Univ Gustave Eiffel - Cosys/Grettia

Reinforcement Learning & Optimal Control

Advanced Topics in RL

Nadir Farhi

chargé de recherche, UGE - Cosys/Grettia

nadir.farhi@univ-eiffel.fr

ENSIA - 23 April 2025

Outline

1- Model-based RL

- Introduction
- Dyna-Q (review)
- World models

2- Imitation learning

(a) Directly learning policy

- Behavior Cloning
- DAgger algorithm

(a) Indirectly learning policy (Inverse RL)

- Apprenticeship learning

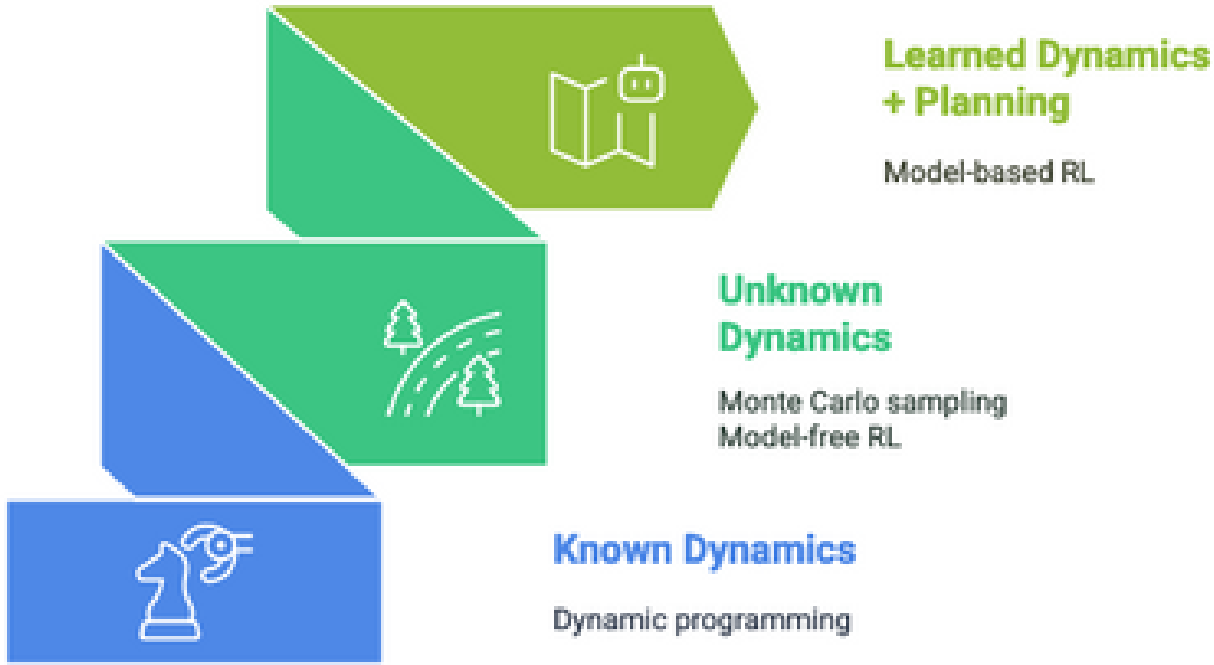
3- Muti-agent RL

4- Hierarchical RL

1 - Model-based RL

Model-based RL

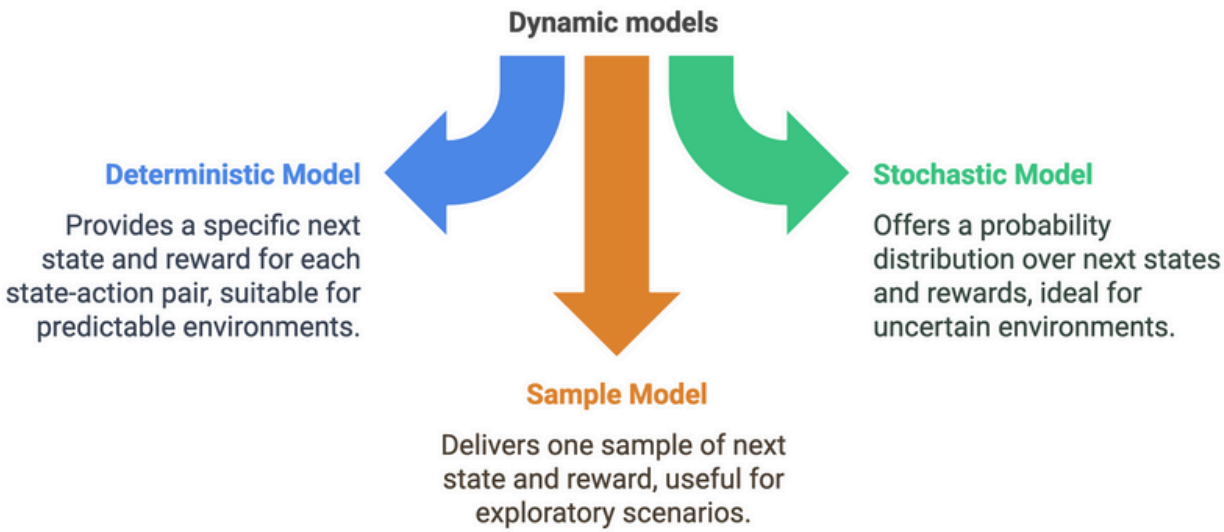
Planning and Learning with Tabular Methods



Model-based RL

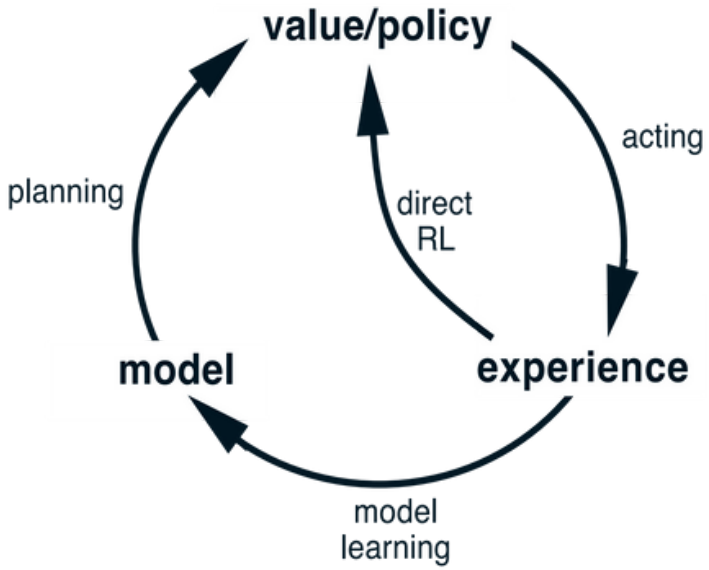
Planning and Learning with Tabular Methods

- Model-free methods rely on learning.
- Model-based methods rely on planning.
- Both approaches compute value functions.



Leanring & Planning

- Learning uses real experience generated by the environment.
- planning uses simulated experience generated by a model.
- Planning: model → new or improved policy.
- Dyna: Integrated Planning, Acting, and Learning
  - Tabular method
  - Deterministic environment



