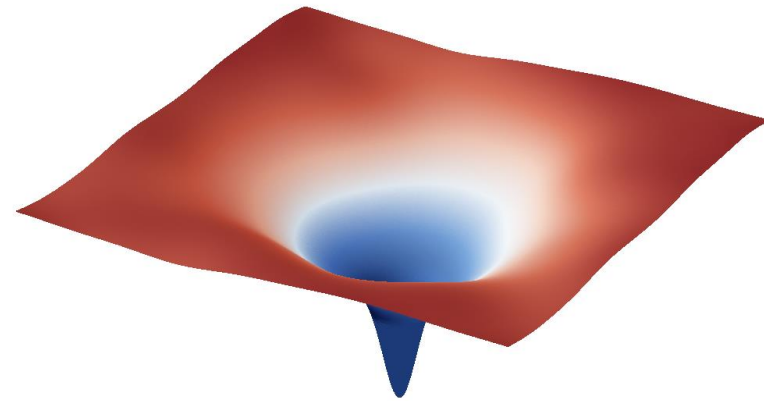
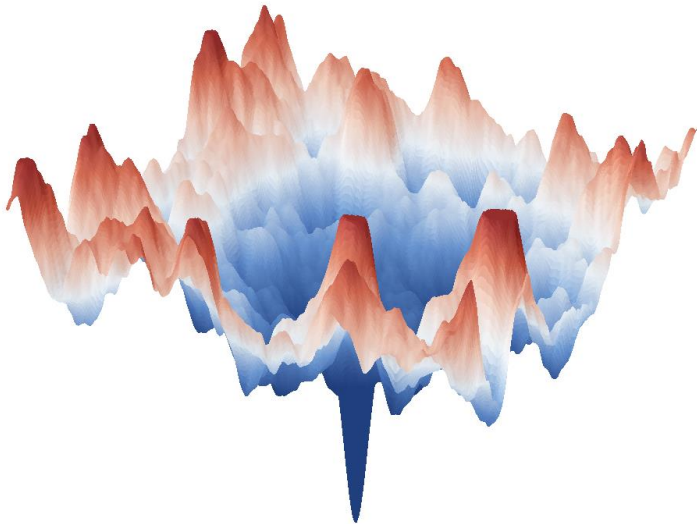


Deep Learning Optimization

Mohammad Salameh and Pat Virtue



Functions and the Objective

A model has parameters θ . The **objective** $J(\theta)$ is a single number measuring how badly the model does with those parameters, usually an average over the N training examples:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^N J^{(i)}(\theta)$$

Training means finding parameters that make $J(\theta)$ as small as possible

The derivative: slope with one parameter

$$J(\theta) = \frac{1}{2}(\theta - 1)^2 + \frac{1}{2}$$

How to read a derivative:

Sign = direction

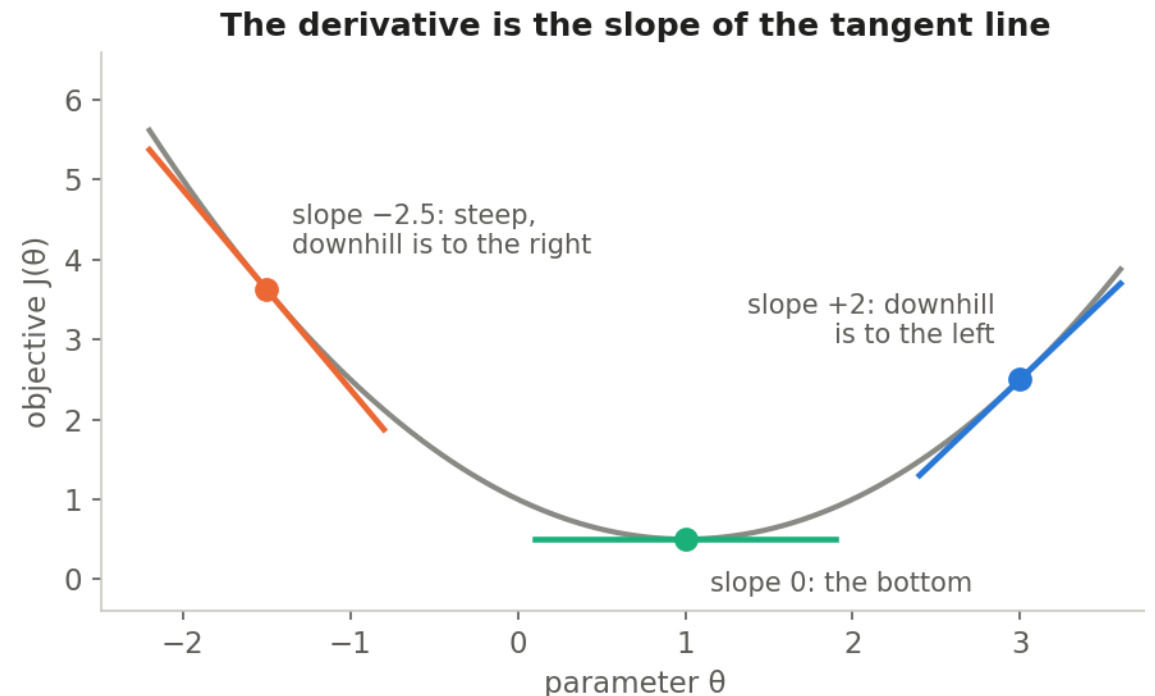
- Negative slope: the J goes *down* as θ increases, so move right.
- Positive slope: move left.

Size = steepness

A slope of -2.5 is steeper than -0.5 .

Zero = flat

At the bottom of the bowl the slope is 0; that is how we recognize a minimum.



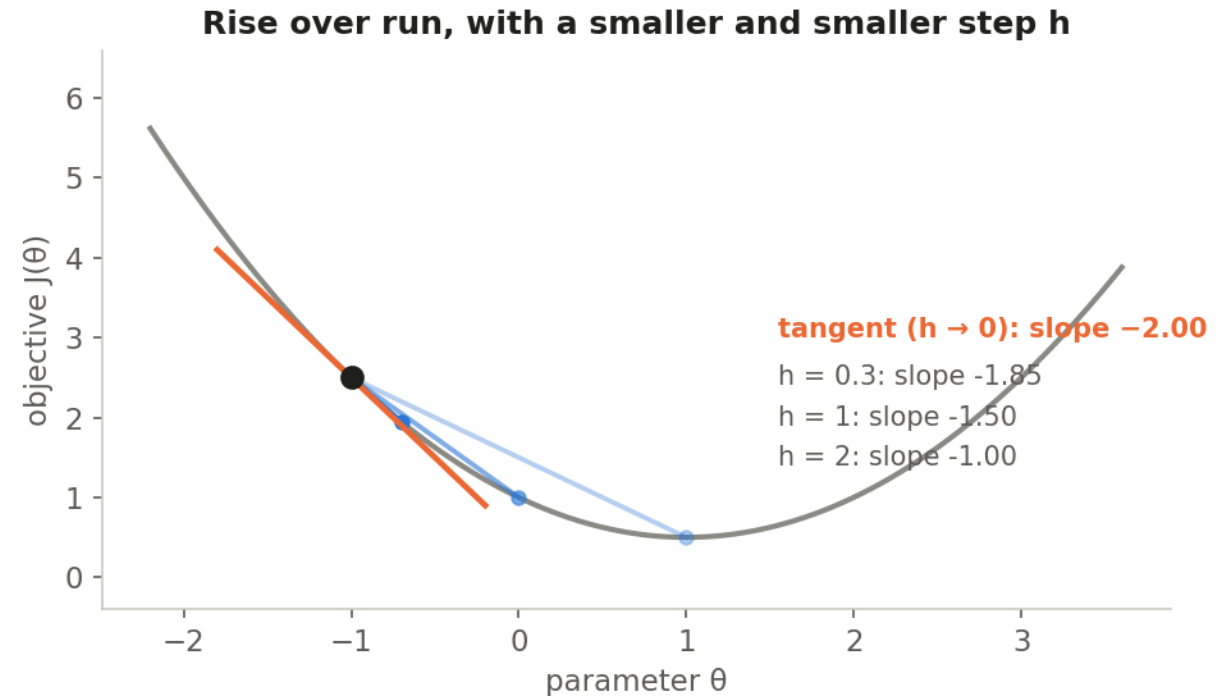
The derivative: slope with one parameter

The **derivative** $dJ/d\theta$ (also written $J'(\theta)$) is the slope of the objective curve at θ :

Indicates how J changes per unit change in θ .

Measure it as "rise over run" with a small step h , and let h decrease

$$\frac{dJ}{d\theta} = \lim_{h \rightarrow 0} \frac{J(\theta+h) - J(\theta)}{h}$$



Partial derivatives: slopes of objective for each parameter

With M parameters, calculate the slope for one parameter at a time, while holding the other fixed.

That is a **partial derivative**, written with a curly $\partial J / \partial \theta_m$

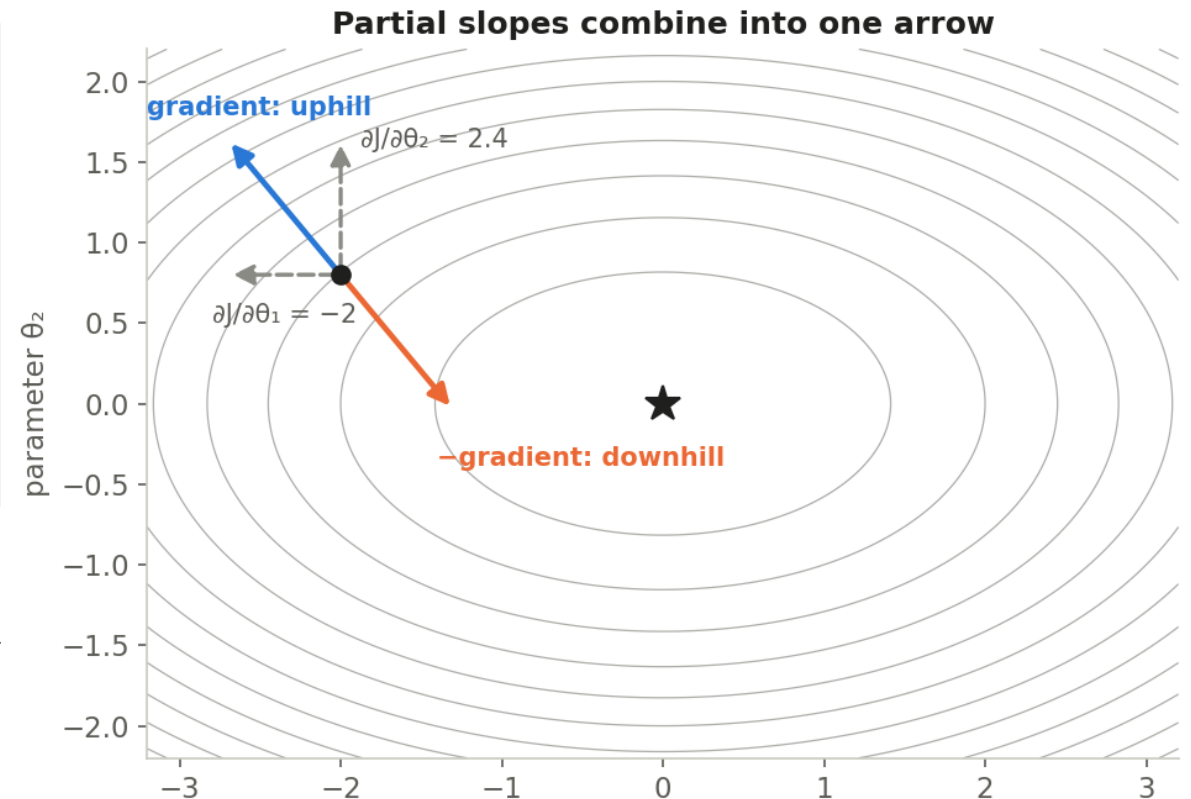
The **gradient** stacks all M partial derivatives into a column vector

$$\nabla J(\boldsymbol{\theta}) = \left[\frac{\partial J}{\partial \theta_1}, \dots, \frac{\partial J}{\partial \theta_M} \right]^\top$$

Example with $M = 2$

$$J(\boldsymbol{\theta}) = \frac{1}{2}\theta_1^2 + \frac{3}{2}\theta_2^2 \quad \Rightarrow \quad \nabla J(\boldsymbol{\theta}) = [\theta_1, 3\theta_2]^\top$$

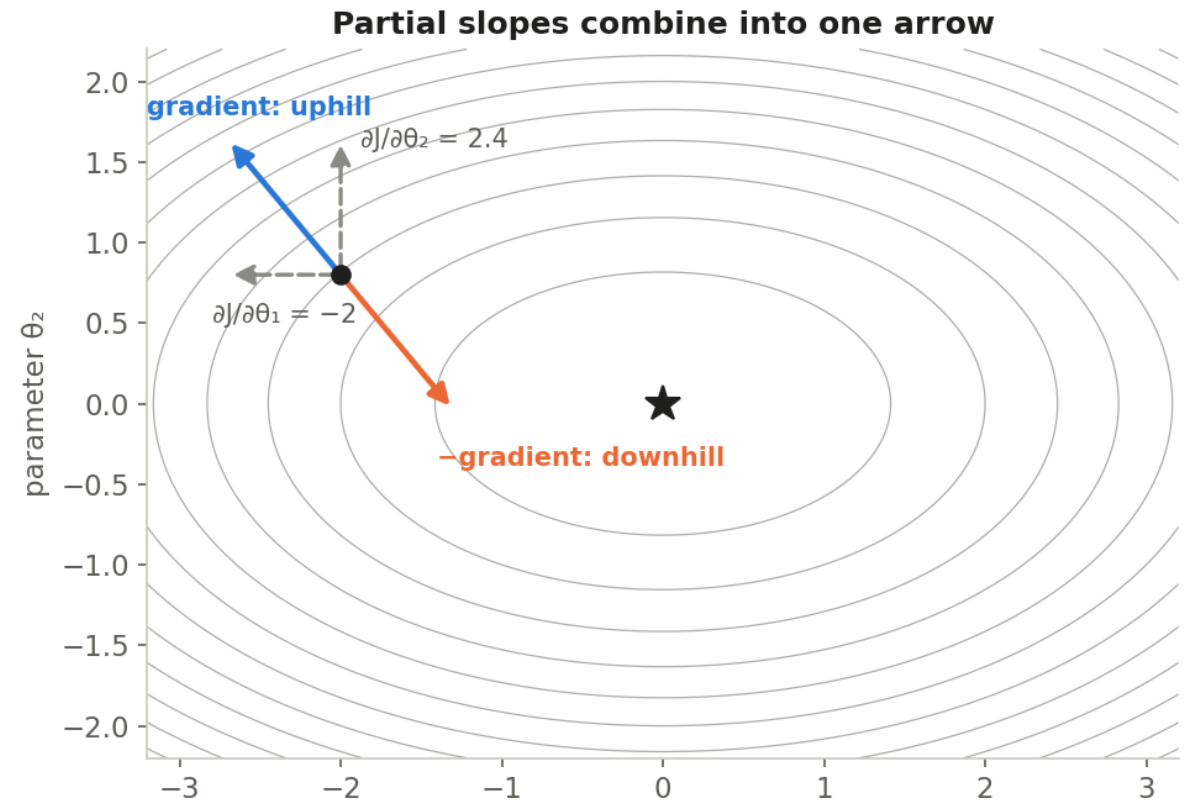
At $\boldsymbol{\theta} = [-2, 0.8]^\top$, the gradient is $[-2, 2.4]^\top$



The gradient: all the slopes in one arrow

The grey ovals are **contour lines**: each joins points with the same value of J

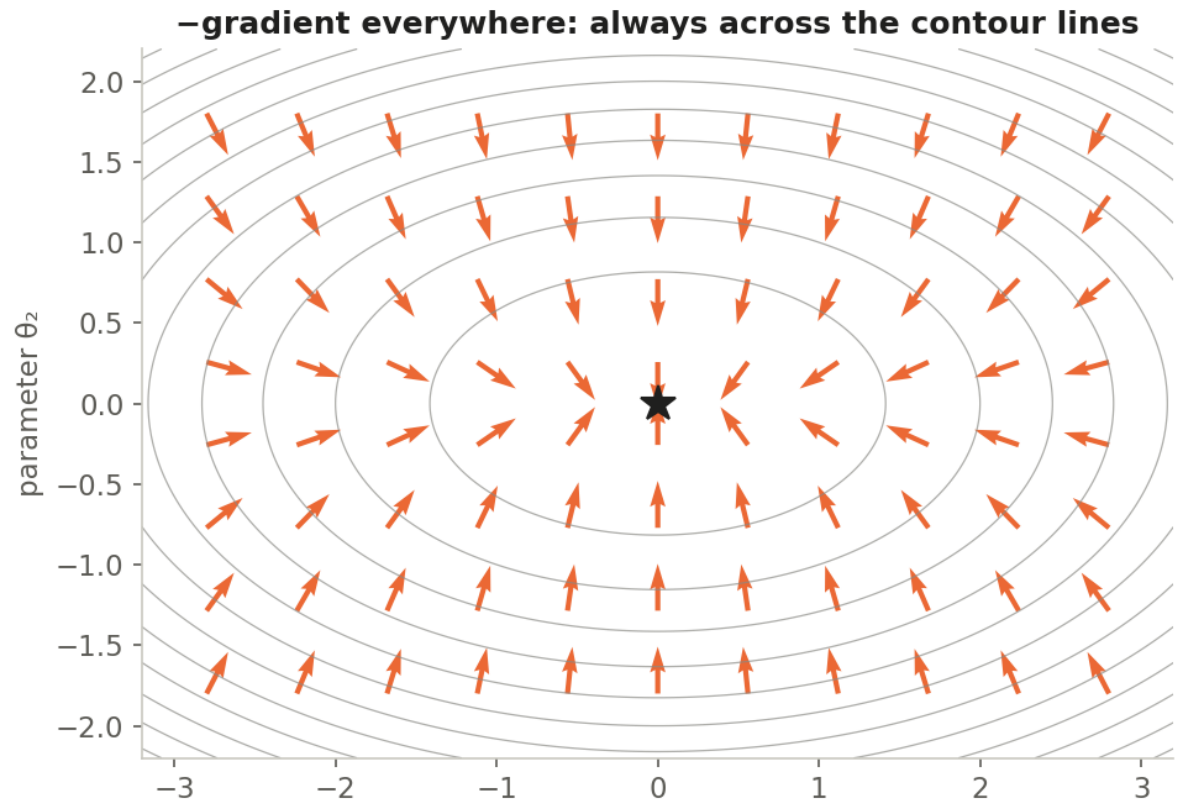
- The gradient points **uphill**, so $-\nabla J(\theta)$ points downhill.
- It is always **perpendicular to the contour lines**
- Its **length** measures the steepness; at a minimum the gradient is the zero vector.



The gradient: all the slopes in one arrow

The grey ovals are **contour lines**: each joins points with the same value of J

- The gradient points **uphill**, so $-\nabla J(\theta)$ points downhill.
- It is always **perpendicular to the contour lines**
- Its **length** measures the steepness; at a minimum the gradient is the zero vector.



Curvature

Slope tells us which way is downhill.

Curvature tells us how quickly that slope changes and that decides how large a **step** you can safely take.

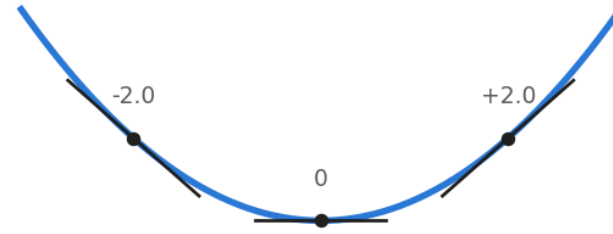
The **second derivative** $J''(\theta)$ is the slope of the slope, how fast the slope changes as you move.

Its **sign** tells us the shape of the objective.

Its **size** tells us how sharply the objective bends.

Positive: a bowl

$$J = \theta^2, J'' = +2$$



slope rises from $-$ to $+$
→ a minimum at the bottom

parameter θ

Negative: a hilltop

$$J = -\theta^2, J'' = -2$$

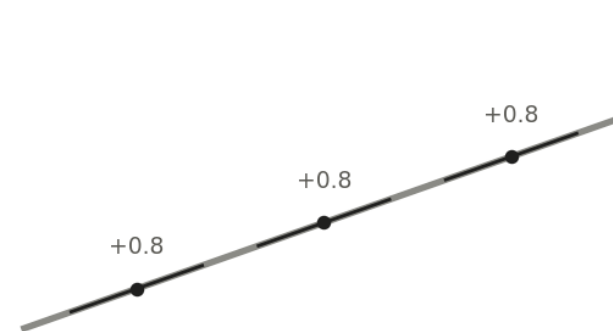


slope falls from $+$ to $-$
→ a maximum at the top

parameter θ

Zero: a straight slope

$$J = 0.8\theta, J'' = 0$$

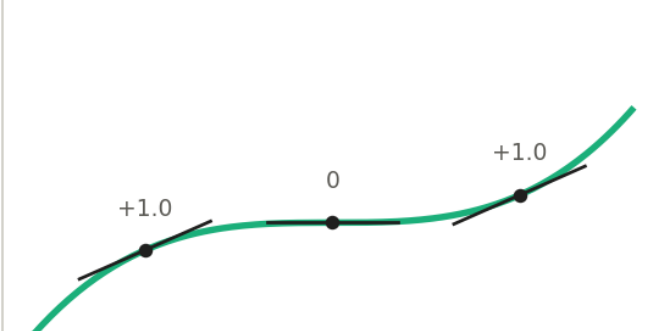


slope is the same
everywhere: no bend

parameter θ

Changing sign: inflection

$$J = \theta^3/3, J'' = 2\theta$$



bends down on the left,
up on the right; flat at 0

parameter θ

Curvature

Most of the trouble in optimization comes from curvature varying widely across directions.

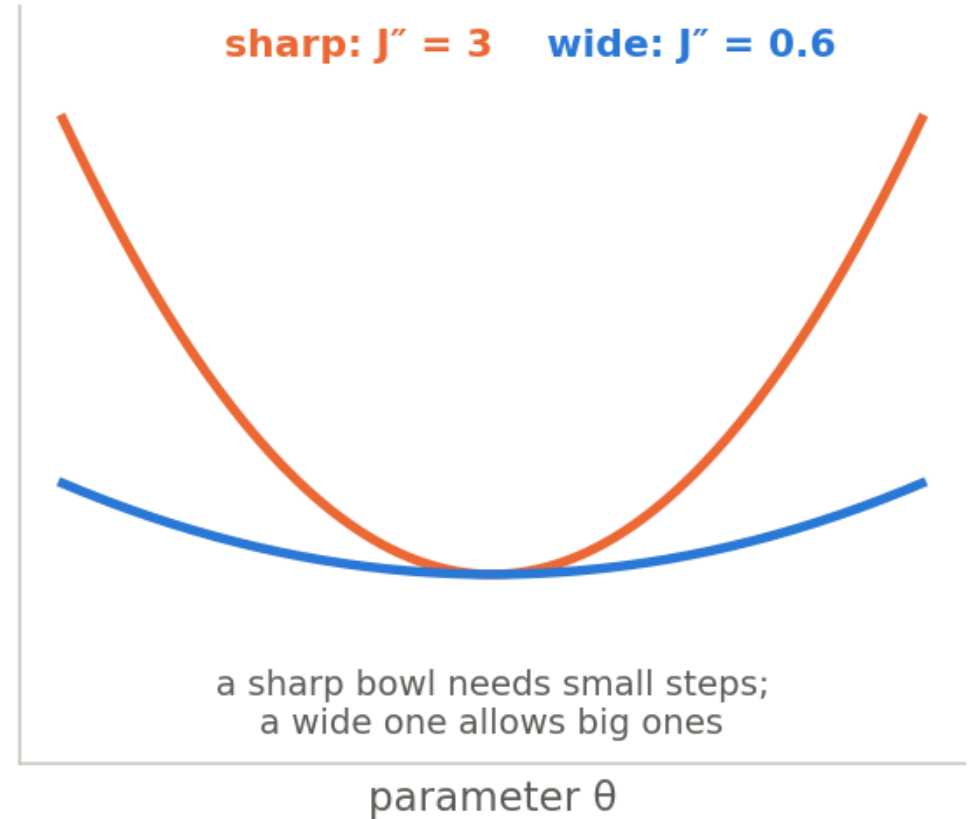
Two bowls can both have positive curvature but bend very differently.

