DISTILLING NEURAL NETWORKS FOR FASTER AND GREENER DEPENDENCY PARSING

Anonymous authors
Paper under double-blind review

ABSTRACT

The carbon footprint of natural language processing (NLP) research has been increasing in recent years due to its reliance on large and inefficient neural network implementations. Distillation is a network compression technique which attempts to impart knowledge from a large model to a smaller one. We use *teacher-student* distillation to improve the efficiency of the Biaffine dependency parser which obtains state-of-the-art performance with respect to accuracy and parsing speed (Dozat & Manning, 2016). When distilling to 20% of the original model's trainable parameters, we only observe an average decrease of $\sim$1 point for both UAS and LAS across a number of diverse Universal Dependency treebanks while being 2.26x (1.21x) faster than the baseline model on CPU (GPU) at inference time. We also observe a small increase in performance when compressing to 80% for some treebanks. Finally, through distillation we attain a parser which is not only faster but also more accurate than the fastest modern parser on the Penn Treebank.

1 INTRODUCTION

Ethical NLP research has recently gained attention (Kurita et al., 2019; Sun et al., 2019). For example, the environmental cost of AI research has become a focus of the community, especially with regards to the development of deep neural networks (Schwartz et al., 2019; Strubell et al., 2019). Beyond developing systems to be greener, increasing the efficiency of models makes them more cost-effective, which is a compelling argument even for people who might downplay the extent of anthropogenic climate change.

In conjunction with this push for greener AI, NLP practitioners have turned to the problem of developing models that are not only accurate but also efficient, so as to make them more readily deployable across different machines with varying computational capabilities (Strzyz et al., 2019; Clark et al., 2019; Vilares et al., 2019; Junczys-Dowmunt et al., 2018). This is in contrast with the recently popular principle of *make it bigger, make it better* (Devlin et al., 2019; Radford et al., 2019).

Here we explore *teacher-student* distillation as a means of increasing the efficiency of neural network systems used to undertake a core task in NLP, dependency parsing. To do so, we take a state-of-the-art (SoTA) Biaffine parser from Dozat & Manning (2016). The Biaffine parser is not only one of the most accurate parsers, it is the fastest implementation by almost an order of magnitude among state-of-the-art performing parsers.

Contribution We utilise *teacher-student* distillation to compress Biaffine parsers trained on a diverse subset of Universal Dependency (UD) treebanks. We find that distillation maintains accuracy performance close to that of the full model and obtains far better accuracy than simply implementing equivalent model size reductions by changing the parser's network size and training regularly. Furthermore, we can compress a parser to 20% of its trainable parameters with minimal loss in accuracy and with a speed 2.26x (1.21x) faster than that of the original model on CPU (GPU).

2 DEPENDENCY PARSING

Dependency parsing is a core NLP task where the syntactic relations of words in a sentence are encoded as a well-formed tree with each word attached to a head via a labelled arc. Figure 1 shows

an example of such a tree. The syntactic information attained from parsers has been shown to benefit a number of other NLP tasks such as relation extraction (Zhang et al., 2018), machine translation (Chen et al., 2018), and sentiment analysis (Poria et al., 2014; Vilares et al., 2017).

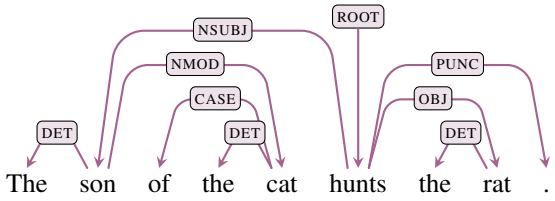


Figure 1: Dependency tree example.

2.1 CURRENT PARSER PERFORMANCE

Table 1 shows performance details of current SoTA dependency parsers on the English Penn Treebank (PTB) with predicted POS tags from the Stanford POS tagger (Marcus & Marcinkiewicz, 1993; Toutanova et al., 2003). The Biaffine parser of Dozat & Manning (2016) offers the best trade-off between accuracy and parsing speed with the HPSG parser of Zhou & Zhao (2019) achieving the absolute best reported accuracy but with a reported parsing speed of roughly one third of the Biaffine's parsing speed. It is important to note that direct comparisons between systems with respect to parsing speed are wrought with compounding variables, e.g. different GPUs or CPUs used, different number of CPU cores, different batch sizes, and often hardware is not even reported.

	speed (sent/s)			
	GPU	**CPU**	**UAS**	**LAS**
Pointer-TD (Ma et al., 2018)	-	$10.2^{\dagger}$	$95.87^{\dagger}$	$94.19^{\dagger}$
Pointer-LR (Fernández-González & Gómez-Rodríguez, 2019)	-	$23.1^{\dagger}$	$96.04^{\dagger}$	$94.43^{\dagger}$
HPSG (Zhou & Zhao, 2019)	$158.7^{\dagger}$	-	$96.09^{\dagger}$	$94.68^{\dagger}$
BIST - Transition (Kiperwasser & Goldberg, 2016)	-	$76\pm1^{\ddagger}$	$93.9^{\dagger}$	$91.9^{\dagger}$
BIST - Graph (Kiperwasser & Goldberg, 2016)	-	$80\pm0^{\ddagger}$	$93.1^{\dagger}$	$91.0^{\dagger}$
Biaffine (Dozat & Manning, 2016)	$411^{\dagger}$	-	$95.74^{\dagger}$	$94.08^{\dagger}$
CM (Chen & Manning, 2014)	-	$654^{\dagger}$	$91.80^{\dagger}$	$89.60^{\dagger}$
SeqLab (Strzyz et al., 2019)	$648\pm20^{\ddagger}$	$101\pm2^{\ddagger}$	$93.67^{\ddagger}$	$91.72^{\ddagger}$
UUParser (Smith et al., 2018)	-	42 ± 1	94.63	92.77
Biaffine (PyTorch)	957 ± 2	57 ± 0	95.74	94.07
SeqLab	1061 ± 6	97 ± 0	93.46	91.49
Biaffine-D20	1161 ± 9	414 ± 2	92.84	90.73
Biaffine-D40	1123 ± 6	101 ± 0	94.59	92.64
Biaffine-D60	1088 ± 3	80 ± 0	94.78	92.86
Biaffine-D80	987 ± 3	66 ± 0	94.84	92.95

