Adaptive Parameter Compression for Language Models

Jeremias Bohn and Frederic Mrozinski and Georg Groh
Technical University of Munich
School of Computation, Information and Technology
Germany
{jeremias.bohn, frederic.mrozinski}@tum.de, grohg@cit.tum.de

Abstract

Over the last years, state-of-the-art AI models have grown to a point where their use bears significant economic and environmental cost. At the same time, investigation of NLP models has shown that they are often overparameterized, giving rise to research of compression approaches. Such approaches often suffer the trade-off between hardware requirements and classification performance. In this work, we propose the hardware-independent compression strategy Adaptive Parameter Compression (APC). We extend the Weight Squeezing approach by introducing compression biases and weights, as well as investigating multiple initialization strategies for these weights and the application of APC to transformer model components. Experiments with $\text{BERT}_{\text{base}}$ show the compression's effectiveness, slightly outperforming DistilBERT while being significantly more efficient.

1 Introduction

Since the introduction of the Transformer architecture (Vaswani et al., 2017) and its widespread adoption in the NLP community, significant performance gains were achieved by increasing the parameter count of models, with OpenAI's GPT-2 using about 1.5 billion parameters (Radford et al., 2019) and growing to 175 billion in the next iteration GPT-3 (Brown et al., 2020), as well as Megatron Turing NLG (530 billion parameters) (Smith et al., 2022) and PaLM (540 billion parameters) (Chowdhery et al., 2023). This race for the largest language model had recently culminated in GPT-4 which, while officially not disclosed, is estimated to comprise a total of 1.8 trillion parameters (Schreiner, 2023). While these models usually outperform their predecessors in various benchmark tests, the efficiency of this exponential growth is questionable, in particular when considering the economical and ecological costs of training and using these models; e.g., the training of GPT-3 is estimated to have emitted about 550 tons CO_2e (Patterson et al., 2021) and consumed about 700 tons of water (Li et al., 2022). Further, the increase in model size sets a high bar for their training and inference, resulting in strong dependencies on a small number of tech companies (Kak and Myers Wes, 2023), which limits independent research and bears great ethical risks (Müller, 2020).

However, as the Lottery Ticket Hypothesis by Frankle and Carbin (2019) suggests, dense neural networks (as used in the Transformer architecture) contain small subnetworks which, when trained solely without the remaining weights, can match the performance of a fully trained network, which could indicate that at least for inference a great amount of the weights are not necessary. In fact, multiple approaches have shown that parameters can be removed or nullified to a great extent with small performance sacrifices (e.g. Zafrir et al., 2021; Michel et al., 2019; Elkerdawy et al., 2020).

In our work, we present Adaptive Parameter Compression (hereafter referred to as APC), an algorithm based on Weight Squeezing (Chumachenko et al., 2020), which we show is a generalization of neuron pruning (e.g. Jiang et al., 2018) and neuron merging (e.g. Yvinec et al., 2023). APC removes weights by "rewiring" connections with varying compression ratios across different layers, allowing us to adapt to the complexity of feature extraction at each layer. To this end, we introduce projection matrices which are initialized so that they locally reconstruct the output optimally and are then globally optimized by gradient descent on the pretraining task. With our experiments, we show that APC is on par with similar compression techniques in terms of performance, but poses no constraints with respect to hardware specializations and finds suitable compressions quickly.

Findings of the Association for Computational Linguistics:
NAACL 2025, pages 7284–7301

2 Background & Related Work

2.1 Compression Techniques

Since the training and inference of Deep Neural Networks (DNNs) was and still is expensive, the study of methods to reduce model sizes has been around for decades (e.g., Janowsky, 1989; LeCun et al., 1989). We briefly describe some common compression techniques and their relation to APC.

Pruning Pruning is a technique to remove weights which have the least effect on the model's performance. Commonly, the model is fine-tuned after the removal of weights to "patch" it, mitigating the negative effect of pruning. We can distinguish between unstructured and structured pruning: Unstructured pruning was already discussed in early works (LeCun et al., 1989; Hassibi et al., 1993), which measured performance degradation by pruning parameters by means of the effect on the loss's second order derivatives (which are expensive to calculate). An approximation of the second order derivatives can be calculated via Fisher information, as used in more recent approaches (Tu et al., 2016; Molchanov et al., 2019; Theis et al., 2018). Other popular techniques use a magnitude-based importance measurement, typically the l_1 norm (Han et al., 2015; Frankle and Carbin, 2019), which minimizes the Frobenius norm of the difference between the weight matrix before and after pruning. While unstructured pruning yields high compression ratios with minimal performance loss (e.g., Zafrir et al.'s (2021) BERT model with 90% pruning ratio), exploiting these sparse matrices is non-trivial and either requires specialized hardware (Mishra et al., 2021) or cannot gain similar inference acceleration to structured pruning when optimizing with software only (Wang, 2021).

Structured pruning constrains the pruning mask to specific structures, e.g. to block structures in weight matrices (Chandy et al., 2023), filters in convolutional neural networks (CNNs) (Li et al., 2016; Hu et al., 2016) (extendable to single neurons), attention heads (Michel et al., 2019), and entire layers (O'Neill et al., 2020). A notable neuron pruning approach is described by Molchanov et al. (2016) which defines a mask $M \coloneqq \mathbb{1} \in \mathbb{R}^{\text{out}}$ and changes the output of a linear layer to $M \odot (x^\top W + b^\top)$ so that M's gradient provides insight into the effect of pruning a neuron. This approach, to which we will refer to as *neuron sensitivity pruning*, has also been used by Prasanna et al. (2020).

A special case of neuron pruning is neuron merging, which identifies "similar" neurons (e.g. by clustering) which are then collapsed into a single neuron (Srinivas and Babu, 2015; Zhong et al., 2018).

Since models edited with structured pruning are in most cases indistinguishable from models with different dimension sizes, their pruning ratio usually corresponds to an asymptotically identical inference acceleration, but the given constraints on pruning selection affect performance stronger than unstructured pruning. Neuron pruning and hence also neuron merging are special cases of APC, as described in Subsection 4.2.

Knowledge Distillation Knowledge Distillation, also known as Teacher-Student-Training, was introduced by Hinton et al. (2015). The core idea is to imitate a (smoothed) probability distribution of a teacher model (of large size) using a new student model (which can be of smaller size), e.g., by matching the teacher's logits. Thus, the student does not learn directly from the hard gold label standard of the data, but instead reproduces a softer distribution which is, according to Hinton et al. (2015), easier and results in superior performance compared to training a small model without knowledge distillation. Since then, knowledge distillation has been adapted to not only match the final output, but also aligning intermediate hidden representations (Zagoruyko and Komodakis, 2017), often by introducing alignment weights (which can be discarded after training) to match sizes between the student and teacher models' hidden embeddings (Romero et al., 2015; Li et al., 2020; Zhou et al., 2020).

Knowledge Distillation was extensively applied to language models, in particular BERT (Devlin et al., 2019), e.g., the well-known DistilBERT (Sanh et al., 2019) which considered the masked language modelling (MLM) task and removed half of $\text{BERT}_{\text{base}}$'s layers, Turc et al.'s (2019) models which also considered the next sentence prediction (NSP) for distillation and further pre-trained student models, and Weight Squeezing (Chumachenko et al., 2020), the algorithm APC extends, which uses knowledge distillation on the final outputs to learn a reparameterization by means of projections of the original weight matrices instead of learning a freshly initialized student model.

