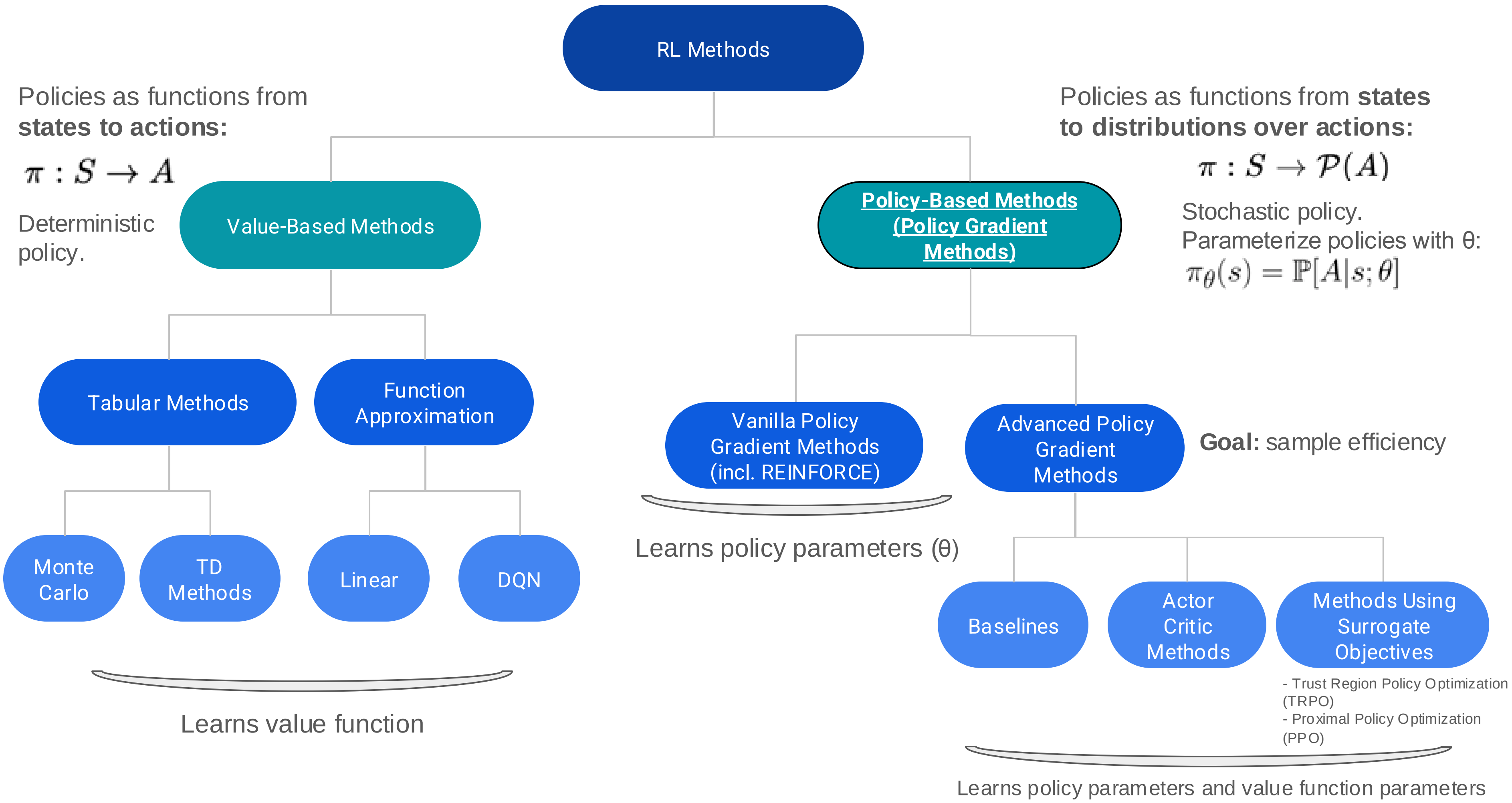
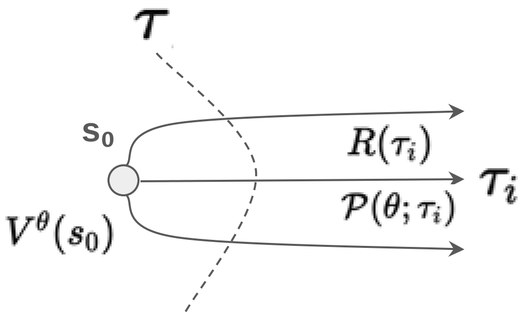
Part 2: Advanced Policy Gradient Methods

Lain Mustafaoglu

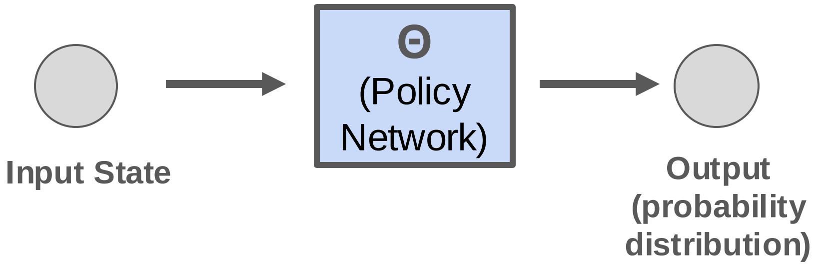

RL Methods
Policies as functions from **states to actions:**
$\pi : S \rightarrow A$
Deterministic policy.
Value-Based Methods
Tabular Methods
Function Approximation
Monte Carlo
TD Methods
Linear
DQN
Learns value function
Policies as functions from **states to distributions over actions:**
$\pi : S \rightarrow \mathcal{P}(A)$
Stochastic policy.
Parameterize policies with θ:
$\pi_\theta(s) = \mathbb{P}[A|s;\theta]$
Policy-Based Methods (Policy Gradient Methods)
Vanilla Policy Gradient Methods (incl. REINFORCE)
Learns policy parameters (θ)
Advanced Policy Gradient Methods
Goal: sample efficiency
Baselines
Actor Critic Methods
Methods Using Surrogate Objectives
- Trust Region Policy Optimization (TRPO)
- Proximal Policy Optimization (PPO)
Learns policy parameters and value function parameters