Table 1: Speed and accuracy performance for SoTA parsers and parsers from our distillation method, Biaffine-Dπ compressing to π% of the original model, for the English PTB with POS tags predicted from the Stanford POS tagger. In the first table block, † denotes values taken from the original paper, ‡ from Strzyz et al. (2019). Values with no superscript (second and third blocks) are from running the models on our system locally with a single CPU core for both CPU and GPU speeds (averaged over 5 runs) and with a batch size of 4096 with GloVe 100 dimension embeddings.

We therefore run a subset of parsers locally to achieve speed measurements in a controlled environment, also shown in Table 1: we compare a PyTorch implentation of the Biaffine parser (which runs more than twice as fast as the reported speed of the original implementation); the UUParser from Smith et al. (2018) which is one of the leading parsers for Universal Dependency (UD) parsing; a sequence-labelling dependency parser from Strzyz et al. (2019) which has the fastest reported parsing speed amongst modern parsers; and also distilled Biaffine parsers from our implementation

described below. All speeds measured here are with the system run with a single CPU core for both GPU and CPU runs.[1]

Biaffine parser is a graph-based parser extended from the graph-based BIST parser (Kiperwasser & Goldberg, 2016) to use a deep self-attention mechanism. This results in a fast and accurate parser, as described above, and is used as the parser architecture for our experiments. More details of the system can be found in Dozat & Manning (2016).

3 NETWORK COMPRESSION

Model compression has been under consideration for almost as long as neural networks have been utilised, e.g. LeCun et al. (1990) introduced a pruning technique which removed weights based on a locally predicted contribution from each weight so as to minimise the perturbation to the error function. More recently, Han et al. (2015) introduced a means of pruning a network up to 40 times smaller with minimal affect on performance. Hagiwara (1994) and Wan et al. (2009) utilised magnitude-based pruning to increase network generalisation. More specific to NLP, See et al. (2016) used absolute-magnitude pruning to compress neural machine translation systems by 40% with minimal loss in performance. However, pruning networks leaves them in an irregularly sparse state which cannot be trivially re-cast into less sparse architectures. Sparse tensors could be used for network layers to obtain real-life decreases in computational complexity, however, current deep learning libraries lack this feature. Anwar et al. (2017) introduced structured pruning to account for this, but this kernel-based technique is restricted to convolutional networks. More recently Voita et al. (2019) pruned the heads of the attention mechanism in their neural machine translation system and found that the remaining heads were linguistically salient with respect to syntax, suggesting that pruning could also be used to undertake more interesting analyses beyond merely compressing models and helping generalisation.

Ba & Caruana (2014) and Hinton et al. (2015) developed distillation as a means of network compression from the work of Bucilu et al. (2006), who compressed a large ensemble of networks into one smaller network. *Teacher-student* distillation is the process of taking a large network, the *teacher*, and transferring its knowledge to a smaller network, the *student*. *Teacher-student* distillation has successfully been exploited in NLP for machine translation, language modelling, and speech recognition (Kim & Rush, 2016; Yu et al., 2018; Lu et al., 2017). Latterly, it has also been used to distill task-specific knowledge from BERT (Tang et al., 2019).

Other compression techniques have been used such as low-rank approximation decomposition (Yu et al., 2017), vector quantisation (Wu et al., 2016), and Huffman coding (Han et al., 2016). For a more thorough survey of current neural network compression methods see Cheng et al. (2018).

4 TEACHER-STUDENT DISTILLATION

The essence of model distillation is to train a model and subsequently use the patterns it learnt to influence the training of a smaller model. For *teacher-student* distillation, the smaller model, the *student*, explicitly uses the information learnt by the larger original model, the *teacher*, by comparing the distribution of each model's output layer. We use the Kullback-Leibler divergence to calculate the loss between the teacher and the student:

$$\mathcal{L}_{KL} = -\sum_{t \in T} \sum_{x \in X} P(x) \log \frac{P(x)}{Q(x)} \tag{1}$$

where P is the probability distribution from the teacher's softmax layer, Q is the probability distribution from the student's, and x is the input to the target layer for token w_x in a given tree, t.

For our implementation, there are two probability distributions for each model, one for the arc prediction and one for the label prediction. By using the distributions of the teacher rather than just using the predicted arc and label, the student can learn more comprehensively about which arcs and

[1]This is for ease of comparability. Parsing can trivially be parallelised by allocating sentences to different cores, so speed per core is an informative metric to compare parsers (Hall et al., 2014).

labels are very unlikely in a given context, i.e. if the teacher makes a mistake in its prediction, the distribution might still carry useful information such as having a similar probability for y_g and y_p which can help guide the student better rather than just learning to copy the teacher's predictions.

