Adam, AdamW & LR Schedules

Lecture 5 · ES 667: Deep Learning

Prof. Nipun Batra
IIT Gandhinagar · Aug 2026

Learning outcomes

By the end of this lecture you will be able to:

1. Explain why **per-parameter** LR is helpful (sparse vs dense gradients).
2. Derive **Adam** as momentum + RMSProp + bias correction.
3. Compute **bias-correction** at small t and show why it matters.
4. Distinguish **Adam vs AdamW** and pick AdamW for regularized training.
5. Pick a **schedule** (constant / step / cosine / warmup+cosine) per use-case.
6. Explain **why warmup** is essential for Transformers.

Recap · where we left off

Lecture 4 · **momentum** = EMA of past gradients. Damps ravine oscillation and speeds training.

But momentum still uses a **single learning rate** for all parameters.

> REFERENCE
>
> Today maps to **UDL Ch 6** (Adam) and **Ch 7** (gradients + initialization revisited).

Q. Is that always the right thing?

Not always — here's why

Imagine training a model where:

- word-embedding parameters are updated by rare tokens → gradients are **large and sparse**
- hidden-layer weights are updated every step → gradients are **small but constant**

A single LR that is right for one is wrong for the other.

KEY IDEA

Today · **per-parameter adaptive learning rates** — AdaGrad → RMSProp → Adam → AdamW — plus the schedule we wrap around them.

Four questions

1. How do we get a **per-parameter** learning rate?
2. What is **Adam** actually doing, piece by piece?
3. Why is L2 "broken" inside Adam, and how does **AdamW** fix it?
4. What **schedule** should you use, and why do Transformers need warmup?

Pop quiz · which optimizer would you bet on?

You are training a 1B-parameter Transformer on text · gradients are sparse for embeddings, dense for attention, and large for a few outlier weights.

POP QUIZ

(a) SGD with momentum 0.9.
(b) AdaGrad.
(c) AdamW with warmup + cosine.
(d) Plain Adam, constant LR.

Stop and decide. By the end of today the answer should be obvious — and you'll know **why** each of the others fails.

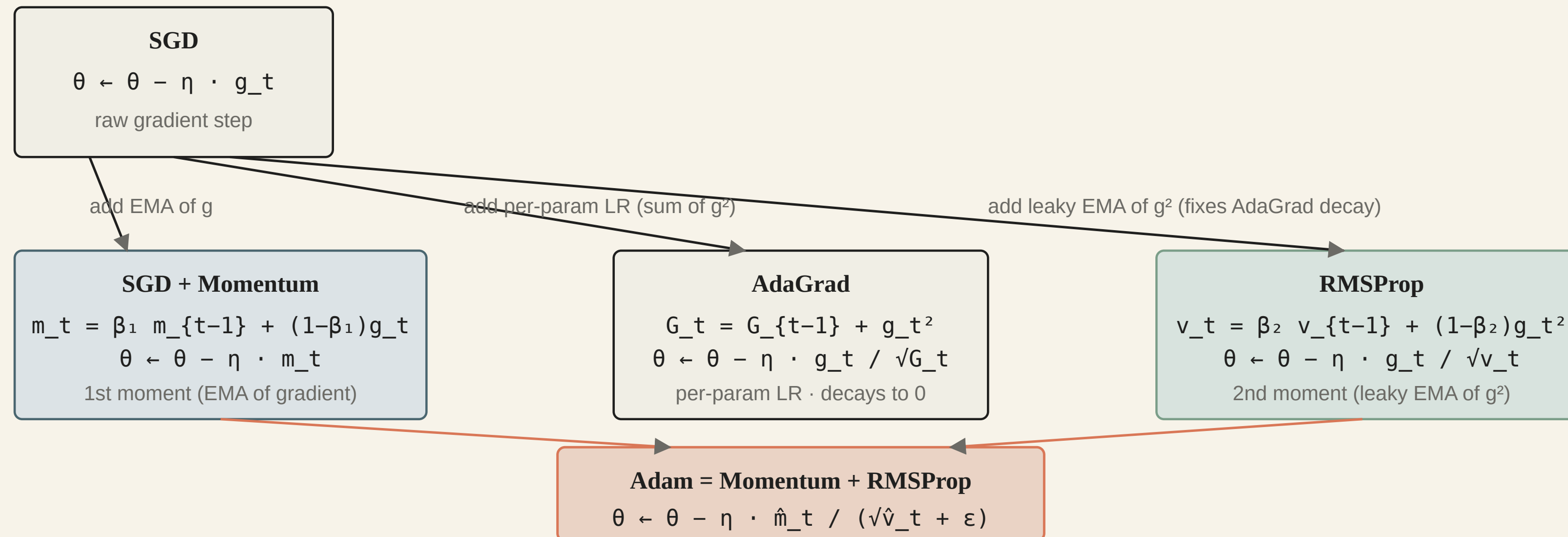
The exact same loss-curve shape would be diagnosed differently for each optimizer. That diagnostic is what this lecture trains.

PART 1

The family tree

SGD → AdaGrad → RMSProp → Adam

The lineage



Per-parameter LR · the two-knobs analogy

KEY IDEA

Imagine tuning two knobs · one is sensitive (you've already moved it a lot) · the other you've barely touched.

Which deserves a bigger turn? Obviously **the untouched one**.

