Optimization for Deep Learning

How gradients become reliable parameter updates

Prof. Nipun Batra

ES 667 · Deep Learning · IIT Gandhinagar

You already know the two ends of a training step

From backprop, you know how to compute every sensitivity

$$g_t = \nabla_\theta \mathcal{L}_{\mathcal{B}_t}(\theta_t).$$

From gradient descent, you know the simplest parameter update

$$\theta_{t+1} = \theta_t - \eta g_t.$$

Our first new idea—momentum—will remember recent gradients instead of trusting one minibatch at a time.

> Today's question is the missing middle: **what state should an optimizer remember, and how should that state shape the next step?**

Commit: one parameter is seen often; one is seen rarely

Q QUESTION

Two coordinates receive this minibatch-gradient history:

coordinate	g_1	g_2	g_3
dense d	4	4	4
rare r	0	0	0.4

At step 3, which information could tell us that r is rare?

A. only the current gradient g_3

B. the signs in g_1, g_2, g_3

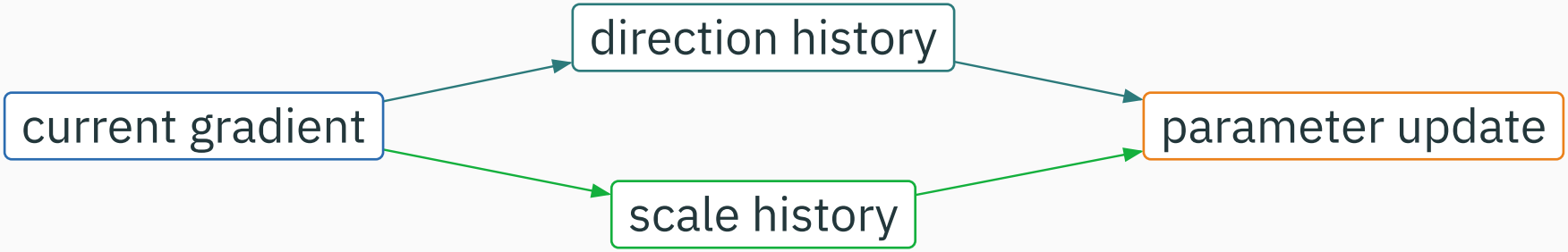
C. past squared gradients per coordinate

D. a larger global learning rate

Keep your choice; we will earn the answer

We start at $\theta_0 = (1, 1)$ and repeatedly reuse the same stream:

$$g_1 = (4, 0), \quad g_2 = (4, 0), \quad g_3 = (4, 0.4).$$



Every method today will be one new answer to: “what should the past change about this step?”

Outline: from gradient to update

V VISUAL

At every beat: identify the stored state, calculate one update, then test the predicted behaviour.

measured g_t direction m_t scale v_t or G_t applied $\Delta\theta_t$

The optimizer is the middle of the loop

Backprop measures; the optimizer acts

V VISUAL

Backprop · g_t

What would a tiny parameter change do to this minibatch loss?

Optimizer · $\Delta\theta_t$

Given this measurement and stored history, what change should we apply?

$\theta_{t+1} = \theta_t + \Delta\theta_t$ **— the gradient and the update are not the same object.**

A learning rate converts sensitivity into distance

For plain minibatch SGD,

$\Delta\theta_t = -\eta g_t.$

On our recurring stream, with $\eta = 0.1$:

step	g_t	$\Delta\theta_t$	θ_t
0	—	—	$(1, 1)$
1	$(4, 0)$	$(-0.4, 0)$	$(0.6, 1)$
2	$(4, 0)$	$(-0.4, 0)$	$(0.2, 1)$
3	$(4, 0.4)$	$(-0.4, -0.04)$	$(-0.2, 0.96)$

> One global η preserves the $10\times$ magnitude gap in g_3. It knows nothing about how often each coordinate has appeared.

Why the negative gradient is locally downhill

D DERIVATION

For a small proposed step Δ, the first-order model is

$$\mathcal{L}(\theta + \Delta) \approx \mathcal{L}(\theta) + g^\top \Delta.$$

Among steps with $\|\Delta\| \le r$, the smallest predicted change occurs at

$$\Delta^\star = -r \frac{g}{\|g\|}.$$

> The result is local and Euclidean: after moving we must measure a new gradient, and a change of parameterization changes what “steepest” means.

One quadratic makes learning-rate behaviour predictable

D DERIVATION

Let $\mathcal{L}(\theta) = (\theta - 3)^2$ and $e_t = \theta_t - 3$. Then

$$e_{t+1} = (1 - 2\eta)e_t.$$

$0 < \eta < 0.5$	$0.5 < \eta < 1$	$\eta > 1$
positive multiplier: approach from one side	negative multiplier: alternate and shrink	magnitude above one: diverge

The loss surface stays fixed; the discrete step can crawl, settle, oscillate, or diverge.

Predict before revealing the trajectory

Q QUESTION

For $\mathcal{L}(\theta) = (\theta - 3)^2$, which rate makes the signed error alternate while shrinking by half?

