MAY 27, 2020

Introduction to (Deep) Reinforcement Learning

Deep Learning Sessions Lisbon

Motivation

Humans learn autonomously how to make decisions

Why are humans so good at RL?

- People have prior experience
  - Humans learn from very few samples *(trial-and-error basis)*
- People have an existing representation of the world
  - Humans (subconsciously) use discrimination and generalization to identify and classify the world
- People are goal-oriented
  - Active behavior with sequential interactions
- Open questions about human behavior
  - Knowledge from evolution, culture, experience, and so on …

Mapping questions

- Is world model learning an essential step to be learned before or in parallel with policy model?
- Can we learn a representation under which RL solve new tasks under very small amount of experience?

Type of Learning

- **Optimization**
- **Generalization**

Supervised Learning

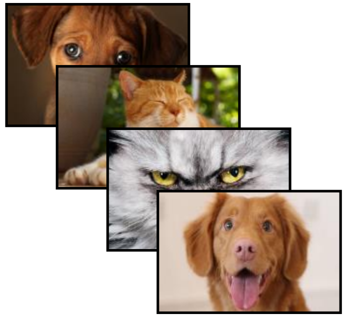
***"Teach by example"** – design of the annotations*

N samples → Any Encoder → Representation → Prediction ← - - - Sample; Prediction

Human annotated GT

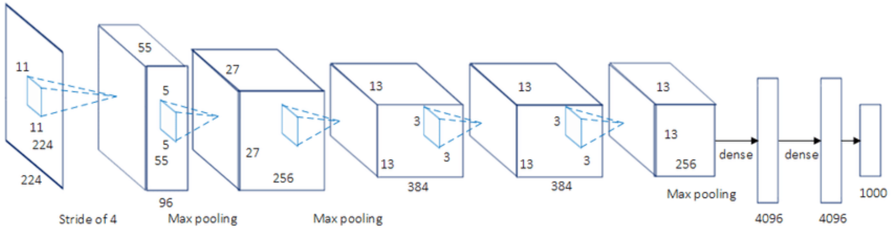
Input

x



Network

$y = f(x;\ \theta)$



Output

y

Data

$\{(x, y)_i\}$

dog
cat
cat
dog

Type of Learning

- **Optimization**
- **Generalization**
- **Delayed consequences**

Imitation Learning

behavioral cloning (supervised learning) for sake of simplicity

"Teach by expert" ***– design to mimic***

N environment samples → Any Encoder → Representation → Action ⇠ Observation; Action

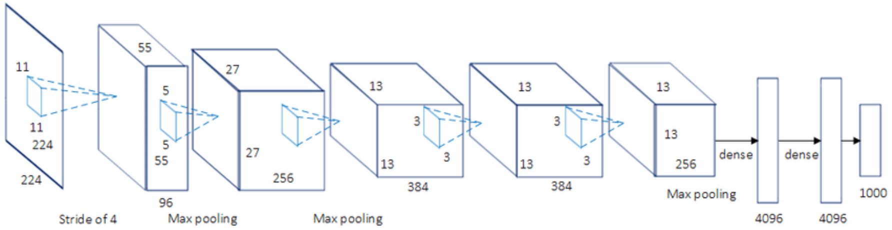
Expert behavior

Input

s_t

Network

$a_t = \pi(s_t;\ \theta)$



Output

a_t

Data

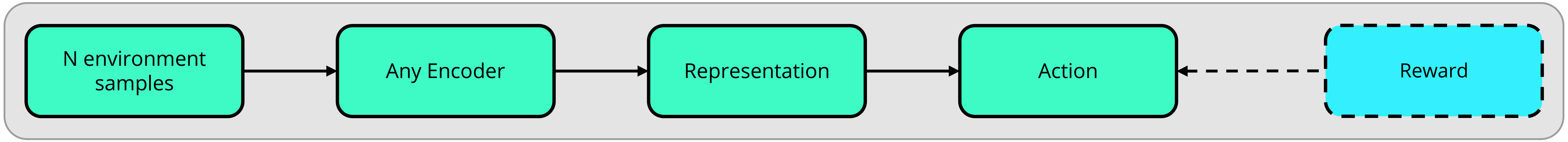
$\{(s_t, a_t^*)\}$

Type of Learning

Reinforcement Learning

- **Optimization**
- **Generalization**
- **Delayed consequences**
- **Exploration**

