

1 Bias & Variance

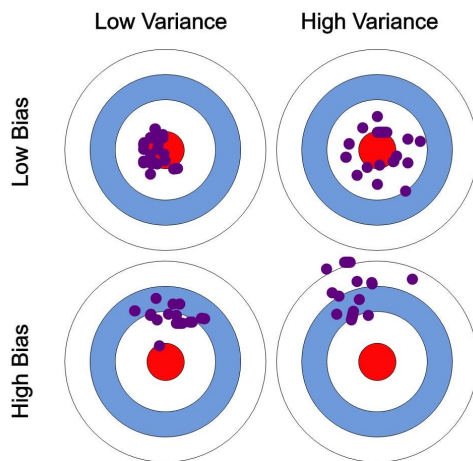
We will be using two terms very frequently from now on, which you may have heard of previously.

- **Bias** is a *systematic* error: how far off an estimate is *on average* from the true quantity it is trying to estimate. If we could repeat the whole procedure many times (new data, new samples) and average the results, the bias is the gap that would remain, $\text{Bias}(\hat{x}) = \mathbb{E}[\hat{x}] - x_{\text{true}}$. An estimator with zero bias is called *unbiased*.

Bias shows up in both deep-learning optimization and REINFORCE (which we will cover in this recitation). For example, an exponential moving average that starts at $\mathbf{0}$ is biased toward $\mathbf{0}$ in its first steps, which is what Adam's bias correction fixes. In REINFORCE, we ask whether $\mathbb{E}[g] = \nabla_{\theta} J(\theta)$, i.e. whether our gradient estimate points the right way on average.

- **Variance** is the *random* error: how much an estimate would change from one repetition of the procedure to the next, $\text{Var}(\hat{x}) = \mathbb{E}[(\hat{x} - \mathbb{E}[\hat{x}])^2]$. A low-variance estimator gives nearly the same answer every time, while a high-variance one is noisy, even if it is correct on average.

Variance also shows up in REINFORCE, where each gradient estimate is computed from a few randomly sampled trajectories.



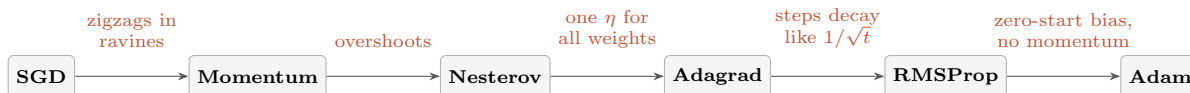
Often in machine learning, we have to deal with what is called the *bias/variance tradeoff*. In general, this refers to decisions we have to make about our model, algorithm (like in reinforcement learning), or training (like in DL optimization) that try to reduce bias or variance, but have to sacrifice the other.

For example, in DL optimization, a minibatch gradient is an unbiased but noisy estimate of the full gradient. Averaging past gradients with an EMA (as in momentum, RMSProp, and Adam) smooths out the noise and lowers the variance, but it mixes in stale gradients from earlier parameter values, which adds bias.

We will see the same tradeoff when we talk about policy gradient methods: REINFORCE is unbiased but has high variance, and later improvements (like adding a baseline and bootstrapping in actor-critic) lowers the variance at the price of some bias.

2 Definitions: Optimization for Deep Learning

Training a network means minimizing a loss $J(\theta)$ over millions of parameters $\theta \in \mathbb{R}^M$. Every optimizer below computes the gradient $\mathbf{g}^{(t)} = \nabla J(\theta^{(t)})$ and then decides how far to move each parameter; the methods differ only in what they remember about past gradients. Each one fixes a weakness of the one before it:



We start at $\theta^{(1)}$, every running quantity $(\mathbf{m}, \mathbf{s}, \mathbf{v})$ starts at $\mathbf{0}$, and squares, square roots, products $\odot$ and divisions of vectors act entry by entry.

2.1 Building Block: Exponential Moving Average (EMA)

1. **EMA:** A running average that weights recent values more. Keep a fraction β of the old average and mix in a fraction $1 - \beta$ of the newest value:

$$s_t = \beta s_{t-1} + (1 - \beta) x_t = (1 - \beta) (x_t + \beta x_{t-1} + \beta^2 x_{t-2} + \cdots + \beta^{t-1} x_1), \quad s_0 = 0,$$

so the value from j steps ago, x_{t-j} , gets weight $(1 - \beta)\beta^j$: each step back multiplies its weight by β .

