

Optimization: Advanced Topic

Variance Reduction and Momentum Methods

Mathematical Foundation of Data Science
Difan Zou

Nov. 25, 2025

Outline

- 1 Introduction and Motivation
- 2 SGD and the Variance Problem
- 3 Variance Reduction Methods
- 4 Momentum Methods
- 5 Adaptive Gradient Methods
- 6 Summary

Part 1

Introduction and Motivation

Review of Gradient Descent and Its Limitations

Review: Gradient Descent

Standard Gradient Descent Update:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla L(\theta^{(t)})$$

Convergence Rates (Previously Covered):

- **Smooth functions:** $\|\nabla L(\theta^{(t)})\|^2 = O(1/t)$
- **Strongly convex:** $L(\theta^{(t)}) - L(\theta^*) = O(e^{-t/\kappa})$

where $\kappa = L/\mu$ is the **condition number**

- L = smoothness constant (Lipschitz gradient)
- μ = strong convexity constant

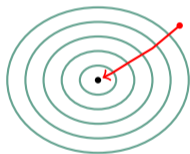
Condition Number: Definition and Impact

Definition: The **condition number** is $\kappa = \frac{L}{\mu}$

- $L =$ **Smoothness** (largest curvature), $\mu =$ **Strong convexity** (smallest curvature)

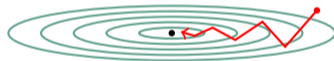
Impact on Convergence: $L(\theta^{(t)}) - L(\theta^*) \leq \left(1 - \frac{1}{\kappa}\right)^t (L(\theta^{(0)}) - L(\theta^*)) \Rightarrow$ **Iterations:** $O(\kappa \log(1/\varepsilon))$

$$f = \frac{1}{2}(x^2 + 1.5y^2)$$



Small $\kappa \approx 1$
Direct path
Fast convergence

$$f = \frac{1}{2}(x^2 + 100y^2)$$



Large $\kappa \gg 1$
Zig-zag path
Slow convergence

Key insight: Large κ forces small step size (due to L), causing slow progress along low-curvature directions.

Example: Condition Number Impact

Consider quadratic: $f(\theta) = \frac{1}{2}\theta^T A\theta$ where $A = \text{diag}(\mu, \mu, \dots, L)$

Gradient: $\nabla f(\theta) = A\theta$

Optimal step size: $\eta = \frac{2}{L+\mu}$

Convergence:

$$\|\theta^{(t)} - \theta^*\| \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^t \|\theta^{(0)} - \theta^*\|$$

Example values:

- $\kappa = 10$: $(9/11)^t \approx 0.82^t$ — converges in ~ 20 iterations
- $\kappa = 100$: $(99/101)^t \approx 0.98^t$ — converges in ~ 200 iterations

Finite-Sum Optimization Problem

Many machine learning problems have the form:

$$\min_{\theta} L(\theta) = \frac{1}{n} \sum_{i=1}^n L_i(\theta)$$

Examples:

- Empirical Risk Minimization
- Logistic Regression: $L_i(\theta) = \log(1 + e^{-y_i \theta^T x_i})$
- Linear Regression: $L_i(\theta) = \frac{1}{2} (y_i - \theta^T x_i)^2$
- SVM: $L_i(\theta) = \max(0, 1 - y_i \theta^T x_i)$

Cost per iteration: $O(n \cdot d)$ for full gradient

- When n is large (millions of samples), this is expensive!

Concrete Example: Large-Scale Training

ImageNet classification:

- $n = 1.2$ million images
- $d \approx 25$ million parameters (ResNet-50)
- Full gradient: $1.2M \times 25M = 3 \times 10^{13}$ operations

On modern GPU:

- ~ 10 TFLOPs \Rightarrow 50 seconds per iteration!
- Need thousands of iterations \Rightarrow days of training

With mini-batch SGD (batch size 256):

- $256 \times 25M = 6.4 \times 10^9$ operations
- ~ 0.01 seconds per iteration
- Thousands of iterations \Rightarrow hours of training

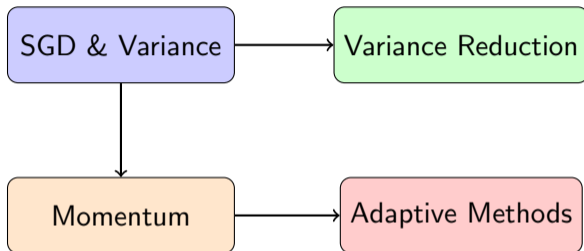
This is why we need stochastic methods!

Limitations of Standard GD

- ① **Computational Cost:** $O(n)$ per iteration for finite-sum problems
 - Need to compute gradients for ALL n samples
- ② **Uniform Learning Rate:** Same η for all directions
 - Different parameters may have different scales/curvatures
- ③ **Condition Number Dependence:** Convergence $\propto e^{-t/\kappa}$
 - If κ is large, convergence is slow
 - Ill-conditioned problems: $\kappa \gg 1$

Goal: Develop methods that address these limitations

Today's Roadmap



Key Ideas:

- Reduce variance \rightarrow faster convergence with constant step size
- Add momentum \rightarrow acceleration, smoother optimization
- Adapt learning rates \rightarrow handle different scales

Part 2

SGD and the Variance Problem

Understanding the Fundamental Trade-off

Mini-batch SGD Review

Update Rule:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \frac{1}{B} \sum_{i \in \mathcal{I}^{(t)}} \nabla L_i(\theta^{(t)})$$

where $\mathcal{I}^{(t)}$ is a random mini-batch of size B

Key Property - Unbiasedness:

$$\mathbb{E}[g_t \mid \theta^{(t)}] = \nabla L(\theta^{(t)})$$

Per-iteration cost: $O(B \cdot d)$ instead of $O(n \cdot d)$

- Huge savings when $B \ll n$

The Variance Problem - Core Issue

Stochastic gradient has variance that limits convergence!

Bounded Variance Assumption:

$$\mathbb{E} \left[\|g_t - \nabla L(\theta^{(t)})\|^2 \right] \leq \sigma^2$$

Convergence Bound for SGD:

$$\mathbb{E}[L(\theta^{(t+1)})] \leq L(\theta^{(t)}) - \underbrace{\left(\eta - \frac{L\eta^2}{2} \right) \|\nabla L(\theta^{(t)})\|^2}_{\text{Decreasing term}} + \underbrace{\frac{L\eta^2\sigma^2}{2}}_{\text{Variance term}}$$

- The variance term $\frac{L\eta^2\sigma^2}{2}$ does NOT decrease!
- It prevents convergence to exact optimum

Derivation: SGD Convergence Bound

Setup: L is L -smooth, i.e., $\|\nabla L(\theta) - \nabla L(\theta')\| \leq L\|\theta - \theta'\|$

Step 1: By smoothness,

$$\begin{aligned} L(\theta^{(t+1)}) &\leq L(\theta^{(t)}) + \langle \nabla L(\theta^{(t)}), \theta^{(t+1)} - \theta^{(t)} \rangle \\ &\quad + \frac{L}{2} \|\theta^{(t+1)} - \theta^{(t)}\|^2 \end{aligned}$$

Step 2: Substitute $\theta^{(t+1)} = \theta^{(t)} - \eta g_t$:

$$L(\theta^{(t+1)}) \leq L(\theta^{(t)}) - \eta \langle \nabla L(\theta^{(t)}), g_t \rangle + \frac{L\eta^2}{2} \|g_t\|^2$$

Step 3: Take expectation and use $\mathbb{E}[g_t] = \nabla L(\theta^{(t)})$...