***"Teach by experience"** – design of the world*

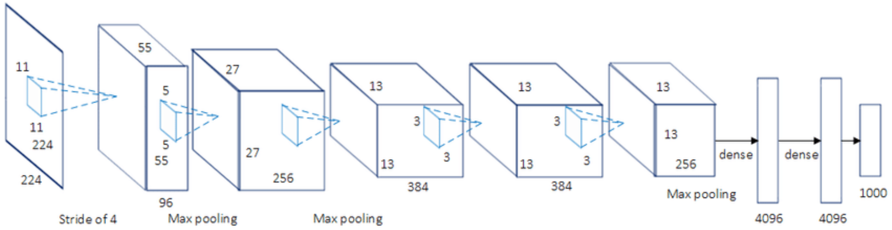


Input

s_t

Network

$a_t = \pi(s_t;\ \theta)$



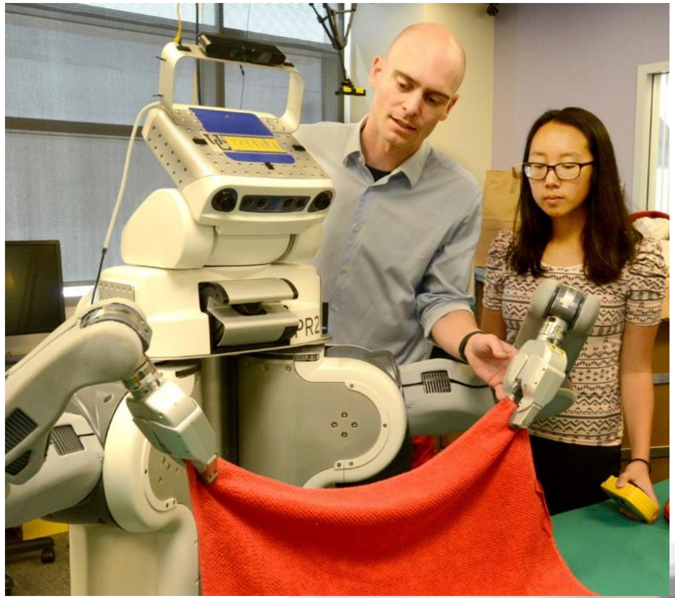
Output

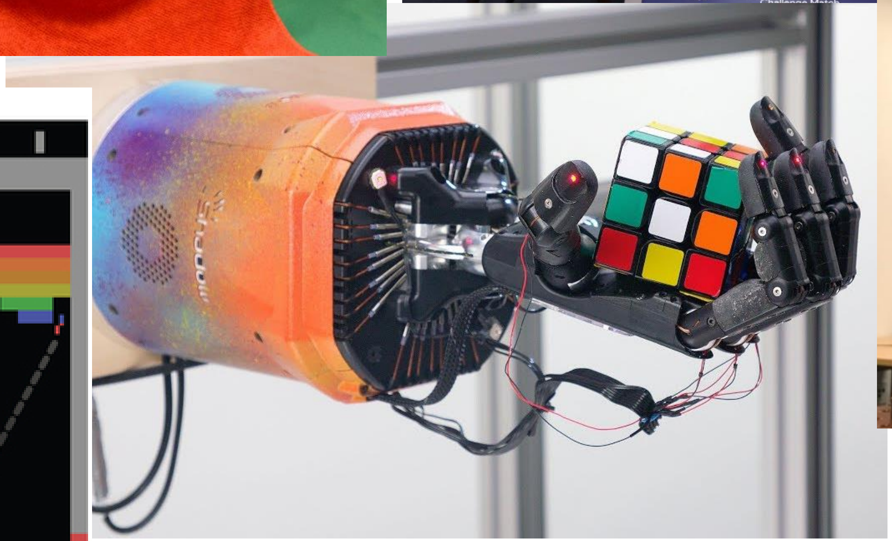
a_t

Data

$\{(s_t, a_t, r_t, s_{t+1})\}$

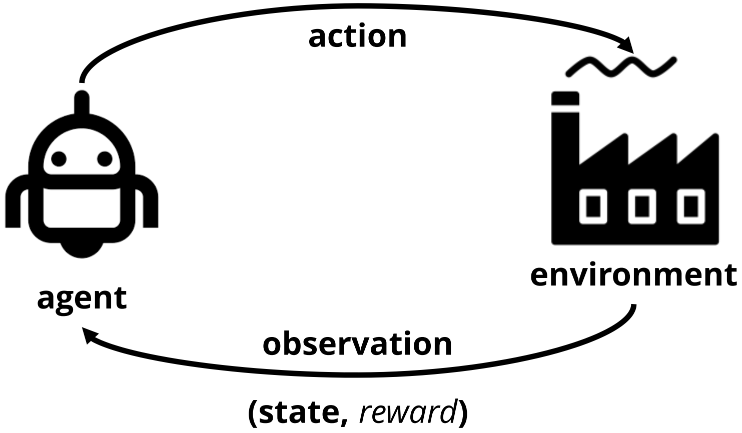
Applications





Interaction Loop

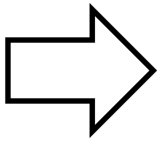
Agent interacts with an environment by following a policy



- The agent contains:
  - Agent state
  - Policy
  - Value function *(probably)*
  - Model *(optionally)*

- At each step t the agent:
  - Receives observation o_t *(and reward r_t)*
  - Executes a_t
- The environment:
  - Receives action a_t
  - Emits observation o_{t+1} *(and reward r_{t+1})*

- Aspire learning to make decisions from interaction …
  - … time;
  - … long-term consequence of actions;
  - … actively gathering experience;
  - … predict the future;
  - … deal with uncertainty



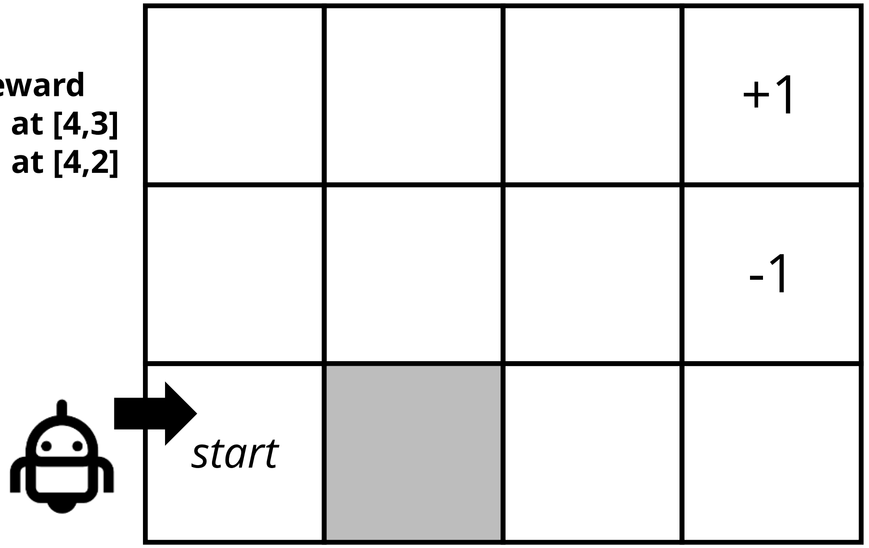
- **Optimization**
- **Generalization**
- **Delayed consequences**
- **Exploration**

Final goal: to learn an optimal policy which maximizes the long-term cumulative rewards

Example #1

Robot labyrinth

Reward
+1 at [4,3]
-1 at [4,2]



Actions

{UP, DOWN, LEFT, RIGHT}

States

Cells

Questions & Options

What’s the strategy to achieve max reward?

- Learn the model and plan
- Learn the value of (action, state) pairs and act greedy/non-greedy
- Learn the policy directly while sampling from it

Deterministic model of the world

Policy:
- **Shortest path**

Stochastic model of the world

Policy:
Reward for each step: **[-2, -0.04]**
- **Shortest path**, however the structure changes; the higher negative reward for each step, the urgency of the agent increases, and vice-versa

Reward for each step: **+**
- **Longest path**

Notation

States & Observations

- A *state* $\boldsymbol{s}$ is a complete description of the state – no hidden information
- An *observation* $\boldsymbol{o}$ is a partial description of a state – omit information
- Both agent and environment may have an internal state

Environment State

- It is the environment's internal state
- *Fully observed* – agent is able to observe the complete state of the environment
- *Partially observed* – agent only see a partial observation

Agent State

- Actions depend on the state
- A history is a sequence of

$$\mathcal{H}_t = o_0, a_0, r_1, o_1, \dots, o_{t-1}, a_{t-1}, r_t, o_t$$

- The history can be used to construct an agent state s_t

Notation

Trajectories

- A trajectory $\boldsymbol{\tau}$ is a sequence of states and actions in the world

$$\tau = (s_0, a_0, s_1, a_1, \dots)$$

- Initial state s_0 randomly sampled from the start-state distribution

$$s_0 \sim \rho_0(\cdot)$$

- State transitions are governed by the laws of the environment, and depend on only the most recent action, $\boldsymbol{a_t}$:

$$deterministic \;\rightarrow\; s_{t+1} = f(s_t, a_t)$$
$$stochastic \;\rightarrow\; s_{t+1} \sim P(\cdot\,|s_t, a_t)$$

Notation

Reward & Return

- A reward r_t is a scalar feedback signal
- Indicates how well the agent is doing at step t
- Depends on the current state of the environment, the action just taken, and the next state of the world

$$r_t = R(s_t, a_t, s_{t+1})$$

- The goal of the agent is to maximize some notion of cumulative reward over a trajectory, so-called **return** $R(\tau)$
  - *Finite-horizon undiscounted return*: sum of rewards obtained in a fixed window of steps

$$R(\tau) = \sum_{t=0}^{T} r_t$$

  - *Infinite-horizon discounted return:* sum of all rewards ever obtained by the agent, but discounted by how far off in the future they are obtained

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t \, r_t, \text{ where } \gamma \in [0, 1]$$

Notation

Value Function

- The value of state (*or state-action pair*) is the expected cumulative reward, from a state s, and then act according to a particular policy π forever after

$$V(s) = \mathbb{E}[R(\tau) \mid s_t = s]$$

- The goal is to maximize value by picking suitable actions
- Rewards and values define desirability of a state or action
- There are four main functions:
  - On-policy value function $V^{\pi}(s) = \mathbb{E}_{\tau \sim \pi}[R(\tau) \mid s_t = s]$
  - On-policy action-value function $Q^{\pi}(s, a) = \mathbb{E}_{\tau \sim \pi}[R(\tau) \mid s_t = s, a_t = a]$
  - Optimal value function $V^{*}(s) = max_{\pi}\mathbb{E}_{\tau \sim \pi}[R(\tau) \mid s_t = s]$
  - Optimal action-value function $Q^{*}(s, a) = max_{\pi}\mathbb{E}_{\tau \sim \pi}[R(\tau) \mid s_t = s, a_t = a]$

