Reinforcement Learning
Sequential Decision Making

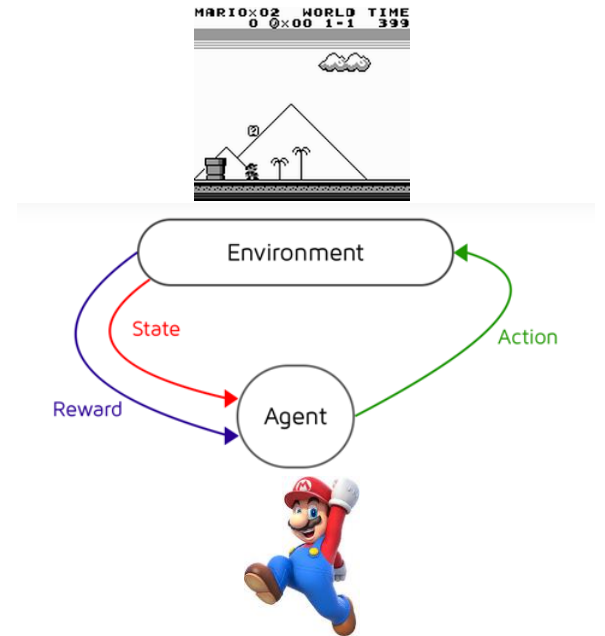Understanding Machine Learning

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)

from Sutton
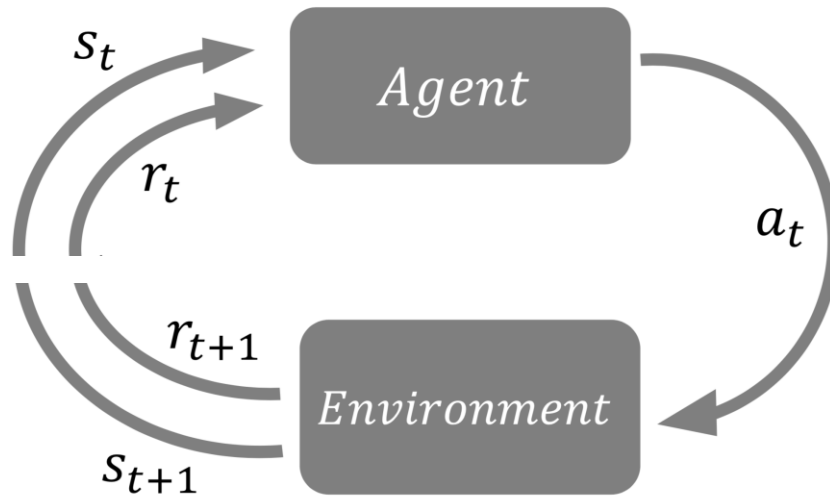
4
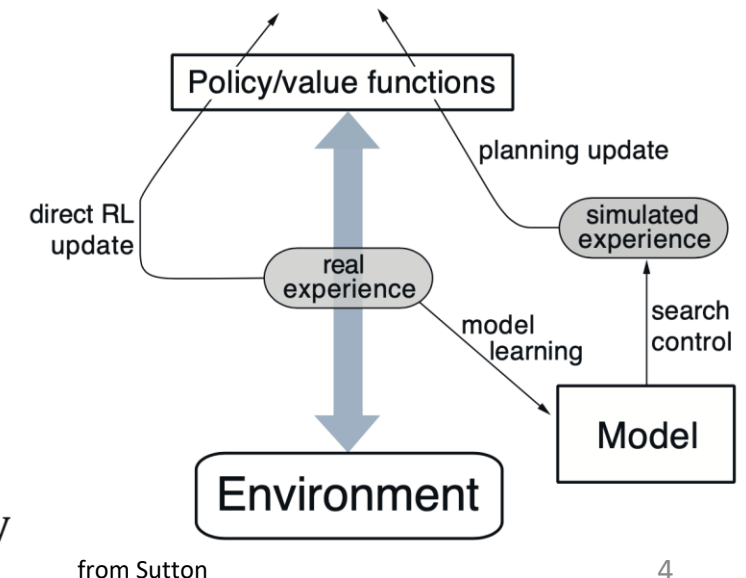
Markov Decision Process (MDP)

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

return

discount rate

(needed for all states)

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

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

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
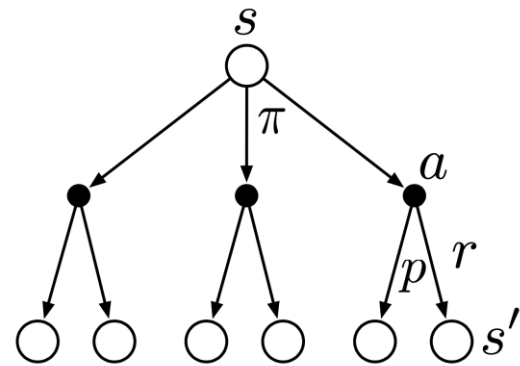
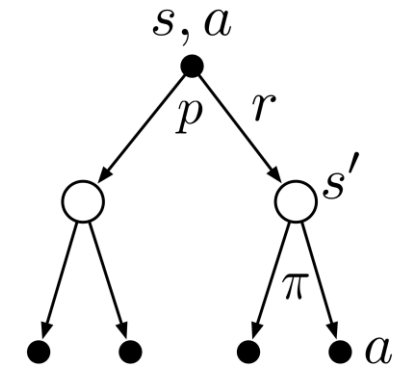Bellman (expectation) equation: recursion

(sweep through entire state space)

