

07-380 Notes

Pre-reading: Deep Learning Optimization

Carnegie Mellon University
School of Computer Science

Contents

1	The Objective Function	1
2	Slope: The Derivative and the Gradient	2
2.1	One Parameter: The Derivative	2
2.2	Many Parameters: Partial Derivatives and the Gradient	2
3	Curvature: How the Slope Changes	3
3.1	One Parameter: The Second Derivative	3
3.2	Sharp and Wide Bowls	4
3.3	Many Parameters: The Hessian	4
3.4	Eigenvectors, Eigenvalues, and the Condition Number	5
3.5	Up Close, Every Objective Looks Like a Quadratic	6
4	Gradient Descent	6
4.1	The Update	6
4.2	The Learning Rate and Curvature	7
4.3	Stochastic Gradient Descent	7
4.4	Problems with SGD	8
5	Helpful Statistical Tools	9
5.1	Mean, Variance, and Root Mean Square	9
5.2	Exponential Moving Average	10
6	Summary	13

1 The Objective Function

A model has **parameters** θ . For a neural network these are all of the weights and biases, stacked into one long vector $\theta \in \mathbb{R}^M$, where M is often in the millions or billions. 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:

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

where $J^{(i)}(\theta)$ is the loss on training example i . **Training** means finding parameters that make $J(\theta)$ as small as possible.

In 07-280 we minimized objectives like this with gradient descent, and that method is the starting point for this lecture. The lecture asks why plain gradient descent is slow or unstable on deep networks, and what the optimizers used in practice (momentum, RMSProp, Adam) change about it. The answer depends on two properties of the *shape* of J : its **slope**, which is review, and its **curvature**, which is new. This pre-reading covers both, then re-reads gradient descent in that language, and ends with a few statistical tools (mean, variance, root mean square, and the exponential moving average) that the lecture's optimizers are built from.

Notation: θ in bold is the whole parameter vector, θ_m is its m -th entry, and plain θ is used when there is only one parameter. A superscript (t) is the iteration number of the optimizer, and a superscript (i) is a training example.

2 Slope: The Derivative and the Gradient

This section is review from 07-280. It is here to fix notation and to set up the one-parameter pictures that the rest of the notes build on.

2.1 One Parameter: The Derivative

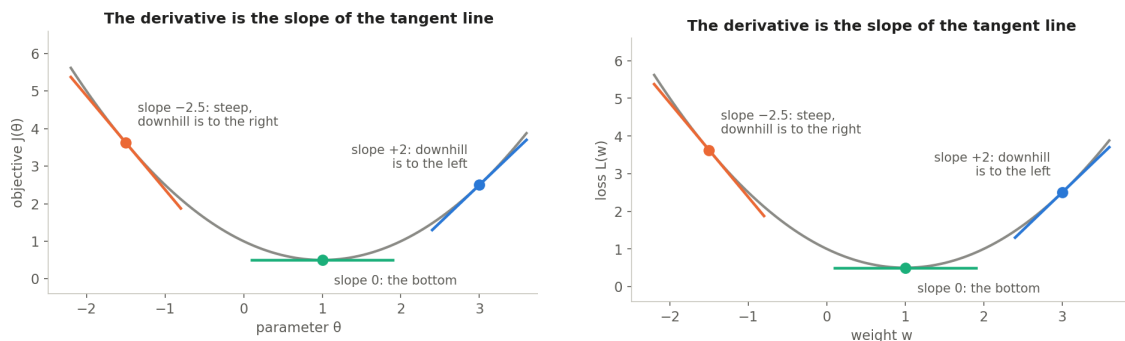
Suppose the model has a single parameter θ and the objective is

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

The **derivative** $dJ/d\theta$, also written $J'(\theta)$, is the slope of the objective curve at θ : how much J changes per unit change in θ . We measure it as “rise over run” with a small step h , and let h shrink to zero:

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

For our example, $J'(\theta) = \theta - 1$.



Left: the derivative is the slope of the tangent line. Right: as the step h shrinks, the rise-over-run slope settles on the tangent slope, here -2 at $\theta = -1$.

How to read a derivative:

- **Sign = direction.** A negative slope means J goes *down* as θ increases, so downhill is to the right. A positive slope means downhill is to the 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.

