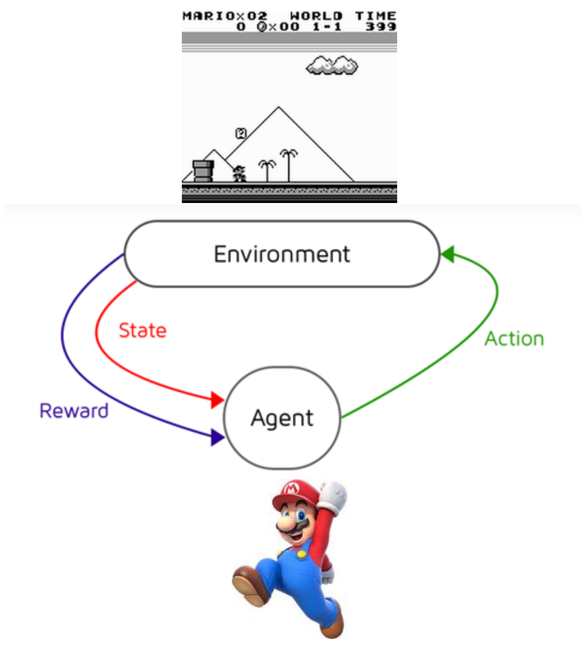
Reinforcement Learning

Fundamentals of Artificial Intelligence

Sequential Decision Making

reinforcement learning (RL):

formalization of sequential decision making (action policy) of software agent interacting with environment



corresponds to search for best (or rather good) action policy to reach a given goal (e.g., win a game)

using learning from examples (data) to guide the search

RL usually more difficult (e.g., non-differentiable as a whole) than supervised learning (which can be seen as “generalized optimization”, often of proxy metric)

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 generating observations to learn from (exploration) independent from updated policy (current best)

feedback from environment: goal-directed, no supervision

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

Optional Elements of RL

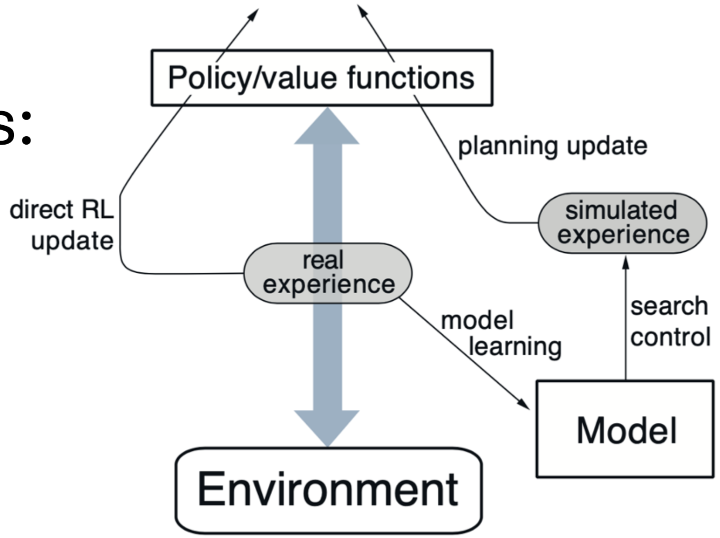
value functions for states or actions: improve efficiency of search in vast action policy space (alternative: direct policy search)

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)

$$\text{model} \xrightarrow{\text{planning}} \text{policy}$$



from Sutton

Markov Decision Process (MDP)

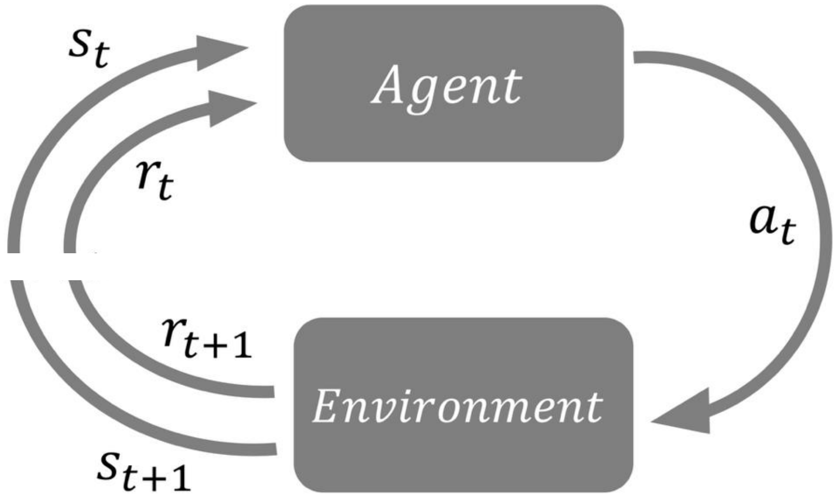
idea: current state includes all information about past