Notation

Policy

- Actions may have long term consequences
- Reward may be delayed
- A mapping from states to actions is called a policy – rule used by an agent to decide what actions to take
- *Parameterized policies*, whose ouputs are computable functions that depend on a set of parameters
  - *Deterministic policy -* μ: $a_t = \mu_\theta(s_t) \equiv \pi(s) = a$
  - *Stochastic policy -* π: $a_t \sim \pi_\theta(\cdot|s_t) \equiv \pi(a|s) = p(a_t = a|s_t = s)$

Notation

Model

- Agent's representation of the environment

Transition $\boxed{s_0, a_0, r_1, s_1,}\ a_1, r_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n$

State **Next state** **Terminal State**

Action

Reward

- A model predicts what the environment will do next

$$transition\ function \ \rightarrow P(s', r|s, a) = \ p(s_{t+1} = s', r_{t+1} = r|s_t = s, a_t = a)$$

$$next\ state \ \rightarrow P(s'|s, a) = \ p(s_{t+1} = s'|s_t = s, a_t = a)$$

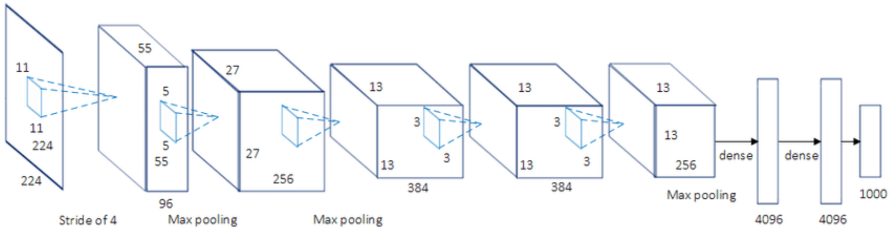
$$next\ reward \ \rightarrow R(s, a, s') \approx \ \mathbb{E}[r_{t+1}|s_t = s, a_t = a, s_{t+1} = s']$$

- A model does not immediately provide a good policy – planning is still needed
- Stochastic (generative) models can be used

Formalizing RL Interaction

$\boldsymbol{s}$

$\boldsymbol{a_t = \pi(a|s)}$



$\boldsymbol{a_t}$

- Which action is better or worse?
- $r(s, a)$: reward function, tell us which states and actions are better
- The mathematical formulation of the agent-environment interaction is called a Markov decision process (MDP)

$$s, a, R(s, a, s'), P(s' |\, s, a)$$

- Suppose the agent sees the full environment state (*fully observable*) $s_t = o_t = environment\ state$
  - Then the agent is in a MDP
- Suppose the agent gets partial information (*partially observable*)
  - Then it is a partially observable MDP (POMDP), where the observation is not Markov, the environment state can still be Markov, but the agent does not know it

MDP Context

Definition

- *"The future is independent of the past given the present"*
- Considering a sequence of random states, $\{S_t\}_{t \in \mathbb{N}}$, indexed by time, a state s has the *Markov* property when for states $\forall s' \in S$ and all rewards $r \in \mathbb{R}$

$$p(r_{t+1} = r, s_{t+1} = s' | s_t = s) = p(r_{t+1} = r, s_{t+1} = s' | s_1, \dots, s_{t-1}, s_t = s), \text{ for all possible } s_1, \dots, s_{t-1}$$

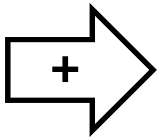
Markov property

- A MDP is a tuple (S, A, P, R, γ), where
  - S is the set of all possible states
  - A is the set of all possible actions
  - $P(s' |\, s,\ a)$ is the conditional probability of next state s', given a state s and action a – defines the dynamics
  - $R(s, a, s')$ is the joint probability of how much reward – defines the reward
  - $\gamma \in [0,1]$ is a discount factor that gives more importance to early rewards – rewards and discount define the goals

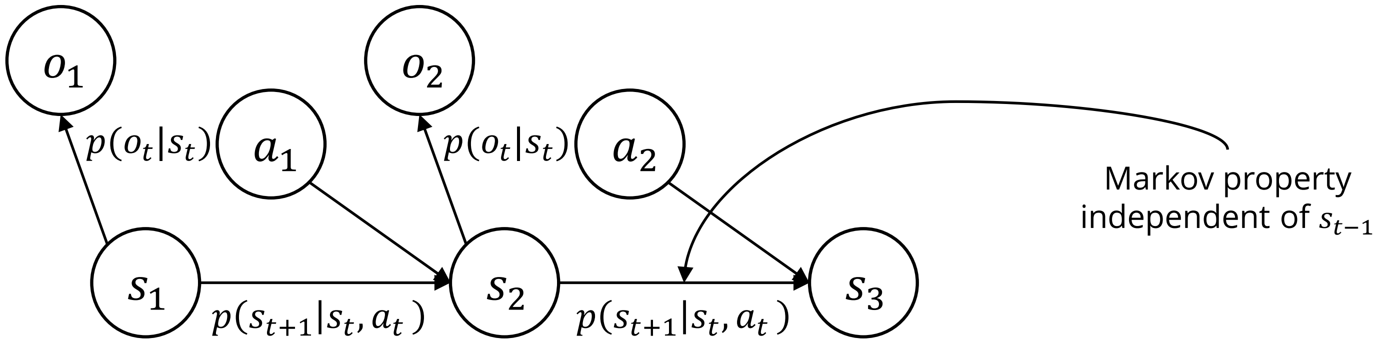
RL as MDP

Definition

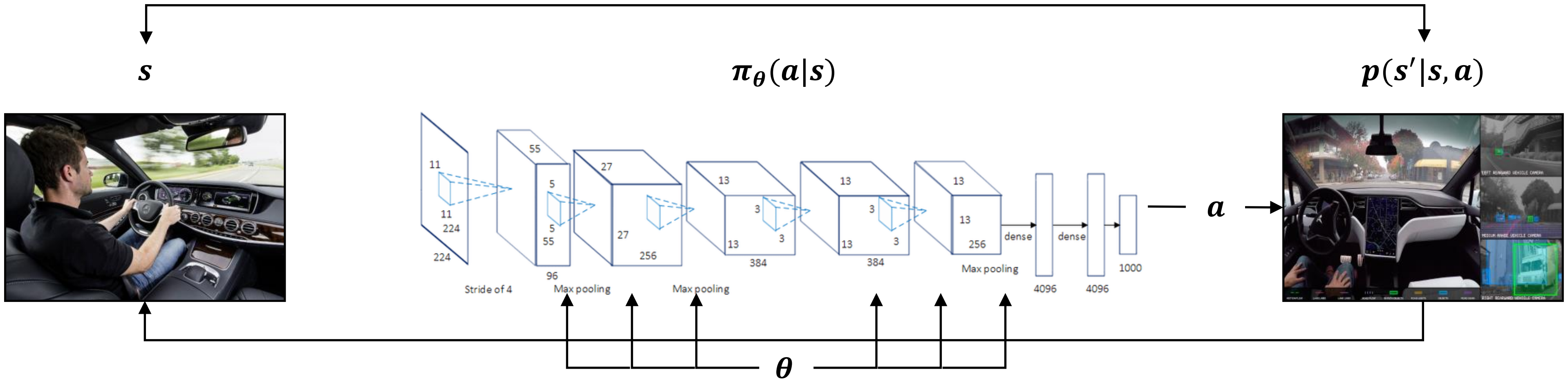
- MDP → $M = \{S, A, P, R, \gamma\}$
  - S – state space $s \in S$ (discrete or continuous)
  - A – action space $a \in A$ (discrete or continuous)
  - P – transition operator (tensor)
  - R – reward function $R: S \times A \rightarrow \mathbb{R}$
  - $\gamma \in [0,1]$ is a discount factor



- POMDP → $M = \{S, A, O, T, \mathcal{E}, r, \gamma\}$
  - O – observation space $o \in O$ (discrete or continuous)
  - $\mathcal{E}$ – emission probability $p(o_t|s_t)$