An advantage of knowledge distillation is that it is independent of other compression techniques described here, allowing to combine strategies, e.g.,

with neuron pruning (Mao et al., 2020) or unstructured pruning (Zafrir et al., 2021).

Quantization A major current research focus of model compression is quantization, which aims at reducing the precision of values used in computation to reduce model size, training time and latency. In general, quantization approaches reduce the precision of the weights of a model (e.g. Courbariaux et al., 2015; Zafrir et al., 2019; Zadeh et al., 2020; Razani et al., 2021), which allows to reduce weights in LLMs to as low as a ternary representation without performance loss (Ma et al., 2024). However, to effectively make use of quantization at inference, sometimes specific hardware is needed which is suitable for the used precision level.

APC's design is orthogonal to quantization, thus quantization can be applied after APC or vice versa.

2.2 Layer Importance

In our proposed method, we can choose the compression ratio for each layer independently. Since the hyperparameter space is commonly large and a direct search or other methods are cumbersome (White et al., 2023), we aim to group layers with similar importance, which serves as an indicator for compressibility (Zhang et al., 2022; Michel et al., 2019; Elkerdawy et al., 2020; O'Neill et al., 2020). To this end, we use three metrics: weight imprinting, layer dropping, and fisher information.

Weight Imprinting Proposed by Qi et al. (2018) to extend an existing classifier by an additional unseen class, the authors argue that in a normalized output the new class can be well-separated from the already learnt classes, allowing us to add the expected embedding of the output as an additional column to our weight matrix. By adding a classifier that discriminates the expected hidden embeddings for all output classes after each layer and measuring these classifier's relative performance differences with that of the previous layer, we can calculate an importance measurement for each layer (Elkerdawy et al., 2020, 2021; Liu et al., 2021).

Layer Dropping Layer dropping is a simple importance heuristic which removes an entire layer and compares the performance of this model with the full model, motivated by the assumption that layers which contribute strongly to a classification transform their input to a significantly different sub-manifold of the data space (Sajjad et al., 2023). Note that this approach fails if the input and output dimensions of a layer do not match.

Chatterji et al. (2019) and Zhang et al. (2022) propose to instead replace the layer with its state after initialization for a more accurate score. However, having access to the original initialized values of foundation models is rare, which makes this approach only viable when training from scratch.

Fisher Information As described in Subsection 2.1, Fisher information has been used in unstructured pruning to identify weights with low effect on the performance as it provides an estimate of the second order derivative for converged models. By averaging over the fisher information of all weights in a layer, we get an approximation of the compressibility of a layer since many "unimportant" weights and few "important" weights usually yield a smaller average value, indicating a higher compression tolerance.

3 Adaptive Parameter Compression

APC is a projection-based compression algorithm which allows us to learn lower-dimensional approximations of the parameter space of a model.

3.1 General Structure

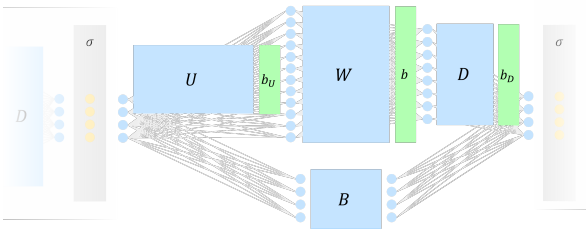


Figure 1: Adaptive Parameter Compression of a single intermediate perceptron layer.

Consider an affine-linear layer l defined by $Y = XW + b$ with $Y \in \mathbb{R}^{\text{batch}\times\text{out}}$, $X \in \mathbb{R}^{\text{batch}\times\text{in}}$, $W \in \mathbb{R}^{\text{in}\times\text{out}}$, and $b \in \mathbb{R}^{\text{out}}$. Then for some compressed input dimensionality $\text{cmp}_{l-1} < \text{in}$ and compressed output dimensionality $\text{cmp}_l < \text{out}$, we introduce trainable tensors $U \in \mathbb{R}^{\text{cmp}_{l-1}\times\text{in}}$, $D \in \mathbb{R}^{\text{out}\times\text{cmp}_l}$, $B \in \mathbb{R}^{\text{cmp}_{l-1}\times\text{cmp}_l}$, $b_U \in \mathbb{R}^{\text{in}}$, and $b_D \in \mathbb{R}^{\text{out}}$ to transform the layer l to $\check{Y} = \check{X}UWD + \check{X}B + D^\top W^\top b_U + D^\top b + b_D$ where $\check{Y}, \check{X}$ denote compressed outputs and inputs, respectively (see also Figure 1). Then with

$$\check{W} := UWD + B \tag{1}$$

$$\check{b} := D^\top W^\top b_U + D^\top b + b_D \tag{2}$$

we can compute compressed parameters $\check{W} \in \mathbb{R}^{\text{cmp}_{l-1} \times \text{cmp}_l}$, $\check{b} \in \mathbb{R}^{\text{cmp}_l}$ so that $\check{Y} = \check{X}\check{W} + \check{b}$. We refer to a model with computed $\check{W}$ and $\check{b}$ as *finalized*. Thus, during compression training (before finalization), the model has access to all original weights W and b. After its finalization, the model is actually compressed for faster inference.

Now consider an activation function σ with $X^{l+1} = \sigma(X^l W + b)$. From a reconstruction perspective, APC calculates

$$\widetilde{X}^{l+1} = \sigma((X^l W + b)D + b_D + X^l \widetilde{B})U + b_u \quad (3)$$

for $B = U^l \widetilde{B}$ (since X^l is not compressed here), which approximates X^{l+1} with a bottleneck at the activation function. This motivates the error term

$$\mathbb{E}_{X^l} \left\| X^{l+1} - \widetilde{X}^{l+1} \right\|_F, \quad (4)$$

hereafter referred to as reconstruction error, which indicates that our compressed network locally models a similar function as the uncompressed one. Since we introduce the bias terms b_D and b_U (unlike Chumachenko et al. (2020)), we map to a larger set of hyperplanes, theoretically allowing for a lower reconstruction error.

The introduction of the corrective weight bias B is motivated by the fact that the function $c_W(U, D) = UWD$ is generally not surjective, hence an optimal compressed weight matrix $\check{W}$ is not necessarily in the image of c_W, which can be compensated for with B. Further, this serves as a gateway to pass the information flow directly from layer $l-1$ to layer $l+1$ (or vice versa for gradients).

In experiments, we found that using the biases b_D, b_U, and B significantly improves the compressed model's performance, see Appendix G.

3.2 APC Compression Types for Transformer Blocks

So far we have only discussed APC for Multi Layer Perceptrons (MLP), which allows a straightforward pairing of D and U: As they are positioned around the bottleneck, they can be jointly optimized. In a Transformer, however, we can distinguish between the size of the hidden embedding within each block and further the size within the Feed Forward Network (FFN) and the attention heads, compressing them separately. Each compression is orthogonal to the others, allowing us to combine them freely afterward.

For simplicity, we will refer to each compression only by its matrices U, D, but implicitly also add the biases and the bias correction matrix to every pair of following U and D, as seen in Figure 1.

Embedding Compression Embedding Compression reduces the hidden size inside a Transformer block without affecting the dimensionality of FFNs or attention heads (see Figure 2). Since we have residual connections within the block, we must ensure the dimension stays the same within it.