Action-Value Function

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

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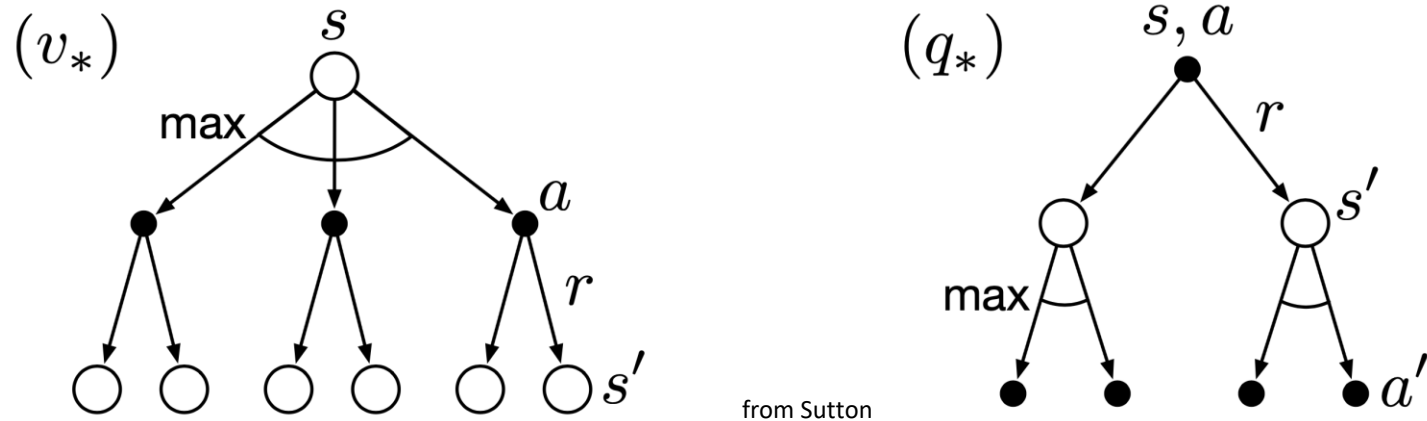
Backup diagram for v_π

q_π backup diagram

10

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: $\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$

- value iteration: truncated policy evaluation using Bellman optimality equation as update rule (stopped after one update of each state)

generalized policy iteration (GPI):

evaluation

$V \rightsquigarrow v_\pi$

π V

$\pi \rightsquigarrow \text{greedy}(V)$

improvement

π_* v_*
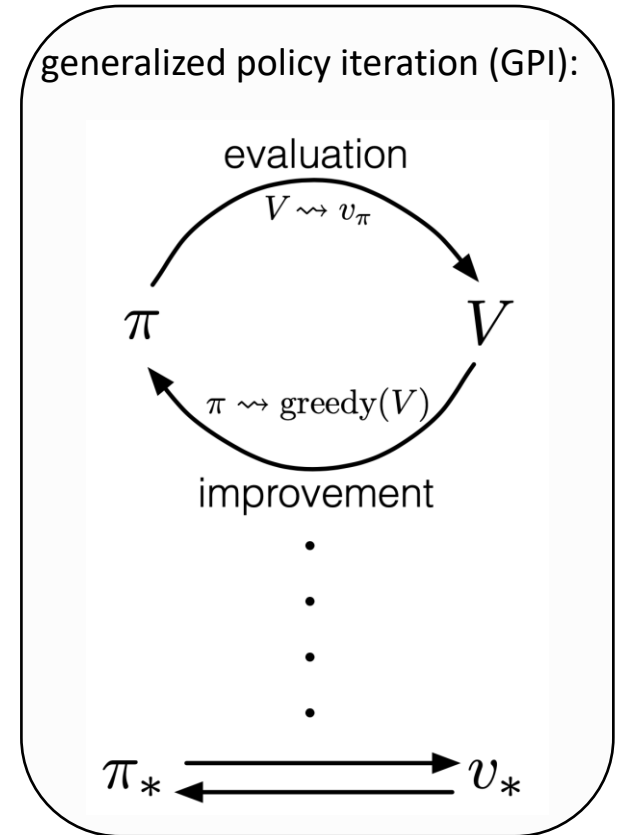
from Sutton

GPI also followed by MC and TD methods ...

12

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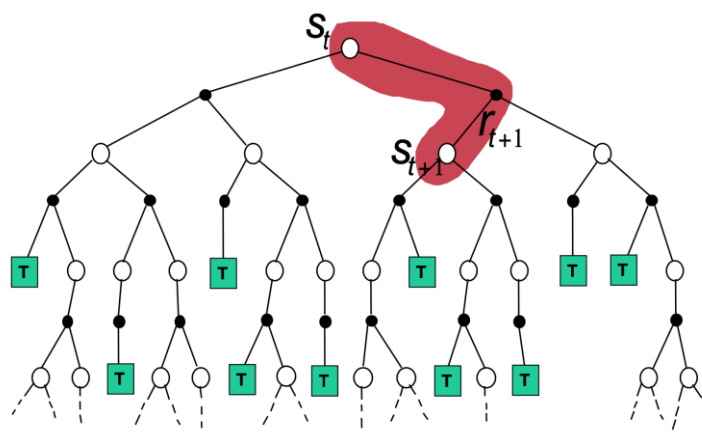**sampling**: experience of sample sequences (no need for complete knowledge of environment)