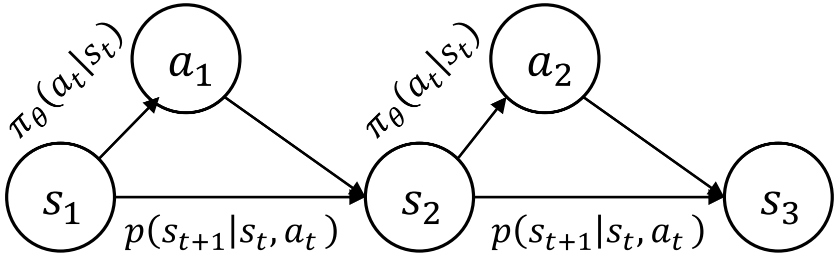


(Deep) RL Problem



- The goal is to select a policy which maximizes expected return when the agent acts according to it
- Assuming that both state transitions and policy are stochastic, the probability of a *T-step trajectory* is

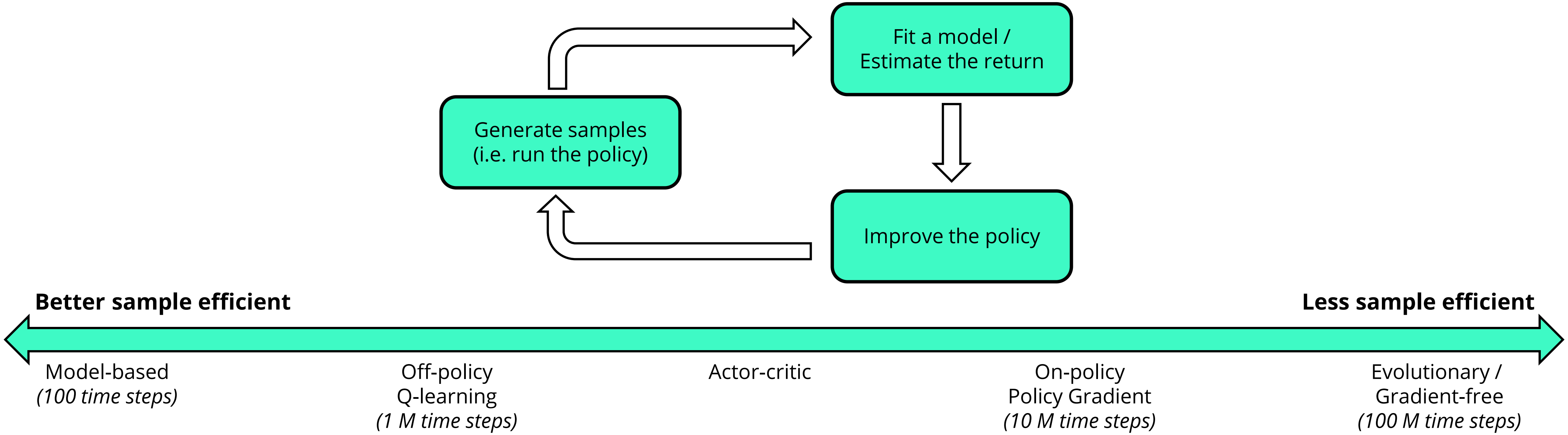
$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} \underbrace{P(s_{t+1}|s_t, a_t)\, \pi(a_t|\, s_t)}_{\text{Markov chain on } (s,a)}$$



- The expected return is given by $J(\pi) = E_{\tau\sim\pi}[R(\tau)]$
- The optimal policy is given by $\pi^* = \underset{\pi}{\operatorname{argmax}}\, J(\pi)$

Types of RL

Anatomy of RL algorithms



Model-based

Estimate the transition model to

- Use it for planning (no explicit policy)
- Use it to improve a policy

Update model and re-plan often

Value-based

Estimate value function or Q-function of the optimal policy (no explicit policy)

Act by using best action in state

Exploration is a necessary add-on

Actor-critic

Estimate value function or Q-function of the current policy

Use it to improve policy

Policy-based

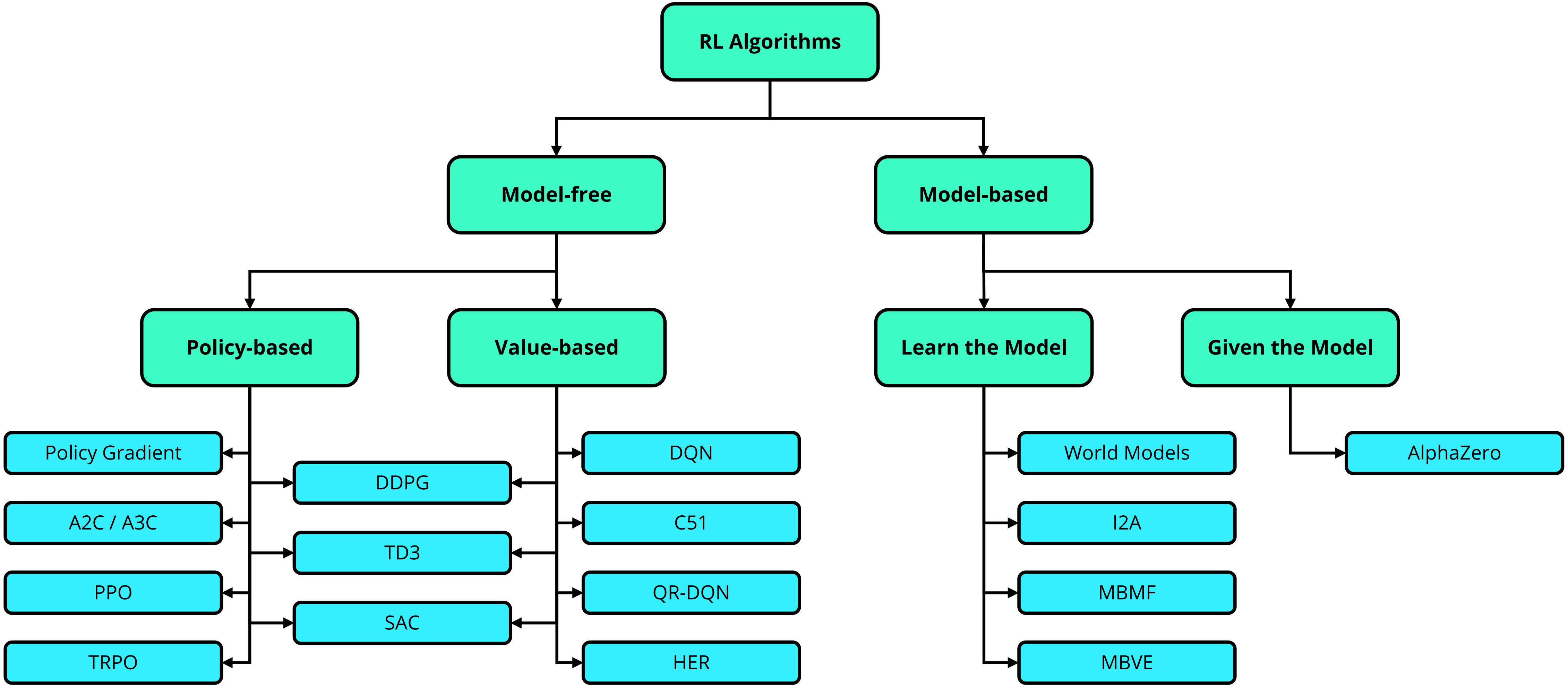
Learn the stochastic policy function that maps state-to-action