To this end, for an embedding compression dimension cmp^e we introduce the pre- and post-scaling matrices D^{PS}, U^{PS}, outer attention upscaling U^V, U^K, U^Q (which are shared between all attention heads) and downscaling D^O, as well as U^α and U^β for the 2-layer FFN in the standard Transformer. Consider two adjacent blocks b and $b+1$. Then $J_b := U_b^{PS} D_{b+1}^{PS}$ is the linear transformation between these two blocks. If the compression dimension of both blocks is identical, this becomes a square matrix, which poses no problem as long as it has full rank, but can affect the residual flow through the Transformer. Thus, initializing these two matrices in a way that J_b is the identity matrix (or omitting them entirely) stands to reason.

To maximize the residual flow, we initialize as many J_b as identity matrices as possible and otherwise pair adjacent D and U matrices (see Appendix A for exact pairings). For any two paired matrices, we can then optimize the reconstruction error (see Equation 4) with X^l the input before D, X^{l+1} the output without compression, and $\widetilde{X}^{l+1}$ the output after U, e.g., if D^{PS} and $U^{\{V,K,Q\}}$ are paired, we minimize the three errors $\mathbb{E}_X \left\| X - X D^{PS} U^i \right\|_F$ with $i \in \{V, K, Q\}$.

Attention Compression Attention Compression reduces the dimensionality of the embeddings of the keys, queries and values within the multi-head attention to a dimension cmp^a. We introduce the inner attention downscaling matrices D^V, D^K, D^Q (which are of block diagonal structure, processing the concatenated outputs of the linear layers simultaneously, i.e., each head has its own block in the inner downscaling, unlike the outer upscaling, which is shared), as well as the inner upscaling U^O.

Consider the compressed Multi-Head Attention $\check{MHA}(X)$ where $\check{SA}_i(X)$ is the compressed at-

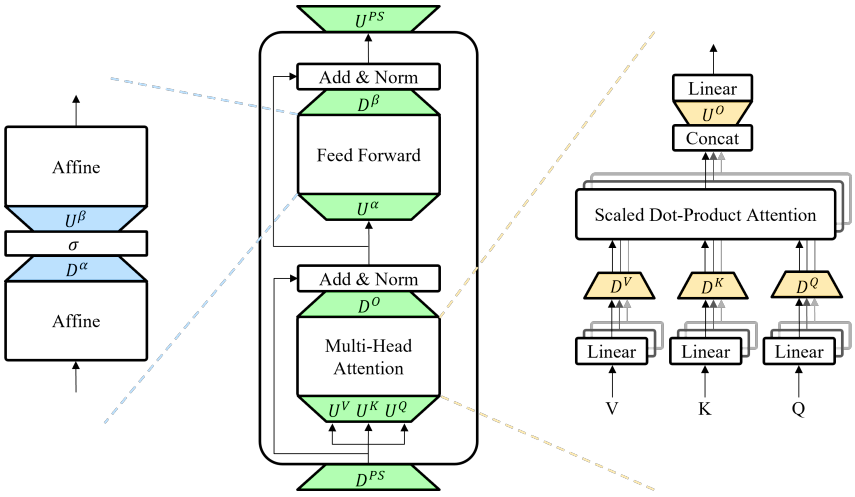


Figure 2: Left: Feed Forward Compression (blue). Center: Embedding Compression (green). Right: Attention Compression (orange).

tention head i:

$$\check{MHA}(X) \coloneqq \check{SA}(X)U^O W^O \quad (5)$$

$$\check{SA}(X) \coloneqq \left(\check{SA}_1(X) \mid \ldots \mid \check{SA}_H(X)\right) \quad (6)$$

$$\check{SA}_i(X) \coloneqq \mathrm{softmax}(\check{Z}_i(X))XW_i^V D_i^V \quad (7)$$

$$\check{Z}_i(X) \coloneqq \frac{XW_i^Q D_i^Q \left(XW_i^K D_i^K\right)^\top}{\sqrt{d_K}} \quad (8)$$

Here, $D_i^{\{V,K,Q\}}$ are the blocks i of the corresponding $D^{\{V,K,Q\}}$ matrix. By rearranging we have

$$\check{MHA}(X) = (Y_1 \mid \ldots \mid Y_H)\, D^V U^O W^O \quad (9)$$

with $Y_i \coloneqq \mathrm{softmax}(\check{Z}_i(X))XW_i^V$. Then in particular we have that D^V and U^O are adjacent in our computation, and we can jointly optimize them by minimizing the error $\mathbb{E}_X \left\|SA(X) - SA(X)D^V U^O\right\|_F$ where $SA(X)$ denotes the uncompressed self-attention.

This leaves us with the matrices D^K, D^Q which we pair, thus aiming to minimize the error $\mathbb{E}_X \left\|\mathrm{softmax}(Z_i(X)) - \mathrm{softmax}(\check{Z}_i(X))\right\|_F$ with $Z_i(X) \coloneqq \frac{XW_i^Q(W_i^K)^\top X^\top}{\sqrt{d_K}}$, which is non-trivial due to the $\mathrm{softmax}$ function. Instead, we replace this objective by

$$\mathbb{E}_X \left\|(Z_i(X) - \check{Z}_i(X)) \cdot \sqrt{d_K}\right\|_F = \quad (10)$$

$$\mathbb{E}_X \left\|XW_i^Q(I - D_i^Q(D_i^K)^\top)(XW_i^K)^\top\right\|_F \quad (11)$$

which is an upper bound for the original error since $\mathrm{softmax}$ is Lipschitz-continuous w.r.t. the Frobenius norm with Lipschitz constant 1 (Gao and Pavel, 2017). With $L\Sigma R^\top$ the SVD of $XW_i^Q(XW_i^K)^\top$ and $\check{L}, \check{R}$ the cmp^a first columns of $L\Sigma^{\frac{1}{2}}$ resp. $R\Sigma^{\frac{1}{2}}$, $D_i^Q = (XW_i^Q)^\dagger \check{L}$ and $D_i^K = (XW_i^K)^\dagger \check{R}$ then minimize the objective in Equation 11. The proof for this can be found in Appendix B.

Feed Forward Compression Feed Forward Compression is straightforward and follows the same optimization approach as described in Subsection 3.1. We introduce the pre- and postscaling matrices D^α, U^β which create a bottleneck with dimension cmp^f and get the compressed network

$$\check{FF}(X) \coloneqq \sigma((XW^\alpha + b^\alpha)D^\alpha)U^\beta W^\beta + b^\beta \quad (12)$$

in which we pair the newly introduced matrices.

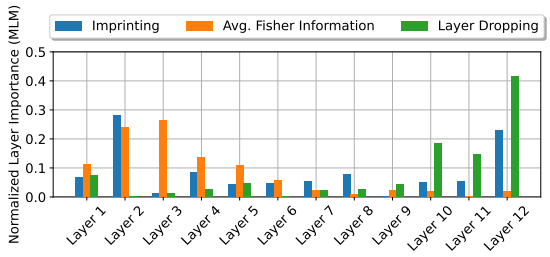
4 Experiments

We now investigate how to apply APC in practice. To this end, we compress the well-studied $\mathrm{BERT}_{\mathrm{base}}$ model (Devlin et al., 2019). All experiments were run on one Nvidia A40 paired with an AMD Epyc 7413 and 512 GB RAM. Throughout our experiments, we use Python 3.10 with the libraries transformers (4.36.2), torch (2.1.2), and optuna (3.5.0).

