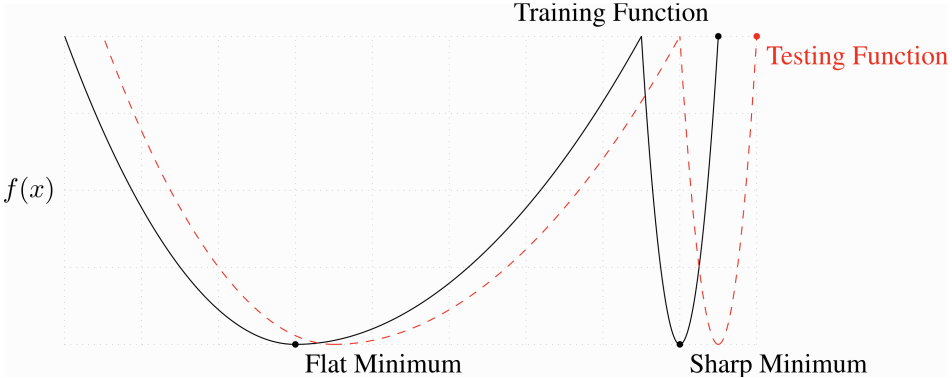
Sharpness-Aware Minimization. Mode Connectivity. Grokking. Double Descent.

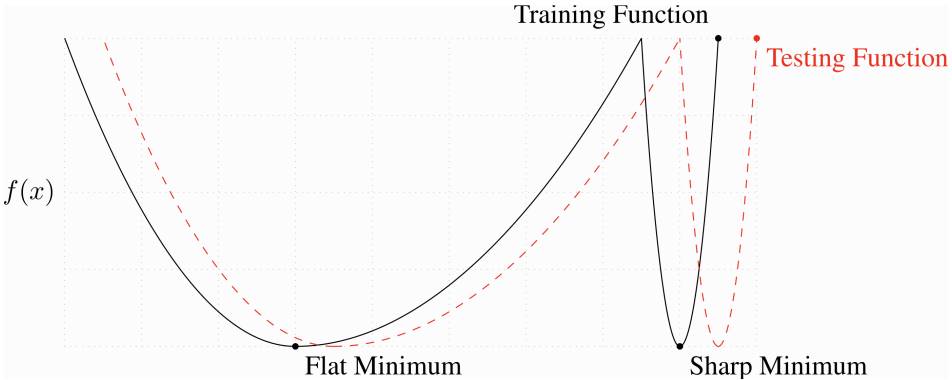
Seminar

Optimization for ML. Faculty of Computer Science. HSE University

Flat Minimum vs Sharp Minimum



Flat Minimum vs Sharp Minimum



Question

What's wrong with Sharp Minimum?

Sharpness-Aware Minimization [1]

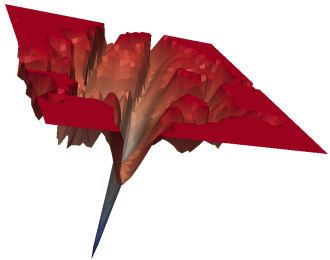


Figure 1: A sharp minimum to which a ResNet trained with SGD converged.

[1] Foret, Pierre, et al. "Sharpness-aware minimization for efficiently improving generalization." (2020)

Sharpness-Aware Minimization [1]

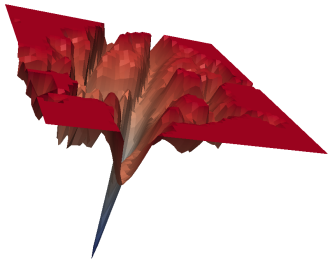


Figure 1: A sharp minimum to which a ResNet trained with SGD converged.

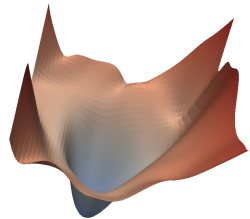


Figure 2: A wide minimum to which the same ResNet trained with SAM converged.

[1] Foret, Pierre, et al. "Sharpness-aware minimization for efficiently improving generalization." (2020)

Sharpness-Aware Minimization [1]

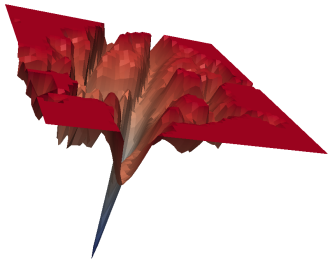


Figure 1: A sharp minimum to which a ResNet trained with SGD converged.

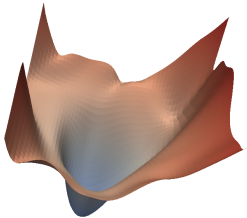


Figure 2: A wide minimum to which the same ResNet trained with SAM converged.

