Deep Reinforcement Learning (Deep RL)

Reinforcement Learning

- Optimal control
  - Take a sequence of actions under uncertainty to reach a goal while optimizing some objective function

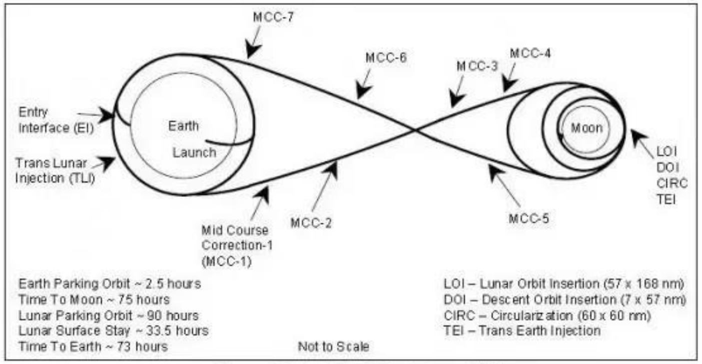


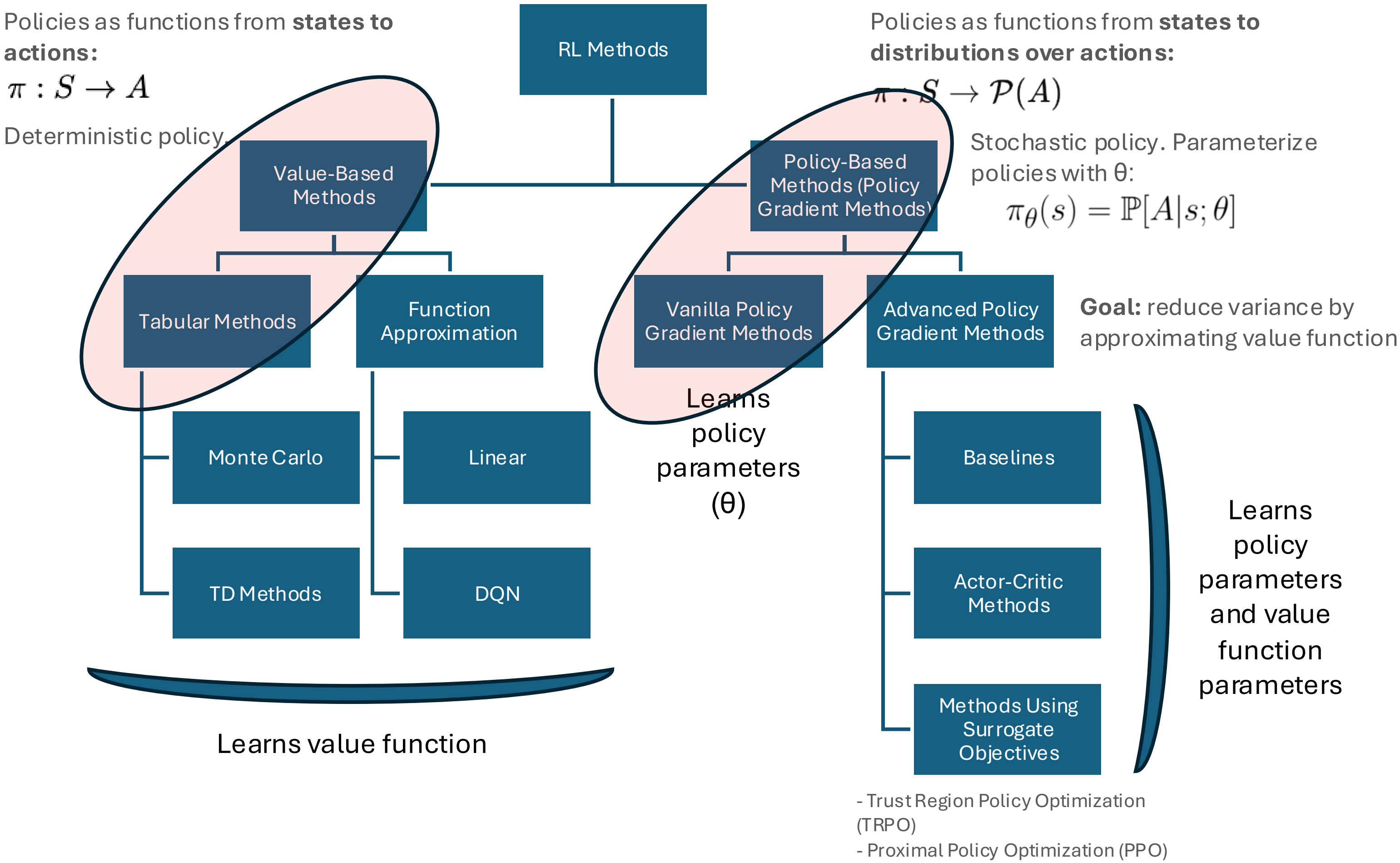
Figure 4. Nominal Apollo 13 mission profile.[12]

- Examples:
  - Moon shot:
    - > Goal: reach the moon
    - > Burn minimal amount of fuel
    - > Uncertainty: imperfect modeling of rocket and gravitational effects
    - > Solution: series of mid-course corrections till you reach moon

- Reinforcement learning
  - Optimal control **when you have little to no knowledge of the dynamics of system**
  - However, you are allowed to perform many experiments to build a working model of the dynamics

