Recursive LSTM Tree Representation for Arc-Standard Transition-Based Dependency Parsing

Mohab Elkaref*
IBM Research
Daresbury, UK
mohab.elkaref@ibm.com

Bernd Bohnet
Google Inc.
London, UK
bohnetbd@google.com

Abstract

We propose a method to represent dependency trees as dense vectors through the recursive application of Long Short-Term Memory networks to build Recursive LSTM Trees (RLTs). We show that the dense vectors produced by Recursive LSTM Trees replace the need for structural features by using them as feature vectors for a greedy Arc-Standard transition-based dependency parser. We also show that RLTs have the ability to incorporate useful information from the bi-LSTM contextualized representation used by Cross and Huang (2016) and Kiperwasser and Goldberg (2016b). The resulting dense vectors are able to express both structural information relating to the dependency tree, as well as sequential information relating to the position in the sentence. The resulting parser only requires the vector representations of the top two items on the parser stack, which is, to the best of our knowledge, the smallest feature set ever published for Arc-Standard parsers to date, while still managing to achieve competitive results.

1 Introduction

Neural network-based dependency parsers have typically relied on combination of raw features, as represented by their dense vector embeddings to represent features of a sentence as well as the parser state (Chen and Manning, 2014; Weiss et al., 2015; Andor et al., 2016; Zhou et al., 2015). On the other hand, there has been substantial work on using innovative deep learning architectures to build more informative feature representations.

One approach has been to model an input sentence using bi-directional Long Short-Term Memory Networks (bi-lstms) (Cross and Huang, 2016; Kiperwasser and Goldberg, 2016b). The result is a vector for each word that encodes both its information, and relevant information from other parts of the sentence. This approach enabled better results with fewer features than was possible before (Cross and Huang, 2016; Shi et al., 2017).

Another approach has been to represent the dependency tree itself with some form of recursive network, either bottom-up (Dyer et al., 2015; Kiperwasser and Goldberg, 2016a; Stenetorp, 2013), or top-down (Le and Zuidema, 2014).

In this paper we propose a new method of recursively modelling dependency trees using LSTMs, which we call Recursive Tree LSTMs. Our experiments show that this method of representation is very powerful, and can even be used as an additional layer of encoding over bi-lstm feature representation, which results in a more informative model. The final parser is capable of achieving competitive results with a feature set consisting of only the top two items on the stack, which is the smallest feature set for an Arc-Standard dependency parser used successfully to date.

2 Recursive LSTM Trees (RLTs)

We propose a method of representing a dependency tree as a single dense vector that results from the repeat application of an encoding mechanism to sequences of head-dependent pairs.

*Work done while at University of Birmingham.

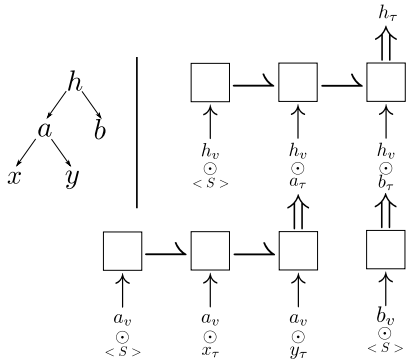

Figure 1: A compact representation showing how a subtree (left) is arranged as a sequence to produce a tree vector for the head token h_τ (right). The encoding mechanism here is a single forward LSTM. The operation $\odot$ is concatenation, $\uparrow$ is input, $\rightharpoonup$ is the passing of the internal state from one LSTM step to another, and $\Uparrow$ is the output of the LSTM. $< S >$ represents the start tag, and is used both to signifiy the start of the head/child sequence and as a base case for leaf nodes, such as in the case of b_τ. The construction of x_τ and y_τ is done in the same way as for b_τ and is omitted for brevity.

Each node in the tree represents a head token's interaction with the representations of all of its immediate dependents. Similarly, the representations of each of these dependents are themselves the result of the interaction between their token and the representations of their corresponding dependents.