Valuations



Trajectories starting at S_0

$$V^\theta(S_0) = E_{\tau\sim\pi_\theta}[R(\tau)] = \sum_{\tau_i} R(\tau_i)\mathcal{P}(\theta;\tau_i)$$

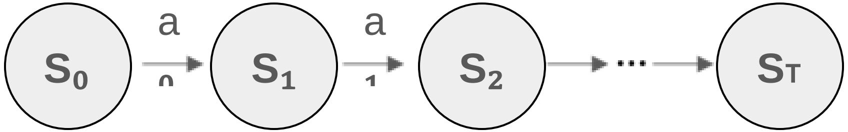


Gradient-ascent for policy improvement

$$\theta \leftarrow \theta + \alpha * \nabla_\theta V^\theta(S_0)$$

$$\nabla_\theta V^\theta(S_0) = \sum_{\tau_i} R(\tau_i)\nabla_\theta \mathcal{P}(\theta;\tau_i)$$

Policy-Based Monte-Carlo: REINFORCE



$R(S_0, a_0, S_1)$

$\pi_\theta(a_0|S_0) * P(S_1|S_0, a_0)$

$R(S_1, a_1, S_2)$

$\pi_\theta(a_1|S_1) * P(S_2|S_1, a_1)$

$$\nabla_\theta V^\theta(S_0) = \sum_{\tau_i} \underbrace{P(\tau_i)\pi_\theta(\tau_i)}_{\text{probability of trajectory}} \underbrace{R(\tau_i) * \nabla_\theta log(\pi_\theta(\tau_i))}_{\text{contribution to gradient}}$$

Construct **multiset** of episodes by Monte Carlo sampling: probability ≈ frequency

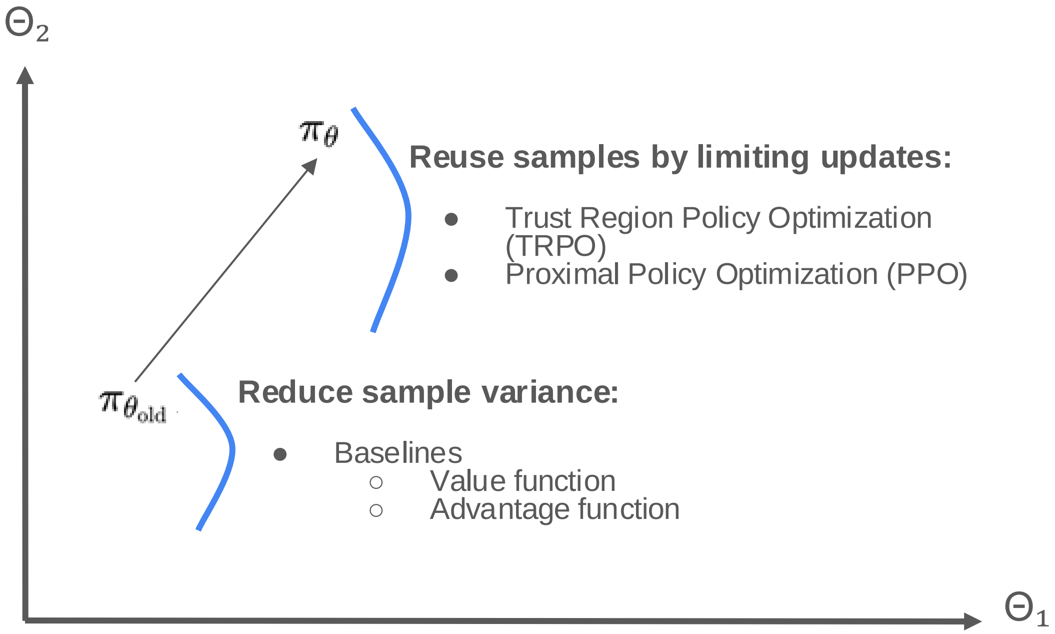
$$\nabla_\theta \widehat{V^\theta}(S_0) \approx \frac{1}{|Episodes|} \sum_{e_i} R(e_i) * \nabla_\theta log(\pi_\theta(e_i))$$

REINFORCE algorithm:

1. Sample multiset of episodes $\{e_i\}$ from π_θ.
2. $\nabla_\theta \widehat{V^\theta}(S_0) \approx \frac{1}{|Episodes|} \sum_{e_i} R(e_i) * \nabla_\theta log(\pi_\theta(e_i))$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta \widehat{V^\theta}(S_0)$

Improvements Over Vanilla Policy Gradient Methods

- **Sample efficiency: getting more accurate gradient estimates without collecting more samples**
    a. **For given policy:** reduce variance across samples (episodes)
    b. **Across different policies:** limit policy update to permit reuse of samples from previous policy



Baselines

- Use a **baseline b(s)** in gradient calculation:
  - Value subtracted from reward to reduce variance of the gradient estimate without changing the expectation
  - Compare current episode to some measure of average performance
  - Increase/decrease probability of episode based on how much better/worse it performed compared to average
- **Gradient calculation with baseline:**

$$\nabla_\theta \hat{V}^\theta(S_0) = \frac{1}{|Episodes|} \sum_{e_i \in Episodes} \nabla_\theta \log \pi_\theta(e_i) \cdot R(e_i)$$

$$\nabla_\theta \hat{V}^\theta = \frac{1}{|Episodes|} \sum_{e_i \in Episodes} \nabla_\theta \log \pi_\theta(e_i) \cdot (R(e_i) - \boxed{b(s_i)})$$

- Baseline can be constant or state-dependent
  - **Common baseline choice for state-dependent baselines:** for each state, approximation of value function by MC or TD, Q-value function, advantage function