AdaGrad does this for every parameter · if a parameter has accumulated lots of gradient (big knob movement so far), shrink its effective LR. If it's been quiet, leave its LR large. This is the core idea behind every adaptive optimizer (RMSProp, Adam, AdamW).

How do we give each parameter its own LR?

INTUITION

Analogy · audio mixing board. Imagine you're an engineer with a giant board of thousands of knobs (parameters).

- Some knobs are very sensitive — you've already moved them a lot. Make tiny adjustments now.
- Other knobs you've barely touched. You can afford to turn them aggressively.

AdaGrad's idea · keep a *total movement history* for each knob. The more a knob has moved, the smaller its future turns.

AdaGrad · build the update step-by-step

For a *single* parameter θ_i:

1. **Gradient** at step t for parameter i: $g_{t,i}$.
2. **Running sum of squared gradients** (the "history"):

$$G_{t,i} = G_{t-1,i} + g_{t,i}^2, \qquad G_{0,i} = 0$$

3. **Standard SGD** would do $\theta_{t,i} = \theta_{t-1,i} - \eta\, g_{t,i}$.
4. **AdaGrad** divides η by $\sqrt{G_{t,i} + \epsilon}$:

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}}\, g_{t,i}$$

$\epsilon = 10^{-8}$ avoids division-by-zero. Vector form (element-wise):

$$G_t = G_{t-1} + g_t^2, \qquad \theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}}\, g_t$$

Each parameter gets its **own** effective LR.

Worked numeric · AdaGrad on two parameters

θ_1 (dense, small gradients) and θ_2 (sparse, occasionally huge). $\eta = 0.1$.

θ_1 · steady $g = 0.2$

- $t = 1$: $G = 0.04 \Rightarrow \eta_{\text{eff}} = 0.1/\sqrt{0.04} = 0.5$
- $t = 2$: $G = 0.08 \Rightarrow \eta_{\text{eff}} \approx 0.354$
- $t = 10$: $G = 0.40 \Rightarrow \eta_{\text{eff}} \approx 0.158$

LR shrinks gently as updates accumulate.

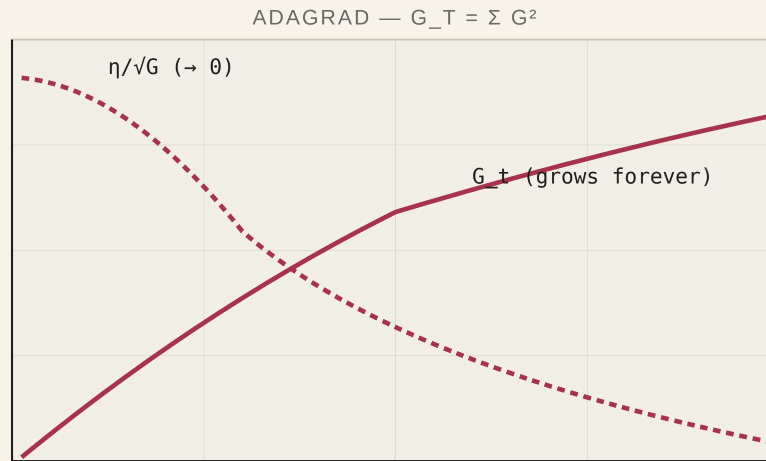
θ_2 · sparse: $g_1 \approx 0$, $g_2 = 5.0$

- $t = 1$: $G \approx 0 \Rightarrow$ tiny update (nothing to update toward)
- $t = 2$: $G = 25 \Rightarrow \eta_{\text{eff}} = 0.1/5 = 0.02$

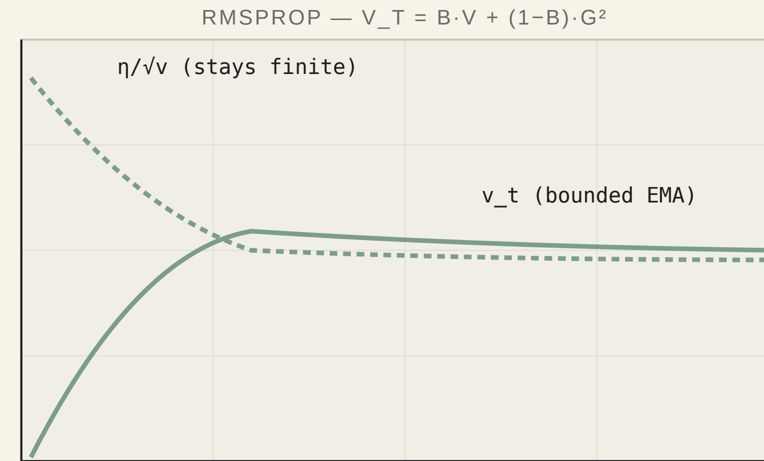
One big gradient → LR for θ_2 collapses immediately. Past sparsity protected; future moves are careful.

AdaGrad's problem · LR decays to zero

AdaGrad's flaw · G_t never forgets → effective LR decays to zero



training step → LR **dies** before convergence



training step → LR **lives** as long as gradient keeps flowing

*Hinton's insight · replace the sum with a **leaky EMA** — old squared gradients fade out.*

How do we fix AdaGrad's dying LR?

INTUITION

Analogy · perfect vs. fading memory. AdaGrad has *infinite memory* — every gradient since step 1 is in G_t. After a million steps, G_t is huge and the effective LR is essentially zero.

