

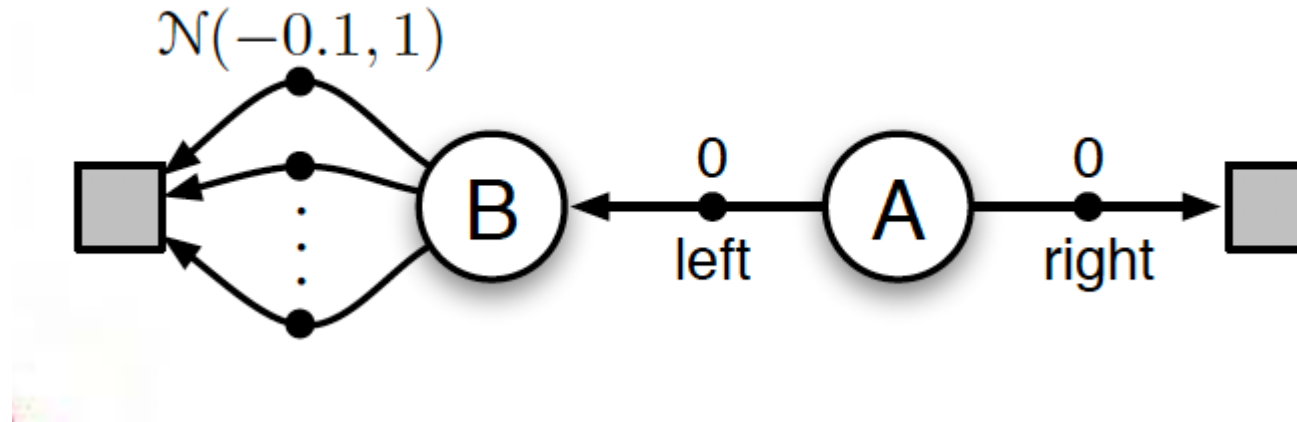
DOUBLE DQN

DQN RECAP

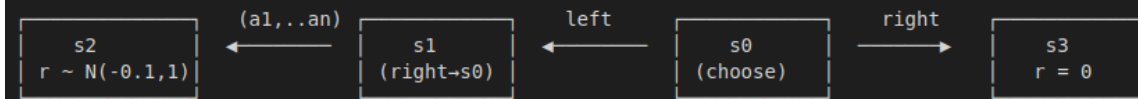
- Agent takes action a from state s and lands at s' and gets reward r .
- Tabular logic:
- $Q[s,a]$ was my current estimate of the Expected Future reward.
- My current observation says it should be $r + \gamma * Q[s', \text{next_best_action}]$
- So let's just update the table and put new estimate/ update to a moving average with some learning rate.
- In Q Network, no table but we have parameters(θ) that control the Mapping $(s,a) \rightarrow Q$. We thus treat our observation as a target
- $Y_{\text{target}} = r + \gamma * Q[s', \text{next_best_action}]$
- Tweak θ parameters such that $Q(s,a)$ is updated to the new values.
- How? Regression / Gradient Descent any other optimizer.

How do the Q updates look like?

Overestimation in Q Learning



Consider the following MDP:



We always start at the state s_0 . We can go either right or left.

Lets initialize $Q=0$ for each state action pair

Suppose we take the (LEFT, a_1) actions and recieve a reward $+0.2$ at the end of the episodes,

Q learnin episodes look like this:

$Q[s_0, \text{left}] = 0 + \max([0, 0])$

$Q[s_1, a_1] = 0.2 + 0 = 0.2$

```

+-----+-----+-----+-----+
|       | a1  | a2  | a3  | a4  |
+-----+-----+-----+-----+
| s1    | 0.2 | 0   | 0   | ... |
+-----+-----+-----+-----+
  
```

Suppose we go (LEFT, a_2) again and recieve a reward -0.1 at the end of the episode