Optimal Constant Baseline

Baselines do not affect the expectation, they only affect the variance:

$$\mathrm{Var}[x] = E[x^2] - E[x]^2$$

$$\nabla_\theta \hat{V}^\theta = \frac{1}{|Episodes|} \sum_{e \in Episodes} \nabla_\theta \log \pi_\theta(e)\,(R(e) - b)$$

$$\mathrm{Var} = \mathbb{E}_{e\sim\pi_\theta(e)}\left[(\nabla_\theta \log \pi_\theta(e)(R(e)-b))^2\right] - \left(\mathbb{E}_{e\sim\pi_\theta(e)}\boxed{[\nabla_\theta \log \pi_\theta(e)(R(e)-b)]}\right)^2$$

this bit is just $\mathbb{E}_{e\sim\pi_\theta(e)}[\nabla_\theta \log \pi_\theta(e) R(e)]$ (baselines are unbiased in expectation)

$$\frac{d\,\mathrm{Var}}{db} = \frac{d}{db}\mathbb{E}\left[\boxed{g(e)}^2(R(e)-b)^2\right] = \frac{d}{db}\left(\mathbb{E}[g(e)^2R(e)^2] - 2b\mathbb{E}[g(e)^2R(e)] + b^2\mathbb{E}[g(e)^2]\right)$$

$$\boxed{g(e) = \nabla_\theta \log \pi_\theta(e)} \qquad = -2\mathbb{E}[g(e)^2R(e)] + 2b\mathbb{E}[g(e)^2] = 0$$

$$b = \frac{\mathbb{E}[g(e)^2R(e)]}{\mathbb{E}[g(e)^2]}$$

Optimal constant baseline

Source

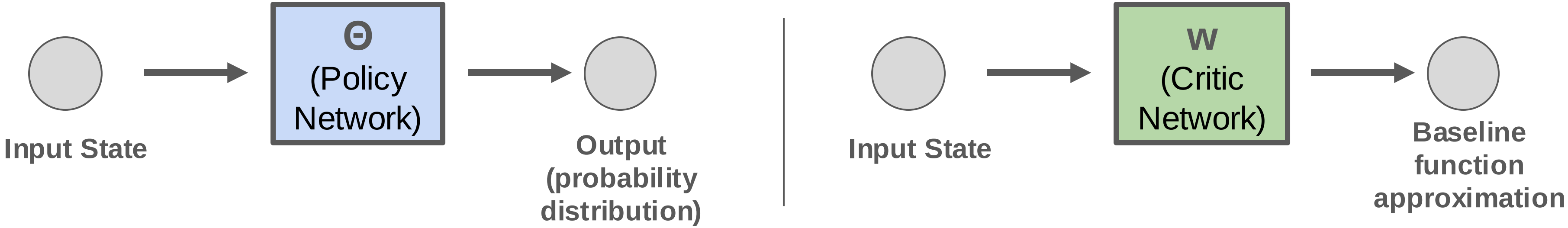
Reducing Variance with Function Approximation

- **Key idea:** Use different baseline value for each state
- **New framework:** Have another network, w, that approximates baseline value to inform gradient updates made by Θ, policy network
- Methods that follow this framework are **Actor-Critic Methods:**
    - Policy gradient methods with baseline approximation
    - Combines policy-based (actor) and value-based (critic) methods
    - **Critic** evaluates policy by estimating the baseline and updating **baseline parameters w**
    - **Actor** updates **policy parameters Θ** by gradient ascent using the value estimated by the critic

$$\delta_t \leftarrow R_t - V(S_t; w)$$

$$\theta \leftarrow \theta + \alpha\gamma^t \delta_t \nabla_\theta \log \pi_\theta(A_t|S_t)$$

$$w \leftarrow w + \alpha_w \delta_t \nabla_w V(S_t; w)$$

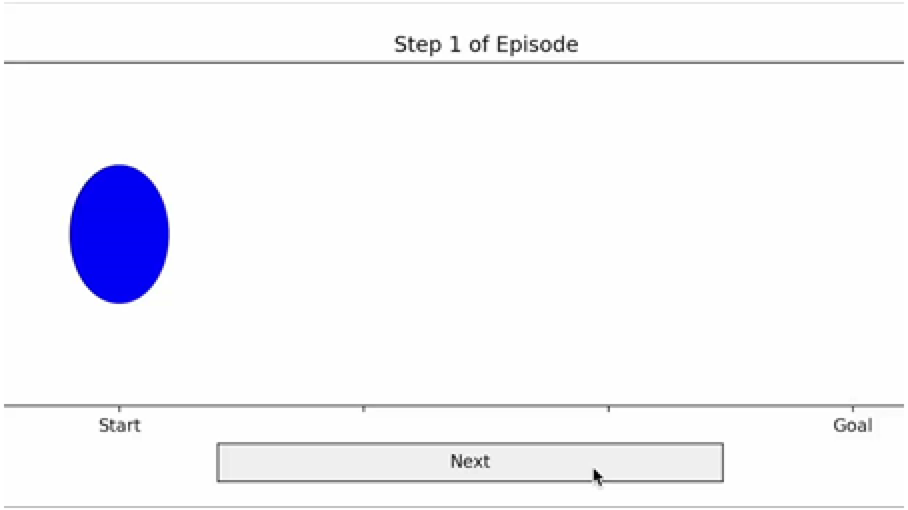


REINFORCE Demo: Example 13.1 from Barto and Sutton

Short corridor grid world with switched actions: The environment consists of a sequence of states arranged in a line (or corridor), with a start state at one end and a goal state at the other. The agent's objective is to reach the goal state by taking a series of actions (moving left or right), and it receives a reward for each action.

Environment setup:

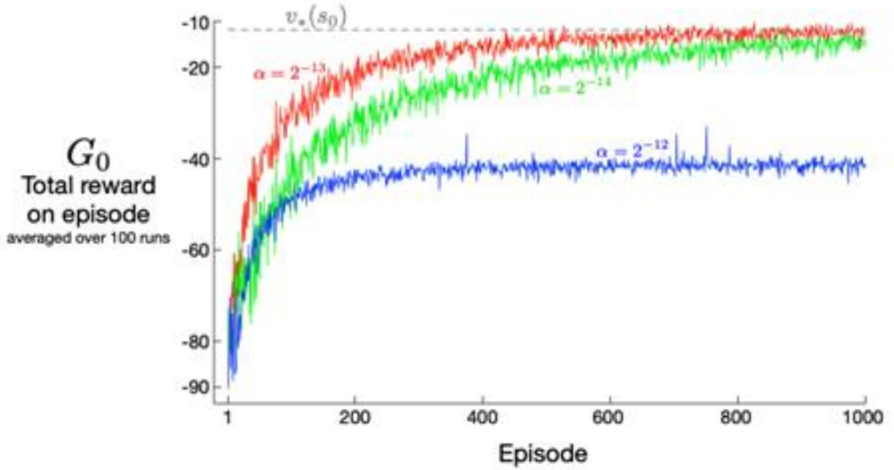
- **States:** $\{0, 1, 2, 3\}$
- **Actions:** $\{0 \text{ (right)}, 1 \text{ (left)}\}$
- **Start State:** 0
- **Goal State:** 3
- **Reward:** -1 per step
- **Discount Factor** (γ): 0.9
- **Episodes:** 1000
- **Runs:** 100