2. **β Sets the Memory:** The weights of the N most recent values add up to

$$(1 - \beta) (1 + \beta + \cdots + \beta^{N-1}) = 1 - \beta^N \quad (\text{multiply out: the middle terms cancel}).$$

3. **Bias at the Start:** A proper average has weights that sum to 1, so that a constant sequence $x_k = c$ averages to c . At step t only t values exist, so by the formula above with $N = t$ their weights sum to just $1 - \beta^t < 1$, and a constant sequence gives $s_t = (1 - \beta^t)c$: too small. The missing weight β^t went to the starting value $s_0 = 0$ (e.g. with $\beta = 0.9$ and $x_1 = 5$, $s_1 = 0.5$). Dividing by the weight sum removes this bias: $\hat{s}_t = s_t / (1 - \beta^t)$, so $\hat{s}_1 = x_1$.

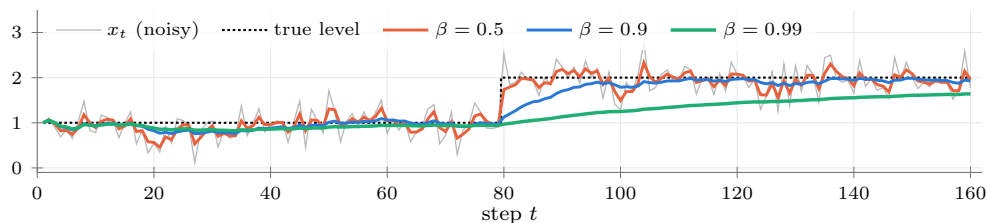


Figure 1: Bias-corrected EMAs of a noisy signal whose true level jumps from 1 to 2 at step 80. $\beta = 0.5$ follows the jump at once but keeps the noise; $\beta = 0.99$ is smooth but still well below 2.

2.2 From SGD to Adam

1. **Stochastic Gradient Descent (SGD):** Move every parameter against its own slope, with a single learning rate η :

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

Issue: loss surfaces have ravines, steep in some directions and flat in others. η must be small enough for the steepest direction or the iterates diverge, so the path zigzags across the ravine and crawls along its floor.

2. **Momentum** (fixes the zigzag): Keep a velocity, a decaying sum of past gradients, and move along it:

$$\mathbf{m}^{(t)} = \gamma \mathbf{m}^{(t-1)} + \mathbf{g}^{(t)} = \mathbf{g}^{(t)} + \gamma \mathbf{g}^{(t-1)} + \dots + \gamma^{t-1} \mathbf{g}^{(1)}, \quad \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \mathbf{m}^{(t)}.$$

Gradient components that flip sign every step (across the ravine) cancel, while those that keep their sign (along the floor) add up to $\mathbf{g}/(1-\gamma)$: a $10\times$ longer step for the usual $\gamma = 0.9$. Setting $\gamma = 0$ gives back SGD.

Issue: the correction uses the slope where the step started, so a fast-moving ball overshoots the minimum.

3. **Nesterov Momentum** (fixes overshooting): First coast on the old velocity to a look-ahead point, then measure the slope there:

$$\tilde{\boldsymbol{\theta}}^{(t)} = \boldsymbol{\theta}^{(t)} - \eta \gamma \mathbf{m}^{(t-1)}, \quad \mathbf{m}^{(t)} = \gamma \mathbf{m}^{(t-1)} + \nabla J(\tilde{\boldsymbol{\theta}}^{(t)}), \quad \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \mathbf{m}^{(t)}.$$

Substituting, $\boldsymbol{\theta}^{(t+1)} = [\boldsymbol{\theta}^{(t)} - \eta \gamma \mathbf{m}^{(t-1)}] - \eta \nabla J(\tilde{\boldsymbol{\theta}}^{(t)})$: coast to $\tilde{\boldsymbol{\theta}}^{(t)}$, then correct with the slope measured there. When the velocity is about to carry the parameters past the minimum, the slope at the landing point already points back, so Nesterov brakes one step earlier. It still costs one gradient per step.

Issue: SGD, momentum and Nesterov use the same η for every parameter, although steep directions need small steps and flat directions need big ones.

4. **Adagrad** (fixes one η for all weights): Divide each parameter's gradient by the square root of its running sum of squared gradients:

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

Writing $s_i^{(t)} = \sum_{k=1}^t (g_i^{(k)})^2$ for the i -th entry of $\mathbf{s}^{(t)}$, parameter θ_i ($i = 1, \dots, M$) gets its own **effective step size** $\eta/(\sqrt{s_i^{(t)}} + \epsilon)$: large past gradients (a steep direction) give small steps, small ones give big steps. The constant ϵ (e.g. 10^{-10}) only prevents division by zero. Because each coordinate is rescaled by a different amount, this changes the step's direction, not just its length: at $t = 1$ every weight moves by exactly η , so the first step is $-\eta \text{sign}(\mathbf{g}^{(1)})$, whereas SGD, momentum and Nesterov all start along $-\mathbf{g}^{(1)}$ (for the last two, $\mathbf{m}^{(1)} = \mathbf{g}^{(1)}$).

