

[2.2] DQN and VPG

David Quarel

ARENA

June 9, 2026

Beyond Tabular Learning

- All the methods up to this point assume sweeping through all state-action pairs is tractable

Beyond Tabular Learning

- All the methods up to this point assume sweeping through all state-action pairs is tractable
- What about large/continuous state spaces?
 - State aggregation?
 - Parameterised policy π_{θ} , learn best θ ?
 - Craft a heuristic by hand?
- In general, would like the agent to learn useful features for us

Beyond Tabular Learning

- All the methods up to this point assume sweeping through all state-action pairs is tractable
- What about large/continuous state spaces?
 - State aggregation?
 - Parameterised policy π_{θ} , learn best θ ?
 - Craft a heuristic by hand?
- In general, would like the agent to learn useful features for us
 - Something deep learning excels at!

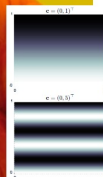
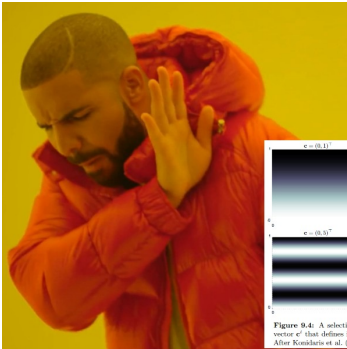


Figure 9.4: A selection of six vector e^i that defines it (s_1) is t . After Konradis et al. (2011).



9.5 Feature Construction for Linear Methods

- 9.5.1 Polynomials
- 9.5.2 Fourier Basis
- 9.5.3 Coarse Coding
- 9.5.4 Tile Coding
- 9.5.5 Radial Basis Functions

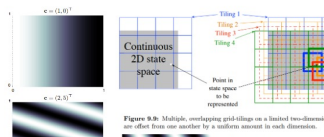


Figure 9.9: Multiple, overlapping grid-tilings on a limited two-dimension are offset from one another by a uniform amount in each dimension.

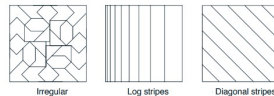
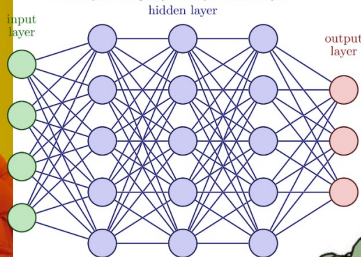


Figure 9.12: Tilings need not be grids. They can be arbitrarily shaped and non-regular, still in many cases being computationally efficient to compute.



Difficulties with using Neural Networks for RL

Neural networks expect to be trained in a supervised learning fashion, with batches of data fed in, loss computed, and gradients backpropagated.

Idea: Reduce the reinforcement learning problem to a supervised learning problem?

Difficulties with using Neural Networks for RL

Neural networks expect to be trained in a supervised learning fashion, with batches of data fed in, loss computed, and gradients backpropagated.

Idea: Reduce the reinforcement learning problem to a supervised learning problem?

- Interaction with environment is *NOT* i.i.d

Difficulties with using Neural Networks for RL

Neural networks expect to be trained in a supervised learning fashion, with batches of data fed in, loss computed, and gradients backpropagated.

Idea: Reduce the reinforcement learning problem to a supervised learning problem?

- Interaction with environment is *NOT* i.i.d
 - Collect many trajectories, dump into a buffer and shuffle until “i.i.d enough”

Difficulties with using Neural Networks for RL

Neural networks expect to be trained in a supervised learning fashion, with batches of data fed in, loss computed, and gradients backpropagated.

Idea: Reduce the reinforcement learning problem to a supervised learning problem?

- Interaction with environment is *NOT* i.i.d
 - Collect many trajectories, dump into a buffer and shuffle until “i.i.d enough”
- Rewards are sparse

Difficulties with using Neural Networks for RL

Neural networks expect to be trained in a supervised learning fashion, with batches of data fed in, loss computed, and gradients backpropagated.

Idea: Reduce the reinforcement learning problem to a supervised learning problem?