Sharpness-Aware Minimization (SAM) is a procedure that aims to improve model generalization by simultaneously minimizing loss value and **loss sharpness**.

[1] Foret, Pierre, et al. "Sharpness-aware minimization for efficiently improving generalization." (2020)

Learning setup

The training dataset drawn $i.i.d.$ from a distribution D:

$$S = \{(x_i, y_i)\}_{i=1}^{n},$$

where x_i – feature vector and y_i – label.

Learning setup

The training dataset drawn $i.i.d.$ from a distribution D:

$$S = \{(x_i, y_i)\}_{i=1}^{n},$$

where x_i – feature vector and y_i – label.

The training set loss:

$$L_S = \frac{1}{n} \sum_{i=1}^{n} l(\boldsymbol{w}, x_i, y_i),$$

where l – per-data-point loss function, $\boldsymbol{w}$ – parameters.

Learning setup

The training dataset drawn $i.i.d.$ from a distribution D:

$$S = \{(x_i, y_i)\}_{i=1}^{n},$$

where x_i – feature vector and y_i – label.

The training set loss:

$$L_S = \frac{1}{n} \sum_{i=1}^{n} l(\boldsymbol{w}, x_i, y_i),$$

where l – per-data-point loss function, $\boldsymbol{w}$ – parameters.

The population loss:

$$L_D = \mathbb{E}_{(x,y)}[l(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y})]$$

What is sharpness?

Theorem

For any $\rho > 0$, with high probability over training set S generated from distribution D,

$$L_D(\boldsymbol{w}) \leq \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}) + h\left(\|\boldsymbol{w}\|_2^2 / \rho^2\right),$$

where $h : \mathbb{R}_+ \to \mathbb{R}_+$is a strictly increasing function (under some technical conditions on $L_D(\boldsymbol{w})$).

What is sharpness?

Theorem

For any $\rho > 0$, with high probability over training set S generated from distribution D,

$$L_D(\boldsymbol{w}) \leq \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}) + h\left(\|\boldsymbol{w}\|_2^2 / \rho^2\right),$$

where $h : \mathbb{R}_+ \to \mathbb{R}_+$is a strictly increasing function (under some technical conditions on $L_D(\boldsymbol{w})$).

Adding and subtracting $L_S(\boldsymbol{w})$:

$$\left[\max_{\|\in\|_2 \leq \rho} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}) - L_S(\boldsymbol{w})\right] + L_S(\boldsymbol{w}) + h\left(\|\boldsymbol{w}\|_2^2 / \rho^2\right)$$

The term in square brackets captures the **sharpness** of L_S at $\boldsymbol{w}$ by measuring how quickly the training loss can be increased by moving from $\boldsymbol{w}$ to a nearby parameter value.

Sharpness-Aware Minimization

The function h is removed in favor of a simpler constant λ. The authors propose selecting parameter values by solving the following Sharpness-Aware Minimization (SAM) problem:

$$\min_{\boldsymbol{w}} L_S^{SAM}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2 \quad \text{where} \quad L_S^{SAM}(\boldsymbol{w}) \triangleq \max_{\|\epsilon\|_p \leq \rho} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}),$$

with $\rho \geq 0$ as hyperparameter and p in $[1, \infty]$ (a little generalization, though $p = 2$ is empirically the best choice).

How to minimize L_S^{SAM}?

In order to minimize L_S^{SAM} an efficient approximation of its gradient is used. A first step is to consider the first-order Taylor expansion of $L_S(\boldsymbol{w} + \boldsymbol{\epsilon})$:

$$\boldsymbol{\epsilon}^*(\boldsymbol{w}) \triangleq \underset{\|\epsilon\|_p \leq \rho}{\arg\max} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}) \approx \underset{\|\epsilon\|_p \leq \rho}{\arg\max} L_S(\boldsymbol{w}) + \boldsymbol{\epsilon}^T \nabla_{\boldsymbol{w}} L_S(\boldsymbol{w}) = \underset{\|\epsilon\|_p \leq \rho}{\arg\max} \boldsymbol{\epsilon}^T \nabla_{\boldsymbol{w}} L_S(\boldsymbol{w}).$$

How to minimize L_S^{SAM}?