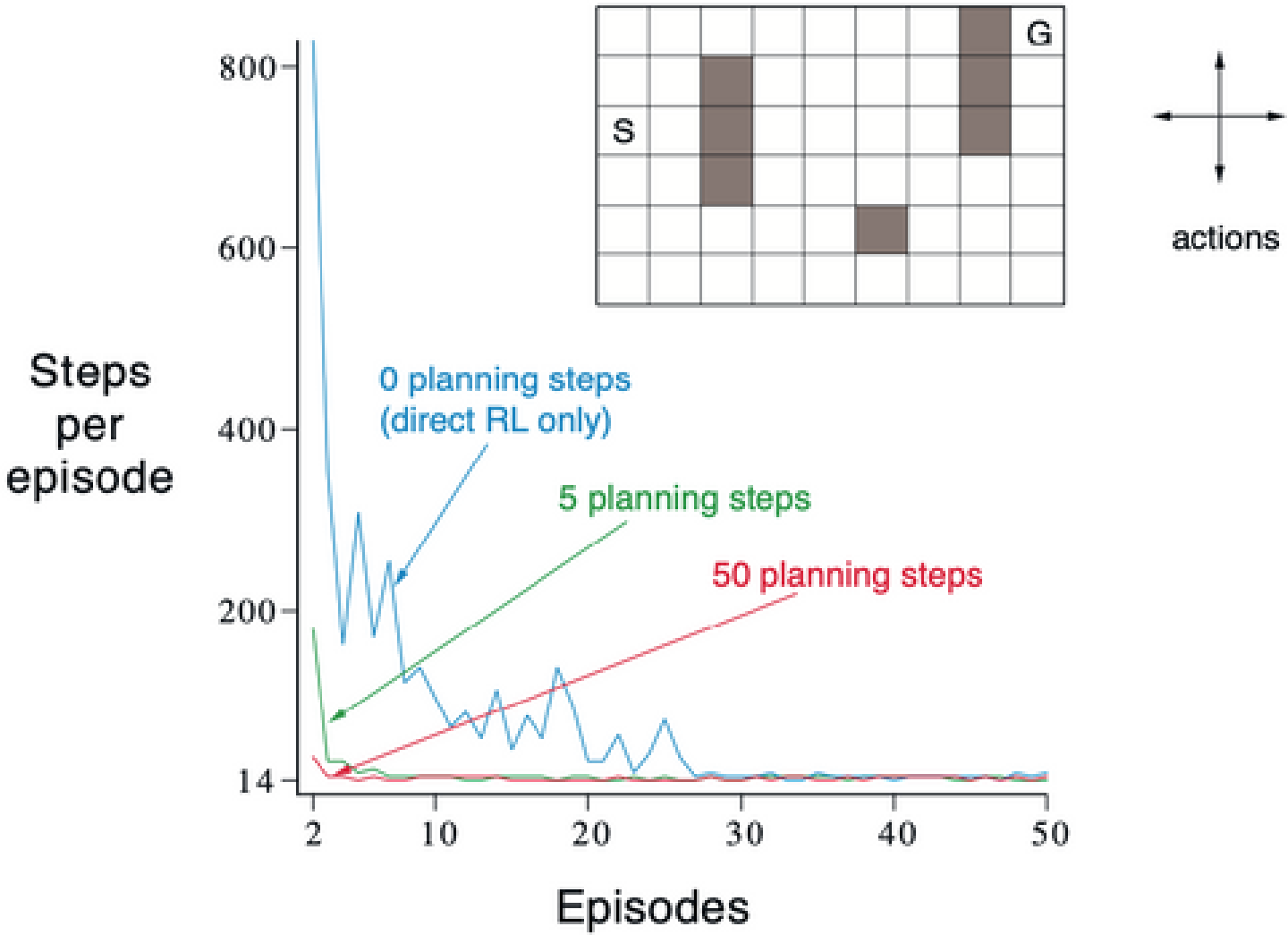
Tabular Dyna-Q

Tabular Dyna-Q

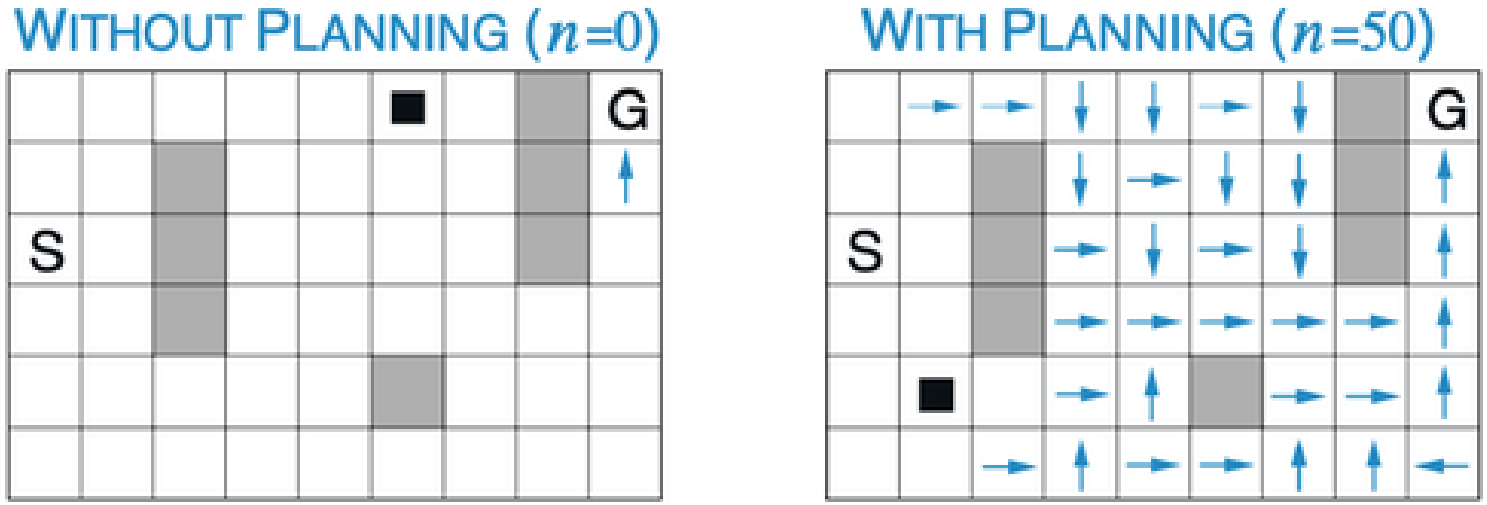
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \varepsilon$-greedy(S,Q)
- (c) Take action A; observe resultant reward, R, and state, S'
- (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
- (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Loop repeat n times:
  - $S \leftarrow$ random previously observed state
  - $A \leftarrow$ random action previously taken in S
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

(d) : direct RL (Q-learning)
(e) : model learning
(f) : planning

Simple maze S → G with Dyna-Q



Dyna-Q planning

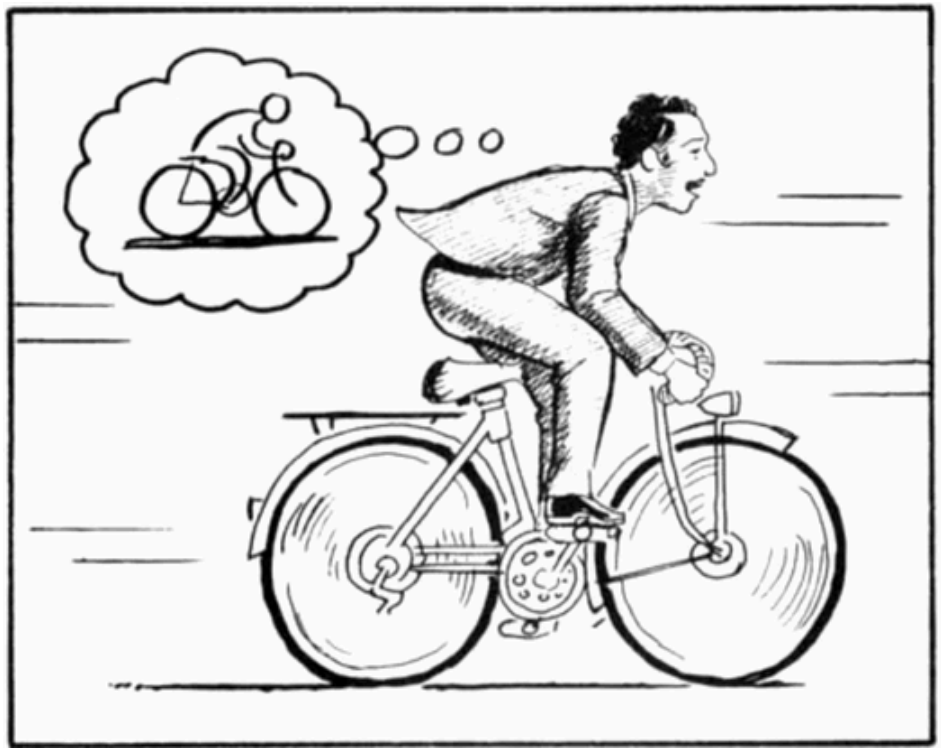


Without planning (n = 0): Each episode adds only one additional step to the policy

With planning : An extensive policy has been developed
By the end of the episode, it will reach almost back to the start state.

World Models

Can agents learn inside of their own dreams?



"The image of the world around us, which we carry in our head, is just a model. Nobody in his head imagines all the world, government or country. He has only selected concepts, and relationships between them, and uses those to represent the real system."

Jay Wright Forrester, , the father of system dynamics.

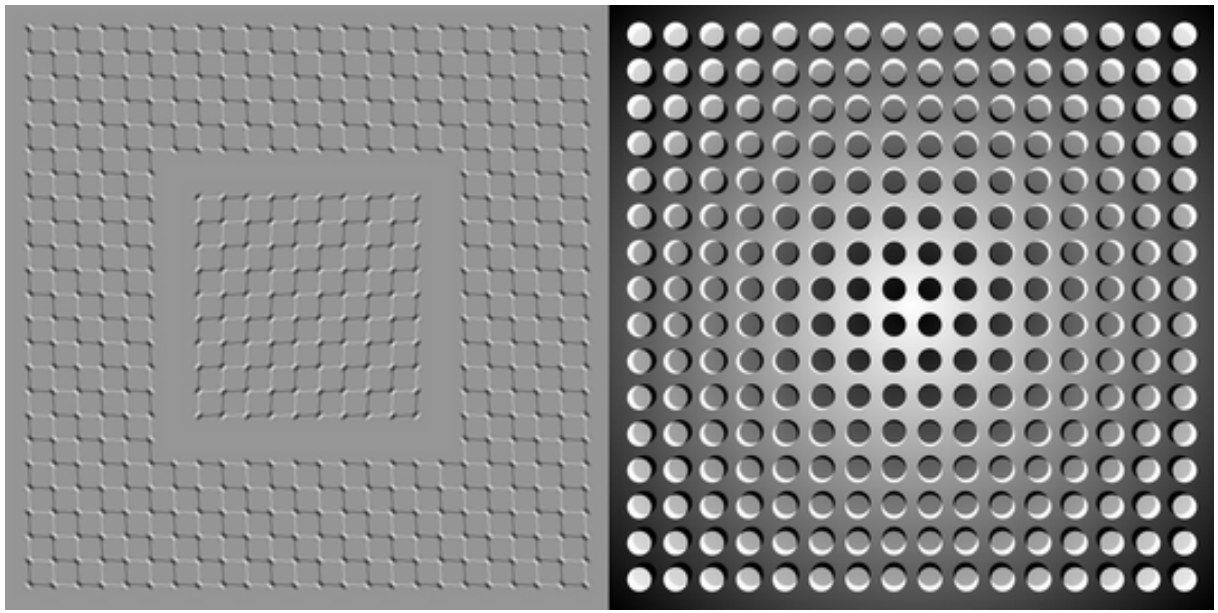
World models

Internal model, prediction & perception

Internal model:

- Our brain learns an abstract representation of both spatial and temporal aspects.
- observe a scene → Remember an abstract description.

Internal model → Prediction of the future → Our perception.



E.g. when we face a danger → Act on internal model → Perform fast reflexive behavior → without need to plan out a course of action.

World models

World model & controller model

Most existing model-based RL approaches:

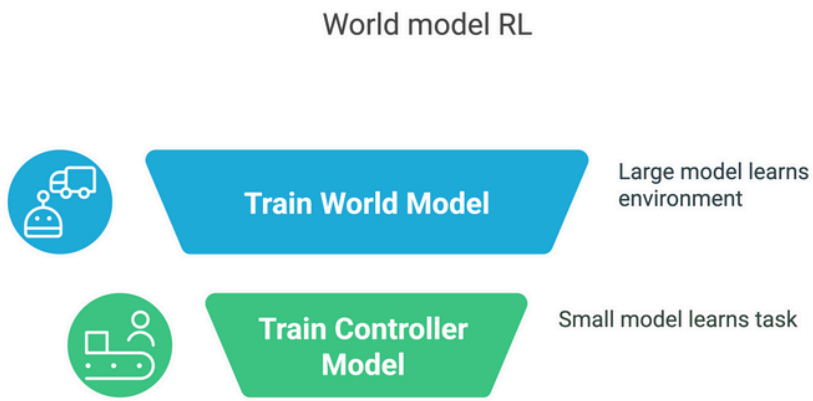
==> Learn a model of the RL environment, but still train on the actual environment.

