Pruning Neural Machine Translation for Speed Using Group Lasso

Maximiliana Behnke
University of Edinburgh
maximiliana.behnke@ed.ac.uk

Kenneth Heafield
University of Edinburgh
kheafiel@inf.ed.ac.uk

Abstract

Unlike most work on pruning neural networks, we make inference faster. Group lasso regularisation enables pruning entire rows, columns or blocks of parameters that result in a smaller dense network. Because the network is still dense, efficient matrix multiply routines are still used and only minimal software changes are required to support variable layer sizes. Moreover, pruning is applied during training so there is no separate pruning step. Experiments on top of English→German models, which already have state-of-the-art speed and size, show that two-thirds of feedforward connections can be removed with 0.2 BLEU loss. With 6 decoder layers, the pruned model is 34% faster; with 2 tied decoder layers, the pruned model is 14% faster. Pruning entire heads and feedforward connections in a 12-1 encoder-decoder architecture gains an additional 51% speed-up. These push the Pareto frontier with respect to the trade-off between time and quality compared to strong baselines. In the WMT 2021 Efficiency Task, our pruned and quantised models are 1.9–2.7× faster at the cost 0.9–1.7 BLEU in comparison to the unoptimised baselines. Across language pairs, we see similar sparsity patterns: an ascending or U-shaped distribution in encoder feedforward and attention layers and an ascending distribution in the decoder.

1 Introduction

Making transformer-based machine translation models (Vaswani et al., 2017) faster and smaller is a common requirement for server and mobile deployment. We focus on pruning methods that actually improve speed upon strong baselines. There is a variety of work on pruning individual parameters (See et al., 2016; Brix et al., 2020), structures like attention heads (Voita et al., 2019; Behnke and Heafield, 2020), and even whole layers (Sajjad et al., 2020). Unfortunately, much of the prior work on pruning does not report speed or makes inference slower: Brix et al. (2020) achieved no speed-up while Yao et al. (2019) report a 87.5% sparse model took 1.6× as long using cuSPARSE. However, Gale et al. (2020) point out that coefficient-sparse kernels like cuSPARSE are highly unoptimised. Even block-sparse kernels are 1.8× slower at 70% sparsity (Gray et al., 2017) though they did eventually achieve a 1.4× speed-up with "balanced pruning". We propose pruning entire rows or columns and even whole submatrices of a tensor, resulting in a smaller dense matrix. Because the inference problem remains dense, we sidestep the need for sparse kernels to improve speed.

We use group lasso (Yuan and Lin, 2006) regularisation, which encourages groups to diminish together, during the usual training procedure. Murray et al. (2019) used group lasso to prune feedforward layers in their submission to the Efficient Translation Task at WNGT 2019 (Kim et al., 2019a). Their submissions, which "eliminate more than 25% of the model's parameters while suffering a decrease of only 1.1 BLEU" were at best 6% faster than their baseline. When tuned for the same quality loss, our method reduces size by 66% and translates 52% faster. Moreover, their submissions were slower than higher-quality competitors by an order of magnitude, whereas our baselines are state-of-the-art.

Too much work (Gu et al., 2018; Lee et al., 2020; Wang et al., 2020b) on efficiency compares a baseline unoptimized system with their optimized system, which is smaller or faster in exchange for some reduction in BLEU. What these papers fail to prove is that their method works better than existing methods that also make models smaller or faster in exchange for some reduction in BLEU like knowledge distillation (Kim and Rush, 2016), quantisation, reducing the number of layers, prior work on pruning, or simply training a smaller model. In other words, is the trade-off offered by their method any better than the trade-offs already available, regardless of the type of method? Stacking the exist-

Proceedings of the Sixth Conference on Machine Translation (WMT), pages 1074–1086
November 10–11, 2021.

ing methods produces a variety of data points with different speed and quality. The Pareto frontier is the set of data points that a practitioner would choose from: no other data point is simultaneously faster (or smaller) and of higher quality. We argue that a new method's empirical justification should advance the Pareto frontier. In this work, we build upon and compare to strong baselines to show the frontier advances.

To compare with the state-of-the-art in terms of speed and to investigate the impact this pruning makes on different languages, we build upon English→German, Spanish→English and Estonian→English student models trained with sequence-level knowledge distillation (Kim and Rush, 2016). We experiment with four architecture variations: a typical decoder with 6 layers and faster variations with shallow decoder of only 1–2 layers. We also include our experiments from the WMT 2021 Efficiency Shared Task.

Our key findings show that:

1. It is possible to prune entire nodes from feed-forward layers early during training, resulting in Pareto optimal architectures (quality vs speed). Similarly, pruning entire heads on the top of it results in even faster models.
2. Different language pairs exhibit similar structural sparsity patterns.
3. Pruning during training matches, and sometimes outperforms, retraining the pruned model from scratch.
4. Among the English→German Pareto optimal models, the notable examples include a model with a 6-layered decoder being 34% faster at the cost of 0.2 BLEU and a model with 12-1 encoder-decoder ratio gaining additional 51% speed-up costing 0.3 BLEU.
5. This type of pruning combined with quantisation gives a significant speed boost. Our models are 1.9–2.7× faster at the cost of 0.9–1.7 BLEU.

2 Related work

Extensive research to reduce workload, compress and speed-up neural machine translation models includes methods such as knowledge distillation (Kim and Rush, 2016), quantisation (Quinn and Ballesteros, 2018; Aji and Heafield, 2020), layer approximation (Kim et al., 2019b) and pruning. For the best results, they can be stacked together to train the efficient state-of-the-art model.

In their analysis, Dalvi et al. (2020) claim that 85% of transformer neurons are redundant across the network. Using transfer learning, they find the minimal set of neurons that achieve optimum performance given the task. However, that method requires a fully pretrained model to perform a brute-force search on it, making overall training time too long.

Pruning techniques are usually split into two groups: unstructured and structured. Unstructured removes individual coefficients. It is straightforward to apply and yields good quality results simultaneously, which makes it popular. Unstructured magnitude pruning, while successfully applied to NMT (See et al., 2016), often needs retraining to recover from quality damage. Moreover, it also requires an efficient matrix multiplication routine to get any speed-up besides size compression. The latest research on combining lottery ticket hypothesis with other methods (Brix et al., 2020) sparsified NMT models by 70 to 90% while losing between 0.6 to 3.3 BLEU points in quality. They used a sparse matrix representation for compression but did not report any speed gains.