2.2 Many Parameters: Partial Derivatives and the Gradient

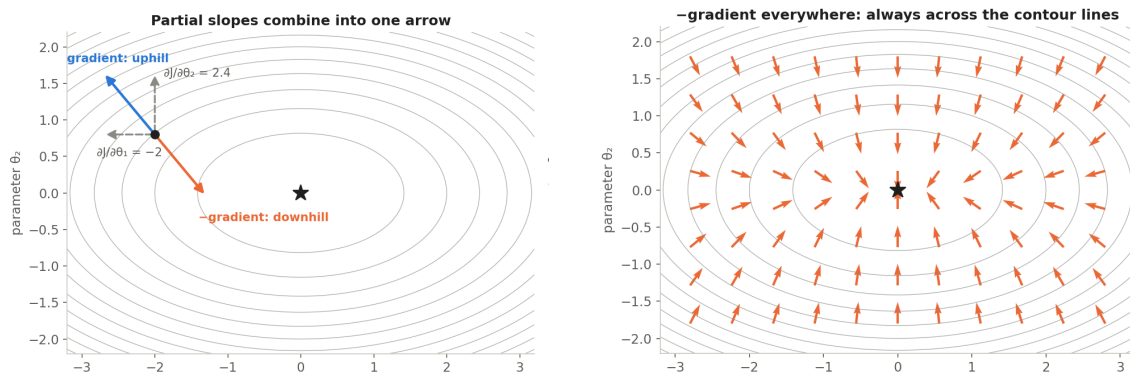
With M parameters, we calculate the slope for one parameter at a time, holding the others fixed. That is a **partial derivative**, written with a curly ∂ as $\partial J / \partial \theta_m$. The **gradient** stacks all M partial derivatives into a column vector:

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

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 grey ovals are **contour lines**: each joins points with the same value of J , and the star is the minimum. On the left, the two partial slopes at $[-2, 0.8]^\top$ (grey dashed arrows) combine into one gradient arrow. On the right, the arrows show $-\nabla J(\theta)$ at many points.

- The gradient points **uphill**, in the direction of steepest increase, 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 contours are ovals rather than circles because the slope grows three times faster along θ_2 than along θ_1 . That difference is the subject of the next section.

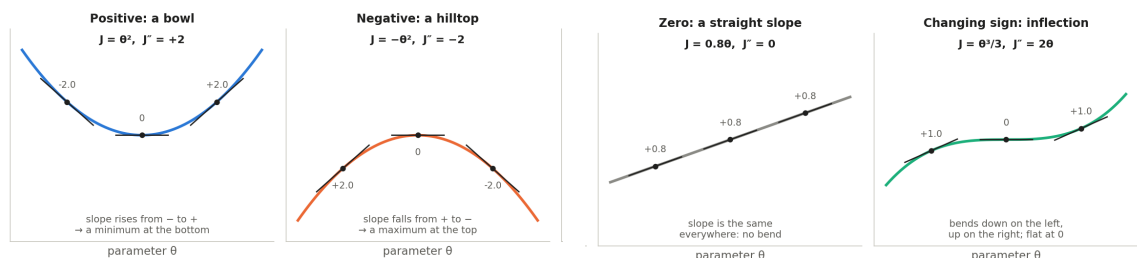
3 Curvature: How the Slope Changes

3.1 One Parameter: The Second Derivative

Slope tells us which way is downhill. **Curvature** tells us how quickly that slope changes as we move, and that decides how large a step we can safely take (Section 4.2).

The **second derivative** $J''(\theta)$ is the slope of the slope. We compute it by differentiating twice. For the bowl of Section 2.1, $J(\theta) = \frac{1}{2}(\theta - 1)^2 + \frac{1}{2}$ gives $J'(\theta) = \theta - 1$ and then $J''(\theta) = 1$.

- Its **sign** tells us the shape of the objective. Positive curvature is a bowl: the slope rises from negative to positive, so there is a minimum at the bottom. Negative curvature is a hilltop, with a maximum at the top. Zero curvature is a straight slope with no bend. Where the sign changes, the curve bends one way and then the other, which is an inflection.
- Its **size** tells us how sharply the objective bends.



A note on the word “curvature”: this word is defined differently in different contexts. In a calculus or geometry class, the curvature of a curve measures how fast its direction turns per unit

of distance along the curve, which for the graph of J works out to $J''/(1 + J'^2)^{3/2}$, a quantity that also depends on the slope. In optimization, and everywhere in these notes and in lecture, *curvature means the second derivative $J''(\theta)$ itself*, the rate at which the slope changes per unit of θ , including its sign. With many parameters it means the Hessian of Section 3.3.

3.2 Sharp and Wide Bowls

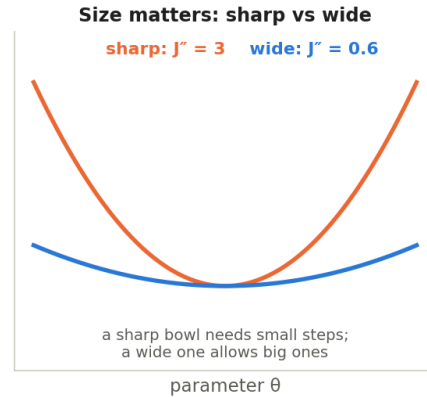
Two bowls can both have positive curvature but bend very differently.

