Reinforcement Learning (RL)

Advanced Machine Learning – IBIO Uniandes

September 7, 2020

By: Daniela Tamayo & Isabella Ramos

Tutors: Laura Daza, Catalina Gómez

Contents

- **Introduction**
  - History and Context
  - What is RL?
  - Basic elements and key concepts
- **Theory**
  - Categorizing RL Algorithms
  - Markov Decision Process (MDP)
  - Bellman's equation & Q-function
  - Credit assignment and reward function
  - Exploration vs. Exploitation
  - Monte Carlo methods
- **Publications**
  - **Paper 1**: Playing Atari with Deep Reinforcement Learning
  - **Paper 2**: Discovering Reinforcement Learning Algorithms
- **Practice**
  - Understanding RL code
  - Gym tutorial
  - Homework

Textbook references:

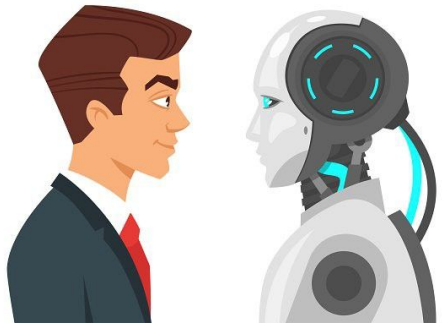
[1] "Reinforcement Learning an introduction" Sutton R.S, Barto A.G. Adaptive Computation and Machine Learning

[2] "Reinforcement Learning algorithms with Python" Lonza A. Packt Birmingham - Mumbai

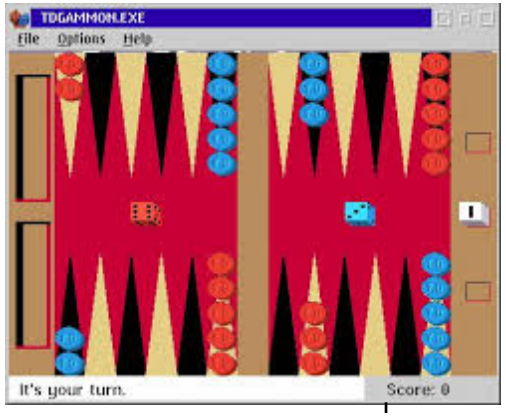
Learning: Human vs. Machine

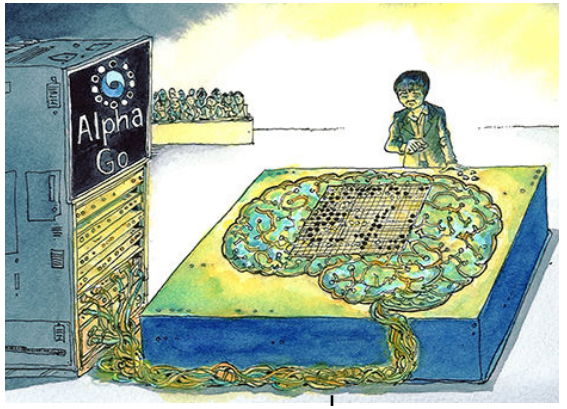
- Machines limited by set of actions, humans limited by freedom and laws of physics

- Repetitive interaction with our environment
  - Discover cause and effect
  - Understand consequences of our actions
  - Determine ways to achieve goals
- There might be more than a single answer



Timeline of RL





Google DeepMind

OpenAI

1950-1980: Early History
Optimal control and
Temporal-difference
learning

1992: TD-Gammon

2013: Playing Atari
(Deep Q-Network)

2017: AlphaGO
AlphaStar

Today: More and
more research…

[1], [2]

Early History

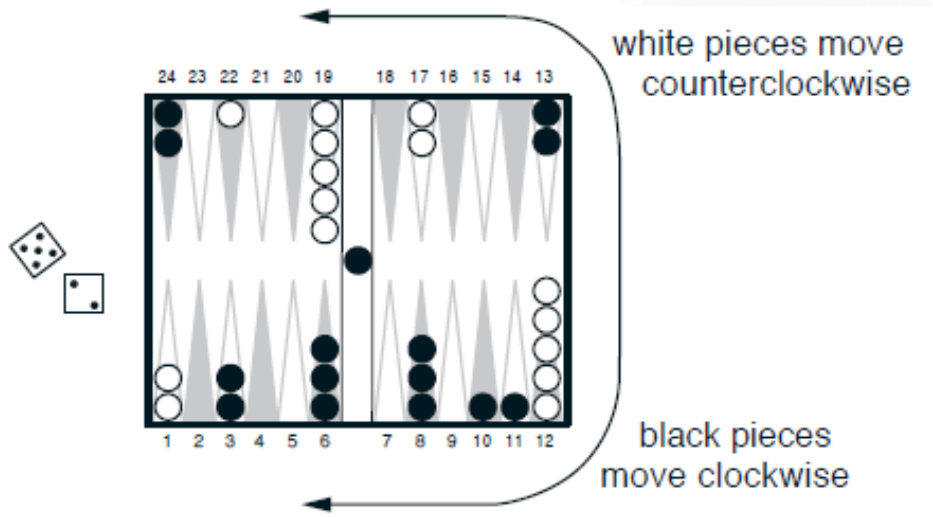
Optimal Control (1960-1970): Control of dynamical system over a period of time. Design a controller to optimize a behavior's measure of the dynamic system over time.