Dynamic Programming   Temporal Difference (TD) Learning   Monte Carlo (MC)
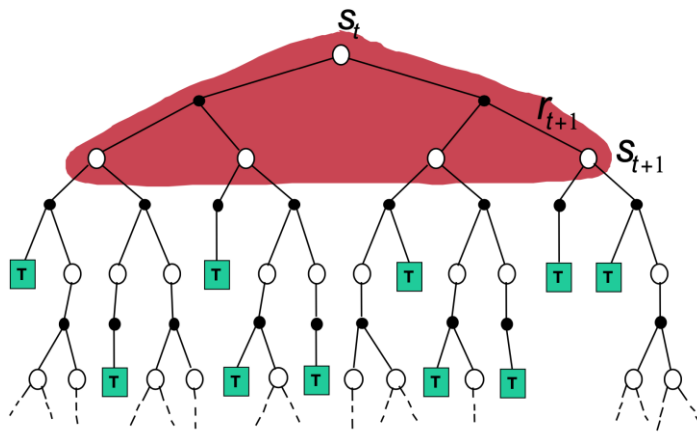
from Sutton

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

- bootstrapping
- sampling → model-free
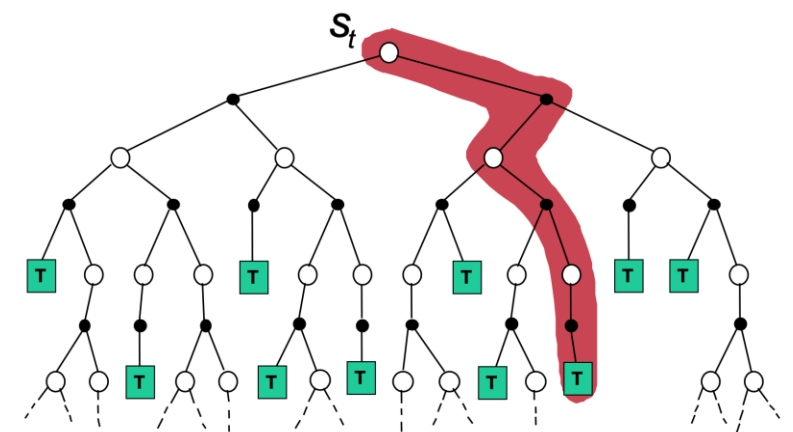
- no bootstrapping
- sampling → model-free

14

Sampling Update Rule

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

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

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

S   A   R   S   A

from Sutton
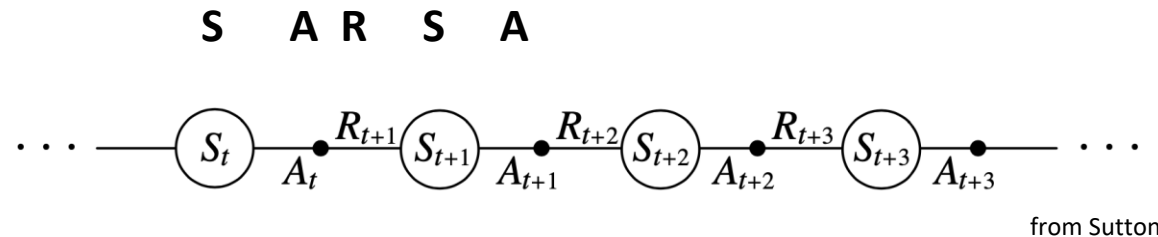
following pattern of GPI:

- 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
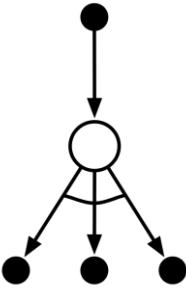
compare to expected Sarsa:

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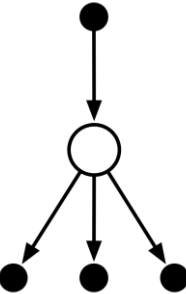
Summary: Update Characteristics