In addition to the loss with respect to the teacher's distributions, the student model is also trained using the loss on the gold labels in the training data. We use cross entropy to calculate the loss on the student's predicted head classifications:

$$\mathcal{L}_{CE} = -\sum_{t \in T} \sum_{h \in H} p(h) \log p(\hat{h}) \tag{2}$$

where t is a tree in the treebank T, h is a head position for the set of heads H for a given tree, and $\hat{h}$ is the head position predicted by the student model. Similarly, cross entropy is used to calculate the loss on the predicted arc labels for the student model. The total loss for the student model is therefore:

$$\mathcal{L} = \mathcal{L}_{KL}(T_h, S_h) + \mathcal{L}_{KL}(T_{lab}, S_{lab}) + \mathcal{L}_{CE}(h) + \mathcal{L}_{CE}(lab) \tag{3}$$

where $\mathcal{L}_{CE}(h)$ is the loss for the student's predicted head positions, $\mathcal{L}_{CE}(lab)$ is the loss for the student's predicted arc label, $\mathcal{L}_{KL}(T_h, S_h)$ is the loss between the teacher's probability distribution for arc predictions and that of the student, and $\mathcal{L}_{KL}(T_{lab}, S_{lab})$ is the loss between label distributions.

5 METHODOLOGY

We train a Biaffine parser for a number of Universal Treebanks v2.4 (UD) (Nivre et al., 2019) and apply the *teacher-student* distillation method to compress these models into a number of different sizes. We use the hyperparameters from Dozat & Manning (2016), but use a PyTorch implementation for our experiments which obtains the same parsing results and runs faster than the reported speed of the original (see Table 1).[2] The hyperparameter values can be seen in Table 4. During distillation dropout is not used. Beyond lexical features, the model only utilises universal part-of-speech (UPOS) tags. Gold UPOS tags were used for training and at runtime. Also, we used gold sentence segmentation and tokenisation. We opted to use these settings to compare models under homogeneous settings, so as to make reproducibility of and comparability with our results easier.

Data We use the subset of UD treebanks suggested by de Lhoneux et al. (2017) from v2.4, so as to cover a wide range of linguistic features, linguistic typologies, and different dataset sizes. We make some changes as this set of treebanks was chosen from a previous UD version. We exchange Kazakh with Uyghur because the Kazakh data does not include a development set and Uyghur is a closely related language. We also exchange Ancient-Greek-Proiel for Ancient-Greek-Perseus because it contains more non-projective arcs (the number of arcs which cross another arc in a given tree) as this was the original justification for including Ancient Greek. We also included Wolof as African languages were wholly unrepresented in the original collection of suggested treebanks. Details of the treebanks pertinent to parsing can be seen in Table 2. We use pretrained word embeddings from FastText (Grave et al., 2018) for all but Ancient Greek, for which we used embeddings from Ginter et al. (2017), and Wolof, for which we used embeddings from Heinzerling & Strube (2018). When necessary, we used the algorithm of Raunak (2017) to reduce the embeddings to 100 dimensions.

For each treebank we then acquired the following models:

i **Baseline 1**: Full-sized model is trained as normal and undergoes no compression technique.

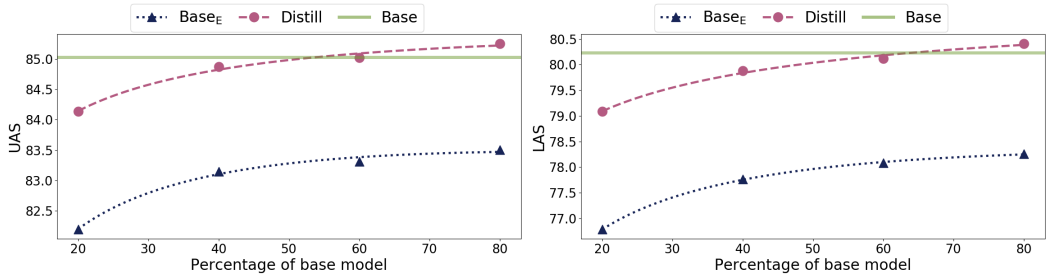
ii **Baseline 2**: Model is trained as normal but with equivalent sizes of the distilled models (20%, 40%, 60%, and 80% of the original size) and undergoes no compression technique. These models have the same overall structure of baseline 1, with just the number of dimensions of each layer changed to result in a specific percentage of trainable parameters of the full model.

iii **Distilled**: Model is distilled using the *teacher-student* method. We have four models were the first is distilled into a smaller network with 20% of the parameters of the original, the second 40%, the third 60%, and the last 80%. The network structure and parameters of the distilled models are the exact same as those of the baseline 2 models.

[2]The implementation can be found at `github.com/zysite/biaffine-parser`. Beyond adding our distillation method, we also included the Chu-Liu/Edmonds' algorithm, as used in the original, to enforce well-formed trees.

	number of trees			average tree length			average arc length			non-proj. arc pct		
	train	dev	test	train	dev	test	train	dev	test	train	dev	test
Ancient-Greek-Perseus	11476	1137	1306	14.9	20.5	17.0	4.1	4.5	4.1	23.9	23.2	23.5
Chinese-GSD	3997	500	500	25.7	26.3	25.0	4.7	4.9	4.7	0.1	0.0	0.3
English-EWT	12543	2002	2077	17.3	13.6	13.1	3.7	3.5	3.6	1.0	0.6	0.6
Finnish-TDT	12217	1364	1555	14.3	14.4	14.5	3.4	3.4	3.4	1.6	1.9	1.8
Hebrew-HTB	5241	484	491	27.3	24.6	26.0	3.9	3.8	3.7	0.8	0.8	0.9
Russian-GSD	3850	579	601	20.5	21.2	19.9	3.5	3.7	3.7	1.1	1.0	1.2
Tamil-TTB	400	80	120	16.8	16.8	17.6	3.5	3.7	3.7	0.3	0.0	0.2
Uyghur-UDT	1656	900	900	12.6	12.8	12.5	3.5	3.5	3.5	1.1	1.3	1.4
Wolof-WTB	1188	449	470	20.8	23.9	23.1	3.5	3.8	3.6	0.4	0.4	0.5

Table 2: Statistics for salient features with respect to parsing difficulty for each UD treebank used: number of trees, the number of data instances; average tree length, the length of each data instance on average; average arc length, the mean distance between heads and dependents; non.proj. arc pct, the percentage of non-projective arcs in a treebank.