Derivation: SGD Convergence Bound (cont.)

Step 3 (continued): Taking expectation,

$$\mathbb{E}[L(\theta^{(t+1)})] \leq L(\theta^{(t)}) - \eta \|\nabla L(\theta^{(t)})\|^2 + \frac{L\eta^2}{2} \mathbb{E}[\|g_t\|^2]$$

Step 4: Decompose $\mathbb{E}[\|g_t\|^2]$:

$$\begin{aligned}\mathbb{E}[\|g_t\|^2] &= \mathbb{E}[\|g_t - \nabla L + \nabla L\|^2] \\ &= \mathbb{E}[\|g_t - \nabla L\|^2] + 2\langle \mathbb{E}[g_t - \nabla L], \nabla L \rangle + \|\nabla L\|^2 \\ &= \sigma^2 + \|\nabla L(\theta^{(t)})\|^2\end{aligned}$$

Step 5: Substitute back:

$$\mathbb{E}[L(\theta^{(t+1)})] \leq L(\theta^{(t)}) - \left(\eta - \frac{L\eta^2}{2}\right) \|\nabla L(\theta^{(t)})\|^2 + \frac{L\eta^2\sigma^2}{2}$$

Consequences of Variance

Problem: Cannot use constant step size

- SGD oscillates around optimum instead of converging

Solution: Decrease $\eta_t \rightarrow 0$

- Common choice: $\eta_t = O(1/t)$ or $\eta_t = O(1/\sqrt{t})$
- But this leads to slower convergence!

Convergence Rate Comparison (Strongly Convex):

| Method | Rate |
|--------------------------|----------------------------------|
| GD | $O(e^{-t/\kappa})$ (exponential) |
| SGD (decreasing η) | $O(1/t)$ (sublinear) |

SGD is much slower than GD near the optimum!

Example: SGD vs GD Iterations

Strongly convex problem with $\kappa = 100$, target accuracy $\varepsilon = 10^{-6}$

GD iterations needed:

$$t \approx \kappa \log(1/\varepsilon) = 100 \times 14 = 1400 \text{ iterations}$$

SGD iterations needed (with $\eta_t = 1/t$):

$$t \approx \frac{1}{\varepsilon} = 10^6 \text{ iterations}$$

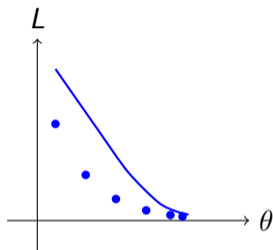
However:

- GD cost per iteration: $O(n)$
- SGD cost per iteration: $O(1)$

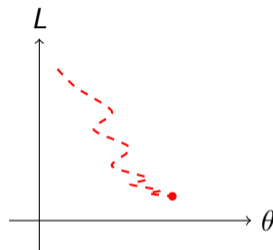
Total cost: GD needs $1400n$, SGD needs 10^6 operations

- If $n > 714$, SGD is still faster overall!

Visualizing the Variance Problem



GD: Smooth



SGD: Oscillates

Key Insight: If we can reduce variance while keeping cheap iterations, we get the best of both worlds!

Part 3

Variance Reduction Methods

SVRG, SAGA, and SPIDER

The Core Idea of Variance Reduction

Goal: Construct gradient estimator g_t such that:

- 1 **Unbiased:** $\mathbb{E}[g_t] = \nabla L(\theta^{(t)})$
- 2 **Reduced Variance:** $\text{Var}(g_t) \rightarrow 0$ as $\theta \rightarrow \theta^*$

Key Technique: Control Variates

- Use historical gradient information to “correct” stochastic gradients
- Maintain the same expectation (unbiased)
- Cancel out some of the randomness (lower variance)

Benefit: Can use constant step size \rightarrow linear convergence!

SVRG: Stochastic Variance Reduced Gradient

Algorithm Structure:

- 1: **Outer loop** (every m iterations):
- 2: Compute full gradient: $\mu = \nabla L(\tilde{\theta})$
- 3: Store reference point $\tilde{\theta}$
- 4:
- 5: **Inner loop** (for $t = 1$ to m):
- 6: Sample index i uniformly from $\{1, \dots, n\}$
- 7: $g_t = \nabla L_i(\theta^{(t)}) - \nabla L_i(\tilde{\theta}) + \mu$
- 8: $\theta^{(t+1)} = \theta^{(t)} - \eta \cdot g_t$

Complexity per epoch: $O(n + m)$, typically $m = O(n)$

Why SVRG Works

Unbiasedness:

$$\begin{aligned}\mathbb{E}[g_t] &= \mathbb{E}[\nabla L_i(\theta^{(t)}) - \nabla L_i(\tilde{\theta}) + \mu] \\ &= \nabla L(\theta^{(t)}) - \nabla L(\tilde{\theta}) + \nabla L(\tilde{\theta}) \\ &= \nabla L(\theta^{(t)}) \quad \checkmark\end{aligned}$$

Variance Reduction:

$$\text{Var}(g_t) \leq 2L\|\theta^{(t)} - \tilde{\theta}\|^2$$

- As $\theta^{(t)} \rightarrow \tilde{\theta}$ (within inner loop), variance $\rightarrow 0$
- The “correction term” $\nabla L_i(\theta^{(t)}) - \nabla L_i(\tilde{\theta})$ has small variance when $\theta \approx \tilde{\theta}$

SVRG Variance Analysis

Lemma: For L -smooth L_i ,

$$\mathbb{E}[\|g_t - \nabla L(\theta^{(t)})\|^2] \leq 2L\mathbb{E}[\|\theta^{(t)} - \tilde{\theta}\|^2]$$

Proof sketch:

$$g_t - \nabla L(\theta^{(t)}) = \nabla L_i(\theta^{(t)}) - \nabla L_i(\tilde{\theta}) - (\nabla L(\theta^{(t)}) - \nabla L(\tilde{\theta}))$$

Taking expectation over i :

$$\begin{aligned}\mathbb{E}[\|g_t - \nabla L(\theta^{(t)})\|^2] &= \mathbb{E}_i[\|\nabla L_i(\theta^{(t)}) - \nabla L_i(\tilde{\theta})\|^2] - \|\nabla L(\theta^{(t)}) - \nabla L(\tilde{\theta})\|^2 \\ &\leq \mathbb{E}_i[\|\nabla L_i(\theta^{(t)}) - \nabla L_i(\tilde{\theta})\|^2] \\ &\leq L^2\|\theta^{(t)} - \tilde{\theta}\|^2 \quad (\text{by smoothness})\end{aligned}$$

Wait, this gives L^2 not $2L$. The correct bound uses

$$\mathbb{E}_i[\|\nabla L_i(\theta) - \nabla L_i(\tilde{\theta})\|^2] \leq 2L(L_i(\theta) - L_i(\tilde{\theta}) - \langle \nabla L_i(\tilde{\theta}), \theta - \tilde{\theta} \rangle).$$

SVRG Convergence Theorem

Theorem: For strongly convex and smooth L with step size $\eta = \frac{1}{10L}$ and $m = 20\kappa$,

$$\mathbb{E}[L(\tilde{\theta}^{(s)})] - L(\theta^*) \leq \left(\frac{2}{3}\right)^s (L(\theta^{(0)}) - L(\theta^*))$$

where $\tilde{\theta}^{(s)}$ is the snapshot at epoch s .