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 up on the other side. A small J'' (here 0.6) is a **wide** bowl where big steps are safe.

If a bowl has constant curvature λ and its minimum is at θ^* , then $J(\theta) = \frac{1}{2}\lambda(\theta - \theta^*)^2 + \text{const}$, and its slope is

$$\text{slope} = \lambda \cdot d, \quad d = \theta - \theta^*,$$

where d is how far we are from the minimum. In a sharp direction, even a small distance produces a big slope.



We will see in Section 4.2 that the learning rate must stay below $2/J''$, so a sharp bowl forces small steps. **Most of the trouble in optimization comes from curvature varying widely across directions**, and that is the theme of the lecture. To talk about it, we need curvature with many parameters.

3.3 Many Parameters: The Hessian

With M parameters, there is a second derivative for every *pair* of parameters: differentiate with respect to one parameter and then with respect to the other. The **Hessian** $H = \nabla^2 J(\theta)$ collects them in an $M \times M$ matrix:

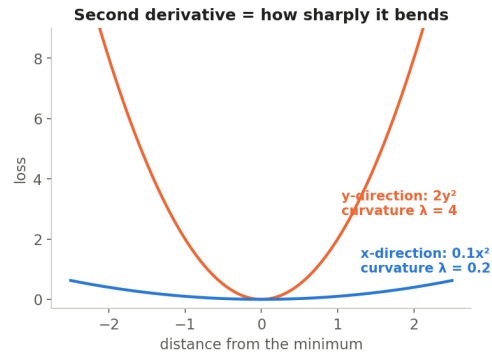
$$H_{jk} = \frac{\partial^2 J}{\partial \theta_j \partial \theta_k}, \quad H = \nabla^2 J(\theta) = \begin{bmatrix} \frac{\partial^2 J}{\partial \theta_1^2} & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_M} \\ \frac{\partial^2 J}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 J}{\partial \theta_2^2} & \cdots & \frac{\partial^2 J}{\partial \theta_2 \partial \theta_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial \theta_M \partial \theta_1} & \frac{\partial^2 J}{\partial \theta_M \partial \theta_2} & \cdots & \frac{\partial^2 J}{\partial \theta_M^2} \end{bmatrix}.$$

- The **diagonal** entries H_{mm} are the curvatures along each parameter.
- The **off-diagonal** entries say how the slope in one parameter changes when another parameter moves. The matrix is symmetric, $H_{jk} = H_{kj}$.

The ravine: This is the example we will use for the rest of these notes, and the one the lecture uses to compare optimizers:

$$J(\theta) = 0.1\theta_1^2 + 2\theta_2^2, \quad \nabla J(\theta) = [0.2\theta_1, 4\theta_2]^\top, \quad H = \begin{bmatrix} 0.2 & 0 \\ 0 & 4 \end{bmatrix}.$$

The off-diagonal entries are zero because $\partial J / \partial \theta_1 = 0.2 \theta_1$ does not involve θ_2 at all. The curvature along θ_1 is 0.2, a wide bowl, and along θ_2 it is 4, a sharp one. Cutting through the landscape along each axis, through the minimum, shows the two bowls side by side. In the figure, x is θ_1 and y is θ_2 .



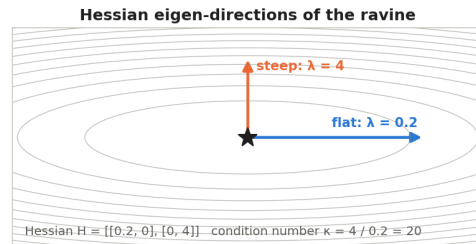
3.4 Eigenvectors, Eigenvalues, and the Condition Number

The diagonal of H only gives the curvatures along the parameter axes. The directions to look along are the **eigenvectors** of H : an eigenvector $\mathbf{v}$ is a direction along which the objective, viewed along that line, is a plain parabola, and its curvature in that direction is the **eigenvalue** λ . In symbols,