What if we gave it a **fading memory**, like momentum? An **EMA of squared gradients** weights recent gradients more than old ones. That's **RMSProp**.

RMSProp · AdaGrad with a fading memory (2012)

Replace AdaGrad's accumulator with an EMA v_t:

	ADAGRAD	RMSPROP
Update	$G_t = G_{t-1} + g_t^2$ (forever)	$v_t = \beta_2 \, v_{t-1} + (1 - \beta_2) \, g_t^2$
Behaviour	Grows without bound	Stabilizes

Update rule:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{v_t} + \epsilon} \, g_t$$

Typical $\beta_2 = 0.999$ — keep 99.9% of old, mix in 0.1% of new.

Worked numeric · AdaGrad vs. RMSProp

Constant gradient $g = 2.0$, $\eta = 0.1$, $\beta_2 = 0.9$ for clarity.

AdaGrad · keeps shrinking

- $t = 1$: $G = 4 \Rightarrow \eta_{\text{eff}} = 0.05$
- $t = 2$: $G = 8 \Rightarrow \eta_{\text{eff}} \approx 0.035$
- $t = 10$: $G = 40 \Rightarrow \eta_{\text{eff}} \approx 0.016$
- $t = 100$: $\eta_{\text{eff}} \to 0$

RMSProp · stabilizes

- $t = 1$: $v = 0.4 \Rightarrow \eta_{\text{eff}} \approx 0.158$
- $t = 2$: $v = 0.76 \Rightarrow \eta_{\text{eff}} \approx 0.115$
- $t \to \infty$: $v \to g^2 = 4 \Rightarrow \eta_{\text{eff}} = 0.05$

RMSProp's effective LR **converges to a positive value** rather than decaying to zero.

The big idea · Adam = Momentum + RMSProp

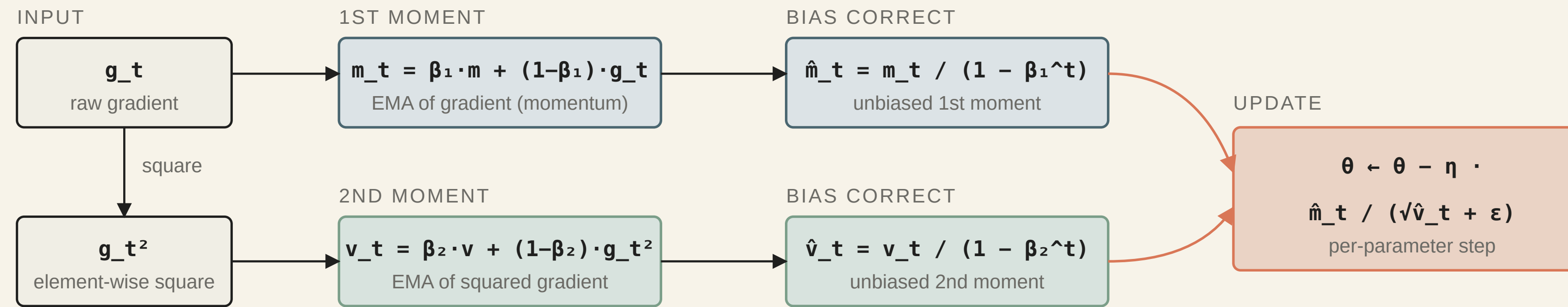
Combine the best of both worlds:

1. **From momentum** — EMA of gradients gives a smoother, less noisy direction. Call it m_t (first moment).
2. **From RMSProp** — EMA of *squared* gradients gives a per-parameter scale. Call it v_t (second moment).

Adam does both at once.

Adam · Momentum + RMSProp

Adam — four pieces wired together



DEFAULTS THAT WORK

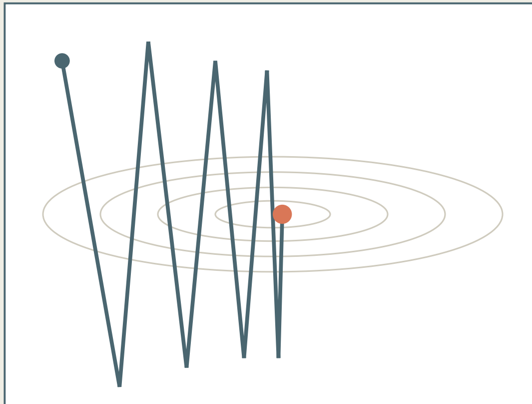
$\beta_1 = 0.9$ $\beta_2 = 0.999$ $\varepsilon = 1e\text{-}8$ $\eta = 1e\text{-}3$ (CNN) · 3e-4 (Transformer)

Kingma & Ba 2015 · "a single algorithm that works across a wide range of problems and hyperparameters"

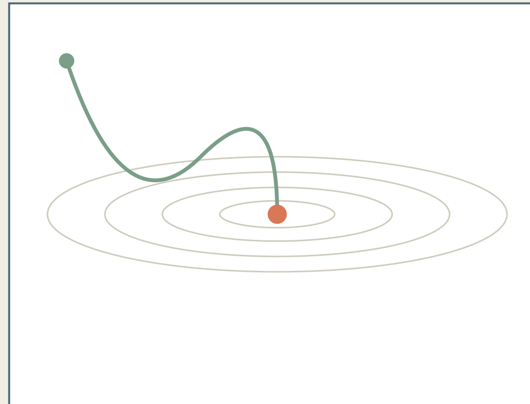
Trajectories · SGD vs momentum vs Adam

Optimizer trajectories on an ill-conditioned quadratic (κ=10)