- Bellman's equation: optimal return function
- **Markov's decision process** (MDP): general framework for modeling decision-making in stochastic situations.
- **Dynamic programming:** Solution method for optimal control problems. **->** curse of dimensionality

Temporal-difference learning (TD) (1980): Predict variable over multiple time steps. Sampling from the environment (Monte Carlo methods) **+** updates based on current estimates or bootstrapping (DP).

[1], [2]

TD-Gammon

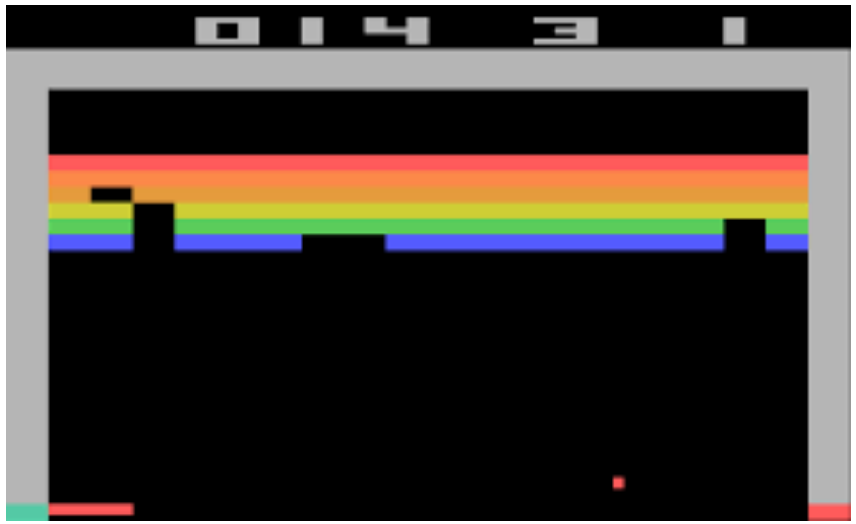


A backgammon position [1]

1992: Gerald Tesauro at IBM's Thomas J. Watson research Center.

- Neurogammon
- Possible moves **->** resulting board **->** evaluation function
- **TD**: update weights after each turn

Playing Atari with RL



2013: DeepMind

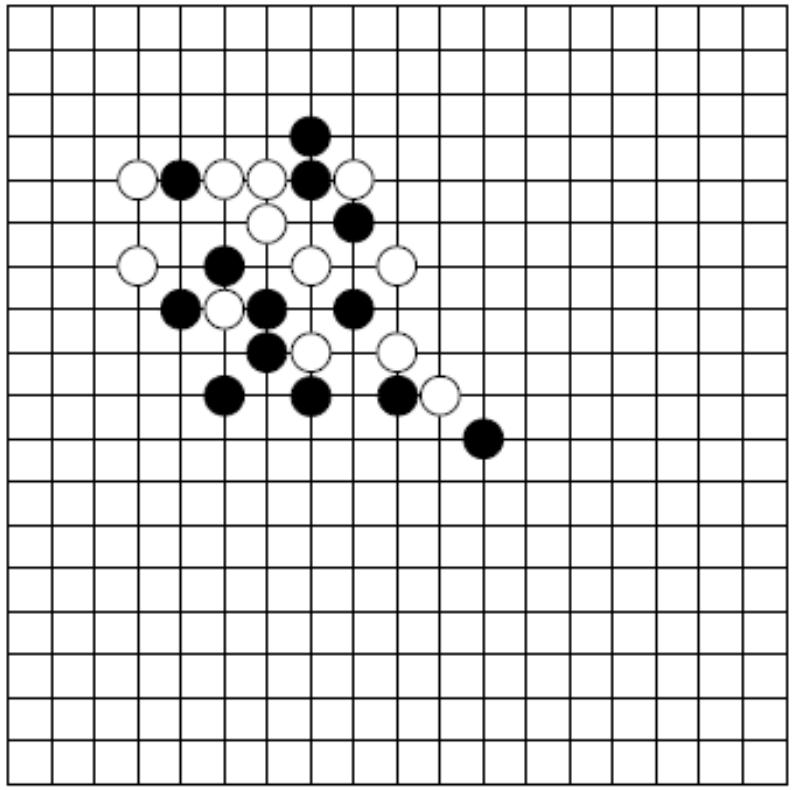
- Playing seven Atari 2600 games
- Deep reinforcement learning
- Q - learning