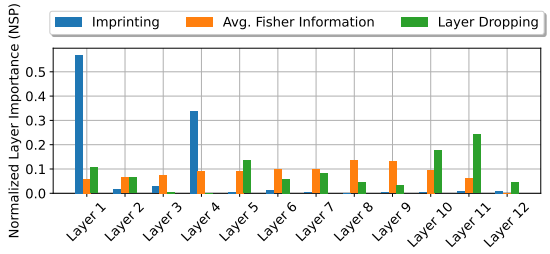
4.1 Compression Sizes

As laid out in the previous section, APC allows us to choose different compression dimensions at each bottleneck. To reduce the combinatorial explosion resulting from 3 different compressions per layer and 12 layers, we first seek to group layers which will then share compression dimensions. To this end, we measure the importance of each layer using the methods described in Section 2.2 both

for the MLM and NSP tasks, which can be seen in Figure 3. While Fisher information is almost



(a) Layer Importance w.r.t. MLM accuracy



(b) Layer Importance w.r.t. NSP accuracy

Figure 3: Layer Importances in $\text{BERT}_{\text{base}}$. Layer Importances are normalized to sum to 1.

uniformly distributed for NSP, weight imprinting suggests a strong importance of layers 1 and 4, and layer dropping for layers 10 and 11. For the MLM task, importance distributions are significantly different: While Fisher information suggests high relevance of layer 2 and 3, layer dropping indicates a strong influence of the last three layers, and weight imprinting sees layer 2 and 11 as the most important. For both tasks, layers 5 to 9 have only slight influence on the accuracy of the network, motivating us to compress them with identical ratios. Further, we group layers 10 to 12, given their strong values for layer dropping for MLM, layer 1 and 2, as well as layer 3 and 4.

Given these groups, we continue with a hyperparameter search to find optimal dimensions for the embedding, attention, and feed forward compressions, using the Optuna optimization engine with a tree-structured Parzen estimator sampler (Akiba et al., 2019) with 100 trials per search.

We search compression dimensions for a total compression rate of 90% (APC-90-opt), 50% (APC-50-opt), and 15% (APC-15-opt). Since the found architecture for a compression of 50% barely compressed layers 1 and 2, we ran an additional search, restricting the compression only to the remaining layers, which yielded a better performance on the pre-training tasks during the parameter search, hence we will keep this architecture. Further, we include a manually chosen architecture with a 50% compression rate which only uses feed-forward compression on layers 3 to 12 (APC-50-ff), limiting them to a dimension of 256.

The found architectures mostly follow the relevance assigned to layers by weight imprinting and average fisher information, indicated by a low compression ratio on lower layers and stronger compression in higher layers. The exact compression dimensions can be found in Appendix F in Table 8.

4.2 Initialization Strategies

We look into different initialization strategies to reduce the reconstruction error and maximize accuracy on our pre-training dataset. For all approaches, we set b_U, b_D, and B to 0.

Random Initialization For random initialization, we sample all entries for U and D independently from a zero-centered Gaussian distribution with a tiny, fixed variance of 10^{-6}. In our experiments, this variance has shown to be most effective compared with input-output variance-preserving and gradient variance-preserving initializations (Glorot and Bengio, 2010), which showed slow convergence behavior. Thus, we omit the derivations and definitions of the necessary variance values for the latter two cases for the sake of brevity.

Reconstructive Random Initialization As for the random initialization, we sample all entries of D from the distribution as described above, but calculate the entries of U by means of a multivariate linear regression, minimizing the reconstruction errors described in Subsection 3.2, using a randomly sampled batch of training data of size 5000.

Reconstructive SVD Initialization The reconstructive SVD initialization is inspired by Weight Factorization as done by Chen et al. (2021), essentially pruning in an orthogonalized column space of the original matrix W. To this end, we initialize D with the right singular vectors of W corresponding to the cmp-highest singular values. U is then calculated as in the reconstructive random initialization to minimize the reconstructive error.

Neuron Sensitivity Pruning Initialization We use Molchanov et al.'s (2016) approach (see Subsection 2.1) to determine the cmp most important columns in our weight matrix W and initialize D s.t. WD corresponds to exactly these cmp columns.

Initialization strategy	CE	KD	actKD
Random	0.308 / 0.924	0.295 / 0.916	**0.317 / 0.929**
Reconstr. Random	0.044 / **0.484**	0.044 / **0.484**	**0.045 / 0.484**
Sens. Neuron Pruning	0.434 / **0.972**	0.425 / **0.972**	**0.439 / 0.972**
Reconstr. SVD	0.047 / 0.516	0.044 / 0.516	**0.120 / 0.726**
k-Means	**0.397 / 0.963**	0.388 / 0.958	0.383 / 0.948

Table 1: MLM-/NSP-accuracies after 100 steps of APC-50-ff training. Blue values indicate best scores per loss, **bold** values best scores per initialization strategy.

	# Parameters	GFLOPs per inf.	Inference time
BERT	110M	3032	744ms
Zafrir et al.	33M	3032	734ms
DistilBERT	67M	1823	441ms
APC-50-ff	67M	1926	572ms

Table 2: Comparing model size and cost, averaged over 50 runs.

U is then set to $D^\top$. The formal algorithm can be found in Appendix C. This initialization directly corresponds to neuron pruning, thus APC generalizes this approach.

k-Means Initialization This approach is based on Zhong et al.'s (2018) neuron merging: We cluster all columns of W into cmp many sets and initialize D s.t. each column of WD is the average of the columns in the corresponding cluster. For U, we have $U_{i,j} \coloneqq 1$ if $D_{j,i} \neq 0$ and 0 otherwise. The formal algorithm can be found in Appendix C.

When comparing the reconstruction errors of these initialization techniques on our APC-50-ff architecture, both reconstructive approaches scored best since they actively minimize this error, with the neuron pruning and merging approaches following, most likely due to their preserving nature of the original weight matrix, and random scoring worst, which is to be expected. The exact error terms can be found in Appendix F, Table 9. However, one must note that while a low reconstruction error indicates a better mimicking of the original network, it does not necessarily correlate with stronger language modelling performance. Hence, we continue with training our new parameters and evaluating the resulting models on the pretraining task.

4.3 Training and Loss Functions

While we can use a standard cross entropy (CE) loss $\mathcal{L}_{\mathrm{CE}}$ (in BERT the sum of the MLM-loss $\mathcal{L}_{\mathrm{MLM}}$ and the NSP-loss $\mathcal{L}_{\mathrm{NSP}}$), knowledge distillation motivates fitting the output distribution of our compressed network to the original one's by means of a student-teacher regularization term. We define $\mathcal{L}_{\mathrm{student-teacher}}$ as $\frac{5}{10^4} \cdot \mathcal{L}_{\mathrm{MLM}}^{\mathrm{s-t}} + \frac{5}{2} \cdot \mathcal{L}_{\mathrm{NSP}}^{\mathrm{s-t}}$ with $\mathcal{L}^{\mathrm{s-t}}$ the soft target loss to balance the differences in numbers of classes between both tasks. Our knowledge distillation (KD) loss is then $\mathcal{L}_{\mathrm{total}} = 0.7\mathcal{L}_{\mathrm{CE}} + 0.3\mathcal{L}_{\mathrm{student-teacher}}$. Further, as applied by Chumachenko et al. (2020), we also test aligning the outputs of the activation functions (Zagoruyko and Komodakis, 2017) to further minimize the reconstruction error (4) by means of $\mathcal{L}_{\mathrm{total}} = 0.7\mathcal{L}_{CE} + 0.3\beta \sum_{l=1}^{L} \left\| X^{l+1} - \widetilde{X}^{l+1} \right\|_F$ with $\beta = 10^5$, the activation knowledge distillation (actKD) loss. Whenever applying knowledge distillation, we use a temperature of 1. To compress $\mathrm{BERT}_{\mathrm{base}}$ we randomly sample 1,000 Wikipedia articles and 10,000 sentences from the BooksCorpus (Zhu et al., 2015) for training with a 90-10 train-dev split to match the model's pre-training data distribution. The training hyperparameters are found in Table 5, Appendix D. We showcase the results in Table 1 for APC-50-ff, the best-performing architecture, results for the remaining compressions can be found in Appendix F.