Issue: $\mathbf{s}$ only grows. If θ_i 's gradients have a steady size σ , then $s_i^{(t)} \approx t\sigma^2$ and its step is $\eta\sigma/\sqrt{t\sigma^2} = \eta/\sqrt{t}$, so learning stalls.

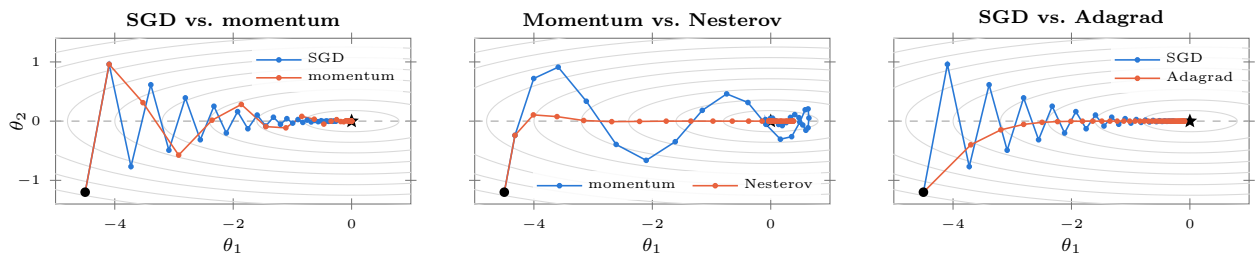


Figure 2: All panels: the ravine $J(\boldsymbol{\theta}) = \frac{1}{2}(\theta_1^2 + 20\theta_2^2)$, same start (●) and minimum (★). Left: momentum ($\gamma = 0.5$) cancels SGD's zigzag (both $\eta = 0.09$). Middle: Nesterov damps momentum's bounces and overshoot ($\eta = 0.04$, $\gamma = 0.8$). Right: Adagrad ($\eta = 0.8$) rescales each coordinate separately, so its first step is diagonal instead of along $-\mathbf{g}$.

5. **RMSProp** (fixes decaying steps): Replace Adagrad's running sum with an EMA, so the step reflects how big the gradients have been recently:

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

$\sqrt{v}$ is the **root mean square (RMS)** of recent gradients: their typical size, even when their sign keeps flipping. With gradients of steady size σ , $v \approx \sigma^2$ and the step is $\eta\sigma/\sqrt{\sigma^2} = \eta$ at every step instead of $\eta/\sqrt{t}$. Old gradients fade within a few multiples of $1/(1-\rho)$ steps (typically $\rho = 0.9$).

Issue: $\mathbf{v}$ starts at $\mathbf{0}$, so $v^{(1)} = (1-\rho)(g^{(1)})^2$ is too small and the first step is $\eta/\sqrt{1-\rho} \approx 3.2\eta$. RMSProp also has no momentum.

6. **Adam** (momentum + RMSProp + bias correction): Keep an EMA of the gradients for a steady direction (like momentum) and an EMA of the squared gradients for a step size per weight (like RMSProp), and bias-correct both:

$$\begin{aligned}\mathbf{m}^{(t)} &= \beta_1 \mathbf{m}^{(t-1)} + (1-\beta_1) \mathbf{g}^{(t)}, & \hat{\mathbf{m}}^{(t)} &= \mathbf{m}^{(t)} / (1-\beta_1^t), \\ \mathbf{v}^{(t)} &= \beta_2 \mathbf{v}^{(t-1)} + (1-\beta_2) \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}, & \hat{\mathbf{v}}^{(t)} &= \mathbf{v}^{(t)} / (1-\beta_2^t),\end{aligned}$$

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

defaults: $\eta = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$,

where β^t is a power, not an iteration index. Why the correction: both EMAs start at $\mathbf{0}$, so for a constant gradient g they give $m^{(t)} = (1-\beta_1^t)g$ and $v^{(t)} = (1-\beta_2^t)g^2$. The factor $1-\beta_1^t$ approaches 1 on the scale of $1/(1-\beta_1) = 10$ steps, but $1-\beta_2^t$ only on the scale of $1/(1-\beta_2) = 1000$ steps. The uncorrected step $\eta m^{(t)}/\sqrt{v^{(t)}}$ therefore divides by a number that is too small, and the step is too large: 3.2η at $t = 1$, up to 6.6η at $t = 12$. After correction, $\hat{m}^{(t)} = g$ and $\hat{v}^{(t)} = g^2$, so the step is exactly η .