On the other hand, structured pruning removes whole layers or groups of parameters, such as blocks (Narang et al., 2017), which makes it easier to optimise on hardware via a special block-sparse matrix multiply (Gray et al., 2017). We apply block sparsity, but the blocks are entire rows or columns so that the usual dense matrix multiply can be used with less overhead. Another line of work prunes entire attention heads from a model (Voita et al., 2019; Behnke and Heafield, 2020), which we also explore in our approach.

Yao et al. (2019) combine unstructured sparsity with a light structure that aims to balance parallel workloads. They introduce a specialised kernel for their structured sparsity. Our workloads are easier to balance because they retain density. The idea that different levels of coarseness can be combined may also extend to prune coefficients, rows, columns, and layers simultaneously in future work.

Wang et al. (2020c) parametrise weight matrices with low-rank factorisations and remove rank-1 components during training, which is said to better preserve linear transformation of uncompressed matrices. They report compressing a Transformer-XL language model by 90% with 1.6× speed-up during inference. Low-rank approximations preserve density, albeit at the cost of doing serial ma-

trix multiplications.

Fan et al. (2019) explored a structured dropout that allows users to prune models for inference. Unfortunately they fail short on NMT experiments. They call WMT14 en-de a 'competitive benchmark' which it has not been for many years. Most problematically, they use tokenised BLEU, which has been noted to be harmful and gives false 'boost' of multiple points on tokenised data. They do not specify the tokeniser or script they use either. Again, there do not include any report on speed or model sizes despite claiming to have much smaller models as a result.

Dodge et al. (2019) used group lasso to sparsify a variant of RNN for text classification, which is an easier task to learn than NMT. They have to train until convergence twice, which we avoid. They provide no speed or model size analysis, suggesting that there is no improvement or proper implementation.

Group lasso has also been previously used by Wuebker et al. (2018) to compress the delta between a base model and a domain adapted version of the model. They still have to run a full-size model in inference, so they have no overall speed gain. They also have to store the full base model; compression only refers to the delta. In contrast, our work makes the base model faster and smaller. The different goals also mean different groups: they focused on embeddings that update in domain adaptation while we focus on costly parts of the architecture.

Though we use the same algorithm of group lasso, our method differs in several ways from (Murray et al., 2019). We prune submatrices in addition to rows and columns, though experiments on just rows and columns show better performance than theirs. They pruned only feedforward layers; we see more speed-up from feedforward layers and additionally prune attention. Finally, we use the normal Adam optimiser (Kingma and Ba, 2014) instead of proximal gradient descent (Parikh and Boyd, 2014). Empirically, we find turning regularisation off after some training is important to quality. Overall, we achieve a much better trade-off between quality and speed/compression.

Most of the methods above need either tuning or retraining, often multiple times. They are usually treated as techniques to compress already existing models. Still, there are ongoing research efforts on training a reduced model from start to finish in one go. For example, Golub et al. (2018) pruned weights with the lowest total accumulated gradients and reduced the memory footprint to allow training much larger models than possible on available hardware. Some methods prune immediately after initialisation, in either unstructured (Lee et al., 2019) or structured (Wang et al., 2020a) way. Our method is orthogonal and is integrated into a training scheme instead.

Using regularisation to sparsify groups of parameters was introduced by Yuan and Lin (2006) and has been since then built upon in the machine learning field (Scardapane et al., 2017; Wen et al., 2016). In this paper, we use group lasso in its simplest form to achieve structural sparsity in transformer layers, focusing on inference speed of machine translation.

3 Methodology

To allow regularisation to remove parameters structurally, we need to define how we group parameters. Depending on which matrix it is, we treat parameters in its rows, columns or heads as one entity to be pruned together. Thus, we apply a group lasso over them. A bias term, if necessary, is treated as a part of regularised groups. We want such a sparsity pattern to emerge early into training so that there is no need to retrain or tune it later.

Group lasso regularisation

Given a matrix w split into non-overlapping groups of parameters G, the group lasso is defined as:

$$R(w) = \sum_{g=1}^{G} \gamma \left\| w_g \right\|_2 = \sum_{g=1}^{G} \gamma \sqrt{\sum_{j=1}^{|G_g|} \left(w_g^j \right)^2}. \quad (1)$$

This penalty term applies L_2 norm over the parameters in each group to force them to go towards 0 together, with L_1 on top of it to enforce overall sparsification. γ is a scalar that orthonormalises groups of different sizes (Simon and Tibshirani, 2012), scaling by the number of elements in a group $\sqrt{d_g}$. If regularising only rows and columns, all groups are of the same size. However, in later experiments, we also regularise whole attention heads alongside individual feedforward connections.

In the end, the penalty for each layer is added to the cost function and scaled by λ and averaged

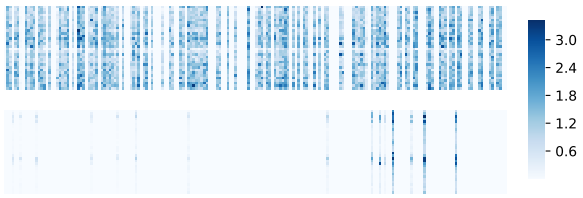

Figure 1: An example of block-sparse matrices in the first layer of decoder (top) and encoder (bottom) pruned by group lasso regularisation.

over words in a batch along with cross-entropy:

$$\frac{1}{|B|}(\sum_{x \in B} CE(x) + \lambda * \sum_{w \in W} R(w)) \,. \quad (2)$$

Initially we experimented with 8x8 blocks in group lasso. However, this approach removed entire rows and columns that correspond to pruning connections. Figure 1 shows an example of this effect on parameter matrices. When an entire row or column is zero, it can be deleted to form a smaller dense matrix. If an input connection is ignored and not used elsewhere, it can be removed from the upstream matrix. If an output connection is constant, the constant can be folded into downstream bias. These optimisations are typically discovered automatically by regularisation. We mainly focus on pruning feedforward layers, but later include experiments that prune attention heads as well.

4 Setup

4.1 Data & architectures

We concentrate on three language pairs: English→German, Spanish→English and Estonian→English. We use knowledge distillation (Kim and Rush, 2016) under teacher-student regime.