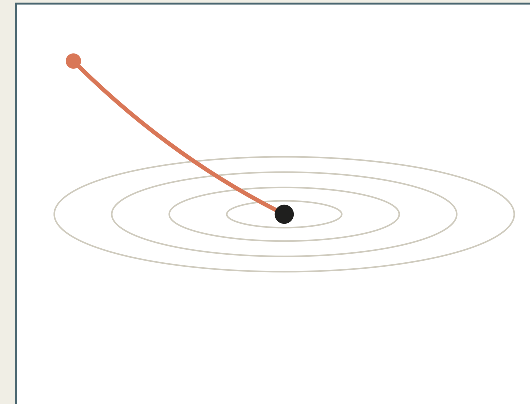
Vanilla SGD
zig-zags across ravine · 180 steps



SGD + Momentum
damped zig-zag · 60 steps



Adam / AdamW
near-direct · 30 steps



How to read

Each panel · same ill-conditioned quadratic $f(x,y) = 10\,x^2 + y^2$ · Hessian eigenvalues 10, 1 · κ=10.
Same LR for all three. SGD oscillates, momentum smooths, Adam rescales per-parameter → ~6× fewer steps.

Adam · the full update

DERIVATION

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t \qquad \text{(1st moment)}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 \qquad \text{(2nd moment)}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \qquad \text{(bias corr.)}$$

$$\theta_t = \theta_{t-1} - \eta\, \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Defaults · $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \eta = 10^{-3}$.

Adam · worked example · 3 steps on a single parameter

Suppose · gradients $g_1, g_2, g_3 = 2.0, 1.0, 1.5$. Defaults; $\theta_0 = 0$.

DERIVATION

STEP	m_t	$\hat{m}_t$	v_t	$\hat{v}_t$
1	$0.9 \cdot 0 + 0.1 \cdot 2.0 = 0.20$	$0.20/0.1 = 2.00$	$0.001 \cdot 4 = 0.004$	$0.004/0.001 = 4.0$
2	$0.9 \cdot 0.20 + 0.1 \cdot 1.0 = 0.28$	$0.28/0.19 = 1.47$	$0.999 \cdot 0.004 + 0.001 \cdot 1 = 0.005$	$0.005/0.001999 \approx 2.$
3	$0.9 \cdot 0.28 + 0.1 \cdot 1.5 = 0.402$	$0.402/0.271 = 1.48$	$0.999 \cdot 0.005 + 0.001 \cdot 2.25 = 0.0072$	≈ 2.4

Note · the per-parameter update size is **roughly constant ($\sim 10^{-3}$)** even though the gradients change. That's Adam's per-parameter adaptive scaling at work.

PART 2

The bias-correction detail

Why we divide by $(1 - \beta^t)$

The cold-start problem

Our moving averages m_t and v_t are **initialized at zero**.

INTUITION

Analogy · making hot chocolate. You start with a cup of hot water ($m_0 = 0$) and add one spoonful of cocoa (gradient g_1). The first sip is mostly water with a hint of cocoa — *biased toward the starting point*.

After many spoonfuls the mix is right. Bias correction is the math trick to "undilute" the early steps and get the right strength immediately.

What goes wrong at $t = 1$?

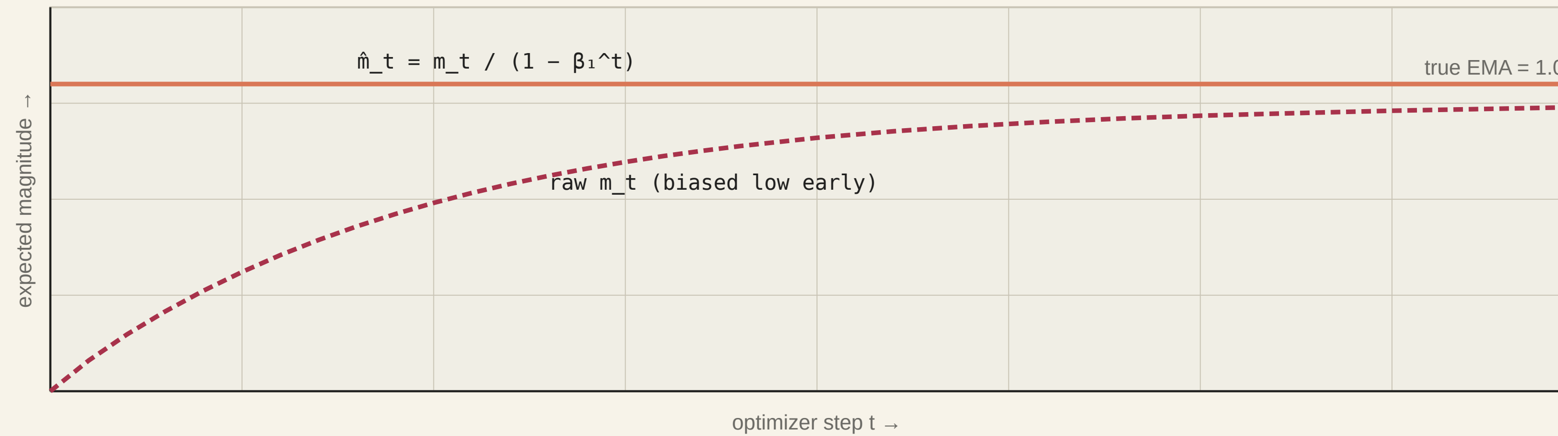
Initialize $m_0 = 0$. At step 1:

$$m_1 = \beta_1 \cdot 0 + (1 - \beta_1)\, g_1 = 0.1\, g_1$$

The EMA is **10× smaller** than the true gradient — purely because it started from zero.

