Structural Pruning for Speed in Neural Machine Translation

Maximiliana Behnke

Doctor of Philosophy

THE UNIVERSITY OF EDINBURGH

2022

To My Teenage Self and

the Future Yet to Come

Abstract

Neural machine translation (NMT) strongly outperforms previous statistical techniques. With the emergence of a transformer architecture, we consistently train and deploy deeper and larger models, often with billions of parameters, as an ongoing effort to achieve even better quality. On the other hand, there is also a constant pursuit for optimisation opportunities to reduce inference runtime.

Parameter pruning is one of the staple optimisation techniques. Even though coefficient-wise sparsity is the most popular for compression purposes, it is not easy to make a model run faster. Sparse matrix multiplication routines require custom approaches, usually depending on low-level hardware implementations for the most efficiency. In my thesis, I focus on structural pruning in the field of NMT, which results in smaller but still dense architectures that do not need any further modifications to work efficiently.

My research focuses on two main directions. The first one explores Lottery Ticket Hypothesis (LTH), a well-known pruning algorithm, but this time in a structural setup with a custom pruning criterion. It involves partial training and pruning steps performed in a loop. Experiments with LTH produced substantial speed-up when applied to prune heads in the attention mechanism of a transformer. While this method has proven successful, it carries the burden of prolonged training cost that makes an already expensive training routine even more so.

From that point, I exclusively concentrate on research incorporating pruning into training via regularisation. I experiment with a standard group lasso, which zeroes-out parameters together in a structural pre-defined way. By targeting feedforward and attention layers in a transformer, group lasso significantly improves inference speed with already optimised state-of-the-art fast models. Improving upon that work, I designed a novel approach called aided regularisation, where every layer penalty is scaled based on statistics gathered as training progresses. Both gradient- and parameter-based approaches aim to decrease the depth of a model, further optimising speed while maintaining the translation quality of an unpruned baseline.

The goal of this dissertation is to advance the state-of-the-art efficient NMT with simple but tangible structural sparsity methods. The majority of all experiments in the thesis involve highly-optimised models as baselines to show that this work pushes the Pareto frontier of quality vs speed trade-off forward. For example, it is possible to prune a model to be 50% faster with no change in translation quality.

Lay Summary

Machine translation has become widely popular in recent years, it is now easily accessible on mobile phones or through web browsers. A system like that consists of millions or billions of parameters that perform numerous mathematical calculations to provide high-quality automatic translations. The enormous scale of such models is highly expensive to run on hardware. In my thesis, I focus on reducing this computational workload by a smaller number of performed calculations. Pruning individual parameters in a network of those calculations is inefficient and requires complicated memory solutions to make a model work faster. However, it is possible to structurally remove many parameters to make the architecture smaller and faster without the need for complex memory algorithms.

The crucial point of this research is to perform the pruning in a way that damages quality the least at a given translation speed gain. I carefully examine structural pruning approaches to achieve this goal. Most experiments in this thesis involve highly-optimised models as baselines to show that my work advances the current state-of-the-art efficient machine translation through simple techniques that result in actual speed-up. For example, it is possible to prune an already optimised model to make it 50% faster with no change to translation quality.

Acknowledgements

This work would not have been possible without the unconditional support of my family, friends and professional peers.

I want to thank my mum and grandma for being my top cheerleaders and my sister and stepfather for never-ending jokes and encouragement. Many thanks to my dearest friends Olga, Miho, Darca, Gosia, Kas, Kasia and Amiko for being here for me. I would like to honour my late grandfather who passed away after a long fight with cancer just before me starting my programme. I would not be here without his support until his last days. I would also like to commemorate my cat Molko who passed away due to cancer during my last year of PhD.

I am extremely grateful to my supervisor, Kenneth Heafield, for the constant support, on-point comments, and many puns. I extend my thanks to Roman Grundkiewicz, my second supervisor, for numerous impromptu meetings whenever I got scientifically stuck.

I could not have undertaken this journey without Marcin Junczys-Dowmunt, who saw my potential and helped me grow into the researcher I am today.

I would like to extend my gratitude to my research group full of wonderful people who motivated me along the way. Special thanks to Nick, who helped me fight any doubts and supported me with his PhD-crisis knowledge and 'how-to' tips.

Lastly, I would like to recognise Darren O'Reilly, my university councillor, whose weekly assistance has been invaluable throughout these years and allowed me to finish my studies in good mental health.

Thank you all so much!

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 was performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service (www.csd3.cam.ac.uk), provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/P020259/1), and DiRAC funding from the Science and Technology Facilities Council (www.dirac.ac.uk).

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Maximiliana Behnke

Contents

Figures and Tables

Figures

Tables

Chapter 1

Introduction

The never-ending expansion of deep neural networks demands efficiency and better optimisation methods. Neural models have become so large that it is often difficult to train them on available hardware, let alone deploy them. On the other hand, larger models keep outperforming previous cutting-edge architectures. The recent research on large language models focuses on drastically increasing the number of parameters, with the popular GPT-3 (T. Brown et al., 2020) having 175 billion parameters and the recent MT-NLG (Smith et al., 2022) amounting to a total of 530 billion parameters. Even if we train such an enormous model, it is almost impossible to run efficiently, especially in mobile deployment or web browsers.

It is no different for neural machine translation (NMT), one of the most challenging tasks in deep learning since it requires an enormous amount of data and parameters to achieve good quality. In turn, a single model takes weeks to train on it. Due to the generative nature of translation, the inference speed quickly becomes an issue, given that we want the best quality possible. While on a much smaller but still impressive scale (up to 8 billion), Huang et al. (2019) have shown that translation quality improves logarithmically with the increased number of parameters as presented in Fig. 1.1.

Even with a few billion parameters, the training of such a model poses a significant challenge. Not only training of such models is not feasible without appropriate hardware, the increased depth and complexity may lead to overfitting Any model should generalise well on rarely or even never seen data. It is especially crucial in NMT, in which the input space is infinite and highly varied.

Hoffmann et al. (2022) found that “for compute-optimal training, the model size and the number of training tokens should be scaled equally.” This is not a case in most circumstances as we struggle with obtaining more good quality data for many language pairs. In the long run, just utilising more parameters and bloating models up is not the answer if we do not feed more data into a system. It may suggest that we are over-utilising deep neural networks to achieve our goals. At the end of a day, those neural models are deployed for users on either GPU or CPU.

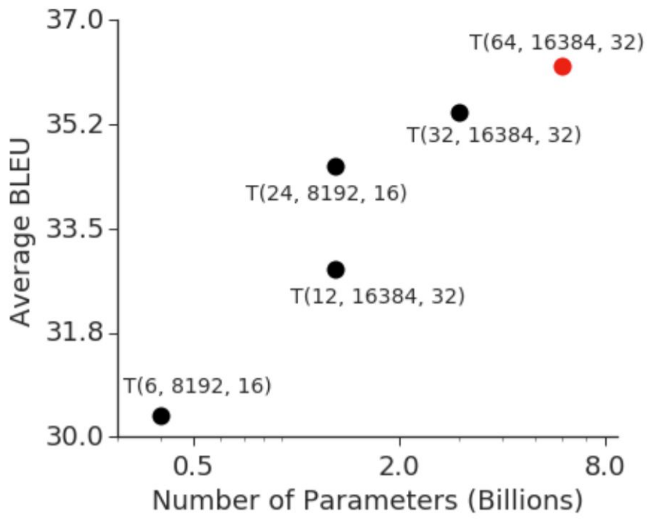


Figure 1.1: A strong correlation between translation quality and the number of parameters in a transformer architecture (Huang et al., 2019).

As we will see in Tab. 3.11 and 3.12, switching from 1 GPU to 1 CPU core slows translations down fourfold. There are many ongoing efforts to make neural networks faster, specifically on the CPU due to this performance gap. As will be presented in Sect. 4.15, most of those optimiation methods can be stacked together to boost efficiency even more.

The blatant overparametrisation of neural networks combined with expensive maintainance costs leads to a natural question: if this large number of parameters is required to achieve good quality, can most of them be removed from a model down the line? Through *parameter pruning*, a model can be optimised towards speed or memory size to make it faster or smaller. There is a variety of work on pruning individual parameters (Brix, Bahar, & Ney, 2020; See, Luong, & Manning, 2016), larger sub-network structures (Behnke & Heafield, 2020; Molchanov, Tyree, Karras, Aila, & Kautz, 2016; Voita et al., 2019), and even whole layers Sajjad, Dalvi, Durrani, and Nakov (2020). Coefficient-wise pruning remains one of the most popular approaches due to its straightforwardness and good results. Most scattered and coefficient-wise sparsity patterns require specialised memory representations and matrix multiplication routines to avoid unnecessarily processing zeroed-out parameters.

Unfortunately, much of the prior work on pruning does not report speed or makes inference slower. For example, Brix et al. (2020) achieved no speed-up despite removing 70% of all parameters in a model in exchange for -0.6 BLEU loss. The available sparse kernels are often difficult to optimise. For example, Yao, Cao, Xiao, Zhang, and Nie (2019) reported that their model with 87.5% parameters pruned is 1.6× as slow when using cuSPARSE kernels in comparison to the basic dense operator. Gale, Zaharia, Young, and Elsen (2020) further support this by pointing out that coefficient-sparse kernels like cuSPARSE are highly non-optimal for less than 95% sparse matrices.

Blockwise sparsity, while more memory friendly due to more even spread out of parameters, still struggles with efficiency. Gray, Radford, and Kingma (2017) reported block-sparse kernels being 1.8× slower at 70% sparsity. After developing a custom “balanced pruning” method, they eventually achieved 1.4× faster inference.

Due to these optimisation issues, many papers represent their speed gains in FLOPs only. While sparse operations have fewer FLOPs, they have overhead to encode sparsity and in less regular memory access. Gale, Elsen, and Hooker (2019) point out that FLOP is often inconsistently defined across papers making it impossible to compare and gauge actual advancement. This calls into question the utility of FLOPs as a unit of work depsite widespread use in pruning literature. In the end, the struggle to optimise sparse models for speed shifts the focus of researchers towards compression only.

Another critical issue in the research field of pruning is the lack of transparency in the results regarding the potential improvements of new methods. Too much work (Gu, Bradbury, Xiong, Li, & Socher, 2018; J. Lee, Shu, & Cho, 2020; Y. Wang, Wang, Li, & Tu, 2020) on efficiency compares completely unoptimised systems with their optimised solutions, which are smaller or faster in exchange for quality performance. Despite the popularity of pruning, engineers in practice prefer other more straightforward methods such as using fewer layers in a model, reducing the depth of a decoder or training smaller architectures under a knowledge-distillation regime Y. Kim and Rush (2016a) to apply quantisation at the end of it then. Most efficiency papers fail to prove that their new method offers a better trade-off than those mentioned above (and more) existing methods.

Still, it is possible to achieve faster and smaller models with no need for sparsity operators. Image recognition is one of the most prevalent deep learning tasks, which utilise convolutional networks (LeCun et al., 1989). Layers in such a model contain a set of convolutional filters (kernels) which are applied onto 2D image to transform it. Removing kernels from a model prunes parameters structurally in a cascading way, resulting in a still dense but smaller architecture. The insight into the pruning subject shows the severe lack of efficient sparsity solutions outside of convolutional networks. It served as an inspiration to explore and contribute to the development of similar straightforward structural pruning approaches outside of image recognition, in this case for machine translation. As my research will show, the state-of-the-art transformer model (Vaswani et al., 2017a) can be sliced and pruned to produce a smaller but still dense architecture — this way, the need for sparse kernels is sidestepped. At the same time, the number of calculations gets reduced.

1.1 Thesis statement

In my thesis, I make inference speed of NMT models faster by exploring structural pruning methods and improving upon them. I provide pruning regimes that result in tangibly faster translation through simple training steps that prune and collapse a model into a smaller but still dense architecture that requires time to perform calculations. In this thesis, I perform most experiments on highly-optimised fast and robust architectures that represent the state-of-the-art in the optimisation field. Pruning results on such models show a practical and cut-edge advancement in the optimisation of neural networks. The pruning methods developed in this thesis result in *Pareto optimal* models that are the fastest at given translation quality and vice versa.

1.2 Thesis contributions

This work consists of three main content chapters. Each one focuses on a different structural pruning approach and includes extensive experimental sections and analysis of their results. At the end of every chapter, Pareto analysis is performed, comparing various baselines and the specific pruning method and debating which strategy provides the best quality-speed trade-off. This sets up a new Pareto frontier to build upon in the following chapter.

1.2.1 Structural Lottery Ticket Hypothesis

Among the more successful and well-known pruning methods recently is the *Lottery Ticket Hypothesis* (Frankle & Carbin, 2019). It assumes that a model can be stripped of individual obsolete parameters as long as it gets retrained without them, while the other parameters get initialised to the same values. The best way to find a winning combination of parameters (*ticket*) has been under constant refinement (Brix et al., 2020; Frankle, Dziugaite, Roy, & Carbin, 2019; Morcos, Yu, Paganini, & Tian, 2019; Yu, Edunov, Tian, & Morcos, 2020), but the core idea was always to prune individual parameters.

> I expand upon the Lottery Ticket Hypothesis to prove it holds in a structural setup, not just on a coefficient level and that it makes neural architectures fast and compact.

I apply this well-established pruning technique to remove attention heads from a transformer architecture. My research shows that this algorithm, combined with a custom pruning thresholding function, can be used to remove whole structures/blocks of parameters instead of doing so on a coefficient level.

Inspired by the attention confidence defined by Voita et al. (2019), I prune the least confident attention heads to translate faster. The experiments demonstrate that most attention heads can be removed without significant damage to the quality. Pruning about half of the attention heads does not affect the quality, while removing three-quarters of them damages a model by 0.1

to 0.2 BLEU. The inference speed is also substantially faster. For example, pruning attention heads in a base transformer architecture leads to 1.6–$2.2\times$ faster translation at 0.1–0.3 BLEU loss in quality. Training the identical architectures from scratch underperforms in the quality area, proving that the hypothesis holds in the structural approach.

The work is then further expanded beyond just attention onto block-sparse feedforward layers as well.

1.2.2 Structural Regularisation

Regularisation techniques prevent a model from overfitting by reducing its complexity. One of the possible side-effects is parameter pruning. *Group lasso* (Yuan & Lin, 2006) is a structural regulariser that penalises a cost function, pruning groups of parameters in the process.

> I create a customised pipeline that incorporates structural regularisation early into training and apply it to prune a transformer architecture for faster inference.

Pruning and training progress together with a model converging in a similar time to a baseline. This reduces the overall deployment time by eliminating complete or partial retrainings required in most pruning techniques.

My approach produces Pareto optimal models in terms of quality and speed. Group lasso applied over nodes in feedforward layers boosts inference speed significantly. Similarly, pruning entire heads on top results in even faster translation. For example, removing half of the attention heads and two-thirds of feedforward connections makes a ‘12–1” encoder-decoder model $1.5\times$ faster with no change to quality in COMET. Similarly to the previous research direction, there is no need for elaborative sparsity representations or matrix multiplications. Combined with quantisation, the speed-up is even more substantial.

1.2.3 Aided Regularisation

Standard regularisation penalises all layers equally across a model. It can have an adverse impact on quality as some parts of the architecture may be a bottleneck, with particular layers more crucial to the performance than others.

> To improve upon the previous work, I develop and explore a new regularisation technique that scales penalties for each layer, instead of having one global constant scalar as is the case in a typical regulariser.

Aided regularisation, as the name suggests, independently scales penalties per layer, aided by external information. I define two scaling variants that keep track of the magnitude of either gradients or parameters and change penalties accordingly. A close analysis of this regularisation method clearly shows that it greatly boosts inference speed and pushes highly-optimised state-of-the-art forward. For instance, with just pruning, a model can get $1.8\times$ faster at the cost of half a BLEU point. The aided regulariser amplifies sparsity patterns exhibited by the pruning methods from my previous works on the Lottery Ticket Hypothesis and the group lasso. When regularised with this technique, a model concentrates on reducing its architectural depth by pruning and collapsing its middle layers. Experiments on regularising and then rejuvenating parameters showed that reducing a 12-layered encoder to 7 layers is possible with no loss in quality.

1.3 Thesis structure

The road-map of this thesis presents as follows:

Chapter 1: Introduction You are here.

Chapter 2: Background This chapter introduces all topics and essential knowledge required to understand and follow the whole thesis.

Chapter 3: Structural Pruning of Transformer with Lottery Ticket Hypothesis The chapter explores the Lottery Ticket Hypothesis, this time in a structural setup.

Chapter 4: Structural Pruning for Speed Using Group Lasso This chapter switches focus towards structural regularisation using a group lasso incorporated early into training.

Chapter 5: Improving Structural Pruning through Aided Regularisation This chapter explores a novel regularisation approach that scales penalties for each layer as training progresses.

Chapter 6: Conclusions This chapter summarises the entire thesis.

Chapter 2

Background

Machine translation is a sequence to sequence task that generates a translation based on its likelihood given a source sentence. Nowadays, we employ neural networks to directly optimise the problem defined as:

$$\underset{e}{\operatorname{argmax}} P(e \mid f) = \prod_{i=1}^{|e|} P(e_i \mid e_0, ..., e_{i-1}, f_1, ..., f_{|f|})$$

with a foreign sentence *f* and a hypothetic translation *e*. The goal is to find the best possible translation *e*. The quality of machine translation is evaluated using human judgement or automatic scoring such as BLEU (Papineni, Roukos, Ward, & Zhu, 2002).

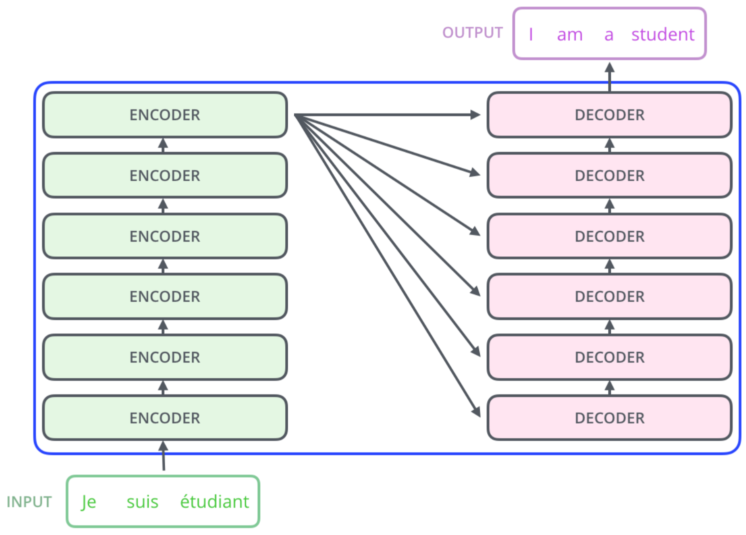


Figure 2.1: A typical encoder–decoder architecture (illustrated by Alammar (2018)).

In neural networks, *embeddings* represent words in a vectorised form that map discrete word entities into continuous space. In sequence to sequence tasks, the most popular architecture approach is *encoder–decoder*. As seen in Fig. 2.1, both encoder and decoder consist of a stack of identical layers. An encoder is responsible for digesting an input and creating its hidden representation. An encoder can serve as a language model to determine the probability of a given sequence of words. Then, given an encoded source, a decoder generates a hypothesis.

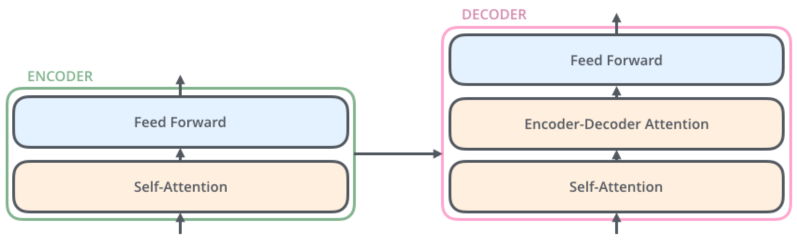


Figure 2.2: A single layer unit in a transformer (illustrated by Alammar (2018)).

The current state-of-the-art architecture used in neural machine translation (NMT) is a transformer (Vaswani et al., 2017a). It is an encoder-decoder architecture as illustrated in Fig. 2.2. It consists of three key components:

Embeddings Numerical vector representations of words in vocabularies. They are the largest matrices in a model with the size of (D_{vocab}, D_{model}) with D_{vocab} usually being $32\,000$. Press and Wolf (2017) has shown that using the same embeddings for both source and target languages, especially if they use similar alphabets, reduces a model size without compromising on quality. Thus, shared embeddings are a standard approach.

Feedforward (FFN) layers A layer of linear transformation followed by a non-linear function applied on top of it.

Attention mechanism A layer that learns alignments. Attention layer has three trainable parameters: keys ($\mathbf{W}^K$), queries ($\mathbf{W}^Q$) and values ($\mathbf{W}^V$). Keys and queries process two inputs and create the actual alignment distribution between them. This distribution, in turn, scales the generated values. If there is only one input and alignment is done over words in the same sentence, it is called *self-attention*. If done between source and target sentences, it is called *context* or *encoder–decoder* attention. Both encoder and decoder perform self-attention analysis of their respective sentences: input and translation. The bridge between the encoder and decoder is context attention that directly creates an alignment between a source and its target translation.

Given parameters $\mathbf{W}^K$, $\mathbf{W}^Q$ and $\mathbf{W}^V$ with an assigned input $\mathbf{X}$, the keys, queries and values are generated as follows:

$$\begin{aligned}\mathbf{Q} &= \mathbf{X}^Q\mathbf{W}^Q \\ \mathbf{K} &= \mathbf{X}^K\mathbf{W}^K \\ \mathbf{V} &= \mathbf{X}^V\mathbf{W}^V\end{aligned} \tag{2.1}$$

The attention mechanism is then defined as:

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \sigma\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}}\right)\mathbf{V} \tag{2.2}$$

with σ being a softmax function that creates a probability distribution:

$$\sigma(\mathbf{X})_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{X_j}} \quad \text{for } i = 1,\dots,N \text{ and } \mathbf{X} = (X_1,\dots,X_N) \in \mathbb{R}^N \tag{2.3}$$

One instance of calculations in Eq. 2.2 is refered as an attention *head*. *Multi-head* attention runs multiple alignments at the same time to then concatenate and scale the results into a final output:

$$\begin{aligned}\text{MultiHead}(\mathbf{Q},\mathbf{K},\mathbf{V}) &= [\text{H}_1 \dots \text{H}_h]\,\mathbf{W}^O \\ \text{where } \text{H}_i &= \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V\right) \\ \text{and } h &= \text{ is the number of attention heads.}\end{aligned} \tag{2.4}$$

The example of attention alignment is presented in Fig. 2.3. Many attention heads learn to produce interpretable alignment distributions such as syntactic relations, pronoun recognition, et cetera (Serrano & Smith, 2019; Voita et al., 2019).

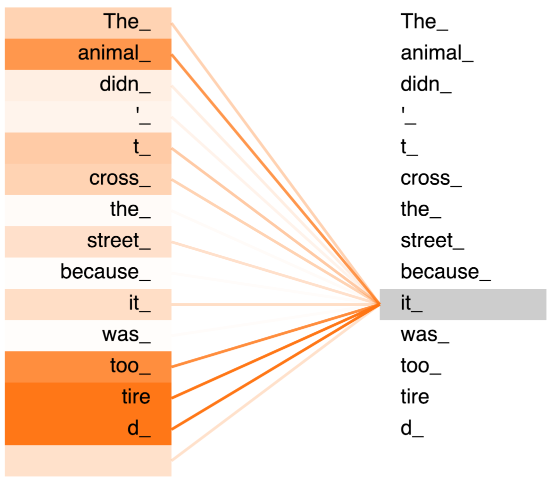


Figure 2.3: A visualisation of attention alignment for a single head in a transformer (from Tensor2Tensor tutorial by Vaswani et al. (2018)).

In order to successfully train a deep model, a transformer architecture includes *residual networks*, also known as *skip connections*. They are an additional shortcut with layer input added to its output. In Fig. 2.4, skip connections are represented as dashed lines with the example input $\mathbf{X}$ added to the output $\mathbf{Z}$ to be then normalised.

Residual connections may help avoid a vanishing gradient problem (Informatik, Bengio, Frasconi, & Schmidhuber, 2003) where the first layers struggle to learn due to diminishing returns of backpropagation. Skip connections are also crucial when pruning parameters from a model. Without skip connections, if a layer gets pruned too much, it would zero out the output completely and break a model from producing any meaningful predictions. Residual paths ensure that unchanged input gets propagated forward, even if a layer is sparsified or removed.

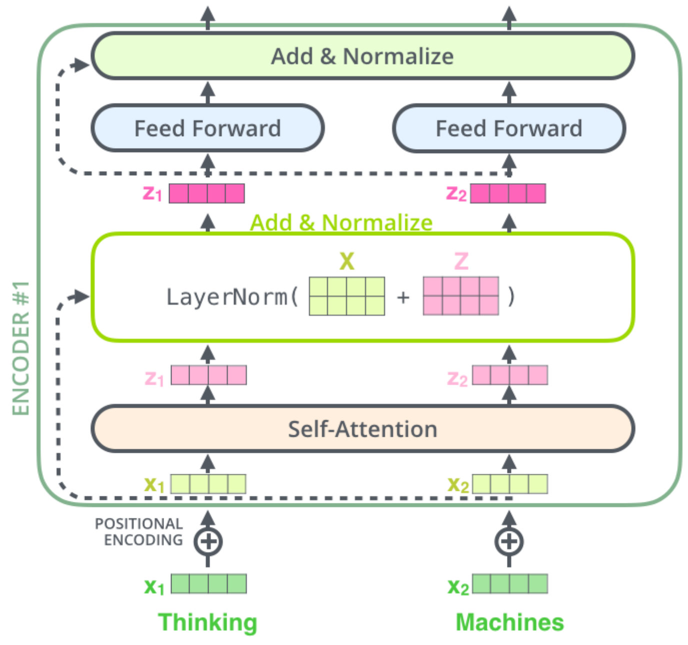


Figure 2.4: A skip/residual connections passed forward transformer layers (illustrated by Alammar (2018)).

A transformer architecture has many advantages. It outperforms the previous state-of-the-art approaches such as LSTM (long-short term memory) recurrent neural networks (RNN) (Vaswani et al., 2017a). Since each layer of the encoder depends only on the layer below it, the input sequence can be processed in parallel. Thus, batched input sequence is fed into the network in one go, allowing a model to see complete sentences. A parallel dataset (corpus) provides a reference translation during training that allows a model to compare it with its output. At inference, though, the model only has source information. For that reason, translation is generated word by word from a probability distribution and keeping track of the top-k translation hypotheses through the combined scores of the histories and their continuations when new words get predicted. Those selected words are then fed back into a loop to generate the next words. This search process is called *beam search*. It is time-consuming and computationally as a decoder is running sequentially, considering multiple hypotheses along the way; meanwhile, this is not a concern in an encoder as it gets executed once.

On top of this, the attention mechanism is $O(n^2)$, with n being the length of an analysed sequence length. The aforementioned reasons contribute to the overall large computational costs of translating using the transformer architecture. There is an ongoing effort to reduce the complexity of the attention mechanism. For example, Sparse Transformer (Child, Gray, Radford, & Sutskever, 2019) reduced attention complexity from $O(n^2)$ to $O(n\sqrt{n})$, mostly for image recognition purposes. In sequence-to-sequence tasks, Simpler Simple Recurrent Unit (SSRU) (see Sec. 2.2.2) reduces self-attention in the decoder from $O(n^2)$ to $O(n)$.

In the following sections, I will discuss the most popular approaches used to build state-of-the-art efficient machine translation models.

2.1 Knowledge distillation

Quality and efficiency are (almost) always conflicting: good quality requires complexity, while efficiency requires quality sacrifices. In the long run, we want much smaller, less complicated architectures. However, those trained from scratch will never reach the learning potential and quality of those large state-of-the-art models mentioned above. The best quality-wise models presented in the Workshop on Machine Translation (WMT) usually include many deep transformer models ensembles. Each translates at a large beam size, generating 6–12 (or more) hypotheses per output word. Such models are costly to run and, in practice, are only reasonable for a shared task or offline usage.

Knowledge distillation is a compression method that effectively allows training of smaller robust architectures with much higher quality using knowledge acquired (distilled) from deeper and better models.

The strategy of transferring knowledge from numerous ensembles into a single model was introduced by Buciluǎ, Caruana, and Niculescu-Mizil (2006). The idea of knowledge distillation has been further solidified and generalised by Hinton, Vinyals, and Dean (2015). Usually, an objective function optimises toward given data with yet-not-seen input in mind. A well-versed generalisation on new data characterises a good model. It is vital in natural language processing (NLP) considering language flexibility, as smaller models struggle with generalisation due to low architecture complexity.

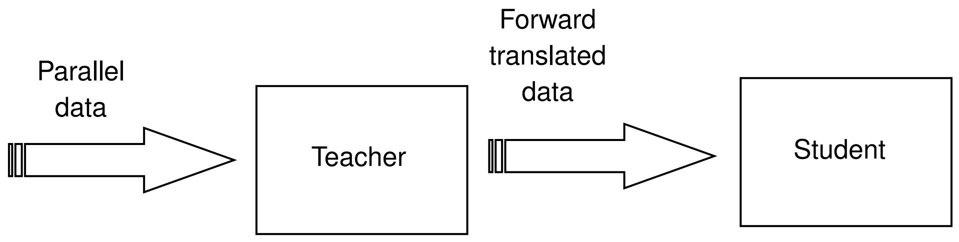


Figure 2.5: A visualisation of a teacher–student training regime for machine translation. The student is trained on text forward-translated by the teacher with the goal of overfitting towards its distributions.

In knowledge distillation, instead of aiming for *golden standard* set by a reference, a smaller model is taught to approximate the distribution of another model instead. The most popular form of knowledge distillation in machine translation is a *teacher-student regime* introduced by Y. Kim and Rush (2016a). In it, a smaller *student* model trains on data generated by a larger *teacher* model with a wide beam search space and picking its best translations. A student's goal is to overfit the teacher's distribution as much as possible. The student cannot reach the quality akin to the teacher's on its own, but learning to mimic another model's distribution through distillation allows it to learn better than when trained on parallel data directly.

	Teacher	Parallel data	Distilled data
Size	2.3GB	85MB	85MB
BLEU	45.1	31.3	37.6
WPS	559	5828	5958

Table 2.1: A comparison of small models trained using normal and knowledge-distilled parallel data alongside the large teacher model from the WMT2021 Efficiency Shared Task. Speed and quality were calculated on a single GeForce 1080 GPU averaged for WMT16–19 testsets.

In Tab. 2.1, I present an English$\rightarrow$German teacher and student, further explored in Sec. 4.16. On a GPU, the best-quality teacher translates $10.7\times$ fewer words per second (WPS) than its optimised knowledge-distilled counterpart. Most importantly, I also compare a tiny transformer trained on a parallel data directly and through forward-translated knowledge-distillation. Despite both models being trained on the same amount of data (66.5M sentences), the knowledge-distilled student is 6.3 BLEU points better than a model trained on pure parallel data. This shows the clear advantage of knowledge-distillation for better quality and smaller architectures than models trained in usual fashion. When trained on much more data, the gap between a teacher and student closes even further. In Tab. 2.2, I present an example of an English$\rightarrow$German WMT19 teacher consisting of four ensembled models with an enormous total amount of 1.5 billion parameters. The distillation results in a model with 100$\times$ fewer parameters at the cost of only 1 BLEU point in quality. This kind of trade-off is highly efficient and often pursued for model deployment. Both of these models are further explored in Sec. 3.4.5 and 3.4.16.

Model	# parameters	BLEU
Teacher	1,542,059,008	42.5
Student	15,722,752	41.5

Table 2.2: A comparison between a state-of-the-art WMT19 English→German teacher model and its distilled more robust student counterpart evaluated on WMT19 testset.

One of the best side-effects of sequence-level distillation in NMT is that a student has a peaked probability distribution and therefore beam search does not yield much improvement. Greedy search of chosing the top-1 word is near sufficient, which means a distilled model does not require beam search (Y. Kim & Rush, 2016a). Since a student overfits a teacher's distribution, it is possible to take the most probable word every time a student generates a translation. As shown before in Tab. 2.1, it speeds up inference by an order of magnitude. It also reduces the overall size of a neural architecture as it usually compresses an architecture consisting of multiple ensembled models into a single network. For those reasons, knowledge distillation became one of the first optimisation steps in machine translation.

2.2 Architecture optimisation

It is not an easy task to develop an entirely new and better architecture. The advancement is usually made by slowly improving upon specific parts of the current networks, often with explicit efficiency in mind. A current state-of-the-art approach to a task is a mix of such techniques until more progress is made in the future. In this section, I describe a few methods widely used in NMT and have been proven reliable and positively impact efficiency.

2.2.1 Tied embeddings

Embeddings are the largest matrices in the neural architecture. Typically, each language from source and target has its vocabulary, which almost doubles a model's size. However, many language pairs either share an alphabet or jargon. Even if that is not the case, source and target sentences may use the same entity names, numbers, date representation, et cetera. Due to the nature of backpropagation, most word embeddings do not update as frequently as we would like them to be, leading to suboptimal gradient learning. Inan, Khosravi, and Socher (2016); Press and Wolf (2017) have shown that sharing embeddings between input and output layers reduces the perplexity of language models and compresses them to less than half of their original size. Thus, tied embeddings and a single shared vocabulary have become a staple in NMT.

2.2.2 Simplifying decoder attention

The attention mechanism is quite an expensive operation with a complexity of $O(n^2)$, where n is the length of a sequence it attends to. Moreover, decoding in a transformer is not as parallelisable due to the auto-regressive nature of translation. While context attention seems to be a transformer's crucial and indispensable component, there is an ongoing effort to simplify decoder self-attention and make it more efficient.

Average Attention Network (AAN), introduced by Zhang, Xiong, and Su (2018), replaces calculating dynamic weights with their simple fixed average. The AAN network consists of an average layer (Eq. 2.5), a gating layer (Eq. 2.6) and a normalised output layer (Eq. 2.7).

Given an input $y = \{y_1, y_2, ..., y_n\}$, the average layer is defined as:

$$\mathbf{g}_j = \mathrm{FFN}\left(\frac{1}{j}\sum_{k=1}^{j}\mathbf{y}_k\right) \tag{2.5}$$

with FFN being a feedforward layer operations.

The average g_j and input y_j are then fed into a gating network:

$$\begin{aligned} \mathbf{i}_j, \mathbf{f}_j &= \sigma\left(\mathbf{W}\left[\mathbf{y}_j; \mathbf{g}_j\right]\right) \\ \tilde{\mathbf{h}}_j &= \mathbf{i}_j \odot \mathbf{y}_j + \mathbf{f}_j \odot \mathbf{g}_j \end{aligned} \tag{2.6}$$

where $\mathbf{i}_j, \mathbf{f}_j$ are input and forget gates respectively. $[\cdot;\cdot]$ is a concatenation, $\odot$ denotes an element-wise multiplication and σ is a *sigmoid* activation function, often replaced with *tanh*.

Finally, the gating output is then added with the input again through the skip connection and normalised:

$$\mathbf{h}_j = \text{ LayerNorm } \left(\mathbf{y}_j + \tilde{\mathbf{h}}_j\right) \tag{2.7}$$

In contrast to the standard self-attention defined in Eq. 2.2 and 2.4, the average network can be updated incrementally through dynamic programming, with the average in Eq. 2.5 simply updated. This means the attention does not have to be completely recalculated from scratch. Using a constant mask matrix, the calculations of cumulative averages over inputs can be easily parallelised. This way, the complexity gets reduced from $O(n^2)$ to linear.

According to Hsu, Garg, Liao, and Chatsviorkin (2020), removing the gating layer does not affect quality. Junczys-Dowmunt, Heafield, Hoang, Grundkiewicz, and Aue (2018) have gone even further and removed feedforward calculations as well to no quality loss. Further research in that direction resulted in *Simpler Simple Recurrent Unit* (SSRU) (Y. J. Kim et al., 2019) which does not suffer from performance degradation similarly to self-attention, AAN or original SRU cell (Lei, Zhang, Wang, Dai, & Artzi, 2017). At the same time, SSRU drops one more matrix multiplication in comparison to SRU and replaced *tanh* with a simpler *ReLU*.

Given input x_t and trainable parameters W_t and W, SSRU cell is defined as follows:

$$\begin{aligned} f_t &= \sigma\left(W_t x_t + b_f\right) \\ c_t &= f_t \odot c_{t-1} + (1 - f_t) \odot W x_t \\ o_t &= \mathrm{ReLU}\left(c_t\right) \end{aligned} \tag{2.8}$$

where $\odot$ denotes an element-wise multiplication. f_t is a forget gate and c_t is a cell state.

SSRU is the current state-of-the-art approach for replacing a decoder self-attention with linear approximation (Bogoychev et al., 2020; Hsu et al., 2020; Y. J. Kim et al., 2019).

As mentioned at the beginning, all these methods only replace the self-attention in the decoder, not the context attention. The same with the encoder self-attention, which is left unchanged. While the encoder is only executed once per batch, there is still a room for improvement in its efficiency.

2.2.3 Lexical shortlisting

The last matrix multiplication in NMT models is the most expensive since it requires producing probabilities over the whole vocabulary. As a model generates translation word by word, it is highly wasteful to do so over the *entire* vocabulary. To speed inference up and lower memory consumption, we can create a vocabulary shortlist (Y. J. Kim et al., 2019) of the most probable translation candidates within a batch and predict only using those. Given parallel data, we can extract the topmost probable translations based on alignments between source and target sentences. During the translation, this reduces output embedding sizes from, for example, $32,000$ vocabulary words to a union of 50 most probable translations per batch word and 50 most frequent words in the vocabulary. It means that both batch size and sentence lengths affect the final matrix dimension. For example, the shortlisted embeddings of the model from Tab. 2.3 fluctuate between 13560 and 1272 when translating with top 50 most probable and common words per batch of 32 sentences.

Shortlisting is especially effective on the CPU as it significantly reduces matrix multiplication workload, fitting more into cache. As seen in Tab. 2.3, a model with shortlisting translates $1.66\times$ faster at no change in quality in comparison to the baseline.

	Baseline	+ shortlist
BLEU	38.2	38.2
WPS	1275	2111
Speed-up	1.00	1.66

Table 2.3: A speed evaluation of a tiny transformer English→German model (from Tab. 5.11) evaluated with and without shortlisting on WMT16–19 testsets.

2.3 Quantisation

Matrix multiplications in deep learning are resource-demanding. Those calculations not only are slow, but they also consume a lot of memory, bandwidth and disk space. Neural network parameters are typically represented in a single-precision floating-point format using 32-bits (*fp32*). Can we use fewer bits than that to represent and run a model while keeping its quality intact? Even going from single- to half-precision (fp16) (Khudia et al., 2021) allows us to compress a model by $\times 2$. However, what if we want to use fewer bits than that? As seen in Fig. 2.6, floating-point representation loses its flexibility if it uses fewer bits in either exponent or mantissa.

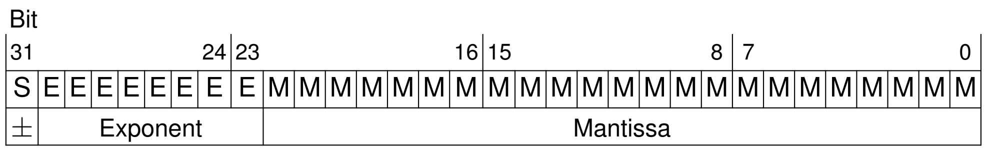

Bit
31
24
23
16
15
8
7
0
S E E E E E E E E M M M M M M M M M M M M M M M M M M M M M M M
±
Exponent
Mantissa

Figure 2.6: Single floating point representation (based on IEEE754 standard).

Matrix multiplication, the staple operation in neural networks, is a memory-bound problem. For this reason, there has been extensive research done into using low-precision fixed point representations instead of floating point numbers. Integer arithmetics fit more values into memory, in turn allowing more multiplications to be performed in parallel. Integers load faster to cache, fitting more values into registers. This makes integers especially friendly on CPU, embedded systems or mobiles.

Quantisation is a process of mapping continuous floating point numbers into discrete integer values at the cost of precision. It is required that a zero is represented without any rounding errors. Given a scale s and a zero point z, the quantisation of input x is:

$$x_q = \text{round}\left(\frac{1}{s}x + z\right) \tag{2.9}$$

and the dequantisation is:

$$x = s\,(x_q - z) \tag{2.10}$$

We also clip quantised values to avoid any out of bound cases in practice. The scale and zero variables are are defined for each tensor or layer-type individually.

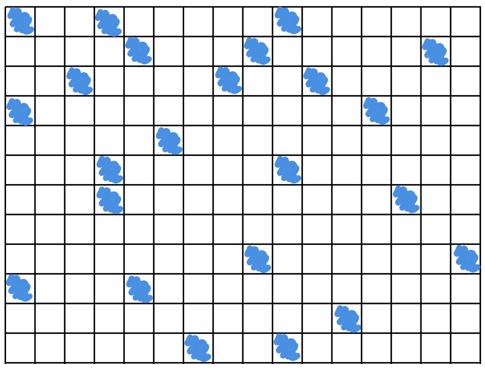
The intuition behind quantisation is that it classifies continuous values into discrete integer *bins*. Those bins do not have to spread out evenly; quite the opposite. Neural networks require more accuracy for numbers close to zero than those for larger but less frequent values; thus it would be more beneficial to have more bins cover area around zeroes. Depending on the implementation and quality tolerance, quantisation may be applied to a varying degree.

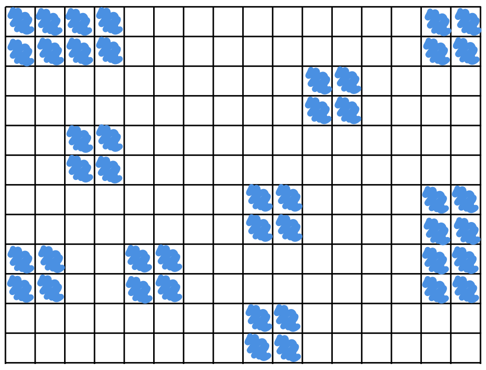
Besides parameters, activations and gradients may be quantised on a fly to operate entirely within integer arithmetic with no float–integer conversions in-between. For example, Bogoychev et al. (2020) quantises activations to values between $[-127, 127]$ by extracting minimum and maximum values from a tensor and multiplying every float with by $\frac{127}{\max}$. This can be done statically once with a preset range values or calculated dynamically during inference.

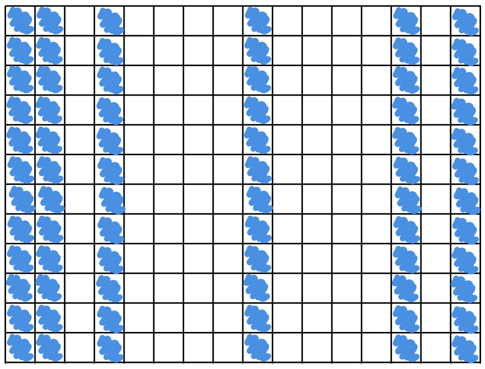
Quantised training is difficult since rounding errors accumulate during backpropagation, but not impossible (Courbariaux, Bengio, & David, 2015; Gupta, Agrawal, Gopalakrishnan, & Narayanan, 2015; H. Li et al., 2017). Most efforts concentrate on quantised inference as parameters are static, which has been quickly popularised in image recognition (Jacob et al., 2018; Miyashita, Lee, & Murmann, 2016), but has been the focus of sequence-to-sequence research as well, including machine translation (Bogoychev et al., 2020; Junczys-Dowmunt, 2018). 8-bit quantisation has been accepted as a standard as it usually does not compromise quality on the variety of tasks and is widely supported by hardware. More aggressive quantisation includes 4-bits (Aji & Heafield, 2020; Miyashita et al., 2016) or even binary networks (Courbariaux & Bengio, 2016; Rastegari, Ordonez, Redmon, & Farhadi, 2016). Replacing 32-bit floats with 8-bit integers compresses a model $\times 4$ and leads to even twice as fast inference in NMT (Bogoychev et al., 2020).

2.4 Sparsity & pruning

One of the methods to prevent overfitting and reduce resource consumption is *pruning*. Removing parameters from a neural network was partially inspired by how our brains develop. The number of neural synapses grows in orders of magnitude during our childhood to help us learn, memorise and adapt, and as we mature, the brain *prunes* synapses that it no longer needs (Chechik, Meilijson, & Ruppin, 1998). Optimal Brain Damage (OBD) (LeCun, Denker, & Solla, 1990) is one such pruning method that has been successfully applied to neural networks. In it, parameters are pruned based on the information carried by their second derivatives and the approximate effect their removal has on a model. Since then, pruning has become a staple in neural network optimisation.


(a) A coefficient-wise sparsity.


(b) A block-wise sparsity.


(c) A neuron-wise sparsity.

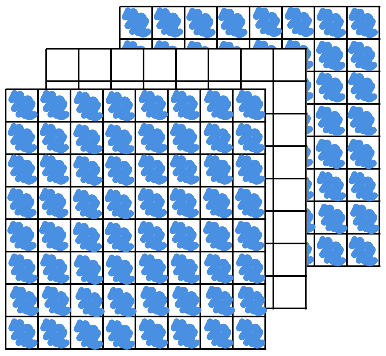

(d) A subtensor/layer-wise sparsity.

Figure 2.7: Examples of sparsity patterns.

A typical neural model consists of a collection of tensors and bias terms for each layer. Parameters may be removed in a specific fashion to achieve different goals. Pruning usually focuses only on matrices, not biases, unless in specific cases. Pruned tensors can be still dense and only masked with zeroes, with no optimisation benefits. If possible, pruned matrices should take advantage of fewer parameters in memory consumption and matrix multiplication routines. Here I introduce the most popular sparsity approaches alongside their optimised representations.

Coefficient pruning is the most popular pruning configuration due to its lack of structure: a model removes individual parameters in a scattered pattern. It is usually much easier for a model to remove individual parameters without damaging quality. Fig. 2.7a presents an example of a coefficient-wise sparse matrix. A randomly scattered pattern is not particularly memory friendly. A matrix multiplication routine loads and performs calculations on smaller matrix chunks in parallel. It is possible to represent a sparse matrix in a way that reduces memory accesses. However, there is still a significant overhead as hardware arranges values to be multiplied efficiently. If we multiply block of 4 floats in one register with 4 floats in another, sparse parameters need to be organised accordingly. Ideally, this block could skip this operation entirely, reducing the workload. *Block-wise sparsity* is one such case. An example of a matrix with 2×2 block sparsity pattern is presented in Fig. 2.7b. Both coefficient- and block-sparse matrices require specialised memory representation and multiplication routines to avoid unnecessarily processing zeroed-out parameters.

On the other hand, it is possible to remove more coarse structures such as layer connections (Fig. 2.7c), which are usually rows and columns. A popular approach in convolutional networks is to remove entire kernels, which allows a network to skip calculations down the line (Fig. 2.7d). Similarly, attention heads in a transformer can be pruned. This type of sparsity is advantageous as tensors can be *sliced*, redundant parameters wholly removed, with a final architecture being smaller but still dense. There is no requirement for specialised algebra or algorithms, with minor or no code modifications to toolkits. Smaller architectures are especially CPU-friendly as more fits into their cache, which results in a significant speed-up.

Let us take a look at examples of storage formats for sparse matrices. The sparse examples have been provided by Intel® oneAPI Math Kernel Library (*Developer Reference for Intel® oneAPI Math Kernel Library - C*, n.d.).

Given a sparse matrix M:

$$M = \begin{pmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & * \\ * & * & 4 & 6 & 4 \\ -4 & * & 2 & 7 & * \\ * & 8 & * & * & -5 \end{pmatrix}$$

Coordinate Matrix Storage Format (COO) The simplest form of storing sparse matrices. Each value with its row and column indices is stored side by side. Straightforward for human interpretation.

$$\begin{array}{lccccccccccccccc} \mathit{values} & = & 1 & -1 & -3 & -2 & 5 & 4 & 6 & 4 & -4 & 2 & 7 & 8 & -5 \\ \mathit{rows} & = & 0 & 0 & 0 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 \\ \mathit{columns} & = & 0 & 1 & 2 & 0 & 1 & 2 & 3 & 4 & 0 & 2 & 3 & 1 & 4 \end{array}$$

Compressed Sparse Row Matrix Storage Format (CSR) One of the most popular sparse matrix formats. It also has a column-based variant (CSC) with values transposed. A typical 3-array variation can be extended into 4 arrays where the pointers for row boundaries are kept separately: beginnings of rows in one array and end of rows in another. The mechanism behind the pointers is visualised in Fig. 2.8.

$$\begin{array}{lccccccccccccccc} \text{values} & = & 1 & -1 & -3 & -2 & 5 & 4 & 6 & 4 & -4 & 2 & 7 & 8 & -5 \\ \text{columns} & = & 0 & 1 & 3 & 0 & 1 & 2 & 3 & 4 & 0 & 2 & 3 & 1 & 4 \\ \text{4-array variation:} \\ \text{pointerB} & = & 0 & 3 & 5 & 8 & 11 \\ \text{pointerE} & = & 3 & 5 & 8 & 11 & 13 \\ \text{3-array variation:} \\ \text{pointer} & = & 0 & 3 & 5 & 8 & 11 & 13 \end{array}$$

$$\begin{array}{cccccccccccccc}
0 & & & 3 & & 5 & & & 8 & & & 11 & & 13 \\
[1 & -1 & -3] & [-2 & 5] & [4 & 6 & 4] & [-4 & 2 & 7] & [8 & -5] & \\
[0 & 1 & 3] & [\ 0 & 1] & [2 & 3 & 4] & [\ 0 & 2 & 3] & [1 & 4] &
\end{array}$$

Figure 2.8: A sparse matrix represented in a CSR format. The red pointers indicate columns in which the corresponding values are stored. The green pointers show boundaries for each matrix row in the memory array.

A similar approach can be adapted for more coarse structures such as blocks. Given the 2×2 block-sparse matrix E that can be represented as $\hat{E}$:

$$E = \begin{pmatrix} 1 & 0 & 6 & 7 & * & * \\ 2 & 1 & 8 & 2 & * & * \\ * & * & 1 & 4 & * & * \\ * & * & 5 & 1 & * & * \\ * & * & 4 & 3 & 7 & 2 \\ * & * & 0 & 0 & 0 & 0 \end{pmatrix} \implies \hat{E} = \begin{pmatrix} L & M & * \\ * & N & * \\ * & P & Q \end{pmatrix}$$

where

$$L = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}, \quad M = \begin{pmatrix} 6 & 7 \\ 8 & 2 \end{pmatrix}, \quad N = \begin{pmatrix} 1 & 4 \\ 5 & 1 \end{pmatrix}, \quad P = \begin{pmatrix} 4 & 3 \\ 0 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 7 & 2 \\ 0 & 0 \end{pmatrix}$$

Block Sparse Row Matrix Storage Format (BSR) A variant of CSR adapted for blocks. Elements of each block are stored consecutively in an array.

values $= 1\ 0\ 2\ 1\ 6\ 7\ 8\ 2\ 1\ 4\ 5\ 1\ 4\ 3\ 0\ 0\ 7\ 2\ 0\ 0$

columns $= 0\ 1\ 1\ 1\ 2$

4-array variation:

pointerB $= 0\ 2\ 3$

pointerE $= 2\ 3\ 5$

3-array variation:

pointer $= 0\ 2\ 3\ 5$

2.4.1 Approaches

The ideal parameter candidate to be removed from a network should do no (or little to no) damage to a model's performance. The pruning procedure can be split into the following categories:

1. Scheduling: how and when do we prune, and what does the overall training looks like?
2. Location: which exactly parameters do we remove and do we make a decision layer-wise or globally across a whole model?
3. Threshold: how do we judge which parameters are more important to keep than others?

