Journal of Computer Science and Cybernetics, V.39, N.3 (2023), 207–221
DOI no 10.15625/1813-9663/18363

IN-ORDER TRANSITION-BASED PARSING FOR VIETNAMESE

JOHN BAUER[1], HUNG BUI[2], VY THAI[2], CHRISTOPHER D. MANNING[3]

[1]*HAI, Stanford University, 353 Jane Stanford Way Stanford, CA 94305, United States of America*

[2]*Department of Computer Science, Stanford University, 353 Jane Stanford Way Stanford, CA 94305, United States of America*

[3]*Linguistics & Computer Science, Stanford University, 353 Jane Stanford Way Stanford, CA 94305, United States of America*

Abstract. In this paper, we implement a general neural constituency parser based on an in-order parser. We apply this parser to the VLSP 2022 Vietnamese treebank, obtaining a test score of .8393 F1, top of the private test leaderboard. Earlier versions of the parser for languages other than Vietnamese have already been included in the publicly released Python package Stanza. The next Stanza release will include the Vietnamese model, along with all of the code used in this project.

Keywords. Constituency parsing; Vietnamese constituency parsing; Transition parsing; Parser; Dynamic oracle.

1. INTRODUCTION

Constituency parsing is the process of turning a sentence, either raw words or with part of speech tags, into a tree with each subtree labeled with a phrase structure label. For example, in English, the sentence *My hip hurts because of arthritis* is parsed as in Figure 1.

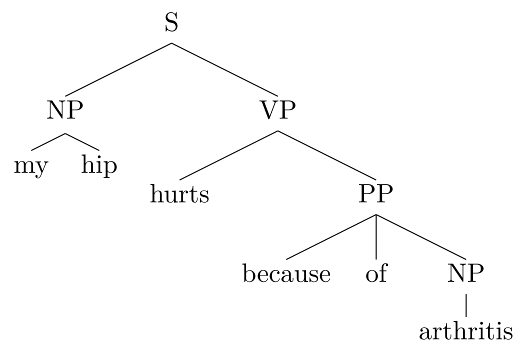


Figure 1: Sample English parse tree

*Corresponding author.
E-mail addresses: horatio@cs.stanford.edu (J. Bauer); hung0411@stanford.edu (H. Bui); vythai@stanford.edu (V. Thai); manning@stanford.edu (C.D. Manning).

© 2023 Vietnam Academy of Science & Technology

There are many possible models for parsing. Broadly, they usually fall into two categories: dynamic programming or chart parsers, which, for example, use the CKY algorithm to build up progressively larger subtrees until they reach the top, and transition-based parsers, which add one word at a time to a partially finished tree until the entire tree is finished.

As part of the publicly released Stanza open-source software distribution [1],* we have implemented a language-agnostic neural in-order transition-based parser [2]. The model already supports several human languages, including English [3], Chinese [4], Portuguese [5], Turkish [6], Japanese [7], Italian [8], [9], and Danish [10].

When applied to Vietnamese, we are pleased to report that we finished first on the private test section, with an F1 of 0.8393. In this report, we describe how the in-order transition-based parser works and describe the work done specifically for Vietnamese.

2. IN-ORDER PARSING

2.1. Transition-based parsing

To understand in-order parsing, it is first necessary to understand the basics of transition-based parsing. The underlying mechanism is similar to that of a shift/reduce compiler.

To implement a shift/reduce compiler without a recursive stack, the compiler maintains a state with two data structures: a stack of components it has already built, which will have zero or more pieces on it, and a queue of tokens remaining to be processed. The operations allowed are to shift a new item from the queue onto the stack and to reduce some number of items from the stack into a larger, combined item.

Whereas a compiler has deterministic rules for when these operations are applied, typically by looking at the stack and the next one or more tokens on the queue, a transition-based parser for dependencies and/or constituencies predicts the next transition at a given stack and queue state. The stack will be a list of partially constructed trees, and the queue is the unparsed words. At each time step, the predicted transition is applied, updating the stack and queue, and the process repeats until the queue is exhausted and the stack has only one item on it. The equivalent shift action moves a single word onto the stack of partial trees, and the reduce action combines two or more subtrees into a larger tree. In treebanks with unary transitions, those can be represented either by a unary transition which operates on a single node, or by a reduce action which simultaneously combines multiple nodes and applies multiple layers of the tree at the same time.

The transition scheme just described is a bottom-up transition scheme, and early constituency parsers such as [11] used this scheme to achieve efficient and accurate results. Since then, more advanced transition schemes have been proposed such as top-down [12], in-order [2], or attach/juxtapose [13]. In this project, we implemented both top-down and in-order transition schemes. Although the top-down scheme is less accurate than the in-order, it proved useful for producing high-accuracy training data (see Subsection 4.3).

