

Reinforcement Learning

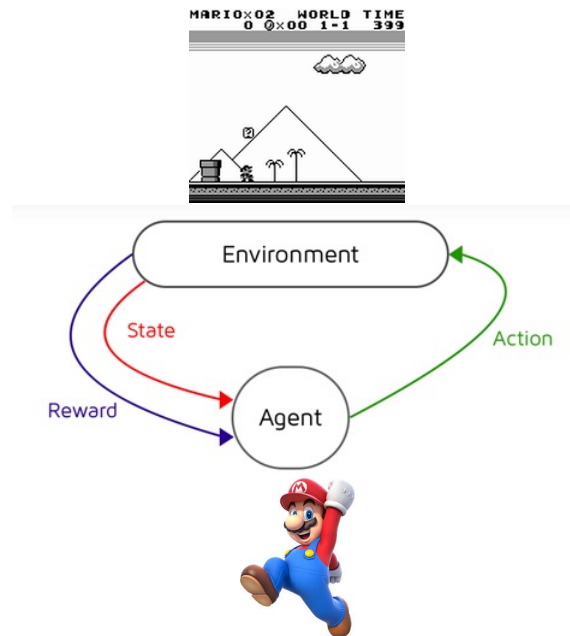
Sequential Decision Making

Understanding Machine Learning

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)

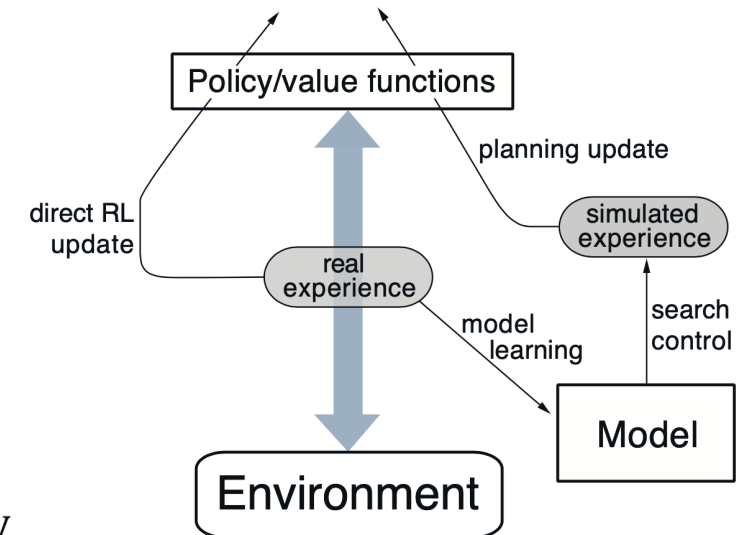
Optional Elements of RL

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

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

model of environment can be used in different ways:

- simulate experience from model (for learning)
- decision-time planning (e.g., heuristic search or model predictive control)



from Sutton

Markov Decision Process (MDP)

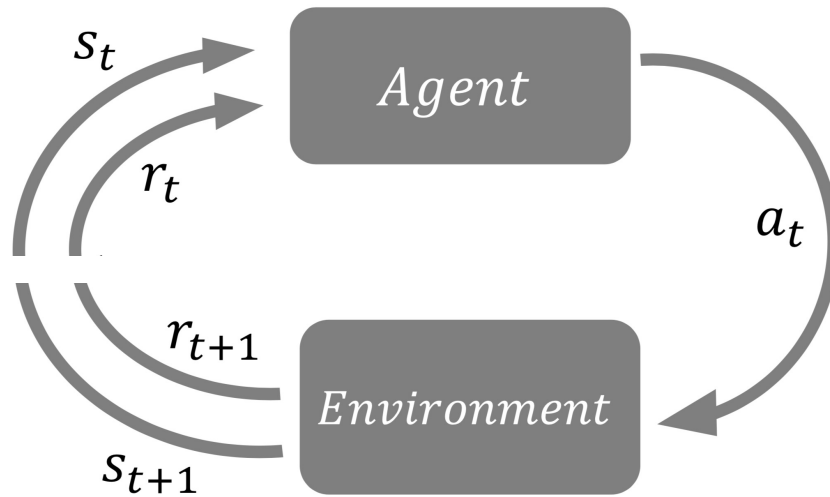
idea: current state includes all information about past

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

(needed for all states)

$$v_{\pi}(s_t) = E_{\pi} \left[\overbrace{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}}^{\text{return}} \mid s_t \right] = E_{\pi} [r_{t+1} + \underbrace{\gamma v_{\pi}(s_{t+1})}_{\text{discount rate}} \mid s_t]$$
$$= \sum_{a_t} \pi(a_t \mid s_t) \sum_{s'_{t+1}, r_{t+1}} p(s'_{t+1}, r_{t+1} \mid s_t, a_t) [r_{t+1} + \gamma v_{\pi}(s'_{t+1})]$$