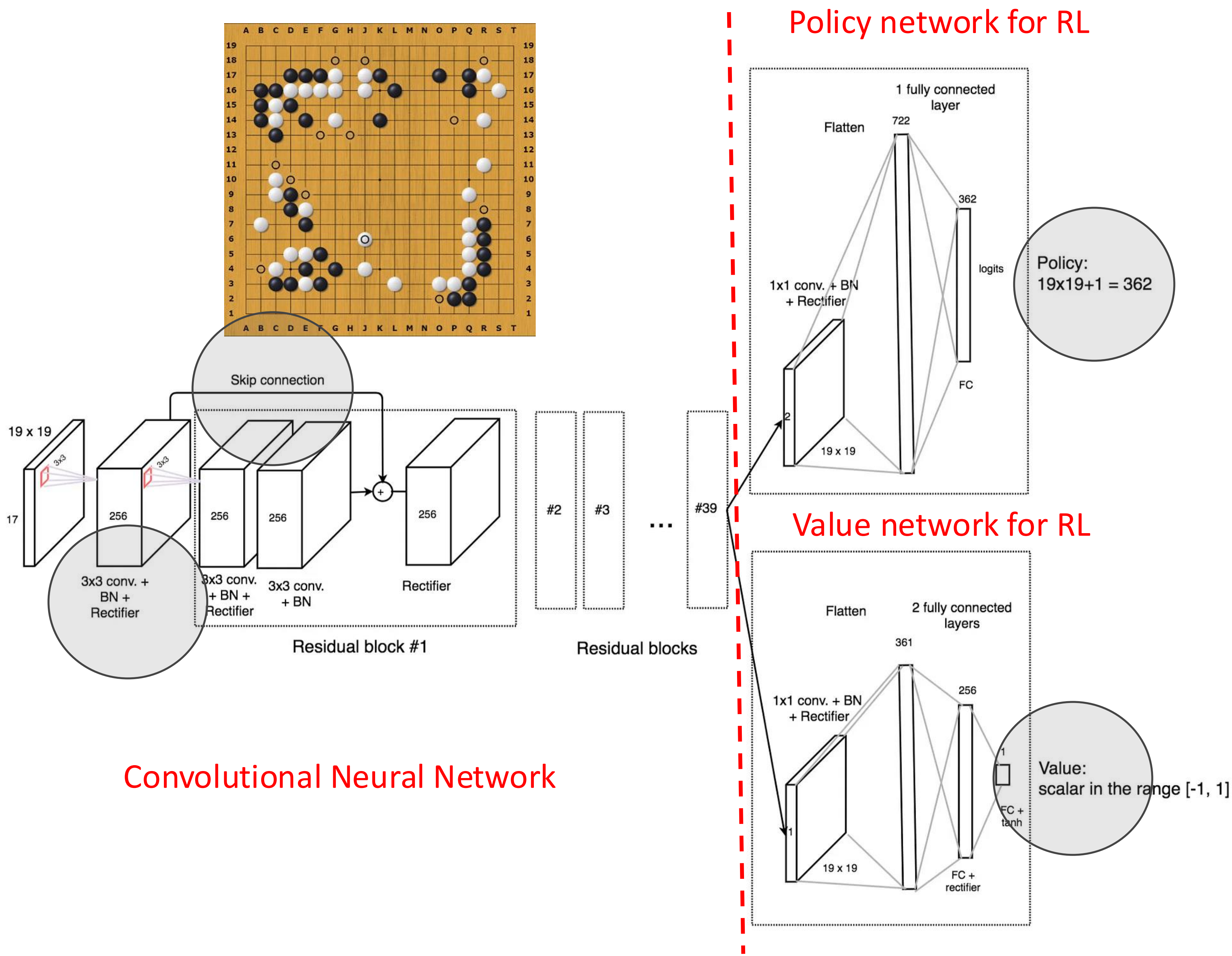
Reinforcement learning = Optimal control while learning system dynamics

RL methods



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

AlphaGo Zero (2017) uses Deep RL



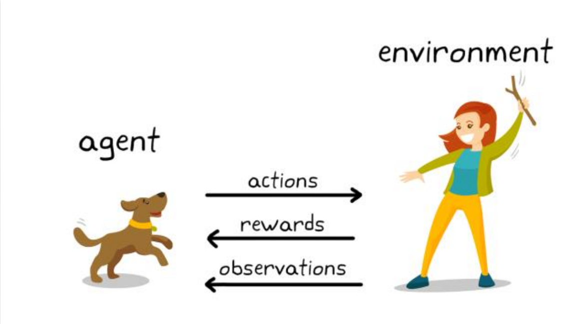
Organization

- Vanilla RL
  - Assume environment dynamics are known
  - System model: Markov Decision Process (MDP)
  - Solving optimal control problem in MDPs (*planning*)
    - **Value iteration** aka Bellman iteration (1957)
    - **Policy iteration**: Ron Howard (1960)
  - Tabular implementations
    - Represent data using state tables

- Deep RL
  - Large or continuous state spaces
    - Backgammon: 10^{20} states, Chess: 10^{44} states, Go: 10^{170} states (atoms ~10^{80})
    - Helicopter: Continuous state space
  - Use neural networks to approximate state tables