Each token has 2 representations, a vector representation v and a tree representation τ. The vector representation is the raw description of a token in its sentence, which in the most basic form can simply be the concatenation of the word and part-of-speech vectors of that token. However, contextualized representation has been shown to be a richer, more informative feature about a token and its position in a sentence (Cross and Huang, 2016; Kiperwasser and Goldberg, 2016b). We experiment with both approaches and confirm that a contextualized vector does improve the performance of RLTs, in addition to its properties being carried over to the RLTs themselves, meaning that parsing can be done with minimal features.

The tree representation of a token, on the other hand, encodes the dependency information of a token and its dependents. Consider a simple subtree consisting of a head token h and its dependents a and b as illustrated in Figure 1. The subtree is represented as a sequence of pairs of head vectors (h_v) and child trees (a_τ, b_τ). These pairs are then input to the encoding mechanism with the final output being the head tree vector h_τ.

The first pair in the sequence representation is always $(h_v, < S >)$, where $< S >$ has the same size as the output size of the encoding mechanism and represents the start tag of the sequence. This also serves as the base case for leaf nodes in the dependency tree as well as for tokens without dependents in partially built trees while parsing.

Each input pair uses the same h_v which is then concatenated with the tree representation of the dependent. The dependents are presented in their order of appearance in the sentence, and the encoding mechanism output at each step can be taken to represent the subtree of h including the dependents introduced up to that step. The recursive element of this formulation is the repeat application of the encoding mechanism, in a bottom-up approach, in order to produce tree representations for tokens that are then used in turn to produce the tree representations of their corresponding heads.

The head token h has one dependent who also has dependents and another which has none. b_τ is represented by the base case with $(b_v \odot < S >)$, while a_τ requires 2 additional steps to incorporate information from x_τ and y_τ.

The dependents a_τ and b_τ must be calculated first before h_τ can be produced, and by extension x_τ and y_τ are required first in order to calculate a_τ. In this way the final dependency tree representation is built recursively, bottom-up, with the final representation being the tree representation $ROOT_\tau$.

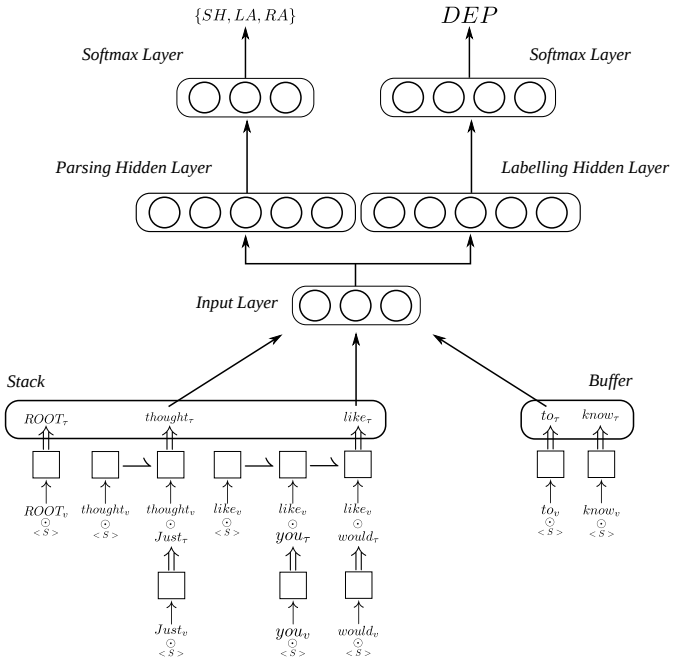

Figure 2: Example of a parser configuration with features using RLTs.

3 Implementation & Training Details

We implemented[1] our model in python using the DyNet framework (Neubig et al., 2017). The encoding mechanisms used by the RLTs in our experiments used 2 layers of LSTMs of size 256. For experiments using bi-lstm contextualized representation, we also used 2 layers of bi-lstms of size 256 in each direction. For the basic vector representations we used randomly initialized part-of-speech tag vectors of size 50, and for word embeddings we used vectors of size 100 initialized using the GloVe vectors (Pennington et al., 2014) trained on 6B. tokens with 400k vocabulary.