- Interaction with environment is *NOT* i.i.d
 - Collect many trajectories, dump into a buffer and shuffle until “i.i.d enough”
- Rewards are sparse
 - Q-Learning bootstraps using estimates of the future
 - VPG already handles exploration as we will see
 - Can use a critic to learn estimates of the future (tomorrow)

Difficulties with using Neural Networks for RL

Neural networks expect to be trained in a supervised learning fashion, with batches of data fed in, loss computed, and gradients backpropagated.

Idea: Reduce the reinforcement learning problem to a supervised learning problem?

- Interaction with environment is *NOT* i.i.d
 - Collect many trajectories, dump into a buffer and shuffle until “i.i.d enough”
- Rewards are sparse
 - Q-Learning bootstraps using estimates of the future
 - VPG already handles exploration as we will see
 - Can use a critic to learn estimates of the future (tomorrow)
- No ground truth to compare against

Difficulties with using Neural Networks for RL

Neural networks expect to be trained in a supervised learning fashion, with batches of data fed in, loss computed, and gradients backpropagated.

Idea: Reduce the reinforcement learning problem to a supervised learning problem?

- Interaction with environment is *NOT* i.i.d
 - Collect many trajectories, dump into a buffer and shuffle until “i.i.d enough”
- Rewards are sparse
 - Q-Learning bootstraps using estimates of the future
 - VPG already handles exploration as we will see
 - Can use a critic to learn estimates of the future (tomorrow)
- No ground truth to compare against
 - Bootstrap from current estimates (i.e. Q-Learning)

Deep Q-Networks (DQN)

- The Optimal Q-Value estimate $\hat{Q}_*(s, a; \theta)$ is now stored as a network, with parameters θ . Recall the TD-error for Q-Learning

$$\delta_t = r_{t+1} + \gamma \max_{a'} \hat{Q}_*(s_{t+1}, a'; \theta) - \hat{Q}_*(s_t, a_t; \theta)$$

Deep Q-Networks (DQN)

- The Optimal Q-Value estimate $\hat{Q}_*(s, a; \theta)$ is now stored as a network, with parameters θ . Recall the TD-error for Q-Learning

$$\delta_t = r_{t+1} + \gamma \max_{a'} \hat{Q}_*(s_{t+1}, a'; \theta) - \hat{Q}_*(s_t, a_t; \theta)$$

- Idea:** Accumulate experience $(s^i, a^i, r^i, s_{\text{new}}^i)$ via interaction, optimise θ to minimise loss $L(\theta) = \mathbb{E}[\sum_t \delta_t^2]$

Deep Q-Networks (DQN)

- The Optimal Q-Value estimate $\hat{Q}_*(s, a; \theta)$ is now stored as a network, with parameters θ . Recall the TD-error for Q-Learning

$$\delta_t = r_{t+1} + \gamma \max_{a'} \hat{Q}_*(s_{t+1}, a'; \theta) - \hat{Q}_*(s_t, a_t; \theta)$$

- **Idea:** Accumulate experience $(s^i, a^i, r^i, s_{\text{new}}^i)$ via interaction, optimise θ to minimise loss $L(\theta) = \mathbb{E}[\sum_t \delta_t^2]$
 - In practice, experience is accumulated in a buffer, and batches are sampled at random to make data “more i.i.d”

Deep Q-Networks (DQN)

- The Optimal Q-Value estimate $\hat{Q}_*(s, a; \theta)$ is now stored as a network, with parameters θ . Recall the TD-error for Q-Learning

$$\delta_t = r_{t+1} + \gamma \max_{a'} \hat{Q}_*(s_{t+1}, a'; \theta) - \hat{Q}_*(s_t, a_t; \theta)$$

- **Idea:** Accumulate experience $(s^i, a^i, r^i, s_{\text{new}}^i)$ via interaction, optimise θ to minimise loss $L(\theta) = \mathbb{E}[\sum_t \delta_t^2]$
 - In practice, experience is accumulated in a buffer, and batches are sampled at random to make data “more i.i.d”
 - Also use separate set of parameters θ_{target} for the target network, copy weights every so often for stability