(a) (b)

Figure 2: UAS (a) and LAS (a) against the model size relative to the original full-sized model: Base$_E$, the baseline models of equivalent size to the distilled models; Distill, the distilled models; Base, the performance of the original full-sized model.

Hardware For evaluating the speed of each model when parsing the test sets of each treebank we set the number of CPU cores to be one and either ran the parser using that solitary core or using a GPU (using a single CPU core too). The CPU used was an Intel Core i7-7700 and the GPU was an Nvidia GeForce GTX 1080.[3]

Experiment We compare the performance of each model on the aforementioned UD treebanks with respect to the unlabelled attachment score (UAS) which evaluates the accuracy of the arcs, and the labelled attachment score (LAS) which also includes the accuracy of the arc labels. We also evaluate the differences in inference time for each model on CPU and GPU with respect to sentences per second and tokens per second. We report sentences per second as this has been the measurement traditionally used in most of the literature, but we also use tokens per second as this more readily captures the difference in speed across parsers for different treebanks where the sentence length varies considerably. We also report the number of trainable parameters of each distilled model and how they compare to the baseline, as this is considered a good measure of how green a model is in lieu of the number of floating point operations (FPO) (Schwartz et al., 2019).[4]

6 Results and Discussion

Figure 2a shows the average attachment scores across all treebanks for the distilled models and the equivalent-sized base models against the size of the model relative to the original full model. There is a clear gap in performance between these two sets of models with roughly 2 points of UAS and LAS more for the distilled models. This shows that the distilled models do actually manage to

[3]Using Python 3.70, PyTorch 1.0.0, and CUDA 8.0.

[4]There exist a number of packages for computing the FPO of a model but, to our knowledge, as of yet they do not include the capability of dealing with LSTMs.

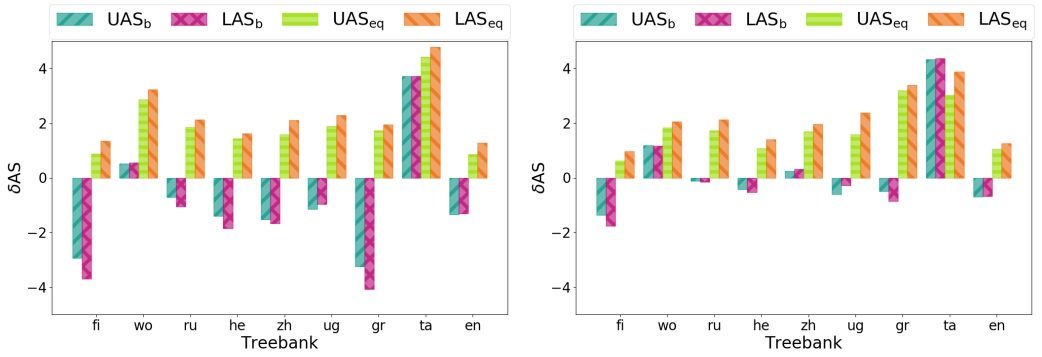


Figure 3: Delta UAS and LAS for when comparing both the original base model and equivalent-sized base models for each treebank for two of our distilled models: (a) D-20, 20% of original model and (b) D-80, 80% of original model.

	gr	**zh**	**en**	**fi**	**he**	**ru**	**ta**	**ug**	**wo**
F	12.28	11.98	12.23	12.77	12.04	11.92	11.22	11.45	11.39
20	2.47 (19.7)	2.42 (20.2)	2.44 (19.7)	2.56 (19.7)	2.39 (19.2)	2.36 (19.3)	2.25 (19.6)	2.30 (20.2)	2.27 (19.5)
40	4.88 (39.3)	4.79 (39.5)	4.86 (39.3)	5.12 (40.2)	4.80 (40.0)	4.73 (39.5)	4.49 (39.3)	4.60 (40.4)	4.57 (39.8)
60	7.35 (59.8)	7.24 (60.5)	7.33 (59.8)	7.66 (59.8)	7.19 (59.2)	7.18 (59.7)	6.71 (59.8)	6.90 (60.5)	6.84 (60.2)
80	9.80 (80.3)	9.57 (79.8)	9.75 (79.5)	10.23 (80.3)	9.59 (79.2)	9.52 (79.8)	8.94 (79.5)	9.19 (79.8)	9.12 (80.5)

Table 3: Trainable model parameters ($\times 10^6$) with percentage of full model in parentheses.

leverage the information from the original full model. The full model's scores are also shown and it is clear that on average the model can be distilled to 60% with no loss in performance. When compressing to 20% of the full model, the performance only decreases by about 1 point for both UAS and LAS.

Figures 3a and 3b show the differences in UAS and LAS for the models distilled to 20% and 80% respectively for each treebank when compared to the equivalent sized baseline model and the full baseline model. The distilled models far outperform the equivalent-sized baselines for all treebanks. It is clear that for the smaller model that some treebanks suffer more when compressed to 20% than others when compared to the full baseline model, e.g. Finnish-TDT and Ancient-Greek-Perseus. These two treebanks have the largest percentage of non-projective arcs (as can be seen in Table 2) which could account for the decrease in performance, with a more powerful model required to account for this added syntactic complexity.