While these results show a quick improvement of APC-50-ff with neuron pruning (and random initialization for the other architectures) when combined with the actKD loss, running the training for 5 epochs with early stopping actually indicates better performance of the standard CE loss, with a final accuracy of 0.488 resp. 0.981 versus 0.462 resp. 0.979 for actKD (with 0.468 resp. 0.981 for KD), with similar observations for the other architectures. Given this plus the additional cost of calculating the regularization terms leads us to the decision of using the CE loss instead.
We further trained the model until full convergence. The training hyperparameters are listed in Table 6, Appendix D.

4.4 Pre-Trained Model's Compute Resources

On our setup, it took APC-50-ff about 90 minutes to train to full convergence (ca. 404 PFLOPs). In comparison, performing neuron pruning to the same architecture (see Appendix E for details) took about 3 hours (ca. 808 PFLOPs), and training Distil-

	SST2	MNLI (m. / mm.)	RTE	MRPC (acc. / F1)	Avg.
$BERT_{base}$	**0.927**	**0.844 /** –	[0.664]	0.867 / –	**0.837**
Zafrir et al.	0.909	0.815 / 0.824	–	– / –	–
DistilBERT	0.913	0.822 / –	0.599	**0.875 /** –	0.802
Turc et al.	0.911	0.825 / 0.834	0.667	0.849 / 0.894	0.813
Neuron Pruning	0.905	0.788 / 0.797	0.657	0.853 / 0.896	0.801
Nova et al. (60%)	0.911	0.772 / –	–	0.842 / –	–
APC-50-ff	0.901	0.799 / 0.806	**0.682**	0.853 / 0.894	0.809

Table 3: Fine-tuning results of multiple (compressed) $BERT_{base}$-models on various GLUE tasks. Best scores per task are highlighted in bold. Scores are reported over dev sets or, when in brackets, over the closed test sets. Further, we report the average overall task scores, where we only count matched MNLI, and for MRPC, only accuracy.

BERT took about 90 hours on 8 V100 GPUs (Sanh et al., 2019) (ca. 324 EFLOPs). In Table 2, we compare the required compute resources for performing inferences with a batch size of 64.

4.5 Fine-Tuning Strategies

	Finalized	Compr. only	All Params	FT + Compr.
SST-2	**0.901**	0.899	0.900	0.898
MNLI (m. / mm. acc.)	0.792 / 0.799	0.776 / 0.789	**0.799 / 0.806**	0.765 / 0.772
RTE	0.675	0.664	**0.682**	0.646
MRPC (acc. / F1)	0.831 / 0.884	0.821 / 0.877	**0.853 / 0.894**	0.772 / 0.847

Table 4: Fine-tuning results of APC-50-ff on various GLUE tasks with different fine-tuning strategies. Scores of the best fine-tuning strategy per architecture are highlighted in **bold**. Scores are reported over dev sets.

There are multiple strategies to fine-tune pre-trained APC models: We can either finalize the model by replacing all W, b with $\check{W}, \check{b}$ as in Equations 1,2 and then train those new parameters, fine-tune only the compression parameters, fine-tune all parameters after pre-training or simultaneously fine-tune and compress the model by APC. We test our architectures on four datasets from the GLUE benchmark suite (Wang et al., 2018), namely SST-2 (Socher et al., 2013), RTE (Bentivogli et al., 2009), MRPC (Dolan and Brockett, 2005), and MNLI (Williams et al., 2018). Training hyperparameters can be found in Table 7, Appendix D. The varying amount of epochs across different tasks aim to counteract different dataset sizes. The results for APC-50-ff can be found in Table 4, the results for the other architectures in Appendix F. The results suggest a generally higher performance when training both compression and original weights, however, this comes with a significant increase of necessary compute power compared to training the finalized network (38% and 72% more time for compression weights only and for all parameters, respectively, see Table 15, Appendix H for details) for marginal performance gains, hence we recommend fine-tuning the finalized model instead.

5 Results

Eventually, we would like to relate APC to other compression approaches. To this end, we compare the best dev set scores of APC-50-ff for the given downstream tasks with the numbers officially provided for the models of Zafrir et al. (2021), Turc et al. (2019) (see Subsection 2.1), DistilBERT, $BERT_{base}$ with 50% neuron sensitivity pruning on layers 3 to 12, and $BERT_{base}$ as a baseline. The results can be found in Table 3. It is notable that our architecture retains about 97% of $BERT_{base}$'s average performance, and performs slightly better on average than DistilBERT. This is, however, only a result of the significantly better score for the RTE dataset. APC performs slightly worse than Zafrir et al.'s (2021) model, but provides gains in computational speed even on regular consumer hardware. In general, APC-50-ff shows an on-par performance with other methods of similar compression size, but is not hardware-dependent for speed gains and can be trained very fast.

6 Conclusion

We have presented Adaptive Parameter Compression, a flexible technique to compress DNNs, which learns low-dimensional approximations through affine projections in the parameter space. APC extends and modifies the Weight Squeezing approach to improve its performance, matching the ones of similarly strongly compressed approaches at a reduced training time. We provided theoretical and empirical insights into different optimization

techniques for APC and guidelines on how to apply it to a BERT model. Future work will focus on testing APC on larger LMs, improving its performance for lower compression ratios, and extending the approach to also reduce the number of layers.

7 Limitations

In our work, we present how to apply APC to $\text{BERT}_{\text{base}}$ and investigated how we can optimize it for this specific model. Due to limited computational resources, we could not evaluate our method on more recent, larger language models yet, which may show a different behavior when applying the same initialization techniques and loss functions we chose for $\text{BERT}_{\text{base}}$. While we assume that APC should work even better on larger, overparameterized models, more extensive testing with a variety of models is required to show scalability of our method.

Further, while pruning algorithms follow an iterative paradigm of removing parameters and thus slowly decrease the performance of the original model, APC starts with a worse initial performance which is increased by training, similar to distillation approaches. This makes it rather unsuitable for low compression ratios as we have to introduce more parameters, which increase training time and, thus, provide a worse tradeoff between computational cost for finding the optimal compressed model and performance than neuron pruning.

Currently, APC also requires to choose a fixed compression architecture, making it cumbersome to find the best tradeoff between the model's performance and inference cost. We have started to conceptualize methods to overcome this, however further research and testing is required, hence we did not include these in this paper.

While some of our initialization strategies aimed at minimizing the reconstructive error in each layer, which theoretically should align the original and the compressed model, the language modelling performance did not confirm this assumption. We conjecture that this is caused by the depth of the network since we only minimize the error locally, thus potentially propagating errors throughout multiple up- and downscaling pairs, which cumulates and affects performance negatively. Thus, extending the approach to reduce the number of layers and optimizing the global reconstruction error are future directions we want to look into.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. *TAC*, 7(8):1.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Sarin Eapen Chandy, Varun Prashant Gangal, Yi Yang, and Gabriel Maggiotti. 2023. DYAD: A descriptive yet abjuring density efficient approximation to linear neural network layers. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@NeurIPS 2023)*.