A large J'' (here 3) is a narrow, **sharp** bowl: the slope changes quickly, so a step that looks reasonable from one side lands high on the other side.

A small J'' (here 0.6) is a **wide** bowl where big steps are safe. The learning rate must stay below $2/J''$.

Size matters: sharp vs wide

sharp: $J'' = 3$ wide: $J'' = 0.6$



Curvature

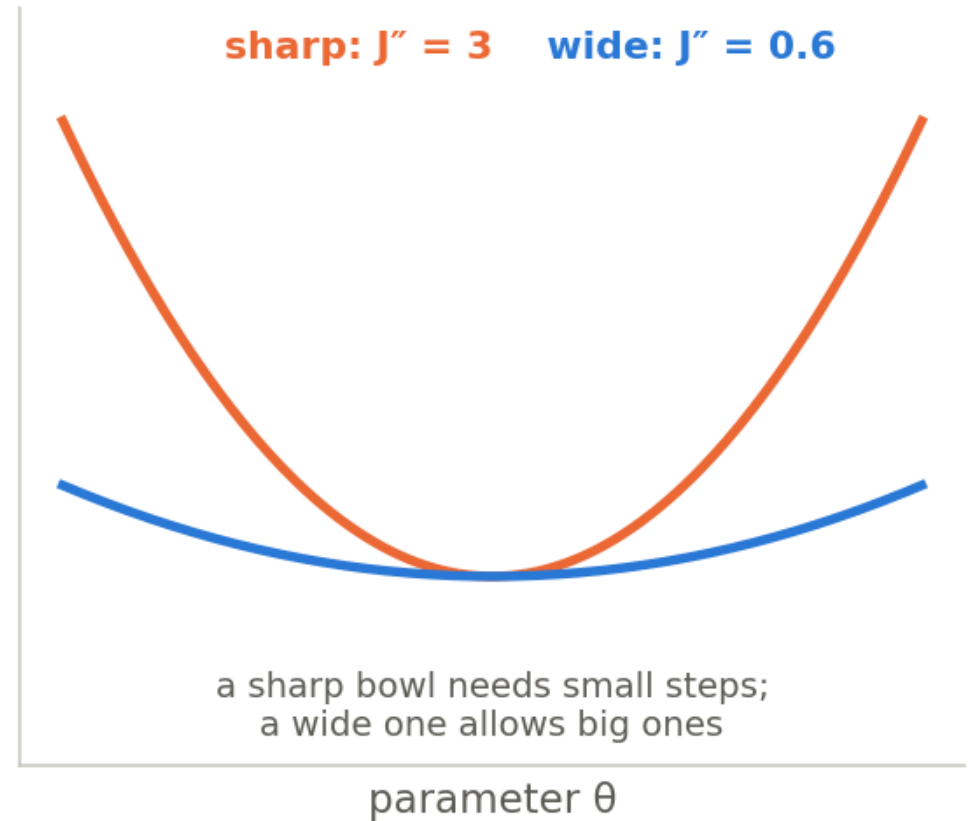
$$\text{slope} = \lambda \cdot d$$

d is the distance from the minimum
 λ is the curvature

In a sharp direction, even a small distance produces a big slope.

Size matters: sharp vs wide

sharp: $J'' = 3$ wide: $J'' = 0.6$



The Hessian: all the curvatures in one matrix

With M parameters, there is a **second derivative** for every pair of parameters.

The **Hessian** H collects them in an $M \times M$ matrix.

- The **diagonal** entries H_{mm} are the curvatures along each parameter.
- The **off-diagonal** ones say how the slope in one parameter changes when another parameter moves

$$H = \nabla^2 J(\boldsymbol{\theta}), \quad H_{jk} = \frac{\partial^2 J}{\partial \theta_j \partial \theta_k}$$

$$\mathbf{H} = \nabla^2 J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial^2 J}{\partial \theta_1^2} & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_n} \\ \frac{\partial^2 J}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 J}{\partial \theta_2^2} & \cdots & \frac{\partial^2 J}{\partial \theta_2 \partial \theta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial \theta_n \partial \theta_1} & \frac{\partial^2 J}{\partial \theta_n \partial \theta_2} & \cdots & \frac{\partial^2 J}{\partial \theta_n^2} \end{bmatrix}$$

For the ravine used in our example,

$$J(\boldsymbol{\theta}) = 0.1 \theta_1^2 + 2 \theta_2^2$$

$$\nabla J(\boldsymbol{\theta}) = [0.2 \theta_1, 4 \theta_2]^\top$$

$$H = \begin{pmatrix} 0.2 & 0 \\ 0 & 4 \end{pmatrix}$$

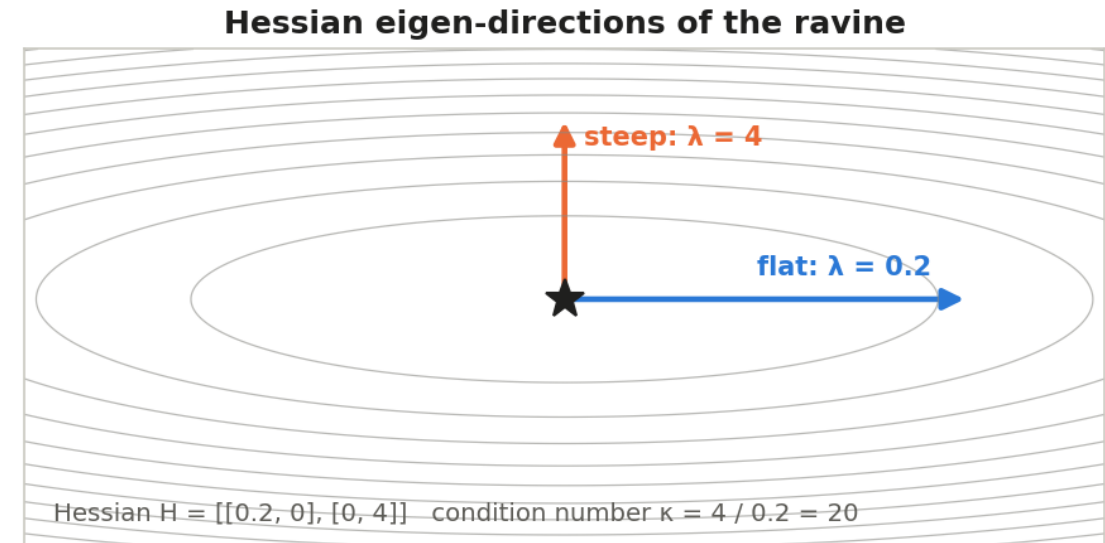
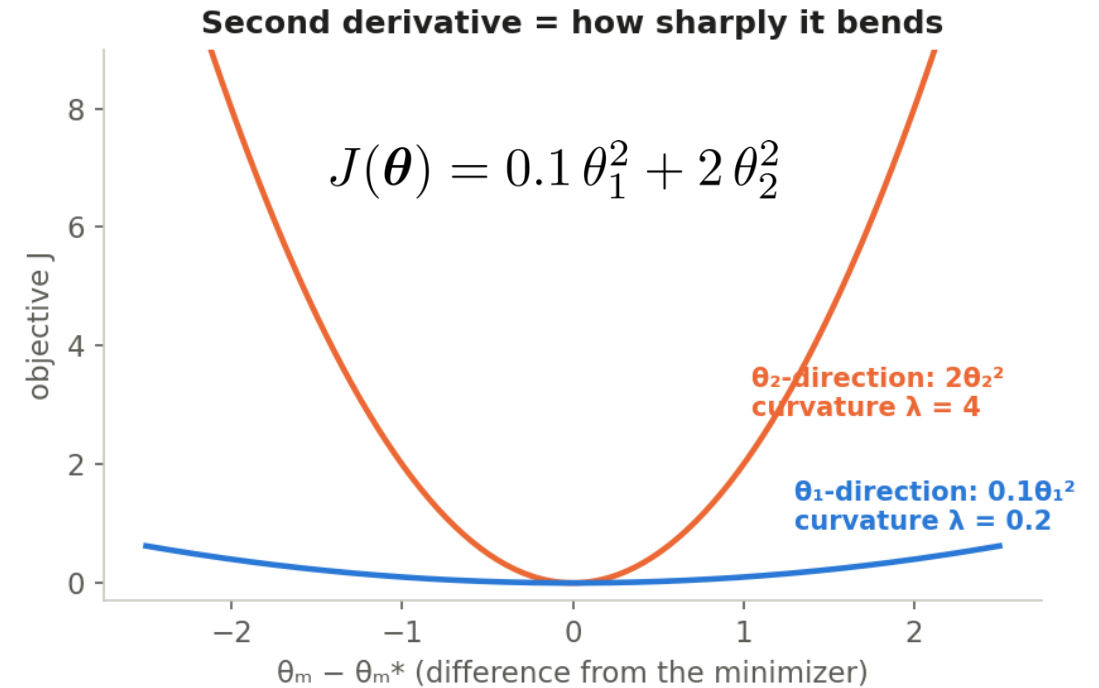
Eigenvector, eigenvalues and the condition number

An **eigenvector** $\mathbf{e}$ of H is a direction along which the objective is a plain parabola

Its curvature in that direction is the **eigenvalue** λ .

In symbols, $Hv = \lambda v$: multiplying the direction v by H only stretches it by λ

- For our ravine, the eigen-directions are simply the x-axis ($\lambda = 0.2$, flat) and the y-axis ($\lambda = 4$, steep)
- If the oval contours were tilted, the eigen-directions would be the tilted long and short axes of the ovals.



Eigenvector, eigenvalues and the condition number

The **condition number** compares the steepest and flattest directions:

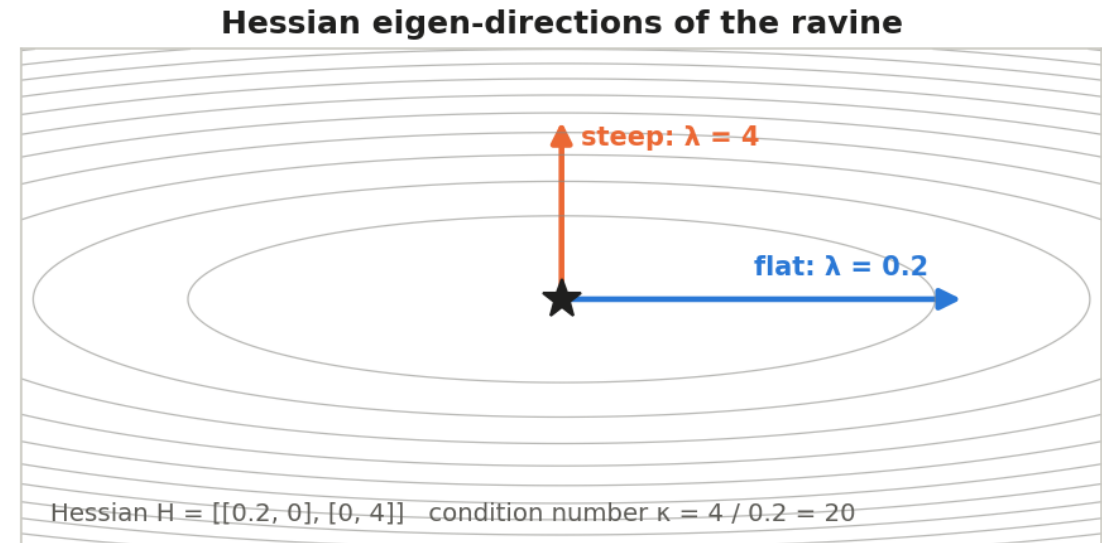
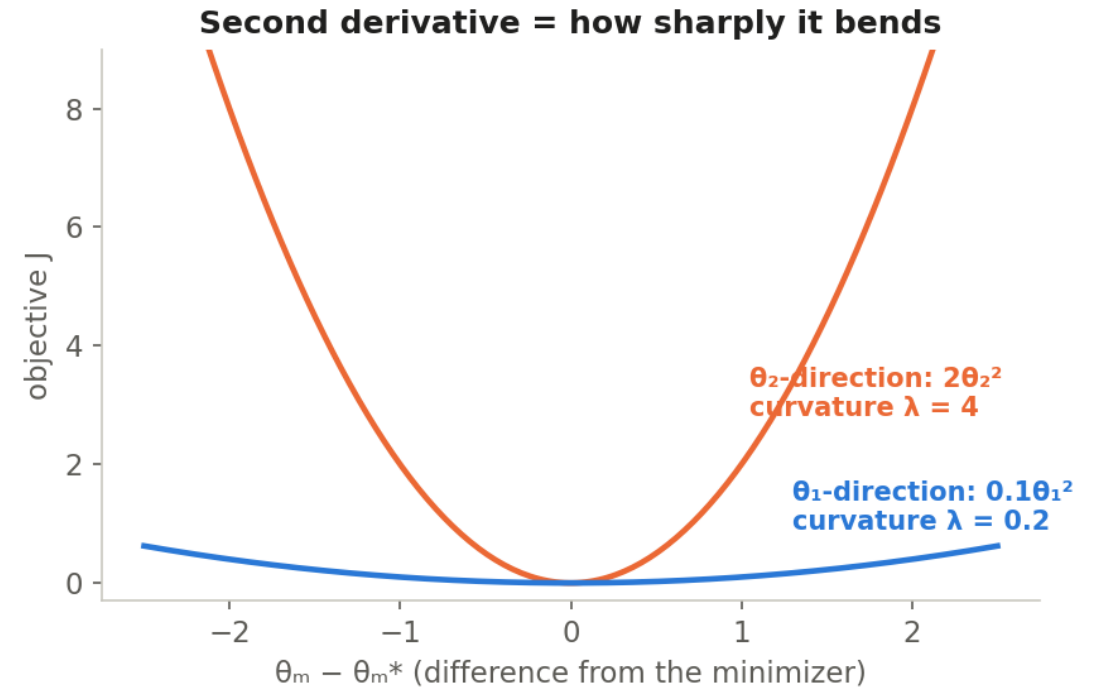
$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$$

$\kappa = 1$ is a perfectly round bowl.

Our ravine has $\kappa = 4/0.2 = 20$, a stretched oval.

Real networks can have κ in the thousands or more.

A large κ means a long, narrow ravine, and that is exactly what makes gradient descent slow.



Eigenvector, eigenvalues and the condition number

With two parameters, there are two eigenvalues, and their signs and sizes name the shape:

Curvature in 2-D: the Hessian eigenvalues decide the shape (top: surface, bottom: seen from above)

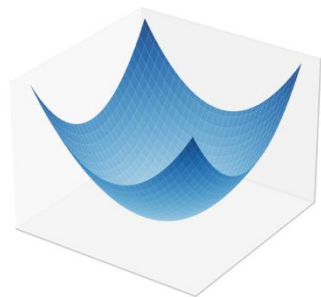
$\kappa = 1$ is a perfectly round bowl.

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$$

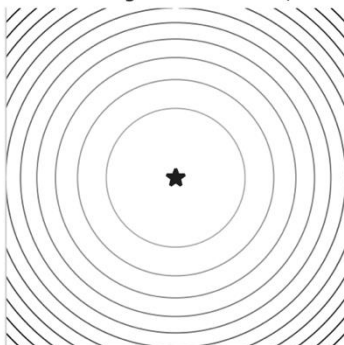
Real networks can have κ in the thousands or more.

A large κ means a long, narrow ravine, and that is exactly what makes gradient descent slow.

Bowl



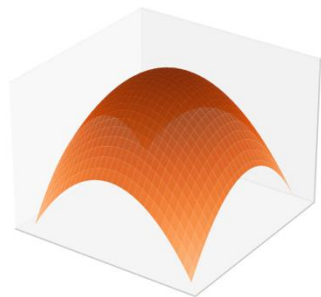
Hessian eigenvalues $\lambda = +2, +2$



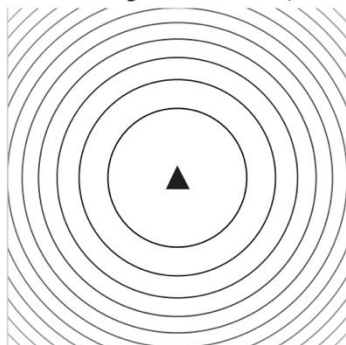
both positive:
a minimum

Heads straight in

Dome



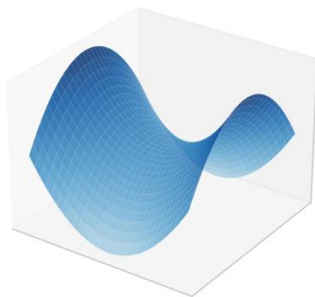
Hessian eigenvalues $\lambda = -2, -2$



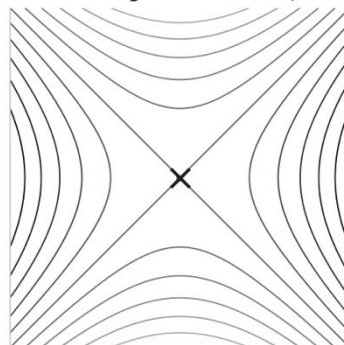
both negative:
a maximum

Rolls away in every
direction

Saddle



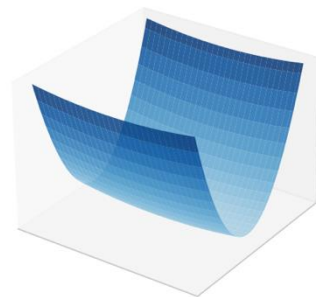
Hessian eigenvalues $\lambda = +2, -2$



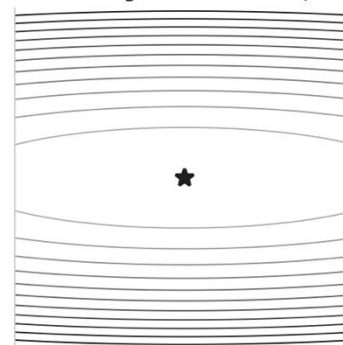
mixed signs: up one way,
down the other

Slows near the middle,
then escapes downhill

Ravine



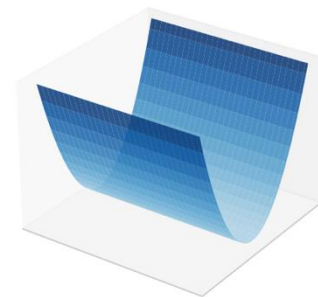
Hessian eigenvalues $\lambda = +0.2, +4$



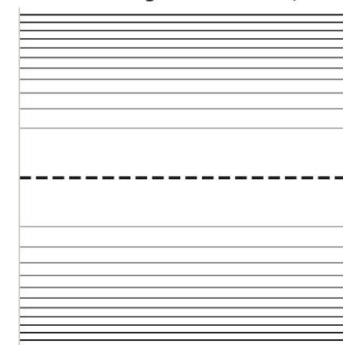
both positive, very unequal:
long narrow valley

Zig-zags across,
crawls along

Flat trough



Hessian eigenvalues $\lambda = 0, +2$



one zero: a whole line
of equally good points

Drops to the floor, then
feels no push along it

What
gradient
descent
does

Mean, variance and root mean square

1. Mean: the typical direction.

$$\bar{x} = \frac{1}{T} \sum_{t=1}^T x^{(t)}$$

2. Mean square (MS): square first, then average. It measures size and ignores sign.

$$\text{MS} = \frac{1}{T} \sum_{t=1}^T (x^{(t)})^2$$

3. Variance σ^2 : the average squared difference from the mean, which measures the noise. Its square root σ is the standard deviation.

$$\sigma^2 = \frac{1}{T} \sum_{t=1}^T (x^{(t)} - \bar{x})^2$$

4. Root mean square (RMS): the typical size, in the same units as x .

$$\text{RMS} = \sqrt{\text{MS}}$$

t	$x^{(t)}$	$(x^{(t)})^2$	$x^{(t)} - \bar{x}$	$(x^{(t)} - \bar{x})^2$
1	3	9	2	4
2	-1	1	-2	4
3	2	4	1	1
4	0	0	-1	1
	$\bar{x} = 1$	MS = 3.5	0 (always)	$\sigma^2 = 2.5$

So $\text{RMS} = \sqrt{3.5} \approx 1.87$ and $\sigma = \sqrt{2.5} \approx 1.58$

Mean tells the direction, RMS tells the size: $\text{mean square} = \text{mean}^2 + \text{variance}$

averages and noise

$$\sigma^2 = \frac{1}{T} \sum_{t=1}^T (x^{(t)} - \bar{x})^2$$

$$\sigma^2 = \frac{1}{T} \sum_t \left((x^{(t)})^2 - 2\bar{x}x^{(t)} + \bar{x}^2 \right)$$

$$= \text{MS} - 2\bar{x} \cdot \bar{x} + \bar{x}^2$$

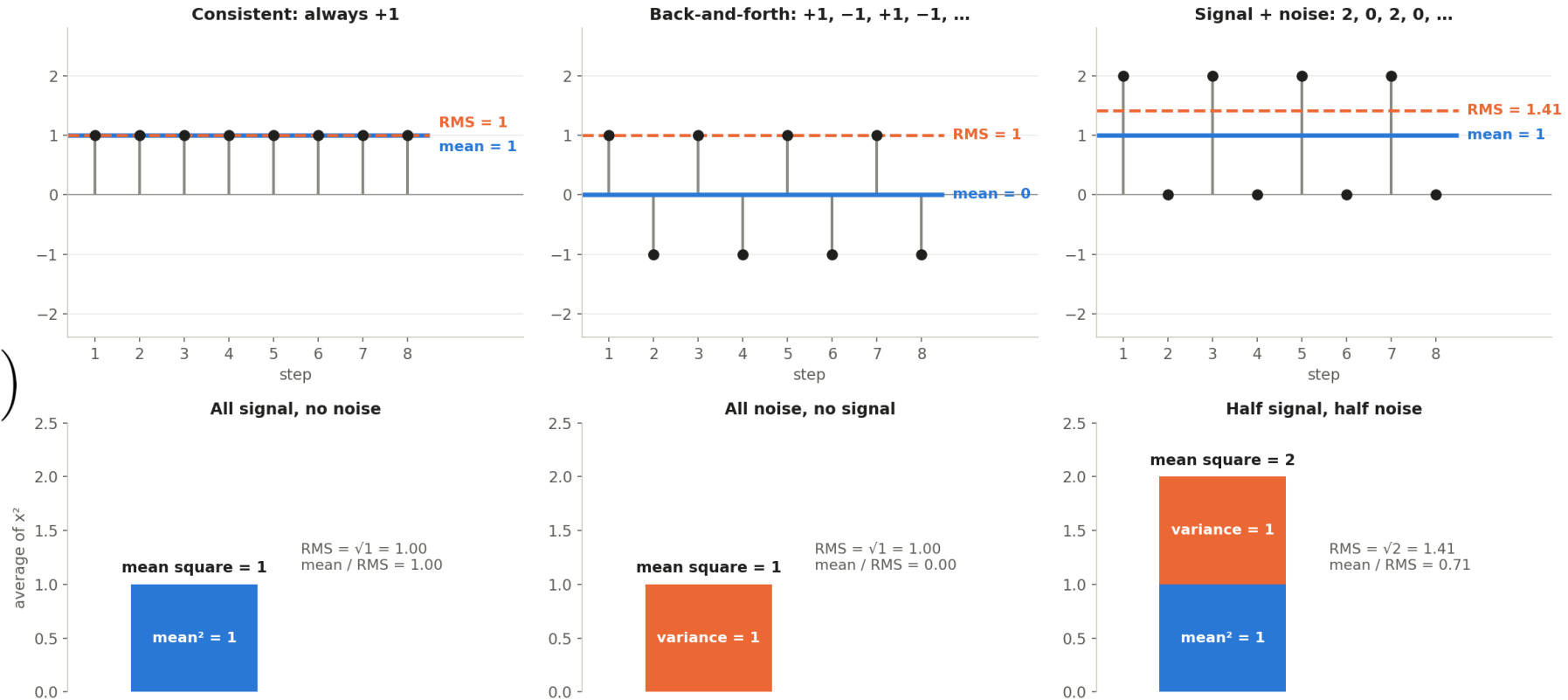
$$= \text{MS} - \bar{x}^2$$

$$\Rightarrow \boxed{\text{MS} = \bar{x}^2 + \sigma^2}$$

mean square = mean² + variance

$$\mathbb{E}[X^2] = (\mathbb{E}[X])^2 + \sigma^2$$

$$\sigma^2 = \mathbb{E}[(X - \mathbb{E}[X])^2]$$



- $RMS \geq |\bar{x}|$, with equality only when there is no noise.
- **The ratio $\bar{x}/RMS$ lies between -1 and 1.** It is near ± 1 for a consistent signal and near 0 for pure noise. Adam's step is essentially λ times this ratio.
- $\sigma^2 = MS - \bar{x}^2$. This is how RMSprop and Adam work in practice: they keep averages of g and of g^2 , never differences from the mean.

