# Machine Learning and Optimization

## Lecture 13: Adaptive Gradient Descent Methods

CSCI 4140/6960 — Spring 2026

Course website: `cs.rpi.edu/~gittea/.../mlandopt.html`

# Learning Objectives

By the end of this lecture, you should be able to:

1. **Explain** why standard GD/SGD is sensitive to hyperparameters and fails on ill-conditioned problems
2. **Derive** the momentum, exponential averaging, and preconditioning updates and state what each fixes
3. **Trace** the progression from heavy ball $\rightarrow$ Nesterov $\rightarrow$ AdaGrad $\rightarrow$ RMSProp $\rightarrow$ Adam $\rightarrow$ AdamW, identifying the one design choice that changes at each step
4. **Classify** adaptive methods by their whitening type (element-wise vs. spectral) and state the practical implications

# Lecture Roadmap


Why Adaptive Methods?

A Concrete Problem: Ill-Conditioned Quadratics

Momentum Methods

Preconditioning

Adaptive Learning Rates: AdaGrad, RMSProp, Adam

AdamW: The Method Everyone Uses

The Big Picture: A Taxonomy of Adaptive Methods

Summary

# The Story So Far

We have two families of first-order methods:

**Gradient Descent:** cheap per step, exact gradients, converges at rate controlled by the condition number $\kappa = \beta/\mu$.

**SGD:** even cheaper per step (one sample), but noisy, and convergence is dominated by the noise floor $O(\sigma^2/T)$.

Both share two fundamental weaknesses:

1. They use a **single scalar stepsize** for all coordinates.
2. They **ignore curvature** — the Hessian never appears.

# Three Tools to Fix These Weaknesses

The adaptive methods in this lecture all combine three ideas:

1. **Momentum:** Keep moving in directions that historically made progress. *Fixes oscillation.*
2. **Exponential averaging:** Forget old information at a controlled rate. *Balances local and global information.*
3. **Preconditioning:** Use different stepsizes for different coordinates (or directions). *Models curvature cheaply.*

Every method we will see today is a specific combination of these three ingredients.

# First-Order Methods: Strengths and Limits

**Strengths:**

- Scalable: no Hessian, no linear system solves, low memory
- Easily randomizable: replace $\nabla f$ with a stochastic estimate
- Generally applicable: work without strong convexity or even differentiability

**Limits:**

- Sensitive to the stepsize $\alpha$ — too large diverges, too small stalls
- Convergence rate depends on $\kappa$, which can be enormous

**Central question:** Can we build first-order methods that are *less sensitive to hyperparameters* and *better model curvature*?

# Notation for This Lecture

Throughout this lecture, we use a unified notation:

- $f$: the objective function we want to minimize
- $g_t$: the **gradient estimate** used at step $t$
- In the **deterministic** setting: $g_t = \nabla f(x_t)$ (exact gradient)
- In the **stochastic/online** setting: $g_t = \nabla f_t(x_t)$, where $f_t$ is the loss on a sample (or an adversarially chosen function). The online setting will be introduced formally when we reach AdaGrad.

Unless otherwise stated, $\beta$ denotes the smoothness constant, $\mu$ the strong convexity constant, $\kappa = \beta/\mu$ the condition number, and $\sigma^2$ the variance bound $\mathbb{E}[\|g_t - \nabla f(x_t)\|^2] \leq \sigma^2$.

# An Ill-Conditioned Quadratic

Consider the simplest nontrivial example:

$$f(x) = \tfrac{1}{2}\|Dx\|_2^2 = \tfrac{1}{2}x_1^2 + 50\,x_2^2, \qquad D = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}$$

The Hessian is $D^2 = \text{diag}(1, 100)$. So $\mu = 1$, $\beta = 100$, and $\kappa = 100$.

The gradient is $\nabla f(x) = D^2 x = (x_1, 100\,x_2)^T$.

GD with stepsize $\alpha$ gives:

$$x_{t+1} = (I - \alpha D^2)\,x_t = \begin{pmatrix} (1-\alpha)^{t+1} & 0 \\ 0 & (1-100\alpha)^{t+1} \end{pmatrix} x_0$$

# The Stepsize Dilemma

For stability, we need $|1 - 100\alpha| < 1$, so $\alpha < 1/50$.