In English→German, we follow the Workshop on Neural Generation and Translation 2020 Efficiency shared task (WNGT2020) [1] under the WMT 2019 data condition (Barrault et al., 2019). As a teacher, we use a WMT 2019 system submitted by Microsoft to news translation task (Junczys-Dowmunt, 2019). It is an ensemble of four deep transformer-big models (Vaswani et al., 2017) with 12 layers in encoder and decoder, embedding size of 1024 and feedforward size 4096 and 8 attention heads. For Spanish→English and Estonian→English, we use teachers provided by the Bergamot,[2] which is an ensemble of two similar architectures but with 6 layers instead.

We start with strong student baselines, which are already very small and fast, closely following the latest trends set by WNGT2020.

Students for all language pairs have an embedding dimension of 256 and feedforward of 1536, based on "tiny" architecture from Kim et al. (2019a). The attention has 8 heads in each layer except for decoder self-attention, which is replaced by a faster SSRU (Simpler Simple Recurrent Unit) (Kim et al., 2019b). The models use a shared vocabulary of 32,000 subword units generated by SentencePiece (Kudo and Richardson, 2018) and translate using shortlists of top 50 words.

We tried different configurations of layers to investigate trade-offs between them and potential bottlenecks. We describe each architecture by layer number in encoder and decoder and whether the decoder layers are tied. Thus, we investigate the following architectures (chronologically): "6-2tied", "6-6", "12-1" and "6-2".

Other training hyperparameters were Marian defaults for training a transformer base model.[3] We used dynamic batching, filling a 10GB workspace on each of 4 GPUs, resulting in about 71,000 words per batch in a "6-2tied" student and about 46,000 words per batch in a "6-6" student. As is more effective in the teacher-student regime, we did not use dropout or label smoothing. We use the Adam optimiser (Kingma and Ba, 2014).

The English→German models were trained on 13M sentences of available parallel data, using the concatenated English-German WMT testsets from 2016–2018 as a development set. The Spanish→English students were trained on 242M sentences which included about 15M of mixed forward- and backtranslations. We used a WMT13 testset for development. Estonian→English students were trained on 132M sentences which included about 30M of mixed forward- and backtranslations, and WMT18/dev was used for development.

We trained and decoded all our models using the Marian NMT toolkit (Junczys-Dowmunt et al., 2018). We evaluate quality and speed on 1 CPU core. In order to expand beyond BLEU and incentivise others to do the same, we additionally

[1] https://sites.google.com/view/wngt20

[2] https://github.com/browsermt/students

[3] Available via `--task transformer-base`.

evaluate with chrF (Popović, 2015) and COMET[4] (Rei et al., 2020) as well. We use SacreBLEU (Post, 2018) for BLEU and chrF. Training progressed until BLEU stopped improving for 20 consecutive validations. The checkpoint with the highest BLEU score was selected.

4.2 Training regime

Our training regime for all of ours models has three phases:

1. Pretrain for 25k batches.
2. Train with a regulariser for 250k batches.
3. Remove rows/columns with a sum less than $1e-5$, collapse a model and then train without regularisation until convergence.

It is well known that initial transformer training is problematic and sensitive to model hyperparameters (Nguyen and Salazar, 2019; Aji et al., 2019; Liu et al., 2020). A transformer starts training with 1–2 BLEU and quickly jumps over to 15–30 or more within a short training period, then slows down. Thus, we start pruning after BLEU improvement slowed down to be less than 1 BLEU point in a single checkpoint. This way, we avoid any potential damage to a model during the critical initial period. In this case, we pretrain for 25k batches.

Next, we had to decide how long to regularise our model to achieve a good trade-off between quality and sparsity levels. We started with regularising a model until convergence. As shown in Fig. 2, most of the parameters are already pruned in the first half of the training. Since students require significantly more updates to train than standard models, we want to give a model enough time to sparsify and converge without any constraints. For this reason, we split an average training time into two halves: the first with pruning, the second without it with normal convergence. We found that switching the regulariser off at some point is less aggressive and allows a model to recover some of its lost quality. We chose 250k updates as a pivot as it is about halfway to when the model has begun stalling in Fig. 2.

After each step, we copy the latest checkpoint and start a fresh training round. Thus, all training hyperparameters are reset. We checked and found no additional advantage to our baselines by refreshing learning rate scheduling or Adam optimiser. We do so to avoid partially retraining the same settings during our development phase, but Brix et al. (2020) found it beneficial in their pruning scheme.

[4]We used the default 'wmt20-comet-da' metric model.

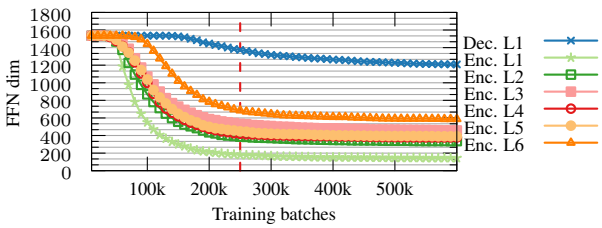

Figure 2: An example of pruning FFN layers in an English→German tied model ($\lambda = 0.5$). About "halfway" through, most parameters are already removed.

Reg. $\lambda \longrightarrow$	Base	0.3	0.4	0.5	0.7	1.0
BLEU	37.2	37.8	37.4	36.8	36.5	36.1
chrF	63.3	63.8	63.4	63.1	62.8	62.5
COMET	49.7	51.1	50.4	48.9	47.2	46.0
FFN sparsity	0%	45%	63%	73%	85%	92%
Size (MB)	61	51	47	45	43	41
WPS	2404	2613	2748	3067	3215	3420
Speed-up	1.00	1.09	1.14	1.28	1.34	1.42

(a) With pruned encoder + decoder.

Reg. $\lambda \longrightarrow$	Base	0.3	0.4	0.5	0.7	1.0
BLEU	37.2	37.6	37.3	36.8	36.8	36.4
chrF	63.3	63.4	63.4	63.1	62.9	62.8
COMET	49.7	50.4	49.8	49.8	48.3	47.8
FFN sparsity	0%	48%	64%	72%	79%	82%
Size (MB)	61	50	47	45	44	43
Words per sec	2404	2748	2916	2929	3054	3096
Speed-up	1.00	1.14	1.21	1.22	1.27	1.29

(b) With pruned encoder.

Table 1: The evaluation of English→German "6-2tied" students pruned using group lasso.

5 Experiments

5.1 Pruning "6-2tied" models (English→German)

We begin our experiments with the state-of-the-art English→German student with a tied decoder (Bogoychev et al., 2020). This is their fastest architecture and we want to investigate how much further it can be pushed in that regard. In terms what is a typical difference in inference speed, a tiny distilled model is usually at least 20× faster than its teacher (Germann et al., 2020).