Deep Q-Networks (DQN)

- The Optimal Q-Value estimate $\hat{Q}_*(s, a; \theta)$ is now stored as a network, with parameters θ . Recall the TD-error for Q-Learning

$$\delta_t = r_{t+1} + \gamma \max_{a'} \hat{Q}_*(s_{t+1}, a'; \theta) - \hat{Q}_*(s_t, a_t; \theta)$$

- Idea:** Accumulate experience $(s^i, a^i, r^i, s_{\text{new}}^i)$ via interaction, optimise θ to minimise loss $L(\theta) = \mathbb{E}[\sum_t \delta_t^2]$
 - In practice, experience is accumulated in a buffer, and batches are sampled at random to make data “more i.i.d”
 - Also use separate set of parameters θ_{target} for the target network, copy weights every so often for stability

$$\begin{aligned} L(\theta) &= \mathbb{E} \left[\left(r_{t+1} + \gamma \max_{a'} \hat{Q}_*(s_{t+1}, a'; \theta_{\text{target}}) - \hat{Q}_*(s_t, a_t; \theta) \right)^2 \right] \\ &= \frac{1}{N} \sum_{i=1}^N \left(r^i + \gamma \max_{a'} \hat{Q}_*(s_{\text{new}}^i, a'; \theta_{\text{target}}) - \hat{Q}_*(s^i, a^i; \theta) \right)^2 \end{aligned}$$

Deep Q-Networks (DQN)

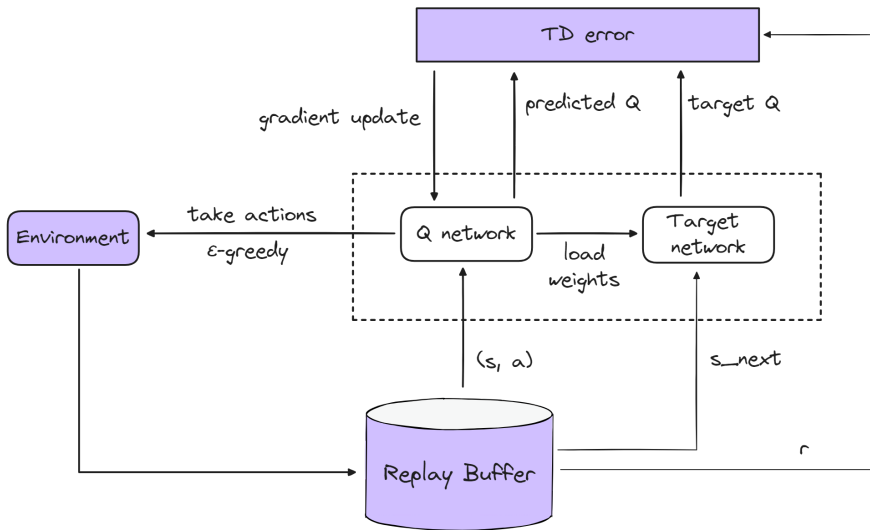
- The Optimal Q-Value estimate $\hat{Q}_*(s, a; \theta)$ is now stored as a network, with parameters θ . Recall the TD-error for Q-Learning

$$\delta_t = r_{t+1} + \gamma \max_{a'} \hat{Q}_*(s_{t+1}, a'; \theta) - \hat{Q}_*(s_t, a_t; \theta)$$

- Idea:** Accumulate experience $(s^i, a^i, r^i, s_{\text{new}}^i)$ via interaction, optimise θ to minimise loss $L(\theta) = \mathbb{E}[\sum_t \delta_t^2]$
 - In practice, experience is accumulated in a buffer, and batches are sampled at random to make data “more i.i.d”
 - Also use separate set of parameters θ_{target} for the target network, copy weights every so often for stability

$$\begin{aligned} L(\theta) &= \mathbb{E} \left[\left(r_{t+1} + \gamma \max_{a'} \hat{Q}_*(s_{t+1}, a'; \theta_{\text{target}}) - \hat{Q}_*(s_t, a_t; \theta) \right)^2 \right] \\ &= \frac{1}{N} \sum_{i=1}^N \left(r^i + \gamma \max_{a'} \hat{Q}_*(s_{\text{new}}^i, a'; \theta_{\text{target}}) - \hat{Q}_*(s^i, a^i; \theta) \right)^2 \end{aligned}$$