In order to minimize L_S^{SAM} an efficient approximation of its gradient is used. A first step is to consider the first-order Taylor expansion of $L_S(\boldsymbol{w}+\boldsymbol{\epsilon})$:

$$\boldsymbol{\epsilon}^*(\boldsymbol{w}) \triangleq \underset{\|\epsilon\|_p \leq \rho}{\arg\max} L_S(\boldsymbol{w}+\boldsymbol{\epsilon}) \approx \underset{\|\epsilon\|_p \leq \rho}{\arg\max} L_S(\boldsymbol{w}) + \boldsymbol{\epsilon}^T \nabla_{\boldsymbol{w}} L_S(\boldsymbol{w}) = \underset{\|\epsilon\|_p \leq \rho}{\arg\max} \boldsymbol{\epsilon}^T \nabla_{\boldsymbol{w}} L_S(\boldsymbol{w}).$$

The last expression is just the argmax of the dot product of the vectors $\boldsymbol{\epsilon}$ and $\nabla_{\boldsymbol{w}} L_S(\boldsymbol{w})$, and it is well known which is the argument that maximizes it:

$$\hat{\boldsymbol{\epsilon}}(\boldsymbol{w}) = \rho \operatorname{sign}\left(\nabla_{\boldsymbol{w}} L_S(\boldsymbol{w})\right) \left|\nabla_{\boldsymbol{w}} L_S(\boldsymbol{w})\right|^{q-1} / \left(\|\nabla_{\boldsymbol{w}} L_S(\boldsymbol{w})\|_q^q\right)^{1/p},$$

where $1/p + 1/q = 1$.

How to minimize L_S^{SAM}?

In order to minimize L_S^{SAM} an efficient approximation of its gradient is used. A first step is to consider the first-order Taylor expansion of $L_S(\boldsymbol{w}+\boldsymbol{\epsilon})$:

$$\boldsymbol{\epsilon}^*(\boldsymbol{w}) \triangleq \underset{\|\epsilon\|_p \leq \rho}{\arg\max} L_S(\boldsymbol{w}+\boldsymbol{\epsilon}) \approx \underset{\|\epsilon\|_p \leq \rho}{\arg\max} L_S(\boldsymbol{w}) + \boldsymbol{\epsilon}^T \nabla_{\boldsymbol{w}} L_S(\boldsymbol{w}) = \underset{\|\epsilon\|_p \leq \rho}{\arg\max} \boldsymbol{\epsilon}^T \nabla_{\boldsymbol{w}} L_S(\boldsymbol{w}).$$

The last expression is just the argmax of the dot product of the vectors $\boldsymbol{\epsilon}$ and $\nabla_{\boldsymbol{w}} L_S(\boldsymbol{w})$, and it is well known which is the argument that maximizes it:

$$\hat{\boldsymbol{\epsilon}}(\boldsymbol{w}) = \rho \operatorname{sign}\left(\nabla_{\boldsymbol{w}} L_S(\boldsymbol{w})\right) \left|\nabla_{\boldsymbol{w}} L_S(\boldsymbol{w})\right|^{q-1} / \left(\|\nabla_{\boldsymbol{w}} L_S(\boldsymbol{w})\|_q^q\right)^{1/p},$$

where $1/p + 1/q = 1$.

Thus

$$\nabla_{\boldsymbol{w}} L_S^{SAM}(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} L_S(\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})) = \left.\frac{d(\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w}))}{d\boldsymbol{w}} \nabla_{\boldsymbol{w}} L_S(\boldsymbol{w})\right|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(w)}$$

$$= \nabla_w L_S(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})} + \left.\frac{d\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}{d\boldsymbol{w}} \nabla_{\boldsymbol{w}} L_S(\boldsymbol{w})\right|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}$$

Sharpness-Aware Minimization

Modern frameworks can easily compute the preceding approximation. However, to speed up the computation, second-order terms can be dropped obtaining:

$$\nabla_{\boldsymbol{w}} L_S^{SAM}(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} L_S(w)\Big|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}$$

Sharpness-Aware Minimization

Modern frameworks can easily compute the preceding approximation. However, to speed up the computation, second-order terms can be dropped obtaining:

$$\nabla_{\boldsymbol{w}} L_S^{SAM}(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} L_S(w)\big|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}$$

Input: Training set $\mathcal{S} \triangleq \cup_{i=1}^{n}\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$, Loss function $l : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$, Batch size b, Step size $\eta > 0$, Neighborhood size $\rho > 0$.
Output: Model trained with SAM
Initialize weights $\boldsymbol{w}_0$, $t = 0$;
while *not converged* **do**
  Sample batch $\mathcal{B} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), ...(\boldsymbol{x}_b, \boldsymbol{y}_b)\}$;
  Compute gradient $\nabla_{\boldsymbol{w}} L_{\mathcal{B}}(\boldsymbol{w})$ of the batch's training loss;
  Compute $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})$ per equation 2;
  Compute gradient approximation for the SAM objective (equation 3): $\boldsymbol{g} = \nabla_w L_{\mathcal{B}}(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}$;
  Update weights: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{g}$;
  $t = t + 1$;
end
return $\boldsymbol{w}_t$

Algorithm 1: SAM algorithm

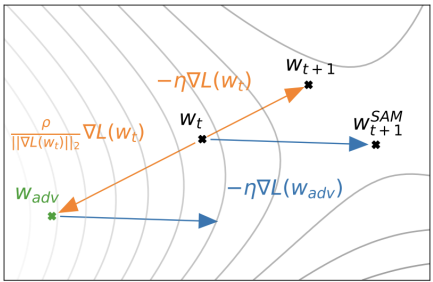


Figure 2: Schematic of the SAM parameter update.

