Revisiting Tri-training of Dependency Parsers

Joachim Wagner and **Jennifer Foster**
School of Computing
Dublin City University
Dublin, Ireland
firstname.lastname@dcu.ie

Abstract

We compare two orthogonal semi-supervised learning techniques, namely tri-training and pretrained word embeddings, in the task of dependency parsing. We explore language-specific FastText and ELMo embeddings and multilingual BERT embeddings. We focus on a low resource scenario as semi-supervised learning can be expected to have the most impact here. Based on treebank size and available ELMo models, we select Hungarian, Uyghur (a zero-shot language for mBERT) and Vietnamese. Furthermore, we include English in a simulated low-resource setting. We find that pretrained word embeddings make more effective use of unlabelled data than tri-training but that the two approaches can be successfully combined.

1 Introduction

Pre-trained neural architectures and contextualised word embeddings are state-of-the-art approaches to combining labelled and unlabelled data in natural language processing tasks taking text as input. A large corpus of unlabelled text is processed once and the resulting model is either fine-tuned for a specific task or its hidden states are used as input for a separate model. In the task of dependency parsing, recent work is no exception to the above. However, earlier, pre-neural work explored many other ways to use unlabelled data to enrich a parsing model. Among these, self-, co- and tri-training had most impact (Charniak, 1997; Steedman et al., 2003; McClosky et al., 2006a,b; Søgaard and Rishøj, 2010; Sagae, 2010).

Self-training augments the labelled training data with automatically labelled parse trees predicted by a baseline model in an iterative process:

1. Select unlabelled sentences to be parsed in this iteration
2. Parse sentences with current model
3. Optionally discard some of the parse trees, e. g. based on parser confidence
4. Optionally oversample the original labelled data to give it more weight
5. Train a new model on the concatenation of manually labelled and automatically labelled data
6. Check a stopping criterion

Co-training proceeds similarly to self-training but uses two different learners, each teaching the other learner, i. e. output of learner A is added to the training data of learner B and vice versa. *Tri-training* uses three learners and only adds predictions to a learner that the other two learners, the teachers, agree on. As with co-training, the roles of teachers are rotated so that all three learners can receive newly labelled data.

We compare tri-training and contextualised word embeddings in the task of dependency parsing, using the same unlabelled data for both approaches. In this comparison, we will try to answer:

1. How does semi-supervised learning with tri-training compare to semi-supervised learning with a combination of context-independent and contextualised word embeddings?
2. Are the above two approaches orthogonal, i. e. do we get an additional boost if we combine them?
3. How do these three approaches compare to the baseline of using only the manually labelled data?

We focus on low-resource languages as *(a)* maximising the benefits from semi-supervised learning even at high computational costs is most needed for low-resource languages, e. g. to reducing editing effort in the manual annotation of additional data, and *(b)* tri-training with high-resource languages comes at much higher computational costs as not

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 9457–9473
November 7–11, 2021.

only is the manually labelled data much larger but also the automatically labelled data can be expected to need to be at least equally larger to have a relevant effect. We select three low-resource languages, namely Hungarian, Uyghur and Vietnamese, (see Section 3.3 for selection criteria) and, English, simulating a low-resource scenario by sampling a subset of the available data.

The results of our experiments show that 1) both tri-training and pretrained word embeddings offer an obvious improvement over a fully supervised approach, 2) pretrained word embeddings clearly outperform tri-training (between 2 and 5 LAS points, depending on the language), and 3) there is some merit in combining the two approaches since the best performing model for each of the four languages is one in which tri-training is applied with models which use pretrained embeddings.

2 Background

2.1 Tri-training

Tri-training has been used to tackle various natural language processing problems including dependency parsing (Søgaard and Rishøj, 2010), part-of-speech tagging (Søgaard, 2010; Ruder and Plank, 2018), chunking (Chen et al., 2006), authorship attribution (Qian et al., 2014) and sentiment analysis (Ruder and Plank, 2018). Approaches differ not only in the type of task (sequence labelling, classification, structured prediction) but also in the flavour of tri-training applied. These differences take the form of the method used to introduce diversity into the three learners, the number of tri-training iterations and whether a stopping criterion is employed, the balance between manually and automatically labelled data, the selection criteria used to add an automatically labelled instance to the training pool, and whether automatic labels from previous iterations are retained.

Zhou and Li (2005) introduce tri-training. They experiment with 12 binary classification tasks with data sets from the UCI machine learning repository, using bootstrap samples for model diversity. Each pair of learners, the teachers, sends their unanimous predictions to the remaining third learner if *(a)* the error rate as measured on the subset of the manually labelled data which the two learners agree on is below a threshold and *(b)* the total number of items that the teachers agree on and therefore can hand over to the learner reaches a minimum number that is adjusted in each round for each learner. There also is an upper limit for the size of the new data received that is enforced by down-sampling if exceeded. A learner's model is updated using the concatenation of the full set of manually labelled data (before sampling) and the predictions received from teachers. If no predictions are received a learner's model is not updated. Tri-training stops when no model is updated.

Chen et al. (2006) apply tri-training to a sequence labelling task, namely chunking, and discuss sentence-level instance selection as a deviation from vanilla tri-training. They propose a "two agree one disagree method" in which the learner only accepts a prediction from its teachers when it disagrees with the teachers. Søgaard (2010) reinvents this method and coins the term *tri-training with disagreement* for it.

Li and Zhou (2007) extend tri-training to more than three learners and relax the requirement that all teachers must agree by using their ensemble prediction. They apply this to an ensemble of decision trees, i. e. a random forest, and call the method *co-forest*. As to the risk of deterioration of performance due to wrong labelling decisions, they point to previous work showing that the effect can be compensated with a sufficient amount of data if certain conditions are met, and they include these conditions in the co-forest algorithm.

Guo and Li (2012) identify issues with the update criterion of tri-training and with the estimation of error rates on training data and propose two modified methods, one improving performance in 19 of 33 test cases (eleven tasks and three learning algorithms) and the other improving performance in 29 of 33 cases.

Fazakis et al. (2016) compare self-, co- and tri-training combined with a selection of machine learning algorithms on 52 datasets and include a setting where self-training is carried out with logistic model trees, a type of decision tree classifier that has logistic regression models at its leaf nodes. Tri-training with C45 decision trees comes second in their performance ranking after self-training with logistic model trees. However, logistic model trees are not tested with co- or tri-training.

Chen et al. (2018) adjust tri-training to neural networks by sharing parameters between learners for efficiency. Furthermore, they add random noise to the automatic labels to encourage model diversity and to regularize the models and in addition to teacher agreement they require teacher predictions

made with dropout (as in training) to be stable.

Ruder and Plank (2018) also propose to share all but the final layers of a neural model between the three learners in tri-training for sentiment analysis and POS tagging. They add an orthogonality constraint on the features used by two of the three learners to encourage diversity. Furthermore, they apply multi-task training in tri-training and they modify the tri-training algorithm to exclude the manually labelled data from the training data of the third learner.

2.2 Tri-training in Dependency Parsing

Tri-training was first applied in dependency parsing by Søgaard and Rishøj (2010), who combine tri-training with stacked learning in multilingual graph-based dependency parsing. 100k sentences per language are automatically labelled using three different stacks of token-level classifiers for arcs and labels, resulting in state-of-the-art performance on the CONLL-X Shared Task (Buchholz and Marsi, 2006).