Exponential Moving Average (EMA)

At every step, the optimizer receives a new, noisy gradient.

We do ***not*** want to store the whole history of gradients for every step

We want a ***running estimate*** of what the gradients have *typically* been lately, updated cheaply at each step

$$s_t = \beta s_{t-1} + (1 - \beta) x_t$$

Exponential Moving Average (EMA)

At every step, the optimizer receives a new, noisy gradient.

We do **not** want to store the whole history of gradients for every step

We want a **running estimate** of what the gradients have *typically* been lately, updated cheaply at each step

start the average
at $s_0 = 0$

$$s_t = \beta s_{t-1} + (1 - \beta) x_t$$

keeps a fraction β of
the old average

mixes in a fraction $1 - \beta$
of the new value

new average = $\beta \times$ **old average** + $(1 - \beta) \times$ **new value**.

Exponential Moving Average (EMA)

At every step, the optimizer receives a new, noisy gradient.

We do **not** want to store the whole history of gradients for every step

We want a **running estimate** of what the gradients have *typically* been lately, updated cheaply at each step

β is a number between 0 and 1, indicating how much of the past to keep.

EMA effectively remembers about the last $1/(1 - \beta)$ values:

- 2 steps for $\beta = 0.5$
- 10 for $\beta = 0.9$
- 100 for $\beta = 0.99$
- 1,000 for $\beta = 0.999$

Larger β means smoother but slower to react

start the average
at $s_0 = 0$

$$s_t = \beta s_{t-1} + (1 - \beta) x_t$$

keeps a fraction β of
the old average

mixes in a fraction $1 - \beta$
of the new value

new average = $\beta \times$ old average + $(1 - \beta) \times$ new value.

Exponential Moving Average (EMA)

At every step, the optimizer receives a new, noisy gradient.

We do **not** want to store the whole history of gradients for every step

We want a **running estimate** of what the gradients have *typically* been lately, updated cheaply at each step

β is a number between 0 and 1, indicating how much of the past to keep.

EMA effectively remembers about the last $1/(1 - \beta)$ values:

- 2 steps for $\beta = 0.5$
- 10 for $\beta = 0.9$
- 100 for $\beta = 0.99$
- 1,000 for $\beta = 0.999$

Larger β means smoother but slower to react

start the average
at $s_0 = 0$

$$s_t = \beta s_{t-1} + (1 - \beta) x_t$$

keeps a fraction β of
the old average

mixes in a fraction $1 - \beta$
of the new value

new average = $\beta \times$ **old average** + $(1 - \beta) \times$ **new value**.

Rolling:

$$s_t = \beta s_{t-1} + (1 - \beta) x_t$$

$$s_t = \beta [\beta s_{t-2} + (1 - \beta) x_{t-1}] + (1 - \beta) x_t$$

$$\vdots$$

$$s_t = s_0 + \dots + \beta^2(1 - \beta)x_{t-2} + \beta^1(1 - \beta)x_{t-1} + (1 - \beta)x_t$$

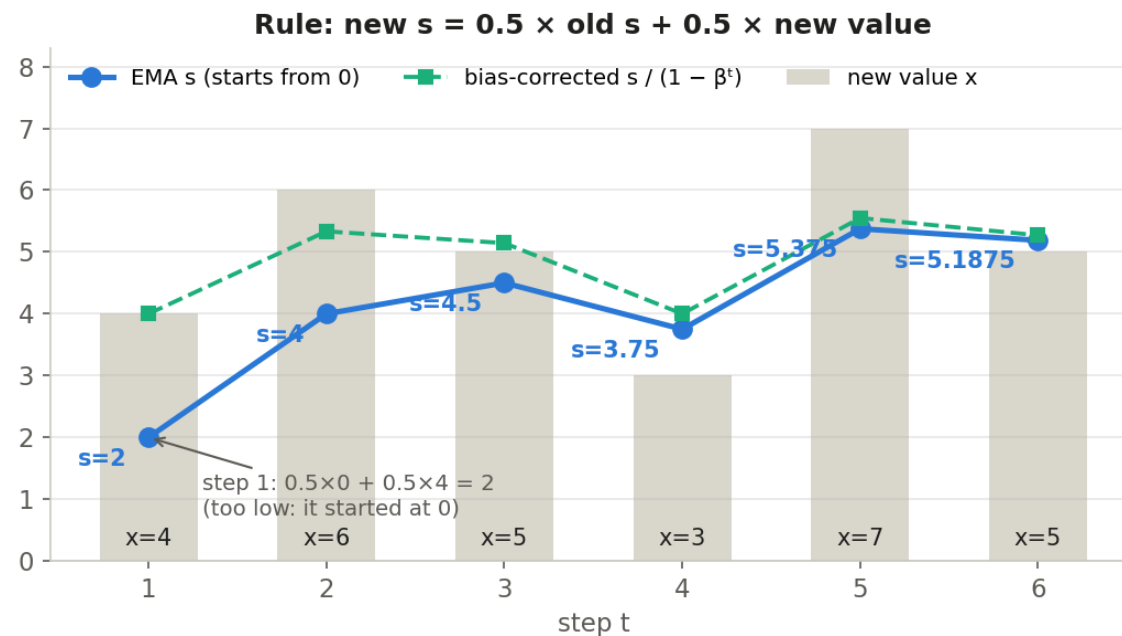
β sets how far back it remembers. Because each older value counts β times less, only the last $1/(1 - \beta)$ or so values really matter

Exponential Moving Average (EMA)

$$s_t = \beta s_{t-1} + (1 - \beta) x_t$$

$$\text{new average} = \beta \times \text{old average} + (1 - \beta) \times \text{new value}$$

Step t	New value x	Calculation ($\beta = 0.5$): $0.5 \times s_{t-1} + 0.5 \times x_t$	New average s
1	4	$0.5 \times 2 + 0.5 \times 6$	2
2	6	$0.5 \times 4 + 0.5 \times 5$	4
3	5	$0.5 \times 4.5 + 0.5 \times 3$	4.5
4	3	$0.5 \times 3.75 + 0.5 \times 7$	3.75
5	7	$0.5 \times 0 + 0.5 \times 4$	5.375
6	5	$0.5 \times 0 + 0.5 \times 4$	5.1875



Exponential Moving Average (EMA)

$$s_t = \beta s_{t-1} + (1 - \beta) x_t$$

$$s_3 = 0.5 x_3 + 0.5 s_2$$

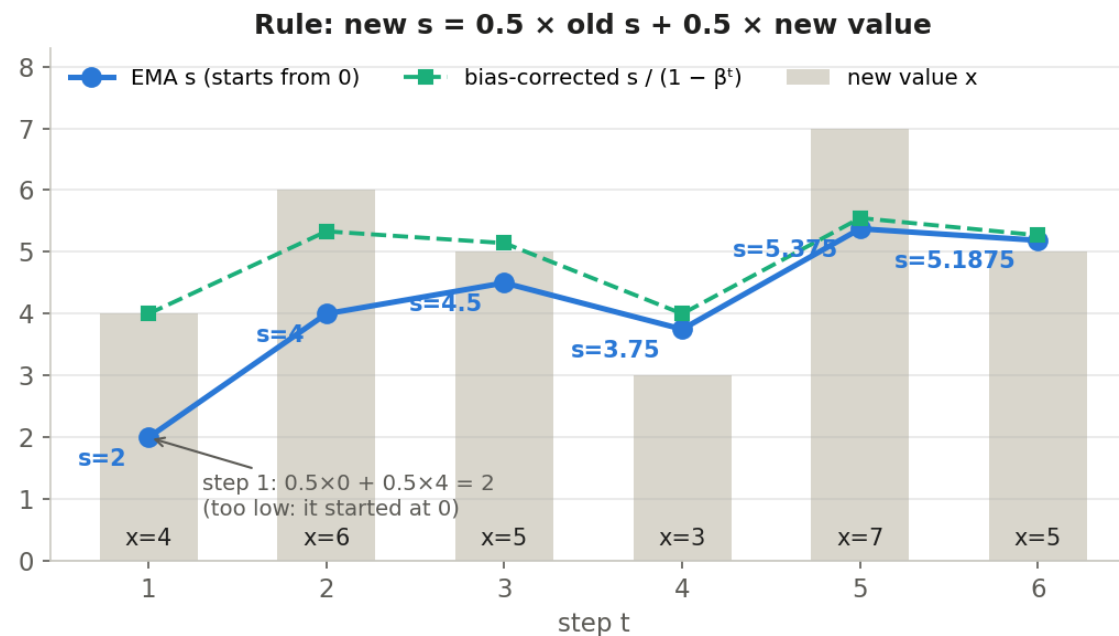
$$= 0.5 x_3 + 0.5 (0.5 x_2 + 0.5 s_1)$$

$$= 0.5 x_3 + 0.25 x_2 + 0.125 x_1$$

Notice that s_3 's shares do not add up to 1
 $0.5 + 0.25 + 0.125 = 0.875$

$$\text{new average} = \beta \times \text{old average} + (1 - \beta) \times \text{new value}$$

Step t	New value x	Calculation ($\beta = 0.5$): $0.5 \times s_{t-1} + 0.5 \times x_t$	New average s
1	4	$0.5 \times 2 + 0.5 \times 6$	2
2	6	$0.5 \times 4 + 0.5 \times 5$	4
3	5	$0.5 \times 4.5 + 0.5 \times 3$	4.5
4	3	$0.5 \times 3.75 + 0.5 \times 7$	3.75
5	7	$0.5 \times 0 + 0.5 \times 4$	5.375
6	5	$0.5 \times 0 + 0.5 \times 4$	5.1875



Exponential Moving Average (EMA)

$$s_t = \beta s_{t-1} + (1 - \beta) x_t$$

$$\begin{aligned} s_3 &= 0.5 x_3 + 0.5 s_2 \\ &= 0.5 x_3 + 0.5 (0.5 x_2 + 0.5 s_1) \\ &= 0.5 x_3 + 0.25 x_2 + 0.125 x_1 \end{aligned}$$

Notice that s_3 's shares do not add up to 1
 $0.5 + 0.25 + 0.125 = 0.875$

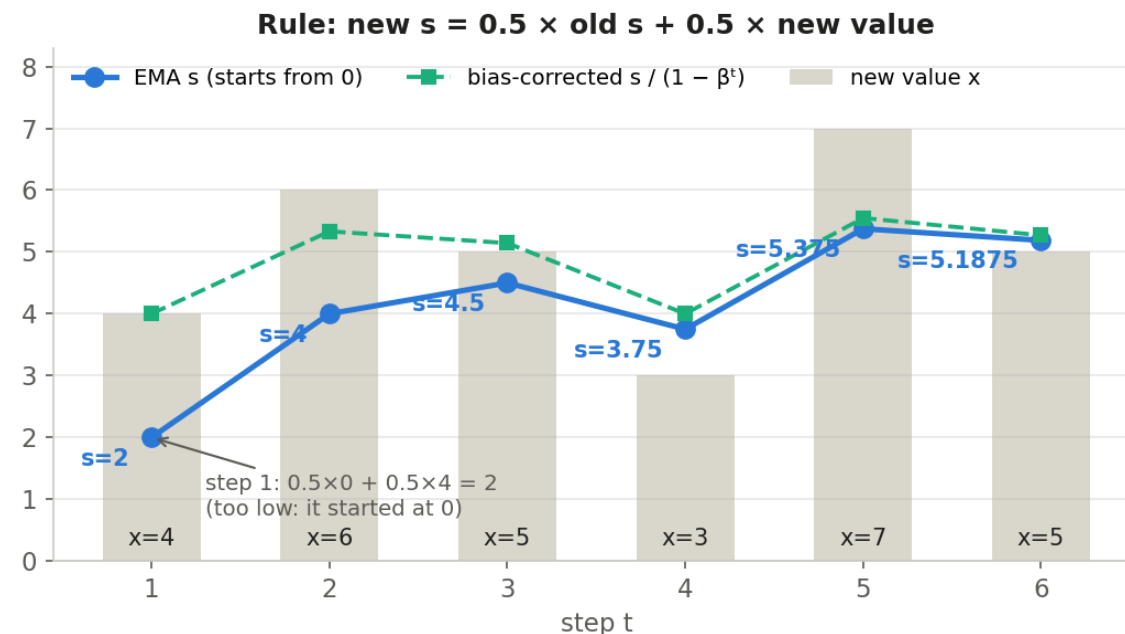
$$s_t = s_0 + \dots + \beta^2(1 - \beta)x_{t-2} + \beta^1(1 - \beta)x_{t-1} + (1 - \beta)x_t$$

$$1 + \beta + \beta^2 + \dots + \beta^{t-1} = \frac{1 - \beta^t}{1 - \beta}$$

$$\Rightarrow \text{shares add up to } (1 - \beta) \cdot \frac{1 - \beta^t}{1 - \beta} = 1 - \beta^t$$

$$\text{new average} = \beta \times \text{old average} + (1 - \beta) \times \text{new value}$$

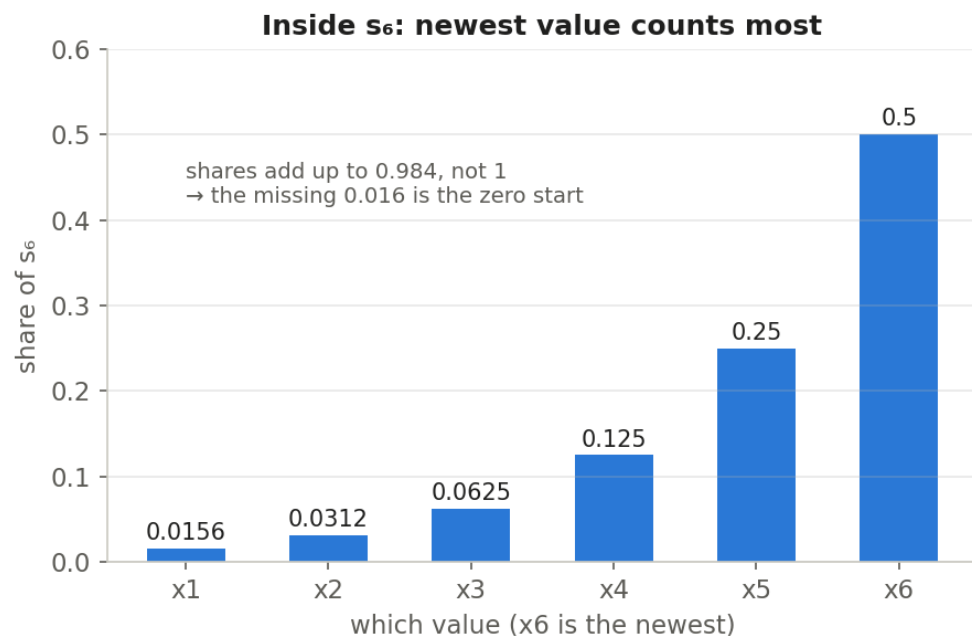
Step t	New value x	Calculation ($\beta = 0.5$): $0.5 \times s_{t-1} + 0.5 \times x_t$	New average s
1	4	$0.5 \times 2 + 0.5 \times 6$	2
2	6	$0.5 \times 4 + 0.5 \times 5$	4
3	5	$0.5 \times 4.5 + 0.5 \times 3$	4.5
4	3	$0.5 \times 3.75 + 0.5 \times 7$	3.75
5	7	$0.5 \times 0 + 0.5 \times 4$	5.375
6	5	$0.5 \times 0 + 0.5 \times 4$	5.1875



Exponential Moving Average (EMA)

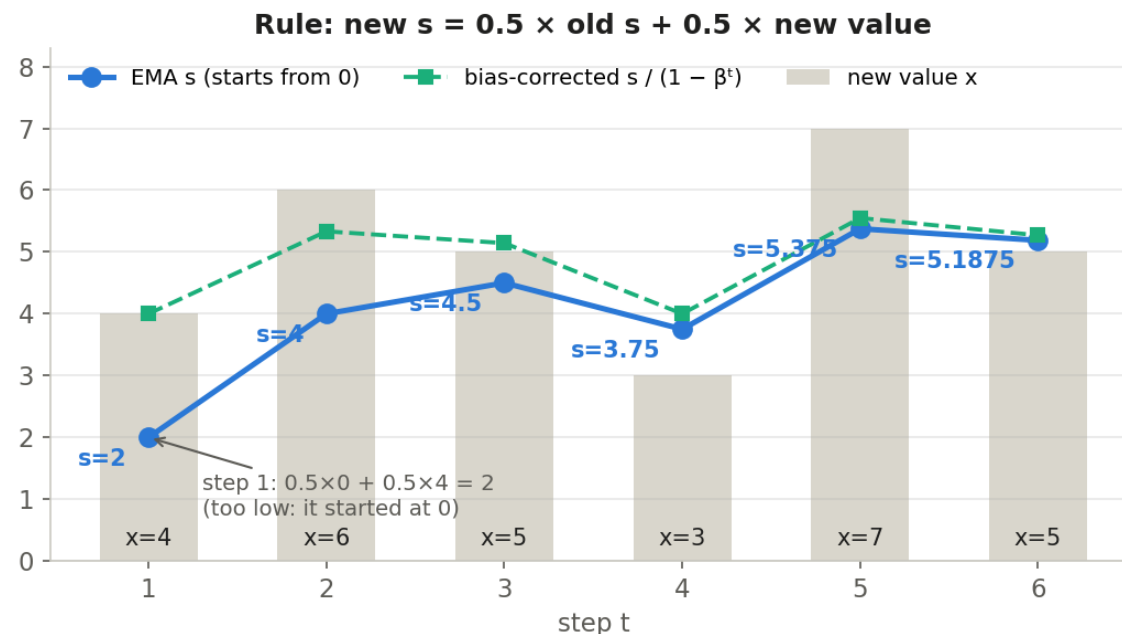
$$s_t = \beta s_{t-1} + (1 - \beta) x_t$$

$$\begin{aligned} s_3 &= 0.5 x_3 + 0.5 s_2 \\ &= 0.5 x_3 + 0.5 (0.5 x_2 + 0.5 s_1) \\ &= 0.5 x_3 + 0.25 x_2 + 0.125 x_1 \end{aligned}$$



$$\text{new average} = \beta \times \text{old average} + (1 - \beta) \times \text{new value}$$

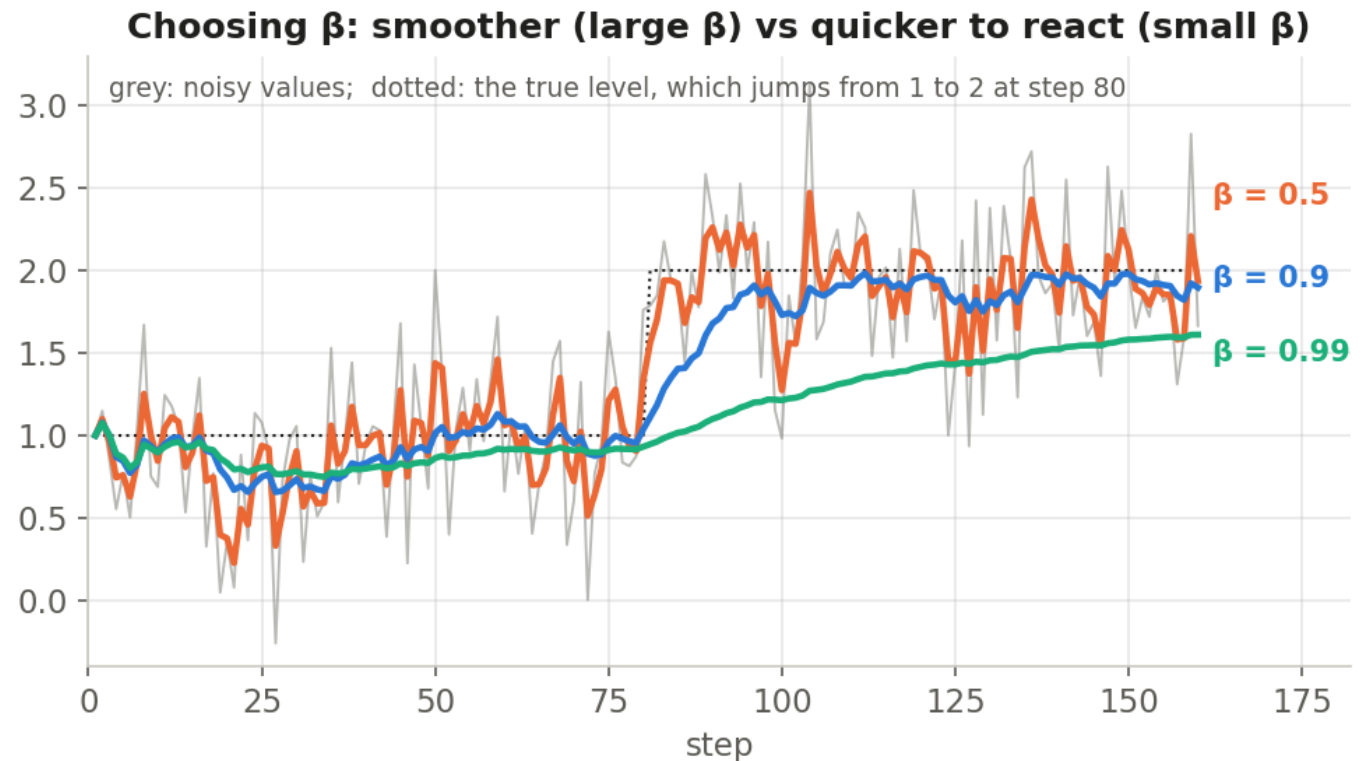
Step t	New value x	Calculation ($\beta = 0.5$): $0.5 \times s_{t-1} + 0.5 \times x_t$	New average s
1	4	$0.5 \times 2 + 0.5 \times 6$	2
2	6	$0.5 \times 4 + 0.5 \times 5$	4
3	5	$0.5 \times 4.5 + 0.5 \times 3$	4.5
4	3	$0.5 \times 3.75 + 0.5 \times 7$	3.75
5	7	$0.5 \times 0 + 0.5 \times 4$	5.375
6	5	$0.5 \times 0 + 0.5 \times 4$	5.1875



Exponential Moving Average (EMA)

β sets the memory

- Small β (quick to react): the average forgets the old level fast, so it follows the jump within a few steps.
- Large β (slow to react): the old values keep most of the weight, so the average lags behind the change for a long time.



$$s_t = (1 - \beta) (x_t + \beta x_{t-1} + \beta^2 x_{t-2} + \dots + \beta^{t-1} x_1)$$

$$1 + \beta + \beta^2 + \dots + \beta^{t-1} = \frac{1 - \beta^t}{1 - \beta} \xrightarrow{t \rightarrow \infty} \frac{1}{1 - \beta}$$

$$\Rightarrow \text{shares add up to } (1 - \beta) \cdot \frac{1 - \beta^t}{1 - \beta} = 1 - \beta^t$$

Exponential Moving Average (EMA)

Bias with the zero start

Because the shares add up to only $1 - \beta^t$, an EMA that starts at 0 is **too small at first**.

