Memory-Efficient LLM Training with Online Subspace Descent

Kaizhao Liang, Bo Liu, Lizhang Chen, Qiang Liu

Online Subspace Descent

TLDR:

- **AdamW is good, but (memory) expensive**
- **A general online subspace framework for memory efficient optimization**
- **Subspaces can be updated arbitrarily – via hamiltonian view**

Algorithm 1 Online Subspace Descent

Required: Optimizer `OptimizerW`, learning rate ϵ_t^W, weight decay λ^W for model weights $\boldsymbol{W}_t$; and $\{$`OptimizerP`$, \epsilon_t^P, \lambda^P\}$ for the projection matrix $\boldsymbol{P}_t$. Proper initialization.

for iteration t **do**

Calculate gradient $\boldsymbol{G}_t = \nabla L(\boldsymbol{W}_t)$; Update model weights $\boldsymbol{W}_t$ by

$$(\hat{\boldsymbol{\Delta}}_t,\ \hat{\boldsymbol{S}}_t) = \texttt{OptimizerW}(\boldsymbol{P}_t^\top \boldsymbol{G}_t, \hat{\boldsymbol{S}}_{t-1}), \quad \boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \epsilon_t^W(\boldsymbol{P}_t\hat{\boldsymbol{\Delta}}_{t+1} - \lambda^W \boldsymbol{W}_t)$$

Calculate $\boldsymbol{G}_t^P = \nabla L_{\boldsymbol{G}_t}(\boldsymbol{P}_t)$ for $L_{\boldsymbol{G}_t}(\cdot)$ in Eq (6); Update the projection $\boldsymbol{P}_t$ by

$$(\boldsymbol{\Delta}_t^P, \boldsymbol{S}_t^P) = \texttt{OptimizerP}(\boldsymbol{G}_t^P,\ \boldsymbol{S}_{t-1}^P), \qquad \boldsymbol{P}_{t+1} = \boldsymbol{P}_t + \epsilon_t^P(\boldsymbol{\Delta}_t^P - \lambda^P \boldsymbol{P}_t)$$

end for

Remark: We added weight decay as a common heuristic. We recommend using Adam for both optimizers, and set $\epsilon_t^P = \alpha\epsilon_t^W$ with a constant α (e.g., $\alpha = 5$), and $\lambda^W = \lambda^P$. [2]

Common Optimizers

$$\begin{aligned}
\textit{Gradient Descent}: \quad & \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \epsilon_t \boldsymbol{P}_t \boldsymbol{P}_t^\top \boldsymbol{G}_t, && \boldsymbol{G}_t = \nabla L(\boldsymbol{W}_t),\\
\textit{Momentum}: \quad & \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \epsilon_t \boldsymbol{P}_t \hat{\boldsymbol{M}}_t, && \hat{\boldsymbol{M}}_t = (1-\beta)\boldsymbol{P}_t^\top \boldsymbol{G}_t + \beta \hat{\boldsymbol{M}}_{t-1},\\
\textit{Lion-}\mathcal{K}: \quad & \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \epsilon_t \boldsymbol{P}_t \nabla\mathcal{K}(\hat{\boldsymbol{N}}_t), && \hat{\boldsymbol{G}}_t = \boldsymbol{P}_t^\top \boldsymbol{G}_t\\
& \hat{\boldsymbol{N}}_t = (1-\beta_1)\hat{\boldsymbol{G}}_t + \beta_1 \hat{\boldsymbol{M}}_t, && \hat{\boldsymbol{M}}_t = (1-\beta_2)\hat{\boldsymbol{G}}_t + \beta_2 \hat{\boldsymbol{M}}_{t-1},\\
\textit{Adam}: \quad & \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \epsilon_t \boldsymbol{P}_t \frac{\hat{\boldsymbol{M}}_t}{\sqrt{\hat{\boldsymbol{V}}_t} + e}, && \hat{\boldsymbol{G}}_t = \boldsymbol{P}_t^\top \boldsymbol{G}_t,\\
& \hat{\boldsymbol{M}}_t = (1-\beta_{1t})\hat{\boldsymbol{G}}_t + \beta_{1t}\hat{\boldsymbol{M}}_{t-1}, && \hat{\boldsymbol{V}}_t = (1-\beta_{2t})\hat{\boldsymbol{G}}_t^{\odot 2} + \beta_{2t}\hat{\boldsymbol{V}}_{t-1}.
\end{aligned}$$

