Generalised Advantage Estimation

v0.3 · June 2026

Matthew Willetts, with assistance from Codex and Claude

1. The quantity

For a policy π, the advantage at (s_t, a_t) is

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t),$$

which says: *how much better was this action than the average action π would have taken from this state.* It is the natural multiplier in the policy gradient — $\nabla J \approx \mathbb{E}[\nabla \log \pi(a_t \mid s_t)\, A^\pi(s_t, a_t)]$ — because it gives the same gradient as Q but with lower variance (the V baseline has zero expectation against the score function).

The problem: Q^π is not observed. We have observed rewards $r_t, r_{t+1}, \ldots$ and a learned estimate $V_\phi(s)$.

2. The family of k-step estimators

Bellman's equation says $Q(s_t, a_t) = \mathbb{E}[r_t + \gamma V(s_{t+1})]$. So we can estimate Q by using k observed rewards before falling back on the learned V:

$$\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t) =: \delta_t$$

$$\hat{A}_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)$$

$$\hat{A}_t^{(k)} = \sum_{i=0}^{k-1} \gamma^i\, r_{t+i} + \gamma^k V(s_{t+k}) - V(s_t)$$

$$\hat{A}_t^{(\infty)} = G_t - V(s_t), \qquad G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}.$$

Each k has a different bias-variance profile:

k	bias	variance
1 (pure TD)	high (heavily relies on V)	low (only one noisy reward summed)
intermediate	intermediate	intermediate
∞ (pure MC)	zero (no bootstrap, V only enters as a mean-zero baseline)	high (full noisy trajectory return summed)

A useful identity: each k-step estimator telescopes to a sum of TD residuals,

$$\hat{A}_t^{(k)} = \sum_{i=0}^{k-1} \gamma^i \, \delta_{t+i}.$$

The V terms in the sum cancel in adjacent pairs, leaving only the δs. This makes the next step possible.

3. GAE: exponentially-weighted average

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = (1-\lambda) \sum_{k=1}^{\infty} \lambda^{k-1} \, \hat{A}_t^{(k)}.$$

The weights $(1-\lambda)\lambda^{k-1}$ form a normalised geometric distribution over k. The parameter $\lambda \in [0,1]$ slides between the two extremes:

- $\lambda = 0$: all mass on $k = 1 \to \hat{A}_t = \delta_t$ (pure TD).
- $\lambda = 1$: all mass at $k \to \infty \to \hat{A}_t = G_t - V(s_t)$ (pure MC).
- $\lambda = 0.95$ (typical): mostly long-horizon (low bias) with a fast-decaying short-horizon contribution (variance control).

4. The recursion

Substituting the telescoping identity into the weighted sum:

$$\hat{A}_t^{\mathrm{GAE}} = (1-\lambda) \sum_{k=1}^{\infty} \lambda^{k-1} \sum_{i=0}^{k-1} \gamma^i \delta_{t+i} = \sum_{i=0}^{\infty} (\gamma\lambda)^i \, \delta_{t+i}.$$

Reading the sum off the front gives the recursion:

$$\boxed{\hat{A}_t = \delta_t + \gamma\lambda \, \hat{A}_{t+1}}$$

walked **backward** over time, with boundary $\hat{A}_T = 0$. This is what you implement: one pass from $T-1$ down to 0, carrying a scalar.

5. Boundary condition

$\hat{A}_T = 0$ because past the end of collected data there are no δs to add — anything known about the future is already absorbed into $V(s_T)$, which entered the recursion via δ_{T-1}. No further correction is warranted.

If the rollout was *truncated* mid-episode, $V(s_T)$ provides a bootstrap estimate of remaining return. If the episode *terminated* naturally, $V(s_T) = 0$ (the done-mask handles this).