$$H\mathbf{v} = \lambda\mathbf{v},$$

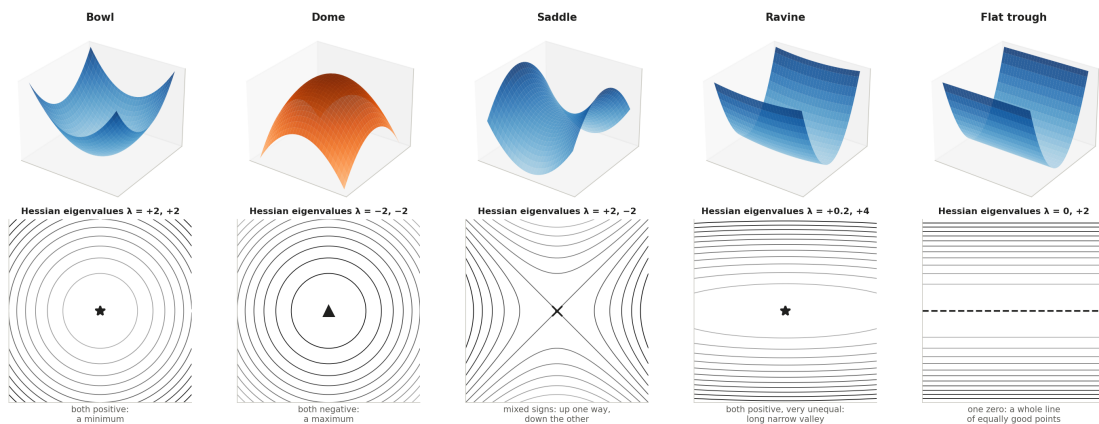
multiplying the direction $\mathbf{v}$ by H only stretches it by λ . (These are the same eigenvectors and eigenvalues that PCA uses, now of the Hessian instead of the covariance matrix.)

For our ravine, H is diagonal, so the eigen-directions are simply the θ_1 axis, with $\lambda = 0.2$ (flat), and the θ_2 axis, with $\lambda = 4$ (steep). If the oval contours were tilted, the eigen-directions would be the tilted long and short axes of the ovals, and neither would line up with a single parameter.



The signs of the eigenvalues decide the shape of the landscape, just as the sign of J'' did with one parameter:

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



Both eigenvalues positive is a bowl with a minimum; both negative is a dome with a maximum; mixed signs is a saddle, up one way and down the other; both positive but very unequal is our ravine; and one eigenvalue of zero is a flat trough with a whole line of equally good points.

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 what makes gradient descent slow (Section 4.4).

3.5 Up Close, Every Objective Looks Like a Quadratic

Near a reference point θ_0 , a smooth objective is well approximated by a quadratic built from its value, slope, and curvature at that point. This is the **second-order Taylor approximation**:

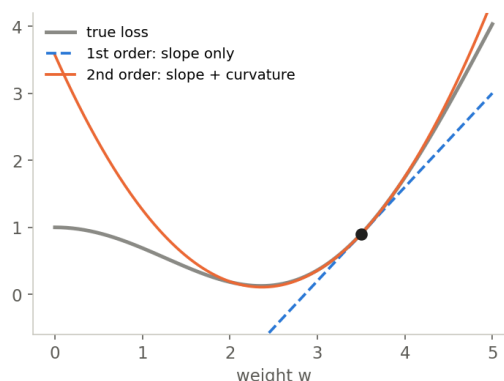
$$J(\theta) \approx J(\theta_0) + J'(\theta_0)(\theta - \theta_0) + \frac{1}{2} J''(\theta_0)(\theta - \theta_0)^2.$$

With M parameters and a small move $\theta - \theta_0$, the same idea reads

$$J(\theta) \approx J(\theta_0) + \nabla J(\theta_0)^\top (\theta - \theta_0) + \frac{1}{2} (\theta - \theta_0)^\top H (\theta - \theta_0).$$

The straight line (slope only) is good just next to the point; the parabola (slope plus curvature) hugs the true objective much longer.

For the ravine with $\theta_0 = \mathbf{0}$, the gradient term is zero and the last term is $\frac{1}{2}(0.2\theta_1^2 + 4\theta_2^2) = 0.1\theta_1^2 + 2\theta_2^2$: each entry of $\theta - \theta_0$ squared, times its curvature, times $\frac{1}{2}$. Here the approximation is exact, since a quadratic is its own second-order Taylor approximation.



The approximation is always a quadratic, but its shape depends on where we take it: the eigenvalues of H at θ_0 pick out one of the landscapes of Section 3.4, so up close an objective can look like a bowl, a dome, a saddle, or a ravine. At a minimum θ^* the gradient is zero, so near it the slope term drops out and what remains is $\frac{1}{2}(\theta - \theta^*)^\top H(\theta - \theta^*)$, and at a minimum none of the eigenvalues of H is negative. **Near a minimum, every smooth objective is approximately a quadratic bowl**, with the eigenvalues of H as its curvatures. So watching optimizers on simple quadratic bowls like the ravine tells us how they behave close to a real minimum, and that is why the rest of these notes, and the lecture, use them as the test case.