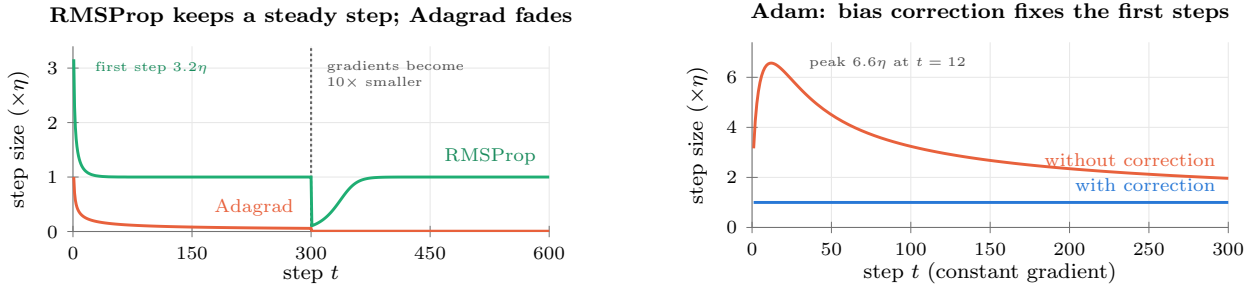


Figure 3: Step size in units of η . Left: the gradient has size 1 for 300 steps, then becomes $10\times$ smaller ($\rho = 0.9$). Adagrad's step fades like $1/\sqrt{t}$; RMSProp's settles at η and recovers within about 60 steps, after a first step of 3.2η . Right: Adam with a constant gradient. Without bias correction the early steps are up to $6.6\times$ too large.

3 DL Optimization: EMA, Gradient Descent, and Momentum

3.1 EMA by Hand

A sequence starts $x_1 = 4$, $x_2 = 2$, $x_3 = 6$. Use an EMA with $\beta = 0.5$ and $s_0 = 0$.

- (a) Compute s_1, s_2, s_3 .

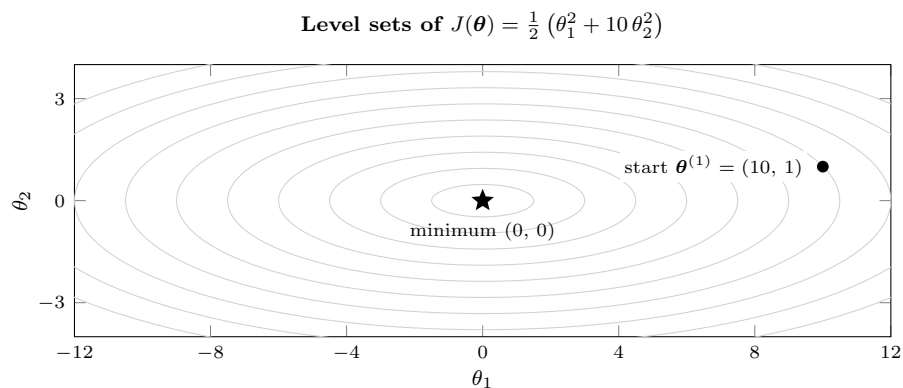
$$s_1 = 0.5 \cdot 0 + 0.5 \cdot 4 = 2, \quad s_2 = 0.5 \cdot 2 + 0.5 \cdot 2 = 2, \quad s_3 = 0.5 \cdot 2 + 0.5 \cdot 6 = 4.$$

- (b) Compute the bias-corrected values $\hat{s}_1, \hat{s}_2, \hat{s}_3$. Check that $\hat{s}_3$ is a weighted average of x_1, x_2, x_3 whose weights sum to 1.

The weight sums $1 - 0.5^t$ are 0.5, 0.75, 0.875, so $\hat{s}_1 = 2/0.5 = 4$, $\hat{s}_2 = 2/0.75 = 8/3 \approx 2.67$, $\hat{s}_3 = 4/0.875 = 32/7 \approx 4.57$. Note that $\hat{s}_1 = x_1$: the first corrected value is simply the first observation. Unrolled, $s_3 = 0.5x_3 + 0.25x_2 + 0.125x_1$. Dividing by 0.875 gives the weights $\frac{4}{7}, \frac{2}{7}, \frac{1}{7}$, which sum to 1, and $\frac{4}{7} \cdot 6 + \frac{2}{7} \cdot 2 + \frac{1}{7} \cdot 4 = \frac{32}{7}$.