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

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

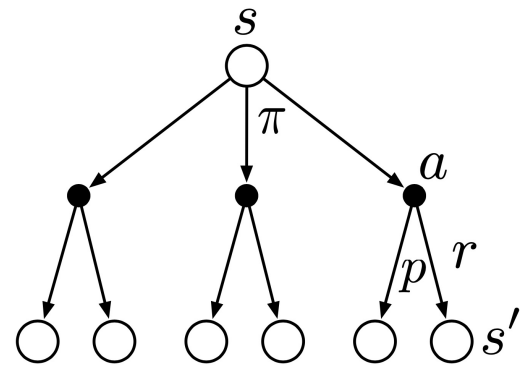
(sweep through entire state space)

Bellman (expectation) equation: recursion

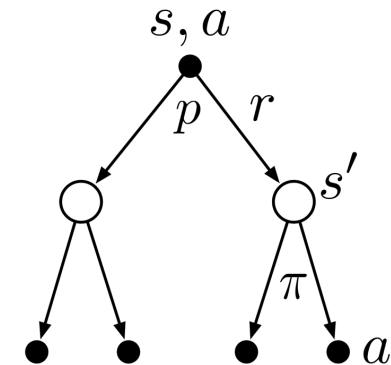
Action-Value Function

$$q_{\pi}(s_t, a_t) = E_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t, a_t \right] = E_{\pi} [r_{t+1} + \gamma q_{\pi}(s_{t+1}, a_{t+1}) \mid s_t, a_t]$$