The correction in one picture

Why Adam needs bias correction · the first few steps are not unbiased



Because moments are initialized at zero, the raw EMA is biased low for small t. The fix — divide by `(1 − β^t)` *— is Adam's bias correction.*

Deriving the bias-correction factor

Unroll the EMA from $m_0 = 0$, assuming the gradient is approximately constant $g_k = g$:

- $m_1 = (1 - \beta_1)\, g$
- $m_2 = \beta_1(1 - \beta_1)\, g + (1 - \beta_1)\, g$
- $m_3 = \beta_1^2(1 - \beta_1)\, g + \beta_1(1 - \beta_1)\, g + (1 - \beta_1)\, g$

In general:

$$m_t = (1 - \beta_1)\, g \left(\beta_1^{t-1} + \beta_1^{t-2} + \cdots + 1\right)$$

The bracketed sum is a **geometric series** with t terms:

$$\sum_{i=0}^{t-1} \beta_1^i = \frac{1 - \beta_1^t}{1 - \beta_1}$$

So $m_t = g\,(1 - \beta_1^t)$. To recover g, divide:

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

Worked example · bias correction in action

DERIVATION

Suppose the true gradient is $g = 1.0$ at every step. With $\beta_1 = 0.9$, starting $m_0 = 0$:

STEP t	$m_t = 0.9m_{t-1} + 0.1g$	$\hat{m}_t = m_t/(1 - 0.9^t)$
1	0.100	**1.000**
2	0.190	**1.000**
3	0.271	**1.000**
10	0.651	**1.000**
100	1.000	1.000

Without correction, the first step would use $m_1 = 0.1$ — ten times too small. Adam's step would therefore be **10× smaller than the optimal at t=1**, wasting the early training phase. The $(1 - \beta_1^t)$ denominator fixes it exactly.

When does bias correction matter?

For $\beta_1 = 0.9$:

- $t = 1$: factor $\frac{1}{1-0.9} = 10 \leftarrow$ huge
- $t = 10$: $\frac{1}{1-0.9^{10}} \approx 1.54$
- $t = 100$: $\approx 1.000026 \leftarrow$ negligible

Bias correction is a first-few-steps phenomenon. It keeps early steps the right magnitude; after that it fades to the identity.

PART 3

Adam → AdamW

The decoupled-weight-decay fix

L2 · quick recap from ES 654

L2 regularization adds $\frac{\lambda}{2}||\theta||^2$ to the loss. Gradient contribution: $\lambda\,\theta$.

In plain SGD, this equals *weight decay* — each step shrinks weights by a factor of $(1-\eta\lambda)$:

$$\theta_t = \theta_{t-1} - \eta\,\nabla\mathcal{L} - \eta\lambda\,\theta_{t-1}$$

For SGD these are the same. **Not so for Adam.**

AdamW · the fix

Adam vs AdamW · one-step worked numeric

Setup · $\theta = 1.0$, $g = 0.5$, $\lambda = 0.1$ (weight decay), $\eta = 10^{-3}$, $\hat{v} = 4.0$ (from RMSProp).

DERIVATION

Adam (L2 in gradient)

- Effective gradient · $g + \lambda\theta = 0.5 + 0.1 = 0.6$
- Update · $-\eta \cdot 0.6/\sqrt{4} = -0.0003$
- Note · the "regularization" $\lambda\theta$ got divided by $\sqrt{\hat{v}}$ · weakened on high-gradient params!

AdamW (decoupled)

- Adaptive update · $-\eta \cdot 0.5/\sqrt{4} = -0.00025$
- Plus uniform decay · $-\eta\lambda\theta = -0.0001$
- Total · -0.00035

In AdamW, weight decay is **uniform** for every parameter. In Adam, parameters with large past gradients are decayed less. AdamW's behavior matches the regularization theory · use it.

AdamW in PyTorch · one line

```
# the right default for almost everything in 2026
opt = torch.optim.AdamW(model.parameters(),
                        lr=3e-4,
                        betas=(0.9, 0.999),
                        weight_decay=0.1)   # typical for LLMs
```

IN PRACTICE

LLMs: `weight_decay=0.1`. Fine-tuning: `0.01-0.05`. Vision fine-tune: `0.001-0.01`.