Solve $e_{t+1} = (1 - 2\eta)e_t$ mentally.

A. $\eta = 0.25$

B. $\eta = 0.50$

C. $\eta = 0.75$

D. $\eta = 1.25$

Answer: the multiplier must be -0.5

A ANSWER

$$1 - 2\eta = -0.5 \quad \rightarrow \quad \eta = 0.75.$$

$$e_0 \rightarrow -0.5e_0 \rightarrow 0.25e_0 \rightarrow -0.125e_0 \rightarrow \ldots$$

Correct: C — $\eta = 0.75$

Why: the sign creates alternation; the magnitude 0.5 creates geometric decay.

Follow along: see the multiplier become a path

I INTERACTIVE

INTERACTIVE

↗ colab.research.google.com/github/nipunbatra/dl-teaching/blob/master/notebooks/L05/01_gd_quadratic.ipynb

Before each run:

- predict the sign and magnitude of $1 - 2\eta_t$;
- calculate the next parameter value;
- inspect both signed error and loss;
- explain any disagreement between prediction and plot.

> Complete the fixed-rate section now. Return to the warmup–cosine section after schedules are introduced.

Momentum remembers direction

Minibatches make a useful gradient noisy

For a uniformly sampled minibatch $\mathcal{B}_t$,

$$g_t = \frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_t} \nabla \ell_{i(\theta_t)}, \quad \mathbb{E}[g_t \mid \theta_t] = \nabla \mathcal{L}(\theta_t).$$

One update

may point away from the full-batch downhill direction

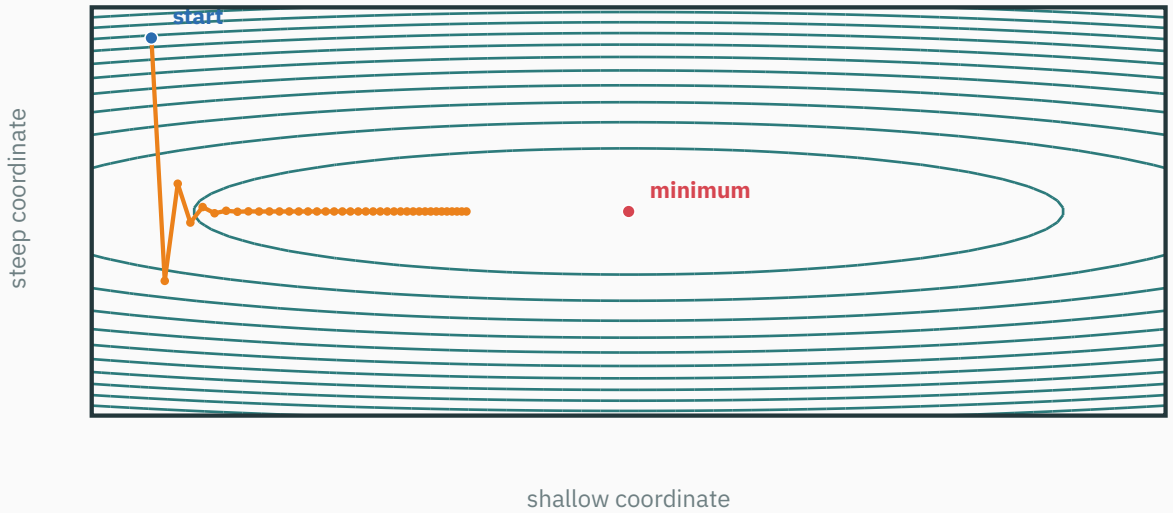
Repeated sampling

recovers the full gradient in expectation

> “Unbiased” is an average statement, not a promise that every minibatch step lowers training loss.

A ravine makes gradients alternate across one axis

V VISUAL

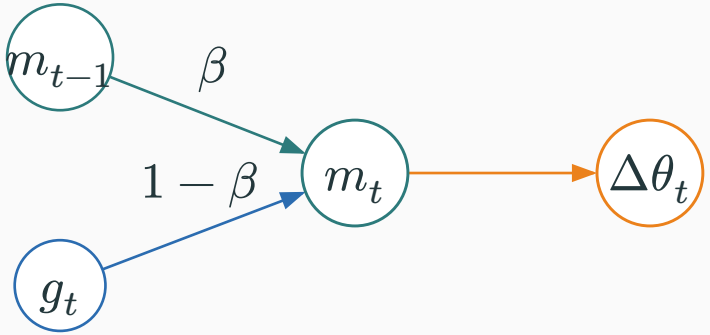


The plotted path is computed from the displayed loss. Stability along steep y forces slow progress along shallow x.

Momentum averages direction before stepping

Using the normalized moving-average convention,

$$m_t = \beta m_{t-1} + (1-\beta) g_t, \quad \Delta\theta_t = -\eta m_t.$$



Alternating components cancel; persistent components accumulate.