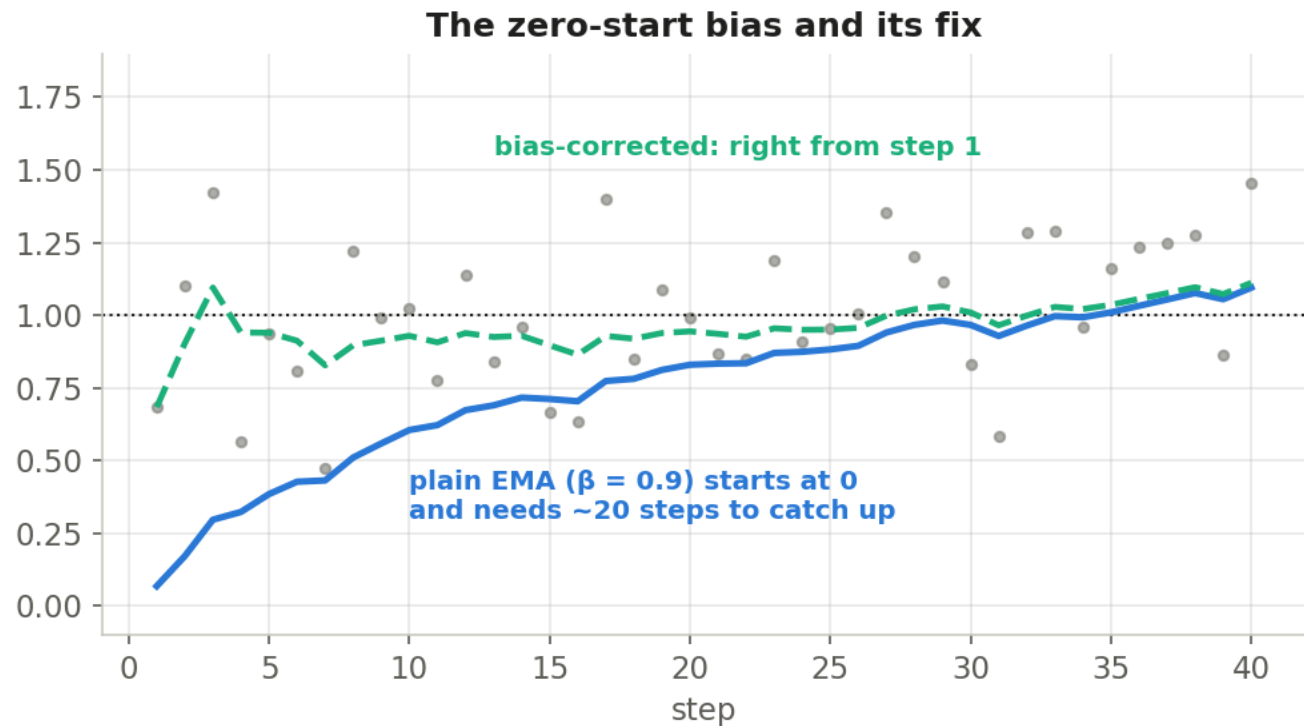
If every value equals x , the average after t steps is exactly $(1 - \beta^t) \cdot x$.

The fix is to divide that factor out:

$$\hat{s}_t = \frac{s_t}{1 - \beta^t}$$

Fixing the start matters a lot (for Adam):

- with $\beta = 0.9$ the plain EMA needs about 20 steps to catch up
- with $\beta = 0.999$ it would need thousands.



$$\hat{s}_1 = 2 / (1 - 0.5) = 4$$

$$\hat{s}_3 = 4.5 / 0.875 \approx 5.14$$

After many steps, β^t is almost 0 and the correction stops mattering

Gradient Descent



$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla J(\boldsymbol{\theta}^{(t)})$$

$$\text{new weights} = \text{old weights} - \text{learning rate} \times \text{gradient}$$

Each parameter moves against its own slope, by an amount proportional to how steep that slope

$$\text{entry by entry: } \theta_m^{(t+1)} = \theta_m^{(t)} - \eta \frac{\partial J}{\partial \theta_m}(\boldsymbol{\theta}^{(t)})$$

Gradient Descent

Let $J(\theta) = \theta^2$

Step	$\theta^{(t)}$	Slope $J'(\theta^{(t)}) = 2\theta^{(t)}$	Update $\theta^{(t+1)} = \theta^{(t)} - 0.3 \cdot J'(\theta^{(t)})$
0	-4.00	-8.00	$-4 + 2.4 = -1.60$
1	-1.60	-3.20	$-1.6 + 0.96 = -0.64$
2	-0.64	-1.28	$-0.64 + 0.384 = -0.256$

Each step multiplies θ by the same factor, 0.4.
Near a minimizer with curvature λ , the slope is

$$J'(\theta) = \lambda(\theta - \theta^*)$$

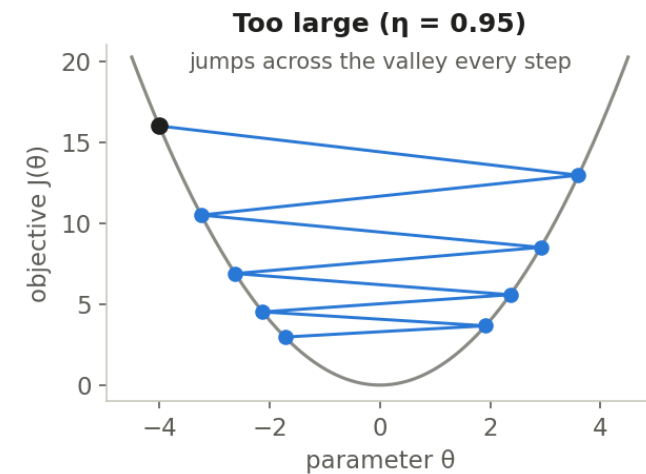
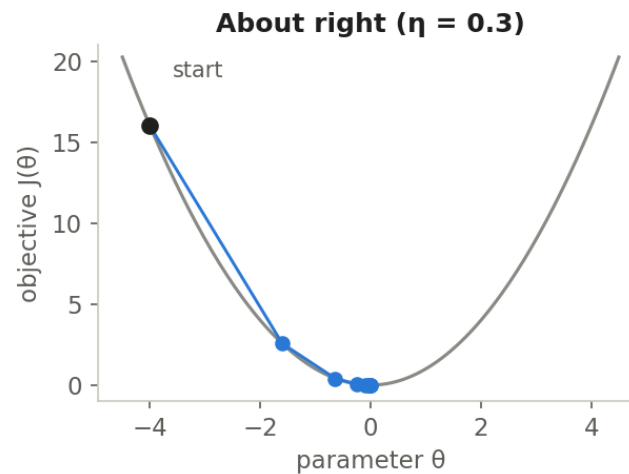
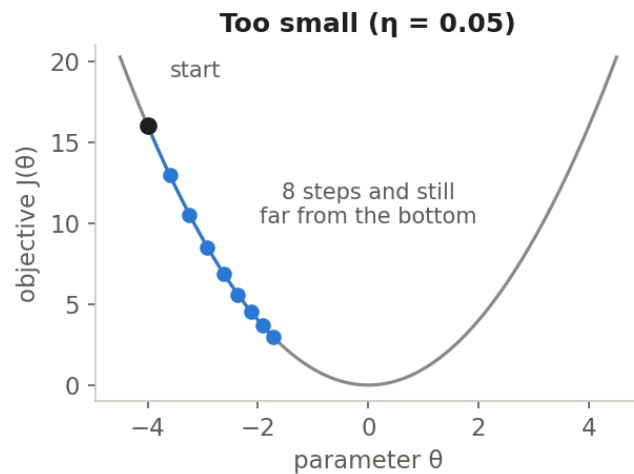
So subtracting θ^* from both sides of the update gives:

$$\theta^{(t+1)} - \theta^* = (1 - \eta\lambda) (\theta^{(t)} - \theta^*)$$

How big the learning rate can be depends on this

The safe range is $0 < \eta < \frac{2}{\lambda}$

Gradient descent: step downhill; the step size η sets how far



Gradient Descent

$\eta\lambda$	Factor	What happens
0.1	0.9	Shrinks 10% per step: slow but steady
0.6	0.4	Shrinks fast: a good choice
1	0	Lands exactly on the minimum in one step
1.9	-0.9	Sign flips every step: bounces across the valley, slowly settling
above 2	below -1	Each bounce is bigger than the last: diverges

Each step multiplies θ by the same factor, 0.4.
Near a minimizer with curvature λ , the slope is

$$J'(\theta) = \lambda(\theta - \theta^*)$$

So subtracting θ^* from both sides of the update gives:

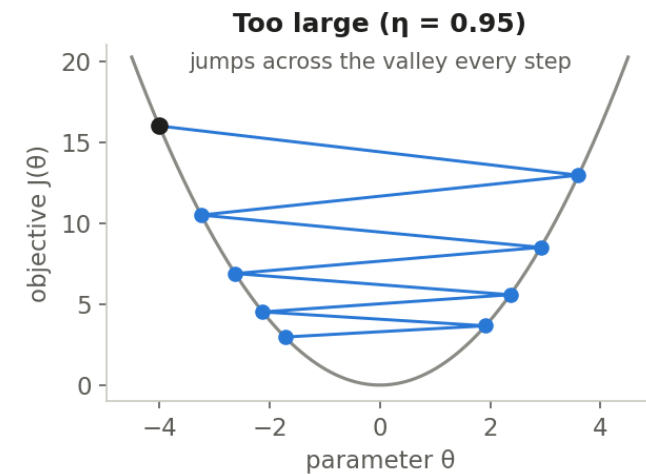
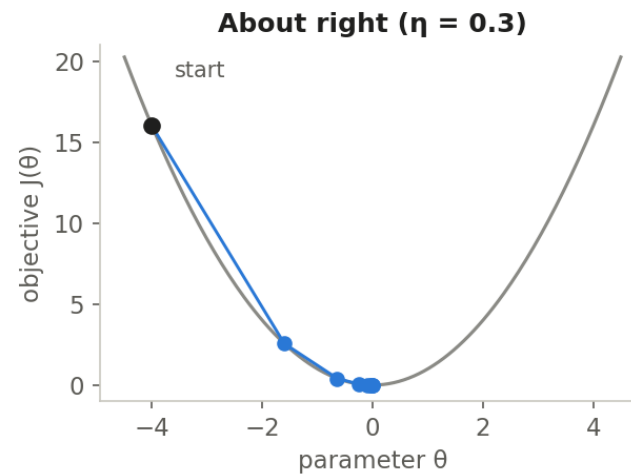
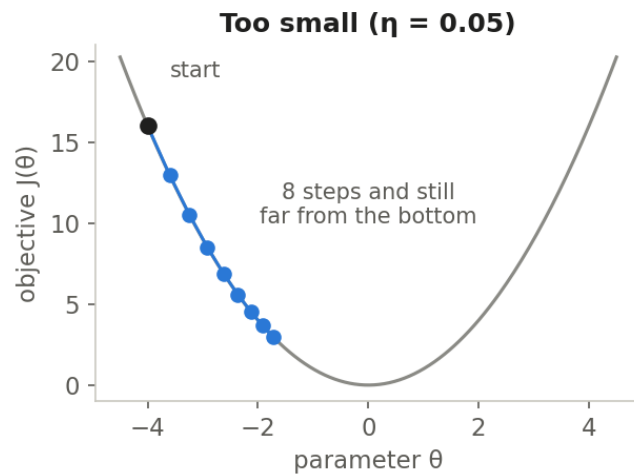
$$\theta^{(t+1)} - \theta^* = (1 - \eta\lambda) (\theta^{(t)} - \theta^*)$$

How big the learning rate can be depends on this

The safe range is

Each step multiplies θ by the same factor, 0.4.
Near a minimizer with curvature λ , the slope is
 $J'(\theta) = \lambda(\theta - \theta^*)$
So subtracting θ^* from both sides of the update gives:

Gradient descent: step downhill; the step size η sets how far



Stochastic Gradient Descent

Gradient Descent:

Calculating the slope with Gradient Descent requires every training example, which is expensive

The loss is an average over all N training examples, each with its own loss ℓ_i , so the gradient is an average too

$$J(\theta) = \frac{1}{N} \sum_{i=1}^N J^{(i)}(\theta) \quad \Rightarrow \quad \nabla J(\theta) = \frac{1}{N} \sum_{i=1}^N \nabla J^{(i)}(\theta)$$

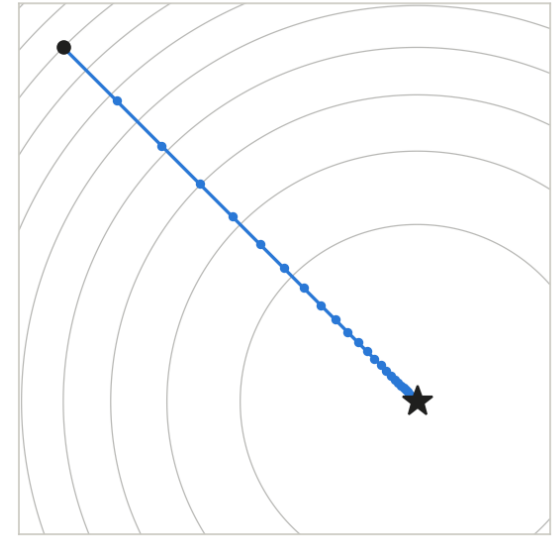
Stochastic Gradient Descent:

SGD picks a random **minibatch** B of b examples and averages only those

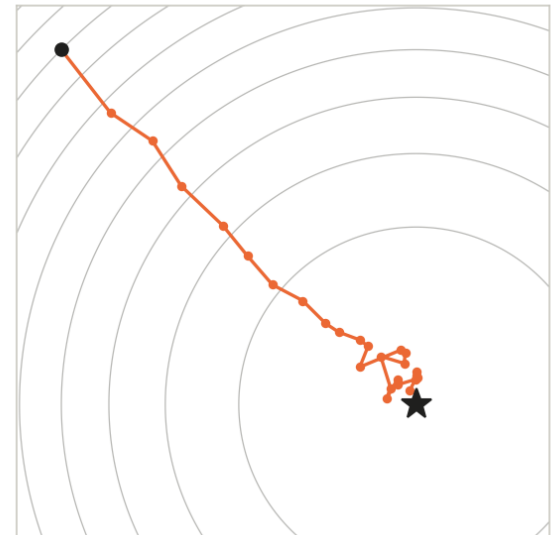
$$\mathbf{g}^{(t)} = \frac{1}{B} \sum_{i \in \mathcal{B}^{(t)}} \nabla J^{(i)}(\theta^{(t)}), \quad \theta^{(t+1)} = \theta^{(t)} - \eta \mathbf{g}^{(t)}$$

MNIST has $N = 60,000$ training images. Full-batch gradient descent makes 1 update per pass over the data; SGD with $b = 128$ makes about 470, each one 470 times cheaper

Full-batch gradient descent: smooth



Stochastic (minibatch) GD: noisy but cheap



Problems with SGD

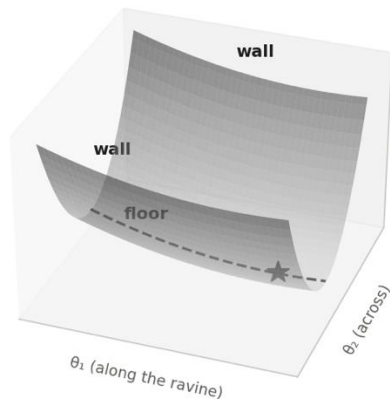
Real loss landscapes are often shaped like long, narrow ravines.

$$J(\boldsymbol{\theta}) = 0.1 \theta_1^2 + 2 \theta_2^2, \quad \nabla J(\boldsymbol{\theta}) = [0.2 \theta_1, 4 \theta_2]^\top$$
$$\lambda_1 = 0.2, \lambda_2 = 4, \kappa = 20$$

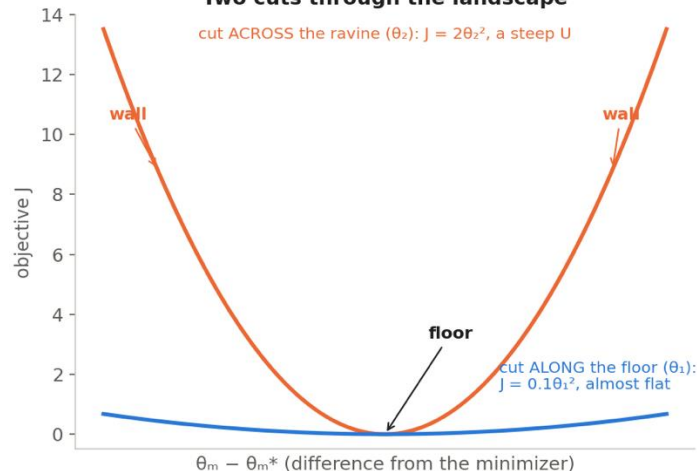
$$\theta_1^{(t+1)} = (1 - 0.2 \eta) \theta_1^{(t)}, \quad \theta_2^{(t+1)} = (1 - 4 \eta) \theta_2^{(t)}$$

The ravine: a long valley floor between two steep walls

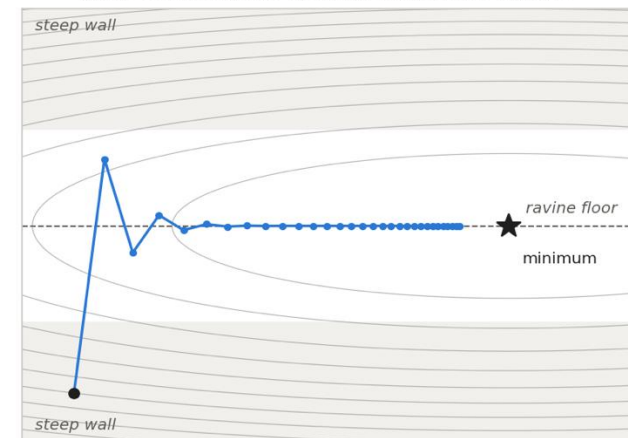
The landscape: a long trough with steep sides



Two cuts through the landscape

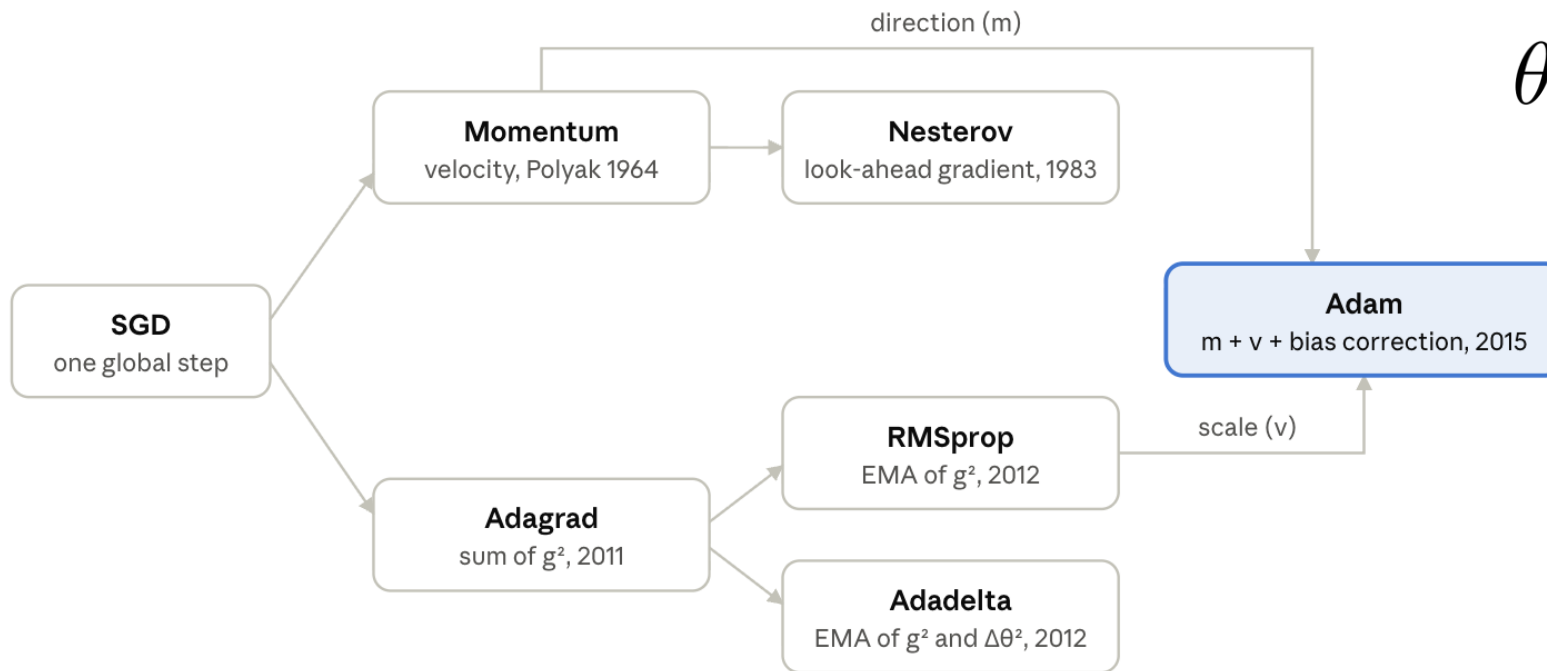


Seen from above: SGD bounces wall to wall



1. **The direction problem.** Real loss landscapes often look like long, narrow ravines: steep walls on the sides, a gentle slope along the floor. The steepest direction points mostly across the ravine, not along it. So SGD bounces from wall to wall, zig-zagging, while making slow progress along the floor.
2. **The step-size problem.** SGD uses one learning rate for every weight. Some weights sit in steep directions and need tiny steps, others in flat directions and need big ones. Any single learning rate is too big for some and too small for others.