Gradient complexity: To achieve $\mathbb{E}[L(\theta)] - L(\theta^*) \leq \varepsilon$,

$$\# \text{ gradient evaluations} = O\left((n + \kappa) \log \frac{1}{\varepsilon}\right)$$

Comparison:

- GD: $O(n\kappa \log(1/\varepsilon))$
- SVRG: $O((n + \kappa) \log(1/\varepsilon))$ — **much better when $n \gg \kappa$!**

SAGA: Stochastic Average Gradient Augmented

Key Idea: Store a table of gradients rather than the average of gradients, update incrementally

- 1: **Initialize:** Store $\alpha_i = \nabla L_i(\theta^{(0)})$ for all $i = 1, \dots, n$
- 2:
- 3: **for** $t = 1, 2, \dots$ **do**
- 4: Sample index i uniformly from $\{1, \dots, n\}$
- 5: $g_t = \nabla L_i(\theta^{(t)}) - \alpha_i + \frac{1}{n} \sum_{j=1}^n \alpha_j$
- 6: $\theta^{(t+1)} = \theta^{(t)} - \eta \cdot g_t$
- 7: Update table: $\alpha_i \leftarrow \nabla L_i(\theta^{(t)})$
- 8: **end for**

Note: Single-loop structure (simpler than SVRG)

Why SAGA Works

Unbiasedness:

$$\begin{aligned}\mathbb{E}[g_t] &= \mathbb{E} \left[\nabla L_i(\theta^{(t)}) - \alpha_i + \bar{\alpha} \right] \\ &= \nabla L(\theta^{(t)}) - \bar{\alpha} + \bar{\alpha} \\ &= \nabla L(\theta^{(t)}) \quad \checkmark\end{aligned}$$

Variance Reduction:

- Table $\{\alpha_i\}$ gradually converges to $\{\nabla L_i(\theta^*)\}$
- At optimum: $\alpha_i \approx \nabla L_i(\theta^*)$, so correction term is small

Trade-offs:

- Memory: $O(nd)$ to store gradient table
- Simpler single-loop structure
- Supports proximal operators naturally

SAGA Convergence Theorem

Theorem: For strongly convex and L -smooth L with step size $\eta = \frac{1}{3(\mu n + L)}$,

$$\mathbb{E}[\|\theta^{(t)} - \theta^*\|^2] \leq \left(1 - \min\left\{\frac{\mu}{3L}, \frac{1}{3n}\right\}\right)^t \|\theta^{(0)} - \theta^*\|^2$$

Gradient complexity:

$$\# \text{ gradient evaluations} = O\left((n + \kappa) \log \frac{1}{\varepsilon}\right)$$

Same complexity as SVRG, but:

- Single-loop (easier to implement)
- Requires $O(nd)$ memory
- Better for non-smooth objectives with proximal operators

SPIDER: Stochastic Path-Integrated Differential Estimator

Motivation: Achieve near-optimal complexity for non-convex problems

```
1: for  $k = 0, 1, 2, \dots$  (epochs) do  
2:   Compute full gradient:  $v_0 = \nabla L(\theta^{(k,0)})$   
3:  
4:   for  $t = 1$  to  $q$  do  
5:     Sample mini-batch  $\mathcal{S}$  of size  $B$   
6:      $v_t = \frac{1}{B} \sum_{i \in \mathcal{S}} [\nabla L_i(\theta^{(k,t)}) - \nabla L_i(\theta^{(k,t-1)})] + v_{t-1}$   
7:      $\theta^{(k,t+1)} = \theta^{(k,t)} - \eta \cdot v_t$   
8:   end for  
9: end for
```

Why SPIDER is Powerful

Key Innovation: Recursive variance reduction along optimization path

$$v_t = v_{t-1} + \frac{1}{B} \sum_{i \in \mathcal{S}} \left[\nabla L_i(\theta^{(t)}) - \nabla L_i(\theta^{(t-1)}) \right]$$

- v_t tracks the gradient using differential estimator
- Small steps \Rightarrow small difference $\nabla L_i(\theta^{(t)}) - \nabla L_i(\theta^{(t-1)})$
- Mini-batch size B can be much smaller than n

Gradient Complexity:

- Strongly convex: $O((n + \kappa) \log(1/\varepsilon))$
- Non-convex: $O(n^{1/2} \varepsilon^{-2})$ to find $\mathbb{E}[\|\nabla L\|^2] \leq \varepsilon$
- SPIDER exhibits advantages for non-convex optimization.

Intuition: Comparing Gradient Estimators

Standard SGD: $g_t = \nabla L_i(\theta^{(t)})$

[High variance]

SVRG: $g_t = \underbrace{\nabla L_i(\theta^{(t)}) - \nabla L_i(\tilde{\theta})}_{\text{Small variance}} + \underbrace{\nabla L(\tilde{\theta})}_{\text{Anchor}}$

SAGA: $g_t = \underbrace{\nabla L_i(\theta^{(t)}) - \alpha_i}_{\text{Correction}} + \underbrace{\bar{\alpha}}_{\text{Table avg}}$

SPIDER: $v_t = \underbrace{v_{t-1}}_{\text{Previous}} + \underbrace{\nabla L_i(\theta^{(t)}) - \nabla L_i(\theta^{(t-1)})}_{\text{Recursive correction}}$

Comparison: Memory and Structure

| Method | Memory | Per-iter | Structure |
|--------|---------|----------|-------------|
| SGD | $O(d)$ | $O(d)$ | Single loop |
| SAGA | $O(nd)$ | $O(d)$ | Single loop |
| SVRG | $O(d)$ | $O(d)$ | Double loop |
| SPIDER | $O(d)$ | $O(Bd)$ | Double loop |

Summary:

- SAGA requires $O(nd)$ memory but has simpler single-loop structure
- SVRG and SPIDER are memory-efficient but use double loops
- SPIDER offers flexibility in mini-batch size B