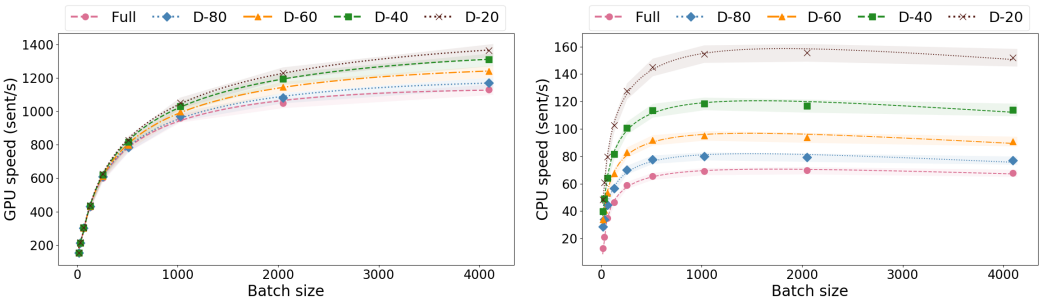
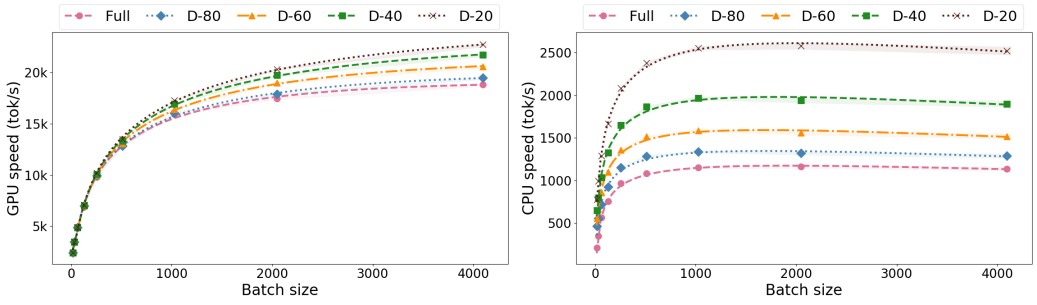
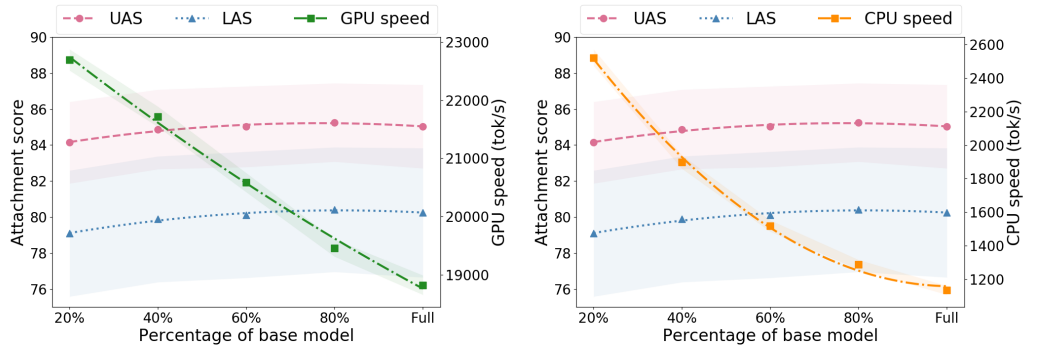


Figure 4: GPU (a) and single core CPU (b) speeds in sentence per second with varying batch sizes for distilled models (D-X) and full-sized base model (Full). Shaded areas show the standard error.



(a) (b)

Figure 5: GPU (a) and single core CPU (b) speeds in tokens per second with varying batch sizes for distilled models (D-X) and full-sized base model (Full). Shaded areas show the standard error.



(a) (b)

Figure 6: Comparison of attachment scores and percentage increase of speed (tok/s) for different distilled models with batch size 4096: speed on GPU (a) and speed on CPU (b). Shaded areas show the standard error.

However, the two smallest treebanks, Tamil-TTB and Wolof-WTB, actually increase in accuracy when using distillation, especially Tamil-TTB, which is by far the smallest treebank, with an increase in UAS and LAS of about 4 points over the full base model. This is likely the result of over-fitting when using the larger, more powerful model, so that reducing the model size actually helps with generalisation.

These observations are echoed in the results for the model distilled to 80%, where most treebanks lose less than a point for UAS and LAS against the full baseline, but have a smaller increase in performance over the equivalent-sized baseline. This makes sense as the model is still close in size to the full baseline and still similarly powerful. The increase in performance for Tamil-TTB and Wolof-WTB are greater for this distilled model, which suggests the full model doesn't need to be compressed to such a small model to help with generalisation. The full set of attachment scores from our experiments can be seen in Table 5 in the Appendix.

With respect to how green our distilled models are, Table 3 shows the number of trainable parameters for each distilled model for each treebank alongside its corresponding full-scale baseline. We report these in lieu of FPO as, to our knowledge, no packages exist to calculate the FPO for neural network layers like LSTMs which are used in our network. These numbers do not depend on the hardware used and strongly correlate with the amount of memory a model consumes. Different algorithms do utilise parameters differently, however, the models compared here are of the same structure and use the same algorithm, so comparisons of the number of trainable model parameters do relate to how much work each respective model does compared to another.