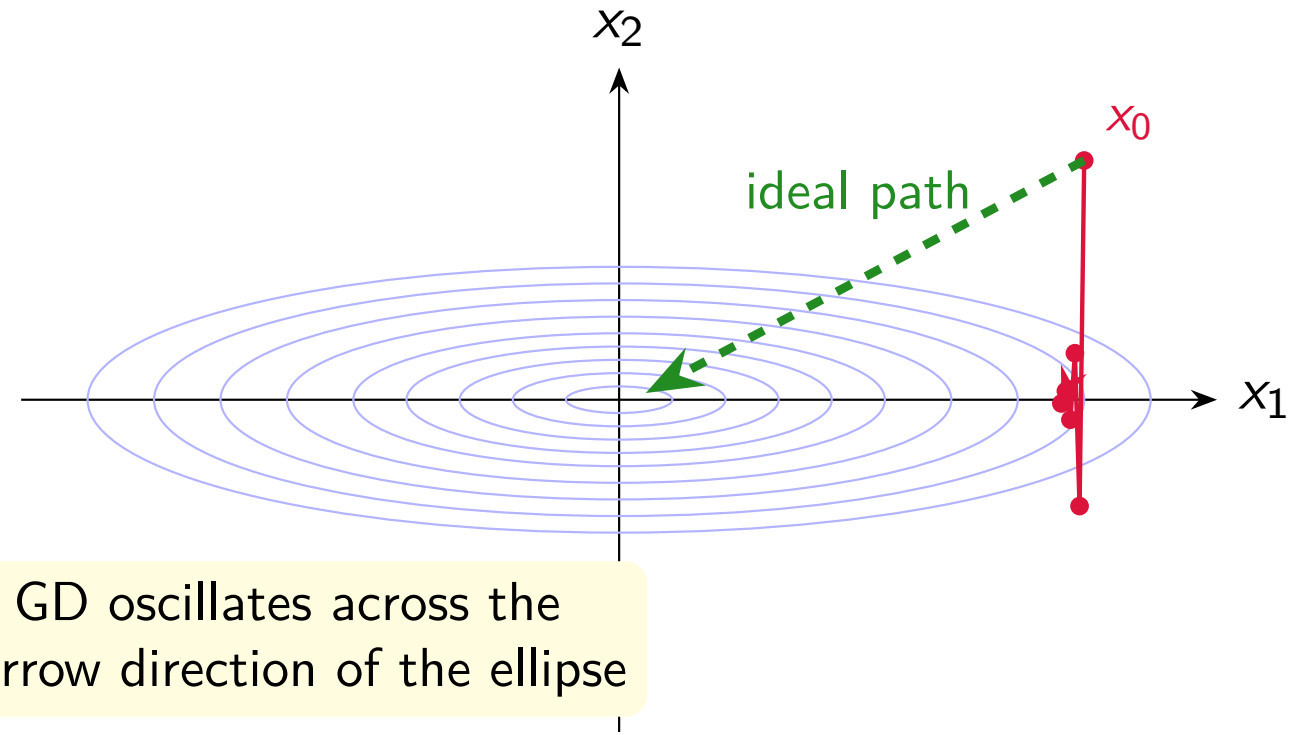
The safe choice is $\alpha = 1/\beta = 1/100$:

- $x_2$ direction: rate $(1 - 100 \cdot 1/100)^t = 0$ — converges instantly
- $x_1$ direction: rate $(1 - 1/100)^t = (99/100)^t$ — **very slow**

The problem: $x_1$ *wants* $\alpha = 1$, but $x_2$ *forces* $\alpha = 1/100$.

A single scalar stepsize cannot serve both coordinates well.

# Visualizing the Problem




GD oscillates across the narrow direction of the ellipse

# Three Lessons

**Lesson 1:** **Per-coordinate stepsizes** would help. Coordinate $x_1$ wants $\alpha = 1$; coordinate $x_2$ wants $\alpha = 1/100$.

**Lesson 2:** **Accounting for curvature** matters. The eigenvalues of $D^2$ control the convergence rate in each direction.

**Lesson 3:** **Dampening oscillations** is valuable. The zigzag pattern wastes most of the gradient signal.

The methods in this lecture address these lessons one at a time, then combine them.

# Historical Timeline

| Year | Method |
|---|---|
| 1964 | SGD with momentum (Polyak's heavy ball) |
| 1983 | Nesterov's accelerated gradient descent (NAG) |
| 2011 | AdaGrad (Duchi, Hazan, Singer) |
| 2012 | RMSProp (Hinton, unpublished lecture notes) |
| 2012 | AdaDelta (Zeiler) |
| 2015 | Adam (Kingma & Ba) |
| 2019 | AdamW (Loshchilov & Hutter) |

We will trace the design decisions connecting each method to the next.

# SGD with Momentum (Polyak 1964)

**Idea:** Dampen oscillations by continuing to move in historically useful directions.

**Update rules:**

$$\mu_t = \theta_t \, \mu_{t-1} + \alpha_t \, g_t$$
$$x_{t+1} = x_t - \mu_t$$

Here $\theta_t \approx 0.9$ is the **momentum parameter**, $\mu_0 = 0$, and $g_t$ is the gradient estimate at step $t$ (exact or stochastic).

This subsumes both GD ($g_t = \nabla f(x_t)$) and SGD ($g_t = \nabla f_t(x_t)$) as special cases.

# Momentum: Equivalent Viewpoint

Expanding the update:

$$x_{t+1} = \underbrace{x_t - \alpha_t g_t}_{\text{local model (GD/SGD step)}} + \underbrace{\theta_t(x_t - x_{t-1})}_{\text{history: keep moving}}$$

**The "heavy-ball" mechanism:**

- The first term makes local progress using the current gradient.
- The second term adds inertia: the algorithm resists sudden direction changes.