Comparison: Convergence Rates (Strongly Convex)

| Method | Gradient Evaluations to ε -accuracy |
|--------|---|
| GD | $O(n\kappa \log(1/\varepsilon))$ |
| SGD | $O(1/\varepsilon)$ |
| SAGA | $O((n + \kappa) \log(1/\varepsilon))$ |
| SVRG | $O((n + \kappa) \log(1/\varepsilon))$ |
| SPIDER | $O((n + \kappa) \log(1/\varepsilon))$ |

Key Insights:

- Variance reduction methods achieve **linear convergence** (log factor)
- When $n \gg \kappa$: SAGA/SVRG cost $\approx O(n \log(1/\varepsilon))$

Numerical Example: Convergence Comparison

Problem: Logistic regression, $n = 10000$, $d = 100$, $\kappa = 50$, $\varepsilon = 10^{-6}$

GD: $10000 \times 50 \times \log(10^6) \approx 6.9 \times 10^6$ gradient evaluations

SGD: 10^6 gradient evaluations (but very slow convergence)

SVRG/SAGA: $(10000 + 50) \times \log(10^6) \approx 1.4 \times 10^5$ gradient evaluations

Speedup over GD:

- SVRG/SAGA: $\sim 50\times$ faster
- SPIDER: $\sim 63\times$ faster

When to Use Which Method?

SAGA:

- When memory $O(nd)$ is affordable
- Want simple implementation (single loop)
- Need proximal operator support

SVRG:

- When memory is limited
- Okay with double-loop structure

SPIDER:

- For non-convex problems (deep learning)
- When n is very large

Standard SGD:

- When problem is not finite-sum (streaming data)
- Very large scale where even $O(d)$ per iteration matters

Practical Example: Memory Considerations

Example: Logistic regression with $n = 10,000$, $d = 100$

SAGA memory requirement:

$$n \times d \times 8 \text{ bytes} = 10,000 \times 100 \times 8 = 8 \text{ MB}$$

⇒ **Affordable!** Use SAGA.

Example: Large-scale with $n = 10,000,000$, $d = 100$

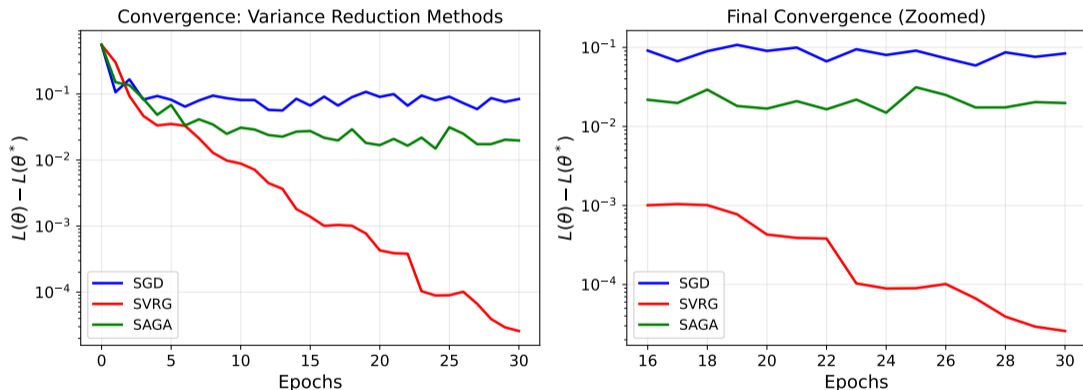
SAGA memory requirement:

$$10^7 \times 100 \times 8 = 8 \text{ GB}$$

⇒ **Too large!** Use SVRG or SPIDER instead.

Numerical Simulation: Variance Reduction Methods

Comparing SGD, SVRG, and SAGA on logistic regression



Observation: Variance reduction methods (SVRG, SAGA) converge linearly to optimum, while standard SGD plateaus due to gradient variance.

Part 4

Momentum Methods

Accelerating Convergence

Motivation for Momentum

Problem with Gradient Descent:

- Oscillates in high-curvature directions
- Moves slowly in low-curvature directions

Physical Analogy: Ball Rolling Downhill

GD: Position \propto slope (teleportation)

Momentum: Slope \rightarrow velocity \rightarrow position (physics)

Intuition:

- Consistent gradients \Rightarrow acceleration (build up speed)
- Oscillating gradients \Rightarrow damping (cancel out)

Physical Derivation: Ball with Friction

Consider a ball rolling in potential $L(\theta)$ with friction

Newton's second law with friction (the acceleration is governed by the force and friction):

$$m\ddot{\theta}(t) = -\nabla L(\theta(t)) - \gamma\dot{\theta}(t)$$

Discretize with time step h :

$$\begin{aligned}\dot{\theta}^{(t+1)} &= \dot{\theta}^{(t)} - \frac{h}{m}\nabla L(\theta^{(t)}) - \frac{h\gamma}{m}\dot{\theta}^{(t)} \\ &= \left(1 - \frac{h\gamma}{m}\right)\dot{\theta}^{(t)} - \frac{h}{m}\nabla L(\theta^{(t)})\end{aligned}$$

Setting $\beta = 1 - \frac{h\gamma}{m}$ and $\eta = \frac{h}{m}$, and $m_t = \dot{\theta}^{(t)}$:

$$m_{t+1} = \beta m_t - \eta \nabla L(\theta^{(t)})$$

This is exactly the momentum update!

Heavy Ball (Momentum) Method

Update Rule:

$$\begin{aligned}m_t &= \beta \cdot m_{t-1} + \eta \cdot \nabla L(\theta^{(t)}) \\ \theta^{(t+1)} &= \theta^{(t)} - m_t\end{aligned}$$

Parameters:

- $\beta \in [0, 1)$: momentum coefficient (typically $\beta = 0.9$)
- m_t : momentum (accumulated gradient)

Interpretation:

$$m_t = \eta \sum_{k=0}^t \beta^{t-k} \nabla L(\theta^{(k)})$$

Exponentially weighted sum of past gradients

Effective learning rate increases by factor $\frac{1}{1-\beta}$ for persistent gradients!

Why Momentum Helps: Example

Consider optimization in a valley (high curvature in one direction)

Without momentum:

- Large gradient \perp valley \Rightarrow large steps, oscillation
- Small gradient \parallel valley \Rightarrow small steps, slow progress

With momentum ($\beta = 0.9$):

- Gradients \perp valley oscillate $\Rightarrow m_t$ averages them out
- Gradients \parallel valley are consistent $\Rightarrow m_t$ accumulates them
- Effective step size \parallel valley: $\frac{\eta}{1-0.9} = 10\eta$

Result: Faster progress along valley, less oscillation across valley

Momentum: Convergence Analysis Setup

Assume: L is μ -strongly convex and L -smooth

Optimal parameters for quadratics:

$$\eta = \frac{4}{(\sqrt{L} + \sqrt{\mu})^2}, \quad \beta = \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^2$$

where $\kappa = L/\mu$

Example: $\kappa = 100$

$$\beta = \left(\frac{9}{11} \right)^2 \approx 0.67$$

Convergence rate:

$$\|\theta^{(t)} - \theta^*\| \leq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^t \|\theta^{(0)} - \theta^*\|$$

Compare to GD: $\left(\frac{\kappa-1}{\kappa+1} \right)^t$ — improvement from κ to $\sqrt{\kappa}$!