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



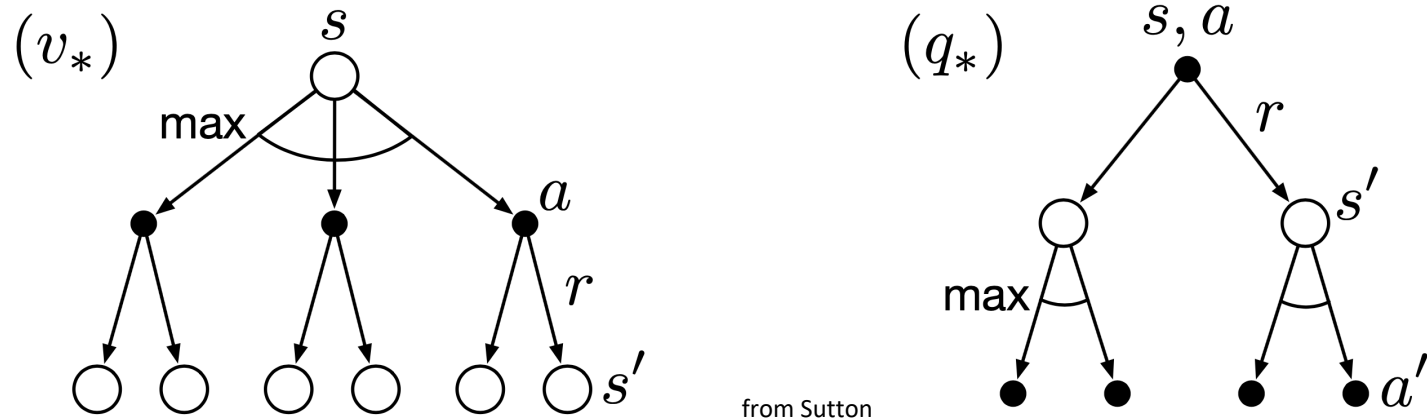
Backup diagram for v_{π}



q_{π} backup diagram

Bellman Optimality Equations

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



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

→ approximate solutions

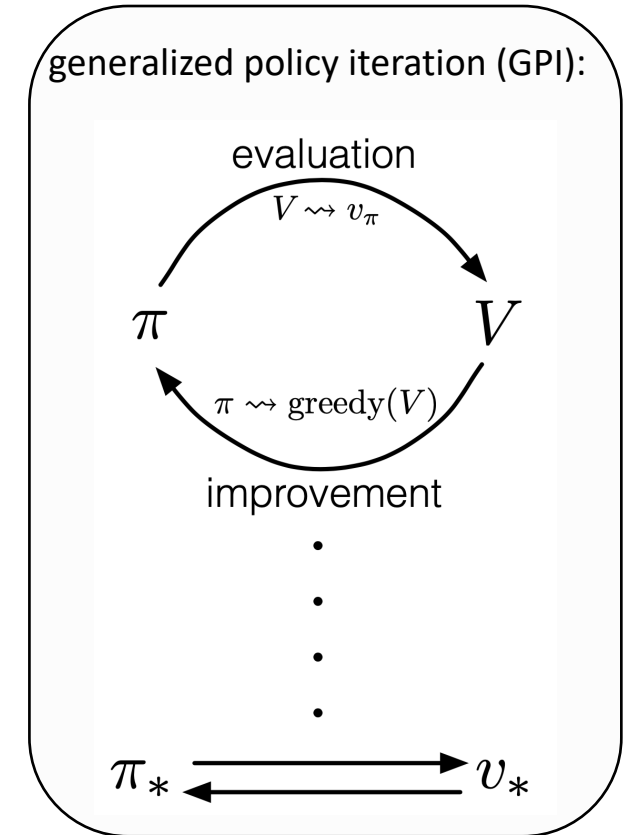
Dynamic Programming

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



from Sutton

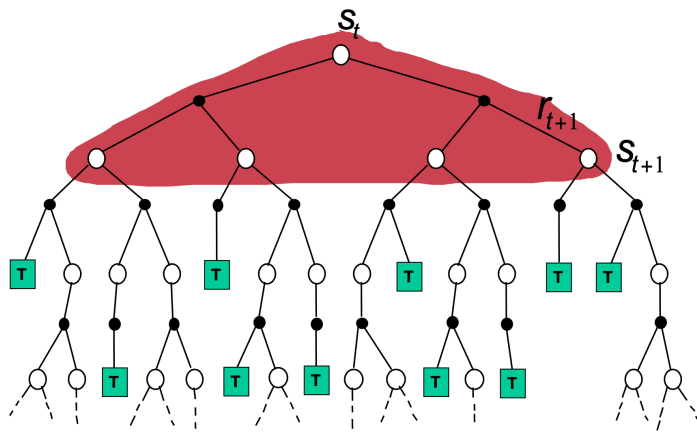
GPI also followed by MC and TD methods ...

Bootstrapping and Sampling

bootstrapping: update estimates of state values based on estimates of values of successor states

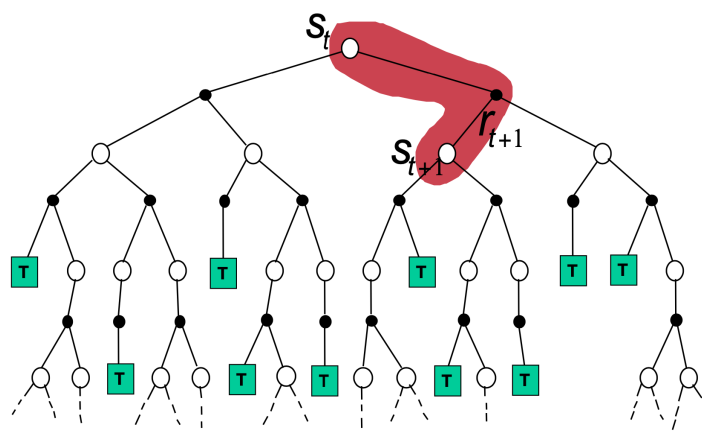
sampling: experience of sample sequences (no need for complete knowledge of environment)

Dynamic Programming



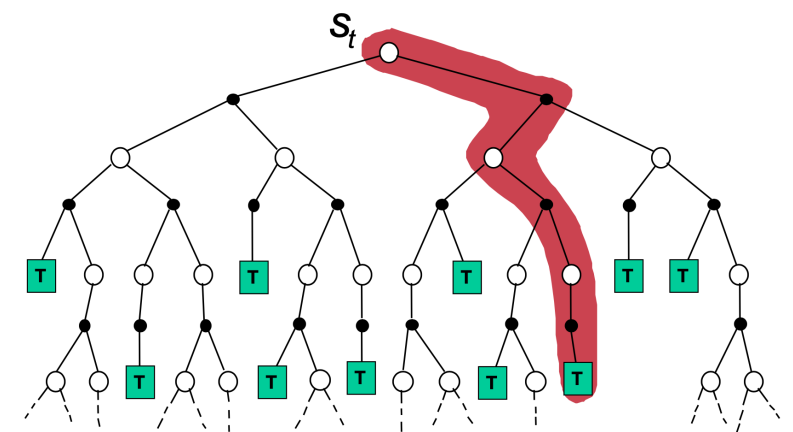
- bootstrapping
- no sampling \rightarrow model-based (transition probabilities needed)

Temporal Difference (TD) Learning



- bootstrapping
- sampling \rightarrow model-free

Monte Carlo (MC)