Top-down adjusts the previous bottom-up scheme by introducing a non-terminal transition. It starts a constituent, allowing the parser to build any number of subtrees until applying the reduce action.

In-order further extends that idea by building the left child of a tree first, *then* adding a

*https://stanfordnlp.github.io/stanza/

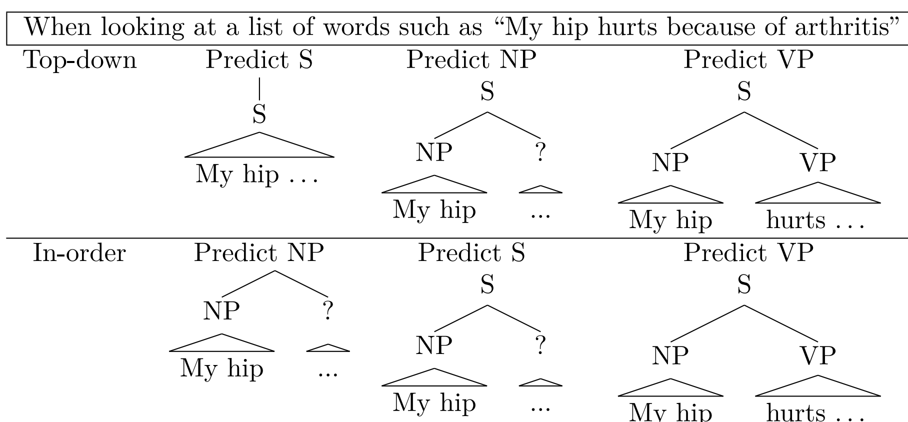


Figure 2: Comparison between top-down and in-order. In-order parsing completes the leftmost NP child of the S before predicting the parent S category.

non-terminal label to the stack, and then building the rest of the subtree before applying a reduce. See Table 1 for a complete example of the (simpler) top-down transition scheme and see Figure 2 for a small illustration of how in-order parsing differs. In early experiments, the added in-order context improved scores from 0.8171 to 0.8206 when splitting the train set into 0.9 train and 0.1 validation.

Depending on the exact transition scheme used, there are likely constraints on the predicted transitions, a full list of which is available in the code for this project but is beyond the scope of this report.

Table 1: Top-down transition example

Stack	Transition	Queue
∅	∅	My, hip, hurts, because, of, arthritis
(ROOT	NT_{ROOT}	My, hip, hurts, because, of, arthritis
(ROOT (S	NT_{S}	My, hip, hurts, because, of, arthritis
(ROOT (S (NP	NT_{NP}	My, hip, hurts, because, of, arthritis
(ROOT (S (NP My	Shift	hip, hurts, because, of, arthritis
(ROOT (S (NP My hip	Shift	hurts, because, of, arthritis
(ROOT (S (NP My hip)	Close	hurts, because, of, arthritis
(ROOT (S (NP My hip) (VP	NT_{VP}	hurts, because, of, arthritis
(ROOT (S (NP My hip) (VP hurts	Shift	because, of, arthritis
(ROOT (S (NP My hip) (VP hurts (PP	NT_{PP}	because, of, arthritis
(ROOT (S (NP My hip) (VP hurts (PP because	Shift	of, arthritis
(ROOT (S (NP My hip) (VP hurts (PP because of	Shift	arthritis
(ROOT (S (NP My hip) (VP hurts (PP because of (NP	NT_{NP}	arthritis
(ROOT (S (NP My hip) (VP hurts (PP because of (NP arthritis	Shift	∅
(ROOT (S (NP My hip) (VP hurts (PP because of (NP arthritis)	Close	∅
(ROOT (S (NP My hip) (VP hurts (PP because of (NP arthritis))	Close	∅
(ROOT (S (NP My hip) (VP hurts (PP because of (NP arthritis)))	Close	∅
(ROOT (S (NP My hip) (VP hurts (PP because of (NP arthritis))))	Close	∅
(ROOT (S (NP My hip) (VP hurts (PP because of (NP arthritis)))))	Close	∅

2.2. Neural transition-based parsing

Early transition-based constituency parsers used perceptrons over features of words and their neighbors [11]. These days, feature-based models have been mostly replaced with neural models. Instead of features, the starting point is some form of embedding of the original text, such as those produced by word2vec [14] or Glove [15].

3. IMPROVEMENTS TO BASE MODEL

The simplest and most effective improvement to the base model of [2] is to make use of stronger word representations (see textcolorredsection 4.1.). Beyond that, there are several improvements made to the base model and its training.

The first is that, whereas [12] and [11] used a Bi-LSTM as a composition function when merging several constituents, we found that a simple *max* function worked better.

