Reinforcement Learning

Understanding Machine Learning

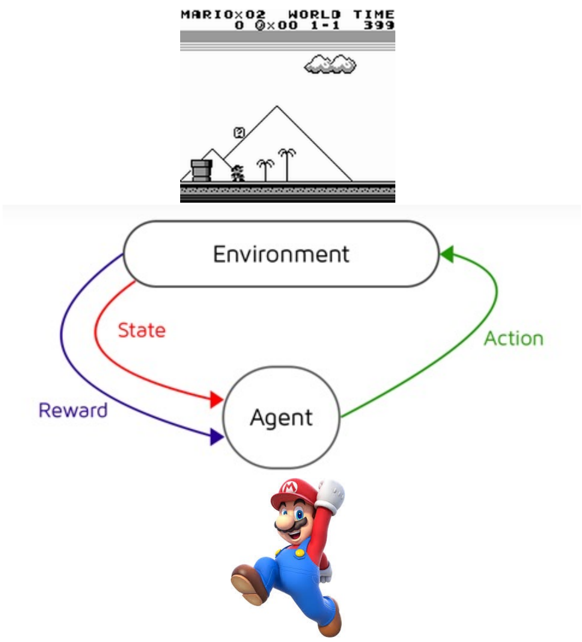
October 2022

Felix Wick

Sequential Decision Making

reinforcement learning (RL):

formalization of sequential decision making of software agent interacting with environment



Main Elements of RL

goal: find action policy maximizing reward from environment

action policy: exploration-exploitation trade-off

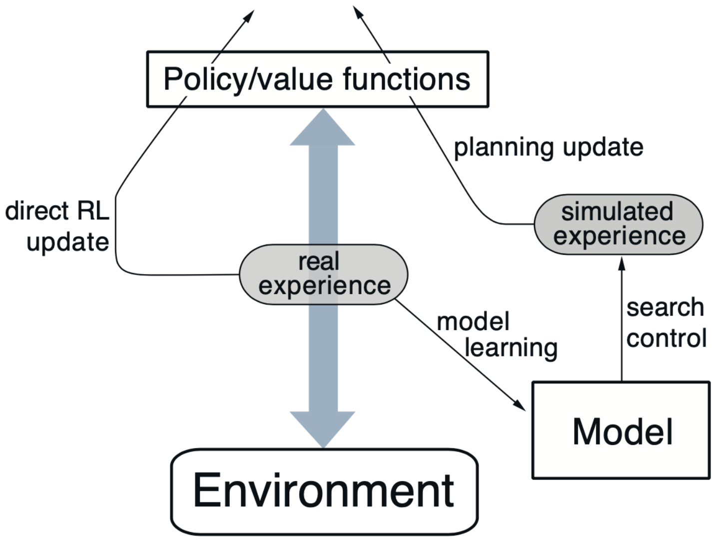
- e.g., epsilon-greedy: random exploration at small fraction of the time
- off-policy instead of on-policy learning: policy for learning different from current best → exploit in application and explore during learning

feedback from environment: goal-directed, no supervision

- scalar reward signal
- cumulative and delayed rewards (credit assignment problem)

Optional Elements of RL

model of environment: (model-free) trial-and-error or planning



from Sutton

value functions for states or actions: improve efficiency of search in vast action policy space (alternative: direct policy search)

Markov Decision Process (MDP)

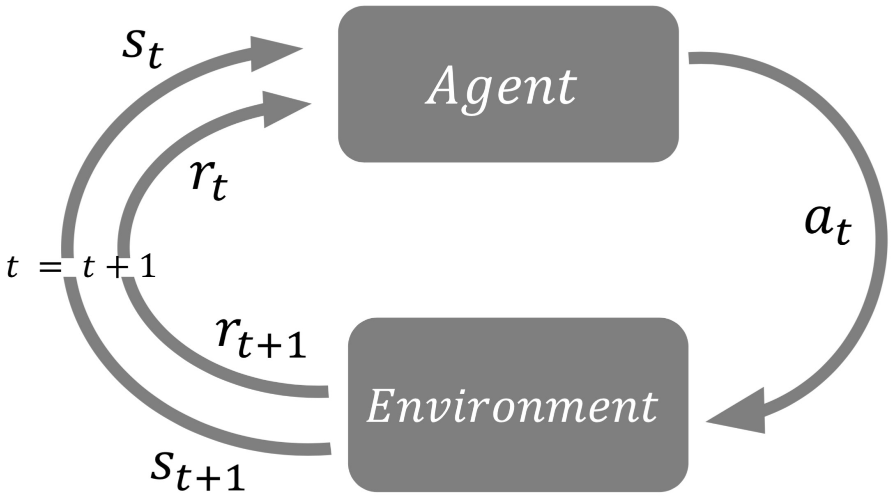
idea: current state includes all information about past

transition probabilities between states describe dynamics of given MDP

action policy: mapping from states to probabilities for selection of different actions

