LEAP: Learnable Pruning for Transformer-based Models

Anonymous ACL submission

Abstract

Pruning is an effective method to reduce the memory footprint and computational cost associated with large natural language processing models. However, current pruning algorithms either only focus on one pruning category, e.g., structured pruning and unstructured, or need extensive hyperparameter tuning in order to get reasonable accuracy performance. To address these challenges, we propose LEArnable Pruning (LEAP), an effective method to gradually prune the model based on thresholds learned by gradient descent. Different than previous learnable pruning methods, which utilize L_0 or L_1 penalty to indirectly affect the final pruning ratio, LEAP introduces a novel regularization function, that directly interacts with the preset target pruning ratio. Moreover, in order to reduce hyperparameter tuning, a novel adaptive regularization coefficient is deployed to control the regularization penalty adaptively. With the new regularization term and its associated adaptive regularization coefficient, LEAP is able to be applied for different pruning granularity, including unstructured pruning, structured pruning, and hybrid pruning, with minimal hyperparameter tuning.

1 Introduction

Since the development of transformer models (Vaswani et al., 2017), the number of parameters for natural language processing (NLP) models has become much larger, e.g., BERT$_{large}$ (330M) (Devlin et al., 2019), Megatron-LM (8.3B) (Shoeybi et al., 2019), T5 (11B) (Raffel et al., 2019), GPT3 (170B) (Brown et al., 2020), and MT-NLG (530B) (Microsoft and Nvidia, 2021). Although larger models tend to exhibit better generalization ability for downstream tasks, the inference time and associated power consumption become critical bottlenecks for deploying those models on both cloud and edge devices.

One promising approach to address the inference time and power consumption issues of these large models is pruning (Sanh et al., 2020; Michel et al., 2019; Wang et al., 2020a). As the nature of neural networks (NNs), different pruning granularity exists, e.g., structured pruning (head pruning for transformers and block-wise pruning for weight matrices) and unstructured pruning (purely sparse-based pruning). Different pruning methods are proposed, but they generally only target one set of pruning granularity. As such, when a new scenario comes, e.g., hybrid pruning, a combination of structured pruning and unstructured pruning, it is unclear how to choose the proper method.

Meanwhile, existing work sometimes sets the same pruning ratio for all layers. However, it is challenging to prune the same amount of parameters of all weights of a general NNs to ultra-low density without significant accuracy loss. This is because not all the layers of an NN allow the same pruning level. A possible approach to address this is to use different pruning ratios. A higher density ratio is needed for certain “sensitive” layers of the network, and a lower density ratio for “non-sensitive” layers. However, manually setting such multi-level pruning ratios is infeasible. Regularization method, e.g., (Sanh et al., 2020), is proposed to address multi-level pruning ratio issue. However, it introduces two drawbacks: (i) a careful hand-tuned threshold schedule is needed to improves the performance; and (ii) the regularization needs heavy tuning to get the desired density ratio in that the regularization term is not directly applied to the pruning ratio,

Motivated by these issues, we propose an effective LEArnable Pruning (LEAP) method to gradually prune the weight matrices based on corresponding thresholds that are learned by gradient descent. We summarize our contributions below,

- LEAP sets a group of learnable pruning ratio parameters, which can be learned by the stochastic gradient descent, for the weight matrices, with a purpose to set a high pruning ratio for insensitive layers and vice versa. As the NN prefers a high-density ratio for higher accuracy and low loss, we introduce a novel regularization function that can directly control the preset target pruning ratio. As such, LEAP can easily achieve the desired compression ratio unlike those L_0 or L_1 penalty-based regularization methods, whose target pruning ratio needs careful tuning.

- To ease hyperparameter search, we design an adaptive regularization magnitude λ_{reg} to adaptively control the contribution to the final loss from the regularization penalty. The coefficient λ_{reg} is automatically adjusted to be large (small) when the current pruning ratio is far away (close to) the target ratio.
- We apply LEAP for BERT$_{\text{base}}$ on three datasets, i.e., QQP/MNLI/SQuAD, under different pruning granularity, including structured, hybrid, and unstructured pruning, with various pruning ratios. Our results demonstrate that LEAP can consistently achieve on-par or better performance as compared to previous heavily tuned methods, with minimal hyperparameter tuning. Moreover, by analyzing the pruned models, two observations are made: (1) early layers are more sensitive to pruning, which results in a higher density ratio at the end; and (2) fully connected layers are less sensitive to pruning, which results in higher pruning ratios than multi-head attention layers.

2 Methodology

2.1 Background and Problems

Regardless of pruning granularity, in order to prune a neural network (NN) there are two approaches: (i) one-time pruning (Yu et al., 2021; Michel et al., 2019) and (ii) multi-stage pruning (Han et al., 2016; Lagunas et al., 2021). The main difference between the two is that one-time pruning directly prunes the NN to a target ratio within one pruning cycle. However, one-time pruning oftentimes requires a pre-trained model on downstream tasks and leads to worse performance as compared to multi-stage pruning. For multi-stage pruning, two main categories are used: (i) one needs multiple rounds for pruning and finetuing (Han et al., 2016); and (ii) another gradually increases pruning ratio within one run (Sanh et al., 2020; Lagunas et al., 2021). Here, we focus on the latter case, where the pruning ratio gradually increases until it reaches the target.

Assume the NN consists of n weight matrices, $\mathcal{W} = \{W_1, \ldots, W_n\}$. To compress $\mathcal{W}$, gradual pruning consists of the following two stages:

- (S1) For each W_i, we initialize a corresponding all-one mask M_i and denote $\mathcal{M} = \{M_1, \ldots, M_n\}$ as the whole set of masks.
- (S2) We train the network with the objective: $\min_{\mathcal{W}} \mathcal{L}_{\text{pure}}(\mathcal{M} \odot \mathcal{W})$, where $\mathcal{M} \odot \mathcal{W}$ means $W_i \odot M_i$ for all $i = 1, \ldots n$, and $\mathcal{L}_{\text{pure}}$ is the standard training objective function of the associated task, e.g., the finite sum problem with cross-entropy loss. As the training proceeds, M_i is gradually updated with more zero, i.e., the cardinality $|M_i| = s_t^i$ becomes smaller.