We tried several variants of the constituent function aside from the Bi-LSTM and *max*. A Tree-LSTM [16] using the previous layers and an embedding over the constituent types as the inputs were competitive, in some cases even better than *max*, especially when also using an embedding to initialize the cell state of the Tree-LSTM. However, on some datasets, we found that it was less effective, and this method was substantially slower, so we left *max* as the default. Taking a max over bigrams was slightly worse than *max*, as was taking the max over node outputs instead of combining the endpoints of the original bi-LSTM method. We also tried multiple forms of self-attention over the nodes, as it seemed compelling that an attention layer should be able to learn which of the subtrees matter most, but none of the variants we tried beat *max*; see Figure 3.

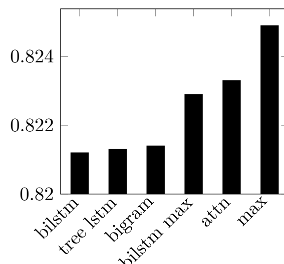


Figure 3: VI dev scores for various constituency composition methods

Both the previous top-down and in-order papers [2], [12] use a one-directional LSTM to represent the incoming word buffer. We found that there is a slight increase in performance using a bi-LSTM instead. It should be noted that using the bi-LSTM rules out using a generative version of the model for reranking, though.

Liu and Zhang briefly discussed the nonlinearity they used between layers. We further explored that concept, testing virtually every nonlinearity available in PyTorch [17]; see Figure 4. Out of all of them, ReLU [18] and GeLU [19] were the most effective. When tested over multiple trials, ReLU slightly beat out GeLU. An A/B test specifically between ReLU and GeLU, training on gold + silver data (see Subsection 4.3) gave 0.8270 averaged over 5 models with ReLU, and 0.8248 averaged over 5 models for GeLU. Accordingly, we kept the ReLU nonlinearity for the remainder of the project. Other nonlinearities we tried include Mish [20], SiLU [21], tanh, ELU [22], and ReLU6 [23], among others.

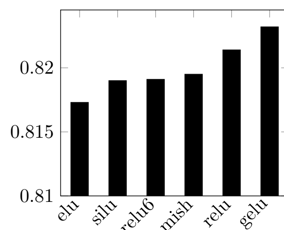


Figure 4: VI dev scores with use of the different non-linearities

We implemented both a partitioned attention layer [24] and a labeled attention layer [25], although unfortunately, we were unable to find a balance between the default encoder and the attention layers which improved scores. Some mechanisms we attempted were to concat the attention outputs to the encoder outputs, replace one with the other, or use various weightings of residual connections. Future work will include tuning these layers to get better results in Vietnamese or the other languages for which we build models.

3.1. Training mechanism

Of the learning algorithms included in PyTorch, we found that AdaDelta [26] worked the best, with AdamW [27] a close second. An externally available algorithm, MADGRAD (and the associated Mirror MADGRAD) [28], was slightly more accurate than AdamW.

An interesting observation, though, was that while AdamW, MADGRAD, and similar optimizers generally achieved their high accuracy scores by making very small adjustments in the tensors of the model, AdaDelta with a high weight decay made much larger changes to the model. For example, the norm of the tensor which converts a word to a constituent gradually changes with MADGRAD, but changes drastically at the beginning with AdaDelta, as shown in Figure 5.

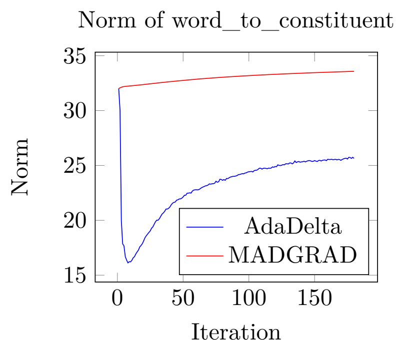


Figure 5: Effect of optimizer choice on the tensor norm

The hypothesis is that AdamW, MADGRAD, and similar optimizers can find a very strong local minimum in the neighborhood of the initial conditions, but the initial starting points for the values of the matrices in the model were far from ideal. AdaDelta with high weight decay, on the other hand, would effectively move to a much better region of the state space, but would then be unable to find the strongest local minimum in the new region of the state space.

Accordingly, we introduced a mechanism where we first trained for 70 epochs on AdaDelta to essentially pick better initial conditions, then trained for another 70 epochs using AdamW or MADGRAD. This led to much higher accuracy than either optimizer by itself (see Figure 6).

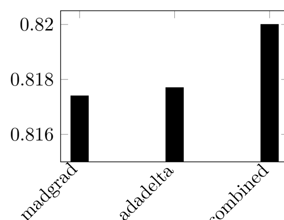


Figure 6: VI dev scores for different choices of optimizer

4. LANGUAGE SPECIFIC PROCESSING

4.1. Encoder

The encoder for the parser is flexible and can make use of several types of components.