On the ill-conditioned quadratic, this dampens the zigzag in $x_2$ while accumulating consistent progress in $x_1$.

# Momentum as Exponential Averaging of Gradients

With constant $\theta_t = \theta$ and $\alpha_t = \alpha$, unroll the recursion:

$$\mu_t = \alpha \sum_{k=0}^{t} \theta^{t-k} g_k$$

The update direction is an **exponentially weighted average** of past gradients, with recent gradients weighted more heavily.

**Key observation:** This is the first appearance of exponential averaging — it will reappear in RMSProp and Adam.

# Convergence Improvement from Momentum

**For $\mu$-strongly convex, $\beta$-smooth objectives (deterministic, $g_t = \nabla f(x_t)$):**

| **Method** | **Linear Rate** | **Depends On** |
|---|---|---|
| GD | $\left(\frac{\kappa - 1}{\kappa + 1}\right)^t$ | $\kappa$ |
| GD + momentum | $\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^t$ | $\sqrt{\kappa}$ |

Optimal parameters: $\alpha^* = \frac{4}{(\sqrt{\beta} + \sqrt{\mu})^2}$, $\quad \theta^* = \left(\frac{\sqrt{\beta} - \sqrt{\mu}}{\sqrt{\beta} + \sqrt{\mu}}\right)^2$.

The dependence on $\sqrt{\kappa}$ instead of $\kappa$ is a dramatic improvement when $\kappa$ is large. Where do these come from?

# Why Does Momentum Accelerate? (Setup)

**Principle.** First-order methods induce *linear recurrences in the error*. Convergence is governed by the *spectral radius* of the recurrence operator.

**Linear dynamics.** If

$$e_{t+1} = Ae_t,$$

then

$$\|e_t\| \leq \|A\|_2^t \|e_0\|$$

**Interpretation.** Linear convergence $\iff \|A\|_2 < 1$. The slowest eigenmode sets the rate.

# Gradient Descent as a Linear Recurrence

Let $f$ be $\mu$-strongly convex and $\beta$-smooth. Define $e_t = x_t - x^*$. Then

$$e_{t+1} = x_{t+1} - x^* = (x_t - \alpha \nabla f(x_t)) - x^*$$

**Mean-value representation.**

$$\nabla f(x_t) = \nabla f(x_t) - \nabla f(x^*) = H_t e_t, \qquad H_t = \int_0^1 \nabla^2 f(x^* + s e_t)\, ds,$$

with $\mu I \preceq H_t \preceq \beta I$.

**GD recurrence.**

$$\begin{aligned} e_{t+1} &= (x_t - \alpha H_t e_t) - x^* \\ &= (x_t - x^*) - \alpha H_t e_t \\ &= (I - \alpha H_t) e_t. \end{aligned}$$

# Gradient Descent convergence rate

It follows that

$$\|e_t\|_2 \leq \prod_{s=1}^{t} \|I - \alpha H_s\|_2 \|e_0\|_2.$$

Using the fact that $H_t$ is the integral of Hessians all satisfying $\mu I \preceq \nabla^2 f(x) \preceq \beta I$,

$$\|e_t\|_2 \leq (\max_{\lambda \in [\mu, \beta]} |1 - \alpha\lambda|)^t \|e_0\|_2$$

We're maximizing a convex function over a compact convex set, so

$$\max_{\lambda \in [\mu, \beta]} |1 - \alpha\lambda| = \max\{|1 - \alpha\mu|, |1 - \alpha\beta|\}.$$

Choosing the stepsize $\alpha = \frac{1}{\beta}$, we see convergence occurs at the rate

$$\left|1 - \frac{\mu}{\beta}\right| = \left(1 - \frac{1}{\kappa}\right),$$

exactly as we expected.

# Momentum as a Second-Order Error Recurrence

Heavy ball update:

$$x_{t+1} = x_t - \alpha \nabla f(x_t) + \theta(x_t - x_{t-1}).$$

**Error form.**

$$e_{t+1} = (I - \alpha H_t + \theta I)e_t - \theta e_{t-1}.$$

**Matrix system.**

$$\begin{pmatrix} e_{t+1} \\ e_t \end{pmatrix} = A_t \begin{pmatrix} e_t \\ e_{t-1} \end{pmatrix}, \qquad A_t = \begin{pmatrix} (1+\theta)I - \alpha H_t & -\theta I \\ I & 0 \end{pmatrix}.$$

# Caveat: the recurrence matrices are not contractions!

We then have that

$$\|e_{t+1}\|_2 \leq \left\| \begin{pmatrix} e_{t+1} \\ e_t \end{pmatrix} \right\|_2 \leq \left\| \prod_{i=0}^{t} A_t \right\|_2 \|e_0\|_2.$$

However, now the recurrence matrices $A_i$ actually have norm greater than 1, so bounding the norm of the product of the matrices by $\prod \|A_t\|_2$ does not give a convergence rate.