REINFORCE Demo: Variance Reduction

Setting: REINFORCE without baseline

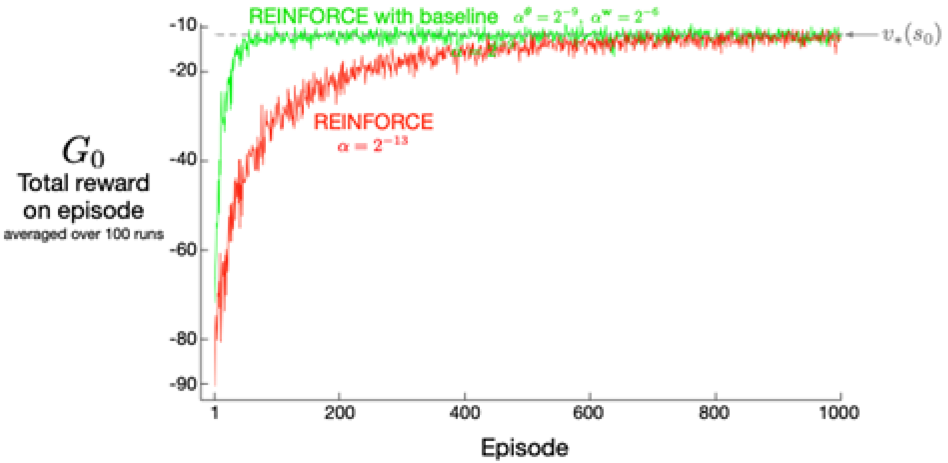
Learning Rate for Policy Parameters (α_θ): $2^{-13}, 2^{-14}, 2^{-12}$



Setting: REINFORCE with **value function** baseline

Learning Rate for Policy Parameters (α_θ): 2^{-9}

Learning Rate for Baseline Parameters (α_w): 2^{-6}



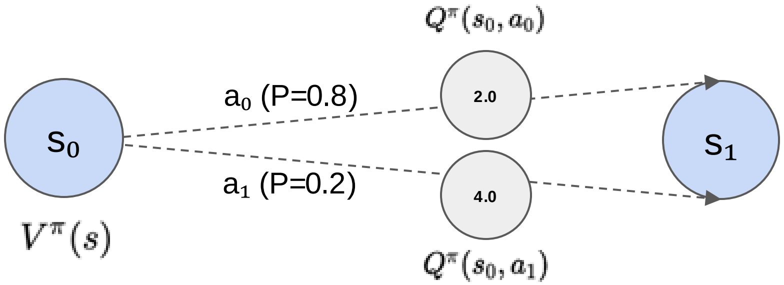
The value function baseline reduces variance -> faster convergence

- What if we use a different baseline that is not only state-dependent, but also action-dependent?

Advantage Functions

- Instead of estimating value function, estimate advantage function instead
  - **Key idea:** What is the *advantage* of an off-policy action compared to following the on-policy action given a state?
  - **Goal:** Take actions with more rewards than previous actions

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$



Assuming $Q^\pi(s_0, a_0) = 2.0$ and $Q^\pi(s_0, a_1) = 4.0$

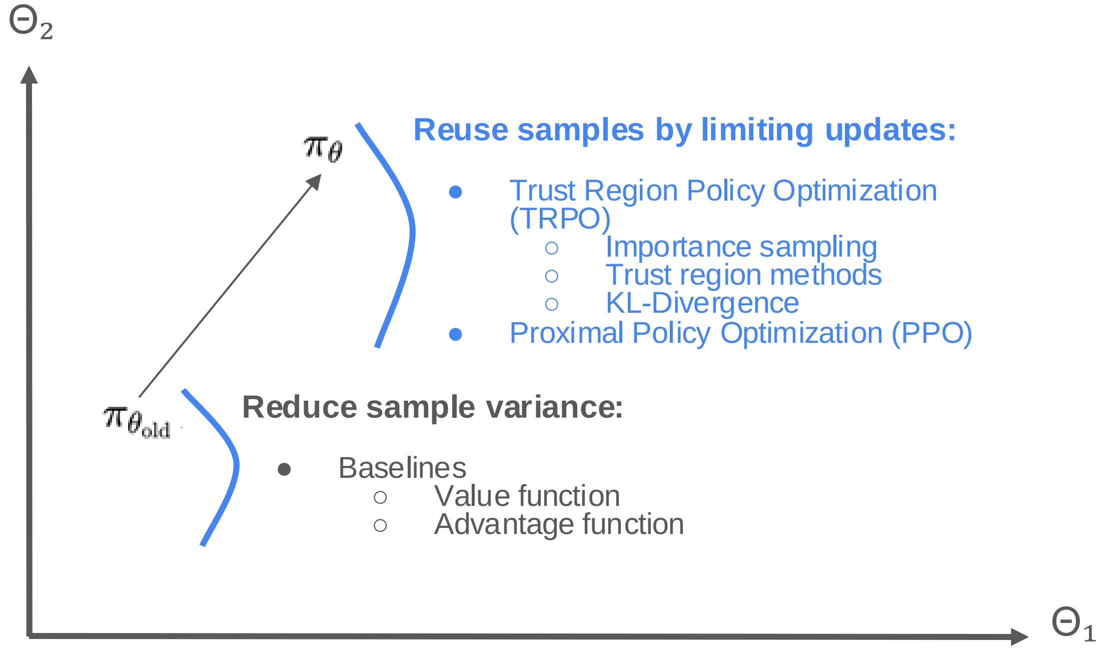
$$V^\pi(s_0) = 2 * 0.8 + 4 * 0.2 = 2.4$$

$$A^\pi(s_0, a_0) = 2.0 - 2.4 = -0.4$$

$$A^\pi(s_0, a_1) = 4.0 - 2.4 = 1.6$$

$\rightarrow$ We want to increase the likelihood of a_1

Methods Using Surrogate Objectives



Solution: Collect sample episodes using the old policy and use them to optimize a **surrogate objective** to limit the size of policy updates