3.2 Gradient Descent in a Ravine

Consider the following loss function, $J(\theta) = \frac{1}{2} (\theta_1^2 + 10\theta_2^2)$, starting point in the weight space $\theta^{(1)} = (10, 1)$.



- (a) Compute ∇J and write the GD update for each coordinate separately.

$$\nabla J = (\theta_1, 10\theta_2), \text{ so } \theta_1^{(t+1)} = (1 - \eta) \theta_1^{(t)} \text{ and } \theta_2^{(t+1)} = (1 - 10\eta) \theta_2^{(t)}.$$

- (b) Take one GD step with $\eta = 0.15$. What happens to θ_2 ? (Hint: reason on the sign of the weight)

$\theta^{(2)} = (0.85 \cdot 10, (1 - 1.5) \cdot 1) = (8.5, -0.5)$. The factor $1 - 10\eta = -0.5$ is negative, so θ_2 jumps past 0 to the other side: it flips sign at every step (bouncing between the walls), even though its size still shrinks.

- (c) For which learning rates does GD converge from every starting point?

We need $|1 - \eta| < 1$ and $|1 - 10\eta| < 1$, i.e. $0 < \eta < 2$ and $0 < \eta < 0.2$. So GD converges for $0 < \eta < 0.2$.

3.3 Momentum vs. Nesterov by Hand

Consider the 1-D objective $J(\theta) = \theta^2$, with $\theta^{(1)} = 1$, $m^{(0)} = 0$, $\eta = 0.1$ and $\gamma = 0.9$.

- (a) Run two steps of momentum: compute $g^{(t)}$, $m^{(t)}$ and $\theta^{(t+1)}$ for $t = 1, 2$.

The gradient is $\nabla J(\theta) = 2\theta$. Each step applies the update rules in order, with $\eta = 0.1$ and $\gamma = 0.9$.

Momentum: $m^{(t)} = \gamma m^{(t-1)} + 2\theta^{(t)}$, $\theta^{(t+1)} = \theta^{(t)} - \eta m^{(t)}$.

$$\begin{aligned} t = 1 : \quad & g^{(1)} = 2\theta^{(1)} = 2 \cdot 1 = 2 \\ & m^{(1)} = \gamma m^{(0)} + g^{(1)} = 0.9 \cdot 0 + 2 = 2 \\ & \theta^{(2)} = \theta^{(1)} - \eta m^{(1)} = 1 - 0.1 \cdot 2 = 0.8 \\ \\ t = 2 : \quad & g^{(2)} = 2\theta^{(2)} = 2 \cdot 0.8 = 1.6 \\ & m^{(2)} = \gamma m^{(1)} + g^{(2)} = 0.9 \cdot 2 + 1.6 = 3.4 \\ & \theta^{(3)} = \theta^{(2)} - \eta m^{(2)} = 0.8 - 0.1 \cdot 3.4 = 0.46 \end{aligned}$$

- (b) Run two steps of Nesterov: compute $\tilde{\theta}^{(t)}$, $\nabla J(\tilde{\theta}^{(t)})$, $m^{(t)}$ and $\theta^{(t+1)}$ for $t = 1, 2$.

Nesterov: first look ahead, $\tilde{\theta}^{(t)} = \theta^{(t)} - \eta \gamma m^{(t-1)}$, then use the slope there.

$$t = 1 : \quad \tilde{\theta}^{(1)} = \theta^{(1)} - \eta \gamma m^{(0)} = 1 - 0.1 \cdot 0.9 \cdot 0 = 1$$

$$\nabla J(\tilde{\theta}^{(1)}) = 2\tilde{\theta}^{(1)} = 2 \cdot 1 = 2$$

$$m^{(1)} = \gamma m^{(0)} + \nabla J(\tilde{\theta}^{(1)}) = 0.9 \cdot 0 + 2 = 2$$

$$\theta^{(2)} = \theta^{(1)} - \eta m^{(1)} = 1 - 0.1 \cdot 2 = 0.8$$

$$t = 2 : \quad \tilde{\theta}^{(2)} = \theta^{(2)} - \eta \gamma m^{(1)} = 0.8 - 0.1 \cdot 0.9 \cdot 2 = 0.62$$

$$\nabla J(\tilde{\theta}^{(2)}) = 2\tilde{\theta}^{(2)} = 2 \cdot 0.62 = 1.24$$

$$m^{(2)} = \gamma m^{(1)} + \nabla J(\tilde{\theta}^{(2)}) = 0.9 \cdot 2 + 1.24 = 3.04$$

$$\theta^{(3)} = \theta^{(2)} - \eta m^{(2)} = 0.8 - 0.1 \cdot 3.04 = 0.496$$

At $t = 1$ the two methods agree, because $m^{(0)} = 0$ makes the look-ahead point $\tilde{\theta}^{(1)}$ equal to $\theta^{(1)}$.