Practically, this means GD + momentum is NOT a descent method. Despite global convergence, it's possible that $f(x_{k+1}) \geq f(x_k)$ at some finite times.

**Momentum breaks descent!**

# Acceleration = Minimax Spectral-Radius Design

Luckily, we find that a global convergence rate can be obtained of the form

$$\|e_t\|_2 \leq C\hat{\rho}^t\|e_0\|_2, \quad \hat{\rho} = \limsup_{k\to\infty} \|A_0 A_1 \dots A_k\|_2^{1/k}$$

for some $C > 0$ that accounts for the non-normality of the $A_t$. Now we want to choose the stepsize $\alpha$ *and* the momentum $\theta$ to minimize the *joint spectral radius* $\hat{\rho}$,

$$\min_{\alpha,\theta} \max_{\mu I \preceq H_t \preceq \beta I} \hat{\rho}.$$

**Sketch (Chebyshev equalization).** The maximum spectral radius is achieved at $\mu I$ or $\beta I$. Choose $(\alpha, \theta)$ so the worst-case contraction is *equalized at* $H_t = \mu I$ *and* $H_t = \mu\beta$. This yields the minimax convergence rate, and optimal parameters.

# SGD with Momentum: Convergence

When $g_t$ is a stochastic gradient ($\mathbb{E}[g_t \mid x_t] = \nabla f(x_t)$, $\mathbb{E}[\|g_t - \nabla f(x_t)\|^2] \le \sigma^2$), the noise changes the picture.

**Theorem** (convex, $\beta$-smooth). With $\alpha_t = O(1/\sqrt{T})$ and fixed $\theta < 1$:

$$\mathbb{E}[f(\bar{x}_T)] - f(x^*) \le O\left(\frac{\|x_0 - x^*\|^2}{\sqrt{T}} + \frac{\sigma}{\sqrt{T}}\right)$$

**Strongly convex refinement.** With $\mu$-SC and $\alpha_t = O(1/(\mu t))$: rate improves to $O(\sigma^2/(\mu T))$.

**Notice:** The stochastic rates are the *same order* as SGD without momentum. The noise floor $O(\sigma^2/T)$ dominates, so momentum primarily helps the deterministic component (dampening oscillations, improving constants) rather than changing the asymptotic rate. Its practical benefit is faster early convergence.

# Nesterov's Accelerated Gradient Descent (1983)

**Key idea:** Evaluate the gradient at a *predicted* future position, not the current one.

**Update rules:**

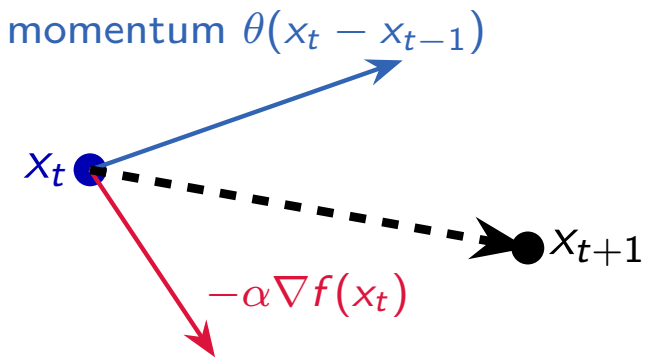
$$y_t = x_t + \theta_t(x_t - x_{t-1}) \quad \text{(lookahead point)}$$
$$x_{t+1} = y_t - \alpha_t \nabla f(y_t) \quad \text{(gradient step from lookahead)}$$

Equivalently:

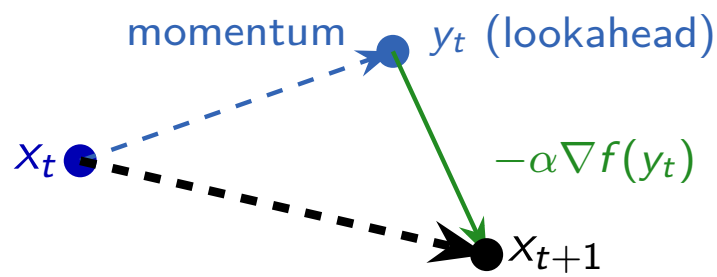
$$x_{t+1} = \underbrace{x_t + \theta_t(x_t - x_{t-1})}_{\text{predict using history}} - \underbrace{\alpha_t \nabla f(y_t)}_{\text{correct at predicted point}}$$

# NAG vs Heavy Ball: The Geometric Difference

**Heavy Ball**




**NAG**




**What changed:** NAG evaluates the gradient at where momentum is *about to take you*, not where you currently are. This provides a correction mechanism that prevents overshooting.