States, Actions, and Rewards

transition probabilities (model of environment): $p(s_{t+1}, r_{t+1} | s_t, a_t)$



reward hypothesis:

- reward as scalar signal
- goal: maximization of expected cumulative sum of received rewards

Value-Based Methods

State and Action Values

state/action value: total amount of expected future reward starting from given state/action (usually with discounting of later steps)

→ indicating long-term desirability of states/actions

main motivation: improve efficiency of search in policy space

(for comparison: evolutionary methods search directly by evaluating entire policies)

State-Value Function

discount rate

$$v_\pi(s_t) = E_\pi\left[\overbrace{\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \,|s_t}^{\text{return}}\right] = E_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})|s_t]$$

$$= \sum_{a_t} \pi(a_t|s_t) \sum_{s'_{t+1}, r_{t+1}} p(s'_{t+1}, r_{t+1}|s_t, a_t)[r_{t+1} + \gamma v_\pi(s'_{t+1})]$$

policy: probabilitiy to take specific action being in a given state

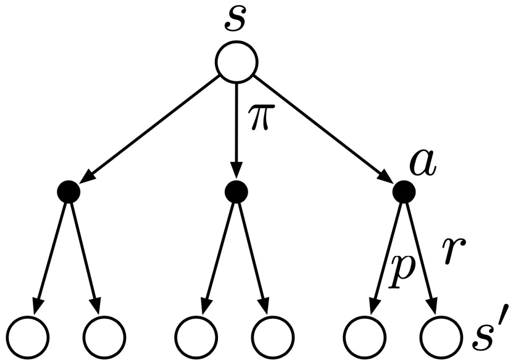
transition probability (depending on environment) from one state to another for a given action

Bellman (expectation) equation: recursion

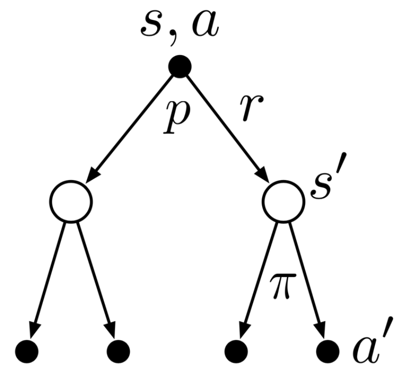
Action-Value Function

$$q_\pi(s_t, a_t) = E_\pi\left[\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \,|s_t, a_t\right] = E_\pi[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})|s_t, a_t]$$

$$= \sum_{s'_{t+1}, r_{t+1}} p(s'_{t+1}, r_{t+1}|s_t, a_t)[r_{t+1} + \gamma v_\pi(s'_{t+1})]$$