Act by sampling policy

Exploration is baked-in

Taxonomy of RL

Different methods of RL



From: Spinningup - Openai

Challenges of (Deep) RL

Learning the components of an agent

- All components are functions
  - Policies map states to actions
  - Value functions map states to values
  - Models map states to states and/or rewards
  - State updates map states and observations to new states
- These functions can be represented as <u>neural networks</u>

challenges

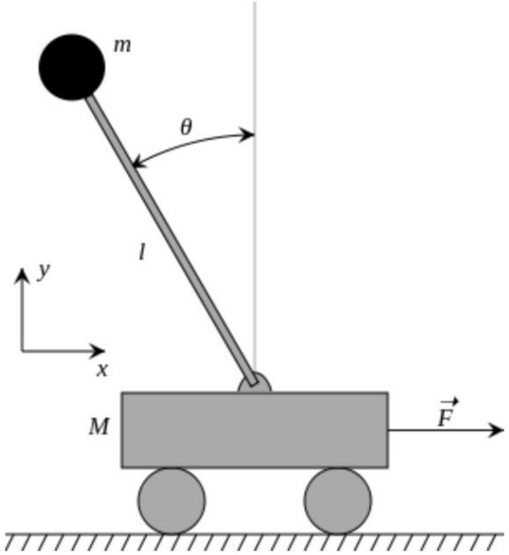
- Learning
  - The environment is initially unknown
  - The agent interacts with the environment
- Planning
  - A model of the environment is given
  - The agent plans in this model
- Prediction
  - Evaluate the future for a given policy
- Control
  - Optimize the future, finding the best policy

Example #2

Cartpole

Observation

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	-41.8°	41.8°
3	Pole Velocity At Tip	-Inf	Inf



Actions

Num	Action
0	Push cart to the left
1	Push cart to the right

Description

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity.

Reward

Reward is 1 for every step taken, including the termination step. The threshold is 475 for v1.

Starting State

All observations are assigned a uniform random value between ±0.05.

Episode Termination

- Pole Angle is more than ±12°
- Cart Position is more than ±2.4 (center of the cart reaches the edge of the display)
- Episode length is greater than 200 (500 for v1).

Bellman Equations

Goal: solve MDP by finding the optimal policy and value functions

- Bellman equations decompose the value function

$$\boldsymbol{v} = \underset{\substack{\uparrow \\ \text{immediate reward}}}{\boldsymbol{r}} + \underbrace{\gamma \boldsymbol{P}^{\pi} \boldsymbol{v}}_{\text{Discounted sum of future rewards}}$$

- The Bellman expectation equations are given by

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s)\left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s')\right)$$

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q^{\pi}(s',a')$$

- The optimal values are given by

$$V^{*}(s) = max_{a \in A}\left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{*}(s')\right)$$

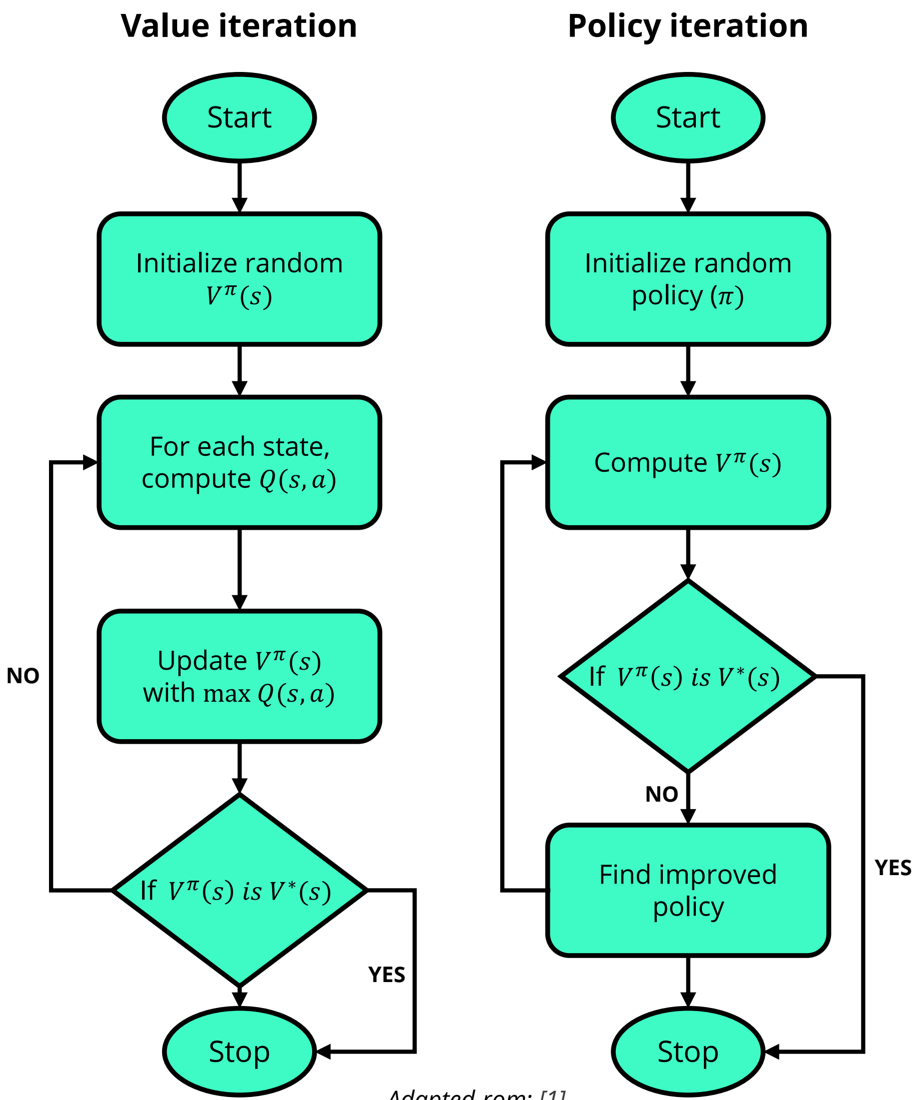
$$Q^{*}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) max_{a' \in A} Q^{*}(s',a')$$

- Optimal policy

$$\pi^{*}(s) = argmax_{\pi} V^{\pi}(s)$$

- The relative advantage of an action (compared to others on average) is

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$



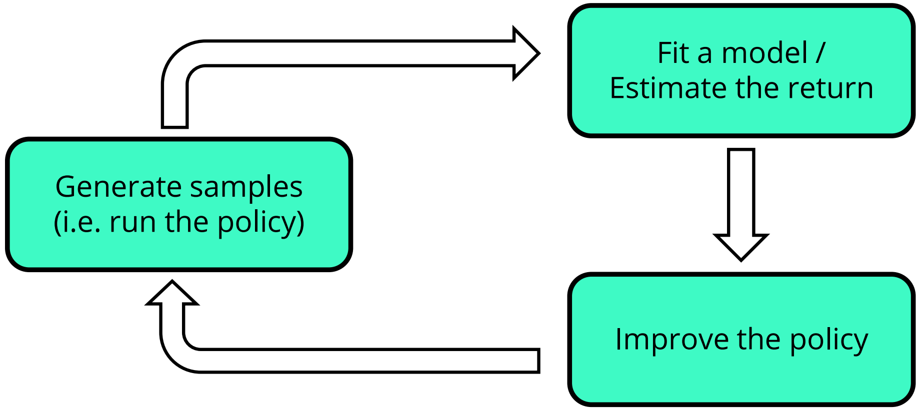
Adapted rom: [1]

Model-based RL

You own the (real or simulated) representation of the environment you are in

Have a representation of

$M = \{S, A, P, R, \gamma\}$



Learn $P(s_{t+1}|s_t, a_t)$

Several approaches

Goal: follow Bellman equations to iteratively evaluate value functions and improve policy

- Planning problem: policy iteration is generally more efficient than enumeration

  Set $i = 0$

  Initialize $\pi_0(s)$ *randomly for all states* s

  While $i == 0$ *or* $\| \pi_i - \pi_{i-1} \|_1 > 0$:

  V^{π_i} $\leftarrow$ *MDP V function* ***policy evaluation*** *of* π_i

  π_{i+1} $\leftarrow$ ***Policy improvement***

  $i = i + 1$

$$\boldsymbol{\pi_0} \xrightarrow[\textbf{evaluation}]{} \boldsymbol{V^{\pi_0}} \xrightarrow{\textbf{improvement}} \boldsymbol{\pi_1} \xrightarrow[\textbf{evaluation}]{} \boldsymbol{V^{\pi_1}} \xrightarrow{\textbf{improvement}} \boldsymbol{\pi_2} \xrightarrow[\textbf{evaluation}]{} \boldsymbol{\dots} \xrightarrow{\textbf{improvement}} \boldsymbol{\pi_*} \xrightarrow[\textbf{evaluation}]{} \boldsymbol{V^*}$$

- Policy evaluation

$$V^{\pi_i}(s) = R\big(s, \pi_i(s)\big) + \gamma \sum_{s' \in S} P(s'|s, \pi_i(s)) V^{\pi_{i-1}}(s')$$

- Policy improvement

$$Q^{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^{\pi_i}(s')$$

$$\pi_{i+1} = argmax_a Q^{\pi_i}(s, a)$$

- Monotonic improvement in policy

$$V^{\pi_1} \geq V^{\pi_2}: V^{\pi_1}(s) \geq V^{\pi_2}(s), \forall s \in S$$

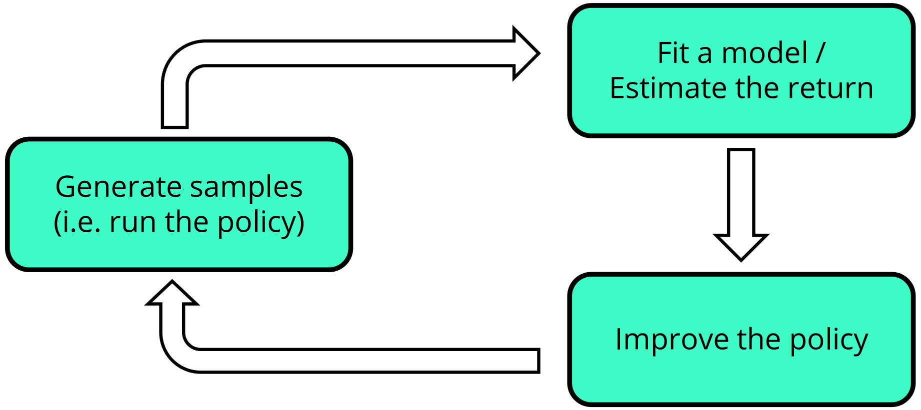
$$Q_\pi\big(s, \pi'(s)\big) = Q_\pi\big(s, argmax_a Q_\pi(s, a)\big)$$

$$= max_a Q_\pi(s, a) \geq Q_\pi\big(s, \pi(s)\big) = V_\pi(s)$$

Model-based RL

You own the (real or simulated) representation of the environment you are in

Have a representation of
$M = \{S, A, P, R, \gamma\}$



Learn $P(s_{t+1}|s_t, a_t)$

Several approaches

Analytic gradient computation

- Assumptions about the form of the dynamics and cost function
    - LQR framework, Receding-horizon control, other analytic gradient-based approaches used for policy improvement

Sampling-based planning

- Generate sampling distributions of action sequences *(continuous space)* or to search over tree structures *(discrete space)*
    - Random shooting, cross-entropy method, path integral optimal control, etc

Model-based data generation

- Increase size of training set for policy optimization
    - Dyna, iLOG, meta-learning, etc

Model-free RL

You don't know the dynamics and reward models

No knowledge of MDP is required, just samples

$$V^{\pi}(s) = \sum_{a\in A} \pi(a|s)(R(s,a) + \gamma \sum_{s'\in S} \boxed{P(s'|s,a)} V^{\pi}(s'))$$

Unknown!!

Value-based

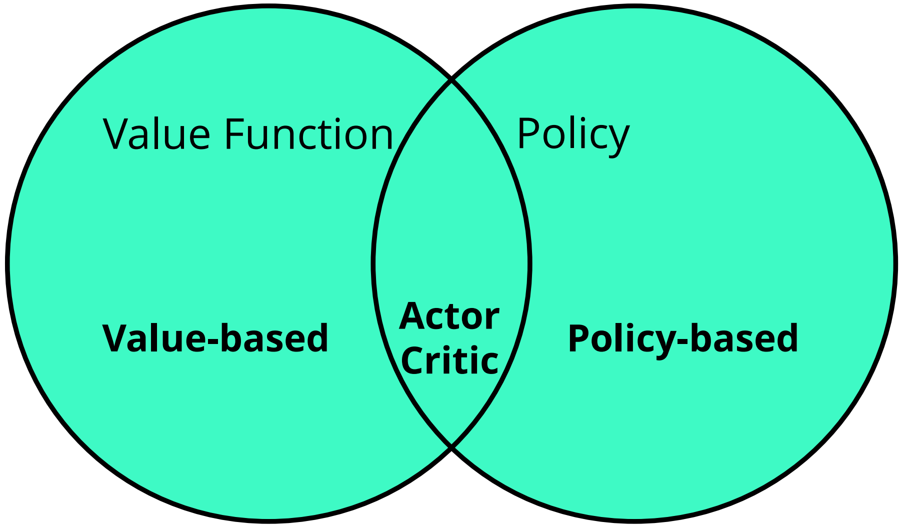
- Learn value function
- Implicit policy (e.g. $\epsilon - greedy$)

Policy-based

- No value function
- Learnt policy

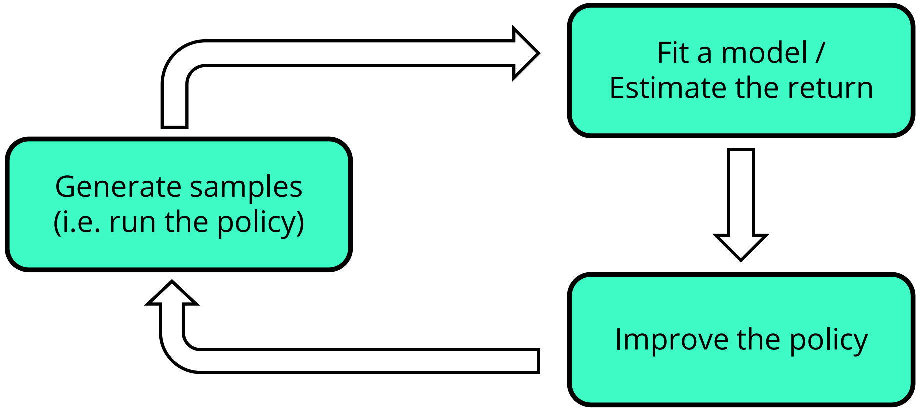
Actor-Critic

- Learnt value function
- Learnt policy



Model-free RL

Direct policy gradients



Evaluate returns
$\boldsymbol{R(\tau) = \sum_{t=0}^{T} r(s_t, a_t)}$

$$\boldsymbol{\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}[\sum_{t=0}^{T} r(s_t, a_t)]}$$

Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find best θ

- The quality of a policy π_θ can be measured in
  - Episodic environments by the start value
    $$J_0(\theta) = V^{\pi_\theta}(s_0)$$
  - Continuing environments by the average value
    $$J_{avV}(\theta) = \sum_{s \in S} \mu^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

  or by the average reward per time-step

$$J_{avR}(\theta) = \sum_{s \in S} \mu^{\pi_\theta}(s) \sum_{a \in A} \pi(s) \sum_{r \in R} p(r|s, a)\, r$$