A Natural Update Rule

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \epsilon_t \boldsymbol{P}_t \boldsymbol{P}_t^\top \boldsymbol{G}_t, \qquad \boldsymbol{G}_t = \nabla L(\boldsymbol{W}_t)$$

$$\boldsymbol{P}_{t+1} = \texttt{OptimizerP.step}(\boldsymbol{P}_t,\ \nabla_{\boldsymbol{P}} L_{\boldsymbol{G}_t}(\boldsymbol{P}_t))$$

Hamiltonian + Descent

W/O Projection

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{W}_t = \partial_{\boldsymbol{S}} H(\boldsymbol{W}_t, \boldsymbol{S}_t) - \Phi(\partial_{\boldsymbol{W}} H(\boldsymbol{W}_t, \boldsymbol{S}_t))$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{S}_t = -\partial_{\boldsymbol{W}} H(\boldsymbol{W}_t, \boldsymbol{S}_t) - \Psi(\partial_{\boldsymbol{S}} H(\boldsymbol{W}_t, \boldsymbol{S}_t)).$$

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t} H(\boldsymbol{W}_t, \boldsymbol{S}_t) &= \left\langle \partial_{\boldsymbol{W}} H_t, \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{W}_t \right\rangle + \left\langle \partial_{\boldsymbol{S}} H_t, \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{S}_t \right\rangle\\
&= \langle \partial_{\boldsymbol{W}} H_t, \partial_{\boldsymbol{S}} H_t - \Phi(\partial_{\boldsymbol{W}} H_t)\rangle + \langle \partial_{\boldsymbol{S}} H_t, -\partial_{\boldsymbol{W}} H_t - \Psi(\partial_{\boldsymbol{S}} H_t)\rangle\\
&= -\|\partial_{\boldsymbol{W}} H_t\|_\Phi^2 - \|\partial_{\boldsymbol{S}} H_t\|_\Psi^2 \le 0,
\end{aligned}$$

With Projection

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{W}_t = \boldsymbol{P}_t \partial_{\hat{\boldsymbol{S}}} H(\boldsymbol{W}_t, \hat{\boldsymbol{S}}_t) - \Phi(\partial_{\boldsymbol{W}} H(\boldsymbol{W}_t, \hat{\boldsymbol{S}}_t))$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{\boldsymbol{S}}_t = -\boldsymbol{P}_t^\top \partial_{\boldsymbol{W}} H(\boldsymbol{W}_t, \hat{\boldsymbol{S}}_t) - \Psi(\partial_{\hat{\boldsymbol{S}}} H(\boldsymbol{W}_t, \hat{\boldsymbol{S}}_t))$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{P}_t = \Gamma(\boldsymbol{P}_t, \nabla L(\boldsymbol{W}_t)),$$

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t} H(\boldsymbol{W}_t, \hat{\boldsymbol{S}}_t) &= -\|\partial_{\boldsymbol{W}} H_t\|_\Phi^2 - \|\partial_{\boldsymbol{S}} H_t\|_\Psi^2 + \langle \partial_{\boldsymbol{W}} H_t, \boldsymbol{P}_t \partial_{\hat{\boldsymbol{S}}} H_t\rangle - \langle \partial_{\hat{\boldsymbol{S}}} H_t, \boldsymbol{P}_t^\top \partial_{\boldsymbol{W}} H_t\rangle\\
&= -\|\partial_{\boldsymbol{W}} H_t\|_\Phi^2 - \|\partial_{\boldsymbol{S}} H_t\|_\Psi^2 \le 0,
\end{aligned}$$

Experiments

TLDR: works better than Galore

Method	Perplexity(↓)		
	60M	350M	1B
8bit-AdamW (Full Rank)	32.75	30.43	29.40
GaLore (Rank = 512)	57.03	44.34	35.52
Ours (Rank = 512)	**56.12**	**43.67**	**31.30**

Pretraining LLaMA 1B SS 256, 10K steps, AdamW8bit

- **7B LLaMA** model
- SS 256
- C4 dataset for 10K steps
- Perplexity Lower the better

Method	Perplexity	Wall Clock Time (hours)
Galore	51.21	9.7439
Ours	**43.72**	**7.1428**