4 Gradient Descent

4.1 The Update

Gradient descent starts from some $\theta^{(0)}$ and repeatedly steps downhill:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla J(\theta^{(t)}), \quad \text{new weights} = \text{old weights} - \text{learning rate} \times \text{gradient}.$$

The **learning rate** η sets how far we go. Entry by entry, the update is

$$\theta_m^{(t+1)} = \theta_m^{(t)} - \eta \frac{\partial J}{\partial \theta_m}(\theta^{(t)}),$$

so each parameter moves against its own slope, by an amount proportional to how steep that slope is. Like a hiker in fog, the algorithm sees only the slope under its feet and takes a step downhill.

4.2 The Learning Rate and Curvature

Let $J(\theta) = \theta^2$, so $J'(\theta) = 2\theta$, and take $\eta = 0.3$ starting from $\theta^{(0)} = -4$:

Step t	$\theta^{(t)}$	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. The reason is that near a minimizer θ^* with curvature λ , the slope is $J'(\theta) = \lambda(\theta - \theta^*)$ (Section 3.2). Substituting that into the update and subtracting θ^* from both sides gives

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

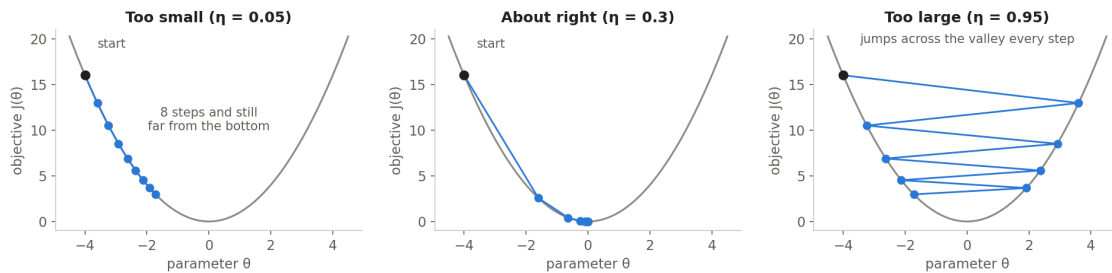
So the distance to the minimum is multiplied by the factor $1 - \eta\lambda$ at every step. For $J(\theta) = \theta^2$ we have $\lambda = 2$ and $\eta\lambda = 0.6$, so the factor is 0.4. How big the learning rate can be depends on this factor:

$\eta\lambda$	Factor $1 - \eta\lambda$	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

The safe range is

$$0 < \eta < \frac{2}{\lambda}.$$

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



With many parameters there is one such factor per eigen-direction, $1 - \eta\lambda_i$, and the steepest direction sets the limit for everyone: $\eta < 2/\lambda_{\max}$. This limit is the root of the problems in Section 4.4.

4.3 Stochastic Gradient Descent

The objective is an average over all N training examples, so its gradient is an average too:

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

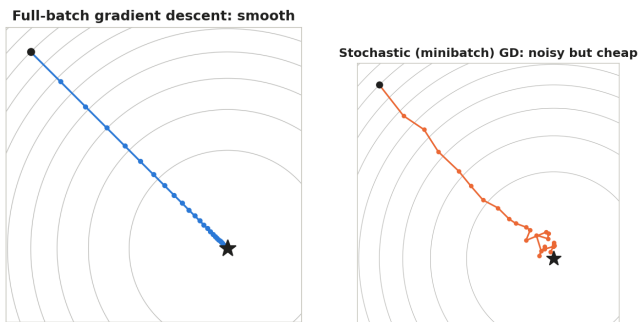
Computing this gradient requires every training example, which is expensive for a single update. **Stochastic gradient descent (SGD)** instead picks a random **minibatch** $\mathcal{B}^{(t)}$ of B examples and averages only those:

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

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

Because the minibatch is random, $\mathbf{g}^{(t)}$ is right on average: $\mathbb{E}[\mathbf{g}^{(t)}] = \nabla J(\boldsymbol{\theta}^{(t)})$. Each step has some noise, but the noise points in random directions and largely cancels over many steps. Near the minimum the true gradient becomes tiny while the noise does not, so SGD keeps jittering around the bottom. That is one reason learning rates are usually lowered later in training.

From here on, “the gradient $\mathbf{g}$ ” means this minibatch estimate.



4.4 Problems with SGD

Real loss landscapes are often shaped like long, narrow ravines: steep walls on the sides, a gentle slope along the floor. Our ravine from Section 3.3 is a small one:

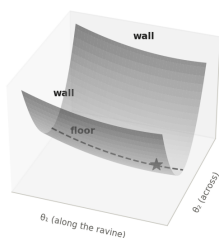
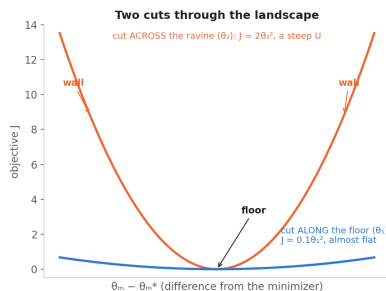
$$J(\boldsymbol{\theta}) = 0.1 \theta_1^2 + 2 \theta_2^2, \quad \nabla J(\boldsymbol{\theta}) = [0.2 \theta_1, 4 \theta_2]^\top, \quad \lambda_1 = 0.2, \quad \lambda_2 = 4, \quad \kappa = 20.$$