In an uptraining scenario, Weiss et al. (2015) train a neural transition-based dependency parser on the unanimous predictions of two slower, more accurate parsers. This can be seen as tri-training with one iteration and with just one learner's model as the final model. Similarly, Vinyals et al. (2015) use single iteration, single direction tri-training in constituency parsing where the final model is a neural sequence-to-sequence model with attention, which learns linearised trees.

2.3 Comparing Cross-view Training and Pretraining in NLP

The only previous work we know of that compares pretrained contextualised word embeddings to another semi-supervised learning approach is the work of Bhattacharjee et al. (2020) who compare three BERT models (Devlin et al., 2019) and a semi-supervised learning method for neural models in three NLP tasks: detecting the target expressions of opinions, named entity recognition (NER) and slot labelling, i. e. populating attributes of movies given reviews of movies. The semi-supervised learning method is cross-view training (Clark et al., 2018), which adds auxiliary tasks to a neural network that are only given access to restricted views of the input, similarly to the learners in co-training, e. g. a view may be the output of the forward LSTM in a Bi-LSTM. The auxiliary tasks are trained to agree with the prediction of the main classifier on unlabelled data. Cross-view training performs best in NER and slot labelling in Bhattacharjee et al. (2020)'s experiments and comes second and third place on two test sets in opinion target expression detection.

3 Experimental Setup

This section describes the technical details of the experimental setup.

3.1 Tri-Training Algorithm

We provide an overview of our tri-training algorithm.[1] Before the first tri-training iteration, three samples of the labelled data are taken and initial models are trained on them. Each tri-training iteration compiles three sets of automatically labelled data, one for each learner, feeding predictions that two learners agree on to the third learner.[2] In case all three learners agree, we randomly pick a receiving learner.[3] At the end of each tri-training iteration, the three models are updated with new models trained on the concatenation of the manually labelled and automatically labelled data selected for the learners.

3.2 Parameter Selection

We explore three tri-training parameters:

- A: the amount of automatically labelled data combined with labelled data when updating a model at the end of a tri-training iteration
- T: the number of tri-training iterations

[1] Full pseudocode is provided in Appendix A. We share our source code, basic documentation and training log files (including development and test scores of each learner for all iterations) on https://github.com/jowagner/mtb-tri-training.

[2] We require all predictions (lemmata, universal and treebank-specific POS tags, morphological features, dependency heads and dependency labels including language-specific subtypes) for all tokens of a sentence to agree. The main reason is simplicity: The parser UDPipe-Future expects training data with all predictions as it jointly trains on them (multi-task learning). If we allowed disagreement between teachers on some of the tag columns we would have to come up with a heuristic to resolve such disagreements, complicating the experiment. Furthermore, we hypothesise that full agreement increases the likelihood of the syntactic prediction to be correct. The agreement can be seen as a confidence measure or quality filter.

[3] Restricting the knowledge transfer to a single learner is a compromise between vanilla tri-training, which lets all three learners learn from unanimous predictions, and tri-training with disagreement (Chen et al., 2006), which lets none of the learners learn from such predictions. Furthermore, this modification (together with rejecting duplicates while sampling the unlabelled data) increases diversity of the sets and therefore may help keeping the learners' models diverse.

- d: how much weight is given to data from previous iterations. The current iteration's data is always used in full. No data from previous iterations is added with $d = 0$. For $d = 1$, all available data is concatenated. With $d < 1$, we apply exponential decay to the dataset weights, e. g. for $d = 0.5$ we take 50% of the data from the previous iteration, 25% from the iteration before the last one, etc.

For a fair comparison of tri-training with and without word embeddings, we take care that A, T and d are explored equally well in both settings and that each comparison is based on results for the same set of parameters. Based on the observations in Appendices B.2 to B.4 and balancing accuracy, number of runs and computational costs, we perform for each language and parser twelve runs:

- one run with $A = 40$k, $T = 12$ and $d = 1$
- one run with $A = 80$k, $T = 8$ and $d = 1$
- two runs with $A = 80$k, $T = 8$ and $d = 0.5$
- two runs with $A = 160$k, $T = 4$ and $d = 0.5$
- the above six runs in a variant where the seed data is oversampled to match the size of unlabelled data for the model updates at the end of each tri-training iteration.

For runs with multilingual BERT, we use $d \in \{0.5, 0.71\}$ instead of $d \in \{0.5, 1\}$ to reduce computational costs.

3.3 Choice of Languages

Since we focus on low-resource languages, we select the three treebanks with the smallest amount of training data from UD v2.3, meeting the following criteria:

- The treebank has a development set.
- An ELMoForManyLangs model (Section 3.5) is available for the target language. Sign languages and transcribed spoken treebanks are not covered.
- Surface tokens are included in the public UD release.

The treebanks selected are Hungarian `hu_szeged` (Vincze et al., 2010), Uyghur `ug_udt` (Eli et al., 2016), and Vietnamese `vi_vtb` (Nguyen et al., 2009). Furthermore, we include English in a simulated low-resource setting using a sample of 1226 sentences (20149 tokens) from the English Web Treebank `en_ewt` (Silveira et al., 2014). Table 1 shows each treebank's training data size and the size of unlabelled data. The sizes of the labelled training data are in a narrow range of 19.3 to 20.3 thousand tokens.

3.4 Unlabelled Data

To match the training data of the word embeddings (Section 3.5), we use the Wikipedia and Common Crawl data of the CoNLL 2017 Shared Task in UD Parsing (Ginter et al., 2017; Zeman et al., 2017) as unlabelled data in tri-training. We downsample the Hungarian data to 12%, the Vietnamese data to 6% and the English data to 2% of sentences to reduce disk storage requirements. All data, including data for Uyghur, is further filtered by removing all sentences with less than five or more than 40 tokens[4] and the order of sentences is randomised. We then further sample the unlabelled data in each tri-training iteration to a subset of fixed size to limit the parsing and training costs. The last two columns of Table 1 show the size of the unlabelled data sets after filtering and sampling.[5]

3.5 Parser and Word Embeddings

For the parsing models of the individual learners in tri-training, we use UDPipe-Future (Straka, 2018). This parser jointly predicts parse tree, lemmata, universal and treebank-specific POS tags and morphological features. Since its input at predict time is just tokenised text, it can be directly applied to unlabelled data while still exploiting lemmata and tags annotated in the labelled data to obtain strong models. We use UDPipe-Future in two configurations:

- **udpf**: UDPipe-Future with internal word and character embeddings only. This parser is for semi-supervised learning via tri-training only, i. e. the unlabelled data only comes into play through tri-training. The parser's word em-

[4] In preliminary experiments with the English LinEs treebank and without a length limit, learners rarely agree on predictions for longer sentences. This means that long sentences are unlikely to be selected by tri-training as new training data and the increased computational costs of parsing long sentences does not seem justified. We also exclude very short sentences as we do not expect them to feature new syntactic patterns and, if they do, to not provide enough context to infer the correct annotation.

[5] These numbers do not reflect the removal of sentences that contain one or more tokens that have over 200 bytes in their UTF-8-encoded form and de-duplication performed before parsing unlabelled data.

	Labelled						Unlabelled	
	Training		Development		Test		(filtered and sampled)	
Language	Tokens	Sent.	Tokens	Sent.	Tokens	Sent.	Tokens	Sentences
English	20,149	1,226	25,148	2,002	25,096	2,077	153,878,772	10,275,582
Hungarian	20,166	910	11,418	441	10,448	449	168,359,253	12,199,371
Uyghur	19,262	1,656	10,644	900	10,330	900	2,537,468	217,950
Vietnamese	20,285	1,400	11,514	800	11,955	800	189,658,820	12,634,409

Table 1: Data statistics (UDPipe sentence splitting and tokenisation for the unlabelled data; right-most columns are for the unlabelled data used in tri-training; for the ELMo and FastText training data, see Section 3.5)

beddings are restricted to the labelled training data.