- no bootstrapping
- sampling \rightarrow model-free

from Sutton

Sampling Update Rule

$$NewEstimate \leftarrow OldEstimate + StepSize [Target - OldEstimate]$$

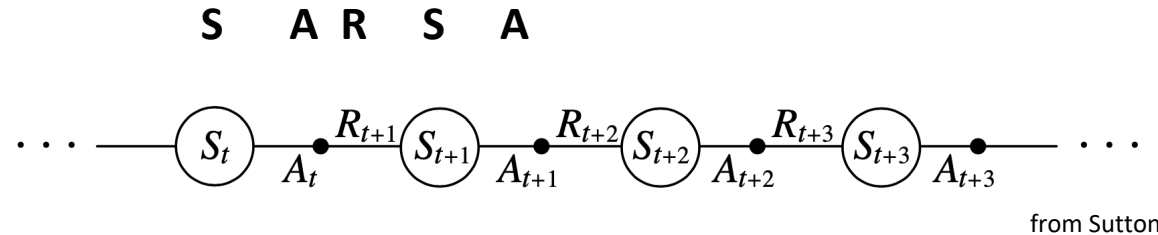
$$MC: \quad v(s_t) \leftarrow v(s_t) + \eta [\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} - v(s_t)]$$

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

bootstrapping

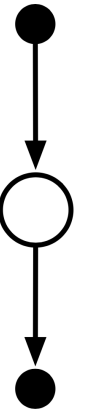


On-Policy TD Control: SARSA



following pattern of GPI:

- estimate action-value function for current behavior policy
$$q_{\pi}(s_t, a_t) \leftarrow q_{\pi}(s_t, a_t) + \eta[r_{t+1} + \gamma q_{\pi}(s_{t+1}, a_{t+1}) - q_{\pi}(s_t, a_t)]$$
- change policy toward greediness with respect to q_{π} (exploration for example via ε -greedy policy)

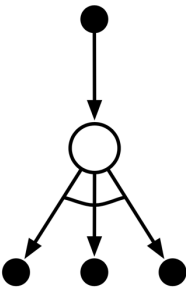


Off-Policy TD Control: Q-Learning

estimate action-value function directly approximating optimal one (independent of behavior policy → potentially off-policy)

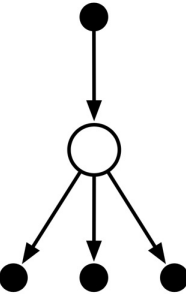
$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \eta \left[r_{t+1} + \gamma \max_a q(s_{t+1}, a_{t+1}) - q(s_t, a_t) \right]$$