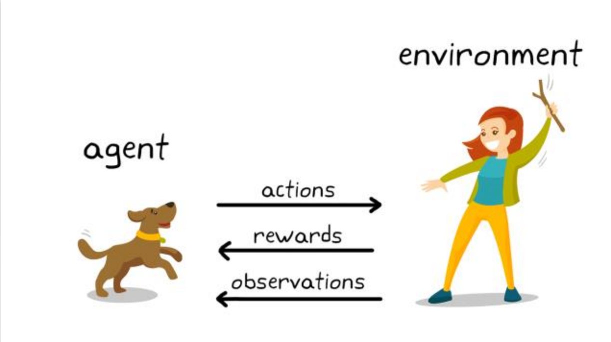
- REINFORCE: simplest deep RL method
  - Policy iteration
  - Policy network
  - Monte Carlo method to optimize policy network

Reinforcement Learning: Terminology and initial assumptions (I)



- Model: Discrete-time Markov decision process (MDP)
- Time: advances in discrete steps
- **Agent states**: finite set S (e.g.) {Sa, Sb, Sc, Sd}
  - Assumption: at each step, agent knows what state it is in
  - Partial observability: only part of the state is known (cf. Kalman filtering)
    - We will not worry about this
- **Action**: finite set A (e.g.) {Go East, Go West, Go North, Go South}
  - Result of taking action in a given state
    - **Probabilistic** transition to another state; uncertainty comes from environment (think Markov processes)
    - **Reward: may depend on new state**
  - Some actions may not be available in some states (e.g. at a wall)
- **End/terminal state**: no actions available

Terminology and initial assumptions (II)



- **Task:**
  - **Episodic:** transitions starting from some initial state until clock runs out (**finite horizon**) or you reach end state

$$Task : State \xrightarrow{Action} Reward, State \xrightarrow{Action} Reward, State.... \xrightarrow{Action} Reward, EndState$$

  - **Continuous:** sequence of transitions but no notion of end

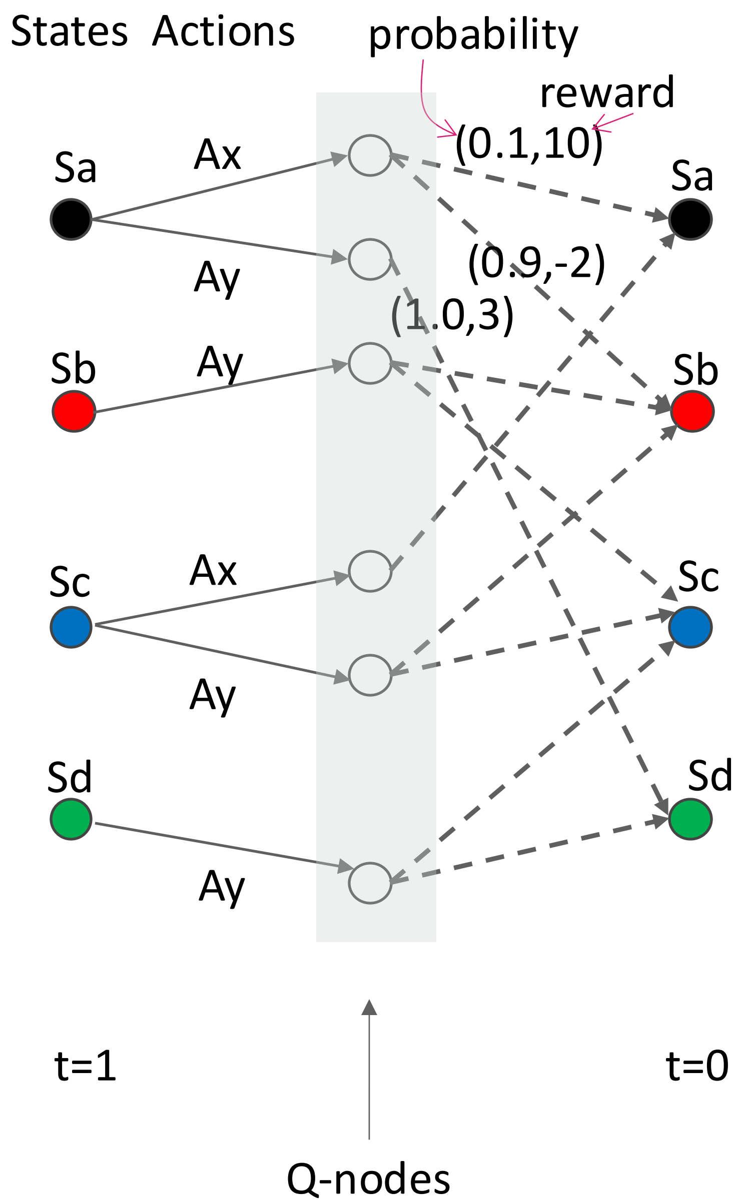
- **Policy:**
  - Action to be taken by agent at each state

Goal: Policy optimization under uncertainty

What policy maximizes expected total reward?