Niladri S Chatterji, Behnam Neyshabur, and Hanie Sedghi. 2019. The intriguing role of module criticality in the generalization of deep networks. In *International Conference on Learning Representations*. OpenReview.net.

Patrick Chen, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. 2021. Drone: Data-aware low-rank compression for large nlp models. In *Advances in Neural Information Processing Systems*, volume 34, pages 29321–29334. Curran Associates, Inc.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Artem Chumachenko, Daniil Gavrilov, Nikita Balagansky, and Pavel Kalaidin. 2020. Weight squeezing: Reparameterization for knowledge transfer and model compression. *arXiv preprint arXiv:2010.06993*.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in Neural Information Processing Systems*, 28.

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

4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Sara Elkerdawy, Mostafa Elhoushi, Abhineet Singh, Hong Zhang, and Nilanjan Ray. 2020. One-shot layer-wise accuracy approximation for layer pruning. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 2940–2944.

Sara Elkerdawy, Mostafa Elhoushi, Abhineet Singh, Hong Zhang, and Nilanjan Ray. 2021. To filter prune, or to layer prune, that is the question. In *Computer Vision – ACCV 2020*, pages 737–753, Cham. Springer International Publishing.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Bolin Gao and Lacra Pavel. 2017. On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv preprint arXiv:1704.00805*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1135–1143, Cambridge, MA, USA. MIT Press.

B. Hassibi, D.G. Stork, and G.J. Wolff. 1993. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, pages 293–299 vol.1.

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

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*.

Steven A. Janowsky. 1989. Pruning versus clipping in neural networks. *Phys. Rev. A*, 39:6600–6603.

Chunhui Jiang, Guiying Li, Chao Qian, and Ke Tang. 2018. Efficient dnn neuron pruning by minimizing layer-wise nonlinear reconstruction error. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2298–2304. International Joint Conferences on Artificial Intelligence Organization.

Amba Kak and Sarah Myers Wes. 2023. 2023 landscape. confronting tech power. https://web.archive.org/web/20240215195947/https://ainowinstitute.org/wp-content/uploads/2023/04/AI-Now-2023-Landscape-Report-FINAL.pdf. Accessed 2024-02-15.

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710.

Pengfei Li, Jianyi Yang, Mohammad A Islam, and Shaolei Ren. 2022. Making ai less "thirsty": Uncovering and addressing the secret water footprint of ai models. *Artificial intelligence (AI)*, page 4.

Tianhong Li, Jianguo Li, Zhuang Liu, and Changshui Zhang. 2020. Few sample knowledge distillation for efficient network compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14639–14647.

Hongyang Liu, Sara Elkerdawy, Nilanjan Ray, and Mostafa Elhoushi. 2021. Layer importance estimation with imprinting for neural network quantization. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2408–2417.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. 2024. The era of 1-bit llms: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*.

Yihuan Mao, Yujing Wang, Chufan Wu, Chen Zhang, Yang Wang, Quanlu Zhang, Yaming Yang, Yunhai Tong, and Jing Bai. 2020. LadaBERT: Lightweight adaptation of BERT through hybrid model compression. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3225–3234, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. 2021. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11256–11264.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *International Conference on Learning Representations*.

Vincent C. Müller. 2020. Ethics of artificial intelligence and robotics. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. The Metaphysics Research Lab Philosophy Department Stanford University. Accessed: 2024-03-08.

Azade Nova, Hanjun Dai, and Dale Schuurmans. 2023. Gradient-free structured pruning with unlabeled data. In *International Conference on Machine Learning*, pages 26326–26341. PMLR.

James O'Neill, Greg Ver Steeg, and Aram Galstyan. 2020. Compressing deep neural networks via layer fusion. *arXiv preprint arXiv:2007.14917*.

David A. Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *CoRR*, abs/2104.10350.

Sai Prasanna, Anna Rogers, and Anna Rumshisky. 2020. When BERT Plays the Lottery, All Tickets Are Winning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3208–3229, Online. Association for Computational Linguistics.

Hang Qi, Matthew Brown, and David G. Lowe. 2018. Low-shot learning with imprinted weights. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5822–5830.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Ryan Razani, Grégoire Morin, Eyyub Sari, and Vahid Partovi Nia. 2021. Adaptive binary-ternary quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4613–4618.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2015. Fitnets: Hints for thin deep nets. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2023. On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language*, 77:101429.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Maximilian Schreiner. 2023. Gpt-4 architecture, datasets, costs and more leaked. https://web.archive.org/web/20240307142710/https://the-decoder.com/gpt-4-architecture-datasets-costs-and-more-leaked/. Accessed: 2024-03-08.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. 2022. Using deepspeed and megatron to train megatronturing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Suraj Srinivas and R Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. In *Procedings of the British Machine Vision Conference 2015*. British Machine Vision Association.

Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. 2018. Faster gaze prediction with dense networks and fisher pruning. *CoRR*, abs/1801.05787.

Ming Tu, Visar Berisha, Yu Cao, and Jae-Sun Seo. 2016. Reducing the model order of deep neural networks using information theory. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 93–98.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Ziheng Wang. 2021. Sparsednn: Fast sparse deep learning inference on cpus. *arXiv preprint arXiv:2101.07948*.

Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. 2023. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Edouard Yvinec, Arnaud Dapogny, Matthieu Cord, and Kevin Bailly. 2023. Red++ : Data-free pruning of deep neural networks via input splitting and output merging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3664–3676.

Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. 2020. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 811–824. IEEE.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pages 36–39.

Ofir Zafrir, Ariel Larey, Guy Boudoukh, Haihao Shen, and Moshe Wasserblat. 2021. Prune once for all: Sparse pre-trained language models. *arXiv preprint arXiv:2111.05754*.

Sergey Zagoruyko and Nikos Komodakis. 2017. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *International Conference on Learning Representations*.

Chiyuan Zhang, Samy Bengio, and Yoram Singer. 2022. Are all layers created equal? *J. Mach. Learn. Res.*, 23(1).

Guoqiang Zhong, Hui Yao, and Huiyu Zhou. 2018. Merging neurons for structure compression of deep networks. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 1462–1467.

Denny Zhou, Mao Ye, Chen Chen, Tianjian Meng, Mingxing Tan, Xiaodan Song, Quoc Le, Qiang Liu, and Dale Schuurmans. 2020. Go wide, then narrow: Efficient training of deep thin networks. In *International Conference on Machine Learning*, pages 11546–11555. PMLR.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19–27.