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

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)

discount rate

$$v_\pi(s_t) = E_\pi\left[\overbrace{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t}^{\text{return}}\right] = E_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})|s_t]$$

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

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

transition probability (depending on environment) from state s_t to state s'_{t+1} for a given action

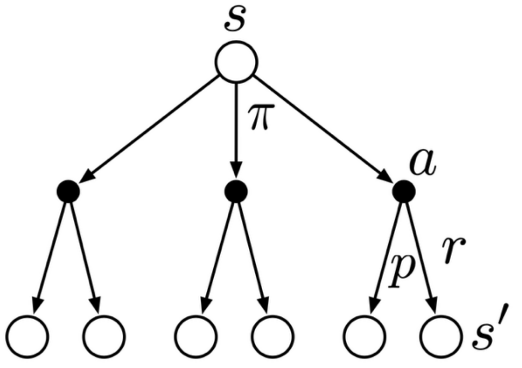
Bellman (expectation) equation: recursion

(sweep through entire state space)

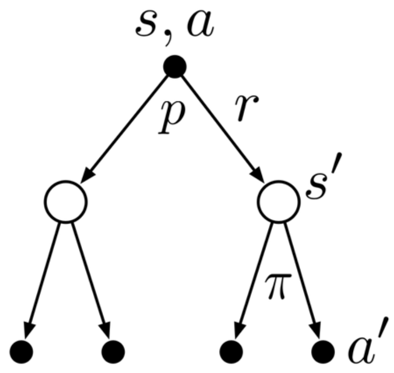
Action-Value Function

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

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



Backup diagram for v_π

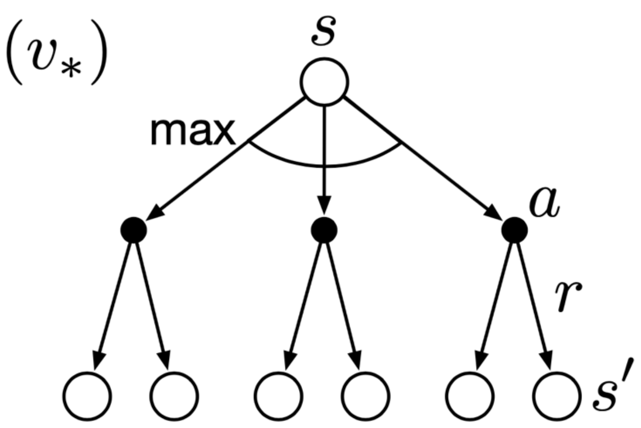


q_π backup diagram

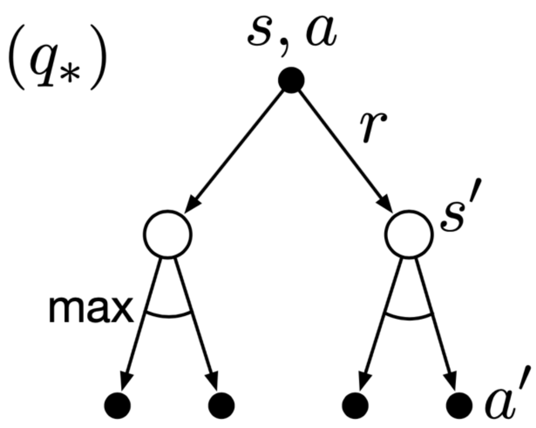
from Sutton

Bellman Optimality Equations

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



from Sutton



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

→ approximate solutions

Dynamic Programming (DP)