World models divide a RL agent: large world model and small control model.

1- Train a large NN to learn a world model in an unsupervised manner.

2- Train the smaller controller model to learn a task using this world model.

3- Transfer this policy back into the actual environment.



World models

The agent model

At each time step, our agent receives an **observation** from the environment.

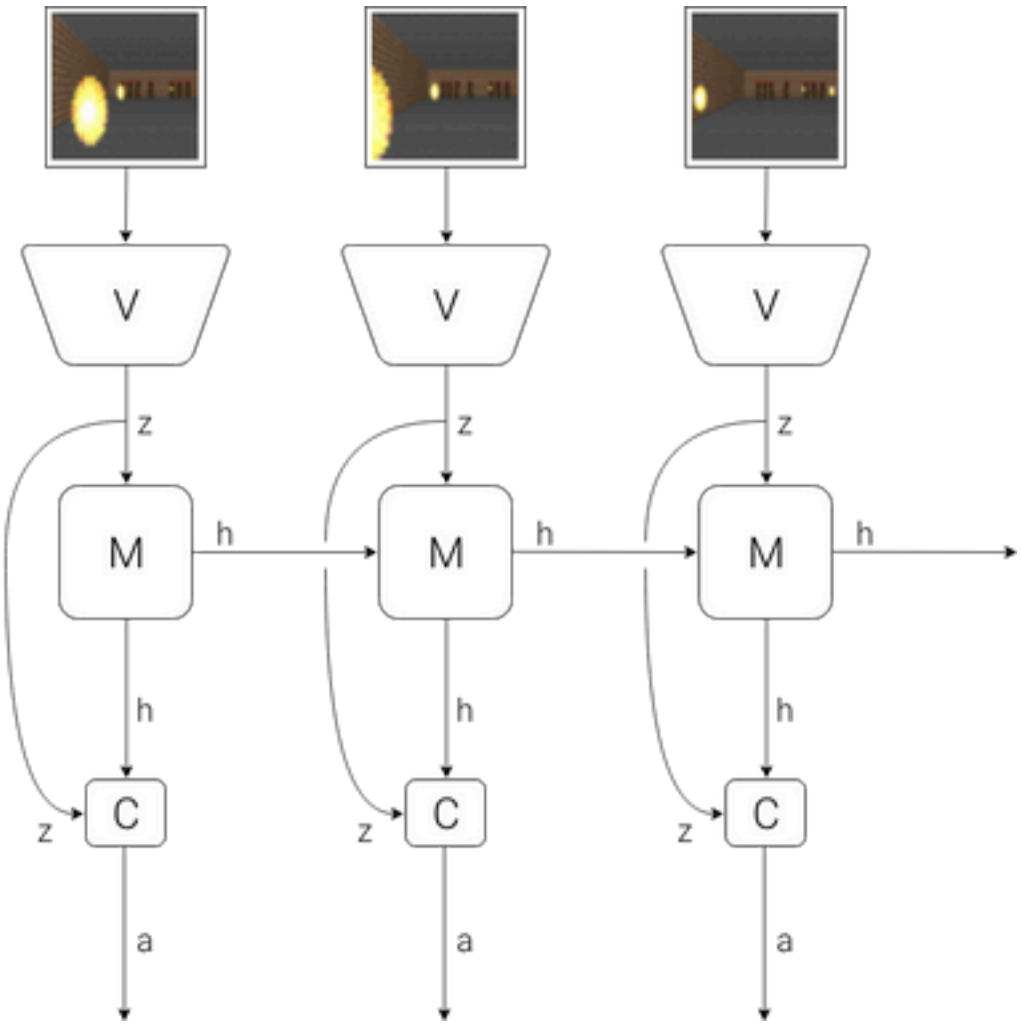
World Model

The **Vision Model (V)** encodes the high-dimensional observation into a low-dimensional latent vector.

The **Memory RNN (M)** integrates the historical codes to create a representation that can predict future states.

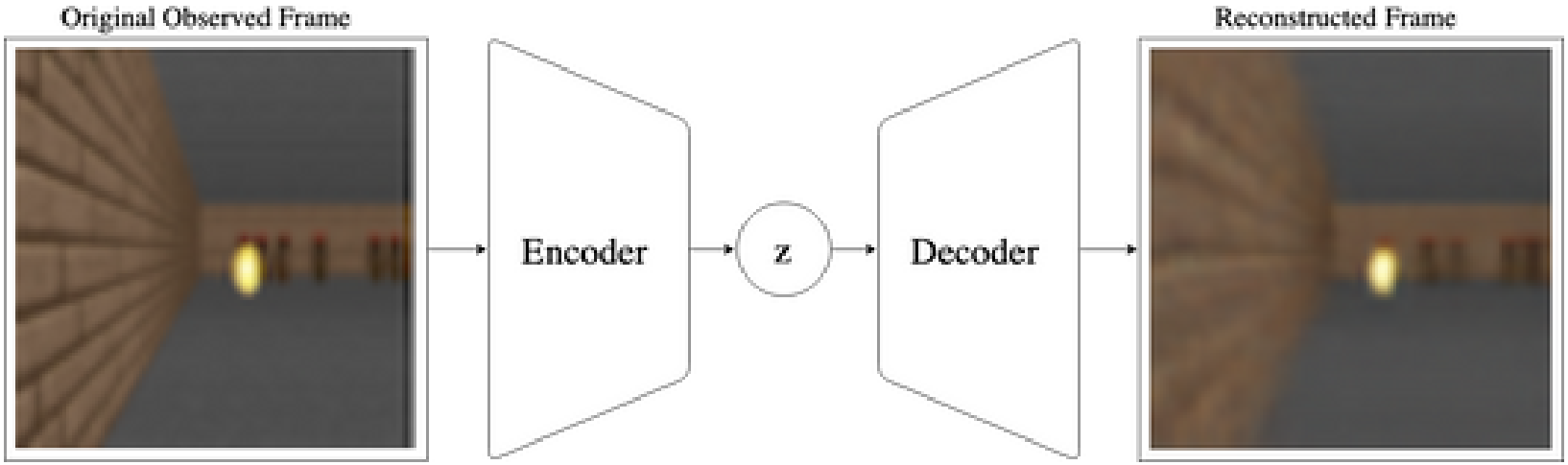
A small **Controller (C)** uses the representations from both **V** and **M** to select good actions.

The agent performs **actions** that go back and affect the environment.



World models

The V Model (vision)

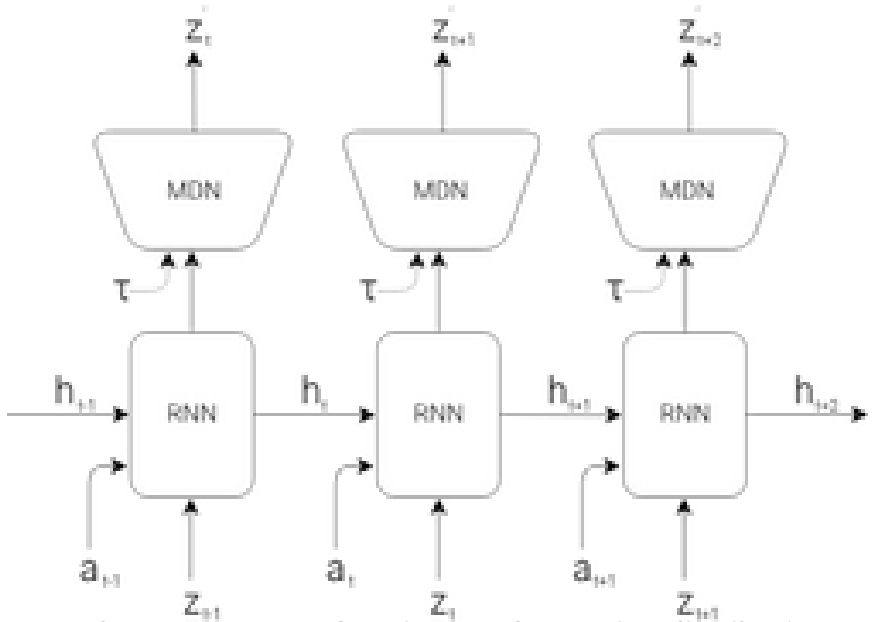


Flow diagram of a Variational Autoencoder.

- Environment → high dimensional observation (2D image frame of a video sequence)
- V model → learns an abstract compressed representation of observed input frame.

World models

The M model (memory RNN)