Backup diagram for v_π

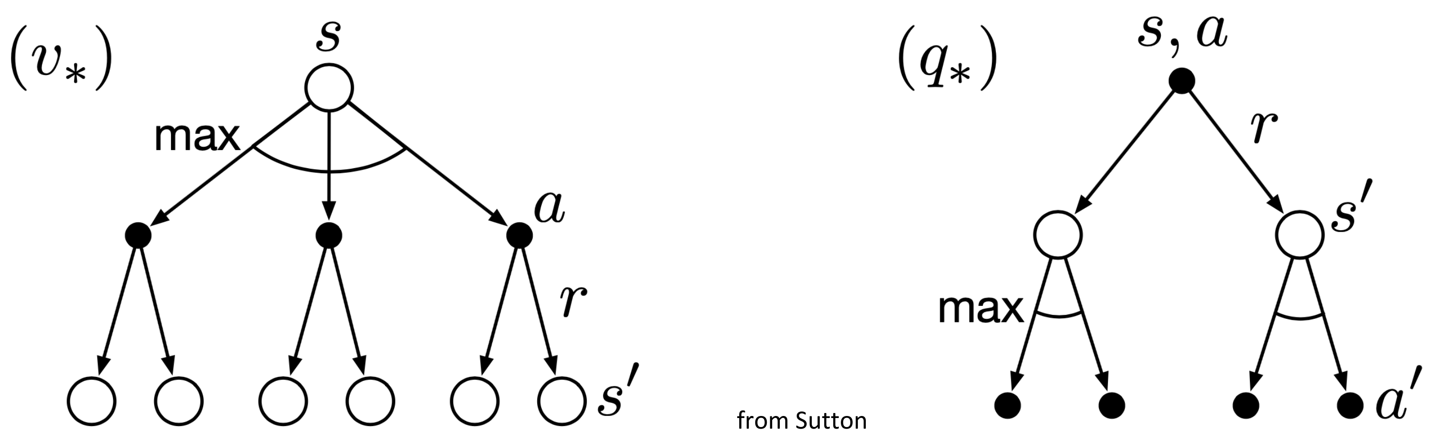


q_π backup diagram

from Sutton

Bellman Optimality Equations

optimal solutions to Bellman equations (directly defining optimal policy):



from Sutton

rarely possible to find in practice (model of environment, Markov property, computational resources)

→ approximate solutions

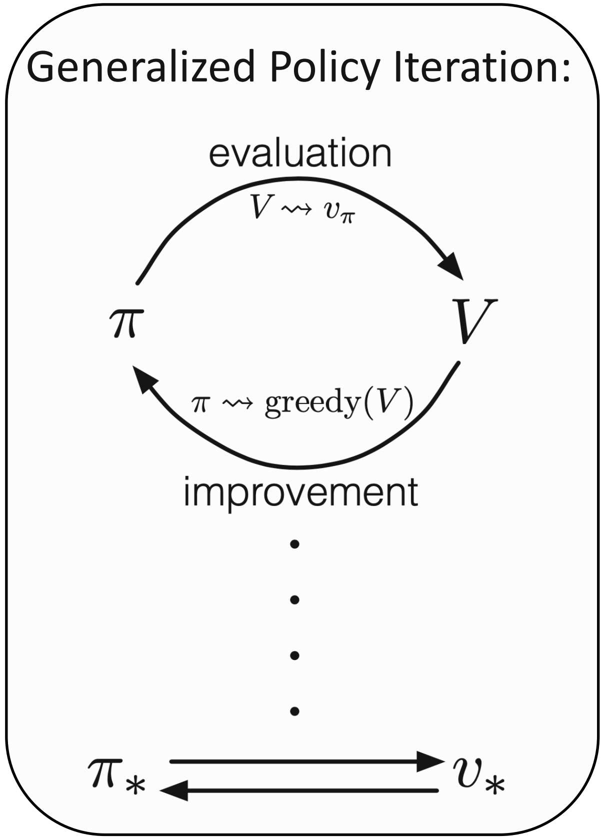
Dynamic Programming

iterative approaches to find approximations for optimal value functions

1. policy evaluation: calculate value function with current policy (Bellman equation as update rule)
2. policy improvement: adjusting policy to act greedy (pick actions with maximum values) with respect to value function of current policy

putting both components together:

- policy iteration: $\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*$
- value iteration: truncated policy evaluation using Bellman optimality equation as update rule