- **fasttext**: This parser is UDPipe-Future with FastText word embeddings (Bojanowski et al., 2017). It is included to be able to tell how much of performance differences of the following two parsers is due to the inclusion of FastText word embeddings.
- **elmo**: This parser combines FastText word embeddings with ELMo (Peters et al., 2018) contextualised word embeddings. This parser is for semi-supervised learning via training word embeddings on unlabelled data and via a combination of word embeddings and tri-training.
- **mbert**: This parser combines FastText word embeddings with multilingual BERT[6] (Devlin et al., 2019) contextualised word embeddings, pre-trained on Wikipedia in just over 100 languages including three of the four languages of our experiments.[7] We include this parser to verify that our findings carry over to a transformer architecture.

UDPipe-Future employs external word representations without fine-tuning the respective models, in our case FastText, ELMo and BERT. Following Straka et al. (2019) we use a fixed vocabulary of the one million most frequent types with FastText. We train FastText on the full CoNLL'17 data for each language separately, i. e. 9.4 billion tokens for English, 1.6 billion tokens for Hungarian, 3.0 million tokens for Uyghur and 4.1 billion tokens for Vietnamese. FastText's feature to produce new word vectors for unseen words is not used. For ELMo, we use the ELMoForManyLangs[8] models provided by Che et al. (2018). They limit training to "20 million words", i. e. 0.2% of the English data, 1.2% of the Hungarian data, 100% of the Uyghur data and 0.5% of the Vietnamese data. Multilingual BERT is trained on Wikipedia only, presumably on a newer data dump than the one used for the Wikipedia part of the CoNLL'17 data.

An unusual feature of UDPipe-Future is that it oversamples the training data to 9,600 sentences in each epoch if the training data is smaller than that. This automatic oversampling enables the parser to perform well on most UD treebanks without tuning the number of training epochs. In our experiments, this behaviour will be triggered in settings with a low or medium augmentation size A, except when data of previous iterations is combined ($d > 0$) and the number of iterations T is not small.[9]

We make a small modification to the default learning rate schedule of UDPipe-Future, softening its single large step from 0.001 to 0.0001 to five smaller steps, keeping the initial and the final learning rate.[10] The seed for pseudo-random initialisation of the neural network of the parser is derived from the seed that randomises the sampling of data in each tri-training experiment, an identifier of the tri-training parameters, an indicator whether the run is a repeat run, the learner number i and the tri-training iteration t.

[6] https://github.com/google-research/bert/blob/master/multilingual.md

[7] To increase parser diversity in tri-training, each tri-training learner uses different BERT layers and subword unit vector pooling methods, see Appendix F.

[8] https://github.com/HIT-SCIR/ELMoForManyLangs

[9] With $d > 0$, the amount of training data grows with each iteration (up to a limit for $d < 1$). For example, in the third tri-training iteration for Hungarian with mBERT, $d = 0.71$ and $A = 40$k, learner 1 receives 3749 sentences (39986 tokens) from the current iteration, 2744 sentences from iteration 2, 2038 sentences from iteration 1 and 2275 sentences from the labelled data (2.5 times 910 sentences), totalling in 10806 sentences which is above UDPipe-Future's oversampling threshold of 9600 sentences.

[10] We run UDPipe-Future with the option `-epochs 30:1e-3,5:6e-4,5:4e-4,5:3e-4,5:2e-4,10:1e-4`

3.6 Ensemble Method for Candidate Models

Candidate models for the final model are created at each tri-training iteration by combining the current models of the three learners in an ensemble using linear tree combination (Attardi and Dell'Orletta, 2009) as implemented[11] by Barry et al. (2020).[12] Candidate ensembles are evaluated on development data using the CoNLL'18 evaluation script.

At tri-training iteration zero, i. e. before any unlabelled data is used, runs with the same language and parser only differ in the random initialisation of models. A large number of additional ensembles can therefore be built, picking a model for each learner from different runs. These ensembles behave like ensembles from new runs, allowing us to study the effect of random initialisation using a much more accurate estimate of the LAS distribution than possible with ensembles of each runs. We obtain 4096 LAS values for each language and parser as follows:

1. For each learner $i \in \{1, 2, 3\}$, we partition the available models into 16 buckets in order of development LAS.
2. We enumerate all $16^3 = 4096$ combinations of buckets, and from each bucket combination, we sample one combinations of three models, one model per bucket.
3. The selected model combination is combined using the linear tree combiner and evaluated.

4 Development Set Results

We compare semi-supervised learning with tri-training to semi-supervised learning with a combination of context-independent and contextualised word embeddings, and we compare these two approaches to combining them and to training without unlabelled data, i. e. supervised learning, addressing the three questions posed in Section 1.

As described in Section 3.2, we obtain twelve LAS scores from tri-training with each parser and language. Tri-training with T iterations can choose an ensemble from $T + 1$ ensembles (Appendix E). To give the baselines the same number of models to choose from, we sample from the baseline ensembles described in Section 3.6. For each tri-training run with T iterations, we select the best score from $T+1$ baseline scores. As the choice of the subset of $T + 1$ scores is random, we repeat the random sampling 250,000 times to get an accurate estimate of the LAS distribution. In other words, we simulate what would happen if the additional data obtained through tri-training had no effect on the parser.

Figure 1 compares the parsing performance with and without tri-training for the four development languages and for the three types of parsing models **udpf**, **elmo** and **mBERT**. Most distributions for each language are clearly separated and the order of methods is the same: both tri-training and external word embeddings yield clear improvements. External word embeddings have a much stronger effect than tri-training. In combination, the two semi-supervised learning methods yield a small additional improvement with an average score difference of over half an LAS point.

5 Error Analysis

In this section, we probe, using the development sets, how the error distribution changes as we add tri-training and/or word ELMo embeddings trained on unlabelled data to the basic parser. We compare the following

1. tokens that are out-of-vocabulary relative to the manually labelled training data versus those that are in the training data ("OOV"/"IV")
2. different sentence length distributions: up to 9, 10 to 19, 20–39, and 40 or more tokens
3. different dependency labels, e.g is there a marked difference in the effect of tri-training or word embeddings for particular label types, e.g. `nsubj`?

How does tri-training help (with no embeddings)? As expected, tri-training brings a clearly greater improvement for OOVs than IVs for all four languages. The role of sentence length is not consistent across languages. For English, tri-training helps most on longer sentences ($>$ 20 words), for Hungarian, short sentences ($<$ 10 words), and for Uyghur, very long sentences ($>$ 40 words). Sentence length does not appear to be a factor for Vietnamese. Regarding dependency labels, there are no clear pattern across languages.[13]

[11] https://github.com/jowagner/ud-combination

[12] As we observed that performance of the ensembles on the development data varies considerably with the random initialisation of the tie breaker in the combiner's greedy search, e. g. obtaining LAS scores from 75.52 to 75.62 for ten repetitions, we run the combiner 21 times with different initialisation and report average LAS.

[13] See Table 7 in Appendix C.