$Q[s_0, \text{left}] = 0 + \max(Q[s_1])$

$Q[s_0, \text{left}] = 0 + 0.2 = 0.2$

$Q[s_1, a_2] = -0.1$

```

+-----+-----+-----+-----+
|       | a1  | a2  | a3  | a4  |
+-----+-----+-----+-----+
| s1    | 0.2 | -0.1| 0   | ... |
+-----+-----+-----+-----+
  
```

Let now run the algorithm for next episode,

Now suppose we go (RIGHT) this time and get a reward of 0 at the end of the episode

$Q[s_0, \text{right}] = 0$

Suppose we go (LEFT, a_3) again and recieve a reward -0.3 at the end of the episode

$Q[s_0, \text{left}] = 0 + \max(Q[s_1])$

$Q[s_0, \text{left}] = 0 + 0.2 = 0.2$

$Q[s_1, a_3] = -0.3$

```

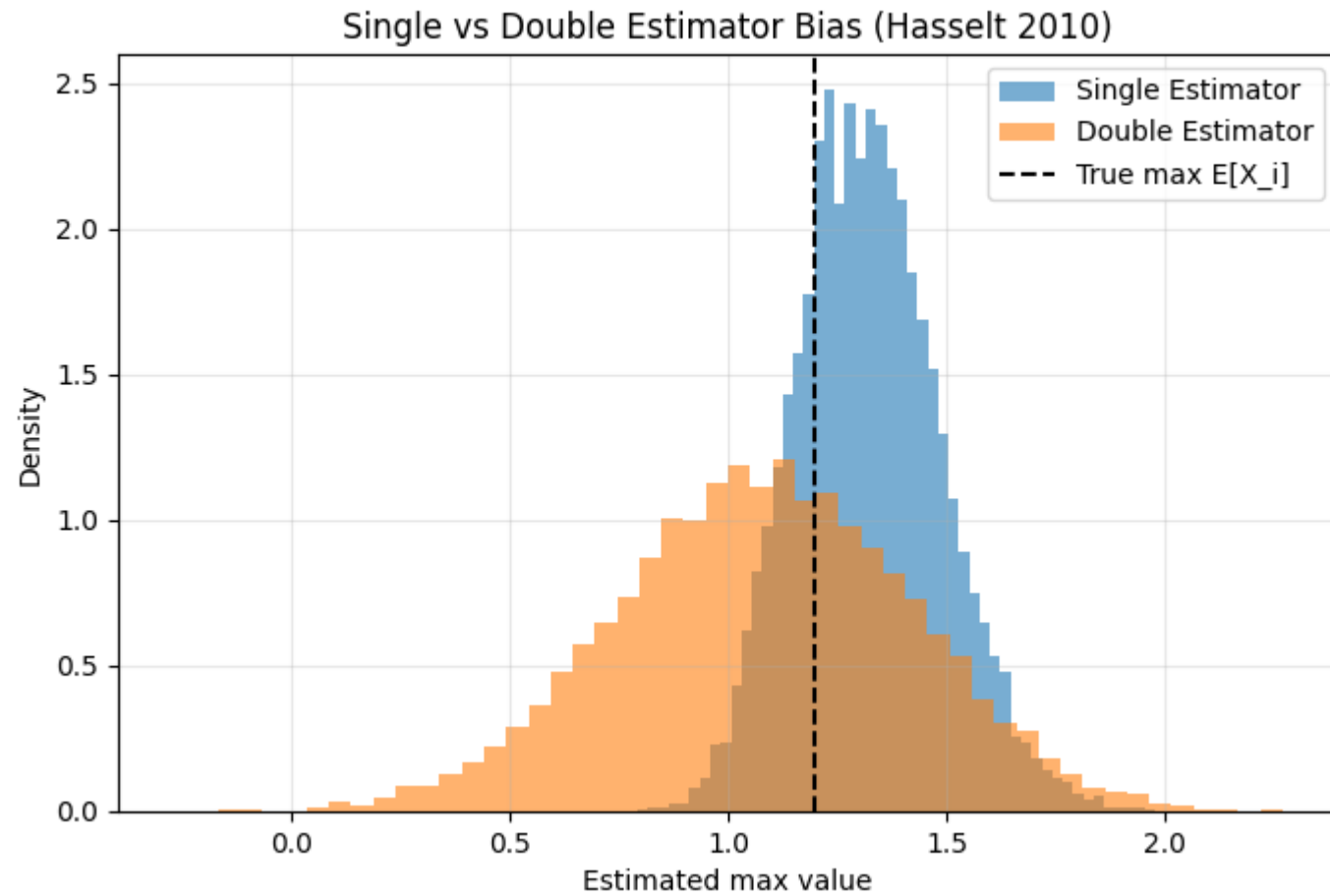
+-----+-----+-----+-----+
|       | a1  | a2  | a3  | a4  |
+-----+-----+-----+-----+
| s1    | 0.2 | -0.1| -0.3| ... |
+-----+-----+-----+-----+
  