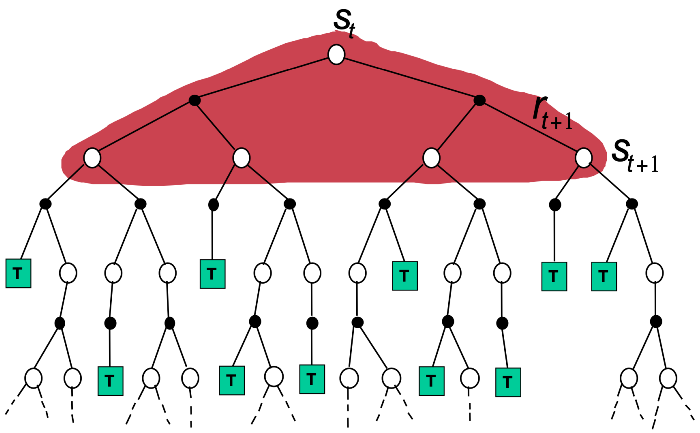


from Sutton

Bootstrapping and Sampling

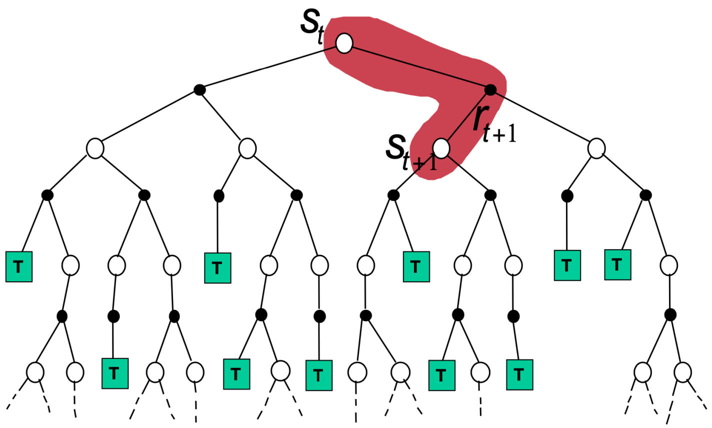
bootstrapping: update estimates of state values based on estimates of values of successor states
sampling: experience of sample sequences (no need for complete knowledge of environment)

Dynamic Programming



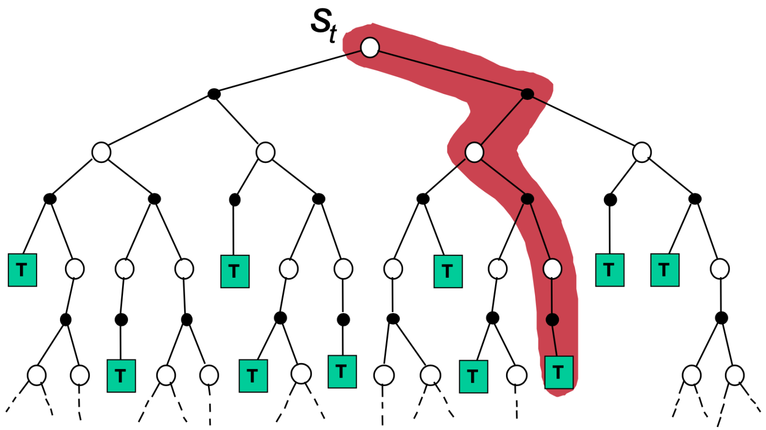
- bootstrapping
- no sampling → model-based (transition probabilities needed)

Temporal Difference (TD) Learning



- bootstrapping
- sampling → model-free

Monte Carlo (MC)



- no bootstrapping
- sampling → model-free

from Sutton

Sampling Update Rule

$$\boxed{NewEstimate \leftarrow OldEstimate \; + \; StepSize \left[Target - OldEstimate \right]}$$

MC: $v(s_t) \leftarrow v(s_t) + \alpha[\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} - v(s_t)]$

TD: $v(s_t) \leftarrow v(s_t) + \alpha[r_{t+1} + \gamma v(s_{t+1}) - v(s_t)]$

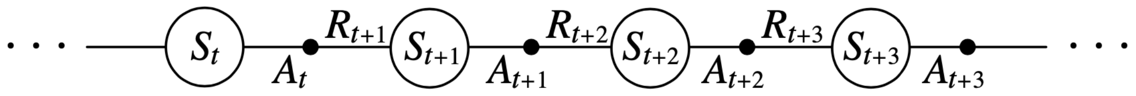
bootstrapping

On-Policy TD Control: SARSA

generalized policy iteration:

estimating action-value function Q for current behavior policy π

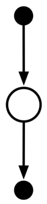
S A R S A



change policy toward greediness with respect to Q (exploration for example via epsilon-greedy policy)

update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$



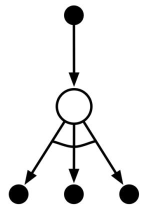
Off-Policy TD Control: Q-Learning

estimate action-value function Q directly approximating optimal action-value function (independent of policy being followed → potentially off-policy)

policy just determines which state-action pairs are visited and updated

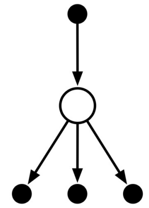
update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$



expected Sarsa:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big]$$
$$\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