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

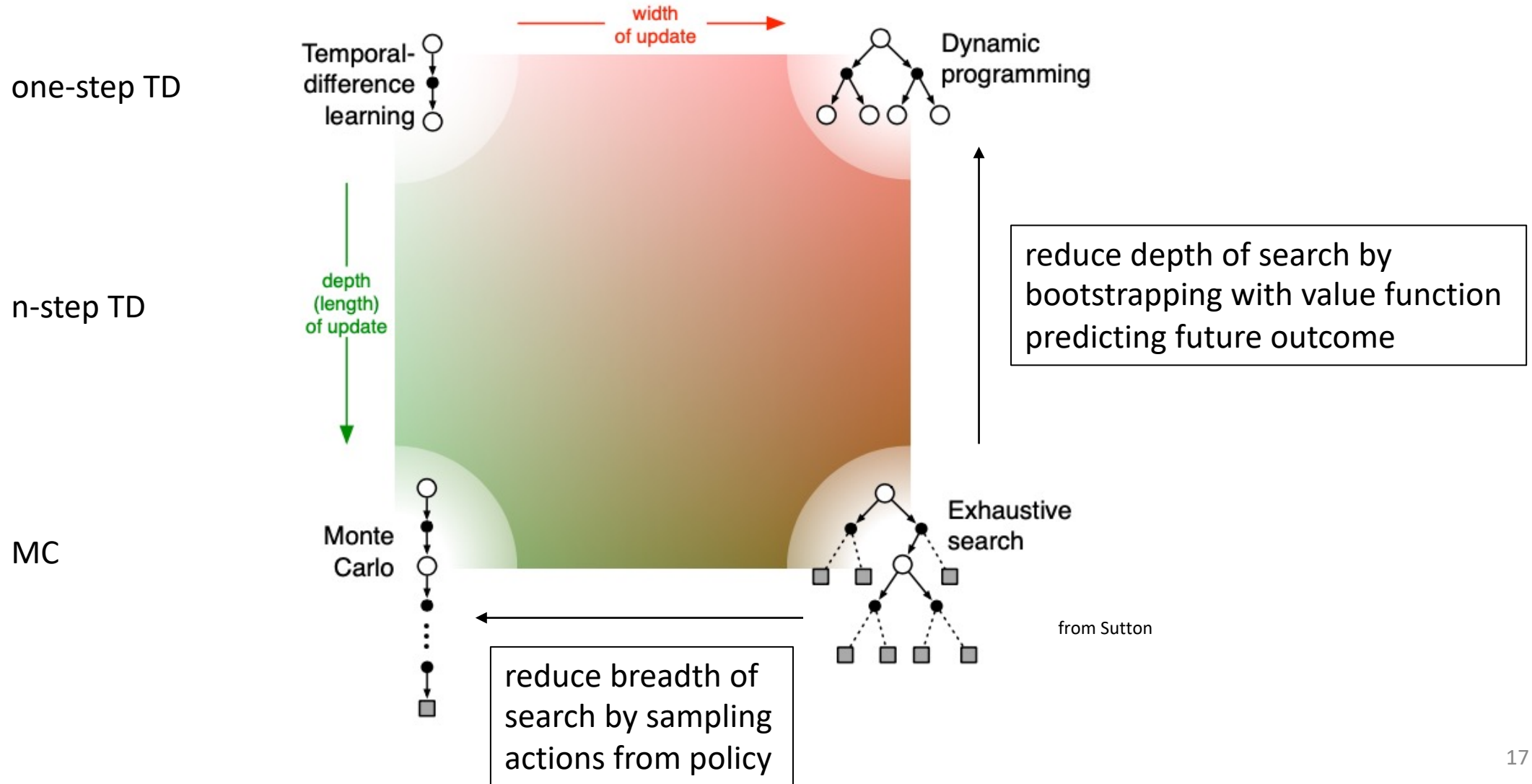


compare to expected Sarsa:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \eta \left[r_{t+1} + \gamma \sum_a \pi(a_{t+1}|s_{t+1}) q(s_{t+1}, a_{t+1}) - q(s_t, a_t) \right]$$



Summary: Update Characteristics



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

state/action values as parametrized function (instead of table)

- variables/features describing different states
- parameters (e.g., connection weights in neural network) to be learned

objective function (e.g., squared error loss):

$$J(\hat{\mathbf{w}}) = \sum_s \left(v_{\pi}(s) - \hat{v}(s; \hat{\mathbf{w}}) \right)^2$$

parameters/weights to be optimized via (stochastic) gradient descent

Deep Q-Network (DQN)

idea: deep neural network(s) approximating tabular action-value function (according to Q-learning): $q(s, a; \hat{w})$ as target of supervised learning model

key components to get it going:

- separate target network: weights only periodically updated with estimated Q-network weights → reducing correlations of Q-network with target
- experience replay: apply Q-learning updates on samples (or mini batches) of experience drawn at random from stored samples (agent's experiences) → removing correlations in observation sequence ("make it i.i.d.")

Side Note: i.i.d. Assumption in ML

assumption of independent and identically distributed sets of random variables $(Y_1, \mathbf{X}_1), (Y_2, \mathbf{X}_2), \dots, (Y_n, \mathbf{X}_n)$ fundamental to statistical (supervised) learning in terms of generalization:

consistent training and test data sets basis of empirical risk minimization
(adversarial vulnerability/attacks: targeted violations of i.i.d. assumption)

