A Results with More PLMs on subset of GLUE

In addition the widely used BERT and GPT-2 models, we also perform pruning experiments upon other two pre-trained language models: Electra$_{\text{base}}$ and MiniLM$_{\text{12L-384H}}$ to further verify the effectiveness of our method.

Due to computing resource constraint, we restrict our experiments on a subset of GLUE task, including RTE, CoLA and QNLI at 80% and 90% sparsity. We compare PINS against IMP and PLATON as two representative baselines for magnitude-based and sensitivity-based pruning methods. We fix the batch size as 32 and learning rate as 3e-5 similar to the BERT experiments. We illustrate the pruning results on Table 1 and Table 2. At both sparsity levels, PINS consistently outperforms IMP and PLATON on all three datasets, verifying the general effectiveness of PINS for language model pruning.

Sparsity	Method	**RTE**	**CoLA**	**QNLI**
		Acc	Mcc	Acc
0%	Fine-tune	73.0	58.5	91.5
80%	IMP	60.5	21.6	87.5
	PLATON	68.2	54.1	89.8
	PINS (ours)	69.5	54.4	90.4
90%	IMP	57.5	14.1	83.9
	PLATON	63.1	38.8	88.0
	PINS (ours)	66.2	44.8	88.6

Table 1: Results with MiniLM$_{\text{12L-384H}}$ on the GLUE development set.

Sparsity	Method	**RTE**	**CoLA**	**QNLI**
		Acc	Mcc	Acc
0%	Fine-tune	81.9	69.0	93.1
80%	IMP	59.9	11.2	87.5
	PLATON	73.6	60.0	91.0
	PINS (ours)	75.5	63.7	92.0
90%	IMP	52.9	0.0	83.0
	PLATON	69.9	48.0	89.7
	PINS (ours)	72.3	49.2	90.2

Table 2: Results with Electra$_{\text{base}}$ on the GLUE development set.

B Proof of Theorem 1

Proof. Let t_i and t_j where $t_i \geq t_j$ denote the time steps at which two different checkpoints are saved; Let $R(f_{\boldsymbol{\theta}^{(t\leftarrow t_i)}})$ and $R(f_{\boldsymbol{\theta}^{(t\leftarrow t_j)}})$ denote the expected generalization error of models learned from two checkpoints $\boldsymbol{\theta}^{(t_i)}$ and $\boldsymbol{\theta}^{(t_j)}$; Let n denotes the size of training data; $|\cdot|_{\text{C}}$ denotes a function class capacity measure like VC-dimension. Based on previous expositions on VC theory, the following asymptotic generalization bound holds:

$$\begin{aligned} R(f_{\boldsymbol{\theta}^{(t\leftarrow t_i)}}) &= R(f_{\boldsymbol{\theta}^{(t\leftarrow t_i)}}) - R(f_{\boldsymbol{\theta}^{(t_i)}}) + R(f_{\boldsymbol{\theta}^{(t_i)}}) \\ &\leq O(\frac{|f_{\boldsymbol{\theta}^{(t)}}|_{\text{C}}}{n^{\alpha_i}}) + \epsilon_{t,t_i} + R(f_{\boldsymbol{\theta}^{(t_i)}}) \\ &= \underbrace{O(\frac{|f_{\boldsymbol{\theta}^{(t)}}|_{\text{C}}}{n^{\alpha_i}}) + \inf_{\boldsymbol{\theta}^{(t)}\in\mathcal{F}_{\boldsymbol{\theta}^{(t\leftarrow t_i)}}} R(f_{\boldsymbol{\theta}^{(t)}})}_{bound(f_{\boldsymbol{\theta}^{(t\leftarrow t_i)}})} \end{aligned}$$

$$\begin{aligned} R(f_{\boldsymbol{\theta}^{(t\leftarrow t_j)}}) &= R(f_{\boldsymbol{\theta}^{(t\leftarrow t_j)}}) - R(f_{\boldsymbol{\theta}^{(t_j)}}) + R(f_{\boldsymbol{\theta}^{(t_j)}}) \\ &\leq O(\frac{|f_{\boldsymbol{\theta}^{(t)}}|_{\text{C}}}{n^{\alpha_j}}) + \epsilon_{t,t_j} + R(f_{\boldsymbol{\theta}^{(t_j)}}) \\ &= \underbrace{O(\frac{|f_{\boldsymbol{\theta}^{(t)}}|_{\text{C}}}{n^{\alpha_j}}) + \inf_{\boldsymbol{\theta}^{(t)}\in\mathcal{F}_{\boldsymbol{\theta}^{(t\leftarrow t_j)}}} R(f_{\boldsymbol{\theta}^{(t)}})}_{bound(f_{\boldsymbol{\theta}^{(t\leftarrow t_j)}})} \end{aligned}$$

where $\epsilon_{t,ti}$ is the approximation error of function class $\mathcal{F}_{\boldsymbol{\theta}^{(t\leftarrow t_i)}}$ with respect to $f_{\boldsymbol{\theta}^{(t_i)}}$. $\epsilon_{t,tj}$ is defined in analogy. Because: (1) $\boldsymbol{\theta}^{(t_i)}$ is a later checkpoint with higher sparsity than $\boldsymbol{\theta}^{(t_j)}$, we have the learning speed $1 \geq \alpha_i \geq \alpha_j \geq \frac{1}{2}$; (2) $\boldsymbol{\theta}^{(t_i)}$ has lower generalization error than $\boldsymbol{\theta}^{(t_j)}$, we have the following inequality holds with large probability:

$$bound(f_{\boldsymbol{\theta}^{(t\leftarrow t_i)}}) \leq bound(f_{\boldsymbol{\theta}^{(t\leftarrow t_j)}})$$

□

C More Post-pruning Analyses

This section presents more visualized analyses of models sparsified by different pruning methods.