SAM results

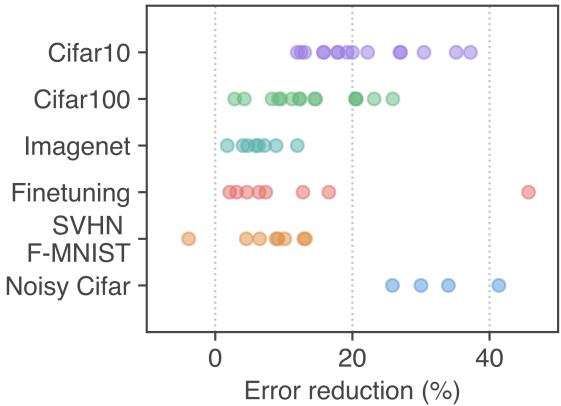


Figure 4: Error rate reduction obtained by switching to SAM. Each point is a different dataset / model / data augmentation.

Why VR methods do not work in training neural networks? [2]

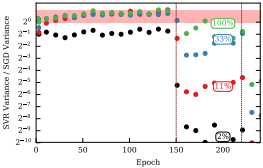


Figure 5: DenseNet

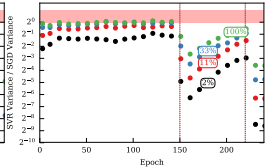


Figure 6: Small ResNet

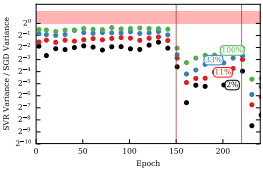


Figure 7: LeNet-5

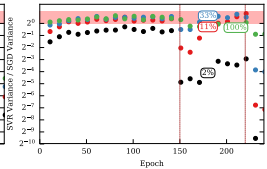


Figure 8: ResNet-110

- **SVRG / SAG** provide convincing gains in convex problems, but on CIFAR-10 (LeNet-5) and ImageNet (ResNet-18), they do not outperform standard SGD.

[2] Defazio, A., Bottou, L. (2019). On the ineffectiveness of variance reduced optimization for deep learning. Advances in Neural Information Processing Systems, 32.

Why VR methods do not work in training neural networks? [2]

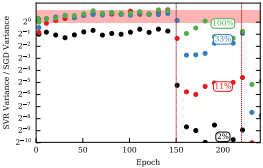


Figure 5: DenseNet

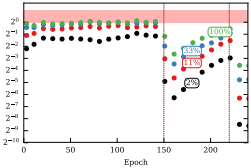


Figure 6: Small ResNet

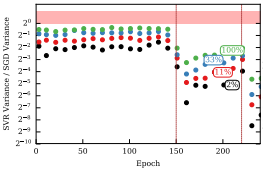


Figure 7: LeNet-5

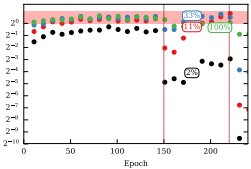


Figure 8: ResNet-110

- **SVRG / SAG** provide convincing gains in convex problems, but on CIFAR-10 (LeNet-5) and ImageNet (ResNet-18), they do not outperform standard SGD.
- The measured ratio "variance of SGD / variance of SVRG" remains $\lesssim 2$ for most layers - meaning the actual noise reduction is minimal.

[2] Defazio, A., Bottou, L. (2019). On the ineffectiveness of variance reduced optimization for deep learning. Advances in Neural Information Processing Systems, 32.

Why VR methods do not work in training neural networks? [2]

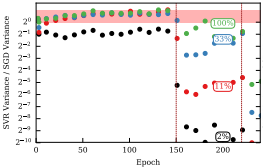


Figure 5: DenseNet

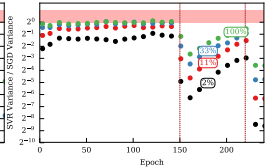


Figure 6: Small ResNet

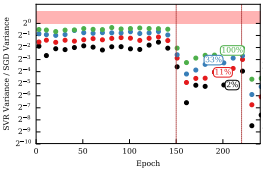


Figure 7: LeNet-5

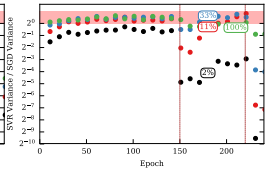


Figure 8: ResNet-110

- **SVRG / SAG** provide convincing gains in convex problems, but on CIFAR-10 (LeNet-5) and ImageNet (ResNet-18), they do not outperform standard SGD.
- The measured ratio "variance of SGD / variance of SVRG" remains $\lesssim 2$ for most layers - meaning the actual noise reduction is minimal.
- Possible reasons:
  - **Data augmentation** makes the reference gradient g_{ref} outdated after just a few minibatches.