6. Done masking across episode boundaries

When a rollout spans multiple episodes, the GAE carry must not bleed across boundaries. With $d_t \in \{0,1\}$ marking terminal transitions:

$$\delta_t = r_t + \gamma \, V(s_{t+1}) \, (1 - d_t) - V(s_t)$$

$$\hat{A}_t = \delta_t + \gamma\lambda \, (1 - d_t) \, \hat{A}_{t+1}.$$

When $d_t = 1$: the $\gamma V(s_{t+1})$ bootstrap is zeroed (the episode ended, no future to bootstrap) **and** the GAE carry from $t+1$ is killed (the next step belongs to a fresh, unrelated episode).

7. Value targets

Training V_ϕ requires targets. The internally consistent choice is

$$R_t = \hat{A}_t + V(s_t).$$

Limits:

- $\lambda = 1$: $R_t = G_t$ (Monte Carlo return).
- $\lambda = 0$: $R_t = r_t + \gamma V(s_{t+1})(1 - d_t)$ (1-step TD target).
- General λ: lambda-weighted bootstrap-aware target.

Using $R_t = \hat{A}_t + V(s_t)$ ensures V_ϕ is trained toward the same kind of quantity that GAE implicitly assumes. Training V_ϕ on MC returns while computing advantages with $\lambda \neq 1$ would create a mismatch — the implicit V inside GAE would not match the trained V.

8. Implementation sketch

```
T = len(rewards)
advantages = torch.zeros(T)
gae = 0.0
for t in reversed(range(T)):
    next_v = values[t+1] if t+1 < T else last_value   # bootstrap or 0
    nonterm = 1.0 - dones[t]
    delta = rewards[t] + gamma * next_v * nonterm - values[t]
    gae = delta + gamma * lam * nonterm * gae
    advantages[t] = gae
returns = advantages + values
```

For PPO:

- `advantages` and `returns` are *targets*; detach before use in losses.
- Normalise advantages for stability: `(adv - adv.mean()) / (adv.std() + 1e-8)`.
- Do **not** normalise returns — they are the regression target for V_ϕ and need to stay in raw scale.

9. What λ means

λ controls how much you trust observed rewards vs the learned value function. $\lambda = 1$ trusts observations entirely (zero bias, high variance); $\lambda = 0$ trusts V heavily (low variance, high bias from imperfect V); $\lambda = 0.95$ is the standard "lean on observations but use V for variance control" choice.

10. Quick worked example

A 4-step trajectory with $\gamma = 0.99$, $\lambda = 0.95$, no terminations until the end ($d_3 = 1$):

t	r_t	$V(s_t)$	$\delta_t = r_t + \gamma V(s_{t+1})(1 - d_t) - V(s_t)$
0	1	4	$1 + 0.99 \cdot 3 - 4 = -0.03$
1	1	3	$1 + 0.99 \cdot 2 - 3 = -0.02$
2	1	2	$1 + 0.99 \cdot 1 - 2 = -0.01$
3	1	1	$1 + 0.99 \cdot 0 \cdot 0 - 1 = 0$ (terminal)

Backward recursion with $\hat{A}_4 = 0$:

t	$\hat{A}_t = \delta_t + \gamma\lambda(1 - d_t)\hat{A}_{t+1}$
3	$0 + 0.99 \cdot 0.95 \cdot 0 \cdot 0 = 0$
2	$-0.01 + 0.99 \cdot 0.95 \cdot 1 \cdot 0 = -0.01$
1	$-0.02 + 0.99 \cdot 0.95 \cdot 1 \cdot (-0.01) \approx -0.0294$
0	$-0.03 + 0.99 \cdot 0.95 \cdot 1 \cdot (-0.0294) \approx -0.0577$

And value targets $R_t = \hat{A}_t + V(s_t)$: $(3.94, 2.97, 1.99, 1.00)$. These are what V_ϕ regresses toward.

The all-negative $\hat{A}_t$ here means the value function was slightly overestimating; advantages are corrections that push V down. In real training the signs and magnitudes will fluctuate as the policy and value function co-evolve.