[1], [2], Paper 1

History and Context

AlphaGo

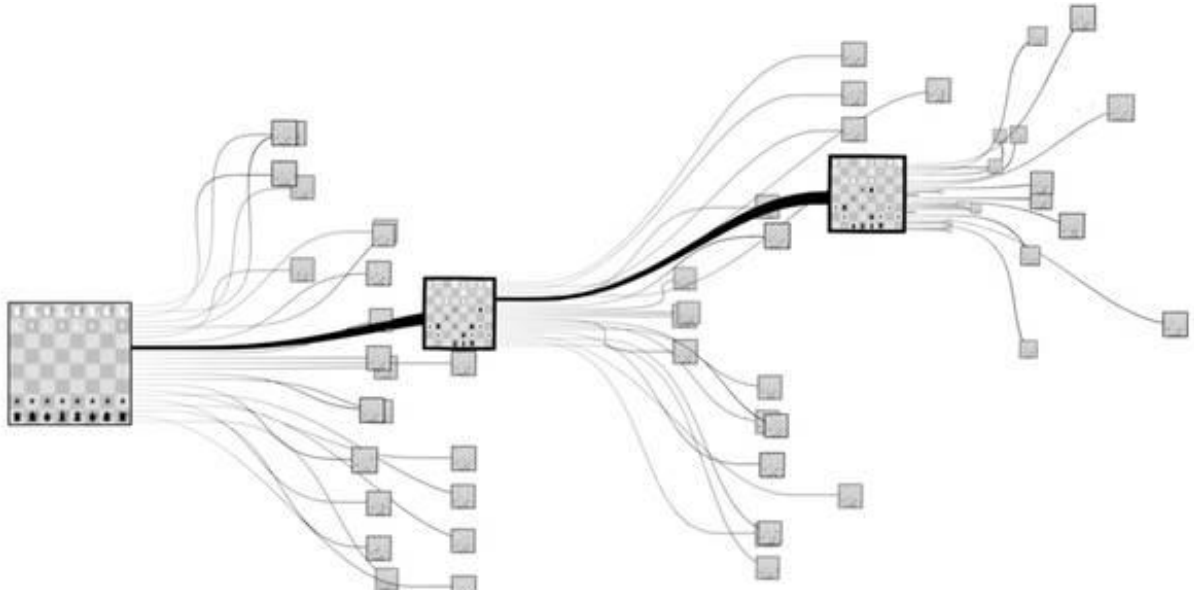
- Ancient Chinese strategic board game GO



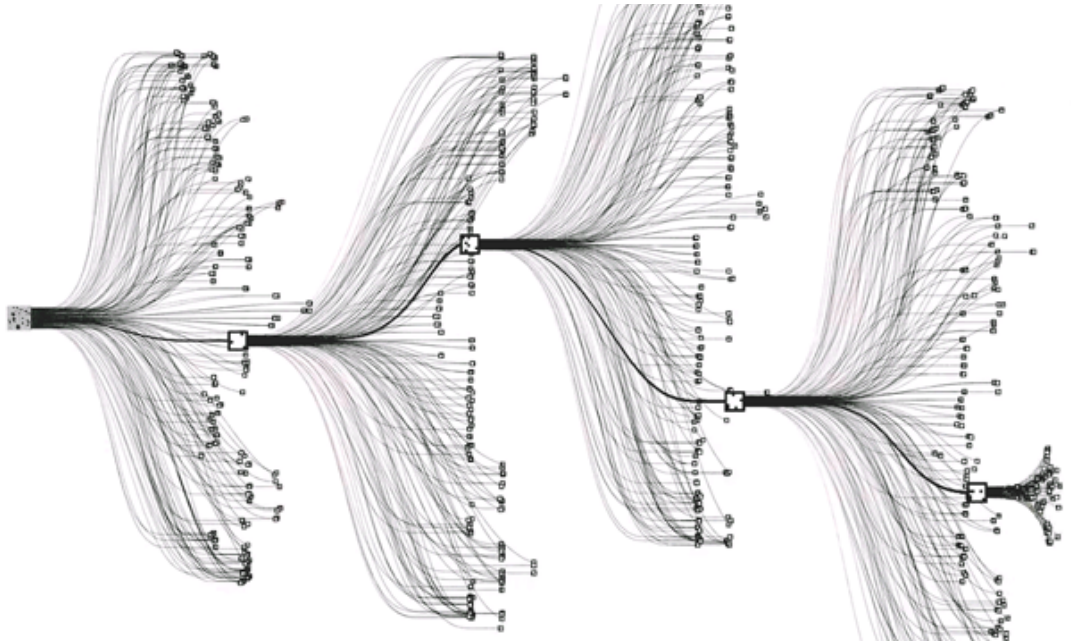
A Go board configuration

[1]