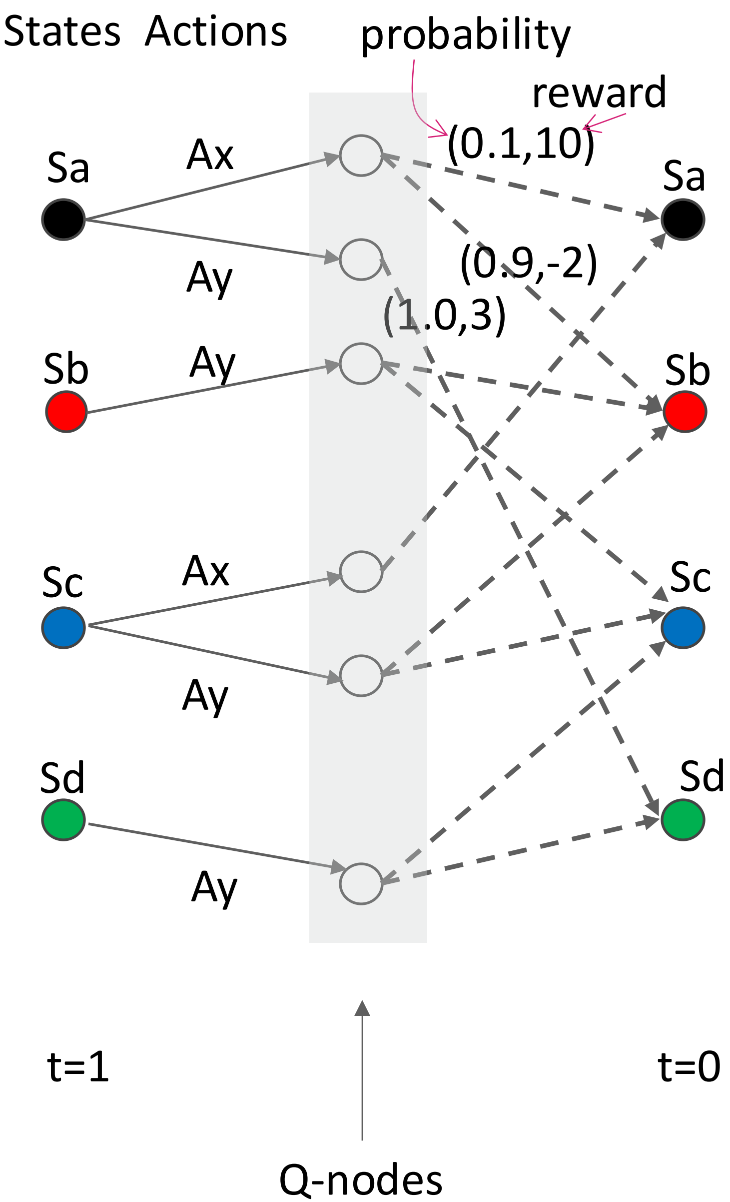
Markov Decision Processes (MDPs)

Markov Decision Process (MDP): Model for RL



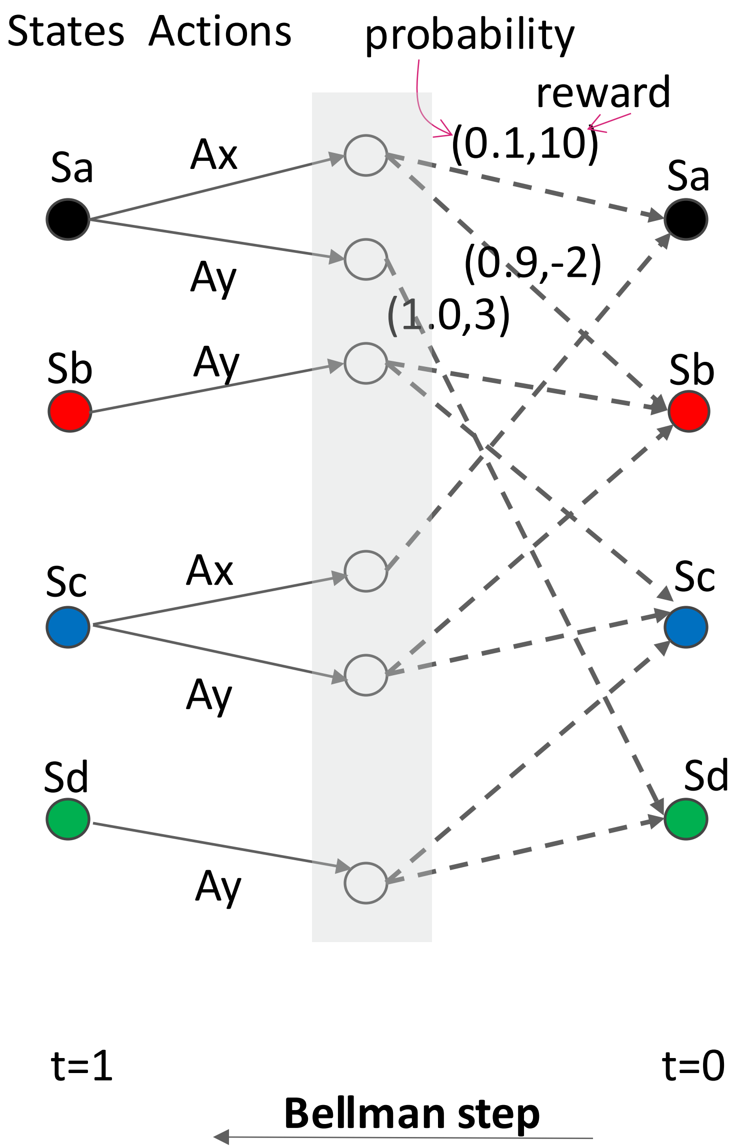
- States: S = {Sa, Sb, Sc, Sd}
- Actions: A = {Ax, Ay}
- Result of action is probabilistic
  - E.g. action Ax at Sa has two possible outcomes
    - Get reward of 10 and end up in Sa with probability 0.1
    - Get reward of -2 and end up in Sb with probability 0.9
- Horizon: 1
  - t =[1,0]
- Policy: function S → A
  - Set of policies: $|A|^{|S|}$
- Policy optimization
  - What policy maximizes total expected reward?