- (c) Compare $m^{(2)}$ for the two methods. Why is Nesterov's velocity smaller, and why is that useful as θ approaches the minimum at 0?

$m^{(2)} = 3.04$ for Nesterov and 3.4 for momentum. Nesterov measured the slope at the look-ahead point 0.62, which is closer to the minimum and therefore gentler (1.24 instead of 1.6), so it adds less speed. Momentum moves further in step 2 ($\theta^{(3)} = 0.46$ vs. 0.496), but it builds up more velocity than it needs. Once θ is about to pass 0, the look-ahead point is already on the far side, where the slope points back, so Nesterov starts braking one step earlier.

4 DL Optimization: RMSProp and Adam

RMSProp and Adam start from Adagrad's per-weight step size and fix its weaknesses one at a time. Throughout, ignore ϵ and take $\rho = 0.9$.

4.1 RMSProp

Recall the two updates (all operations act entry by entry):

$$\begin{aligned} \text{Adagrad: } \mathbf{s}^{(t)} &= \mathbf{s}^{(t-1)} + \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}, & \boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - \eta \frac{\mathbf{g}^{(t)}}{\sqrt{\mathbf{s}^{(t)}}}, \\ \text{RMSProp: } \mathbf{v}^{(t)} &= \rho \mathbf{v}^{(t-1)} + (1 - \rho) \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}, & \boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - \eta \frac{\mathbf{g}^{(t)}}{\sqrt{\mathbf{v}^{(t)}}}. \end{aligned}$$

1. A weight receives the same gradient $g \neq 0$ at every step. Find each method's step size $|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}|$ as a function of t . What happens as t grows, and why?

Adagrad: $s^{(t)} = t g^2$, so the step is $\eta |g| / \sqrt{t g^2} = \eta / \sqrt{t} \rightarrow 0$. The sum never forgets, so learning stalls.

RMSProp: unrolling the EMA, $v^{(t)} = (1 - \rho) g^2 (1 + \rho + \dots + \rho^{t-1}) = (1 - \rho^t) g^2$, so the step is $\eta / \sqrt{1 - \rho^t} \rightarrow \eta$. The EMA forgets old gradients, so $\sqrt{v}$ tracks the recent gradient size and the step stays at η .

2. RMSProp starts from $\mathbf{v}^{(0)} = \mathbf{0}$. Compute the size of its first step ($t = 1$) in terms of η . Is it larger or smaller than the long-run step from Question 1?

$v^{(1)} = (1 - \rho)(g^{(1)})^2 = 0.1 (g^{(1)})^2$, so the step is

$$\eta \frac{|g^{(1)}|}{\sqrt{0.1 |g^{(1)}|}} = \frac{\eta}{\sqrt{0.1}} \approx 3.2 \eta,$$

about $3\times$ larger than η (the $t = 1$ case of $\eta / \sqrt{1 - \rho^t}$). Starting from 0, $v^{(1)}$ underestimates $(g^{(1)})^2$ by the factor $1 - \rho$, and it sits in the denominator.

4.2 Adam Optimizer

Adam (or its variant AdamW) is the default optimizer in much of modern deep learning. It combines an EMA of the gradients (like momentum) with RMSProp's EMA of squared gradients, and bias-corrects both:

$$\begin{aligned} \mathbf{m}^{(t)} &= \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \mathbf{g}^{(t)}, & \hat{\mathbf{m}}^{(t)} &= \frac{\mathbf{m}^{(t)}}{1 - \beta_1^t}, \\ \mathbf{v}^{(t)} &= \beta_2 \mathbf{v}^{(t-1)} + (1 - \beta_2) \mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}, & \hat{\mathbf{v}}^{(t)} &= \frac{\mathbf{v}^{(t)}}{1 - \beta_2^t}, \\ \boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - \eta \frac{\hat{\mathbf{m}}^{(t)}}{\sqrt{\hat{\mathbf{v}}^{(t)}}}, & \mathbf{m}^{(0)} = \mathbf{v}^{(0)} &= \mathbf{0}, \quad \beta_1 = 0.9, \quad \beta_2 = 0.999. \end{aligned}$$