PART 4

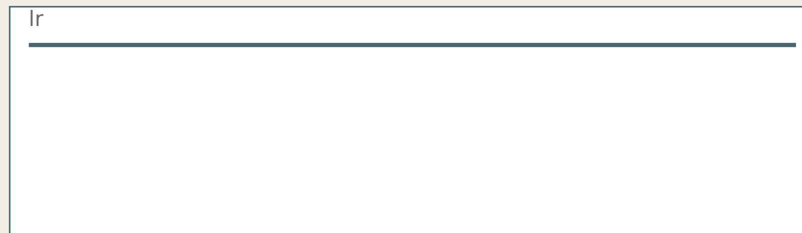
Learning-rate schedules

Why one learning rate isn't enough over a full run

LR schedules · four common shapes

Four learning-rate schedules · shape matters more than magnitude

Constant



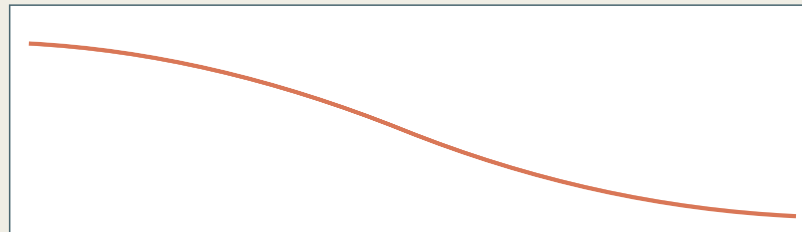
→ fast prototype · rarely final.

steps →

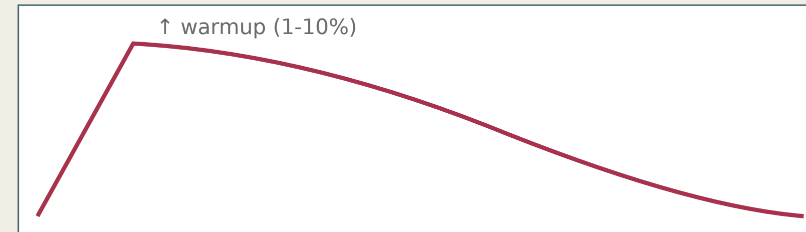
Step decay · milestones

→ classic CV recipe · multiply LR by 0.1 at fixed epochs.

Cosine annealing

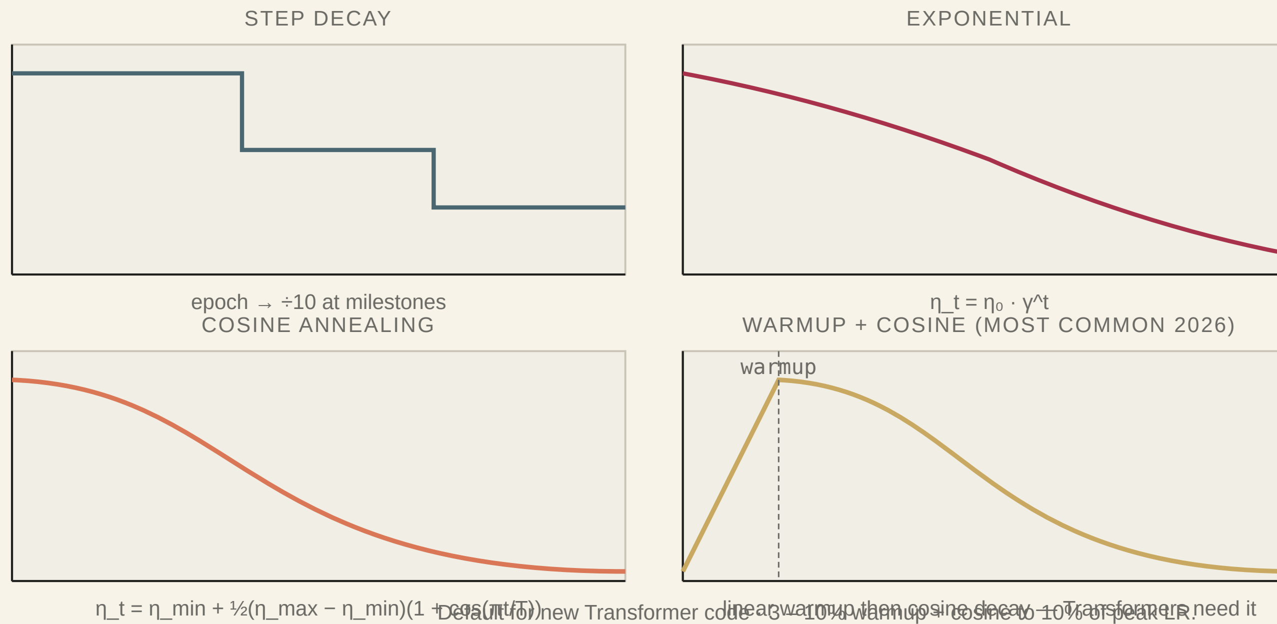


Warmup + cosine



2026 default · warmup+cosine for Transformers · cosine for CNN fine-tune · step-decay for CNN from scratch.

Four common schedules



IN PRACTICE

Interactive: sliders for peak LR and warmup — lr-schedule-visualizer.