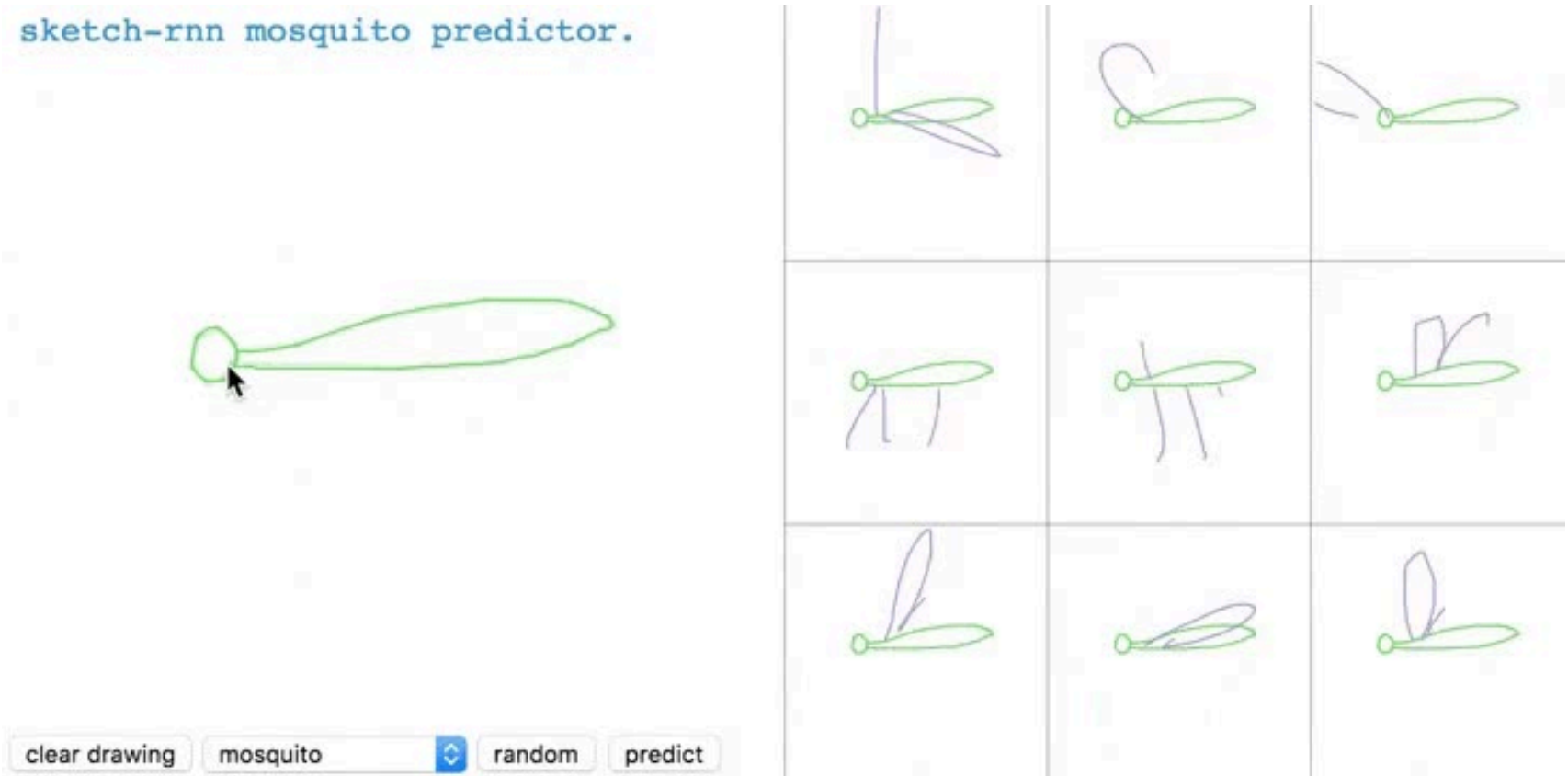


The Mixture Density RNN outputs the parameters of a mixture of Gaussian distribution used to sample a prediction,

- The M model serves as a predictive model of the future z vectors that V is expected to produce.
- The RNN outputs a probability density function (complex environments are stochastic in nature).
- The RNN models P(z_t+1 / a_t, z_t, h_t), where h_t is the hidden state of the RNN at time t.
- During sampling, we can adjust a temperature parameter ττ to control model uncertainty.

World models

SketchRNN [1]



[1] Draw Together with a Neural Network, D. Ha, J. Jongejan, I. Johnson. Google AI Experiments. 2017.

World models

The C model (Controller)

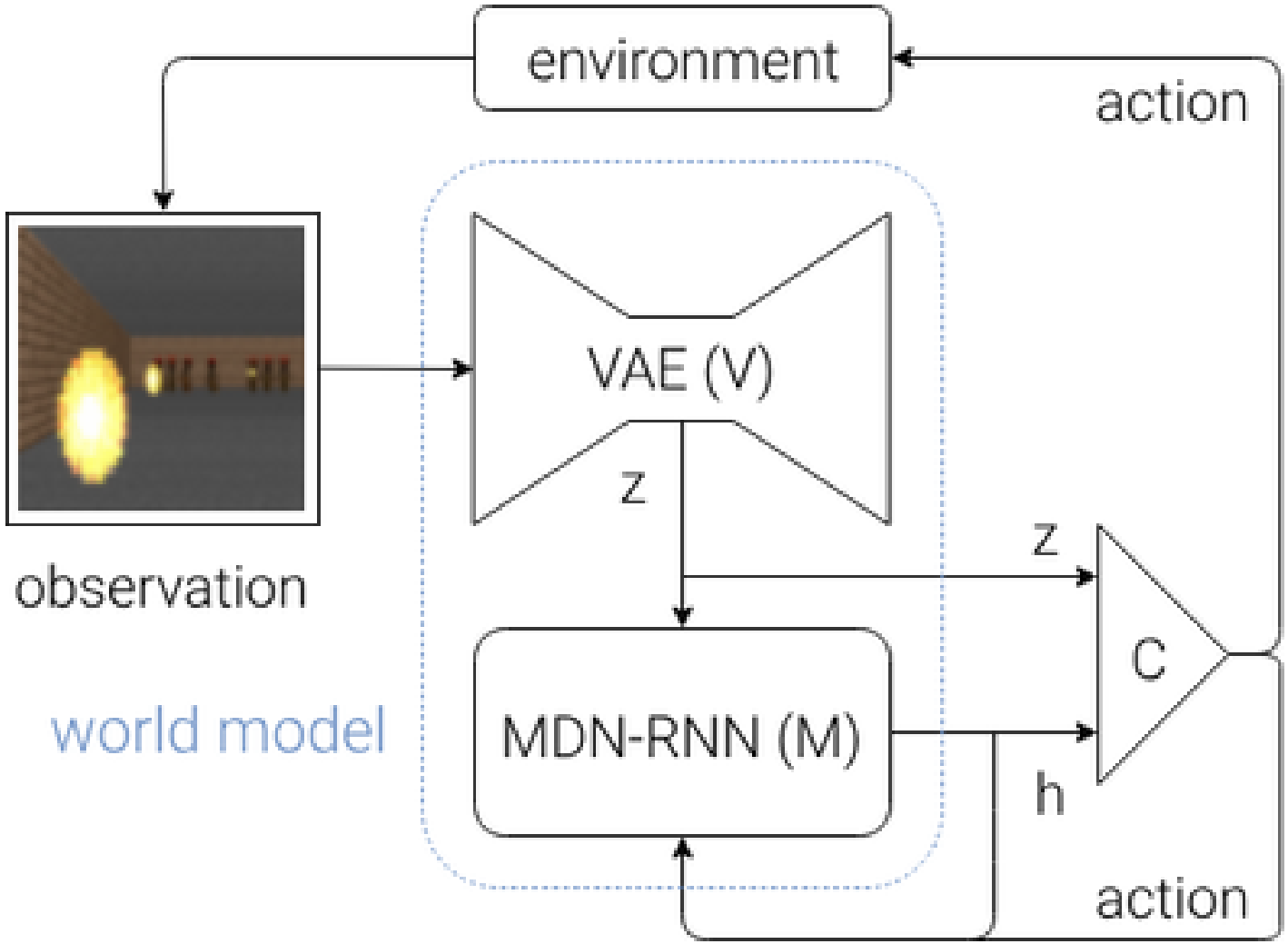
- C determines the course of actions to maximize the expected cumulative reward.
- Most of our agent's complexity resides in the world model (V and M).
- C simple and small, and trained separately from V and M,

$$a_t = W_c [z_t \quad h_t] + b_c$$

  - W_c : weight matrix
  - b_c : bias vector

World models

V , M and C together



World models

Practical benefits

World model

- (V and M) are designed to be trained efficiently using modern GPU accelerators.
- → Put Most of the model's complexity, and model parameters in V and M.

Controller

- The number of parameters of C (linear model) is minimal in comparison.
- Optimize the param. of C: Covariance-Matrix Adaptation Evolution Strategy (CMA-ES) [1,2,3].
- We evolve parameters of C on a single machine with multiple CPU cores running multiple rollouts of the environment in parallel.

[1] The CMA Evolution Strategy: A Tutorial. N. Hansen. ArXiv preprint. 2016.
[2] Completely Derandomized Self-Adaptation in Evolution Strategies. N. Hansen, A. Ostermeier. Evolutionary Computation, Vol 9(2), pp. 159—195. MIT Press. 2001.
[3] A Visual Guide to Evolution Strategies. D. Ha. blog.otoro.net. 2017.

World models

Car racing experiment

Procedure

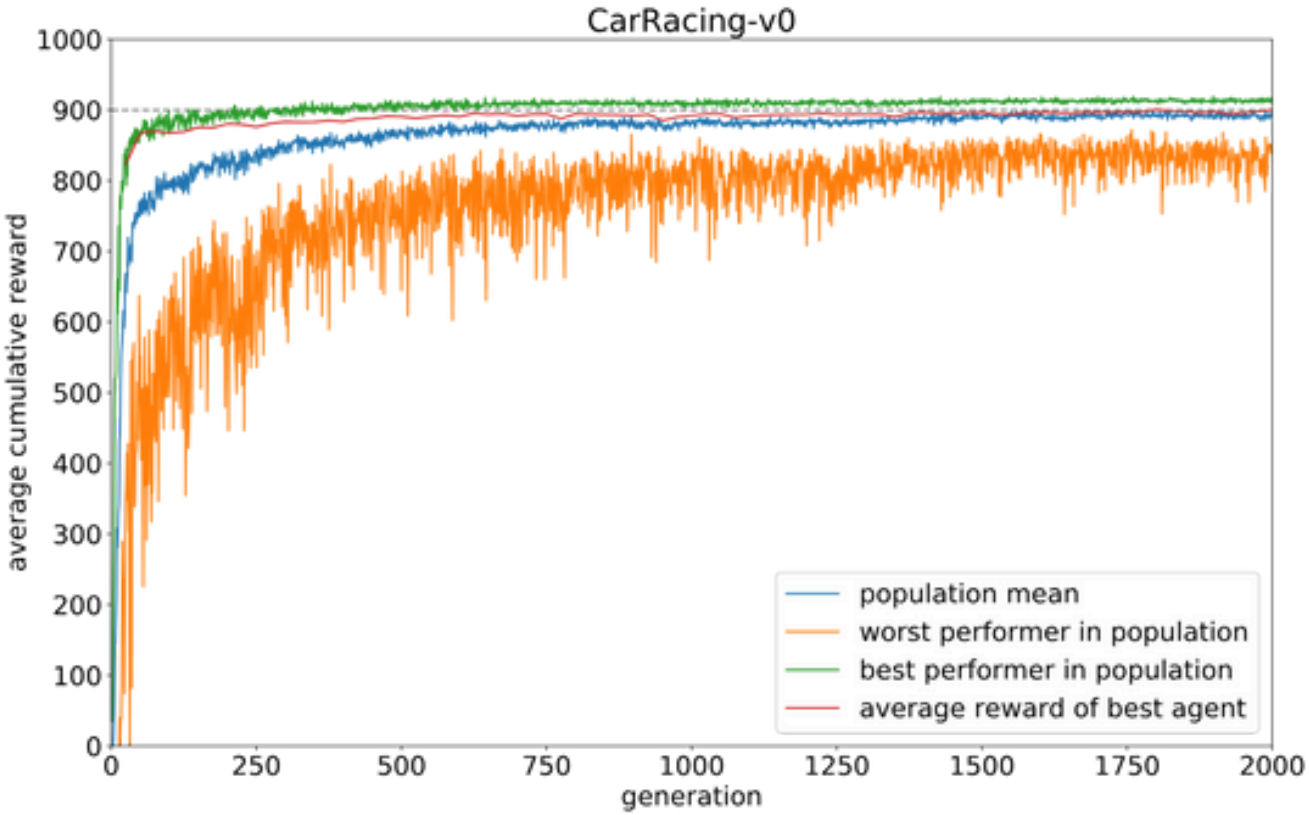
To summarize the Car Racing experiment, below are the steps taken:

1. Collect 10,000 rollouts from a random policy.
2. Train VAE (V) to encode each frame into a latent vector $z \in \mathcal{R}^{32}$.
3. Train MDN-RNN (M) to model $P(z_{t+1} \mid a_t, z_t, h_t)$.
4. Evolve Controller (C) to maximize the expected cumulative reward of a rollout.