Here s_t^i in (S2) could be a simple linear decaying function or more generally a polynomial function based on the user's requirement. Such method is called *hard/soft-threshold pruning*. For both threshold pruning, users have to design sparsity scheduling which raises hyperparameters search issues. Hard-threshold pruning can hardly extend to different pruning granularity, which likely leads to sub-optimal solutions by setting the same pruning ratio for all layers. While soft threshold methods could be a possible solution to resolve part of the problems, it introduces another extra hyperparameter, λ_{reg}, and there are critical concerns on how to obtain the target sparse ratio. See Appendix B for detailed description.

We address the above challenges in the coming section by designing learnable thresholds with (i) a simple yet effective regularization function that can help the users to achieve their target sparse ratio, and (ii) an adaptive regularization magnitude, λ_{reg} to alleviate the hyperparameter tuning.

2.2 LEAP with A New Regularization

We denote the learnable threshold vector $\boldsymbol{\sigma} = [\sigma_i, \ldots, \sigma_n]$ and each σ_i associates with the tuple (W_i, M_i, S_i). With the score $\mathcal{S}$ and learnable threshold vector $\boldsymbol{\sigma}$, LEAP can be smoothly incorporated to Top-k pruning method (Zhu and Gupta, 2017; Sanh et al., 2020).[1]

Recall the Top-K pruning uses the score matrix set $\mathcal{S}$ to compute $\mathcal{M}$, i.e., $M_i = \text{Top-}K(S_i)$ with $K \in [0, 100]$ in a unit of percentage. By sorting the elements of the matrix S_i, Top-K set the mask M_i for the top $K\%$ to be 1, and the bottom $(100-K)\%$ to 0. Mathematically, it expresses as

$$\text{Top-K}(x) = \mathbb{1}_{\{x \in \text{sort}(S_i, K\%)\}} \qquad (1)$$

[1]Our methods can thus be easily applied to magnitude-based pruning methods by setting $\mathcal{S}$ to be identical to $\mathcal{W}$ (Han et al., 2015).

where sort$(S_i, K\%)$ contains the Top $K\%$ of the sorted matrix S_i. Here K is determined by the users, and thus follows various kinds of schedules such as the cubic sparsity scheduling, Eq. 6. As described in Section 2.1, such a schedule usually requires extensive engineering tuning in order to achieve state-of-the-art performance. Moreover, in (Zhu and Gupta, 2017), the Top-K$(\cdot)$ threshold is fixed for all weight matrices. However, different weight matrices have different tolerances/sensitivities to pruning, meaning that a low pruning ratio needs to be applied for sensitive layers, and vice versa. In order to resolve those issues, we propose an algorithm to automatically adjust their thresholds for all weight matrices. More specifically, we define $K(\sigma_i) := 100 \cdot k(\sigma_i)$ for $i = 1, \ldots, n$ with

$$k(\sigma_i) = \text{Sigmoid}(\sigma_i/T) \tag{2}$$

where the Sigmoid function is used to map σ to be in the range of $(0, 1)$. T is a temperature value which critically controls the speed of k transitioning from 1 to 0 as σ decreases. We remark that Sigmoid could be replaced with any continuous function that maps any positive or negative values to $[0, 1]$. Investigating for various such functions could be an interesting future direction.

For a mask $M_i \in \mathbb{R}^{d^i_{\text{in}} \times d^i_{\text{out}}}$, its density ratio $|M_i|/(d^i_{\text{in}} \times d^i_{\text{out}}) = k(\sigma_i)$ is uniquely determined by σ_i. However, directly applying this for our objective function will tend to make $k(\sigma_i)$ always close to 1, since the model prefers no pruning to achieve lower training loss. Therefore, we introduce a novel regularization term to compensate for this. Denote $R(\boldsymbol{\sigma})$ the remaining ratio of weight parameter, which is a function of $\boldsymbol{\sigma}$ (more details of how to calculate $R(\boldsymbol{\sigma})$ are given later). Suppose that our target pruning ratio is R_{target}. We propose the following regularization loss,

$$\mathcal{L}_{reg}(\boldsymbol{\sigma}) = \begin{cases} (R(\boldsymbol{\sigma}) - R_{target})^2 & R(\boldsymbol{\sigma}) \geq R_{target}, \\ 0 & \text{else.} \end{cases} \tag{3}$$

Equipped with Eq. 1, 2, and 3, we then rewrite the training objective as

$$\mathcal{L}_{\text{obj}} = \mathcal{L}_{\text{pure}}(\mathcal{M}_{\boldsymbol{\sigma}} \odot \mathcal{W}) + \lambda_{reg}\mathcal{L}_{reg}(\boldsymbol{\sigma}) \tag{4}$$

where the masks $\mathcal{M}_{\boldsymbol{\sigma}}$ is written in an abstract manner, meaning that each mask M_i is determined by Top-K (defined in Eq. 1). As the Top-k operator is not a smooth operator, we use the so-called Straight-through Estimator (Bengio et al., 2013) to compute the gradient with respect to both $\boldsymbol{\sigma}$ and $\mathcal{S}$. That is to say, the gradient through Top-K operator is artificially set to be 1. With such a regularization defined in Eq. 4, there exits "competition" between σ_i in $\mathcal{L}_{\text{pure}}$ and σ_i in $\mathcal{L}_{reg}$. Particularly, σ_i in $\mathcal{L}_{\text{pure}}$ tends to make $k(\sigma_i)$ close to 1 as the dense model generally gives better accuracy performance, while σ_i in $\mathcal{L}_{reg}$ makes $k(\sigma_i)$ close to the target ratio R_{target}. Notably, our regularization method is fundamentally different from those soft-threshold methods by using L_0 or L_1 regularization. While they apply a penalty to the score matrices with indirect control on final sparsity, our method focus on learnable sparsity thresholds σ_i. Thus, we could easily achieve our target compression ratios. On the other hand, one may add L_0 or L_1 regularization to Eq. 4 as the two are complementary.