- **Surrogate objective:** An approximation of the true objective function that is easier to optimize than the original objective

Importance Sampling

- **Goal:** Compute the expectation of a function f(x) with respect to a distribution p(x)

$$\mathbb{E}_p[f(x)] = \int f(x)p(x)\,dx$$

- **Issue when sampling from a distribution:** Some x_i values contribute very little to the sum because f(x) is too close to 0 in certain regions

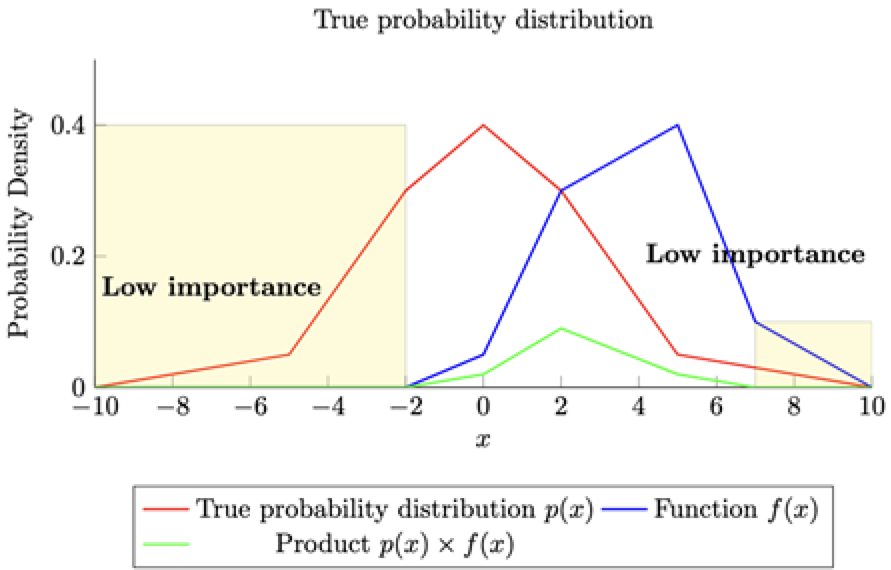
$$\hat{\mathbb{E}}_p[f(x)] = \frac{1}{N}\sum_{i=1}^{N} f(x_i)p(x_i)$$

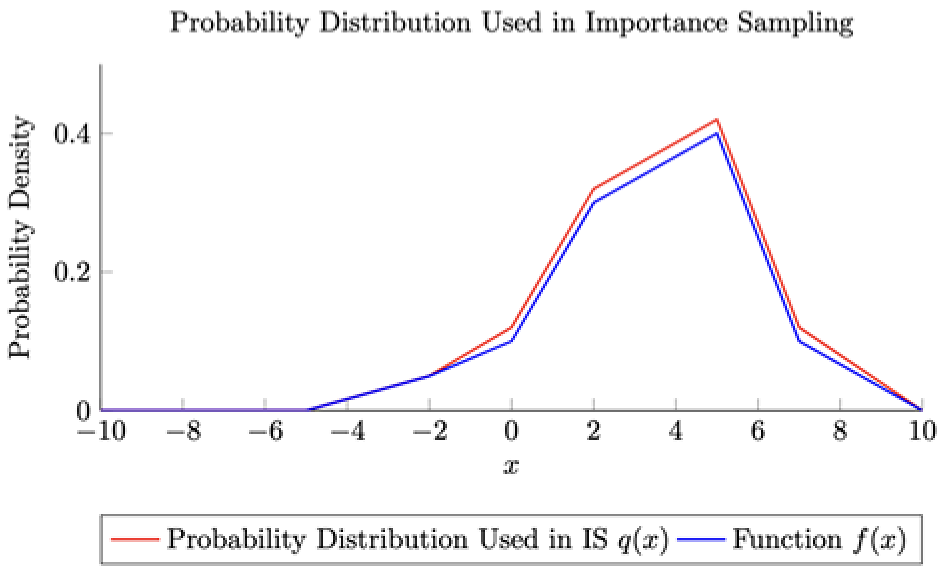
- **Solution:** Sample from a different distribution q(x), which places more weight on values of x where the function f(x) has higher value

$$\mathbb{E}_p[f(x)] = \int f(x)\frac{p(x)}{q(x)}q(x)\,dx = \mathbb{E}_q[f(x)\frac{p(x)}{q(x)}] = \int [f(x)\frac{p(x)}{q(x)}]q(x)\,dx \qquad \hat{\mathbb{E}}_p[f(x)] = \frac{1}{N}\sum_{i=1}^{N} f(x_i)\frac{p(x_i)}{q(x_i)}$$

Likelihood ratio: Ratio between old distribution and new distribution

Importance Sampling





Non-Linear Optimization

- Minimize or maximize non-linear function f
  - No direct methods in general, so iterative approximation (search) needed
- **Two classes of search techniques:** line search, trust region methods
- **Line search**: Given current point x_k, line search determines x_{k+1} by finding
  - Direction of x_{k+1}: d_k
  - Distance along that direction: α_k