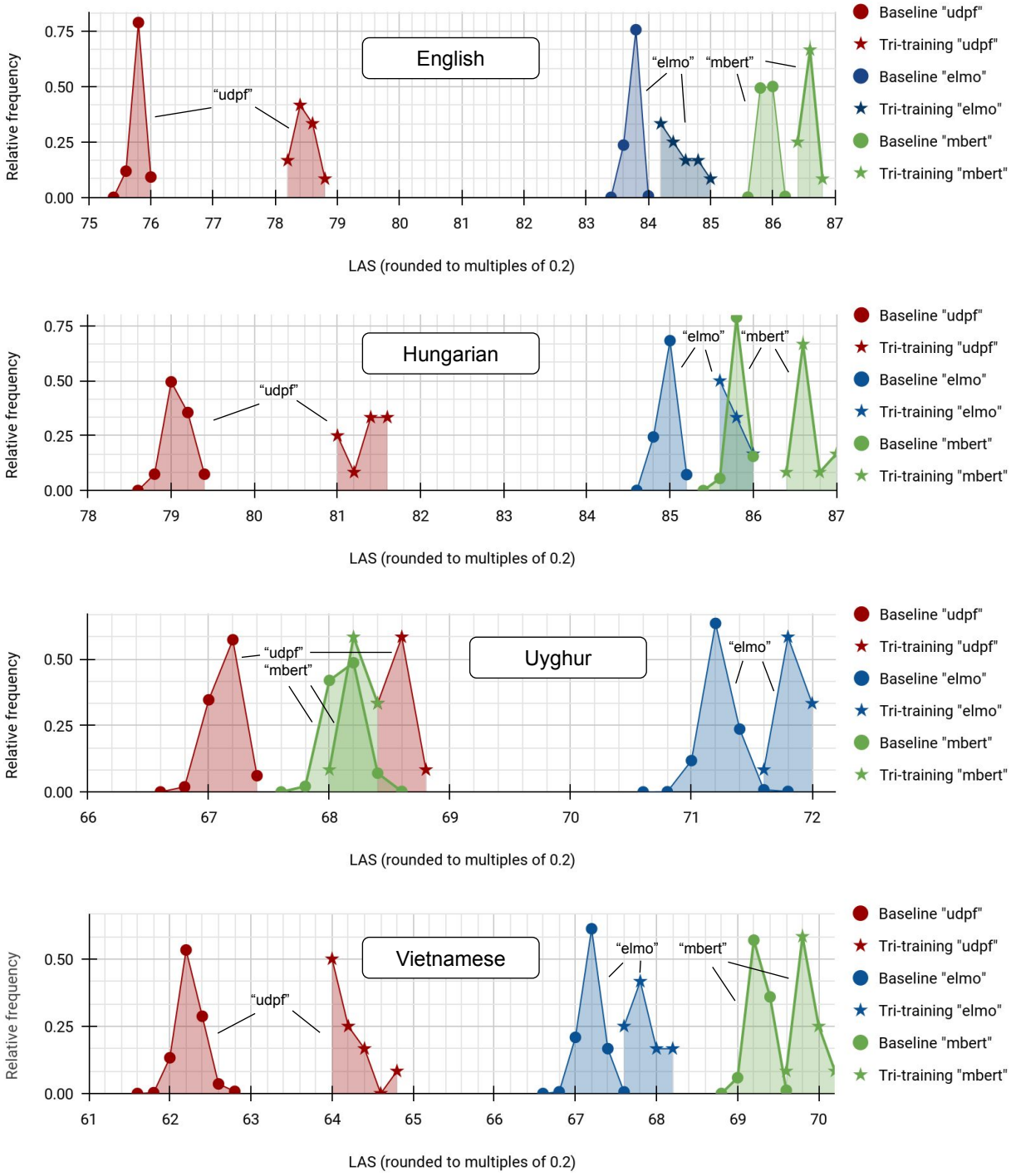

Figure 1: Effects of external word embeddings and tri-training on the LAS distribution: The baseline distributions are based on a large number of ensemble LAS scores. Each tri-training distribution is based on twelve LAS scores.

How do word embeddings help (with no tri-training)? Analysis of improvement by sentence length and by OOV status show similar trends to the tri-training improvements described above. Across languages, the use of pretrained embeddings helps to correctly identify the `flat` relation (which is used in names and dates).

6 Test Set Results

In this section, we verify to what extent our main observations on development data carry over to test data and include results for a parser using only FastText as external word embedding. For each of the LAS distributions using tri-training in Figure 1 and the distribution for **fasttext** not shown, we select the ensemble with highest development LAS for testing. Since we also use model selection based on development LAS to choose the final model of each tri-training run from its $T + 1$ iterations, the best model is selected from a set of 50 models given the values of T listed in Section 3.2, exceeding the number of baseline models available from iteration 0. For a fair comparison, we therefore leverage the 4096 baseline ensembles described in Section 3.6. As a choice of 50 out of 4096 ensembles would introduce noise, we repeatedly draw samples, for each sample find the best model according to development LAS, obtain test LAS and report the average LAS over all samples, i. e. the expectation value. As was the case for development results, we run the linear tree combiner 21 times on the three individual predictions of the tri-training learners and take the average LAS over all combiner runs as the score of the ensemble.

Table 2 shows development and test set LAS for the models selected as described above. The test set results confirm the development result that a combination of tri-training and contextualised word embeddings consistently gives the best results and that the individual methods improve performance. In keeping with the development results, contextualised word embeddings yield higher gains than tri-training. The test results confirm the development observation that multilingual BERT does not work as well as language-specific ELMo for Uyghur, a zero-shot language for multilingual BERT.

7 Conclusion

We compared two semi-supervised learning methods in the task of dependency parsing for three low-resource languages and English in a simulated low-resource setting. Tri-training was effective but could not come close to the performance gains of contextualised word embeddings. Combined, the two learning methods achieved small additional improvements between 0.2 LAS points for Uyghur and 1.3 LAS points for Vietnamese. Whether these gains can justify the additional costs of tri-training will depend on the application.

We recommend that users of tri-training vary settings and repeat runs to find good models. Future work could therefore explore how to best combine the many models or the large amount of automatically labelled data that such experiments produce. To obtain a fast and strong final model, a combination of ensemble search and model distillation or up-training may be the next step. Integrating cross-view training (Clark et al., 2018) into tri-training may also be fruitful similarly to the integration of multi-view learning in co-training (Lim et al., 2020). The requirement of tri-training that two teachers must agree changes the sentence length distribution of the data selected and may introduce other biases. Future work could try to counter this effect be re-sampling the predictions similarly to how Droganova et al. (2018) corrected for such effects in self-training.

While our literature review suggests that tri-training performs better than co- and self-training, it would be interesting how these methods compare under a fixed computation budget as the latter methods train fewer parsing models per iteration.

8 Ethics and Broader Impact

Tri-training uses much smaller amounts of unlabelled data than the state-of-the-art semi-supervised method of self-supervised pre-training and we therefore do not expect tri-training to add new risks from undesired biases in the unlabelled data. The use of tri-training may, however, pose new challenges in detecting problematic effects of issues in unlabelled data as existing inspection methods may not be applicable.