# NAG: Convergence Rates

| **Setting** | **NAG Rate** | **Compare to GD** |
|---|---|---|
| Convex, $\beta$-smooth | $f(x_T) - f^* = O\left(\frac{\beta D^2}{T^2}\right)$ | GD: $O(1/T)$ |
| $\mu$-SC, $\beta$-smooth | $\left(1 - \frac{1}{\sqrt{\kappa}}\right)^T$ | GD: $\left(1 - \frac{1}{\kappa}\right)^T$ |

The $O(1/T^2)$ rate for smooth convex functions is **optimal among all first-order methods** (Nesterov 2004).

**Important caveat:** The heavy ball method does **not** achieve $O(1/T^2)$ for general convex functions — it can even diverge (Lessard, Recht, Packard 2016). The lookahead correction is essential.

# Can We Accelerate Subgradient Methods?

**For nonsmooth convex functions** (e.g., $\ell_1$ regularization, hinge loss), the subgradient method achieves:

$$f(\bar{x}_T) - f^* = O\left(\frac{GD}{\sqrt{T}}\right)$$

where $G$ is a bound on subgradient norms and $D$ is the domain diameter.

**Can momentum improve this? No.**

Nesterov showed that $O(1/\sqrt{T})$ is the **optimal rate** for any first-order method on Lipschitz convex functions. Acceleration is impossible in this setting.

**Intuition:** Acceleration exploits *smoothness* — the gradient changes predictably, so the lookahead correction in NAG is informative. Without smoothness, the function can "kink" arbitrarily between two points, making any prediction useless. The bottleneck is information-theoretic, not algorithmic.

# Preconditioning: Change of Variables

**Setup:** We want to solve $x^* = \operatorname{argmin}_x f(x)$.

**Change of variables:** Let $x = Ru$ where $R$ is invertible. Then:

$$u^* = \operatorname*{argmin}_u f(Ru), \qquad x^* = Ru^*$$

Running GD in $u$-space and converting back:

$$\begin{aligned} & u_{t+1} = u_t - \alpha_t R^T \nabla f(Ru_t) \\ \Longrightarrow \quad & x_{t+1} = x_t - \alpha_t \underbrace{RR^T}_{=:P_t} \nabla f(x_t) \end{aligned}$$

The matrix $P_t = RR^T$ is the **preconditioner**. It rescales the gradient to account for geometry. Question: **How to choose a useful preconditioner?**

# What Is the "Right" Preconditioner?

If $R = [\nabla^2 f(x_t)]^{-1/2}$, then $P_t = [\nabla^2 f(x_t)]^{-1}$ and we recover **Newton's method**.

But Newton's method requires computing and inverting the Hessian: $O(d^3)$ per step.

**Dominant idea for adaptive methods:**
Use a **diagonal preconditioner** $P_t = \mathrm{diag}(p_1, \ldots, p_d)$ — cheap to compute and store, still captures per-coordinate scaling.

This gives per-coordinate stepsizes:

$$(x_{t+1})_j = (x_t)_j - \alpha_t \, p_j \cdot [g_t]_j$$

The question becomes: **how can we learn useful per-coordinate scalings from gradient observations along the trajectory?**

# Why Can Gradient History Reflect Curvature? (SGD Regime)

In ML, SGD samples data terms:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x), \qquad g_t = \nabla f_{i_t}(x_t).$$

**Gradient approximation near $x^*$.** With $H_i = \nabla^2 f_i(x^*)$ and $\nabla f(x^*) = 0$,

$$g_t \;\approx\; H_{i_t}(x_t - x^*), \qquad \nabla^2 f(x^*) = \mathbb{E}_i[H_i].$$

**Second moment of stochastic gradients.** Taking expectation over data sampling and iterate fluctuations,

$$\mathbb{E}[g_t g_t^T] \;\approx\; \mathbb{E}_i[H_i \,\mathrm{Cov}(x_t)\, H_i] \;\approx\; \nabla^2 f(x^*)\,\mathrm{Cov}(x_t)\,\nabla^2 f(x^*),$$

when the $H_i$ share geometry (e.g. $H_i = H + \Delta$ where $\Delta$ is mean zero and bounded covariance).

# The Geometry of the Iterate Cloud

**Balance of forces:** In the SGD steady state, iterates $x_t$ are pushed out by gradient noise $S$ and pulled back by the curvature $H = \nabla^2 f(x^*)$.

