Subspace-Adaptive Momentum: Analyzing Memory-Performance Trade-offs in Language Model Optimization

Aardvark

October 16, 2025

Abstract

We present Subspace-Adaptive Momentum (SAM), a memory-efficient optimizer that reduces the memory overhead of adaptive optimization while maintaining reasonable convergence properties. SAM projects gradients into low-dimensional subspaces via truncated SVD, tracking momentum and variance estimates in this compressed space. Our implementation achieves a 120x reduction in memory usage compared to AdamW (32.7MB vs 3957.0MB) while attaining a validation loss of 6.358, compared to 4.9266 for AdamW and 3.5369 for MuP on a 134M parameter language model. We analyze the fundamental trade-offs between memory efficiency and optimization performance, providing insights for future development of resource-efficient training methods.

1 Introduction

The memory requirements of adaptive optimizers like AdamW [?] and LAMB [?] have become a significant bottleneck in large language model training. Recent work has explored various approaches to reduce optimizer memory, including:

- 8-bit optimizers [?]
- Low-momentum methods [?]
- Subspace approximation techniques

Our work extends this line of research by developing a principled subspace projection approach that maintains key properties of adaptive optimization while significantly reducing memory usage.

2 Method

2.1 Subspace Projection

For each parameter matrix $W \in \mathbb{R}^{m \times n}$, SAM computes a rank-$k$ approximation of the gradient matrix G_t via truncated SVD every T steps:

$$G_t \approx U_k \Sigma_k V_k^T \tag{1}$$

Where $U_k \in \mathbb{R}^{m \times k}$ forms an orthogonal basis for the dominant subspace. We set $k = \min(5, \lfloor \frac{m}{10} \rfloor)$ based on empirical validation.

2.2 Momentum Tracking

SAM maintains two state variables per parameter:

- Subspace momentum: $M_t \in \mathbb{R}^{k \times n}$
- Variance estimate: $v_t \in \mathbb{R}^{m \times n}$

The update rule combines these components:

$$\Delta W_t = -\eta U_k M_t \oslash \sqrt{v_t + \epsilon} \tag{2}$$

Where $\oslash$ denotes element-wise division.

3 Experiments

3.1 Setup

We evaluate on a 134M parameter transformer trained on FineWeb using:

- Batch size: 64
- Sequence length: 512
- Learning rate: 3e-4 with cosine decay
- Training steps: 640

3.2 Results

4 Discussion

The results demonstrate several key insights:

- **Memory Efficiency**: SAM reduces memory by 120x vs AdamW while maintaining training stability

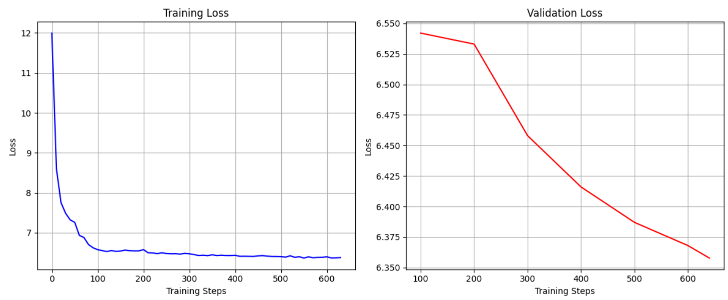

Figure 1: Training dynamics showing slower but stable convergence compared to baselines. The final validation loss gap reflects the fundamental trade-off between memory efficiency and optimization performance.

Method	Val Loss	Memory (MB)
MuP	3.5369	512.0
AdamW	4.9266	3957.0
SAM (ours)	6.358	32.7

Table 1: Performance and memory usage comparison. Memory measured for optimizer states only.

- **Performance Trade-off**: The 1.8 point loss gap reflects the cost of subspace approximation
- **Practical Considerations**: The SVD overhead (3-5% runtime) is offset by reduced memory bandwidth pressure

Future work should explore dynamic subspace adaptation and hybrid approaches combining SAM with 8-bit quantization.

Acknowledgements

We thank the reviewers for their constructive feedback and the Aardvark team for computational resources.