An individual tri-training run with FastText and multilingual BERT word embeddings and $A = 80k$, $T = 8$ and $d = 0.5$ typically takes three days on a single NVIDIA GeForce RTX 2080 Ti GPU. Overall, we estimate that our experiments took 2500 GPU days. This large GPU usage stems from the exploration of tri-training parameters in Appendix B. Future work can build on our observations and thereby reduce computational costs.

	udpf		fasttext		elmo		mbert	
Language	**B**	**TT**	**B**	**TT**	**B**	**TT**	**B**	**TT**
Development								
English	75.8	78.8	78.1	80.6	83.7	84.9	85.9	**86.7**
Hungarian	79.0	81.7	80.9	83.0	85.2	86.1	85.9	**86.9**
Uyghur	67.3	68.8	68.4	69.7	71.2	**72.1**	68.1	68.4
Vietnamese	62.3	64.8	64.0	65.6	67.3	68.3	69.3	**70.2**
Test								
English	76.6	79.3*****	78.9	80.8*****	84.0	84.9*****	85.7	**86.0****
Hungarian	77.6	79.2*****	79.8	81.2*****	84.3	84.9**	85.5	**86.3*****
Uyghur	66.0	67.7*****	66.8	68.2*****	69.8	**70.6*****	67.4	67.5
Vietnamese	61.3	62.6****	62.5	63.1*	65.6	66.8****	69.3	**69.9***

Table 2: Development and test set LAS for selected models (best of 12 according to development LAS); **B** = baseline, **TT** = tri-training; statistical significance of tri-training improvement over baseline (McNemar test; carried out for test set results only): one star for $p \leq 0.05$, two stars for $p \leq 0.01$, three stars for $p \leq 0.001$, four stars for $p \leq 0.0001$ and five stars for $p \leq 0.00001$.

Acknowledgements

This research is supported by Science Foundation Ireland (SFI) through the ADAPT Centre for Digital Content Technology, which is funded under the SFI Research Centres Programme (Grant 13/RC/2106) and is co-funded under the European Regional Development Fund, and through the SFI Frontiers for the Future programme (19/FFP/6942).

References

Giuseppe Attardi and Felice Dell'Orletta. 2009. Reverse revision and linear tree combination for dependency parsing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 261–264, Boulder, Colorado, USA. Association for Computational Linguistics.

James Barry, Joachim Wagner, and Jennifer Foster. 2020. The ADAPT enhanced dependency parser at the IWPT 2020 shared task. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 227–235, Online. Association for Computational Linguistics.

Kasturi Bhattacharjee, Miguel Ballesteros, Rishita Anubhai, Smaranda Muresan, Jie Ma, Faisal Ladhak, and Yaser Al-Onaizan. 2020. To BERT or not to BERT: Comparing task-specific and task-agnostic semi-supervised approaches for sequence tagging. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7927–7934, Online. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City. Association for Computational Linguistics.

Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the The Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 598–603, Providence, Rhode Island, USA. The AAAI Press, Menlo Park, California, USA.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. 2018. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics.

Dong-Dong Chen, Wei Wang, Wei Gao, and Zhi-Hua Zhou. 2018. Tri-net for semi-supervised deep learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, pages 2014–2020. International Joint Conferences on Artificial Intelligence Organization.

Wenliang Chen, Yujie Zhang, and Hitoshi Isahara. 2006. Chinese chunking with tri-training learning. In *Computer processing of oriental languages, Beyond the Orient: The Research Challenges Ahead, 21st International Conference, ICCPOL 2006*, volume 4285, pages 466–473. Springer-Verlag Berlin Heidelberg, Germany.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018. Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925, Brussels, Belgium. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Kira Droganova, Filip Ginter, Jenna Kanerva, and Daniel Zeman. 2018. Mind the gap: Data enrichment in dependency parsing of elliptical constructions. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 47–54, Brussels, Belgium. Association for Computational Linguistics.

Marhaba Eli, Weinila Mushajiang, Tuergen Yibulayin, Kahaerjiang Abiderexiti, and Yan Liu. 2016. Universal dependencies for Uyghur. In *Proceedings of the Third International Workshop on Worldwide Language Service Infrastructure and Second Workshop on Open Infrastructures and Analysis Frameworks for Human Language Technologies (WLSI/OIAF4HLT2016)*, pages 44–50, Osaka, Japan. The COLING 2016 Organizing Committee.

Nikos Fazakis, Stamatis Karlos, Sotiris Kotsiantis, and Kyriakos Sgarbas. 2016. Self-trained LMT for semisupervised learning. *Computational Intelligence and Neuroscience*, 2016.

Filip Ginter, Jan Hajič, Juhani Luotolahti, Milan Straka, and Daniel Zeman. 2017. CoNLL 2017 shared task - automatically annotated raw texts and word embeddings. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Tao Guo and Guiyang Li. 2012. Improved tri-training with unlabeled data. In *Software Engineering and Knowledge Engineering: Theory and Practice*, pages 139–147, Berlin, Heidelberg. Springer Berlin Heidelberg.

M. Li and Z. Zhou. 2007. Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(6):1088–1098.

K. Lim, J. Y. Lee, J. Carbonell, and T. Poibeau. 2020. Semi-supervised learning on meta structure: Multi-task tagging and parsing in low-resource scenarios. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34:(05), pages 8344–8351.

David McClosky, Eugene Charniak, and Mark Johnson. 2006a. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of [sic!] Computational Linguistics (HLT-NAACL 06)*, pages 152–159, New York City, USA. Association for Computational Linguistics.

David McClosky, Eugene Charniak, and Mark Johnson. 2006b. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING ACL 06)*, pages 337–344, Sydney, Australia. Association for Computational Linguistics.

Phuong-Thai Nguyen, Xuan-Luong Vu, Thi-Minh-Huyen Nguyen, Van-Hiep Nguyen, and Hong-Phuong Le. 2009. Building a large syntactically-annotated corpus of Vietnamese. In *Proceedings of the Third Linguistic Annotation Workshop (LAW III)*, pages 182–185, Suntec, Singapore. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, USA. Association for Computational Linguistics.

Tieyun Qian, Bing Liu, Li Chen, and Zhiyong Peng. 2014. Tri-training for authorship attribution with limited training data. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 345–351, Baltimore, Maryland, USA. Association for Computational Linguistics.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Sebastian Ruder and Barbara Plank. 2018. Strong baselines for neural semi-supervised learning under domain shift. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1044–1054, Melbourne, Australia. Association for Computational Linguistics.

Kenji Sagae. 2010. Self-training without reranking for parser domain adaptation and its impact on semantic role labeling. In *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, pages 37–44, Uppsala, Sweden. Association for Computational Linguistics.

Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor,

John Bauer, and Christopher D. Manning. 2014. A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2897–2904, Reykjavik, Iceland. European Language Resources Association (ELRA).

Anders Søgaard. 2010. Simple semi-supervised training of part-of-speech taggers. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 205–208, Uppsala, Sweden. Association for Computational Linguistics.

Anders Søgaard and Christian Rishøj. 2010. Semi-supervised dependency parsing using generalized tri-training. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1065–1073, Beijing, China. Coling 2010 Organizing Committee.

Mark Steedman, Miles Osborne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. 2003. Bootstrapping statistical parsers from small datasets. In *10th Conference of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary. Association for Computational Linguistics.

Milan Straka. 2018. UDPipe 2.0 prototype at CoNLL 2018 UD shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 197–207, Brussels, Belgium. Association for Computational Linguistics.

Milan Straka, Jana Straková, and Jan Hajič. 2019. Evaluating contextualized embeddings on 54 languages in POS tagging, lemmatization and dependency parsing. ArXiv 1908.07448v1.

Veronika Vincze, Dóra Szauter, Attila Almási, György Móra, Zoltán Alexin, and János Csirik. 2010. Hungarian dependency treebank. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2773–2781. Curran Associates, Inc.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. 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 323–333, Beijing, China. Association for Computational Linguistics.