[2] Defazio, A., Bottou, L. (2019). On the ineffectiveness of variance reduced optimization for deep learning. Advances in Neural Information Processing Systems, 32.

Why VR methods do not work in training neural networks? [2]

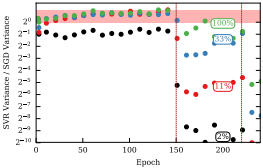


Figure 5: DenseNet

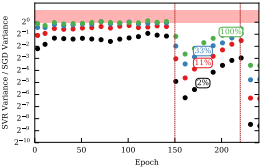


Figure 6: Small ResNet

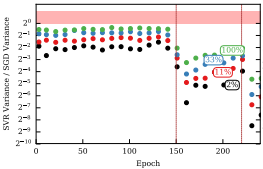


Figure 7: LeNet-5

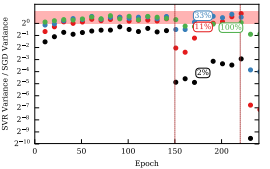


Figure 8: ResNet-110

- **SVRG / SAG** provide convincing gains in convex problems, but on CIFAR-10 (LeNet-5) and ImageNet (ResNet-18), they do not outperform standard SGD.
- The measured ratio "variance of SGD / variance of SVRG" remains $\lesssim 2$ for most layers - meaning the actual noise reduction is minimal.
- Possible reasons:
  - **Data augmentation** makes the reference gradient g_{ref} outdated after just a few minibatches.
  - **BatchNorm** and **Dropout** introduce internal stochasticity that cannot be compensated by the past g_{ref}.

[2] Defazio, A., Bottou, L. (2019). On the ineffectiveness of variance reduced optimization for deep learning. Advances in Neural Information Processing Systems, 32.

Why VR methods do not work in training neural networks? [2]

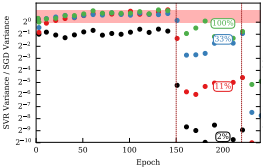


Figure 5: DenseNet

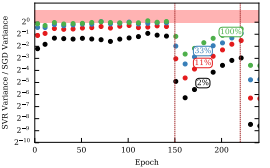


Figure 6: Small ResNet

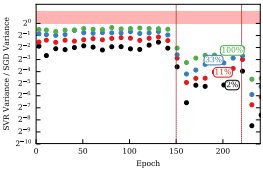


Figure 7: LeNet-5

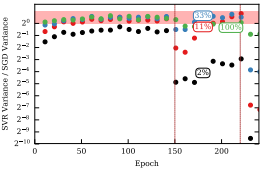


Figure 8: ResNet-110

- **SVRG / SAG** provide convincing gains in convex problems, but on CIFAR-10 (LeNet-5) and ImageNet (ResNet-18), they do not outperform standard SGD.
- The measured ratio "variance of SGD / variance of SVRG" remains $\lesssim 2$ for most layers - meaning the actual noise reduction is minimal.
- Possible reasons:
  - **Data augmentation** makes the reference gradient g_{ref} outdated after just a few minibatches.
  - **BatchNorm** and **Dropout** introduce internal stochasticity that cannot be compensated by the past g_{ref}.
  - An additional full pass through the dataset (to compute g_{ref}) consumes the potential iteration savings.

[2] Defazio, A., Bottou, L. (2019). On the ineffectiveness of variance reduced optimization for deep learning. Advances in Neural Information Processing Systems, 32.

Why VR methods do not work in training neural networks? [2]