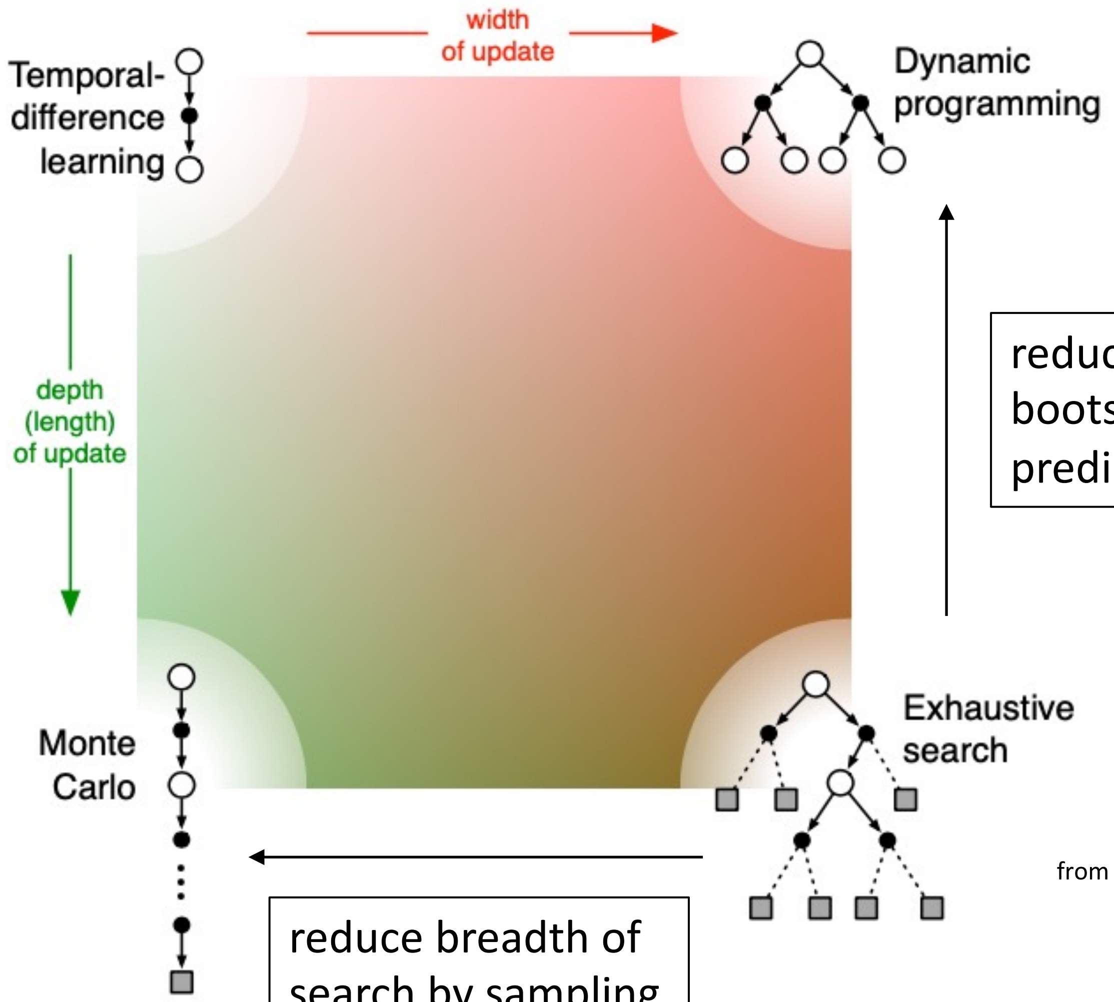


Summary: Update Characteristics

one-step TD

n-step TD

MC



reduce depth of search by bootstrapping with value function predicting future outcome

reduce breadth of search by sampling actions from policy

from Sutton

Deep Reinforcement Learning

Limitation of Tabular Methods

tabular methods simply memorize observed data

problem with tabular solution methods in practice: large state/action spaces → curse of dimensionality

need for generalization: supervised learning to the rescue

- non-linear function approximation
- nowadays often deep learning methods → deep reinforcement learning

Approximate Solution Methods

- value-function as parametrized functional form of state s with weight vector w (instead of table)
- w contains parameters for different features describing *s* (e.g., connection weights in neural network)

objective function
(mean squared value error):

$$\overline{\mathrm{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \Big[v_\pi(s) - \hat{v}(s,\mathbf{w}) \Big]^2$$

$\boldsymbol{\mu}$: state distribution (e.g., fraction of time spent in *s*)

stochastic gradient descent:

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha \nabla \Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t) \Big]^2 \\ &= \mathbf{w}_t + \alpha \Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t) \Big] \nabla \hat{v}(S_t,\mathbf{w}_t) \end{aligned}$$

with

$$\nabla f(\mathbf{w}) \doteq \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right)^\top$$

Deep Q-Network (DQN)

deep neural network to approximate Q-function (Q-value as output for any state-action pair)

Mnih et al. (Google DeepMind): *Human-level control through deep reinforcement learning*

separate target network (weights only periodically updated with Q-network weights)

→ reducing correlations of Q-network with target

experience replay: apply Q-learning updates on samples/minibatches of experience drawn at random from pool of stored samples (agent's experiences at each time-step)

→ removing correlations in observation sequence (make it i.i.d.)

Side Note: …

… i.i.d. as fundamental assumption of ML

… i.i.d. → causality

Famous Example of Deep RL: AlphaGo

Monte Carlo tree search (heuristic search algorithm) for move (action) selection

guided by deep convolutional neural networks for both value function and policy estimation

→ improving search efficiency

- **reduce depth** of search tree by evaluating positions with **value function** (predicting outcome from given position → **bootstrapping**)
- **reduce breath** of search tree by **sampling** actions using **policy network** (probability distribution over possible moves in given position)

Side Note: Model-Predictive Control

… beam-search-based planning conceptually an instance of model-predictive control

Direct Policy Search

Policy Gradient Methods

learning of parametrized policy (without value functions):

$$\pi(a|s,\boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$$

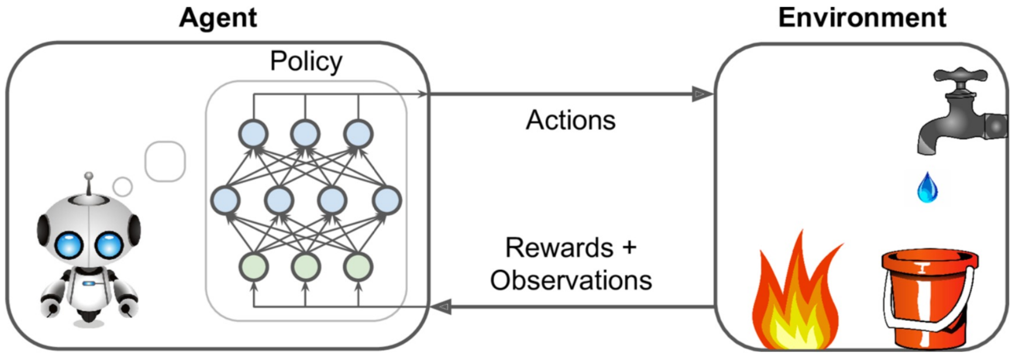
parameters: e.g., neural network weights

maximizing objective $J(\boldsymbol{\theta})$ (expected cumulative rewards)

update rule of REINFORCE method:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \cdot \underbrace{\nabla(\log \pi(A_t|S_t, \boldsymbol{\theta})) \cdot G_t}_{\nabla J(\boldsymbol{\theta})}$$

"weighting" with return



policy gradients $\nabla\pi$:
e.g., neural network gradients

Actor-Critic Methods

hybrid between policy-based and value-based methods (to reduce variance)

value function as critic of policy (instead of return):

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \cdot \nabla(log\ \pi(A_t | S_t, \boldsymbol{\theta})) \cdot \boxed{Q(S_t, A_t)}$$

independent parametrizations for π and Q (e.g., two separate neural networks)

advantage actor-critic: $Q(S_t, A_t) \rightarrow A(S_t, A_t) = \underbrace{Q(S_t, A_t) - V(S_t)}_{\text{can be approximated by TD error}}$

Proximal Policy Optimization (PPO)

state-of-the-art policy gradient method

advantage actor-critic method with clipped surrogate objective function

- surrogate objective from trust region policy optimization → better efficiency
- clipping: limiting policy update at each training step → improved stability of actor

policy gradient

advantage function:
$A(s, a) = Q(s, a) - V(s)$

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta(a_t \mid s_t)\hat{A}_t\right]$$