Critical term $R(\boldsymbol{\sigma})$ We now delve into the calculation of $R(\boldsymbol{\sigma})$. For simplicity, we consider that all three matrices M_i, W_i, and S_i follow the same dimensions $d^i_{\text{in}} \times d^i_{\text{out}}$. Then $R(\boldsymbol{\sigma}) = N_{\text{remain}}(\boldsymbol{\sigma})/N_{\text{total}}$, where the total number of weight parameters $N_{\text{total}} = \sum_{i=1}^{n}(d^i_{\text{in}} \times d^i_{\text{out}})$, and the number of remaining parameters $N_{\text{remain}}(\boldsymbol{\sigma}) = \sum_{i=1}^{n} k(\sigma_i)(d^i_{\text{in}} \times d^i_{\text{out}})$.

Adaptive regularization coefficient λ_{reg} Generally, for regularization-based (e.g., L_1 or L_0 regularization) pruning methods, λ_{reg} needs to be carefully tuned (Sanh et al., 2020). To resolve this tuning issue, we propose an adaptive formula to choose the value:

$$\lambda_{reg} = \max\left\{\lambda_{max}\mathcal{L}_{reg}/(1 - R_{target})^2, \lambda_{min}\right\}, \tag{5}$$

where λ_{max} and λ_{min} are pre-chosen hyperparameters.We found that our results are not sensitive to the choice of these hyper-parameters. The idea is that when $R(\boldsymbol{\sigma})$ is far away from the R_{target}, the new coefficient λ_{reg} in Eq. 5 is close to λ_{max} (when $R(\boldsymbol{\sigma}) = 1$, it is indeed λ_{max}) so that we can have a strong regularization effect; and when R is close to R_{target}, the penalty can be less heavy in Eq. 4. Detailed comparison between constant and our proposed adaptive regularization are referred to Section 3.

3 Experimental Setup and Results

We apply LEAP with task-specific pruning for BERT$_{\text{base}}$, a 12-layer encoder-only Transformer model (Devlin et al., 2019) on three tasks QQP, MNLI and SQUAD for unstructured, hybrid and

	Methods	Density 1	Density 2	Density 3
QQP		9∼10%	3∼4%	1∼2%
	Soft MvP	90.2/86.8	89.1/85.5	N/A
	LEAP	90.4/87.1	90.3/87.0	89.6/86.0
	LEAP-l	**90.9/87.9**	**90.6/87.4**	**90.4/87.1**
MNLI		12∼13%	10∼11%	2∼3%
	Soft MvP	N/A	81.2/81.8	79.5/80.1
	LEAP	81.3/81.5	80.6/81.0	79.0/79.3
	LEAP-l	**82.2/82.2**	**81.7/81.7**	**80.3/80.2**
SQuAD		9∼10%	5∼6%	3∼4%
	Soft MvP	76.6/84.9	N/A	72.7/82.3
	LEAP	77.0/85.4	74.3/83.3	72.9/82.5
	LEAP-l	**78.7/86.7**	**75.9/84.5**	**75.7/84.5**

Table 1: Different density ratios for unstructured pruning. Here Soft MvP is referred to (Lagunas et al., 2021). Here LEAP uses exactly training strategies as (Lagunas et al., 2021) and LEAP-l doubles the training epochs. We report accuracy/F1 socre for QQP, accuracy of match and mis-match sets for for MNLI, exact match/F1 score for SQuAD.

	Methods	Density 1	Density 2	Density 3
QQP		27∼30%	21∼25%	11∼15%
	soft MvP-1 (H_{32})	NA/87.6	NA/87.1	NA/**86.8**
	LEAP-1 (H_{32})	91.2/**88.0**	91.0/**87.9**	90.7/85.5
	LEAP-1 (S_{32})	91.0/87.9	90.9/87.7	90.5/87.3
MNLI		27∼30%	17∼21%	11∼15%
	soft MvP-1 (H_{32})	**83.0/83.6**	**82.3/82.7**	81.1/**81.5**
	LEAP-1 (H_{32})	**83.0**/83.2	82.2/82.4	**81.5/81.5**
	LEAP-1 (S_{32})	82.0/82.1	81.0/81.0	80.0/80.0
SQuAD		27∼30%	21∼25%	15∼19%
	soft MvP-1 (H_{32})	**80.5/88.7**	**79.3**/86.9	**78.8/86.6**
	LEAP-1 (H_{32})	80.1/87.6	**79.3/87.0**	78.2/86.1
	soft MvP-1 (S_{32})	**77.9**/85.6	77.1/85.2	N/A
	LEAP-1 (S_{32})	**77.9/85.9**	**77.2/86.4**	N/A

Table 2: Hybrid and structured pruning comparison between LEAP and Soft MvP (Lagunas et al., 2021).See Table 1 for task metrics and reading instructions

structured block-wise pruning. Please see the detailed experimental setup in Appendix C). We compare with (Sanh et al., 2020; Lagunas et al., 2021) but not other methods for the two reasons: (1) our training setup are close to them and they are the current stat-of-the-art methods for BERT models. (2) in (Sanh et al., 2020; Lagunas et al., 2021), there already exist extensive comparisons between different methods, including hard-threshold and L_0 regularization.

Unstructured pruning We show the results of LEAP under different density ratios of BERT$_{\text{base}}$ on QQP/MNLI/SQuAD. As can be seen, compared to Soft MvP, LEAP achieves better performances on 4 out of 6 direct comparisons (as Soft MvP only provides two pruning ratios per task). Particularly, for QQP, LEAP is able to reduce the density ratio to 1∼2% while achieving similar performance as Soft MvP with 3∼4% density ratio; for MNLI, although LEAP is slightly worse than Soft MvP, the performance gap is within 0.6 for all cases.

We also list the results of LEAP-l, which utilizes more training epochs to boost the performance, in Table 1. One hypothesis to explain why longer training can significantly boost the performance of LEAP is that LEAP introduces both more learnable hyperparameters and the adaptive regularization magnitude. As such, those extra parameters need more iterations to reach the "optimal" values (which is also illustrated in Section E).