The Overall Picture

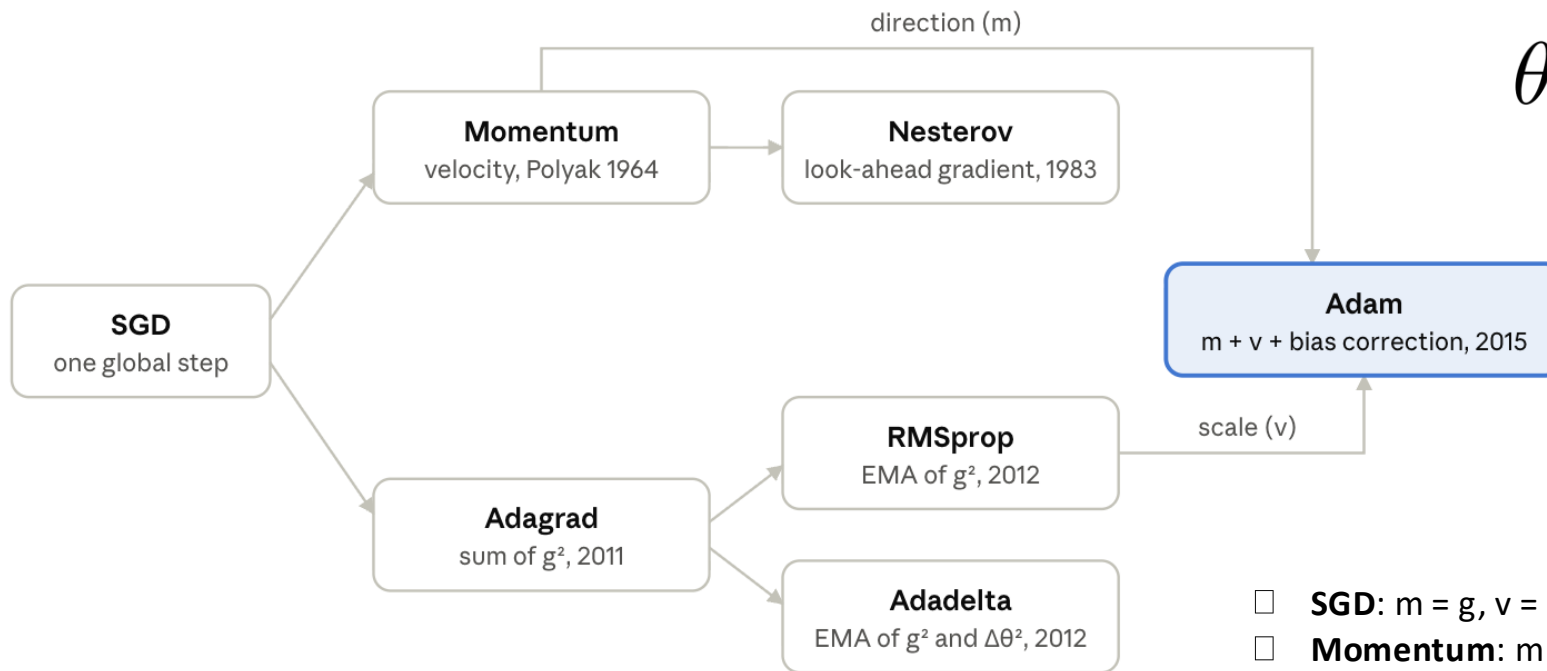


$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} \odot m_t$$

Every method is a choice of

- m (what direction to move) and
- v (how to rescale each coordinate)

The Overall Picture



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

Every method is a choice of

- m (what direction to move) and
- v (how to rescale each coordinate)

- **SGD:** $m = g$, $v = 1$
- **Momentum:** $m = \text{EMA-style sum of past gradients}$, $v = 1$.
- **Nesterov** evaluates the gradient at a look-ahead point)
- **Adagrad:** $m = g$, $v = \text{sum of squared gradients}$.
- **RMSprop:** $m = g$, $v = \text{EMA of squared gradients}$.
- **Adam:** $m = \text{EMA of gradients}$, $v = \text{EMA of squared gradients}$, both bias-corrected.

Momentum



Let us replace the hiker with a ball
A ball doesn't change direction at every bump; it carries speed from before.

- Pushes that keep coming from the same side build up speed, and
- pushes that alternate left and right cancel out.

Momentum

$$\theta^{(t+1)} = \theta^{(t)} - \eta \mathbf{m}^{(t)}$$

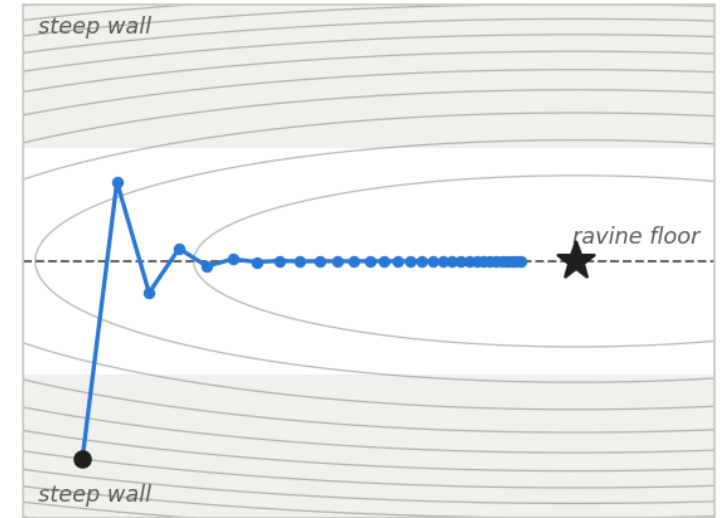
$$\mathbf{m}^{(t)} = \gamma \mathbf{m}^{(t-1)} + \mathbf{g}^{(t)}$$

Keep a **velocity** $\mathbf{m}$ (one number per parameter, starting at 0)

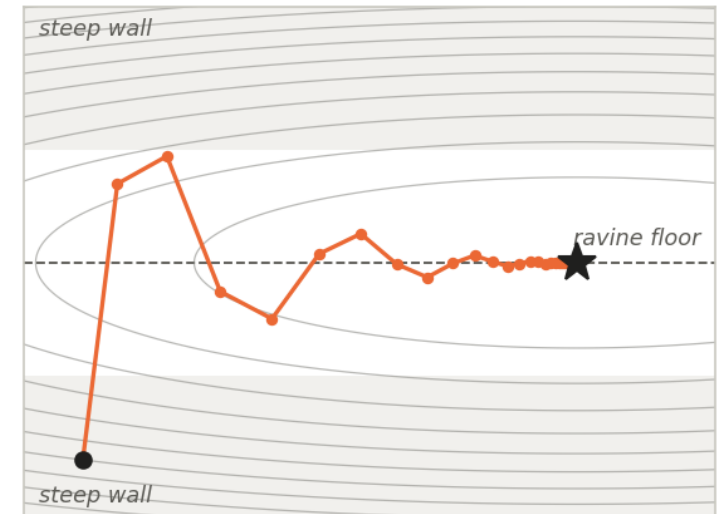
At each step:

1. keep a fraction γ of the old velocity

SGD



SGD + momentum ($\gamma = 0.5$)



Momentum

$$\theta^{(t+1)} = \theta^{(t)} - \eta \mathbf{m}^{(t)}$$

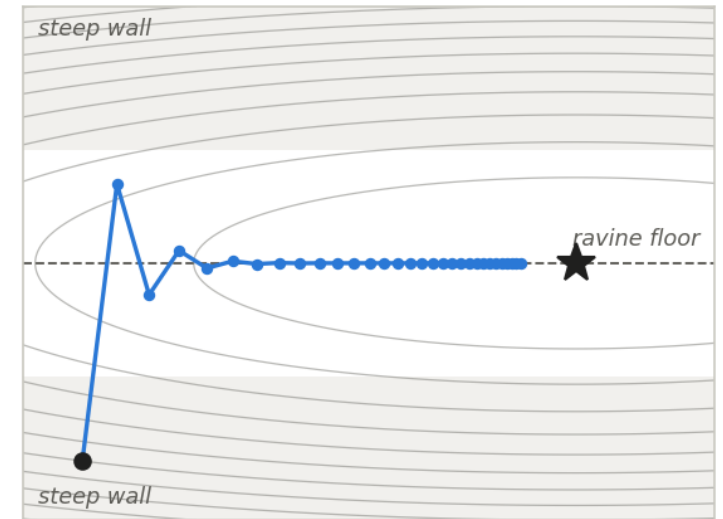
$$\mathbf{m}^{(t)} = \gamma \mathbf{m}^{(t-1)} + \mathbf{g}^{(t)}$$

Keep a **velocity** $\mathbf{m}$ (one number per parameter, starting at 0)

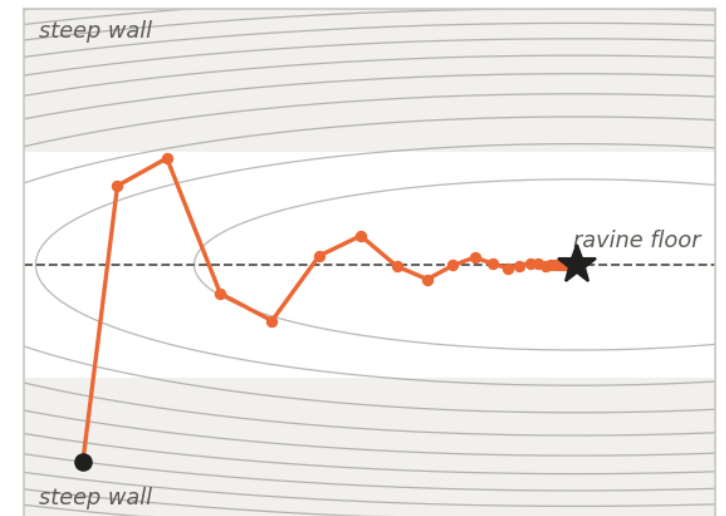
At each step:

1. keep a fraction γ of the old velocity
2. add the new gradient, and

SGD



SGD + momentum ($\gamma = 0.5$)



Momentum

$$\theta^{(t+1)} = \theta^{(t)} - \eta \mathbf{m}^{(t)}$$

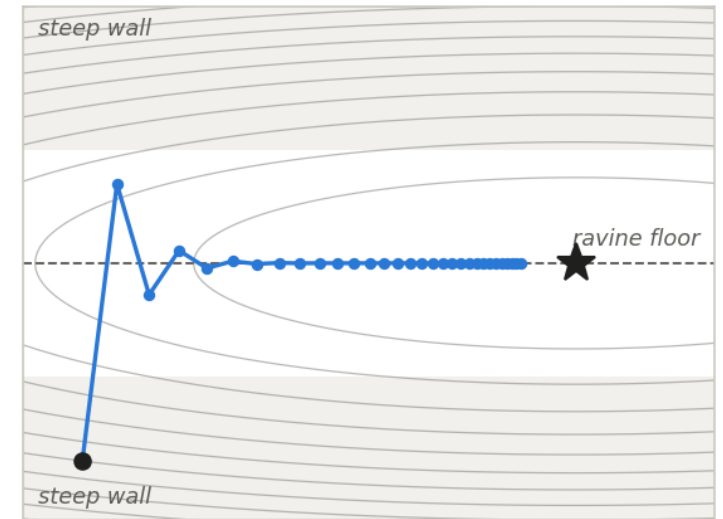
$$\mathbf{m}^{(t)} = \gamma \mathbf{m}^{(t-1)} + \mathbf{g}^{(t)}$$

Keep a **velocity** $\mathbf{m}$ (one number per weight, starting at 0)

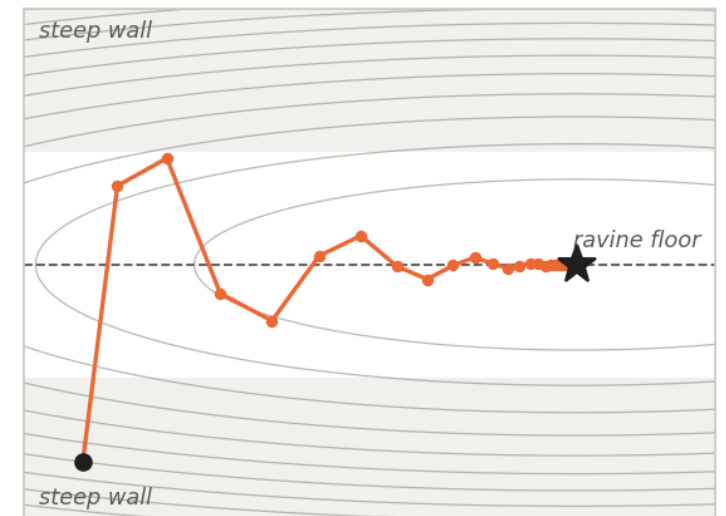
At each step:

1. keep a fraction γ of the old velocity
2. add the new gradient, and
3. move along the velocity

SGD



SGD + momentum ($\gamma = 0.5$)



Momentum

$$\theta^{(t+1)} = \theta^{(t)} - \eta \mathbf{m}^{(t)}$$

$$\mathbf{m}^{(t)} = \gamma \mathbf{m}^{(t-1)} + \mathbf{g}^{(t)}$$

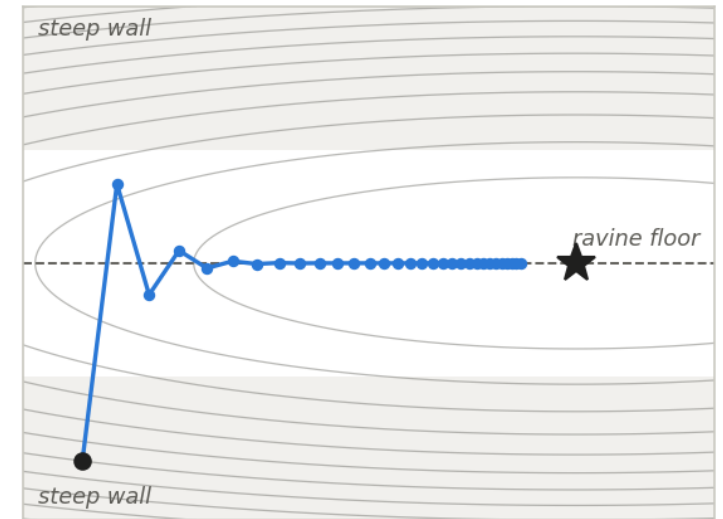
Keep a **velocity** $\mathbf{m}$ (one number per weight, starting at 0)

At each step:

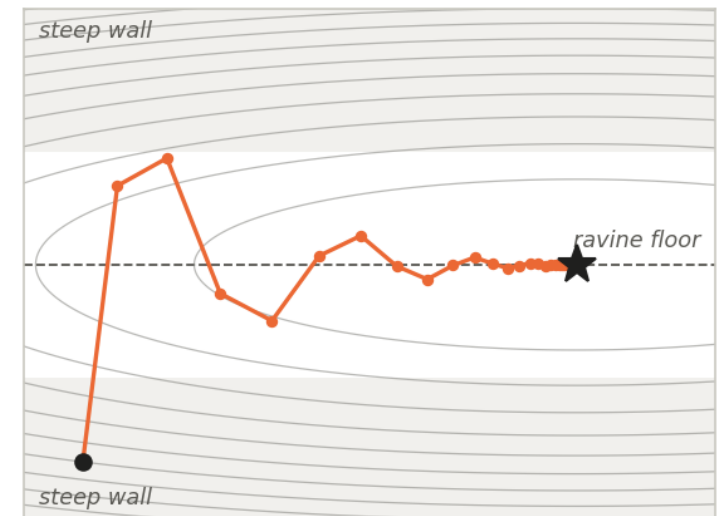
1. keep a fraction γ of the old velocity
2. add the new gradient, and
3. move along the velocity

- $\gamma = 0$ is plain SGD
- γ close to 1 means very little friction
- γ is set to 0.9

SGD



SGD + momentum ($\gamma = 0.5$)



Momentum

$$\theta^{(t+1)} = \theta^{(t)} - \eta \mathbf{m}^{(t)}$$

$$\mathbf{m}^{(t)} = \gamma \mathbf{m}^{(t-1)} + \mathbf{g}^{(t)}$$

Keep a **velocity** $\mathbf{m}$ (one number per weight, starting at 0)

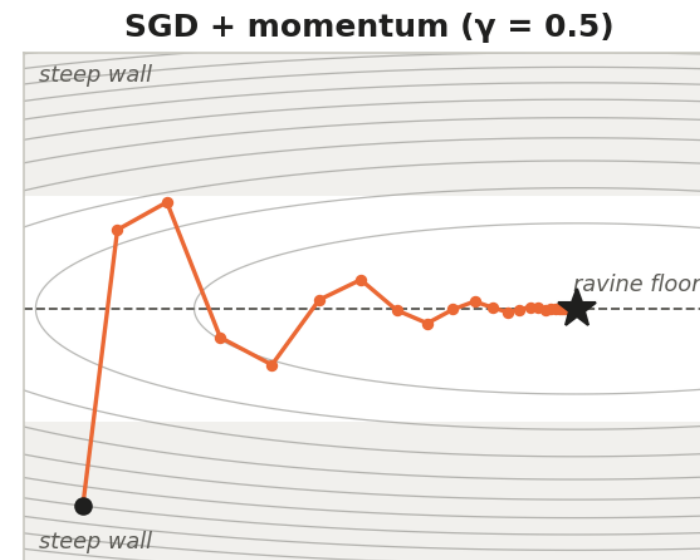
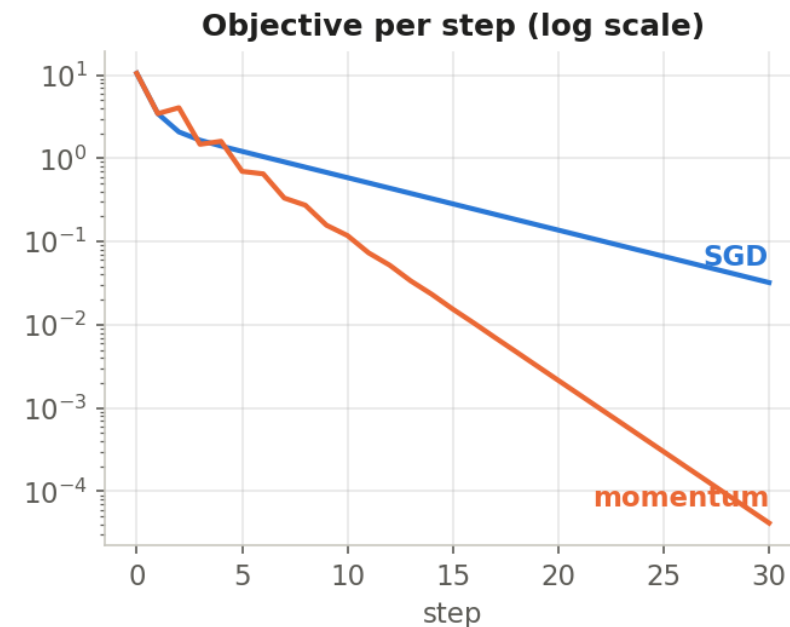
At each step:

1. keep a fraction γ of the old velocity
2. add the new gradient, and
3. move along the velocity

- $\gamma = 0$ is plain SGD
- γ close to 1 means very little friction
- γ is set to 0.9

The **velocity** It is a sum of all past gradients, with older ones counting less

$$\mathbf{m}^{(t)} = \mathbf{g}^{(t)} + \gamma \mathbf{g}^{(t-1)} + \gamma^2 \mathbf{g}^{(t-2)} + \dots + \gamma^{t-1} \mathbf{g}^{(1)}$$



Momentum

The **velocity** It is a sum of all past gradients, older ones counting less

$$\mathbf{m}^{(t)} = \mathbf{g}^{(t)} + \gamma \mathbf{g}^{(t-1)} + \gamma^2 \mathbf{g}^{(t-2)} + \dots + \gamma^{t-1} \mathbf{g}^{(1)}$$

Case 1: the gradient keeps pointing the same way (along the ravine floor). Take $g = 1$ every step and $\gamma = 0.9$:

Step	1	2	3	4	...	long run
b	1	1.9	2.71	3.44	...	10

$$m_m^{(t)} \rightarrow g(1 + \gamma + \gamma^2 + \dots) = \frac{g}{1-\gamma} = 10g$$

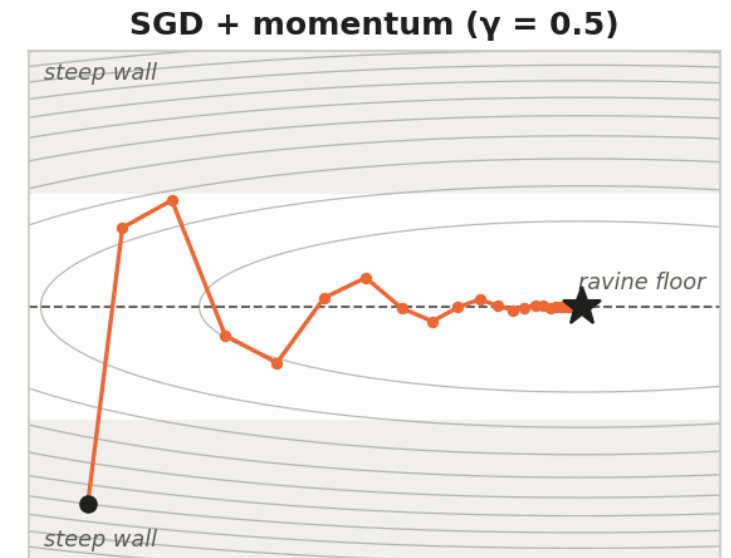
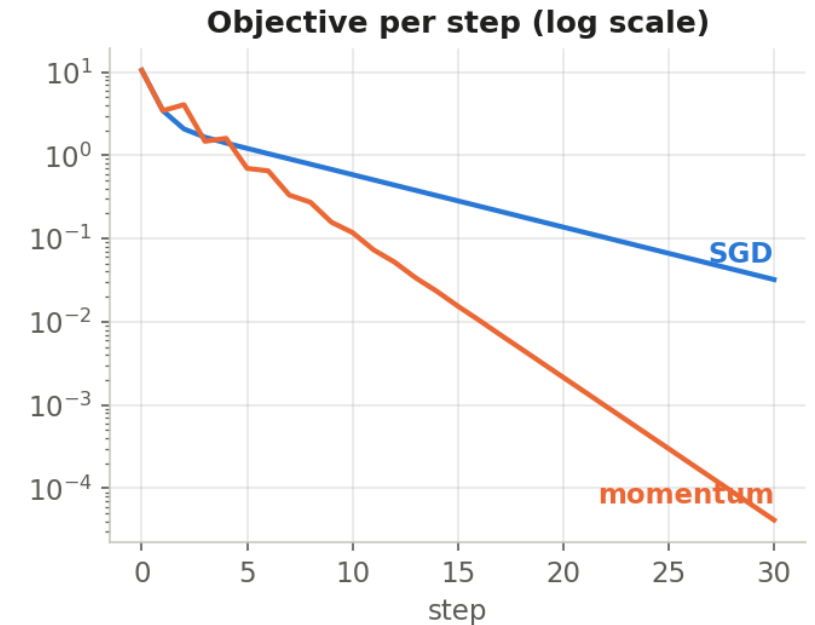
Consistent directions get amplified