We investigate two scenarios: pruning both the encoder and decoder (Tab. 1a) or pruning only the encoder (Tab. 1b). The models were trained with the regularisation term $\lambda \in \{0.3, 0.4, 0.5, 0.7, 1.0\}$.

Since there is only one layer's worth of decoder parameters, the regularisation is reluctant to re-

Reg. $\lambda \longrightarrow$		Base	0.3	0.4	0.5	0.7	1.0
BLEU	Pruned	37.2	37.8	37.4	36.8	36.5	36.1
	Reinit	-	37.1	36.5	36.5	36.1	35.3
chrF	Pruned	63.3	63.8	63.4	63.1	62.8	62.5
	Reinit	-	63.2	62.8	62.7	62.3	62.0
COMET	Pruned	49.7	51.1	50.4	48.9	47.2	46.0
	Reinit	-	48.8	47.3	47.3	45.7	42.7

Table 2: The average BLEU of English→German "6-2tied" students with pruned encoder and decoder (*Pruned*), compared to the same architecture trained from scratch (*Reinit*).

Reg. $\lambda \longrightarrow$	Base	0.1	0.15	0.2	0.3	0.4	0.5	1.0
Enc. L1	1536	821	453	259	100	56	35	12
Enc. L2	1536	932	506	327	166	93	65	22
Enc. L3	1536	1009	502	298	129	79	47	10
Enc. L4	1536	1213	663	384	147	70	41	11
Enc. L5	1536	1149	646	392	186	119	87	16
Enc. L6	1536	1366	919	610	322	208	148	43
Dec. L1	1536	542	231	121	43	15	8	1
Dec. L2	1536	836	435	259	108	50	27	4
Dec. L3	1536	448	227	129	49	26	16	3
Dec. L4	1536	1064	623	422	229	142	111	25
Dec. L5	1536	1528	1260	876	450	276	198	49
Dec. L6	1536	1536	1536	1536	1517	1216	835	178
BLEU	38.5	38.7	38.6	38.3	37.7	37.6	37.4	36.8
chrF	64.2	64.5	64.4	64.1	63.7	63.6	63.5	63.1
COMET	54.8	55.7	54.7	53.8	52.9	52.8	51.9	49.6
FFN sparsity	0%	31%	55%	69%	81%	87%	91%	98%
Size (MB)	83	72	63	58	54	52	50	48
WPS	1225	1377	1543	1639	1741	1827	1867	1976
Speed-up	1.00	1.12	1.26	1.34	1.42	1.49	1.52	1.61

Table 3: The evaluation of English→German "6-6" students pruned with group lasso on 1 CPU core, with the distribution of parameters left in each layer.

move any parameters from it (Tab. 1a). A similar effect was observed by Behnke and Heafield (2020).

Because only the encoder was pruned, the speed-up is relatively small. Still, we successfully prune from half up to two-thirds of feedforward parameters with ± 0.2 BLEU change with 9–14% faster inference. In the most extreme case, it gains 42% speed-up at the cost of 1.5 BLEU.

To investigate whether this pruning just found a new type of architecture structure, we reinitialise and retrain the smaller pruned models from scratch with reduced dimensions. As seen in Tab. 2, the same models achieve noticeably worse translation quality when trained from the get-go in comparison to careful pruning.

Next, we concentrate on pruning encoder only (Tab. 1b). The model with about 50% of feedforward parameters removed is 14% faster with no change to the overall quality. The most aggressive pruning removes almost all feedforward layers in the encoder at the loss of 1.2 BLEU.

Reg. $\lambda \longrightarrow$		Base	0.1	0.15	0.2	0.3	0.4	0.5	1.0
BLEU	Pruned	38.5	38.7	38.6	38.3	37.7	37.6	37.4	36.8
	Reinit	-	38.1	38.0	37.6	37.3	37.2	37.0	36.6
chrF	Pruned	64.2	64.5	64.4	64.1	63.7	63.6	63.5	63.1
	Reinit	-	64.0	63.9	63.7	63.6	63.4	63.3	63.0
COMET	Pruned	54.8	55.7	54.7	53.8	52.9	52.8	51.9	49.6
	Reinit	-	53.8	53.3	52.3	51.5	51.4	49.7	49.1

Table 4: The evaluation of English→German "6-6" students with pruned both encoder and decoder (*Pruned*), compared to the same architecture trained from scratch (*Reinit*).

Reg. $\lambda \rightarrow$	Base	0.1	0.15	0.2	0.3	0.4	0.5	1.0
BLEU	37.3	36.9	36.8	36.6	36.3	36.3	36.1	35.9
chrF	62.6	62.4	62.3	62.3	62.0	62.0	61.9	61.8
COMET	58.1	57.3	56.8	56.2	55.1	55.4	54.6	54.3
FFN sparsity	0%	48%	67%	77%	86%	91%	94%	98%
Size (MB)	83	59	66	55	52	50	49	48
WPS	1407	1655	1811	1891	2017	2071	2112	2204
Speed-up	1.00	1.18	1.29	1.34	1.43	1.47	1.50	1.57

Table 5: The evaluation of Spanish→English "6-6" students pruned with group lasso on 1 CPU core averaged over WMT12–13.

In both cases, only one-third of feedforward parameters is required to perform within a small margin of BLEU loss (-0.2 to -0.3). Removing more than that results in progressively worse quality.

5.2 Pruning "6-6" models (English→German, Spanish→English)

The models with "6-6" architecture were trained with $\lambda = \{0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 1.0\}$ pruning both encoder and decoder. The results are presented in Tab. 4. Additionally, we show the number of remaining rows/columns left in each layer, along with sparsity and inference speed-up.

The pruned models behave similarly to the smaller models in Tab. 1. The regularised models are of a better translation quality than the same architectures trained from scratch (Tab. 6). Similarly, it is possible to remove two-thirds of all feedforward parameters with -0.2 BLEU and +34% speed-up. Pruning more than that causes a notable step down in quality, which may not be worth aiming for since the "6-2tied" architectures outperform that loss. The sparsity pattern follows an ascending trend in both encoder and decoder layers.

We repeat the experiments but this time with Spanish→English using the same "6-6" architecture. The models were trained with regularisation $\lambda = \{0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 1.0\}$. The results are presented in Tab 5.