Model	Parameter Count
VAE	4,348,547
MDN-RNN	422,368
Controller	867

World models

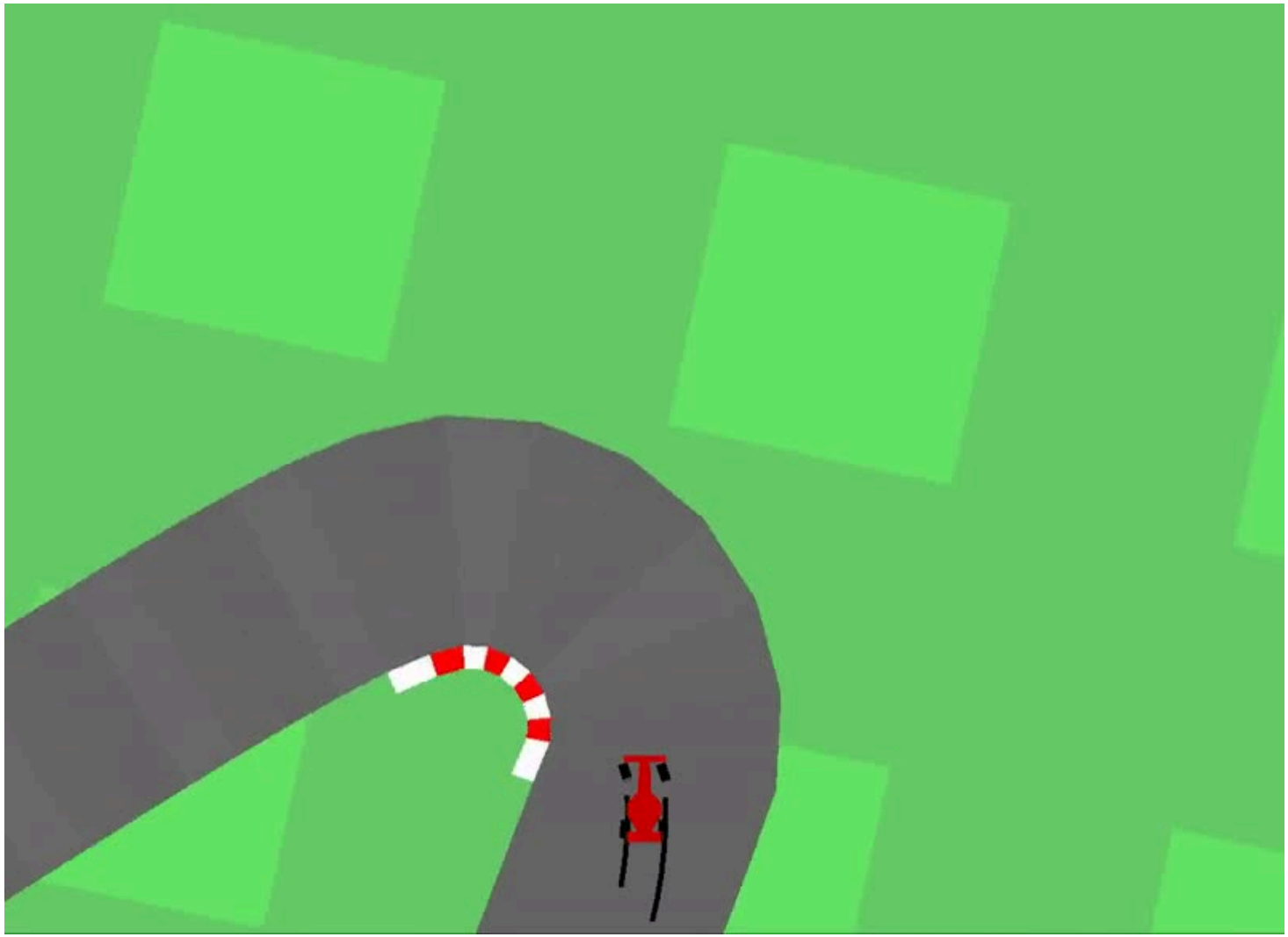
Car racing experiment



Training of CarRacing

World models

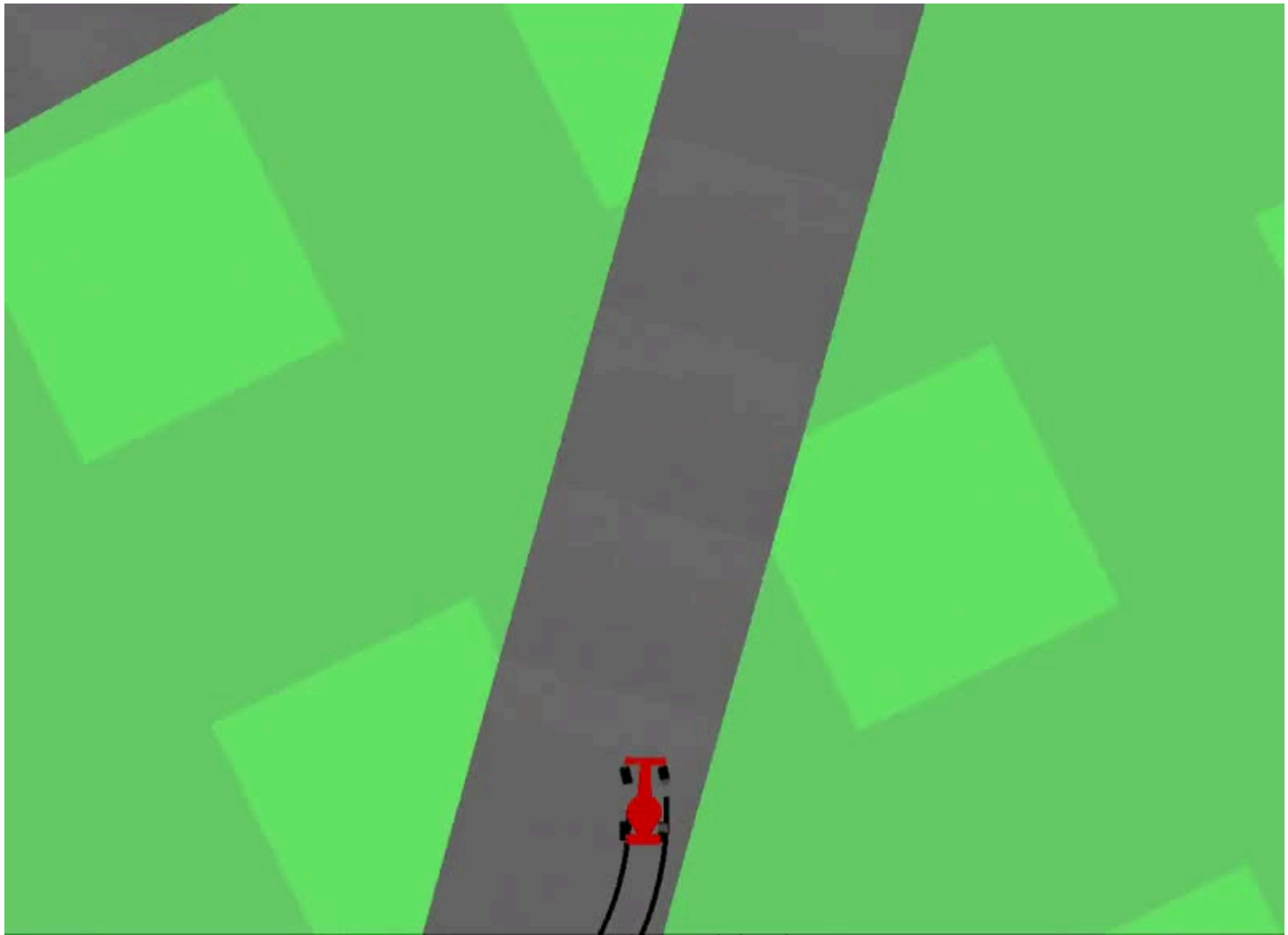
Car racing experiment



Training of CarRacing

World models

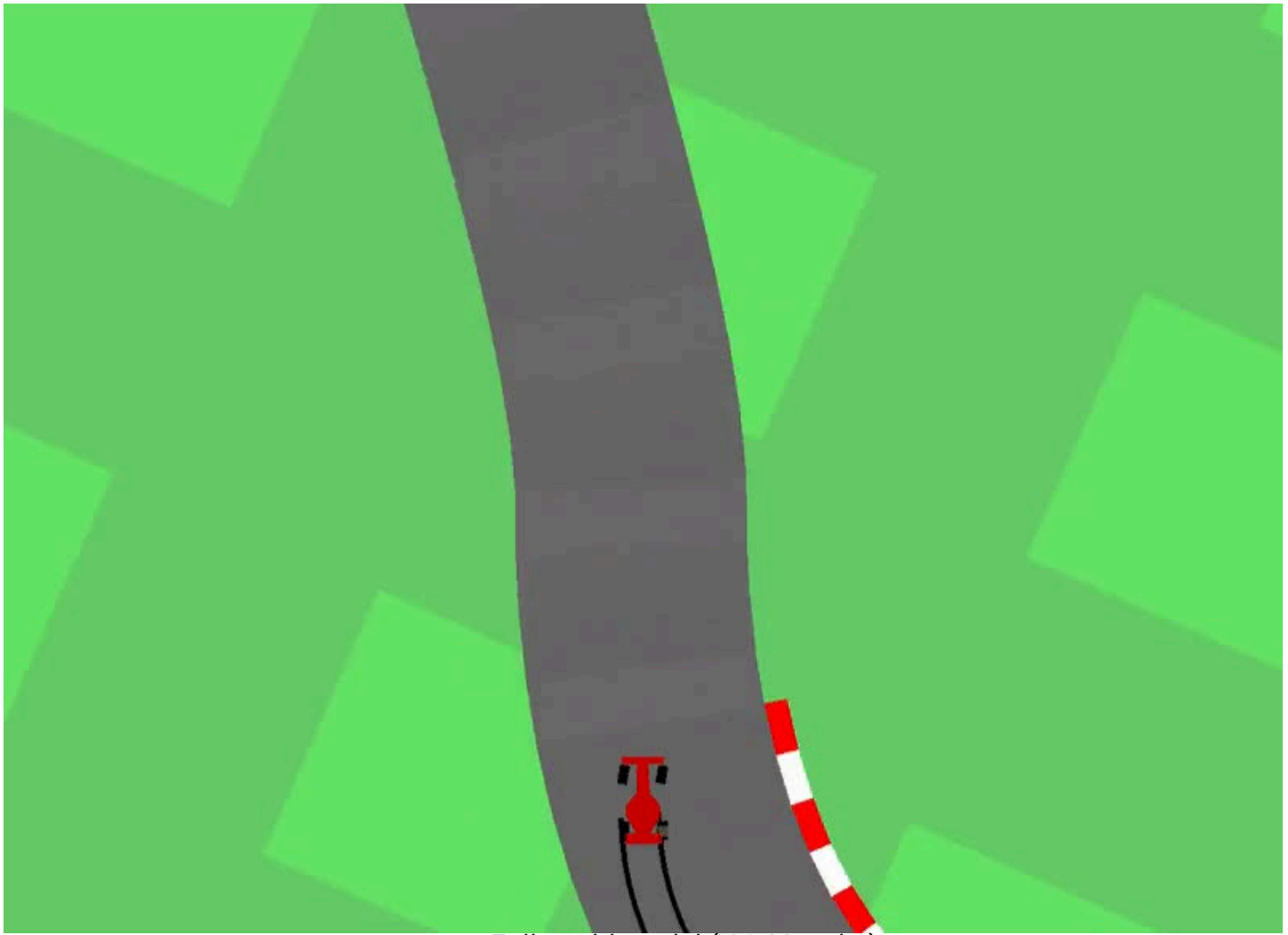
Car racing experiment



V and C model only

World models

Car racing experiment



Full world model (V & M and C)

World models

Car racing experiment

Car racing dreams

- Our world model is able to model the future
- We can ask it to produce P(z_t+1) given the current states
- then sample z_t+1 and use this sample as the real observation
- We can put our trained C back into this dream environment generated by M

Demo in: https://worldmodels.github.io/

2 - Imitation Learning

II -Imitation Learning [1]

- Model-free RL and model-based RL: assume reward function is known.
- Both collect system data to:
  - update a learned model (model-based)
  - or directly update a learned value function or policy (model-free).
- Drawbacks:
  - Accurate representation of the true performance objectives can be challenging.
  - Rewards may be sparse → Learning process expensive (data, time, memory).

Imitation Learning:

- We assume reward function is unknown a priori
- It is described implicitly through expert demonstrations.

[1] Stanford Univ lecture.

Where does the reward function come from ? [1]

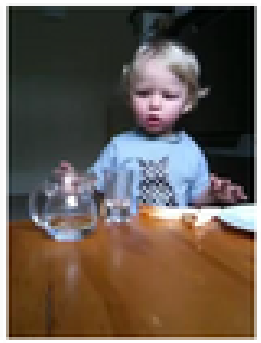
Computer Games

reward

Mnih et al. '15

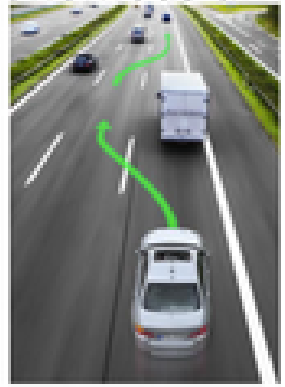
Real World Scenarios

robotics



dialog

autonomous driving



what is the reward?
often use a proxy

→ Infer reward function from roll-outs of expert policy

[1] Sergey Levine lecture CS 294-112.

Why should we learn the reward ?

Why not directly mimic the expert (behavior cloning) ?

- simply “ape” the expert’s motions/actions
- doesn’t necessarily capture the salient parts of the behavior
- what if the expert has different capabilities ?

→ Reason about what the expert is trying to achieve instead ?

Imitation Learning:

- Instead of leveraging an explicit reward function r_t = R(x_t, u_t)
- A set of demonstrations from an expert are provided.

Formulation of Imitation Learnign problem

The dynamics:

$$p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}), \tag{10.1}$$

An expert demonstration ξ consists of a sequence of state-control pairs:

$$\xi = \{(\boldsymbol{x}_0, \boldsymbol{u}_0), (\boldsymbol{x}_1, \boldsymbol{u}_1), \dots\}, \tag{10.3}$$

Definition 10.1.1 (Imitation Learning Problem). *For a system with transition model* (10.1) *with states* $\boldsymbol{x} \in \mathcal{X}$ *and controls* $\boldsymbol{u} \in \mathcal{U}$, *the imitation learning problem is to leverage a set of demonstrations* $\Xi = \{\xi_1, \dots, \xi_D\}$ *from an expert policy* π^* *to find a policy* $\hat{\pi}^*$ *that imitates the expert policy.*

Two approaches:

1 - Directly learn the policy (behavior cloning and the DAgger algorithm)

2 - Indirectly, by learning the expert's reward function (Inverse RL)

Behavior Cloning

Use a set of expert demonstrations $\xi \in \Xi$ to determine a policy π that imitates the expert.

It can be accomplished through supervised learning techniques:

$$\hat{\pi}^* = \arg\min_{\pi} \sum_{\xi \in \Xi} \sum_{x \in \xi} L(\pi(x), \pi^*(x)),$$