Gradient descent to minimize $L(\theta)$



Deep Q-Networks (DQN) for Episodic Environments

- Annoying caveat for episodic environments: Need to modify the TD-error, depending if s_{t+1} is a terminal state.
- Assume environment samples $(s_{t+1}, r_{t+1}, d_{t+1})$ given (s_t, a_t) , where d_{t+1} (done) indicates if the episode ended on timestep $t + 1$ (s_{t+1} is arbitrary in such cases).

$$\begin{aligned}\delta_t &= y_t - \hat{Q}_*(s_t, a_t; \theta) \\ y_t &= \begin{cases} r_{t+1} & d_{t+1} = \text{True} \\ r_{t+1} + \gamma \max_{a'} \hat{Q}_*(s_{t+1}, a'; \theta_{\text{target}}) & d_{t+1} = \text{False} \end{cases} \\ &= r_{t+1} + \gamma \max_{a'} \hat{Q}_*(s_{t+1}, a'; \theta_{\text{target}}) \llbracket \text{not } d_{t+1} \rrbracket\end{aligned}$$

Loss function is now

$$L(\theta) = \frac{1}{N} \sum_{i=1}^N \left(y_t - \hat{Q}_*(s_t, a_t; \theta) \right)^2$$

Algorithm 1 Deep Q-Learning with Replay Buffer

Input: Environment p , Number of episodes M , replay buffer size N

- 1: Initialise replay buffer $\mathcal{D}$ to capacity N
- 2: **for** episode = 1 to M **do**
- 3: Sample initial state s from environment
- 4: $d \leftarrow \text{False}$
- 5: Initialize target parameters $\theta_{\text{target}} \leftarrow \theta$
- 6: **while** $d = \text{False}$ **do**
- 7: $a \leftarrow \begin{cases} \text{random action} & \text{prob } \varepsilon \\ \arg \max_{a'} Q(s, a'; \theta) & \text{prob } 1 - \varepsilon \end{cases}$
- 8: Sample $(s_{\text{new}}, r, d) \sim p(\cdot | s, a)$
- 9: Store experience $(s, a, r, s_{\text{new}}, d)$ in $\mathcal{D}$
- 10: $s_{\text{new}} \leftarrow s$
- 11: **if** Learning on this step **then**
- 12: Sample minibatch $B \leftarrow \{(s^i, a^i, r^i, s_{\text{new}}^i, d^i)\}_{i=1}^{|B|}$ from $\mathcal{D}$
- 13: **for** $j = 1$ to $|B|$ **do**
- 14: $y^j \leftarrow \begin{cases} r^j & d^j = \text{True} \\ r^j + \gamma \max_{a'} Q(s_{\text{new}}^j, a'; \theta_{\text{target}}) & d^j = \text{False} \end{cases}$
- 15: **end for**
- 16: Define loss $L(\theta) = \frac{1}{|B|} \sum_{i=1}^{|B|} (y^i - Q(s^i, a^i; \theta))^2$
- 17: Gradient descent step $\theta \leftarrow \theta - \eta \nabla_{\theta} L(\theta)$
- 18: **end if**
- 19: **if** Update target this step **then**
- 20: $\theta_{\text{target}} \leftarrow \theta$
- 21: **end if**
- 22: **end while**
- 23: **end for**

- **Action space** $\mathcal{A}$ is still discrete, and finite
- **State space** $\mathcal{S}$ is continuous (a subspace of $\mathbb{R}^n$), and each state is a vector $\mathbf{s} \in \mathcal{S}$.
- Environment is **episodic**, and agent always starts at/near **starting state** $\mathbf{s}_0$. We generate a full episode, and then afterwards learn. No discount ($\gamma = 1$).
- Define measure of performance (**joy**) as

$$J(\boldsymbol{\theta}) := V_{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_0)$$