Case 2: the gradient flips sign every step (bouncing between the walls). Take $g = +1, -1, +1, \dots$: $b = 1, -0.1, 0.91, -0.18, \dots$

The signs alternate inside the sum, so it settles at a size of only

$$\frac{g}{1+\gamma} \approx 0.53g$$

Back-and-forth directions get damped

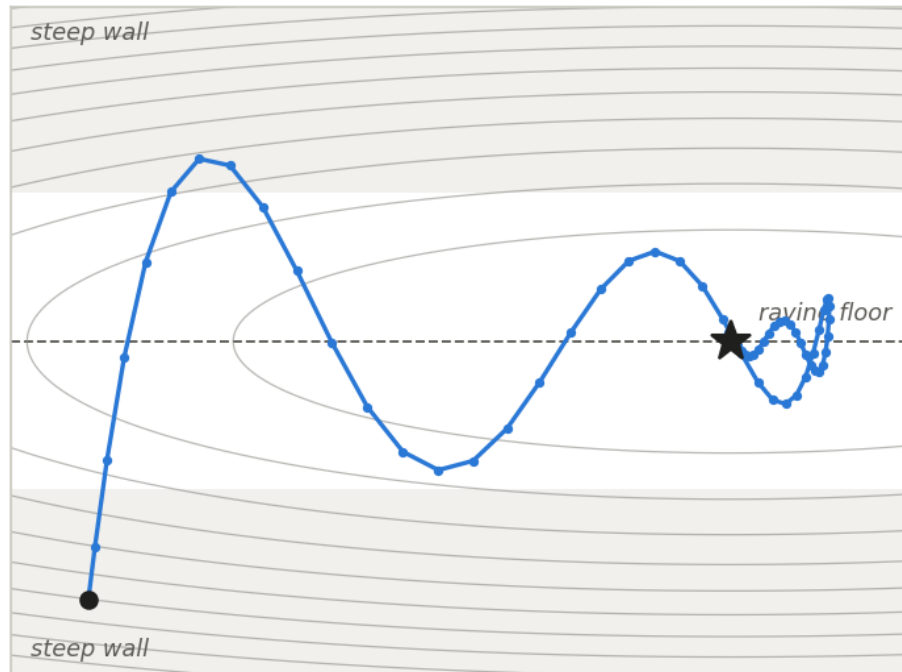


Momentum

New problem: overshoot

A heavy ball builds up speed and rolls past the bottom before friction stops it. With large γ it can circle the minimizer for a long time

Momentum ($\gamma = 0.9$): overshoots, circles

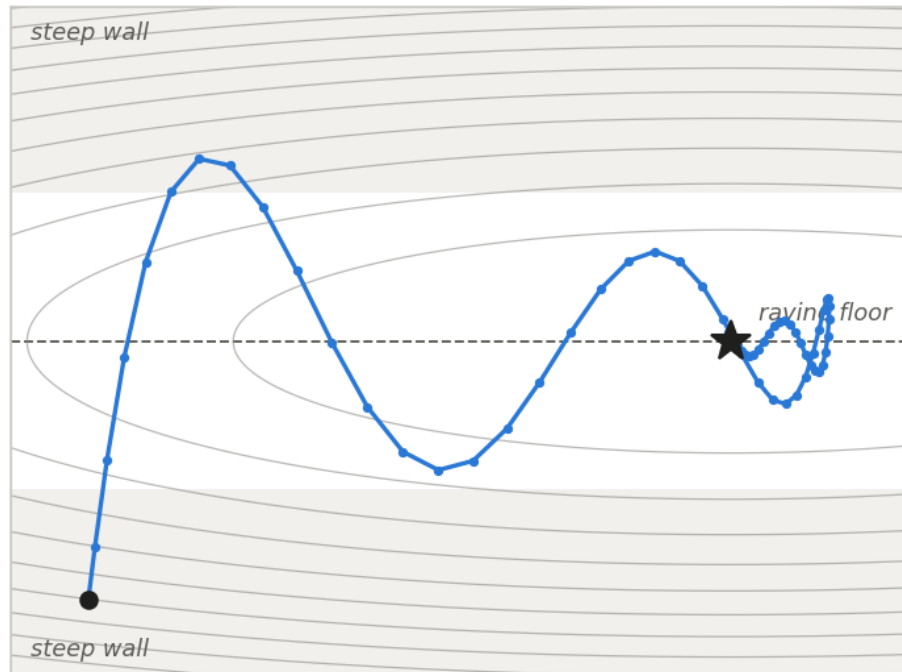


Momentum

New problem: overshoot

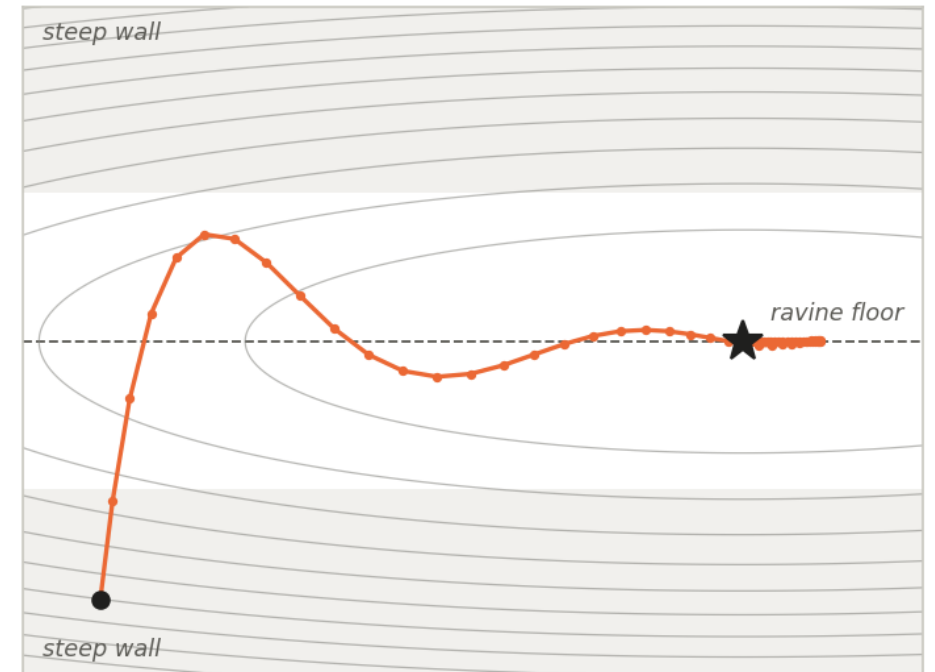
A heavy ball builds up speed and rolls past the bottom before friction stops it. With large γ it can circle the minimizer for a long time

Momentum ($\gamma = 0.9$): overshoots, circles



Solution: Nesterov

Nesterov ($\gamma = 0.9$): brakes early



Nesterov

Momentum vs Nesterov

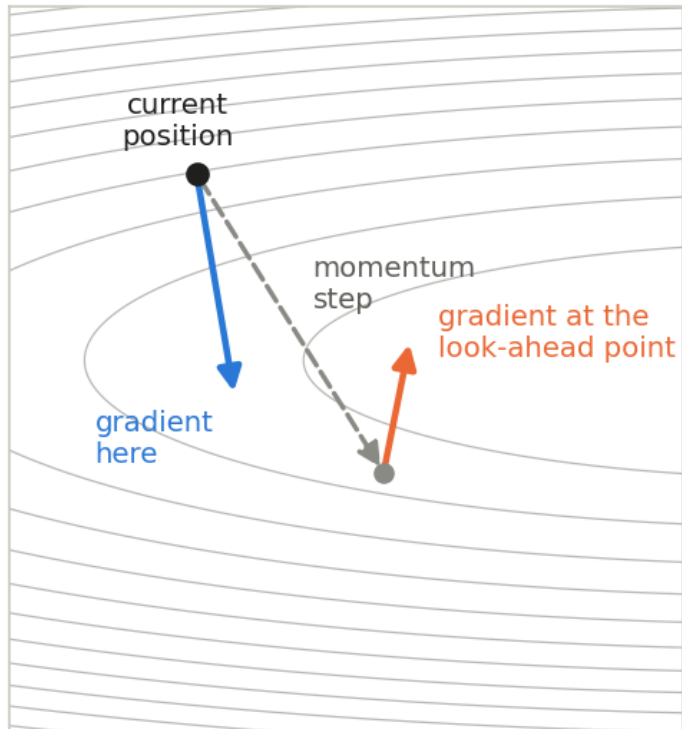
Every momentum step is really two moves:

- a **momentum step** along the accumulated velocity
- the gradient *corrects* it.

Classical momentum computes that correction from the gradient at the current parameters (**where it started**).

Nesterov computes it at the **look-ahead point**, where the momentum step lands.

That is the only difference

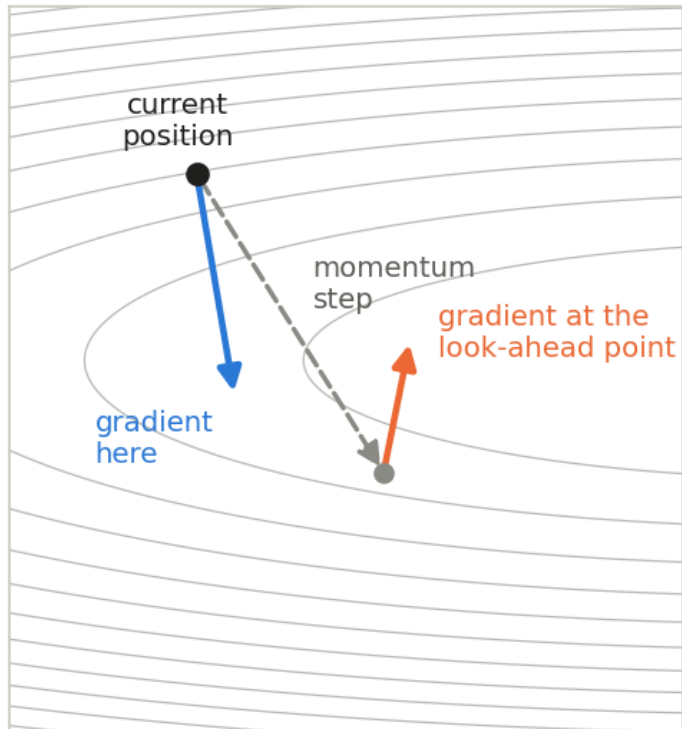


Nesterov

Momentum vs Nesterov

Every momentum step is really two moves:

- a **momentum step** along the accumulated velocity
- the gradient *corrects* it.



Classical momentum computes that correction from the gradient at the current parameters (**where it started**).

Nesterov computes it at the **look-ahead point**, where the momentum step lands.

That is the only difference

1. Look ahead: to where the momentum step would land

$$\tilde{\theta}^{(t)} = \theta^{(t)} - \eta \gamma \mathbf{m}^{(t-1)}$$

look-ahead
point

step
size

momentum
coefficient

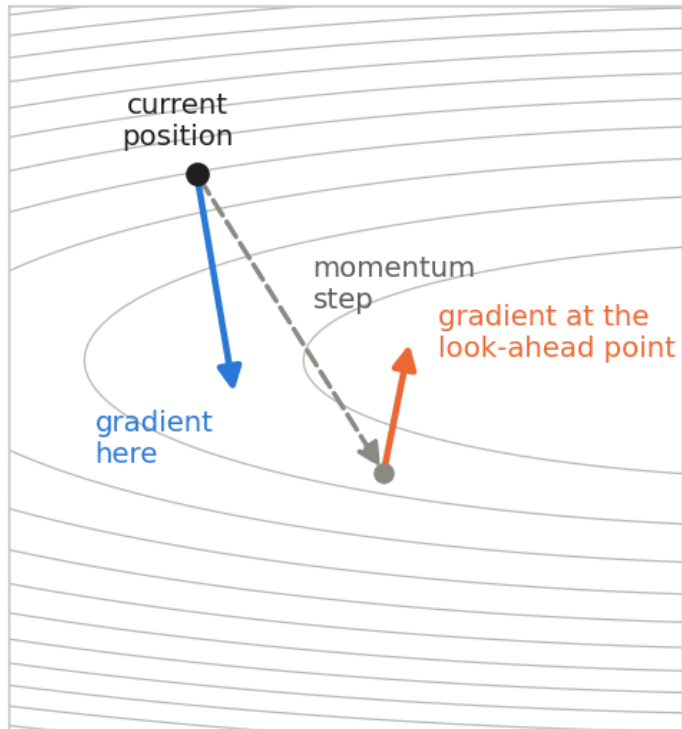
velocity

Nesterov

Momentum vs Nesterov

Every momentum step is really two moves:

- a **momentum step** along the accumulated velocity
- the gradient *corrects* it.



Classical momentum computes that correction from the gradient at the current parameters (**where it started**).

Nesterov computes it at the **look-ahead point**, where the momentum step lands.

That is the only difference

1. Look ahead: to where the momentum step would land

$$\tilde{\theta}^{(t)} = \theta^{(t)} - \eta \gamma \mathbf{m}^{(t-1)}$$

2. Measure the gradient there (one gradient computation, at $\tilde{\theta}$ instead of θ_t)

$$\nabla J(\tilde{\theta}^{(t)})$$

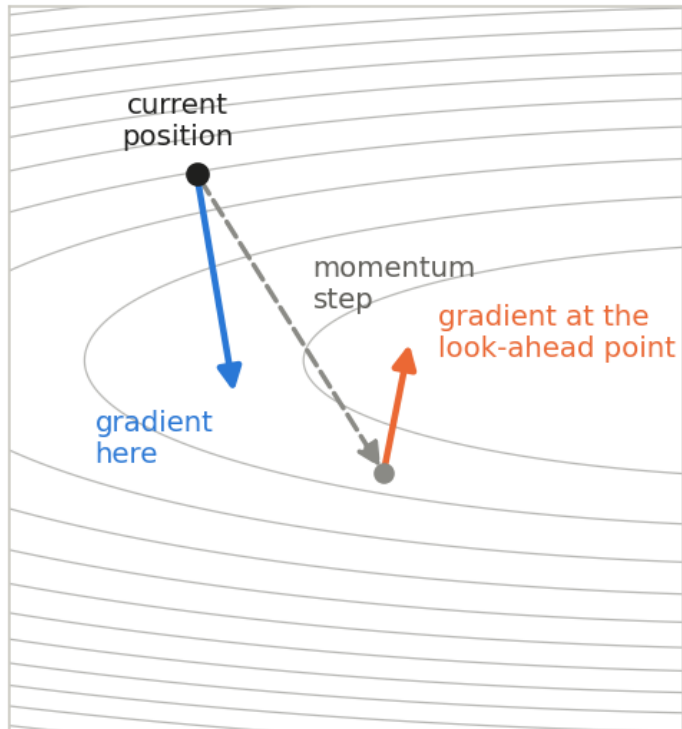
Gradient at
the look-
ahead point

Nesterov

Momentum vs Nesterov

Every momentum step is really two moves:

- a **momentum step** along the accumulated velocity
- the gradient *corrects* it.



Classical momentum computes that correction from the gradient at the current parameters (**where it started**).

Nesterov computes it at the **look-ahead point**, where the momentum step lands.

That is the only difference

1. Look ahead: to where the momentum step would land

2. Measure the gradient there (one gradient computation, at $\tilde{\theta}$ instead of θ_t)

3. Update the velocity: keep γ of the old velocity and add the look-ahead slope

3. Update the velocity: keep γ of the old velocity and add the look-ahead slope

$$\nabla J(\tilde{\theta}^{(t)})$$

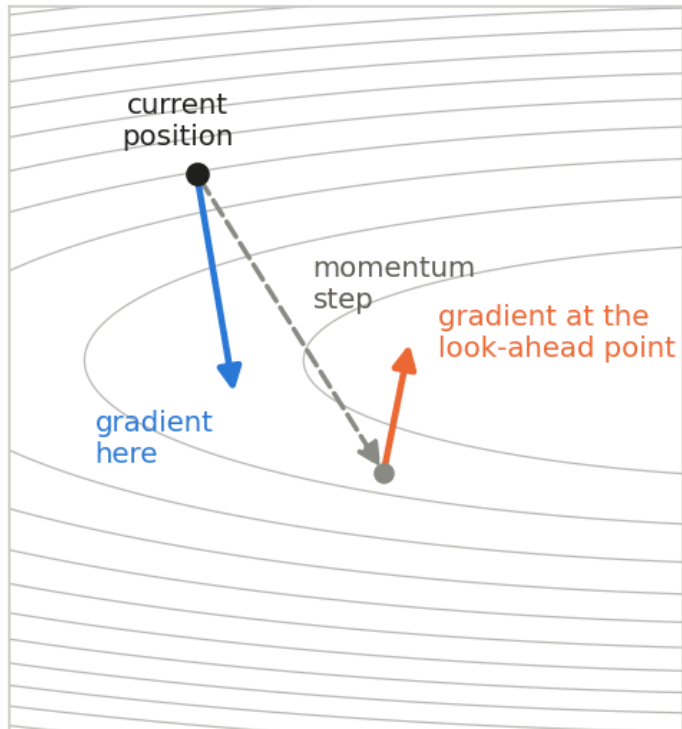
$$\mathbf{m}^{(t)} = \gamma \mathbf{m}^{(t-1)} + \nabla J(\tilde{\theta}^{(t)})$$

Nesterov

Momentum vs Nesterov

Every momentum step is really two moves:

- a **momentum step** along the accumulated velocity
- the gradient *corrects* it.



Classical momentum computes that correction from the gradient at the current parameters (**where it started**).

Nesterov computes it at the **look-ahead point**, where the momentum step lands.

That is the only difference

1. Look ahead: to where the momentum step would land

2. Measure the gradient there (one gradient computation, at $\tilde{\theta}$ instead of θ_t)

3. Update the velocity: keep γ of the old velocity and add the look-ahead slope

4. Update the parameters: move along the new velocity

3. Update the velocity: keep γ of the old velocity and add the look-ahead slope

$$\nabla J(\tilde{\theta}^{(t)})$$

$$\mathbf{m}^{(t)} = \gamma \mathbf{m}^{(t-1)} + \nabla J(\tilde{\theta}^{(t)})$$

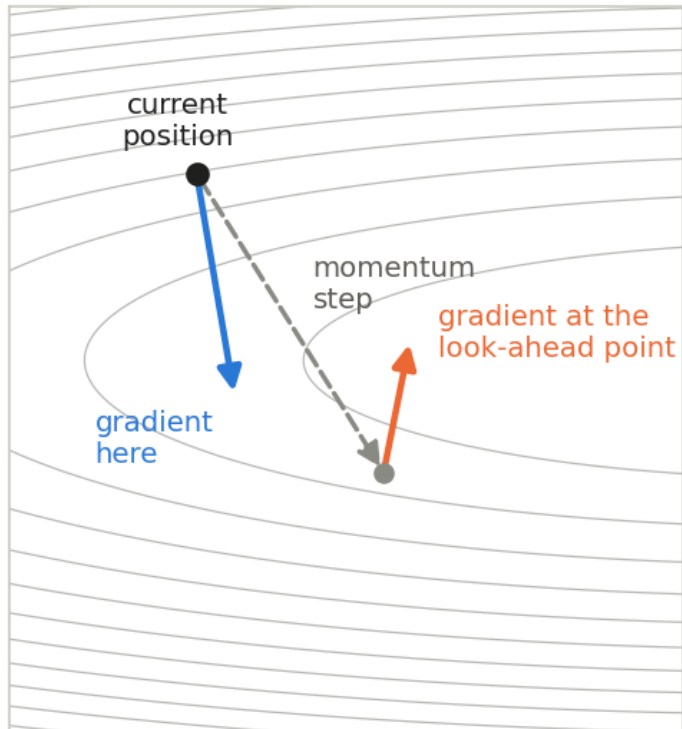
$$\theta^{(t+1)} = \theta^{(t)} - \eta \mathbf{m}^{(t)}$$

Nesterov

Momentum vs Nesterov

Every momentum step is really two moves:

- a **momentum step** along the accumulated velocity
- the gradient *corrects* it.