The simplest foundational component used in neural NLP models is word embeddings. In this case, we used the word2vec embeddings from the CoNLL 2018 Shared Task [29]. While there are more recent and presumably more accurate word vectors available for Vietnamese, such as PhoW2V [30], that package has a restrictive license, and the word vectors chosen have a relatively low impact once larger language models are employed. Using these vectors gives us a score of .7623 on a randomly chosen validation set.

A character language model such as that used in Flair is a significant improvement over using word vectors alone [31]. The CoNLL 2017 shared task included a large repository of Vietnamese data for use in training large language models. Using this dataset, we trained forward and backward character models for Vietnamese. Adding these character models to the encoder improved results to 0.7732. Larger repositories of Vietnamese text are available, such as the Oscar Common Crawl [32] or Wikipedia; again, though, our expectation is larger transformer-based language models would help more than retraining the character model on a larger dataset would have.

Two transformer-based models have been released in recent years, both by the VinAI research group. PhoBERT [33] and BARTpho [34] both drastically improve results. BARTpho improved results to 0.78, or 0.785 when using diacritic normalization, whereas PhoBERT improved results to 0.82+ when using the full architecture described here. Specifically, we obtained the best results when using a learned weighting over 6, 7, or 8 of the final output layers from vinai/phobert-large.

In general, using transformers as the input greatly improves the quality of the final model. For the various languages listed above, we found improvements for most of them using transformers (see Table 2).

4.2. POS tagging

In most parsing tasks, including this one, the dataset itself contains POS tags. However, for best results, the tags should not be used as inputs to the training or evaluation of the

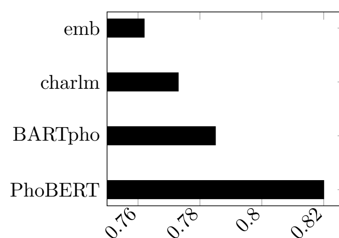


Figure 7: VI dev scores for different choices of pretraining

Table 2: With and without Transformer parsing scores

lang	w/o T	w/ T	Transformer	ref
da	82.70	83.45	bert-base-multilingual-cased	[35]
en	93.21	95.80	roberta-base	[36]
it (tut)	89.42	92.76	Musixmatch/umberto-commoncrawl-cased-v1	[37]
it (vit)	78.52	82.43	Musixmatch/umberto-commoncrawl-cased-v1	[37]
pt	90.98	93.61	neuralmind/bert-large-portuguese-cased	[38]
tr	73.04	75.70	dbmdz/bert-base-turkish-128k-cased	[39]
zh	86.85	90.98	hfl/chinese-roberta-wwm-ext	[40]

parsing model, as gold standard tags will not be available when using the parser on raw text. Indeed, the test section for this task did not even provide the tags.

To handle tagging, we used the Stanza tagger [1] retrained on the gold tags provided for the training portion of the task. When trained on 90% of the data, with 10% held out for validation, a tagger using PhoBERT-Large [33] achieved 94.1% token accuracy.

We then similarly shared the input data with the constituency data, using an ensemble of 5 taggers to produce the tags used to train the model used in the bakeoff.

4.3. Additional silver-standard data

In addition to training on the VLSP dataset, we parsed a large collection of text from Wikipedia† and used this as silver standard data. To split Wikipedia into sentences, we first extracted it with WikiExtractor [41] and then tokenized it with RDRSegmenter [42].

To ensure the accuracy of the silver data, we constructed a top-down version of the parser [12] and only selected trees where the in-order and top-down models aligned, as in Choe-Charniak [43]. After filtering duplicate sentences, and removing small articles in order to eliminate the 200,000 sentences isomorphic to "The dung beetle is a type of beetle", this process resulted in 1.2M silver standard trees.

Retraining this collection of silver trees at a rate of 1 silver tree per gold tree resulted in a noticeable improvement. When using 5 models in an ensemble, scoring on a held-out 1/10th of the data, training on the silver trees improved scores from .8281 to .8315.

Inspecting the results revealed a gap in performance for a specific tree structure, that of identifying questions (SQ) versus statements (S). Partly this was because there were very few questions in the original training data. One would expect ? at the end of a sentence to be

†https://dumps.wikimedia.org/backup-index.html

indicative, but there were some sentences in the training set which did not have an *SQ* label despite the question mark. Furthermore, there were very few trees with any form of *WH-* node in them, leading to a very low probability of the parser predicting *WH-* as a constituent.