Schedules in PyTorch

```python
# Step decay
sched = torch.optim.lr_scheduler.MultiStepLR(opt, milestones=[60, 120], gamma=0.1)

# Cosine annealing
sched = torch.optim.lr_scheduler.CosineAnnealingLR(opt, T_max=n_epochs)

# Warmup + cosine — the 2026 default for Transformers
from torch.optim.lr_scheduler import LambdaLR
def lr_lambda(step):
    if step < warmup: return step / warmup
    progress = (step - warmup) / (total_steps - warmup)
    return 0.5 * (1 + math.cos(math.pi * progress))
sched = LambdaLR(opt, lr_lambda)
```

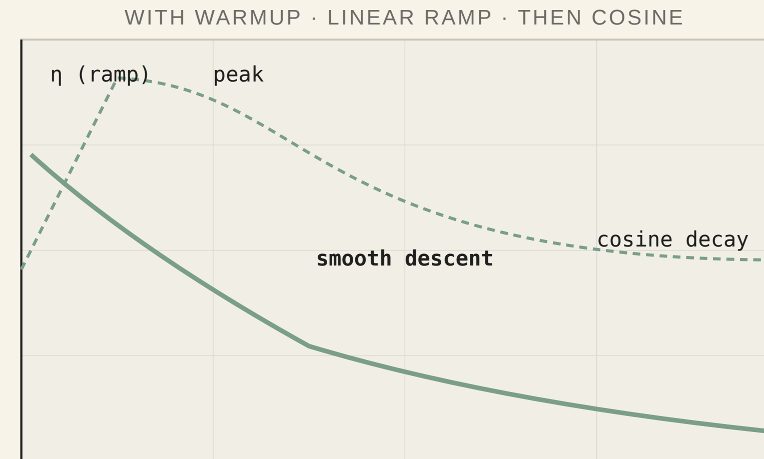
Why Transformers need warmup

Why Transformers need warmup — loss diverges without it

WITHOUT WARMUP · FULL LR FROM STEP 0

steps →

WITH WARMUP · LINEAR RAMP · THEN COSINE



steps →

Warmup = 1–10% of total steps · linear ramp from 0 to peak LR · then cosine decay to 10% of peak. Default for any Transformer.

Why are early gradients so chaotic?

A randomly-initialized network knows **nothing**.

- Loss is high → gradients are large.
- The model makes wildly overconfident-but-wrong predictions (e.g. softmax assigns 99% to the wrong class) → massive corrective gradients.

Two things go wrong **simultaneously** at step 1:

1. **Chaotic, large gradients** from a random network.
2. **Adam's v_t is itself noisy** — only one batch has been seen; the EMA estimate is unreliable.

Big gradient ÷ tiny, unreliable $\sqrt{v_t}$ → **explosive** first step that throws weights into unrecoverable territory.

Why Transformers need warmup

A perfect storm at the start:

1. **Adam's denominator is unstable.** $\hat{v}_t$ is based on a few batches; if those happen to be small, $\sqrt{\hat{v}_t}$ is tiny → updates are **huge**.
2. **Transformer gradients are spiky.** Random init → attention accidentally focuses everything on one irrelevant token → enormous gradient on that head's weights.

Combine:

$$\text{Update} \propto \frac{\text{large } g_t}{\text{tiny } \sqrt{\hat{v}_t}} \implies \text{EXPLOSION}$$

Warmup is a safety valve. Linearly ramp η from 0 over the first 1–10% of training:

- At step 1, $\eta \approx 0$ → tiny update no matter how crazy the gradient.
- This gives m_t, v_t time to **stabilize** over many batches.
- By the time η reaches its target, $\hat{v}_t$ is a reliable estimate.

After warmup, use cosine decay.

PART 5

What to actually use

Warmup · typical schedule

DERIVATION

Typical Transformer schedule:

- **Warmup** · linearly ramp lr 0 → target over first 2000 steps (or 1-10% of total).
- **Plateau** · hold at target lr briefly.
- **Cosine decay** · smooth drop from target to ~1/10 of target over the rest.
- **Final** · ~10% of training at minimum lr to squeeze last gains.

```
def lr_lambda(step, total, warmup):
    if step < warmup:
        return step / warmup        # linear warmup 0 → 1
    progress = (step - warmup) / (total - warmup)
    return 0.5 * (1 + math.cos(math.pi * progress))  # cosine
```

5 lines. Ubiquitous.

Defaults that work

MODEL / REGIME	OPTIMIZER	LR	SCHEDULE
CNN from scratch	**SGD + momentum + Nesterov**	0.1	step decay
CNN fine-tune	AdamW	1e-4	cosine
Transformer pre-train	**AdamW** (β_2 = 0.95)	3e-4	**warmup + cosine**
LoRA fine-tune of LLM	AdamW	1e-4 to 3e-4	cosine
Debugging a new idea	AdamW	3e-4	constant

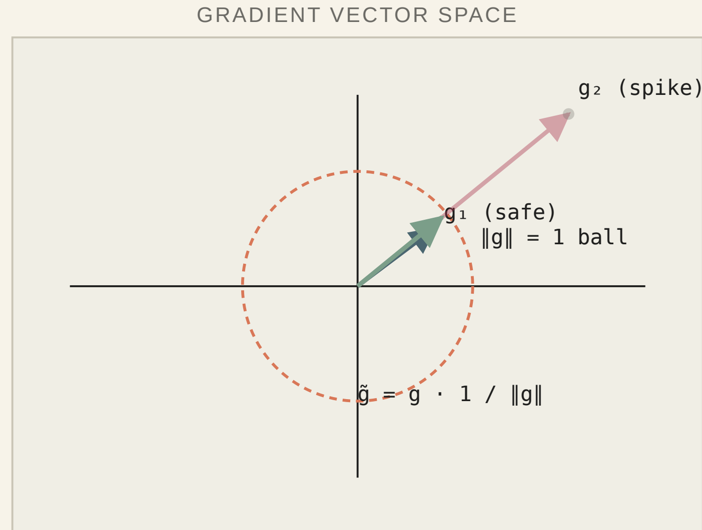
IN PRACTICE

`lr = 3e-4` is not magic — it's the number to use when you don't want to think. For a real run, do the **LR finder** (Lecture 3).