Xiang Yu, Ngoc Thang Vu, and Jonas Kuhn. 2020. Ensemble self-training for low-resource languages: Grapheme-to-phoneme conversion and morphological inflection. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 70–78, Online. Association for Computational Linguistics.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Droganova, Héctor Martínez Alonso, Çağrı Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 2017. CoNLL 2017 shared task: Multilingual parsing from raw text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada. Association for Computational Linguistics.

Zhi-Hua Zhou and Ming Li. 2005. Tri-training: exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1529–1541.

A Tri-training Algorithm

Algorithm 1 shows the tri-training algorithm in the form we use it in this work. An extended version of the description in Seciton 3.1 follows.

Lines 1–3 Before the first tri-training iteration, three samples B_i of the labelled data L are taken and initial models h_i are trained on them ($i \in \{1, 2, 3\}$). We sample without replacement and with a target size 2.5 times the size of L, repopulating the sampling urn each time it becomes empty.[14]

Lines 5–7 For each tri-training iteration, we further sample a de-duplicated subset U' of the unlabelled data as processing all available unlabelled

[14] Zhou and Li (2005) sample with replacement. Section B.1 motivates our choice.

Algorithm 1: Tri-training in this work.

Input: L: Labelled data
U: Unlabelled data
A: Maximum number of items to add per iteration and learner
T: Number of tri-training iterations
d: Decay parameter, $0 \leq d \leq 1$
Output: Models $\{h_1, h_2, h_3\}$ for ensemble.

1 **for** $i \in \{1, 2, 3\}$ **do**
2 $\quad B_i \leftarrow \text{Sample}(L, \text{ size} = 2.5 \times |L|)$
3 $\quad h_i \leftarrow \text{Learn}(B_i)$
4 **end for**
5 **for** $t = 1$ *to* T **do**
6 $\quad U' \leftarrow \text{Sample}(U, \text{ size } = 16 \times A),$
7 $\qquad \text{reject\_duplicates } = \textit{True})$
8 $\quad L_{t,i} \leftarrow \{\}, \quad i = 1, 2, 3$
9 $\quad$ **for** $x \in U'$ **do**
10 $\quad\quad$ **if** $h_j(x) = h_k(x), j \neq k$ **then**
11 $\quad\quad\quad$ **if** $h_1(x) = h_2(x) = h_3(x)$ **then**
12 $\quad\quad\quad\quad C \leftarrow \{1, 2, 3\}$
13 $\quad\quad\quad$ **else**
14 $\quad\quad\quad\quad C \leftarrow \{1, 2, 3\} \setminus \{j, k\}$
15 $\quad\quad\quad$ **end if**
16 $\quad\quad\quad i \leftarrow \text{RandomChoice}(C)$
17 $\quad\quad\quad L_{t,i} \leftarrow L_{t,i} \cup \{(x, h_j(x))\}$
18 $\quad\quad$ **end if**
19 $\quad$ **end for**
20 $\quad$ **for** $i \in \{1, 2, 3\}$ **do**
21 $\quad\quad$ **if** $|L_{t,i}| > A$ **then**
22 $\quad\quad\quad L_{t,i} \leftarrow \text{Sample}(L_{t,i}, \text{ size } = A)$
23 $\quad\quad$ **end if**
24 $\quad\quad R \leftarrow \bigcup_{t'=1}^{t} \text{Sample}(L_{t',i}, \text{size } =$
25 $\quad\quad\quad\quad \min\{|L_{t',i}|, A \times d^{t-t'}\})$
26 $\quad\quad h_i \leftarrow \text{Learn}(B_i \cup R)$
27 $\quad$ **end for**
28 **end for**

data would not be practical for most languages in our experiments (Section 3.4).[15, 16]

Lines 5, 8–18 Each tri-training iteration t compiles three sets of automatically labelled data $L_{t,i}$ one for each learner i, feeding predictions that two learners agree on[17] to the third learner (lines 10–17). In case all three learners agree, we randomly pick a receiving learner.[18] While Zhou and Li (2005) do not state in their pseudo code that j and k must be different, this is clear from their description.

Lines 21 and 22 We limit the size of the data sets $L_{t,i}$ to A, downsampling them if needed.

Lines 19, 23–35 While Zhou and Li (2005) update the models h_i by directly training on $L \cup L_{t,i}$, we experiment with concatenating data from previous tri-training iterations (lines 24–25) and we use B_i instead of L (line 26). [19] The parameter d controls how much weight is given to data from previous iterations. The current iteration's data is always used in full. No data from previous iterations is added with $d = 0$. For $d = 1$, all available data is concatenated. With $d < 1$, we apply exponential decay to the dataset weights, e. g. for $d = 0.5$ we take 50% of the data from the previous iteration, 25% from the iteration before the last one etc. (line 25). [20] At the end of each tri-training iteration in (line 26), the three models h_i are updated with new models trained on the concatenation of the manually labelled and automatically labelled data selected for the learners.[21]

[15] In Zhou and Li (2005)'s experiments, all datasets are small, that largest having 3772 items.

[16] We set the size of U' so that we do not expect it to be a limiting factor. In preliminary experiments with the English LinEs treebank, we observed that $4A$ is sufficient to obtain at least A new labelled items. We set the size of U' to $16A$ to account for likely variation in the rate of agreement between models when switching to other treebanks.

[17] See Footnote 2.

[18] See Footnote 3.

[19] Initially, the latter was an error on our side but, given that we ensure that each item in L is included in each B_i at least twice, keeping the B_i seems better fitted as L no longer provides additional labelled data, the diversity of learner models is improved and moderate oversampling of L can be expected to be helpful.

[20] We do not restrict experiments to a single value of d as tri-training is considerably faster with $d \in \{0, 0.5\}$ than for $d = 1$, see Section B.2.

[21] For the reasons described in Section E, we do not use the model update conditions of Zhou and Li (2005) that are based on *(a)* the estimated label noise in R to be lower than in the previous iteration and on *(b)* R to be sufficiently big for the noise not to be harmful under certain assumptions.

Our training.conllu files produced by tri-training for each parsing model start with the manually labelled data followed by the automatically labelled data. In case of oversampling B_i to match the size of R (option not mentioned in the description above), the oversampling also changes the order of the manually labelled data. Similarly, the $L_{t',i}$ are only re-ordered if $|L_{t',i}| > A \times d^{t-t'}$.

For clarity, when we use the set union operator in Algorithm 1 we mean concatenation of data sets. Duplicates are not removed. It is also clear that vanilla tri-training concatenates sets as set operations in the mathematical sense would damage the samples with replacement of manually labelled data.

B Parameter Search

This section describes our parameter search, analysing four distinct aspects of tri-training: sampling of seed data, reusing data from previous iterations, sample size and oversampling.

B.1 Effect of Sampling of Seed Data

B.1.1 Seed Sampling Methods Considered

The seed data B_i in tri-training is the labelled data that is used to train the initial models. This data is also included in the training data of the remaining tri-training iterations in our version of tri-training, see Algorithm 1. Each learner usually uses a different sample of the original training data.

In initial experiments with the English side of the LinES Parallel Treebank (`en_lines`) as seed data, we observed a degradation of performance of the learners' models when sampling the manually labelled data with replacement – as in vanilla tri-training (Zhou and Li, 2005) – compared to models trained directly on the labelled data. Neither combining three models in an ensemble nor additional training data obtained through tri-training in up to two tri-training iterations compensated for the loss of performance.