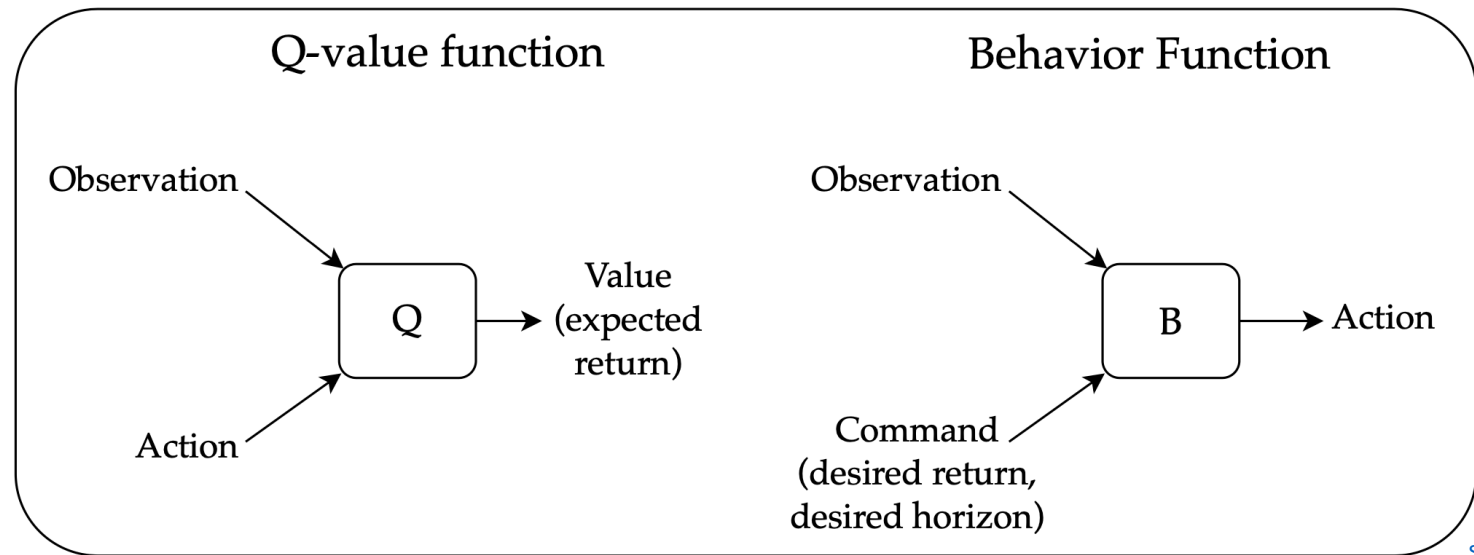
RL: MDP outside of i.i.d. setting (\rightarrow use techniques like experience replay in training of supervised learning models for value functions with observations)

causal models: interventions outside of i.i.d. setting (need for causal model)

The Deadly Triad

issue in deep RL: combination of off-policy bootstrapping (e.g., Q-learning) with high-dimensional function approximation leads to non-stationary targets (unstable)
most popular technique to overcome this: target networks in DQN

alternative: upside-down RL → no bootstrapping



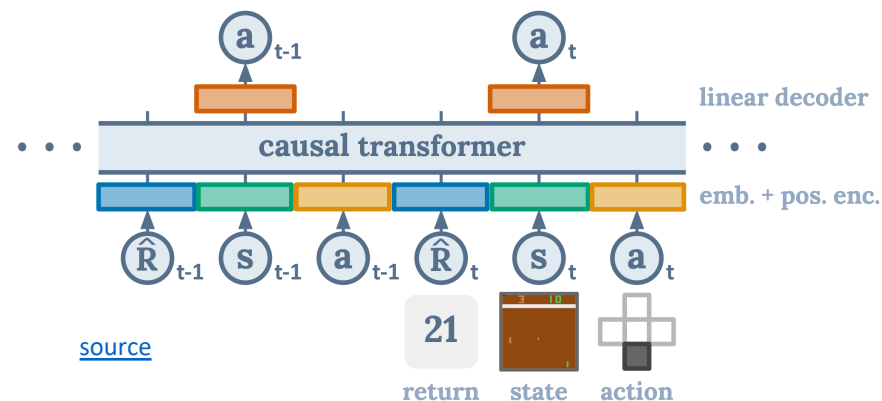
RL via Sequence Modeling

generative: transformer decoder architecture to autoregressively model trajectories

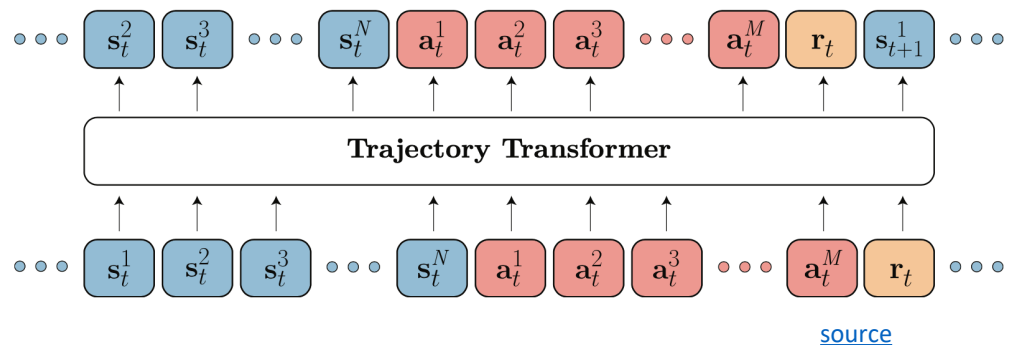
credit assignment directly via self-attention: implicitly forming state-return associations via similarity of query and key vectors (maximizing the dot product)

desired return tokens as prompt for action generation

Decision Transformer: conditioning on desired return, past states and actions to generate future actions



Trajectory Transformer: predicting also states and returns (adding model-based components, planning with beam search)



