Sheared Backpropagation for Fine-tuning Foundation Models

Supplementary Material

7. Proof of Theorem 1

In this section, we prove the main theorem in our paper. We follow Jiang et al. [24] and Liu and Mozafari [30] to make this proof. Consider a single linear layer, according to the definition of S_d-smoothness of $\mathcal{L}$ with respect to θ_d and the update rule $\theta_{t+1,d} = \theta_{t,d} - \alpha_t \tilde{\boldsymbol{g}}'_{t,d}$, we have:

$$\begin{aligned}
&\mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_t) \\
&\leq \sum_{d=1}^{D} \langle \boldsymbol{g}_{t,d}, \theta_{t+1,d} - \theta_{t,d} \rangle + \sum_{d=1}^{D} \frac{S_d}{2} \|\theta_{t+1,d} - \theta_{t,d}\|^2 && (11) \\
&= -\alpha_t \sum_{d=1}^{D} \langle \boldsymbol{g}_{t,d}, \tilde{\boldsymbol{g}}'_{t,d} \rangle + \frac{\alpha_t^2}{2} \sum_{d=1}^{D} S_d \|\tilde{\boldsymbol{g}}'_{t,d}\|^2 && (12) \\
&\leq -\alpha_t \sum_{d=1}^{D} \langle \boldsymbol{g}_{t,d}, \tilde{\boldsymbol{g}}'_{t,d} \rangle + \frac{S_\infty \alpha_t^2}{2} \|\tilde{\boldsymbol{g}}'_t\|^2 && (13) \\
&= -\alpha_t \sum_{d=1}^{D} \langle \boldsymbol{g}_{t,d}, \frac{\tilde{\boldsymbol{g}}_{t,d}}{p_{t,d}} Z_{t,d} \rangle + \frac{S_\infty \alpha_t^2}{2} \| \sum_{d=1}^{D} \frac{\tilde{\boldsymbol{g}}_{t,d}}{p_{t,d}} Z_{t,d} \|^2 && (14) \\
&= -\alpha_t \sum_{d=1}^{D} \frac{Z_{t,d}}{p_{t,d}} \langle \boldsymbol{g}_{t,d}, \tilde{\boldsymbol{g}}_{t,d} \rangle + \frac{S_\infty \alpha_t^2}{2} \| \sum_{d=1}^{D} \frac{\tilde{\boldsymbol{g}}_{t,d}}{p_{t,d}} Z_{t,d} \|^2 && (15) \\
&\leq -\alpha_t \sum_{d=1}^{D} \frac{Z_{t,d}}{p_{t,d}} \langle \boldsymbol{g}_{t,d}, \tilde{\boldsymbol{g}}_{t,d} \rangle + \frac{S_\infty \alpha_t^2}{2} \sum_{d=1}^{D} \frac{\|\tilde{\boldsymbol{g}}_{t,d}\|^2}{p_{t,d}^2} Z_{t,d}^2 && (16)
\end{aligned}$$

Taking expectations on both sides, we have:

$$\begin{aligned}
&\mathbb{E}[\mathcal{L}(\theta_{t+1})] - \mathbb{E}[\mathcal{L}(\theta_t)] \\
&\leq -\alpha_t \sum_{d=1}^{D} \|\boldsymbol{g}_{t,d}\|_2^2 + \frac{S_\infty \alpha_t^2}{2} \sum_{d=1}^{D} \frac{\|\boldsymbol{g}_{t,d}\|_2^2}{p_{t,d}} && (17)
\end{aligned}$$

By summing over T iterations, we have:

$$\begin{aligned}
\mathbb{E}[\frac{1}{T} \sum_{t=1}^{T} \|\boldsymbol{g}_t\|^2] \leq \frac{\mathcal{L}(\theta_0) - L^*}{\alpha T} \\
+ \frac{S_\infty \alpha}{2T} \sum_{t=1}^{T} \mathbb{E}[\sum_{d=1}^{D} \frac{\|\boldsymbol{g}_{t,[d]}\|^2}{p_{t,d}}] && (18)
\end{aligned}$$

So far, we have completed the proof of **Theorem 1**. As the number of training iterations T increase, the two terms on the right-hand side of the equation progressively diminishes to zero, whereas the value of the second term can be further adjusted with the bounds set by the bandit method. For further insight into this aspect, we direct readers to the work of Liu and Mozafari [30], which demonstrates that the average convergence rate of the JOINTSPAR method can attain $O(\frac{1}{\sqrt{T}})$. Distinguishing our work from their approach, which utilizes parameter-wise blocks, we propose an innovative adaptation: transforming these blocks into column-wise patterns for each parameter. This modification, in conjunction with activation pruning, significantly enhances computation and memory efficiency.