Hybrid and structure pruning We start with hybrid pruning and compare LEAP-1 with Soft MvP-1. The results are shown in Table 2. Again, as can be seen, for different tasks with various pruning ratio, the overall performance of LEAP-1 is similar to Soft MvP-1, which demonstrates the easy adoption feature of LEAP. We also present structured pruning results in Table 2. The first noticeable finding as expected here is that the accuracy drop of structured pruning is much higher than hybrid mode, especially for a low density ratio. Compared to Soft MvP-1, LEAP-1 achieves slightly better performance on SQuAD.

We reiterate that *Table 1 and Table 2 are not about beating the state-of-the-art results but emphasizing that LEAP requires much less hyper-parameter tuning but achieves similar performance as Soft MvP that involved a large set of the hyper-parameter sweep.* For details about hyper-parameter tuning of Soft MvP-1 and LEAP, see Appendix D. In addition, we provide in-depth analysis of LEAP in Appendix E.

4 Conclusions

In this work, we present LEAP, a learnable pruning framework for transformer-based models. To alleviate the hyperparameter tuning effort, LEAP introduces a novel regularization function and designs an adaptive regularization magnitude coefficient. By combining these two techniques, LEAP achieves on-par or even better performance for various pruning scenarios as compared to previous methods. LEAP is less sensitive to the newly introduced hyperparameters and show the advance of the proposed adaptive regularization coefficient. Finally, we show that there exists pruning sensitivity associated with the depth of the network.

References

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. 2020. BinaryBERT: Pushing the limit of BERT quantization. *arXiv preprint arXiv:2012.15701*.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model. *arXiv preprint arXiv:1906.00532*.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pre-trained BERT networks. *arXiv preprint arXiv:2007.12223*.

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

Xin Dong, Shangyu Chen, and Sinno Pan. 2017. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867.

Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. 2019. Learned step size quantization. *arXiv preprint arXiv:1902.08153*.

Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*.

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. 2020. Training with quantization noise for extreme fixed-point compression. *arXiv preprint arXiv:2004.07320*.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

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

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2014. Distilling the knowledge in a neural network. *Workshop paper in NIPS*.

Zehao Huang and Naiyan Wang. 2018. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320.

Forrest N Iandola, Albert E Shaw, Ravi Krishna, and Kurt W Keutzer. 2020. SqueezeBERT: What can computer vision teach NLP about efficient neural networks? *arXiv preprint arXiv:2006.11316*.

Shankar Iyer, Nikhil Dandekar, and Kornl Csernai. 2017. First quora dataset release: Question pairs, 2017. *URL https://data. quora. com/First-Quora-Dataset-Release-Question-Pairs*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. TinyBERT: Distilling BERT for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. 2021. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10619–10629.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*.

Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. 2018. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*, pages 2425–2432.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *arXiv preprint arXiv:1905.10650*.

Microsoft and Nvidia. 2021. Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, the World's Largest and Most Powerful Generative Language Model. https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/.

Sharan Narang, Eric Undersander, and Gregory Diamos. 2017. Block-sparse recurrent neural networks. *arXiv preprint arXiv:1711.02782*.

Sejun Park, Jaeho Lee, Sangwoo Mo, and Jinwoo Shin. 2020. Lookahead: a far-sighted alternative of magnitude-based pruning. *arXiv preprint arXiv:2002.04809*.

Sai Prasanna, Anna Rogers, and Anna Rumshisky. 2020. When BERT plays the lottery, all tickets are winning. *arXiv preprint arXiv:2005.00561*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2020. Poor man's BERT: Smaller and faster transformer models. *arXiv preprint arXiv:2004.03844*.

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*.

Victor Sanh, Thomas Wolf, and Alexander M Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *arXiv preprint arXiv:2005.07683*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-BERT: Hessian based ultra low precision quantization of bert. In *AAAI*, pages 8815–8821.

Sheng Shen, Zhewei Yao, Douwe Kiela, Kurt Keutzer, and Michael W Mahoney. 2021. What's hidden in a one-layer randomly weighted transformer? *arXiv preprint arXiv:2109.03939*.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact task-agnostic BERT for resource-limited devices. *arXiv preprint arXiv:2004.02984*.

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

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

Hanrui Wang, Zhekai Zhang, and Song Han. 2020a. Spatten: Efficient sparse attention architecture with cascade token and head pruning. *arXiv preprint arXiv:2012.09852*.

Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. 2020b. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2019. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Xia Xiao, Zigeng Wang, and Sanguthevar Rajasekaran. 2019. Autoprune: Automatic network pruning by regularizing auxiliary parameters. In *Advances in Neural Information Processing Systems*, pages 13681–13691.

Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S Morcos. 2019. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. *arXiv preprint arXiv:1906.02768*.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. 2018. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203.

Shixing Yu, Zhewei Yao, Amir Gholami, Zhen Dong, Michael W Mahoney, and Kurt Keutzer. 2021. Hessian-aware pruning and optimal neural implant. *arXiv preprint arXiv:2101.08940*.

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. *arXiv preprint arXiv:1910.06188*.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*.

Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. 2019. Variational convolutional neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2780–2789.

Mengjie Zhao, Tao Lin, Fei Mi, Martin Jaggi, and Hinrich Schütze. 2020. Masking as an efficient alternative to finetuning for pretrained language models. *arXiv preprint arXiv:2004.12406*.

Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.

A Related Work

Different approaches have been proposed to compress large pre-trained NLP models. These efforts can be generally categorized as follows: (i) knowledge distillation (Jiao et al., 2019; Tang et al., 2019; Sanh et al., 2019; Sun et al., 2019); (ii) quantization (Bhandare et al., 2019; Zafrir et al., 2019; Shen et al., 2020; Fan et al., 2020; Zadeh et al., 2020; Zhang et al., 2020; Bai et al., 2020; Esser et al., 2019); (iii) new architecture design (Sun et al., 2020; Iandola et al., 2020; Lan et al., 2019; Kitaev et al., 2020; Wang et al., 2020b); and (iv) pruning. Pruning can be broadly categorized into unstructured pruning (Dong et al., 2017; Lee et al., 2018; Xiao et al., 2019; Park et al., 2020; Han et al., 2016; Sanh et al., 2020) and structured pruning (Luo et al., 2017; He et al., 2018; Yu et al., 2018; Lin et al., 2018; Huang and Wang, 2018; Zhao et al., 2019; Yu et al., 2021; Michel et al., 2019). Here, we briefly discuss the related pruning work in NLP.