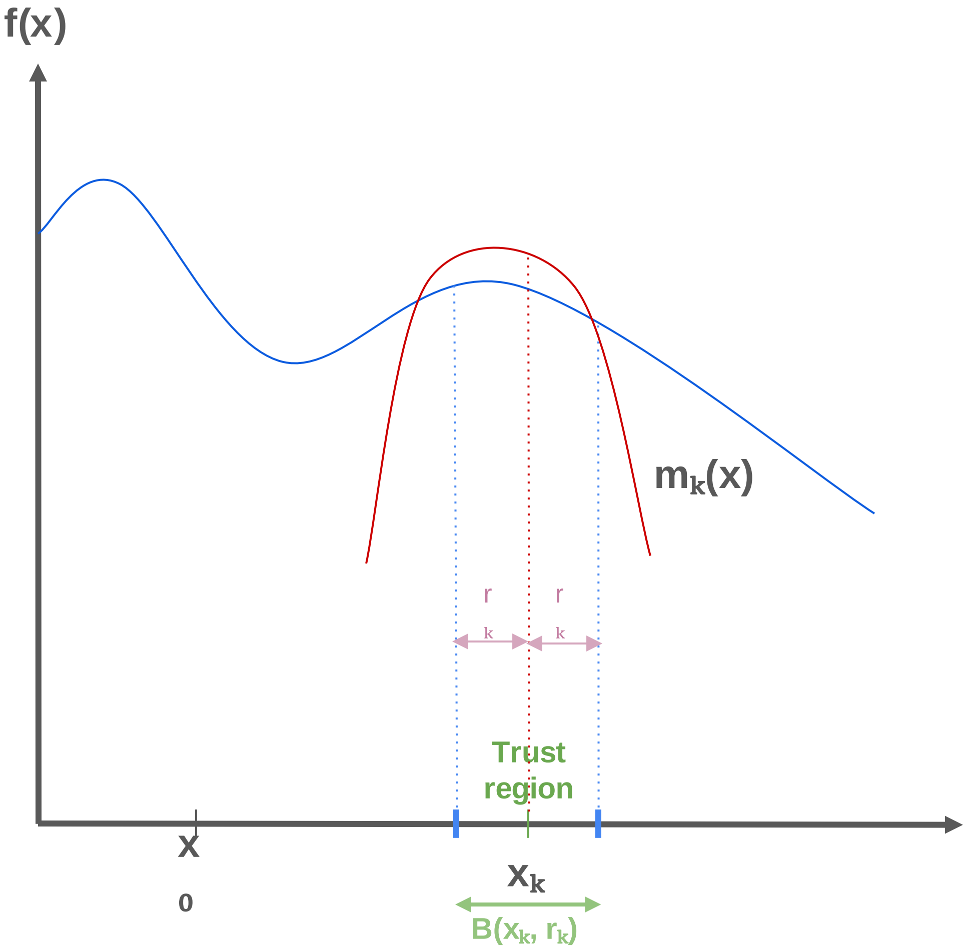
$$x_{k+1} := x_k + \alpha_k d_k.$$

- **Trust region methods:** alternative to line search
  - Optimize **surrogate (model) function $\mathbf{m_k}$** in a small region **(trust region)** around x_k to determine x_{k+1}
  - More robust than line search in practice

Trust Region Method

- **Model function $m_k(x)$:** usually linear or quadratic function obtained by Taylor series expansion of f around current point x_k
- Minimize m_k in **trust region $B(x_k, r_k)$** where r_k is radius of the trust region centered at point x_k, where m_k is good approximation for f
  - **Methods:** Cauchy point, Dogleg
- Radius r_k is chosen adaptively to speed up convergence

$$x_{k+1} := \underset{x \in B(x_k, r_k)}{\operatorname{argmin}} \; m_k(x).$$



Trust-Region Algorithm

Algorithm 1.1 Trust-Region Algorithm

```
procedure TRUST-REGION ALGORITHM
    Choose initial point x_0, initial radius r_0, and threshold η ∈ [0, 0.25).
    while ||∇f(x_k)|| > tol do
        Calculate p_k by solving the sub-problem
        Compute ρ_k.
        if ρ_k < 0.25 then
            r_{k+1} = 0.25 r_k
        else
            if ρ_k > 0.75 and ||p_k|| = r_k then
                r_{k+1} = min(2 r_k, r_max)
            else
                r_{k+1} = r_k
        if ρ_k > η then
            x_{k+1} = x_k + p_k
        else
            x_{k+1} = x_k
```

$(x_{k+1} = x_k + p_k)$

Evaluate accuracy of the model function by computing the ratio between the actual reduction and the predicted reduction, then change the radius accordingly

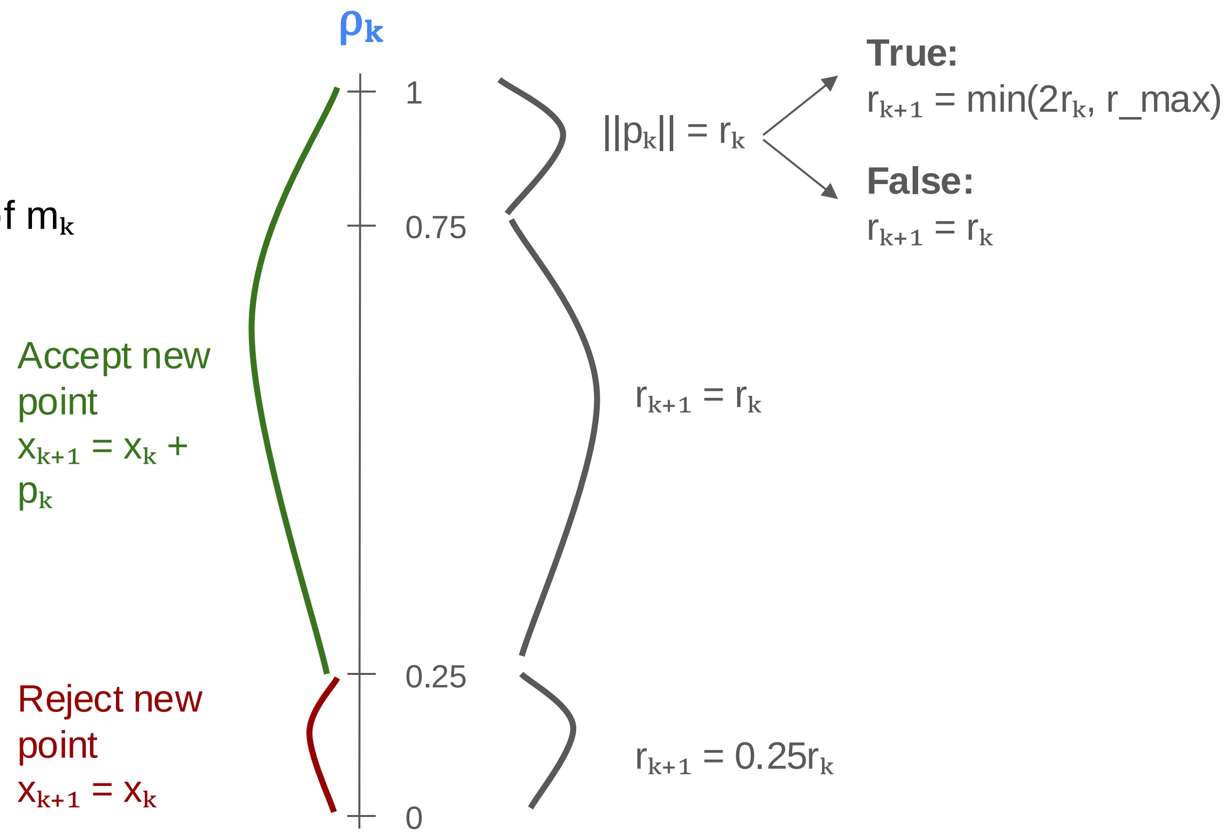
$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}.$$