To compensate for this, we parsed the questions in UIT-ViQuAD [44] and used them to build a silver dataset similar to the Wikipedia dataset. To ensure relevance to the problem of parsing questions correctly, we discarded all sentences which did not have an *SQ* as the top node or a *WH-* constituent somewhere in the tree. Unfortunately, this did not improve performance. When using 5 models with 1/10th held out as a test set, the performance with the Wikipedia silver only was .8315, whereas the performance with the added questions was .8313.

Our best working theory is that the filtering step was too restrictive. The difference between this and the Wikipedia silver trees is that the Wikipedia silver dataset introduced many new words, letting the parser learn on out-of-domain text, whereas the restrictive filtering of UIT-ViQuAD meant the parser only saw trees with question words it already knew to label as *WH-*. Given more time, we should instead either manually adjust UIT-ViQuAD sentences parsed as *S* to have an *SQ* and a *WH-* node, or we should generate multiple trees and take the highest scoring tree with *SQ* and *WH-* in it.

5. DYNAMIC ORACLE

The main theoretical improvement to the parser in this work is a *dynamic oracle.*

In the most commonly used teacher forcing training, at every step of training, the model makes a predicted transition from which a loss is calculated, but then the predicted transition is discarded and the gold transition is applied. This is suboptimal because, at test time, that means the model may make an error, transition into a state space it has not trained on, and have less context to make the following decisions. This leads to one error cascading into more errors, whereas ideally, the model would continue to make the best tree possible after such an error.

Instead, we make a *dynamic oracle* which adjusts the remaining gold transitions to best match the parser's errors at training time. The general intuition is that a missed bracket or extra bracket early in the parse does not affect the transition sequence of later subtrees, so the transition sequence can be repaired to minimize the bracket errors due to the incorrect subtree by keeping the rest of the transition sequence intact.

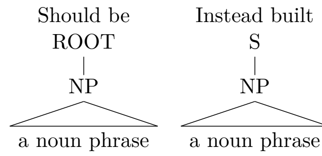
One caveat is that there are sequences where an error can lead to multiple trees with equivalent scores. In such cases, we found that using the dynamic oracle to enforce one or the other result tree leads to worse scores overall. Instead, when an ambiguous tree such as this occurs, we revert to teacher forcing. In our experiments, roughly 1/3rd of the errors which a fully trained model makes are ambiguous in one way or another.

Dynamic oracles have been used in the past for dependency parsing [45] and bottom-up transition parsers [46], but to the best of our knowledge, this is the first use of one in an in-order parser.

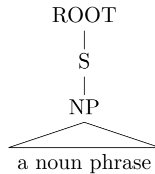
We now discuss some particular types of errors that are addressed by the dynamic oracle.

Root unary. The first error type we fix is one of the simplest. At the end of each sequence when using the standard in-order transition scheme, there is a unary transition to ROOT, executed by $\mathrm{NT}_{\mathrm{ROOT}}$, CLOSE. In the event of parsing a short phrase which does not end in

S, the parser may well transition to S anyway before adding the ROOT:

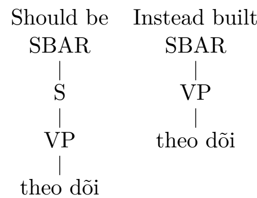


In this case, the simplest fix is to add the transition to ROOT after the incorrect replacement.

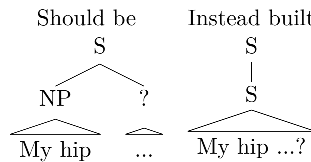


Wrong unary. If a transition is predicted which skips part of a unary chain, it is safe to continue with no further alterations. A long chain of unaries is rare and difficult to get right. Note that without the oracle, the learning algorithm would force the model to make the correct unary chain instead.

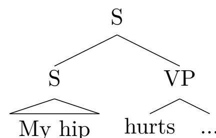
As an example of where this might happen in the VLSP dataset



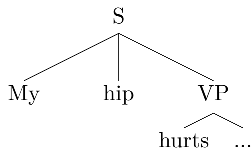
Nested open. This error occurs when a subtree is missed, a different non-terminal is chosen, and the non-terminal matches the surrounding tree.



In this case, there are a few possible solutions. For example, we could keep the wrong transition, ignore the missed subtree, or attempt to add a unary transition to the correct subtree. Using a unary transition to the correct subtree is hard to learn. In terms of final scoring, predicting the wrong label adds a recall error to the final score, unless we use a unary transition to fix the missing subtree. However, in general, learning unary sequences is difficult for the in-order model. Treating the wrong label as a new subtree can lead to unusual structures; in this case, the final tree would have an extra S if we build a subtree with the new label:



Furthermore, this would add a precision error as well as a recall error. To minimize the total errors learned, while not making the structure too difficult to learn, we simply drop the expected subtree (the NP in this case):



It is unclear from experiments whether this helps much, and there is certainly room for other options to work better.