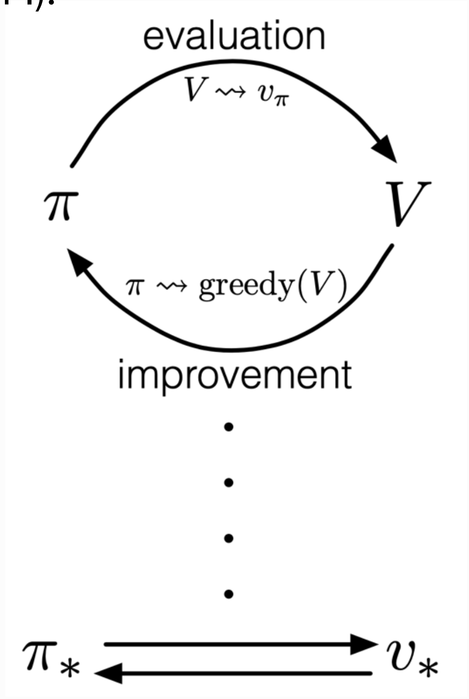
iterative approaches to find approximations for optimal value functions

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 (stopped after one update of each state)

generalized policy iteration (GPI):



from Sutton

GPI also followed by MC and TD methods ...

Limited Utility of DP

requires full model of environment

computationally expensive

- expected update operation (based on values of all possible successor states and their probability)
- for each state (in potentially huge state space)

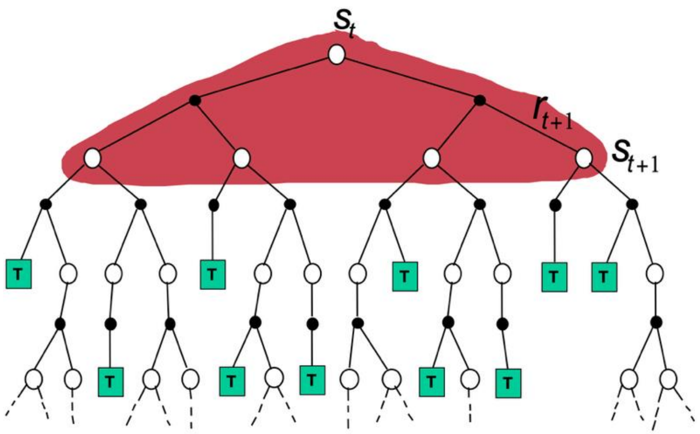
(asynchronous DP at least avoids systematic sweeps over entire state space)

→ need for more efficient methods achieving the same effect as DP, without (perfect) model of environment

Bootstrapping and Sampling

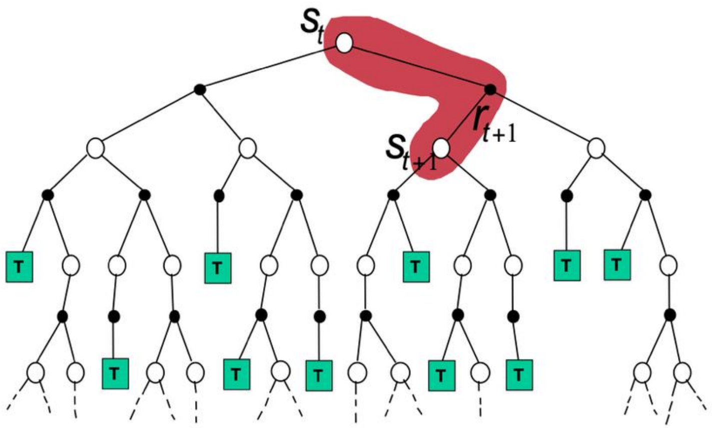
bootstrapping: update estimates of state values based on estimates of values of successor states
sampling: experience of sample sequences (no need for complete knowledge of environment)

Dynamic Programming



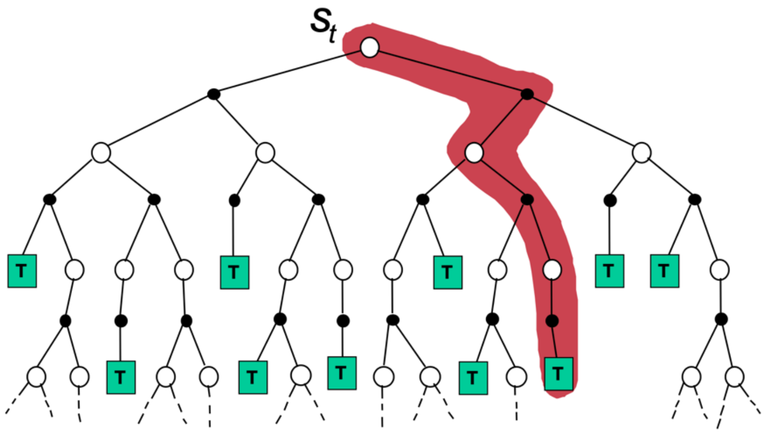
- bootstrapping
- no sampling → model-based (transition probabilities needed)