Work the memory before seeing its effect

D DERIVATION

Take $\beta = 0.9$, $m_0 = 0$, and one coordinate with gradients $10, 8, 11$.

t	g_t	$m_t = 0.9m_{t-1} + 0.1g_t$	m_t
1	10	$0.9(0) + 0.1(10)$	1.00
2	8	$0.9(1.00) + 0.1(8)$	1.70
3	11	$0.9(1.70) + 0.1(11)$	2.63

> The average starts low because its unobserved history is represented by zeros. This cold-start bias returns when Adam combines two moving averages.

Momentum changes direction, not coordinate scale

On our recurring stream, both coordinates use the same β:

$$m_t = \beta m_{t-1} + (1-\beta) g_t.$$

It can smooth the dense coordinate's direction, but when the rare coordinate finally appears, its small gradient is still small.

m_t remembers **sign and direction** $\neq$ a per-coordinate scale

To recognize rarity, the optimizer must remember magnitude separately for every coordinate.

AdaGrad remembers all squared gradients

Squared gradients form a per-coordinate activity ledger

AdaGrad accumulates

$$G_t = G_{t-1} + g_t^2$$

elementwise, then, with small $\varepsilon > 0$ preventing a zero denominator, updates

$$\Delta\theta_t = -\eta \frac{g_t}{\sqrt{G_t} + \varepsilon}.$$

Frequently active

$G_{t,j}$ grows; future steps shrink

Rarely active

$G_{t,j}$ stays small; a rare signal keeps leverage

AdaGrad turns gradient history into a separate effective rate for every coordinate.

Build the ledger for our two coordinates

D DERIVATION

$$g_1 = (4, 0), \quad g_2 = (4, 0), \quad g_3 = (4, 0.4).$$

t	g_t	$G_t = \sum_{\tau \le t} g_\tau^2$	$\frac{g_t}{\sqrt{G_t} + \varepsilon}$
1	$(4, 0)$	$(16, 0)$	$(1, 0)$
2	$(4, 0)$	$(32, 0)$	$(0.707, 0)$
3	$(4, 0.4)$	$(48, 0.16)$	$(0.577, 1)$

> At its first appearance, the rare coordinate receives normalized magnitude 1 even though its raw gradient is $10\times$ smaller.

AdaGrad answers the opening question

A ANSWER

With $\eta = 0.1$ and $\theta_0 = (1, 1)$:

step	dense update	rare update	θ_t
1	-0.100	0	$(0.900, 1)$
2	-0.071	0	$(0.829, 1)$
3	-0.058	-0.100	$(0.772, 0.900)$

Correct: C — past squared gradients per coordinate

Why: only a coordinate-wise history distinguishes “small because rare” from “small on every batch.”

The strength of AdaGrad becomes its failure

G_t only increases.

1 · Observe

more nonzero gradients

→

2 · Accumulate

larger $G_t = \sum g_t^2$

3 · Normalize

smaller effective rate $\frac{\eta}{\sqrt{G_t}}$

←

4 · Consequence

eventual near-freeze

> Old gradients keep equal voting rights forever. On a long nonstationary training run, useful step sizes can decay too aggressively.

RMSProp lets old scale evidence fade

Replace the sum with a moving average

RMSProp stores

$$v_t = \rho v_{t-1} + (1-\rho) g_t^2$$

and applies

$$\Delta\theta_t = -\eta \frac{g_t}{\sqrt{v_t} + \varepsilon}.$$

AdaGrad: all squared gradients $\rightarrow$ RMSProp: recent squared gradients

> The approximate memory window is $\frac{1}{1-\rho}$ steps: about 10 for $\rho = 0.9$, about 100 for $\rho = 0.99$.

Recompute step 3 with a fading ledger

D DERIVATION

For our stream, let $\rho = 0.9$ and $v_0 = (0, 0)$:

t	v_t	$\frac{g_t}{\sqrt{v_t}+\varepsilon}$
1	$(1.600, 0)$	$(3.162, 0)$
2	$(3.040, 0)$	$(2.294, 0)$
3	$(4.336, 0.016)$	$(1.921, 3.162)$

> The rare coordinate is still amplified relative to the dense one, but evidence now fades instead of accumulating forever.

A zero start creates an artificially small denominator

At the first nonzero gradient,

$$v_1 = (1-\rho)g_1^2, \quad \frac{g_1}{\sqrt{v_1}} = \frac{\operatorname{sign}(g_1)}{\sqrt{1-\rho}}.$$

For $\rho = 0.9$, the normalized magnitude is $\frac{1}{\sqrt{0.1}} \approx 3.16$.

> This is a cold-start effect, not evidence that the first gradient is exceptionally trustworthy. Adam will correct this missing moving-average mass explicitly.

What ε does—and does not do

RMSProp and Adam divide by $\sqrt{v_{t,j}} + \varepsilon$.