After pruning, a model may need to re-adjust some of its weights afterwards with additional training. However, if a parameter gets removed prematurely, it may inadvertently curb the architecture's potential. For this reason, we usually prune on fully (pre)trained models. One of the most popular approaches is to evaluate all parameters, prune a given percentage of the least "useful" parameters based on a heuristic and restart training to recover from damage. For example, Optimal Brain Damage (LeCun et al., 1990) trains in such a loop. While highly effective, such pruning is extremely expensive as it requires many training runs to sparsify a single model. Long training is especially an issue in NMT as a typical model trains from a couple of days to a few weeks. Prolonging that is not feasible. There is an ongoing effort to rectify this with pruning during training (Golub, Lemieux, & Lis, 2018), making retraining/tuning shorter (Frankle et al., 2019) or even pruning before training starts at all (N. Lee, Ajanthan, & Torr, 2019).

The critical aspect of pruning is deciding which parameters are worth keeping and vice versa. The most straightforward method is to look at *magnitude* of parameters. Those close to zero do not affect the output activations as much as others. Magnitude pruning has been successfully applied NMT (See et al., 2016) to sparsify recurrent neural networks (RNN), but with no speed-up or direct compression. Magnitude alone may not be enough — just the fact that a parameter is small does not mean it is not performing crucial work since even small outputs can be multiplied by larger coefficients in later layers. *Gradients* are also a viable option for thresholding. If a gradient is consistently close to zero, this parameter is not learning much as it is either obsolete or already well-optimised. The OBD algorithm (LeCun et al., 1990) uses second derivatives to prune, which requires additional calculations outside of backpropagation, less feasible for models with a large number of parameters due to quadratic growth. Molchanov, Mallya, Tyree, Frosio, and Kautz (2019) makes an argument that Taylor expansions, which can simplify into the multiplication of parameters and their gradients, can approximate the contribution those parameters make. They performed pruning on a kernel level in image recognition, but it could also be on a coefficient level. Some methods prune immediately after initialisation, in either unstructured (N. Lee et al., 2019) or structured (C. Wang, Zhang, & Grosse, 2020) way. Yao et al. (2019) combine unstructured sparsity with a light structure that aims to balance parallel workloads. They introduce a specialised matrix multiplication kernel for their structured sparsity. My goal in this thesis is to retain density to avoid extensive implementation optimisations.

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.

Another method that can induce sparsity in neural networks is *regularisation*. By additionally penalising a cost function with the magnitude of parameters (for example, L_1 regularisation), the network itself diminishes parameters towards zero. The regularisation as a sparsification mechanism will be discussed later in Sec. 2.5.

There is no limit to how pruning can be done. Researchers create various heuristics to calculate thresholds either individually across layers or globally. Each new method is supposedly better than the previous ones. Unfortunately, this research field has many issues that are yet to be addressed. Gale et al. (2019) highlight some of them, such as the lack of comparison between other similar methods, no variety or too much ambiguity in settings or architectures, experimental inconsistencies, et cetera. Z. Liu, Sun, Zhou, Huang, and Darrell (2018) also points out that training a large architecture is not necessary to obtain a smaller well-performing model and that pruning may instead serve a role as architecture searching. All the problems above become even worse when looked at through lenses of speed optimisation.

Unfortunately, much of the prior work on pruning and sparsity does not report speed or makes inference slower: Brix et al. (2020) achieved no speed-up. Meanwhile, Yao et al. (2019) reported an 87.5% sparse model took 1.6$\times$ 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$\times$ slower at 70% sparsity (Gray et al., 2017) though they did eventually achieve a 1.4$\times$ speed-up with “balanced pruning”. Among many, (Han, Pool, Tran, & Dally, 2015; Lemaire, Achkar, & Jodoin, 2018) report theoretical speed-up in FLOPs only, which Gale et al. (2019) points out that many papers inconsistently define. Dong, Huang, Yang, and Yan (2017) showed that the actual speed-up is much smaller than the theoretical. Pruning research, in most cases, omits any performance analysis, reporting on just memory compression and omitting speed analysis on purpose as there is none. In my thesis, I want to fill this optimisation gap in pruning while focusing on NMT in highly speed-competitive settings.

Here are some key issues that stem from pruning and sparsity:

1. Pruning is tedious. Using a smaller architecture may be easier to do and result in better quality.
2. Only really sparse matrices get the speed advantage from sparse algorithms and representation.
3. Most popular deep learning toolkits do not fully support sparse algebra. If they do, they are often poorly optimised, obstructed by additional “code layers” like Python et cetera.
4. Knowing that toolkits are suboptimal, researchers focus on compression and quality only.
5. As there is no actual speed-up gained, researchers perform theoretical analysis in FLOPs only, that are often inconsistent between papers.
6. More complicated sparsity patterns require low-level implementations that available libraries may not support. They may be too complex and not worth the re-implementation efforts by others.

7. Pruning research centres around image recognition tasks, which are easy and fast to train with satisfying quality after aggressive sparsification. We do not know the scope of damage offered in exchange for potential speed-up in tasks such as speech recognition or machine translation.
8. Most inference optimisation papers (Gu et al., 2018; J. Lee et al., 2020), including those on pruning (Y. Wang et al., 2020) choose weak baselines as the starting point. It is easier to remove parameters from a bigger architecture than to optimise a smaller, more robust model that is considered a current "state-of-the-art" in terms of speed. Combined with outdated training datasets, we do not know whether existing methods perform well under real-life, "in-production" conditions.

This thesis aims to address these problems by providing simple and reliable methods for structural pruning that result in actual faster inference. My experiments revolve around machine translation as the field of my choice but are universal for other sequence-to-sequence tasks such as language modelling or speech recognition. I avoid issues of optimising sparse operators by pruning structurally to create smaller but still dense architectures that can be used straightforwardly.

2.5 Regularisation

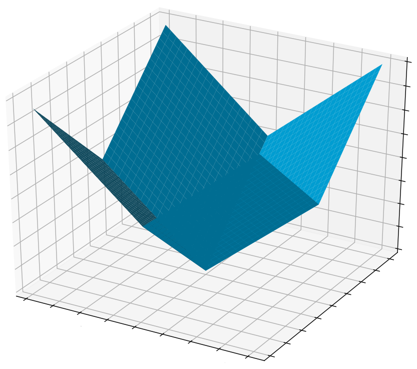
Regularisation is a process of reducing the complexity of a model to improve generalisation and prevent overfitting. For example, *dropout* masks input or parameters randomly during training at each update to make a network more flexible when still learning. However, this term primarily encompasses techniques that impose additional restrictions on a cost function, forcing a model to optimise towards a compromise between its quality and complexity.

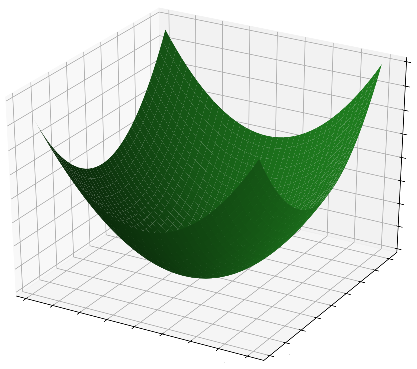
Given an input x, a label y and a predictive function f, the regularised cost function is defined as follows:

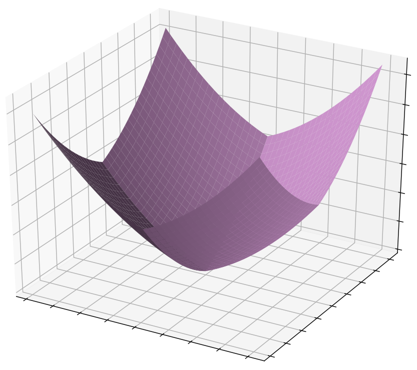
$$\min_{f} E\left(f\left(x\right), y\right) + R(f) \tag{2.11}$$

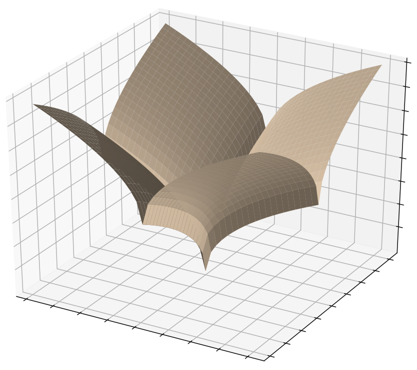
with E being a typical cost function predicting $f(x)$ (cross-entropy, mean squared error etc.) and R being a regularisation term.

In my research, I investigate regularisation methods and their pruning effects. Usually, a regulariser sparsifies a model by penalising the cost function with the magnitude of its parameters. Then, an optimiser diminishes selected parameters during backpropagation to reduce the overall cost while simultaneously minimising a typical loss.


(a) L_1 penalty (Eq. 2.14)


(b) L_2 penalty (Eq. 2.15)


(c) Elastic Net (Eq. 2.16)


(d) $L_{0.5}$ penalty (Eq. 2.17)

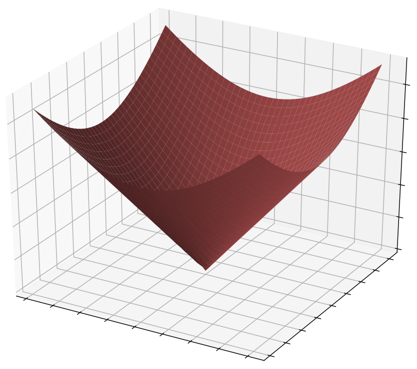
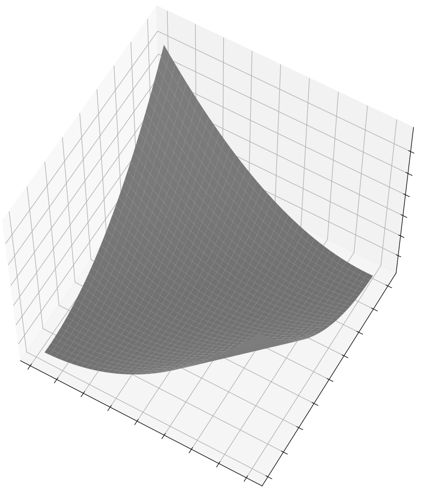
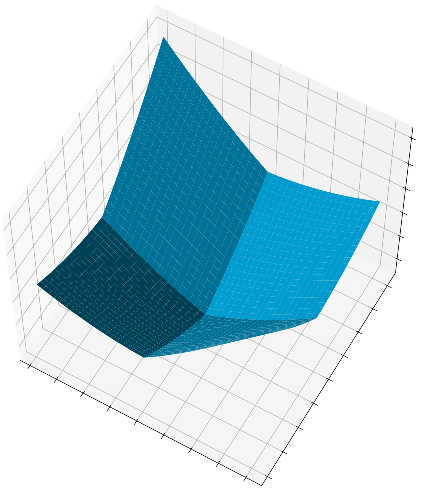

(e) Group Lasso (Eq. 2.20)

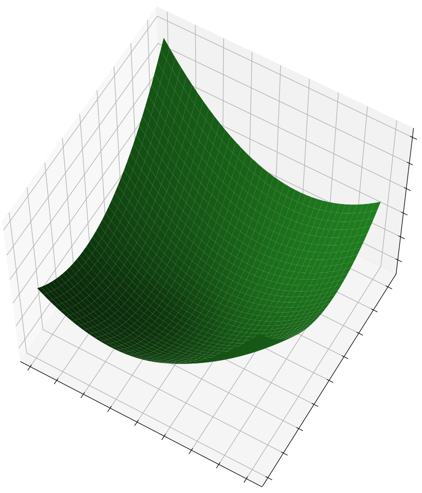
Figure 2.9: Regularisation penalties visualised for a model with two parameters.



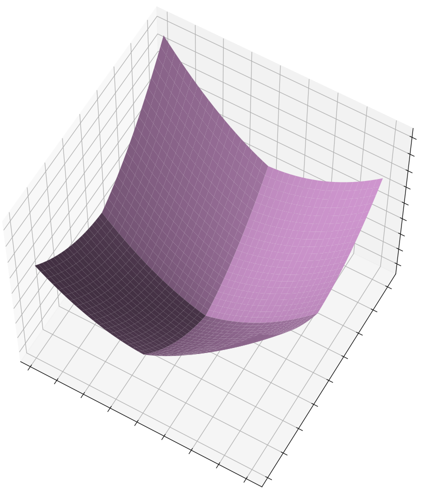
(a) Error (Eq. 2.13)



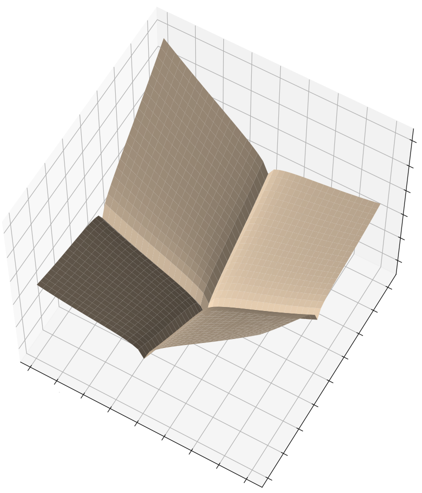
(b) Error + L_1 penalty (Eq. 2.14)



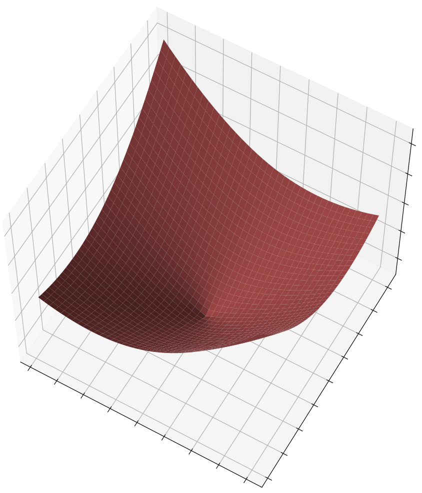
(c) Error + L_2 penalty (Eq. 2.15)



(d) Error + Elastic Net (Eq. 2.16)



(e) Error + $L_{0.5}$ penalty (Eq. 2.17)



(f) Error + Group Lasso (Eq. 2.20)

Figure 2.10: Regularised cost functions visualised for a model with two parameters.

The generalised notion behind such regularisation is to use L_p norm. To quickly explain and visualise the behaviour of the most popular regularisation functions, let us start with a toy model.

Let f be a simple logistic function with two inputs x_1 and x_2 defined as:

$$f(x_1,x_2,w_1,w_2) = \frac{1}{1+e^{-(w_1x_1+w_2x_2)}} \tag{2.12}$$

parametrised with two weights w_1 and w_2.

In the following analysis, we will approximate $y = f(x_1,x_2,w_1 = 0.5,w_2 = 0.25)$ for $x_1,x_2 \in [-1,1]$ and predicted $\hat{w}_1,\hat{w}_2 \in [-1,1]$.

Let us define E as a mean squared error function:

$$E = (y - f(x_1,x_2,\hat{w}_1,\hat{w}_2))^2 \tag{2.13}$$

Since we only have two variables, the error function can be visualised as presented in Fig. 2.10a.

Now, let us look into the effect of a few popular regularisation methods on this model. In all of them, a hyperparameter λ controls the strength of the regulariser and its contribution towards a total cost.

2.5.1 L_1 regularisation (LASSO)

L_1 regularisation, also known as LASSO (Least Absolute Shrinkage And Selection) (Tibshirani, 1996), penalises the cost function by adding a sum of absolute parameters to it:

$$R(f) = \lambda \sum_{i=1}^{n} |w_i| \tag{2.14}$$

When minimising a loss, an L_1 penalty strives to zero-out selected parameters. In Fig. 2.9a, the *sharp edges* correspond to solutions with specific parameters being zero. When added to a cost function (Fig. 2.10b, it steers the gradient descent to fall into those narrow paths, encouraging zero coefficients. In addition, L_1 is robust to outliers and has multiple possible solutions since it applies Manhattan distance. Due to this behaviour, L_1 regularisation is a popular choice for *coefficient pruning* as it does not require additional expensive analysis. It is easy to utilise and produces solid baseline results in many tasks.

2.5.2 L_2 regularisation (Ridge regression)

Rather than aiming to prune individual coefficients, L_2 regularisation (also called Ridge regression) (Hilt & Seegrist, 1977) tends to reduce the magnitude of parameters evenly by adding L_2 norm to the cost function:

$$R(f) = \lambda \sum_{i=1}^{n} (w_i)^2 \tag{2.15}$$

The root function in L_2 norm is omitted for gradient simplicity. The derivative of a root is $\frac{1}{2\sqrt{x}}$ which makes gradient form a little less elegant. Minimising the cost function subject to $||w||_2 < c$ is equivalent to minimising $||w||_2^2 < c^2$. Root function scaling the sum of parameters down can be considered a part of the λ scalar and thus omitted.

This type of regularisation is effective in codependent features as L_2 reduces their variance. As illustrated in Fig. 2.9b and 2.10c, the penalty has an *oval* shape, which encourages the parameters to get smaller but not necessarily extremely small, as in almost zero. It can constrain coefficient norm while keeping all parameters. For this reason, it is not the best choice to inflict explicit sparsity on a model.

2.5.3 L_1 + L_2 regularisation (Elastic Net)

Individually, L_1 and L_2 have their advantages and disadvantages. For example, LASSO may select only one variable from a group of highly correlated parameters, which may not be ideal. The addition of the quadratic penalty eases the limitation mentioned above. *Elastic net* combines both LASSO and ridge regression:

$$R(f) = \lambda_1 \sum_{i=1}^{n} |w_i| + \lambda_2 \sum_{i=1}^{n} (w_i)^2 \tag{2.16}$$

λ_1 and λ_2 are usually scaled as $\lambda_1 = \lambda$ and $\lambda_2 = 1 - \lambda$, so that the penalty can be balanced and, if needed, easily turned to only L_1 or L_2 respectively.

2.5.4 Non-convex and non-differentiable L_p regularisations

L_p norm can be generalised as:

$$R(f) = \lambda \sum_{i=1}^{n} ||w_i||_p = \lambda \sqrt[p]{\sum_{i=1}^{n} (w_i)^p} \tag{2.17}$$

LASSO can zero-out coefficients but may also over-shrink those it decided to retain. This behaviour inspired an interest in using lower norms. L_p with $p \in [0,1)$ results in a spiky non-convex space which, in theory, should favour edges and axes during optimisation. However, the non-convexity of the problem leads to computational complexity, making its optimisation NP-hard. Fig. 2.9d and 2.10e show $L_{0.5}$ penalty and it being applied to the cost function.

The edge case of L_0 is not a proper norm as it corresponds to the total number of non-zero parameters from the best-subset selection. It is non-differentiable, making it a discrete optimisation problem. Reparametrisation of a cost function can achieve optimise it through gradient descent by adding binary gates with L_0 "norm" indicating the number of gates being active.

With z_i being a gate for a parameter $w_i = \hat{w}_i z_i$, let us define $q(z_i|\pi_i) = Bernoulli(\pi_i)$ as a Bernoulli distribution over each gate z_i. The L_0 penalty can be then reparametrised as:

$$R(f) = \lambda \sum_{i=1}^{n} \pi_i \tag{2.18}$$

Concrete hard distribution (Louizos, Welling, & Kingma, 2018) can be used to smooth out the z_i gates by using hard sigmoid:

$$\hat{z}_i = \begin{cases} 1 & \text{if } z_i > 1 \\ 0 & \text{if } z_i < 0 \\ z_i & \text{otherwise} \end{cases} \tag{2.19}$$

The gradient descent optimiser updates the parameters π_i of the distribution over those gates, resulting in them being open ("1") or closed ("0").

2.5.5 Group Lasso

So far, I have discussed techniques that regularise and sparsify a model on a coefficient level. However, there are many cases where we may want to impose a structural sparsity instead. *Group lasso* (Yuan & Lin, 2006) is an umbrella term for methods that simultaneously diminish parameters in clusters. In neural networks, we are primarily interested in the most naive case of removing *non-overlapping* groups.

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

$$R(f) = \lambda \sum_{g=1}^{|G|} \sqrt{\sum_{i=1}^{|G_g|} \left(w_i^g\right)^2} \tag{2.20}$$

This penalty can be interpreted as applying ridge regression over parameters within each group and LASSO over all groups. Simon and Tibshirani (2012) recommends additionally orthonormalising the L_2 penalty by the number of elements if groups are of different sizes.

As can be seen in Fig. 2.9e and 2.10f, a group lasso has a funnel shape which encourages the optimiser to zero out both parameters at the same time, in contrast to a simple lasso which may end up pruning only one of them. Fig. 4.3 shows the outcome of applying group lasso over 8x8 blocks of parameters in a feedforward layer of a transformer.

In Chapter 4, I further explore structural pruning using group lasso regularisation in the context of improving inference speed in NMT.

2.5.6 Multi-loss aggregation

A regularisation term is an additional loss added to a typical cost function that optimises towards translation quality. As typical for sequence-to-sequence tasks, batches are not necessarily constant due to sentences having different lengths and how they are packed in memory. Thus, averaging a cross entropy cost over the batch size is a standard practice. In the multiloss scenario, we need to consider scaling the penalty by batch size as well. There are various ways to reduce multiple losses into a single cost function:

Sum Both cross-entropy and regularisation terms are summed together first and then averaged over the batch size.

$$E(batch) = \frac{1}{|batch|}\left(\sum_{x\in batch} CE(x) + \lambda * \sum_{l\in layers} R(l)\right) \tag{2.21}$$

Mean Each loss is averaged individually: perplexity by the batch size, regularisation penalties by, for example, the number of layers regularised et cetera, and then summed up.

$$E(batch) = \frac{1}{|batch|}\sum_{x\in batch} CE(x) + \lambda \sum_{l\in layers} R(l) \tag{2.22}$$

Scaled The perplexity is averaged as usual, but all other losses are scaled up by the batch size instead. Usually, a smaller λ is used here.

$$E(batch) = \frac{1}{|batch|}\sum_{x\in batch} CE(x) + \lambda |batch| \sum_{l\in layers} R(l) \tag{2.23}$$

2.6 Architecture search

It is not easy to develop a brand new architecture design and a set of hyperparameters working well with it. Doing so by hand takes endless seeking and so-called intuition.
Neural architecture search (NAS) automates it by exploring an ample search space of layers in models while estimating their performance. NAS requires training a considerable number of models to test their performance which is extremely costly. For example, Zoph and Le (2016) used reinforcement learning to concurrently train 800 models at any time using 800 GPUs for 3–4 weeks. Evolutionary algorithms are also a prevalent approach (Real, Aggarwal, Huang, & Le, 2019). In each "mutation" step, parent models and their hyperparameters get altered (by adding/removing/changing a type of some layers), trained, evaluated and added back to the population. The process repeats in a loop until it finds a satisfying model.

Most architecture search research concentrates on image classification problems as they are fast to train, unlike NMT. For sequence-to-sequence tasks, So, Liang, and Le (2019) have found a new transformer architecture named an "evolved" transformer. The model is static — it does not adapt itself to datasets but can be considered another architecture flavour to try. So et al. (2019) report that their big model tested on NMT has only 7M parameters and is smaller by 37.6 than a standard big transformer. However, they do not mention actual inference speed. There is no direct indication that this brand new architecture is any better at inference speed despite its compression.

Pruning itself can be interpreted as a "downward" form of architecture search, in which we only remove paths, structures and individual parameters. This search space is usually much smaller and manageable to deal with.

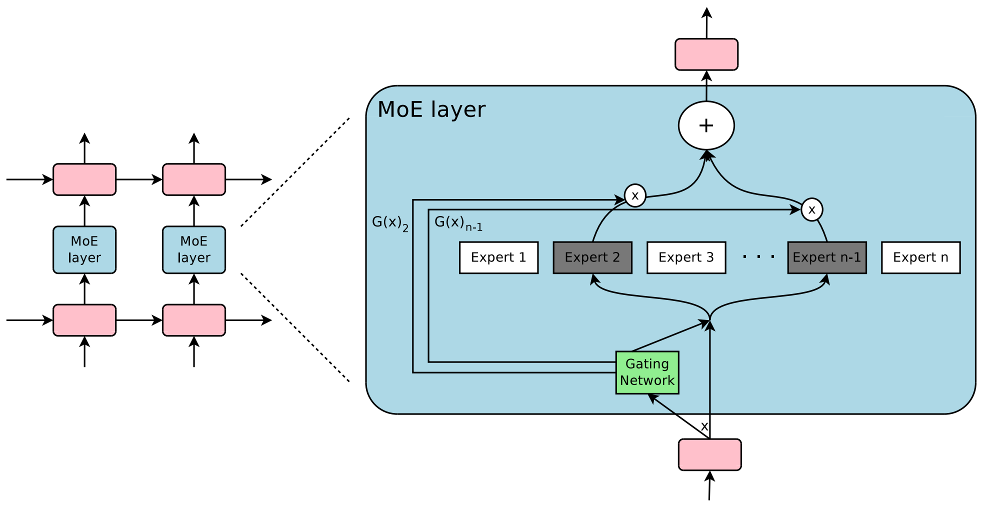


Figure 2.11: A Mixture of Experts layer (illustrated by Shazeer et al. (2017)). A gating network controls which subnetworks get activated in a forward pass.

2.7 Mixture of Experts

Mixture of Experts (MoE) is another machine learning technique akin to an architecture search In it, only parts of a neural network are active for a specific input, which reduces the computational workload for huge models. Shazeer et al. (2017) introduced an architecture consisting of up to thousands of feed-forward *experts* with trainable gates that determine which of these sub-networks to use for each example. The main problem with this method is that experts can become *undertrained* as they do not update with all the training data; this is especially problematic in machine translation due to the complexity of the task.

Chapter 3

Structural Pruning of Transformer with the Lottery Ticket Hypothesis

As introduced in Chapter 2 (see Eq. 2.4 and Fig. 2.3), the attention mechanism is quadratically complex with regards to the length of a sentence it attends over. This cost becomes even more evident as we stack many layers in a model to get the best possible quality. For example, a popular neural network used for language modelling called *BERT* (Bidirectional Encoder Representations from Transformers) is a transformer architecture that typically consists of 12 to 24 layers, with each having 12 to 16 attention heads per layer. That means between 144 and 384 attention heads in a single model make it substantially resource-hungry to train and translate. The rapid growth in the size of neural architectures creates a necessity for improved generalisation and *pruning*, which directed research into investigating the inner workings of the attention mechanism in particular.

3.1 Motivation

The question is: **do we need all of those attention heads, or can they be pruned? If so, how much inference time will that save?** Michel, Levy, and Neubig (2019) observed that many heads “can be removed at test time without significantly impacting performance”, with some layers in a model even being reduced from sixteen to a single head. Moreover, those heads that are particularly important seem to emerge at the beginning of training. Serrano and Smith (2019) further explored the notion of head *importance* and attention interpretability. Using a text classification task, they analysed if “high attention weights correlate with greater impact on model predictions” as they are often treated as human-interpretable and of higher significance. Their experiments concluded that “anything may flip a model’s decision for tasks with a much larger output space such as language modelling or machine translation”. This means that attention heads that may seem insignificant at first glance still contribute to the output.

Most of the attention heads do not perform identifiable roles, and many of their calculations are excessive. Xiao, Li, Zhu, Yu, and Liu (2019) noted that plenty of attention layers share similar distributions, which suggests a high level of redundancy. Since attention is expensive to use in a decoder, replacing it entirely with a less costly alternative would be the best option. For example, a linear approximation such as *SSRU* (Simpler Simple Recurrent Unit) can replace a decoder self-attention. However, this does not optimise encoder or context attention.

All the research above indicates that attention has the potential to be downsized. Voita et al. (2019) analysed the attention mechanism further and noticed that majority of heads are "useless". They either do not have linguistically interpretable roles or cannot make reliable choices when making alignments. While contradicting the previously mentioned analysis by Serrano and Smith (2019), Voita et al. (2019) deems *confident* heads — those that assign large weights on average — to be overall more important to a model as they seem to perform most of the work in it. The essential contribution of their analysis says that "specialised heads do the heavy lifting, the rest can be pruned". To put it to the test, they pruned attention heads using a structural L_0 regularisation (see Sec. 2.5.4). They showed it is possible to prune a significant amount of attention heads from NMT models with negligible damage to quality. However, they did not investigate how slicing and removing heads affects speed. This lack of analysis inspired me to look into pruning of the attention mechanism in the context of actually speeding up inference. In particular, I want to prune models in highly-optimised deployment settings characterised by a different model configuration than usual such as fewer layers or reduced layer dimensions. It is possible to prune all attention heads from a layer (as well as other parameters such as feedforward layers) as it will simply pass the input forward through residual connections (Fig. 2.4).

The study's primary purpose in this chapter is to understand the workload reduction of the attention mechanism and the potential of extending these solutions to other transformer structures such as feedforward layers. The chapter is split into three major sections that present the research findings focusing on key themes that have led towards the overall direction of this dissertation. The first idea looks into reinventing the traditional attention mechanism and its heads into a Mixture-of-Experts approach. The next one discusses the reduction of attention entropy to force it to make confident decisions which, in turn, may require fewer calculations to perform. The last idea directly follows the previous two, inspired by their outcome, and focuses on the structural pruning of a transformer architecture using the Lottery Ticket Hypothesis (LTH) (Frankle & Carbin, 2019). In particular, it analyses attention pruning for faster inference in machine translation. Later in this chapter, I apply blockwise pruning on feedforward layers, generalising structural block sparsity with the LTH method and showing the results of stacking both attention and feedforward layers being pruned. The analysis of pruning results in this chapter serves as a direct motivation for future work in the thesis.

The research on pruning attention using lottery tickets presented in this chapter has been published in Behnke and Heafield (2020) and further applied in Bogoychev et al. (2020).

3.2 Attention mechanism as a Mixture-of-Experts

As a first exploration of the attention mechanism topic, I speculated that a transformer could *choose* which attention heads to use. As Voita et al. (2019) has later shown in their research, attention heads perform different functions within a model, often corresponding to specific linguistic roles. However, not every word in a sentence needs all of that (grammatical or not) analysis. Most probably only require peeking at neighbouring words, connecting a noun to its verb, et cetera.

3.2.1 Methodology

An attention head could be described as an *expert* that has an assigned job. Inspired by Shazeer et al. (2017), I modified the attention mechanism to have it learn which heads to use **per word** and mask the calculations appropriately. I apply *Mixture-of-Experts* network by training a *gating network* alongside a typical architecture. The gate is calculated based on attention input (query) and scales the attention output on a word level.

With $\mathbf{Q},\mathbf{K},\mathbf{V}$ defined as keys, queries and values (see Eq. 2.1 and 2.2), σ being a softmax function (see Eq. 2.3), the gated attention is defined as follows:

$$\begin{aligned}\text{Gated Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) &= Gate(\mathbf{Q}) * \text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) \\ &= Gate(\mathbf{Q}) * \sigma\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}}\right)\mathbf{V}\end{aligned} \tag{3.1}$$

Following Shazeer et al. (2017), I try two gating approaches:

Softmax Gating All attention heads are used but scaled to varied degrees.

With an additional trainable layer W_g, the softmax gate is simply defined as:

$$Gate(\mathbf{Q}) = \sigma(\mathbf{Q}W_g) \tag{3.2}$$

Noisy Top-K Gating One of the issues in the Mixture-of-Experts approach is balancing the number of training examples seen by each subnetwork. A model may easily favour a few selected experts simply because the gates chose them in the first training batches. According to Shazeer et al. (2017), adding a tunable Gaussian noise helps with load balancing. Besides the W_g layer, there is also W_{noise} that controls how much noise is added to the gate during training. It is disabled during inference.

Finally, the Top-K gating looks as follows:

$$H(x)_i = (\mathbf{Q}W_g)_i + \phi \cdot \hat{\sigma}(\mathbf{Q}W_{noise})_i \tag{3.3}$$

$$TopK(z_i, K) = \begin{cases} z_i, & \text{if } z_i \text{ in top K elements of } z \\ -\infty, & \text{otherwise} \end{cases} \tag{3.4}$$

$$Gate(\mathbf{Q}) = \sigma(TopK(H(\mathbf{Q}), K)) \tag{3.5}$$

$$\tag{3.6}$$

with $\phi = X \sim \mathcal{N}(\mu, \sigma^2)$ being a sample from a Gaussian distribution.
$\hat{\sigma}$ represents a softplus function that is a smooth approximation to ReLU:

$$\hat{\sigma}(x) = \log(1 + e^x) \tag{3.7}$$

3.2.2 Setup: a base transformer (English→German)

Data

I use English→German parallel data allowed by the constrained condition of the WMT17 news task (Bojar et al., 2017). The corpus consists of $\sim 4.56M$ sentences.

The preprocessing steps are as follows:

1. Normalisation — the standardisation of punctuation (such as quotations styles across different languages), removing non-Unicode characters, removing extra spaces, et cetera.
2. Tokenisation — splitting words from punctuation.
3. Truecasing — lowercasing all words beside entities, names, et cetera.
4. BPE segmentation — splittings compound words to smaller subwords using a Byte Pair Encoding algorithm (Sennrich, Haddow, & Birch, 2016).

Normalisation, tokenisation, truecasing are done using Moses[1] scripts. BPE was trained jointly with 36000 operations. The vocabularies are shared for both languages in a pair and contain 36000 words.

All models are evaluated on untokenised WMT data using BLEU calculated with sacreBLEU (Post, 2018). I develop models on the WMT13 devset and use testsets from 2014, 2015 and 2016 for testing.

1. `https://github.com/moses-smt/mosesdecoder`

Architecture

All models follow the settings of a base transformer with hyperparameters predefined by Vaswani et al. (2017a). Both the encoder and decoder have 6 stacked layers with 8 *fixed* attention heads of size 64 (see Tab. 3.1).

Type	Size
dim_{emb}	512
dim_{FFN}	2048
dim_{head}	64
Heads per layer	8
Encoder layers	6
Decoder layers	6

Table 3.1: A base transformer base architecture setup.

The gradient descent optimizer is Adam (Kingma & Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\varepsilon = 10^{-9}$. The learning rate is set to 0.0003, which warms up for the first 16000 updates and then decays following the inverse square scheme (Vaswani et al., 2017a). Each model is exponentially smoothed with 0.0001 and optimised for cross-entropy averaged over words in a batch. A model stops training after ten consecutive stalled checkpoints.

All experiments are trained using Marian NMT[2] (Junczys-Dowmunt, Grundkiewicz, et al., 2018), a C++ toolkit, on 4 $\times$ NVIDIA P100 (16GB) GPUs. The batch size is dynamic, aiming to fill 13 GB of each GPU memory, with approximately 700 sentences per batch.

3.2.3 Experiments

To test the mixture-of-experts methods, I train English$\rightarrow$German base transformer models (see Sec. 3.2.2) with both softmax and Top-4 attention mechanisms. The training progression is presented in Fig. 3.1. The average cross-entropy overfits faster than in the baseline, though the best BLEU validation stays competitive. Most interestingly, the models trained with gated attention have a smoother training progression at the beginning — instead of starting from 1–3 BLEU points as is typical in a transformer, those models validate with 11-12 BLEU in the first checkpoint.

2. `https://marian-nmt.github.io`

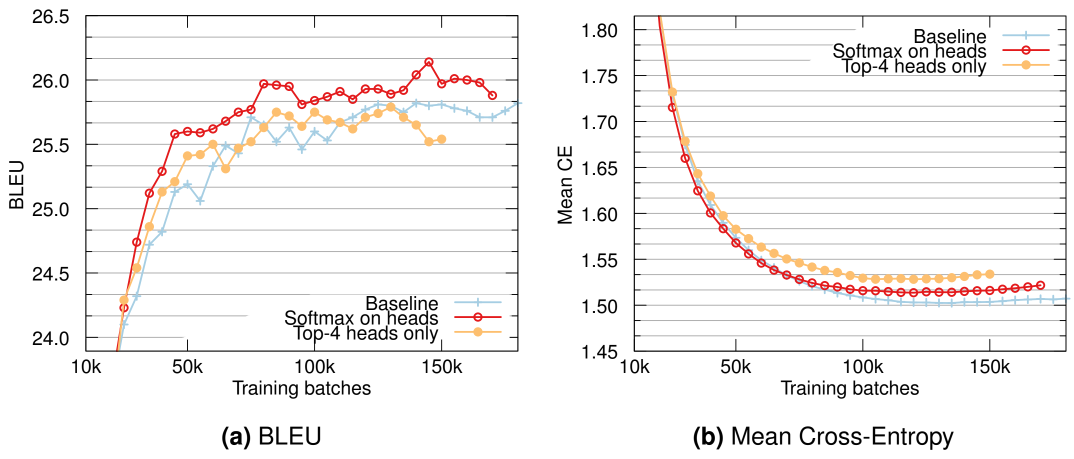


Figure 3.1: Training validation of transformer models with a gated attention.

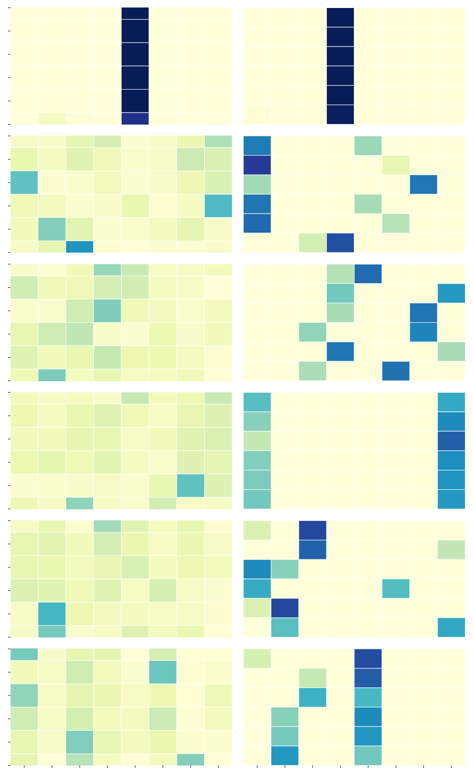
The most interesting part of this experiment is how the gating network behaves during the inference. Hence in Fig. 3.2 I visualise the attention gates for a short input sentence *hi how are you ? EOS*.[3] The input has six subwords, and the output is generated with a beam size of 6. Each pair of heatmaps represents a layer in the encoder, decoder and context attention, with 6 stacked heatmaps representing 6 layers. The heatmaps on the left represent softmax gates, and the ones on the right represent Top-K gates. Since the gates mask the attention **per word**, each heatmap is 6×8 for 6 words in an input/beam and 8 attention heads per layer.

Overall, Top-4 gates confidently favour specific heads rather than being spread thin as much as possible within the constraints of 4 heads. However, even softmax gates often align to a single chosen head, especially while decoding. In many cases, the gating mechanism strongly selects the same *single head for the whole sentence*. This direct observation inspired me to look into attention pruning.

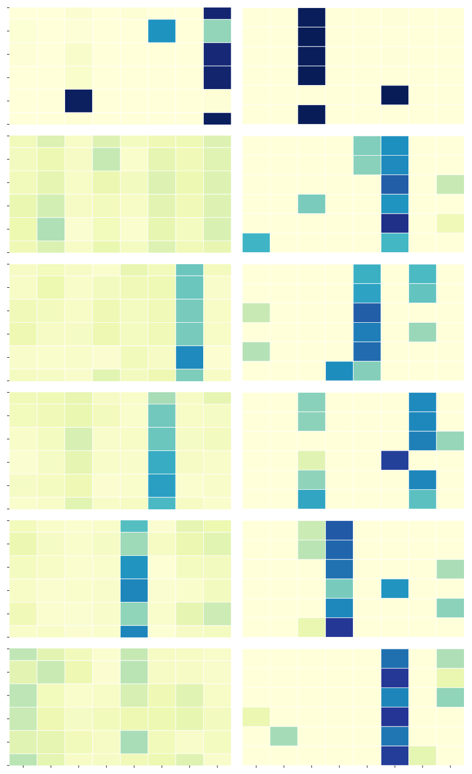
3.2.4 Conclusions

Mixture-of-experts are getting increasingly popular in large language models (Artetxe et al., 2021; Gao, Liu, Zhao, Lu, & Wen, 2022) and multilingual models (Y. J. Kim et al., 2021). In this research, I investigated applying a mixture-of-experts approach onto the attention mechanism. Unfortunately, the word-level gating methods would require dynamic slicing, which would be challenging to optimise speed-wise across batches. Since the results were not groundbreaking and the potential for speed-up small, I decided to give up this research direction of dynamic attention *selection* to instead focus on attention *pruning* with efficiency in mind.

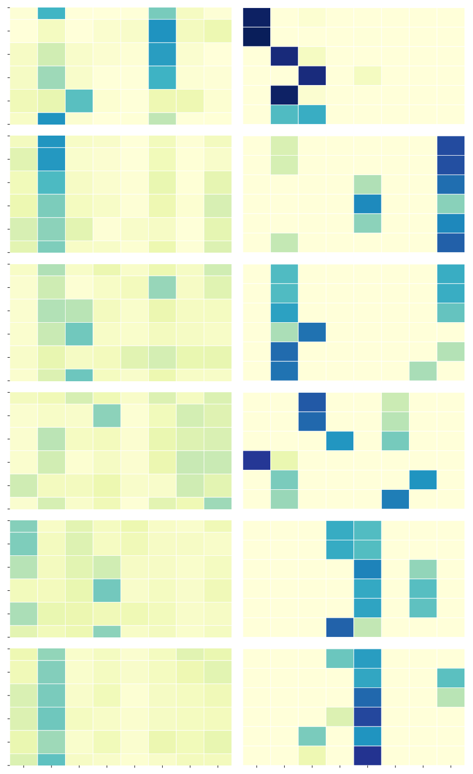
3. EOS = a special “End of Sentence” token, it signalises to a network to stop generating translations.



(a) Encoder



(b) Decoder



(c) Context

Figure 3.2: An example output of the softmax (left columns) and noisy Top-K (right columns) gating networks in encoder, decoder and context attention layers. Each heatmap is $6 \text{x} 8$ for 6 words in an input/beam and 8 attention heads per layer.

It is worth noting that recent work by J. Li, Cotterell, and Sachan (2021) has also utilised a similar approach to prune attention heads. Their paper introduces a "differentiable subset pruning" of attention heads, which trains Top-K gates with a Gumbel noise instead of Gaussian, resulting in a hard sparsity at test time. This method is less dynamic than a typical Mixture-of-Experts as it requires a user-specified hard constraint on the number of used heads, and the rest is subsequently pruned rather than selected on the fly. They report a 33% speed-up in inference time and 24% decrease in model size while maintaining over 80% accuracy on the MNLI dataset using a transformer-based language model BERT (Devlin, Chang, Lee, & Toutanova, 2019).

3.3 Reducing attention entropy to make it confident

In their paper, Voita et al. (2019) notice that heads of high quality are those that make "confident" decisions, as in assigning large weights to one specific word. Partially inspired by the notion of IBM alignment models (P. F. Brown, Della Pietra, Della Pietra, & Mercer, 1993) incentivising alignment fertility, I want to test whether we can explicitly train models with more confident alignment decisions.

The most confident head regularly assigns a weight close to 1 to a single word and almost 0 to others. In other words, the entropy of such heads should be minimal. Otherwise, a model with a higher penalty added to the cost function should get punished. This should minimise the alignment fertility into the minimal subset of the most important words.

3.3.1 Methodology

I modify a cost function to include the sum of entropies from weights of all attention heads in a model:

$$E^*(x) = E(x) + \lambda \sum_{h \in heads} \sigma(\frac{\mathbf{Q_h K_h}^\top}{\sqrt{n}}) \cdot log(\sigma(\frac{\mathbf{Q_h K_h^\top}}{\sqrt{n}}))$$

where $E(x)$ is a typical cross-entropy loss, σ is a softmax function and $\mathbf{Q}$ and $\mathbf{K}$ being queries and keys in the attention mechanism.

The additional penalty is scaled by a hyperparameter λ so that it does not overpower the standard cost.

3.3.2 Experiments

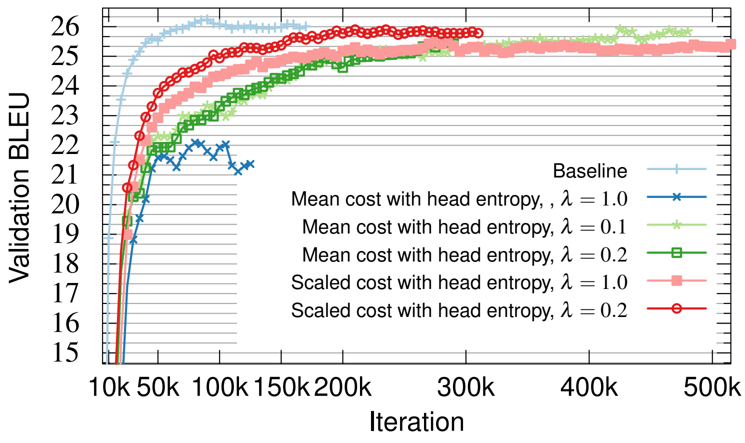


Figure 3.3: Training progression for models trained with head entropy penalty with mean and scaled cost functions.

I train English→German models using the setup described in Sect. 3.1 with the additional entropy penalty added to the cost function. I experiment with two types of cost functions: mean (Eq. 2.22) and scaled (Eq. 2.23). As can be seen in Fig. 3.3, forcing a cost function to include entropies of attention weights has considerably slowed down the convergence of the models. A model with an entropy penalty requires two to five times more batch steps to reach its best validation results. I observed no distinctive difference in attention behaviour between a baseline model and the ones trained with this method. Since the main goal was to maintain the translation quality at least, and there are no other advantages, this research direction has been abandoned instead to focus on pruning.

3.4 Structural pruning of attention heads using Lottery Ticket Hypothesis

Pruning is fast becoming a staple approach to optimisation, and it is no different for machine translation. Around the time of the research presented so far, Voita et al. (2019) have published their work on the analysis of attention mechanisms in a transformer and, most importantly, pruning of it. Unfortunately, their paper did not focus on the inference optimisation aspect of pruning. This, in turn, actively motivated me to pursue this direction myself and look into attention pruning for faster translation.

The following section presents the setup for all languages and architectures used in experiments. I follow with the introduction of a pruning method called the *Lottery Ticket Hypothesis* which serves as a basis for my pruning approach. Next, I proceed to outline the methodology of my attention pruning method. Then, I establish solid baselines with related work. Finally, I present the results of the experiments themselves with in-depth analysis.

In order to investigate how generalisable my pruning methods are, I explore a wide variety of use cases. There are two “types” of experiments: regular and optimised.

First, I focus my research on exploring standard transformer base and big models. Here I concentrate on two language pairs: English→German (base) and Turkish→English (big). The first one is considered a high-resource language pair, with English not being a target language. In contrast, Turkish→English is low-resource, even with additional back-translated data.

Last but not least, I apply my pruning method to a highly-optimised English→German model trained under a knowledge distillation scheme for a WNGT2020 efficiency shared task (Heafield et al., 2020). The goal is to test the real impact of this pruning method on translation speed when used “straight out of the box” in efficiency-focused settings.

3.4.1 Background: Lottery Ticket Hypothesis

As introduced in Sect. 2.4, coefficient sparsity is the most popular form of pruning due to its relative non-invasive manner: it is easier to remove individual parameters across dimensions without compromising on quality than removing larger groups of parameters in a single instance. The usual approach to pruning assumes that a model is converged first and pruned second, which requires either extended tuning or complete retraining.

Frankle and Carbin (2019) have introduced **Lottery Ticket Hypothesis**, a pruning method that leads to superior results to training from scratch and in comparison to other pruning regimes. The hypothesis states as follows:

> *A randomly-initialised, dense neural network contains a subnetwork such that – when trained in isolation – it can match the test accuracy of the original network after training for at most the same number of iterations.*

In other words, according to the hypothesis, some parts of the neural network are luckily initialised, resulting in them performing most of the work. Larger networks usually outperform smaller ones because they have more chances to sample “good” parameters in the initialisation lottery. A *winning ticket* is a combination of parameters that, when trained in isolation, achieves similar or better quality in contrast to the whole network. This ticket can be formally expressed as a binary mask applied to parameters.

As defined by Frankle and Carbin (2019), the simplest variant of this hypothesis implies that this winning ticket is generated by training at least twice. Given a model $f(x;\theta)$, the following steps form the procedure:

1. Initialise parameters randomly to θ_0 (e.g. Glorot distribution (Glorot & Bengio, 2010)).
2. Train a model for j iterations, resulting in parameters θ_j.
3. Prune $p\%$ of the parameters, creating a mask m.
4. Apply the mask m to θ_0, resetting the rest of parameters to initial values.
5. $f(x; m \odot \theta_0)$ is the winning ticket of the lottery.
6. Train $f(x; m \odot \theta_0)$ until convergence.

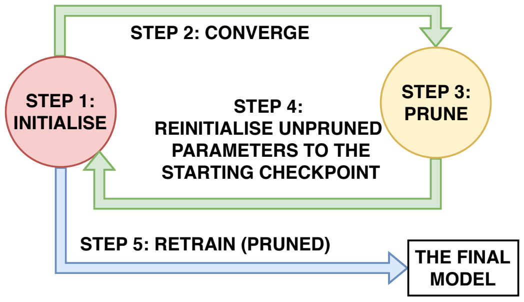


Figure 3.4: One-shot lottery ticket pruning.

This method is quite aggressive as parameters get removed in one swoop, which may be detrimental to overall quality. For this reason, Frankle and Carbin (2019) focused on *iterative* pruning in their work. In this approach, steps 2–4 are repeated n times, with each iteration pruning a fraction of parameters from a model. In their follow-up work, Frankle et al. (2019) have shown that iteratively pruning a model uncovers smaller and better quality subnetworks in comparison to pruning just once at the end. To avoid full training loops in their iterative approach, Frankle et al. (2019) introduced *late resetting* and *early turnaround*. Late resetting reverts parameters after pruning back to a checkpoint from the early stages of training ($\theta_{i>0}$), not to the starting initialisation (θ_0). It can just be the same checkpoint every time, meaning we pretrain a model first. This checkpoint may also dynamically change. For example, Brix et al. (2020) used θ_{j-1} instead of θ_0. Early turnaround means a model does not need to be fully trained to make a pruning decision but can do so much earlier into training. Both of these methods combined shorten the training time of each step in the iterative lottery scheme.