Policy optimization in MDPs (I): policy iteration



- Policy iteration
  - Set of policies is finite
  - Search over set of policies, performing *policy evaluation* for each policy
- Policy evaluation
  - Compute expected reward at each state at t=1
- Example: focus on Sa at t=1
  - (Sa,Ax) → Expected reward = -0.8
  - (Sa,Ay) → Expected reward = 3
  - Optimal policy at Sa:
    - Sa → Ay
  - Similar computation at all other starting states
- Valuations:
  - $V_1^{\pi}(s)$: Expected reward at state s at t = 1 under policy π
  - $V_1^{*}(s)$: Expected reward at state s at t = 1 under optimal policy
- Exhaustive search over policies not required
  - Policy improvement

Policy optimization in MDPs (II): value iteration

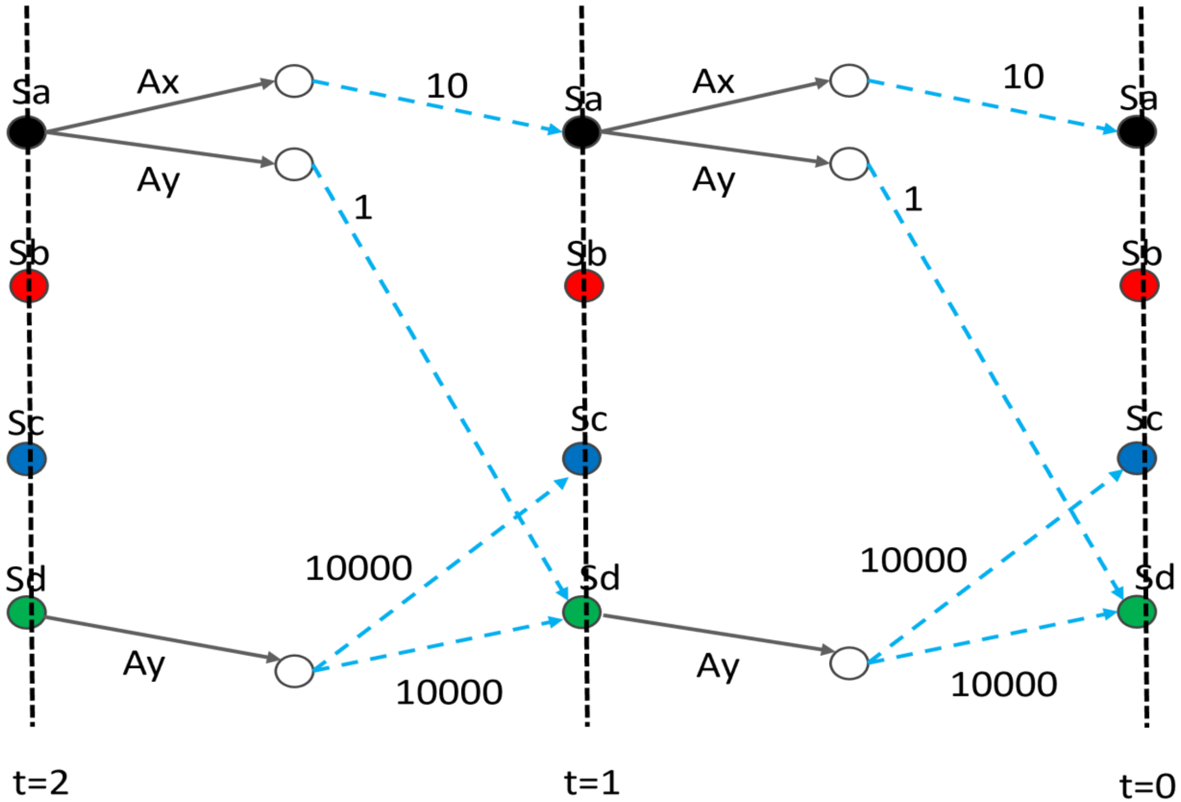


- Value iteration: expectimax
  - Backward dataflow algorithm starting at t = 0
- Initialization: $V_0(s) = 0 \ (\forall s \in S)$
- Transfer function on transition edge (Qi, p, r, Sj)