$\sqrt{v_{t,j}} gg\varepsilon$

ε has almost no effect

$\sqrt{v_{t,j}} \approx 0$

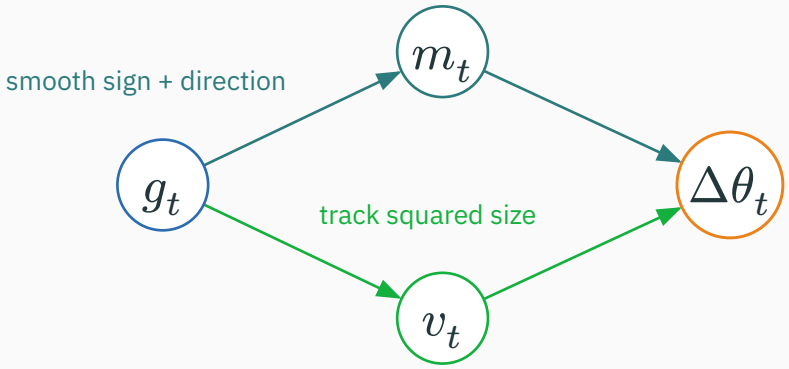
the denominator stays nonzero and the step finite

> ε is numerical protection, but it can also change effective rates in extremely low-variance coordinates. It is part of the optimizer definition, not decorative notation.

Adam combines direction and scale

Two memories solve two different problems

V VISUAL



m_t · **first moment**

smooth noisy signs and preserve persistent direction

v_t · **second raw moment**

normalize coordinate-wise gradient scale

Adam is momentum in the numerator and RMSProp-like scaling in the denominator.

Write Adam as state, correction, update

direction	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
scale	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
correct	$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$
update	$\Delta\theta_t = -\eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\varepsilon}$

measured g_t direction m_t scale v_t or G_t applied $\Delta\theta_t$

Why bias correction has exactly that denominator

D DERIVATION

If $g_t = g$ is constant and $m_0 = 0$, unrolling gives

$$m_t = (1 - \beta_1^t)g.$$

Similarly,

$$v_t = (1 - \beta_2^t)g^2.$$

Dividing by the missing mass restores the constant moments:

$$\hat{m}_t = g, \quad \hat{v}_t = g^2.$$

> The correction removes initialization bias under a constant-gradient thought experiment; it does not make a noisy gradient statistically unbiased.

Predict Adam's first update

Q QUESTION

Let $g_1 = (4, 0.4)$, $m_0 = v_0 = 0$, $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Ignoring ε, what is $\frac{\widehat{m}_1}{\sqrt{\widehat{v}_1}}$?

A. $(4, 0.4)$

B. $(1, 1)$

C. $(0.1, 0.001)$

D. $(0.4, 0.00016)$

Answer: correction restores both scales

A ANSWER

$$m_1 = (0.4, 0.04), \quad v_1 = (0.016, 0.00016).$$

$$\hat{m}_1 = (4, 0.4), \quad \hat{v}_1 = (16, 0.16).$$

$$\frac{\hat{m}_1}{\sqrt{\hat{v}_1}} = (1, 1).$$

Correct: B — $(1, 1)$

Why: on the first nonzero observation, bias-corrected Adam retains sign while normalizing coordinate magnitude.

Carry our recurring stream through Adam

D DERIVATION

For $g_1 = (4, 0), g_2 = (4, 0), g_3 = (4, 0.4)$:

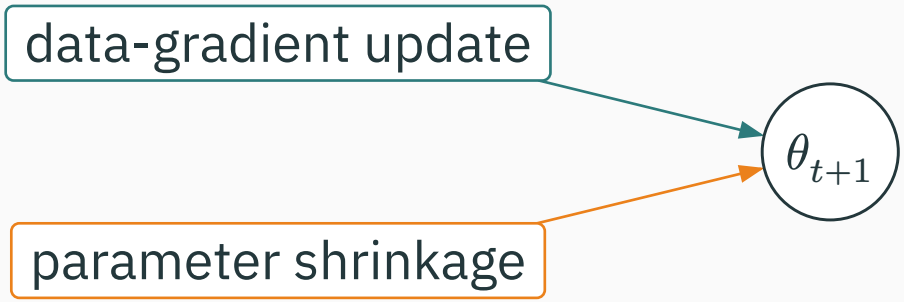
quantity at $t = 3$	dense d	rare r
m_3	1.084	0.040
$\hat{m}_3$	4.000	0.148
v_3	0.047952	0.000160
$\hat{v}_3$	16.000	0.05339
$\frac{\hat{m}_3}{\sqrt{\hat{v}_3}}$	1.000	0.639

> Scale normalization lifts the rare coordinate; direction memory tempers it because two previous batches contained zero evidence for that coordinate.

AdamW keeps weight decay out of the adaptive denominator

Adam supplies an adaptive data-gradient step. AdamW adds shrinkage separately:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} - \eta \lambda \theta_t.$$



> Decoupling makes the intended shrinkage independent of Adam's coordinate-wise rescaling. Bias parameters and normalization gains are commonly excluded, but the correct choice is model- and recipe-dependent.

Predict: should equal parameters shrink unequally?

Q QUESTION

Return to dense d and rare r. Suppose

$$\theta = (1, 1), \quad \hat{v} = (16, 0.16), \quad \eta = 0.01, \quad \lambda = 0.1.$$

Hold $\hat{v}$ fixed to isolate the geometry. If an L2 gradient $\lambda\theta$ is sent through Adam's adaptive denominator, its approximate contribution is

$$-\eta\lambda\frac{\theta}{\sqrt{\hat{v}}}.$$

How do the two coordinates' shrinkage contributions compare?