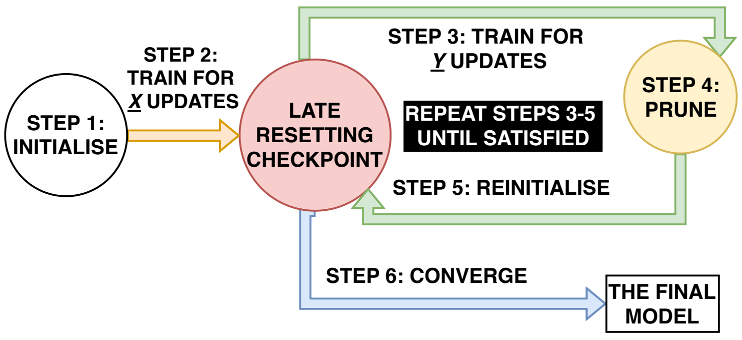

STEP 1: INITIALISE
STEP 2: TRAIN FOR X UPDATES
LATE RESETTING CHECKPOINT
STEP 3: TRAIN FOR Y UPDATES
STEP 4: PRUNE
REPEAT STEPS 3-5 UNTIL SATISFIED
STEP 5: REINITIALISE
STEP 6: CONVERGE
THE FINAL MODEL

Figure 3.5: Iterative lottery ticket pruning.

Lottery ticket pruning has been successfully applied to natural language processing tasks, including NMT (Yu et al., 2020). The winning ticket for that task was “remarkably robust to pruning” of singular weights if embeddings were spared from pruning. However, they also noted a linear drop in BLEU with sparsity. In turn, Brix et al. (2020) succeeded in applying the stabilised lottery ticket hypothesis to prune individual coefficients from a transformer in NMT settings with good results. In their experiments, a stabilised version of pruning with the lottery ticket method damages translation quality by 2 BLEU points while removing 80% of all parameters. They improved upon that further by proposing a mix of the lottery ticket and magnitude pruning. They aimed to compress a model and did not report any speed results, subsequently clarifying after their conference presentation that they did not achieve a speed improvement.

3.4.2 Methodology

Voita et al. (2019) showed that many attention heads could be pruned in a fully trained NMT model, but removing the same heads before training yielded lower quality. In my research, I investigate the third way: pruning heads in early training using the lottery ticket hypothesis. Empirically, this method enables even more pruning, which is helpful for faster machine translation.

Reinitialisation of a model with the same pruned structure underperformed in Voita et al. (2019), which is consistent with the lottery ticket hypothesis. Prior lottery ticket research prunes individual parameters to form a sparse network; I show that this logic extends to entire structures such as transformer heads. I follow stabilised lottery ticket strategies (Frankle et al., 2019) to prune in early training, achieving a better trade-off between pruning and quality than pruning after training (Voita et al., 2019).

The main goal is faster inference speed for machine translation deployment with minimal impact on quality. Pruning heads means they can be removed from the model entirely (with other heads shifted down), resulting in a layer configured to have fewer heads. Unlike most work on pruning (Gale et al., 2019; Zhu & Gupta, 2017), there is no need for sparse matrices, block-sparse matrix operators, or additional masking. In particular, we go further than Voita et al. (2019) by removing heads rather than masking them.

In this work, I combine findings of both Voita et al. (2019) (“what”) and Frankle and Carbin (2019) (“how”) to prune attention heads. First, we define a training scheme based on an iterative approach that does not require full convergence of a model each time partial pruning occurs. To analyse the impact of pruning in various settings, I experiment with a stock system across two language pairs: Turkish$\rightarrow$English and English$\rightarrow$German and a highly optimised setup with English→German. The experiments show the remaining attention heads creating similar architecture patterns across both high- and low-resource language pairs. My work also confirms the findings of both previously mentioned papers that it is impossible to train a model with such reduced architectures from scratch without damaging the quality.

"How to prune?" — Structural Lottery Ticket Hypothesis I use a stabilised variant of the lottery ticket procedure as a basis for the pruning approach to the problem. In structural pruning, one could train a whole model, identify unlucky heads with a pruning heuristic, and retrain the pruned model starting with the same initialisation. However, removing most attention heads in one go seems too drastic using a simple heuristic since other heads in layers may adapt to having fewer parameters. The roles of pruned heads may even transfer to those still active. For all these reasons, I apply a loop that iteratively prunes attention heads guided by partial training as described by Frankle et al. (2019). More details can be found in Sec. 3.4.1 describing the lottery ticket hypothesis. The detailed training scheme is presented in Figure 3.5.

First, I train a model for a set number of updates and keep it as a late resetting checkpoint. Then the pruning phase starts — the model trains for a while, and selected heads are removed to have other parameters reinitialised to the checkpoint mentioned earlier at the end. That loop repeats until we are satisfied with how many attention heads got pruned. Finally, the smaller model can be converged.

"Why prune it?" — Attention Confidence The lottery ticket hypothesis explains how pruning should progress, but the question remains: which heads qualify as pruning candidates in each lottery iteration? Inspired by Voita et al. (2019), we are mostly interested in heads that are *confident* in their decisions. Their paper defines an attention head as *confident* when it assigns a large weight to one of the words within a sentence. That head should routinely make strong alignments to be considered a candidate to remain in a model.

When a head appears, its softmax layer computes a probability distribution over the words it attends to. I record the maximum of this probability distribution as confidence. For example, a context head attends over source words s and the confidence score can be formally defined as:

$$c = \max_s \text{attention}(s) = \max_s \sigma(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V} \tag{3.8}$$

for $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ being query, key and value with σ being a softmax function.

These confidence values get averaged over all times the head appears while translating a development corpus. For example, a context head appears once per output word, so its confidence is averaged over all output words.

All heads are assigned a confidence score $c \in [0,1]$ by which they are sorted. The attention heads with the lowest confidence are good candidates to be pruned.

3.4.3 Setup: a base transformer (English$\rightarrow$German)

I reuse the same setup as described in Sect. 3.1.

3.4.4 Setup: a big transformer (Turkish$\rightarrow$English)

Data

I use all the parallel data allowed by the constrained condition of the WMT18 new task (Bojar et al., 2018). The corpora consist of $1M$ sentences in total: ~$200k$ parallel sentences plus an additional ~$800k$ sampled from News Crawl and back-translated using a shallow NMT model trained on the existing small bilingual corpora (Haddow et al., 2018).

The preprocessing follows the same steps that were previously set for English$\rightarrow$German:

1. Normalisation — the standardisation of punctuation (such as quotations styles across different languages), removing non-Unicode characters, removing extra spaces, et cetera.
2. Tokenisation — splitting words from punctuation.
3. Truecasing — lowercasing all words beside entities, names, et cetera.
4. BPE segmentation — splittings compound words to smaller subwords using a Byte Pair Encoding algorithm (Sennrich et al., 2016).

Normalisation, tokenisation, truecasing are done using Moses[4] scripts. BPE was trained jointly with 36000 operations. The vocabularies are shared for both languages in a pair and contain 36000 words.

All models are evaluated on untokenised WMT data using BLEU calculated with sacreBLEU (Post, 2018). I use the WMT16 devset for the development, with the final evaluation on the 2016, 2017 and 2018 testsets.

Model settings

For Turkish$\rightarrow$English, I experiment with big transformer models. The settings and hyperparameters are predefined by Vaswani et al. (2017a). Both the encoder and decoder have 6 stacked layers with 8 attention heads, rather than the standard 16 in a big transformer as defined by Vaswani et al. (2017a) motivated by no quality change in a big transformer with halved heads exhibited in Tab. 3.4. To keep experiments consistent, each head has a *fixed* size of 64 (Tab. 3.2). As shown later in Tab. 3.4, halving the number of heads does not affect the

4. `https://github.com/moses-smt/mosesdecoder`

quality but greatly reduces the pool of pruning candidates. Thus, all big and base models have 144 attention heads: 48 (6 layers with 8 heads each) self-attention heads in the encoder, 48 self-attention heads in the decoder, and 48 context heads in the decoder that attend to the encoder.

The gradient descent optimizer is Adam (Kingma & Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\varepsilon = 10^{-9}$. The learning rate is 0.0006, warming up for the first 8000 updates. Each model is exponentially smoothed with 0.0001 and optimised for cross-entropy averaged over words in a batch. A model stops training after 10 consecutive stalled checkpoints.

All experiments are trained on 4 $\times$ NVIDIA P100 (16GB) GPUs. The batch size is dynamic, aiming to fill 13 GB of each GPU memory, with approximately 700 sentences per batch.

Type	Size
dim_{emb}	1024
dim_{FFN}	4096
dim_{head}	64
Heads per layer	8
Encoder layers	6
Decoder layers	6

Table 3.2: A modified big transformer architecture used in experiments.

3.4.5 Setup: A knowledge-distilled tiny transformer (English$\rightarrow$German)

Data

This optimised *tiny* transformer architecture follows the deployment directions set by our work in WNGT2020 Efficiency Shared Task Bogoychev et al. (2020). Tiny models were data-distilled from an ensemble of larger architectures under the teacher-student regime (Y. Kim & Rush, 2016a). For the teacher, I used the sentence-level English-German system from Microsoft's constrained submission to the WMT'19 News Translation Task (Junczys-Dowmunt, 2019). It is an ensemble of four deep transformer-big models (Vaswani et al., 2017a), each with 12 blocks of layers in encoder and decoder, model size of 1024, filter size of 4096, and 8 transformer heads.

The student models get trained on pairs of the source and teacher-translated target sentences generated from parallel English-German datasets and English News Crawl data available for WMT19 (Barrault et al., 2019). For parallel data, we generated 8-best lists and selected translations with the highest sentence-level BLEU to reference sentences. Monolingual data was translated with a beam size of 4.

The final training set, which was used in Edinburgh's submission to WNGT2020 (Bogoychev et al., 2020), consisted of 185M sentences, including 20M of initial parallel data. In my experiments with knowledge distillation, I use a smaller subset consisting of 13.5M parallel sentences.

Model settings

All tiny students have standard transformer encoders (Vaswani et al., 2017a) and light-weight RNN-based decoders with SSRU (Y. J. Kim et al., 2019) with the dimensions presented in Tab. 3.3. The vocabulary is shared with 32,000 subword units preprocessed with SentencePiece (Kudo & Richardson, 2018).

Type	Size
dim_{emb}	256
dim_{FFN}	1536
dim_{head}	32
Heads per layer	8
Encoder layers	6
Decoder layers	2 (tied)

Table 3.3: The tiny transformer architecture used in WNGT2020 experiments.

Since a student model should closely mimic the teacher, I did not use regularisation techniques like dropout or label smoothing. The models were trained using the concatenated English-German WMT testsets from 2016–2018 as a development set[5] until BLEU has stopped improving for 20 consecutive validations. Then, the checkpoints with the highest BLEU scores were selected.

Other training hyperparameters were Marian defaults for training a base transformer model.[6] Student models have sharp probability distributions (as they overfit teacher-translated data), so it is possible to use beam size 1 without a quality loss. A baseline student model translates about 2335 words per second on a single CPU core, thanks to all the optimisation settings.

3.4.6 Related work

Coefficient-wise pruning is not hardware friendly. On the other hand, block-wise sparsity (Narang, Undersander, & Diamos, 2017) can practically skip loading pieces of a tensor into GPU's memory. In this paper, we concentrate on a specific case of block sparsity that removes entire attention heads from a model (no masking)

Xiao et al. (2019) reused attention output within adjacent layers in a model to save on computations. This reuse of parameters could be interpreted as a pruning method that concentrates on removing vertical redundancy, in contrast to our research, which is more horizontal.

5. The validation sentences are not teacher-translated.
6. Available via `-task transformer-base`.

SSRU (Y. J. Kim et al., 2019) replaces the decoder self-attention mechanism but leaves an encoder and context between them unchanged. The lottery ticket pruning of attention heads is complementary and can remove encoder and context heads on top, further reducing computational load.

The main focus of Voita et al. (2019) was an analysis of attention and its behaviour, rather than pruning and efficiency. As shown in Sect. 3.4.7, I could not reproduce the positive results on my dataset and got major damage to translation quality.

3.4.7 Establishing baselines for attention pruning

Just fewer attention heads

Do we even need to prune attention heads at all? Can we train a model that has fewer heads from the beginning? Are more heads better for quality? The typical transformer implementation described by Vaswani et al. (2017a) initialises attention matrices based on the embedding dimension, and those matrices are split into separate heads. That means the fewer heads there are set to be in a model, the larger they are. To compare models with a different number of heads reasonably, I fixed their size to a constant instead.

I experimented with the following model to explore how the number of attention heads affects quality. I used all the parallel data allowed by the constrained condition of the WMT17 news task (Bojar et al., 2017) for English→German ($4.56M$ sentences) following a standard preprocessing: normalisation, tokenisation, truecasing using Moses scripts [7], and BPE segmentation (Sennrich et al., 2016) with 36000 subwords. I tried training a model with 32 heads but could not due to memory constraints. For that reason, we start with a typical big transformer (Vaswani et al., 2017a) architecture using recommended hyperparameters. It has 16 heads of size 64 ($64 \times 16 = 1024$). Then, I trained the same model but with 8, 4 and 2 heads of the same size. The results are below in Table 3.4.

Model	Heads	wmt14	wmt15	wmt16	Avg.
Big transformer	16	26.7	29.8	33.9	30.1
Big transformer-8	8	27.2	29.7	34.2	30.4
Big transformer-4	4	26.1	29.0	34.2	29.8
Big transformer-2	2	26.0	29.0	33.6	29.5

Table 3.4: The quality of big transformer models with different number of attention heads for English→German.

7. `https://github.com/marian-nmt/moses-scripts`

The model needs a reasonable number of attention heads to perform well when it comes to quality. The more this number gets reduced, the worse the quality. However, more heads do not necessarily equal better translation quality. Using 8 heads per layer strikes a perfect balance between memory consumption and quality degradation. This quality of models with reduced attention heads is damaged even more the smaller the overall architecture is.

Michel et al. (2019) pruning

Michel et al. (2019) experimented with pruning attention heads during and after training using a different heuristic: they introduced a mask variable for each head then defined *importance* as the gradient of the loss with respect to the mask variable. Their reported results are poor: pruning 40% of the total heads results in "staying within 85–90% of the original BLEU score". Pruning after training is even worse: about 3 BLEU points were lost with 40% sparsity, and 10 BLEU points were lost with 60% sparsity. Comparisons are based on their reported numbers, which use non-standard tokenized BLEU that is known to boost scores falsely (Post, 2018).

Voita et al. (2019) pruning

Using the same language pair and dataset, I tried a pruning method presented by Voita et al. (2019). I used their Tensorflow implementation[8] with their training scripts, in which they set up a base transformer architecture that gets pruned globally.

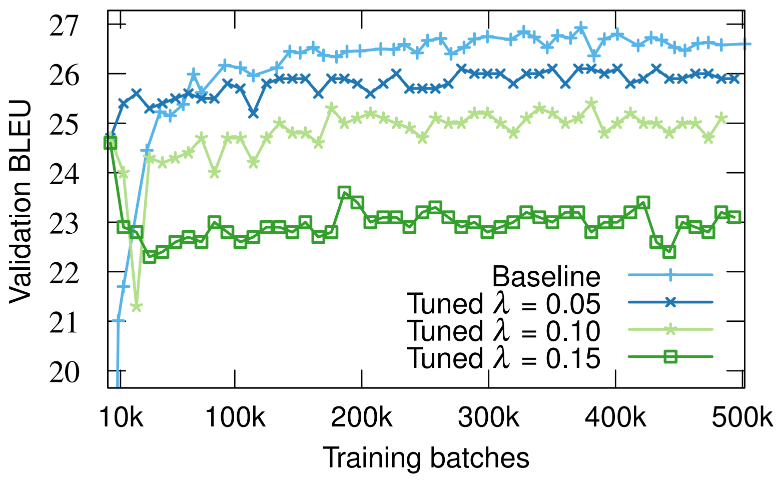


Figure 3.6: Validation BLEU for English→German transformer-base baseline and pruned with Voita et al. (2019) models.

This pruning scheme requires a baseline model to be fully converged first and then tuned with a L_0 regulariser that masks the heads. A λ hyperparameter controls the attention sparsity.

8. `https://github.com/lena-voita/the-story-of-heads`

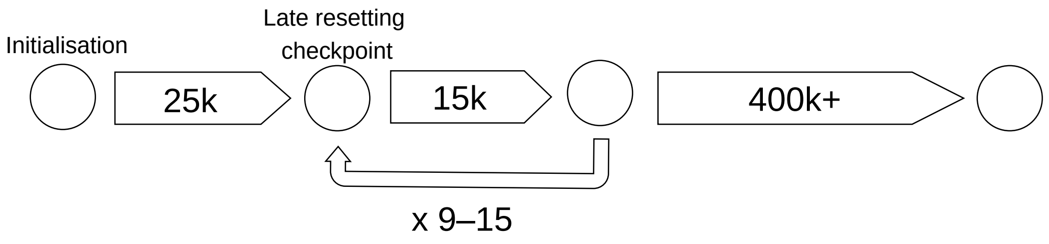
Model	Sparsity	wmt14	wmt15	wmt16	Avg.	Δ
Baseline	0%	26.7	29.8	34.5	30.3	-
$\lambda = 0.05$	22%	26.4	29.7	34.0	30.0	-0.30
$\lambda = 0.10$	53%	25.1	27.9	31.8	28.3	-2.00
$\lambda = 0.15$	67%	23.5	25.8	28.8	26.0	-4.30

Table 3.5: Evaluation BLEU of English→German transformer-base models pruned with Voita et al. (2019)

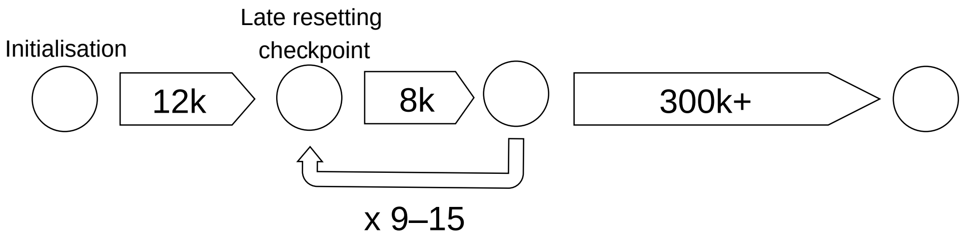
Even though I used the authors' implementation and the baseline achieved a good score, pruning degraded its quality. Looking at Figure 3.6, the more sparsity was enforced with regularisation, the lower the translation quality. Removing about half of the attention heads lost about 2 BLEU points, and removing two-thirds damaged the quality by more than 4 BLEU points. Even though we tuned for as long as the baseline training, the models did not recover. I tried experimenting with various hyperparameters settings, such as learning rate and its scheduling, but with no further success. Finally, I could not reproduce the positive results presented by Voita et al. (2019) and focused on improving upon that work.

3.4.8 Experiments: Foreword

These experiments aim to prune as many attention heads from a transformer as possible without damaging translation quality.



(a) English→German



(b) Turkish→English

Figure 3.7: Visualisation of the stabilised lottery ticket pruning with the hyperparameters selected for the experiments.

The pruning procedure has some hyperparameters: the late resetting point, how long to train before making a pruning decision and how many heads to prune each iteration. Exploring this space is expensive; I arbitrarily set these to the following values and found them working well within my experiments. I pretrain models for about 5–6 saving checkpoints (25k batches for English$\rightarrow$German, 12k for Turkish$\rightarrow$English). By this time, the translation quality stabilises quickly by jumping from 1-2 BLEU points towards 15-30 BLEU. Then, each pruning iteration lasts about 3–4 checkpoints (15k batches for English$\rightarrow$German, 8k for Turkish$\rightarrow$English), after which selected attention heads are removed. After each iteration, I change a seed value to make a model see data in a different order when I restart training.

The number of heads removed is roughly half of a total number of layers containing attention. Removing less than that makes pruning slow. Removing more in one go results in a unified distribution of attention heads, as the algorithm usually picks one head per layer, which may be too aggressive. There are 18 attention layers in this specific case: 6 layers for encoder, decoder, and context. For both language pairs, I prune **8 heads per iteration** (assuming the number of attention layers being 18 divided by two is relatively close to 8).

I focus on results roughly within 50% to 83% heads removed. This range covers the interesting part from minor to noticeable degradation in translation quality. For evaluation of an iteration, heads are pruned as usual; then, I reset the model back to the late resetting checkpoint to continue training until convergence.

3.4.9 Experiments: Pruning big and base transformer models

Since I have shown that there is no need for having 16 heads per layer in a big transformer architecture (see Tab. 3.4), I halve attention matrices to start pruning from 8 heads per layer to save time. Thus, all big and base models have 144 attention heads in total. The models are pruned with the workflow presented in Fig. 3.7a and 3.7b.

After every pruning iteration, the models are immediately evaluated on their development sets, as illustrated in Fig. 3.8. With each pruning step, the models slowly deteriorate in quality to finally take considerable damage at roughly 70% of all heads removed (iter. 11–12). Having that in mind, even if a model reaches low BLEU after the limited number of updates used to make pruning decisions, the model, in most cases, recovers from most of that damage when finally allowed to converge. The convergence progression of each pruned checkpoint for both language pairs is presented in Fig. 3.9.

The baselines reach the top BLEU scores quicker during training, but many pruned models still achieve competitive results later in training as they recover from pruning damage. The dashed vertical lines in Fig. 3.9 show the late resetting checkpoints. Pruning from half up to two-thirds of the attention heads leads to longer convergence times, but similar BLEU results on the development set. As each pruning iteration removes additional 8 heads, the

		Heads			BLEU				
Model	Sparsity	Encoder	Context	Decoder	WMT14	WMT15	WMT16	Avg.	Δ
Baseline	0%	8 8 8 8 8 8	8 8 8 8 8 8	8 8 8 8 8 8	27.2	29.7	34.2	30.4	—
Pruned i=9	50%	7 1 3 4 7 8	0 1 2 3 6 6	8 5 5 1 1 4	26.7	29.9	33.8	30.1	-0.3
Pruned i=10	56%	6 1 2 4 6 8	0 1 2 3 6 6	8 4 4 0 0 3	27.0	29.7	34.2	30.3	-0.1
Pruned i=11	61%	5 1 2 4 5 7	0 1 2 3 5 5	8 3 3 0 0 2	26.7	29.7	34.3	30.2	-0.2
Pruned i=12	67%	4 1 2 4 4 6	0 1 2 3 4 4	8 2 2 0 0 1	27.0	29.4	33.7	30.0	-0.4
Pruned i=13	72%	3 1 2 3 4 6	0 1 1 3 3 3	8 1 1 0 0 0	27.0	29.6	33.9	30.2	-0.2
Pruned i=14	78%	2 1 2 2 4 5	0 1 0 2 3 2	7 1 0 0 0 0	27.0	29.5	33.8	30.1	-0.3
Pruned i=15	83%	1 1 1 1 3 5	0 0 0 2 3 1	6 0 0 0 0 0	26.4	29.1	32.8	29.4	-1.0

(a) English→German (a base transformer)

		Heads			BLEU				
Model	Sparsity	Encoder	Context	Decoder	WMT16	WMT17	WMT18	Avg.	Δ
Baseline	0%	8 8 8 8 8 8	8 8 8 8 8 8	8 8 8 8 8 8	22.7	21.9	23.1	22.6	—
Pruned i=9	50%	7 2 4 6 7 8	0 1 2 3 6 7	8 3 2 4 1 1	22.7	22.4	23.5	22.9	0.3
Pruned i=10	56%	7 1 3 6 7 8	0 1 2 3 5 6	8 2 2 3 0 0	23.1	22.2	23.5	22.9	0.3
Pruned i=11	61%	6 1 3 5 7 8	0 1 2 2 4 5	8 1 1 2 0 0	22.9	22.0	23.4	22.8	0.2
Pruned i=12	67%	5 1 3 5 6 7	0 1 2 2 3 4	7 1 0 1 0 0	22.5	22.1	23.5	22.7	0.1
Pruned i=13	72%	4 1 2 5 5 6	0 1 2 2 2 3	6 1 0 0 0 0	22.8	21.6	23.1	22.5	-0.1
Pruned i=14	78%	3 1 2 5 4 5	0 0 2 2 1 2	5 0 0 0 0 0	22.0	21.3	22.6	22.0	-0.6
Pruned i=15	83%	2 1 1 4 4 4	0 0 1 2 0 1	4 0 0 0 0 0	21.9	21.2	22.6	21.9	-0.7

(b) Turkish→English (a big transformer)

			Heads	BLEU					
Model	Params	Sparsity	Encoder	WMT16	WMT17	WMT18	WMT19	Avg.	Δ
Baseline	15,7M	0%	8 8 8 8 8 8	36.4	29.3	43.2	39.9	37.2	—
Pruned i=9	14,8M	56%	5 1 3 1 5 6	36.5	29.1	43.4	40.0	37.3	0.1
Pruned i=10	14,7M	63%	4 1 2 1 4 6	36.5	29.0	43.5	39.7	37.2	0.0
Pruned i=11	14,6M	69%	3 0 2 1 4 5	36.3	29.0	43.3	39.9	37.1	-0.1
Pruned i=12	14,5M	75%	2 0 2 1 3 4	36.3	28.8	43.1	39.8	37.0	-0.2
Pruned i=13	14,4M	81%	1 0 2 1 2 3	36.3	28.9	42.7	39.5	36.9	-0.3
Pruned i=14	14,3M	88%	0 0 1 1 1 3	35.7	28.4	42.1	38.9	36.3	-0.9

(c) English→German (a knowledge-distilled tiny transformer)

Table 3.6: Evaluation of various transformer models converged at i^{th} pruning iteration with the following distributions of attention heads.

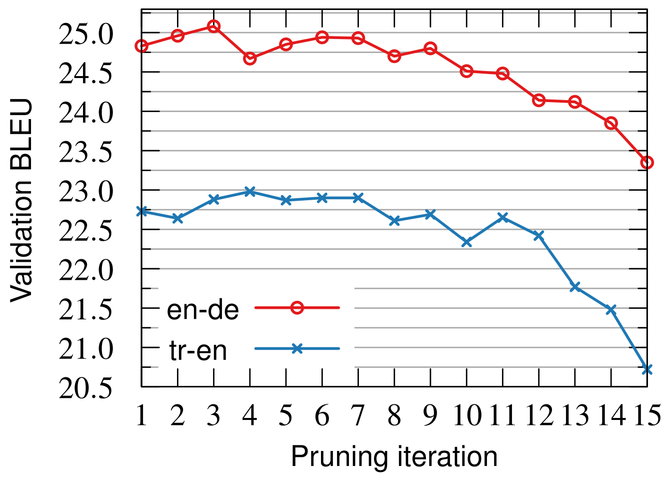
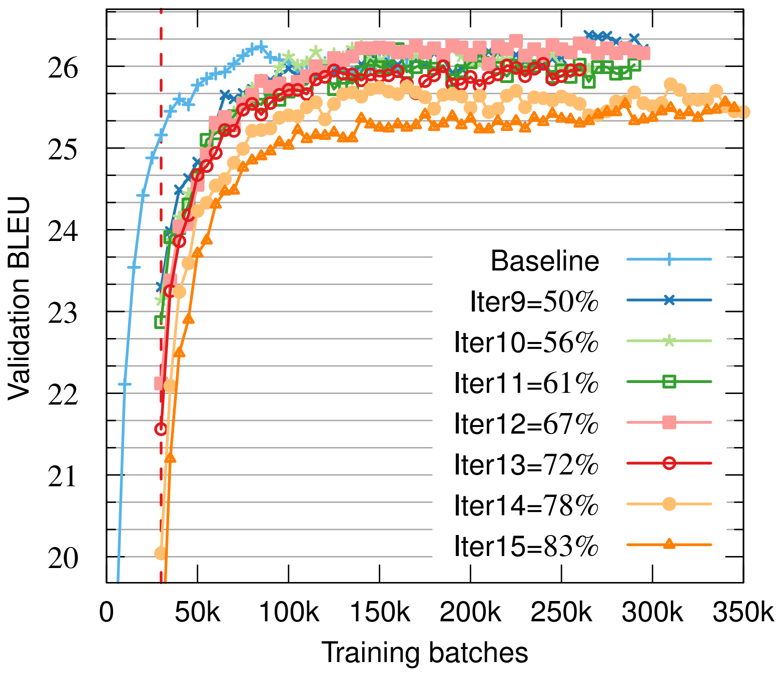


Figure 3.8: Validation BLEU after each pruning iteration for English→German and Turkish→English.

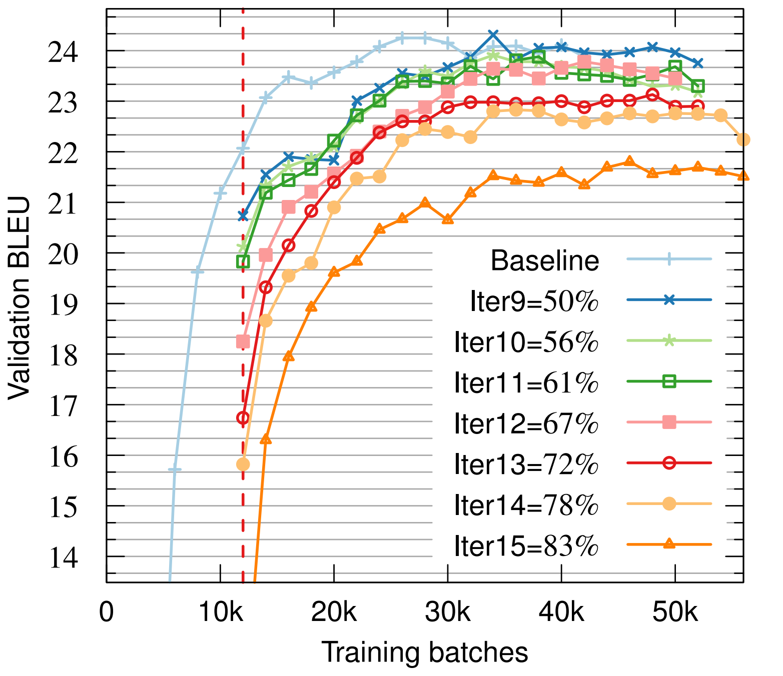
final convergence of those models ends up with progressively worse quality. Using different pruning hyperparameters, such as longer pretraining or pruning loops, could improve training convergence. There is a breaking point of considerable damage at about $78–83\%$ heads removed, especially for Turkish→English. At 83% sparsity, there are about 8 heads on the average left across all layers per encoder, decoder and context. Doing more pruning at this stage would mean obliterating most if not all attention heads from at least one of those, making a model incapable of performing translating tasks.

In Tab. 3.6a and 3.6b, I perform the final evaluation on “unseen” testsets and calculate the average difference in BLEU between the unpruned baselines and pruned models. I also show the total attention sparsity with the distribution of heads in each layer. In terms of quality, even the harshest pruning results in about 1 BLEU point damage. However, most customers may prioritise quality over sparsity and would not tolerate such quality degradation. They may agree to compromise on quality if the BLEU drop is relatively small $(0.1–0.3)$. I keep that in mind when analysing the results.

As evident in Tab. 3.6a and 3.6b, pruning $72–78\%$ of all attention heads mostly maintains the quality set by the baselines to then sharply degrade beyond that point. It indicates that there exists a minimum set of attention heads needed to perform at a given quality level.



(a) English→German



(b) Turkish→English

Figure 3.9: Convergence of big transformer models after removing a given percentage of all attention heads.

3.4.10 Experiments: Pruning knowledge-distilled tiny transformer models

I repeat the English$\rightarrow$German experiments, but this time in a highly-optimised teacher-student training setting. The decoder is already reduced to two tied layers in this knowledge-distilled architecture. In practice, it makes the decoder a single layer of parameters. The single context layer becomes a bottleneck since decoder self-attention gets replaced with an SSRU. I decided not to prioritise context in the pruning algorithm and focus on pruning the encoder only. I follow the workflow set in Fig. 3.7a with pretraining the model for 25k batches and pruning iterations lasting 15k updates. I remove 3 heads from the encoder in each pruning step across 14 iterations. The final evaluation results are presented in Tab. 3.6c.

The pruning of student models follows the trend set by the big transformer experiments: 75–81% of encoder heads can be removed with slight (0.2–0.3 BLEU) damage to the quality. Pruning more than that, like in the previous experiments, damages the model more aggressively by 0.9 BLEU point and can be treated as a trade-off between sparsity and quality with the focus on sparsity instead.

3.4.11 Quality analysis: Is pruning worth it?

Overall, the lottery ticket approach successfully prunes entire attention heads in both a large transformer model and a tiny student architecture based on a simple heuristic; most heads can be removed from a model with negligible damage to the translation quality.

Structural pruning can be interpreted as a form of architecture searching. To confirm whether the lottery ticket hypothesis remains valid or if the architecture of a winning ticket is the key aspect, I run the following experiments. Each English$\rightarrow$German pruned model I trained so far is *reinitialised*, so that the identical architecture and the number of parameters are kept. Then, I re-train these models from scratch to see if there is no need to bother with pruning at all, as we could use found sparsity patterns. The results are presented in Tab. 3.7 with big transformer experiments in Tab. 3.7a and knowledge-distilled tiny students in Tab. 3.7b. The pruned models in red columns accompany the same models trained from scratch (*Reinit*). Among the notable examples is the big model with 78% of all attention heads removed, losing about 0.3 BLEU with the reinitialised model losing 1.0 point. Similarly, the knowledge-distilled student is damaged by about 0.2 BLEU at 75% attention sparsity in the encoder, while the same reinitialised baseline loses 0.6 BLEU point. The quality gap gets smaller with the most aggressive pruning, again implying a minimum set of attention heads is required for a model to perform well without severe damage to quality.

	Base 0%	Iter 9 50%	Reinit	Iter 10 56%	Reinit	Iter 11 61%	Reinit	Iter 12 67%	Reinit	Iter 13 72%	Reinit	Iter 14 78%	Reinit	Iter 15 83%	Reinit
WMT14	27.2	26.7	26.8	27.0	26.5	26.7	26.5	27.0	26.5	27.0	26.6	27.0	26.0	26.4	25.9
WMT15	29.7	29.9	29.4	29.7	29.2	29.7	29.2	29.4	29.2	29.6	29.0	29.5	28.9	29.1	28.7
WMT16	34.2	33.8	34.2	34.2	34.1	34.3	33.8	33.7	33.7	33.9	33.9	33.8	33.2	32.8	32.6
Avg.	30.4	30.1	30.1	30.3	29.9	30.2	29.8	30.0	29.8	30.2	29.8	30.1	29.4	29.4	29.1
Δ	—	-0.2	-0.2	-0.1	-0.4	-0.2	-0.6	-0.4	-0.6	-0.2	-0.6	-0.3	-1.0	-1.0	-1.3

(a) English→German (a big transformer)

	Base 0%	Iter 9 56%	Reinit	Iter 10 63%	Reinit	Iter 11 69%	Reinit	Iter 12 75%	Reinit	Iter 13 81%	Reinit	Iter 14 88%	Reinit
WMT16	36.4	36.5	36.5	36.5	36.4	36.3	36.1	36.3	36.2	36.3	35.9	35.7	35.5
WMT17	29.3	29.1	28.9	29.0	28.9	29.0	28.9	28.8	28.5	28.9	28.5	28.4	28.2
WMT18	43.2	43.4	43.1	43.5	43.0	43.3	42.7	43.1	42.4	42.7	42.3	42.1	41.7
WMT19	39.9	40.0	40.0	39.7	39.6	39.9	39.4	39.8	39.4	39.5	39.5	38.9	38.9
Avg.	37.2	37.3	37.1	37.2	37.0	37.1	36.8	37.0	36.6	36.9	36.6	36.3	36.1
Δ	—	0.0	-0.1	0.0	-0.2	-0.1	-0.4	-0.2	-0.6	-0.5	-0.6	-0.9	-1.1

(b) English→German (a knowledge-distilled tiny transformer)

Table 3.7: The comparison of translation quality between pruned models (grey columns) and those having the same pruned architecture but with reinitialised parameters and trained from scratch. Lottery ticket pruning ensures better quality due to careful parameter selection, which is nullified when reinitialised.

To summarise, the pruned models outperform the identical standalone architectures across a variety of model sizes (big vs base vs tiny) and training methods (normal vs knowledge-distillation). The results demonstrate that the advantage of lottery ticket pruning comes from random initialisation, not the architecture itself. The careful selection of parameters with a rewind mechanism in place leads to better translation quality in NMT.

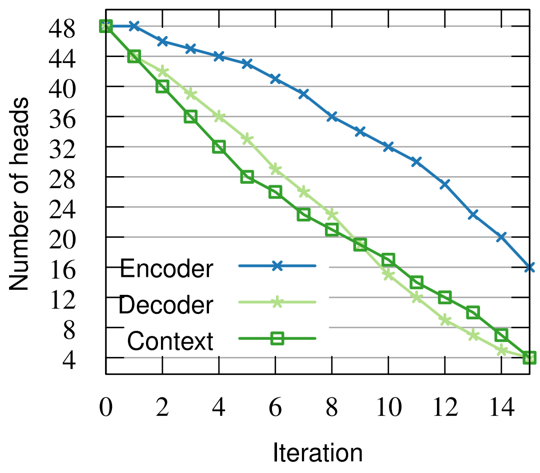


Figure 3.10: The total number of remaining attention heads per *encoder*, *decoder* and *context* in a big transformer (English→German). The decoder and context attention heads are pruned rather quickly and harshly in comparison to those in the encoder.

3.4.12 Sparsity analysis: Attention distribution

In this section, I take a closer look at the distribution of attention heads that remain after pruning. Tab. 3.6 presents the number of heads left in each layer after every pruning iteration, which is further visualised in Fig. 3.10, 3.11 and 3.12.

First, the heuristic based on attention confidence favours keeping encoder heads over those in the decoder and context. The plot in Fig. 3.10, which visualises the total number of heads left in a model, shows that the encoder is the least pruned part of a model. If we follow the logic of “confident heads equals important heads”, then the encoder seems to hold more of the important heads overall.

Self-attention heads in the decoder seem especially useless. As shown in Tab. 3.6a and its visualisation in Fig. 3.11, all self-attention layers in the decoder except the first one get aggressively pruned, aiming to remove them from a model completely. This finding is quite interesting in itself as this seems to support the claim of other researchers that deem decoder self-attention redundant (Xiao et al., 2019) or approximable (Y. J. Kim et al., 2019). Positive pruning results further affirm this. For example, one of the Turkish→English models (Tab. 3.6b) removed more than 85% of decoder and context heads at the cost of only 0.1 BLEU point. The remaining heads are concentrated within a few layers, reducing the overall depth of a model.

There are two (and possibly more) interpretations of this. Foremost, a model may not need such a deep decoder, as endorsed by the general popularity of reduced decoders (1–2 layers). On the other hand, the gap left by redundant attention after pruning is probably minimised by feedforward layers as they take over the network workload and address potential calculative limitations.

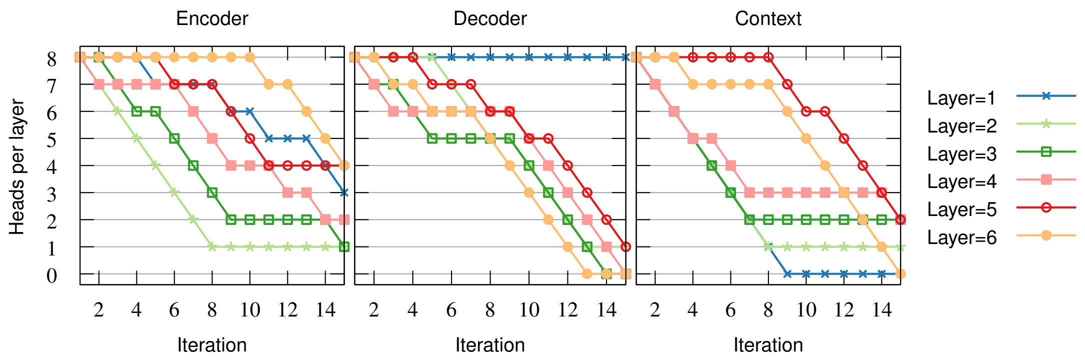


Figure 3.11: Attention distribution in each layer per pruning iterations for English→German from Tab. 3.6a.

Most interestingly, different language pairs also exhibit similar patterns in attention distribution. In Fig. 3.12, I present the histogram of attention heads in big transformer models with 67% heads removed for English→German and Turkish→English. Both experiments share a comparable structure. The encoder is valley-shaped, with the second layer being its bottom. The self-attention in the decoder keeps the first layer intact and (almost) obliterates the rest. The reverse is true for the context attention: the first layer has been removed, and further layers participate progressively more. This pattern reveals that a model prefers to first attend to its translation and then confront it with a source context later.

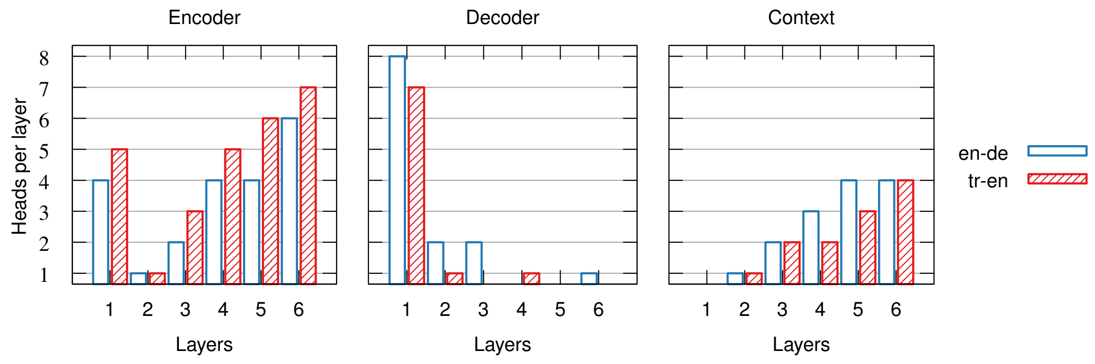


Figure 3.12: Attention distribution in the pruned models for English→German and Turkish→English. The pruned models for both language pairs follow similar patterns.

Model	Pretrain	Pruning	Convergence	Total
Baseline	-	-	475k	475k
Pruned 81%	25k	15k $\times$ 13 iterations	400k	620k

Table 3.8: The number of training updates in the baseline and the pruned English$\rightarrow$German student model.

Since this appears to be the case for two diverse training data sets, a transformer architecture may require less attention in general, with a varying number of heads up to the task in each layer. If initialised luckily, it could be potentially trained from scratch with good quality results. The lottery ticket hypothesis only uncovers the winning ticket that may be, to some degree, universal across different language pairs.

However, it is important to remember that this distribution pattern is the direct outcome of using attention *confidence* as a threshold function. It is entirely possible that using an entirely different heuristic in the lottery ticket scheme may produce other architecture variants.

3.4.13 Speed analysis: Foreword

The main objective of this research is to remove heads from a transformer to make inference faster. Due to resetting the parameters in the lottery ticket hypothesis, a model requires a longer time to train than when trained from scratch. Because of that, I make a trade-off between a total training time and inference speed, which is particularly useful in an industrial production environment, where we may afford to train slightly longer but have a faster model to deploy. In Table 3.8, I show an example comparison of how long it takes to prune and train a model in contrast to the baseline. In practice, if a model trains for 2–3 days, an additional day may be needed for a pruning procedure on top of it.

Given two objectives that we have (inference speed and quality), there is no single *best* model. Instead, we look at *the best possible speed at given quality*. Depending on the usage, we may be more interested in the best translation quality possible, but we have to compromise on speed. If we do not care about quality, we may prioritise speed instead. Despite attention heads being just a small fraction of all parameters ($\sim 5\%$ fewer parameters with about 10% size reduction), pruning them lessens the burden on inference significantly, especially for models with a deeper decoder. In the knowledge-distillation settings, Junczys-Dowmunt, Heafield, et al. (2018) achieved 8.57$\times$ speed-up with -0.8 BLEU loss on GPU when scaling down from a big transformer teacher to a base transformer student. In another experiment, they gained 1.31$\times$ speed-up with -0.6 BLEU when using 8-bit quantisation on the CPU. My pruning method complements those as lottery ticket pruning can always remove heads on top of existing solutions. For this reason, it is important to explore the benefit of pruning in different environments. The following sections evaluate and analyse experiments regarding speed for various architecture sizes: a typical base transformer architecture and a highly-optimised model based on the efficiency shared task.

3.4.14 Speed analysis: A non-optimised transformer

Let us start by evaluating a larger base transformer with English$\rightarrow$German. Tab. 3.9 presents the results of the speed evaluation. A decoder is the most expensive part of a transformer architecture, so aggressively pruning its attention leads to a significant speed-up. As seen in Tab. 3.9, removing more than half of all attention heads leads to $1.6\times$ faster translation with only 0.1 change in BLEU. Pruning about three-quarters results in more than twice faster translation speed at a small (0.2–0.3) BLEU cost. This model reaches about 2000 words per second in inference speed. This shows that the lottery ticket pruning could reduce the cost of using the largest and most expensive models, prioritising quality over the speed at the end of the day.

	Architecture		Average BLEU		Inference speed		
Model	Att. sparsity	Model size	WMT14–16	Δ	Avg. time	Speed-up	WPS
Baseline	0%	241MB	30.4	-	75.7	1.00	897
Pruned i=9	50%	201MB	30.1	-0.3	52.1	1.45	1304
Pruned i=10	56%	197MB	30.3	-0.1	45.9	1.65	1479
Pruned i=11	61%	193MB	30.2	-0.2	42.7	1.77	1593
Pruned i=12	67%	189MB	30.0	-0.4	39.0	1.94	1739
Pruned i=13	72%	185MB	30.2	-0.2	35.9	2.11	1892
Pruned i=14	78%	181MB	30.1	-0.3	33.8	2.24	2011
Pruned i=15	83%	177MB	29.4	-1.0	32.0	2.37	2124

Table 3.9: Inference speed analysis of base transformer models for English$\rightarrow$German after removing a given percentage of *all attention heads*. WPS is words per second. All models run on 16 CPU cores.

3.4.15 Speed analysis: A highly-optimised transformer

Knowledge-distillation is the state-of-the-art approach to getting the fastest possible translation with a negligible quality compromise. The base and tiny experiments are not comparable due to training for different tasks/data sets. However, in terms of speed, the most pruned base model reaches about 2000–2100 words per second, while the tiny baseline translates about 2300 words per second (see Tab. 3.10). These numbers show that, depending on the task, data, and target deployment, it may be better to utilise a smaller, more robust architecture rather than bother with pruning to get similar speed-up effects.

A small student model translates about 10% faster when pruned at 75% of sparsity at the cost of 0.1 BLEU point. It is important to remember that the decoder is the key reason the transformer architecture is slow, despite being optimised with an SSRU. Thus, this type of model has a smaller margin of improvement. The hardware and low-level code optimisations limit further advancement. Again, in this case, attention pruning is complementary and pushes the state-of-the-art even further.

Just for comparison, I also include simple baseline models trained with half (4) and one (1) attention head in each layer (including context attention). The model with just one head everywhere is slightly faster than our pruned model but at the cost of 2 BLEU points. This loss clearly shows again that careful pruning gives much better results than just training a smaller model from the start.

	Architecture		BLEU		Inference speed		
	Att. sparsity	# params	WMT19	Δ	Time	Speed-up	WPS
Baseline	0%	15,7M	39.9	0.0	18.1	1.00	2283
Baseline.half	50%	14.8M	39.0	-0.9	18.0	1.01	2300
Baseline.one	88%	14.1M	37.8	-2.1	15.2	1.20	2729
Pruned i=9	56%	14,8M	40.0	0.1	16.9	1.07	2443
Pruned i=10	63%	14,7M	39.7	-0.2	16.7	1.08	2475
Pruned i=11	69%	14,6M	39.9	0.0	16.6	1.09	2489
Pruned i=12	75%	14,5M	39.8	-0.1	16.3	1.11	2542
Pruned i=13	81%	14,4M	39.5	-0.4	16.4	1.11	2527
Pruned i=14	88%	14,3M	38.9	-1.0	16.2	1.12	2558

Table 3.10: Inference speed analysis of knowledge-distilled tiny transformer models for English$\rightarrow$German after removing a given percentage of *encoder attention heads*. WPS is words per second. All models run on 1 CPU core.

3.4.16 Speed analysis: The Efficiency Shared Task (WNGT2020)

To compare my work with the state-of-the-art in machine translation speed, I submitted English$\rightarrow$German student models to the WNGT2020 efficiency shared task (Bogoychev et al., 2020) as a part of the Edinburgh team. These submissions were converged on a larger amount of data (185M sentences instead of 13.5M) for maximised quality. The speed got evaluated by translating 1M sentences on a single CPU core and a single GPU. The quality was evaluated on WMT1*, which is an average over WMT10–19 testsets, excluding WMT12. The results are presented in Tab. 3.11 and 3.12.

Since the pruning method usually selects one head to remove per layer, I experimented with more aggressive (*pushy*) and lenient (*steady*) pruning by removing 6 and 3 heads per iteration, respectively. The *steady* pruning results in a uniform distributions. My pruned submissions were on the Pareto frontier for speed and quality, meaning that no other submission was simultaneously faster and higher quality.

On CPU, the speed-up is about 10% with 75% encoder heads removed (Tab. 3.11). In terms of GPU, the best pruned model gains 15% speed-up w.r.t. words per second, losing 0.1 BLEU in comparison to an unpruned model (Tab. 3.12). These results show that the pruned models achieve comparable quality with faster translation even when tested on a larger scale and push the Pareto frontier forward.

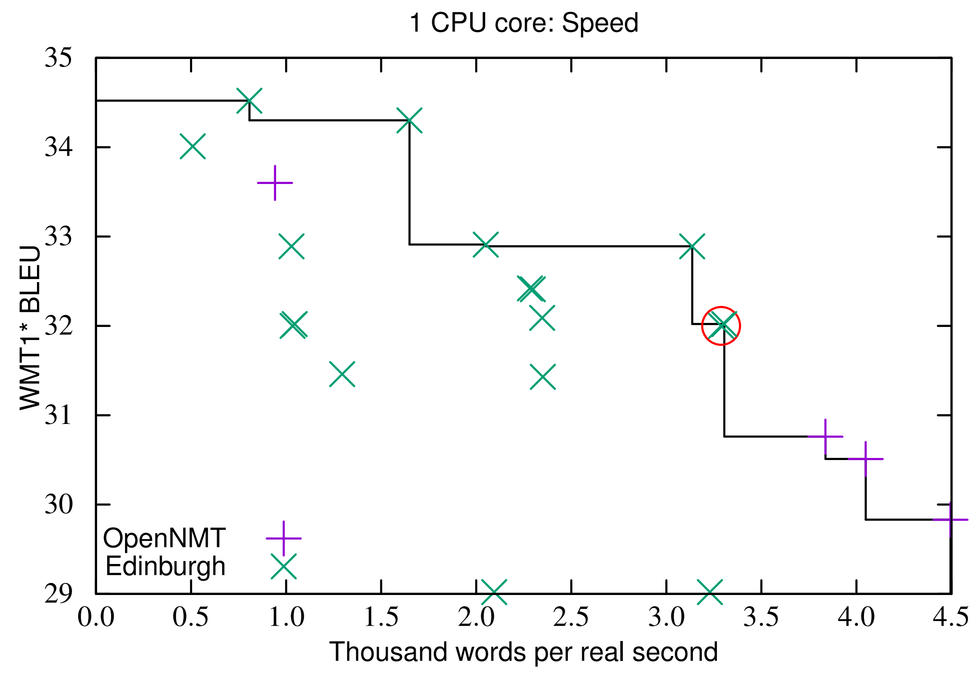
				BLEU		
Model	Enc. heads	Params.	Size	WMT19	WMT1*	WPS
Tiny	8 8 8 8 8 8	15.7M	61MB	41.5	32.9	2050
Tiny.Steady i=12	2 0 2 1 3 4	14.5M	56MB	41.4	32.4	2282
Tiny.Steady i=14	0 0 1 1 1 3	14.3M	55MB	40.2	31.4	2350
Tiny.Pushy i=6	2 2 2 2 2 2	14.5M	56MB	41.1	32.4	2298
Tiny.Pushy i=7	1 1 1 1 1 1	14.3M	55MB	40.8	32.1	2346

Table 3.11: Quality and inference speed of our WNGT2020 models with pruned attention on CPU. Words per second (WPS) is evaluated in *float32* with a single CPU core on the official WNGT2020 input of 1M sentences. WMT1* is an average over WMT10–19 testsets, excluding WMT12.