$$p * (r + V_0(Sj))$$

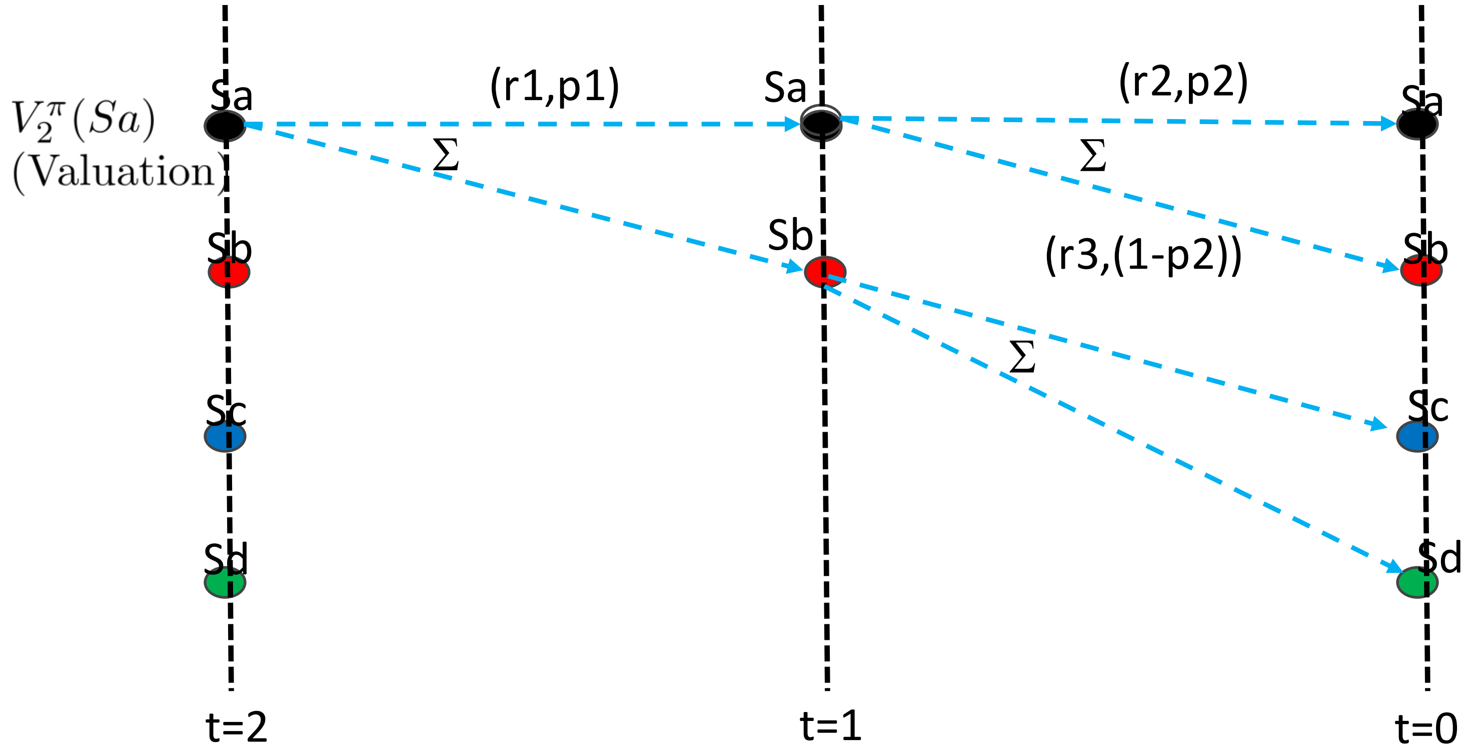
- Confluence operations
  - Q-nodes: sum incoming values
  - S-nodes: max incoming values
- Computes $V_1^*(s)$ directly

Generalizations



- Episodic tasks with time horizon T > 1
  - Policy: SxT → A
  - Obvious extension to value and policy iteration
- Continuous tasks
  - Policy: S→ A
  - Discount factor γ for delayed rewards
  - Policy iteration: search algorithm (policy improvement)
  - Value iteration: fixpoint equation (Banach fixpoint theorem)

Reformulate Policy Iteration Valuations as Expectations



- Useful to express valuations for policy π as expectations

$\tau_2^\pi(Sa)$ = set of terminating paths from Sa at t=2 consistent with π
$R(q)$ = sum of rewards on path q
Claim: $V_2^\pi(Sa) = E_{q \sim \tau_2^\pi(Sa)}\,[R(q)]$

- Intuition: express valuation as sum of products (like in gradient computation)
  - p1(r1 + p2*r2 + (1-p2)*r3) = p1p2(r1+r2) + p1(1-p2)(r1+r3)

- Easy proof by induction: bottom-up in transition diagram

Deep RL

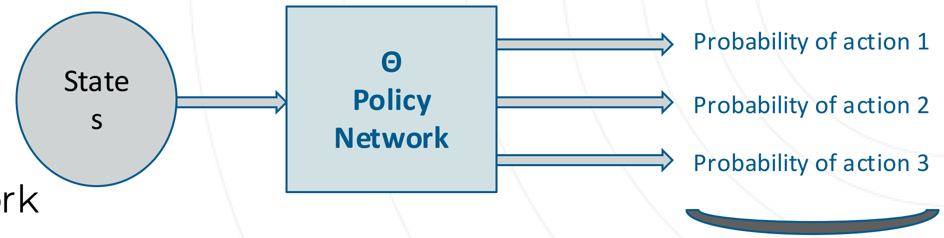
Motivation

- One implementation of π: S → A
  - Assume |S| and |A| are finite
  - Table mapping states to actions

State Action

1	A1
2	A6
3	A1
….	….
|S|	A9

- Disadvantages
  - Table becomes bigger as number of states increases
  - Cannot handle continuous state spaces
    - > Backgammon: 10^{20} states, Chess: 10^{44} states, Go: 10^{170} states
    - > Helicopter: Continuous state space



π: S $\rightarrow$ Distribution(A)