Substitute step 3 into step 4 and the update splits into the two moves:

$$\theta^{(t+1)} = \underbrace{\theta^{(t)} - \eta \gamma \mathbf{m}^{(t-1)}}_{\text{momentum step to } \tilde{\theta}^{(t)}} - \underbrace{\eta \nabla J(\tilde{\theta}^{(t)})}_{\text{gradient correction}}$$

1. Look ahead: to where the momentum step would land

2. Measure the gradient there (one gradient computation, at $\tilde{\theta}$ instead of θ_t)

3. Update the velocity: keep γ of the old velocity and add the look-ahead slope

4. Update the parameters: move along the new velocity

3. Update the velocity: keep γ of the old velocity and add the look-ahead slope

$$\nabla J(\tilde{\theta}^{(t)})$$

$$\mathbf{m}^{(t)} = \gamma \mathbf{m}^{(t-1)} + \nabla J(\tilde{\theta}^{(t)})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \mathbf{m}^{(t)}$$

Nesterov

$$J(\theta) = \frac{1}{2}\theta^2$$

$$J'(\theta) = \theta$$

$$\theta^* = 0$$

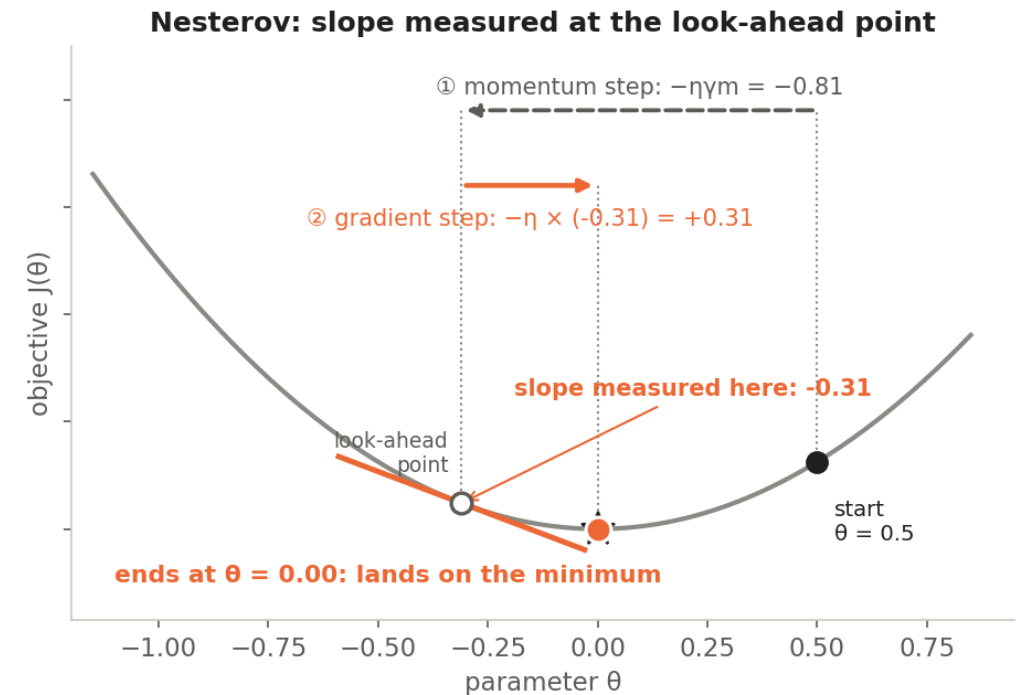
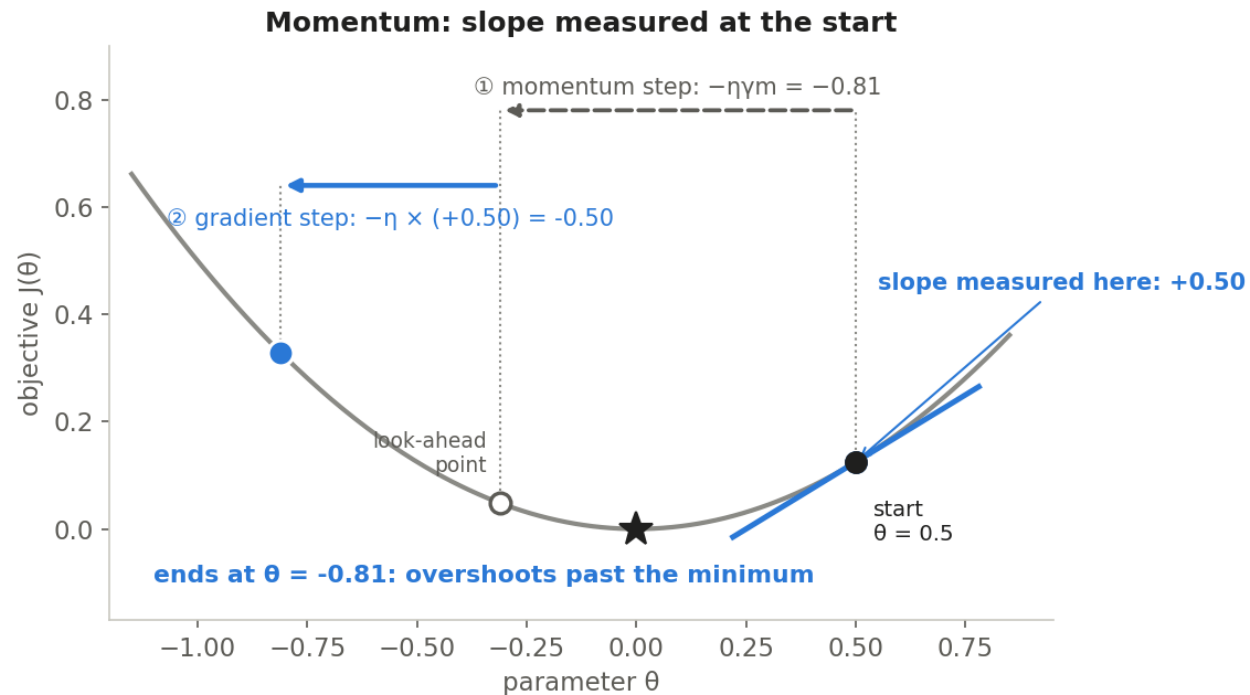
$$\eta = 1$$

$$\gamma = 0.9$$

$$\theta^{(t)} = 0.5$$

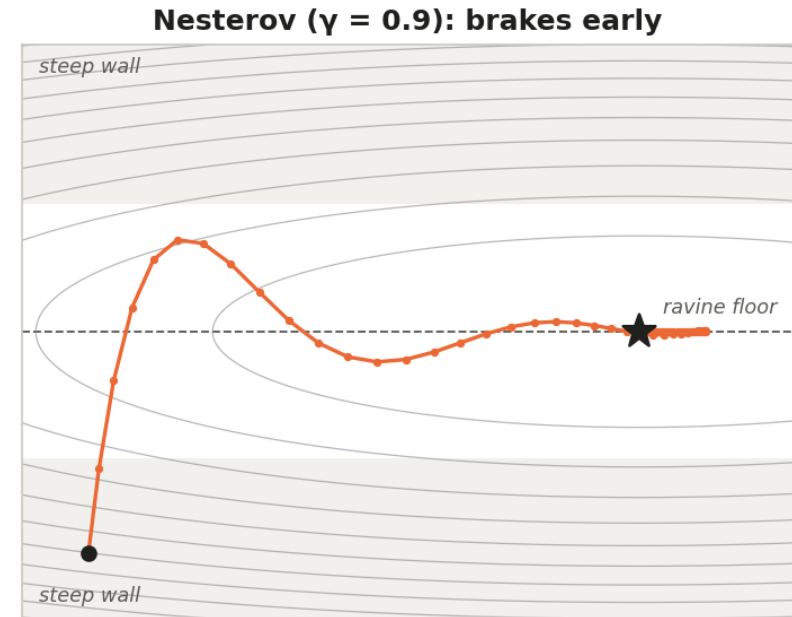
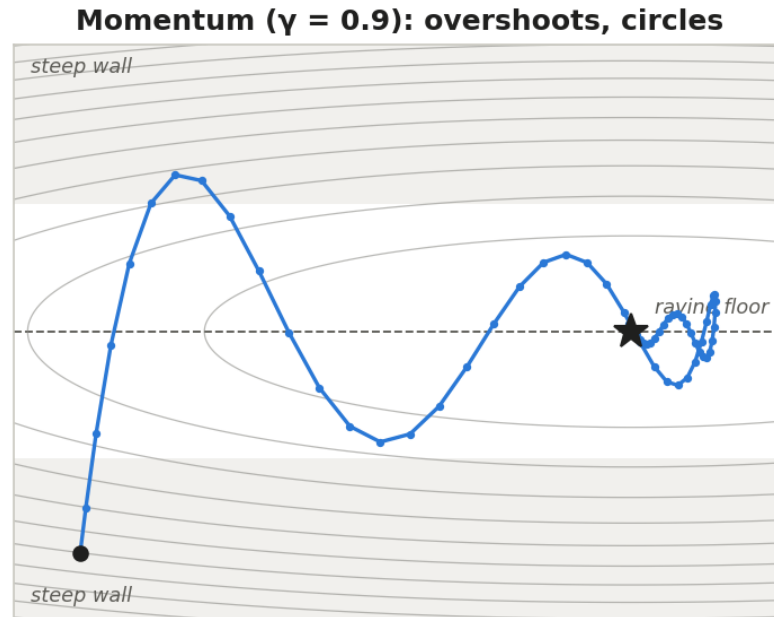
$$m^{(t-1)} = 0.9$$

Both take the momentum step first; the only difference is where the gradient for the correction is measured



Nesterov

With $\gamma = 0.9$ and the same learning rate, momentum swings past the minimum and circles it, while Nesterov settles smoothly



Why it brakes where it should?

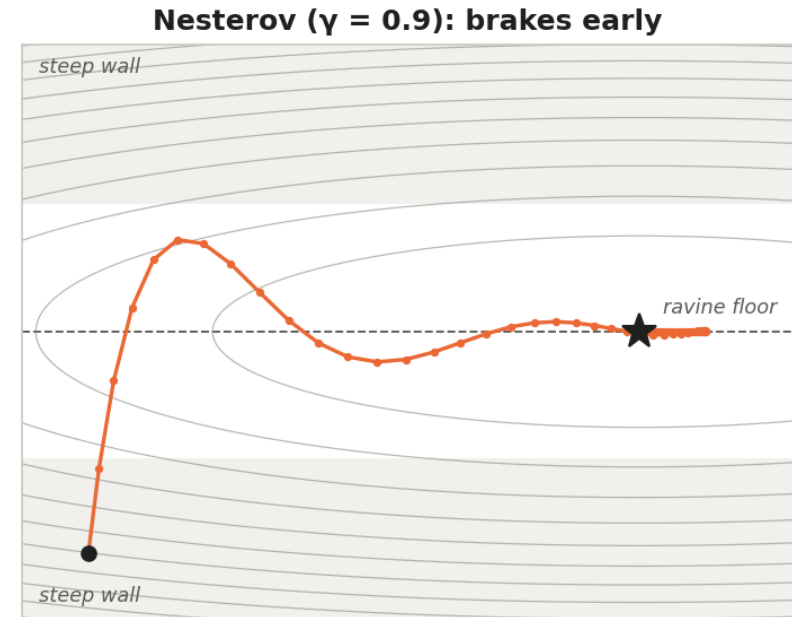
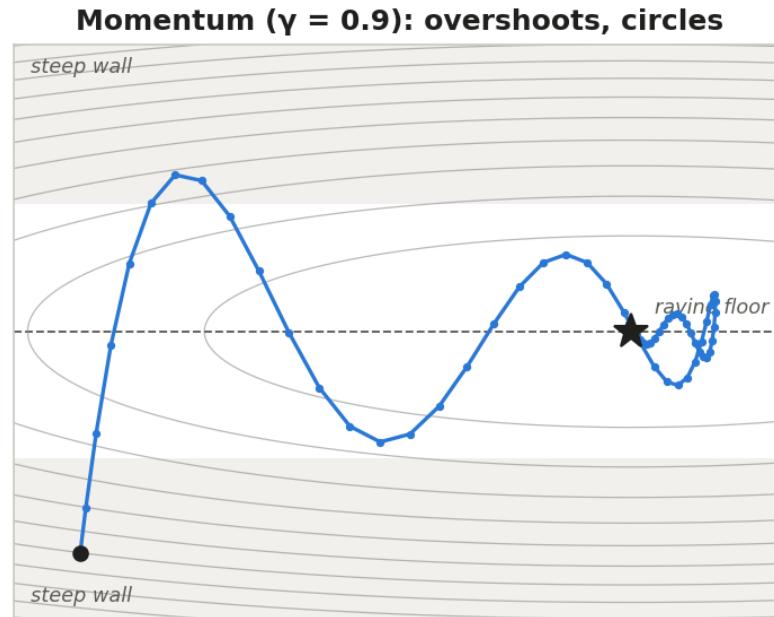
Near a minimizer with curvature λ , the look-ahead gradient is

$$J'(\tilde{\theta}^{(t)}) = \lambda (\tilde{\theta}^{(t)} - \theta^*) = \underbrace{\lambda (\theta^{(t)} - \theta^*)}_{\text{gradient at } \theta^{(t)}} - \underbrace{\eta \lambda \cdot \gamma m^{(t-1)}}_{\text{braking term}}$$

- In flat directions (small λ) the braking term is tiny and Nesterov keeps its speed.
- In steep directions (large λ) it cancels much of the momentum step.
- Each direction effectively gets a momentum coefficient of $\gamma(1 - \eta\lambda)$

Nesterov

With $\gamma = 0.9$ and the same learning rate, momentum swings past the minimum and circles it, while Nesterov settles smoothly



Why it brakes where it should?

Near a minimizer with curvature λ , the look-ahead gradient is

$$J'(\tilde{\theta}^{(t)}) = \lambda (\tilde{\theta}^{(t)} - \theta^*) = \underbrace{\lambda (\theta^{(t)} - \theta^*)}_{\text{gradient at } \theta^{(t)}} - \underbrace{\eta \lambda \cdot \gamma m^{(t-1)}}_{\text{braking term}}$$

- In flat directions (small λ) the braking term is tiny and Nesterov keeps its speed.
- In steep directions (large λ) it cancels much of the momentum step.
- Each direction effectively gets a momentum coefficient of $\gamma(1 - \eta\lambda)$

Both momentum methods use one step size for every parameter.

Adagrad

In the ravine, one step size η has to serve two very different directions:

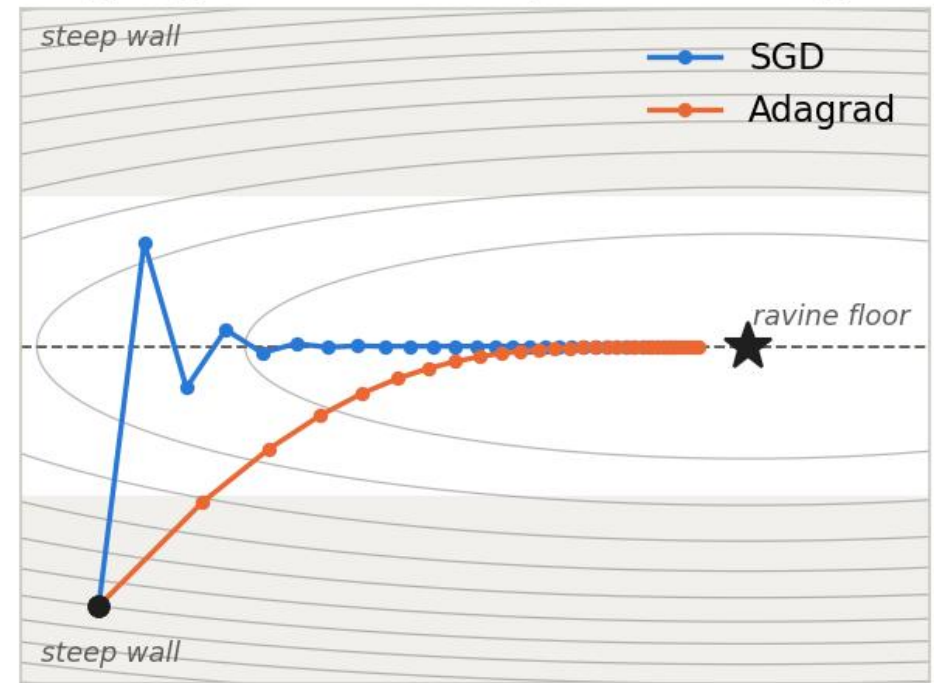
- the steep one (large curvature λ) needs small steps,
- the flat one (small λ) needs big steps

Momentum only softens this.

Adagrad gives each parameter θ_m its own step size, set by how large its gradients have been so far (based on its own history).

- a parameter whose gradients have been large (a steep direction) gets smaller steps
- a parameter whose gradients have been small gets bigger step.

Each weight gets its own step size: a straighter path



Adagrad

For every parameter, keep a running **sum** of **squared** gradients ($\mathbf{s}^{(0)} = \mathbf{0}$):

$$\mathbf{s}^{(t)} = \mathbf{s}^{(t-1)} + \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$$

$\mathbf{g}^{(t)}$ is the **minibatch gradient at iteration t**

$$\mathbf{g}^{(t)} = \frac{1}{B} \sum_{i \in \mathcal{B}^{(t)}} \nabla J^{(i)}(\boldsymbol{\theta}^{(t)})$$

Element-wise product results with squaring the gradient, entry by entry, where the sign is dropped and only the size is kept.

$$\mathbf{g} \odot \mathbf{g} = \begin{bmatrix} g_1^2 \\ g_2^2 \\ \vdots \\ g_M^2 \end{bmatrix}$$

Adagrad

For every parameter, keep a running **sum** of **squared** gradients ($\mathbf{s}^{(0)} = \mathbf{0}$):

Then divide each entry of the gradient by the square root of its sum:

$$\mathbf{s}^{(t)} = \mathbf{s}^{(t-1)} + \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{\mathbf{g}^{(t)}}{\sqrt{\mathbf{s}^{(t)}} + \epsilon}$$

The small constant ϵ (e.g. 10^{-10}) only prevents division by zero

global step
size

$\sqrt{s_m^{(t)}}$ measures how large the gradients of θ_m have been so far.

Adagrad

For every parameter, keep a running **sum** of squared gradients ($\mathbf{s}^{(0)} = \mathbf{0}$):

Then divide each entry of the gradient by the square root of its sum:

$$\mathbf{s}^{(t)} = \mathbf{s}^{(t-1)} + \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$$

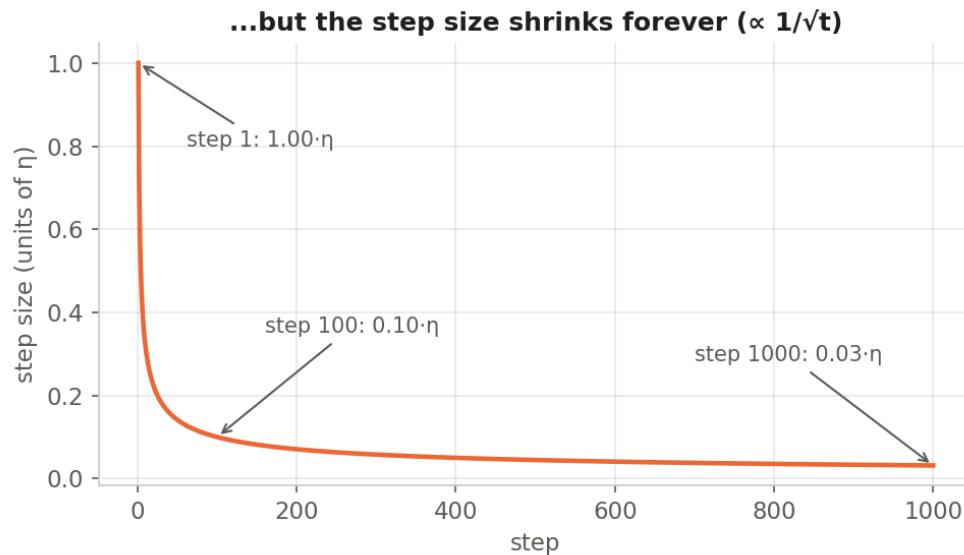
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{\mathbf{g}^{(t)}}{\sqrt{\mathbf{s}^{(t)}} + \epsilon}$$

For one parameter the division becomes a step size of its own. Parameter θ_m has its own **effective step size**

$$\frac{\eta}{\sqrt{s_m^{(t)}} + \epsilon}$$

$$\theta_m^{(t+1)} = \theta_m^{(t)} - \underbrace{\frac{\eta}{\sqrt{s_m^{(t)}} + \epsilon}}_{\text{effective step size } \eta_m^{(t)}} g_m^{(t)}$$

Adagrad



Three consequences of dividing by $\sqrt{s_m}$:

1. **The gradient's size cancels.** Plain gradient descent would move θ_2 twenty times farther than θ_1 . Adagrad moves them the same amount, so the flat direction is no longer neglected.
2. **The step keeps shrinking.** With a constant gradient g , $s^{(t)} = tg^2$, so the step is $\frac{\eta g}{g\sqrt{t}} = \frac{\eta}{\sqrt{t}}$. After 100 steps it is 0.1η ; after 10,000 steps, 0.01η .
3. **Gradient scale doesn't matter.** Multiply every gradient of a weight by 1,000: G grows by 1,000,000, $\sqrt{G}$ by 1,000, and the step is unchanged. Weights with naturally tiny or huge gradients are put on the same footing.

$$\theta^{(t+1)} = \theta^{(t)} - \eta \frac{\mathbf{g}^{(t)}}{\sqrt{\mathbf{s}^{(t)} + \epsilon}}$$

Example with two parameters (Ignore ϵ)

- θ_1 sits in a **flat** direction and always sees gradient 0.1
- θ_2 sits in a **steep** direction and always sees gradient 2.

Step t	$s_1^{(t)}$	Step taken by θ_1	$s_2^{(t)}$	Step taken by θ_2
1	0.01	$\eta \cdot 0.1 / 0.1 = 1.00\eta$	4	$\eta \cdot 2 / 2 = 1.00\eta$
2	0.02	$\eta \cdot 0.1 / 0.141 = 0.71\eta$	8	$\eta \cdot 2 / 2.83 = 0.71\eta$
3	0.03	0.58η	12	0.58η
4	0.04	0.50η	16	0.50η

Big and small gradients get the same step.

Adagrad divides each gradient by its own past size, so both parameters move the same amount, $\eta/\sqrt{t}$. The flat direction is no longer neglected, and the steep one no longer overshoots.

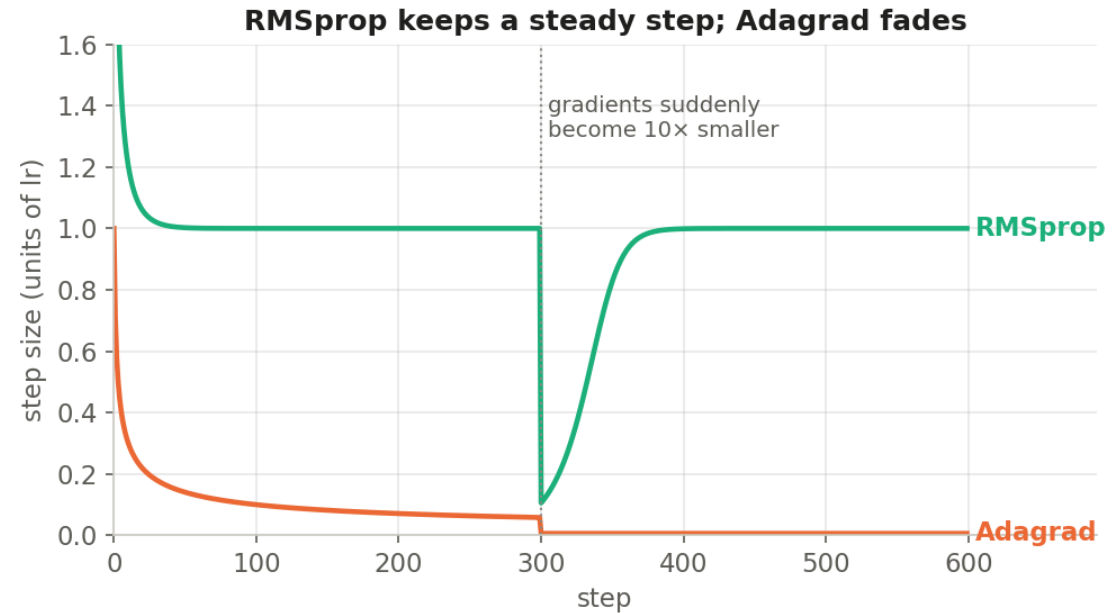
$$\frac{\eta g}{g\sqrt{t}} = \frac{\eta}{\sqrt{t}}$$

RMSProp

Adagrad divides each slope by a running total that keeps growing and never forgets.

RMSprop divides by the size of *recent* gradients only, using an **exponential moving average**, so each parameter's step size can go down and back up and never wears out.

The step size then reflects how big the gradients have been *recently*.



RMSProp

Adagrad divides each slope by a running total that keeps growing and never forgets.

RMSprop divides by the size of *recent* gradients only, using an **exponential moving average**, so each parameter's step size can go down and back up and never wears out.

The step size then reflects how big the gradients have been *recently*.

Start with $\mathbf{v}^{(0)} = \mathbf{0}$. At every iteration $t = 1, 2, \dots$:

1. **Square the gradient, entry by entry.** $\mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$ has entries $(g_m^{(t)})^2$. Squaring drops the sign and keeps the size.

2. **Update the moving average of the squares:**

$$\mathbf{v}^{(t)} = \beta \mathbf{v}^{(t-1)} + (1 - \beta) \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$$

decay rate: the fraction of the old average kept each step.

For $\beta = 0.9$, v keeps 90% of what it knew and adds 10% of the newest squared slope

EMA of squared gradients; $v_m^{(t)}$ estimates the mean square $\mathbb{E}[g_m^2]$, starts at 0

RMSProp

Adagrad divides each slope by a running total that keeps growing and never forgets.

RMSprop divides by the size of *recent* gradients only, using an **exponential moving average**, so each parameter's step size can go down and back up and never wears out.

The step size then reflects how big the gradients have been *recently*.

Start with $\mathbf{v}^{(0)} = \mathbf{0}$. At every iteration $t = 1, 2, \dots$:

1. Square the gradient, entry by entry. $\mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$ has entries $(g_m^{(t)})^2$. Squaring drops the sign and keeps the size.

2. Update the moving average of the squares:

$$\mathbf{v}^{(t)} = \beta \mathbf{v}^{(t-1)} + (1 - \beta) \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$$

3. Take the square root. $\sqrt{v_m^{(t)}}$ is the root mean square (RMS) of the recent gradients of θ_m : their typical size, in the same units as the gradient.