one-step TD

n-step TD

MC

width of update

Temporal-difference learning

Dynamic programming

depth (length) of update

Monte Carlo

Exhaustive search

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

reduce breadth of search by sampling actions from policy

from Sutton
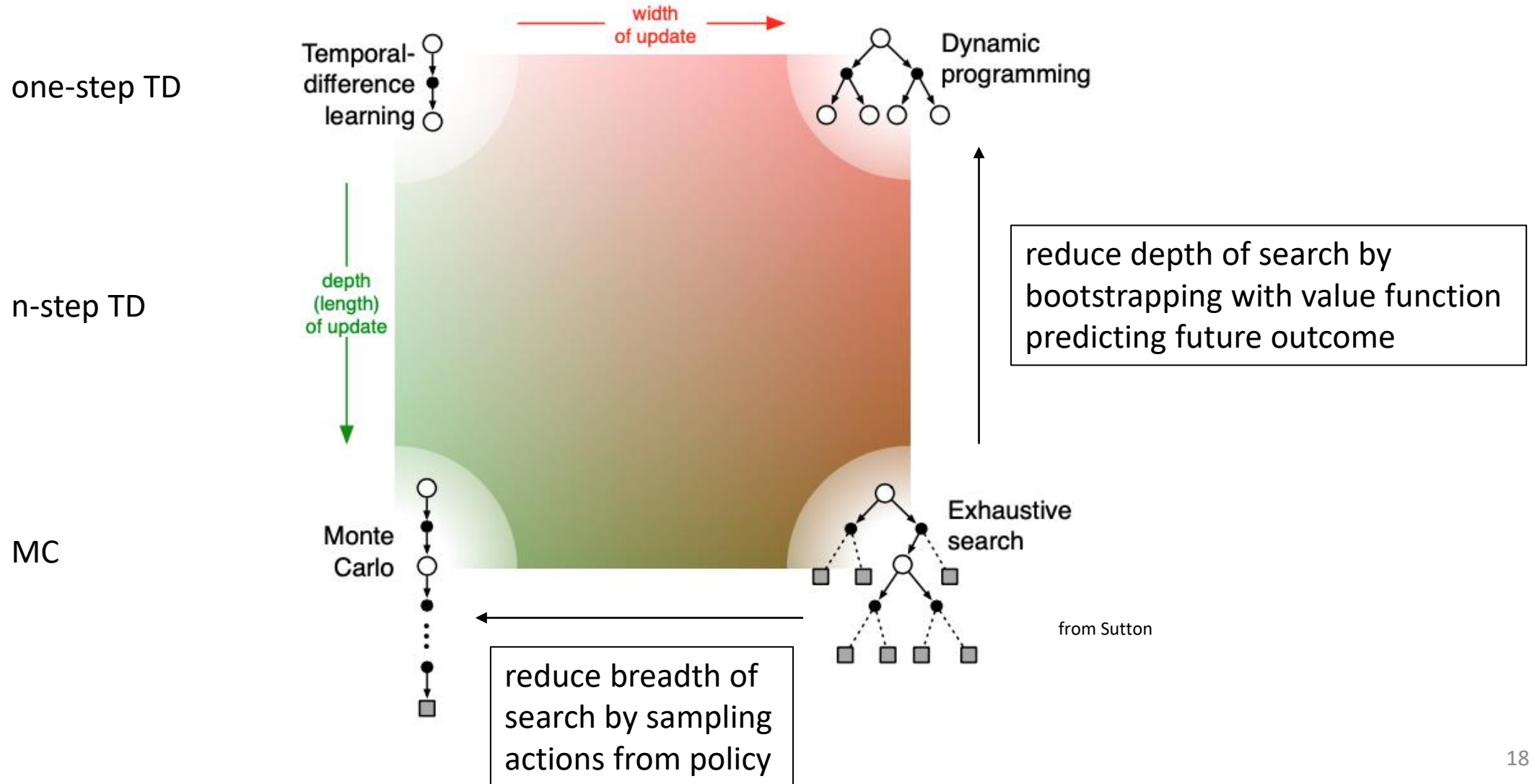
Deep Reinforcement Learning

Limitation of Tabular Methods

tabular methods (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)

$\rightarrow$ need for generalization: supervised learning to the rescue

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

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

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

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)

Deep Q-Network (DQN)

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

key components to get it going:

- separate target network: weights only periodically updated with estimated Q-network weights $\rightarrow$ 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) $\rightarrow$ 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 $(Y_1, \boldsymbol{X}_1), (Y_2, \boldsymbol{X}_2), \ldots, (Y_n, \boldsymbol{X}_n)$ 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 ($\rightarrow$ 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 (to conventional RL): upside–down RL

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

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

Q-value function   Behavior Function

Observation → Q → Value (expected return)
Action → Q

Observation → B → Action
Command (desired return, desired horizon) → B

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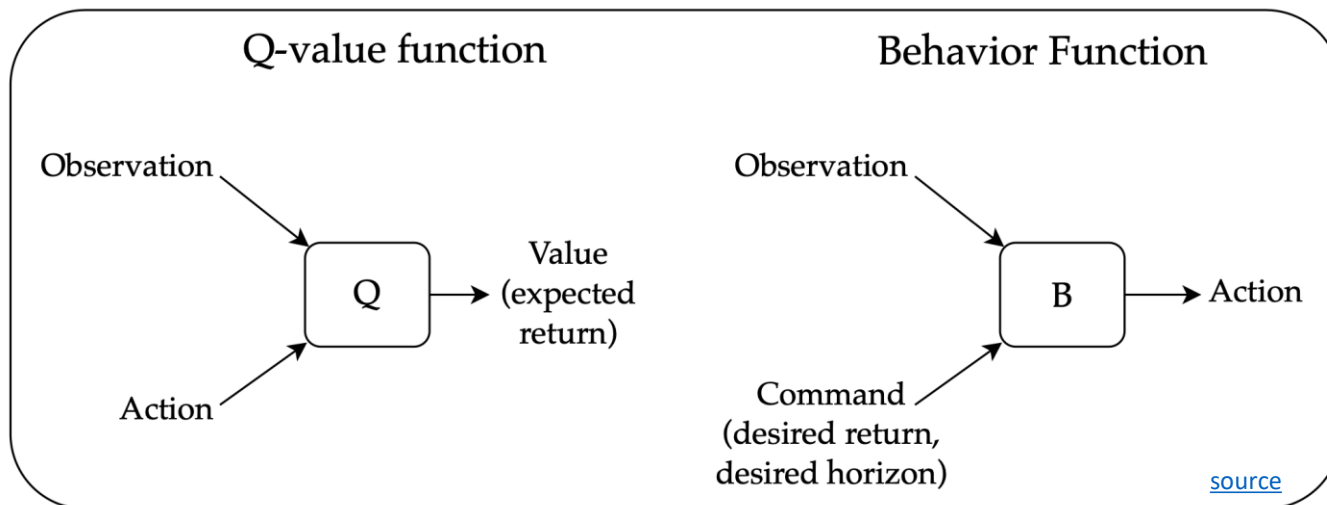
But policy improvement (i.e., higher return) beyond training examples (extrapolation) usually still requires policy iteration (here: iteratively updated trainings with new data with higher returns).