Direct Policy Search

Policy Gradient Methods

learning of parametrized policy (without value functions) $\pi(a_t|s_t; \hat{\theta})$:
probability to take different actions (target) given a state
(variables/features) and parameters (e.g., neural network weights)

goal maximizing expected cumulative rewards

→ objective function corresponds to true state value: $J(\hat{\theta}) = v_{\pi}(s_t)$

policy gradient theorem:

$$\nabla_{\hat{\theta}} J(\hat{\theta}) \propto \sum_{a_t} q_{\pi}(s_t, a_t) \nabla_{\hat{\theta}} \pi(a_t|s_t; \hat{\theta})$$

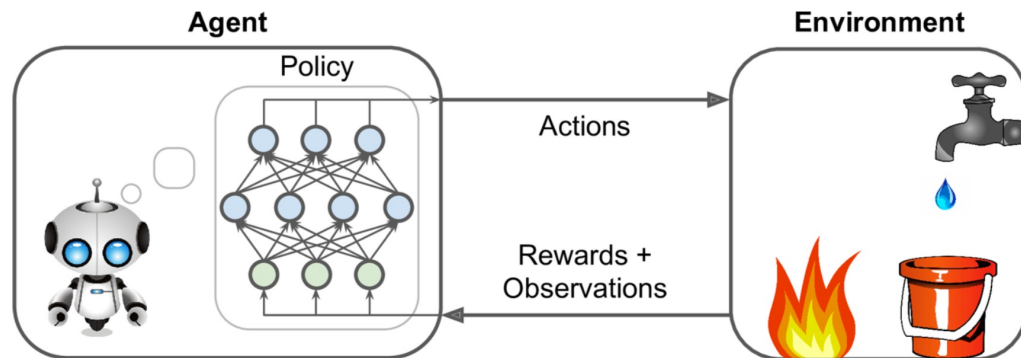
REINFORCE

REINFORCE method (MC method following from policy gradient theorem):

$$\hat{\theta} \leftarrow \hat{\theta} + \eta \cdot \underbrace{\nabla_{\hat{\theta}} [\log \pi(a_t | s_t; \hat{\theta})] \cdot (r_{t+1} + \gamma r_{t+2} + \dots)}_{\nabla_{\hat{\theta}} J(\hat{\theta})}$$

policy gradients \rightarrow neural network gradients

“weighting” with observed
(discounted) return



REINFORCE with Baseline

policy gradient theorem unchanged by subtracting an action-independent baseline, e.g., an estimate of the state-value function:

$$\nabla_{\hat{\theta}} J(\hat{\theta}) \propto \sum_{a_t} [q_{\pi}(s_t, a_t) - \hat{v}(s_t; \hat{\mathbf{w}})] \nabla_{\hat{\theta}} \pi(a_t | s_t; \hat{\theta})$$

e.g., separate
networks

$$\hat{\theta} \leftarrow \hat{\theta} + \eta \cdot \nabla_{\hat{\theta}} [\log \pi(a_t | s_t; \hat{\theta})] \cdot [(r_{t+1} + \gamma r_{t+2} + \dots) - \hat{v}(s_t; \hat{\mathbf{w}})]$$

hybrid between policy-based and value-based methods

→ reduction of variance

Actor-Critic Methods

using state-value function for bootstrapping \rightarrow critic of policy:

$$\hat{\boldsymbol{\theta}} \leftarrow \hat{\boldsymbol{\theta}} + \eta \cdot \nabla_{\hat{\boldsymbol{\theta}}} [\log \pi(a_t | s_t; \hat{\boldsymbol{\theta}})] \cdot \underbrace{[(r_{t+1} + \gamma \hat{v}(s_{t+1}; \hat{\mathbf{w}})) - \hat{v}(s_t; \hat{\mathbf{w}})]}_{\text{TD error}}$$

turning MC (observed return) into TD method

\rightarrow introduction of bias, but further reduction of variance

Synonym: Advantage Actor-Critic

for the critic of the action policy (actor):

interpret TD error $r_{t+1} + \gamma \hat{v}(s_{t+1}; \hat{\mathbf{w}}) - \hat{v}(s_t; \hat{\mathbf{w}})$

as advantage function $\hat{q}(s_t, a_t; \hat{\mathbf{w}}) - \hat{v}(s_t; \hat{\mathbf{w}})$

idea: calculates extra reward for specific action compared to average action in given state (expected state value)

Proximal Policy Optimization (PPO): prominent advantage actor-critic method with some tricks

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

RL from Human Feedback

example for supporting
large language models
(transformers) with RL

used in famous ChatGPT

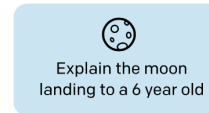
goal: improve alignment
with user intentions

→ learn from human
preferences

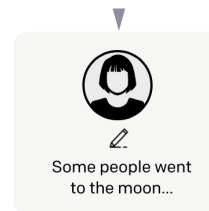
Step 1

**Collect demonstration data,
and train a supervised policy.**

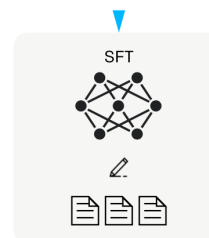
A prompt is
sampled from our
prompt dataset.