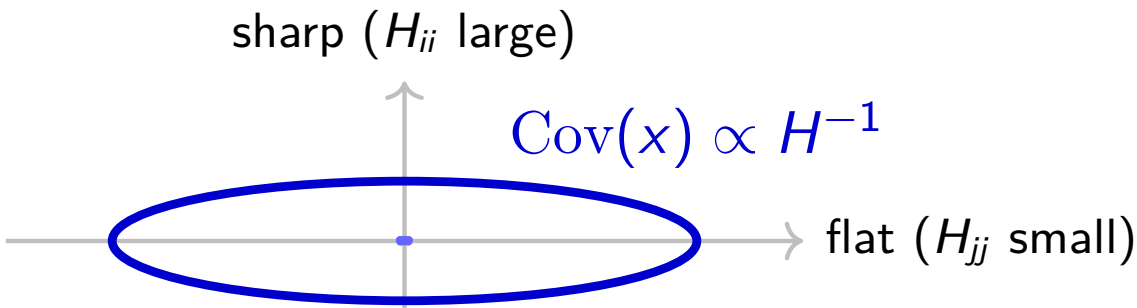
**The Stationary Distribution:** Under a constant step size $\alpha$, the iterates form a "cloud" around $x^*$ with covariance:

$$\mathrm{Cov}(x_t) \approx \frac{\alpha}{2} H^{-1} S H^{-1}$$

where $S$ is the gradient noise covariance.

**The $H^{-1}$ Law:** In many ML settings (e.g., Least Squares), the noise is itself shaped by the geometry, $S \approx \sigma^2 H$. This simplifies the cloud to:

$$\mathrm{Cov}(x_t) \propto H^{-1}$$



# Closing the Loop: Why $G \approx H$

We previously established that $\mathbb{E}[g_t g_t^T] \approx H \operatorname{Cov}(x_t) H$.
Since the iterate cloud is elongated exactly where the curvature is shallow ($H^{-1}$), the effects cancel:

$$\mathbb{E}[g_t g_t^T] \approx H \underbrace{(H^{-1})}_{\text{cloud}} H = H$$

# AdaGrad (Duchi, Hazan, Singer 2011)

**Idea:** Learn a different stepsize for each coordinate, using the history of observed gradients $g_1, \ldots, g_t$.

**Preconditioner:** Accumulate squared gradients per coordinate:

$$(D_t)_{jj} = \left( \sum_{i=1}^{t} g_i g_i^t \right)_{jj} = \sum_{i=1}^{t} [g_i]_j^2$$

**Update:**

$$x_{t+1} = x_t - \alpha (D_t + \varepsilon I)^{-1/2} g_t,$$

i.e.,

$$(x_{t+1})_j = (x_t)_j - \frac{\alpha}{\sqrt{(D_t)_{jj} + \varepsilon}} \cdot [g_t]_j$$

# AdaGrad: Two Key Phenomena

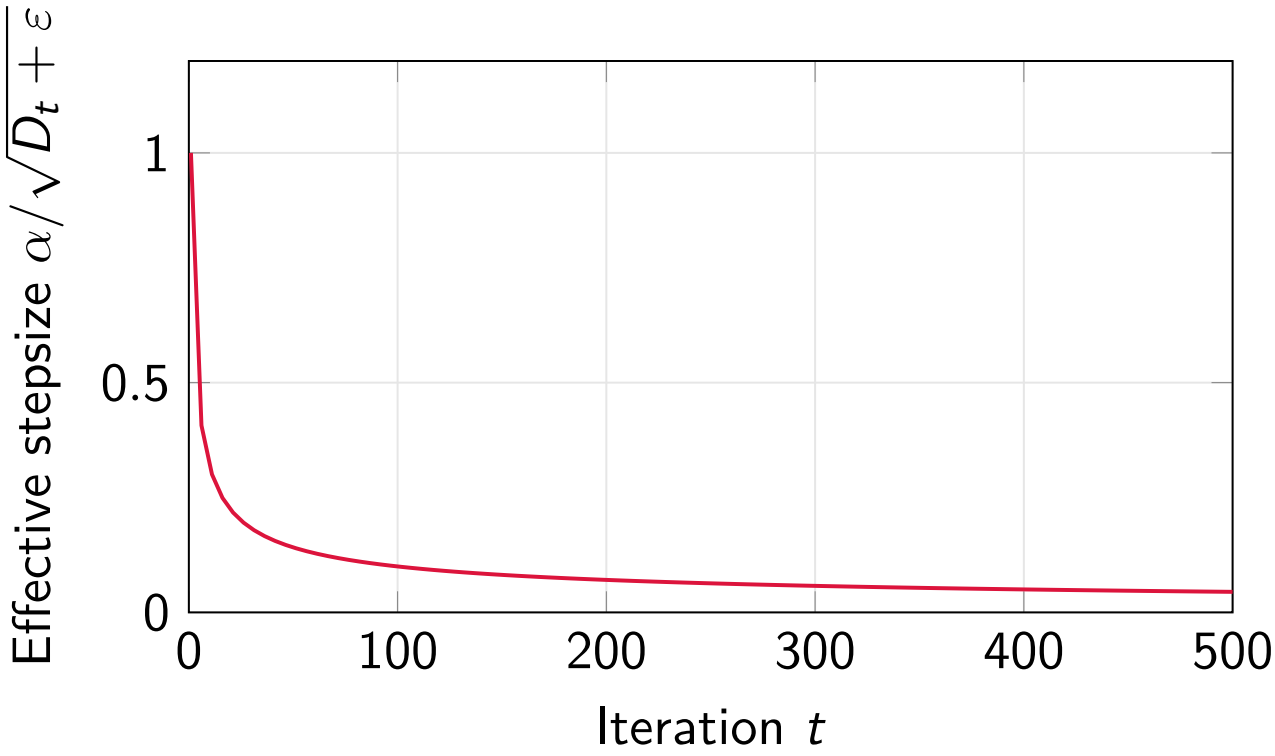
**Phenomenon 1:** All coordinates' learning rates **decrease** over time (the accumulated sum only grows).