For unstructured pruning, (Yu et al., 2019; Chen et al., 2020; Prasanna et al., 2020; Shen et al., 2021) explore the lottery-ticket hypothesis (Frankle and Carbin, 2018) for transformer-based models; (Zhao et al., 2020) shows that pruning is an alternative effective way to fine-tune pre-trained language models on downstream tasks; and (Sanh et al., 2020) proposes the so-called movement pruning, which considers the changes in weights during fine-tuning for a better pruning strategy, and which achieves significant accuracy improvements in high sparsity regimes. However, as an extension of (Narang et al., 2017), (Sanh et al., 2020) requires non-trivial hyperparameter tuning to achieve better performance as well as desired pruning ratio.

For structured pruning, (Fan et al., 2019; Sajjad et al., 2020) uses LayerDrop to train the model and observes that small/efficient models can be extracted from the pre-trained model; (Wang et al., 2019) uses a low-rank factorization of the weight matrix and adaptively removes rank-1 components during training; and (Michel et al., 2019) tests head drop for multi-head attention and concludes that a large percentage of attention heads can be removed during inference without significantly affecting the performance. More recently, (Lagunas et al., 2021) extends (Sanh et al., 2020) from unstructured pruning to block-wise structured pruning. As a continuing work of (Sanh et al., 2020; Narang et al., 2017), hyperparameter tuning is also critical for (Lagunas et al., 2021).

Although fruitful pruning algorithms are proposed, most methods generally only work for specific pruning scenarios, e.g., unstructured or structured pruning. Also, a lot of algorithms either (i) need a hand-tuned threshold (aka pruning ratio) to achieve good performances; or (ii) require careful regularization magnitude/schedule to control the final pruning ratio and retain the model quality. Our LEAP is a general pruning algorithm that achieves on-par or even better performance under similar pruning ratio across various pruning scenarios as compared to previous methods, and LEAP achieves this with very minimal hyperparameter tuning by introducing a new regularization term and a self-adaptive regularization magnitude.

B Problems of Existing Hard/Soft-threshold Pruning Methods

In this section, we continue a more detailed description on problems of existing hard/soft-threshold pruning methods. In (Zhu and Gupta, 2017; Sanh et al., 2020; Lagunas et al., 2021), s_t^i in (S2) Section 2.1 is set to be the same across all the weight matrices, i.e., $s_t^i := s_t$ and they use a cubic sparsity scheduling for the target sparsity s_f given a total iterations of t_f:

$$s_t = \begin{cases} s_0 & 0 \le t < t_0, \\ s_f + (s_0 - s_f)(1 - \frac{t-(t_0+t_c)}{t_f-(t_0+t_c)})^3 & t_0 \le t < t_f - t_c, \\ s_f & t \ge t_c. \end{cases} \tag{6}$$

Although threshold methods achieve reasonably advanced pruning ratios along with high model qualities, they also exhibit various issues.

Common issues Both hard- and soft- threshold pruning introduce three hyperparameters: the initial sparsity value s_0, the warmup step t_0, and the cool-down steps t_c. As a common practical issue, more hyperparameters need more tuning efforts, and the question, how to choose the hyperparameters v_0, t_0 and t_f, is by no means resolved.

Issues of hard-threshold pruning It is natural for weight matrices to have different tolerances/sensitivities to pruning, which means that a high pruning ratio needs to be applied for insensitive layers, and vice versa. However, for hard-threshold pruning, which sorts the weight in one layer by absolute values and masks the smaller portion (i.e., s_t^i) to zero, it uses the same pruning ratio across all layers. That oftentimes leads to a sub-optimal solution for hard-threshold pruning.

As such instead of using a single s_t schedule, a more suitable way is to use different $s_t^i,\ i = 1, \dots n$ for each weight matrix W_i. However, this leads the number of tuning hyperparameters to be a linear function as the number of weight matrices, i.e., $3n$. For instance, there are $3 \times 6 \times 12 = 216$ hyperparameters for the popular NLP model–BERT$_{\text{base}}$, a 12-layer encoder-only Transformer of which each layer consists of 6 weight matrices (Devlin et al., 2019). Extensively searching for these many hyperparameters over a large space is impractical.

Except for the single threshold issue, the hard-threshold method is hard to extend to different pruning scenarios, e.g., block-pruning, head pruning for attention heads, and filter pruning for fully connected layers. The reason is that the importance of those structured patterns cannot be simply determined by their sum of absolute values or other norms such as the Euclidean norm.

Issues of soft-threshold pruning One way to resolve the above issues is through soft-threshold methods. Instead of using the magnitude (aka absolute value) of the weight matrix to generate the mask, soft-threshold methods introduce a regularization (penalty) function $\mathcal{L}_{reg}(\mathcal{S})$ to control the sparsity of the weight parameters (for instance, L_p-norm, $\mathcal{L}_{reg} = \|\cdot\|_p$, with $p = 0$ or $p = 1$). Here, $\mathcal{S} := \{S_i\}_{i=1}^n$ and each S_i refers to the associated importance score matrix of W_i, which is learnable during training. Particularly, (i) this S_i can be adopted to different pruning granularity, e.g., structured and unstructured pruning, and (ii) the final pruning ratio of each weight matrix can be varied thanks to the learnable nature of S_i.

For soft-threshold pruning, the mask, M_i, is generated by the learnable importance score S_i and s_t[2] using the comparison function, $M_i = f(S_i) > s_t$.[3] Where $f(\cdot)$ is any function that maps real values to $[0,\ 1]$. As $f(S_i)$ will prefer larger values as smaller loss will be introduced to training procedure, a regularization term is added to the training objective,

$$\mathcal{L}_{\text{obj}}(\mathcal{M} \odot \mathcal{W}) = \mathcal{L}_{\text{pure}}(\mathcal{M} \odot \mathcal{W}) + \lambda_{reg}\mathcal{L}_{reg}(f(\mathcal{S})) \tag{7}$$

The coefficient λ_{reg} is used to adjust the magnitude of the penalty (the larger λ_{reg}, the sparser the $\mathcal{W}$). Although soft-threshold pruning methods achieve better performance as compared to hard-threshold pruning methods, it introduces another hyperparameter λ_{reg}. More importantly, as the final sparsity is controlled indirectly by the regularization term, it requires sizable laborious experiments to achieve the desired compression ratio.