A. both are -0.001

B. dense shrinks $10\times$ more

C. rare shrinks $10\times$ more

D. both are -0.01

Answer: preconditioning changes the intended penalty

A ANSWER

L2 inside adaptive update	Decoupled AdamW
dense: $-0.01\frac{0.1}{4} = -0.00025$	dense: $-\eta\lambda\theta_d = -0.001$
rare: $-0.01\frac{0.1}{0.4} = -0.0025$	rare: $-\eta\lambda\theta_r = -0.001$

Correct: C — the rare coordinate shrinks $10\times$ more

Why: coupled L2 is filtered by the coordinate-wise preconditioner; AdamW applies the intended proportional shrinkage separately.

The coupled calculation holds $\hat{v}$ fixed for interpretation. In an actual coupled optimizer, adding $\lambda\theta$ also changes the moment estimates.

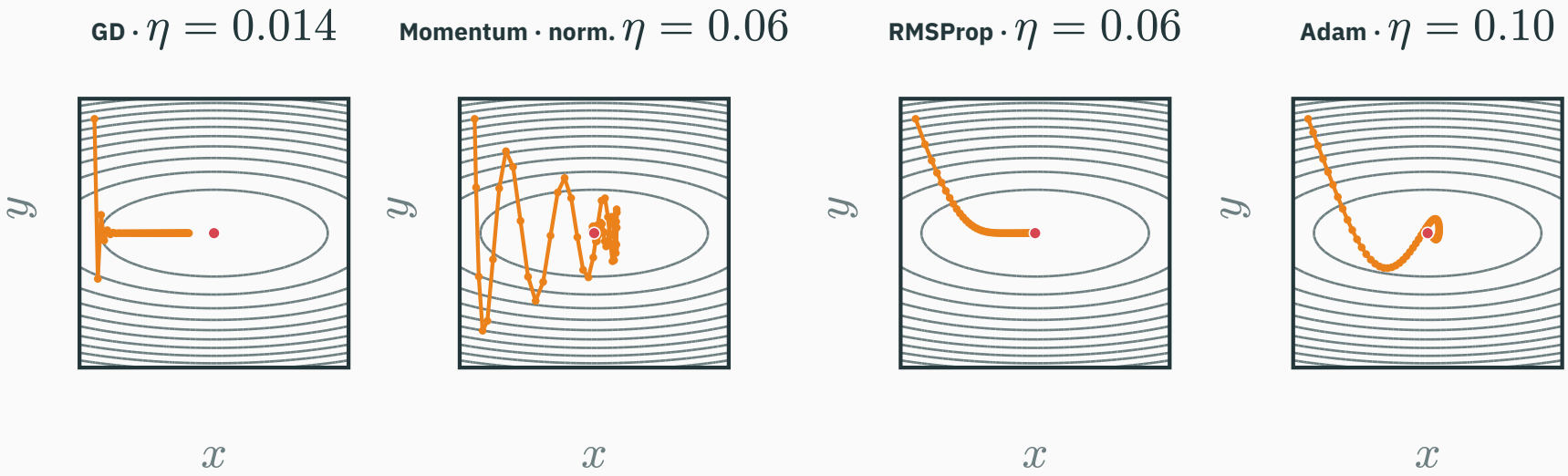
Compare mechanisms, not brand names

method	direction memory	scale memory	mechanism
SGD	—	—	current minibatch gradient
momentum	m_t	—	smooth direction
AdaGrad	—	$\sum g_t^2$	protect rare coordinates; rate only shrinks
RMSProp	—	EMA of g_t^2	adapt to recent scale
Adam / AdamW	m_t	v_t	combine both; optionally decouple decay

> There is no universally best optimizer. Any comparison also changes learning rate, schedule, update budget, batch size, and often regularization.

Same ravine, four explicitly different recipes

V VISUAL



Same loss and start. Momentum's normalized $\eta = 0.06$ equals buffer rate 0.006. Method-specific rates make this a mechanism illustration—not a ranking.

Starting recipes depend on the model regime

regime	reasonable first baseline	what to test next
CNN from scratch	established SGD + momentum or AdamW recipe	sweep rate/schedule; compare equal update budgets
Transformer / sparse model	established AdamW, normalization, and schedule recipe	test peak rate, warmup, decay, exclusions
pretrained fine-tune	task or adapter recipe; lower rate than pretraining	test forgetting and layer/adapter rates

> Starting hypotheses, not universal defaults: a validated architecture recipe plus a controlled rate sweep outranks a generic number.