objective

surrogate

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

conservative policy iteration

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

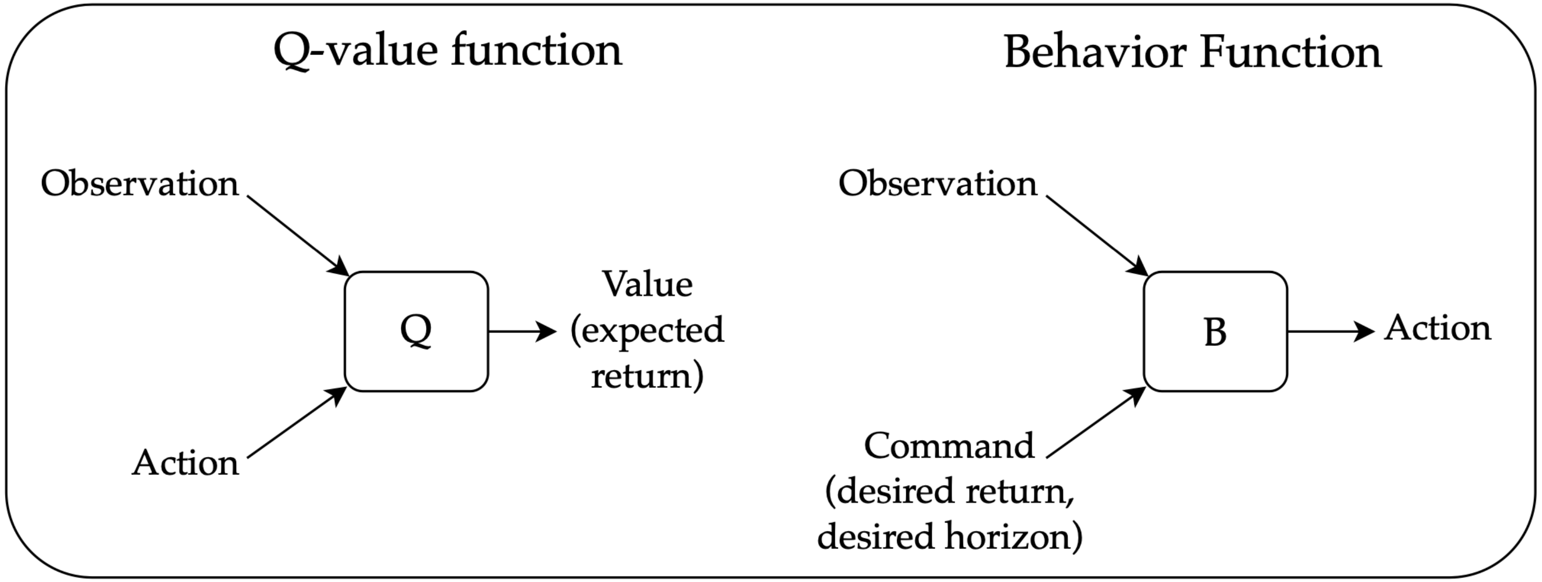
trust-region methods: first choose size of trust region, then direction
line-search methods: first choose direction, then step size

Upside-Down RL

combintion of off-policy bootstrapping (e.g., Q-learning) with high-dimensional function approximation leads to non-stationary targets (deadly triad)

most popular technique to overcome this: target networks (a copy of an agent's value function is frozen and stored periodically to provide stationary learning targets for temporal-difference learning)

upside –down RL as alternative



Generative Trajectory Modeling

transformer (sequence model) trained on fixed, limited experience consisting of trajectory rollouts of arbitrary policies (offline RL)