Reg. $\lambda \longrightarrow$		Base	0.1	0.15	0.2	0.3	0.4	0.5	1.0
BLEU	Pruned	37.3	36.9	36.8	36.6	36.3	36.3	36.1	35.9
	Reinit	-	37.0	36.7	36.5	36.3	36.2	36.2	35.8
chrF	Pruned	62.6	62.4	62.3	62.3	62.0	62.0	61.9	61.8
	Reinit	-	62.5	62.2	62.1	62.0	61.9	61.9	61.7
COMET	Pruned	58.1	57.3	56.8	56.2	55.1	55.4	54.6	54.3
	Reinit	-	57.3	56.7	55.9	55.1	54.6	54.5	53.0

Table 6: The evaluation of Spanish→English "6-6" students with pruned encoder and decoder (*Pruned*), compared to the same architecture trained from scratch (*Reinit*) averaged over WMT12–13.

Reg. $\lambda \longrightarrow$	Base	0.2	0.3	0.4	0.5	0.7
BLEU	38.2	37.9	37.3	37.0	37.0	36.6
chrF	63.9	63.6	63.2	62.9	62.9	62.5
COMET	49.5	49.6	48.2	46.5	45.5	44.6
Att. sparsity	0%	48%	56%	58%	57%	59%
FFN sparsity	0%	63%	76%	81%	84%	87%
Size (MB)	85	54	48	45	44	43
WPS	1930	2918	3029	3430	3446	3485
Speed-up	1.00	1.51	1.57	1.78	1.79	1.81

Table 7: The evaluation of English→German "12-1" students pruned with group lasso on 1 CPU core.

The only differences between German and Spanish experiments are the languages involved and the scale of the training data: Spanish students trained on a 19× larger corpus. Experiments of such scale are still widely unexplored in machine translation, raising the question of whether known methods are beneficial in real-life scenarios. Most pruning papers use English→German models under WMT14 constraints (Bojar et al.), which is only 4.5M sentences (See et al., 2016; Brix et al., 2020; Hsu et al., 2020), sometimes branching into different languages such as Russian or French in a similar scope (Voita et al., 2019; Kasai et al., 2020). For that reason, we sampled 13M from 242M sentences (the same amount as English→German) and repeated the experiments. In the end, we came to similar conclusions, meaning that the Spanish subpar results are not related to architecture or data size. The reinitialised models (Tab. 6) on full dataset achieve comparable quality to their pruned counterparts. We conclude that in some cases, structural pruning serves as an architecture search method to find Pareto optimal quality-speed trade-off.

5.3 Pruning "12-1" models (English→German)

Kasai et al. (2020) argues that shifting layers from decoder to encoder makes a model much faster at almost no cost in translation quality. Their experiments have shown that 12-1 encoder-decoder layer proportions perform as good as 6-6. Pruning an already reduced decoder may cause a bottleneck that damages quality too much. However, if we shift most of the workload into an encoder, we can focus on pruning it exclusively.

This time we prune attention layers as well. Pruning attention structurally is more tricky — you cannot remove individual connections easily due to how matrix multiplications perform their routine. The only option is to remove respective heads or an entire layer. To keep it simple, we regularise individual connections and remove an entire attention head if at least half of its connections are dead (its rows/columns $< 1e-5$). The results are in Tab. 7, with an extended version of it in the appendix.

In terms of quality and speed-up, it outperforms other models presented so far. This type of pruning was not aggressive on attention, preferring to prune feedforward layers instead, indicating that attention connections perform more critical work in a model. At the small cost of 0.3 BLEU, the model is 51% faster than the baseline.

		Baseline		rc+heads		rc+rc	
		FFN	Heads	FFN	Heads	FFN	Heads
Encoder 1		1536	8	579	0	210	7
Encoder 2		1536	8	793	1	552	1
Encoder 3		1536	8	959	0	712	3
Encoder 4		1536	8	913	0	459	6
Encoder 5		1536	8	1212	3	708	4
Encoder 6		1536	8	1523	2	1033	8
Decoder 1		1536	8	1536	2	1535	7
Decoder 2		1536	8	1536	7	1536	8
BLEU	Pruned	31.5		29.8		30.4	
	Reinit	-		28.5		30.3	
chrF	Pruned	58.4		57.0		57.6	
	Reinit			56.0		57.5	
COMET	Pruned	54.8		49.9		53.0	
	Reinit	-		46.8		50.7	
Time		21.57		15.16		18.9	
WPS		1414		2012		1614	
Speed-up		1.00		1.42		1.14	

Table 8: The WMT18 testset evaluation of Estonian→English "6-2" students pruned with group lasso on 1 CPU core with the same architectures trained from scratch (*Reinit*).

5.4 Pruning "6-2" models with head lasso (Estonian→English)

Finally, we train Estonian→English models, pruning both feedforward and attention layers across the whole model. We do not sweep parameters, choosing $\lambda = 0.3$. The results are in Tab. 8.

This time we try two options:

- regularising individual connections and then removing heads with more than half of con-

	BLEU		COMET		Sparsity		
	WMT20	WMT21	WMT20	WMT21	Att.	FFN	Speed (s)
12-1.tiny	36.1	27.6	48.2	41.9	0%	0%	19.2
+ head-lasso pruning	34.7	27.0	42.9	38.8	3%	75%	14.5
+ 8bit quantisation	33.9	26.2	38.8	33.6	3%	75%	9.3
+ 8bit finetuning	34.1	26.7	39.8	33.0	3%	75%	9.3
+ rowcol-lasso pruning	33.8	26.3	39.3	34.2	68%	73%	11.6
+ 8bit quantisation	32.9	25.6	33.7	28.7	68%	73%	6.9
+ 8bit finetuning	32.9	26.0	35.7	31.3	68%	73%	7.1
12-1.micro	35.4	27.6	46.2	40.2	0%	0%	17.1
+ head-lasso pruning	34.6	26.7	43.0	35.4	3%	72%	14.1
+ 8bit quantisation	33.4	26.0	36.7	31.2	3%	72%	9.2
+ 8bit finetuning	33.7	26.5	38.3	33.3	3%	72%	9.2
+ rowcol-lasso pruning	34.3	26.4	40.7	35.1	60%	59%	12.0
+ 8bit quantisation	32.7	25.5	34.2	29.1	60%	59%	7.5
+ 8bit finetuning	33.3	25.9	35.2	30.5	60%	59%	7.5

Table 9: 8-bit model performance. BLEU score is calculated from WMT20. Speed is measured on a single core CPU with a mini-batch of 32. We experimented with two types of pruning. Head pruning removes entire heads. Row and column pruning removes entire rows or columns of matrices, resulting in a smaller matrix.