Figures 4 and 5 show the parsing speeds on CPU and GPU for the distilled models and for the full baseline model for sentence per second and token per second, respectively. The speeds are reported for different batch sizes as this obviously affects the speed at which a neural network can make

predictions, but the maximum batch size that can be used on different systems varies significantly. As can be seen in Figures 4a and 5a, the limiting factor in parsing speed is the bottleneck of loading the data onto the GPU when using a batch size less than $\sim$1000 sentences. However, with a batch size of 4096 sentences, we achieve an increase in parsing speed of 21% over the full baseline model when considering tokens per second.

As expected, a much smaller batch size is required to achieve increases in parsing speed when using a CPU. Even with a batch size of 32 sentences, the smallest model more than doubles the speed of the baseline. For a batch size of 4096, the distilled model compressed to 20% increases the speed of the baseline by 126% when considering tokens per second. A full breakdown of the parsing speeds for each treebank and each model when using a batch size of 4096 sentences is given in Table 6 in the Appendix.

Figure 6 shows the attachment scores and the corresponding parsing speed against model size for the distilled model and the full baseline model. These plots clearly show that the cost in accuracy is neglible when compared to the large increase in parsing speed. So not only does this *teacher-student* distillation technique maintain the accuracy of the baseline model, but it achieves real compression and with it practical increases in parsing speed and with a greener implementation. In absolute terms, our distilled models are faster than the previously fastest parser using sequence labelling, as can be seen explicitly in Table 1 for PTB, and outperforms it by over 1 point with respect to UAS and LAS when compressing to 40%. Distilling to 20% results in a speed 4x that of the sequence labelling model on CPU but comes at a cost of 0.62 points for UAS and 0.76 for LAS compared to the sequence labelling accuracies.

Furthermore, the increase in parsing accuracy for the smaller treebanks suggests that distillation could be used as a more efficient way of finding optimal hyperparameters depending on the available data, rather than training numerous models with varying hyperparameter settings.

6.1 Future work

There are numerous ways in which this distillation technique could be augmented to potentially retain more performance and even outperform the large baseline models, such as using *teacher annealing* introduced by Clark et al. (2019) where the distillation process gradually secedes to standard training. Beyond this, the structure of the distilled models can be altered, e.g. student models which are more shallow than the teacher models (Ba & Caruana, 2014). This technique could further improve the efficiency of models and make them more environmentally friendly by reducing the depth of the models and therefore the total number of trainable parameters.

Distillation techniques can also be easily expanded to other NLP tasks. Already attempts have been made to make BERT more wieldy by compressing the information it contains into task-specific models (Tang et al., 2019). But this can be extended to other tasks more specifically and potentially reduce the environmental impact of NLP research and deployable NLP systems.

7 Conclusion

We have shown the efficacy of using the *teacher-student* distillation technique for dependency parsing by distilling a state-of-the-art parser implementation. The parser used for our experiments was not only accurate but already fast, meaning it was a strong baseline from which to see improvements. We obtained parsing speeds up to 2.26x (1.21x) faster on CPU (GPU) while only losing $\sim$1 point for both UAS and LAS when compared to the original sized model. Furthermore, the smallest model which obtains these results only has 20% of the original model's trainable parameters, vastly reducing its environmental impact.

REFERENCES

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, pp. 2654–2662, 2014.

Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 535–541. ACM, 2006.

Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 740–750, 2014.

Kehai Chen, Rui Wang, Masao Utiyama, Eiichiro Sumita, and Tiejun Zhao. Syntax-directed attention for neural machine translation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35 (1):126–136, 2018.

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D Manning, and Quoc Le. BAM! Born-again multi-task networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5931–5937, 2019.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. Old school vs. new school: Comparing transition-based parsers with and without neural network enhancement. In *TLT*, pp. 99–110, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.

Timothy Dozat and Christopher D Manning. Deep biaffine attention for neural dependency parsing. *Proceedings of the 5th International Conference on Learning Representations*, 2016.

Daniel Fernández-González and Carlos Gómez-Rodríguez. Left-to-right dependency parsing with pointer networks. In *Proceedings of NAACL-HLT*, pp. 710–716, 2019.

Filip Ginter, Jan Hajic, Juhani Luotolahti, Milan Straka, and Daniel Zeman. CoNLL 2017 shared task-automatically annotated raw texts and word embeddings., 2017. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

Masafumi Hagiwara. A simple and effective method for removal of hidden units and weights. *Neurocomputing*, 6(2):207–218, 1994.

David Hall, Taylor Berg-Kirkpatrick, and Dan Klein. Sparser, better, faster GPU parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 208–217, 2014.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

Benjamin Heinzerling and Michael Strube. BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, May 7-12, 2018 2018.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, et al. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pp. 116–121, 2018.

Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of EMNLP*, pp. 1317–1327, 2016.

Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016.

Keita Kurita, Nidhi Vyas, Ayush Pareek, Alan W Black, and Yulia Tsvetkov. Measuring bias in contextualized word representations. *Proceedings of the 1st Workshop on Gender Bias in Natural Language Processing*, pp. 166172, 2019.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Liang Lu, Michelle Guo, and Steve Renals. Knowledge distillation for small-footprint highway networks. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4820–4824. IEEE, 2017.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1403–1414, 2018.

Mitchell P Marcus and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2), 1993.

Joakim Nivre, Mitchell Abrams, Željko Agić, et al. Universal Dependencies 2.4, 2019. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Soujanya Poria, Erik Cambria, Grégoire Winterstein, and Guang-Bin Huang. Sentic patterns: Dependency-based rules for concept-level sentiment analysis. *Knowledge-Based Systems*, 69: 45–63, 2014.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

Vikas Raunak. Simple and effective dimensionality reduction for word embeddings. *Proceedings of NIPS LLD Workshop*, 2017.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green AI. *arXiv preprint arXiv:1907.10597*, 2019.