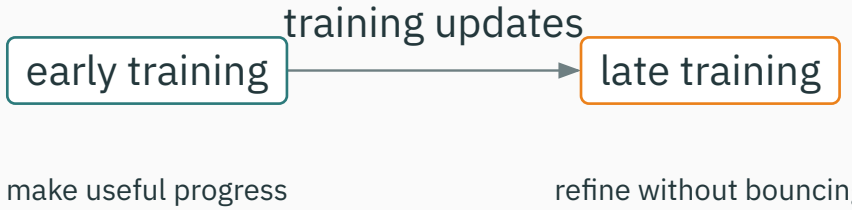
Make an optimizer comparison interpretable

discipline	record or test
record the complete recipe	optimizer, base rate, betas, ε, decay, schedule, batch, update count, seeds
inspect the mechanism	learning rate, gradient norm, update norm, train loss, validation metric
verify before scaling	one-batch overfit and a controlled learning-rate sweep

> Changing optimizer and learning-rate schedule together does not identify which change caused the result.

The schedule changes the step scale over time

A fixed rate asks one number to do two jobs

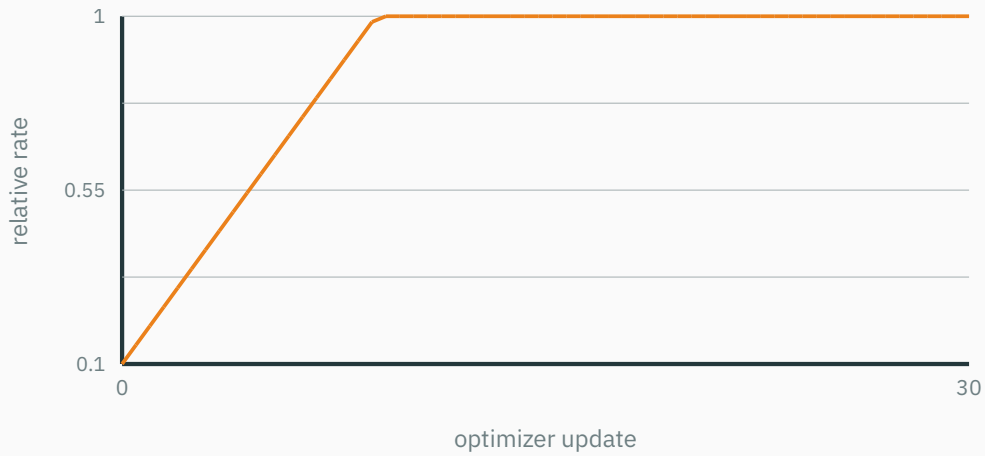


The optimizer chooses a direction and normalization; the schedule multiplies the overall step scale.

Warmup limits abrupt early updates

For T_w warmup updates, one simple convention is

$$\eta_t = \eta_{\max} \frac{t+1}{T_w}, \quad t = 0, \ldots, T_w - 1.$$

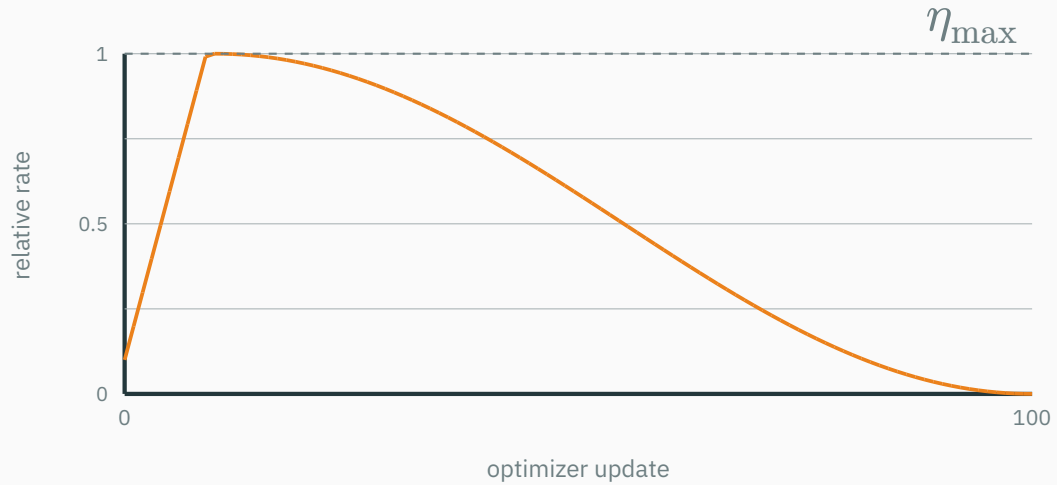


> Warmup reduces the initial applied scale; it does not repair a fundamentally unstable peak rate, bad data, or exploding activations.

Cosine decay turns progress into refinement

After warmup, let $u = \frac{t - T_w}{T - T_w}$ be progress from 0 to 1:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})(1 + \cos(\pi u)).$$



Warmup changes the beginning; cosine decay changes the remainder. Neither replaces optimizer state.