C Experimental Setup

We apply LEAP with task-specific pruning for BERT$_{\text{base}}$, a 12-layer encoder-only Transformer model (Devlin et al., 2019), with approximately 85M parameters excluding the first embedding layer. We focus on three monolingual (English) tasks: question answer (SQuAD v1.1) (Rajpurkar et al., 2016); sentence similarity (QQP) (Iyer et al., 2017); and natural language inference (MNLI) (Williams et al., 2017). For SQuAD, QQP, and MNLI, there are 88K, 364K, 392K training examples respectively.

In order to do a fair comparison with Soft MvP (Lagunas et al., 2021), for all tasks, we perform logit distillation to boost the performance (Hinton et al., 2014). That is,

$$\mathcal{L}_{obj} = \alpha\mathcal{L}_{ds} + (1-\alpha)\mathcal{L}_{\text{ce}}(\mathcal{M}(\boldsymbol{\sigma}) \odot \mathcal{W}) + \lambda_{reg}\mathcal{L}_{reg}(\boldsymbol{\sigma}) \tag{8}$$

Here, $\mathcal{L}_{ds}$ is the KL-divergence between the predictions of the student and the teacher, $\mathcal{L}_{\text{ce}}$ is the original cross entropy loss function between the student and the true label, and α is the hyperparameter that balances the cross-entropy loss and the distillation loss. We let $\alpha = 0.9$ by default for fair comparison

[2]When all s_t^i are the same, we drop the superscript for simplicity.

[3]Here both the function $f(\cdot)$ and the comparison are element wise and the comparison returns either 1 or 0.

Pruning Settings	MHA	FC
Hybrid (H_{32})	32×32	1×1
Structure (S_{32})	32×32	32×32
Structure (S_{16})	16×16	16×16
Structure (S_8)	8×8	8×8
Unstructure (S_1)	1×1	1×1

Table 3: Summary of different pruning settings. Here the first column shows the abbreviate name we will refer to later, the second column shows the block size used for multi-head attention (MHA), and the third column shows the block size used for fully-connected layers (FC).

(One might be able to improve the results further with more careful hyperparameter tuning and more sophisticated distillation methods).

Training Details For all three tasks, the temperature parameter T for $k(\sigma_i)$ is chosen between $\{16, 32, 48, 64\}$ and $\lambda_{\max}$ varies between $\{40, 80, 160, 320\}$ with $\lambda_{\min} = 10$. For initialization of σ_i, we set it to be $5T$. We use a batch size 32, a sequence 128 for QQP/MNLI and 11000/12000 warmup steps (about 1 epoch) for learning rate. As for SQuAD, we use a batch size 16 and a sequence 384, and we use 5400 warmup steps (about 1 epoch). We use a learning rate of 3e-5 (1e-2) for the original weights (for pruning-rated parameters, i.e., $\mathcal{S}$ and $\boldsymbol{\sigma}$). We set all the training to be deterministic with the random seed of 17. All the models are trained using FP32 with PyTorch on a single V100 GPU. Note that these configurations strictly follow the experimental setup in (Sanh et al., 2020; Lagunas et al., 2021); readers could check more details there. For the results in Table 1, the entire epoch using LEAP is 10, 6, and 10, respectively, for QQP, MNLI, and SQuAD. For the results of LEAP-1 and the results in Table 2, we simply double training epochs correspondingly (i.e., 20, 12, and 20).

Structured/Unstructured/Hybrid pruning The basic ingredients of the transformer-based layer consist of multi-headed attention (MHA) and fully connected (FC) sub-layers. We denote the sets of weight matrices $\mathcal{W}_{\text{att}}$ for MHA and $\mathcal{W}_{\text{fc}}$ for FC. Before we give details on the three pruning settings (Structured, Unstructured, and Hybrid), we first explain square $d \times d$ block-wise pruning. Consider an output matrix as $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, where $d_{\text{in}} = dr$ and $d_{\text{out}} = dc$ (here r and c are integer by design). We will define a mask M and a score S with the dimension of $r \times c$ for the matrix W. Given a score $[S]_{i,j}$, if the Top-K operator returns $[M]_{i,j} = 0$, then all d^2 elements in the (i, j)-th block of W will be set to 0; otherwise, $[M]_{i,j} = 1$ means keeping those elements.

In our experiments, *structured pruning* refers to applying block-wise pruning to both sets, i.e., $\mathcal{W}_{\text{att}}$ and $\mathcal{W}_{\text{fc}}$. In addition, we make the square block size the same in both MHA and FC sub-layers and we choose $d = 32$. *Unstructured pruning* is using $d = 1$ for MHA and FC. *Hybrid pruning* means using structured pruning for MHA (setting the block size to be $d = 32$) and using unstructured one for FC ($d = 1$). As such, there are three different sets of experiments and we summarize them in Table 3. We test our methods with the scores $\mathcal{S}$ described in movement pruning (Sanh et al., 2020; Lagunas et al., 2021) over three datasets across unstructured, hybrid, and structured pruning setups. Moreover, we follow strictly (Lagunas et al., 2021) (referred to as Soft Pruning in later text) on the learning rate including warm-up and decaying schedules as well as the total training epochs to make sure the comparison is fair. Let LEAP-l and soft MvP-1 denote a double-epoch training setup compared to LEAP and Soft Pruning (Sanh et al., 2020). For more training details, see Appendix C.

D More Results Details

Smaller tasks Note larger datasets (QQP/MNLI/SQuAD) to evaluate the performance of the pruning method is very common due to the evaluation robustness. However, to illustrate the generalization ability of LEAP, we also tested its performance on two smaller datasets STS-B and MPRC, using block pruning with size 32x32. The results are shown in Table 1. As can be seen, with around 20% density ratio, LEAP still achieves marginal accuracy degradation compared to baseline.