**Phenomenon 2:** Coordinates with historically **large gradients** have learning rates that decay **faster**.

**Interpretation:** AdaGrad gives more aggressive steps to infrequent features (small accumulated gradient) and cautious steps to frequent features (large accumulated gradient).

This is especially useful for sparse data (NLP, recommendation systems).

# AdaGrad's Weakness




The effective stepsize decays as $O(1/\sqrt{t})$. In non-stationary settings or long training runs, this means AdaGrad **eventually stops learning**.

**The fix:** Use exponential averaging instead of cumulative summation.

# RMSProp (Hinton 2012)

**One change:** Replace the cumulative sum with an exponential moving average.

| | **AdaGrad** | **RMSProp** |
|---|---|---|
| $D_t$ | $D_{t-1} + \text{diag}(g_t g_t^T)$ | $\beta_2 D_{t-1} + (1-\beta_2)\,\text{diag}(g_t g_t^T)$ |

where $g_t$ is the gradient estimate at step $t$ and $\beta_2 \approx 0.999$.

**Update (same form):**

$$x_{t+1} = x_t - \alpha(D_t + \varepsilon I)^{-1/2}\, g_t$$

The exponential averaging means old gradient information is forgotten, so the effective stepsize no longer decays to zero.

# Adam (Kingma & Ba 2015)

**Two changes from RMSProp:**

1. Add momentum on the gradient (first moment estimate).
2. Add **bias correction** for the initialization.

**Update rules:**

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1)\, g_t && \text{(momentum)} \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2)\, g_t \odot g_t && \text{(preconditioner, per-coord)} \\
\hat{m}_t &= m_t / (1-\beta_1^t) && \text{(bias correction)} \\
\hat{v}_t &= v_t / (1-\beta_2^t) && \text{(bias correction)} \\
x_{t+1} &= x_t - \alpha\, \hat{v}_t^{\,-1/2} \odot \hat{m}_t
\end{aligned}
$$

# Adam: Defaults and Interpretation

**Standard defaults:** $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, $\alpha = 10^{-3}$.

**Why bias correction?**
At initialization ($m_0 = 0$, $v_0 = 0$), the exponential averages are biased toward zero. Since $\mathbb{E}[m_t] = (1 - \beta_1^t)\,\mathbb{E}[g_t]$, dividing by $(1 - \beta_1^t)$ corrects this.

**The genealogy:**



# Problem 1: Forgetting large gradients

## Adam Can Diverge (Reddi, Kale, Kumar 2018)

The original convergence proof of Adam (Kingma & Ba 2015) contains a bug. There exist simple convex online problems where Adam **diverges**.

**The issue:** The exponential moving average of $v_t$ can "forget" large past gradients, causing the effective stepsize to *increase* at bad times.

**The fix (AMSGrad):** Replace $\hat{v}_t$ with $\tilde{v}_t = \max(\tilde{v}_{t-1}, \hat{v}_t)$ element-wise. This ensures the effective stepsize is non-increasing, recovering AdaGrad-style guarantees.

In practice, Adam works well with $\beta_2 = 0.999$ (close to 1), which makes the "forgetting" issue negligible.

# Problem 2: L2 Regularization $\neq$ Weight Decay for Adam

For SGD, adding $\frac{\lambda}{2}\|x\|^2$ to the loss is equivalent to subtracting $\lambda x_t$ at each step (weight decay), up to rescaling by $\alpha$.

For **Adam, this equivalence breaks**.

With L2 regularization in Adam, the gradient of the penalty $\lambda x_t$ gets divided by the preconditioner $(D_t + \varepsilon I)^{-1/2}$:

$$\text{Adam + L2:} \quad x_{t+1} = x_t - \alpha\, \hat{v}_t^{-1/2} \odot (\hat{m}_t + \lambda x_t)$$

Parameters with large historical gradients are regularized **less** — the opposite of what we want.

# AdamW: Decoupled Weight Decay (Loshchilov & Hutter 2019)

**The fix:** Apply weight decay *directly* to the parameters, not through the preconditioner.

| | |
|---|---|
| **Adam + L2:** | $x_{t+1} = x_t - \alpha\, \hat{v}_t^{-1/2} \odot (\hat{m}_t + \lambda x_t)$ |
| **AdamW:** | $x_{t+1} = (1 - \lambda\alpha)\, x_t - \alpha\, \hat{v}_t^{-1/2} \odot \hat{m}_t$ |

This is a $\sim$1 line code change with profound consequences.