Figure 2 shows the layerwise rank distribution of BERT$_{\text{base}}$ pruned using different importance criteria on the RTE dataset. The observation here is similar to what is discussed in the main body of the paper: PINS exhibits the lowest average matrix rank in the sparsified model compared to the other two criteria.

Figure 1 illustrates the weight distribution of BERT$_{\text{base}}$ pruning using different importance criteria. From the left figure we can see that magnitude-based pruning tends to keep parameters with high absolute values, which is expected based on its definition. Sensitivity and PINS produce similar weight value distribution mainly because the two methods both contain the $g\theta$ term

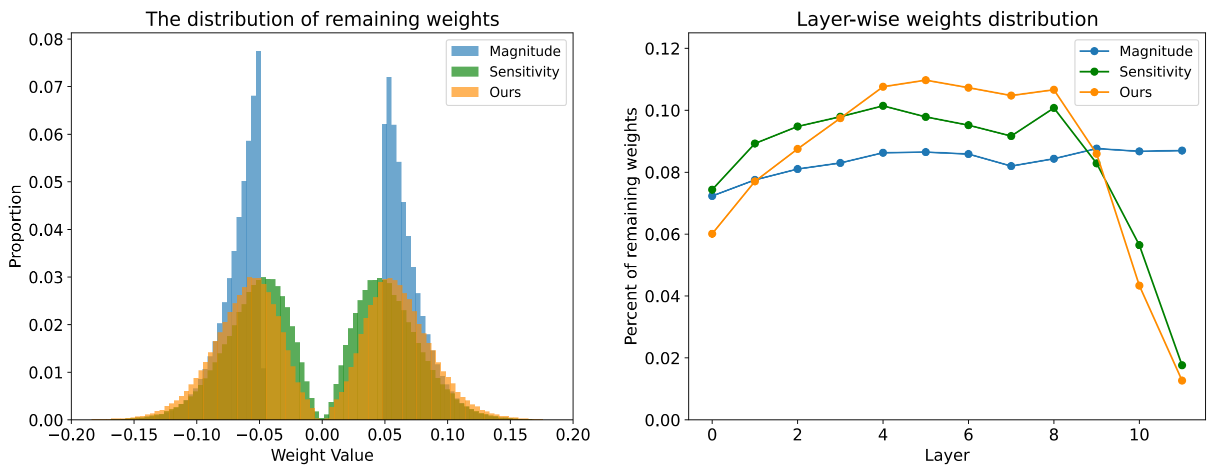

Figure 1: Weight distributions of BERT$_{\text{base}}$ pruned using different importance criteria on RTE dataset. Left figure shows the value distribution and the right figure shows how remaining parameters are distributed at different model layers.

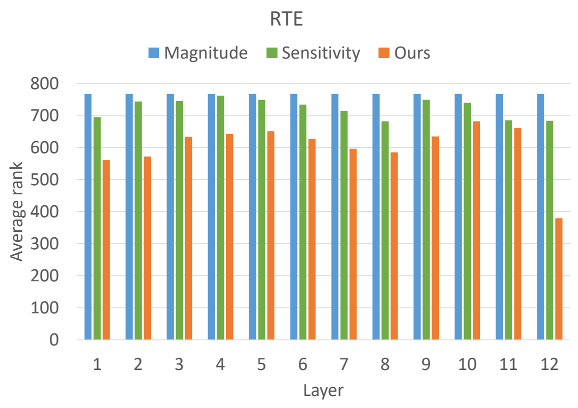

Figure 2: Layerwise rank distribution of BERT$_{\text{base}}$ pruning using different importance criteria on RTE dataset.

in their importance calculation. Despite the similarity, we can still observe that PINS produces smoother distribution than sensitivity and covers more weights with larger absolute values.

The right figure shows the layerwise distribution of remaining parameters after pruning. A clear trend is that PINS tends to retain more parameters in the middle layers (4-7), which also coincided with the inter-model sparsity pattern analysis in the main body of our paper. Both sensitivity and PINS remove a large proportion of parameters in the top layers (10-12) while magnitude-based pruning has no preference for model layers.

D Sparsity Scheduler

The proportion of remaining weights is controlled by the sparsity scheduler, here we adopt the commonly used cubic sparsity schedule to progressively reach target sparsity, i.e., $v^{(t)}$ at time step t within the maximum time steps T is given by:

$$\begin{cases} v_i & t \in [0, t_i) \\ v_f + (v_i - v_f)(\frac{T-t_f-t}{T-t_f-t_i})^3 & t \in [t_i, T-t_f) \\ v_f & \text{otherwise} \end{cases} \tag{1}$$

where $v_i = 1.0$, v_f is the final percent of remained parameters, t_i and t_f are the warmup and cool-down steps.

E Accelerating Inference and Reducing Storage

We attain practical efficiency gain in terms of inference time and disk storage space using different sets of off-the-shelf techniques. Specifically, we use DeepSparse[1], a sparsity-aware inference runtime to accelerate inference of sparse model on CPUs. We also utilize the Pytorch built-in quantization function[2] and Compressed Sparse Row (CSR) format[3] to achieve a much smaller disk space requirement.

[1]https://github.com/neuralmagic/deepsparse

[2]https://pytorch.org/docs/stable/quantization.html

[3]https://github.com/huggingface/block_movement_pruning/blob/master/Saving_PruneBERT.ipynb