A Pairings for Embedding Compression

Consider three Transformer blocks $b-1, b, b+1$, then we have the following cases in which we pair the matrices in embedding compression as follows:

- $\text{cmp}^e_{b-1} \neq \text{cmp}^e_b$ (or b is the first block) and $\text{cmp}^e_b \neq \text{cmp}^e_{b+1}$ (or b is the last block): We pair D^{PS}_b with $U^{\{V,K,Q\}}_b$, D^O_b with U^α_b, and D^β_b with U^{PS}_b.
- $\text{cmp}^e_{b-1} \neq \text{cmp}^e_b$ (or b is the first block) and $\text{cmp}^e_b = \text{cmp}^e_{b+1}$: We pair D^{PS}_b with $U^{\{V,K,Q\}}_b$, D^O_b with U^α_b, D^β_b with $U^{\{V,K,Q\}}_{b+1}$, and U^{PS}_b with D^{PS}_{b+1}.
- $\text{cmp}^e_{b-1} = \text{cmp}^e_b$ and $\text{cmp}^e_b \neq \text{cmp}^e_{b+1}$ (or b is the last block): We pair D^{PS}_b with U^{PS}_{b-1}, D^β_{b-1} with $U^{\{V,K,Q\}}_b$, D^O_b with U^α_b, and D^β_b with U^{PS}_b.
- $\text{cmp}^e_{b-1} = \text{cmp}^e_b$ and $\text{cmp}^e_b = \text{cmp}^e_{b+1}$: We pair D^{PS}_b with U^{PS}_{b-1}, D^β_{b-1} with $U^{\{V,K,Q\}}_b$, D^O_b with U^α_b, D^β_b with $U^{\{V,K,Q\}}_{b+1}$, and U^{PS}_b with D^{PS}_{b+1}.

B Proof for Optimality

Claim: Let $A_1, A_2 \in \mathbb{R}^{n\times a}$ be of maximal rank. Define the optimization problem

$$\min_{B_1,B_2\in\mathbb{R}^{a\times b}} \left\|A_1(I_a - B_1B_2^T)A_2^T\right\|_F, \qquad (13)$$

for some $n \geq a > b$. An optimal solution to (13) is then given by

$$B_1 \coloneqq A_1^\dagger \hat{L}, \qquad B_2 \coloneqq A_2^\dagger \hat{R}, \qquad (14)$$

for $\hat{L}$ and $\hat{R}$ being the submatrices of the b first columns of $\tilde{L}$ and $\tilde{R}$ respectively, where $\tilde{L} \coloneqq L\Sigma^{1/2}$ and $\tilde{R} \coloneqq R\Sigma^{1/2}$ for L, Σ, R denoting the singular value decomposition of $A_1A_2^T$, i.e., $A_1A_2^T = L\Sigma R^T$.

Proof:

Firstly, we rewrite (13) as

$$\min_{B_1,B_2\in\mathbb{R}^{a\times b}} \left\|A_1A_2^T - A_1B_1B_2^TA_2^T\right\|_F, \qquad (15)$$

and note that $A_1 B_1 B_2^T A_2^T$ is of at most rank b because B_1 (and B_2) are of at most rank b. Therefore, because $\hat{L}\hat{R}^T$ is by definition the best rank-b approximation of $A_1 A_2^T$, we get

$$\min_{B_1, B_2 \in \mathbb{R}^{a \times b}} \left\| A_1 A_2^T - A_1 B_1 B_2^T A_2^T \right\|_F \geq \left\| A_1 A_2^T - \hat{L}\hat{R}^T \right\|_F . \tag{16}$$

On the other hand, let B_1 and B_2 be defined as in (14). Then

$$\begin{aligned}
&\left\| A_1 (I_a - B_1 B_2^T) A_2^T \right\|_F && (17)\\
=&\left\| A_1 (I_a - A_1^\dagger \hat{L}\hat{R}^T (A_2^T)^\dagger) A_2^T \right\|_F && (18)\\
=&\left\| A_1 A_1^\dagger (A_1 - \hat{L}\hat{R}^T (A_2^T)^\dagger) A_2^T \right\|_F && (19)\\
=&\left\| A_1 A_1^\dagger (A_1 A_2^T - \hat{L}\hat{R}^T)(A_2^T)^\dagger A_2^T \right\|_F && (20)\\
=&\left\| P_{A_1} (A_1 A_2^T - \hat{L}\hat{R}^T) P_{A_2}^T \right\|_F , && (21)
\end{aligned}$$

where P_* denotes the orthogonal projection onto the range of $*$. Because the range of $A_1 A_2^T$ and of $\hat{L}\hat{R}^T$ are a subspace of the range of A_1, we further get

$$\ldots = \left\| (A_1 A_2^T - \hat{L}\hat{R}^T) P_{A_2}^T \right\|_F , \tag{22}$$

and with analogous reasoning for A_2 by transposition,

$$\ldots = \left\| A_1 A_2^T - \hat{L}\hat{R}^T \right\|_F . \tag{23}$$

Thus, the bound from (16) holds with equality for the choice of B_1 and B_2 as defined in (14), which concludes the proof.

C Algorithms

Algorithm 1 Neuron Pruning Initialization

Require: $\mathrm{in}, \mathrm{out}, \mathrm{cmp}, W \in \mathbb{R}^{\mathrm{in} \times \mathrm{out}}$

$D \leftarrow$ zero-matrix of size $\mathrm{out} \times \mathrm{cmp}$
$U \leftarrow$ zero-matrix of size $\mathrm{cmp} \times \mathrm{out}$
$S \leftarrow$ compute index set of cmp-many columns of W corresponding to most "important" neurons, according to neuron sensitivity pruning mask M.
$j \leftarrow 0$
for $i \in \{1, \ldots, \mathrm{out}\}$ **do**
  if $i \in S$ **then**
    $D_{i,j} \leftarrow 1$
    $j \leftarrow j + 1$
  end if
end for
$U \leftarrow D^\top$
$b_D, b_U, B \leftarrow$ initialize with zero entries
return U, b_U, D, b_D, B

Algorithm 2 k-Means Initialization

Require: $\mathrm{in}, \mathrm{out}, \mathrm{cmp}, W \in \mathbb{R}^{\mathrm{in} \times \mathrm{out}}$

$D \leftarrow$ zero-matrix of size $\mathrm{out} \times \mathrm{cmp}$
$U \leftarrow$ zero-matrix of size $\mathrm{cmp} \times \mathrm{out}$
$(S_i)_i \leftarrow \mathrm{cmp}$-many sets of neuron indices belonging to cluster i, retrieved by k-means
for $i \in \{1, \ldots, \mathrm{cmp}\}$ **do**
  $\mathrm{cluster\_size} \leftarrow |S_i|$
  for $j \in S_i$ **do**
    $D_{j,i} \leftarrow 1/\mathrm{cluster\_size}$
    $U_{i,j} \leftarrow 1$
  end for
end for
$b_D, b_U, B \leftarrow$ initialize with zero entries
return U, b_U, D, b_D, B

D Hyperparameters

Parameters	Value
Batch size	24
Learning rate	10^{-4} to 0
Learning rate scheduler	Linear learning rate decay
Training steps	100
Optimizer	Adam $\beta_1 = 0.9, \beta_2 = 0.999$
Trainable weights	U, D, b_U, b_D

Table 5: Hyperparameters used for comparing the different initialization techniques.

Parameters	Value
Batch size	24
Learning rate	$5 \cdot 10^{-5}$ to 0
Warmup-steps	100
Learning rate scheduler	Linear learning rate decay
Epochs	5
Optimizer	Adam $\beta_1 = 0.9, \beta_2 = 0.999$
APC Initialization	Neuron sensitivity pruning
Trainable weights	U, W, D, b_U, b, b_D, B

Table 6: Hyperparameters used for the pre-training of all our APC architectures.

Parameters	Value
Batch size	32
Learning rate	$2 \cdot 10^{-5}$ to 0
Warmup-steps	100
Learning rate scheduler	Linear learning rate decay
Epochs	MNLI: 1 MRPC: 163 RTE: 240 SST-2: 8
Optimizer	Adam $\beta_1 = 0.9, \beta_2 = 0.999$

Table 7: Hyperparameters used for fine-tuning our APC architectures on downstream tasks.