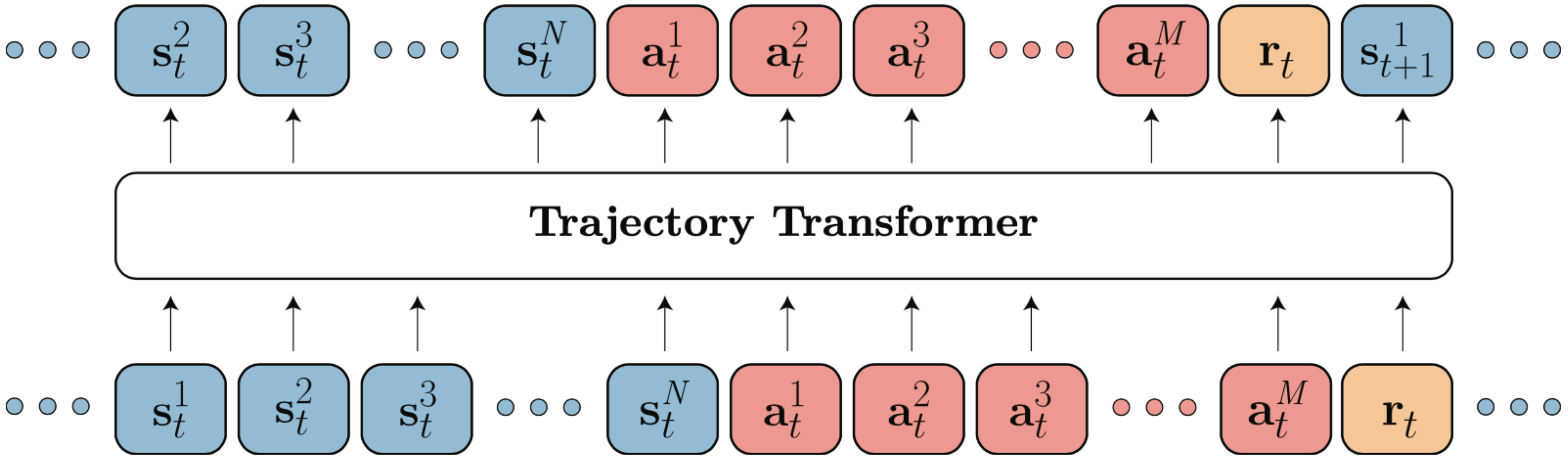
→ no need for bootstrapping

perform credit assignment directly via self-attention: implicitly forming state-return associations via similarity of query and key vectors (maximizing the dot product)

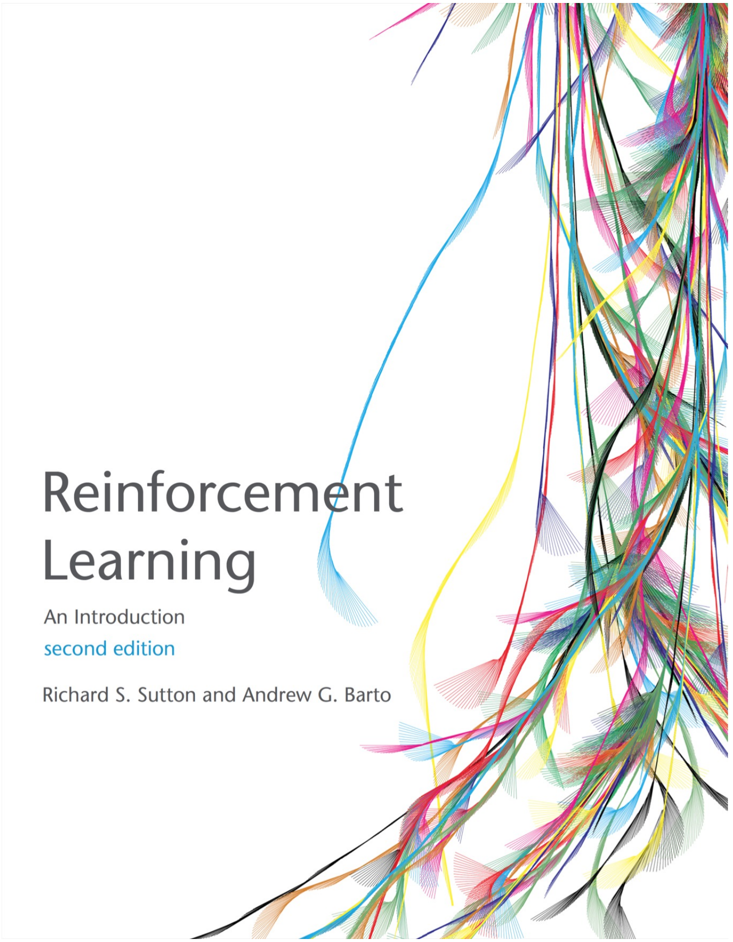
decoder architecture to autoregressively model trajectories

- Trajectory Transformer: sequence model for joint distribution of states, actions, and rewards
- Decision Transformer: conditional sequence model, conditioning on desired return (reward), past states, and actions to generate future actions

planning mirrors sampling procedure used to generate sequences from language model: selecting desired return tokens, acting as prompt for generation



Literature



papers:

- ...

Automation

...one of most impactful goals of AI

…computer vision, NLP

next step:
automated decision-making/control (e.g., autonomous driving)
…but also … [nuclear fusion plasma stabilization]

…control, robotics