In the work of BACKRAZOR [24], a convergence analysis is conducted based on the results of applying top-k pruning to activation matrices. Our research takes this a step further by altering the mask pattern to a column-wise configuration. Based on **Lemma 1**, our modification seamlessly integrates the probabilistic elements of the bandit method across the activation columns into the gradient approximation, thus creating a methodology distinct from that of Jiang et al. [24].

8. Implementation Details

In this discussion, we elaborate on the efficient implementation of our methods, building upon PyTorch [39] and BACKRAZOR [24].

Considering a linear layer as an example, our approach involves maintaining a probability distribution over the activation matrix. This is achieved by using an array, sized according to the number of columns in the input matrix x, to record the probability of Bernoulli sampling over the coordinates. To acquire gradient information, we utilize the `register_backward_hook` API from PyTorch. This function is crucial for harnessing gradients during the backpropagation of loss, when gradients with respect to the module are computed. For a specified update frequency spanning several iterations, the probability distribution is updated based on this gradient information. This process is detailed in Algorithm 2.

In terms of gradient computation, our method involves using a column-wise mask to reconstruct the remaining activation coordinates into a smaller matrix. This approach reduces the workload of computing the full gradient. Subsequently, the smaller matrix is expanded back to its original size. This technique bears resemblance to the one employed in JOINTSPAR [30], which utilizes the `requires_grad_(False)` API to inhibit parameter-wise gradient computation.

Table 6. Results of fine-tuning ViT on Flowers-102.

Dataset	**Method**	**Memory Peak Usage**	**Finish Time**	**Accuracy(%)**
Flowers	FULL-FT	19600MB	17.18min	99.3
	BACKRAZOR	8190MB	34.84min	99.5
	(ours) PREBR++	7800MB	22.65min	99.4

Table 7. Results of fine-tuning ViT on CUB-200.

Dataset	**Method**	**Memory Peak Usage**	**Epoch Time**	**Accuracy(%)**
CUB-200	FULL-FT	19600 MB	22s	85.5
	BACKRAZOR	8210MB	47s	86.6
	(ours) PREBR++	8100MB	30s	86.4

Regarding our proposed PREBACKRAZOR-, the principal alteration consists of transforming the element-wise mask used in BACKRAZOR into a column-wise pattern. This revamped mask is generated by considering the top-k norm values from each column of the activation matrix. It's important to note that while this represents a simplified implementation, it nonetheless yields a moderate improvement in computation efficiency, particularly noticeable when the pruning ratio is not aggressive (80%).

As outlined in Sec. 3, integrating dense FLASHATTENTION [16, 17] kernels significantly enhances computation but also increases memory usage. To bring the memory consumption below that of BACKRAZOR, we need to introduce sparsity into the process. During our experiments with asymmetric backpropagation, where we use sparsified QKV matrices solely for computing backward gradients, we encountered `nan` values, signaling computational instability. To address this issue, we adopted a recomputation strategy, which involves performing an additional sparse attention forward pass using the sparsified matrices during the backward pass, prior to computing gradients. We then utilize the outcomes of this pass for the backward process. By integrating sparse and dense kernels, we maintain memory usage at the same level as that of BACKRAZOR. We have empirically validated the effectiveness of this method through our experiments.

9. More Experiment Results

We present the results of fine-tuning the ViT-B/16 model on the Flowers-102 [37] and CUB-200 [52] datasets, with a pruning ratio of 80%. Specifically for the Flowers-102 dataset, the model is fine-tuned over 2,000 steps with a maximum learning rate of 0.03. All other experimental settings are consistent with those described in Sec. 5.

As demonstrated in Tabs. 6 and 7, our proposed PREBACKRAZOR not only attains an accuracy comparable to that of BACKRAZOR but also delivers a notable speed improvement while maintaining similar memory efficiency. The results on CUB-200 also indicate that at a less aggressive pruning ratio of 80%, our method shows a significant improvement compared to the results presented in Tab. 1, closely approaching the accuracy of BackRazor.