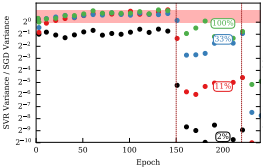


Figure 5: DenseNet

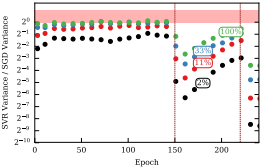


Figure 6: Small ResNet

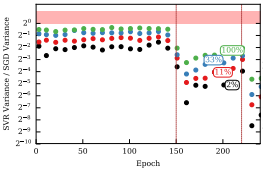


Figure 7: LeNet-5

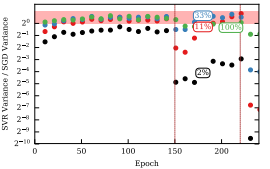


Figure 8: ResNet-110

- **SVRG / SAG** provide convincing gains in convex problems, but on CIFAR-10 (LeNet-5) and ImageNet (ResNet-18), they do not outperform standard SGD.
- The measured ratio "variance of SGD / variance of SVRG" remains $\lesssim 2$ for most layers - meaning the actual noise reduction is minimal.
- Possible reasons:
  - **Data augmentation** makes the reference gradient g_{ref} outdated after just a few minibatches.
  - **BatchNorm** and **Dropout** introduce internal stochasticity that cannot be compensated by the past g_{ref}.
  - An additional full pass through the dataset (to compute g_{ref}) consumes the potential iteration savings.
- "Streaming" modifications of SVRG, designed to handle augmentation, reduce theoretical bias but still lose to SGD in both time and quality.

[2] Defazio, A., Bottou, L. (2019). On the ineffectiveness of variance reduced optimization for deep learning. Advances in Neural Information Processing Systems, 32.

Why VR methods do not work in training neural networks? [2]

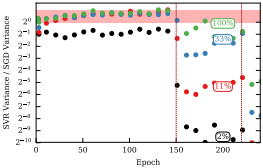


Figure 5: DenseNet

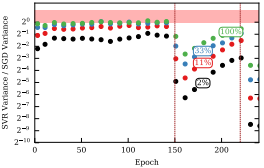


Figure 6: Small ResNet

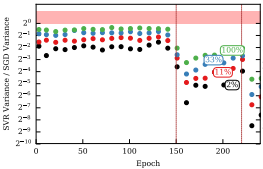


Figure 7: LeNet-5

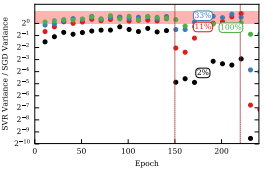


Figure 8: ResNet-110

- **SVRG / SAG** provide convincing gains in convex problems, but on CIFAR-10 (LeNet-5) and ImageNet (ResNet-18), they do not outperform standard SGD.
- The measured ratio "variance of SGD / variance of SVRG" remains $\lesssim 2$ for most layers - meaning the actual noise reduction is minimal.
- Possible reasons:
  - **Data augmentation** makes the reference gradient g_{ref} outdated after just a few minibatches.
  - **BatchNorm** and **Dropout** introduce internal stochasticity that cannot be compensated by the past g_{ref}.
  - An additional full pass through the dataset (to compute g_{ref}) consumes the potential iteration savings.
- "Streaming" modifications of SVRG, designed to handle augmentation, reduce theoretical bias but still lose to SGD in both time and quality.
- **Conclusion**: Existing variance reduction methods are impractical for modern deep networks; future solutions should take into account the stochastic nature of the architecture and data (augmentation, BatchNorm, Dropout).

[2] Defazio, A., Bottou, L. (2019). On the ineffectiveness of variance reduced optimization for deep learning. Advances in Neural Information Processing Systems, 32.

Mode Connectivity [3]

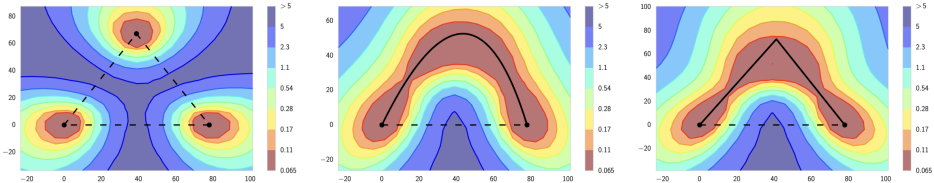


Figure 9: The l_2-regularized cross-entropy train loss surface of a ResNet-164 on CIFAR-100, as a function of network weights in a two-dimensional subspace. In each panel, the horizontal axis is fixed and is attached to the optima of two independently trained networks. The vertical axis changes between panels as we change planes (defined in the main text). Left: Three optima for independently trained networks. Middle and Right: A quadratic Bezier curve, and a polygonal chain with one bend, connecting the lower two optima on the left panel along a path of near-constant loss. Notice that in each panel a direct linear path between each mode would incur high loss.

[3] Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P., Wilson, A. G. (2018). Loss surfaces, mode connectivity, and fast ensembling of dnns. Advances in neural information processing systems, 31.