Other oracle fixes These are just the first few oracle fixes; the remainder are documented in the source code for the parser.

The impact of oracle fixes applied cumulatively on just one training run is shown in Figure 8.

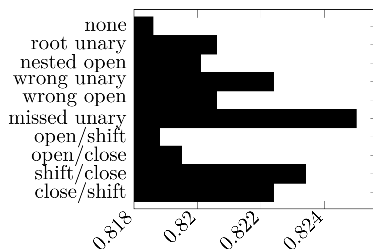


Figure 8: VI dev scores for cumulatively adding individual oracle fixes

Most of the oracle fixes produce small gains. However, although some variation is expected in the scores of different models, the large drop when adding the open/shift correction is concerning and should be further investigated to make sure the oracle is working for that particular transition error.

6. EXPERIMENTAL SETUP

We trained multiple versions of the model on the Stanford computer cluster, A/B testing various code changes and parameter sweeping across various hyperparameters. The hyperparameters of our most successful model are in Table 3. In particular, we increased the hidden size of the model compared to [2], as the total encoder dimension is much higher (WV + tag embedding + charlm + BERT) compared to the original in-order parser.

The training data for the VLSP dataset consisted of 8160 sentences. The unparsed data for the bakeoff consisted of 5000 sentences, of which 1204 were later released as gold-parsed test sentences. One model takes roughly 12 hours to converge on an Nvidia 2080ti GPU when trained on the VLSP dataset, and 24 hours when trained with the silver dataset as well.

For searching over hyperparameters and testing the model improvements described here, we started from the 8160 training trees and held out 10% as a validation set. The final model submitted to VLSP was an ensemble of 5 models trained on a different 4/5th of the given training data, using the remaining 1/5th for validation for that specific model. We used AdaDelta + MADGRAD as the optimizer for the final models and included the corpus of silver trees described above.

Table 3: Hyperparameters

Parameter	Value
Optimizer	AdaDelta + MADGRAD
AdaDelta LR	1.0
AdaDelta WD	0.02
MADGRAD LR	7e-7
MADGRAD WD	2e-6
Trans. embedding	20
Hidden size	512
Nonlinearity	ReLU
Constituency Composition	Max
Dynamic Oracle	Yes
Dropout	0.2
LSTM Layers	2
FC Layers at Output	3
Ensemble size	5

Table 4: Hyperparameters

Model	Dev % F1
No Oracle, No Silver	81.48
Oracle, No Silver	81.91
Oracle and Silver	82.65

When tested on the held-out portion of the training data, we achieved the results shown in Table 4.

7. FUTURE WORK

In addition to making use of partitioned attention and labeled attention (see section 3.), another technique which has seen great success is building a reranker over multiple candidate parses. We tried multiple reranking techniques, but none of them proved beneficial. Finding a reranked implementation which improves the scores should be a good source of improved F1. Reducing the ambiguities in the dynamic oracle should make the oracle more effective, also leading to higher scores overall.

Several varieties of transition sequences are theoretically possible. Currently, the non-terminals are labeled. Labeling the reduces instead or in addition to the non-terminals may give the model more context to use when making the final decision on a constituent's label. Furthermore, unary transitions in an in-order sequence are implemented in a somewhat awkward manner: first a non-terminal is pushed, then it is immediately closed. Initial experiments to replace that with a single transition were quite promising, but unfortunately adding such a transition would require reimplementing the entire dynamic oracle, so the net result was negative. Further refining that mechanism and updating the oracle to be compatible with the new transition scheme may improve performance.

8. CONCLUSION

We introduce the Stanza constituency parser, based on an in-order transition-based parser, with language-agnostic accuracy improvements and additional work specific to Vietnamese

parsing. The high-accuracy language-agnostic model, combined with Vietnamese-specific work, is shown to work well on Vietnamese constituency parsing.

ACKNOWLEDGEMENTS

We would like to thank VLSP for organizing an interesting and challenging project. We would also like to thank Jiangming Liu, author of In-Order Parsing [2], for clarifying a couple of points on the implementation of a transition parser. Rodolfo Delmonte held extensive discussions with us regarding the Italian version of the parser, although those discussions were not directly applicable to the Vietnamese results.

At Stanford, we would like to thank Drew Hudson and Sidd Karamcheti for valuable discussions regarding model architecture and model training.

REFERENCES

[1] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning, *Stanza: A Python natural language processing toolkit for many human languages*, 2020. DOI: 10.48550/ARXIV.2003.07082. [Online]. Available: https://arxiv.org/abs/2003.07082.