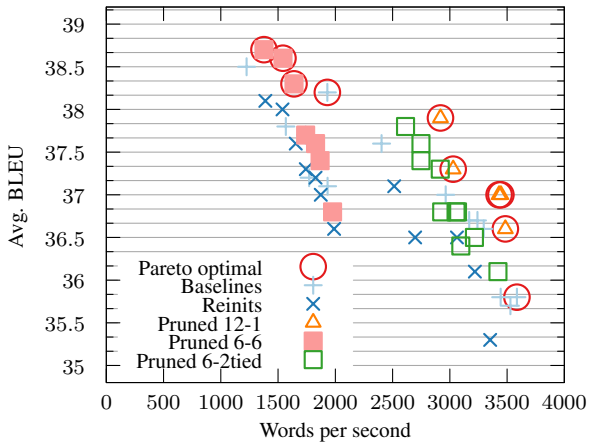

Figure 3: Pareto trade-off between average translation quality and average translation time for English→German students of different architectures.

nections inactive (rc+rc = rows/columns in FFN and attention both)

- regularising entire heads with group lasso (rc+heads = rows/columns in FFN, heads in attention)

Due to how the penalty is scaled with γ in Eq. 1, the regularisation of entire heads is much more aggressive towards them, removing some layers entirely, which we skip during inference. Both pruning methods perform within a -0.1 to -0.3 BLEU difference compared to the same architecture trained from scratch. However, despite only 0.1 BLEU difference, the same model loses 2.3 COMET points, further validating the fact that training from scratch is subpar. Those results show that there is a potential in regularising larger structures and even entire layers as a way of architecture searching. We leave the improvement of the method for future work.

6 Pareto trade-off (English→German)

In this section, we look at the Pareto trade-off between the translation quality and speed for all our English→German models (Fig. 3). To be fair in our comparison, we trained several simpler baselines with uniformly smaller feedforward dimensions set to $\{768, 384, 192, 96\}$. For"12-1" we additionally set heads per layer to 4 to roughly reflect sparsity percentages of pruned models.

We pit against each other the said baselines, the pruned models and their reinitialised counterparts. Naturally, the models with 6 decoder layers are slower but of a higher quality. However, it is better to switch to fewer decoder layers than to prune too far. Our experiments on "12-1" architecture show that its pruned models outperform all others (including all simpler baselines), being a leader in the Pareto frontier.

7 WMT2021 Efficiency Shared Task

To put our method to the final test, we participated in WMT2021 Efficiency Task[5] (Behnke et al., 2021). Under the task constraints, we trained, pruned and quantised 12–1.tiny and 12–1.micro architectures. We tried two pruning settings, following the directions set in Sect. 5.3 and 5.4: *rowcol-*

[5] http://www.statmt.org/wmt21/efficiency-task.html

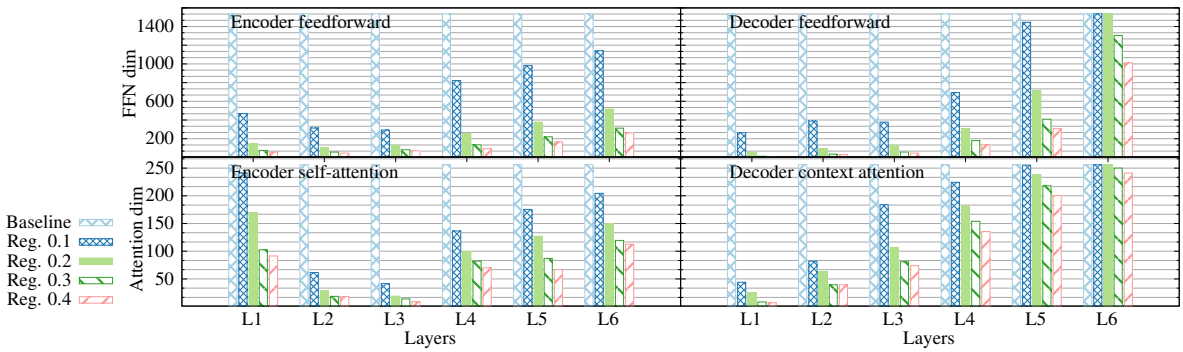

Figure 4: The distribution of feedforward and attention connections in Spanish→English "6-6" pruned students.

lasso and *head-lasso*. Both prune feedforward and attention layers in the encoder. *rowcol-lasso* regularised individual connections and removed an entire attention head if at least half of its connections are dead. *head-lasso* applied lasso to a whole head submatrix. Due to the scale of the task, we had no opportunity to grid-search for the best pruning hyperparameters, thus the experiments are as close to 'out-of-the-box' usage as they can be. We used $\lambda = 0.5$ for both methods. The models were pretrained for 50k updates and regularised for 150k, after which the models were sliced and trained until convergence. The results are presented in Tab. 9.

head-lasso left attention layers almost completely unpruned, focusing on removing connections from feedforward layers instead. *rowcol-lasso* was much more aggressive in both layers at the cost of quality. To further optimise the models, they were quantised to 8bit. However, we observe that the smaller a model is, the larger the quality drop after its quantisation. Additional finetuning allows us to recover at least partially from the quantisation damage. Evaluating on the latest testset WMT21, our pruned models are 1.2–1.7× faster at the cost of 0.6–1.3 BLEU. With quantisation, those models are 1.9–2.7× faster losing 0.9–1.7 BLEU in comparison to the unpruned and unquantised baselines.

8 Analysis

To analyse sparse architecture patterns, we experiment with Spanish→English models. We prune individual connections in all attention and feedforward layers. In Fig. 4, we present the distribution of remaining parameters for the baseline and the models regularised with $\lambda = \{0.1, 0.2, 0.3, 0.4\}$.

Since the decoder self-attention is replaced with SSRU (Kim et al., 2019b), we only show two "pairs" of parameters: encoder self-attention and decoder context attention, with their feedforward counterparts.

Both encoder layer types follow a similar sparsity pattern making a "U-shape", with the second and third ones being the most aggressively pruned. On the other hand, the decoder parameters are pruned less and less with each subsequent layer. This arrangement of parameters is identical to that exhibited by pruned attention heads in Behnke and Heafield (2020). In that paper, the attention in the encoder also prunes middle layers, and the context attention retains more heads in further layers. It strongly indicates that the decoder prefers to attend to itself first and confront context later.