Nesterov Accelerated Gradient (NAG)

Update Rule:

$$\begin{aligned}m_t &= \beta \cdot m_{t-1} + \eta \cdot \nabla L(\theta^{(t)} - \beta \cdot m_{t-1}) \\ \theta^{(t+1)} &= \theta^{(t)} - m_t\end{aligned}$$

Key Difference: Compute gradient at “look-ahead” position

$$\theta^{(t)} - \beta \cdot m_{t-1}$$

Intuition:

- First, take a step in the momentum direction
- Then, compute gradient at the anticipated position
- This provides a “correction” to the momentum
- Prevents overshooting!

NAG: Why Look-Ahead Helps

Heavy Ball: Blind momentum - might overshoot

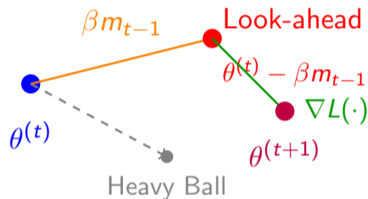
- Compute gradient at current position
- Add to existing momentum
- Can oscillate if momentum is too large

NAG: Corrective momentum - anticipates overshoot

- Predict where momentum will take us
- Compute gradient at that future position
- Adjust momentum based on future gradient
- Dampens oscillation automatically

Result: More stable, provably optimal convergence rate

NAG Look-Ahead Gradient Visualization



NAG anticipates where momentum will take us, then corrects based on gradient there.

NAG: Convergence Analysis

Theorem (Nesterov 1983): For μ -strongly convex and L -smooth L ,

With optimal parameters:

$$L(\theta^{(t)}) - L(\theta^*) \leq \frac{L\|\theta^{(0)} - \theta^*\|^2}{2} \left(1 - \frac{1}{\sqrt{\kappa}}\right)^t$$

Iteration complexity to reach ε -accuracy:

$$t = O(\sqrt{\kappa} \log(1/\varepsilon))$$

Comparison:

- GD: $O(\kappa \log(1/\varepsilon))$
- Heavy Ball: $O(\sqrt{\kappa} \log(1/\varepsilon))$ (on quadratics)
- NAG: $O(\sqrt{\kappa} \log(1/\varepsilon))$ (on general strongly convex)

NAG is optimal for first-order methods!

NAG Convergence: Proof Sketch

Key idea: Use an “estimate sequence” $\phi_t(\theta)$

Define:

$$\phi_t(\theta) = \frac{\mu}{2} \|\theta - v_t\|^2 + \psi_t$$

where v_t and ψ_t are updated each iteration.

Key properties:

- ① $\phi_t(\theta^*) \geq L(\theta^*)$ for all t
- ② $L(\theta^{(t)}) \leq \phi_t^* = \min_{\theta} \phi_t(\theta)$
- ③ $\phi_{t+1}^* \leq (1 - 1/\sqrt{\kappa})\phi_t^*$

Combining these:

$$L(\theta^{(t)}) - L(\theta^*) \leq \phi_t^* - L(\theta^*) \leq (1 - 1/\sqrt{\kappa})^t (\phi_0^* - L(\theta^*))$$

Convergence Rates: GD vs Momentum vs NAG

General Convex Functions:

| Method | $L(\theta^{(t)}) - L(\theta^*)$ |
|------------|---------------------------------|
| GD | $O(1/t)$ |
| Heavy Ball | $O(1/t)$ |
| NAG | $O(1/t^2)$ |

Strongly Convex Functions (condition number $\kappa = L/\mu$):

| Method | $L(\theta^{(t)}) - L(\theta^*)$ |
|------------|--|
| GD | $O(e^{-t/\kappa})$ |
| Heavy Ball | $O(e^{-t/\sqrt{\kappa}})$ (quadratics) |
| NAG | $O(e^{-t/\sqrt{\kappa}})$ (general) |

NAG provides quadratic speedup in convergence rate!

Numerical Example: Speedup from Momentum

Problem: $f(\theta) = \frac{1}{2}\theta^T A \theta$ with $\kappa = 100$, $\varepsilon = 10^{-6}$

GD iterations:

$$t \approx \kappa \log(1/\varepsilon) = 100 \times 14 = 1400$$

NAG iterations:

$$t \approx \sqrt{\kappa} \log(1/\varepsilon) = 10 \times 14 = 140$$

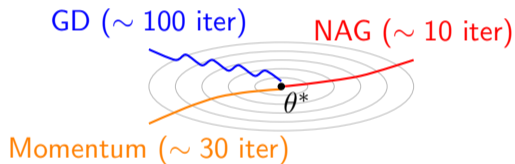
Speedup: 10× fewer iterations!

For $\kappa = 10000$:

- GD: 140,000 iterations
- NAG: 1,400 iterations
- Speedup: 100×!

Visual Comparison: Optimization Trajectories

Example: $f(x, y) = x^2 + 100y^2$ (condition number $\kappa = 100$)



NAG takes the most direct path with fewest oscillations

Momentum in Practice: Hyperparameter Tuning

Common values for β :

- $\beta = 0.9$: standard choice for most problems
- $\beta = 0.99$: for very ill-conditioned problems
- $\beta = 0$: reduces to standard GD

Rule of thumb:

$$\beta \approx 1 - \frac{2}{\sqrt{\kappa} + 1}$$

Examples:

- $\kappa = 10$: $\beta \approx 0.5$
- $\kappa = 100$: $\beta \approx 0.82$
- $\kappa = 10000$: $\beta \approx 0.98$

In practice: Start with $\beta = 0.9$, increase if training is slow

SGD with Momentum (Deep Learning)

Update Rule:

$$m_t = \beta \cdot m_{t-1} + \eta \cdot g_t$$
$$\theta^{(t+1)} = \theta^{(t)} - m_t$$

where g_t is the stochastic gradient (mini-batch)

Benefits in Deep Learning:

- Smooths out noise in stochastic gradients
- Helps escape saddle points and local minima
- Reduces oscillation from high variance
- Standard choice: $\beta = 0.9$

This is the default optimizer in many vision tasks!