3. Compute the size of Adam's first step ($t = 1$) in terms of η : (a) without bias correction (use $\mathbf{m}$ and $\mathbf{v}$ in place of $\hat{\mathbf{m}}$ and $\hat{\mathbf{v}}$), and (b) with it.

$$m^{(1)} = (1 - \beta_1) g^{(1)} = 0.1 g^{(1)} \text{ and } v^{(1)} = (1 - \beta_2) (g^{(1)})^2 = 0.001 (g^{(1)})^2.$$

$$(a) \eta \frac{0.1 |g^{(1)}|}{\sqrt{0.001 |g^{(1)}|}} = \frac{0.1}{\sqrt{0.001}} \eta \approx 3.2 \eta.$$

$$(b) \hat{m}^{(1)} = 0.1 g^{(1)} / 0.1 = g^{(1)} \text{ and } \hat{v}^{(1)} = 0.001 (g^{(1)})^2 / 0.001 = (g^{(1)})^2, \text{ so the step is } \eta |g^{(1)}| / |g^{(1)}| = \eta.$$

5 Reinforcement Learning Review

Recall the general goal in policy gradient reinforcement learning: we want to learn a policy $\pi_\theta(a \mid s)$ that maximizes the expected cumulative discounted reward:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T \gamma^t r_t \right] = \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)]$$

where:

- $\tau = [(s_0, a_0, r_0), \dots, (s_T, a_T, r_T)]$ is a trajectory or episode,
- $R(\tau)$ is the total reward of the trajectory,
- $\gamma \in [0, 1]$ is the discount factor, and
- the expectation is taken over T trajectories collected by deploying the policy π_θ in the environment.

$\nabla_\theta J(\theta)$ is called the **policy gradient**, and a policy gradient method learns the parameters θ of the policy π_θ by performing gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

where η is the learning rate.

5.1 Bias and Variance of the Gradient Estimator

A gradient estimator g is an estimate of the true policy gradient $\nabla_\theta J(\theta)$. We focus on two properties of g :

- **Bias:** $\text{Bias}(g) = \mathbb{E}[g] - \nabla_\theta J(\theta)$. The estimator is *unbiased* if $\text{Bias}(g) = 0$, or $\mathbb{E}[g] = \nabla_\theta J(\theta)$ (on average it points in the true gradient direction).
- **Variance:** $\text{Var}(g) = \mathbb{E}[\|g - \mathbb{E}[g]\|_2^2]$, how much g fluctuates from one batch of sampled trajectories to the next.

5.2 Policy Gradient Method Comparisons

Each algorithm below keeps the same form and only changes the estimate of the future reward, and differs in the tradeoff between bias and variance:

1. All of the algorithms compute a gradient of the form

$$g_\theta = \sum_{t=0}^T \tilde{G}_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

where $\tilde{G}_t$ is an estimate of the expected discounted future reward from time step t .

2. Some use a learned value function V_ϕ , and some do not:

Algorithm	$\tilde{G}_t$ estimate	V_ϕ ?	Biased?	High variance?
REINFORCE	$\bar{R}_t$	$\times$	No	Severely
REINFORCE with Baseline	$\bar{R}_t - V_\phi(s_t)$	$\checkmark$	No	Moderately

6 REINFORCE

Recall the REINFORCE algorithm from lecture:

```

1: procedure REINFORCE( $\eta, \gamma$ )
2:   Initialize policy parameters  $\theta$  randomly
3:   while not converged do
4:     Sample a trajectory  $\tau = [(s_0, a_0, r_0), \dots, (s_T, a_T, r_T)]$  from  $\pi_\theta$ 
5:      $g \leftarrow \sum_{t=0}^T \bar{R}_t \nabla_\theta \log \pi_\theta(a_t | s_t)$  where  $\bar{R}_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ 
6:      $\theta \leftarrow \theta + \eta g$ 
7:   end while
8:   return  $\pi_\theta$ 
9: end procedure

```

1. **Policy-Based Methods vs. Value-Based Methods.** Recall that there is another “family” of reinforcement learning algorithms, called value iteration, or value-based methods. Algorithms such as Q-learning and Deep Q-Networks (DQN) belong to this family.

What is the difference between policy-based methods (such as REINFORCE) and value-based methods (such as Q-learning/DQN)?