Temporal Difference (TD) Learning



- bootstrapping
- sampling → model-free

Monte Carlo (MC)



- no bootstrapping
- sampling → model-free

from Sutton

Sampling Update Rule

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

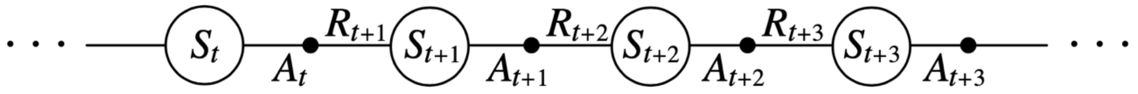
MC: $v(s_t) \leftarrow v(s_t) + \eta[\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} - v(s_t)]$

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

bootstrapping

On-Policy TD Control: SARSA

S A R S A



from Sutton

following pattern of GPI:

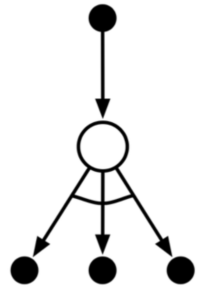
- estimate action-value function for current behavior policy
  $q_\pi(s_t, a_t) \leftarrow q_\pi(s_t, a_t) + \eta[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1}) - q_\pi(s_t, a_t)]$

- 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)

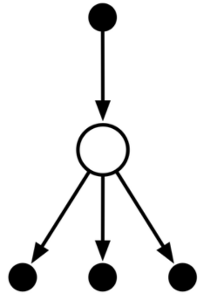
$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \eta \left[r_{t+1} + \gamma \max_a q(s_{t+1}, a_{t+1}) - q(s_t, a_t) \right]$$



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

compare to expected Sarsa:

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

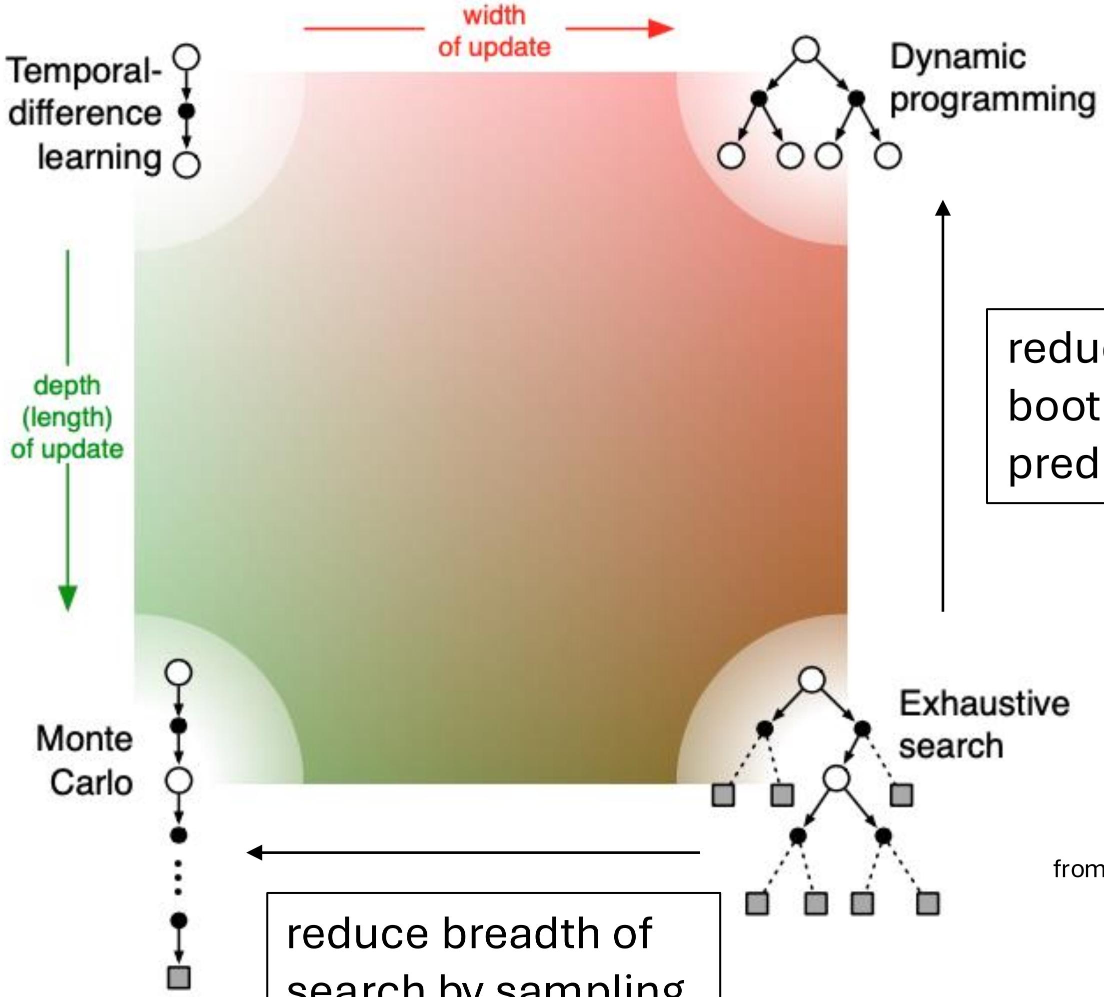


Summary: Update Characteristics

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

Limitation of Tabular Methods

tabular methods (calculating values for each state/action) simply memorize observed data

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

→ need for generalization: supervised learning to the rescue

- non-linear function approximation over state/action space
- nowadays often deep learning methods → deep RL

Approximate Solution Methods

online setting: learning while interacting with environment
offline setting: learning from limited data without further interaction with environment

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 for supervised learning (e.g., squared error loss):

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

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

→ RL problem expressed in supervised learning setup (potentially offline/batch data)

but $v_{\pi}(s)$ still calculated via RL methods (e.g., bootstrapping)

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

assumption of independent and identically distributed sets of random variables $(Y_1, \boldsymbol{X}_1), (Y_2, \boldsymbol{X}_2), \dots, (Y_n, \boldsymbol{X}_n)$ 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 (→ 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

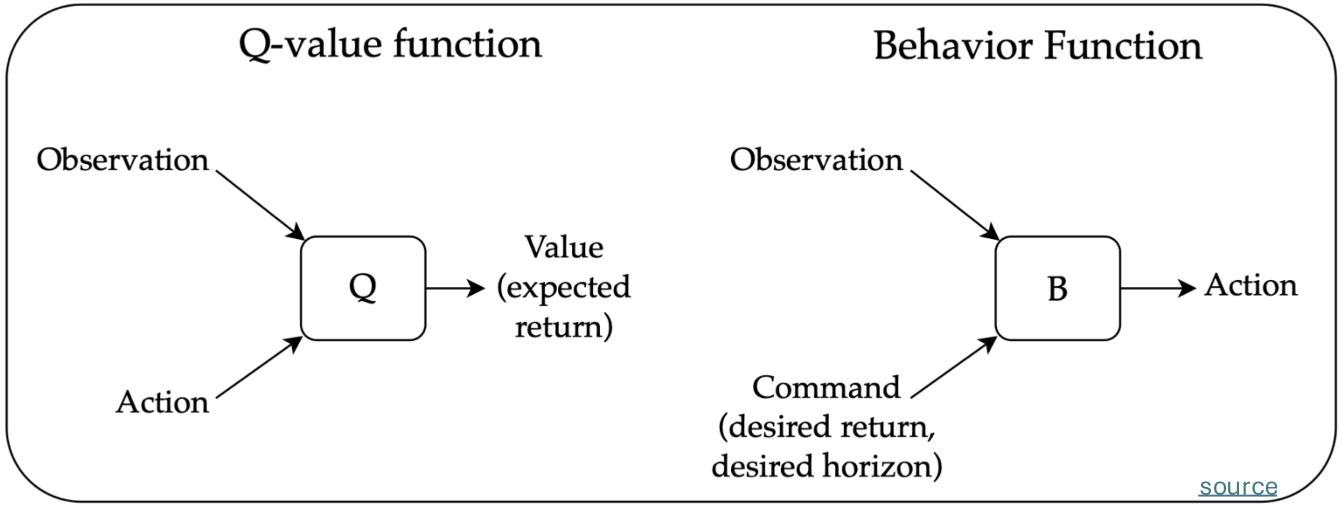
issue in deep RL: combination of off-policy bootstrapping (e.g., Q-learning) with high-dimensional function approximation leads to non-stationary targets (unstable)