Part 5

Adaptive Gradient Methods

AdaGrad, RMSProp, Adam, and Coordinate-wise Preconditioning

Motivation: Non-uniform Curvature

Problem: Different parameters need different learning rates

Ideal Update (Newton's Method):

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1} \nabla L(\theta^{(t)})$$

where $H = \nabla^2 L$ is the Hessian

Per-coordinate ideal:

$$\Delta\theta_i \propto \frac{\nabla_{\theta_i} L}{\partial^2 L / \partial \theta_i^2}$$

Problem: Computing H^{-1} is expensive ($O(d^3)$)

Solution: Approximate with diagonal scaling using squared gradients

Example: Why Uniform Learning Rate Fails

Consider: $f(\theta_1, \theta_2) = \frac{1}{2}\theta_1^2 + 50\theta_2^2$

Curvatures: $\frac{\partial^2 f}{\partial \theta_1^2} = 1$, $\frac{\partial^2 f}{\partial \theta_2^2} = 100$

With uniform $\eta = 0.01$:

- θ_1 direction: reasonable progress
- θ_2 direction: very slow (high curvature needs smaller step)

Ideal adaptive rates:

$$\eta_1 = 0.01, \quad \eta_2 = 0.01/100 = 0.0001$$

Adaptive methods automatically discover these different rates!

Coordinate-wise Preconditioning

Key Insight: Use squared gradients to estimate curvature

The second derivative can be approximated:

$$\frac{\partial^2 L}{\partial \theta_i^2} \approx \mathbb{E} \left[\left(\frac{\partial L}{\partial \theta_i} \right)^2 \right]$$

Coordinate-wise Preconditioned Update:

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{v_i + \varepsilon}} \cdot g_i^{(t)}$$

where v_i is an estimate of $\mathbb{E}[g_i^2]$

Effect: Large gradients \Rightarrow small effective step size

Small gradients \Rightarrow large effective step size

This is the foundation of AdaGrad, RMSProp, and Adam!

AdaGrad

Update Rule:

$$G_t = \sum_{k=0}^t (\nabla L(\theta^{(k)}))^2 \quad (\text{element-wise})$$
$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{\sqrt{G_t} + \varepsilon} \odot \nabla L(\theta^{(t)})$$

Properties:

- Automatically adapts learning rate per parameter
- Good for sparse data (infrequent features get larger updates)
- Convergence guarantee for convex problems

Limitation: Learning rate monotonically decreases

- G_t only grows \Rightarrow can stop learning too early
- Not suitable for non-convex optimization (deep learning)

AdaGrad Example: Sparse Features

Problem: Text classification with word features

Feature 1: Common word “the” - appears in 90% of documents

- Gradients computed frequently $\Rightarrow G_t$ grows fast
- Effective learning rate decreases quickly

Feature 2: Rare word “cryptocurrency” - appears in 1% of documents

- Gradients computed rarely $\Rightarrow G_t$ grows slowly
- Effective learning rate stays large
- Can make large updates when this word appears!

Result: Rare features can still learn effectively despite infrequent updates

RMSProp

Fix AdaGrad's limitation: Use exponential moving average

Update Rule:

$$G_t = \gamma \cdot G_{t-1} + (1 - \gamma) \cdot (\nabla L(\theta^{(t)}))^2$$
$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{\sqrt{G_t} + \varepsilon} \odot \nabla L(\theta^{(t)})$$

Parameters:

- $\gamma \approx 0.9$: decay rate
- Learning rate can now recover (old gradients decay)

Advantages over AdaGrad:

- Works for non-convex optimization
- Particularly good for RNNs (handles varying curvature in sequences)

Adam: Adaptive Moment Estimation

Combines momentum + adaptive learning rate

Update Rule:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \quad \text{(First moment)}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \quad \text{(Second moment)}$$

Bias Correction:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Parameter Update:

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

Adam: Default Hyperparameters

Recommended values (work well in practice):

- $\beta_1 = 0.9$ (momentum decay)
- $\beta_2 = 0.999$ (second moment decay)
- $\varepsilon = 10^{-8}$ (numerical stability)
- $\eta = 0.001$ (learning rate)

Why these values?

- $\beta_1 = 0.9$: effective window $\approx \frac{1}{1-0.9} = 10$ gradients
- $\beta_2 = 0.999$: effective window ≈ 1000 squared gradients
- Higher β_2 for more stable curvature estimate
- Different time scales for mean vs. variance estimation

Why Bias Correction?

Problem: Initial estimates are biased toward zero

At $t = 0$: $m_0 = 0$, $v_0 = 0$

After t steps:

$$\mathbb{E}[m_t] = (1 - \beta_1^t) \cdot \mathbb{E}[g]$$

Example: $\beta_1 = 0.9$, $t = 1$

$$\mathbb{E}[m_1] = (1 - \beta_1)g_1 = 0.1g_1 \quad (\text{only 10\% of true value!})$$

Correction: Divide by $(1 - \beta_1^t)$

$$\mathbb{E}[\hat{m}_t] = \frac{(1 - \beta_1^t)\mathbb{E}[g]}{1 - \beta_1^t} = \mathbb{E}[g] \quad \checkmark$$

As $t \rightarrow \infty$, $(1 - \beta_1^t) \rightarrow 1$, correction becomes negligible

Bias Correction: Visual Impact

First 10 iterations without bias correction:

Assume constant gradient $g = 1$, $\beta_1 = 0.9$:

| t | m_t (biased) | \hat{m}_t (corrected) |
|----------|----------------|-------------------------|
| 1 | 0.1 | 1.0 |
| 2 | 0.19 | 1.0 |
| 3 | 0.271 | 1.0 |
| 5 | 0.410 | 1.0 |
| 10 | 0.651 | 1.0 |
| ∞ | 1.0 | 1.0 |

Without correction: First updates are too small!

With correction: Proper step size from the start

Connection to SignGD

SignGD Update:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \text{sign}(\nabla L(\theta^{(t)}))$$

Observation: Adam approximates SignGD!

When $\beta_1 \approx \beta_2$ and gradients are consistent:

$$\frac{\hat{m}_t}{\sqrt{\hat{v}_t}} \approx \frac{\mathbb{E}[g]}{\sqrt{\mathbb{E}[g^2]}} \approx \frac{g}{|g|} = \text{sign}(g)$$

Intuition: Adam normalizes each coordinate by its typical magnitude

- Large persistent gradients \Rightarrow normalized to ± 1
- Small noisy gradients \Rightarrow also normalized to ± 1
- All directions take similar-sized steps!