RMSProp

Adagrad divides each slope by a running total that keeps growing and never forgets.

RMSprop divides by the size of *recent* gradients only, using an **exponential moving average**, so each parameter's step size can go down and back up and never wears out.

The step size then reflects how big the gradients have been *recently*.

Start with $\mathbf{v}^{(0)} = \mathbf{0}$. At every iteration $t = 1, 2, \dots$:

1. **Square the gradient, entry by entry.** $\mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$ has entries $(g_m^{(t)})^2$. Squaring drops the sign and keeps the size.

2. **Update the moving average of the squares:**

$$\mathbf{v}^{(t)} = \beta \mathbf{v}^{(t-1)} + (1 - \beta) \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$$

3. **Take the square root.** $\sqrt{\mathbf{v}_m^{(t)}}$ is the root mean square (RMS) of the recent gradients of θ_m : their typical size, in the same units as the gradient.

4. **Divide and step:**

$$\theta^{(t+1)} = \theta^{(t)} - \eta \frac{\mathbf{g}^{(t)}}{\sqrt{\mathbf{v}^{(t)}} + \epsilon}$$

RMS of the recent
gradients of θ_m

RMSProp

Adagrad divides each slope by a running total that keeps growing and never forgets.

RMSprop divides by the size of *recent* gradients only, using an **exponential moving average**, so each parameter's step size can go down and back up and never wears out.

The step size then reflects how big the gradients have been *recently*.

Start with $\mathbf{v}^{(0)} = \mathbf{0}$. At every iteration $t = 1, 2, \dots$:

1. Square the gradient, entry by entry. $\mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$ has entries $(g_m^{(t)})^2$. Squaring drops the sign and keeps the size.

2. Update the moving average of the squares:

$$\mathbf{v}^{(t)} = \beta \mathbf{v}^{(t-1)} + (1 - \beta) \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$$

3. Take the square root. $\sqrt{v_m^{(t)}}$ is the root mean square (RMS) of the recent gradients of θ_m : their typical size, in the same units as the gradient.

4. Divide and step:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{\mathbf{g}^{(t)}}{\sqrt{\mathbf{v}^{(t)} + \epsilon}}$$

Written for one parameter, the division becomes a step size of its own:

$$\theta_m^{(t+1)} = \theta_m^{(t)} - \underbrace{\frac{\eta}{\sqrt{v_m^{(t)} + \epsilon}}}_{\text{effective step size } \eta_m^{(t)}} g_m^{(t)}$$

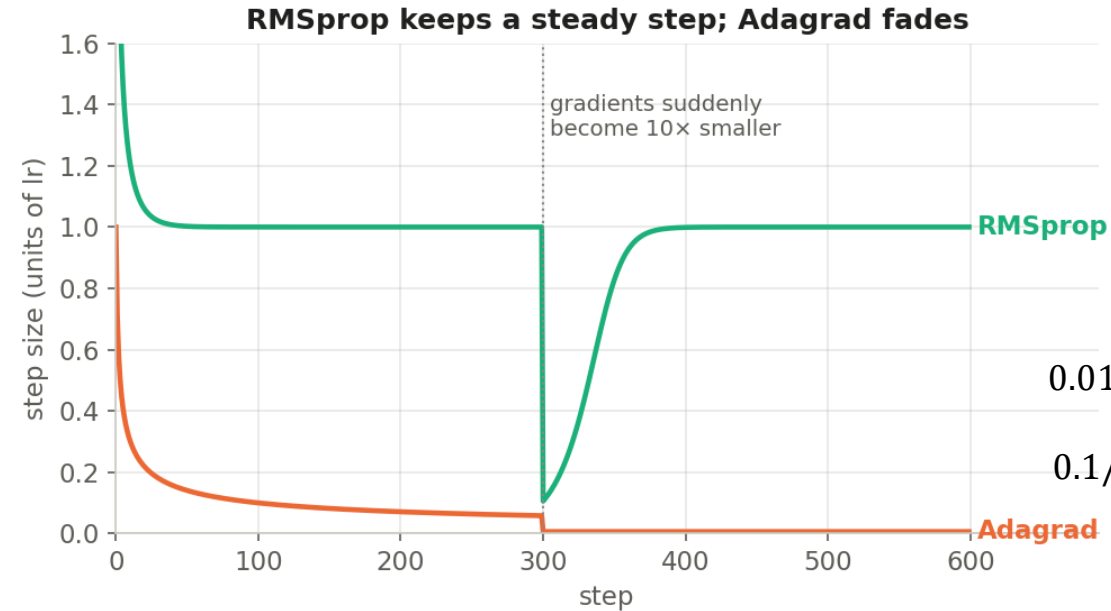
RMSProp

Adagrad vs RMSProp

$\beta = 0.9$, one parameter, ε ignored

Step	Slope	RMS props $\sqrt{v}$	RMS prop step	Adagrad step
1	1	0.32	3.16	1.00
3	1	0.52	1.92	0.58
20	1	0.94	1.07	0.22
300	1	1.00	1.00	0.06
301	0.1	0.95	0.105	0.006
310	0.1	0.60	0.17	0.006
330	0.1	0.23	0.44	0.006
350	0.1	0.12	0.81	0.006
360	0.1	0.11	0.92	0.006

(slope drops)



$$0.01 + 0.99 \times 0.9^{300} \approx 0.01$$

$$0.1 / \sqrt{300} \approx 0.006$$

Why the steps no longer die out?

The step no longer depends on the scale of the gradient. If the gradients of θ_m are steady at some value g , then $v_m \rightarrow g^2$ and

$$\frac{g_m}{\sqrt{v_m}} \approx \frac{g}{\sqrt{g^2}} = \pm 1 \quad \Rightarrow \quad \text{step} \approx \pm \eta$$

Multiply the objective by 100: every gradient grows 100x, $\sqrt{v}$ grows 100x, and the step does not change. Plain SGD's step would grow 100x.

While in Adagrad it is $\frac{\eta}{\sqrt{t}}$

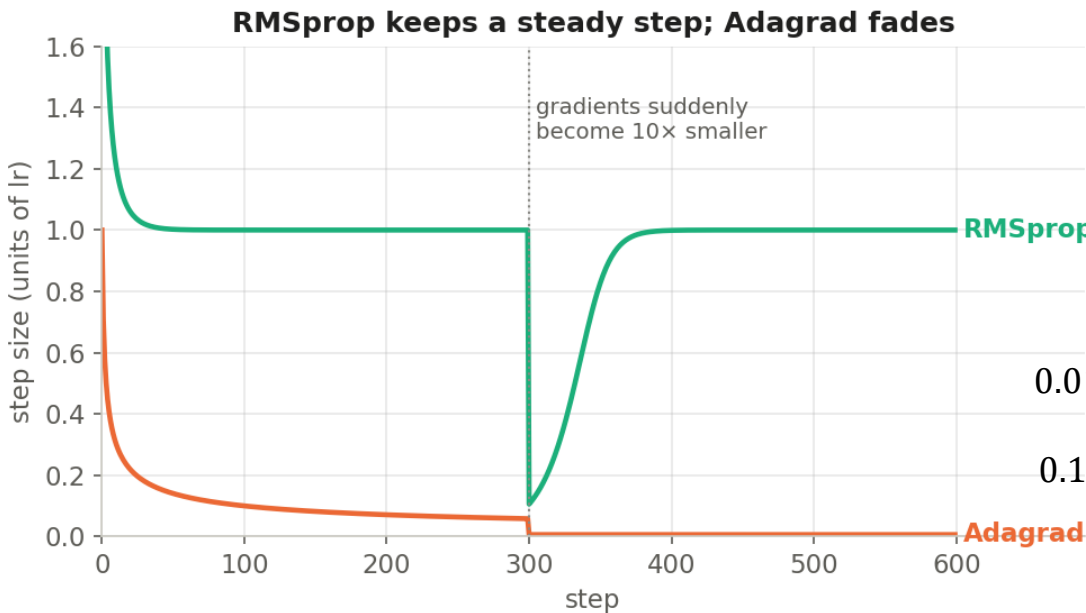
RMSProp

Adagrad vs RMSProp

$\beta = 0.9$, one parameter, ε ignored

Step	Slope	RMS props $\sqrt{v}$	RMS prop step	Adagrad step
1	1	0.32	3.16	1.00
3	1	0.52	1.92	0.58
20	1	0.94	1.07	0.22
300	1	1.00	1.00	0.06
301	0.1	0.95	0.105	0.006
310	0.1	0.60	0.17	0.006
330	0.1	0.23	0.44	0.006
350	0.1	0.12	0.81	0.006
360	0.1	0.11	0.92	0.006

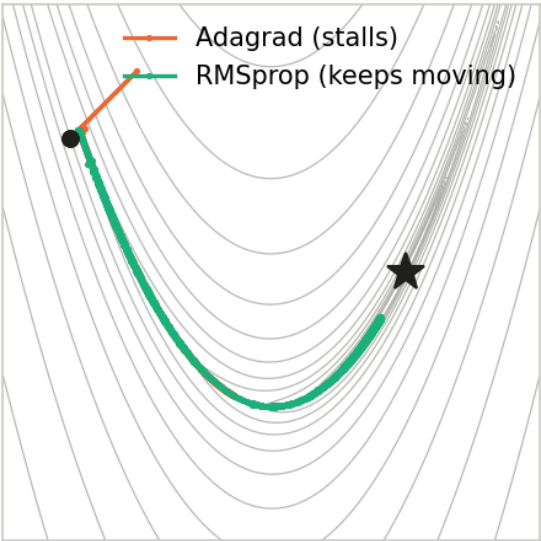
(slope drops)



$$0.01 + 0.99 \times 0.9^{300} \approx 0.01$$

$$0.1 / \sqrt{300} \approx 0.006$$

Curved valley, 1000 steps



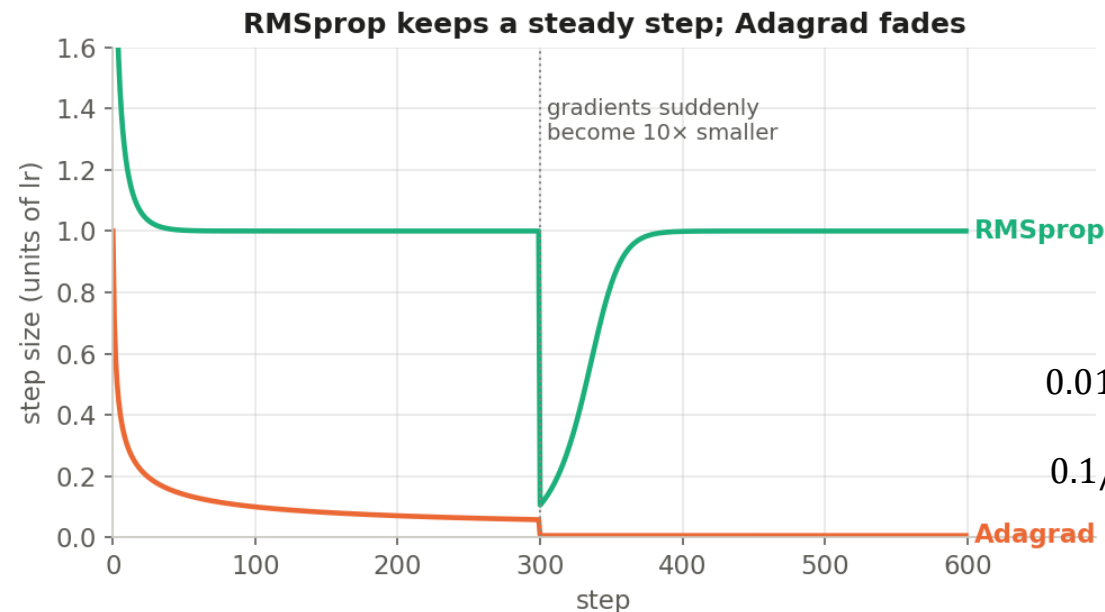
RMSProp

Adagrad vs RMSProp

$\beta = 0.9$, one parameter, ε ignored

Step	Slope	RMS props $\sqrt{v}$	RMS prop step	Adagrad step
1	1	0.32	3.16	1.00
3	1	0.52	1.92	0.58
20	1	0.94	1.07	0.22
300	1	1.00	1.00	0.06
301	0.1	0.95	0.105	0.006
310	0.1	0.60	0.17	0.006
330	0.1	0.23	0.44	0.006
350	0.1	0.12	0.81	0.006
360	0.1	0.11	0.92	0.006

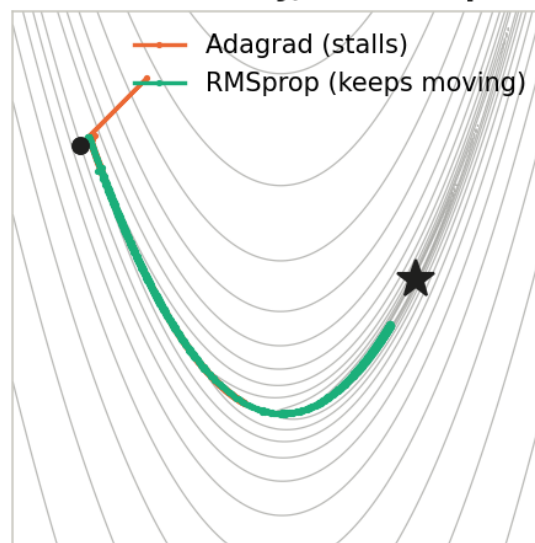
(slope drops)



$$0.01 + 0.99 \times 0.9^{300} \approx 0.01$$

$$0.1 / \sqrt{300} \approx 0.006$$

Curved valley, 1000 steps



RMSprop has no momentum. Each step points along the current gradient only, so in a ravine it still zig-zags somewhat between the walls, and its first steps are inflated by the zero start.

Adam

By now there are two good ideas:

1. **momentum**: average recent gradients for a steady **direction**
2. **RMSprop** : divide by the recent gradient size for a sensible **step size per weight**

Adam ("adaptive moment estimation") uses both, and fixes the zero-start bias that RMSprop left open.

For each parameter, it keeps

- a running average of the gradient (the direction) and
- a running average of the squared gradient (the size),
- and steps by direction divided by size, **with a fix for the first few steps**

Adam

By now there are two good ideas:

1. **momentum**: average recent gradients for a steady **direction**
2. **RMSprop** : divide by the recent gradient size for a sensible **step size per weight**

Adam ("adaptive moment estimation") uses both, and fixes the zero-start bias that RMSprop left open.

The update

Start with $\mathbf{m}^{(0)} = \mathbf{v}^{(0)} = \mathbf{0}$. At every iteration $t = 1, 2, \dots$:

1. **Average the gradients (the direction)**. This is the *first moment*:

$$\mathbf{m}^{(t)} = \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \mathbf{g}^{(t)}$$

Adam's $\mathbf{m}$ is an average and not a plain sum.
The factor $(1 - \beta_1)$ keeps it on the scale of one gradient instead of $1/(1 - \gamma)$ gradients.
Both point in the same direction.

Adam

By now there are two good ideas:

1. **momentum**: average recent gradients for a steady **direction**
2. **RMSprop** : divide by the recent gradient size for a sensible **step size per weight**

Adam ("adaptive moment estimation") uses both, and fixes the zero-start bias that RMSprop left open.

The update

Start with $\mathbf{m}^{(0)} = \mathbf{v}^{(0)} = \mathbf{0}$. At every iteration $t = 1, 2, \dots$:

1. **Average the gradients (the direction)**. This is the *first moment*:

$$\mathbf{m}^{(t)} = \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \mathbf{g}^{(t)}$$

2. **Average the squared gradients (the size)**. This is the *second moment*, exactly RMSprop's $\mathbf{v}$ with decay β_2 :

$$\mathbf{v}^{(t)} = \beta_2 \mathbf{v}^{(t-1)} + (1 - \beta_2) \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$$

Why "moments". In statistics the first moment of X is its mean $\mathbb{E}[X]$ and the second moment is its mean square $\mathbb{E}[X^2]$. Adam keeps EMA estimates of both for every gradient entry.

Adam

By now there are two good ideas:

1. **momentum**: average recent gradients for a steady **direction**
2. **RMSprop** : divide by the recent gradient size for a sensible **step size per weight**

Adam ("adaptive moment estimation") uses both, and fixes the zero-start bias that RMSprop left open.

Why does bias correction matter?

Without it, the first steps are several times too large. With it, the step is the right size from step 1.

The update

Start with $\mathbf{m}^{(0)} = \mathbf{v}^{(0)} = \mathbf{0}$. At every iteration $t = 1, 2, \dots$:

1. **Average the gradients (the direction)**. This is the *first moment*:

$$\mathbf{m}^{(t)} = \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \mathbf{g}^{(t)}$$

2. **Average the squared gradients (the size)**. This is the *second moment*, exactly RMSprop's $\mathbf{v}$ with decay β_2 :

$$\mathbf{v}^{(t)} = \beta_2 \mathbf{v}^{(t-1)} + (1 - \beta_2) \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$$

3. **Bias Correction for the zero start:**

$$\hat{\mathbf{m}}^{(t)} = \frac{\mathbf{m}^{(t)}}{1 - \beta_1^t}, \quad \hat{\mathbf{v}}^{(t)} = \frac{\mathbf{v}^{(t)}}{1 - \beta_2^t}$$

β_1^t and β_2^t are powers (β_1 raised to the power t), not iteration indices.

Adam

By now there are two good ideas:

1. **momentum**: average recent gradients for a steady **direction**
2. **RMSprop** : divide by the recent gradient size for a sensible **step size per weight**

Adam ("adaptive moment estimation") uses both, and fixes the zero-start bias that RMSprop left open.

The step is about η times a “confidence” ratio.

$\hat{\mathbf{m}}/\sqrt{\hat{\mathbf{v}}} \approx \text{mean}/\text{RMS}$ of the gradient, which always lies between -1 and 1 :

- consistent gradients give a full step of about η ;
- noisy gradients that flip sign give tiny steps;
- so no parameter moves much more than η per step, whatever the scale of the objective.

The update

Start with $\mathbf{m}^{(0)} = \mathbf{v}^{(0)} = \mathbf{0}$. At every iteration $t = 1, 2, \dots$:

1. **Average the gradients (the direction)**. This is the *first moment*:

$$\mathbf{m}^{(t)} = \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \mathbf{g}^{(t)}$$

2. **Average the squared gradients (the size)**. This is the *second moment*, exactly RMSprop's $\mathbf{v}$ with decay β_2 :

$$\mathbf{v}^{(t)} = \beta_2 \mathbf{v}^{(t-1)} + (1 - \beta_2) \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$$

3. **Bias Correction for the zero start:**

$$\hat{\mathbf{m}}^{(t)} = \frac{\mathbf{m}^{(t)}}{1 - \beta_1^t}, \quad \hat{\mathbf{v}}^{(t)} = \frac{\mathbf{v}^{(t)}}{1 - \beta_2^t}$$

4. **Step**: the averaged direction divided by the RMS:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{\hat{\mathbf{m}}^{(t)}}{\sqrt{\hat{\mathbf{v}}^{(t)}} + \epsilon}$$

Adam

By now there are two good ideas:

1. **momentum**: average recent gradients for a steady **direction**
2. **RMSprop** : divide by the recent gradient size for a sensible **step size per weight**

Adam ("adaptive moment estimation") uses both, and fixes the zero-start bias that RMSprop left open.

For one parameter:

$$\theta_m^{(t+1)} = \theta_m^{(t)} - \eta \frac{\hat{m}_m^{(t)}}{\sqrt{\hat{v}_m^{(t)} + \epsilon}}$$

The update

Start with $\mathbf{m}^{(0)} = \mathbf{v}^{(0)} = \mathbf{0}$. At every iteration $t = 1, 2, \dots$:

1. **Average the gradients (the direction)**. This is the *first moment*:

$$\mathbf{m}^{(t)} = \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \mathbf{g}^{(t)}$$

2. **Average the squared gradients (the size)**. This is the *second moment*, exactly RMSprop's $\mathbf{v}$ with decay β_2 :

$$\mathbf{v}^{(t)} = \beta_2 \mathbf{v}^{(t-1)} + (1 - \beta_2) \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$$

3. **Bias Correction for the zero start**:

$$\hat{\mathbf{m}}^{(t)} = \frac{\mathbf{m}^{(t)}}{1 - \beta_1^t}, \quad \hat{\mathbf{v}}^{(t)} = \frac{\mathbf{v}^{(t)}}{1 - \beta_2^t}$$

4. **Step**: the averaged direction divided by the RMS:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{\hat{\mathbf{m}}^{(t)}}{\sqrt{\hat{\mathbf{v}}^{(t)} + \epsilon}}$$

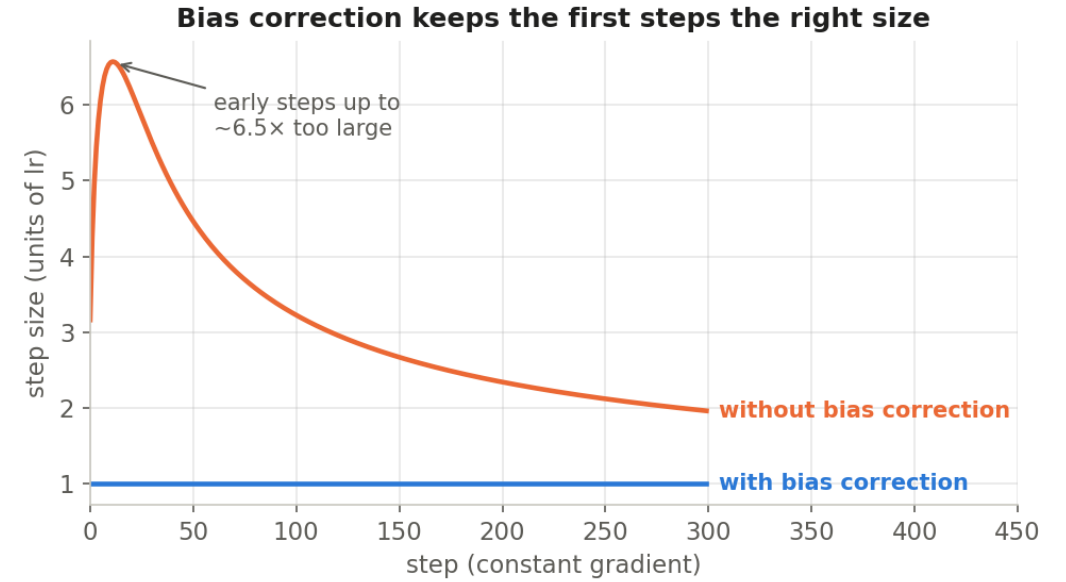
Adam

Consider two parameters on very different scales

θ_1 always sees gradient 0.01 and θ_2 always sees gradient 100.

With bias correction, $\hat{m} = g$ and $\hat{v} = g^2$, so both steps are $\eta \cdot g / |g| = \eta$. Plain SGD would move θ_2 ten thousand times farther than θ_1 .

Why Adam is the default. It inherits momentum's smoothing and RMSprop's per-parameter scaling, works out of the box with $\eta = 10^{-3}$ on most problems, and its step size is roughly bounded by η , which makes η easy to reason about. In practice many models use AdamW, which adds decoupled weight decay on top of this update



ways sees gradient
ient 100.

with bias correction,
 $\eta \cdot g / |g| = \eta$. Plain
and times farth