Accept new point

Model did not do well enough so reject new point

Trust-Region Algorithm

x_k: current point
$x_{k+1} = x_k + p_k$ (next point)
ρ_k: measure of accuracy of m_k
r_k: radius of trust region



Trust Region Policy Optimization (TRPO)

Goal: Reuse episodes collected using the old policy for sample efficiency

- New policy should not be too different from old policy
  - Updates constrained to trust region bound by KL-divergence
- Make correction using importance sampling

Surrogate objective:

$$L^{\text{TRPO}}(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s,a)\right]$$

Likelihood (probability) ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Advantage function

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Trust Region Policy Optimization (TRPO)

Our policies are distributions so we need a measure that captures the difference between two distributions p and q: **Kullback–Leibler (KL) divergence**

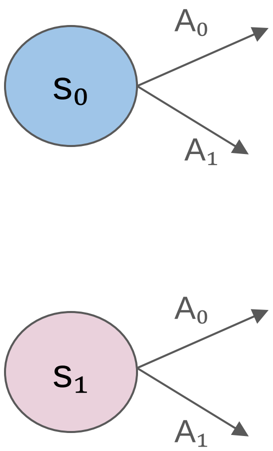
In TRPO, our surrogate objective is subject to a KL-divergence constraint to bound the trust region for policy updates:

$$\begin{aligned} \max_{\theta} \quad & L_{\theta_{\text{old}}}(\theta) \\ \text{subject to} \quad & \bar{D}_{\text{KL}}(\theta_{\text{old}}, \theta) \leq \delta \end{aligned}$$

$$\bar{D}_{\text{KL}}(\theta_{\text{old}}, \theta) = \mathbb{E}_{s \sim \rho_{\theta_{\text{old}}}} \left[D_{\text{KL}} \left(\pi_{\theta_{\text{old}}}(\cdot|s) \| \pi_{\theta}(\cdot|s) \right) \right]$$

$\rho_{\theta_{\text{old}}}$: State visitation distribution under the old policy

Trust Region Constraint Using KL-Divergence



	$\pi_{\theta_{\text{old}}}$	π_θ
s_0, A_0	0.7	0.5
s_0, A_1	0.3	0.5
s_1, A_0	0.6	0.4
s_1, A_1	0.4	0.6

$$\left[D_{\mathrm{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot|s)\|\pi_\theta(\cdot|s)\right)\right] = 1.7$$

$$\left[D_{\mathrm{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot|s)\|\pi_\theta(\cdot|s)\right)\right] = 0.4$$

s_0	1.7
s_1	0.4

Assuming s_0 was visited 4 times and s_1 was visited 8 times under the old policy:

$$\mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[D_{\mathrm{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot|s)\|\pi_\theta(\cdot|s)\right)\right]$$

= 4/12 * 1.7 + 8/12 * 0.4
= 0.833 + 0.267 = **1.100**

Proximal Policy Optimization (PPO)

Key idea: Modify objective to reduce the constrained optimization problem in TRPO to an unconstrained optimization problem

Clipped surrogate objective:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t\left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right]$$

Take the minimum of clipped and unclipped (probability ratio x advantage) values to pick the smallest update possible

Likelihood (probability) ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Advantage function

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Implicit constraint from clipping:

$$(1-\epsilon)\pi_{\theta_{\text{old}}} \leq \pi_\theta \leq (1+\epsilon)\pi_{\theta_{\text{old}}}$$

Proximal Policy Optimization (PPO)