E Neuron Pruning

In our work, we compare APC to neuron pruning. We prune BERT using as pruning criterion *neuron sensitivity pruning*, as introduced in 2.1. In contrast to magnitude based pruning, the former also considers the derivative of an entire neuron and can thus arguably be seen as of one error convergence order higher than magnitude based pruning (Molchanov et al., 2016). We iteratively prune BERT to the same size as APC-50-ff in 10 pruning steps. Each steps starts by pruning a fixed ratio of weights, followed by fine-tuning the model until convergence. Convergence is determined by the sum of MLM- and NSP-accuracy not increasing for three evaluation steps, where an evaluation step is performed on the validation set after every 60 fine-tuning steps. All other hyperparameters equal those used to compress via APC.

F Compression Architectures & Performance

	Layer group (1, 2)	Layer group (3, 4)	Layer group (5, 6, 7, 8, 9)	Layer group (10, 11, 12)
APC-15-opt (16.9%) compression				MLM / NSP Accuracy: 0.124 / 0.729
cmp^e	654	753	768	720
cmp^a	588	756	708	768
cmp^f	2251	1399	2527	3057
Cmp. ratio	36.6%	38.0%	14.4%	6.6%
APC-50-opt (51.7%) compression				MLM / NSP Accuracy: 0.257 / 0.911
cmp^e	-	767	654	669
cmp^a	-	768	252	732
cmp^f	-	3072	171	421
Cmp. ratio	0%	4.9%	78.1%	61.1%
APC-90-opt (88.8%) compression				MLM / NSP Accuracy: 0.121 / 0.541
cmp^e	631	159	113	102
cmp^a	720	72	252	46
cmp^f	200	2418	1636	468
Cmp. ratio	70.7%	88.4%	93.1%	97.1%
APC-50-ff (50.9%) compression				MLM / NSP Accuracy: 0.308 / 0.924
cmp^e	-	-	-	-
cmp^a	-	-	-	-
cmp^f	-	256	256	256
Cmp. ratio	0%	61.1%	61.1%	61.1%

Table 8: APC architectures returned by a hyperparameter search. The original values for embedding, attention and feed-forward dimensions were 768, 768, and 3072, respectively. Further, we include the compression ratio of each layer group, excluding pre- and post-scalings. Lastly, we report the MLM- and NSP accuracy after 100 steps of training as performed by the hyperparameter search.

Initialization Strategy	Average Reconstruction Error
Random	$1.067 \cdot 10^3$
Reconstructive Random	$0.627 \cdot 10^3$
Reconstructive SVD	$0.637 \cdot 10^3$
Neuron Sensitivity Pruning	$0.900 \cdot 10^3$
k-Means	$0.856 \cdot 10^3$

Table 9: Comparing reconstruction errors of all compressed layers of APC-50-ff. Reconstruction errors have been averaged over all compressed layers.

Initialization Strategy	CE	KD	actKD
Random	0.045 / 0.479	0.044 / 0.484	**0.122 / 0.567**
Reconstr. Random	0.044 / **0.484**	0.044 / **0.484**	**0.046 / 0.484**
Sens. Neuron Pruning	**0.044 / 0.516**	**0.044** / 0.484	0.000 / **0.516**
Reconstr. SVD	0.044 / **0.516**	0.044 / 0.484	**0.045** / 0.484
k-Means	**0.044** / 0.484	**0.044** / 0.484	0.000 / **0.516**

Table 10: With the same format as in Table 1, we report the MLM- / NSP-accuracies after 100 steps of APC-15-opt training.

Initialization Strategy	CE	KD	actKD
Random	0.257 / 0.847	0.254 / **0.875**	**0.259** / 0.737
Reconstr. Random	0.044 / 0.484	0.044 / 0.484	**0.045 / 0.490**
Sens. Neuron Pruning	**0.044 / 0.516**	**0.044 / 0.516**	0.000 / **0.516**
Reconstr. SVD	0.136 / 0.906	0.043 / 0.516	**0.220 / 0.947**
k-Means	**0.044** / 0.484	**0.044 / 0.516**	0.000 / **0.516**

Table 11: With the same format as in Table 1, we report the MLM- / NSP-accuracies after 100 steps of APC-50-opt training.

Initialization Strategy	CE	KD	actKD
Random	0.119 / **0.484**	**0.125 / 0.484**	0.116 / **0.484**
Reconstr. Random	0.044 / 0.484	0.044 / 0.484	**0.045 / 0.516**
Sens. Neuron Pruning	**0.044 / 0.516**	**0.044** / 0.484	0.000 / **0.516**
Reconstr. SVD	0.044 / 0.484	0.044 / 0.484	**0.045 / 0.516**
k-Means	**0.047** / 0.484	0.044 / 0.484	0.000 / **0.516**

Table 12: With the same format as in Table 1, we report the MLM- / NSP-accuracies after 100 steps of APC-90-opt training.

	Finalized	Compr. only	All Params	FT + Compr.
		SST2		
APC-90-opt	0.837	**0.839**	0.838	0.825
APC-50-opt	0.877	0.876	**0.882**	0.825
		MNLI (m. / mm. acc.)		
APC-90-opt	0.616 / 0.612	0.652 / 0.658	**0.658 / 0.665**	0.443 / 0.468
APC-50-opt	0.727 / 0.733	0.721 / 0.732	**0.737 / 0.744**	0.354 / 0.352
		RTE		
APC-90-opt	0.552	0.578	**0.592**	0.563
APC-50-opt	**0.621**	**0.621**	0.610	0.542
		MRPC (acc. / F1)		
APC-90-opt	0.686 / 0.790	**0.689** / 0.797	0.684 / 0.781	0.684 / **0.812**
APC-50-opt	0.760 / 0.843	**0.767 / 0.850**	0.757 / 0.841	0.723 / 0.825

Table 13: Fine-tuning results of multiple APC architectures on various GLUE tasks with different fine-tuning strategies. The scores of the best fine-tuning strategy per architecture are highlighted in **bold**. The scores are reported over dev sets.

G Effects of Using Biases

To quantify the effect of compression biases on overall compressed model performance, we trained two different APC-architectures once without biases, once with b_D and b_U, and lastly with all biases, including B. Each setting is trained over max. five epochs of our training data with a learning rate of 10^{-4} and linear learning rate decay over those five epochs. We use early stopping with the patience of three model evaluations on the validation set and convergence criterion being the sum of the model's MLM and NSP accuracy. The model is evaluated every 100 steps. As an initialization strategy, we use random initialization with a tiny fixed variance. No knowledge distillation and activation knowledge distillation is used. The results are presented in Table 14. We found that using b_U and b_D performs significantly better than when not using biases. Introducing B significantly improves the performance of APC-90-opt while it slightly decreases the performance in APC-50-ff.

	APC-90-opt	APC-50-ff
b_U, b_D, and B	**0.142 / 0.896**	0.478 / **0.981**
b_U and b_D	0.138 / 0.857	**0.488 / 0.981**
No biases	0.139 / 0.828	0.439 / 0.976

Table 14: Comparing MLM-/NSP-accuracies for different bias approaches in the APC architecture after full training.

H Fine-tuning Training Times

Strategy	Training Time
Finalized	149 min
Compression only	205 min
All Parameters	256 min
FT + Compression	254 min

Table 15: Comparing fine-tuning training times for different APC fine-tuning strategies. The reported times are accumulated over fine-tuning on MNLI, MRPC, RTE, and SST-2.