24

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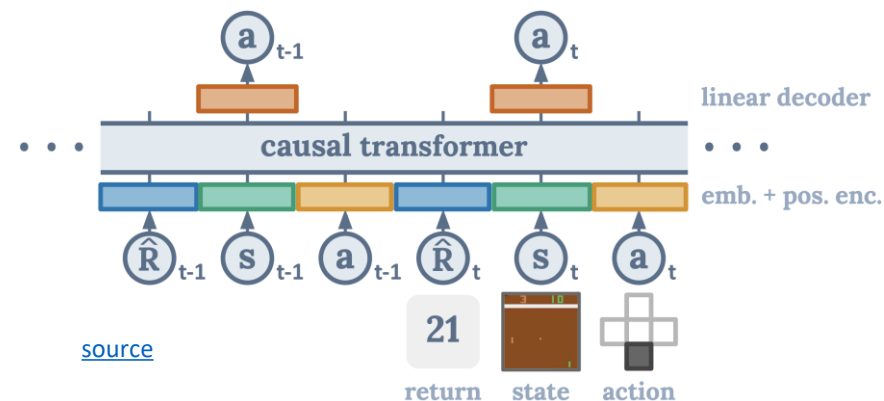
Trajectory Transformer: predicting also states and returns (adding model-based components, planning with beam search)
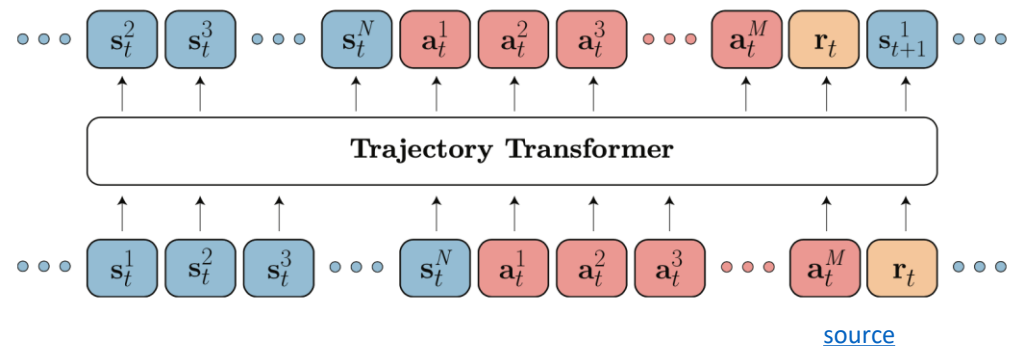
source

return   state   action

source

25

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

Internet-Scale VQA + Robot Action Data

Q: What is happening in the image?
A: 311 423 170 55 244
A grey donkey walks down the street.

Q: Que puis-je faire avec ces objets?
A: 3455 1144 189 25673
Faire cuire un gâteau.