SignGD Convergence Analysis

SignGD Update: $\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \text{sign}(\nabla L(\theta^{(t)}))$

For L -smooth functions:

$$\begin{aligned} L(\theta^{(t+1)}) &\leq L(\theta^{(t)}) + \langle \nabla L(\theta^{(t)}), \theta^{(t+1)} - \theta^{(t)} \rangle + \frac{L}{2} \|\theta^{(t+1)} - \theta^{(t)}\|^2 \\ &= L(\theta^{(t)}) - \eta \langle \nabla L(\theta^{(t)}), \text{sign}(\nabla L(\theta^{(t)})) \rangle + \frac{L\eta^2 d}{2} \\ &= L(\theta^{(t)}) - \eta \|\nabla L(\theta^{(t)})\|_1 + \frac{L\eta^2 d}{2} \end{aligned}$$

Note: $\|\theta^{(t+1)} - \theta^{(t)}\|^2 = \eta^2 \|\text{sign}(\nabla L)\|^2 = \eta^2 d$

Using $\|\nabla L\|_1 \geq \|\nabla L\|_2$:

$$L(\theta^{(t+1)}) \leq L(\theta^{(t)}) - \eta \|\nabla L(\theta^{(t)})\|_2 + \frac{L\eta^2 d}{2}$$

SignGD Convergence (Continued)

Summing over T iterations:

$$\sum_{t=0}^{T-1} \|\nabla L(\theta^{(t)})\|_2 \leq \frac{L(\theta^{(0)}) - L(\theta^*)}{\eta} + \frac{L\eta dT}{2}$$

Optimal step size: $\eta = \sqrt{\frac{2(L(\theta^{(0)}) - L(\theta^*))}{LdT}}$

Convergence Rate:

$$\min_{t < T} \|\nabla L(\theta^{(t)})\|_2 \leq \sqrt{\frac{2L(L(\theta^{(0)}) - L(\theta^*))d}{T}}$$

Rate: $O(\sqrt{d/T})$

- Dimension-dependent but can be fast for large gradients
- Worse than GD's $O(1/T)$ by factor \sqrt{d}
- But very robust to gradient magnitudes!

AdamW: Decoupled Weight Decay

Problem with L2 regularization in Adam:

Standard L2: Add $\lambda\theta$ to gradient before adaptive scaling

$$g_t \leftarrow g_t + \lambda\theta^{(t)}$$

This gets scaled by $1/\sqrt{\hat{v}_t}$, weakening regularization!

AdamW Solution: Decouple weight decay

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} - \eta\lambda\theta^{(t)}$$

Result: Better generalization in practice

- Now the standard for training transformers
- Weight decay applied uniformly, not scaled by gradient magnitude

Example: L2 vs Weight Decay in Adam

Consider parameter with small gradients $g \approx 0$:

Adam with L2:

- Effective gradient: $g + \lambda\theta$
- If $g \approx 0$, then $\hat{v}_t \approx \lambda^2\theta^2$
- Update: $\theta \leftarrow \theta - \frac{\eta\lambda\theta}{\sqrt{\lambda^2\theta^2}} = \theta - \frac{\eta\lambda\theta}{|\lambda\theta|}$
- Regularization effect scaled by adaptive term!

AdamW:

- Gradient: g (unmodified)
- Update: $\theta \leftarrow \theta - \frac{\eta g}{\sqrt{\hat{v}_t}} - \eta\lambda\theta$
- Consistent weight decay regardless of gradient magnitude!

Comparison: Adaptive Methods

| Method | 1st moment | 2nd moment | Bias corr. |
|----------|------------|------------|------------|
| SGD | No | No | N/A |
| Momentum | EMA | No | No |
| AdaGrad | No | Sum | No |
| RMSProp | No | EMA | No |
| Adam | EMA | EMA | Yes |
| AdamW | EMA | EMA | Yes |

Summary:

- AdaGrad: coordinate-wise preconditioning with cumulative sum
- RMSProp: fixes AdaGrad with exponential moving average
- Adam: combines momentum with RMSProp + bias correction
- AdamW: Adam with proper weight decay

When to Use Which Optimizer?

SGD + Momentum:

- Computer vision tasks (CNNs, ResNets)
- When you can carefully tune learning rate schedule
- Often achieves best final accuracy (with tuning)

Adam / AdamW:

- NLP tasks (Transformers, BERT, GPT)
- Fast prototyping (robust to hyperparameters)
- When training is unstable
- Default choice for most deep learning

Adam Algorithm: Complete Pseudocode

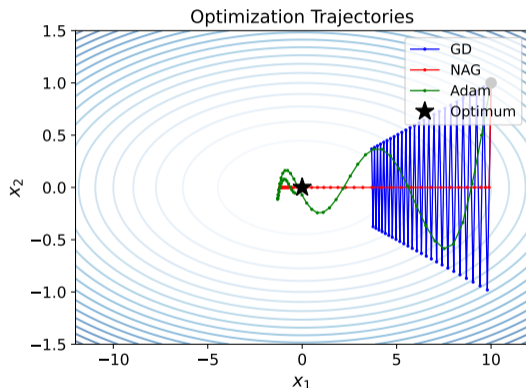
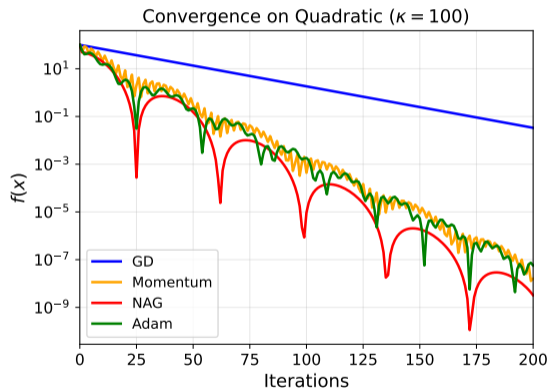
- 1: **Input:** $\eta, \beta_1, \beta_2, \varepsilon$, initial $\theta^{(0)}$
- 2: Initialize: $m_0 = 0, v_0 = 0$
- 3: **for** $t = 1, 2, \dots$ **do**
- 4: $g_t = \nabla L(\theta^{(t-1)})$ (or mini-batch gradient)
- 5: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
- 6: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
- 7: $\hat{m}_t = m_t / (1 - \beta_1^t)$
- 8: $\hat{v}_t = v_t / (1 - \beta_2^t)$
- 9: $\theta^{(t)} = \theta^{(t-1)} - \eta \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon)$
- 10: **end for**

All operations are element-wise

Typical settings: $\eta = 10^{-3}, \beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}$

Numerical Simulation: Momentum and Adaptive Methods

Comparing GD, Momentum, NAG, and Adam on ill-conditioned quadratic ($\kappa = 100$)



Observation: NAG and Adam converge significantly faster than vanilla GD on ill-conditioned problems. Adam adapts learning rates automatically, NAG provides optimal acceleration.

Practical Tips: Learning Rate Schedules

Learning rate scheduling is crucial for final performance

Common Schedules:

1. Step Decay:

$$\eta_t = \eta_0 \cdot \gamma^{\lfloor t/T \rfloor}, \quad \gamma \in (0, 1)$$

Drop learning rate by factor γ every T epochs

2. Cosine Annealing:

$$\eta_t = \frac{\eta_0}{2} \left(1 + \cos \left(\frac{t\pi}{T} \right) \right)$$

Smooth decay, popular for deep learning

3. Warmup + Decay:

- Linear warmup for first T_w steps: $\eta_t = \frac{t}{T_w} \eta_0$
- Then decay (cosine or exponential)
- Critical for large batch training and transformers

Example: Learning Rate Schedule Impact

Training ResNet-50 on ImageNet

Setup: SGD with momentum ($\beta = 0.9$), batch size 256

Constant $\eta = 0.1$:

- Training accuracy: 95%
- Validation accuracy: 72%
- Training unstable near end

Step decay ($\eta_0 = 0.1$, $\gamma = 0.1$ at epochs 30, 60, 90):