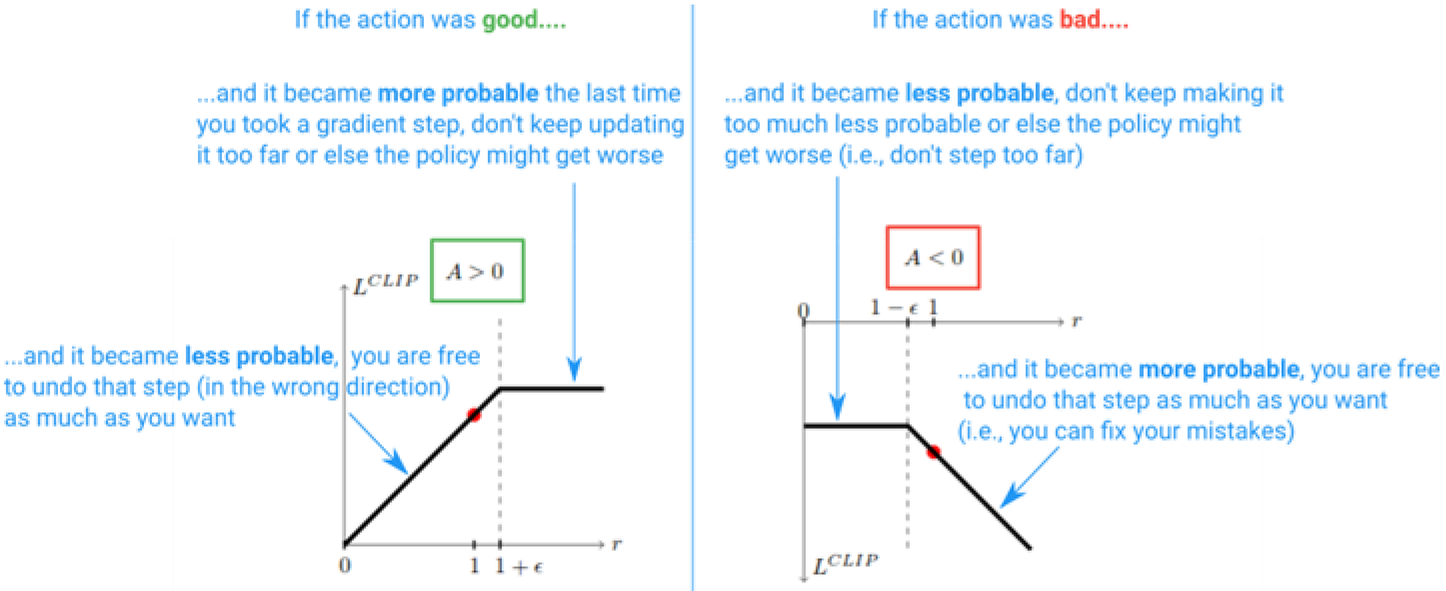
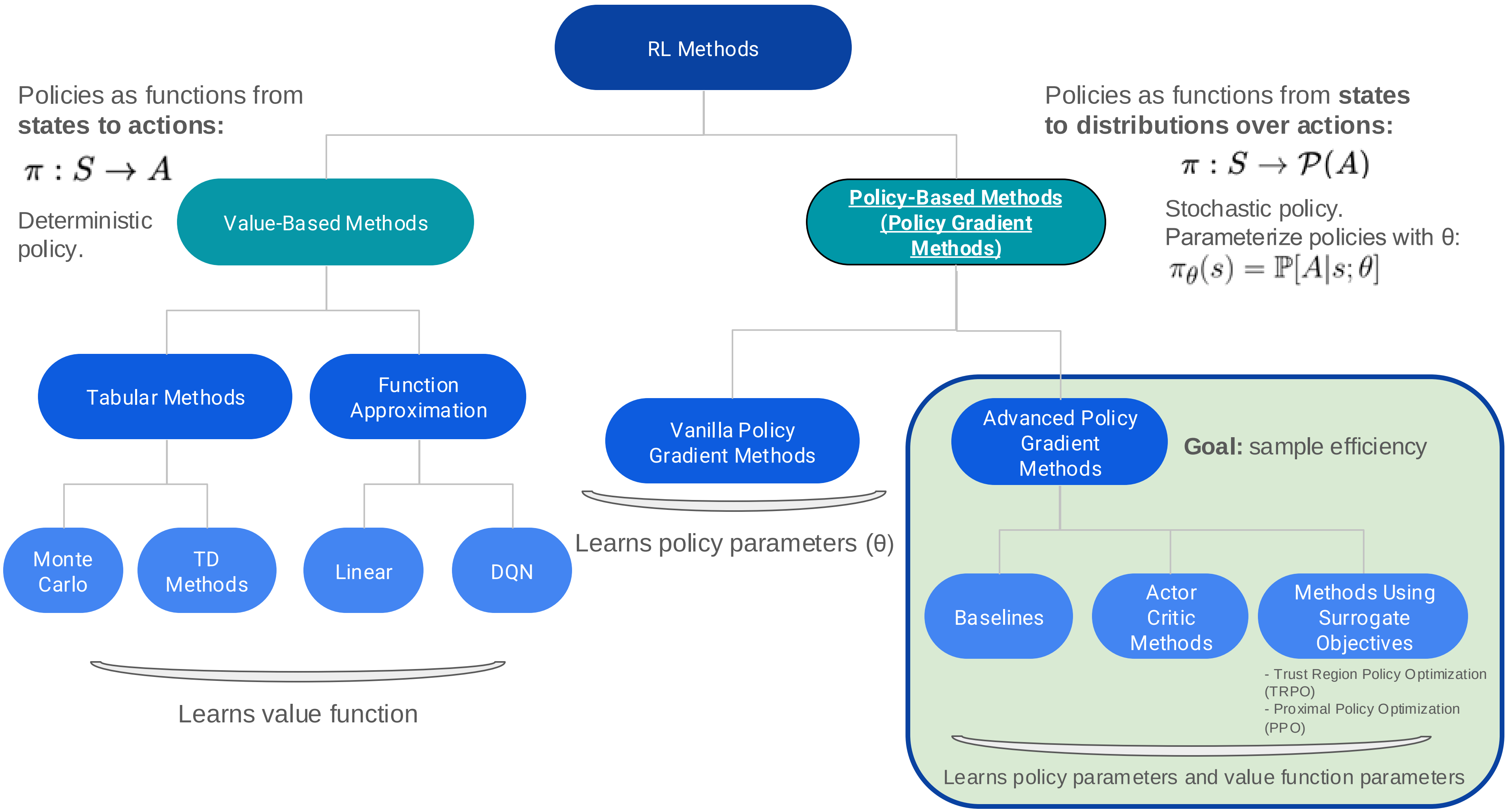


Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these terms.


RL Methods
Policies as functions from states to actions:
$\pi : S \to A$
Deterministic policy.
Value-Based Methods
Tabular Methods
Function Approximation
Monte Carlo
TD Methods
Linear
DQN
Learns value function
Policies as functions from states to distributions over actions:
$\pi : S \to \mathcal{P}(A)$
Stochastic policy.
Parameterize policies with θ:
$\pi_\theta(s) = \mathbb{P}[A|s;\theta]$
Policy-Based Methods (Policy Gradient Methods)
Vanilla Policy Gradient Methods
Learns policy parameters (θ)
Advanced Policy Gradient Methods
Goal: sample efficiency
Baselines
Actor Critic Methods
Methods Using Surrogate Objectives
- Trust Region Policy Optimization (TRPO)
- Proximal Policy Optimization (PPO)
Learns policy parameters and value function parameters

Key Takeaway

- **Sample efficiency: getting more accurate gradient estimates without collecting more samples**
    a. **For given policy:** reduce variance across samples (episodes)
    b. **Across different policies:** limit policy update to permit reuse of samples from previous policy