Value-based methods (like Q-learning) learn an action-value function $Q_\phi(s, a)$ and acts by taking $\operatorname{argmax}_a Q_\phi(s, a)$ (with some exploration). REINFORCE learns the policy $\pi_\theta(a | s)$ directly, and acts by sampling from it.

2. Give one advantage and one disadvantage of REINFORCE compared to Q-Learning.

Advantage: The policy is differentiable, so REINFORCE can handle continuous or very large action spaces (we never perform argmax).
 Disadvantage: REINFORCE is on-policy and uses whole sampled trajectories, so its gradient estimates have high variance and it is sample-inefficient. Q-learning is off-policy, so it can reuse old experience (e.g. from a replay buffer).

Now, we will investigate the bias and variance of the REINFORCE algorithm, and how it affects its performance.

3. Show that the REINFORCE policy gradient estimator is unbiased, i.e. that $\mathbb{E}_{\tau \sim \pi_\theta} [g] = \nabla_\theta J(\theta)$.
Hint: You can use the fact that $\nabla_\theta p = p \nabla_\theta \log p$.

By definition, $g = \sum_{t=0}^T \bar{R}_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$ with $\bar{R}_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$. By linearity of expectation,

$$\mathbb{E}_{\tau \sim \pi_{\theta}}[g] = \sum_{t=0}^T \sum_{t'=t}^T \gamma^{t'-t} \mathbb{E}[r_{t'} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)]. \quad (1)$$

Now compare with $\nabla_{\theta} J(\theta)$. Let $p_{\theta}(\tau) = p(s_0) \prod_{t=0}^T \pi_{\theta}(a_t | s_t) p(s_{t+1} | s_t, a_t)$. Only the policy terms depend on θ , so $\nabla_{\theta} \log p_{\theta}(\tau) = \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$, and by the log-derivative trick,

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \nabla_{\theta} \sum_{\tau} p_{\theta}(\tau) R(\tau) = \sum_{\tau} p_{\theta}(\tau) R(\tau) \nabla_{\theta} \log p_{\theta}(\tau) \\ &= \sum_{t=0}^T \sum_{t'=0}^T \gamma^{t'} \mathbb{E}[r_{t'} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)] \end{aligned}$$

Terms with $t' < t$ vanish:

$$\mathbb{E}[r_{t'} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)] = \mathbb{E}\left[r_{t'} \underbrace{\mathbb{E}_{a_t \sim \pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(a_t | s_t)]}_{= \sum_a \nabla_{\theta} \pi_{\theta}(a | s_t) = \nabla_{\theta} 1 = 0}\right] = 0.$$

Dropping them leaves $\nabla_{\theta} J(\theta) = \sum_t \sum_{t' \geq t} \gamma^{t'} \mathbb{E}[r_{t'} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)]$, which is exactly (1) with an extra factor γ^t on the t -th term. Hence $\mathbb{E}[g] = \nabla_{\theta} J(\theta)$ (with the usual convention of dropping this outer γ^t factor, which is exact when $\gamma = 1$).

4. Why is having an unbiased estimator good?

An unbiased estimator has no systematic error: $\mathbb{E}[g] = \nabla_{\theta} J(\theta)$, so on average each update moves θ in the true ascent direction. With a small enough learning rate, stochastic gradient ascent then converges to a local optimum of J , and averaging more samples removes the remaining noise, whereas a biased estimator can converge to the wrong place no matter how much data we collect.

5. What does variance mean, and where does the variance in REINFORCE come from?

Variance measures how much the estimate g changes from one sampled batch of trajectories to the next, $\text{Var}(g) = \mathbb{E}[\|g - \mathbb{E}[g]\|_2^2]$. In REINFORCE it comes from the randomness in each trajectory: the sampled actions $a_t \sim \pi_{\theta}$, the stochastic transitions and rewards of the environment, and the fact that $\bar{R}_t$ adds up all remaining rewards, so more steps means more random terms. The raw (uncentered) magnitude of $\bar{R}_t$ also scales every term in g .

6. In which of the following scenarios would REINFORCE commonly suffer from high variance?

- ☐ A deterministic environment with a deterministic policy.
- ☒ $T = 1000$ with a stochastic policy.
- ☒ A stochastic environment (the same action in the same state may lead to different rewards).
- ☒ An environment where every reward is exactly 0 except for the termination state.
- ☐ A scenario with a short horizon (e.g. $T = 3$).

Common ways to fix the issue of high variance in REINFORCE is adding a baseline and moving toward other actor-critic methods (which we will look at in the homework).