L is a cost function

can include p-norms (e.g. Euclidean norm) or f -divergences (e.g. KL divergence).

This approach may not yield very good performance

since the learning process is only based on a set of samples provided by the expert.

In many cases expert demonstrations are not uniformly sampled across the entire state space

E.g. Expert demonstrations come from a trajectory of sequential states and actions

DAgger: Dataset Aggregation

Idea : collect new expert data as needed

Assuming the expert can be queried on demand.

when the learned policy leads to unseen states → just query the expert

Algorithm 1: DAgger: Dataset Aggregation

Data: π^*

Result: $\hat{\pi}^*$

$\mathcal{D} \leftarrow 0$

Initialize $\hat{\pi}$

for $i = 1$ **to** N **do**

  $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}$

  Rollout policy π_i to sample trajectory $\tau = \{x_0, x_1, \dots\}$

  Query expert to generate dataset $\mathcal{D}_i = \{(x_0, \pi^*(x_0)), (x_1, \pi^*(x_1)), \dots\}$

  Aggregate datsets, $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$

  Retrain policy $\hat{\pi}$ using aggregated dataset $\mathcal{D}$

return $\hat{\pi}$

Inverse RL

Drawbacks of the direct approach (learn policies)

- no way to understand the underlying reasons for the expert behavior (no reasoning about outcomes or intentions).
- The "expert" may actually be suboptimal.
- A policy that is optimal for the expert may not be optimal for the agent (if they have different dynamics, morphologies, or capabilities.)

Alternative: Learn a representation of the underlying reward function

→ Learn the expert's intent

→ Potentially outperform the expert

→ or adjust for differences in capabilities

This approach is known as: Inverse RL

Inverse RL

Inverse RL assume a parameterization of the reward function

$$R(x, u) = w^T \phi(x, u),$$

w in R^n is a weight vector
φ(x,u) : X × U → R^n is a feature map

Recall: the total (discounted) reward under a policy π :

$$V_T^\pi(x) = E\Big[\sum_{t=0}^{T-1} \gamma^t R(x_t, \pi(x_t)) \mid x_0 = x\Big].$$

Then:

$$V_T^\pi(x) = w^T \mu(\pi, x), \quad \mu(\pi, x) = E_\pi\Big[\sum_{t=0}^{T-1} \gamma^t \phi(x_t, \pi(x_t)) \mid x_0 = x\Big],$$

Feature expectations μ(π, x) are often computed using a Monte Carlo technique (e.g. using the set of demonstrations for the expert policy).

Inverse RL

Optimiality of the expert policy $\pi*$

$$V_T^{\pi^*}(x) \geq V_T^{\pi}(x), \quad \forall x \in \mathcal{X}, \quad \forall \pi,$$

Which can be written:

$$w^{*T}\mu(\pi^*, x) \geq w^{*T}\mu(\pi, x), \quad \forall x \in \mathcal{X}, \quad \forall \pi.$$

→ Find a vector w that satisfies this condition.

Reward ambiguity: E.g. w = 0 satisfies this condition trivially!

Apprenticeship Learning [1]

Two main ideas:

1 - It doesn't matter how well $\boldsymbol{w}*$ is, as long as the *the feature expectations match*.

$$\|\mu(\pi, x) - \mu(\pi^*, x)\|_2 \leq \epsilon \implies |w^T \mu(\pi, x) - w^T \mu(\pi^*, x)| \leq \epsilon$$

for any w as long as // w // 2 ≤ 1

2 - The initial state x_0 is drawn from a distribution D :

$$E_{x_0 \sim D}\left[V_T^{\pi}(x_0)\right] = w^T \mu(\pi), \quad \mu(\pi) = E_{\pi}\left[\sum_{t=0}^{T-1} \gamma^t \phi(x_t, \pi(x_t))\right].$$

This is useful to avoid having to consider all x ∈ X when matching features.