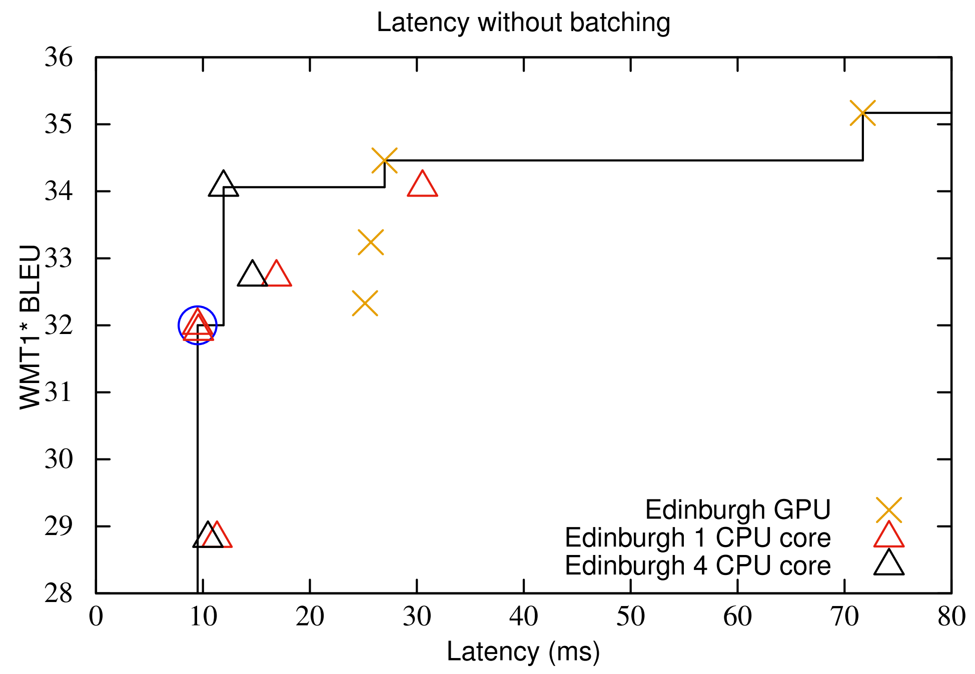
				BLEU		
Model	Enc. heads	Params.	Size	WMT19	WMT1*	WPS
Tiny	8 8 8 8 8 8	15.7M	61MB	41.5	32.9	8210
Tiny.Steady i=12	2 0 2 1 3 4	14.5M	56MB	41.4	32.4	9518
Tiny.Pushy i=6	2 2 2 2 2 2	14.5M	56MB	41.0	32.4	9508

Table 3.12: Quality and inference speed of our WNGT2020 models with pruned attention on GPU. Words per second (WPS) measured on an AWS *g4dn.xlarge* instance with one NVidia T4 GPU. WMT1* is an average over WMT10–19 testsets, excluding WMT12.

We chose two models for our CPU submissions: those with “202134” and “222222” distributions of attention heads (*Tiny.Steady i=12* and *Tiny.Pushy i=6*). They were evaluated batch-wise on 1 CPU core as well as in latency with a batch set to a single sentence. Due to a memory leak our team encountered during our submission period, the organisers allowed a few models to be re-evaluated down the line. The pruned models achieved about 32 BLEU points averaged over WMT1* testsets, which is slightly different to Tab. 3.11. The official Pareto analysis is presented in Fig. 3.13 for both single-core and latency tasks. I highlighted the pruned models on the plots to easily distinguish them from all other submissions. The models with the attention pruned through the structural lottery ticket approach are on the Pareto frontier in the tasks, proving that this method produces state-of-the-art robust architectures in quality and speed.



(a)



(b)

Figure 3.13: The official Pareto trade-offs for English→German models in WNGT2020 Efficiency Shared Task (Heafield et al., 2020). The pruned models are circled.

3.5 Exploring block-wise pruning outside of attention

So far, the lottery ticket approach successfully pruned entire attention heads based on a simple heuristic without damaging the quality. This section looks into a general use case of a block-wise lottery ticket. Attention pruning is a special case of it, in which quite large blocks (e.g. 256×32) can be entirely removed from a model without enforcing sparsity in matrices. Slicing is possible due to the nature of the attention mechanism and its calculations. The generalisation of block-wise lottery ticket pruning needs to be adapted appropriately.

3.5.1 Methodology

First and foremost, blocks in feedforward layers must be much smaller: their dimensions are usually from 8 to 32. In their work on block sparsity and lottery ticket hypothesis, Siswanto (2021) gauged that coarser block sizes lead to worse prediction accuracy in image recognition models trained on MNIST dataset.

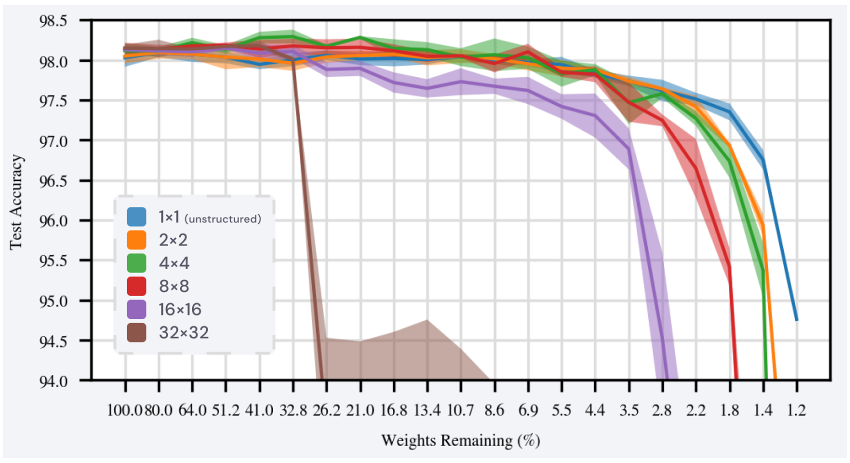


Figure 3.14: The effect of block sizes in block-sparse lottery ticket pruning performed on a LeNet architecture on MNIST image recognition benchmark (taken from (Siswanto, 2021)). Block sparse networks perform best quality-wise when sparsified more granularly.

Larger blocks are hardware friendly as matrix multiplication algorithms may require less overhead, but, in turn, they are less flexible in sparsity patterns, especially in smaller models. Next, not all layers may benefit from block-wise sparsification. For example, each row in an embedding layer corresponds to a word in a vocabulary. Applying square block sparsity patterns over the embedding matrix would insinuate some relations between neighbouring words while they are not related or sorted. The embedding matrices are frequently sliced with words selected in shortlisting. It would be difficult to take advantage of structural sparsity in such case without impacting multiplication overhead negatively.

			BLEU					
Model	Block size	FFN Sparsity	WMT16	WMT17	WMT18	WMT19	Avg.	Δ
Baseline	-	0%	36.4	29.3	43.2	39.9	37.2	-
+ pruned attention	-	0%	36.3	28.8	43.1	39.8	37.0	-0.2
+ pruned FFN	8	50%	35.7	28.2	41.5	38.6	36.0	-1.2
+ pruned FFN	16	50%	35.8	28.3	41.9	38.7	36.2	-1.0
+ pruned FFN	32	50%	35.4	28.1	41.7	39.0	36.1	-1.2
+ pruned FFN	8	75%	34.6	27.2	40.5	37.6	35.0	-2.2
+ pruned FFN	16	75%	34.6	27.2	40.6	37.6	35.0	-2.2
+ pruned FFN	32	75%	34.5	27.1	40.5	37.4	34.9	-2.3

Table 3.13: The results of simple feedforward pruning on a top of a model with attention heads pruned with the lottery ticket. Each layer has been uniformly pruned with 50%/75% of blocks removed. Pruning was performed in a single step on a late resetting checkpoint and then subsequently trained until convergence.

3.5.2 Setup

Feedforward layers are much simpler and, just like attention, significantly contribute to computations. As a follow-up for my work, my next goal is to build upon the attention sparse results I have gotten so far. For this reason, I chose an English→German knowledge-distilled model (see Sect. 3.4.5) with an already pruned encoder attention. I selected a checkpoint with 75% of encoder heads removed (see Tab. 3.6c, *Pruned i=12*) as it only lost -0.2 BLEU in quality.

3.5.3 Experiments: A one-off block-sparse pruning

In this experiment, I test how a model behaves when feedforward layers are pruned straightaway on a slightly pretrained and already attention-sparse model. I masked 50% and 75% of blocks *layerwise* in feedforward layers based on the largest parameter in each block. Tab. 3.13 provides the breakdown on the results for different block sizes: 8×8, 16×16 and 32×32. In the end, there is not much difference in translation quality between those models. The most probable explanation is that this pruning is simplistic and too aggressive. Still, since there is no major difference for this baseline between block sizes and Fig. 3.14 has shown that smaller blocks are better for quaility, I focus on experimenting with 8×8 blocks due to their flexibility from now on.

3.5.4 Experiments: A block-sparse feedforward pruning with a lottery ticket approach

Having defined a block size, I proceed with proper experiments on iterative lottery ticket pruning. In each pruning iteration, I mask an additional 10% of blocks with the lowest maximum magnitude. Then, I converge the checkpoints with induced 50–90% sparsity.

	Sparsity		BLEU					
Model	Attention	FFN	WMT16	WMT17	WMT18	WMT19	Avg.	Δ
Baseline	0%	0%	36.4	29.3	43.2	39.9	37.2	-
Pruned baseline	75%	0%	36.3	28.8	43.1	39.8	37.0	-0.2
Pruned iter=5	75%	50%	35.5	28.0	41.9	39.1	36.1	-1.1
Pruned iter=6	75%	60%	35.9	28.1	41.9	38.7	36.2	-1.0
Pruned iter=7	75%	70%	35.1	27.9	41.1	38.7	35.7	-1.5
Pruned iter=8	75%	80%	35.0	27.4	40.7	37.9	35.3	-1.9
Pruned iter=9	75%	90%	34.2	26.9	39.3	36.9	34.3	-2.9

(a) Uniform pruning with iter. step = 10k

	Sparsity		BLEU					
Model	Attention	FFN	WMT16	WMT17	WMT18	WMT19	Avg.	Δ
Baseline	0%	0%	36.4	29.3	43.2	39.9	37.2	-
Pruned baseline	75%	0%	36.3	28.8	43.1	39.8	37.0	-0.2
Pruned iter=5	75%	50%	35.7	28.6	42.7	39.1	36.5	-0.7
Pruned iter=6	75%	60%	36.0	28.4	42.3	39.1	36.5	-0.7
Pruned iter=7	75%	70%	35.6	28.0	41.6	38.9	36.0	-1.2
Pruned iter=8	75%	80%	35.1	27.6	41.1	38.5	35.6	-1.6
Pruned iter=9	75%	90%	34.6	27.4	40.1	37.4	34.9	-2.3

(b) Uniform pruning with iter. step = 25k

	Sparsity		BLEU					
Model	Attention	FFN	WMT16	WMT17	WMT18	WMT19	Avg.	Δ
Baseline	0%	0%	36.4	29.3	43.2	39.9	37.2	-
Pruned baseline	75%	0%	36.3	28.8	43.1	39.8	37.0	-0.2
Pruned iter=5	75%	50%	36.1	28.6	42.6	40.0	36.8	-0.4
Pruned iter=6	75%	60%	36.1	28.6	42.3	39.1	36.5	-0.7
Pruned iter=7	75%	70%	35.7	28.4	42.2	39.1	36.4	-0.8
Pruned iter=8	75%	80%	35.2	27.9	41.5	38.4	35.8	-1.4
Pruned iter=9	75%	90%	34.6	27.2	40.4	37.6	35.0	-2.2

(c) Uniform pruning with iter. step = 45k

	Sparsity		BLEU					
Model	Attention	FFN	WMT16	WMT17	WMT18	WMT19	Avg.	Δ
Baseline	0%	0%	36.4	29.3	43.2	39.9	37.2	-
Pruned baseline	75%	0%	36.3	28.8	43.1	39.8	37.0	-0.2
Pruned iter=5	75%	50%	36.0	28.5	42.6	39.2	36.6	-0.6
Pruned iter=6	75%	60%	35.5	28.4	42.4	38.9	36.3	-0.9
Pruned iter=7	75%	70%	35.6	28.1	41.7	38.7	36.0	-1.2
Pruned iter=8	75%	80%	35.1	27.7	41.2	38.2	35.6	-1.6
Pruned iter=9	75%	90%	34.5	27.2	40.5	37.7	35.0	-2.2

(d) Global pruning with iter. step = 25k

Table 3.14: Evaluation of knowledge-distilled transformer models converged at i$^{\text{th}}$ pruning iteration with the following sparsity of attention and feedforward layers. Feedforward layers were pruned *uniformly* (a, b, c) and *globally* (d) with an 8×8 block pattern at given sparsity level.

To further investigate how the length of pruning loops affects the final quality, I prune and reset after $10k$ (Tab. 3.14a), $25k$ (Tab. 3.14b) and $45k$ (Tab. 3.14c) batches. Additionally, I also compare *layerwise* and *global* pruning in $25k$ pruning loops (Tab. 3.14b and 3.14d). Alongside the attention-pruned baseline model that I use as a starting checkpoints in these experiments, I present the unpruned baseline as well to fairly compare the overall translation quality.

It is apparent that, within the constraints of this specific pruning heuristic, longer pruning steps in the iterative lottery ticket method lead to better translation quality when converged. For instance, the encoder with three-quarters of attention heads removed, and 70% of parameters blocks in feedforward layers masked damaged the model by 1.5 BLEU when trained with $10k$ pruning steps. Meanwhile, the damage gets reduced by almost half (-0.8) with $45k$ steps. Ideally, one would fully train a model for as long as possible, but time constraints and computational costs are a limit here. It is up to an engineer to decide whether they can afford longer training times; however, it seems a good idea to have a loop at least see a significant portion of data before making a pruning decision.

Next, I investigate a contrast between local and global pruning, as in uniform and irregular sparsity patterns and how they affect the quality. The layerwise results from Tab. 3.14b can be directly compared to Tab. 3.14d with global ones. The two tables do not show any meaningful difference in BLEU. It is important to remember that it is not a general conclusion but one drawn just on this specific pruning heuristic. It could be that another pruning function would favour a different outcome.

In terms of the overall translation quality of the pruned models, pruning half of the parameters in encoder feedforward layers on top of already pruned attention loses about 0.4 BLEU point when compared to a fully dense baseline. Being more aggressive than that results in 70% of feedforward sparsity at the cost of less than 1 BLEU point and 90% sparsity with slightly more than 2 BLEU in damage. These numbers show that it is possible to prune transformer layers extensively with a trade-off in translation quality. The question is: can this pruning method lead to a faster inference? Individual blocks in feedforward layers cannot be entirely removed without utilising a block-sparse memory representation and corresponding matrix multiplication routines. As will be shown in the next chapter (see Sect. 4.9), optimising sparse matrices is quite difficult. A matrix needs to be at least 70–80% sparse to get any more efficient in compression and memory consumption. Even if we ignore that aspect and focus on speed, we need at least 50% sparsity to get a similar speed level of a dense operator.

At that point, I investigated the possibility of incorporating sparse operators into a machine translation toolkit (Marian NMT), at least for CPU and found the support for those operators severely lacking. Without fully customised machine-learning focused operators, any modifications I would have to perform to force a toolkit to work around compatibility issues would diminish any potential speed gains due to required overhead operations. It would also require substantial amount of time of code development. There was no issue with attention pruning as a model can be sliced into a smaller but still dense architecture, leading to faster inference without specialised kernels. Given the problems above and many more, I shift my research focus to methods that would alleviate the issues by remaining dense and simplify/shorten a whole pruning process while potentially achieving better translation quality.

3.6 Conclusions

In this work, I investigated iterative pruning with rewinding to apply blockwise pruning to a transformer model. Specifically, I targeted the attention mechanism to shrink a model and make it faster. As experiments have shown, I achieved a real inference speed-up, which does not require any sparse matrix multiplication routines or low-level code optimisations. The lottery ticket algorithm removes a majority of attention heads in a model, making them faster with negligible or no change to translation quality. This pruning method complements other optimisation techniques such as knowledge distillation and quantisation, further pushing the Pareto frontier of the state-of-the-art approaches. Attention is one of the most expensive layers in a model, and the structural pruning applied to it proved to be effective in optimising them. It does not only removes individual heads but leads to pruning whole layers as well, which allows skipping most calculations altogether. The experiments on NMT have proved that it is possible to remove a significant percentage of all heads (50–72%) in a large transformer, which directly results in 1.6–$2.2\times$ faster translation. It has also successfully been applied in a competitive efficiency shared task setting, resulting in 10–15% faster translation at only 0.1 BLEU cost for an already highly-optimised architecture.

The natural progress of this research would be to prune other parts of the network — with the lottery ticket approach or not — to see how far block pruning can go without too much impact on quality. Besides attention, embedding and feedforward layers also contribute to a workload and may also benefit from pruning. I have performed experiments on blockwise pruning of feedforward layers, stacking on already attention-pruned architectures. The results, although quite promising, have highlighted the limitations of sparse calculations in the context of optimisation. Moreover, the lottery ticket approach is a rather tiresome method. Among its constraints is training time. The lottery ticket algorithm requires training and resetting parameters in a loop, making a model "stuck" until pruning finishes. Even though it is a simple method, the lottery ticket pruning takes too long to perform, especially for NMT models. The heuristic algorithms I chose can be improved upon, especially since there is still room for translation quality improvement and a better speed-BLEU trade-off. Ideally, a pruning method would require only one training pass to prune concurrently without damaging the quality. Long pruning time is one of the lottery ticket's weak points, and I plan to explore pruning methods outside of it in my next work.

Chapter 4

Structural Pruning of Transformer for Speed Using Group Lasso

The main disadvantage of the lottery ticket hypothesis is the enormous time it takes to train a model from start to finish. Empirically, it can make training progress twice as long for a large state-of-the-art model on 4 GPUs. Moreover, there are many hyperparameters, such as the lengths of pretraining and pruning phases, sparsity level for each iteration, et cetera. The grid search makes deployment even more costly.

There is no direct feedback to a neural network during the process in a typical pruning approach. Usually, parameters are chosen based on some heuristic (like their magnitude) and then masked for the rest of the training. Backpropagation has to abruptly accommodate their absence, even if a pruning process only removes a minuscule subset of parameters every so often. Moreover, the complexity of pruning and excessive training time continues to be a concern in the thesis. The previous study highlighted a necessity for simpler and better structural pruning methods in a transformer. The key research question of this chapter is whether or not a less time-consuming regularisation method can result in a Pareto-optimal inference speed for machine translation.

4.1 Motivation

In this chapter, I investigate structural pruning through *regularisation*. As introduced in Sect. 2.5, regularisation is a technique that reduces the complexity of a model to make it more flexible and prevent overfitting. Sparsity gets achieved by penalising a cost function with parameters themselves. Then, the training objective forces the parameters to go down alongside the penalty. Various regularisation terms have different characteristics. In particular, I examine the usage of *group lasso* to sparsify parameters in a transformer architecture *structurally*. The group lasso penalty offers an effective way to prune continuously during training without expensive lottery ticket loops that do not move the training progress forward. Structural sparsity such as block-wise has a potential for significant inference speed-up as it is more hardware friendly than coefficient-wise pruning, which I also plan to explore in this work.

4.2 Background: Group Lasso

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

$$R(f) = \lambda \sum_{g=1}^{|G|} \sqrt{\sum_{i=1}^{|G_g|} \left(w_i^g\right)^2} \tag{2.20 revisited}$$

λ controls the overall strength of the regularisation term. The larger it is, the more aggressive sparsity it enforces. The example use case of a group lasso and its impact on a cost function is visualised in Fig 2.9e and 2.10f.

The group lasso penalty is, just like any other regulariser, added to the cost function and further scaled by a batch size as well:

$$E(batch) = \frac{1}{|batch|} \left(\sum_{x \in batch} CE(x) + \sum_{l \in layers} R(l) \right) \tag{2.21 revisited}$$

4.3 Research outline

First, I continue my research into block sparsity by applying block-sparse group lasso onto feedforward layers in various configurations. These preliminary experiments serve as an exploratory ground in quality performance and clarify the methodology used in the following research. Next, I proceed with an independent analysis of sparse kernels and their efficiency, looking for potential optimisation opportunities. Taking block-sparse matrices extracted directly from a pruned model, I run an evaluation of sparse$\times$dense matrix multiplication operators using various matrix representations. This analysis highlights that, despite quite positive speed-up in multiplication routines, the overhead to perform them is substantial. Moreover, the re-implementation efforts to adapt sparse kernels for deep learning usage would be too much time-consuming for the scope of this PhD.

Even though I found this research direction fruitless in terms of real-life efficiency, an interesting pattern emerged from those experiments. A block-sparse transformer prefers to prune parameters within the same neural connections, as in *rows and columns*. It allows for easy *model slicing*: its sparse matrices can be sliced and collapsed to be smaller but still dense in a way that does not affect the flow of calculations. Further experiments support this notion: models that get sliced at the end of training maintain their quality while collapsing zeroed-out connections and significantly boosting inference speed with no need for specialised kernels.

This observation pushed the direction of research toward group lasso applied to individual neurons directly instead of blocks. I investigate neuron-level regularisation of transformer layers in Sect. 4.11 and 4.12. Sect. 4.13 extends this work into larger structures with regularisation over entire attention heads.

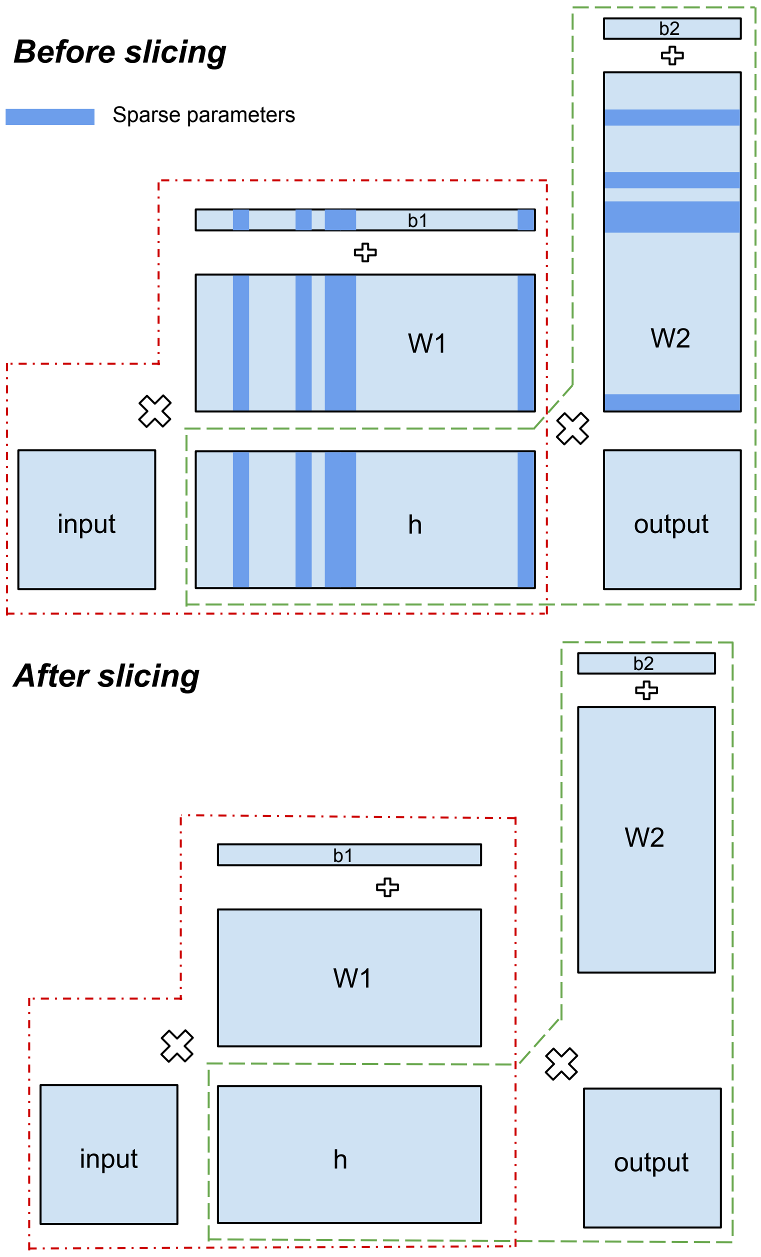


Figure 4.1: Visualisation of feedforward calculations with pruned connections corresponding to rows and columns. Sparse parameters can be removed and matrices collapsed without affecting input and output.

Next, I analyse the resulting models in parameter distribution and Pareto optimality in speed vs quality. Following that, I present the findings of the Efficiency Shared Task, in which I participated and applied this structural pruning approach in practice. Last but not least, I look into the human evaluation and its subsequent judgement.

The research on group lasso pruning presented in this chapter has been published in Behnke et al. (2021) and subsequently explored in Behnke and Heafield (2021).

4.4 Methodology

4.4.1 Model slicing

Due to the nature of affine calculations, it is possible to slice the matrices in a feedforward layer to obtain a smaller but still dense architecture. To show it is possible, I illustrate matrix calculations performed in one feedforward layer in Fig. 4.1. The first affine operation is in a dot-dashed line, in which the input tensor is multiplied with the W_1 parameter matrix and the bias term b_1 is added. The second affine operation is in a green dashed line, and the process is repeated with the other set of parameters instead. The dark blue segments of W_1 and W_2 (and their biases) are to be removed. It is possible to slice corresponding rows and columns to collapse matrices in a way that affects neither input, output, nor the hidden representation h in-between.

4.4.2 Pruning scheme

Throughout the preliminary experiments in Sect. 4.8.3, I explore various pruning schedules to achieve the best translation quality without compromising on training time. Inspired directly by the three steps in the lottery ticket method (see Fig. 3.7), I adapt a similar approach with structural regularisation. Fig. 4.2 presents the general regularisation scheme used in this chapter.

Initial stages of transformer training are known to be problematic and sensitive to model hyperparameters (Aji, Heafield, & Bogoychev, 2019; L. Liu, Liu, Gao, Chen, & Han, 2020; Nguyen & Salazar, 2019). During a learning rate warm-up, 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, the first step of the pruning is short *pretraining* to stabilise the quality in early stages of training, just like in the previous work. BLEU improvement slows down to be less than 1 BLEU point in a single checkpoint. This way, I avoid potential damage to a model during this critical initial period. The next phase turns regularisation on and prunes parameters structurally at the end of it. Finally, a model is allowed to converge with its architecture collapsed into a smaller one after removing sparse parameters. In case of the regularisation being on until convergence, the architecture is only sliced at the end.

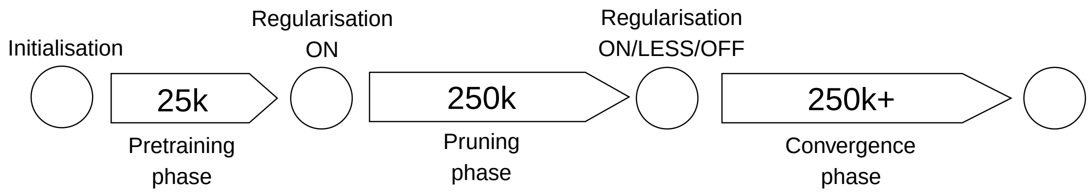


Figure 4.2: Visualisation of pruning schemes using structural regularisation.

Each experiment in this chapter has its own scheduling methodology variant explained in their specific sections.

4.5 Related work

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, Comminiello, Hussain, & Uncini, 2017; Wen, Wu, Wang, Chen, & Li, 2016). Dodge, Schwartz, Peng, and Smith (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 I avoid. They provide no speed or model size analysis, suggesting that there is no improvement or proper implementation.

Wuebker, Simianer, and DeNero (2018) previously used the group lasso to compress the delta between a base model and a domain adapted version of the model. They still have to run a full-sized 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, my work makes the base model faster and smaller. The different goals also mean different groups: they focus on embeddings that update in domain adaptation, while I focus on costly parts of the architecture.

Though we use the same algorithm of group lasso, our method differs in several ways from (Murray, DuSell, & Chiang, 2019). In my later experiments, I 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; I see more speed-up from feedforward layers and additionally prune attention. Finally, I use the normal Adam optimiser Kingma and Ba (2014) instead of proximal gradient descent (Parikh & Boyd, 2014). Empirically, I find turning regularisation off after some training is important to quality. Overall, I achieve a much better trade-off between quality and speed/compression, which I will proceed to show in this chapter.

4.6 Problems with existing sparse kernels

A primary concern of machine translation deployment is efficiency. In the previous chapter, I focused on pruning attention heads first and foremost, and there are multiple key issues in the pruning research field. Unfortunately, the further I looked into researching the topic during my PhD, the more disappointed I got with the current solutions. For instance, many existing methods, such as Voita et al. (2019), often turn out not to be reproducible on large-scale datasets. Most pruning papers suffer from the overly theoretical analysis without a clear demonstration that their brand new method is better than the current state-of-the-art. For the most part, they report size compression but not speed as its optimisation is difficult. They usually present potential boost in FLOPs that does not reflect the real-time impact. Those papers that *do* speed analysis often report slower performance (e.g. Yao et al. (2019) report a 87.5% sparse model being $1.6\times$ slower using cuSPARSE that they eventually improve upon with their custom balanced pruning being $2.4\times$ faster instead) or no gains at all (Brix et al., 2020).

Researchers' huge oversight in the field is comparing their methods to bad baselines. Too much work (Gu et al., 2018; J. Lee et al., 2020; Y. Wang et al., 2020) on efficiency compares a bare baseline model with their optimised system, which is smaller or faster in exchange for some reduction in BLEU. These papers fail to prove that their method works better than existing approaches that also provide a similar quality–efficiency exchange. Among them are techniques such as knowledge distillation Y. Kim and Rush (2016a), quantisation (Aji & Heafield, 2020) or simple benchmarks like training a smaller model with fewer layers, reduced dimensions et cetera. Comparing to a pure unoptimised baseline skews the perception, resulting in "orders of magnitude faster performance" when it does not apply; engineers prefer smaller robust models over new complicated methods with questionable payoff. Blalock, Ortiz, Frankle, and Guttag (2020) has summarised and highlighted many such issues in the pruning research field. The question is whether the trade-off offered by a new method is any better than the trade-offs already available, regardless of the type of method. Stacking the existing methods produces a variety of data points with different speeds 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. I strongly believe that a new method's empirical justification should advance the Pareto frontier. The following research seeks to remedy these shortcomings by building upon and comparing them to strong baselines to show the frontier advances.

4.7 Setup: knowledge-distilled tiny transformers

In the experiments, I concentrate on three language pairs: English→German, Spanish→English and Estonian→English. I use knowledge distillation (Y. Kim & Rush, 2016a) under teacher-student regime. Let us start with tiny student models, which are already very small and fast, making them a solid baseline for optimisation research.

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

I tested different configurations of layers to examine trade-offs between them and potential bottlenecks. The number of layers in both encoder and decoder describe each architecture and whether the decoder is tied. Thus, I investigate the following architectures: "6–2 tied", "6–2", "6–6" and "12–1".

I evaluate the quality and speed on a single CPU core. In order to expand beyond BLEU and to further explore the impact of pruning on translation quality, I additionally evaluate some experiments with chrF (Popović, 2017) and COMET[1] (Rei, Stewart, Farinha, & Lavie, 2020), with a human evaluation down the line as well. I use SacreBLEU (Post, 2018) for BLEU and chrF. Training finishes when BLEU stops improving for 20 consecutive validations. The checkpoint with the highest BLEU score is then selected.

Other training hyperparameters were Marian defaults for training a base transformer model.[2] I used dynamic batching, filling a 10GB workspace on each of 4 GPUs, resulting in about 71,000 words per batch in a "6–2 tied" student and about 46,000 words per batch in a "6–6" student. As is more effective in the teacher-student regime, no dropout or label smoothing is applied. The optimiser is Adam (Kingma & Ba, 2014).

1. I used the default 'wmt20-comet-da' metric model.
2. Available via `-task transformer-base`.

English→German

For English→German, I re-use the same knowledge-distilled setup as described in Sect. 3.4.5. I follow the Workshop on Neural Generation and Translation 2020 Efficiency shared task (WNGT2020) [3] under the WMT 2019 data condition (Barrault et al., 2019). The training corpus consists of 13.5M parallel sentences, using the concatenated English→German WMT testsets from 2016–2018 as a development set.

Spanish→English

For Spanish→English, I use teachers provided by the Bergamot project,[4] which is an ensemble of two big transformer models (Vaswani et al., 2017a) with 6 layers in encoder and decoder, embedding size of 1024 and feedforward size 4096 and 8 attention heads. The students trained on 242M sentences which included about 15M of mixed forward- and backtranslations. The development set is the WMT13 testset.

Estonian→English

Similarly to Spanish, I use Estonian→English teachers provided by the Bergamot project with the same architecture described above. The students trained on 132M sentences which included about 30M of mixed forward- and backtranslations, and a WMT18/dev was used for development.

4.8 Block-sparse regularisation of feedforward layers

Expanding the work from the previous research, I proceed with block sparsity over feedforward layers. The lottery ticket approach turned out to be quite expensive to perform, and while it resulted in good translation quality, there is a wide array of possible improvements. Instead of using the lottery ticket hypothesis, I achieve block-wise sparsity in feedforward layers through a group lasso. Since block sparsity should be hardware friendly, I perform additional analysis of matrix multiplication routines on parameters using sparse kernels. The goal is to get an even better quality-sparsity trade-off and show a real-time inference speed-up on CPU hardware.

3. `https://sites.google.com/view/wngt20`
4. `https://github.com/browsermt/students`

4.8.1 Setup

I begin with English$\rightarrow$German experiments as described in the Setup (Sect. 4.7). The architecture is 6–2 with a tied decoder.

4.8.2 Methodology

I follow the general schedule defined in Sect. 4.11.1. I decided to pretrain for 25k batches in these experiments. After the pretraining phase, I take the checkpoint and start a fresh training round with *the regularisation turned on until a model converges*.

4.8.3 Experiments: Preliminary

For the first experiments described in this section, I focus on pruning feedforward layers, which consist of two affine operations with parameter matrices W_1 and W_2. I apply a block-wise group lasso with a block size set to 8×8 to both encoder and decoder layers. The bias terms are included in regularisation with 1×8 block sizes instead.

The λ is grid-searched over $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1.0\}$ to see a variety of sparsity levels. The results are presented in Tab 4.1, which specifies the percentage of blocks pruned in each feedforward layer. A block is considered pruned if a sum of absolute parameters within it is less than $1\mathrm{e}{-6}$.

Again, the single decoder layer is a bottleneck, reluctantly pruned only with increasingly stronger regularisation. There is a jump in a sparsity level between λ being 0.1 and 0.2, which means there exists a point where the regularisation fully launches for most layers. The first encoder layer is prioritised but reaches similar sparsity to other layers when aggressively pruned in general.

In terms of quality, removing about half the blocks damages the model by -1.1 BLEU point. Pruning more than 90% results in 2.0–2.7 BLEU loss. It is substantial damage, given that we do not know yet how much faster these models could potentially be with block-sparse kernels in place. Moreover, the models were trained with regularisation until convergence. It may impair the overall quality as a network does not have the opportunity to recover from pruning without the constraint of a penalty.

	Reg. $\lambda \rightarrow$	Base	0.1	0.2	0.3	0.4	0.5	0.75	1.0
Block sparsity	Enc. 1	0%	42%	84%	93%	94%	95%	96%	96%
	Enc. 2	0%	6%	65%	86%	90%	93%	92%	96%
	Enc. 3	0%	0%	54%	79%	89%	93%	96%	97%
	Enc. 4	0%	5%	64%	85%	90%	95%	96%	98%
	Enc. 5	0%	5%	63%	84%	89%	92%	96%	98%
	Enc. 6	0%	0%	40%	71%	81%	87%	93%	97%
	Dec. 1	0%	0%	0%	38%	58%	73%	86%	91%
	Avg.	0%	8%	53%	77%	84%	90%	93%	96%
BLEU	WMT16	36.7	36.8	36.0	35.4	35.2	35.2	34.6	34.6
	WMT17	29.6	29.1	28.4	28.0	27.8	27.9	27.3	27.2
	WMT18	44.0	43.4	42.4	41.8	41.4	40.9	40.5	40.2
	WMT19	40.0	40.6	39.2	39.1	38.3	38.2	37.5	37.4
	Avg.	37.6	37.5	36.5	36.1	35.7	35.6	35.0	34.9
	Δ	—	-0.1	-1.1	-1.5	-1.9	-2.0	-2.6	-2.7

Table 4.1: Quality analysis of tiny knowledge-distilled transformer models for English→German after removing a given percentage of *blocks*.

	Reg. $\lambda \rightarrow$	Base	0.3	0.3 → 0.1	0.5	0.5 → 0.1
Block sparsity	Enc. 1	0%	93%	92%	95%	94%
	Enc. 2	0%	86%	86%	93%	93%
	Enc. 3	0%	79%	78%	93%	93%
	Enc. 4	0%	85%	85%	95%	94%
	Enc. 5	0%	84%	84%	92%	91%
	Enc. 6	0%	71%	70%	87%	86%
	Dec.1	0%	38%	37%	73%	69%
	Avg.	0%	77%	76%	90%	89%
BLEU	WMT16	36.7	35.4	35.8	35.2	35.6
	WMT17	29.6	28.0	28.4	27.9	28.0
	WMT18	44.0	41.8	42.4	40.9	41.6
	WMT19	40.0	39.1	39.4	38.2	38.9
	Avg.	37.6	36.1	36.5	35.6	36.0
	Δ	—	-1.5	-1.1	-2.0	-1.6

Table 4.2: Quality analysis of tiny knowledge-distilled transformer models for English→German with block-sparse group lasso compared to the same model trained with λ reduced to 0.1 about halfway through training ($250k$ batches).

4.8.4 Experiments: Reducing λ halfway through training

For this experiment, I select the two models trained with $\lambda \in \{0.3, 0.5\}$. They converged in about $500k$ batches, but most parameters got pruned much earlier. For this reason, I re-run the experiment but this time reducing λ to 0.1 about halfway through training. This change should positively impact quality while still enforcing sparsity at the same time. I arbitrarily set the pivot to reduce λ at $250k$ batches as it is roughly halfway through training.

As presented in Tab. 4.2, the decrease of λ reactivated some blocks (about 1% per layer) but improved the translation quality by 0.4 BLEU on average. It indicates that it could be a good idea to introduce a form of regularisation scheduling that either reduces λ over time or stops it altogether to allow a model to focus on training towards quality after most pruning phase finishes.

4.9 Optimising matrix multiplications with sparse kernels

Despite the popularity of the pruning topic in deep learning research, there are rarely any clear advantages in speed. If they are, they are unavailable without custom implementations or specialised hardware. There is an ongoing effort to include pruning as a 'staple' optimisation approach performed alongside others. However, for a long time, pruning has been known to be not worth the effort unless it can achieve high ($> 90\%$) sparsity levels without compromising on quality. The fact that image recognition is the most popular task for pruning research obscures the extent of damage pruning may cause to other tasks such as those from natural language processing, which are more sensitive to parameter reduction.

The question is: *what is a potential speed-up if the neural network toolkit properly implements and supports sparse calculations?* The goal of this section is to analyse potential potential speed and compression gains from block-wise sparse operators in matrix multiplications. I select the model from the previous section trained with $\lambda = 0.3 \rightarrow 0.1$ (see Tab. 4.2), which loses about 1.1 BLEU points and has varied sparsity rates for each layer.

I evaluate various matrix multiplication routines directly in C++ using Intel MKL library[5]. To emulate similar circumstances within the neural network, I loaded matrices from the model and multiplied them with random dense matrices . I declare A being a sparse matrix of $(256, 1536)$ dimensions with parameters taken a tiny transformer and B being a randomly initialised dense matrix of size $(1536, 512)$ which simulates typical sizes for activation matrices. Then, I measure the execution of matrix multiplication routines ($C = AB$) and memory consumption using the following sparse representations: COO, CSR and BSR as introduced in Sect. 2.4. The last one should be the most suitable since the model has been specifically pruned block-wise. GEMM is a standard dense matrix multiplication routine. The matrices are between 37% to 92% sparse, which should be an informative enough range to gauge the speed impact.

5. Intel(R) Math Kernel Library Version 2019.0.4 Product Build 20190411 for Intel(R) 64 architecture applications

		Byte compression		
Layer	Sparsity	COO	CSR	BSR
Encoder 1	92%	×9.31	×9.68	×12.31
Encoder 2	86%	×2.61	×2.16	×2.18
Encoder 4	85%	×2.39	×1.96	×1.98
Encoder 5	84%	×2.10	×1.70	×1.71
Encoder 3	78%	×1.66	×1.32	×1.33
Encoder 6	70%	×1.12	×0.87	×0.88
Decoder 1	37%	×0.53	×0.40	×0.41

Table 4.3: The comparison of memory byte compression between dense and various sparse representations of matrices from the block-sparse model regularised with $\lambda = 0.3 \rightarrow 0.1$. Layers are sorted by sparsity.

		Execution time (s)			
Layer	Sparsity	GEMM	COO	BSR	CSR
Encoder 1	92%	18.74	1.77	2.60	2.53
Encoder 2	86%	18.66	3.78	4.64	5.56
Encoder 4	85%	18.71	3.95	4.82	5.76
Encoder 5	84%	18.65	4.48	5.32	6.23
Encoder 3	78%	18.72	5.38	7.21	8.64
Encoder 6	70%	18.61	7.46	9.95	11.94
Decoder 1	37%	18.73	15.60	20.67	24.74

Table 4.4: A comparison of speed between dense and sparse matrix multiplication routines on matrices from the block-sparse model regularised with $\lambda = 0.3 \rightarrow 0.1$. Most sparse multiplication routines require matrices to be at least 50% sparse to be noticeably faster. Layers are sorted by sparsity.

4.9.1 Analysis: Byte compression

First, I looked into each layer's byte compression of allocated memory. COO has two arrays holding row and column pointers for each value, which triples the number of bytes. Both CSR and BSR use their 3-array representations. As shown in Tab. 4.3, a matrix needs to be at least at least two-thirds sparse to be storage-efficient. It is a considerable threshold to uphold, given that not all layers have been pruned that much (e.g., the decoder is only 37% sparse). As the quality evaluation shows in Tab. 4.1 and 4.2, there may be a significant damage required to trade for such sparsity.

		Execution time (s)			
Layer	Sparsity	GEMM	COO	BSR	CSR
Encoder 1	92%	18.72	2.24	32.56	2.88
Encoder 2	86%	18.64	3.96	32.77	5.28
Encoder 4	85%	18.64	4.31	32.77	5.70
Encoder 5	84%	18.54	4.98	32.69	6.55
Encoder 3	78%	18.71	6.33	32.84	8.88
Encoder 6	70%	18.70	8.10	32.70	11.55
Decoder 1	37%	18.74	15.47	32.77	23.08

Table 4.5: A comparison of speed between dense and *shuffled* sparse matrix multiplication routines on matrices from the block-sparse model regularised with $\lambda = 0.3 \to 0.1$. When block structures are destroyed, BSR has no advantage anymore. Layers are sorted by sparsity.

4.9.2 Analysis: Execution timing

Next, I looked into evaluating the speed of matrix multiplication routines. Each sparse multiplication executes 5 times to allow the library to run its internal optimisations, and then it is averaged across 20 runs. For dense operations, I used the standard *cblas_gemm* function. For sparse multiplications, I used *mkl_sparse_optimize* and *mkl_sparse_s_mm* functions. The results are presented in Tab. 4.4. Surprisingly, the simplest coordinate representation COO is the fastest of all. BSR comes second, outperforming CSR due to explicit block-wise optimisation. However, both BSR and CSR surpass dense GEMM (general matrix multiplication) at about 50% sparsity. To confirm the advantage of BSR for block sparsity, I shuffled the matrices and re-ran the evaluations again as presented in Tab. 4.5. The sparsity of each layer is still the same, but there is no block-wise structure anymore, which is reflected in BSR being almost twice as slow as a dense multiplication.

Of all three sparse representations, it is surprising to see COO outperform BSR, specifically implemented to work with blocks. It indicates that the Intel MKL library may be unoptimised for typical NMT use cases. The next issue is that this specific library only supports sparse $\times$ dense matrix multiplication but not dense $\times$ sparse, which are required in a deep learning toolkit. A similar effect could be achieved by double transposing both matrices before and after the multiplication, but that would negatively balance any speed-up we may gain. Due to increased matrix multiplication overhead and the significant amount of work required for proper sparse assimilation into a deep learning toolkit, I decided to abandon this research direction favouring simpler sparse solutions. This analysis inspired me to look into sparsity in NMT that does not require any elaborative sparse routines to achieve actual speed-up.

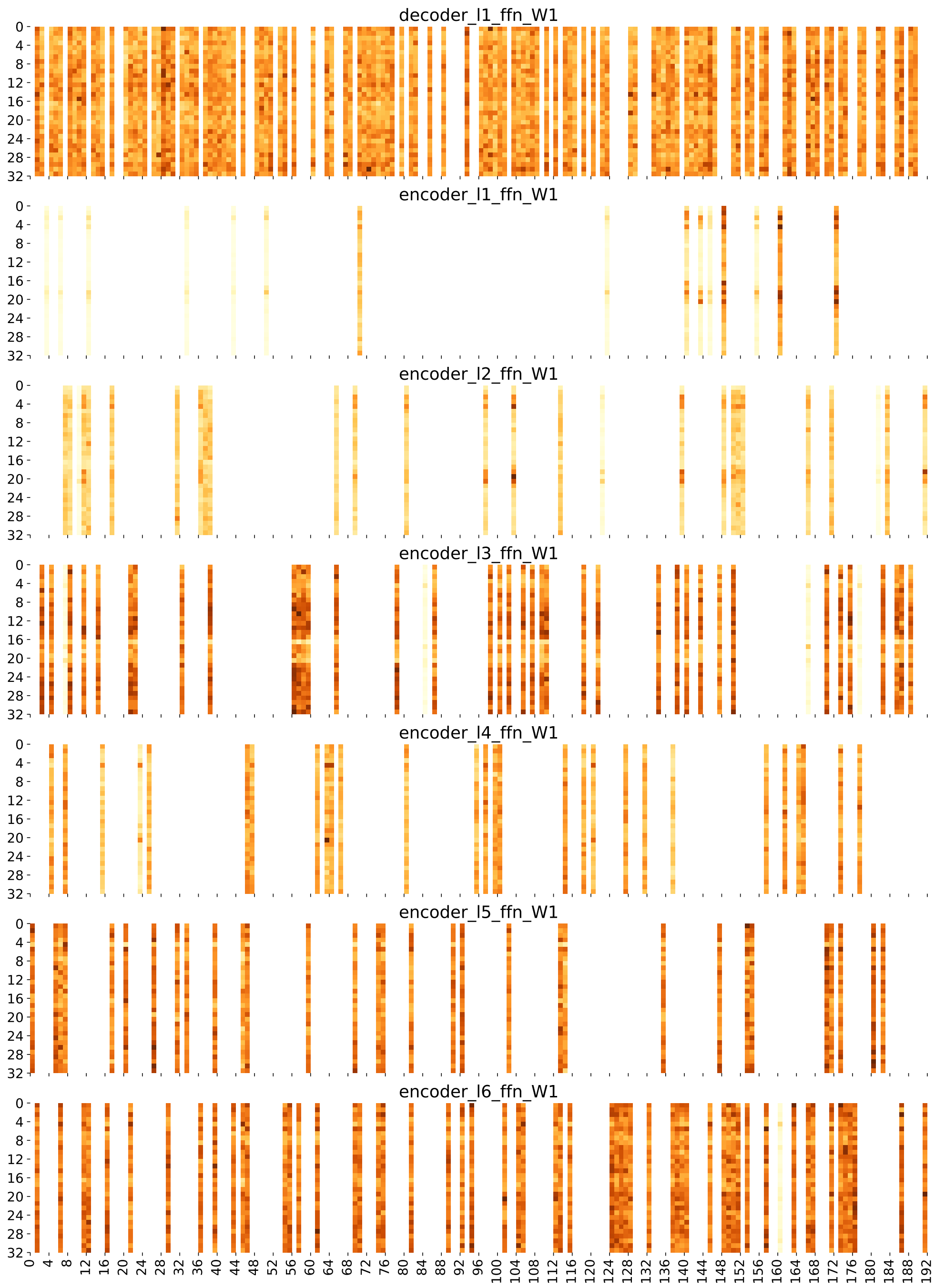


Figure 4.3: A visualisation of feedforward layers in a block-sparse model regularised with $\lambda = 0.3 \rightarrow 0.1$. Each colored block represents an active 8×8 block of parameters in a 256×1536 matrix.

4.10 Slicing and collapsing sparse models for speed

During the last experiment on the block-sparse NMT, I looked into the pruned model's parameters and discovered something peculiar. The group lasso regulariser pruned blocks of parameters *concurrently within the same rows and columns*. To further demonstrate it, I visualise the first feedforward matrix (W_1) of the model in Fig. 4.3 for all layers. Each coloured dot represents the sum of absolute values in an 8×8 block, with white areas depicting pruned parameters. The second matrices (W_2), though not shown here, look almost identical in their sparsity to the first ones but are flipped instead.

This lack of randomness in sparsity induced by a block-wise regularisation strongly suggests that a transformer may prefer fewer but still dense subnetworks in it. Most importantly, the fact that blocks align within the same *rows and columns* implies that backpropagation aims to remove *neural connections*. It is not the case when using a coefficient-wise regularisation such as L_1 or Elastic Net, where individual parameters are scattered around.

4.10.1 Experiment

As explained in Sect. 4.11.1 and Fig. 4.1, neural connections can be easily sliced out and removed entirely from a model. It is especially exciting from the optimisation point of view. If we can slice feedforward layers straightforwardly like attention heads, we can get a more noticeable inference speed-up with no need for specialised matrix multiplication kernels. To test this notion in practice, I slice the models from Tab. 4.1 and 4.2 and remove the inactive connections. A neuron is considered inactive if a sum of absolute parameters in a row/column is less than $1\text{e}{-6}$. There is no additional training involved, just simple slicing of the already converged models.