Gradient clipping · cheap insurance

Gradient clipping — project long gradients back onto the unit-norm ball

GRADIENT VECTOR SPACE



THE RULE

```
if ‖g‖ > max_norm:
g ← g · (max_norm / ‖g‖)
```

WHY THIS MATTERS

- Adam scales steps by $1/\sqrt{\hat{v}}$ — a single g^2 spike briefly produces a huge step.
- RNNs / Transformers often see gradient spikes from rare tokens or long contexts.
- Clipping is **cheap insurance** — costs nothing when gradients behave, saves the run when they don't.

LLMs: 1.0 · RNNs: 5.0 · CNNs: often unneeded

```
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0) # between loss.backward() and optim.step()
```

Common mistakes

WATCH OUT

Leaving `weight_decay` at 0 for AdamW. You get plain Adam with no regularization. Surprisingly common.

No LR schedule for an LLM. Training plateaus early and you blame the architecture.

Warmup of 10 steps for a 100k-step run. Far too short. Warmup = 1–10% of total.

`lr = 3e-4` for SGD. That's the Adam default. SGD usually wants `lr = 0.01` to `0.1`.

Putting it all together · the L05 master sentence

DERIVATION

Adam = momentum on g, RMSProp on g^2, with a bias-correction at the start. AdamW peels weight-decay out of the adaptive scaling so it actually regularizes. A **schedule** wraps everything · warmup at the start (gradients are chaotic), cosine at the end (anneal toward a minimum).

SYMBOL	ROLE	UPDATE AT STEP t
m_t	momentum	$\beta_1 m_{t-1} + (1-\beta_1) g_t$
v_t	per-param scale	$\beta_2 v_{t-1} + (1-\beta_2) g_t^2$
$\hat{m}_t, \hat{v}_t$	bias-corrected	$m_t/(1-\beta_1^t)$, $v_t/(1-\beta_2^t)$
θ_{t+1}	step	$\theta_t - \eta\,\hat{m}_t/(\sqrt{\hat{v}_t}+\epsilon) - \eta\,\lambda\,\theta_t$ (AdamW)

The final $-\eta\lambda\theta_t$ term — applied **directly to weights, not through** $\hat{v}$ — is the only difference between Adam+L2 and AdamW. That single change is why AdamW is the 2026 default for every Transformer in the world.

Pop quiz · revisit

The 1B-parameter Transformer? The answer is **(c) AdamW with warmup + cosine**.

KEY IDEA

(a) SGD · single LR can't cope with sparse vs dense gradients · slow.

(b) AdaGrad · LR collapses to zero on dense layers within 10k steps.

(c) ✓ AdamW + warmup absorbs the chaotic early gradients · cosine anneals.

(d) Plain Adam · L2 silently broken; no warmup → first-step explosions.

Practice problems

DERIVATION

P1. Show that AdaGrad's effective LR for parameter θ_i at step t is $\eta/\sqrt{\sum_{s=1}^{t} g_{s,i}^2}$. Argue why this monotonically decreases.

P2. A parameter has constant gradient $g = 1$. Compute Adam's $\hat{m}_t$ and $\hat{v}_t$ at $t = 1, 2, 10, 100$. Verify the bias-correction recovers $\hat{m}_1 = g$ and $\hat{v}_1 = g^2$.

P3. State the **exact** difference between Adam-with-L2 and AdamW. Show that for a fixed gradient and fixed $\hat{v}_t$ they give different updates.

P4. A Transformer is trained for $T = 100\text{k}$ steps. Sketch the cosine schedule with $T_{\text{warm}} = 2000$ warmup steps and peak LR $= 6 \times 10^{-4}$. Give the LR at $t = 0, 2000, 50000, 100000$.

P5. Why does Adam with $\beta_2 = 0.999$ need bias correction at $t = 1$ but not at $t = 10000$? Compute $1 - \beta_2^t$ for both.

P6. You ran AdamW with `weight_decay=0` for 50 epochs. Train loss is 0.001, val loss is 0.7. Name two changes from this lecture (not L06's regularization) that would help and why.

Summary · Lecture 5 — summary

- **AdaGrad** gave per-parameter LR — but G_t never forgets, so LRs decay to zero.
- **RMSProp** fixed that with EMA of g_t^2.
- **Adam = momentum + RMSProp + bias correction.** Robust first-try optimizer.
- **Bias correction** matters for the first ~10–100 steps; $\hat{m}_t = m_t/(1-\beta^t)$.
- **AdamW** decouples weight decay from adaptive scaling. Use it, not Adam+L2.
- **Schedules** — cosine is the clean default; **warmup + cosine** is the Transformer default.
- **Gradient clipping at 1.0** — cheap insurance.

Read before Lecture 6

Prince — Ch 6 §6.7 (Adam), Ch 7. Free at udlbook.github.io.

Next lecture

Regularization I — bias-variance in the overparameterized regime, double descent, weight decay as prior, early stopping, data augmentation, Mixup, label smoothing.

> NOTEBOOK
>
> **Notebook 5** · `05-adam-schedules.ipynb` — implement Adam and AdamW from scratch; sweep step-decay vs cosine on CIFAR-10.