Possible moves in Chess →



Possible moves in GO →

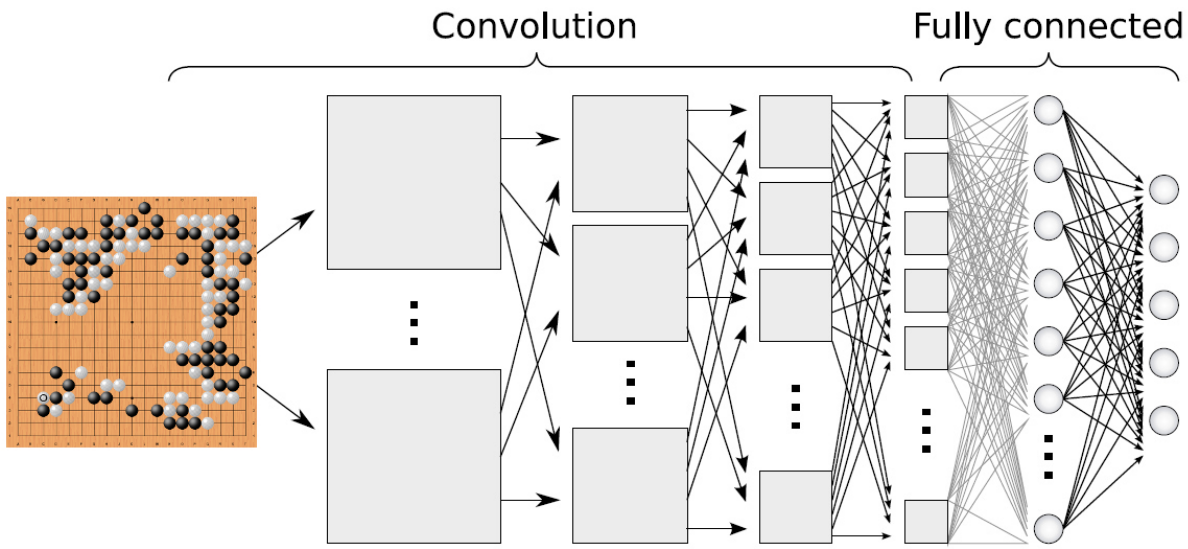


History and Context

AlphaGo

2017: Google Deepmind