Having done that, I re-evaluate the models, noting a speed-up on a single CPU core. The results are presented in Tab. 4.6. The quality is mostly the same, with a few occasional ± 0.1 BLEU changes, which can be attributed to a slight decrease in numerical precision due to slicing parameters that are not exactly zero but close to it. Moreover, the models are neither fine-tuned nor further trained so a model cannot accomodate to sudden removal of parameters. Still, the quality change is negligible, showing that the architectures do not depend on those parameters anymore.

Each feedforward layer has a new custom dimension now. I also highlight two representative models in terms of achieved quality. Removing three-quarters of feedforward neurons leads to $1.36\times$ faster translation at the cost of 1.1 BLEU points on average, while being more aggressive with 90% sparsity results in $1.45\times$ speed-up with -1.6 BLEU in damage.

		Base	0.1	0.2	0.3	0.3→0.1	0.4	0.5	0.5→0.1	0.75	1.0
FFN dimension	Enc. 1	1536	782	143	47	55	42	25	28	20	21
	Enc. 2	1536	1402	478	187	196	129	74	77	43	22
	Enc. 3	1536	1528	686	305	308	155	98	98	45	27
	Enc. 4	1536	1416	531	201	214	134	67	68	34	15
	Enc. 5	1536	1430	548	244	244	168	119	134	58	24
	Enc. 6	1536	1536	889	440	458	287	200	201	96	40
	Dec. 1	1536	1536	1536	889	968	572	394	465	196	123
BLEU	WMT16	36.7	36.8	36.0	35.4	35.8	35.2	35.2	35.6	34.6	34.6
	WMT17	29.6	29.1	28.4	28.1	28.4	27.8	27.9	28.0	27.3	27.2
	WMT18	44.0	43.4	42.4	41.8	42.4	41.4	40.9	41.6	40.5	40.2
	WMT19	40.0	40.6	39.2	38.9	39.4	38.3	38.2	38.9	37.5	37.4
	Avg.	37.6	37.5	36.5	36.1	36.5	35.7	35.6	36.0	35.0	34.9
	Δ	—	-0.1	-1.1	-1.5	-1.1	-1.9	-2.0	-1.6	-2.6	-2.7
	Avg. time	38.6	37.7	32.5	27.9	28.3	26.3	25.7	26.6	24.9	24.6
	Sparsity	0%	10%	55%	78%	77%	86%	91%	90%	95%	97%
	Speed-up	1.00	1.02	1.19	1.38	1.36	1.47	1.50	1.45	1.55	1.57

Table 4.6: Quality and speed analysis of "6–2 tied" tiny transformer models for English→German with block sparsity after slicing inactive neurons out.

4.10.2 Efficiency of previous attention pruning and new results

I performed a similar speed analysis on the same knowledge-distilled architecture in my previous research on attention pruning. I put the direct comparison between two models of similar translation quality but with different pruning approaches in Tab. 4.7.

Model	Att. sparsity	FFN sparsity	WMT19 BLEU	Speed-up
Lottery ticket iter=14	88%	0%	38.9	1.12
Group lasso $\lambda = 0.5 \rightarrow 0.1$	0%	90%	38.9	1.45

Table 4.7: Direct comparison between two models pruned with the lottery ticket and group lasso of similar translation quality but different speed-up gains.

They both evaluate 38.9 BLEU on the WMT19 testset but get largely different inference speed-up compared to the baseline. In the previous research on the lottery ticket, I only achieved $1.12\times$ speed-up when removing 80% of all attention heads. With the similar quality trade-off on WMT2019, the same model is $1.45\times$ faster with 90% of feedforward neurons removed. At that time, I argued that optimising small state-of-the-art architectures is quite difficult, and there may be a limit on how far we can push. However, these recent experiments on group lasso pruning of feedforward layers clearly show that there is still more room for improving efficiency.

		Execution time (s)		
		Full		Sliced
Layer	Sparsity	GEMM	COO	GEMM
Encoder 1	92%	21.10	0.79	1.01
Encoder 2	86%	21.03	3.87	2.67
Encoder 4	85%	21.04	4.31	3.63
Encoder 5	84%	20.94	5.16	3.52
Encoder 3	78%	21.06	6.29	4.29
Encoder 6	70%	20.98	9.12	6.97
Decoder 1	37%	21.13	19.25	13.72

Table 4.8: A comparison of speed between sparse and *dense sliced* matrix multiplication routines on matrices from the block-sparse model regularised with $\lambda = 0.3 \rightarrow 0.1$. Slicing removed inactive connections (rows and columns), making dense GEMM faster.

4.10.3 Speed evaluation of sliced matrices and sparse kernels

Previously in Sect. 4.9, I analysed the performance of various sparse multiplication routines using matrices from one of the pruned models. Here, I will show that sparse kernels are not required to be speed-efficient and that, in fact, they perform worse than dense calculations on smaller sliced matrices.

Tab. 4.8 presents execution times similar to those in Tab. 4.4. First, I evaluate the same block-sparse model regularised with $\lambda = 0.3 \rightarrow 0.1$ but with all connections intact (see Tab. 4.6). I re-run GEMM afresh for a fair evaluation. Coordinate (COO) sparse representation was the fastest in the Intel MKL library in the previous evaluation in Tab. 4.4, so I chose it as a representative of sparse kernels. Next, I take the same model but with sliced and collapsed architecture. This model performs the same quality-wise in both full and sliced circumstances. In the sliced case, the multiplication is dense with a reduced "sparse" matrix $(256, X)$ and dense activations $(X, 512)$, with $X \in [0, 1536]$ being a new smaller inner dimension.

As you can see in Tab. 4.8, sliced matrices outperform the full block-sparse matrices when using dense GEMM as well as COO multiplier. The simplicity of removing connections out from a model and collapsing matrices combined with significantly faster calculations makes this method the best choice to handle sparsity. Not only slicing does not require sparse kernels to be efficient, but dense GEMM is also better on small matrices than the fastest sparse kernel I have tested. For these reasons, I proceed with this easy yet extremely effective approach further into my research.

Given that the block-wise group lasso pruned parameters in a lined-up way, I shift my attention towards a specific case of block sparsity, in which a block is $1 \times N$ with N being the dimension of a layer. In other words, my goal is to apply group lasso over *rows and columns* directly to avoid forcing a network to prune neighbouring connections unnecessarily with blocks.

4.11 Neuron-level regularisation of feedforward layers

The following study aims to assess neuron-level group lasso pruning of feedforward layers to produce smaller but still dense architectures. This section systematically explores the best approaches to regularisation and subsequent pruning. I continue to work with robust knowledge-distilled models to show that this method pushes the state-of-the-art forward in highly-optimised settings.

Multiple directions will be examined. First of all, the previous experiments have shown that reducing λ improved translation quality without compromising on final sparsity. I hypothesise that regularisation should stop to allow a model to recover better from pruning damage. A model could even be sliced during training to remove parameters and then continue training with lower memory usage. Furthermore, I intend to address the potential decoder bottleneck. A decoder is expensive, and its pruning is highly beneficial to efficiency. On the other hand, it is often not as eagerly pruned when regularised and being too aggressive may damage the quality too much.

Also, I expand my experiments into a variety of languages and datasets to consider different scenarios. One of the unanswered questions in the pruning research field is how pruning behaves under high-resource conditions. We do not know how reliable new pruning methods are, so I include an analysis of a setup with 242M parallel sentences in a corpus. Experiments of such scale are still widely unexplored in machine translation, raising whether known methods are beneficial in real-life scenarios. Most pruning papers use English→German models under WMT14 constraints (Bojar et al., n.d.), which is only 4.5M sentences (Brix et al., 2020; Hsu et al., 2020; See et al., 2016), sometimes branching into different languages such as Russian or French in a similar scope of data size(Kasai, Pappas, Peng, Cross, & Smith, 2020; Voita et al., 2019).

Last but not least, recent reports from the machine translation community (Kocmi et al., 2021) highlight the need to go beyond BLEU in quality evaluations as it is not a perfect measure by itself to judge whether a model is “good” for deployment or not. For that reason, I provide additional analysis with chrF and COMET scores.

4.11.1 Methodology

I follow the same approach as described in Sect. 4.2, but this time I apply group lasso over rows and columns in parameter matrices. I am yet to gauge what is the best scheduling for regularised pruning in NMT, but there are overall three steps to the procedure:

1. Pretraining phase — to avoid issues in the early stages of transformer training. It is quite short as it only serves to stabilise BLEU performance.
2. Regularisation phase — this is the key step in which the penalties zero out most of the parameters it is supposed to.

3. Convergence phase — at this point, the regularisation strength is either decreased or entirely halted to allow a model to regain its lost quality.

The next experiment determines the approximate length and variants of these phases. After each step, I copy the latest checkpoint and start a fresh training round. Thus, all training hyperparameters (learning rate etc.) and optimizer state get reset. I checked and found no additional advantage to the baselines by refreshing learning rate scheduling or Adam optimiser. I do so to avoid partially retraining the same settings during the development phase, but Brix et al. (2020) found it beneficial in their pruning scheme.

4.11.2 Experiments: Exploring pruning phases

Let us start with the most straightforward approach: prune everything (as in feedforward layers in both encoder and decoder) by regularising until convergence. The slicing happens at the end as the last step just before the evaluation. The models are trained with $\lambda \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 1.0\}$ to check a whole range of sparsities. Results are presented in Tab. 4.9a. A target dimension of each feedforward layer after slicing is specified, with the overall percentage of feedforward connections removed. Evaluation BLEU is averaged over the WMT testsets from 16 to 19 for more concise reporting, with speed-up expressed in words per second (WPS). I highlight the trade-off between the BLEU difference and gained inference acceleration on a single CPU core compared to the baseline.

From the quality point of view, the drop is quite considerable as the regularisation gets more intensive in its pruning. Even though the only decoder layer is the least pruned among all, it is still greatly sparsified, which, in turn, accelerates translation. Removing about half of the feedforward connections results in $1.13\times$ faster inference at the cost of -0.4 BLEU. Considering this is a highly optimised setup, it is quite promising. It shows that directly targeting rows and columns (connections) instead of blocks with similar aligned sparse patterns in the aftermath is generally better on quality. In Tab. 4.6, removing a little more than half of the connections through a block-sparse training caused -1.1 BLEU in damage in contrast to the -0.4 mentioned above. It is a vast improvement, and there is still room for more.

To understand how pruning progresses, I visualise neuron distribution for each feedforward layer during training for one of the models in Fig. 4.4. Like in the previous block-sparse experiments, most parameters get removed rather early. The later sparsity gains are not as impressive since a model most probably focuses on keeping penalties in check while still optimising translation quality as much as possible. I set the pivot at $250k$ batches once again: it proved to be a reliable choice given that it is roughly halfway through the training process for most of my experiments. By that point, most parameters that were to be pruned had been done so.

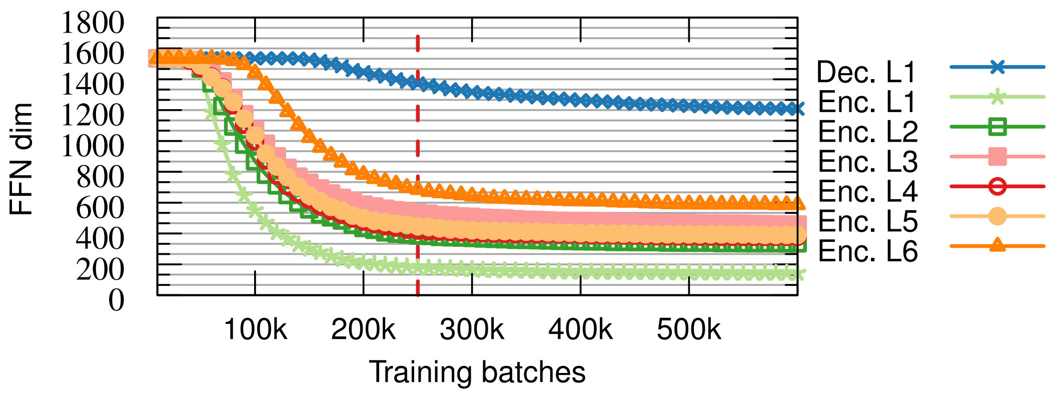


Figure 4.4: Pruning progression of feedforward layers in an English$\rightarrow$German tiny transformer model ($\lambda = 0.5$). About "halfway" through, most parameters are already removed. The dashed line represents the checkpoint I selected as a pivot at $250k$ batches.

So far, reducing λ in the middle of the training regime has helped maintain the sparsity levels and improve the final translation quality. Now I repeat the experiments, but this time with neuron-wise regularisation. I train selected checkpoints with λ reduced to 0.1 after $250k$ updates. I omit setups with $\lambda \in \{0.1, 0.2\}$ due to their redundancy. The results are presented in Tab. 4.9b. Removing about half of the parameters maintains the average quality instead of losing -0.4 BLEU point, which was the case when regularising until convergence. About 2% of connections got reactivated due to the smaller λ used, but it does not affect speed-up ($1.14\times$ vs $1.13\times$).

If reducing λ does help, what about just turning it off completely? I repeat the experiments, now disabling the regularisation at $250k$ checkpoints, with the results in Tab. 4.9c. The difference in target sparsity between models regularised until convergence and those tuned without it is small: between 2 to 5%. Only a few parameters get removed through the second part of training. Compared to reducing λ, turning regularisation off leads to further improvements, especially in the most pruned models. For example, a model with 85% parameters removed lost -1.1 BLEU when the equivalent model with the reduced λ lost -1.5.

In summary, I explored three variants of the approach described in Methodology (Sect. 4.11.1):

Train until convergence Regularisation phase lasts until convergence, with a model sliced only at the end of training.

Reduce λ After $250k$ updates, reduce λ to 0.1, continue training with it until convergence. Slice a model at the end.

Stop regularisation After $250k$, stop regularisation. Slice a model and continue training a smaller architecture until convergence.

	Reg. $\lambda \longrightarrow$	Base	0.1	0.2	0.3	0.4	0.5	0.7	1.0
FFN dimension	Enc. 1	1536	1301	681	281	144	87	39	21
	Enc. 2	1536	1475	1111	551	337	222	112	47
	Enc. 3	1536	1521	1393	794	459	306	151	74
	Enc. 4	1536	1453	1291	673	379	242	126	56
	Enc. 5	1536	1467	1207	645	391	279	164	90
	Enc. 6	1536	1536	1506	930	589	421	253	135
	Dec. 1	1536	1536	1536	1520	1211	890	513	255
BLEU	WMT16-19	37.6	37.8	37.6	37.2	36.7	36.3	35.9	35.6
	Δ	—	0.2	0.0	-0.4	-0.9	-1.3	-1.7	-2.0
Efficiency	FFN sparsity	0%	4%	18%	50%	67%	77%	87%	94%
	WPS	2404	2400	2440	2728	2799	3006	3291	3419
	Speed-up	1.00	1.00	1.01	1.13	1.16	1.25	1.37	1.42

(a) With the regulariser on until convergence.

	Reg. $\lambda \longrightarrow$	Base	0.1	0.2	0.3	0.4	0.5	0.7	1.0
FFN dimension	Enc. 1	1536	——	——-	303	157	99	42	25
	Enc. 2	1536	——	——-	576	356	240	118	55
	Enc. 3	1536	——	——-	827	493	324	161	79
	Enc. 4	1536	——	——-	698	397	251	133	60
	Enc. 5	1536	——	——-	668	414	288	170	93
	Enc. 6	1536	——	——-	974	626	451	269	148
	Dec. 1	1536	——	——-	1528	1293	981	554	295
BLEU	WMT16-19	37.6	——	——-	37.6	37.3	36.7	36.1	35.8
	Δ	—	——	——-	0.0	-0.3	-0.9	-1.5	-1.8
Efficiency	FFN Sparsity	0%	——	——-	48%	65%	76%	87%	93%
	WPS	2404	——	——-	2738	2796	2992	3252	3462
	Speed-up	1.00	——	——-	1.14	1.16	1.24	1.35	1.44

(b) With the regulariser reduced to $\lambda = 0.1$.

	Reg. $\lambda \longrightarrow$	Base	0.1	0.2	0.3	0.4	0.5	0.7	1.0
FFN dimension	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	WMT16-19	37.6	——-	——-	37.8	37.4	36.8	36.5	36.1
	Δ	—	——-	——-	0.2	-0.2	-0.8	-1.1	-1.5
Efficiency	FFN sparsity	0%	——-	——-	45%	63%	73%	85%	92%
	WPS	2404	——-	——-	2613	2748	3067	3215	3420
	Speed-up	1.00	——-	——-	1.09	1.14	1.28	1.34	1.42

(c) With the regulariser switched off.

Table 4.9: "6–2 tied" transformer models for English$\rightarrow$German with feedforward connections in *encoder and decoder* pruned. Evaluated on 1 CPU core.

	Reg. $\lambda \longrightarrow$	Base	0.1	0.2	0.3	0.4	0.5	0.7	1.0
FFN dimension	Enc. 1	1536	1291	621	258	146	77	34	25
	Enc. 2	1536	1474	1030	523	345	202	88	52
	Enc. 3	1536	1522	1332	732	440	263	116	59
	Enc. 4	1536	1448	1248	615	342	208	97	50
	Enc. 5	1536	1462	1136	578	367	224	131	75
	Enc. 6	1536	1536	1436	795	515	344	180	114
	Dec. 1	1536	1536	1536	1536	1536	1536	1536	1536
BLEU	WMT16-19	37.6	37.9	37.5	37.1	36.7	36.4	36.2	35.9
	Δ	—	0.3	0.0	-0.5	-0.9	-1.2	-1.4	-1.7
Efficiency	FFN sparsity	0%	4%	22%	53%	66%	73%	80%	82%
	WPS	2404	2402	2497	2719	2895	2916	3066	3072
	Speed-up	1.00	1.00	1.04	1.13	1.20	1.21	1.28	1.28

(a) With the regulariser until convergence.

	Reg. $\lambda \longrightarrow$	Base	0.1	0.2	0.3	0.4	0.5	0.7	1.0
FFN dimension	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	WMT16-19	37.6	——-	——-	37.6	37.3	36.8	36.8	36.4
	Δ	—	——-	——-	0.0	-0.3	-0.8	-0.8	-1.2
Efficiency	FFN sparsity	0%	——-	——-	48%	64%	72%	79%	82%
	WPS	2404	——-	——-	2748	2916	2929	3054	3096
	Speed-up	1.00	——-	——-	1.14	1.21	1.22	1.27	1.29

(b) With the regulariser switched off.

Table 4.10: "6–2 tied" transformer models for English→German with feedforward connections in *encoder* pruned. Evaluated on 1 CPU core.

The gap in quality between Tab. 4.9a and 4.9c is quite substantial. Explicitly turning the regularisation off results in up to 0.7 better ΔBLEU trade-off with respect to the baseline. It is proof that a model needs time to finish training successfully without penalty restrictions in place. Suppose we have an approximate 1.0 BLEU point budget for potential damage. In that case, we can remove two-thirds of feedforward connections if regularising until the end or remove 85% when converging a sliced model with the regulariser turned off. Naturally, this results in faster inference as well.

4.11.3 Experiments: Skipping a decoder to avoid a bottleneck

Having a tied decoder means only one layer of parameters to prune potentially. Regularising both encoder and decoder, as visualised in Fig. 4.4, shows that the decoder is the least prioritised. The sudden quality decrease when the regularisation starts enforcing pruning over a decoder illustrates a bottleneck. In the following experiment, I examine applying a pruning scheme to only an encoder.

The results are presented in Tab. 4.10 with regularising until convergence (Tab. 4.10a) and turned off (Tab. 4.10b). Again, disabling regularisation proves to be more beneficial to the quality. However, the peak speed performance is worse than that with a pruned decoder. Here, the fastest model is $1.29\times$ better than the baseline. Meanwhile, the one with a pruned decoder on top of it is $1.42\times$ faster in the most aggressive case. With no change in BLEU, it removes about half of the encoder parameters. At the -0.3 BLEU cost, two-thirds of parameters get pruned with $1.21\times$ speed-up.

All the pruned models trained so far, either with the decoder included or not, offer different quality vs speed trade-offs. I leave a more extensive Pareto analysis for later in this chapter.

4.11.4 Experiments: Pruning models with a deep decoder

In the previous section, the hypothesis was that a shallow decoder could be a bottleneck when pruned. Preferably, a decoder should have one or two (ideally tied) layers to reduce computations of a single translation step in a loop and better fit a model into a CPU's cache. However, a typical 6-layered architecture is still often used as it provides a better quality despite slower translation.

I proceed with applying group lasso pruning over the encoder and decoder in the same setup I have been using but this time with 6 decoder layers. The previous experiments proved that stopping regularisation and allowing a model to converge freely achieves the best quality. For this reason, I move forward solely with this approach with the results presented in Tab. 4.11.

Since there are many decoders layers to prune parameters from, a potential speed-up is even higher. Removing only one-third of parameters does not affect the quality but gives a good $1.12\times$ in a speed boost. For a small cost of 0.2 BLEU, two-thirds of parameters can be removed and achieve a $1.34\times$ speed-up. In the most extreme case, where almost all feedforward layers get obliterated (98% sparsity), translation is $1.61\times$ faster for less than 2 BLEU points.

The last decoder layer does not start shedding parameters until the 80% sparsity mark is reached. The reluctance to prune it is understandable: this is the last layer that generates a probability distribution for translation, and trimming this specific layer impacts an output directly.

	Reg. $\lambda \longrightarrow$	Base	0.1	0.15	0.2	0.3	0.4	0.5	1.0
FFN dimension	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	WMT16–19	38.5	38.7	38.6	38.3	37.7	37.6	37.4	36.8
	Δ	—	0.2	0.1	-0.2	-0.8	-0.9	-1.1	-1.7
Efficiency	FFN sparsity	0%	31%	55%	69%	81%	87%	91%	98%
	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 4.11: "6–6" transformer models for English→German with feedforward connections in *encoder and decoder* pruned. Evaluated on 1 CPU core.

This deep architecture prioritises quality over speed, but the experiment has shown that it is easy to make it faster without compromising quality. It can be useful in offline translation, which requires many resources to obtain the best quality.

4.11.5 Analysis: Quality outside of BLEU

Though simple to use, a BLEU score does not correlate well with human evaluation. There are many instances of models maintaining BLEU after optimisation or other modification, but the user-perceived quality goes down. Human evaluation is expensive and time-consuming to perform, often completely inaccessible for researchers. However, I had an opportunity to evaluate models pruned in this chapter later in Sect. 4.17 as a part of the Efficiency Shared Task.

To check whether my pruning method performs well and if there are no false-positive outcomes obscured by using just BLEU, I further analyse models with chrF (Popović, 2017) and COMET (Rei et al., 2020). I evaluate all best English→German models trained so far in this chapter with the regulariser switched off halfway. The review is presented in Tab. 4.12 ("6–2 tied", encoder and decoder pruned), Tab. 4.13 ("6–2 tied", encoder pruned only) and Tab. 4.14 ("6–6", pruned both). Besides the *Quality* section containing BLEU, chrF and COMET scores averaged over WMT16–19 testsets, I also provide total sparsity across feedforward layers, inference speed-up and the disk size of a model in megabytes in *Efficiency*.

	Reg. $\lambda \longrightarrow$	Base	0.3	0.4	0.5	0.7	1.0
Quality	BLEU	37.6	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
Efficiency	FFN sparsity	0%	45%	63%	73%	85%	92%
	Size (MB)	61	51	47	45	43	41
	Speed-up	1.00	1.09	1.14	1.28	1.34	1.42

Table 4.12: Quality evaluation of “6–2 tied” tiny transformer models for English→German with feedforward connections in *encoder and decoder* pruned. Averaged over WMT16–19 testsets. I highlight in red a good representative model that maintains baseline quality.

	Reg. $\lambda \longrightarrow$	Base	0.3	0.4	0.5	0.7	1.0
Quality	BLEU	37.6	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
Efficiency	FFN sparsity	0%	48%	64%	72%	79%	82%
	Size (MB)	61	50	47	45	44	43
	Speed-up	1.00	1.14	1.21	1.22	1.27	1.29

Table 4.13: Quality evaluation of “6–2 tied” tiny transformer models for English→German with feedforward connections in *encoder* pruned. Averaged over WMT16–19 testsets.

	Reg. $\lambda \longrightarrow$	Base	0.1	0.15	0.2	0.3	0.4	0.5	1.0
Quality	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
Efficiency	FFN sparsity	0%	31%	55%	69%	81%	87%	91%	98%
	Size (MB)	83	72	63	58	54	52	50	48
	Speed-up	1.00	1.12	1.26	1.34	1.42	1.49	1.52	1.61

Table 4.14: Quality evaluation of “6–6” tiny transformer models for English→German with feedforward connections in *encoder and decoder* pruned. Averaged over WMT16–19 testsets.

Both chrF and COMET follow the trend set by the BLEU results, which further solidifies this pruning method as reliable. I highlight the representative models which maintain quality across most scores. These models prune 63–72% of all feedforward connections leading to about 30–40% compression in disk size. Depending on the architecture, translation is $1.22\times$ faster in a shallow model and $1.34\times$ with a deep decoder.

4.11.6 Experiments: Pruning or architecture search?

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 4.15: The evaluation of English→German “6–2 tied” students with pruned both 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
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 4.16: The evaluation of English→German “6–6” students with pruned both encoder and decoder (*Pruned*), compared to the same architecture trained from scratch (*Reinit*).

Akin to the lottery ticket approaches, the natural question is whether the pruning procedure is necessary or does it serve as an architecture searching method. If the latter is true, then a model can be trained from scratch with the identical architecture without additional steps.

To test this hypothesis, I perform the following experiment. I take English→German models with “6–2 tied” and “6–6” architectures with feedforward layers pruned neuron-wise. Now, I reinitialise parameters with the same sliced dimensions and train these models again from scratch. I evaluate them once more with BLEU, chrF and COMET to compare with those pruned. The results are presented in Tab. 4.15 and 4.16.

The same models achieve noticeably worse translation quality when trained from the get-go than when carefully pruned through regularisation. A possible explanation is that a model adapts to the removal of selected parameters allowing for a better training flow. Another key aspect is that a larger parameter capacity early into training may be beneficial. While crucial for training purposes, most parameters become obsolete in later stages, which is a fundamental reason why pruning of neural networks works in general. It also justifies why most pruning methods prefer to prune parameters from an already converged model. The gap in quality between pruned and trained from scratch models is the widest for lower and mid sparsity ranges, becoming smaller at the other end of the spectrum as pruning is the most extensive. The contrast between pruned and reinitialised models is especially evident in COMET scores, which supposedly reflect well on human evaluation.

	Reg. $\lambda \longrightarrow$	Base	0.05	0.1	0.15	0.2	0.3	0.4
FFN dimension	Enc. L1	1536	1159	484	203	103	34	25
	Enc. L2	1536	1317	380	182	105	48	29
	Enc. L3	1536	1320	358	169	97	49	33
	Enc. L4	1536	1435	638	255	132	55	28
	Enc. L5	1536	1404	817	400	239	108	63
	Enc. L6	1536	1506	964	462	275	128	75
	Dec. L1	1536	1536	1536	1536	1536	1536	1536
	Dec. L2	1536	1536	1536	1536	1536	1536	1536
	Dec. L3	1536	1536	1536	1536	1536	1536	1536
	Dec. L4	1536	1536	1536	1536	1536	1536	1536
	Dec. L5	1536	1536	1536	1536	1536	1536	1536
	Dec. L6	1536	1536	1536	1536	1536	1536	1536
BLEU	WMT12–13	37.3	37.5	37.0	36.9	36.8	36.9	36.7
	Δ	—	0.3	-0.3	-0.4	-0.5	-0.4	-0.6
	FFN sparsity	0%	6%	30%	41%	45%	48%	49%
	WPS	1407	1444	1508	1558	1540	1591	1577
	Speed-up	1.0	1.03	1.07	1.11	1.09	1.13	1.12

Table 4.17: “6–6” transformer models for Spanish$\rightarrow$English with feedforward connections in *encoder* pruned. Evaluated on 1 CPU core.

4.11.7 Experiments: Pruning in high-resource settings

To continue the investigation into group lasso pruning, let us expand into a high-resource data setting with a “6–6” architecture for Spanish$\rightarrow$English. The corpus contains 242M sentences which is much larger than 13.5M for English$\rightarrow$German. Experiments of such scale are still widely unexplored in research, raising the question of whether known optimisation methods are beneficial in real-life scenarios. Most pruning papers use English$\rightarrow$German models under WMT14 constraints (Bojar et al., n.d.), which is only 4.5M sentences (Brix et al., 2020; Hsu et al., 2020; See et al., 2016), sometimes branching into different languages such as Russian or French in a similar scope of data size (Kasai et al., 2020; Voita et al., 2019).

In the following experiment, I prune the encoder only to show how much a decoder contributes to an inference cost in a deep architectural setup. I use $\lambda \in \{0.05, 0.1, 0.15, 0.2, 0.3, 0.4\}$ with the results presented in Tab 4.17. Removing almost all feedforward parameters in the encoder makes inference only 13% faster. The damage in quality is less than 0.6 BLEU, which is substantial given the small speed-up.

I repeat the experiments, this time pruning both encoder and decoder. The regularisation is set to $\lambda \in \{0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 1.0\}$. The results are presented in Tab 4.18. The overall damage to quality is larger as now pruning is more broadened. However, sparsifying a deep decoder is worthwhile from the optimisation perspective. At the cost of half a BLEU point,

	Reg. $\lambda \rightarrow$	Base	0.1	0.15	0.2	0.3	0.4	0.5	1.0
FFN dimension	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	WMT12–13	37.3	36.9	36.8	36.6	36.3	36.3	36.1	35.9
	Δ	0.0	-0.4	-0.5	-0.6	-1.0	-1.0	-1.2	-1.4
Efficiency	FFN sparsity	0%	48%	67%	77%	86%	91%	94%	98%
	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 4.18: "6–6" transformer models for Spanish→English with feedforward connections in *encoder and decoder* pruned. Evaluated on 1 CPU core.

the model is $1.29\times$ faster than the baseline. In the most aggressive case, pruning 98% of all feedforward parameters results in $1.57\times$ faster translation with -1.4 BLEU loss. The choice between pruning a decoder or not is clearly in favour of doing it. At a similar quality loss, the speed-ups are $1.13\times$ and $1.29\times$ respectively for encoder only and both.

Next, I proceed with reinitialising these models with the quality analysis in Tab. 4.19. Most interestingly, the models trained from scratch achieve comparable quality to their pruned counterparts, despite previous experiments on English→German having a significant gap between those. 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. To check whether the data size affects the pruning outcome, I sampled 13M sentences from the corpus of 242M, the same amount as English→German) and repeated the experiment. Experimenting with that data, I came to similar conclusions, meaning that the Spanish subpar results are not related to architecture or data size. In the experiments so far, the quality of pruned models either surpasses or is at least as good as those trained from scratch. Thus, I conclude that in some cases, structural pruning serves as an architecture search method to find a Pareto optimal trade-off.

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 4.19: 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.

4.12 Neuron-level regularisation of both feedforward and attention layers

The extensive study in the previous section shows that most feedforward connections can be regularised and sliced out of a model, making it smaller and faster at negligible quality change. Given that attention layers have been unaltered so far, it is safe to assume that attention overtakes the brunt of the neural workload, allowing a network to keep the quality high. On the other hand, there is yet untapped optimisation potential in attention layers.

Due to how a transformer architecture is built like, individual connections cannot be easily sliced out of attention layers, unless whole heads are removed. Still, I proceed with regularising rows and columns in attention layers. Next, I approximate the desire of a model to remove a particular attention head based on how many connections got sparsified. In the end, an entire head gets sliced if at least half of its connections are dead (the sum of each row/column being less than $1\mathrm{e}{-5}$). I experimented with English→German using the "6–2 tied" architecture. As it is the first time pruning attention on top of feedforward layers, I once more investigate the bottleneck by regularising the encoder only as well as both encoder and decoder. I use $\lambda \in \{0.2, 0.3, 0.4, 0.5, 0.7\}$ and prune in three stages as usual: pretraining for $25k$ batches, regularising for $250k$ batches with convergence after slicing. The results are presented in Tab. 4.20.

Because of lenient thresholding combined with a non-coarse regularisation of attention layers, the pruning of heads is not as invasive as neuron pruning in feedforward layers. Nonetheless, up to half of all heads were removed from the models in these experiments. In terms of quality, both approaches of regularising only the encoder (Tab. 4.20b) and regularising the whole model (Tab. 4.20a) rank on a similar BLEU level. The models with pruned decoders achieve a slightly higher speed-up in larger λ experiments as they reluctantly start stripping feedforward connections from the decoder.

Reg. $\lambda \longrightarrow$		Base		0.2		0.3		0.4		0.5		0.7	
Layer type $\longrightarrow$		FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads
Dimension / Heads	Enc. 1	1536	8	738	6	238	6	229	5	151	5	88	3
	Enc. 2	1536	8	1180	2	471	3	412	2	277	2	183	2
	Enc. 3	1536	8	1429	4	538	2	674	3	470	3	284	3
	Enc. 4	1536	8	1342	4	732	4	532	4	358	4	214	4
	Enc. 5	1536	8	1270	4	1207	5	554	3	389	3	253	3
	Enc. 6	1536	8	1517	7	1331	5	753	6	557	5	378	5
	Dec. 1	1536	8	1536	8	1530	8	1423	8	1131	8	728	8
BLEU	WMT16–19		37.6		37.7		37.4		37.1		36.8		36.0
	Δ		—		0.1		-0.2		-0.5		-0.8		-1.6
Efficiency	Att. sparsity		0%		38%		41%		45%		46%		50%
	FFN sparsity		0%		16%		44%		57%		69%		80%
	WPS		2404		2605		2723		2896		3089		3531
	Speed-up		1.00		1.08		1.13		1.20		1.28		1.47

(a) With encoder and decoder pruned.

Reg. $\lambda \longrightarrow$		Base		0.2		0.3		0.4		0.5		0.7	
Layer type $\longrightarrow$		FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads
Dimension / Heads	Enc. 1	1536	8	698	6	368	5	209	5	141	5	82	3
	Enc. 2	1536	8	1152	2	620	2	393	2	274	2	172	2
	Enc. 3	1536	8	1426	4	953	4	621	4	437	3	232	3
	Enc. 4	1536	8	1327	4	808	4	482	4	320	4	192	4
	Enc. 5	1536	8	1237	6	768	4	490	3	344	3	204	3
	Enc. 6	1536	8	1508	7	1065	6	687	5	488	5	302	4
	Dec. 1	1536	8	1536	8	1536	8	1536	8	1536	8	1536	8
BLEU	WMT16–19		37.6		37.8		37.5		37.1		36.7		36.0
	Δ		—		0.2		-0.1		-0.5		-0.9		-1.6
Efficiency	Att. sparsity		0%		34%		41%		45%		46%		52%
	FFN sparsity		0%		17%		43%		59%		67%		75%
	WPS		2404		2609	2933			3013		3155		3336
	Speed-up		1.00		1.09		1.22		1.25		1.31		1.39

(b) With encoder pruned.

Table 4.20: “6–2 tied” transformer models for English→German with feedforward connections and attention heads pruned.

The most striking observation is that not a single head gets removed from the context attention in all these experiments. This outcome identifies the true bottleneck in a model, the context attention, which serves as a bridge between an encoder and decoder. Thus, removing parameters from the only layer performing this function could directly prevent a model from being able to successfully translate *at all*. This may not a problem with many decoder layers to prune from, but it becomes an issue in shallow decoder architectures with a single layer.

Overall, the regulariser can remove about 40% of encoder parameters at a small change of -0.1 BLEU and make inference 22% faster. Being harsher than that, it can push speed-up up to 47% at the cost of 1.6 BLEU by removing half of the attention heads and 80% of feedforward parameters. These results are quite promising, but the quality loss is still substantial for more drastic optimisation. A tied decoder is favoured as it allows to fit a model into a cache, but it still stacks N repeated layers (2 in the case of “6–2 tied”). Ideally, a model should have a small shallow decoder from the beginning so that we can shift our focus to prune an encoder part of the neural network solely. Unfortunately, a “6–6” architecture outperforms “6–2 tied” quality-wise as the consensus is that the more parameters, the better. Naturally, more layers mean slower translation, but those who prioritise quality may be unwilling to compromise on it even with a faster inference.

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” layer proportions perform as good as “6–6”. As already demonstrated, pruning an already reduced decoder may cause a bottleneck that damages quality too much. However, if most of the workload gets shifted into an encoder, we can focus on pruning it exclusively. To check whether this claim is true, I train “12–1” architecture and compare its quality with “6–6” in Tab.4.22.

Indeed, a simple shift of layers from the decoder to the encoder caused -0.3 BLEU in damage but at the same time made inference $1.58\times$ faster. This trade-off is a competitive option to consider when training models for deployment. Thus, I will concentrate on this architecture in the next pruning experiments. The results on pruning only the encoder are in Tab. 4.21.

Now that there are 12 layers to prune from, the potential speed-up is even larger. Removing 57% of attention heads and 87% of feedforward connections leads to -1.2 BLEU loss, but a model translates $1.79\times$ faster. It is the largest ratio jump in speed compared to an unpruned baseline there has been so far. One of the less pruned models offers a satisfying trade-off of 0.3 BLEU in quality for 51% faster translation. This specific model removed half of the attention heads and two-thirds of feedforward parameters.

Reg. $\lambda \longrightarrow$		Base		0.2		0.3		0.4		0.5		0.7	
Layer type $\longrightarrow$		FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads
Dimension / Heads	Enc. L1	1536	8	728	5	414	5	250	4	183	4	108	4
	Enc. L2	1536	8	927	3	619	2	436	2	325	2	202	2
	Enc. L3	1536	8	540	4	338	4	222	4	173	4	105	3
	Enc. L4	1536	8	415	4	250	3	166	3	123	3	77	3
	Enc. L5	1536	8	429	5	255	4	167	4	116	5	66	5
	Enc. L6	1536	8	382	4	191	4	123	4	89	3	60	3
	Enc. L7	1536	8	334	4	138	4	81	4	50	4	30	4
	Enc. L8	1536	8	297	3	129	1	69	1	50	1	19	1
	Enc. L9	1536	8	321	3	135	1	69	1	44	3	28	3
	Enc. L10	1536	8	319	3	174	2	117	1	88	1	48	1
	Enc. L11	1536	8	474	4	298	4	214	4	165	3	112	2
	Enc. L12	1536	8	635	4	376	4	264	4	184	4	114	4
	Dec. L1	1536	8	1536	8	1536	8	1536	8	1536	8	1536	8
BLEU	WMT16–19		38.2		37.9		37.3		37.0		37.0		36.6
	Δ		—		-0.3		-0.9		-1.2		-1.2		-1.6
Efficiency	Att. sparsity		0%		48%		56%		58%		57%		59%
	FFN sparsity		0%		63%		76%		81%		84%		87%
	WPS		1930		2918		3029		3430		3446		3485
	Speed-up		1.00		1.51		1.57		1.78		1.79		1.81

Table 4.21: “12–1” transformer models for English→German with feedforward connections and attention heads.

Architecture	BLEU	COMET	WPS
"6–6"	38.5	54.8	1225
"12–1"	38.2	49.5	1930

Table 4.22: A quality comparison between "6–6" and "12–1"' architectures for English→German.

	Reg. $\lambda \longrightarrow$	Base	0.2	0.3	0.4	0.5	0.7
Quality	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
Efficiency	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 4.23: Quality evaluation of "12–1" tiny transformer models for English→German with feedforward connections in *encoder and decoder* pruned. Averaged over WMT16–19 testsets.

Let us go ahead with the extended quality analysis in Tab. 4.23. It turns out that there is a considerable drop in COMET between the "6–6" and "12–1" architectures, with the former scoring 54.8 points and the latter getting 49.5. This gap raises the question of how valid is the hypothesis stated by Kasai et al. (2020), especially when human evaluators judge models. I leave this question open for future work.

4.13 Head-level regularisation of attention layers

So far, I have regularised individual neurons in feedforward and attention layers. While it is easy to slice those out from the former, it is not as trivial in the latter case. I removed entire heads in the previous experiments if at least half of their connections were dead. Then, inactive neurons in the remaining heads are quickly reactivated during a convergence phase to get fully utilised again. Y. Wang et al. (2020) shows that *rejuvenating* parameters after they were pruned allows a model to achieve higher BLEU quality and may serve as a method to prevent overfitting. My pruning approach for attention is a heuristic with a custom threshold as a model does not make an explicit decision to remove a specific head all at once. At the same time, it is easier for a model to give up single connections instead of larger structures as it could lead to sudden quality degradation. If we zero out connections in an attention head but decide to keep the entire head, in the end, those connections will get rejuvenated once more when the model is allowed to converge.

The *groups* in a group lasso have been of the same size until now: all rows and columns pruned from a model were $1 \times N$ with N representing a hidden dimension of a model. In the knowledge-distilled students, it would be 1×256. Equal size of groups means that all of them are penalised on the same scale. The more generalisable variant of group lasso *orthonormalises* groups of different sizes by scaling them with a ν scalar (Simon & Tibshirani, 2012; Yuan & Lin, 2006).

Given parameters w split into groups G, the scaled version of a group lasso penalty is defined as:

$$R(w) = \sum_{g=1}^{|G|} \nu \, \|w_g\|_2 = \sum_{g=1}^{|G|} \nu \sqrt{\sum_{j=1}^{|G_g|} \left(w_g^j\right)^2}. \tag{4.1}$$

Usually, we scale by the number of covariants in a group. I chose to scale by the inner dimension of a parameter matrix as it is the main difference between groups: $\nu = \sqrt{d_g}$. When regularising only rows and columns, I left $\nu = 1$ as it would be the same value everywhere and can be thus included in the global λ scalar.

Now, I proceed with the experiment on the regularisation applied to connections in feedforward layers, and entire attention heads *together* to compare it with the exclusively neuron-wise approach so far. I train Estonian→English models with a “6–2” architecture to diversify experiments. As there are two decoder layers instead of one, I regularise the entire model to gauge the potential impact of a bottleneck and see how context attention behaves under these pruning circumstances. I do not sweep hyperparameters, opting for $\lambda = 0.3$ as it provided a middle-ground sparsity in the previous experiments. Additionally, I reinitialise these models and train them from scratch to compare with the pruned ones. The results are presented in Tab. 4.24.

As aforementioned, I compare two pruning approaches:

- regularising individual connections and then removing heads with more than half of connections inactive (rowcol + rowcol = rows/columns in both feedforward and attention layers)
- regularising entire heads with group lasso (rowcol + heads = rows/columns in feedforward layers, blockwise heads in attention)

Due to how the penalty is scaled with γ in Eq. 4.1, the regularisation of entire heads is much more aggressive towards them, removing some layers entirely, which get skipped 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 a 0.1 BLEU difference, the same model loses 2.3 COMET points, further validating that training from scratch is subpar. Those results show potential in regularising larger structures and even entire layers as a way of architecture searching. I leave the improvement of the method for the upcoming chapter.

			Base		rowcol + heads		rowcol + rowcol	
Model ⟶								
Layer type ⟶			FFN	Heads	FFN	Heads	FFN	Heads
Dimension / Heads	Enc. L1		1536	8	579	0	210	7
	Enc. L2		1536	8	793	1	552	1
	Enc. L3		1536	8	959	0	712	3
	Enc. L4		1536	8	913	0	459	6
	Enc. L5		1536	8	1212	3	708	4
	Enc. L6		1536	8	1523	2	1033	8
	Dec. L1		1536	8	1536	2	1535	7
	Dec. L2		1536	8	1536	7	1536	8
Quality	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	
	Sparsity		0%	0%	26%	77%	45%	31%
	WPS		1414		2012		1614	
	Speed-up		1.00		1.42		1.14	

Table 4.24: 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*).

4.14 Analysis: Parameter distribution

This section looks into the distribution of parameters in models left after pruning and analyses general sparsity patterns emerging across different layers.

Fig. 4.5 visualises the reduction of feedforward dimensions as λ increases in the neuron-level regularisation done on the English→German “6–2 tied” architecture. As observed before, the only decoder layer is a bottleneck and is the least prioritised in pruning. The opposite is true for the first encoder layer, as it is the most pruned. There is an emerging pattern of each next layer in the encoder being less and less sparse.

To further analyse these patterns, I train Spanish→English models with the “6–6” architecture, in which I regularise individual connections in both feedforward and attention layers. The models are trained with $\lambda \in \{0.1, 0.2, 0.3, 0.4\}$ for $250k$ updates. Then, I account for still active connections. Fig. 4.6 presents the histogram of parameters per layer in the baseline and pruned models. Since SSRU (Y. J. Kim et al., 2019) replaces the decoder self-attention, I only show two “pairs” of parameters: encoder self-attention and decoder context attention, with their feedforward counterparts.

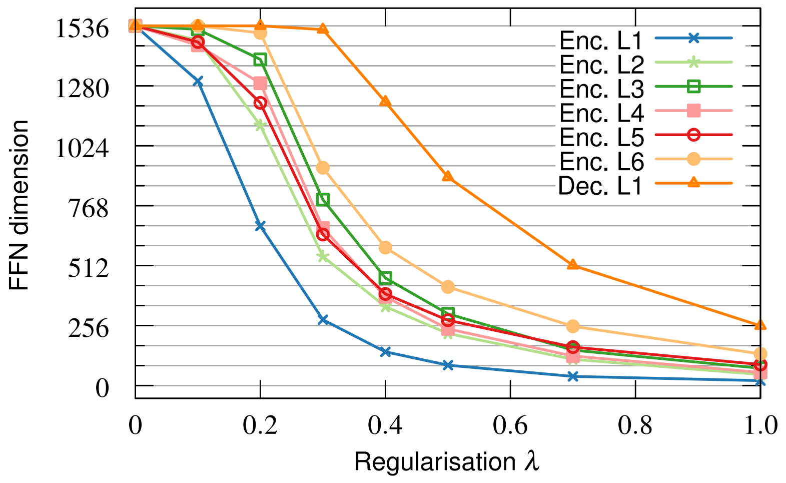


Figure 4.5: Pruning of FFN layers in English→German student models.

The research shows that feedforward parameters also seem to follow the same pattern on the top of attention. The last decoder layer is the least pruned across the whole architecture, which makes sense because this is the last layer before producing the output. The feedforward and attention layers appear to be tied together even though regularisation itself does not enforce it. Since attention has a more defined role, feedforward parameters probably just complement it. There seems to be no need for model layers where feedforward parameters perform their work alone. This is similar to findings from Bogoychev (2021).

Both encoder layer types follow a similar sparsity pattern making a “U-shape”, with the second and third layers 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 my previous work Behnke and Heafield (2020) on lottery ticket pruning of attention heads. As could be previously seen in Fig. 3.11, the attention in the encoder also prunes middle layers, and the context attention retains more heads in further layers. This specific distribution strongly indicates that the decoder prefers to attend to itself first and confront context later. The Estonian models, in which I regularise and prune entire attention heads (see Sect. 4.13), exhibit a roughly similar pattern, which is a signal that structural group lasso pruning with its architecture search may have a broader generalisation. Furthermore, a model may seek to remove entire transformer layers. I leave the exploration of that for the next chapter.

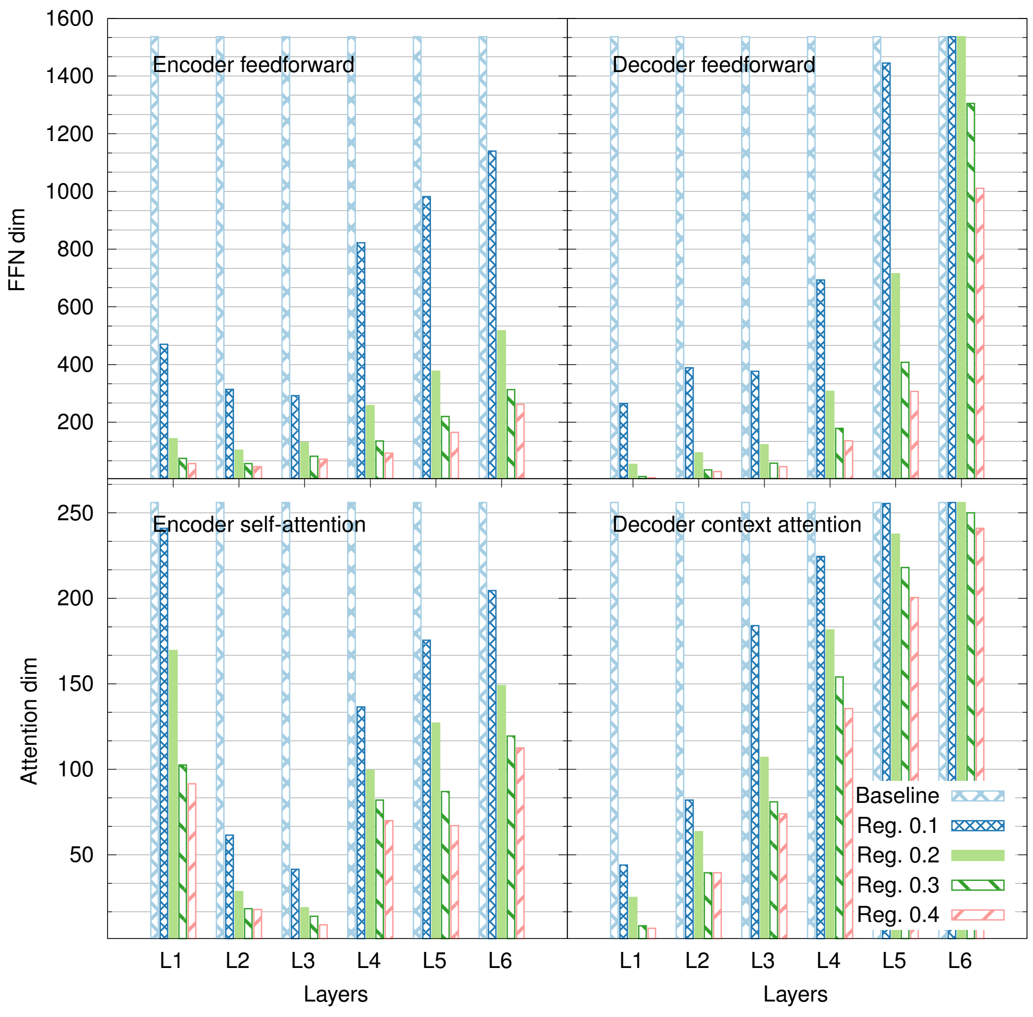


Figure 4.6: The distribution of feedforward and attention connections left in Spanish→English "6–6" architectures when regularised on a neuron level.

4.15 Analysis: Pareto trade-off

Throughout this chapter, I have run numerous experiments on English→German with various architectural setups and training regimes. I noted the achieved quality and translation speed expressed in words per second in each instance. Additionally, I highlighted how faster a model was than its corresponding baseline. Also, I trained many models from scratch to show that pruning outperforms it. In the end, optimisation always means trading-off quality for efficiency. An NMT model is *optimal in a Pareto sense* when there is no other model that produces higher quality while being faster at the same time. In practice, there is no single "best" model but rather a set of experiment points representing the *Pareto frontier*.