Since each entry of the gradient involves only its own parameter, the update acts on the two parameters separately, each with its own factor from Section 4.2:

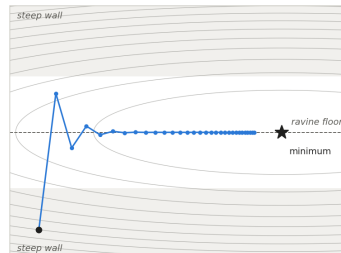
$$\theta_1^{(t+1)} = (1 - 0.2 \eta) \theta_1^{(t)}, \quad \theta_2^{(t+1)} = (1 - 4 \eta) \theta_2^{(t)}.$$

The ravine: a long valley floor between two steep walls

The landscape: a long trough with steep sides



Seen from above: SGD bounces wall to wall



1. **The direction problem.** 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. The steep direction θ_2 needs $\eta < 2/4 = 0.5$ to avoid diverging, but at any such η the flat direction θ_1 shrinks by at most 10% per step, since $1 - 0.2\eta > 0.9$. Some weights sit in steep directions and need tiny steps, others sit in flat directions and need big ones. Any single learning rate is too big for some and too small for others.

The optimizers in the lecture attack these two problems. **Momentum** addresses the direction problem by averaging recent gradients, so that the wall-to-wall bounces cancel and the pushes along the floor add up. **RMSProp** and **Adam** address the step-size problem by giving each weight its own step size, based on how large its gradients have typically been. Both are built from running averages of the gradient and of its square, the statistics of Section 5.

5 Helpful Statistical Tools

These are the quantities the lecture's optimizers keep track of. Throughout, $x^{(1)}, \dots, x^{(T)}$ is a sequence of T numbers; think of the gradient of one weight at T successive steps.

5.1 Mean, Variance, and Root Mean Square

1. **Mean**: the typical value, which for gradients means 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 .

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

Example: For the four values 3, -1, 2, 0,

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
average	$\bar{x} = 1$	MS = 3.5	0 (always)	$\sigma^2 = 2.5$

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

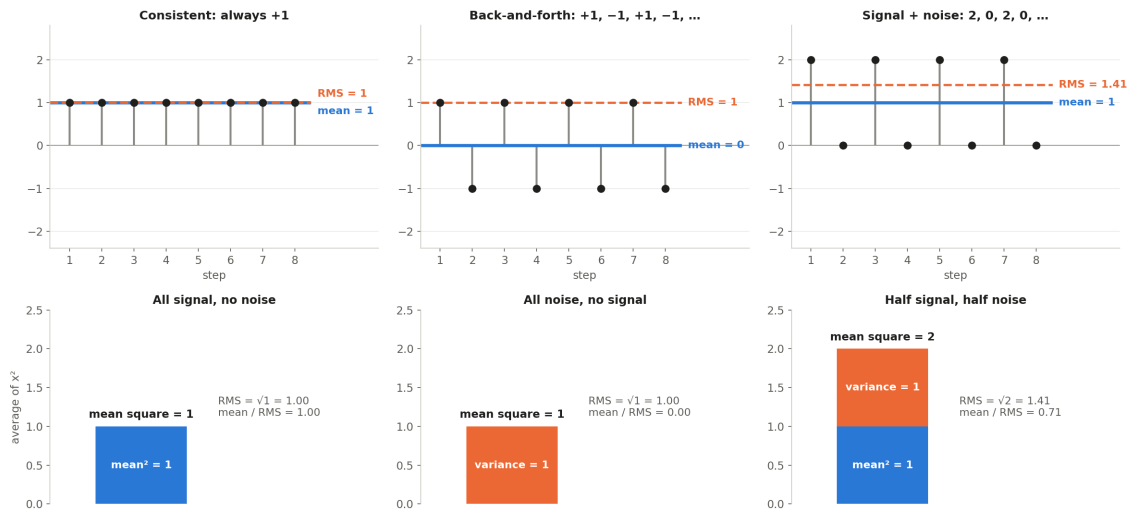
Mean square = mean² + variance: Expanding the square inside the variance,

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

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

In the example, $1^2 + 2.5 = 3.5$. In the language of random variables this is $\mathbb{E}[X^2] = (\mathbb{E}[X])^2 + \sigma^2$. The figure below shows three sequences: all signal, all noise, and half and half.