A labeler
demonstrates the
desired output
behavior.



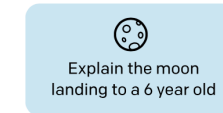
This data is used
to fine-tune GPT-3
with supervised
learning.



Step 2

**Collect comparison data,
and train a reward model.**

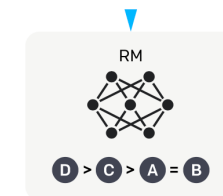
A prompt and
several model
outputs are
sampled.



A labeler ranks
the outputs from
best to worst.



This data is used
to train our
reward model.



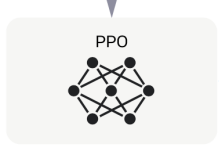
Step 3

**Optimize a policy against
the reward model using
reinforcement learning.**

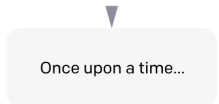
A new prompt
is sampled from
the dataset.



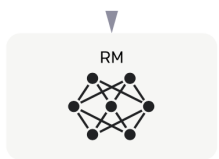
The policy
generates an output.



The reward model
calculates a
reward for
the output.



The reward is
used to update
the policy
using PPO.



[source](#)

Famous Example of Deep RL: AlphaGo

Monte Carlo tree search (heuristic search algorithm) for move (action) selection (focus on current state rather than full state space)

→ decision-time planning

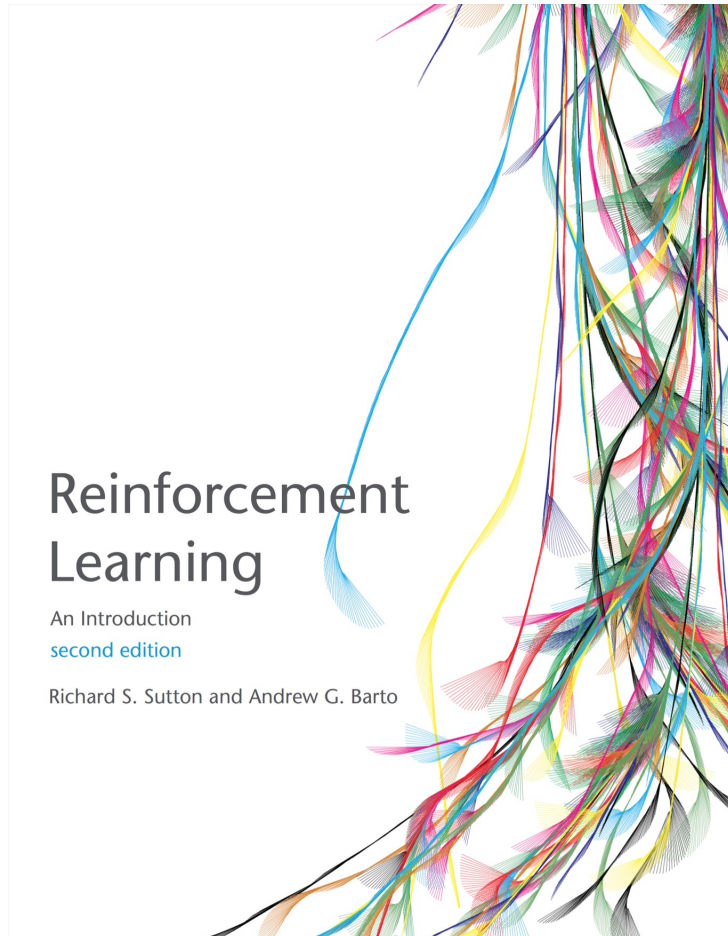
guided by deep convolutional neural networks for both value function and policy estimation

→ 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)

Literature



[pdf](#)

papers:

- [DQN](#), [Atari](#)
- [AlphaGo](#), [AlphaGo Zero](#)
- [PPO](#)

Automation

one of most impactful goals of AI (e.g., get rid of repetitive tasks)

so far mainly for tasks in computer vision, NLP, but also structured data (e.g., automated replenishment)

next step: autonomous decision-making (e.g., autonomous driving, robotics)

→ support technology challenges like [nuclear fusion plasma stabilization](#)