	Model ⟶	base	base ½	base ¼	base ⅛	base ¹⁄₁₆
	Enc. L1–L6	1536	768	384	192	96
	Dec. L1	1536	768	384	192	96
Quality	BLEU	37.6	37.0	36.6	35.8	35.8
	Δ	—	-0.5	-1.0	-1.8	-1.8
	chrF	63.3	63.3	62.9	62.3	62.4
	COMET	49.7	48.6	47.7	44.1	44.5
Efficiency	Sparsity	0%	50%	75%	88%	94%
	WPS	2404	2964	3296	3444	3586
	Speed-up	1.00	1.23	1.37	1.43	1.49

Table 4.25: The evaluation of English→German "6–2 tied" baselines averaged over WMT16–19 testsets.

	Model ⟶	base	base ½	base ¼	base ⅛	base ¹⁄₁₆
	Enc. L1–L6	1536	768	384	192	96
	Dec. L1–L6	1536	768	384	192	96
Quality	BLEU	38.5	37.8	37.7	37.15	37.1
	Δ	—	-0.7	-0.8	-1.3	-1.4
	chrF	64.2	64	63.9	63.3	63.3
	COMET	54.8	53.6	52.9	50.5	50.3
Efficiency	Sparsity	0%	50%	75%	88%	94%
	WPS	1225	1568	1740	1771	1932
	Speed-up	1.00	1.28	1.42	1.45	1.58

Table 4.26: The evaluation of English→German "6–2 tied" baselines.

	Model ⟶	base		base ½		base ¼		base ⅛	
	Layer type ⟶	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads
	Enc. L1–L12	1536	8	768	4	384	4	192	4
	Dec. L1	1536	8	1536	8	1536	8	1536	8
Quality	BLEU	38.2		36.7		36.7		35.7	
	Δ	—		-1.5		-1.5		-2.5	
	chrF	63.9		63		62.6		62.3	
	COMET	49.5		48.3		45.9		44.4	
Efficiency	FFN sparsity	0%		50%		75%		88%	
	Att. sparsity	0%		50%		50%		50%	
	WPS	1930		3171		3241		3532	
	Speed-up	1.00		1.64		1.68		1.83	

Table 4.27: The evaluation of English→German "12–1" baselines averaged over WMT16–19 testsets.

I trained several simple baselines with reduced dimensions to provide an unbiased comparison in the competition. It is an extremely easy method to get a smaller and more robust model from the beginning. I reduce feedforward dimensions uniformly to $\{768, 384, 192, 96\}$ in both encoder and decoder "6–2 tied" and "6–6" architectures. For "12-1" models, I reduce feedforward layers to $\{768, 384, 192\}$ in the encoder and additionally halve encoder heads from 8 to 4 to reflect the sparsity percentages of pruned models roughly. The baseline results are presented in Tab. 4.25, 4.26 and 4.27.

Finally, I pit against each other the said baselines, the pruned models and their reinitialised counterparts. The Pareto trade-off is visualised in Fig. 4.7. It includes every pruned model for specific architectures grouped in the “Pruned” labels. All the baselines and their reinitialised models are rendered in “Baselines” and “Reinits” respectively. The data points representing the fastest models at given quality are circled in the “Pareto optimal” category. In theory, the best possible outcome would be in the upper-right corner of the plot.

The points scattered across the left side of the plot are the “6–6” models. This architecture offers the best quality at slower inference, with some gain when pruned. Eventually, it is better to switch to an architecture with a shallow decoder instead of continuing to prune the “6–6” layers. The points scattered to the right side are the architectures with a shallow decoder: “6–2 tied” and “12–1”. The Pareto plot shows that pruning a deep encoder in “12–1” aggressively offers a superior trade-off than “6–2 tied” pruned in various ways. All the reinitialised models are subpar and perform below the optimal frontier. The only noteworthy optimal baseline is the unpruned “12–1” model placed in the middle of the plot. Overall, the “12–1” architecture is the best choice to achieve the best speed performance, especially when pruned. If quality is the priority instead, a deeper decoder is required, pruned or not.

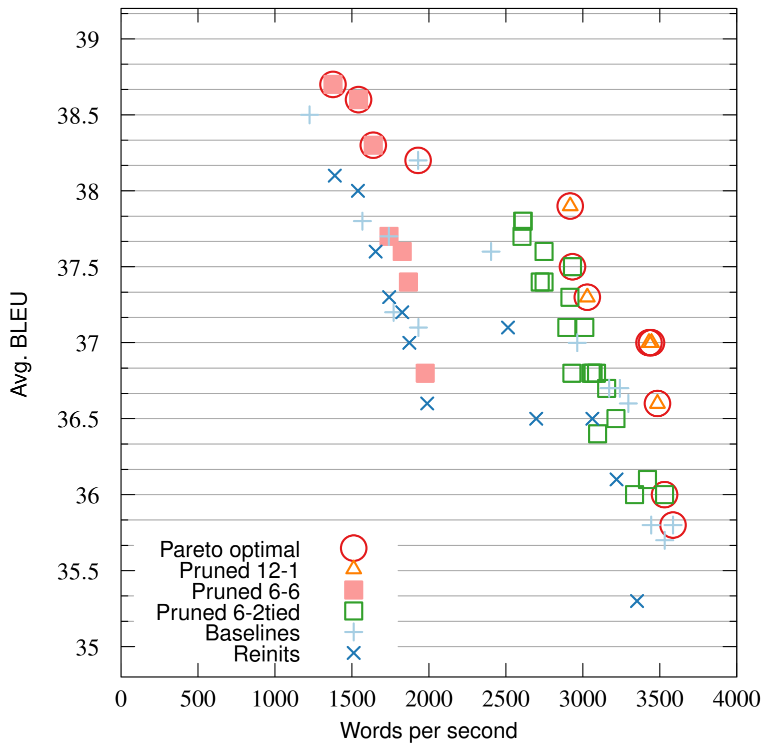


Figure 4.7: Pareto trade-off between average quality in BLEU and average translation time for English→German students of different architectures.

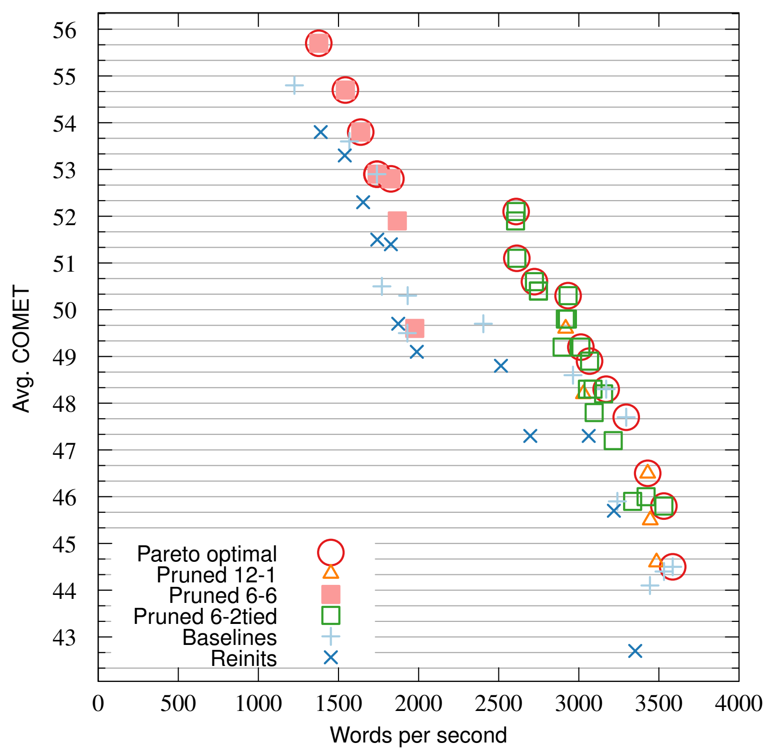


Figure 4.8: Pareto trade-off between average COMET quality and average translation time for English→German students of different architectures.

As I mentioned before in this thesis, BLEU alone is not the best indicator of good performance. Thus, I proceed with another Pareto analysis with COMET scores instead. It is presented in Fig. 4.8.

The consensus remains: pruned models lead the Pareto frontier and provide the best quality-speed trade-off. In contrast to Fig. 4.7, “12–1” architectures are outperformed by “6-2 tied” in terms of quality. This difference can be attributed to the quality gap in COMET shown in Tab. 4.22. These results serve as a reminder to not depend on just BLEU scores. Ideally, researchers should use multiple scoring systems alongside human evaluation before making a deployment decision.

4.16 Efficiency Shared Task (WMT2021)

To put my pruning method to the final test, I participated in the WMT2021 Efficiency Shared Task[6] as a researcher who established the direction of the submission (Behnke et al., 2021).

6. `http://www.statmt.org/wmt21/efficiency-task.html`

The WMT21 efficiency shared task consists of two sub-tasks: throughput and latency. Systems should translate English to German under the constrained conditions of the WMT21 news task. For each task, systems are provided 1 million lines of raw English input with at most 150 space-separated words. The throughput task receives this input directly. The latency task, introduced this year, is fed input one sentence at a time, waiting for the translation output before providing the next sentence. Throughput is measured on multi-core CPU or GPU systems, and latency is measured on single-core CPU or GPU systems. Entries to both tasks are measured on quality, approximated via BLEU score, speed, model size, Docker image size, and memory consumption. We did not optimise specifically for the latency task beyond configuring the relevant batch sizes to one.

This section shows how my research has aided the development of this efficiency shared task submission. We combined knowledge distillation, structural pruning and quantisation to achieve the state-of-the-art English→German models, providing a proven recipe for optimised NMT. In our models, I applied both neuron-level and head-level regularisation to prune feedforward and attention layers structurally. Models were sliced and collapsed, resulting in smaller and more robust architectures.

4.16.1 Setup

Our submission builds upon the work of last year's submission (Bogoychev et al., 2020). We trained our models in a teacher-student setting (Y. Kim & Rush, 2016b), using Edinburgh's En-De system submitted to the WMT2021 news translation task as the teacher model. For the students, we used a Simpler Simple Recurrent Unit (SSRU) Y. J. Kim et al. (2019) decoder, used a target vocabulary shortlist, and experimented with pruning the student models by removing component- and block-level parameters to improve speed. We experimented with quantising into smaller numerical formats, including fixed point 8-bit quantisation on the CPU.

Teachers

We used Edinburgh's English↔German systems submitted to the WMT2021 news translation task as teacher models (Chen et al., 2021). We trained transformer-big models Vaswani et al. (2017b), using a shared 32K SentencePiece (Kudo & Richardson, 2018) vocabulary, built in three stages: corpus filtering, back-translation and fine-tuning. The models achieved 29.90 and 51.78 BLEU on English→German and German→English WMT 2021 test, respectively (scored by the task organisers, with multiple references).

We used sequence-level knowledge distillation Y. Kim and Rush (2016b) to synthesise forward and backward translations using the teachers. We filtered the synthesised parallel data using handcrafted rules[7], followed by removing the bottom 5% according to cross-entropy per word on the generated side using KenLM (Heafield, Pouzyrevsky, Clark, & Koehn, 2013).

Students

We ran experiments using different combinations of teacher-synthesised corpora. The variant I used for experiments in this section included the synthesised data: parallel, monolingual backward and forward. All student models were trained with a validation set consisting of the subset of sentences in the English-German WMT test sets from 2015–2019 that were originally in English. The training concluded after reaching 20 consecutive validations without improving the BLEU score. The student models used the same shared vocabulary as the teacher ensemble. During decoding, we used a lexical shortlist (Devlin et al., 2014; Le, Allauzen, & Yvon, 2012; Schwenk, R. Costa-jussà, & R. Fonollosa, 2007) of the top 50 most probable alignments, combined through a union with the top 50 most frequent vocabulary items. Other than this, we used the default training hyperparameters from Marian for the base transformer model.

4.16.2 Experiment: Building the baselines

Each of the student models used transformer encoders (Vaswani et al., 2017b) and RNN-based decoders with SSRU attention Y. J. Kim et al. (2019). Several different architectures were explored; these differ in the number of encoder and decoder blocks as well as in the sizes of the embedding and FFN layers. Further to this, some of our transformer architectures use a modified attention matrix of shape $(d_{\text{emb}}, n_{\text{head}} \times d_{\text{head}})$ rather than the typical $(d_{\text{emb}}, d_{\text{emb}})$. In all cases, we use 8 transformer heads per layer and set $d_{\text{head}} = 32$ across all modified attention models.

As evident in my research on pruning and finding the Pareto optimal machine translation architectures, a “12–1” tiny transformer performed quite promisingly. Following that direction, I have selected a variety of student architectures to train for the shared task. I focused mostly on “12–1” but also tried “6–2” and tied variants such as “8–4 tied” and ”6–2 tied”. Depending on embedding and feedforward dimensions, each architecture is defined as either *large*, *base*, *tiny* or *micro*. All the student models are summarised in Tab. 4.28. On the newest WMT21 testset, the students are within -0.1 to -1.4 BLEU difference towards the ensembled teacher.

7. `https://github.com/browsermt/students/tree/master/train-student/clean`

	Depth		Dimensions						BLEU		COMET		
Model	Enc	Dec	Emb.	FFN	Att.	Heads	Params.	Size	WMT20	WMT21	WMT20	WMT21	Speed (s)
teacher x 3	6	6/6/8	1024	4096	1024	16	619.0M	2.30GB	38.3	28.8	56.8	50.8	-
12-1.large	12	1	1024	3072	256	8	130.5M	498MB	37.6	28.7	54.0	47.7	92.2
12-1.base	12	1	512	2048	256	8	51.1M	195MB	36.7	28.2	50.7	44.1	38.9
12-1.tiny	12	1	256	1536	256	8	22.0M	85MB	36.1	27.6	48.2	41.9	19.2
12-1.micro	12	1	256	1024	256	8	18.6M	72MB	35.4	27.6	46.2	40.2	17.1
8-4.tied.tiny	8	4	256	1536	256	8	17.8M	69MB	35.7	27.8	50.3	43.9	30.4
6-2.tied.tiny	6	2	256	1536	256	8	15.7M	61MB	34.9	27.4	47.4	42.1	18.6
6-2.base	6	2	512	2048	512	8	42.7M	163MB	37.7	28.7	54.3	48.5	56.2
6-2.tiny	6	2	256	1536	256	8	16.9M	65MB	35.8	27.4	50.2	44.5	19.2

Table 4.28: Architectures for the different student models. The number of encoder/decoder layers are reported with the size of the embedding, attention and FFN layers, the total number of parameters, the model size on disk, quality in both BLEU and COMET as well as speed on WMT21 testset. The first and second groups use a modified attention matrix shape, with second group consisting of tied models. The third group uses the typical shape attention matrices.

4.16.3 Experiment: Pruning and quantisation

Under the task constraints, we trained, pruned and quantised 12–1.tiny and 12–1.micro architectures. I tried two pruning settings, following the directions set in Sect. 4.13: *rowcol-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 were 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 possible. 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. 4.29.

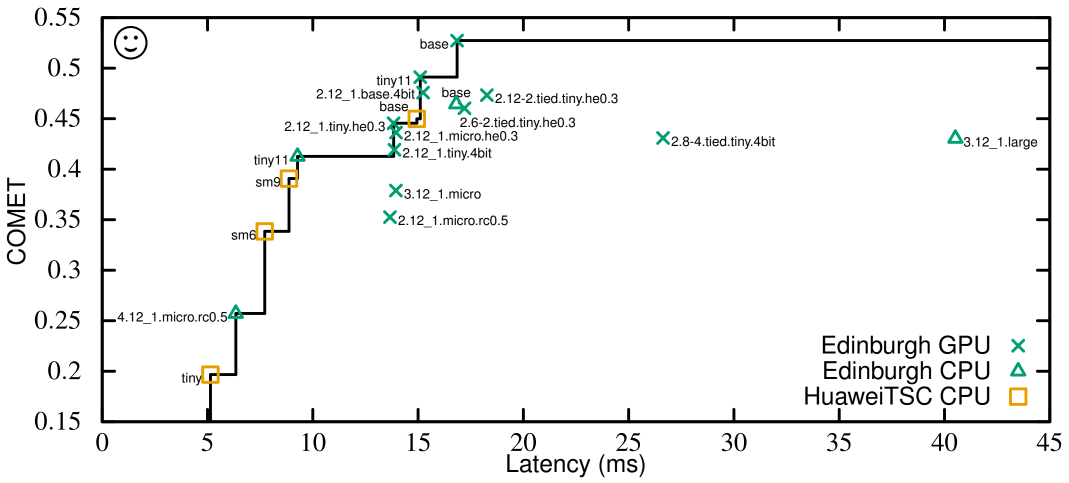
head-lasso left attention layers almost completely unpruned, focusing on removing connections from feedforward layers instead. It could be that $150k$ batches were not enough for this lambda to suppress larger structures of parameters. *rowcol-lasso* was much more aggressive in both layers at the cost of quality. To further optimise the models, they were quantised to 8-bit. However, 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–$1.7\times$ faster at 0.6–1.3 BLEU. With quantisation, those models are 1.9–$2.7\times$ faster, losing 0.9–1.7 BLEU compared to the unpruned and unquantised baselines.

	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

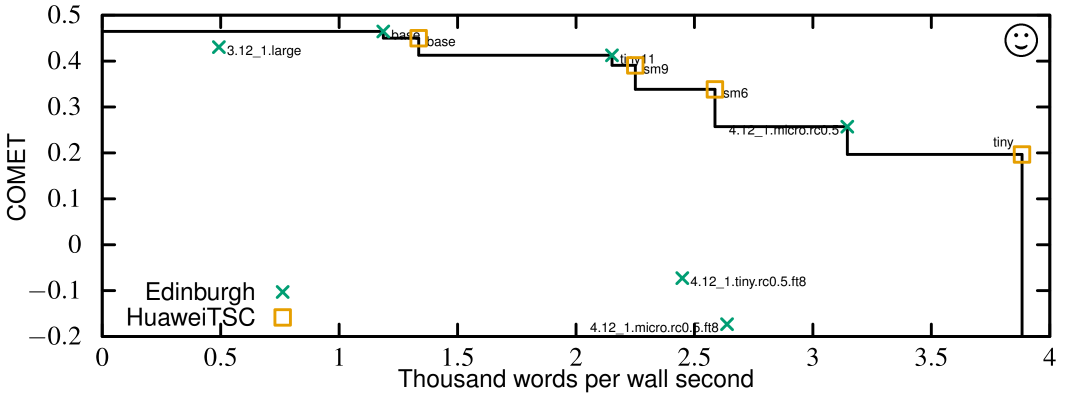
Table 4.29: 8bit model performance. BLEU score is calculated from WMT20. Speed is measured on a single core CPU with a mini-batch of 32.

4.16.4 Analysis: Pareto frontier

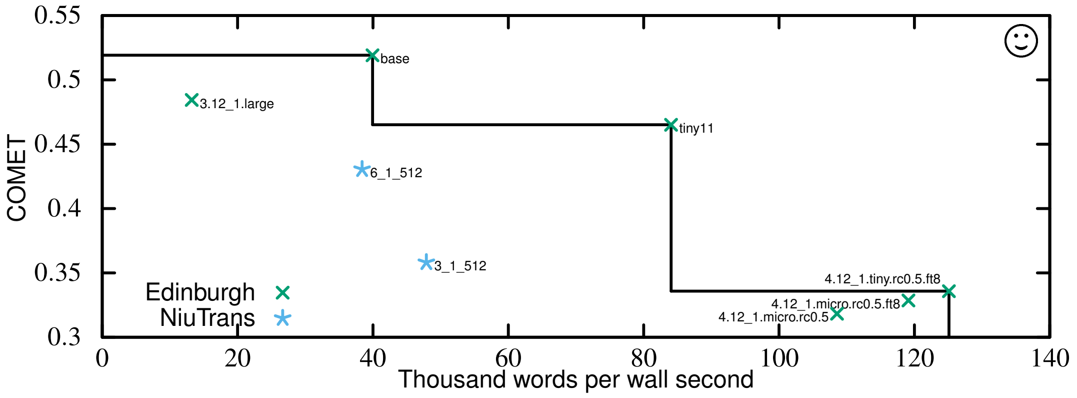
Fig. 4.9 present the Pareto trade-offs for latency and throughput tasks, which include some of the models mentioned above and a few more as well. Many of our submissions, including those pruned, landed in the Pareto frontier. An interesting example of how pruning and quantisation create a winning combination is the difference between Niutrans and Edinburgh models in Fig. 4.9c. “3-1-512” submission by Niutrans means that this architecture has 3 encoders layers, a 1 decoder layer with feedforward dimensions set to 512 (C. Wang et al., 2021). Despite being an overall shallow model with only 4 layers, our aggressively pruned ‘12–1” models outperform it in speed. Of course, many other aspects differentiate these submissions, such as toolkits used, code-level optimisation, et cetera. Still, given the number of layers, the performance gap is interesting.



(a) Latency of combined CPU and GPU systems with COMET scores.



(b) Speed on 1 Core with COMET for all systems.

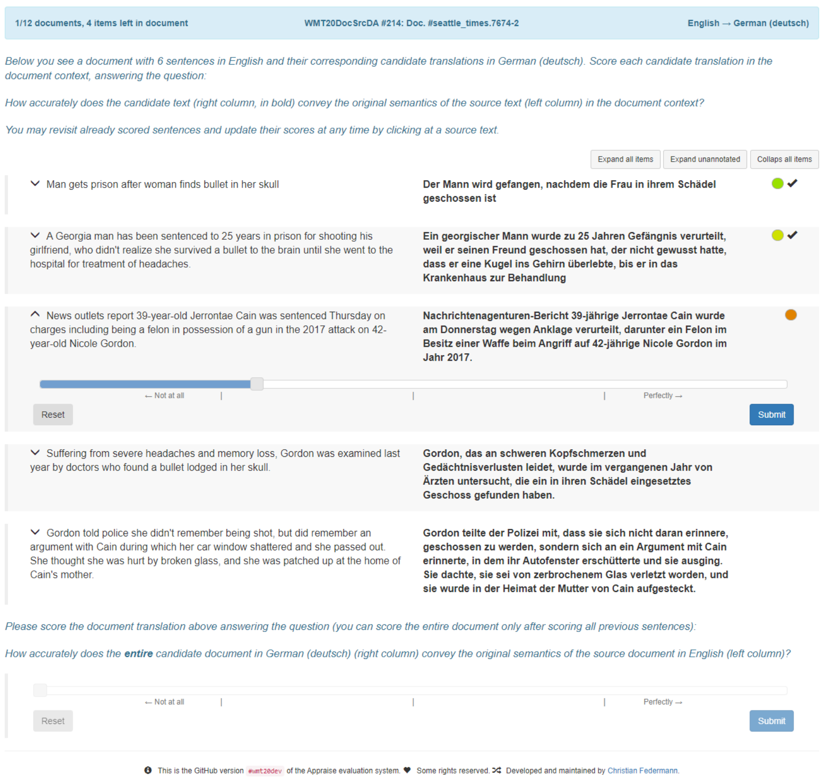


(c) Speed on 36 Cores with COMET for all systems.

Figure 4.9: Pareto trade-offs of the official WMT2021 Efficiency Task submissions.

4.17 Analysis: Human evaluation

Every year, the WMT conference provides human evaluation for the news translation shared task; WMT2021 was no exception (Akhbardeh et al., 2021). The Efficiency Task at WMT2021 participated in the human evaluation as well, which was sponsored by Microsoft (Heafield, Zhu, & Grundkiewicz, 2021). This human evaluation process has been acknowledged as trustworthy method of quality judgement, treated as a golden standard in the research field (Kocmi et al., 2021). In order to assess the impact of pruning and quantisation on translation quality, I submitted selected models for this evaluation procedure. I chose models with an identical starting architecture with the same hyperparameters alongside those with additional pruning and quantisation. In particular, I chose the “12–1” tiny transformer (*12-1.tiny*) from Tab. 4.29 with both types of pruning applied and fine-tuned with an 8-bit quantisation. These models are now abbreviated to *pruned-heads* and *pruned-rowcol* with the *ft8bit* suffix when the quantisation is applied.



1/12 documents, 4 items left in document — WMT20DocSrcDA #214: Doc. #seattle_times.7674-2 — English → German (deutsch)

Below you see a document with 6 sentences in English and their corresponding candidate translations in German (deutsch). Score each candidate translation in the document context, answering the question:

How accurately does the candidate text (right column, in bold) convey the original semantics of the source text (left column) in the document context?

You may revisit already scored sentences and update their scores at any time by clicking at a source text.

Man gets prison after woman finds bullet in her skull	**Der Mann wird gefangen, nachdem die Frau in ihrem Schädel geschossen ist**	● ✓
A Georgia man has been sentenced to 25 years in prison for shooting his girlfriend, who didn't realize she survived a bullet to the brain until she went to the hospital for treatment of headaches.	**Ein georgischer Mann wurde zu 25 Jahren Gefängnis verurteilt, weil er seinen Freund geschossen hat, der nicht gewusst hatte, dass er eine Kugel ins Gehirn überlebte, bis er in das Krankenhaus zur Behandlung**	● ✓
News outlets report 39-year-old Jerrontae Cain was sentenced Thursday on charges including being a felon in possession of a gun in the 2017 attack on 42-year-old Nicole Gordon.	**Nachrichtenagenturen-Bericht 39-jährige Jerrontae Cain wurde am Donnerstag wegen Anklage verurteilt, darunter ein Felon im Besitz einer Waffe beim Angriff auf 42-jährige Nicole Gordon im Jahr 2017.**	●
Suffering from severe headaches and memory loss, Gordon was examined last year by doctors who found a bullet lodged in her skull.	**Gordon, das an schweren Kopfschmerzen und Gedächtnisverlusten leidet, wurde im vergangenen Jahr von Ärzten untersucht, die ein in ihren Schädel eingesetztes Geschoss gefunden haben.**	
Gordon told police she didn't remember being shot, but did remember an argument with Cain during which her car window shattered and she passed out. She thought she was hurt by broken glass, and she was patched up at the home of Cain's mother.	**Gordon teilte der Polizei mit, dass sie sich nicht daran erinnere, geschossen zu werden, sondern sich an ein Argument mit Cain erinnerte, in dem ihr Autofenster erschütterte und sie ausging. Sie dachte, sie sei von zerbrochenem Glas verletzt worden, und sie wurde in der Heimat der Mutter von Cain aufgesteckt.**	

Please score the document translation above answering the question (you can score the entire document only after scoring all previous sentences):

How accurately does the ***entire*** *candidate document in German (deutsch) (right column) convey the original semantics of the source document in English (left column)?*

This is the GitHub version #wmt20dev of the Appraise evaluation system. Some rights reserved. Developed and maintained by Christian Federmann.

Figure 4.10: The example of annotating environment for a standard DA in a human evaluation (Akhbardeh et al., 2021). Annotators move the slides left and right, reflecting the quality of given translations.

The human evaluation is performed through the direct assessment (DA). A standard DA judges the absolute quality of models and allows us to compare them with each other. This type of DA (Cettolo et al., 2017; Graham, Baldwin, Moffat, & Zobel, 2013) has been performed following the directions set by WMT2020 News Translation Task (Barrault et al., 2020). Annotators rate a translation on an analogue scale using a slider corresponding to an underlying absolute range between 0 and 100, representing how adequate a translation is when confronted with a source. These scores get averaged over sentences (*Ave.*). Various annotators may favour different scales when judging translations. For that reason, the scores are standardised (*Ave.* z). The context was provided for the annotators, with individual sentences shown chronologically. All annotators are bilingual native German speakers with a translation or linguistics background. Annotations were collected using Appraise[8] Federmann (2018). Fig. 4.10 shows an example of annotating procedure for a standard DA.

	Human Evaluation		COMET	Sparsity		
System	Ave. z	Ave.	WMT21	Att.	FFN	Speed (s)
12-1.tiny	0.130	94.6	41.9	0%	0%	19.2
12-1.tiny.pruned-heads	0.042	80.7	38.8	3%	75%	14.5
12-1.tiny.pruned-heads.ft8bit	0.036	76.2	33.0	3%	75%	9.3
12-1.tiny.pruned-rowcol	-0.100	83.5	34.2	68%	73%	11.6
12-1.tiny.pruned-rowcol.ft8bit	-0.044	85.7	31.3	68%	73%	7.1

Table 4.30: The human evaluation of WNGT2021 models based on a standard direct assessment.

The results based on the standard DA are presented in Tab. 4.30. Most surprisingly, the *rowcol* model outperforms the *heads* one despite being more aggressively pruned. The automatic metric, COMET on the WMT2021 testset, reflects the opposite. Another interesting observation of this model is that its quantised variant has a higher human-evaluated score than when it is not. This gap does exist for the *heads* experiments.

There are several reasons for such atypical results. A standard DA is a less sensitive method: unless the quality drop is large, it may be difficult to say with certainty which one is better. Independent annotators generally evaluate models on a different subset of sentences, and too much variability can cloud the interpretation.

For this reason, a pairwise contrastive DA has been performed as well. In this type of evaluation, the same annotator compares outputs from two systems (A and B) concurrently. The annotators do not change within a pair, which further reduces irregularities. The results based on the pairwise DA are presented in Tab. 4.31. To highlight how significant Δ between models is, a

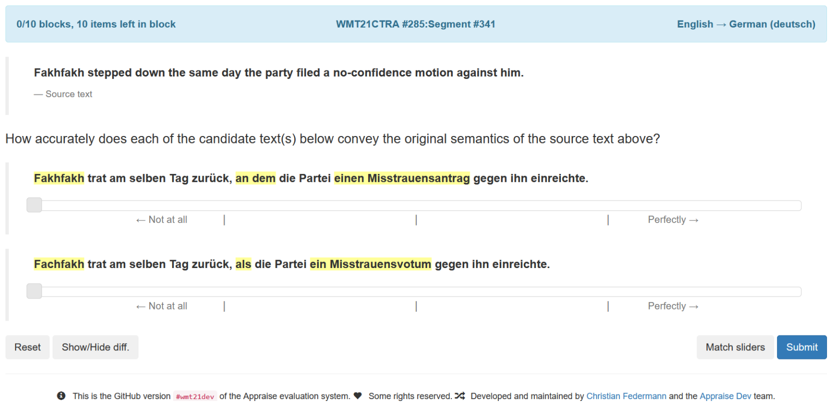
8. `https://github.com/AppraiseDev/Appraise`

Wilcoxon rank-sum test was carried out with $p < 0.05$ for $*$, $p < 0.01$ for $**$ and $p < 0.001$ for $***$. In simpler words, the more $*$ there are in Tab. 4.31, the more significant the quality gap between the models in a pair with a model A on the left being better than a B one. The example of annotating procedure for a pairwise contrastive DA is shown in Fig. 4.11.

When the pruned models are compared to the baseline, the human evaluation scores follow the pattern exhibited by the automatic metrics. The Δ for *rowcol* is larger than for *heads* (3.3 vs 1.1 in Δ). That makes sense, as the former one was more sparsified. Comparing the pruned models results in a similar conclusion: more pruning leads to worse human-perceived quality. Furthermore, we pit the standard pruned models against the quantised experiments. Overall, the quantised models are worse in quality at the cost of faster translation. However, the Δ between *rowcol* and its quantised counterpart is only 0.3. Such a small value does not necessarily indicate that those models are on a similar level. However, statistically speaking, it is difficult to say which one is better or worse based on their translation. The same comparison for *heads* yields $\Delta = 2.7$, which means there is a notable difference between the models with and without the quantisation, further supported by $p < 0.01$.

System A	System B	Ave. A	Ave. B	Δ	p-val
12-1.tiny	12-1.tiny.pruned-rowcol	93.3	90.0	3.3	***
12-1.tiny	12-1.tiny.pruned-heads	85.8	84.6	1.1	
12-1.tiny.pruned-heads	12-1.tiny.pruned-heads.ft8bit	82.0	79.3	2.7	**
12-1.tiny.pruned-rowcol	12-1.tiny.pruned-rowcol.ft8bit	95.3	94.9	0.3	
12-1.tiny.pruned-heads	12-1.tiny.pruned-rowcol	75.4	70.7	4.7	**
12-1.tiny.pruned-heads.ft8bit	12-1.tiny.pruned-rowcol.ft8bit	85.8	82.2	3.6	**

Table 4.31: The human evaluation of WNGT2021 models based on a pairwise direct assessment.


0/10 blocks, 10 items left in block WMT21CTRA #285:Segment #341 English → German (deutsch)

Fakhfakh stepped down the same day the party filed a no-confidence motion against him.

— Source text

How accurately does each of the candidate text(s) below convey the original semantics of the source text above?

Fakhfakh trat am selben Tag zurück, an dem die Partei einen Misstrauensantrag gegen ihn einreichte.

← Not at all | | | Perfectly →

Fachfakh trat am selben Tag zurück, als die Partei ein Misstrauensvotum gegen ihn einreichte.

← Not at all | | | Perfectly →

Reset Show/Hide diff. Match sliders Submit

This is the GitHub version #wmt21dev of the Appraise evaluation system. Some rights reserved. Developed and maintained by Christian Federmann and the Appraise Dev team.

Figure 4.11: The example of annotating environment for a pairwise contrastive DA in a human evaluation (Akhbardeh et al., 2021). Annotators move the slides left and right, reflecting the quality of given translations.

The automatic metrics such as COMET are the cheapest and the fastest option during the development. However, a human evaluation allows for the best analysis that reflects a user's experience, especially when analysing models for potential deployment.

4.18 Conclusions

This chapter investigated the structural pruning of a transformer architecture integrated into a typical training routine. The research focused on shedding feedforward nodes and whole attention heads as training progresses, and I achieved that through a group lasso regularisation, which zeroes parameters out structurally. The experiments on various knowledge-distilled robust models with deep and shallow decoders have shown that this type of pruning leads to Pareto optimal architectures in both quality and speed. In general, pruning entire heads on the top of feedforward nodes results in even faster models.

In contrast to the previous lottery ticket hypothesis method, the main advantage of this pruning approach is that it converges in a single "pass" just like a baseline. There is no need to repeat an entire or a part of training. Moreover, the resulting sparsity patterns exhibit similar distributions across different language pairs. Whether it is in an encoder or a decoder, the first and middle layers are priorities the most in pruning. On the other hand, the experiments on sparsifying both feedforward and attention layers reveal that some of them, such as the last context attention layer, distinctively avoid being pruned.

Among the English$\rightarrow$German experiments, the notable Pareto optimal examples include a model with 6 decoder layers being 34% faster at 0.2 BLEU loss and a model with "12–1" encoder-decoder ratio gaining an additional 51% speed-up costing only 0.3 BLEU point. This pruning approach combined with quantisation gives an even more significant speed boost. The pruned and quantised models trained for the efficiency shared task translate 1.9–$2.7\times$ faster at the cost of 0.9–1.7 BLEU.

The natural progression of this research would be to implement a matrix multiplication routine that allows the attention mechanism to have heads of different sizes. The alternative would be to improve the regularisation of whole heads, but experiments so far indicate that pruning prefers to have multiple heads of various but often small sizes. The group lasso regularisation already prunes some layers almost completely. Instead of leaving graph nodes with few parameters, it may be possible to skip them altogether. In the case of pruning both attention and feedforward, the algorithm could remove whole layers, focusing on those that are redundant. For example, Sajjad et al. (2020) remove entire layers in BERT; however, they require the model to be fully pretrained first.

The group lasso regularisation has been shown to work with block and row/column sparsity. Potentially, a different pattern shape could perform better quality-wise with proper speed optimisation. The group penalty applied to block sparsity currently prioritises pruning them within the same rows and columns. Applied to entire heads results in a sliceable architecture, but the quality may suffer as larger structures imposed in regularisation may damage quality. Given that specific layers get omitted during pruning and vice versa, I would like to explore the direction of forcing more sparsity in obsolete networks and reducing a regulariser's force for bottleneck layers.

Chapter 5

Improving Structural Pruning through Aided Regularisation

Group lasso regularisation has proved to be a reliable method of structural pruning when applied to a transformer, producing state-of-the-art Pareto optimal architectures in terms of translation quality and inference speed. So far in my research, I have worked on the most granular level that allows the effortless slicing of parameters while still maintaining the density of a model. Specifically, it means pruning on a neuron level in feedforward layers and on a head level in attention mechanisms.

The current pruning approach has several drawbacks. Firstly, regularising coarser structures forces a model to decide to zero out a large number of parameters simultaneously, leading to a higher quality drop. For example, removing entire layer at once damages quality more than doing so gradually neuron-wise. On the other hand, removing larger subnetworks makes a transformer even faster. Ideally, entire layers could be skipped for the most optimal outcome. In the research I have performed until now, there has been some indication that particular layers get actively more sparsified than others and vice versa. Those layers that are not eager to be pruned or even proactively refuse to do so may cause a bottleneck with quality worsening as a model is forced to balance perplexity and a regularisation penalty it cannot effectively decrease.

Typical regularisation does not differentiate between layers, and it is up to an engineer to pick which parts of a model to regularise or not. For example, I opted to not apply to apply regularisation to the decoder in “12–1” models. I empirically recognised that a single decoder layer is not worth the regularisation effort, given no sparsity was enforced onto it, which negatively impacted quality with no additional speed gain. The current research on regularisation lacks a more nuanced approach with distinct behaviour towards specific parts of a model. Most pruning methods externally evaluate parameters in a heuristic manner to then mask and exclude them from further training. A model is not informed of it beforehand, and it just accommodates around sudden removal of those parameters. Having a regulariser push and ease off certain layers to a varying degree with additional information plugged into its function is a research direction I plan to explore in this chapter.

This chapter introduces a method called aided regularisation that scales penalties for each layer based on either gradients or parameters. In Sect. 5.10, I provide a Pareto analysis of all variants of these new pruning methods, just like in the previous chapters, to show that aided regularisation results in models on the optimal frontier.

5.1 Motivation

Continuing the course set by research in the previous chapter, I look into improving the group lasso approach to push towards removing entire layers in the process. Rather than pruning whole layers in one go by deciding to keep them entirely or not, a regulariser serves as a halfway measure, with a portion of parameters removed from a layer. However, it could also be more strict if it recognises a layer to be obsolete during training. Then, it could aim to eradicate this layer from the architecture. Preferably, the regulariser should also be able to ignore those layers deemed too important to be pruned at all to maintain the highest translation quality possible. This effect could be achieved by scaling λ individually for each layer depending on some criteria. In other words, the scaling of λ directly modifies the actual regularisation function and its behaviour. Last but not least, the quality vs speed trade-off still has room for further improvement, and any advancement to the Pareto frontier is always welcome.

5.2 Analysis: Execution time of pruned graph nodes

Empirically, it has been shown that decoder calculations are more expensive than those in an encoder, with attention presumed to be a really expensive operation in a transformer. In this analysis, I shall look into these claims by directly measuring the execution time of nodes on a computational graph in a knowledge-distilled transformer.

To understand how dimension reduction affects the speed of individual matrix multiplications, I measure CPU time per operation on the graph. Since each operation is performed in a really short time, I aggregate all node calculations done within a layer and sum them together. For feedforward layers, that means assessing two affine operations with parameters (W_1, b_1) and (W_2, b_2). In an attention mechanism, I consider all operations performed on keys, queries and values (W_k, W_q, W_v with biases) as well as the final affine operation produced on the output (W_o, b_o).

All node execution times were measured with `std::chrono` C++ library on a single CPU thread, expressed in seconds. I start by evaluating encoder layers across "12–1" architectures, including the baseline and the pruned models on the WMT16 testset. The batch size is set to 32. Fig. 5.1a presents a total timing of a feedforward layer given its dimension up to 1536.

Fig. 5.1b does the same but for a self-attention layer given the number of heads in it. Both feedforward and self-attention layers in an encoder get faster *linearly* as the dimension shrinks. For instance, removing half of the neural connections from a feedforward layer makes it twice as fast.

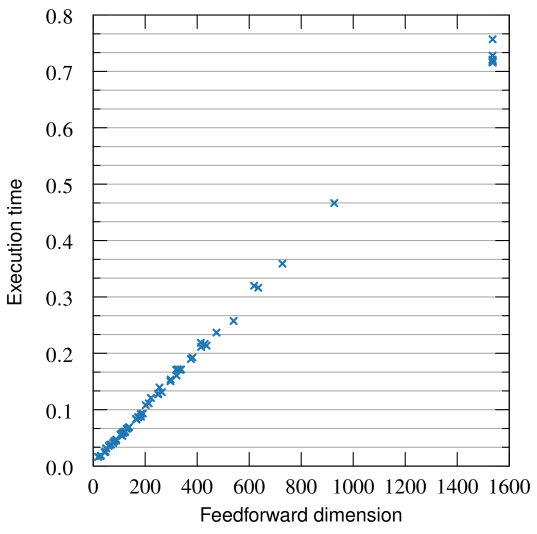
Regarding decoder analysis, I have omitted pruning context attention in my experiments as it affected the quality, with self-attention replaced by SSRU. Because of that, I analyse “6–6” architectures with pruned feedforward parameters instead. I accumulate every instance of each decoder layer executing calculations during a decoding loop and present the results in Fig. 5.1c. There is also a linear decrease of time utilised per layer in this case. A decoder feedforward layer takes twice as much time as the encoder ($1.6s$ vs $0.8s$ for the baseline dimensions of 1536). The decoder is run many times, which makes $2\times$ time cost not as large as it may seem. This efficiency is most probably due to repeated operations held in a CPU cache.

The most interesting part of this analysis is that a feedforward layer is almost twice as expensive to run in contrast to an attention layer in an encoder. Even though it is indeed true that attention in itself is expensive, especially the context one, which loops alongside the decoder, these results show that feedforward optimisation is important to explore. In my regularisation experiments, I first targeted feedforward layers, and it turns out it was the right path to progress. Naturally, pruning of both layers gives the fastest results, as evident in the Pareto frontier, but sparsifying just feedforward parameters also significantly boosts inference speed.

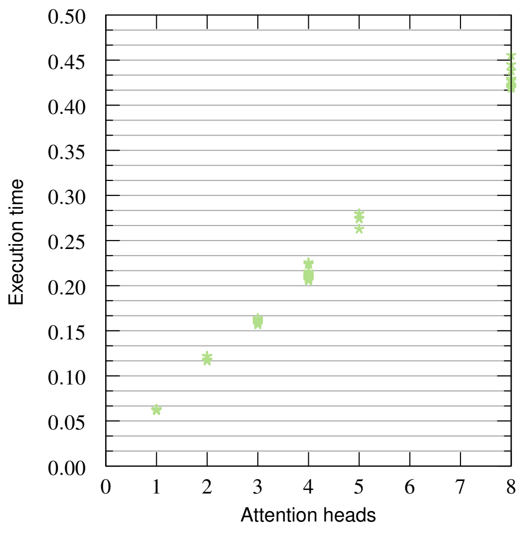
5.3 Analysis: The effect of regularisation on knowledge-distilled models

As I trained models with the group lasso regularisation, I noticed something peculiar in the validation of knowledge-distilled models. As a rule, in the knowledge distillation approach, a student should overfit a teacher’s distribution. On the other hand, regularisation prevents overfitting by design. In NMT, a distilled model is considered converged when BLEU stalls 10–20 times consecutively. It is quite the opposite of a typical training routine where cross-entropy is observed instead.

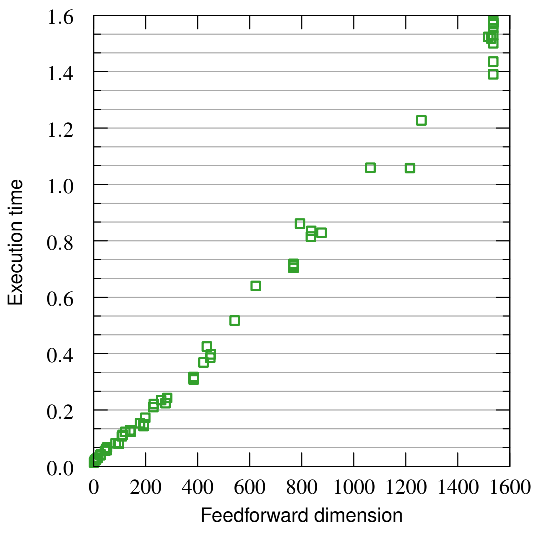
Fig. 5.2 visualises the training progression of English$\rightarrow$German students with “6–6” architectures, regularised under a group lasso using the provided λ. As can be seen in Fig. 5.2a, the most striking result to emerge from it is that the cross-entropy skyrockets for the baseline model. While expected, I overestimated how much overfitting happens in a student model. The regularisation applied throughout the training process plateaus the cross-entropy that still, in most cases, outperforms the baseline validation. This discrepancy may indicate that there could exist a better training regime for the knowledge-distillation, which could still aim to overfit a teacher while resulting in better final quality. I leave this observation as an open question for future research.



(a) Encoder feedforwad nodes.



(b) Encoder self-attention heads.

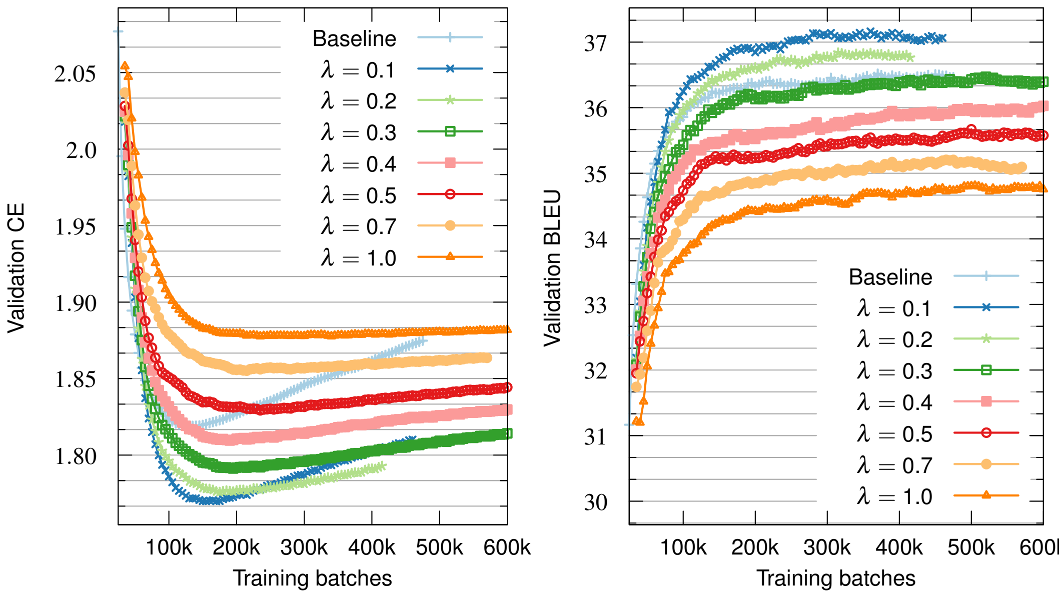


(c) Decoder feedforward nodes.

Figure 5.1: Execution times for feedforward and attention layers for English→German students based on their dimensions or the number of heads.

5.4 Analysis: Perplexity vs regularisation penalty

Next, in the ongoing investigation into regularisation behaviour, I look into how penalties scale during training. It is especially interesting in the context of potential bottlenecks. For this analysis, I take the English→German student model with the “12–1” architecture and apply a neuron-wise group lasso over feedforward and attention layers with $\lambda = 0.03$. I use the



(a) Cross-Entropy **(b)** BLEU

Figure 5.2: The training progression of English→German students with "6–6" architecture (see Tab. 4.11 regularised until convergence.

pretrained checkpoint as the starting point as usual. Then, I print all partial losses participating in training averaged over every $1k$ batches, including the perplexity responsible for the quality and individual regularisation penalties over layers. This way, I scrutinise the training progress and relations between these losses.

In Fig. 5.3, I visualise the perplexity and the total regularisation penalty sanctioned upon both encoder and decoder (Fig. 5.3a). In the early stages of training, the penalty must be larger than the perplexity to incentivise a model to begin zeroing parameters out. A model may focus on quality if a penalty is too small due to λ. Naturally, the former happens in this case. As soon as the model start to reduce the magnitude of parameters, the penalty goes towards zero, and the perplexity takes over.

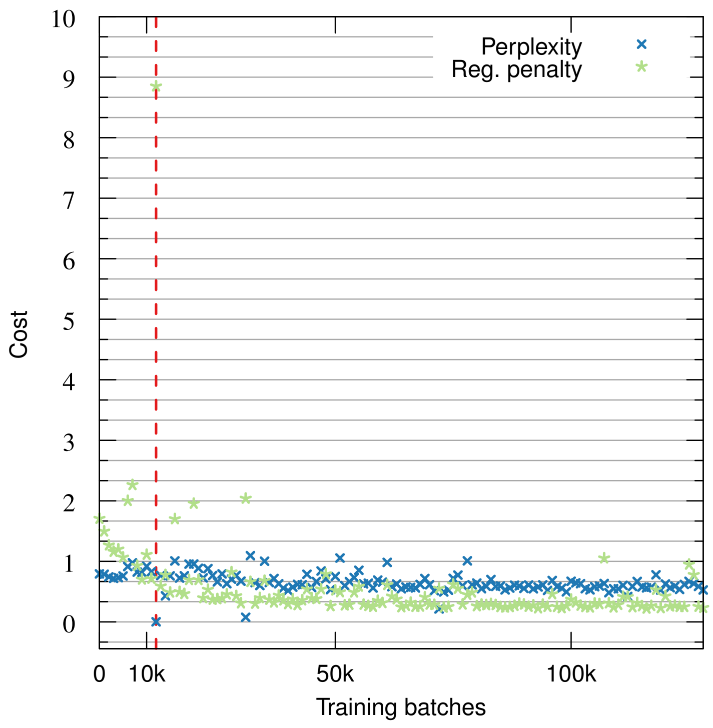
Just a reminder, the group lasso penalties are not scaled down only by λ but also by batch size (words per batch). If a penalty is roughly within the same range and is not scaled by batch, the perplexity will keep being either too large or too small all the time. Scaling the penalty by a batch size makes it follow the similar trajectory up or down. This cost scaling is necessary so that the penalty volume is steady and not constantly subdued by a perplexity that may shift depending on a batch size as well as on how well a model deals with sentences in particular updates.

Batches also vary in how many sentences they carry. Using a static batching routine could solve this issue. However, it would be grossly inefficient for training purposes in NMT as dynamic batching fits as many sentences on GPUs as possible. As can be seen in Fig. 5.3, both the perplexity and the penalty follow a similar trajectory in tandem. However, a few peculiar exceptions shifted my focus in the direction of the outliers.