The Estonian architectures, in which we pruned entire attention heads, exhibit a roughly similar structure. For us, this is a strong signal that structural pruning with its architecture search may have a broader generalisation.

9 Conclusions

This paper investigated the structural pruning of a transformer incorporated into a typical training routine. We focused on shredding nodes in feedforward layers and whole attention heads as training progresses. Our experiments on knowledge-distilled models with deep and shallow decoders have shown that this type of pruning leads to Pareto optimal architectures in quality and speed. Moreover, it converges in just one "pass" like a baseline since there is no need to repeat an entire or a part of the training. The resulting sparsity patterns are similar across different languages, with the first and middle layers being the most prioritised during pruning. On the other hand, our experiments on pruning both feedforward and attention layers reveal that some of them, such as the last context attention layer, distinctively avoid being pruned.

Acknowledgments

This work was supported by the Engineering and Physical Sciences Research Council (grant EP/L01503X/1), EPSRC Centre for Doctoral Training in Pervasive Parallelism at the University of Edinburgh, School of Informatics.

This work has been performed using resources provided by the Cambridge Tier-2 system operated by the University of Cambridge Research Computing Service (www.hpc.cam.ac.uk) funded by EPSRC Tier-2 capital grant EP/P020259/1.

References

Alham Fikri Aji and Kenneth Heafield. 2020. Compressing neural machine translation models with 4-bit precision. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, pages 35–42, Online. Association for Computational Linguistics.

Alham Fikri Aji, Kenneth Heafield, and Nikolay Bogoychev. 2019. Combining global sparse gradients with local gradients in distributed neural network training. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3626–3631, Hong Kong, China. Association for Computational Linguistics.

Loïc Barrault, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, Christof Monz, Mathias Müller, Santanu Pal, Matt Post, and Marcos Zampieri. 2019. Findings of the 2019 conference on machine translation (WMT19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61, Florence, Italy. Association for Computational Linguistics.

Maximiliana Behnke, Nikolay Bogoychev, Alham Fikri Aji, Heafield Heafield, Graeme Nail, Qianqian Zhu, Svetlana Tchistiakova, Jelmer van der Linde, Pinzhen Chen, Sidharth Kashyap, and Roman Grundkiewicz. 2021. Efficient machine translation with model pruning and quantization. In *Proceedings of the Six Conference on Machine Translation*, Online. Association for Computational Linguistics.

Maximiliana Behnke and Kenneth Heafield. 2020. Losing heads in the lottery: Pruning transformer attention in neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2664–2674, Online. Association for Computational Linguistics.

Nikolay Bogoychev, Roman Grundkiewicz, Alham Fikri Aji, Maximiliana Behnke, Kenneth Heafield, Sidharth Kashyap, Emmanouil-Ioannis Farsarakis, and Mateusz Chudyk. 2020. Edinburgh's submissions to the 2020 machine translation efficiency task. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, pages 218–224, Online. Association for Computational Linguistics.

Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna.

Christopher Brix, Parnia Bahar, and Hermann Ney. 2020. Successfully applying the stabilized lottery ticket hypothesis to the transformer architecture. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3909–3915, Online. Association for Computational Linguistics.

Fahim Dalvi, Hassan Sajjad, Nadir Durrani, and Yonatan Belinkov. 2020. Analyzing redundancy in pretrained transformer models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4908–4926, Online. Association for Computational Linguistics.

Jesse Dodge, Roy Schwartz, Hao Peng, and Noah A. Smith. 2019. RNN architecture learning with sparse regularization. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1179–1184, Hong Kong, China. Association for Computational Linguistics.

Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. *CoRR*, abs/1909.11556.

Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. 2020. Sparse gpu kernels for deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press.

Ulrich Germann, Roman Grundkiewicz, Martin Popel, Radina Dobreva, Nikolay Bogoychev, and Kenneth Heafield. 2020. Speed-optimized, compact student models that distill knowledge from a larger teacher model: the UEDIN-CUNI submission to the WMT 2020 news translation task. In *Proceedings of the Fifth Conference on Machine Translation*, pages 191–196, Online. Association for Computational Linguistics.

Maximilian Golub, Guy Lemieux, and Mieszko Lis. 2018. Dropback: Continuous pruning during training. *CoRR*, abs/1806.06949.

Scott Gray, Alec Radford, and Diederik P Kingma. 2017. Gpu kernels for block-sparse weights.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.

Yi-Te Hsu, Sarthak Garg, Yi-Hsiu Liao, and Ilya Chatsviorkin. 2020. Efficient inference for neural machine translation. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 48–53, Online. Association for Computational Linguistics.

Marcin Junczys-Dowmunt. 2019. Microsoft translator at WMT 2019: Towards large-scale document-level neural machine translation. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 225–233, Florence, Italy. Association for Computational Linguistics.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. 2018. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2020. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. 2019a. From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 280–288, Hong Kong. Association for Computational Linguistics.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. 2019b. From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 280–288, Hong Kong. Association for Computational Linguistics.

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

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71.

Jason Lee, Raphael Shu, and Kyunghyun Cho. 2020. Iterative refinement in the continuous space for non-autoregressive neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1006–1015, Online. Association for Computational Linguistics.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. 2019. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. 2020. Understanding the difficulty of training transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5747–5763, Online. Association for Computational Linguistics.

Kenton Murray, Brian DuSell, and David Chiang. 2019. Efficiency through auto-sizing: Notre Dame NLP's submission to the WNGT 2019 efficiency task. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 297–301, Hong Kong. Association for Computational Linguistics.

Sharan Narang, Eric Undersander, and Gregory F. Diamos. 2017. Block-sparse recurrent neural networks. *CoRR*, abs/1711.02782.

Toan Q. Nguyen and Julian Salazar. 2019. Transformers without tears: Improving the normalization of self-attention. *CoRR*, abs/1910.05895.

Neal Parikh and Stephen Boyd. 2014. Proximal algorithms. *Found. Trends Optim.*, 1(3):127–239.

Maja Popović. 2015. chrF: character n-gram f-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Jerry Quinn and Miguel Ballesteros. 2018. Pieces of eight: 8-bit neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 114–120, New Orleans - Louisiana. Association for Computational Linguistics.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2020. Poor man's bert: Smaller and faster transformer models.

Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. 2017. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89.