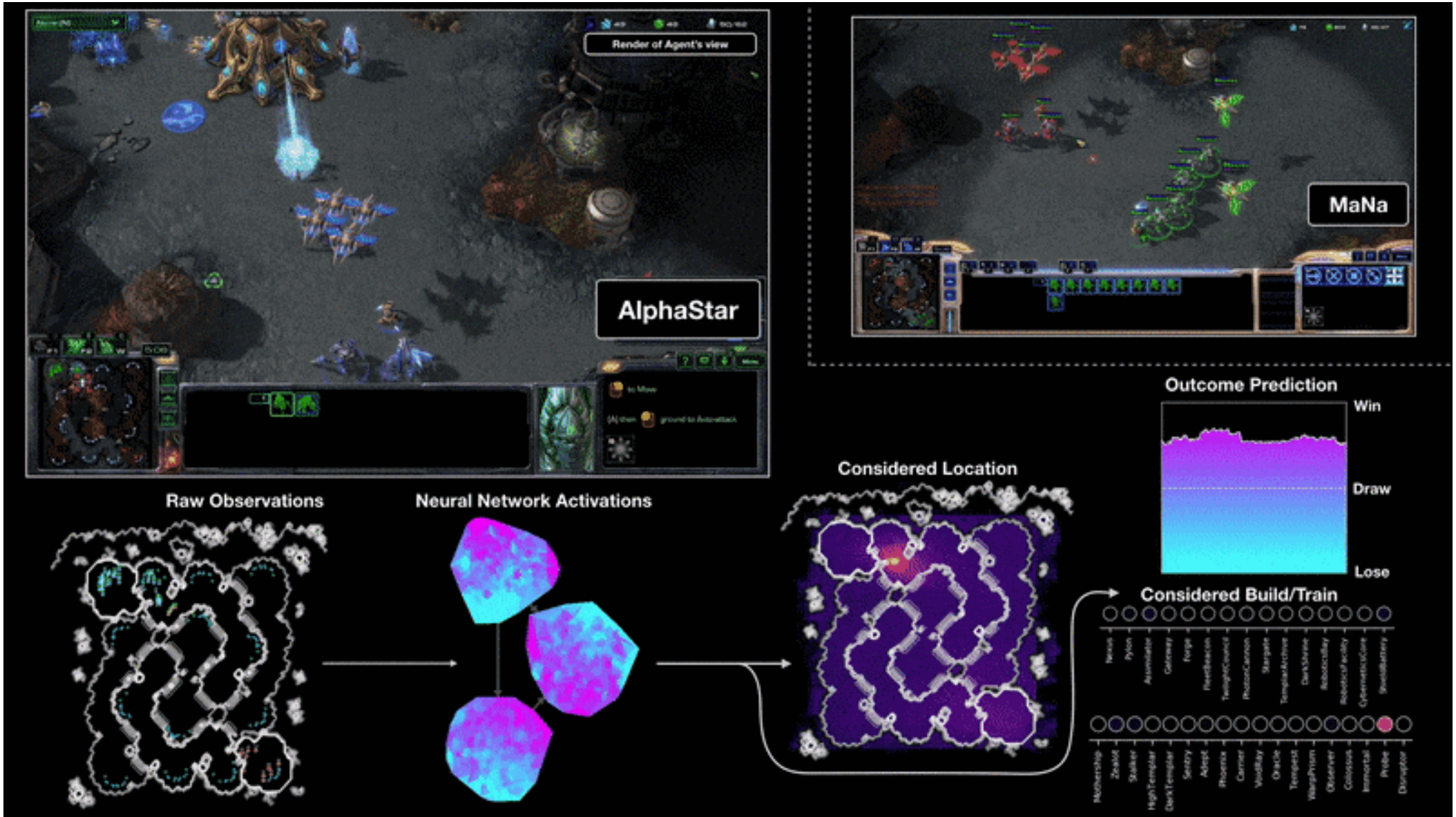
- Reinforcement learning + Monte Carlo tree search (MCTS)
- General Purpose
- **AlphaGo Zero:** completely self-thought