Mean tells the direction, RMS tells the size: mean square = mean² + variance



Three consequences:

- $\text{RMS} \geq |\bar{x}|$, with equality only when there is no noise.
- The ratio $\bar{x}/\text{RMS}$ lies between -1 and 1 . It is near ± 1 for a consistent signal and near 0 for pure noise. (In lecture, Adam's step turns out to be essentially η times this ratio.)
- $\sigma^2 = \text{MS} - \bar{x}^2$, so the noise can be computed from the average of x and the average of x^2 alone. This is how RMSProp and Adam work in practice: they keep running averages of $\mathbf{g}$ and of $\mathbf{g}^2$, never differences from the mean.

5.2 Exponential Moving Average

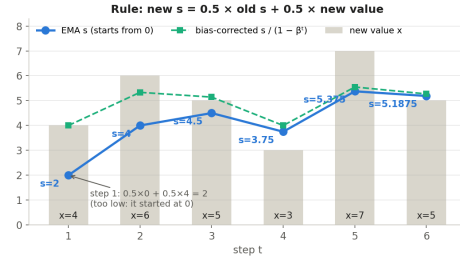
At every step, the optimizer receives a new, noisy gradient. We do *not* want to store the whole history of gradients. We want a **running estimate** of what the values have *typically* been lately, updated cheaply at each step. The **exponential moving average (EMA)** does this while storing a single number:

$$s_t = \beta s_{t-1} + (1 - \beta) x_t, \quad \text{starting from } s_0 = 0.$$

In words, new average = $\beta \times$ old average + $(1 - \beta) \times$ new value: keep a fraction β of the old average and mix in a fraction $1 - \beta$ of the new value. The number β , between 0 and 1 , sets how much of the past to keep. (Following the lecture slides, the EMA writes the step as a subscript: x_t is the value that arrives at step t and s_t is the average after it.)

Example with $\beta = 0.5$ and the values $4, 6, 5, 3, 7, 5$:

Step t	New x_t	$0.5 s_{t-1} + 0.5 x_t$	New s_t
1	4	$0.5 \times 0 + 0.5 \times 4$	2
2	6	$0.5 \times 2 + 0.5 \times 6$	4
3	5	$0.5 \times 4 + 0.5 \times 5$	4.5
4	3	$0.5 \times 4.5 + 0.5 \times 3$	3.75
5	7	$0.5 \times 3.75 + 0.5 \times 7$	5.375
6	5	$0.5 \times 5.375 + 0.5 \times 5$	5.1875



What is inside s_t : Unrolling the rule shows that the EMA is a weighted sum of all of the past values. With $\beta = 0.5$,

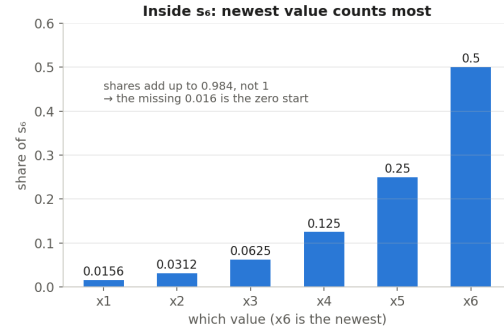
$$\begin{aligned}
 s_3 &= 0.5 x_3 + 0.5 s_2 \\
 &= 0.5 x_3 + 0.5 (0.5 x_2 + 0.5 s_1) \\
 &= 0.5 x_3 + 0.25 x_2 + 0.125 x_1,
 \end{aligned}$$

using $s_1 = 0.5 x_1$ because $s_0 = 0$. In general,

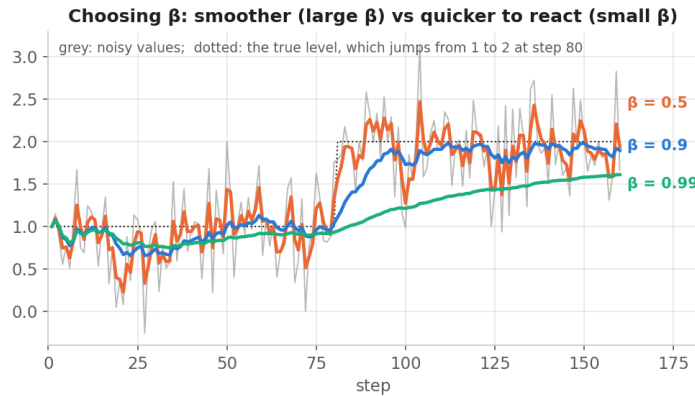
$$s_t = (1 - \beta) (x_t + \beta x_{t-1} + \beta^2 x_{t-2} + \dots + \beta^{t-1} x_1).$$

Each older value counts β times less than the one after it, so the newest value counts most. Because the shares fall off geometrically, only the last $1/(1 - \beta)$ or so values really matter: the EMA effectively remembers about

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



Choosing β : β sets the memory. A small β is quick to react: the average forgets the old level fast, so it follows a jump in the values within a few steps. A large β is slow to react: the old values keep most of the weight, so the average lags behind a change for a long time. Larger β means smoother but slower.