[1] P. Abbeel and A. Ng. "Apprenticeship Learning via Inverse Reinforcement Learning". In: Proceedings of the Twenty- First International Conference on Machine Learning. 2004

Apprenticeship Learning [1]

Objective: Find π that makes $\mu(\pi)$ as similar as possible to $\mu(\pi*)$

Algorithm 2: Apprenticeship Learning

Data: $\mu(\pi^*), \epsilon$
Result: $\hat{\pi}^*$
Initialize policy π_0
for $i = 1$ **to** ... **do**
  Compute $\mu(\pi_{i-1})$ (or approximate via Monte Carlo)
  Solve problem (10.5) with policies $\{\pi_0, \ldots, \pi_{i-1}\}$ to compute w_i and t_i

$$(w_i, t_i) = \arg\max_{w,t} \; t,$$
$$\text{s.t. } w^T\mu(\pi^*) \geq w^T\mu(\pi) + t, \quad \forall \pi \in \{\pi_0, \ldots, \pi_{i-1}\},$$
$$\|w\|_2 \leq 1. \tag{10.5}$$

  if $t_i \leq \epsilon$ **then**
    $\hat{\pi}^* \leftarrow$ best feature matching policy from $\{\pi_0, \ldots, \pi_{i-1}\}$
    return $\hat{\pi}^*$
  Use RL to find an optimal policy π_i for reward function defined by w_i

[1] P. Abbeel and A. Ng. "Apprenticeship Learning via Inverse Reinforcement Learning". In: Proceedings of the Twenty- First International Conference on Machine Learning. 2004

Apprenticeship Learning

Example: optimal routing across a city

The reward function is not known

There is an expert who shows how to drive across the city (i.e. what routes to take).

1 - Behavior Cloning

- Simply try to mimic the actions taken by the expert
- E. g. memorize the actions (turns) at a particular intersections
- This approach is not robust at intersections that the expert never visited!

2 - Apprenticeship Learning

- Identify features of the expert's trajectories that are more generalizable
- Develop a policy that experiences the same feature expectations as the expert
- Examples of features:

  Take routes without stop signs

  Take routes with higher speed limits

3 - Multi-agent RL

Multi-agent RL - Introduction

MARL – Multi-agent RL (book) – 2024

Authors : Stefano V. Albrecht, Filippos Christianos, Lukas Schäfer

Link to pdf : https://www.marl-book.com/

codebase written in the Python - implementations of several MARL algorithms

Chapter 10 describes of how the algorithms are implemented in the codebase.

For familiars with RL, read chapter 3 then skip to chapter 9 and onward

MARL Slides (pdf and tex) : https://github.com/marl-book/slides

The book does not cover all aspects of MARL (e.g. communication in MARL)

Evolutionary game theory for multi-agent learning is not covered in this book
→ survey of Bloembergen et al. (2015).

Multi-agent RL - Introduction

Multiple autonomous agents

Decisions : each agent is capable of making its own decisions

Environment : the agents interact in a shared environment

Goal :

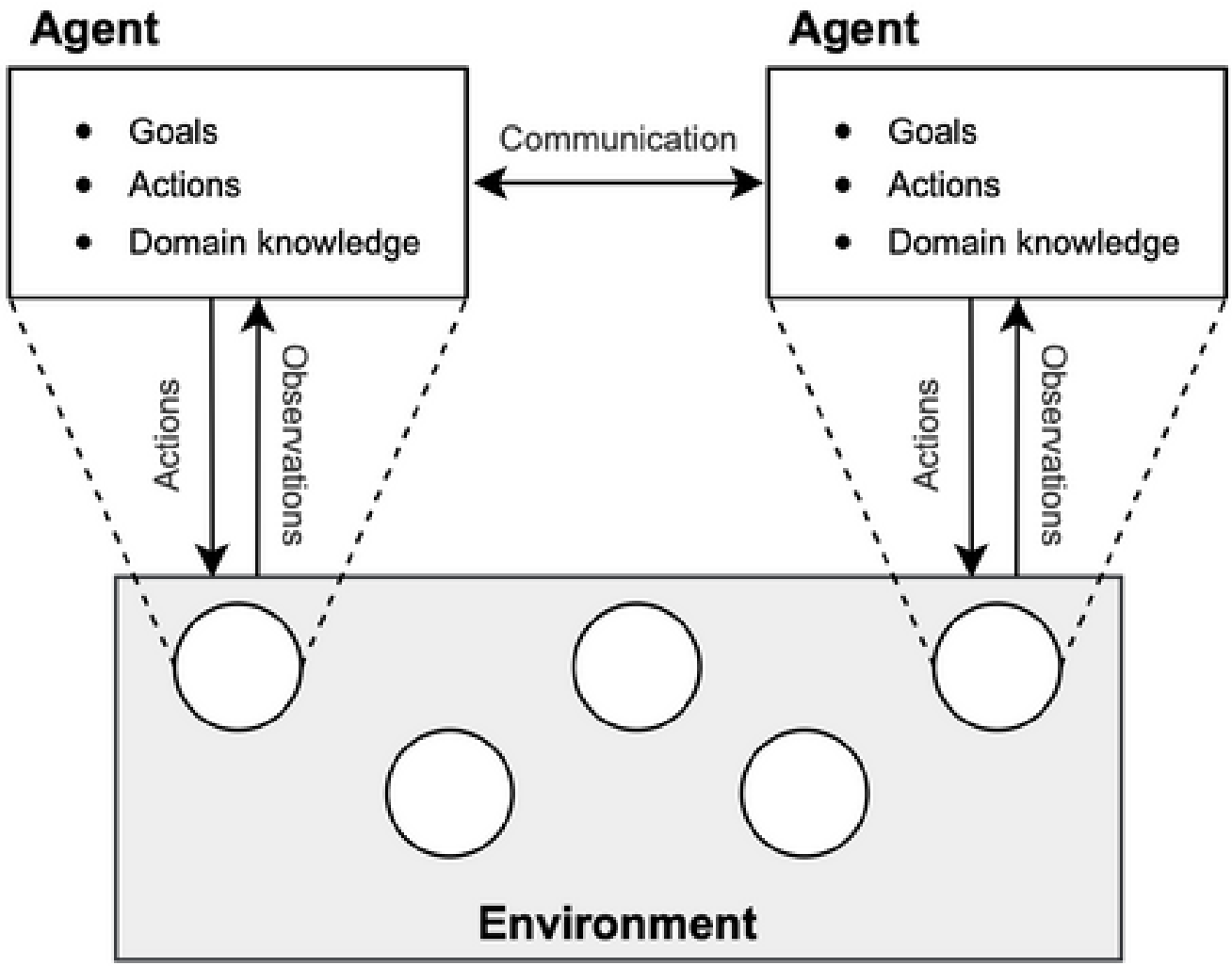
Shared goal:

- a fleet of mobile robots to collect and deliver goods within a large warehouse
- a team of drones tasked with monitoring a power plant.

Conflicting goals:

- agents trading goods in a virtual market: each agent seeks to maximize its own gains.

Multi-agent RL - Introduction



Multi-agent RL - Introduction

Learning :

- The agents begin to try actions in their environment
- Collect experiences about how the environment changes as a result of their actions,
- Learn how the other agents behave.
- Learn skills needed to solve their task
- Learn how to coordinate their actions with other agents
- May even learn to develop a shared language to enable communication
- Finally, they reach a certain level of proficiency
  and have become experts at interacting optimally to achieve their goals.

Multi-agent RL - Introduction

Three scenarios:

- Fully cooperative scenario, the agents collaborate toward achieving a shared goal.

  E.g. all agents receive a reward +1 whenever an agent performs a successful action.

- Competitive scenario, the agents are in direct competition with each other.

  E. g. two agents playing a game of chess, the winner gets +1, the loser –1

- Both cooperation and competition

  E.g. each agent gets a positive reward whenever it performs a successful action, once the goal is attained, all the agents receive a positive reward.

Multi-agent RL - Introduction

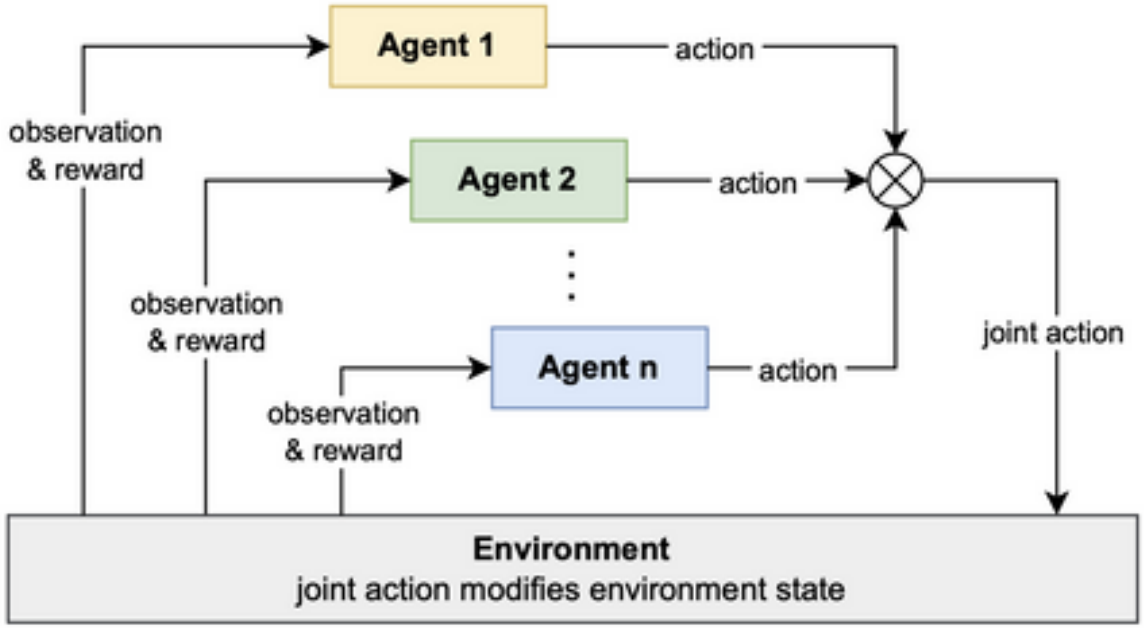
Formal (mathematical) definition

- Normal-form games (specification of players' strategy spaces and payoff functions)
- Stochastic games,
- Partially observable stochastic games

Solution for a game model: consists of a set of policies for the agents that satisfies certain desired properties.