Q: What should the robot do to <task>?
A: 132 114 128 5 25 156
ΔTranslation = [0.1, -0.2, 0]
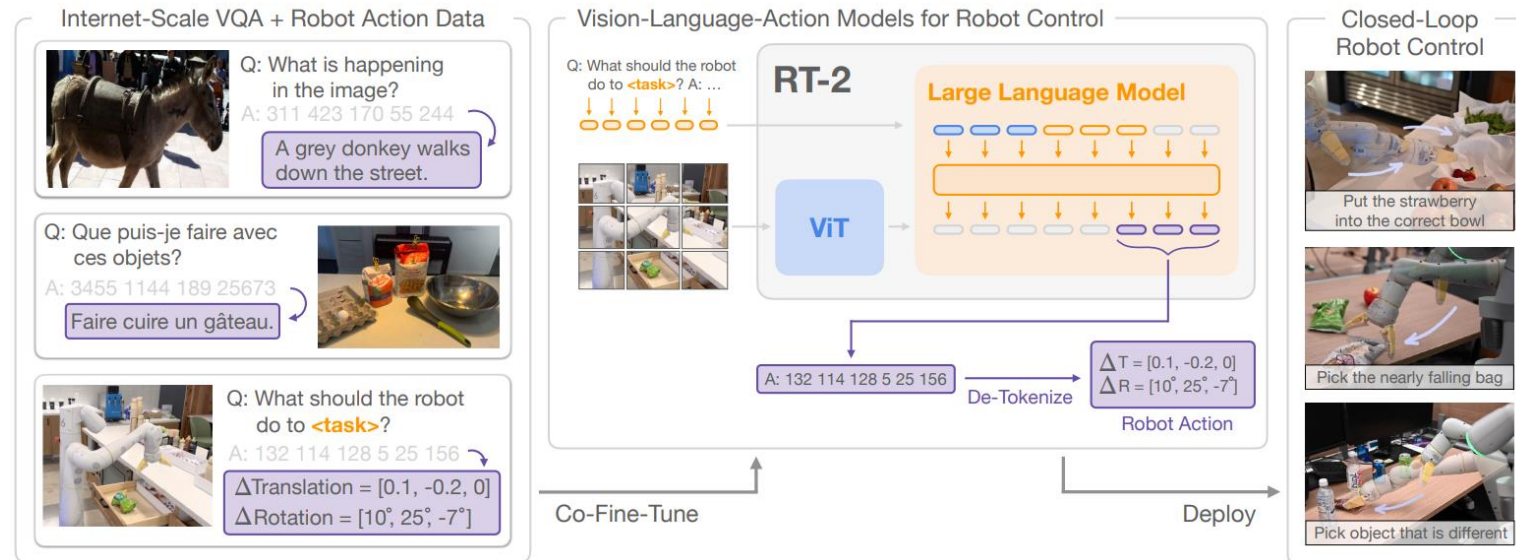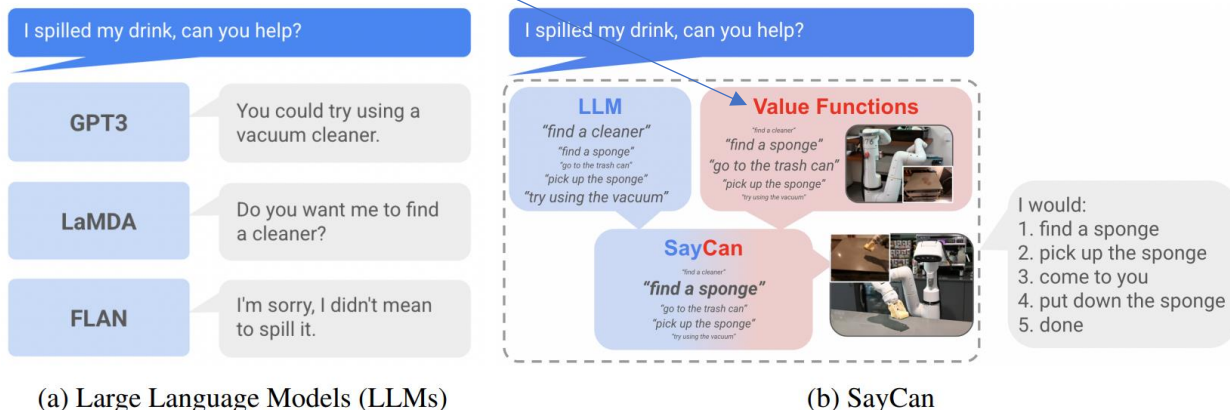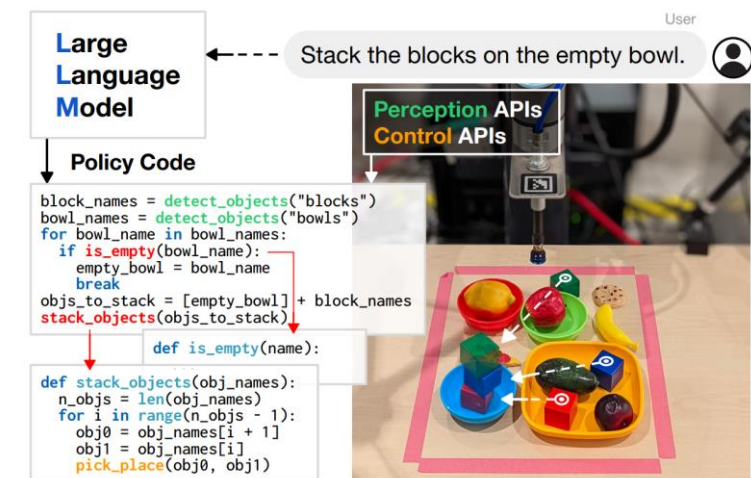ΔRotation = [10°, 25°, -7°]

Vision-Language-Action Models for Robot Control

Q: What should the robot do to <task>? A: …

RT-2

Large Language Model

ViT

A: 132 114 128 5 25 156
De-Tokenize
ΔT = [0.1, -0.2, 0]
ΔR = [10°, 25°, -7°]
Robot Action

Co-Fine-Tune   Deploy

Closed-Loop Robot Control

Put the strawberry into the correct bowl

Pick the nearly falling bag

Pick object that is different

Code as Policies:

grounding with pre-trained skills (SayCan):

I spilled my drink, can you help?

GPT3 — You could try using a vacuum cleaner.

LaMDA — Do you want me to find a cleaner?

FLAN — I'm sorry, I didn't mean to spill it.

(a) Large Language Models (LLMs)

I spilled my drink, can you help?

LLM
"find a cleaner"
"find a sponge"
"go to the trash can"
"pick up the sponge"
"try using the vacuum"

Value Functions
"find a cleaner"
"find a sponge"
"go to the trash can"
"pick up the sponge"
"try using the vacuum"

SayCan
"find a sponge"
"go to the trash can"
"pick up the sponge"
"try using the vacuum"

I would:
1. find a sponge
2. pick up the sponge
3. come to you
4. put down the sponge
5. done

(b) SayCan

Large Language Model — Stack the blocks on the empty bowl.

Policy Code
```
block_names = detect_objects("blocks")
bowl_names = detect_objects("bowls")
for bowl_name in bowl_names:
    if is_empty(bowl_name):
        empty_bowl = bowl_name
        break
objs_to_stack = [empty_bowl] + block_names
stack_objects(objs_to_stack)

def is_empty(name):

def stack_objects(obj_names):
    n_objs = len(obj_names)
    for i in range(n_objs - 1):
        obj0 = obj_names[i + 1]
        obj1 = obj_names[i]
        pick_place(obj0, obj1)
```

Perception APIs
Control APIs

26

Direct Policy Search

Policy Gradient Methods

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

policy gradient theorem:

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

REINFORCE

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

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

$$\nabla_{\widehat{\boldsymbol{\theta}}} J(\widehat{\boldsymbol{\theta}})$$

policy gradients → neural network gradients

"weighting" with observed (discounted) return

Agent
Policy

Actions

Rewards + Observations

Environment
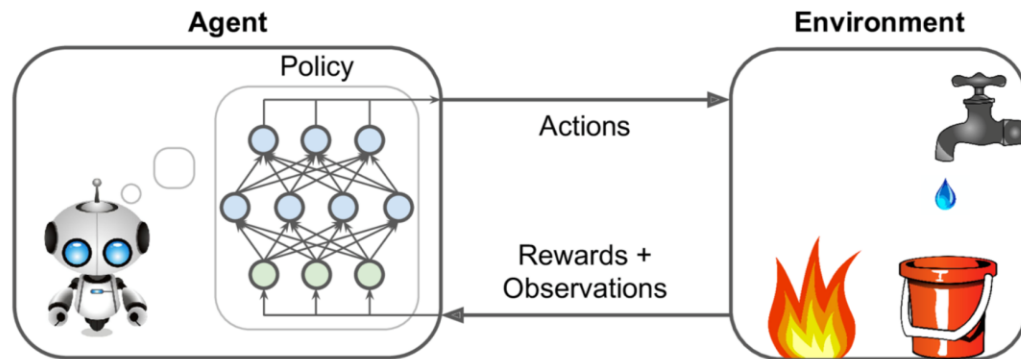
policy gradient methods: on-policy learning

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

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

hybrid between policy-based and value-based methods
→ reduction of variance

Actor-Critic Methods

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

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

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 $\qquad r_{t+1} + \gamma \hat{v}(s_{t+1}; \widehat{\boldsymbol{w}}) - \hat{v}(s_t; \widehat{\boldsymbol{w}})$

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

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

RL from Human Feedback

example for supporting large language models (transformers) with RL

used in famous ChatGPT

goal: improve alignment with user intentions
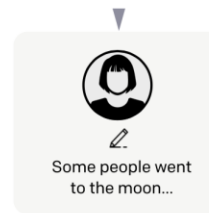
→ learn from human preferences

Step 1
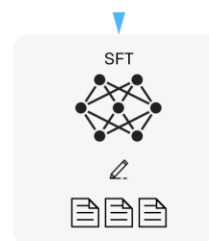**Collect demonstration data, and train a supervised policy.**

A prompt is sampled from our prompt dataset.

Explain the moon landing to a 6 year old
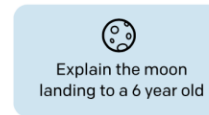
A labeler demonstrates the desired output behavior.
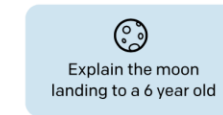
Some people went to the moon...
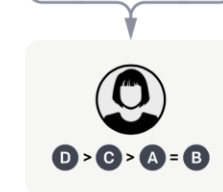
This data is used to fine-tune GPT-3 with supervised learning.

SFT

Step 2
Collect comparison data, and train a reward model.

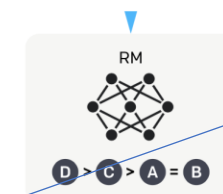A prompt and several model outputs are sampled.

Explain the moon landing to a 6 year old

A: Explain gravity...
B: Explain war...
C: Moon is natural satellite of...
D: People went to the moon...
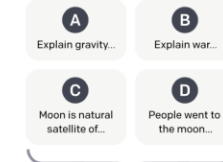
A labeler ranks the outputs from best to worst.

D > C > A = B

This data is used to train our reward model.

RM

D > C > A = B

Step 3
Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

Write a story about frogs

The policy generates an output.

PPO

Once upon a time...

The reward model calculates a reward for the output.

RM
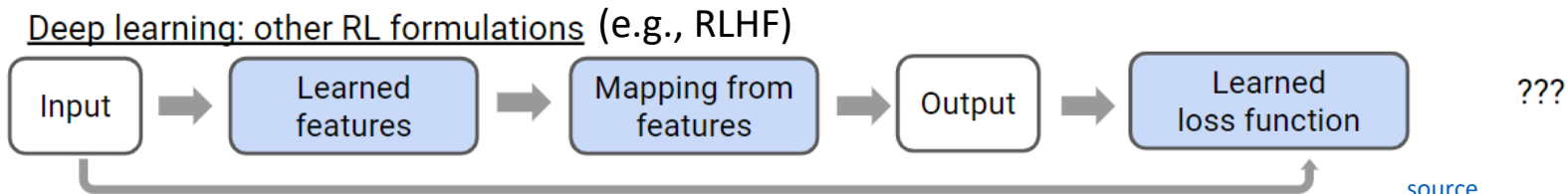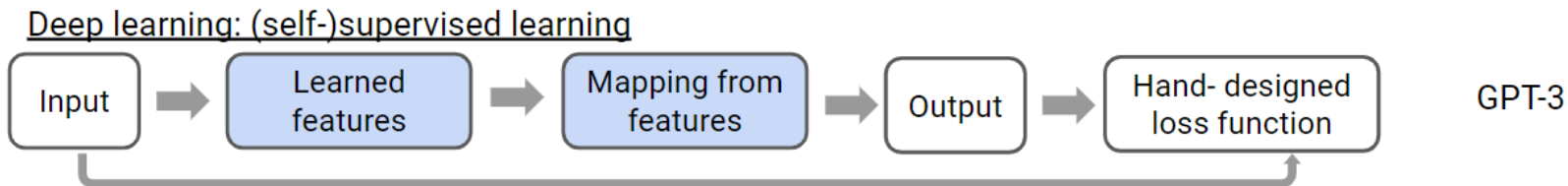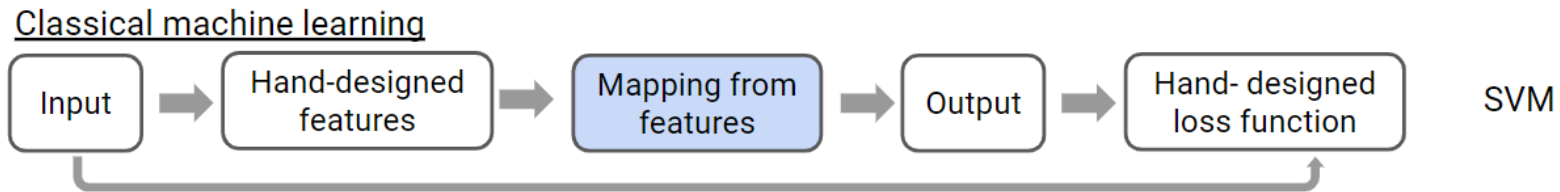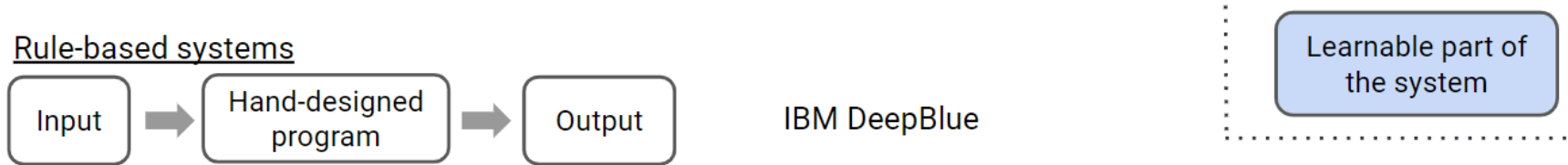
The reward is used to update the policy using PPO.