Curve-Finding Procedure

- Weights of pretrained networks:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

- Define parametric curve: $\phi_\theta(\cdot) : [0,1] \to \mathbb{R}^{|\text{net}|}$

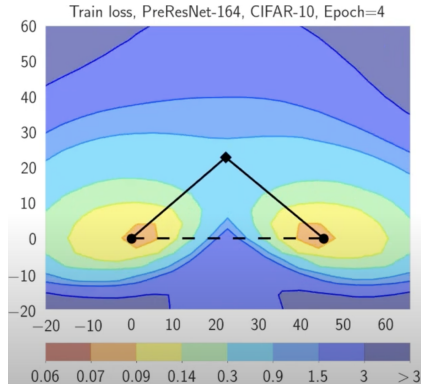
$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

- DNN loss function:

$$\mathcal{L}(w)$$

- Minimize averaged loss w.r.t. θ:

$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t\sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$



Curve-Finding Procedure

- Weights of pretrained networks:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

- Define parametric curve: $\phi_\theta(\cdot) : [0, 1] \to \mathbb{R}^{|\text{net}|}$

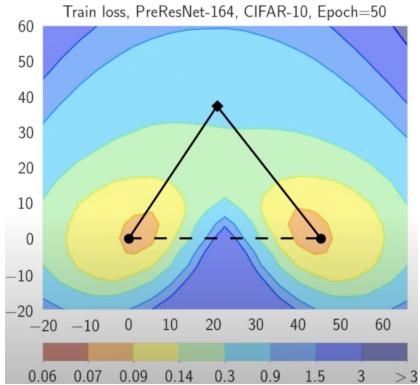
$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

- DNN loss function:

$$\mathcal{L}(w)$$

- Minimize averaged loss w.r.t. θ:

$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t\sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$



Curve-Finding Procedure

- Weights of pretrained networks:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

- Define parametric curve: $\phi_\theta(\cdot) : [0,1] \to \mathbb{R}^{|\text{net}|}$

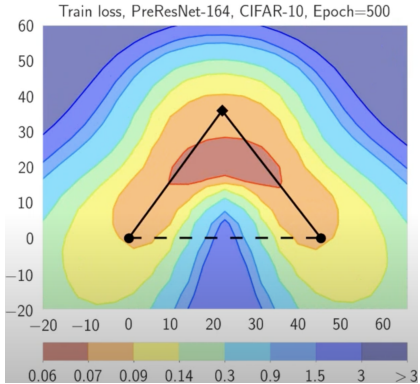
$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

- DNN loss function:

$$\mathcal{L}(w)$$

- Minimize averaged loss w.r.t. θ:

$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t\sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$



Grokking[4]

- After achieving zero train loss the weights continue evolving in a kind of random walk manner
- It is possible that they slowly drift to a wider minima
- Recently discovered grokking effect confirms this hypo

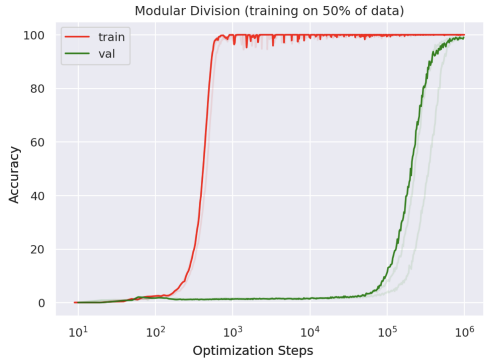


Figure 10: Grokking: A dramatic example of generalization far after overfitting on an algorithmic dataset.

[4] Power, Alethea, et al. "Grokking: Generalization beyond overfitting on small algorithmic datasets." (2022).

Double Descent[5]