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

Noah Simon and Robert Tibshirani. 2012. Standardization and the group lasso penalty. *Statistica Sinica*, 22(3):983.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. 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*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. 2020a. Picking winning tickets before training by preserving gradient flow.

Yong Wang, Longyue Wang, Victor Li, and Zhaopeng Tu. 2020b. On the sparsity of neural machine translation models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1060–1066, Online. Association for Computational Linguistics.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2020c. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162, Online. Association for Computational Linguistics.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 2082–2090, Red Hook, NY, USA. Curran Associates Inc.

Joern Wuebker, Patrick Simianer, and John DeNero. 2018. Compact personalized models for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 881–886, Brussels, Belgium. Association for Computational Linguistics.

Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang, and Lanshun Nie. 2019. Balanced sparsity for efficient dnn inference on gpu. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:5676–5683.

Ming Yuan and Yi Lin. 2006. Model selection and estimation in regression with grouped variables. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 68:49–67.

10 Appendix

Reg. $\lambda \longrightarrow$	Base	0.3	0.4	0.5	0.7	1.0
Enc. L1	1536	330	179	113	52	29
Enc. L2	1536	633	387	266	135	57
Enc. L3	1536	882	533	351	175	82
Enc. L4	1536	738	420	269	137	66
Enc. L5	1536	720	447	309	179	100
Enc. L6	1536	1079	686	488	293	166
Dec. L1	1536	1534	1373	1072	626	339
BLEU	37.2	37.8	37.4	36.8	36.5	36.1
chrF	63.3	63.8	63.4	63.1	62.8	62.5
COMET	49.7	51.1	50.4	48.9	47.2	46.0
FFN sparsity	0%	45%	63%	73%	85%	92%
Size (MB)	61	51	47	45	43	41
WPS	2404	2613	2748	3067	3215	3420
Speed-up	1.00	1.09	1.14	1.28	1.34	1.42

(a) With pruned encoder + decoder.

Reg. $\lambda \longrightarrow$	Base	0.3	0.4	0.5	0.7	1.0
Enc. L1	1536	310	164	98	42	24
Enc. L2	1536	597	372	239	104	50
Enc. L3	1536	831	480	302	143	59
Enc. L4	1536	692	376	234	115	48
Enc. L5	1536	664	400	253	142	73
Enc. L6	1536	948	575	399	209	112
Dec. L1	1536	1536	1536	1536	1536	1536
BLEU	37.2	37.6	37.3	36.8	36.8	36.4
chrF	63.3	63.4	63.4	63.1	62.9	62.8
COMET	49.7	50.4	49.8	49.8	48.3	47.8
FFN sparsity	0%	48%	64%	72%	79%	82%
Size (MB)	61	50	47	45	44	43
Words per sec	2404	2748	2916	2929	3054	3096
Speed-up	1.00	1.14	1.21	1.22	1.27	1.29

(b) With pruned encoder.

Table 10: The evaluation of English→German "6-2tied" students pruned using group lasso on 1 CPU core.

Reg. $\lambda \rightarrow$	Base	0.1	0.15	0.2	0.3	0.4	0.5	1.0
Enc. L1	1536	563	258	137	52	30	24	12
Enc. L2	1536	454	236	156	73	47	31	15
Enc. L3	1536	421	221	135	73	47	37	19
Enc. L4	1536	799	368	197	96	52	34	16
Enc. L5	1536	999	565	350	189	114	83	32
Enc. L6	1536	1227	751	472	258	166	115	40
Dec. L1	1536	418	167	77	15	5	2	1
Dec. L2	1536	491	229	117	38	18	10	1
Dec. L3	1536	448	227	129	49	26	16	3
Dec. L4	1536	787	443	294	143	104	68	24
Dec. L5	1536	1475	1037	684	343	214	156	33
Dec. L6	1536	1536	1536	1533	1220	753	459	138
BLEU	37.3	36.9	36.8	36.6	36.3	36.3	36.1	35.9
chrF	62.6	62.4	62.3	62.3	62.0	62.0	61.9	61.8
COMET	58.1	57.3	56.8	56.2	55.1	55.4	54.6	54.3
FFN sparsity	0%	48%	67%	77%	86%	91%	94%	98%
Size (MB)	83	59	66	55	52	50	49	48
WPS	1407	1655	1811	1891	2017	2071	2112	2204
Speed-up	1.00	1.18	1.29	1.34	1.43	1.47	1.50	1.57

Table 11: The evaluation of Spanish→English 6–6 students pruned with group lasso on 1 CPU core.

Reg. $\lambda \longrightarrow$	Base	0.2	0.3	0.4	0.5	0.7
Enc. L1	1536	728	414	250	183	108
Enc. L2	1536	927	619	436	325	202
Enc. L3	1536	540	338	222	173	105
Enc. L4	1536	415	250	166	123	77
Enc. L5	1536	429	255	167	116	66
Enc. L6	1536	382	191	123	89	60
Enc. L7	1536	334	138	81	50	30
Enc. L8	1536	297	129	69	50	19
Enc. L9	1536	321	135	69	44	28
Enc. L10	1536	319	174	117	88	48
Enc. L11	1536	474	298	214	165	112
Enc. L12	1536	635	376	264	184	114
Dec. L1	1536	1536	1536	1536	1536	1536
Self att. L1	8	5	5	4	4	4
Self att. L2	8	3	2	2	2	2
Self att. L3	8	4	4	4	4	3
Self att. L4	8	4	3	3	3	3
Self att. L5	8	5	4	4	5	5
Self att. L6	8	4	4	4	3	3
Self att. L7	8	4	4	4	4	4
Self att. L8	8	3	1	1	1	1
Self att. L9	8	3	1	1	3	3
Self att. L10	8	3	2	1	1	1
Self att. L11	8	4	4	4	3	2
Self att. L12	8	4	4	4	4	4
Context att. L1	8	8	8	8	8	8
BLEU	38.2	37.9	37.3	37.0	37.0	36.6
chrF	63.9	63.6	63.2	62.9	62.9	62.5
COMET	49.5	49.6	48.2	46.5	45.5	44.6
Att. sparsity	0%	48%	56%	58%	57%	59%
FFN sparsity	0%	63%	76%	81%	84%	87%
Size (MB)	85	54	48	45	44	43
WPS	1930	2918	3029	3430	3446	3485
Speed-up	1	1.51	1.57	1.78	1.79	1.81

Table 12: The evaluation of English→German 12–1 students pruned with group lasso on 1 CPU core.