Abigail See, Minh-Thang Luong, and Christopher D Manning. Compression of neural machine translation models via pruning. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 291–301, 2016.

Aaron Smith, Bernd Bohnet, Miryam de Lhoneux, Joakim Nivre, Yan Shao, and Sara Stymne. 82 treebanks, 34 models: Universal dependency parsing with multi-treebank models. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 113–123, 2018.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2019.

Michalina Strzyz, David Vilares, and Carlos Gómez-Rodrıguez. Viable dependency parsing as sequence labeling. In *Proceedings of NAACL-HLT*, pp. 717–723, 2019.

Tony Sun, Andrew Gaut, Shirlyn Tang, Yuxin Huang, Mai ElSherief, Jieyu Zhao, Diba Mirza, Elizabeth Belding, Kai-Wei Chang, and William Yang Wang. Mitigating gender bias in natural language processing: Literature review. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1630–1640, 2019.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from BERT into simple neural networks. *arXiv preprint arXiv:1903.12136*, 2019.

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

David Vilares, Carlos Gómez-Rodríguez, and Miguel A Alonso. Universal, unsupervised (rule-based), uncovered sentiment analysis. *Knowledge-Based Systems*, 118:45–55, 2017.

David Vilares, Mostafa Abdou, and Anders Søgaard. Better, faster, stronger sequence tagging constituent parsers. 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)*, pp. 3372–3383, 2019.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5797–5808, 2019.

Weishui Wan, Shingo Mabu, Kaoru Shimada, Kotaro Hirasawa, and Jinglu Hu. Enhancing the generalization ability of neural networks through controlling the hidden layers. *Applied Soft Computing*, 9(1):404–414, 2009.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4820–4828, 2016.

Seunghak Yu, Nilesh Kulkarni, Haejun Lee, and Jihie Kim. On-device neural language model based word prediction. In *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pp. 128–131, 2018.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7370–7379, 2017.

Yuhao Zhang, Peng Qi, and Christopher D Manning. Graph convolution over pruned dependency trees improves relation extraction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2205–2215, 2018.

Junru Zhou and Hai Zhao. Head-driven phrase structure grammar parsing on Penn Treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2396–2408, 2019.

A APPENDIX

hyperparameter	**value**
word embedding dimensions	100
pos embedding dimensions	100
embedding dropout	0.33
BiLSTM dimensions	400
BiLSTM layers	3
arc MLP dimensions	500
label MLP dimensions	100
MLP layers	1
learning rate	0.2
dropout	0.33
momentum	0.9
L2 norm λ	0.9
annealing	$0.75^{\wedge}(t/5000)$
ϵ	1×10^{-12}
optimiser	Adam
loss function	cross entropy
epochs	100
batch size	1024

Table 4: Hyperparameters for full-sized baseline models.

	gr		**zh**		**en**		**fi**		**he**		**ru**		**ta**		**ug**		**wo**		**avg**	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Full	75.5	88.2	90.8	90.5	90.8	88.9	76.9	75.2	88.5	85.0	70.4	85.9	89.0	88.6	88.6	85.2	71.0	58.9	84.5	80.2
B-20	70.5	85.1	88.6	86.7	87.9	86.3	76.2	72.2	86.1	82.2	64.4	82.1	86.4	83.6	85.1	82.0	69.9	55.6	81.8	76.8
B-40	72.2	86.1	88.9	87.7	88.5	87.1	78.4	73.0	86.5	83.2	66.4	83.5	86.8	84.8	85.6	83.1	71.8	55.7	82.2	77.8
B-60	72.0	86.7	89.5	88.1	88.7	87.1	77.5	72.7	87.5	83.3	66.4	84.0	87.5	85.5	86.3	83.1	70.9	55.9	83.1	78.1
B-80	71.8	86.7	89.1	88.5	89.3	87.1	78.2	73.0	87.8	83.5	66.2	84.3	87.1	85.9	86.6	82.9	71.5	56.2	83.6	78.3
D-20	72.3	86.7	89.5	87.6	89.4	88.2	80.6	74.1	89.0	84.1	66.4	84.2	87.7	84.9	86.7	84.2	74.7	57.9	85.0	79.1
D-40	74.0	87.9	89.9	89.5	89.4	88.4	80.9	74.5	89.4	84.9	68.3	85.6	88.0	86.9	87.0	84.6	74.7	58.3	85.5	79.9
D-60	74.2	88.3	90.1	89.4	90.0	88.6	80.4	74.5	89.5	85.0	68.7	85.9	88.3	87.1	87.5	84.7	74.5	58.6	85.8	80.1
D-80	75.0	88.4	90.1	89.2	90.3	88.8	81.2	74.6	89.6	85.3	69.6	86.2	88.3	86.9	88.0	85.0	75.4	58.6	85.7	80.4