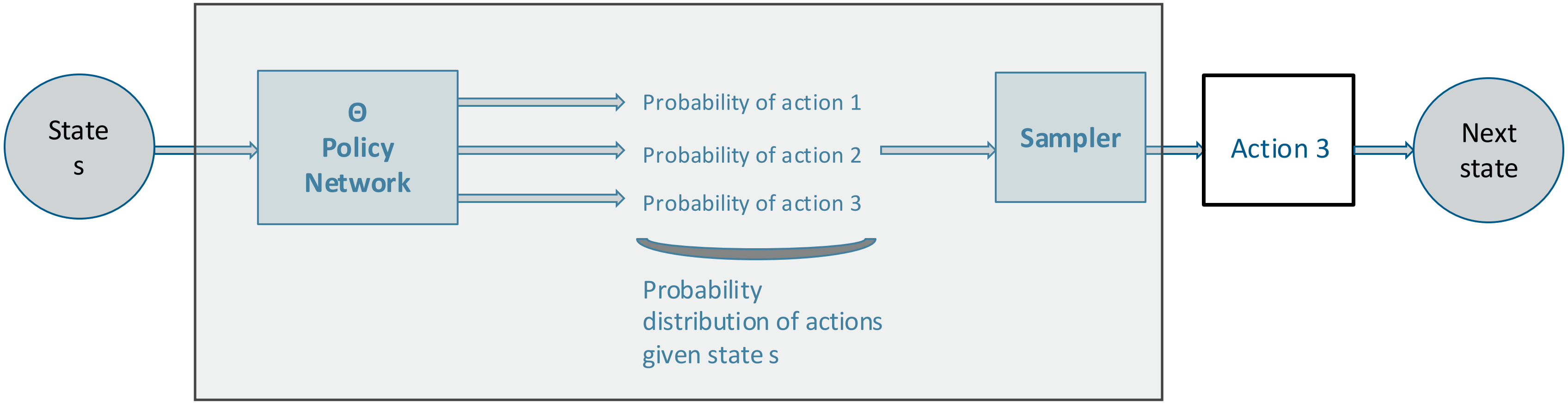
Probability distribution of actions given state s

- Policy network
  - Instead of table, implement a neural network
    - > Input is state, which can even be a real number
    - > Output is a distribution over actions
  - Intuition: think of the table as implementing a classifier

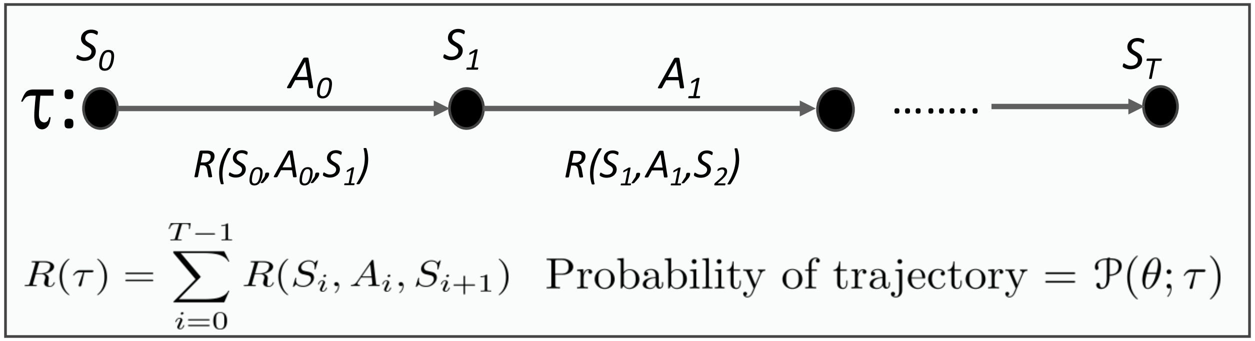
- Action space is still finite, although easy to extend to continuous action spaces

State transitions and episodes

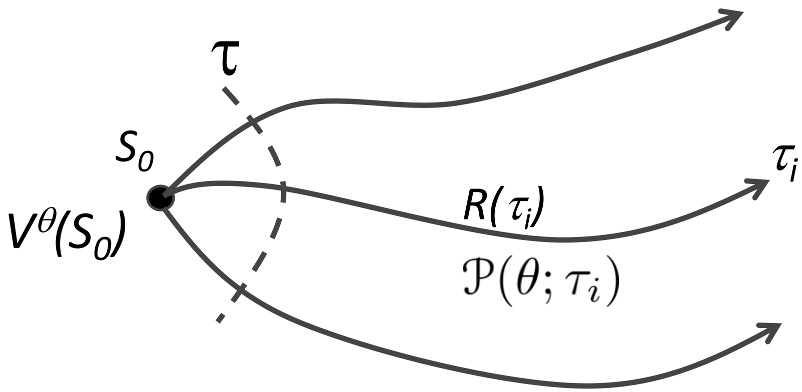
- Making state transitions
  - Policy network maps current state to action distribution
  - Sampler samples distribution to generate an action
  - Agent performs action and observes reward



- Episode/trajectory
  - Perform state transitions from start state until a terminal state is reached