[2] J. Liu and Y. Zhang, "In-order transition-based constituent parsing," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 413–424, 2017. DOI: 10.1162/tacl_a_00070. [Online]. Available: https://aclanthology.org/Q17-1029.

[3] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993. [Online]. Available: https://aclanthology.org/J93-2004.

[4] N. Xue, F. Xia, F.-D. Chiou, and M. Palmer, "The Penn Chinese Treebank : Phrase structure annotation of a large corpus," *Natural Language Engineering*, vol. 11, pp. 207–238, 2005.

[5] J. Silva, A. Branco, S. Castro, and R. Reis, *Out-of-the-Box Robust Parsing of Portuguese*, Springer, Berlin, 2010.

[6] N. Kara, B. Marşan, M. Özçelik, *et al.*, "Creating a syntactically felicitous constituency treebank for turkish," in *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2020, pp. 1–6. DOI: 10.1109/ASYU50717.2020.9259873.

[7] Y. K. Thu, W. P. Pa, M. Utiyama, A. Finch, and E. Sumita, "Introducing the Asian language treebank (ALT)," in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, Portorož, Slovenia: European Language Resources Association (ELRA), May 2016, pp. 1574–1578. [Online]. Available: https://aclanthology.org/L16-1249.

[8] R. Delmonte, A. Bristot, and S. Tonelli, "VIT – Venice Italian Treebank: Syntactic and quantitative features," in *Proceedings of the Sixth International Workshop on Treebanks and Linguistic Theories*, 2007, pp. 43–54.

[9] C. Bosco, "Multiple-step treebank conversion: From dependency to Penn format," in *Proceedings of the Linguistic Annotation Workshop*, Prague, Czech Republic: Association for Computational Linguistics, Jun. 2007, pp. 164–167. [Online]. Available: https://aclanthology.org/W07-1526.

[10] E. Bick, "Arboretum, a hybrid treebank for Danish," in *Proceedings of Treebanks and Linguistic Theory*, J. Nivre and E. Hinrich, Eds., 2003, pp. 9–20.

[11] M. Zhu, Y. Zhang, W. Chen, M. Zhang, and J. Zhu, "Fast and accurate shift-reduce constituent parsing," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 434–443. [Online]. Available: https://aclanthology.org/P13-1043.

[12] C. Dyer, A. Kuncoro, M. Ballesteros, and N. A. Smith, "Recurrent neural network grammars," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 199–209. DOI: 10.18653/v1/N16-1024. [Online]. Available: https://aclanthology.org/N16-1024.

[13] K. Yang and J. Deng, "Strongly incremental constituency parsing with graph neural networks," in *Neural Information Processing Systems (NeurIPS)*, 2020.

[14] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *International Conference on Learning Representations*, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:5959482.

[15] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. [Online]. Available: https://aclanthology.org/D14-1162.

[16] K. S. Tai, R. Socher, and C. D. Manning, "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)*, Beijing, China: Association for Computational Linguistics, Jul. 2015, pp. 1556–1566. DOI: 10.3115/v1/P15-1150. [Online]. Available: https://aclanthology.org/P15-1150.

[17] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[18] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological Cybernetics*, vol. 20, no. 3, pp. 121–136, 1975.

[19] D. Hendrycks and K. Gimpel, *Gaussian error linear units (GELUs)*, 2016. DOI: 10.48550/ARXIV.1606.08415. [Online]. Available: https://arxiv.org/abs/1606.08415.

[20] D. Misra, *Mish: A self regularized non-monotonic activation function*, 2019. DOI: 10.48550/ARXIV.1908.08681. [Online]. Available: https://arxiv.org/abs/1908.08681.

[21] S. Elfwing, E. Uchibe, and K. Doya, *Sigmoid-weighted linear units for neural network function approximation in reinforcement learning*, 2017. DOI: 10.48550/ARXIV.1702.03118. [Online]. Available: https://arxiv.org/abs/1702.03118.

[22] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (ELUs)*, 2015. DOI: 10.48550/ARXIV.1511.07289. [Online]. Available: https://arxiv.org/abs/1511.07289.

[23] A. G. Howard, M. Zhu, B. Chen, *et al.*, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, 2017. DOI: 10.48550/ARXIV.1704.04861. [Online]. Available: https://arxiv.org/abs/1704.04861.

[24] N. Kitaev and D. Klein, *Constituency parsing with a self-attentive encoder*, 2018. DOI: 10.48550/ARXIV.1805.01052. [Online]. Available: https://arxiv.org/abs/1805.01052.

[25] K. Mrini, F. Dernoncourt, Q. Tran, T. Bui, W. Chang, and N. Nakashole, *Rethinking self-attention: Towards interpretability in neural parsing*, 2019. DOI: 10.48550/ARXIV.1911.03875. [Online]. Available: https://arxiv.org/abs/1911.03875.