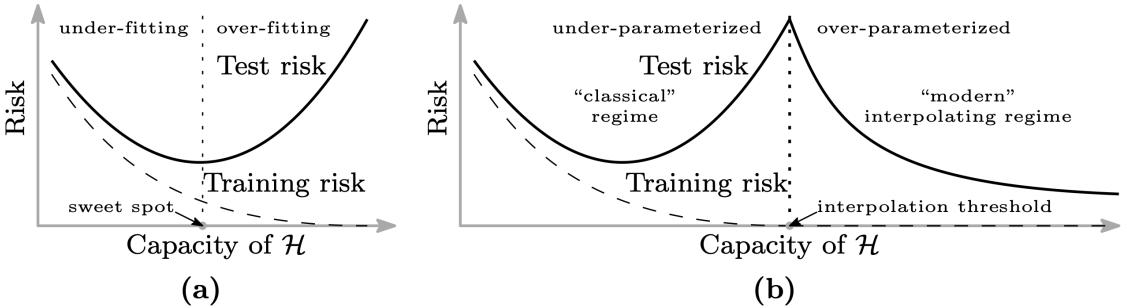
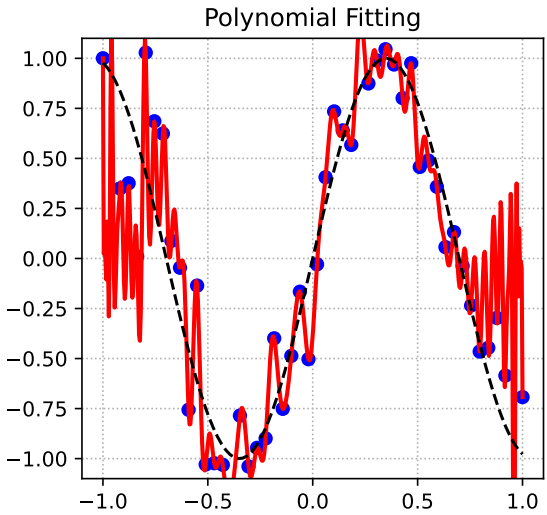
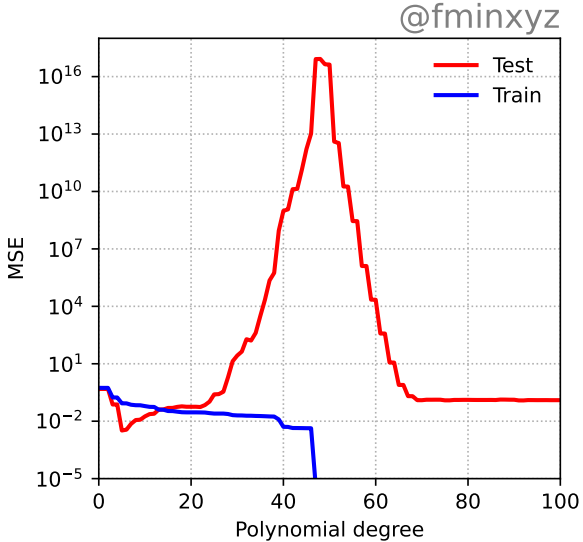


Figure 11: Curves for training risk (dashed line) and test risk (solid line). (a) The classical U-shaped risk curve arising from the bias-variance trade-off. (b) The double descent risk curve, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high capacity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

[5] Belkin, Mikhail, et al. "Reconciling modern machine-learning practice and the classical bias–variance trade-off." (2019)

Double Descent





Shampoo [6]

Stands for **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. It's a method inspired by second-order optimization designed for large-scale deep learning.

Core Idea: Approximates the full-matrix AdaGrad pre conditioner using efficient matrix structures, specifically Kronecker products.

For a weight matrix $W \in \mathbb{R}^{m \times n}$, the update involves preconditioning using approximations of the statistics matrices $L \approx \sum_k G_k G_k^T$ and $R \approx \sum_k G_k^T G_k$, where G_k are the gradients.

Simplified concept:

1. Compute gradient G_k.
2. Update statistics $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ and $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Compute preconditioners $P_L = L_k^{-1/4}$ and $P_R = R_k^{-1/4}$. (Inverse matrix root)
4. Update: $W_{k+1} = W_k - \alpha P_L G_k P_R$.

Notes:

- Aims to capture curvature information more effectively than first-order methods.
- Computationally more expensive than Adam but can converge faster or to better solutions in terms of steps.
- Requires careful implementation for efficiency (e.g., efficient computation of inverse matrix roots, handling large matrices).
- Variants exist for different tensor shapes (e.g., convolutional layers).

[6] Gupta, V., Koren, T. and Singer, Y., 2018, July. Shampoo: Preconditioned stochastic tensor optimization. In International Conference on Machine Learning (pp. 1842-1850). PMLR.

Muon [7] [8] [9]

$$
\begin{aligned}
W_{t+1} &= W_t - \eta (G_t G_t^\top)^{-1/4} G_t (G_t^\top G_t)^{-1/4} \\
&= W_t - \eta (U S^2 U^\top)^{-1/4} (USV^\top)(VS^2V^\top)^{-1/4} \\
&= W_t - \eta (U S^{-1/2} U^\top)(USV^\top)(VS^{-1/2}V^\top) \\
&= W_t - \eta U S^{-1/2} S S^{-1/2} V^\top \\
&= W_t - \eta U V^\top
\end{aligned}
$$

[7] K. Jordan blogpost "Muon: An optimizer for hidden layers in neural networks". 2024.

[8] J. Bernstein blogpost "Deriving Muon". 2025.

[9] Kovalev, D. (2025). Understanding Gradient Orthogonalization for Deep Learning via Non-Euclidean Trust-Region Optimization. arXiv preprint arXiv:2503.12645.

Muon comparison with AdamW on LogReg

- Simple comparison of Muon and AdamW on small LogReg problem

Additional materials

- D. Vetrov "Surprising properties of loss lansdcape in overparametrized models"
- V. Goloshapov "What grokking is not about"