```

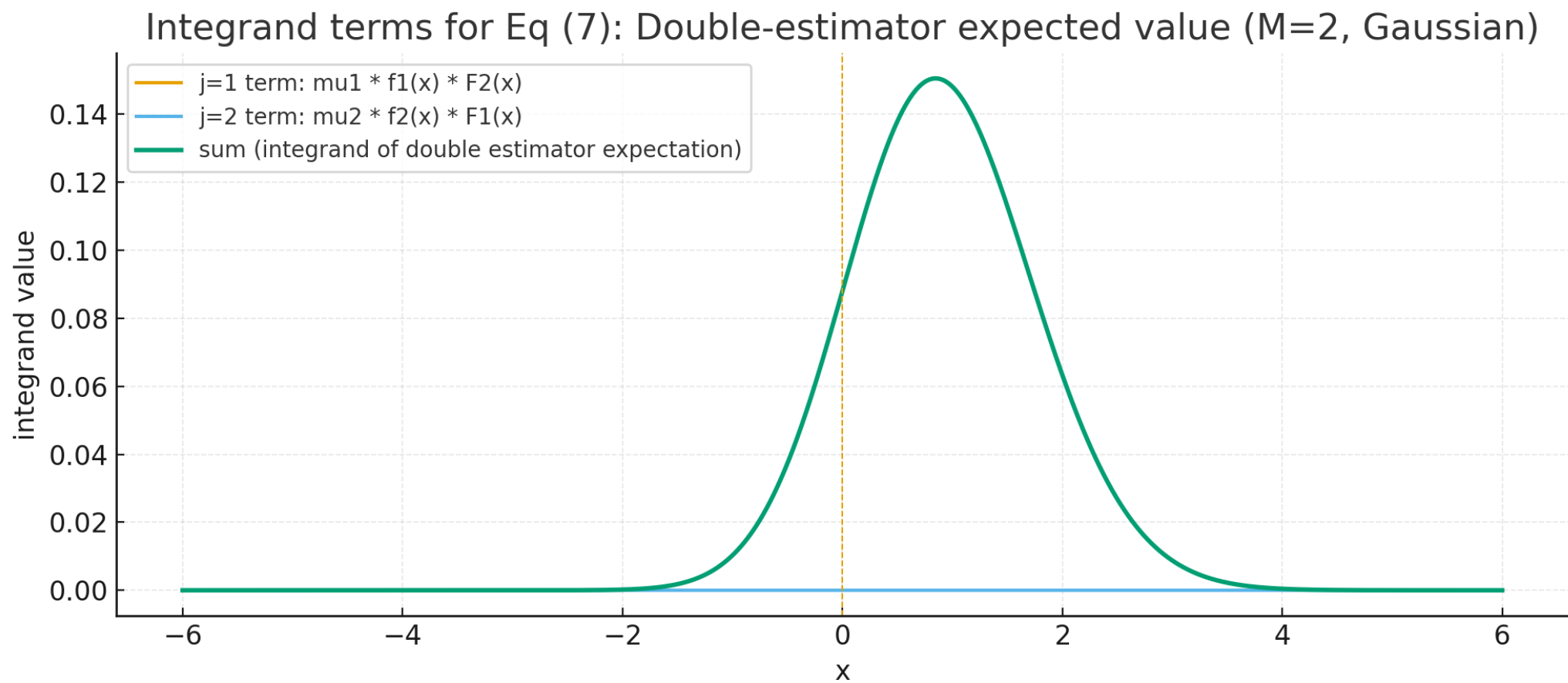
$Q[s_0, \text{left}]$ continues to be overestimated in Q learning.