[1]

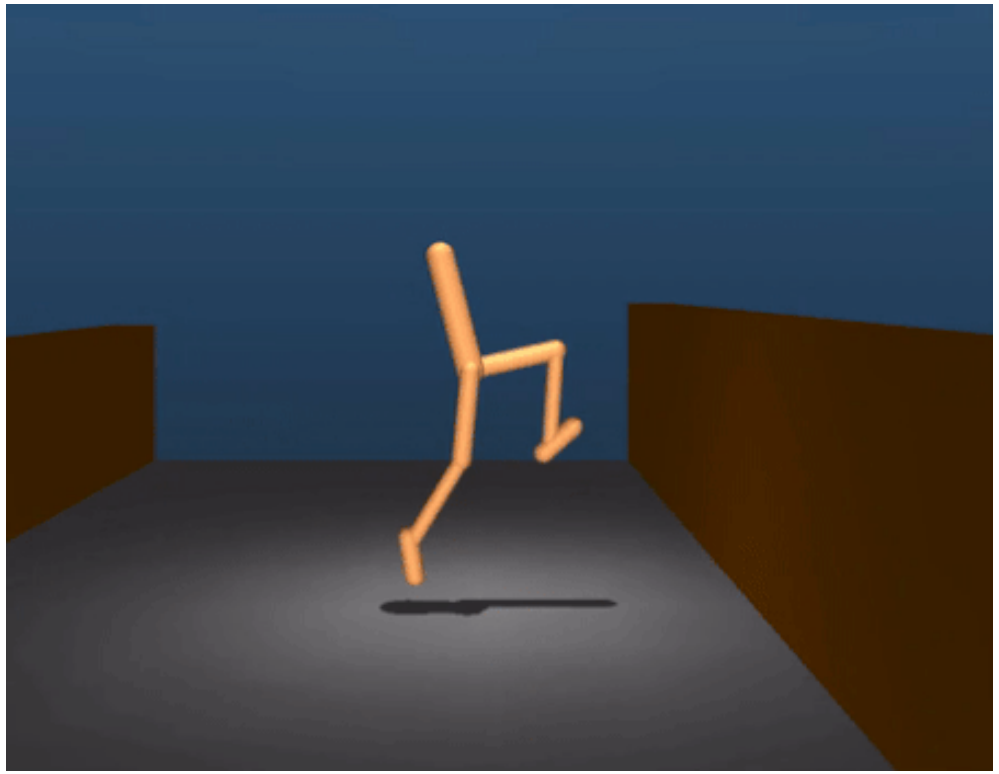
AlphaStar

2019: Google Deepmind vs StarCraft II pro players



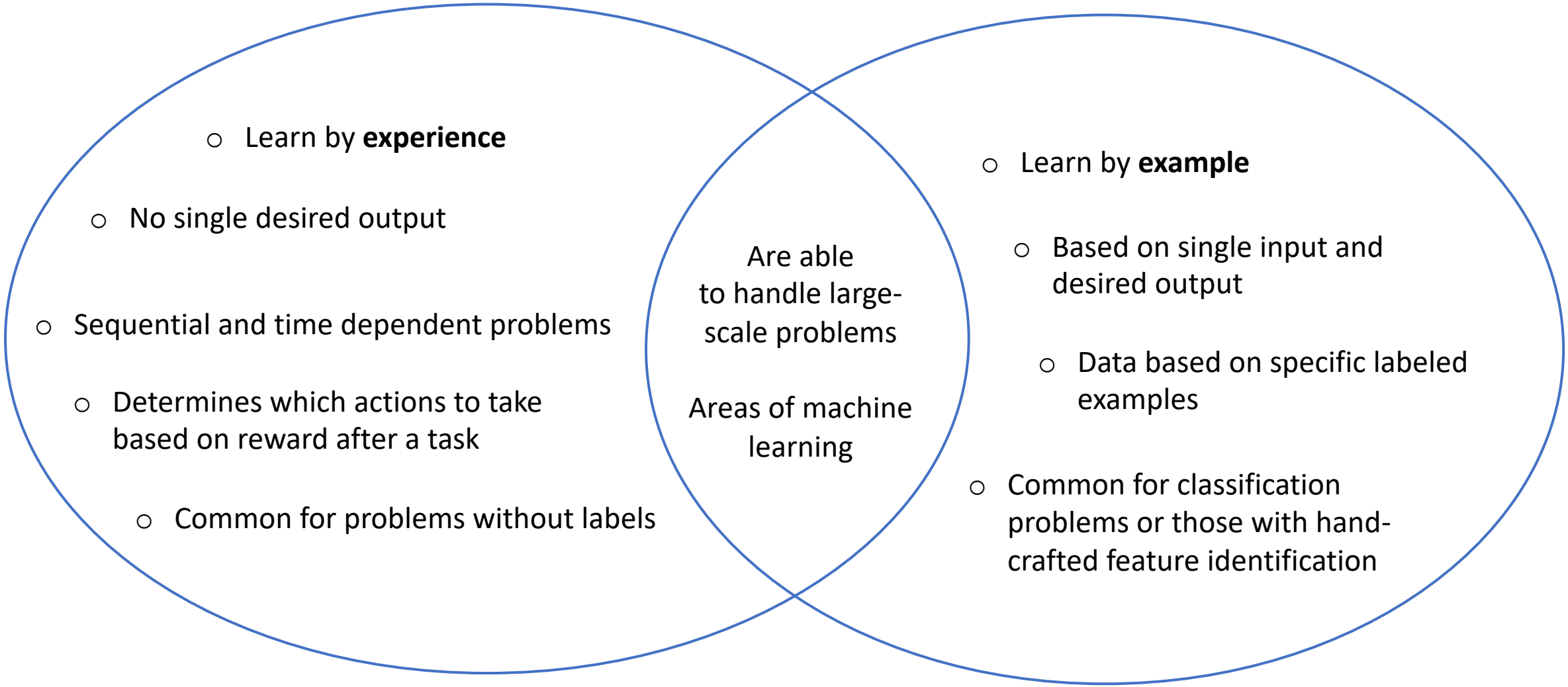
Reinforcement Learning

- Focused on **goal-directed** learning, like a human
- Discover which actions yield the most reward by **trial and error**
- No single label with the "correct response"



[1]