- Training accuracy: 99%
- Validation accuracy: 76%
- Smooth convergence

Key lesson: Decreasing learning rate helps fine-tune the solution

Hyperparameter Tuning: Practical Guidelines

Priority Order (from most to least important):

1. Learning Rate η :

- Try: $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$
- Use largest value that doesn't diverge
- For Adam, often 10^{-3} or 10^{-4}

2. Batch Size:

- Larger is better (up to memory limits)
- Typical: 32, 64, 128, 256, 512
- Scale learning rate with batch size: $\eta \propto \sqrt{B}$

3. Momentum β :

- Default 0.9 works well
- Increase to 0.95 or 0.99 for very ill-conditioned problems

4. Adam's β_1, β_2 : Usually keep at defaults (0.9, 0.999)

Combining Techniques: Best Practices

Modern deep learning typically combines multiple techniques:

For Vision (CNNs):

- Optimizer: SGD + Momentum ($\beta = 0.9$)
- Initial LR: 0.1 (scaled with batch size)
- Schedule: Step decay or cosine annealing
- Regularization: Weight decay ($\lambda = 10^{-4}$), data augmentation
- Warmup: 5 epochs linear warmup

For NLP (Transformers):

- Optimizer: AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.98$ or 0.999)
- Initial LR: 10^{-4} to 10^{-3}
- Schedule: Linear decay or inverse sqrt
- Warmup: Critical! 4000-10000 steps
- Gradient clipping: norm = 1.0

Batch Size vs Learning Rate Scaling

Linear Scaling Rule (Goyal et al., 2017):

When increasing batch size from B_1 to B_2 , scale learning rate:

$$\eta_2 = \frac{B_2}{B_1} \eta_1$$

Intuition: Larger batches \Rightarrow lower gradient variance \Rightarrow can use larger LR

Example:

- Baseline: batch 256, LR = 0.1
- Large batch: batch 8192, LR = $0.1 \times 8192/256 = 3.2$

Limitation: Doesn't work beyond certain batch size

- Very large batches may hurt generalization
- Need longer warmup for large batches

Part 6

Summary

Putting It All Together

Three Design Principles

① Reduce Variance

- Use control variates (SVRG, SAGA, SPIDER)
- Enables constant step size \rightarrow faster convergence
- Key for finite-sum problems

② Accumulate Momentum

- Smooth out oscillations
- Accelerate in consistent directions ($\sqrt{\kappa}$ improvement)
- Help escape local minima

③ Adapt Learning Rates

- Coordinate-wise preconditioning
- Handle different scales automatically
- More robust to hyperparameter choices

Method Selection Guide

| Scenario | Recommended Method |
|------------------------------------|--------------------|
| Convex, finite-sum, memory OK | SAGA |
| Convex, finite-sum, memory limited | SVRG |
| Non-convex, finite-sum | SPIDER |
| Deep learning (vision) | SGD + Momentum |
| Deep learning (NLP) | Adam / AdamW |

Note: These are guidelines based on empirical performance. Always validate on your specific problem!

Complexity Summary

| Method | Memory | Per-iter | Best for |
|--------|---------|----------|------------------------|
| SGD | $O(d)$ | $O(d)$ | Large scale, streaming |
| SVRG | $O(d)$ | $O(d)$ | Convex, mem-limited |
| SAGA | $O(nd)$ | $O(d)$ | Convex, simple impl. |
| SPIDER | $O(d)$ | $O(Bd)$ | Non-convex, large n |
| Adam | $O(d)$ | $O(d)$ | Deep learning |

Key insight: Choose based on:

- Problem structure (convexity, finite-sum, scale)
- Practical constraints (memory, implementation complexity)
- Domain (vision vs NLP vs sparse features)

Convergence Rate Summary

For ε -accuracy on strongly convex finite-sum with condition number κ :

| Method | Gradient Complexity |
|-----------------|---|
| GD | $O(n\kappa \log(1/\varepsilon))$ |
| SGD | $O(1/\varepsilon)$ |
| SAGA/SVRG | $O((n + \kappa) \log(1/\varepsilon))$ |
| SPIDER | $O((n + \sqrt{n\kappa}) \log(1/\varepsilon))$ |
| NAG (full grad) | $O(n\sqrt{\kappa} \log(1/\varepsilon))$ |

Takeaway:

- Variance reduction: eliminates κ dependence when $n \gg \kappa$
- Momentum: improves κ to $\sqrt{\kappa}$ for full gradient
- Combined: best results when both apply

Practical Recommendations

For Convex Optimization:

- ① Small-medium data ($n < 10^5$): Use SAGA or L-BFGS
- ② Large data ($n > 10^6$): Use SVRG or SPIDER
- ③ Very large data: SGD with decreasing step size

For Deep Learning:

- ① Start with Adam/AdamW (robust, fast prototyping)
- ② For final models: Try SGD + Momentum with tuned schedule
- ③ Vision: SGD + Momentum often best (with warmup + decay)
- ④ NLP/Transformers: AdamW is standard

General Tips:

- Always use momentum ($\beta = 0.9$ is a good default)
- Monitor both training loss and validation metric
- Learning rate is the most important hyperparameter

Key Takeaways

① Variance is the enemy of SGD

- Forces decreasing step size \rightarrow slow convergence
- Variance reduction restores linear convergence for finite-sum

② Momentum accelerates optimization

- NAG achieves optimal $O(1/t^2)$ rate for smooth convex
- Improves condition number dependence: $\kappa \rightarrow \sqrt{\kappa}$
- Critical for ill-conditioned problems

③ Adaptive methods are practical

- Adam combines best of both worlds
- AdamW is standard for transformers
- Inherently perform coordinate-wise preconditioning
- Trade theoretical guarantees for empirical robustness

Open Questions and Recent Advances

Theoretical Understanding:

- Why does Adam sometimes fail to converge? (AMSGrad fixes)
- Optimal variance reduction for non-convex?
- Lower bounds for adaptive methods?

Recent Advances:

- μp : Hypparameter transfer from small model to large models
- Layer-wise Adaptive Rate Scaling (LARS): large-batch training
- Muon: A new optimizer for LLM training with better stability and convergence
- Adafactor: memory-efficient Adam for transformers

Future Directions:

- Better understanding of Adam in non-convex settings
- Adaptive methods with convergence guarantees
- Efficient second-order methods

Further Reading

Variance Reduction:

- Johnson & Zhang (2013): SVRG - original paper
- Defazio et al. (2014): SAGA - single-loop variant
- Fang et al. (2018): SPIDER - non-convex optimal

Momentum:

- Nesterov (1983): Accelerated gradient method
- Polyak (1964): Heavy ball method
- Sutskever et al. (2013): Momentum in deep learning

Adaptive Methods:

- Duchi et al. (2011): AdaGrad
- Kingma & Ba (2015): Adam
- Loshchilov & Hutter (2019): AdamW and weight decay

Thank You!

Questions?