most popular technique to overcome this: target networks in DQN

offline RL: no interaction with environment, just fixed data set of trajectory rollouts of arbitrary policies

alternative (to conventional RL): upside–down RL

→ no bootstrapping, just supervised learning with "command" features (hindsight return in training, kind of prompt in inference)



source

But policy improvement (i.e., higher return) beyond training examples (extrapolation) usually still requires policy iteration (here: iteratively updated trainings with new data with higher returns).

Deep Q-Network (DQN)

idea: deep neural network(s) approximating tabular action-value function (according to Q-learning): $q(s, a; \hat{\boldsymbol{w}})$ 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 (due to bootstrapping)
- 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.")

Famous Example of Deep RL: AlphaGo

Monte Carlo tree search (heuristic, lookahead search) for move (i.e., action) selection

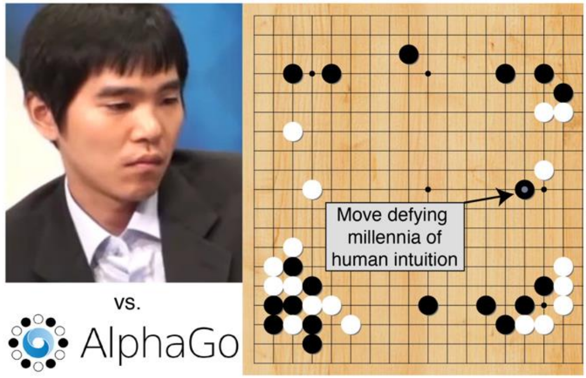
→ decision-time planning (advantage: focus on current state rather than full state space)

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)

famous move 37:



from Max Tegmark

Policy Gradient Methods

learning of parametrized policy (without value functions)
$\pi(a_t|s_t; \widehat{\boldsymbol{\theta}})$: 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: $J(\widehat{\boldsymbol{\theta}}) = v_\pi(s_t)$

policy gradient theorem:

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

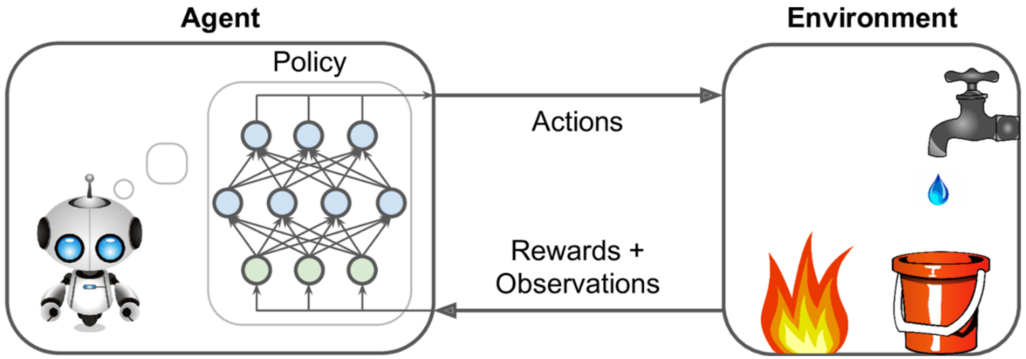
REINFORCE

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

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

"weighting" with observed (discounted) return

policy gradients → neural network gradients



policy gradient methods:
on-policy learning

REINFORCE with Baseline

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

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

e.g., separate networks

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

hybrid between policy-based and value-based methods

→ reduction of variance

Actor-Critic Methods

using state-value function for bootstrapping → critic of policy:

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

turning MC (observed return) into TD method

→ introduction of bias, but further reduction of variance

Synonym: Advantage Actor-Critic

for the critic of the action policy (actor):