	Spearman correlation				Density ratio		
STS-B	T=1	T=2	T=4		T=1	T=2	T=4
λ_{max} =80	85.68	85.86	85.96		20.0	20.1	22.5
λ_{max} =160	85.73	85.91	86.19		20.0	20.1	26.0
λ_{max} =320	85.72	86.01	86.46		20.0	20.3	28.5
	Accuracy				Density ratio		
MRPC	T=1	T=2	T=4		T=1	T=2	T=4
λ_{max} =80	82.1	82.6	79.16		20.0	20.0	20.3
λ_{max} =160	81.37	82.35	79.65		20.0	20.0	21.3
λ_{max} =320	80.88	81.37	78.67		20.0	20.0	22.7

Table 4: Results for STS-B (baseline is 88.71) and MRPC (baseline is 87.01%) with different temperature T and adaptive λ_{max} for structure pruning (block size 32x32).

Hyper-parameter We emphasize again the results of soft mvp is a strong baseline, and our goal is not to purely beat soft mvp from accuracy perspective. However, their results require extensive hyperparameter tuning (see directory), while ours require to only tune T. To show the generalization of the best hyperparameter, we include the results for various λ_{max} and T on multiple tasks in Table 2. Note that when T is fixed, different λ_{max} gives similar results over various tasks.

	Accuracy				Density ratio		
QQP	T=16	T=32	T=48		T=16	T=32	T=48
λ_{max}=160	90.68	90.87	90.7		20.07	20.07	20.1
λ_{max}=320	90.79	90.78	90.6		20.08	20.06	20.1
	Accuracy (MNLI/MNLI-MM)				Density ratio		
MNLI	T=16	T=32	T=48		T=16	T=32	T=48
λ_{max}=40	80.41/81.06	80.79/81.12	80.87/81.22		21.07	21.52	22.25
λ_{max}=160	80.56/80.98	80.88/80.81	81.02/81.17		21.07	21.46	22.15

Table 5: Results for QQP and MRPC with different temperature T and adaptive λ_{max} for structure pruning (block size 32x32).

E Analysis

As mentioned, LEAP is a learnable pruning method with a minimal requirement of hyperparameter tuning. In order to demonstrate this, we analyze LEAP by delving into the key components of LEAP: the initialization of our thresholds $\boldsymbol{\sigma}$, the temperature T, and the regularization term λ_{reg}.

Temperature T As T defined in Eq. 2, it plays a critical role in determining the rate at which the threshold curve $k(\sigma_i)$ falls. In addition, T also directly links to the initialization of σ_i which is set to be $5T$ for all i such that Sigmoid$(\sigma_i/T) \approx 1$. This allows the model to have sufficient time to identify the layers which are insensitive for aggressive pruning and vice versa. To understand how T influences the performances of the Bert model, we conduct an unstructured pruning on the QQP dataset by varying $T \in \{64, 48, 32, 16\}$ and keeping all other hyperparameters to be the same. We plot the objective loss $\mathcal{L}_{\text{obj}}$ (loss), the regularization loss $\mathcal{L}_{\text{reg}}$ (regu_loss), the density ratio $R(\boldsymbol{\sigma})$, and F1 accuracy, with respect to the iterations in Figure 1.

Temperature	$\lambda_{reg} = 50$		$\lambda_{reg} = 160$		$\lambda_{reg} = 320$	
	acc/f1	density	acc/f1	density	acc/f1	density
$T = 16$	76.58/84.94	10.66	76.49/85.01	10.27	76.33/84.85	10.2
$T = 32$	77.11/85.49	11.46	76.97/85.47	10.39	76.96/85.36	10.23

Table 6: Unstructured pruning on SQuAD with epoch 10 using various values of regularization coefficient λ_{reg} in Eq. 4. It shows that our LEAP is not too sensitive to the hyper-parameter choices T and λ_{reg}.

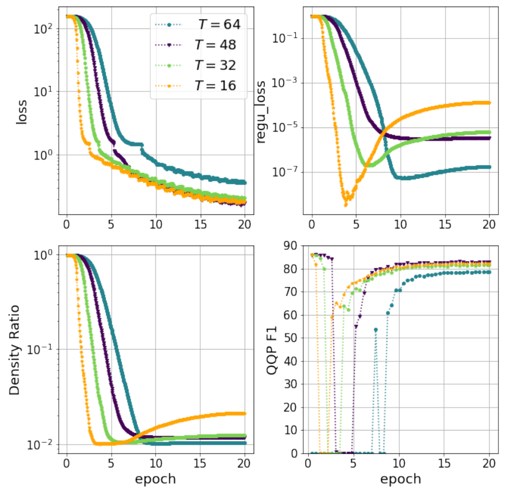

Figure 1: Effect of temperature T for unstructured pruning on QQP. The density ratio is set to be 1%.

Among the four curves in Figure 1, $T = 48$ gives the best F1 accuracy while achieving $\sim$1% density, which clearly demonstrates the significance of T for LEAP. Meanwhile, we see that the gaps between the performance for all T except 64 are close, thus it shows that LEAP is not sensitive to T. A possible explanation why $T = 64$ gives the worse performance is that the density ratio of $T = 64$ decays relatively slower compared to rest curves. As such, when it is close to the desired pruning regime, the learning rate is relatively small and so it cannot be able to recover the accuracy. On the other hand, it is interesting to note that using the temperature $T = 16$ (orange curve), the density ratio increases after around five epochs and keeps increasing to the end[4], which results in a much better performance even though it experiences the most accuracy drop in the beginning. This in some scenes illustrates the "competition" between σ_i in $\mathcal{L}_{\text{pure}}$ and σ_i in $\mathcal{L}_{reg}$ mentioned in Section 2.2: the accuracy increases at epoch 5 meaning that $\mathcal{L}_{\text{pure}}$ is decreasing effectively and the $\mathcal{L}_{reg}$ increases (compromises). Compared to those manual scheduling thresholds, this increasing phenomena of σ_i also shows the advantage of learnable thresholds verifying that the model can figure out automatically when to prune and when not.