[26] M. D. Zeiler, *Adadelta: An adaptive learning rate method*, Dec. 2012. [Online]. Available: https://arxiv.org/abs/1212.5701.

[27] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, 2017. DOI: 10.48550/ARXIV.1711.05101. [Online]. Available: https://arxiv.org/abs/1711.05101.

[28] A. Defazio and S. Jelassi, *Adaptivity without compromise: A momentumized, adaptive, dual averaged gradient method for stochastic optimization*, 2021. arXiv: 2101.11075 [cs.LG].

[29] D. Zeman, J. Hajič, M. Popel, *et al.*, "CoNLL 2018 shared task: Multilingual parsing from raw text to Universal Dependencies," in *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 1–21. DOI: 10.18653/v1/K18-2001. [Online]. Available: https://aclanthology.org/K18-2001.

[30] A. T. Nguyen, M. H. Dao, and D. Q. Nguyen, "A pilot study of text-to-SQL semantic parsing for Vietnamese," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 4079–4085.

[31] A. Akbik, D. Blythe, and R. Vollgraf, "Contextual string embeddings for sequence labeling," in *Proceedings of the 27th International Conference on Computational Linguistics*, Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 1638–1649. [Online]. Available: https://aclanthology.org/C18-1139.

[32] J. Abadji, P. Ortiz Suarez, L. Romary, and B. Sagot, "Towards a cleaner document-oriented multilingual crawled corpus," *arXiv e-prints*, arXiv:2201.06642, arXiv:2201.06642, Jan. 2022. arXiv: 2201.06642 [cs.CL].

[33] D. Q. Nguyen and A. T. Nguyen, "PhoBERT: Pre-trained language models for Vietnamese," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 1037–1042.

[34] N. L. Tran, D. M. Le, and D. Q. Nguyen, "BARTpho: Pre-trained sequence-to-sequence models for Vietnamese," in *Proceedings of the 23rd Annual Conference of the International Speech Communication Association*, 2022.

[35] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805.

[36] Y. Liu, M. Ott, N. Goyal, *et al.*, *RoBERTa: A robustly optimized BERT pretraining approach*, 2019. DOI: 10.48550/ARXIV.1907.11692. [Online]. Available: https://arxiv.org/abs/1907.11692.

[37] L. Parisi, S. Francia, and P. Magnani, *Umberto: An italian language model trained with whole word masking*, https://github.com/musixmatchresearch/umberto, 2020.

[38] F. Souza, R. Nogueira, and R. Lotufo, "BERTimbau: Pretrained BERT models for Brazilian Portuguese," in *9th Brazilian Conference on Intelligent Systems, BRACIS, Rio Grande do Sul, Brazil, October 20-23 (to appear)*, 2020.

[39] S. Schweter, *BERTurk – BERT models for Turkish*, version 1.0.0, Apr. 2020. DOI: 10.5281/zenodo.3770924. [Online]. Available: https://doi.org/10.5281/zenodo.3770924.

[40] Y. Cui, W. Che, T. Liu, B. Qin, S. Wang, and G. Hu, "Revisiting pre-trained models for Chinese natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, Online: Association for Computational Linguistics, Nov. 2020, pp. 657–668. [Online]. Available: https://www.aclweb.org/anthology/2020.findings-emnlp.58.

[41] G. Attardi, *Wikiextractor*, https://github.com/attardi/wikiextractor, 2015.

[42] D. Q. Nguyen, D. Q. Nguyen, T. Vu, M. Dras, and M. Johnson, "A fast and accurate Vietnamese word segmenter," in *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018)*, 2018, pp. 2582–2587.

[43] D. K. Choe and E. Charniak, "Parsing as language modeling," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2331–2336. DOI: 10.18653/v1/D16-1257. [Online]. Available: https://aclanthology.org/D16-1257.

[44] K. Nguyen, V. Nguyen, A. Nguyen, and N. Nguyen, "A Vietnamese dataset for evaluating machine reading comprehension," in *Proceedings of the 28th International Conference on Computational Linguistics*, Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020, pp. 2595–2605. DOI: 10.18653/v1/2020.coling-main.233. [Online]. Available: https://aclanthology.org/2020.coling-main.233.

[45] Y. Goldberg and J. Nivre, "A dynamic oracle for arc-eager dependency parsing," in *COLING*, 2012.

[46] M. Coavoux and B. Crabbé, "Neural greedy constituent parsing with dynamic oracles," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 172–182. DOI: 10.18653/v1/P16-1017. [Online]. Available: https://aclanthology.org/P16-1017.

Received on May 23, 2023
Accepted on August 28, 2023