The reason why vanilla tri-training uses sampling is to ensure variation between the three learners. Neural models, however, naturally vary due to random initialisation of network weights, order of training data, stochastic kernels and numerical effects when intermediary results computed in parallel are combined in unpredictable order. We therefore tried using the original manually labelled training data in all learners and relying on random initialisation to instill variation. This removed the degradation of performance but as tri-training proceeded performance stayed within 0.6 LAS points of the average LAS of ensembles of three initial models. We suspected that more variation is needed. Therefore, we re-introduced sampling but modified it to ensure that all manually labelled data is available to each learner. We change the sampling to pick half of the data twice and the remaining half three times, resulting in a sample size of 250% of the original data.[22] With this sampling, tri-training performance clearly improved in the `en_lines` experiment and exceeded the range of results due to random initialisation and other sources of variation in neural models.

The results for our four development languages shown in Table 3 mostly confirm these findings. Using 2.5 copies consistently gives the highest LAS, though the improvements over vanilla tri-training, which uses sampling only for the initial models and then continues with the full labelled data, and a variant using the full data from the start are small.

B.1.2 Seed Sampling Results

B.2 Effect of Using Data from Previous Iterations

The tri-training parameter d controls how much data from previous tri-training iterations is used in the current iteration. We experiment with $d \in \{0, 0.5, 1\}$ as we expect that training a model on data obtained with different models, initialised with different seeds, may have similar benefits as using ensemble predictions, which Yu et al. (2020) show to improve self-training. Furthermore, data combination may limit negative effects of an iteration with poorly performing models h_i.

The results are shown in Table 4. For all but the Uyghur parser **udpf**, i. e. without external word embeddings, we found the best development results when predictions of all tri-training iterations are combined. The difference in LAS to the combination method that exponentially reduces the size of data taken from previous iterations ($d = 0.5$) is small.

B.3 Effect of Sample Size A

The tri-training parameter A controls how much unlabelled data is combined with labelled data during

[22] We confirmed that UDPipe-Future does not employ methods for handling unknown words based on applying a frequency threshold on the training data as oversampling may interfere with such methods.

Lang.	Parser	W.R.	Vanilla	100%	250%
En	**udpf**	75.3	77.2	77.9	**78.0**
	elmo	82.9	83.8	84.0	**84.3**
Hu	**udpf**	77.9	80.3	80.6	**80.8**
	elmo	84.1	85.3	85.5	**85.5**
Ug	**udpf**	66.4	67.9	68.1	**68.3**
	elmo	70.5	71.4	71.6	**71.7**
Vi	**udpf**	61.6	63.7	63.7	**63.9**
	elmo	66.3	67.5	67.6	**67.6**
Average		73.13	74.63	74.87	**75.01**

Table 3: Effect of seed data sampling on tri-training performance (average development LAS over eight tri-training runs, selecting, for each run, the best tri-training iteration according to development LAS): W.R. is sampling with replacement, Vanilla uses sampling with replacement for the initial models and a full copy of the labelled data for $t > 0$, 100% uses a full copy of the labelled data in all iterations, i. e. the only source of variation is the random seed used in parser training, 250% uses 2.5 copies of L for each learner, providing additional variation due to the random selection of the last half of the data.

		d			
Lang.	**Parser**	0	0.5	0.71	1
En	**udpf**	77.9	78.3	78.4	**78.6**
	elmo	84.1	84.3	84.4	**84.9**
Hu	**udpf**	80.7	81.1	81.5	**81.5**
	elmo	85.5	85.7	**85.8**	85.8
Ug	**udpf**	68.3	**68.7**	68.6	68.6
	elmo	71.6	71.8	71.8	**72.0**
Vi	**udpf**	63.9	64.0	64.2	**64.5**
	elmo	67.3	67.7	**67.9**	67.8

Table 4: Effect of data combination across iterations on tri-training performance (average development LAS over multiple tri-training runs, selecting, for each run, the best tri-training iteration according to development LAS); two runs for each setting (language, parser, data combination method) with $A = 80$k, $T = 8$, seed data sampled as 250% of labelled data and $o \in \{\text{True}, \text{False}\}$.

training. Table 5 presents results for augmentation sizes A from 5k to 160k tokens.[23] We see good improvements for all development languages except Vietnamese as the size of the set of automatically labelled data added in each tri-training round increases. For Vietnamese with parser **elmo**, the range of scores is small and there is no consistent pattern.

B.4 Effect of Oversampling

Table 6 compares average LAS with and without oversampling of the manually labelled data B_i to match the size of the automatically labelled data R. The results suggest that the effects is negligible and since oversampling slows down training we carry out the main experiment without oversampling.[24]

C Error Analysis: Dependency Labels

Table 7 shows the most frequent LAS improvements by dependency label.

D Learning Rate Schedule

Table 8 compares the learning rate schedule we use with UDPipe-Future and its default schedule,

[23] For comparison, the labelled data L has about 20k tokens in our experiments and the samples B_i have about 50k tokens for our best seed data sample size 250%.

[24] Preliminary results for English with oversampling the manually labelled data three times in all iterations, including the seed models (Table 3), however, show a positive effect of oversampling. Maybe oversampling is more important in early iterations where the amount of automatically labelled data is relatively small. Future work should investigate the effect of oversampling further.

Language	Parser	*(i)* Small A 5k	10k	20k	*(ii)* Medium A 20k	40k	80k	*(iii)* Big A 80k	160k
English	**udpf**	77.5	77.9	**78.2**	77.4	77.8	**78.3**	78.0	**78.4**
	elmo	84.1	84.4	**84.6**	84.2	84.4	**84.5**	84.2	**84.4**
Hungarian	**udpf**	80.2	81.1	**81.1**	80.3	80.7	**81.1**	80.8	**81.3**
	elmo	85.3	85.7	**85.7**	85.3	85.4	**85.7**	85.6	**85.8**
Uyghur	**udpf**	67.5	67.9	**68.5**	67.6	67.9	**68.4**	68.2	**68.4**
	elmo	71.6	71.7	**71.7**	71.4	71.5	**71.7**	71.7	**71.9**
Vietnamese	**udpf**	63.6	63.7	**64.2**	63.4	63.8	**64.2**	63.9	**64.0**
	elmo	**67.8**	67.7	67.7	67.5	**67.6**	67.6	67.6	**68.0**

Table 5: Effect of augmentation size A on tri-training performance (average development LAS over multiple tri-training runs): *(i)* two runs with $T = 16$ and $d = 1$, *(ii)* three runs (Uygur) or four runs (other languages) with $T = 8$ and $d = 1$, *(iii)* six runs with $T = 4$ and $d = 0.5$.

Language	Parser	No	Yes	Δ
English	**udpf**	77.4	**77.7**	0.288
	elmo	84.0	**84.0**	0.055
Hungarian	**udpf**	80.2	**80.3**	0.117
	elmo	85.3	**85.3**	0.015
Uyghur	**udpf**	68.1	**68.1**	0.026
	elmo	**71.7**	71.7	-0.001
Vietnamese	**udpf**	**63.8**	63.8	-0.022
	elmo	**67.7**	67.7	-0.048

Table 6: Effect of oversampling the labelled data on tri-training performance (average development LAS over multiple tri-training runs)

E Model Selection

We select the tri-training iteration with the best ensemble performance according to development LAS. We do not use Zhou and Li (2005)'s stopping criterion that is based on conditionally updating the learners' models in line 26 of Algorithm 1 for the following reasons:

- The model update condition is designed for binary classification tasks. It is not clear how the condition would have to be updated for joint prediction of dependency trees, lemmata and multiple tags.
- The model update condition uses the training data to estimate label noise. We do not expect such estimates to be useful for neural models that tend to considerably overfit the training data.
- The model update condition rejects models trained on an amount of automatically labelled data that is too small to avoid harm from label noise under certain assumptions. In Zhou and Li (2005)'s experiments, the size of the unlabelled data is quite small.[25] In contrast, we can avail of orders of magnitude more unlabelled data. Hence, we do not expect the issue of insufficient data to arise.
- The inherent performance variation of neural models, e. g. due to random initialisation, can trigger Zhou and Li (2005)'s stopping criterion too early as it requires the error rate to drop in each iteration. When tri-training is run long enough for the improvements due to the additional training data to be smaller than the performance variation due to randomness in model training, we expect that patience is needed to bridge a temporary degradation.[26]
- Furthermore, tri-training can reduce performance if wrong decisions are amplified.