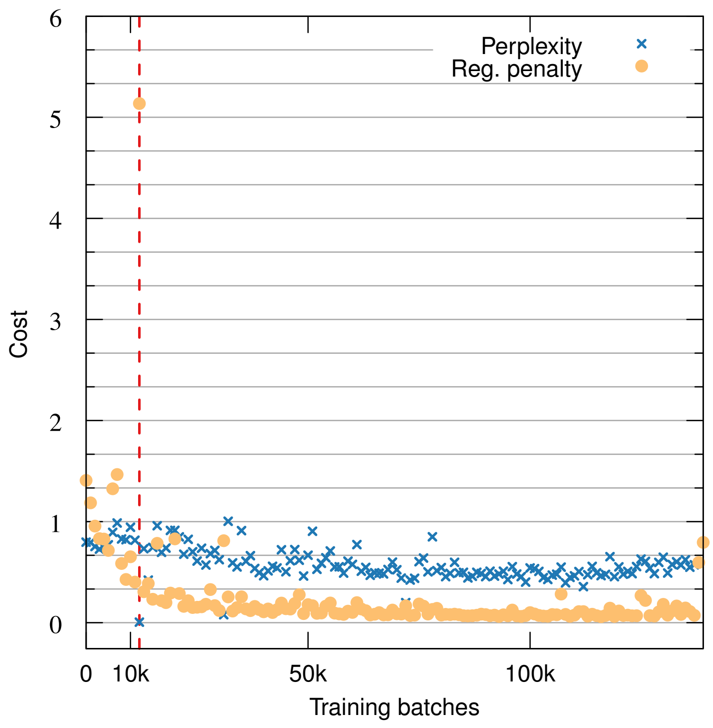
Even though the results are averaged over every thousand batches, there is still a notable checkpoint that seems abnormal. I highlighted it with a dashed vertical line. In this example, the perplexity is small, almost zero, and the regularisation penalty is at least $5–10\times$ larger than usual. Since this is average, multiple issues could be at play here. It is probably a mix of having smaller batches on average combined with "easier" to translate sentences, which causes the regularisation penalty to explode and the perplexity to diminish completely. This behaviour could also mean that some low-quality data disturbs training. It does not seem as damaging during normal training, but it may trigger poor regularisation phenomena. This problem is not isolated to a group lasso alone, but it can happen to any regularisation method. It is more of an issue in natural language processing than in image recognition, which typically trains using the same-sized images and equal batches.

I repeat the same experiment, now regularising only the decoder, now presented in Fig. 5.3b. I increase λ to 0.3 to compensate for fewer layers. The results follow a similar pattern as to the fully regularised architecture. Besides that one extreme outlier, several other points overpower the perplexity. To further illustrate how large this gap between perplexities and penalties is, I compare the perplexity of only one selected penalty: the first affine operation W_1 in the single decoder feedforward layer (Fig. 5.4a) and the same for W_k in the context attention (Fig. 5.4b). A dashed vertical line highlights the outlier checkpoint. Even a single parameter matrix penalty is highly disproportionate to the perplexity.

Overall, this analysis shines a light on the inner workings of regularisation in the context of NMT. The perplexity expresses a model's strive towards a good translation quality with the penalty enforcing sparsity over parameters. The ratio issues between these two may disturb a training flow and impact the gradient descent negatively, possibly leading a model towards suboptimal minima. The results above directly inspired me to look into *scaling* regularisation penalties individually for each layer. Localising λ scaling could theoretically solve an issue of a global ratio being skewed and thus improve training flow, which I leave for future work. Local scaling, by design, could also identify bottlenecks or good candidates for more aggressive pruning. I proceed with exploring the idea in this chapter.

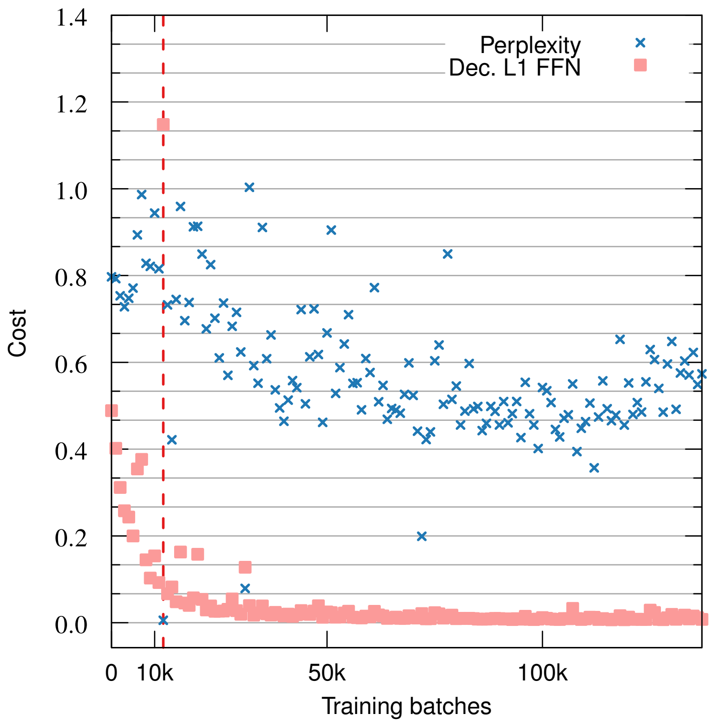


(a) Encoder+Decoder, FFN+Att., $\lambda = 0.03$

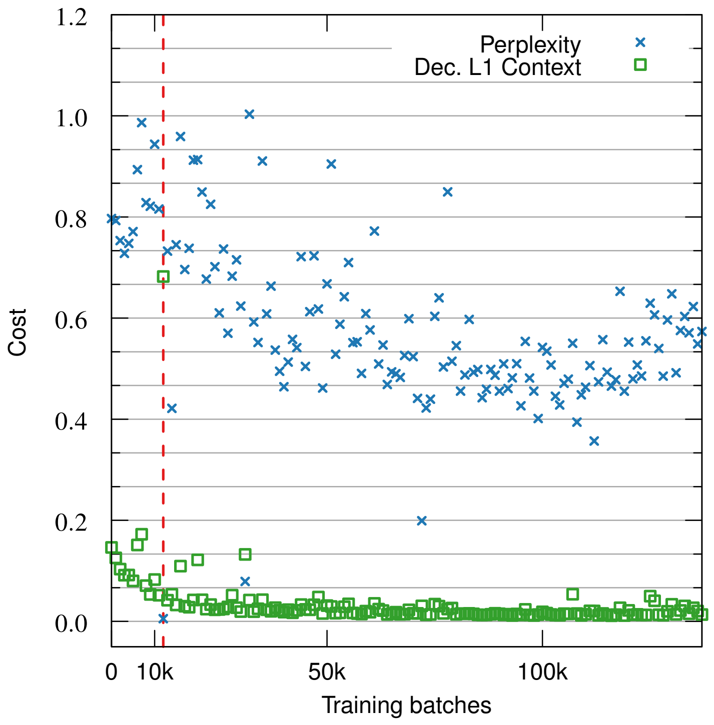


(b) Decoder, FFN+Att, $\lambda = 0.3$

Figure 5.3: The cost comparison between the perplexity and the total regularisation penalty during training of English→German students with "12–1" architecture. The red dashed-line highlights a batch with a disproportional perplexity and a regularisation penalty.



(a) Feedforward W_1 penalty



(b) Attention W_k penalty

Figure 5.4: The cost comparison between the perplexity and the regularisation penalty in the selected decoder layers during training of English→German students with “12–1” architecture.

5.5 Aided Regularisation: Introduction

The previous analysis inspired me to look into scaling regularisation penalties. In other words, it means modifying the regularisation function itself. *Layer-wise* approach is one of the aspects of regularisation that I want to address and expand upon without forcing group lasso over too large structures.

The idea is to use supplementary information to scale the penalties to steer them towards a specific behaviour. In practice, it means adding a new scalar γ alongside an already existing λ:

$$E(batch) = \frac{1}{|batch|}\left(\sum_{x \in batch} CE(x) + \lambda * \sum_{j \in layers} \gamma^j * R(j)\right) \tag{5.1}$$

As shown in Eq. 5.1, each layer has its individual γ, which gets updated after every back-propagation pass. In order to avoid sudden shits in γ between individual batches, which could make a ratio between perplexities and penalties even more unstable, γ are exponentially smoothed as training progresses:

$$\underset{i \in batches}{\forall}\ \underset{j \in layers}{\forall} \gamma^j = \alpha * \hat{\gamma}_i^j + (1-\alpha) * \gamma^{j-1} \tag{5.2}$$

After every batch i, I calculate a local scalar $\hat{\gamma}_i^j$ for each layer j based on information gathered during this specific update, which then updates a smoothed global γ^j scalar. α is a constant used in exponential average that controls the contribution of a new element in a sequence towards the overall average. I use $\alpha = 1\text{e}{-4}$ in my experiments.

“Aided regularisation”, as formally called from now on, uses external information to update and steer these γ scalars per layer. These γ scalars are *not* a part of the computational graph. Instead, they are calculated and updated outside of it and treated as constant nodes in the network.

Pruning methods outside of regularisation assume that some parameters are obsolete due to a predefined heuristic. Thus, they get removed with the hope that a model accommodates their loss independently. In this case, backpropagation does not get any feedback or warning of parameters being about to be removed. A typical regularisation modifies the objective to enforce sparsity upon a model. However, a regulariser does not distinguish which layers or parameters are more important than others. Backpropagation decides the best trade-off between the best-quality objective and a set of parameters required for said quality. At the same time, the force of regularisation remains equally strong, no matter what stage of training it is or how crucial some parameters are to a model. Aided regularisation combines the strengths of the approaches mentioned above to improve upon them.

5.6 Setup

I run all my experiments on the well-familiar English$\rightarrow$German knowledge-distilled student introduced in Sect. 4.7). I focus on the 12–1 architecture as it proved to be a reliable choice and an excellent example for structural pruning. Regularisation is always applied on the neuron level using a group lasso unless stated otherwise.

5.7 Gradient-aided regularisation

The first direction that I explore is *scaling penalties based on gradients*. γ scalars should increase as gradients stop flowing through a layer since it indicates that this layer does not contribute to training as much, possibly stopping learning altogether. I hypothesise that a layer with small gradients is a good candidate to be regularised more aggressively and vice versa. If possible, γ could decrease for layers deemed too important for a model's quality. Similarly, if a regulariser does not prune some parameters over a long period, it can be an accurate prediction that they should be kept in a model and not penalised too much, if at all. Turning a regulariser off serves the same purpose but simultaneously across the whole model. When using local gradients to scale penalties layer-wise, it is important to consider a global gradient norm. The whole point of the gradient descent algorithm is to decrease all gradients as parameters keep getting updated. γ needs to accommodate that to not penalise a model progressively just because it trains successfully.

Feeding gradients into a regulariser that contributes towards the cost function may be problematic. On the other hand, taking backpropagation steps twice to get "pure" partial gradients without penalties involved for γ scaling is too expensive. While not mathematically sound, I use the gradients produced alongside the regularisation as an approximation. This empirical approach should allow me to see how this method performs like in the first place with possible alternatives down the line.

5.7.1 Establishing methodology & scaling variants

The following section explores various designs of γ functions using gradients with experiments and consequent analysis.

Exponential variant

As previously mentioned, the L_2 norm of all gradients needs to be taken into account when designing a γ. For the first experiment, I selected the exponential function $f(x) = e^x$ as the basis for γ.

With W_i being a regularised layer and ∇W as accumulated gradients in a model, the exponential γ is defined as follows:

$$\gamma_i = exp\left(1 - \frac{\|\frac{\partial W_i}{\partial E}\|_2}{\|\nabla W\|_2}\right) \tag{5.3}$$

Here in Eq. 5.3, the numerator representing local gradients of an i^{th} layer is always going to be smaller than the norm of all gradients in the denominator. Thus, subtracting this fraction from 1 results in the range of γ between 0 and e. (approx. 2.72), which scales the regularisation penalty for the relevant layer.

I experiment with this variant of aided regularisation on English$\rightarrow$German knowledge-distilled students (see Setup in Sect. 4.7) with 12–1 architecture. I apply neuron-level regularisation to feedforward layers in both encoder and decoder to investigate the decoder bottleneck and subsequent sparsities. I do not use any tricks to obtain the best quality possible: the regularisation is applied to training from the beginning until the model converges.

The model trained for $450k$ batches with group lasso. After the training finished, I sliced every checkpoint and removed inactive connections that sum up to less than $1\text{e}{-5}$. Since γ additionally scales penalties, the global λ must be smaller to avoid overly aggressive regularisation that may hinder the translation quality. For this reason, I selected $\lambda = 0.1$ in this experiment. I present the visualisation of structural pruning progression using the exponential gradient-based regulariser in Fig. 5.5.

The single decoder layer remains the bottleneck with no parameters removed from it. It took about $200k$ updates for the model to start pruning parameters, most probably due to the low initial λ. However, when selected connections start to zero out, the further middle layers are the first to go (7 to 9). As training progresses, layers 6 and 5 join as well. All other layers are kept intact as more weight is put onto the aforementioned middle layers by the γ.

I do not save smoothing statistics for γ, so whenever a training run restarts in a cluster (about every $80k$ batches), the γ scalars get averaged anew. Besides the natural changes in gradient flows, the occasional larger hiccups in Fig. 5.5 may be caused by these restarts. The sparsity fluctuations may also result from manual thresholding for structural slicing. After a short time,

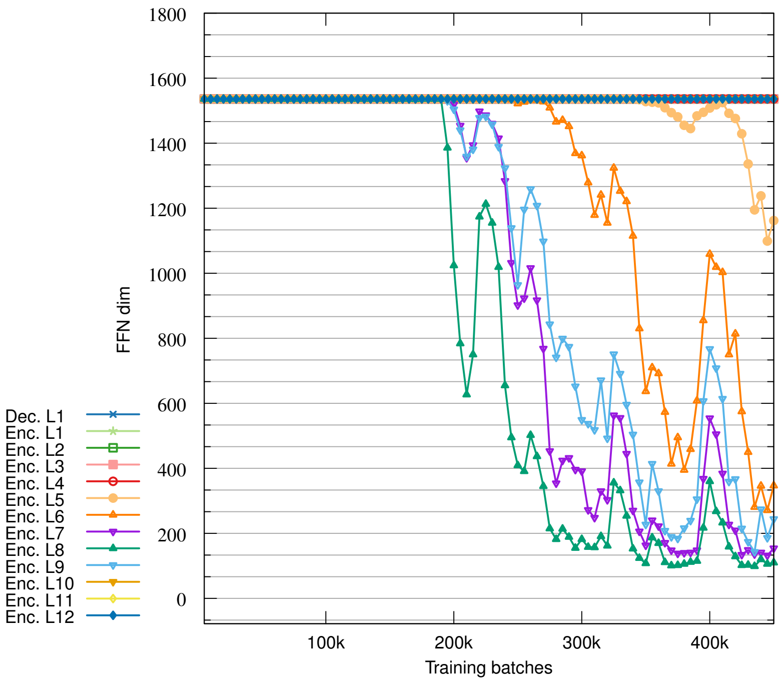


Figure 5.5: The sliced feedforward dimensions of English→German students with "12–1" architecture during training with exponential gradient-aided regularisation.

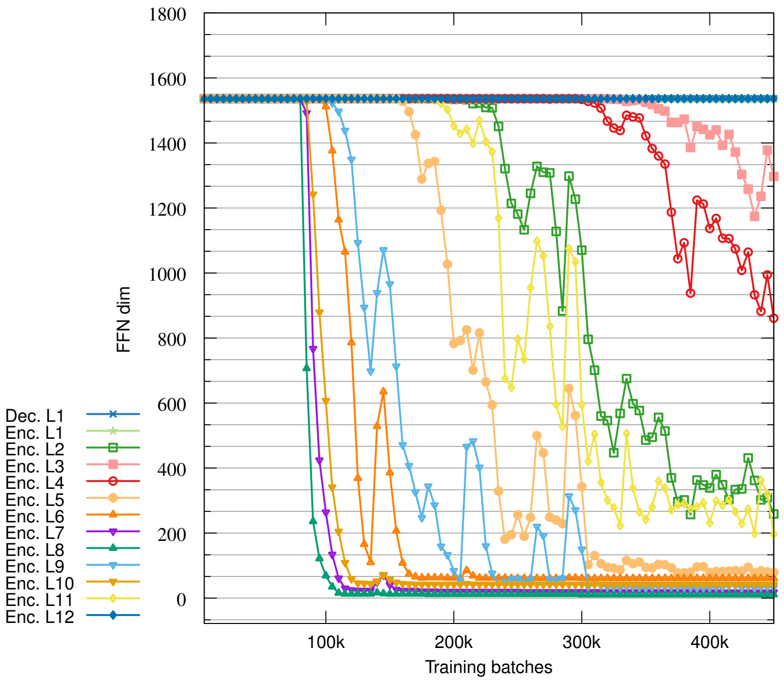


Figure 5.6: The sliced feedforward dimensions of English→German students with "12–1" architecture during training with log-based gradient-aided regularisation.

the parameters quickly catch up to where they were before. This brief reactivation of parameters may serve a good purpose of potentially improving quality while allowing a model to expand its capacity during training temporarily. For example, Y. Wang et al. (2020) have shown that pruning and rejuvenating parameters achieve higher BLEU scores in their experiments.

As I suspected, the gradient flow through the middle layers seems to decrease with training time, putting more pressure on the regulariser. The simple group lasso exhibited similar patterns before, with these layers being top priority from the beginning, but the aided approach tips the pruning further in that direction. Middle layers actively stripped out from a model with a potential decrease in depth is a promising prospect. This variant of aided regularisation is not too pushy: it scales penalties by $\times 2.7$ at most. In this specific example, γ scalars scale the initial $\lambda = 0.1$ up to ~0.27. One could argue that this is not a wide range for a regulariser unless it starts from a much higher λ. In the next section, I look into a more steep γ scaling.

Log-based variant

I proceed with a log function as the base for γ scaling. Following the similar variable definitions as in Eq. 5.3, the log-based γ is defined as follows:

$$\gamma_i = -log\left(\frac{\|\frac{\partial W_i}{\partial E}\|_2}{\|\nabla W\|_2}\right) \tag{5.4}$$

Since the fraction is always smaller than 1, I add minus to the formula to nullify the negation.

I repeat the same experiment as described in the previous section. The progression of structural pruning is presented in Fig. 5.6.

The aggressive regularisation of selected layers immediately stands out from the plot. Overall, the log function leads to a more steep scaling, which suppresses those layers rapidly with the gradient-based approach. This scaling method requires a smaller global λ to moderate the regulariser's strength. It took about $100k$ batches to start pruning, but three layers were completely stripped from the model as soon as it began. Some of the other layers join this endeavour later into training, while many are unaffected, just like in the exponential γ. At that point, I only investigated training stability under such regularisation and did not converge the models to judge their quality. As the log-based γ scaling tends to be more decisive in its pruning (as in, it selects 3–4 layers to be removed swiftly in their entirety from a model), I proceed with more in-depth experiments and studies of this regularisation technique.

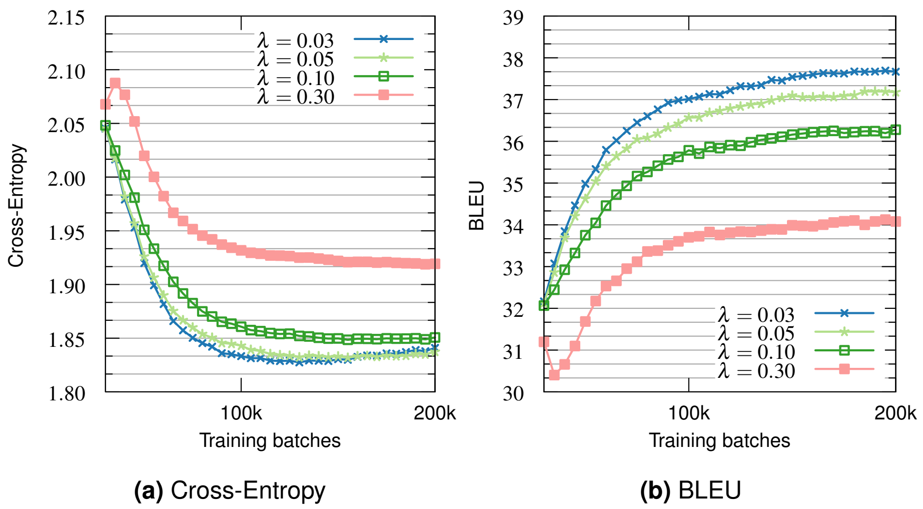


Figure 5.7: The training validation of English→German students with “12–1” architecture regularised with the log-based gradient-aided approach.

5.7.2 Experiment: Training stability

Using the log-based variant of the gradient-aided regularisation, I run experiments pruning feedforward and attention layers in *encoder and decoder* of the “12–1” architecture. This trial run aims to understand better the impact of the modified regularisation on a transformer model and explore potential refinements to the approach. Following the direction set by the previous experiments, all models were pretrained for $25k$ updates. Then on a fresh training run, they are regularised with $\lambda \in \{0.03, 0.05, 0.1, 0.3\}$ for $200k$ batches.

The training progress of models regularised with a gradient-aided approach is presented in Fig. 5.7. The validation is stable in both BLEU and cross-entropy. Naturally, the larger λ is, the more substantial the validation loss. These results show that transformer models successfully train under the gradient-aided regularisation method, producing reasonable validation numbers. Next, I look into how these models sparsify under this regularisation regime.

5.7.3 Analysis: Sparsity progression

The experiments with a standard group lasso have shown (see Fig. 4.4 and 4.5) that some layers have less priority than others, but in general, most of them follow similar patterns to each other. As λ increases, eventually, all layers give in to pruning in a similar fashion. Since aided regularisation scales its penalties, I expect the differences between layers’ sparsity to be more prominent.

Fig. 5.8 presents an extensive visualisation of pruning progress for transformer layers. Each row of plots represents one training run under $\lambda \in \{0.03, 0.05, 0.1, 0.3\}$, which shows how sparsity changes throughout training. I chose to express it as *a percentage of zeroed-out coefficients* (less than $1\mathrm{e}{-6}$) rather than reduced dimensions to minimise fluctuations and provide a smoother picture and analysis. The three columns of plots refer to regularisation progress in the following parameter matrices: W_1 in feedforward and W_k, W_v in attention layers. Group lasso does not tie corresponding layers together, as each row and column for every matrix is regularised independently. It does not change the outcome as structures are connected and sliced out from a model cascadingly. For example, zeroing out columns in W_1 entails pruning of rows in W_2, no matter if they themselves get fully reduced to zero. Similarly, entire attention heads are removed despite all separate parts of attention mechanisms being regularised separately. The regularisation naturally ties corresponding matrices, so there is no need to group them further together. For this reason, I visualise W_k (keys) and W_v (values) only in Fig. 5.8 as they are tied to W_q (queries) and W_o (output) multiplications respectively. They exhibit identical patterns, so I omit them in the multiplot due to redundancy. At the same time, plotting these parts separately allows us to see which parts of the attention mechanism are prioritised by the regulariser.

λ set to 0.03 is too small to prune feedforward layers and requires a larger initial push. However, attention keys are aggressively sparsified from the start. Attention values start to follow the lead with a notable delay. This pattern raises the question of whether it is necessary to apply regularisation to all parts of attention as suppressing values that serve as inputs. It may be counter-intuitive to a model while deciding which attention heads to keep intact or not. In general, attention may be pruned more as its matrices are smaller than feedforward ones, thus increasing the γ. As gradients decrease, the scales for penalties go up drastically. In particular, the regulariser targets encoder layers 7 to 11. As reflected in the previous work on a standard group lasso (see Sect. 4.12), middle layers could become less useful as training continues and, in turn, prove to be good candidates for subsequent pruning. The current results of aided regularisation highlight it even better, with these layers being penalised even more as gradients decrease in magnitude. Many layers reach complete sparsity sooner or later depending on the global scaling with λ. Others remove a portion of their parameters to plateau at a specific sparsity level. It could mean that *layer pruning* does not have to be a binary approach, where a layer is kept entirely or not. Instead, it could be treated as a spectrum but with the emphasis on removing whole layers.

5.7.4 Analysis: γ scalars

In Fig. 5.8, I looked into how sparsity changes during training. This time, I examine the γ scalars directly and how they update for every layer. The visualisation of feedforward γ scaling under different λ is presented in Fig. 5.9.

All the layers get immediately sorted from the start based on the gradients. Most importantly, this order is mostly kept as training goes on. Both Fig. 5.9 and 5.8 are tied together: layers are prioritised during pruning based on γ applied to them. Overall, the penalties have a decreasing tendency, with some drastically rising when a larger global λ is used. Most layers plateau in

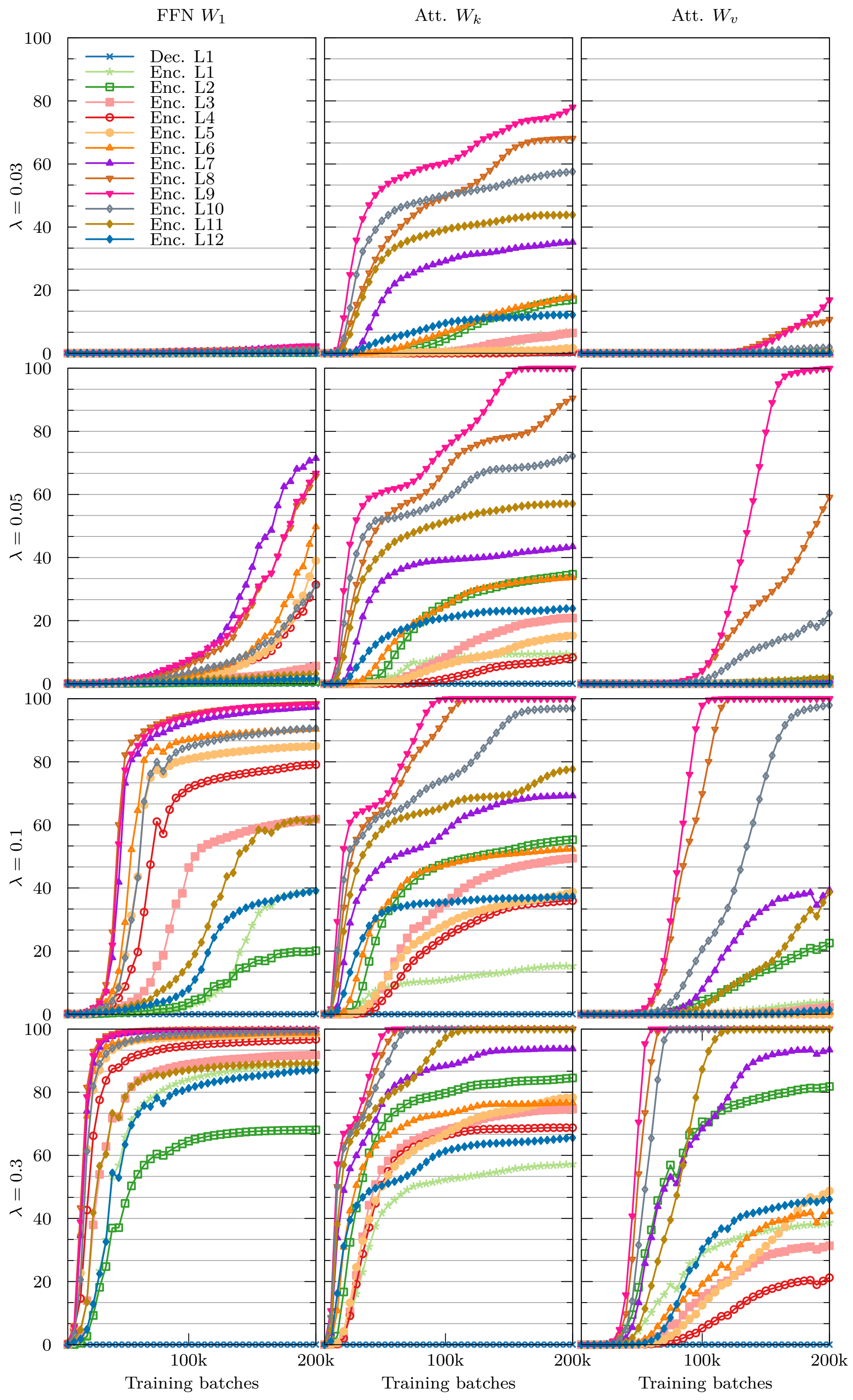


Figure 5.8: The progress of coefficient-wise sparsity in layers for $\lambda \in \{0.03, 0.05, 0.1, 0.3\}$ using gradient-aided regularisation. y-axis represents the percentage of pruned parameters.

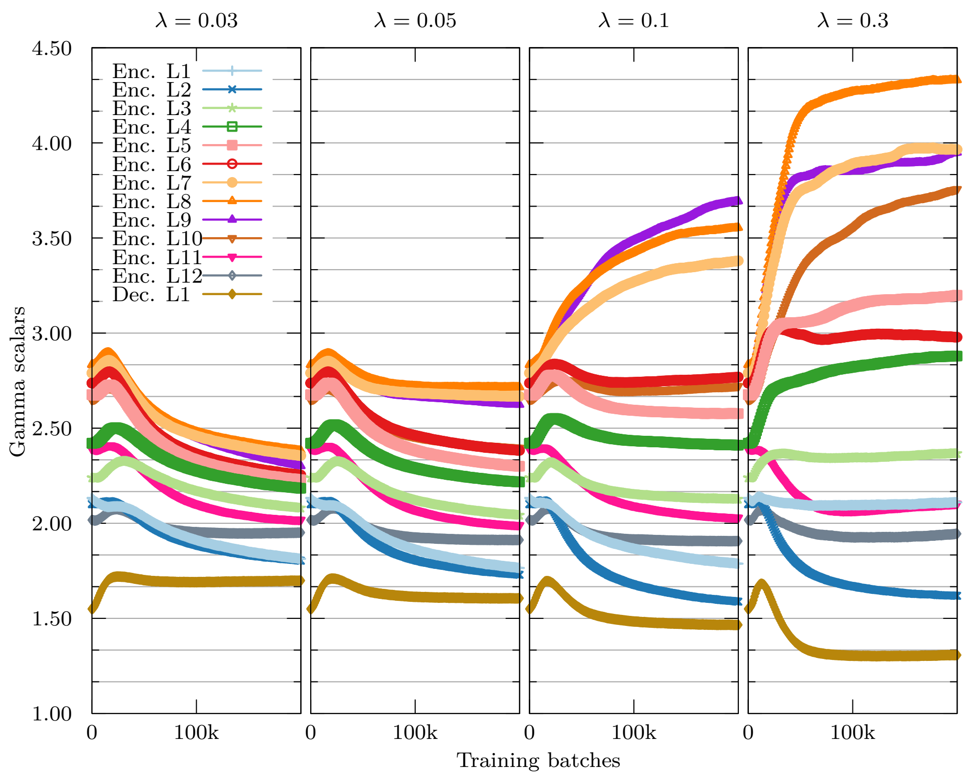


Figure 5.9: The progress of γ scalars for feedforward layers during training with $\lambda \in \{0.03, 0.05, 0.1, 0.3\}$.

the later stages of training, getting reduced from the start value. This pattern can be attributed to gradients, in general, getting progressively larger until they finally stabilise. Those actively pruned layers stop getting updated at some point; thus, their γ rapidly increases. The reverse is true as well: the single decoder layer, which is a known bottleneck, is assigned the lowest γ. It reduces steadily and pulls the pressure off the decoder's penalty. This reduction may have an interesting impact on quality down the line, with negative side-effects of the bottleneck diminished. Instead of forcing all layers to be equally penalised, gradient-aided regularisation scales them, with the previously mentioned middle layers being the main focus of the pruning.

5.7.5 Experiment: Convergence

Now I proceed with slicing and converging the models regularised in the previous section. A neural connection is considered inactive if it sums up to less than $1\mathrm{e}{-4}$. Similarly, an attention head is sliced out if at least half of its connections are dead. The evaluation of the converged architectures is presented in Tab. 5.1. I re-evaluate the inference speed of the baseline model since there have been many new code optimisations done on the toolkit since the efficiency shared task. It translates about 2111 words per second, which slightly improves the previous 1930.

	Reg. $\lambda \longrightarrow$	Base		0.03		0.05		0.1		0.3	
	Layer type $\longrightarrow$	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads
Dimension / Heads	Enc. L1	1536	8	1536	8	1536	7	1494	7	194	4
	Enc. L2	1536	8	1536	8	1536	6	1534	3	502	1
	Enc. L3	1536	8	1536	8	1536	8	683	4	145	0
	Enc. L4	1536	8	1536	8	1536	8	369	7	62	1
	Enc. L5	1536	8	1536	8	1536	8	257	5	31	0
	Enc. L6	1536	8	1536	8	1536	8	168	2	27	0
	Enc. L7	1536	8	1536	8	1536	2	71	2	5	0
	Enc. L8	1536	8	1536	1	1536	0	41	0	1	0
	Enc. L9	1536	8	1536	0	1536	0	41	0	5	0
	Enc. L10	1536	8	1536	1	1536	1	187	0	15	0
	Enc. L11	1536	8	1536	2	1536	2	1536	0	177	0
	Enc. L12	1536	8	1536	8	1536	8	1236	4	235	2
	Dec. L1	1536	8	1536	8	1536	8	1536	8	1536	8
Quality	BLEU		38.2		38.8		38.7		37.8		34.9
	Δ		—		0.6		0.5		-0.4		-3.3
	chrF		63.9		64.0		64.2		63.4		61.5
	COMET		49.5		51.2		51.2		47.4		37.8
Efficiency	Att. sparsity		0%		27%		37%		60%		85%
	FFN sparsity		0%		0%		0%		54%		85%
	WPS		2111		2463		2506		3490		4885
	Speed-up		1.00		1.17		1.19		1.65		2.31

Table 5.1: The results of "12–1" transformer models for English→German regularised with gradient-aided approach on a neuron-level in both encoder and decoder. The layers were subsequently sliced and the models converged. Averaged over WMT16–19 testsets.

In contrast to Fig. 5.8, slicing is more lenient than just counting the percentage of small-valued parameters. For that reason, the feedforward layers do not get pruned until a larger λ is used. The opposite is true for the attention mechanism: many layers are almost completely obliterated. As noted before, layers from about 7 to 11 seem to be the main pruning target. This pattern creates an hourglass shape in the encoder. As expected, the only decoder layer was left completely intact.

In terms of translation quality, there is a slight BLEU improvement for the lesser pruned models, which offer up to $1.19\times$ speed-up. Pruning about half of the attention heads and 60% of feedforward connections results in $1.65\times$ faster inference at the cost of -0.4 BLEU point. The most aggressive 85% sparsity in both parameters makes this model $2.31\times$ faster at the significant -3.3 BLEU loss.

To fully understand the impact of this pruning on quality, I additionally score the models with chrF and COMET in Tab. 5.1. Less sparse models either maintain or surpass the baseline, further proving that a few selected attention layers can be completely removed without affecting the quality. For example, removing almost 5 attention layers completely makes one of the models $1.19\times$ faster with slightly better translation quality. Past that point, speed gains are more significant. The model that is $1.65\times$ faster loses -0.5 chrF and -2.1 COMET points, alongside -0.4 in BLEU.

The results are quite promising, especially given that many layers are (almost) fully pruned in comparison to a typical group lasso regularisation. The past experiments emphasised addressing the bottleneck effect by either omitting the shallow decoder in the regularisation scheme or reducing its force towards the decoder somehow. Therefore, I proceed with the investigation by only applying gradient-aided regularisation to the encoder.

	Reg. $\lambda \longrightarrow$	Base		0.03		0.05		0.1		0.3	
	Layer type $\longrightarrow$	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads
Dimension / Heads	Enc. L1	1536	8	1536	8	1536	7	1501	7	192	4
	Enc. L2	1536	8	1536	8	1536	6	1536	3	494	1
	Enc. L3	1536	8	1536	8	1536	8	649	4	146	0
	Enc. L4	1536	8	1536	8	1536	8	367	7	58	1
	Enc. L5	1536	8	1536	8	1536	8	256	5	29	0
	Enc. L6	1536	8	1536	8	1536	8	161	2	31	0
	Enc. L7	1536	8	1536	8	1535	2	64	2	4	0
	Enc. L8	1536	8	1536	1	1536	0	38	0	1	0
	Enc. L9	1536	8	1536	0	1536	0	44	0	5	0
	Enc. L10	1536	8	1536	1	1536	1	182	0	13	0
	Enc. L11	1536	8	1536	3	1536	2	1532	0	175	0
	Enc. L12	1536	8	1536	8	1536	8	1536	4	277	2
	Dec. L1	1536	8	1536	8	1536	8	1536	8	1536	8
Quality	BLEU	38.2		38.9		38.8		37.8		35.0	
	Δ	—		0.7		0.6		-0.4		-3.2	
	chrF	63.9		64.2		64.1		63.4		61.5	
	COMET	49.5		51.5		51.4		48.3		38.1	
Efficiency	Att. sparsity	0%		26%		37%		60%		85%	
	FFN sparsity	0%		0%		0%		53%		85%	
	WPS	2111		2476		2506		3501		4955	
	Speed-up	1.00		1.17		1.19		1.66		2.35	

Table 5.2: The results of “12–1” transformer models for English→German regularised with gradient-aided approach on a neuron-level in encoder only. The bottleneck effect is minimal in contrast to previous regularisation methods.

5.7.6 Experiment: Decoder bottleneck

I repeat the experiment from the previous section, but I omit the decoder in regularisation this time. The models are sliced after $200k$ regularised batches with $\lambda \in \{0.03, 0.05, 0.1, 0.3\}$ and then trained until convergence. The results are presented in Tab. 5.2.

The sparsity patterns and the quality are close to those wholly regularised models. Direct comparison between Tab. 5.1 and 5.2 shows that the bottleneck effect does not impact the quality in this regularisation scheme, with the difference between the two approaches being ± 0.1 BLEU point. However more extensive quality evaluation using chrF and COMET in Tab. 5.2 shows that this is not entirely true.

Despite the $\lambda = 0.1$ model scoring the same BLEU and chrF in both cases of regularising the decoder or not (Tab. 5.1 vs 5.2), the COMET score is worse by -0.9 when regularisation over decoder is enforced, which is the largest difference in COMET quality between those tables. Nonetheless, this gap in quality is much smaller than the one exhibited by the standard group lasso. The "6–2 tied" architecture lost at most -1.8 COMET points when forced to prune the decoder on top of the encoder. The larger loss can be contributed to a major sparsification of the bottleneck decoder layer as λ increased. This loss is twice as large as the current regularisation method.

On the other hand, the visualisation in Fig. 5.9 shows γ assigned to the decoder layer either being kept constant or decreasing throughout the training. Progressively decreasing γ alleviates the pressure off the decoder and reduces the negative impact of the bottleneck. While not regularising a shallow decoder is still the best option, scaling the penalties through aided regularisation improves the results on this front.

Reg. $\lambda \longrightarrow$		Base	0.03	0.05	0.1	0.3
BLEU	Pruned	38.2	38.9	38.8	37.8	35.0
	Reinit	—	38.0	38.1	37.0	33.6
chrF	Pruned	63.9	64.2	64.1	63.4	61.5
	Reinit	—	63.8	63.6	62.7	60.8
COMET	Pruned	49.5	51.5	51.4	48.3	38.1
	Reinit	—	48.5	48.3	44.2	33.6

Table 5.3: The evaluation of English→German "12–1" students with pruned encoder (*Pruned*) using gradient-aided regularisation, compared to the same architecture trained from scratch (*Reinit*).

5.7.7 Experiment: Training from scratch

As usual, the question is whether the gradient-aided regularisation leads to a better translation quality or if it serves more as an architecture search with the same models achieving similar performance when trained from scratch. To test this hypothesis, I take the converged models pruned with gradient-aided regularisation, reinitialise their parameters and retrain the architectures. I choose models with a pruned encoder only from Tab. 5.2 as they have proven to perform the best quality-wise. The direct comparison of pruned and reinitialised experiments is presented in Tab. 5.3.

Even though the first two models ($\lambda \in \{0.03, 0.05\}$) lose only 0.1–0.3 in BLEU and chrF when trained from scratch relative to the baseline, the quality significantly drops in COMET. Across the experiments, the reinitialised models lose between 3.0 to 4.5 COMET points in comparison to their pruned counterparts. Again, this proves that careful structural pruning, this time through aided regularisation, results in superior translation quality, clearly outperforming models trained with the same architecture from the beginning.

5.7.8 Experiment: Tensor suppression

Slicing an architecture only after a model finishes a regularisation regime means that parameters have the opportunity to reactivate and possibly be pruned again during training. As I previously mentioned, it may even improve the quality of a model. On the other hand, a model needs to actively keep parameters subdued if it decides to prune them, which may necessitate the usage of a larger λ to achieve satisfying sparsity levels. In turn, larger λ damages the quality to a certain extent. If we care about inference speed, we may favour a more stern model in its pruning decisions.

In the following experiment, besides the usual regularisation, I also access parameter tensors and zero-out coefficients that are smaller than $1\mathrm{e}{-6}$. There is no separate masking mechanism in the calculation graph; tensors are accessed directly instead. These parameters are not involved in training calculations and get repressed permanently. I re-run the experiments with the gradient-aided regularisation applied to both encoder and decoder for $200k$ batches after $25k$ updates of pretraining. Every $10k$ batches thresholded coefficients are zeroed-out and disabled from the gradient descent. Similarly to Fig. 5.8, I visualise the sparsity progression for gradient-aided regularisation with tensor masking in Fig. 5.10.

Model checkpoints happening every $5k$ with tensor masking done every $10k$ create the staircase effect in the plot. In contrast to Fig. 5.8, the progress of sparsity levels is much smoother. Notably, W_v (and in turn W_o as well) in the attention mechanism are the least eager to be pruned. Only a few selected layers aim for 100% sparsity, namely layers 8 to 11. The decoder remains unaffected as usual. The γ scalars behave similarly to the patterns presented in Fig. 5.9, so I skipped this analysis due to redundancy.

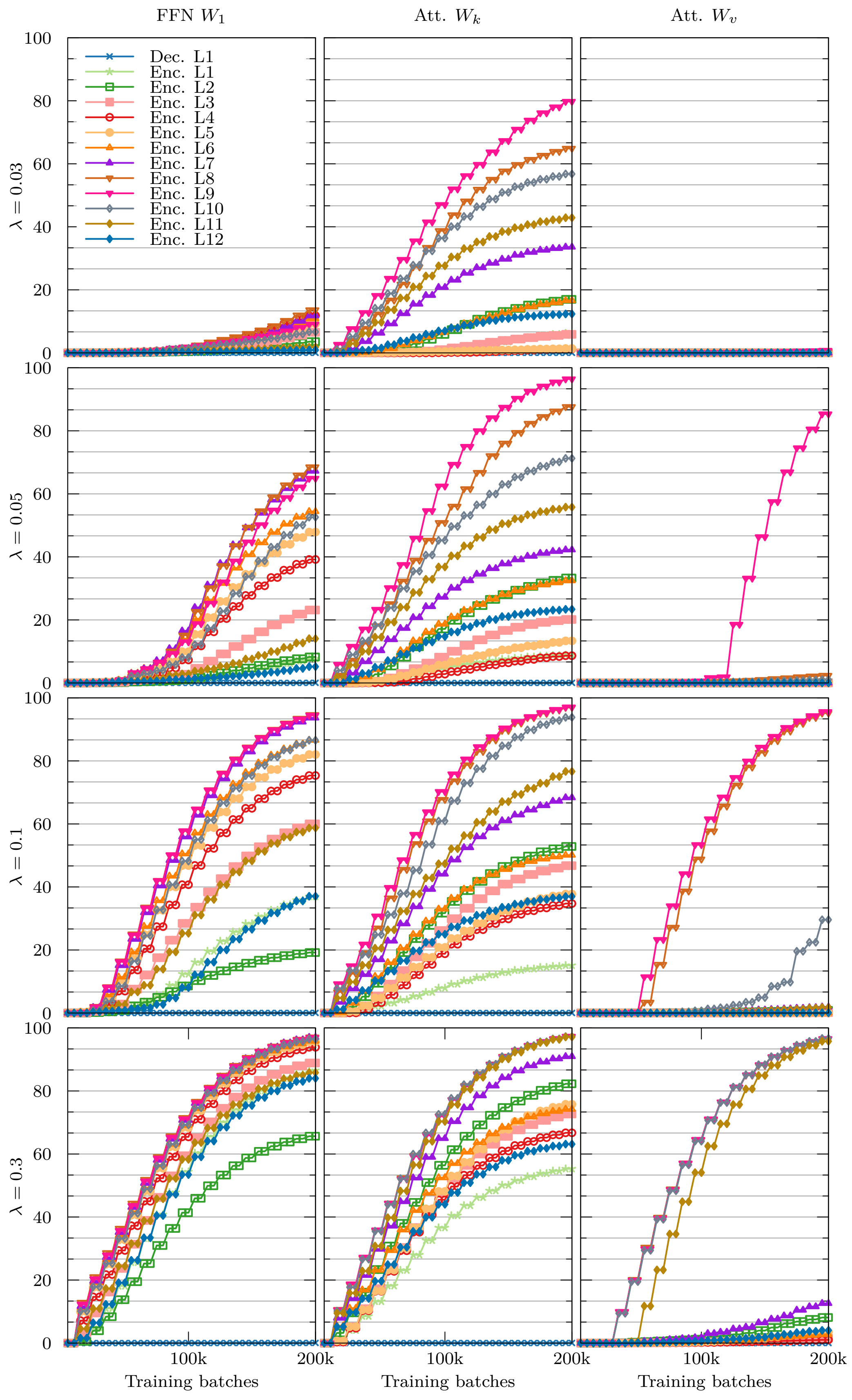


Figure 5.10: The progress of coefficient-wise sparsity in layers for $\lambda \in \{0.03, 0.05, 0.1, 0.3\}$ using gradient-aided regularisation with tensor masking.

	Reg. $\lambda \longrightarrow$	Base		0.03		0.05		0.1		0.3	
	Layer type $\longrightarrow$	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads
Dimension / Heads	Enc. L1	1536	8	1495	7	1372	7	930	7	160	4
	Enc. L2	1536	8	1458	8	1386	6	1222	3	488	1
	Enc. L3	1536	8	1371	8	1123	8	561	4	125	0
	Enc. L4	1536	8	1283	8	869	8	325	7	48	1
	Enc. L5	1536	8	1298	8	723	8	221	5	23	0
	Enc. L6	1536	8	1280	8	619	8	150	2	19	0
	Enc. L7	1536	8	1214	8	413	2	37	2	3	0
	Enc. L8	1536	8	1142	1	381	0	22	0	1	0
	Enc. L9	1536	8	1187	0	429	0	25	0	3	0
	Enc. L10	1536	8	1294	1	616	1	145	0	8	0
	Enc. L11	1536	8	1488	2	1196	2	576	0	172	0
	Enc. L12	1536	8	1511	8	1415	8	920	4	196	2
	Dec. L1	1536	8	1536	8	1536	8	1536	8	1536	8
Quality	BLEU		38.2		38.9		38.6		37.7		34.7
	Δ		—		0.7		0.4		-0.5		-3.5
	chrF		63.9		64.2		64.0		63.3		61.6
	COMET		49.5		52.4		50.6		47.7		37.7
Efficiency	Att. sparsity		0%		28%		37%		60%		85%
	FFN sparsity		0%		12%		40%		67%		86%
	WPS		2111		2511		2938		3761		5177
	Speed-up		1.00		1.19		1.39		1.78		2.45

Table 5.4: The results of "12–1" transformer models for English$\rightarrow$German regularised with gradient-aided approach on a neuron-level in both encoder and decoder. Any coefficients smaller than $1\mathrm{e}{-6}$ are replaced with zeroes and surpressed during training.

Next, the models are sliced just like before and trained until convergence. The results are shown in Tab. 5.4. This regularisation is more aggressive in its structural pruning as suppressing individual coefficients further incentivises a model to remove other parameters from within the same connections. With a slight improvement in BLEU, a model gains about $1.39\times$ faster inference while removing about 40% of parameters across feedforward and attention layers. Pruning about two-thirds of the parameters leads to $1.78\times$ faster translation at the expense of half a BLEU point. It is an excellent example of a really good Pareto quality-speed trade-off we will see in Sect. 5.10. Finally, I confront these pruned architectures with a more extensive quality evaluation with chrF and COMET in Tab. 5.4.

The model mentioned above with $1.39\times$ faster inference also maintains or surpasses its quality performance in both chrF and COMET, just like BLEU. In conclusion, tensor masking guides a model towards sparser and faster architectures while still achieving comparable translation quality to the baseline.

5.8 Parameter-aided regularisation

In the research around aided regularisation, I concentrated the efforts on gradient-based penalty scaling. Despite using regularised gradients as an approximation of pure derivatives, I have empirically shown that gradient-aided regularisation produces good quality models with significantly faster inference. To explore the topic further, I decided to simplify a solution to this optimisation problem and shift focus towards parameter-based γ scaling.

5.8.1 Methodology

The intuition is quite simple: pruning should accelerate as a layer gets sparser. In other words, the more parameters are removed from a layer, the larger its γ scalar should become. *Parameter-aided* regularisation, as the name suggests, uses the magnitude of parameters (instead of gradients) to scale individual layers.

Continuing a log-based approach, I define the parameter-aided γ function as:

$$\gamma_i = log(\|W_i\|_2) \tag{5.5}$$

γ_i is then smoothed (Eq. 5.2) and applied to the cost function (Eq. 5.1) as defined by the aided regularisation approach.

All the other hyperparameters and settings are unchanged for this method.

5.8.2 Analysis: γ scalars

Let us examine the above-defined γ function. I proceed with regularising the “12–1” architecture across both encoder and decoder using the *parameter-aided* method. I visualise the progress of γ for feedforward layers in Fig. 5.11.

As evident from the plots, this is a monotonically increasing function. This monotonicity is only natural as there is no opportunity for the function to decrease: parameters are penalised towards zero, unlike gradients that go up and down. It is worth noting that the layers are in a similar order to that in Fig. 5.9, with the further middle layers being prioritised the most again. The γ range is much wider, scaling even up to $9\times$. This is because the magnitude of parameters is not divided by any scalar or variable with possible future improvement in this aspect. Potentially, it could be averaged over, for example, one of the dimensions or even entirely replaced by a completely novel function.