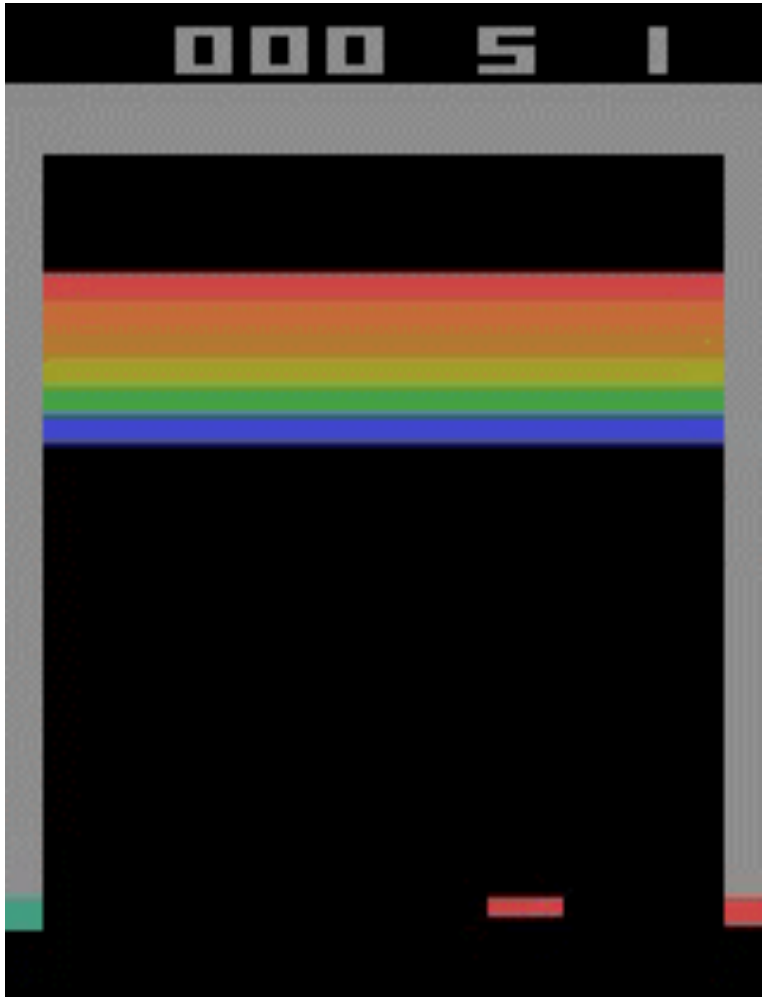
Reinforcement vs. Supervised Learning



[1]

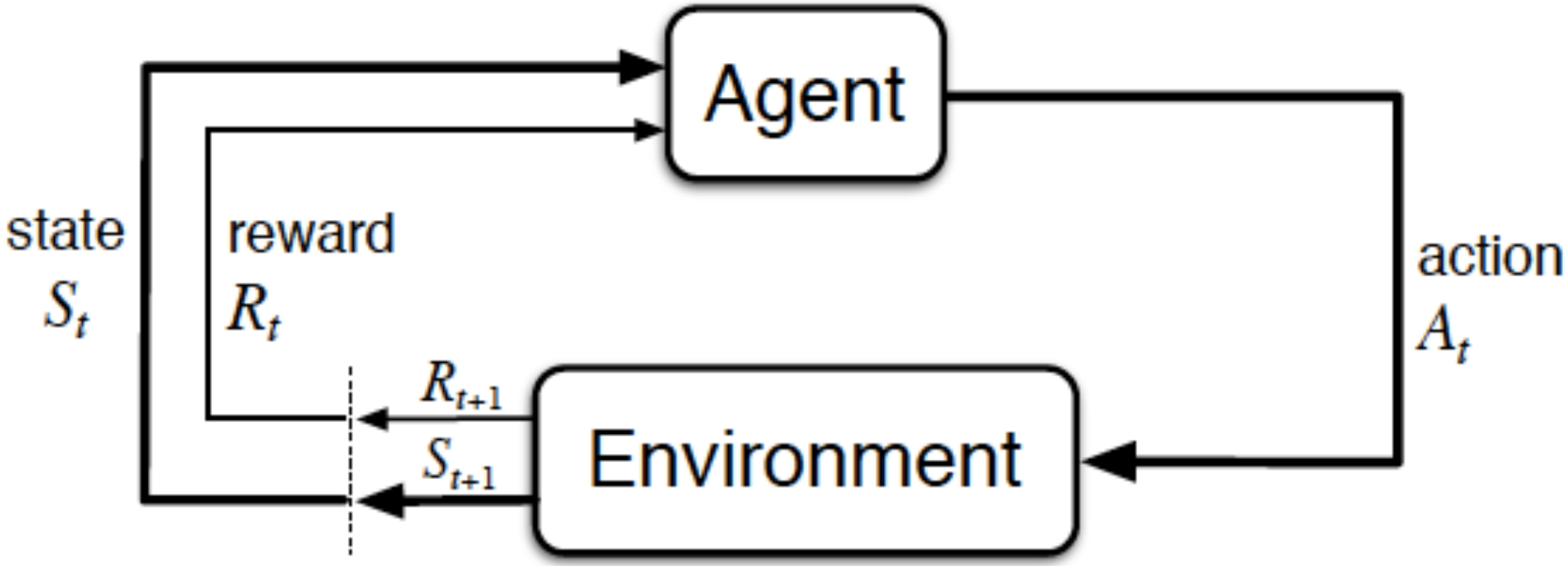
Breakout

- Popular Atari game and part of DeepMind's RL publications
- **Goal**: Bounce ball on movable paddle and walls until all bricks are broken
- Destruction of different colored bricks results in different amount of points
- **Interface**: 210 x 160 RGB video at 60 Hz.