F BERT Layer Selection

We experiment with using different BERT layers and pooling functions for combining BERT's subword vectors to token vectors. We explore 45 settings for each language, nine choices of layers (individual layers and average of layers, excluding bottom layers) and five choices of token pooling functions. For each language, we choose three different settings, one for each learner in tri-training,

[25] Zhou and Li (2005)'s largest dataset has only 3772 items. The size of the unlabelled data never exceeds $3772 \times 0.75 \times 0.80 \approx 2263$ items.

[26] We do find variation in performance and performance recovery after a few iterations looking at a sample of tri-training runs. Future work can analyse our data for the trade-off between the lengths of patience and compute costs (or spending the same compute budget on more runs with lower patience each).

Parsers	English	Hungarian	Uyghur	Vietnamese
base udpf -> tri udpf	acl,fixed compound,xcomp parataxis,iobj	nummod,csubj cop,nsubj case,advmod	parataxis,mark obj,nummod aux,nsubj	compound,mark xcomp,cop case,csubj
base udpf -> base elmo	**flat**,discourse fixed,parataxis acl,appos	cop,advcl nsubj,acl **flat**,ccomp	appos,conj parataxis,**flat** fixed,nummod	amod,ccomp mark,obj discourse,compound
base elmo -> tri elmo	fixed,ccomp advcl,flat obl,punct	nummod,csubj advcl,appos acl,parataxis	discourse,appos cop,compound ccomp,obl	csubj,mark parataxis,compound amod,cop

Table 7: Top 6 largest LAS improvements by dependency type. Those shared by at least 3 languages are highlighted in bold. Labels with fewer than 20 occurrences are excluded.

Setting	Total Epochs	Learning Rate 0.001	0.0006	0.0004	0.0003	0.0002	0.0001
Parser default	60	40	0	0	0	0	20
In this work	60	30	5	5	5	5	10

Table 8: Learning rate schedule used in the experiments and the parser's default learning rate: number of epochs at each learning rate

starting with the top-performing setting, eliminating all settings with the same choice of layers or the same choice of token pooling and then repeating the process for the next learner.

Table 9 shows the results of this experiment. Our observations confirm that middle layers typically perform best (Rogers et al., 2020). Uyghur, for which multilingual BERT does not perform well with UDPipe-Future, has a different pattern showing no large differences and a preference for the top layer that is less informative for the other languages. Different languages seem to prefer different pooling functions for combining vectors of subword units to vectors for UD tokens. Table 9 shows our choices for each development language in bold.

English

Pooling \ Layer	08	09	10	11	12	A4	A4E	A5	A5E	Average
Avg	**85.5**	85.4	85.3	84.7	83.0	85.1	85.2	85.4	85.4	85.0
First	85.3	85.3	85.1	84.6	83.0	84.9	85.1	85.3	85.3	84.9
Last	85.4	85.4	85.2	84.6	83.0	85.0	85.1	85.3	**85.3**	84.9
Max	85.3	85.4	85.2	84.6	83.0	85.0	85.2	85.2	85.2	84.9
Z50	85.4	**85.4**	85.2	84.7	83.0	85.0	85.2	85.3	85.4	85.0
Average	85.4	85.4	85.2	84.7	83.0	85.0	85.2	85.3	85.3	–

Hungarian

Pooling \ Layer	08	09	10	11	12	A4	A4E	A5	A5E	Average
Avg	85.0	85.2	85.1	84.6	83.4	85.0	84.9	85.0	**85.1**	84.8
First	84.9	84.8	84.7	84.4	82.9	84.6	84.6	84.5	84.6	84.4
Last	85.3	**85.5**	85.3	85.0	83.6	85.2	85.3	85.4	85.3	85.1
Max	**84.9**	85.2	84.9	84.5	83.1	84.7	84.7	84.8	84.9	84.6
Z50	84.9	85.1	84.9	84.5	83.3	84.9	84.8	84.8	84.8	84.6
Average	85.0	85.2	85.0	84.6	83.3	84.9	84.9	84.9	84.9	–

Uyghur

Pooling \ Layer	08	09	10	11	12	A4	A4E	A5	A5E	Average
Avg	66.8	67.0	66.8	66.8	67.2	66.8	67.0	**67.2**	66.9	66.9
First	66.9	66.8	66.9	66.9	67.1	66.9	67.0	67.0	67.0	66.9
Last	66.5	66.6	66.8	66.7	66.8	66.8	66.6	66.8	66.8	66.7
Max	**67.0**	66.8	66.7	66.6	67.0	66.9	66.9	66.9	66.8	66.9
Z50	67.0	67.0	66.9	66.8	**67.2**	66.9	66.8	67.0	66.9	66.9
Average	66.8	66.9	66.8	66.8	67.1	66.9	66.9	67.0	66.9	–

Vietnamese

Pooling \ Layer	08	09	10	11	12	A4	A4E	A5	A5E	Average
Avg	68.5	**68.6**	68.4	68.3	67.4	68.5	68.3	68.5	68.4	68.3
First	68.7	68.3	68.4	68.1	67.6	68.2	68.3	**68.7**	68.4	68.3
Last	**68.7**	68.4	68.5	68.2	67.2	68.4	68.3	68.6	68.5	68.3
Max	68.5	68.3	68.3	68.1	67.5	68.4	68.2	68.3	68.4	68.2
Z50	68.5	68.5	68.3	67.9	67.6	68.3	68.3	68.4	68.5	68.3
Average	68.6	68.4	68.4	68.1	67.5	68.4	68.3	68.5	68.5	–

Average over all languages

Pooling \ Layer	08	09	10	11	12	A4	A4E	A5	A5E	Average
Avg	76.5	76.6	76.4	76.1	75.3	76.3	76.4	76.5	76.4	76.3
First	76.4	76.3	76.3	76.0	75.1	76.1	76.2	76.4	76.3	76.1
Last	76.5	76.5	76.4	76.1	75.2	76.4	76.3	76.5	76.5	76.3
Max	76.4	76.5	76.3	76.0	75.1	76.3	76.3	76.3	76.3	76.2
Z50	76.4	76.5	76.3	76.0	75.3	76.3	76.3	76.4	76.4	76.2
Average	76.4	76.5	76.3	76.0	75.2	76.3	76.3	76.4	76.4	–

Table 9: Development set LAS for training UDPipe-Future with word embeddings taken from different BERT layers. Each inner cell shows the average over 25 runs. Pooling methods are average, first, last, maximum and weighted average with binonimal distribition with $p = 0.5$ (Z50). Layer A4 (A5) stands for using the average of the top 4 (5) layers. The E suffix means that the 768-dimensional BERT_BASE vectors are expanded to 1024 components so that all (A4E) or most (A5E) final components can be the average of fewer input components. The three settings selected for the three learners of each language are shown in **bold**.