Bias from the zero start: Notice that the shares in s_3 do not add up to 1: $0.5 + 0.25 + 0.125 = 0.875$. In general, summing the geometric series,

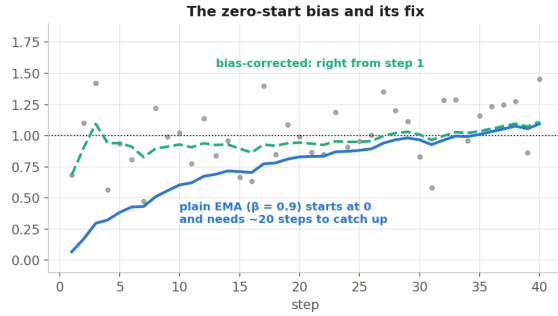
$$1 + \beta + \beta^2 + \dots + \beta^{t-1} = \frac{1 - \beta^t}{1 - \beta} \quad \Rightarrow \quad \text{shares add up to } (1 - \beta) \cdot \frac{1 - \beta^t}{1 - \beta} = 1 - \beta^t.$$

The missing share, β^t , belongs to the zero we started from. So an EMA that starts at 0 is **too small at first**: if every value equals x , the average after t steps is exactly $(1 - \beta^t)x$. The fix is to divide that factor out:

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

In the example, $\hat{s}_1 = 2/(1 - 0.5) = 4$, which is x_1 itself, so the corrected average is right from step 1, and $\hat{s}_3 = 4.5/0.875 \approx 5.14$. After many steps β^t is almost 0 and the correction stops mattering.

Fixing the start matters a lot for Adam, which uses β close to 1. With $\beta = 0.9$ the plain EMA needs about 20 steps to catch up; with $\beta = 0.999$ it would need thousands. The figure shows a noisy sequence whose true level is 1: the plain EMA starts near 0 and creeps up, while the bias-corrected version is right from step 1.



6 Summary

Term	What it is
Objective $J(\theta)$	One number saying how badly parameters θ do, usually the average loss over the N training examples. Training minimizes it.
Gradient $\nabla J(\theta)$	The column vector of partial derivatives. Points uphill, perpendicular to the contour lines; its length is the steepness; zero at a minimum.
Curvature $J''(\theta)$	The slope of the slope. Sign gives the shape (bowl, hilltop, straight, inflection); size says how sharply it bends. In these notes “curvature” always means this second derivative.
Hessian H	The $M \times M$ matrix of second derivatives. Diagonal entries are curvatures along each parameter; off-diagonal entries say how one slope changes as another parameter moves.
Eigenvalues of H	The curvatures along the eigen-directions, where the objective is a plain parabola. Condition number $\kappa = \lambda_{\max}/\lambda_{\min}$; large κ is a long, narrow ravine.
Taylor approximation	$J(\theta) \approx J(\theta_0) + \nabla J(\theta_0)^\top (\theta - \theta_0) + \frac{1}{2}(\theta - \theta_0)^\top H(\theta - \theta_0)$: up close, a smooth objective is approximately a quadratic, and near a minimum that quadratic is a bowl.
Gradient descent	$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla J(\theta^{(t)})$. In a bowl with curvature λ the distance to the minimum is multiplied by $1 - \eta\lambda$ each step, so $0 < \eta < 2/\lambda$; the steepest direction sets $\eta < 2/\lambda_{\max}$.
SGD	The same update with the gradient of a random minibatch of B examples, $\mathbf{g}^{(t)}$: right on average, noisy, far cheaper.
Problems with SGD	In a ravine, SGD zig-zags between the walls (direction problem) and one learning rate is too big for the steep directions and too small for the flat ones (step-size problem).
Mean, MS, variance, RMS	$\bar{x}$ is the typical value, MS the average square, σ^2 the noise, $\text{RMS} = \sqrt{\text{MS}}$ the typical size; $\text{MS} = \bar{x}^2 + \sigma^2$.
EMA	$s_t = \beta s_{t-1} + (1 - \beta)x_t$ from $s_0 = 0$: a running average that remembers about $1/(1 - \beta)$ values. Starts too small by the factor $1 - \beta^t$; bias correction divides it out, $\hat{s}_t = s_t/(1 - \beta^t)$.

In lecture, we will put these pieces together: momentum keeps an EMA-like running sum of the gradient to fix the direction problem, RMSProp keeps an EMA of the squared gradient and divides by its root mean square to fix the step-size problem, and Adam does both, with bias correction.