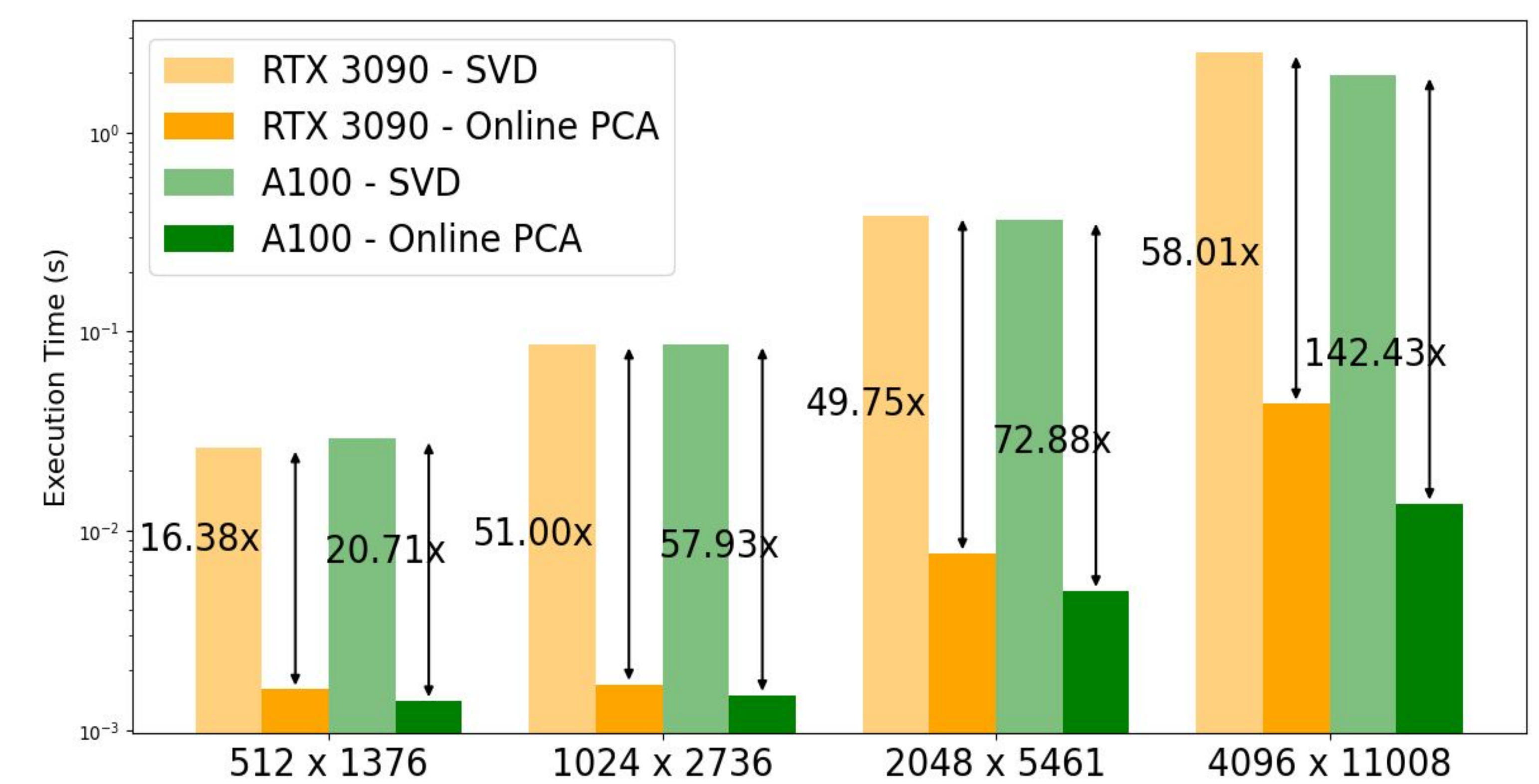
Table 1: Perplexity and Wall Clock Time for 7B

Method	MRPC	RTE	SST2	MNLI	QNLI	QQP	AVG
Galore	0.6838	**0.5018**	0.5183	0.3506	0.4946	0.3682	0.4862
Ours	**0.6982**	0.4901	**0.5233**	**0.3654**	**0.5142**	**0.3795**	**0.4951**

Table 2: GLUE on 7B

System Advantage

Why is it Faster?



- torch.svd is slow
- single-step backward() is fast
- P updates can be executed in parallel, no overhead
- Cost of SVD can't be masked out

References

Chen, Lizhang, et al. "Lion secretly solves constrained optimization: As lyapunov predicts." *arXiv preprint arXiv:2310.05898* (2023).

Loshchilov, I. "Decoupled weight decay regularization." *arXiv preprint arXiv:1711.05101* (2017).

Chen, Xiangning, et al. "Symbolic discovery of optimization algorithms." *Advances in neural information processing systems* 36 (2024).

Anil, Rohan, et al. "Memory efficient adaptive optimization." *Advances in Neural Information Processing Systems* 32 (2019).

Zhao, Jiawei, et al. "Galore: Memory-efficient llm training by gradient low-rank projection." *arXiv preprint arXiv:2403.03507* (2024).