The tree vectors of relevant RLTs are then concatenated and passed as input to two sets of two feed-forward hidden layers of size 256, with rectified linear units (ReLUs) (Nair and Hinton, 2010) as activation functions. The two sets of hidden layers are responsible for modelling the relevant information for dependency parsing and dependency labelling separately, similar to the hierarchical architecture used by Cross and Huang (2016). We set a dropout rate of 0.3 on all LSTMs (Gal, 2015) and the hidden layers (Hinton et al., 2012). In our experiments we tried different dropout rates, but the differences were too small to experiment with separate dropout rates for different layers. The final output layers are two separate softmax layers with the same structure as in the setup of Cross and Huang (2016), in which the scores of the output layer corresponding to $\{SH, LA, RA\}$ uses the output of the dependency parsing hidden layers, and the output layer scoring dependencies $\{DEP\}$ uses the output of the dependency labelling hidden layers, where DEP is the set of all possible dependency labels. An illustration of the architecture of the parser is show in figure 2. All weights and pos tag vectors were initialised uniformly (Glorot and Bengio, 2010). For training we use a negative log likelihood loss function, $-\sum_i log(y_i)$, where y_i is the score of the gold transition from the final softmax layer for the training input/output pair i in the mini-batch. We use mini-batch updates of 10 sentences, and stop training after 30 epochs. We optimize the model parameters using Adam (Kingma and Ba, 2014) with a learning rate $\alpha = 1 \times 10^{-3}$. We train our models using the Wall Street Journal (WSJ) section from the Penn Treebank (Marcus et al., 1993). We use §2-21 for training, §22 for development, and §23 for testing. We use Stanford Dependencies (SD) (De Marneffe et al., 2006) converted from constituency trees using version 3.3.0 of the converter. As is standard we use predicted POS tags for the train, dev, and test sets. We report unlabeled attachment score (UAS) and labeled attachment score (LAS), with punctuation excluded. The models are tuned on

[1]Implementation available at `https://github.com/MohabElkaref/rlt`

Encoding Type	UAS	LAS
word/pos embeddings	92.94	90.61
contextualized vectors	**94.26**	**92.01**
K & G (2016a)	**93.3**	90.8
Dyer et al. (2015)	93.2	**90.9**

Table 1: Development set scores on WSJ (SD) comparing between h_v being a concatenation of the tokens word/pos vectors and h_v being a concatenation of contextualized vectors.

Feature Set	UAS	LAS
$\{s_{0-3}, b_{0-3}\}$	**94.26**	**92.01**
$\{s_{0,1}, b_0\}$	94.23	91.99
$\{s_{0,1}\}$	93.88	91.72

Table 2: Development set scores for different feature sets, using a bi-lstm contextualized vector as h_v, for Forward and Bi-directional encoding.

the development set, with the tuning that produced the highest UAS used to obtain the final scores on the test set. We additionally report results on the Universal Dependency set used in the CoNLL'18 shared task in Table 4 for Catalan, German, English, Spanish, French, Italian, and Norwegian.

4 Experiments & Results

For our initial set of experiments we trained models that used the top 4 RLTs on the stack, and the front 4 on the buffer as input features to the feed forward hidden layer. We compare our results initially to those of Dyer et al. (2015), who used Stack-LSTMs, and Kiperwasser and Goldberg (2016a), who used Hierarchical Tree-LSTMs, since they are the closest in the literature to our approach. We make a more complete comparison with state of the art Transition-based parsers in table 3.

Recursive representation was used by Dyer et al. (2015) to represent elements on the stack, similar to our approach. However, their representation is computed through the recursive application of a feed-forward composition function that encodes a (head, relation, dependent) tuple, encoding children in the order in which they are reduced. Kiperwasser and Goldberg (2016a) uses a bottom up recursive approach to build a tree representation as well, but separates the sequence of children into a left and a right sequence, with the head itself being the start of both sequences, and the final representation of the subtree being a concatenation of the output of both sequences. As in our work, Kiperwasser and Goldberg (2016a) use bi-LSTM vectors to represent words being input to the encoding LSTM.