- Policy $\pi_{\boldsymbol{\theta}} \equiv \pi(\cdot | \mathbf{s}; \boldsymbol{\theta})$ is a conditional probability distribution $\pi_{\boldsymbol{\theta}} : \mathcal{S} \rightarrow \Delta\mathcal{A}$ over actions, mapping state to a logit vector over actions.
- Gradient $\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}$ is well-defined.
- Choose $\boldsymbol{\theta}$ to maximize joy $J(\boldsymbol{\theta})$ via gradient *ascent*

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Softmax policy

- In practice, $\pi_{\theta} : \mathcal{S} \rightarrow \Delta\mathcal{A}$ is a neural network $\mathbf{h}$, mapping a state vector $\mathbf{s}$ to a vector of logits, one for each action $\mathcal{A} = \{a_1, \dots, a_n\}$.

$$\mathbf{h}(\mathbf{s}; \theta) = [h_1(\mathbf{s}; \theta), \dots, h_n(\mathbf{s}; \theta)]$$

from which the action is sampled by softmax'ing the logits

$$\pi(a_i | \mathbf{s}; \theta) := \frac{\exp h_i(\mathbf{s}; \theta)}{\sum_j \exp h_j(\mathbf{s}; \theta)}$$

- More principled than ϵ -greedy, as we get exploration for free, and exploratory actions are not selected at random, but with a bias towards actions believed to be good.

Policy Gradient Framework

- Assume episodic environment, episode length (horizon) T , no discount $\gamma = 1$.
- Assume fixed starting state s_0 . Markov environment $\mu(s'|s, a)$.
- Define $J(\theta) = V_{\pi_\theta}(s_0) = \mathbb{E}_{\tau \sim \pi_\theta}[G(\tau)]$.
- Let $\tau = s_0, a_0, r_1, s_1, \dots, s_{T-1}, a_{T-1}, r_T, s_T$ denote a trajectory
- $G(\tau) = \sum_{t=1}^T r_t$ is the undiscounted return for trajectory τ .

$$\begin{aligned}\Pr(\tau|\theta) &= \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) \mu(s_{t+1}|s_t, a_t) \\ &= \pi_\theta(a_0|s_0) \mu(s_1|s_0, a_0) \dots \pi_\theta(a_{T-1}|s_{T-1}) \mu(s_T|s_{T-1}, a_{T-1})\end{aligned}$$

is the prob. of sampling trajectory τ from enviro. μ given params. θ .

Want to compute $\nabla_\theta J(\theta)$.

Policy Gradient Framework

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [G(\tau)]$$

Policy Gradient Framework

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} [G(\tau)] \\ &= \nabla_{\boldsymbol{\theta}} \sum_{\tau} \text{Pr}(\tau|\boldsymbol{\theta}) G(\tau)\end{aligned}$$

Policy Gradient Framework

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} [G(\tau)] \\ &= \nabla_{\boldsymbol{\theta}} \sum_{\tau} \text{Pr}(\tau|\boldsymbol{\theta}) G(\tau) \\ &= \sum_{\tau} \nabla_{\boldsymbol{\theta}} \text{Pr}(\tau|\boldsymbol{\theta}) G(\tau)\end{aligned}$$

Policy Gradient Framework

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [G(\tau)] \\ &= \nabla_{\theta} \sum_{\tau} \text{Pr}(\tau|\theta) G(\tau) \\ &= \sum_{\tau} \nabla_{\theta} \text{Pr}(\tau|\theta) G(\tau) \\ &= \sum_{\tau} \text{Pr}(\tau|\theta) (\nabla_{\theta} \log \text{Pr}(\tau|\theta) G(\tau)) \quad \text{as } \nabla \log f(x) = \frac{\nabla f(x)}{f(x)}\end{aligned}$$