Calculate the clock before using it

D DERIVATION

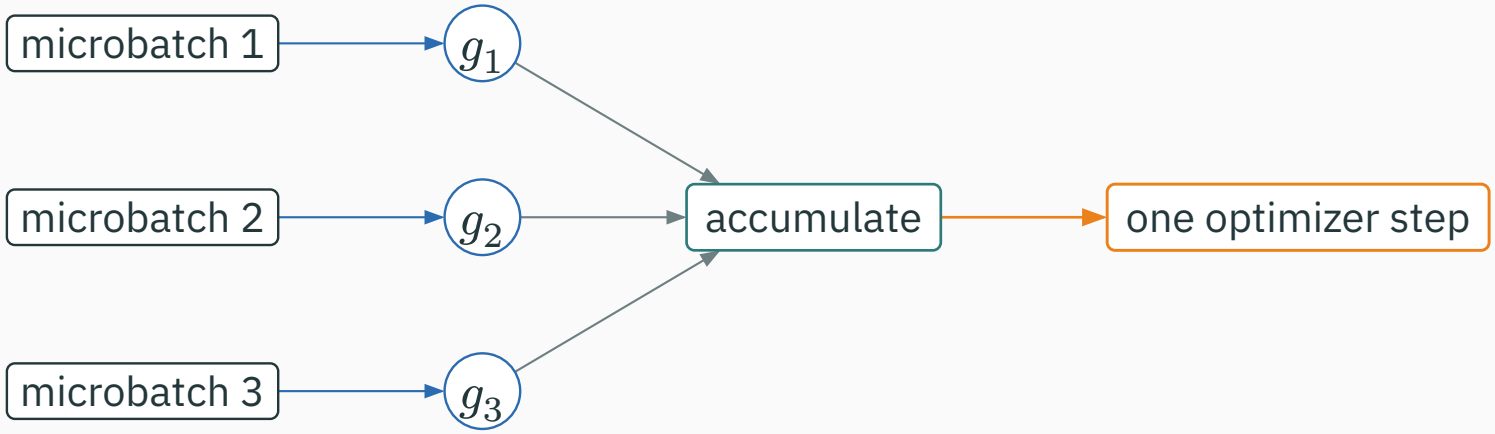
Let $\eta_{\max} = 0.1$, $\eta_{\min} = 0$, $T_w = 10$, and $T = 100$.

update	phase	rate
$t = 0$	first warmup update	$0.1\left(\frac{1}{10}\right) = 0.01$
$t = 4$	fifth warmup update	$0.1\left(\frac{5}{10}\right) = 0.05$
$t = 9$	last warmup update	0.10
$t = 55$	halfway through decay	0.05
$t = 100$	unexecuted endpoint	0

> With this convention, a 100-update run applies $\eta_0, \ldots, \eta_{99}$; $\eta_{100} = 0$ is the unexecuted endpoint. An off-by-one in warmup can silently change the first update from a small nonzero rate to no update at all.

The schedule must count the same update boundary as the optimizer

With gradient accumulation over K microbatches:



> Advancing an update-based scheduler on every microbatch makes its clock run $K\times$ too fast.

One reliable PyTorch update boundary

I INTERACTIVE

```
for x, y in loader:
    optimizer.zero_grad(set_to_none=True)
    loss = loss_fn(model(x), y)
    loss.backward()
    optimizer.step()
    scheduler.step()       # for an update-based schedule
```

Log gradient/update norms at this boundary; clip only under a stated diagnostic or stability policy.

Return to the quadratic: schedule and path together

I INTERACTIVE

INTERACTIVE

↗ colab.research.google.com/github/nipunbatra/dl-teaching/blob/master/notebooks/L05/01_gd_quadratic.ipynb

Now complete the warmup–cosine section:

- predict the time-varying multiplier $1 - 2\eta_t$;
- compare signed error and loss, not loss alone;
- test an overly large peak rate;
- distinguish transient growth from eventual recovery under decay.

Inspect optimizer state, not only a final loss

I INTERACTIVE

INTERACTIVE

↗ colab.research.google.com/github/nipunbatra/dl-teaching/blob/master/notebooks/L05/06_optimizer_comparison_pytorch.ipynb

Use the common ravine and start point to:

- calculate AdaGrad, RMSProp, and Adam state by hand;
- inspect m_t, v_t, and actual parameter updates separately;
- compare paths only with each method's rate visible;
- reproduce the mechanisms with deterministic PyTorch tensors.

What breaks: symptom → suspect → test

Read the symptom before changing the optimizer

symptom	first suspects	controlled test
loss becomes non-finite	rate, bad batch, overflow, exploding gradients	log batch ID, rate, gradient norm; rerun same batch
bounded zigzag	rate too high for a steep direction	lower only the rate; inspect signed updates
no progress	rate too low, dead path, no gradients, saturated units	overfit one batch; inspect per-layer gradient norms
train improves, validation worsens	generalization—not necessarily optimization	hold optimizer fixed; test regularization or data

Change one suspected cause at a time. “Try Adam” is an experiment, not a diagnosis.