When setting h_v to be the concatenation of the word and pos vectors, the resulting accuracy score, shown in table 1, approaches the performance of Dyer et al. (2015) and Kiperwasser and Goldberg (2016a). Using bi-lstm contextualized representation as h_v, however, significantly improves accuracy to 94.26/92.01 on the development set and beating both of our baselines.

Our second set of experiments were to investigate whether or not RLTs retain the properties of the bi-lstm representation in addition to its own, i.e., produce an h_τ that can represent a token's special position in a sentence *in addition to* representing it as the head of its own subtree.

The results shown thus far are the results of a wide feature set, the first 4 items on both structures $\{s_{0-3}, b_{0-3}\}$, which is comparable to earlier feature sets used by Weiss et al. (2015) and Chen and Manning (2014), but without the need for structural features, such as left-most and right-most dependents which are already encoded in the way a tree vector is produced. The results in Table 2 show the performance of our RLT models on increasingly small feature sets. This second set of experiments show that RLTs are also able to represent contextual information about the node from the bi-lstm layer in addition to its own structural information. Interestingly the drop in the accuracy of RLTs with the complete removal of buffer features is limited. Our minimal feature set here consists of only the top 2 items on the stack $\{s_{0,1}\}$. These 2 elements represent the fundamental task of an Arc-Standard parser, which is to decide whether or not these 2 words are related, and so are not themselves contextual features.

5 Discussion

Vector tree representation has a long history, primarily used to model constituency trees using Recursive neural networks (Goller and Kuchler, 1996; Socher et al., 2010). Such networks relied on the repeat application of a feed forward layer to encode a fixed maximum number of relations. Adapting this

	UAS	LAS
This work		
8 feats. + word/pos embeddings	92.72	90.55
8 feats. + contextualized vectors	94.13	92.11
2 feats. + contextualized vectors	94.04	91.93
Recursive Tree		
Le and Zuidema (2014)	93.84	91.51
Dyer et al. (2015)	93.1	90.9
Kiperwasser and Goldberg (2016a)	93.0	90.9
Ballesteros et al. (2016)	93.56	91.42
Feed Forward		
Chen and Manning (2014)	91.80	89.60
Weiss et al. (2015)	93.99	92.05
Andor et al. (2016)	**94.61**	**92.79**
Bi-lstm contextualized representation		
Cross and Huang (2016)	93.42	91.36
Kiperwasser and Goldberg (2016b)	93.9	91.9
Shi et al. (2017)	94.53	N/A

Table 3: Test set scores on WSJ (SD) for some of the highest scoring Transition-based Dependency Parsers in current literature. Contextualized vectors refer to the bi-lstm vector representation used for h_v, and word/pos embeddings refers to the concatenation of these vectors to represent h_v. 8 feats. refers to the use of the top 4 items on the stack and buffer, 2 feats. refers to the use of the top 2 items on the stack.

Corpus	UAS	LAS
ca_ancora	90.34	87.72
de_gsd	76.71	71.56
en_ewt	82.86	80.18
es_ancora	89.78	87.10
fr_gsd	84.15	80.04
it_isdt	90.45	88.22
no_bokmal	85.83	82.70

Table 4: Test set scores on 7 corpuses from the CONLL'18 shared task. These sets use Universal Dependencies, and use an F1 score calculation for UAS and LAS that includes punctuation.

approach to an arbitrary number of dependents results in deep narrow trees and the gradient vanishing problem. One approach to deal with this has been the Tree-LSTM model, an amended gating mechanism proposed by Tai et al. (2015) based on LSTMs.

For transition-based parsing earlier work with recursive representation includes Stenetorp (2013), who uses a recursive layer to model dependency trees in a manner similar to that used in constituency parsers, but does not produce a high accuracy.

Our main comparisons have been with the work of Kiperwasser and Goldberg (2016a) as it is the closest to our work. They use a bottom up recursive approach to build a tree representation as well, but separate the sequence of children into a left and a right sequence, with the head itself being the start of both sequences, and the final representation of the subtree being a concatenation of the output of both sequences. As in our work, Kiperwasser and Goldberg (2016a) use bi-lstm vectors to represent words being input to the encoding LSTM. We note that in the case of dependents that are leaf nodes in the dependency tree, the representation of Kiperwasser and Goldberg (2016a) models the left sequence backwards and the right sequence forwards. They do not encode information considering the entire set of dependents.