Policy Gradient Framework

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [G(\tau)] \\&= \nabla_{\theta} \sum_{\tau} \text{Pr}(\tau|\theta) G(\tau) \\&= \sum_{\tau} \nabla_{\theta} \text{Pr}(\tau|\theta) G(\tau) \\&= \sum_{\tau} \text{Pr}(\tau|\theta) (\nabla_{\theta} \log \text{Pr}(\tau|\theta) G(\tau)) \quad \text{as } \nabla \log f(x) = \frac{\nabla f(x)}{f(x)} \\&= \mathbb{E}_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log \text{Pr}(\tau|\theta) G(\tau)]\end{aligned}$$

Policy Gradient Framework

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [G(\tau)] \\&= \nabla_{\theta} \sum_{\tau} \text{Pr}(\tau|\theta) G(\tau) \\&= \sum_{\tau} \nabla_{\theta} \text{Pr}(\tau|\theta) G(\tau) \\&= \sum_{\tau} \text{Pr}(\tau|\theta) (\nabla_{\theta} \log \text{Pr}(\tau|\theta) G(\tau)) \quad \text{as } \nabla \log f(x) = \frac{\nabla f(x)}{f(x)} \\&= \mathbb{E}_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log \text{Pr}(\tau|\theta) G(\tau)]\end{aligned}$$

Problem: $\text{Pr}(\tau|\theta)$ is unknown

Note that

$$\nabla_{\theta} \log \text{Pr}(\tau|\theta) = \nabla_{\theta} \log \prod_{t=0}^{T-1} \pi_{\theta}(a_t|s_t) \mu(s_{t+1}|s_t, a_t)$$

Note that

$$\begin{aligned}\nabla_{\theta} \log \text{Pr}(\tau|\theta) &= \nabla_{\theta} \log \prod_{t=0}^{T-1} \pi_{\theta}(a_t|s_t) \mu(s_{t+1}|s_t, a_t) \\ &= \nabla_{\theta} \sum_{t=0}^{T-1} \log \left(\pi_{\theta}(a_t|s_t) \mu(s_{t+1}|s_t, a_t) \right)\end{aligned}$$

Note that

$$\begin{aligned}\nabla_{\theta} \log \text{Pr}(\tau|\theta) &= \nabla_{\theta} \log \prod_{t=0}^{T-1} \pi_{\theta}(a_t|s_t) \mu(s_{t+1}|s_t, a_t) \\ &= \nabla_{\theta} \sum_{t=0}^{T-1} \log \left(\pi_{\theta}(a_t|s_t) \mu(s_{t+1}|s_t, a_t) \right) \\ &= \nabla_{\theta} \sum_{t=0}^{T-1} (\log \pi_{\theta}(a_t|s_t) + \log \mu(s_{t+1}|s_t, a_t))\end{aligned}$$

Note that

$$\begin{aligned}\nabla_{\theta} \log \text{Pr}(\tau|\theta) &= \nabla_{\theta} \log \prod_{t=0}^{T-1} \pi_{\theta}(a_t|s_t) \mu(s_{t+1}|s_t, a_t) \\&= \nabla_{\theta} \sum_{t=0}^{T-1} \log \left(\pi_{\theta}(a_t|s_t) \mu(s_{t+1}|s_t, a_t) \right) \\&= \nabla_{\theta} \sum_{t=0}^{T-1} (\log \pi_{\theta}(a_t|s_t) + \log \mu(s_{t+1}|s_t, a_t)) \\&= \nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t|s_t) + \cancel{\nabla_{\theta} \sum_{t=0}^{T-1} \log \mu(s_{t+1}|s_t, a_t)}\end{aligned}$$

Note that

$$\begin{aligned}\nabla_{\theta} \log \text{Pr}(\tau|\theta) &= \nabla_{\theta} \log \prod_{t=0}^{T-1} \pi_{\theta}(a_t|s_t) \mu(s_{t+1}|s_t, a_t) \\&= \nabla_{\theta} \sum_{t=0}^{T-1} \log \left(\pi_{\theta}(a_t|s_t) \mu(s_{t+1}|s_t, a_t) \right) \\&= \nabla_{\theta} \sum_{t=0}^{T-1} (\log \pi_{\theta}(a_t|s_t) + \log \mu(s_{t+1}|s_t, a_t)) \\&= \nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t|s_t) + \cancel{\nabla_{\theta} \sum_{t=0}^{T-1} \log \mu(s_{t+1}|s_t, a_t)} \\&= \nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t|s_t)\end{aligned}$$