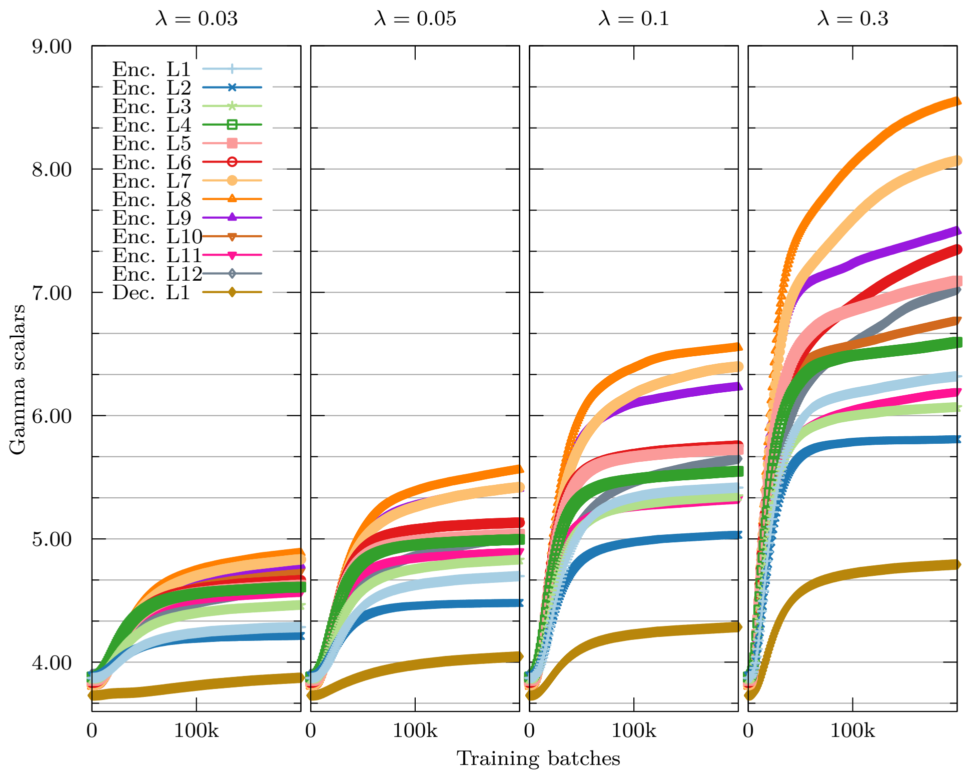


Figure 5.11: The progress of γ scalars in parameter-aided regularisation for feedforward layers during training with $\lambda \in \{0.03, 0.05, 0.1, 0.3\}$.

In contrast to the gradient-based method shown in Fig. 5.9, this is the first time where the single decoder layer gets gradually more penalised as training continues. As will be evident in the next experiment, this does not cause the decoder to be pruned in the end since other layers are still assigned proportionally larger γ. Compared to the previously tested gradient approach, this mirror-like behaviour should broaden our general understanding of which research direction is better and the performance difference between those two.

5.8.3 Experiment: Convergence

Now I proceed with slicing and converging the models regularised in the previous section, similarly how I did in Sect. 5.7.5. Since omitting the decoder during regularisation produces the best quality results, I proceed with regularising the encoder only in the following experiment. I repeat the usual pruning setup: applying the parameter-aided regulariser for $200k$ updates on a model pretrained for $25k$, then slicing an architecture. Finally, a model trains until convergence. The results are presented in Tab. 5.5.

	Reg. $\lambda \longrightarrow$	Base		0.03		0.05		0.1		0.3	
	Layer type $\longrightarrow$	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads
Dimension / Heads	Enc. L1	1536	8	1495	7	1372	7	930	7	160	4
	Enc. L1	1536	8	1536	8	702	7	158	7	23	4
	Enc. L2	1536	8	1536	8	943	7	363	3	54	1
	Enc. L3	1536	8	1536	8	523	8	171	7	32	1
	Enc. L4	1536	8	1536	8	335	8	108	8	13	1
	Enc. L5	1536	8	1494	8	331	8	80	8	5	0
	Enc. L6	1536	8	1536	8	272	8	63	8	7	1
	Enc. L7	1536	8	1536	8	244	8	36	4	4	0
	Enc. L8	1536	8	1536	7	181	2	23	0	1	0
	Enc. L9	1536	8	1536	6	223	0	33	0	4	0
	Enc. L10	1536	8	1536	3	259	2	74	0	10	0
	Enc. L11	1536	8	1536	7	1237	2	184	2	38	0
	Enc. L12	1536	8	1536	8	1442	8	190	4	14	3
	Dec. L1	1536	8	1536	8	1536	8	1536	8	1536	8
Quality	BLEU	38.2		38.9		38.0		37.1		33.5	
	Δ	—		0.7		-0.2		-1.1		-4.7	
	chrF	63.9		64.1		63.7		62.8		60.6	
	COMET	49.5		52.0		51.0		45.3		33.0	
Efficiency	Att. sparsity	0%		9%		27%		43%		82%	
	FFN sparsity	0%		0%		59%		85%		91%	
	WPS	2111		2358		3193		4074		5258	
	Speed-up	1.00		1.12		1.51		1.93		2.49	

Table 5.5: The results of "12–1" transformer models for English→German pruned with feedforward connections and attention heads using parameter-aided regularisation. Averaged over the WMT16–19 testsets.

The parameter distribution of unpruned structures is generally flatter. Feedforward connections get prioritised during pruning, with attention following the lead. Again, the sparse architectures exhibit hourglass shapes, a little bit slimmer on the feedforward side in comparison to the gradient-aided regularisation. Among the representative trade-offs is a model that is $1.51\times$ faster at the cost of 0.2 BLEU point after pruning about 30% of attention and 60% of feedforward layers. Removing more than that leads to at least $1.93\times$ speed-up for a minimum of 1.1 BLEU in damage.

Going ahead with the quality analysis in Tab. 5.5, the previously mentioned model that is one and a half times faster than the baseline loses 0.2 point in both BLEU and chrF while *outperforming* the COMET score by 1.5 points. Like gradient-aided regularisation, using the magnitude of parameters to scale penalties has also proven to train smaller models of comparable quality and actually faster inference.

Reg. $\lambda \longrightarrow$		Base	0.03	0.05	0.1	0.3
BLEU	Pruned	38.2	38.9	38.0	37.1	33.5
	Reinit	—	38.2	37.4	36.1	33.0
chrF	Pruned	63.9	64.1	63.7	62.8	60.6
	Reinit	—	63.8	63.1	62.4	60.4
COMET	Pruned	49.5	52.0	51.0	45.3	33.0
	Reinit	—	49.8	45.7	43.1	30.1

Table 5.6: The evaluation of English→German "12–1" students with pruned encoder (*Pruned*), compared to the same architecture trained from scratch (*Reinit*).

5.8.4 Experiment: Training from scratch

Last but not least, Tab. 5.6 confirms that the same architectures produced by this regularisation method translate significantly worse quality-wise when trained from scratch. The difference in COMET is from 2 up to 5.3 points, the latter corresponding to a 0.8 loss in BLEU. These results suggest that human-perceived quality is much worse in smaller models trained straightforwardly.

5.9 Layerwise pruning by snipping and rejuvenating parameters

Interestingly, there seems to be a sort of a connection between gradient flow and a model's preference for which layers to prune mainly. In the gradient-aided regulariser, layers constantly updated with decreasingly smaller gradients get pruned first and foremost. The parameter-based regularisation accelerates the sparsification of those layers that get subsequently pruned due to the penalty itself. They are the same groups of parameters in both cases: across all 12 encoder layers, the regularisers select layers roughly between 7 to 11. The experiments to this point have shown that the layers can be successfully reduced or even entirely removed.

Until now, models kept at least a small portion of connections per feedforward layer, and I avoided forcefully removing them, believing that maybe these scarce neurons are crucial for a model's performance. A transformer may aim to slim its middle layers while preferring to keep the depth of a model intact, while it may not require as many parameters in the middle layers compared to those on the edges. On the other hand, it is also highly possible that structural regularisation pursues depth reduction, and those few leftover parameters are obsolete.

Continuing the line of thought that rejuvenating diminished parameters improves the quality, the hourglass shape of a pruned architecture may serve as a gradual adaptation for *layerwise* pruning. After removing the heavily sparsified layers, the rest of the parameters can reactivate to recover as much quality as possible. This method could be an ideal solution for those who look for more shallow architectures while still preferring quality over speed.

In this section, the goal is to answer two questions:

> Can the feedforward layers with few connections just be skipped, or are those crucial to the quality?

and

> After removing selected layers, can the rest of the parameters be rejuvenated to recover quality while the overall depth of a model is reduced?

5.9.1 Experiment: Gradient-aided regularisation

Let us start with gradient-aided regularisation. As presented in Tab. 5.7, I select the experiment regularised with $\lambda = 0.1$, is $1.66\times$ faster at the cost of -0.4 BLEU point in comparison to the baseline. I highlight the attention layers that have already been removed and potential candidates for pruning in feedforward layers that have less than 200 neurons left in them, which corresponds to layers 6 to 10.

To answer the question of whether those few neurons are necessary or obsolete to the model, I slice this architecture further after it finished the regularisation phase by *snipping* the highlighted feedforward parameters out. Now, there are 5 feedforward and 4 attention layers, with 3 full layers in the model having both types of parameters removed simultaneously. In Tab. 5.7, this experiment is described as $0.1+S$ as it is the $\lambda = 0.1$ experiment with additional snipping (S).

	Reg. $\lambda \longrightarrow$	Base		0.1		0.1 + S		0.1 + S&R	
	Layer type $\longrightarrow$	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads
Dimension / Heads	Enc. L1	1536	8	1501	7	1501	7	1536	8
	Enc. L2	1536	8	1536	3	1536	3	1536	8
	Enc. L3	1536	8	649	4	649	4	1536	8
	Enc. L4	1536	8	367	7	367	7	1536	8
	Enc. L5	1536	8	256	5	256	5	1536	8
	Enc. L6	1536	8	161	2	0	2	0	8
	Enc. L7	1536	8	64	2	0	2	0	0
	Enc. L8	1536	8	38	0	0	0	0	0
	Enc. L9	1536	8	44	0	0	0	0	0
	Enc. L10	1536	8	182	0	0	0	0	0
	Enc. L11	1536	8	1532	0	1532	0	1536	0
	Enc. L12	1536	8	1536	4	1536	4	1536	8
	Dec. L1	1536	8	1536	8	1536	8	1536	8
BLEU	WMT16–19		38.2		37.8		37.8		38.3
	Δ		—		-0.4		-0.4		0.1
Efficiency	Att. sparsity		0%		60%		60%		38%
	FFN sparsity		0%		53%		63%		54%
	WPS		2111		3501		3605		3052
	Speed-up		1.00		1.66		1.71		1.45

Table 5.7: The results of "12–1" transformer models for English→German pruned with gradient-aided regularisation using $\lambda = 0.1$ compared to it having its layer snipped (*S*) as well as layers snipped and the rest rejuvenated (*S&R*).

Quality-wise, the snipped model performs on par with the few removed parameters. It supports the notion that a model regularised structurally strives to reduce its architectural depth and that middle layers become obsolete as training continues.

In order to confirm it further, as well as to see if rejuvenating the remaining parameters helps a model recover its lost quality, I run another experiment with *snipping and rejuvenating* (S&R). I additionally prune 7th attention layer to have 5 full layers removed in total. This way, the "12–1" architecture becomes "7–1", which significantly reduces the depth. The other layers rejuvenate after being penalised towards zero by the regulariser, resulting in 1536 dimensions across the active feedforward layer and 8 heads per attention layer. Reactivating parameters means that this model is slightly slower as it is less sparse. However, reducing the encoder depth from 12 to 7 results in 45% inference speed-up while at the same time maintaining the quality in BLEU when compared to the baseline.

These results are exciting as they show the possibility of training really deep models but ending up with more shallow and compact architectures that perform at the level of the deep model quality-wise.

Reg. $\lambda \longrightarrow$	Base	0.1	0.1 + S	0.1 + S&R
BLEU	38.2	37.8	37.8	38.3
chrF	63.9	63.4	63.4	63.9
COMET	49.5	48.3	48.1	50.2

Table 5.8: The evaluation of English→German "12–1" students with pruned encoder using the gradient-aided regularisation.

I continue the analysis on quality in Tab. 5.8, where you can see that the "7–1" architecture keeps the baseline performance in chrF as well and slightly outperforms it in COMET ($+0.7$ point).

5.9.2 Experiment: Parameter-aided regularisation

I repeat the same experimental setup described in the previous section, but this time with the parameter-aided regularisation. Again, I select the model penalised with $\lambda = 0.1$. In this case, the nature of this regularisation method resulted in a much sparser architecture. There are 3 attention layers already removed, and I also highlight 6 feedforward layers with less than 100 active neurons. While this pruned model is almost twice as fast as the baseline, it is so at the cost of 1.1 BLEU points. However, snipping the selected feedforward layers only causes a loss of an additional 0.1 BLEU (-1.2 in total). This outcome again confirms that structural regularisation can lead to reducing architectural depth.

	Reg. $\lambda \longrightarrow$	Base		0.1		0.1 + S		0.1 + S&R	
	Layer type $\longrightarrow$	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads
Dimension / Heads	Enc. L1	1536	8	158	7	158	7	1536	8
	Enc. L2	1536	8	363	3	363	3	1536	8
	Enc. L3	1536	8	171	7	171	7	1536	8
	Enc. L4	1536	8	108	8	108	8	1536	8
	Enc. L5	1536	8	80	8	0	8	0	8
	Enc. L6	1536	8	63	8	0	8	0	8
	Enc. L7	1536	8	36	4	0	4	0	8
	Enc. L8	1536	8	23	0	0	0	0	0
	Enc. L9	1536	8	33	0	0	0	0	0
	Enc. L10	1536	8	74	0	0	0	0	0
	Enc. L11	1536	8	184	2	184	2	1536	8
	Enc. L12	1536	8	190	4	190	4	1536	8
	Dec. L1	1536	8	1536	8	1536	8	1536	8
BLEU	WMT16–19	38.2		37.1		37.0		38.0	
	Δ	—		-1.1		-1.2		-0.2	
Efficiency	Att. sparsity	0%		43%		43%		23%	
	FFN sparsity	0%		85%		94%		62%	
	WPS	2111		4074		4152		3046	
	Speed-up	1.00		1.93		1.97		1.44	

Table 5.9: The results of "12–1" transformer models for English→German pruned with parameter-aided regularisation using $\lambda = 0.1$ compared to it having its layer snipped (*S*) as well as layers snipped and the rest rejuvenated (*S&R*).

I proceed with the experiment where the remaining parameters are rejuvenated. I omit forcefully removing layers 5 to 7 in attention to avoid too aggressive snipping since those are almost entirely intact. There are 3 fully skipped layers and further 3 feedforward layers removed as well. While much less fast than the other models, the rejuvenation recovers most of the quality lost through sparsification. It is only -0.2 BLEU worse than the baseline model with $1.44\times$ faster translation.

Reg. $\lambda \longrightarrow$	Base	0.1	0.1 + S	0.1 + S&R
BLEU	38.2	37.1	37.0	38.0
chrF	63.9	62.8	62.9	63.9
COMET	49.5	45.3	45.5	50.2

Table 5.10: The evaluation of English→German "12–1" students with pruned encoder using the parameter-aided regularisation.

As usual, I go ahead with the extended quality evaluation in Tab. 5.10. Like the previous experiments, snipping layers and rejuvenating parameters maintain the baseline's quality and slightly surpass its COMET score.

Given that the final speed vs quality trade-off between snipped and rejuvenated (S&R) models in Tab. 5.8 and 5.10 is similar, a model may not need this aggressive level of regularisation to decide to snip specific layers out from the architecture.

5.10 Analysis: Pareto trade-off

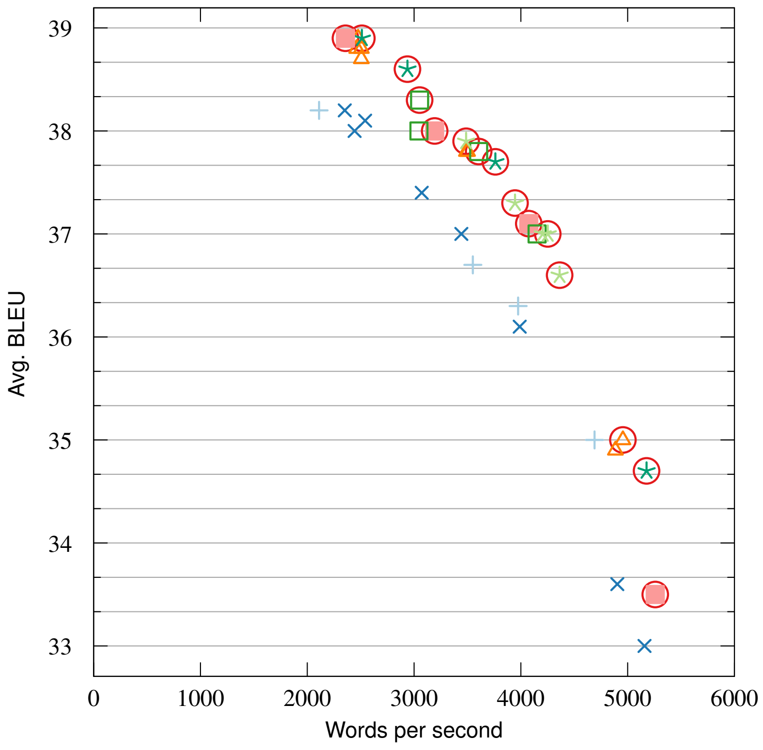
Last but not least, I conclude this chapter with the analysis of the Pareto trade-off, including all models trained in this chapter and more. Since I used an improved code base of the Marian toolkit, the models are faster, and the current numbers cannot be directly compared to those from the previous chapter. Thus, I re-evaluate the speed of the "12–1" experiments from Tab. 4.21, pruned with the standard group lasso, and I take them into account in this Pareto analysis.

When it comes to the baselines, I train models with feedforward layers and attention heads linearly reduced in the encoder. To be precise, the dimensions and the numbers of heads in all baseline models are as follows: $\{1536, 8\}$, $\{768, 4\}$, $\{384, 2\}$ and $\{196, 1\}$. The evaluation of these baseline models is presented in Tab. 5.11. Even though halving all layer parameters in the encoder gains 68% speed-up, it damages quality by -1.5 BLEU points. Training smaller models produce progressively worse quality at the expanse of inference efficiency.

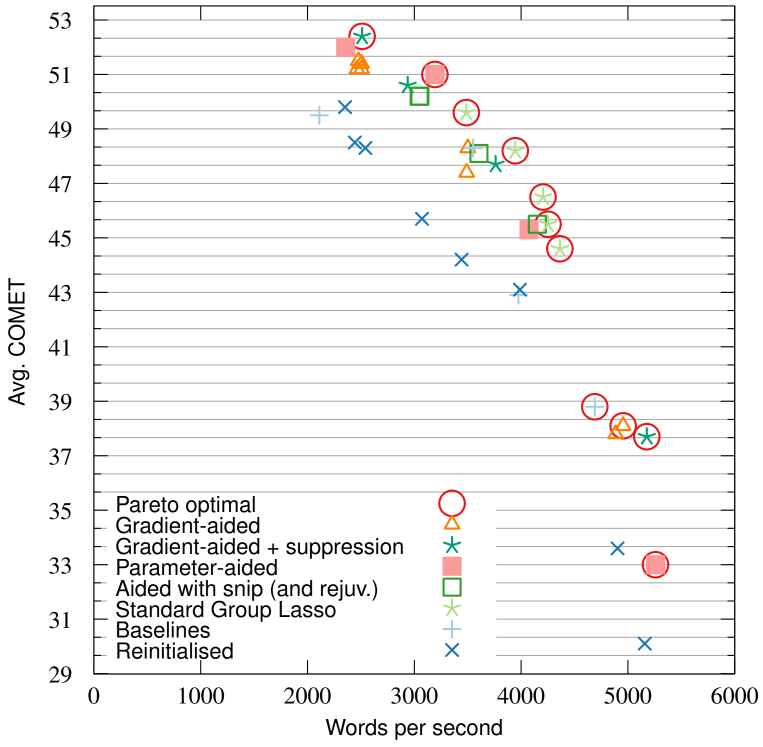
In order to offer a more nuanced perspective, I visualise Pareto trade-offs with both BLEU and COMET against the speed expressed in words per second. The plots are presented in Fig. 5.12. Here I compare various regularisation techniques and the aforementioned baselines with each other. On one side, there are new regularisation methods described in this chapter categorised as "Gradient-aided" and "Parameter-aided" and their variants, including parameter suppression and layer snipping with or without parameter rejuvenation.

	Reg. $\lambda \longrightarrow$	Base		Base $1/2$		Base $1/4$		Base $1/8$	
	Layer type $\longrightarrow$	FFN	Heads	FFN	Heads	FFN	Heads	FFN	Heads
	Enc. L1–12	1536	8	768	4	384	2	192	1
	Dec. L1	1536	8	1536	8	1536	8	1536	8
Quality	BLEU	38.2		36.7		36.3		35.0	
	Δ	—		-1.5		-1.9		-3.2	
	chrF	63.9		63.0		62.4		61.6	
	COMET	49.5		48.3		42.9		38.8	
Efficiency	Att. sparsity	0%		46%		69%		81%	
	FFN sparsity	0%		46%		69%		81%	
	WPS	2111		3550		3974		4690	
	Speed-up	1.00		1.68		1.88		2.22	

Table 5.11: The results of "12–1" transformer baseline models for English→German with simply reduced dimensions in the encoder.



(a) BLEU vs WPS



(b) COMET vs WPS

Figure 5.12: Pareto trade-off for English→German students with “12–1” architecture pruned with aided regularisation methods compared to the baselines, models trained from scratch and those regularised with the standard group lasso.

Due to how sharply γ functions scale penalties up, it is tricky to cover the frontier evenly. Still, the results are positive. There is a notable gap between the pruned and the smaller models trained from scratch. Many pruned models within the proximity of about 2500 to 3000 WPS outperform or maintain the baseline quality that oscillates around 2100 WPS. The pruned models dominate the centre of the plots. The standard group lasso proves to be a reliable method that leads the Pareto frontier, especially in COMET. Less sparsified attention in those models (only up to 50%) is a possible explanation for that outcome. On the other hand, this further solidifies the positive results of the previous research. The most pruned models are also the fastest models on the frontier, such as those with suppressed parameters during training. The smallest baseline model is competitive in this case as well.

5.11 Conclusion

In this chapter, I introduced a novel approach to regularisation, in which external information aids its penalties by scaling them accordingly. Both variants that use the magnitude of gradients and parameters amplify the sparsity patterns exhibited by a standard group lasso. When sparsified, a transformer aims to reduce its architectural depth by focusing on pruning and collapsing middle layers. Most importantly, the analysis of the aided regularisation methods clearly shows that they result in a significant inference speed-up that pushes the state-of-the-art highly optimised architectures forward. For example, a model can get $1.8\times$ faster at the cost of -0.5 BLEU point. Additional experiments on rejuvenating pruned parameters demonstrate that it is possible to entirely remove 5 out of 12 encoder layers with no quality loss as the other layers have an opportunity to reactivate without any penalties enforced.

The work in this chapter further supports regularisation as a valid pruning method incorporated into a training scheme that does not prolong a total training time. Empirically speaking, it can even shorten it as small architectures trained from scratch often take many more epochs to converge in a knowledge-distilled setting. At the same time, it is shorter when a larger model gets sparsified along the way. I strongly believe that the research outlined in this chapter lies a solid foundation for pruning methods for efficient machine translation and sets the direction beyond that for the future.

Chapter 6

Conclusions

Throughout this work, I pursued the goal of improving the inference speed of NMT models with no complicated sparsity support needed. I achieved that by enforcing structural sparsity patterns that easily allow for slicing and reducing the size of a model while still keeping it dense and straightforward to use. Moreover, I closed the gap in the pruning research field of the failure to build upon strong baselines, provide transparency on actual, not theoretical, speed improvements, and make sparse models faster, not just smaller.

Each chapter of this thesis concluded with an extensive Pareto analysis, demonstrating that their experiments actually advanced the frontier of the best state-of-the-art efficient models. I explored various pruning methods such as the Lottery Ticket Hypothesis or many regularisation techniques, and I improved them to suit my agenda to progress with speed efficiency. The research presented in this thesis empirically proves that it is possible to structurally prune small high-end NMT models in ways that maintain their translation quality while at the same time making them considerably faster with no compromise on deployment time.

This project challenged the lack of proper speed analysis perpetuated in the pruning field and delivered a strong directive for current and future work. Such contributions are more important than ever with the ever-growing size of deep neural networks and increased demand for small and fast models deployed on CPUs in mobile or PC environments. There are many possible directions to take my research forward as there is no end game for boosting speed or quality. New, more sophisticated heuristics may exist that could take over the optimisation reins as they better help a model maintain its quality with higher sparsity levels achieved.

There is still an open question for the best “starting point” architecture that suits pruning best. Despite positive reports recommending shifting layers from a decoder to an encoder, extensive analysis in quality casts doubt that it is the case. After all, using a deeper model and pruning it better may be the way to go.

There is also a challenge of improving a ratio between perplexity and regularisation penalties in dynamic batching, which is a prevalent approach in natural language processing. Refining that balance would allow for a better training flow and possibly find better global minima during gradient optimisation.

Bibliography

Aji, A. F., & Heafield, K. (2020, July). Compressing neural machine translation models with 4-bit precision. In *Proceedings of the fourth workshop on neural generation and translation* (pp. 35–42). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2020.ngt-1.4` doi: 10.18653/v1/2020.ngt-1.4

Aji, A. F., Heafield, K., & Bogoychev, N. (2019, November). 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)* (pp. 3626–3631). Hong Kong, China: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/D19-1373` doi: 10.18653/v1/D19-1373

Akhbardeh, F., Arkhangorodsky, A., Biesialska, M., Bojar, O., Chatterjee, R., Chaudhary, V., ... Zampieri, M. (2021, November). Findings of the 2021 conference on machine translation (WMT21). In *Proceedings of the sixth conference on machine translation* (pp. 1–88). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2021.wmt-1.1`

Alammar, J. (2018). *The illustrated transformer.* Retrieved 07.05.2020, from `http://jalammar.github.io/illustrated-transformer/`

Artetxe, M., Bhosale, S., Goyal, N., Mihaylov, T., Ott, M., Shleifer, S., ... Stoyanov, V. (2021). *Efficient large scale language modeling with mixtures of experts.* arXiv. Retrieved from `https://arxiv.org/abs/2112.10684` doi: 10.48550/ARXIV.2112.10684

Barrault, L., Biesialska, M., Bojar, O., Costa-jussà, M. R., Federmann, C., Graham, Y., ... Zampieri, M. (2020, November). Findings of the 2020 conference on machine translation (WMT20). In *Proceedings of the fifth conference on machine translation* (pp. 1–55). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2020.wmt-1.1`

Barrault, L., Bojar, O., Costa-jussà, M. R., Federmann, C., Fishel, M., Graham, Y., ... Zampieri, M. (2019, August). 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)* (pp. 1–61). Florence, Italy: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/W19-5301` doi: 10.18653/v1/W19-5301

Behnke, M., Bogoychev, N., Aji, A. F., Heafield, K., Nail, G., Zhu, Q., ... Grundkiewicz, R. (2021, November). Efficient machine translation with model pruning and quantization. In *Proceedings of the sixth conference on machine translation* (pp. 775–780). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2021.wmt-1.74`

Behnke, M., & Heafield, K. (2020, November). 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)* (pp. 2664–2674). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2020.emnlp-main.211` doi: 10.18653/v1/2020.emnlp-main.211

Behnke, M., & Heafield, K. (2021, November). Pruning neural machine translation for speed using group lasso. In *Proceedings of the sixth conference on machine translation* (pp. 1074–1086). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2021.wmt-1.116`

Blalock, D. W., Ortiz, J. J. G., Frankle, J., & Guttag, J. V. (2020). What is the state of neural network pruning? *CoRR*, *abs/2003.03033*. Retrieved from `https://arxiv.org/abs/2003.03033`

Bogoychev, N. (2021, November). Not all parameters are born equal: Attention is mostly what you need. In *Proceedings of the fourth blackboxnlp workshop on analyzing and interpreting neural networks for nlp* (pp. 363–374). Punta Cana, Dominican Republic: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2021.blackboxnlp-1.28` doi: 10.18653/v1/2021.blackboxnlp-1.28

Bogoychev, N., Grundkiewicz, R., Aji, A. F., Behnke, M., Heafield, K., Kashyap, S., ... Chudyk, M. (2020, July). Edinburgh's submissions to the 2020 machine translation efficiency task. In *Proceedings of the fourth workshop on neural generation and translation* (pp. 218–224). Online: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/2020.ngt-1.26` doi: 10.18653/v1/2020.ngt-1.26

Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Leveling, J., ... Tamchyna, A. (n.d.).

Bojar, O., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huang, S., ... Turchi, M. (2017, September). Findings of the 2017 conference on machine translation (WMT17). In *Proceedings of the second conference on machine translation* (pp. 169–214). Copenhagen, Denmark: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/W17-4717` doi: 10.18653/v1/W17-4717

Bojar, O., Federmann, C., Fishel, M., Graham, Y., Haddow, B., Koehn, P., & Monz, C. (2018, October). Findings of the 2018 conference on machine translation (WMT18). In *Proceedings of the third conference on machine translation: Shared task papers* (pp. 272–303). Belgium, Brussels: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/W18-6401` doi: 10.18653/v1/W18-6401

Brix, C., Bahar, P., & Ney, H. (2020, July). Successfully applying the stabilized lottery ticket hypothesis to the transformer architecture. In *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 3909–3915). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2020.acl-main.360` doi: 10.18653/v1/2020.acl-main.360

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., & Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, *19*(2), 263–311. Retrieved from `https://aclanthology.org/J93-2003`

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., … Amodei, D. (2020). Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (Vol. 33, pp. 1877–1901). Curran Associates, Inc. Retrieved from `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`

Buciluǎ, C., Caruana, R., & Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th acm sigkdd international conference on knowledge discovery and data mining* (p. 535–541). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/1150402.1150464` doi: 10.1145/1150402.1150464

Cettolo, M., Federico, M., Bentivogli, L., Jan, N., Sebastian, S., Katsuitho, S., … Christian, F. (2017). Overview of the IWSLT 2017 evaluation campaign. In *International workshop on spoken language translation* (pp. 2–14).

Chechik, G., Meilijson, I., & Ruppin, E. (1998). Synaptic pruning in development: A computational account. *Neural Comput*, *10*, 2418–2427.

Chen, P., Helcl, J., Germann, U., Burchell, L., Bogoychev, N., Barone, A. V. M., … Heafield, K. (2021). The University of Edinburgh's English-German and English-Hausa submissions to the WMT21 news translation task. In *Proceedings of the sixth conference on machine translation.* Online: Association for Computational Linguistics.

Child, R., Gray, S., Radford, A., & Sutskever, I. (2019). *Generating long sequences with sparse transformers.*

Courbariaux, M., & Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, *abs/1602.02830*. Retrieved from `http://arxiv.org/abs/1602.02830`

Courbariaux, M., Bengio, Y., & David, J.-P. (2015). *Training deep neural networks with low precision multiplications.*

Developer reference for Intel® oneAPI Math Kernel Library - C. (n.d.). Retrieved from `https://software.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top.html`

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, June). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)* (pp. 4171–4186). Minneapolis, Minnesota: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/N19-1423` doi: 10.18653/v1/N19-1423

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R., & Makhoul, J. (2014, June). Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 1370–1380). Baltimore, Maryland: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/P14-1129` doi: 10.3115/v1/P14-1129

Dodge, J., Schwartz, R., Peng, H., & Smith, N. A. (2019, November). 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)* (pp. 1179–1184). Hong Kong, China: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/D19-1110` doi: 10.18653/v1/D19-1110

Dong, X., Huang, J., Yang, Y., & Yan, S. (2017). More is less: A more complicated network with less inference complexity. *CoRR*, *abs/1703.08651*. Retrieved from `http://arxiv.org/abs/1703.08651`

Federmann, C. (2018, August). Appraise evaluation framework for machine translation. In *Proceedings of the 27th international conference on computational linguistics: System demonstrations* (pp. 86–88). Santa Fe, New Mexico: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/C18-2019`

Frankle, J., & Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International conference on learning representations.* Retrieved from `https://openreview.net/forum?id=rJl-b3RcF7`

Frankle, J., Dziugaite, G. K., Roy, D. M., & Carbin, M. (2019). Stabilizing the lottery ticket hypothesis. *CoRR*, *abs/1903.01611*. Retrieved from `http://arxiv.org/abs/1903.01611`

Gale, T., Elsen, E., & Hooker, S. (2019). The state of sparsity in deep neural networks. *CoRR*, *abs/1902.09574*. Retrieved from `http://arxiv.org/abs/1902.09574`

Gale, T., Zaharia, M., Young, C., & Elsen, E. (2020). Sparse GPU kernels for deep learning. In *Proceedings of the international conference for high performance computing, networking, storage and analysis.* IEEE Press.

Gao, Z.-F., Liu, P., Zhao, W. X., Lu, Z.-Y., & Wen, J.-R. (2022). *Parameter-efficient mixture-of-experts architecture for pre-trained language models.* arXiv. Retrieved from `https://arxiv.org/abs/2203.01104` doi: 10.48550/ARXIV.2203.01104

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *In proceedings of the international conference on artificial intelligence and statistics (aistats'10). society for artificial intelligence and statistics.*

Golub, M., Lemieux, G., & Lis, M. (2018). DropBack: Continuous pruning during training. *CoRR*, *abs/1806.06949*. Retrieved from `http://arxiv.org/abs/1806.06949`

Graham, Y., Baldwin, T., Moffat, A., & Zobel, J. (2013, August). Continuous measurement scales in human evaluation of machine translation. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse* (pp. 33–41). Sofia, Bulgaria: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/W13-2305`

Gray, S., Radford, A., & Kingma, D. P. (2017). GPU kernels for block-sparse weights.

Gu, J., Bradbury, J., Xiong, C., Li, V. O., & Socher, R. (2018). Non-autoregressive neural machine translation. In *International conference on learning representations.* Retrieved from `https://openreview.net/forum?id=B1l8BtlCb`

Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015, 07–09 Jul). Deep learning with limited numerical precision. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning* (Vol. 37, pp. 1737–1746). Lille, France: PMLR. Retrieved from `https://proceedings.mlr.press/v37/gupta15.html`

Haddow, B., Bogoychev, N., Emelin, D., Germann, U., Grundkiewicz, R., Heafield, K., ... Sennrich, R. (2018, October). The U niversity of Edinburgh's submissions to the WMT18 news translation task. In *Proceedings of the third conference on machine translation: Shared task papers* (pp. 399–409). Belgium, Brussels: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/W18-6412` doi: 10.18653/v1/W18-6412

Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th international conference on neural information processing systems - volume 1* (p. 1135–1143). Cambridge, MA, USA: MIT Press.

Heafield, K., Hayashi, H., Oda, Y., Konstas, I., Finch, A., Neubig, G., . . . Birch, A. (2020, July). Findings of the fourth workshop on neural generation and translation. In *Proceedings of the fourth workshop on neural generation and translation* (pp. 1–9). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2020.ngt-1.1` doi: 10.18653/v1/2020.ngt-1.1

Heafield, K., Pouzyrevsky, I., Clark, J. H., & Koehn, P. (2013, August). Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st annual meeting of the association for computational linguistics (volume 2: Short papers)* (pp. 690–696). Sofia, Bulgaria: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/P13-2121`

Heafield, K., Zhu, Q., & Grundkiewicz, R. (2021, November). Findings of the WMT 2021 shared task on efficient translation. In *Proceedings of the conference on machine translation at the 2021 conference on empirical methods in natural language processing.* Punta Cana, Dominican Republic. Retrieved from `https://kheafield.com/papers/edinburgh/wmt21-speedtask.pdf`

Hilt, D. E., & Seegrist, D. W. (1977). *Ridge, a computer program for calculating ridge regression estimates* (Vol. no.236). Upper Darby, Pa, Dept. of Agriculture, Forest Service, Northeastern Forest Experiment Station, 1977. Retrieved from `https://www.biodiversitylibrary.org/item/137258` (https://www.biodiversitylibrary.org/bibliography/68934)

Hinton, G., Vinyals, O., & Dean, J. (2015). *Distilling the knowledge in a neural network.*

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., . . . Sifre, L. (2022). *Training compute-optimal large language models.* arXiv. Retrieved from `https://arxiv.org/abs/2203.15556` doi: 10.48550/ARXIV.2203.15556

Hsu, Y.-T., Garg, S., Liao, Y.-H., & Chatsviorkin, I. (2020, November). Efficient inference for neural machine translation. In *Proceedings of sustainlp: Workshop on simple and efficient natural language processing* (pp. 48–53). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2020.sustainlp-1.7` doi: 10.18653/v1/2020.sustainlp-1.7

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, M. X., Chen, D., . . . Chen, Z. (2019). *GPipe: Efficient training of giant neural networks using pipeline parallelism.*

Inan, H., Khosravi, K., & Socher, R. (2016). Tying word vectors and word classifiers: A loss framework for language modeling. *CoRR*, *abs/1611.01462*. Retrieved from `http://arxiv.org/abs/1611.01462`

Informatik, F., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2003, 03). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... Kalenichenko, D. (2018, June). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr).*

Junczys-Dowmunt, M. (2018, October). Microsoft's submission to the WMT2018 news translation task: How I learned to stop worrying and love the data. In *Proceedings of the third conference on machine translation: Shared task papers* (pp. 425–430). Belgium, Brussels: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/W18-6415` doi: 10.18653/v1/W18-6415

Junczys-Dowmunt, M. (2019, August). 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)* (pp. 225–233). Florence, Italy: Association for Computational Linguistics. Retrieved from `http://www.aclweb.org/anthology/W19-5321`

Junczys-Dowmunt, M., Grundkiewicz, R., Dwojak, T., Hoang, H., Heafield, K., Neckermann, T., ... Birch, A. (2018, July). Marian: Fast neural machine translation in C++. In *Proceedings of acl 2018, system demonstrations* (pp. 116–121). Melbourne, Australia: Association for Computational Linguistics. Retrieved from `http://www.aclweb.org/anthology/P18-4020`

Junczys-Dowmunt, M., Heafield, K., Hoang, H., Grundkiewicz, R., & Aue, A. (2018, July). Marian: Cost-effective high-quality neural machine translation in C++. In *Proceedings of the 2nd workshop on neural machine translation and generation* (pp. 129–135). Melbourne, Australia: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/W18-2716` doi: 10.18653/v1/W18-2716

Kasai, J., Pappas, N., Peng, H., Cross, J., & Smith, N. A. (2020). *Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation.*

Khudia, D. S., Huang, J., Basu, P., Deng, S., Liu, H., Park, J., & Smelyanskiy, M. (2021). FBGEMM: enabling high-performance low-precision deep learning inference. *CoRR*, *abs/2101.05615*. Retrieved from `https://arxiv.org/abs/2101.05615`

Kim, Y., & Rush, A. M. (2016a, November). Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing* (pp. 1317–1327). Austin, Texas: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/D16-1139` doi: 10.18653/v1/D16-1139

Kim, Y., & Rush, A. M. (2016b, November). Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing* (pp. 1317–1327). Austin, Texas: Association for Computational Linguistics. Retrieved from https://aclanthology.org/D16-1139 doi: 10.18653/v1/D16-1139

Kim, Y. J., Awan, A. A., Muzio, A., Cruz Salinas, F., Lu, L., Hendy, A., . . . Hassan Awadalla, H. (2021, September). *Scalable and efficient moe training for multitask multilingual models.* Retrieved from https://www.microsoft.com/en-us/research/publication/scalable-and-efficient-moe-training-for-multitask-multilingual-models/

Kim, Y. J., Junczys-Dowmunt, M., Hassan, H., Fikri Aji, A., Heafield, K., Grundkiewicz, R., & Bogoychev, N. (2019, November). From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd workshop on neural generation and translation* (pp. 280–288). Hong Kong: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/D19-5632 doi: 10.18653/v1/D19-5632

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kocmi, T., Federmann, C., Grundkiewicz, R., Junczys-Dowmunt, M., Matsushita, H., & Menezes, A. (2021, November). To ship or not to ship: An extensive evaluation of automatic metrics for machine translation. In *Proceedings of the sixth conference on machine translation* (pp. 478–494). Online: Association for Computational Linguistics. Retrieved from https://aclanthology.org/2021.wmt-1.57

Kudo, T., & Richardson, J. (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* (pp. 66–71). Retrieved from https://arxiv.org/abs/1808.06226

Le, H. S., Allauzen, A., & Yvon, F. (2012, June). Continuous space translation models with neural networks. In *Proceedings of the 2012 conference of the north American chapter of the association for computational linguistics: Human language technologies* (pp. 39–48). Montréal, Canada: Association for Computational Linguistics. Retrieved from https://aclanthology.org/N12-1005

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989, 12). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541-551. Retrieved from https://doi.org/10.1162/neco.1989.1.4.541 doi: 10.1162/neco.1989.1.4.541

LeCun, Y., Denker, J., & Solla, S. (1990). Optimal brain damage. In D. Touretzky (Ed.), *Advances in neural information processing systems* (Vol. 2). Morgan-Kaufmann. Retrieved from `https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf`

Lee, J., Shu, R., & Cho, K. (2020, November). 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)* (pp. 1006–1015). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2020.emnlp-main.73` doi: 10.18653/v1/2020.emnlp-main.73

Lee, N., Ajanthan, T., & Torr, P. (2019). SNIP: Single-Shot network pruning based on connection sensitivity. In *International conference on learning representations.* Retrieved from `https://openreview.net/forum?id=B1VZqjAcYX`

Lei, T., Zhang, Y., Wang, S. I., Dai, H., & Artzi, Y. (2017). *Simple recurrent units for highly parallelizable recurrence.* arXiv. Retrieved from `https://arxiv.org/abs/1709.02755` doi: 10.48550/ARXIV.1709.02755

Lemaire, C., Achkar, A., & Jodoin, P. (2018). Structured pruning of neural networks with budget-aware regularization. *CoRR*, *abs/1811.09332*. Retrieved from `http://arxiv.org/abs/1811.09332`

Li, H., De, S., Xu, Z., Studer, C., Samet, H., & Goldstein, T. (2017). Training quantized nets: A deeper understanding. In I. Guyon et al. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. Retrieved from `https://proceedings.neurips.cc/paper/2017/file/1c303b0eed3133200cf715285011b4e4-Paper.pdf`

Li, J., Cotterell, R., & Sachan, M. (2021). Differentiable subset pruning of transformer heads. *CoRR*, *abs/2108.04657*. Retrieved from `https://arxiv.org/abs/2108.04657`

Liu, L., Liu, X., Gao, J., Chen, W., & Han, J. (2020, November). Understanding the difficulty of training transformers. In *Proceedings of the 2020 conference on empirical methods in natural language processing (emnlp)* (pp. 5747–5763). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2020.emnlp-main.463` doi: 10.18653/v1/2020.emnlp-main.463

Liu, Z., Sun, M., Zhou, T., Huang, G., & Darrell, T. (2018). Rethinking the value of network pruning. *CoRR*, *abs/1810.05270*. Retrieved from `http://arxiv.org/abs/1810.05270`

Louizos, C., Welling, M., & Kingma, D. P. (2018). *Learning sparse neural networks through l_0 regularization.*

Michel, P., Levy, O., & Neubig, G. (2019). Are sixteen heads really better than one? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 14014–14024). Curran Associates, Inc. Retrieved from `http://papers.nips.cc/paper/9551-are-sixteen-heads-really-better-than-one.pdf`

Miyashita, D., Lee, E. H., & Murmann, B. (2016). Convolutional neural networks using logarithmic data representation. *CoRR*, *abs/1603.01025*. Retrieved from `http://arxiv.org/abs/1603.01025`

Molchanov, P., Mallya, A., Tyree, S., Frosio, I., & Kautz, J. (2019). Importance estimation for neural network pruning. *CoRR*, *abs/1906.10771*. Retrieved from `http://arxiv.org/abs/1906.10771`

Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2016). *Pruning convolutional neural networks for resource efficient inference.* arXiv. Retrieved from `https://arxiv.org/abs/1611.06440` doi: 10.48550/ARXIV.1611.06440

Morcos, A. S., Yu, H., Paganini, M., & Tian, Y. (2019). One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *arXiv preprint arXiv:1906.02773*.

Murray, K., DuSell, B., & Chiang, D. (2019, November). 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* (pp. 297–301). Hong Kong: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/D19-5634` doi: 10.18653/v1/D19-5634

Narang, S., Undersander, E., & Diamos, G. F. (2017). Block-sparse recurrent neural networks. *CoRR*, *abs/1711.02782*. Retrieved from `http://arxiv.org/abs/1711.02782`

Nguyen, T. Q., & Salazar, J. (2019). Transformers without tears: Improving the normalization of self-attention. *CoRR*, *abs/1910.05895*. Retrieved from `http://arxiv.org/abs/1910.05895`

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002, July). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the association for computational linguistics* (pp. 311–318). Philadelphia, Pennsylvania, USA: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/P02-1040` doi: 10.3115/1073083.1073135

Parikh, N., & Boyd, S. (2014, January). Proximal algorithms. *Found. Trends Optim.*, *1*(3), 127–239. Retrieved from `https://doi.org/10.1561/2400000003` doi: 10.1561/2400000003

Popović, M. (2017, September). chrF++: words helping character n-grams. In *Proceedings of the second conference on machine translation* (pp. 612–618). Copenhagen, Denmark: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/W17-4770` doi: 10.18653/v1/W17-4770

Post, M. (2018, October). A call for clarity in reporting BLEU scores. In *Proceedings of the third conference on machine translation: Research papers* (pp. 186–191). Brussels, Belgium: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/W18-6319` doi: 10.18653/v1/W18-6319

Press, O., & Wolf, L. (2017, April). Using the output embedding to improve language models. In *Proceedings of the 15th conference of the European chapter of the association for computational linguistics: Volume 2, short papers* (pp. 157–163). Valencia, Spain: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/E17-2025`

Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). XNOR-Net: ImageNet classification using binary convolutional neural networks. *CoRR*, *abs/1603.05279*. Retrieved from `http://arxiv.org/abs/1603.05279`

Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 33, pp. 4780–4789).

Rei, R., Stewart, C., Farinha, A. C., & Lavie, A. (2020, November). COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 conference on empirical methods in natural language processing (emnlp)* (pp. 2685–2702). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2020.emnlp-main.213` doi: 10.18653/v1/2020.emnlp-main.213

Sajjad, H., Dalvi, F., Durrani, N., & Nakov, P. (2020). *Poor man's bert: Smaller and faster transformer models.*

Scardapane, S., Comminiello, D., Hussain, A., & Uncini, A. (2017, Jun). Group sparse regularization for deep neural networks. *Neurocomputing*, *241*, 81–89. Retrieved from `http://dx.doi.org/10.1016/j.neucom.2017.02.029` doi: 10.1016/j.neucom.2017.02.029

Schwenk, H., R. Costa-jussà, M., & R. Fonollosa, J. A. (2007, June). Smooth bilingual n-gram translation. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)* (pp. 430–438). Prague, Czech Republic: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/D07-1045`

See, A., Luong, M.-T., & Manning, C. D. (2016, August). Compression of neural machine translation models via pruning. In *Proceedings of the 20th SIGNLL conference on computational natural language learning* (pp. 291–301). Berlin, Germany: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/K16-1029` doi: 10.18653/v1/K16-1029

Sennrich, R., Haddow, B., & Birch, A. (2016, August). Neural machine translation of rare words with subword units. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 1715–1725). Berlin, Germany: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/P16-1162` doi: 10.18653/v1/P16-1162

Serrano, S., & Smith, N. A. (2019, July). Is attention interpretable? In *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 2931–2951). Florence, Italy: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/P19-1282` doi: 10.18653/v1/P19-1282

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). *Outrageously large neural networks: The sparsely-gated mixture-of-experts layer.*

Simon, N., & Tibshirani, R. (2012). Standardization and the group lasso penalty. *Statistica Sinica*, *22*(3), 983.

Siswanto, A. E. (2021). *Block sparsity and weight initialization in neural network pruning* (Thesis, Massachusetts Institute of Technology). Retrieved 2022-05-30, from `https://dspace.mit.edu/handle/1721.1/130708` (Accepted: 2021-05-24T19:52:33Z)

Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., ... Catanzaro, B. (2022). Using deepspeed and megatron to train megatron-turing NLG 530b, A large-scale generative language model. *CoRR*, *abs/2201.11990*. Retrieved from `https://arxiv.org/abs/2201.11990`

So, D. R., Liang, C., & Le, Q. V. (2019). *The evolved transformer.* arXiv. Retrieved from `https://arxiv.org/abs/1901.11117` doi: 10.48550/ARXIV.1901.11117

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, *58*, 267-288.

Vaswani, A., Bengio, S., Brevdo, E., Chollet, F., Gomez, A. N., Gouws, S., ... Uszkoreit, J. (2018). Tensor2tensor for neural machine translation. *CoRR*, *abs/1803.07416*. Retrieved from `http://arxiv.org/abs/1803.07416`

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017a). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017b). Attention is all you need. In I. Guyon et al. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., & Titov, I. (2019, July). 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). Florence, Italy: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/P19-1580

Wang, C., Hu, C., Mu, Y., Yan, Z., Wu, S., Hu, Y., ... Zhu, J. (2021, November). The NiuTrans system for the WMT 2021 efficiency task. In *Proceedings of the sixth conference on machine translation* (pp. 787–794). Online: Association for Computational Linguistics. Retrieved from https://aclanthology.org/2021.wmt-1.76

Wang, C., Zhang, G., & Grosse, R. (2020). *Picking winning tickets before training by preserving gradient flow.*

Wang, Y., Wang, L., Li, V., & Tu, Z. (2020, November). On the sparsity of neural machine translation models. In *Proceedings of the 2020 conference on empirical methods in natural language processing (emnlp)* (pp. 1060–1066). Online: Association for Computational Linguistics. Retrieved from https://aclanthology.org/2020.emnlp-main.78 doi: 10.18653/v1/2020.emnlp-main.78

Wen, W., Wu, C., Wang, Y., Chen, Y., & Li, H. (2016). Learning structured sparsity in deep neural networks. In *Proceedings of the 30th international conference on neural information processing systems* (p. 2082–2090). Red Hook, NY, USA: Curran Associates Inc.

Wuebker, J., Simianer, P., & DeNero, J. (2018, October-November). Compact personalized models for neural machine translation. In *Proceedings of the 2018 conference on empirical methods in natural language processing* (pp. 881–886). Brussels, Belgium: Association for Computational Linguistics. Retrieved from https://aclanthology.org/D18-1104 doi: 10.18653/v1/D18-1104

Xiao, T., Li, Y., Zhu, J., Yu, Z., & Liu, T. (2019, 7). Sharing attention weights for fast transformer. In *Proceedings of the twenty-eighth international joint conference on artificial intelligence, IJCAI-19* (pp. 5292–5298). International Joint Conferences on Artificial Intelligence Organization. Retrieved from https://doi.org/10.24963/ijcai.2019/735 doi: 10.24963/ijcai.2019/735

Yao, Z., Cao, S., Xiao, W., Zhang, C., & Nie, L. (2019, Jul). Balanced sparsity for efficient dnn inference on gpu. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 5676–5683. Retrieved from `http://dx.doi.org/10.1609/aaai.v33i01.33015676` doi: 10.1609/aaai.v33i01.33015676

Yu, H., Edunov, S., Tian, Y., & Morcos, A. S. (2020). Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. In *International conference on learning representations.* Retrieved from `https://openreview.net/forum?id=S1xnXRVFwH`

Yuan, M., & Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society, Series B*, *68*, 49–67.

Zhang, B., Xiong, D., & Su, J. (2018, July). Accelerating neural transformer via an average attention network. In *Proceedings of the 56th annual meeting of the association for computational linguistics.* Melbourne, Australia: Association for Computational Linguistics.

Zhu, M., & Gupta, S. (2017). *To prune, or not to prune: exploring the efficacy of pruning for model compression.*

Zoph, B., & Le, Q. V. (2016). *Neural architecture search with reinforcement learning.*