Valuations



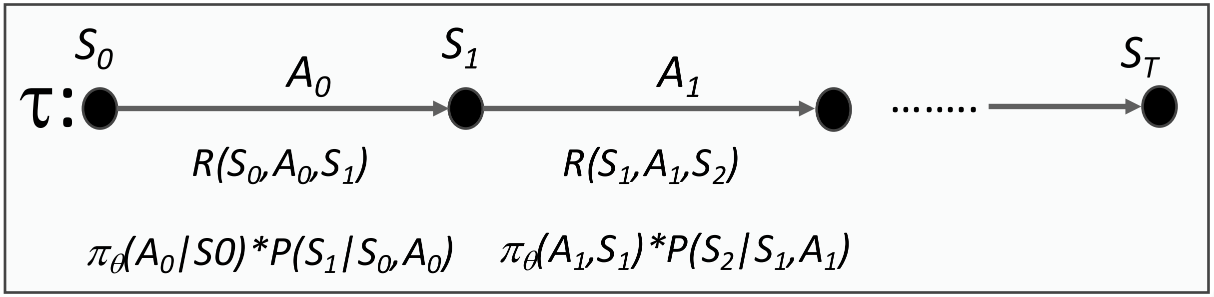
Trajectories starting at S_0

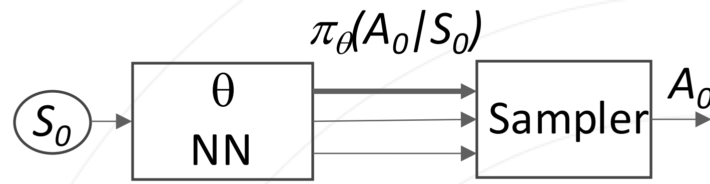
$$
\begin{aligned}
V^\theta(S_0) &= E_{\tau \sim \pi_\theta}[R(\tau)] \\
&= \sum_{\tau_i} R(\tau_i)\mathcal{P}(\theta; \tau_i)
\end{aligned}
$$

Use gradient-ascent for policy improvement

$$
\begin{aligned}
\theta &\leftarrow \theta + \alpha * \nabla_\theta V^\theta(S_0) \\
\nabla_\theta V^\theta(S_0) &= \sum_{\tau_i} R(\tau_i)\nabla_\theta \mathcal{P}(\theta; \tau_i)
\end{aligned}
$$

State transition probabilities





- Each transition probability has two factors
  - Probability that action A_i is selected at S_i = $\pi_\theta(A_i|S_i)$
    - Depends on policy network and sampler
  - Probability that taking action A_i at S_i leads to state S_{i+1}= $P(S_{i+1}|S_i,A_i)$
    - Depends on characteristics of environment as before

- Probability for sequence of transitions τ from S_0 to S_T
  - Product of probabilities of individual transitions

$$\mathcal{P}(\theta;\tau) = \prod_{i=0}^{T-1} \pi_\theta(A_i|S_i) * P(S_{i+1}|S_i,A_i) = \underbrace{(\prod_{i=0}^{T-1} \pi_\theta(A_i|S_i))}_{\pi_\theta(\tau)} * \underbrace{\prod_{i=0}^{T-1} P(S_{i+1}|S_i,A_i)}_{P(\tau)} \quad = P(\tau) * \pi_\theta(\tau)$$

Gradient computation

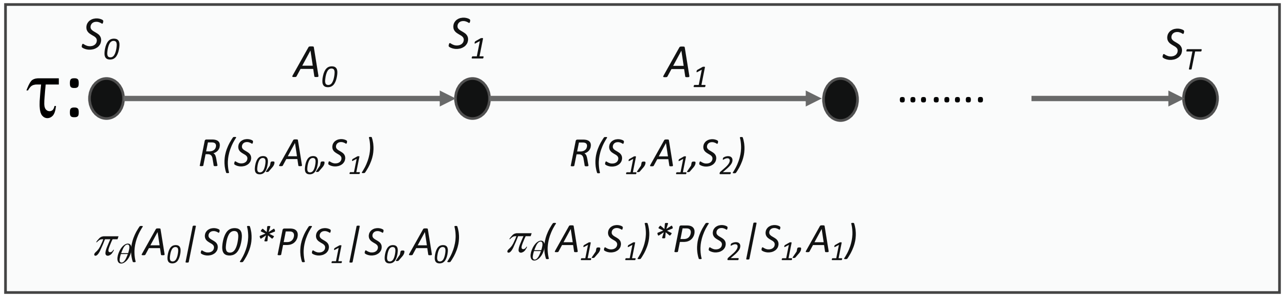
$$
\begin{aligned}
y(x) &= k * f(x) \\
\Longrightarrow \\
y'(x) &= k * f'(x) \\
&= k * f(x) * \frac{f'(x)}{f(x)} \\
&= k * f(x) * \frac{d}{dx} log(f(x))
\end{aligned}
$$

$$\mathcal{P}(\theta;\tau) = P(\tau) * \underbrace{\prod_{i=0}^{T-1} \pi_\theta(A_i|S_i)}_{\pi_\theta(\tau)}$$

$$\nabla_\theta \mathcal{P}(\theta;\tau) = P(\tau)\pi_\theta(\tau)\ \nabla_\theta(log\ \pi_\theta(\tau))$$

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

Monte Carlo Sampling



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

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

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

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. $\theta \leftarrow \theta + \alpha \nabla_\theta \widehat{V^\theta}(S_0)$

Do not need to know environment dynamics

Drawbacks of REINFORCE

- Use of MC results in high variance
  - Gradient updates may result in bad directions
- One solution: slow learning rate
  - Computationally inefficient
- Another solution: average over lots of episodes with given θ
  - Inefficient use of samples
  - Episodes must usually be discarded after θ is updated since π_θ has changed
- Solutions
  - Baseline methods: reduce variance by subtracting a baseline
    - > **Actor-critic method**
  - Limit θ updates by bounding KL-divergence
    - > **Trust Region Policy Optimization** (TRPO): update with region where KL-divergence is bounded
    - > **Proximal Policy Optimization** (PPO): clipping

Thank you