This gives the core equation for Vanilla Policy Gradient (VPG) or **REINFORCE**

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) G(\tau) \right]$$

VPG generates a trajectory, learns, then discards it, generates another.

Clever trick 1: The future cannot affect the past

- In REINFORCE, each $\log \pi_{\theta}(a_t|s_t)$ is multiplied by the full return $G(\tau)$.
- But a_t cannot influence rewards that happened **before** it.
- Can be shown that we can replace $G(\tau)$ with the **reward-to-go**:

$$G_t := \sum_{t'=t+1}^T r_{t'} = r_{t+1} + \dots + r_T$$

- This gives a lower-variance, unbiased estimator.

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) G(\tau) \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) G_t \right]\end{aligned}$$

Algorithm 1 Vanilla Policy Gradient ($\gamma = 1$)

Input: Environment p , Number of episodes M

- 1: **for** episode = 1 to M **do**
 - 2: Generate episode $s_0, a_0, r_1, s_1, \dots, s_{T-1}, a_{T-1}, r_T, s_T$
 - 3: Define $G_t = \sum_{i=t}^{T-1} r_i$ for $0 \leq t \leq T-1$
 - 4: Define gain $J(\boldsymbol{\theta}) = \sum_{t=0}^{T-1} G_t \log \pi_{\boldsymbol{\theta}}(a_t | s_t)$
 - 5: Gradient **ascent** step $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
 - 6: **end for**
-

VPG generates a trajectory, learns, then discards it, generates another.

Importance Sampling Theorem (IST)

For any two probability distributions p and q where $q(x) > 0$ whenever $p(x) > 0$:

$$\mathbb{E}_{x \sim p}[f(x)] = \mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right]$$

Importance Sampling Theorem (IST)

For any two probability distributions p and q where $q(x) > 0$ whenever $p(x) > 0$:

$$\mathbb{E}_{x \sim p}[f(x)] = \mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right]$$

Proof:

Importance Sampling Theorem (IST)

For any two probability distributions p and q where $q(x) > 0$ whenever $p(x) > 0$:

$$\mathbb{E}_{x \sim p}[f(x)] = \mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right]$$

Proof:

$$\mathbb{E}_{x \sim p}[f(x)] = \sum_x p(x) f(x)$$

Importance Sampling Theorem (IST)

For any two probability distributions p and q where $q(x) > 0$ whenever $p(x) > 0$:

$$\mathbb{E}_{x \sim p}[f(x)] = \mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right]$$

Proof:

$$\begin{aligned} \mathbb{E}_{x \sim p}[f(x)] &= \sum_x p(x) f(x) \\ &= \sum_x \frac{p(x)}{q(x)} q(x) f(x) \end{aligned}$$

Importance Sampling Theorem (IST)

For any two probability distributions p and q where $q(x) > 0$ whenever $p(x) > 0$:

$$\mathbb{E}_{x \sim p}[f(x)] = \mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right]$$

Proof:

$$\begin{aligned} \mathbb{E}_{x \sim p}[f(x)] &= \sum_x p(x) f(x) \\ &= \sum_x \frac{p(x)}{q(x)} q(x) f(x) \\ &= \sum_x q(x) \frac{p(x)}{q(x)} f(x) \end{aligned}$$

Importance Sampling Theorem (IST)

For any two probability distributions p and q where $q(x) > 0$ whenever $p(x) > 0$:

$$\mathbb{E}_{x \sim p}[f(x)] = \mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right]$$