We also compare our results with Dyer et al. (2015), who use a bottom encoding to represent words on the stack, and then uses a stack-LSTM to represent the stack and buffer. The main point of interest here is a recursive composition function which encodes a (head, relation, dependent) tuple, and represents heads with multiple dependents by reapplying the composition function with the previous output as the head. The dependents are encoded into this representation as they are added to the tree, which again means

an unordered representation of dependents. Our models suffered a drop in accuracy when we used an unordered sequence of dependents, which could be a possible explanation for the $\sim$1% difference in accuracy scores.

The performance of RLTs shows a considerable ability to encode structural information into a single dense vector. This ability is highlighted when comparing with Weiss et al. (2015), where the resulting accuracy scores are comparable but only with the additional representation of a structured perceptron. Similarly, the scores of Kiperwasser and Goldberg (2016b) improve by using structural features in addition to the initial set of $\{s_{0-2}, b_0\}$, with the left and right-most modifiers of the first 3 and the left-most modifier of the last, for a total of 11 contextualized features. In both of these cases the stack and buffer features are similar, with RLTs showing an ability to implicitly encode useful structural features in the final tree vector τ.

Additionally RLTs gain much from the use of contextualized vectors as the base representation v. The structure of RLTs predictably is not capable of modelling the sequential position of a word in its sentence, but it can retain the information modelled by the bi-lstm representation fed into it.

Finally, our model produces competitive results with a minimal feature set that, to the best of our knowledge, has not yet been achieved for Arc-Standard, but has been achieved for Arc-Eager and Arc-Hybrid by Shi et al. (2017). A key difference is that our minimal features set consisted of the top 2 items on the stack, while Shi et al. (2017) used the first items from the stack and buffer, which did not work for Arc-Standard. This difference could be due to the different definitions of the LA transition in particular which use the front of the buffer as head, while Arc-Standard limits all transition effects to the stack. Our results remain behind those of Andor et al. (2016) and Shi et al. (2017), both of whom used global loss function, in addition to the latter's exact decoding.

6 Conclusion & Future Work

In this work we proposed a recursive tree architecture capable of modelling both subtrees and whole dependency trees. This method exploits the ability of deep learning to model combinations of features as needed in dense vectors, moving further away from feature selection to more expressive architectures. The resulting vector representation for each word encodes information that describes its position in its dependency tree, as well as its sequential position in its original sentence.

Furthermore, the model might be useful for other applications as well, including question answering and sentence similarity, as the final dense representation captures entire sentences.

Acknowledgements

This work was part funded by the STFC Hartree Centres Innovation Return on Research programme, funded by the Department for Business, Energy Industrial Strategy.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack lstm parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2005–2010. Association for Computational Linguistics.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750.

James Cross and Liang Huang. 2016. Incremental parsing with minimal features using bi-directional LSTM. *CoRR*, abs/1606.06406.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*.

Yarin Gal. 2015. A theoretically grounded application of dropout in recurrent neural networks. *arXiv:1512.05287*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May. PMLR.

Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016a. Easy-first dependency parsing with hierarchical tree lstms. *arXiv preprint arXiv:1603.00375*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016b. Simple and accurate dependency parsing using bidirectional lstm feature representations. *arXiv preprint arXiv:1603.04351*.

Phong Le and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 729–739, Doha, Qatar, October. 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.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Tianze Shi, Liang Huang, and Lillian Lee. 2017. Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. *CoRR*, abs/1708.09403.

Richard Socher, Christopher D Manning, and Andrew Y Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, pages 1–9.

Pontus Stenetorp. 2013. Transition-based dependency parsing using recursive neural networks. In *NIPS Workshop on Deep Learning*. Citeseer.

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.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. *arXiv preprint arXiv:1506.06158*.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 1213–1222.