Eg. Nash Equilibrium : No individual agent can deviate from its policy in the solution to improve its outcome.

Multi-agent RL - Introduction



Several questions :

- How to design algorithms where agents choose optimal actions toward their goals ?
- How to design environments to incentivize certain long-term behaviors in agents ?
- How information is communicated and propagated among agents ?
- How norms, conventions, and roles may emerge in a collective of agents ?

Multi-agent RL - Introduction

Centralized vs Decentralized decision-making

Consider 3 agents, each of them with an action space of 10 actions.

- Central RL : we will have $10^3 = 1000$ actions.
- MA decentralized : 3 decentrailized RL agents, each of one with 10 actions.

Challenge : the agents need to coordinate their actions in order to be successful.

MARL may use various approaches to facilitate the learning of coordinated agent policies.

Examples

- Autonomous driving (urban env.): each car requires its own local policy to drive.
- Robots for search-and-rescue: each agent may need to act fully independently.

Multi-agent RL - Introduction

Dimensions in MARL

Size

- How many agents exist in the environment?
- Is the number of agents fixed, or can it change?
- How many states and actions does the environment specify?
- Are states/actions discrete or continuous?
- Are actions defined as single values or multi-valued vectors?

Multi-agent RL - Introduction

Dimensions in MARL

Knowledge

- Do agents know what actions are available to themselves and to others ?
- Do they know their own reward functions, and the reward of other agents?
- Do agents know the state transition probabilities of the environment?

Multi-agent RL - Introduction

Dimensions in MARL

Observability

- What can agents observe about their environment?
- Can agents observe the full environment state, or partial and noisy?
- Can they observe the actions and/or the rewards of other agents?

Multi-agent RL - Introduction

Dimensions in MARL

Rewards

- Are the rewards fully cooperative, competitive, or mixed ?
- Are the agents opponents, with zero-sum rewards?
- Or are agents teammates, with common (shared) rewards?
- Or do agents have to compete and cooperate in some way?

Multi-agent RL - Introduction

Dimensions in MARL

Objective

- Is the agents' goal to learn an equilibrium joint policy?
- What type of equilibrium?
- Is performance during learning important, or only the final learned policies?
- Is the goal to perform well against certain classes of other agents?

Multi-agent RL - Introduction

Dimensions in MARL

Cenralization /communication

- Can agents coordinate their actions via a central controller or coordinator?
- Or do agents learn fully independent policies?
- Can agents share/communicate information during learning and/or after learning ?
- Is the communication channel reliable, or noisy and unreliable?

Multi-agent RL - Introduction

Dimensions in MARL

Training and execution algorithms

Centralized training and execution:

- both stages have access to some centrally shared mechanism or information, such as sharing all observations between agents.

Decentralized training and execution:

- no such centrally shared information
- the learning of an agent's policy only use the local information of that agent

Centralized training with decentralized execution:

- aims to combine the benefits of the two approaches
- assuming that centralization is feasible during training (e.g., in simulation)
- while producing policies that can be executed in a fully decentralized way.

Multi-agent RL - Introduction

Multi-Robot Warehouse Management

Observation ?

Actions ?

Rewards ?

- Many aisles of storage racks that contain all manner of items.
- Orders, which specify certain items and quantities

  to be picked up from the storage racks and delivered to a work station
- Robots moving along the aisles and pick items from the storage racks.
- MARL to train the robots to collaborate optimally

  The goal: complete the orders as quickly and efficiently as possible.

Multi-agent RL - Introduction

Multi-Robot Warehouse Management

Observation :

- location
- current heading within the warehouse,
- the items carried,
- the current order serviced.

+ information of other agents (locations, items, and orders).

Actions :

- physical movements: rotate, accelerate, brake, pick items.
- send communication messages to other robots (travel plans, etc.)

Rewards :

- individual positive reward when completing an order
- reward when any order has been completed by any robot.

Multi-agent RL - Introduction

Autonomous Driving

Observation ?

Actions ?

Rewards ?

Multi-agent RL - Introduction

Autonomous Driving

observations

- own controlled vehicle (e.g., position on lane, orientation, and speed)
- other nearby vehicles (may be uncertain, noisy or incmoplete)

actions

- continuous controls, such as steering and acceleration/braking,
- discrete actions e.g., change lane, turning, overtaking

Rewards

- Penalize collisions (negative reward).
- positive rewards for minimizing driving times,
- negative rewards for abrupt acceleration/braking and frequent lane changes.

Multi-agent RL - Introduction

Autonomous Driving

Collaboration : ?

Collaboration : agents collaborate to avoid collisions

Competition : ?

Competition : minimize driving times and drive smoothly.

4 - Hierarchical Reinforcement Learning (HRL)

Hierarchical Reinforcement Learning (HRL)

Abstract

- HRL extends traditional RL incorporating a hierarchical structure in the learning process.
- Learn multiple levels of policies for different levels of abstraction.
- Tasks are broken down into sub-tasks, ...
- Each level of the hierarchy focuses on solving a specific aspect of the overall task.
- HRL simplifies the learning process and improves the scalability and efficiency.

- This idea is very similar to breaking down large number of lines of code to smaller functions each performing a very specific task.

Hierarchical Reinforcement Learning (HRL)

Key components

1- Hierarchical Policies:

- Lower-level policies:
  focus on achieving specific goals within the context set by higher-level policies.
- Higher-level policies:
  determine which sub-task or lower-level policy to activate.

Hierarchical Reinforcement Learning (HRL)

Key components

2- Options Framework: An option consists of three components:

- An initiation set: I in S
- A policy: $\pi: S \times A \to [0,1]$
- A termination condition: $\beta: S \to [0,1]$

If the option is taken
then actions are selected according to π,
until the option terminates stochastically according to β.

Hierarchical Reinforcement Learning (HRL)

Key components

3- Subgoal Discovery:

- Subgoals act as intermediate milestones
- that the agent needs to achieve on its way to accomplishing the overall task.
- Effective subgoal discovery can significantly enhance the performance of HRL.

Hierarchical Reinforcement Learning (HRL)

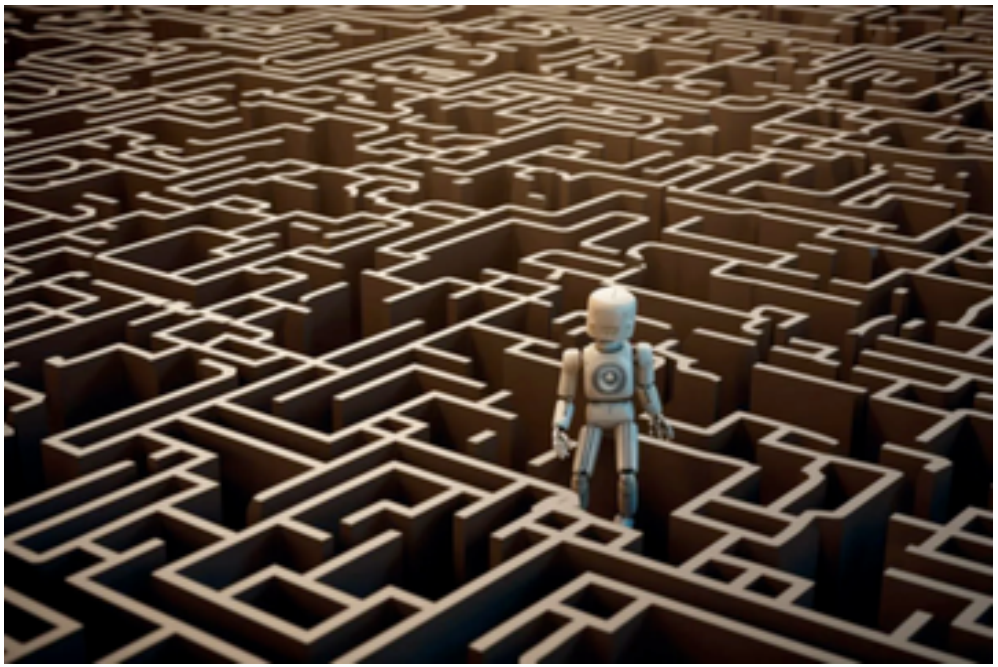
Key components

4- Reward Shaping:

- Reward shaping involves assigning rewards at different levels of the hierarchy to guide the agent's learning process.
- Providing intermediate rewards for achieving subgoals can accelerate convergence and improve learning efficiency.

Hierarchical Reinforcement Learning (HRL)

H-DQN for Autonomous Robot Navigation



- Objective: Enable a robot to navigate a maze-like environment.

- Autonomously reach a target location:
  - Avoiding obstacles
  - and navigating efficiently through the environment.

- The robot has to learn how to make decisions at two levels of abstraction:
  - High-level planning
  - Low-level control.

Hierarchical Reinforcement Learning (HRL)

H-DQN for Autonomous Robot Navigation

Two-level hierarchical model:

High-Level Controller (Meta-Controller):

- Responsible for selecting subgoals for the robot.
- Responsible for selecting subgoals for the robot.
- Subgoals: intermediate states to reach the final destination.
- Operates on a more abstract level
- Focusing on the overall strategy to navigate the environment.

Low-Level Controller (Subgoal Achievement):

- Responsible of achieving the subgoals set by the high-level controller.
- Involves fine-grained control of the robot's movements
- such as turning, moving forward, and avoiding obstacles in the immediate vicinity.
- Uses a standard DQN approach to learn these controls.

Hierarchical Reinforcement Learning (HRL)

H-DQN for Autonomous Robot Navigation

State Representation:

- use sensor inputs, such as LiDAR or depth cameras
- providing information about the robot's surroundings
- including distances to walls and obstacles.

Reward Structure:

- Designed hierarchically.
- The high-level controller received a reward when the robot achieved a subgoal that moved it closer to the target.
- The low-level controller received rewards for successful execution of movements that contributed to achieving these subgoals.

Training:

- The H-DQN is trained in a simulated environment
- The robot learn to navigate mazes of increasing complexity.
- Over time, the robot learns how to decompose the navigation task into subgoals
- and how to execute the necessary actions to achieve these subgoals efficiently.

Thank you !