Proof:

$$\begin{aligned} \mathbb{E}_{x \sim p}[f(x)] &= \sum_x p(x) f(x) \\ &= \sum_x \frac{p(x)}{q(x)} q(x) f(x) \\ &= \sum_x q(x) \frac{p(x)}{q(x)} f(x) \\ &= \mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right] \end{aligned}$$

Importance Sampling Theorem (IST)

For any two probability distributions p and q where $q(x) > 0$ whenever $p(x) > 0$:

$$\mathbb{E}_{x \sim p}[f(x)] = \mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right]$$

Proof:

$$\begin{aligned} \mathbb{E}_{x \sim p}[f(x)] &= \sum_x p(x) f(x) \\ &= \sum_x \frac{p(x)}{q(x)} q(x) f(x) \\ &= \sum_x q(x) \frac{p(x)}{q(x)} f(x) \\ &= \mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right] \end{aligned}$$

The ratio $\frac{p(x)}{q(x)}$ is called the **importance weight** or **importance ratio**.

VPG with Importance Sampling

Problem: VPG is on-policy - we must generate fresh trajectories for each policy update.

Solution: Use importance sampling to reuse old trajectories from $\pi_{\theta_{old}}$.

The **exact** IS-corrected gradient requires the full trajectory probability ratio:

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \mathbb{E}_{\tau \sim \pi_{\theta_{old}}} \left[\frac{\Pr(\tau|\theta)}{\Pr(\tau|\theta_{old})} \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) G_t \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta_{old}}} \left[\prod_{t'=0}^{T-1} \frac{\pi_{\theta}(a_{t'}|s_{t'})}{\pi_{\theta_{old}}(a_{t'}|s_{t'})} \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) G_t \right]\end{aligned}$$

Problem: The product of ratios has *very high variance* for long trajectories.

VPG with Importance Sampling (cont.)

Common approximation: Use per-step importance ratios instead:

$$\begin{aligned}\nabla_{\theta} J(\theta) &\approx \mathbb{E}_{\tau \sim \pi_{\theta_{old}}} \left[\sum_{t=0}^{T-1} \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) G_t \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta_{old}}} \left[\sum_{t=0}^{T-1} \frac{\nabla_{\theta} \pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} G_t \right] \quad (\text{as } \pi \nabla \log \pi = \nabla \pi)\end{aligned}$$

Why use the ratio at timestep t ? For the term $\nabla_{\theta} \log \pi_{\theta}(a_t|s_t) G_t$:

- Action a_t was sampled from $\pi_{\theta_{old}}(\cdot|s_t)$, but we evaluate $\nabla_{\theta} \log \pi_{\theta}(a_t|s_t)$
- Correct for this mismatch with $\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$

What we ignore: State s_t and return G_t also depend on $\pi_{\theta_{old}}$, but when $\pi_{\theta} \approx \pi_{\theta_{old}}$, these effects are small.

VPG with Importance Sampling Algorithm

Input: Environment, policy π_θ , batch size N , learning rate α

Algorithm: Repeat for each iteration:

- ① Collect N episodes in parallel using current policy π_θ :
 - Generate episode $\tau^i = (s_0^i, a_0^i, r_1^i, s_1^i, \dots, s_T^i)$
 - Store log probabilities: $\log \pi_{\text{old}}(a_t^i | s_t^i) = \log \pi_\theta(a_t^i | s_t^i)$
 - Compute returns $G_t^i = \sum_{k=t+1}^T r_k^i$ for all t
 - Store $\{\tau^i, G_t^i, \log \pi_{\text{old}}(a_t^i | s_t^i)\}_{i=1}^N$.
- ② Sample batch $\mathcal{B} = \{\tau^i, G_t^i, \log \pi_{\text{old}}(a_t^i | s_t^i)\}_{i=1}^{|\mathcal{B}|}$.
- ③ Compute importance sampling joy:

$$J(\theta) = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \sum_{t=0}^{T-1} \frac{\pi_\theta(a_t^i | s_t^i)}{\pi_{\text{old}}(a_t^i | s_t^i)} G_t^i$$

- ④ Gradient ascent: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Can reuse trajectories, or slice large rollouts into smaller batches.