Robustness of hyper-parameter tuning T and λ_{reg} We see in the previous section that given the same λ_{reg}, various values of the temperature T lead to similar results although tuning is necessary to achieve the best one. Here we study how robust the coefficient of λ_{reg} in our proposed regularization $\mathcal{L}_{reg}$. We prune BERT$_{\text{base}}$ on the SQuAD task with a target ratio 10% with a combination of $\lambda_{reg} \in \{50, 160, 320\}$ and $T \in \{16, 32\}$, for which the results is in Table 6.

For a given T, it indicates that the results are not highly sensitive to different λ_{reg}s as there is only about 0.1 variation for accuracy. It is worth noticing that a smaller λ_{reg} (here $\lambda_{reg} = 50$) can indeed affect achieving our target sparse ratio. However, the most off pruning ratio is 11.46%, which is reasonably close to the desired target of 10%.

For a given λ_{reg}, larger T leads both the accuracy and the density ratio higher as expected. The reason is that the density ratio function, i.e., Sigmoid(σ_i/T), becomes flatter for larger T, which leads to a higher density ratio by using the same value of σ (Generally, σ is negative to achieve $< 50\%$ density ratio). And higher density ratio results in higher accuracy.

Overall, we can see that LEAP is robust to λ_{reg}. Although one still needs to tune T to obtain the most competitive performance, the tuning efforts are much less than the half/soft-thresthod pruning methods as discussed in Appendix 2.1.

[4]Please note that the y-axis of the density plot is in logarithmic scale. Even $T = 16$ slightly increases the density ratio, it is still very close to 1%.

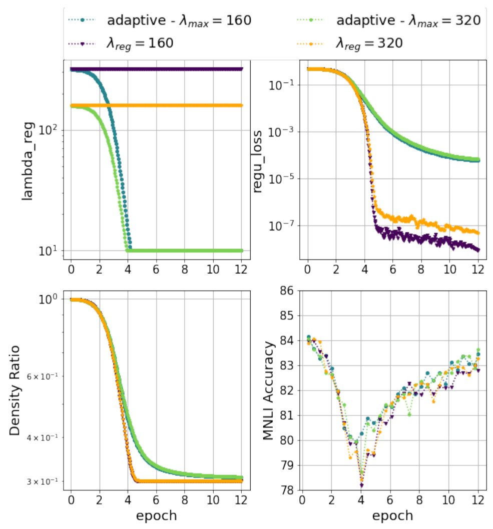

Figure 2: Effect of adaptive regularization T for Hybrid pruning (H_{32}) on MNLI with a target dense ratio of 30%. Note that in the plot of density ratio with respect to epochs (left bottom), the purple (blue) and orange (green) curves are overlapped. Also in the right top bottom, blue and green curves are overlapped.

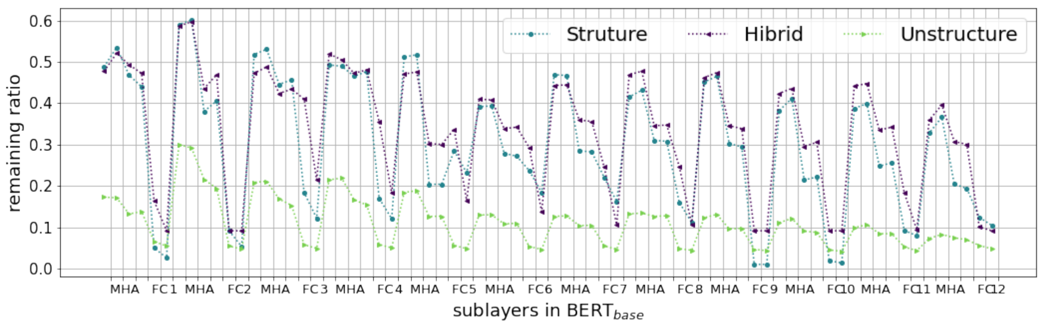

Figure 3: The density ratio $k(\sigma_i)$ to all the weight matrices for structured, hybrid and unstructured pruning on SQuAD, of which the total density ratios are respectively 20% , 16%, and 8%.

The regularization coefficient λ_{reg} To better understand the effect of adaptive λ_{reg} (Eq. 5), we set $\lambda_{max} \in \{160, 320\}$ and fix $\lambda_{min} = 10$ (same as Section 3) to prune BERT$_{\text{base}}$ on the MNLI task with a target ratio 30%. In addition, we also compare this adaptive coefficient with their constant counterparts $\lambda_{reg} \in \{160, 320\}$. We plots the λ_{reg} (lambda_reg), the regularization loss $\mathcal{L}_{\text{reg}}$ (regu_loss), the density ratio $R(\boldsymbol{\sigma})$, and accuracy, with respect to the iterations in Figure 2. First of all, we see that our adaptive coefficient λ_{reg} decreases in a quadratic manner and reaching to the $\lambda_{min} = 10$ after 4 epochs, which slows down the pruning activities after 4 epochs. Also, note that the curves of different λ_{max} are actually overlapped with each other, which also indicates that LEAP is not vulnerable to λ_{reg}. Meanwhile, as λ_{reg} quickly reaches λ_{min}, the importance score $\mathcal{S}$ has more time to figure out the pruning parameters for the last small portion. As such, this slowness can in turn decrease the drop of accuracy and thus eventually recover a much better accuracy than that of the constant regularization.

The effect of learnable pruning for different weight matrices As mentioned, the sensitivities of different weight matrices are different. Therefore, a high pruning ratio should be set for insensitive layers, and a low pruning ratio needs to be used for sensitive layers. To demonstrate LEAP can automatically achieve this, we plot the remaining parameters per layer for different pruning granularity on SQuAD in Figure 3. As can be seen, different layers receive different pruning ratios. Particularly, (i) as compared to MHA layers, FC layers are generally pruned more, which results in a lower density ratio. This might indicate that FC layers are less sensitive as compared to MHA layers; (ii) there is a clear trend that shallow layers (close to inputs) have higher density ratios as compared to deep layers (close to outputs). This finding is very intuitive. If the pruning ratio is too high for shallow layers, the information loss might be too high, and it is hard for the model to propagate the information to the output layer successfully. Therefore, the pruning ratio of shallow layers is smaller.