Table 5: Attachment Scores

		Full	D-20	D-40	D-60	D-80
gr	CPU (tok/s)	1282 ± 3	2936 ± 7	2157 ± 9	1755 ± 1	1425 ± 17
	CPU (sent/s)	79.9 ± 0.2	182.9 ± 0.4	134.4 ± 0.6	109.4 ± 0.1	88.8 ± 1.0
	GPU (tok/s)	19784 ± 38	21658 ± 89	21851 ± 97	20785 ± 66	19495 ± 101
	GPU (sent/s)	1232.8 ± 2.4	1349.6 ± 5.5	1361.6 ± 6.1	1295.2 ± 4.1	1214.8 ± 6.3
zh	CPU (tok/s)	1184 ± 9	2581 ± 33	1999 ± 2	1582 ± 1	1359 ± 4
	CPU (sent/s)	49.3 ± 0.4	107.4 ± 1.4	83.2 ± 0.1	65.9 ± 0.0	56.6 ± 0.2
	GPU (tok/s)	19965 ± 88	24189 ± 76	23393 ± 92	22052 ± 179	20585 ± 249
	GPU (sent/s)	831.0 ± 3.7	1006.9 ± 3.2	973.7 ± 3.8	917.9 ± 7.5	856.9 ± 10.4
en	CPU (tok/s)	965 ± 1	2370 ± 6	1672 ± 2	1323 ± 1	1107 ± 1
	CPU (sent/s)	79.9 ± 0.1	196.1 ± 0.5	138.4 ± 0.2	109.5 ± 0.1	91.6 ± 0.1
	GPU (tok/s)	17436 ± 85	22232 ± 68	21037 ± 67	19952 ± 73	18230 ± 35
	GPU (sent/s)	1443.1 ± 7.0	1840.0 ± 5.6	1741.1 ± 5.6	1651.3 ± 6.1	1508.8 ± 2.9
fi	CPU (tok/s)	1058 ± 2	2691 ± 5	1876 ± 2	1459 ± 2	1231 ± 1
	CPU (sent/s)	78.1 ± 0.2	198.6 ± 0.3	138.4 ± 0.1	107.7 ± 0.2	90.9 ± 0.1
	GPU (tok/s)	19117 ± 45	23513 ± 97	22581 ± 74	21193 ± 86	19804 ± 64
	GPU (sent/s)	1410.8 ± 3.4	1735.3 ± 7.2	1666.5 ± 5.5	1564.1 ± 6.4	1461.6 ± 4.7
he	CPU (tok/s)	1316 ± 1	2833 ± 5	2150 ± 3	1759 ± 3	1487 ± 2
	CPU (sent/s)	52.6 ± 0.1	113.2 ± 0.2	85.9 ± 0.1	70.3 ± 0.1	59.4 ± 0.1
	GPU (tok/s)	21170 ± 136	25277 ± 81	24504 ± 115	23424 ± 104	21602 ± 123
	GPU (sent/s)	846.2 ± 5.4	1010.3 ± 3.2	979.4 ± 4.6	936.3 ± 4.2	863.5 ± 4.9
ru	CPU (tok/s)	764 ± 1	1756 ± 1	1289 ± 1	1021 ± 2	867 ± 1
	CPU (sent/s)	40.3 ± 0.0	92.7 ± 0.1	68.1 ± 0.0	53.9 ± 0.1	45.8 ± 0.1
	GPU (tok/s)	16701 ± 54	20782 ± 119	19102 ± 77	18318 ± 85	17353 ± 78
	GPU (sent/s)	881.6 ± 2.9	1097.0 ± 6.3	1008.3 ± 4.1	967.0 ± 4.5	916.0 ± 4.1
ta	CPU (tok/s)	1199 ± 2	2460 ± 4	1910 ± 2	1575 ± 2	1342 ± 2
	CPU (sent/s)	72.4 ± 0.1	148.4 ± 0.2	115.2 ± 0.1	95.0 ± 0.1	81.0 ± 0.1
	GPU (tok/s)	16507 ± 353	19658 ± 74	19519 ± 67	18402 ± 73	17835 ± 173
	GPU (sent/s)	995.9 ± 21.3	1186.0 ± 4.5	1177.6 ± 4.1	1110.2 ± 4.4	1076.0 ± 10.5
ug	CPU (tok/s)	1111 ± 1	2394 ± 4	1892 ± 1	1480 ± 2	1275 ± 1
	CPU (sent/s)	96.8 ± 0.1	208.6 ± 0.3	164.9 ± 0.1	128.9 ± 0.2	111.1 ± 0.1
	GPU (tok/s)	18912 ± 120	22735 ± 62	22075 ± 61	20496 ± 41	19859 ± 126
	GPU (sent/s)	1647.7 ± 10.5	1980.8 ± 5.4	1923.3 ± 5.3	1785.8 ± 3.6	1730.2 ± 11.0
wo	CPU (tok/s)	1337 ± 1	2671 ± 4	2147 ± 4	1714 ± 2	1498 ± 2
	CPU (sent/s)	60.4 ± 0.1	120.7 ± 0.2	97.0 ± 0.2	77.4 ± 0.1	67.7 ± 0.1
	GPU (tok/s)	19809 ± 85	24189 ± 97	21385 ± 41	20635 ± 78	20363 ± 238
	GPU (sent/s)	895.0 ± 3.8	1092.8 ± 4.4	966.2 ± 1.9	932.3 ± 3.5	920.0 ± 10.8
avg	CPU (tok/s)	1146 ± 23	2550 ± 41	1920 ± 34	1531 ± 29	1302 ± 25
	CPU (sent/s)	68.0 ± 2.2	153.4 ± 5.7	114.6 ± 3.9	91.2 ± 3.0	77.4 ± 2.5
	GPU (tok/s)	18939 ± 139	22903 ± 156	21765 ± 145	20644 ± 136	19572 ± 125
	GPU (sent/s)	1135.4 ± 27.4	1375.2 ± 34.0	1311.9 ± 34.0	1241.5 ± 31.1	1175.4 ± 28.8

Table 6: Speeds with batch-size 4096.