Optimization: find θ that maximizes $J(\theta)$

- No gradient-based
  - Hill climbing
  - Genetic algorithms
- Gradient based (stochastic gradient ascent)
  - Search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t parameters θ
    $$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

  where $\nabla_\theta J(\theta)$ is the policy gradient approximated by

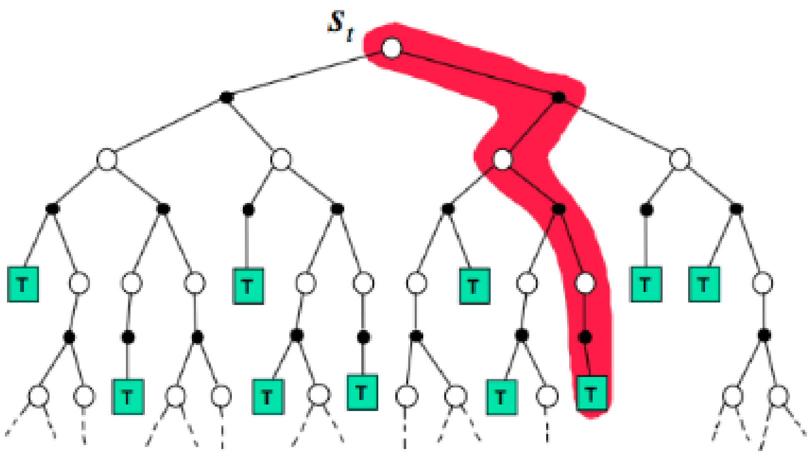
$$\nabla_\theta J(\theta) \propto \sum_{s \in S} \mu(s) \sum_{a \in A} Q_\pi(s, a) \pi(a|s; \theta) = \mathbb{E}^{\pi_\theta}[\nabla log \pi(a|s, \theta) Q_\pi(s, a)]$$

Policy Evaluation Methods

Learning methods

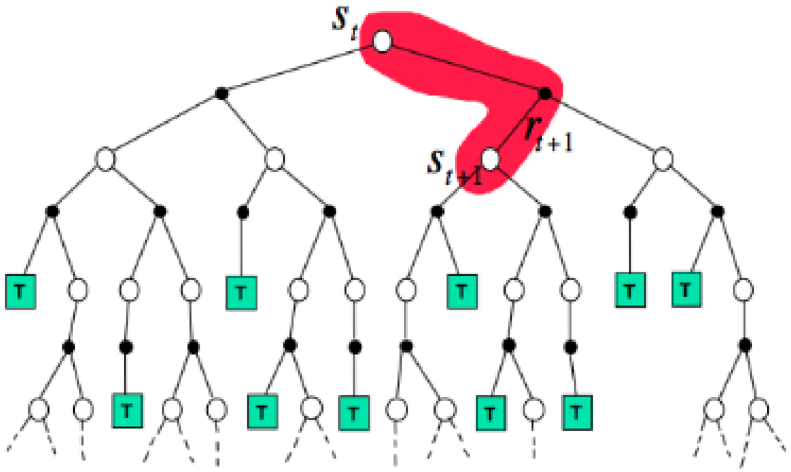
Monte-Carlo (MC)

$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha(R(\tau)_t - V^{\pi}(s_t))$$



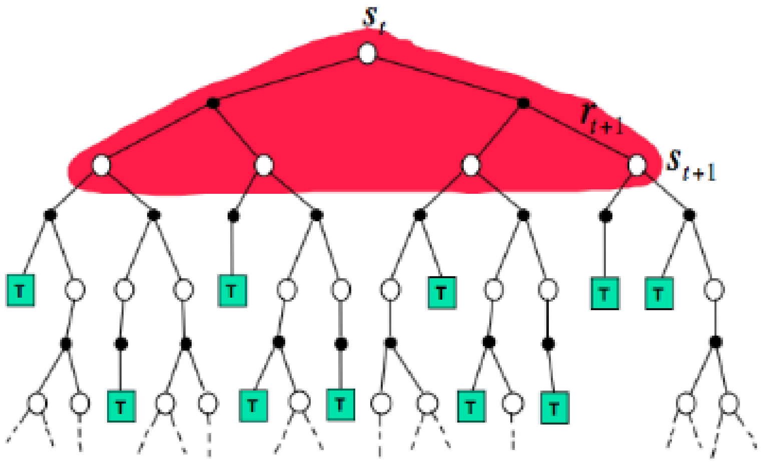
Temporal-Difference (TD)

$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha(R_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t))$$



Dynamic Programming (DP)

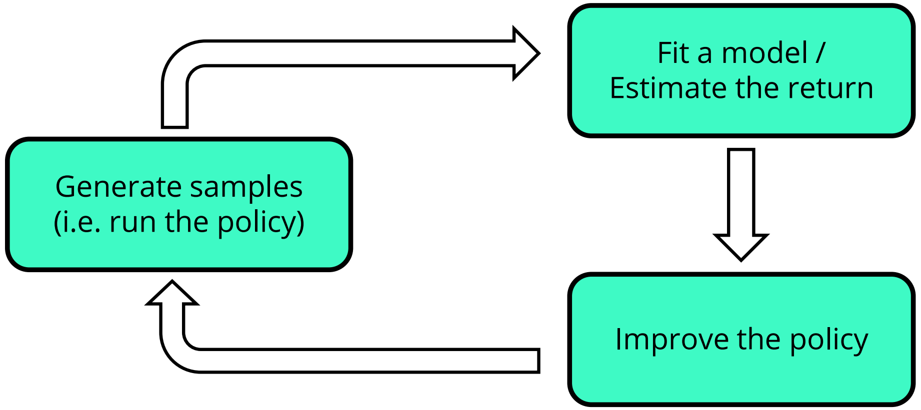
$$V^{\pi}(s_t) = [R_{t+1} + \gamma V^{\pi}(s_{t+1})]$$



From: David Silver's RL course

Model-free RL

Value functions



Fit $V(s)$ or $Q(s,a)$

Set $\pi(s) = argmax_a Q(s,a)$

Goal: measure the (approximated) goodness of a state or how regarding a state or an action is by predicting the future reward

- The objective function is based on the Bellman equation

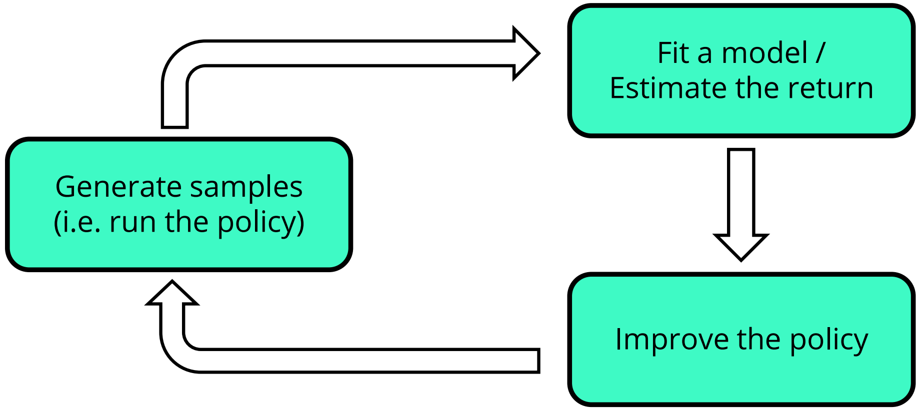
$$\boldsymbol{v} = \boldsymbol{r} + \gamma \boldsymbol{P}^{\pi} \boldsymbol{v}$$

- <u>Monte-Carlo</u>-based (MC) approach
  - Computes the observed mean return as an approximation of the expected return
  - Learns from complete episodes
- <u>Temporal-Difference</u>-based (TD) approach
  - Uses bootstrapping to update $V(s)$ towards an estimated return target
  - Learns from incomplete episodes