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

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

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

overcome Markov property

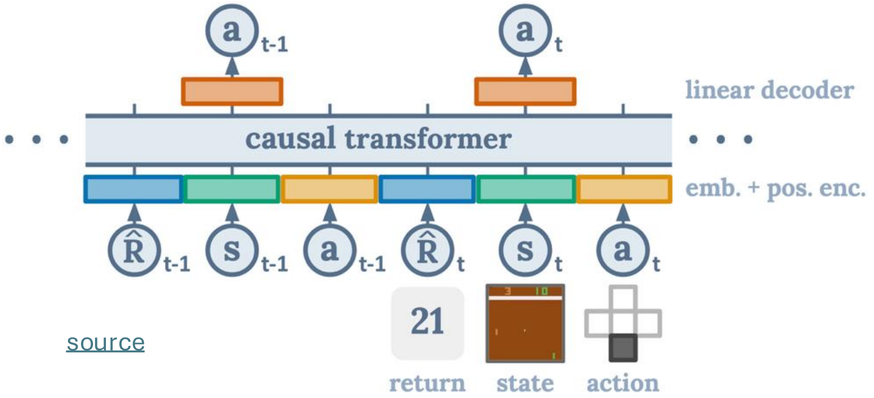
Sequence Modeling for Decisions/Actions

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)

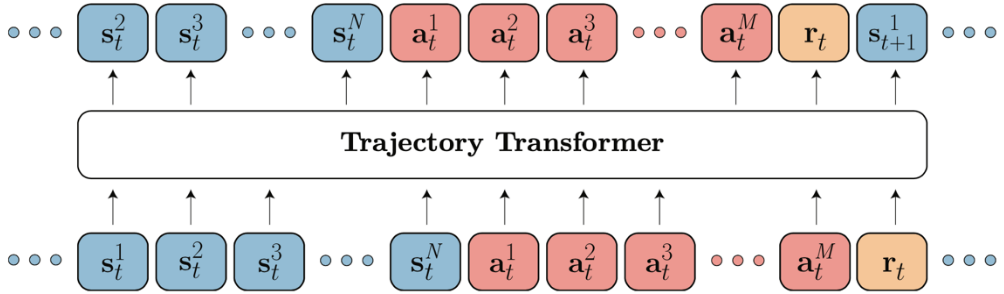
desired return tokens as prompt for action generation