# Why AdamW Matters

1. The optimal weight decay $\lambda$ becomes **independent of the learning rate** $\alpha$. This makes hyperparameter search separable — a much smaller grid.

2. AdamW **closes the generalization gap** between Adam and SGD+momentum that had been observed empirically on vision tasks.

3. AdamW is the **de facto standard** for training transformers (BERT, GPT, ViT, etc.).

**The lesson:** Preconditioning should apply to the *loss gradient*, not to the *regularizer*. A small implementation detail with large theoretical and practical impact.

# Unifying Viewpoint: Whitening the Gradient

All adaptive methods can be viewed as approximately **whitening** the gradient: $g_t \mapsto P_t^{-1} g_t$, so the preconditioned gradient has roughly identity covariance.

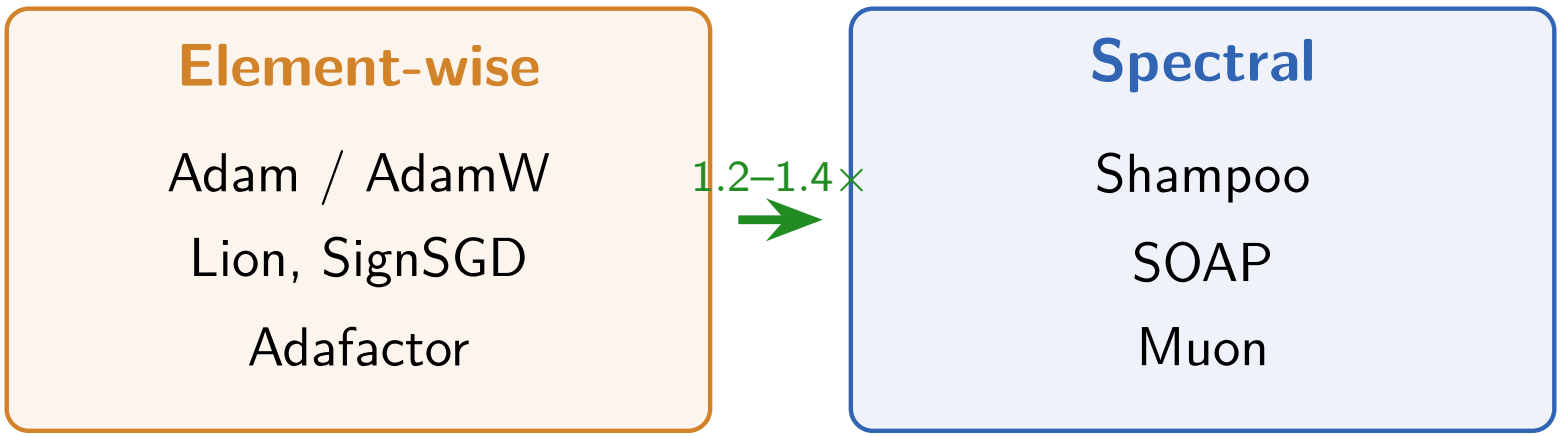
**What differs is the structure of $P_t$:**

**Element-wise** (diagonal): $P_t = \mathrm{diag}(\mathbb{E}[g\, g^T])^{1/2}$. Methods: Adam/AdamW, Lion, SignSGD, Adafactor. Memory: $O(d)$.

**Spectral** (full-matrix, Kronecker approx.): $P_t \approx (\mathbb{E}[g\, g^T])^{1/2}$. Methods: Shampoo, SOAP, Muon. Memory: $O(d^{3/2})$ or less.

**Key empirical finding** (Wen et al. 2025): Within each class, methods perform similarly when properly tuned. But spectral methods consistently beat element-wise methods by $\sim 1.2$–$1.4\times$ in tokens-to-target-loss for LLM pretraining.

# The Two Classes, Visually




The progression AdaGrad $\rightarrow$ RMSProp $\rightarrow$ Adam represents the refinement of *element-wise* whitening. The next frontier is *spectral* whitening — which is exactly what preconditioning (Section 4) was setting up.

**Caveat:** Hyperparameter tuning matters more than optimizer choice within a class. A well-tuned AdamW beats a poorly-tuned Shampoo.

# What We Learned

1. **Ill-conditioning** is the core problem: a single scalar stepsize cannot serve all directions. The condition number $\kappa$ governs convergence.

2. **Momentum** dampens oscillations and gives $\sqrt{\kappa}$-dependence via a second-order recurrence. The **lookahead** in NAG is essential for acceleration in the general convex case.

3. **Preconditioning** gives per-coordinate (or per-direction) stepsizes. Gradient outer products approximate Hessian information, enabling AdaGrad, RMSProp, and Adam.

4. **AdamW** decouples weight decay from the preconditioner — a small change with large practical impact.

5. **The whitening taxonomy** (element-wise vs. spectral) organizes the landscape of adaptive methods and predicts empirical performance.

# Questions?

Next time: neural networks