- Problems of value functions for large MDPs:
  - There are too many states and/or actions to store
  - It is too slow to learn the value of each individual state
  - Individual states are often not fully observable
- Function approximation:
  - Generalize from seen states to unseen states
  - Update parameter θ using TD or MC learning
  - For a partial observable environment, the agent state or a learning state update function can be used
  - ANN, decision tree, NN, fourier / wavelet bases, etc

Model-free RL

Actor-critic: value functions + policy gradients



Fit $\boldsymbol{V(s)}$ or $\boldsymbol{Q(s,a)}$

$$\boldsymbol{\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}[Q(s_t, a_t)]}$$

Goal: learn value function in addition to the policy

Critic : update value function parameters w by (n-step) TD

Actor: update policy parameters θ by policy gradient

- Considering the policy-based gradient

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)\boxed{R(\tau^i)} \longrightarrow \boldsymbol{R(\tau^i) - b(\tau^i)}$$

- Improving high variance of gradients and cumulative reward of 0

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T} \nabla_\theta \underbrace{\boxed{log\pi_\theta(a_t|s_t)}}_{\textbf{Actor}} \underbrace{\underbrace{\boxed{(Q(s_t, a_t) - V_\phi(s_t))}}_{\textbf{Critic}}}_{\textbf{advantage value}}$$

- The **critic** estimates the value function, could be the action-value (*Q* value) or state-value (*V* value)
- The **actor** updates the policy distribution in the direction suggested by the critic (through policy gradients)

Algorithm 1 Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \dots T$: **do**
  Sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a)\nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s,a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a)$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Exploration *vs.* Exploitation

We learn by trial and error

Policy

- Online decision-making involves a fundamental choice to discover a good policy:
  - Exploration – increase knowledge
  - Exploitation – maximize performance based on current knowledge
- The best long-term strategy may involve short-term sacrifices – enough information to make the best overall decision

- $\epsilon - greedy$ policy:
  - Greedy can lock onto a suboptimal action forever
  - $\epsilon - greedy$ continues to explore forever
    - Don't know anything about the environment at the beginning
    - Need to try all actions to find the optimal one
    - With probability $(1-\epsilon)$ select $a = argmax_{a \in A} Q_t(a)$
    - With probability ϵ select a random action

-

Off-policy RL

Q-Learning, an off-policy TD control

Goal: how to act optimally in controlled Markovian domains, by successively improving the evaluations of the quality of particular actions at particular states

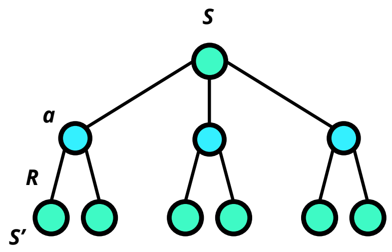
- The optimization is almost always off-policy and the corresponding policy is obtained via the connection between Q^* and π^*

$$a(s) = argmax_a \, Q_\theta(s, a)$$

- Independent of the policy being followed
- $\epsilon - greedy$ is commonly applied
- Only requirement: keep updating each pair (s, a)

- Classes of function approximation
  - Tabular – a table with an entry for each MDP state
  - State aggregation – partition environment states
  - Linear function approximation – fixed features/kernel
  - <u>Differentiable (non-linear) function approximation - NNs</u>

Deep Q-Network



	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

Initialize the *Q'* table with random values

1. Choose an action *a* to perform in the current state, *s*
2. Perform *a* and receive reward *r*
3. Observe the new state, *s'*
4. Update:

learning rate **discount factor** **estimate of optimal future state**

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(\underbrace{R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a)}_{\textbf{learned value}} - Q_t(s_t, a_t) \right)$$

new state **old state** **reward**

$s = s'$

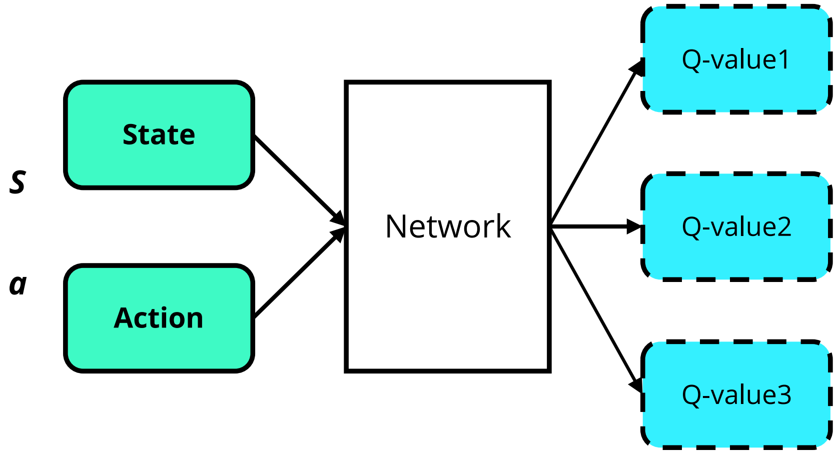
5. If the next state is not terminal, go back to step 1

Off-policy RL

Deep Q-Network, improving Q-Learning

Goal: learn an approximator $Q_\theta(s,a)$ for the optimal action-value function $Q^*(s,a)$

- Q-Learning suffer from
  - Memorization explosion
  - Instability and divergence when combined with a nonlinear Q-value and bootstrapping
- Deep Q-Network improvements
  - Convolutional layers: to consider compact / structural representations
  - Experience replay: storing episode steps into one replay memory D_t, and then randomly drawing batches of them to improve data efficiency and distribution
  - Separate target network: second network to generate the target Q-values that are periodically updated to compute the loss for every action during training



Initialize replay memory *D* to capacity *N*, action-value *Q* function with random weights θ, and target action-value $\hat{Q}$ with weights $\theta^- = \theta$

1. Gather and store samples in a replay buffer with current policy
2. Random sample batches of experiences from the replay buffer
3. Use the sampled experiences to update the Q-network

$$L(\theta_i) = \mathbb{E}_{(s,a,r,s')}[(y_i - \overbrace{Q(s,a;\,\theta_i)}^{\textbf{prediction}})^2]\text{, where}$$

$$y_i = \begin{cases} R_T & \text{for terminal state } s_t \\ \underbrace{R_{t+1}}_{\textbf{reward}} + \overset{\textbf{decay}}{\gamma}\, \underbrace{max_{a'} Q(s_{t+1}, a'; \theta^-)}_{\textbf{target}} & \text{for nonterminal } s_t \end{cases}$$

4. Repeat 1-3

General Conclusions

Model-based

+ *'Easy'* to learn a model (supervised learning)
+ Learns *'all there is to know'* from the data

– Objective captures irrelevant information
– May focus on computing irrelevant details
– Computing policy (i.e. planning) is non-trivial
– Computing policy can be computationally expensive

Value-based

+ Closer to true objective
+ Fairly well understood

– Still not the true objective

Policy-based

+ Right objective
+ Better convergence properties
+ Effective in high-dimensional or continuous action spaces
+ Can learn stochastic policies

– Ignores other learnable knowledge
– Typically converge to a local rather than global optimum
– Evaluating a policy is normally inefficient and high variance is present

Challenges in Deep RL

Final thoughts and research directions

Core algorithms

- Stability – does your algorithm converge? (how many runs consistently)
    - Minimization of fitting error
    - Optimization on true objective
- Efficiency – how long does it take to converge? (how many samples)
    - Impact of off-policy *vs.* on-policy
- Generalization – after it converges, does it generalize?

Core assumptions

- Is this the right problem formulation? (better fit or more practical)
- What is the source of supervision? (efficient source)

Thank you!
Any question?

Eduardo Marques Pereira
edupereira@deloitte.pt