Gradient clipping caps a dangerous update

D DERIVATION

Global-norm clipping replaces g by

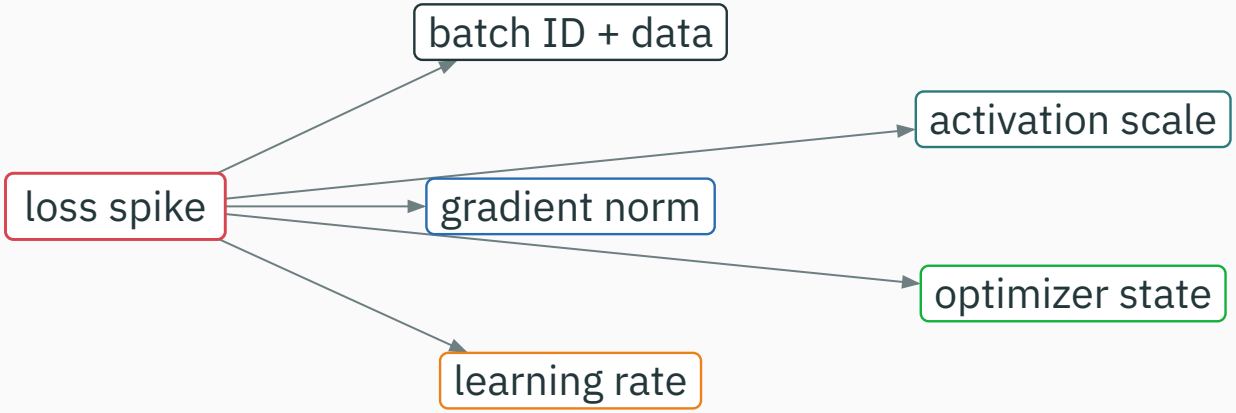
$$\tilde{g} = g \min\left(1, \frac{c}{\|g\|}\right).$$

For $g = (6, 8)$, $\|g\| = 10$. With $c = 5$:

$$\tilde{g} = (6, 8)\left(\frac{5}{10}\right) = (3, 4), \quad \|\tilde{g}\| = 5.$$

> Clipping preserves direction and caps magnitude. It can prevent a single catastrophic step; it does not make an unstable learning rate appropriate or repair chronic exploding gradients.

A loss spike deserves cause-relevant measurements



> For claims about generalization, use held-out performance. Raw “sharp versus flat” pictures depend on parameterization and do not establish generalization by themselves.

Checkpoint: choose the smallest informative intervention

Q QUESTION

A run is stable for 2,000 updates. The loss then spikes on one minibatch, gradient norm jumps $40\times$, and the current rate is unchanged.

What should you do first?

A. switch immediately from AdamW to SGD

B. rerun and inspect that batch plus gradient/activation scales

C. increase weight decay

D. restart with a larger batch and change nothing else

Answer: preserve evidence before changing the algorithm

A ANSWER

Correct: B — inspect and reproduce the triggering batch

Why: the correlated gradient-norm jump points to data or forward/backward scale; changing optimizer first destroys the cleanest evidence.

> If the batch is valid and the spike is reproducible, then test one intervention—rate, clipping threshold, normalization, or architecture—while holding the rest fixed.

Revisit, compress, hand off

Revisit the opening stream

$$g_1 = (4, 0), \quad g_2 = (4, 0), \quad g_3 = (4, 0.4).$$

method	what history says	step-3 consequence
SGD	nothing	raw $10\times$ scale gap remains
momentum	direction has been persistent for d	smooths signs; does not identify rarity
AdaGrad	d has accumulated much more squared activity	rare coordinate gets the larger normalized step
RMSProp	recent squared activity	same idea without permanent accumulation
Adam	direction and scale histories	rare signal is lifted, then tempered by its direction history

The opening answer was not “Adam.” It was the information Adam needs: coordinate-wise squared-gradient history.

Every optimizer answers three questions

Direction	Scale	Clock
current g_t or smoothed m_t?	global η or coordinate-wise v_t?	fixed, warmup, decay—and which updates count?

measure g_t $\rightarrow$ remember direction $+$ remember scale $\rightarrow$ apply a scheduled step

Practical exit ticket

Before trusting a training run, can you answer all five?

1. What exactly does backprop measure?

2. What state does the optimizer store?

3. What is each coordinate's effective scale?

4. Which event advances the schedule?

5. Which logged measurement would falsify your diagnosis?

Reproduce: seed, batch, update count, complete recipe.

If you cannot answer these, the loss curve is a symptom without a mechanism.

Primary references and provenance

- Duchi, Hazan & Singer (2011), **Adaptive Subgradient Methods for Online Learning and Stochastic Optimization**.
- Tieleman & Hinton (2012), **Lecture 6.5—RMSProp**, Neural Networks for Machine Learning.
- Kingma & Ba (2015), **Adam: A Method for Stochastic Optimization**.
- Loshchilov & Hutter (2019), **Decoupled Weight Decay Regularization**.
- Vaswani et al. (2017), **Attention Is All You Need**—a prominent warmup-and-decay recipe, not a claim that warmup is universally required.

> All trajectories in this deck are generated from the loss and update rules shown beside them. Notebook comparisons are deterministic mechanism tests, not benchmark claims.

Adam gives every parameter a history-dependent effective learning rate.

Next: make gradients and activations survive depth—initialization, normalization, and residual paths.