Double estimator example

- Estimating the max of mean of random variables
- Consider 5 random variables with means
- [1.2, 2, 1.7, 1, 0.9]
- And variance 1.
- Goal find $\max(E\{X_i\})$

Double Estimators





Sure – for the Gaussian case we used, the two integrands become:

Eq. (5) – Expected maximum:

$$E[\max(X_1, X_2)] = \int_{-\infty}^{\infty} x [f_1(x)F_2(x) + f_2(x)F_1(x)] dx$$

$$\text{with } f_i(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu_i)^2}{2}}, F_i(x) = \frac{1}{2} \left[1 + \operatorname{erf}\left(\frac{x-\mu_i}{\sqrt{2}}\right) \right].$$

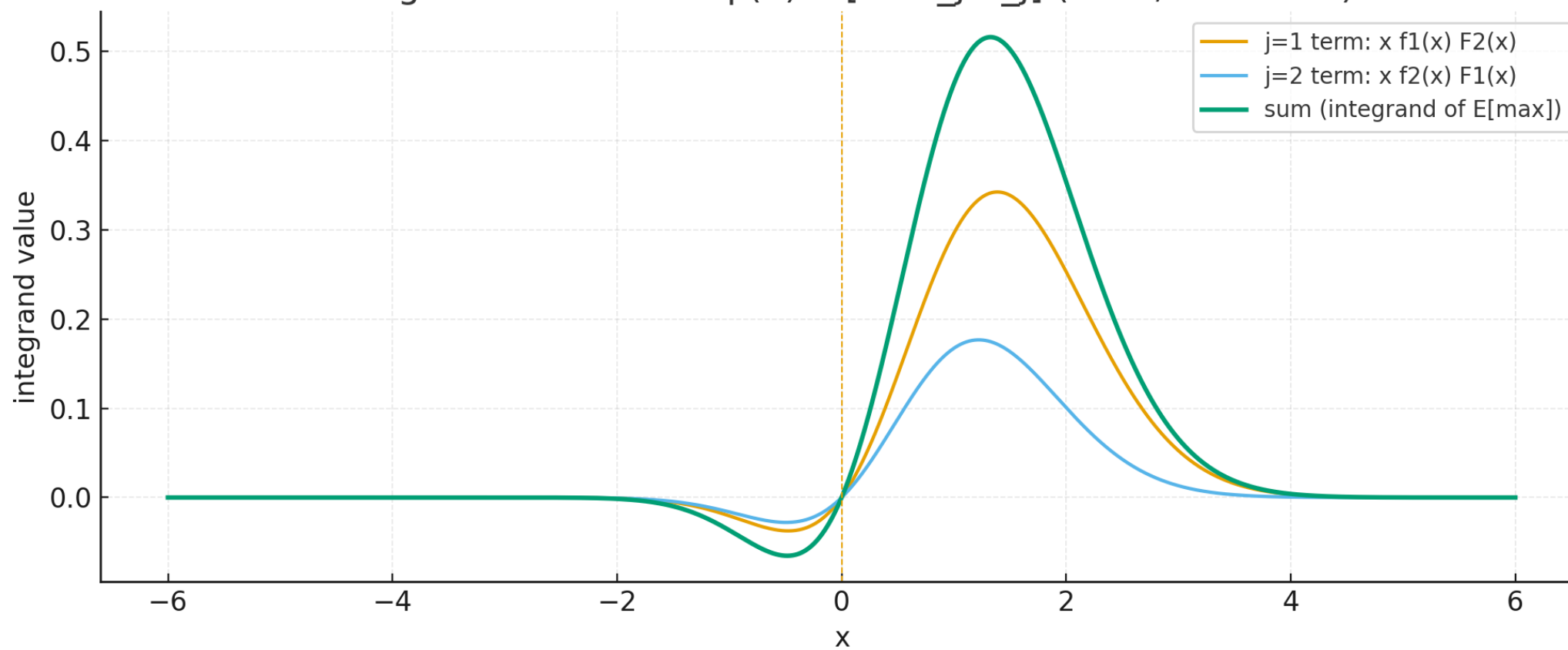
Eq. (7) – Double-estimator expectation:

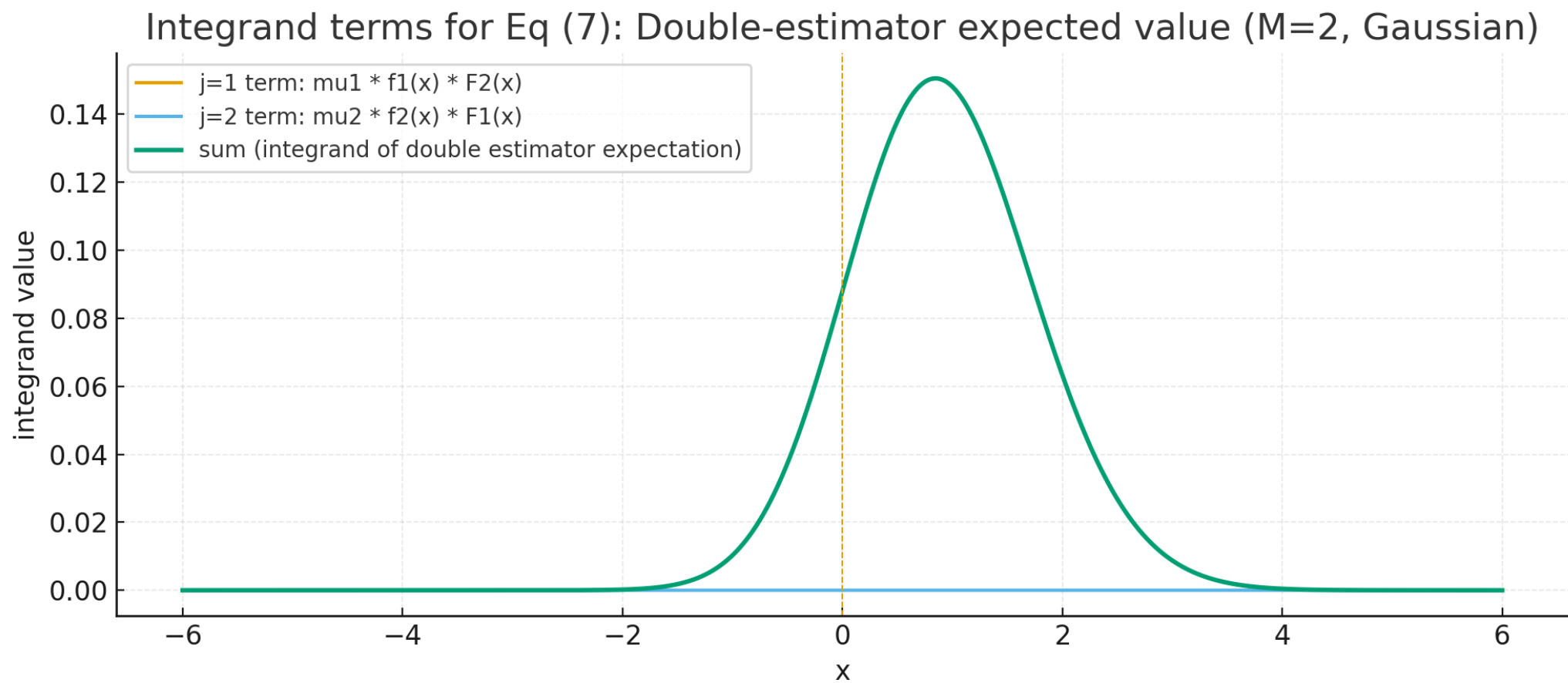
$$E[\mu_{a^*}^B] = \int_{-\infty}^{\infty} [\mu_1 f_1(x)F_2(x) + \mu_2 f_2(x)F_1(x)] dx$$

–in words: Eq. (5) weights by the *sampled value* x , while Eq. (7) replaces x with the *true mean* μ_j .