r_k

RL looks at reward of text output passages as a whole (rather than token-level loss in supervised learning)

33

Reward Model

Rule-based systems

Input → Hand-designed program → Output   IBM DeepBlue

Learnable part of the system

Classical machine learning

Input → Hand-designed features → Mapping from features → Output → Hand-designed loss function   SVM

Deep learning: (self-)supervised learning

Input → Learned features → Mapping from features → Output → Hand-designed loss function   GPT-3

Deep learning: other RL formulations (e.g., RLHF)

Input → Learned features → Mapping from features → Output → Learned loss function   ???

source

loss function for supervised learning sometimes difficult to define

RL allows learning (instead of manually setting) of loss function

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:

vs.

AlphaGo

Move defying millennia of human intuition

from Max Tegmark
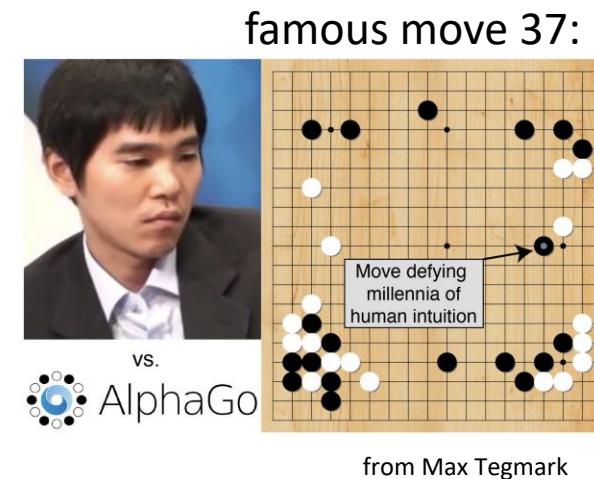
Literature
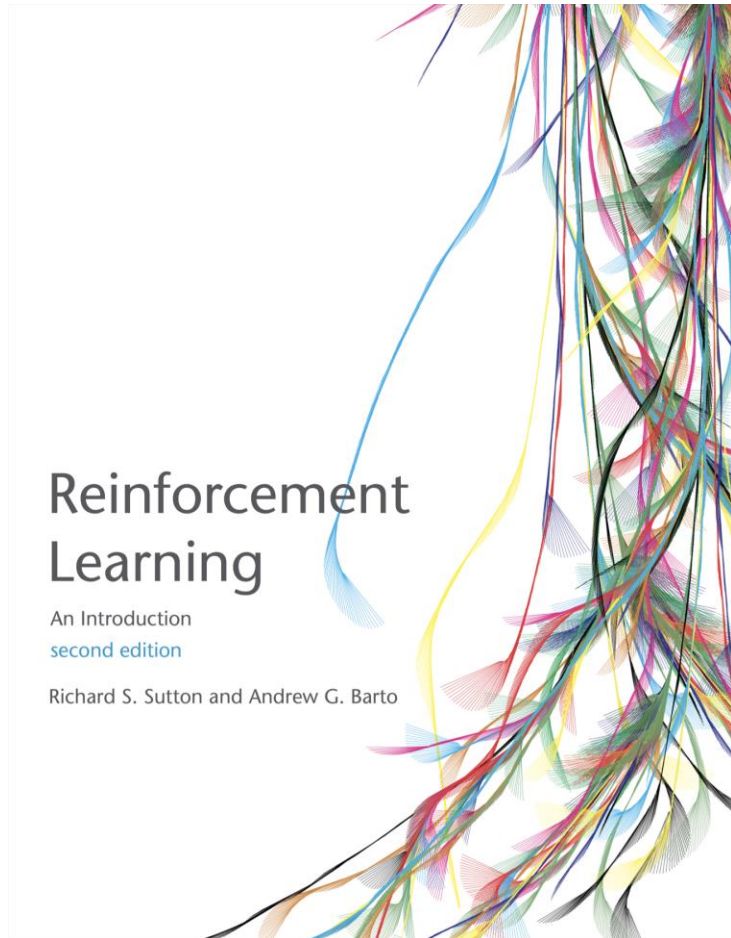
papers:

- [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