LECTURE 7 CHEATSHEET

Momentum → RMSProp → Adam

Three ways to turn a noisy gradient into a useful update direction.

Momentum remembers direction. RMSProp remembers scale. Adam remembers both.

1 · Start with the optimizer contract

On minibatch $\mathcal{B}_t$, compute

$$\hat{\boldsymbol{g}}_t = \nabla_\theta \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta}_t).$$

Every optimizer constructs a direction $\boldsymbol{d}_t$, then applies

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \boldsymbol{d}_t.$$

The learning rate scales the move; the optimizer state changes its direction and coordinate-wise scale.

2 · EWMA is the shared memory primitive

$$m_t = \beta m_{t-1} + (1-\beta)x_t, \quad 0 \le \beta < 1.$$

Recent observations receive weights $(1-\beta), (1-\beta)\beta, \ldots$

Effective memory: roughly $\frac{1}{1-\beta}$ observations.
Zero-start bias: early m_t is too small. Correct with $\hat{m}_t = \frac{m_t}{1-\beta^t}$.

3 · Momentum: consistent direction?

$$\boldsymbol{m}_{t+1} = \beta_1 \boldsymbol{m}_t + (1-\beta_1)\hat{\boldsymbol{g}}_t$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \boldsymbol{m}_{t+1}.$$

Signed gradients that keep the same sign reinforce one another; oscillating components cancel. When $\beta_1 = 0$, this normalized convention becomes ordinary GD.

PyTorch SGD uses an unnormalized momentum buffer, so its `lr` convention differs by a factor of $(1-\beta_1)$ from the normalized rule above.

Ravine intuition

Momentum damps left-right reversals across steep walls while preserving progress along the shallow valley. Standardizing features may remove the ravine first.

4 · RMSProp: large relative to usual?

$$\boldsymbol{v}_{t+1} = \beta_2 \boldsymbol{v}_t + (1-\beta_2)\hat{\boldsymbol{g}}_t^2$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\hat{\boldsymbol{g}}_t}{\sqrt{\boldsymbol{v}_{t+1}} + \varepsilon}$$

Squaring prevents sign cancellation. Dividing by the recent root-mean-square gives each coordinate its own ruler.

$\boldsymbol{v}$ is a second raw moment, not a centred variance. Operations are elementwise.

5 · Adam: direction and scale

$$\boldsymbol{m}_{t+1} = \beta_1 \boldsymbol{m}_t + (1-\beta_1)\hat{\boldsymbol{g}}_t$$

$$\boldsymbol{v}_{t+1} = \beta_2 \boldsymbol{v}_t + (1-\beta_2)\hat{\boldsymbol{g}}_t^2$$

$$\widehat{\boldsymbol{m}}_{t+1} = \frac{\boldsymbol{m}_{t+1}}{1-\beta_1^{t+1}}, \quad \hat{\boldsymbol{v}}_{t+1} = \frac{\boldsymbol{v}_{t+1}}{1-\beta_2^{t+1}}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\widehat{\boldsymbol{m}}_{t+1}}{\sqrt{\hat{\boldsymbol{v}}_{t+1}} + \varepsilon}.$$

Bias correction matters most during the first updates, when both memories start at zero.

6 · What each method stores

Method	State	Question
GD	none	what is the gradient now?
Momentum	$\boldsymbol{m}$	which direction persists?
RMSProp	$\boldsymbol{v}$	which coordinates run large?
Adam	$\boldsymbol{m}, \boldsymbol{v}$	which direction, at what scale?

Adaptive does not mean parameter-free: tune the learning rate for the optimizer and problem.

LECTURE 7 CHEATSHEET

Use adaptive optimizers deliberately

State initialization, step order, diagnostics, and fair comparisons.

1 · One correct Adam step

1. compute the current minibatch gradient
2. update first-moment state $\boldsymbol{m}$
3. update second-moment state $\boldsymbol{v}$
4. bias-correct using update count $t + 1$
5. divide $\widehat{\boldsymbol{m}}$ by $\sqrt{\widehat{\boldsymbol{v}}} + \varepsilon$
6. apply the learning-rate-scaled parameter move

Initialize $\boldsymbol{m} = \mathbf{0}$ and $\boldsymbol{v} = \mathbf{0}$ once—not at every minibatch.

2 · Defaults are starting points

Knob	Typical	Effect
η	10^{-3} Adam	global step scale; tune first
β_1	0.9	longer signed-direction memory as it rises
β_2	0.999	longer squared-gradient memory as it rises
ε	10^{-8}	numerical floor; can affect scale in low precision

The useful learning-rate range depends on architecture, normalization, batch size, loss scale, and optimizer convention.

3 · PyTorch training order

```
opt.zero_grad(set_to_none=True)
pred = model(x)
loss = criterion(pred, y)
loss.backward()
opt.step()
```

Gradients accumulate unless cleared. The optimizer owns its per-parameter state; the model owns parameters and gradients.

AdamW note

AdamW applies weight decay as a separate shrink instead of mixing it into the adaptively scaled gradient. Treat the decay coefficient as a tuned regularization knob.

4 · Diagnose the symptom

Symptom	First checks
loss explodes / NaN	lower η; inspect data, loss scale, logits, and gradients
zigzag in a ravine	standardize; try momentum; reduce η
one coordinate dominates	check feature scale; consider RMSProp/Adam
no early progress	verify gradients/state; raise η carefully
good train, poor validation	this is generalization—not an optimizer-only problem

5 · Compare optimizers fairly

- ✓ Same data split, shuffling policy, initialization, and model.
- ✓ Same update budget and logging frequency.
- ✓ Tune each optimizer's learning rate separately.
- ✓ Report validation performance, not only training loss.
- ✓ Run multiple seeds when differences are small.
- ✓ Log gradient norms, update norms, and optimizer state.

6 · Invariants worth asserting

Shapes: θ, g, m, and v match elementwise.
Finiteness: loss, gradients, and states contain no `NaN`/`Inf`.
Clock: bias correction uses the number of parameter updates.
State: checkpoint optimizer state when resuming training.

Optimizers change the path—not the objective. Always inspect both training dynamics and validation behaviour.