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



source

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



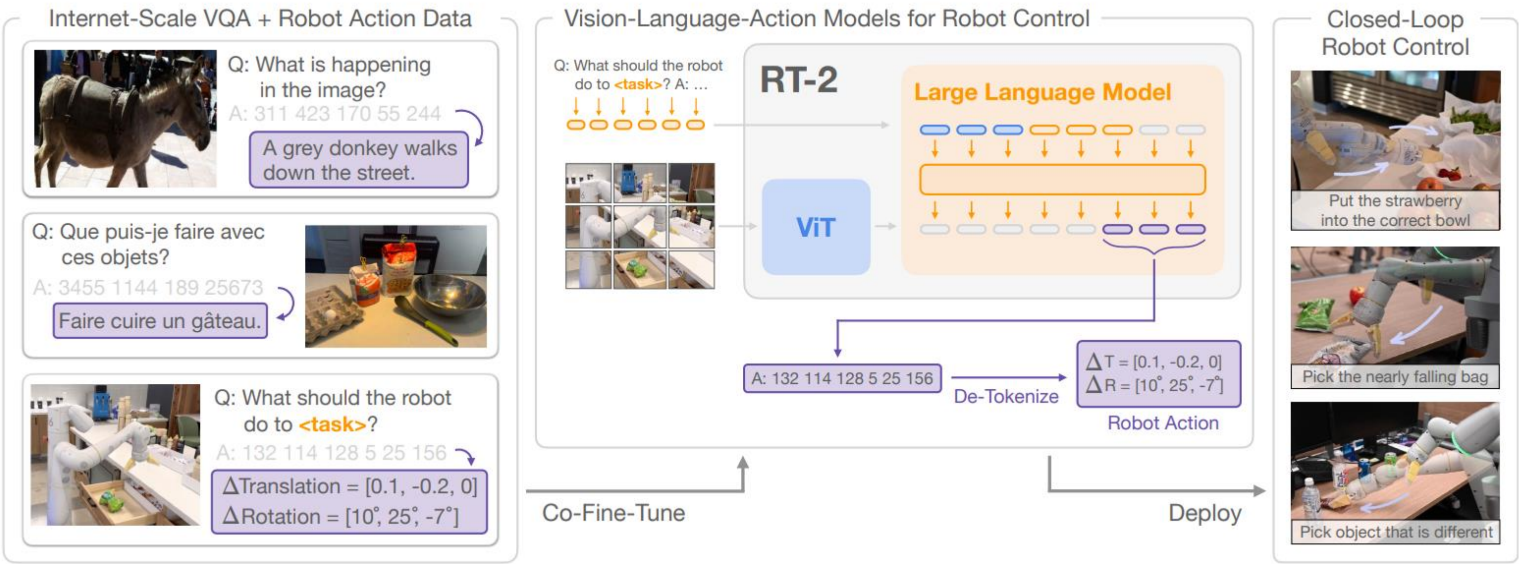
source

can also condition imitation learning
diffusion objective for multimodal outputs

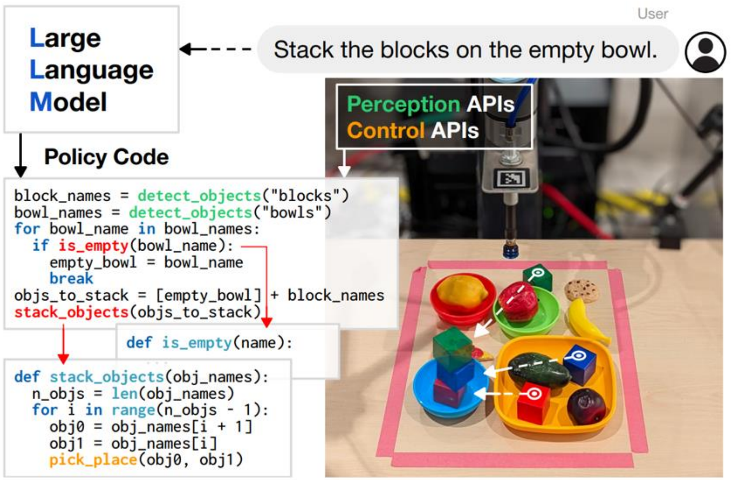
Robotic Control via LLMs (and Vision)

RT-2: vision-language-action model learning from web and robotics data

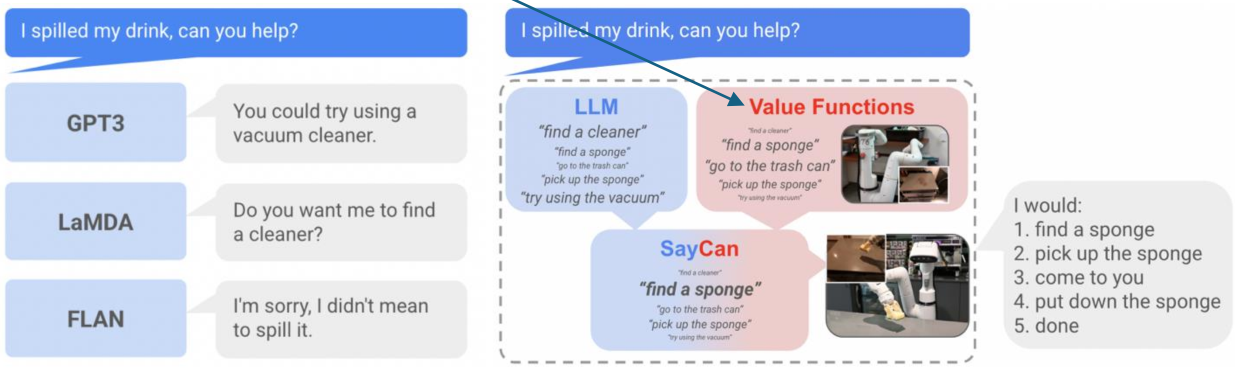
- representation of actions as tokens
- generalization by using pre-trained vision-language models



Code as Policies:



grounding with pre-trained skills (SayCan):



(a) Large Language Models (LLMs)   (b) SayCan