Integrand terms for Eq (5): $E[\max_j X_j]$ (M=2, Gaussian)





Policy Gradient

$$J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)}[r(\tau)] \longrightarrow \text{i.e., } J(\theta) = \int p_{\theta}(\tau) r(\tau) d\tau \longrightarrow \nabla_{\theta} J(\theta) = \int \nabla_{\theta} p_{\theta}(\tau) r(\tau) d\tau$$

$$p_{\theta}(\tau) = p_{\theta}(s_1, a_1, \dots, s_T, a_T) \longrightarrow p_{\theta}(s_1, a_1, \dots, s_T, a_T) = p(s_1) \prod_{t=1}^T \pi_{\theta}(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t)$$

To compute the gradient, we need to know the initial state distribution and transition probability distribution

The Log Trick

$$\nabla_{\theta} J(\theta) = \int \nabla_{\theta} p_{\theta}(\tau) r(\tau) d\tau$$

$$\nabla_{\theta} p_{\theta}(\tau) = p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau)$$

$$\nabla_{\theta} J(\theta) = \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) r(\tau) d\tau$$

$$= \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)]$$

i.e Gradient of objective is the expected value, and we can estimate its gradient using samples drawn from this distribution

What about $p(\tau)$ inside expectation ??

$$p_{\theta}(s_1, a_1, \dots, s_T, a_T) = p(s_1) \prod_{t=1}^T \pi_{\theta}(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t)$$



$$\log p_{\theta}(\tau) = \log p(s_1) + \sum_{t=1}^T \log \pi_{\theta}(a_t \mid s_t) + \log p(s_{t+1} \mid s_t, a_t)$$



$$\nabla_{\theta} \log p_{\theta}(\tau) = \nabla_{\theta} \sum_{t=1}^T \log \pi_{\theta}(a_t \mid s_t)$$



$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t) \right) \left(\sum_{t=1}^T r(s_t, a_t) \right) \right]$$

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)]$$

From previous slide

$$J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[r(\tau) \right]$$

$$\approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T r(s_{i,t}, a_{i,t})$$

Our final objective

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t) \right) \left(\sum_{t=1}^T r(s_t, a_t) \right) \right]$$

Final gradient of objective

No need to know about initial and transition distribution

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$

Repeat forever:

 Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \theta)$

 For each step of the episode $t = 0, \dots, T - 1$:

$G \leftarrow$ return from step t

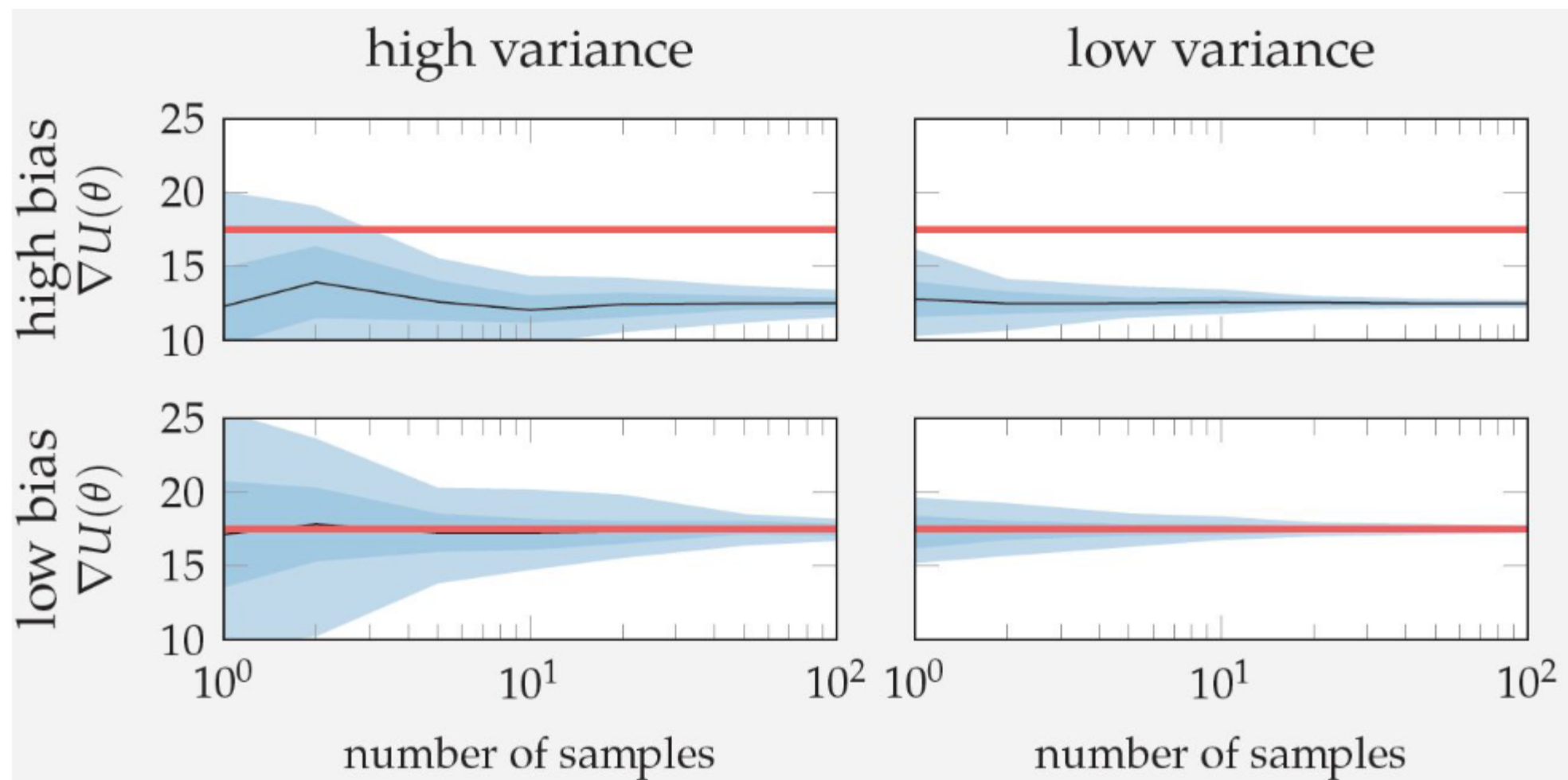
$\theta \leftarrow \theta + \alpha \gamma^t G \nabla_{\theta} \ln \pi(A_t|S_t, \theta)$

[Sutton and Barto, Sec 13.3](#)

```
def finish_episode():
    R = 0
    policy_loss = []
    returns = []
    for r in policy.rewards[::-1]:
        R = r + args.gamma * R
        returns.insert(0, R)
    returns = torch.tensor(returns)
    returns = (returns - returns.mean()) / (returns.std() + eps)
    for log_prob, R in zip(policy.saved_log_probs, returns):
        policy_loss.append(-log_prob * R)
    optimizer.zero_grad()
    policy_loss = torch.cat(policy_loss).sum()
    policy_loss.backward()
    optimizer.step()
    del policy.rewards[:]
    del policy.saved_log_probs[:]
```

[Source](#)

Bias & Variance of gradient estimate



Gradient of objective

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \left(\sum_{t=1}^T r(s_t, a_t) \right) \right]$$

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) \right) \left(\sum_{t=1}^T r(s_{i,t}, a_{i,t}) \right)$$

$$\approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) \left(\sum_{t'=1}^T r(s_{i,t'}, a_{i,t'}) \right)$$

$$\approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) \left(\sum_{t'=t}^T r(s_{i,t'}, a_{i,t'}) \right)$$

$$\approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) \hat{Q}_{i,t}$$

Reward to go

Reduced variance since we reduced the sum to be smaller number