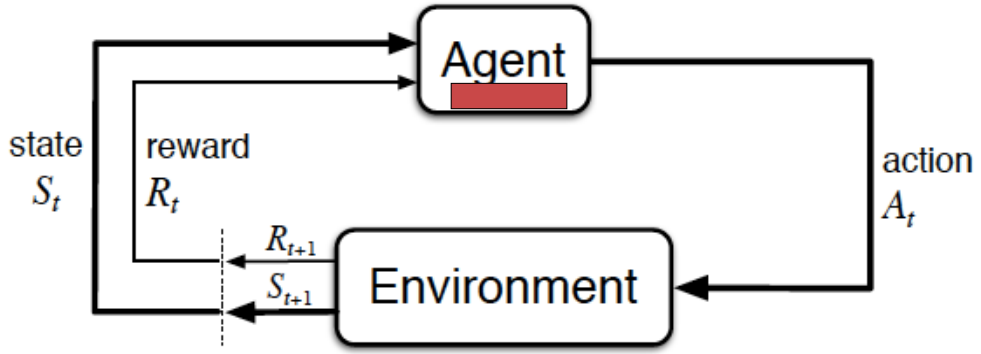
[9]

Basic elements

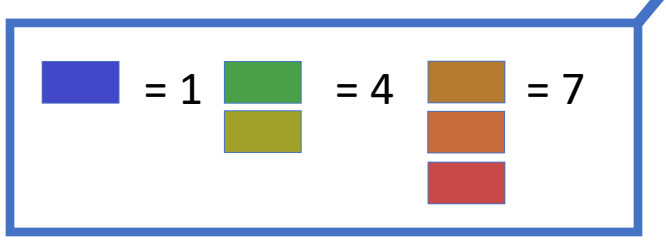


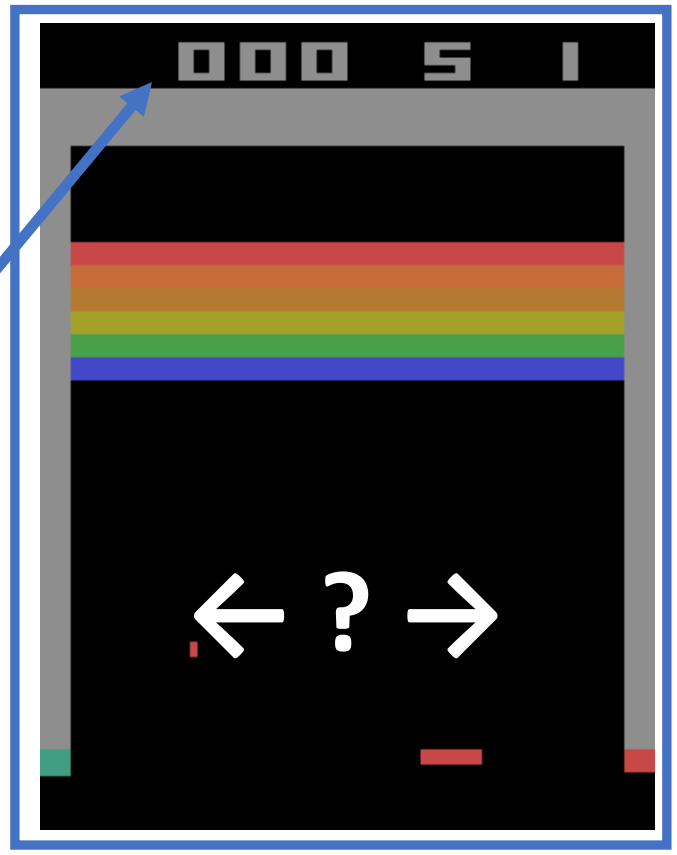
[1]

Basic elements



- **State**
  - **Information** available to the agent **about its environment**
- **Policy (π)**
  - Defines the **way to behave** at a given time
  - Mapping from perceived **states** of the environment to **actions** to be taken when in those states
- **Reward signal**
  - Defines the **goal** of the problem
  - Total reward must be **maximized** in the long run
  - Immediately determines **desirability of a state** (what is good or bad)

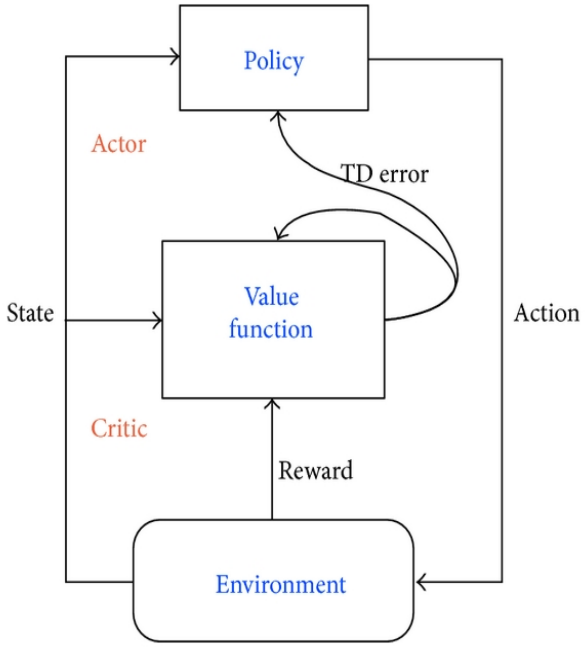




[1]

Basic elements

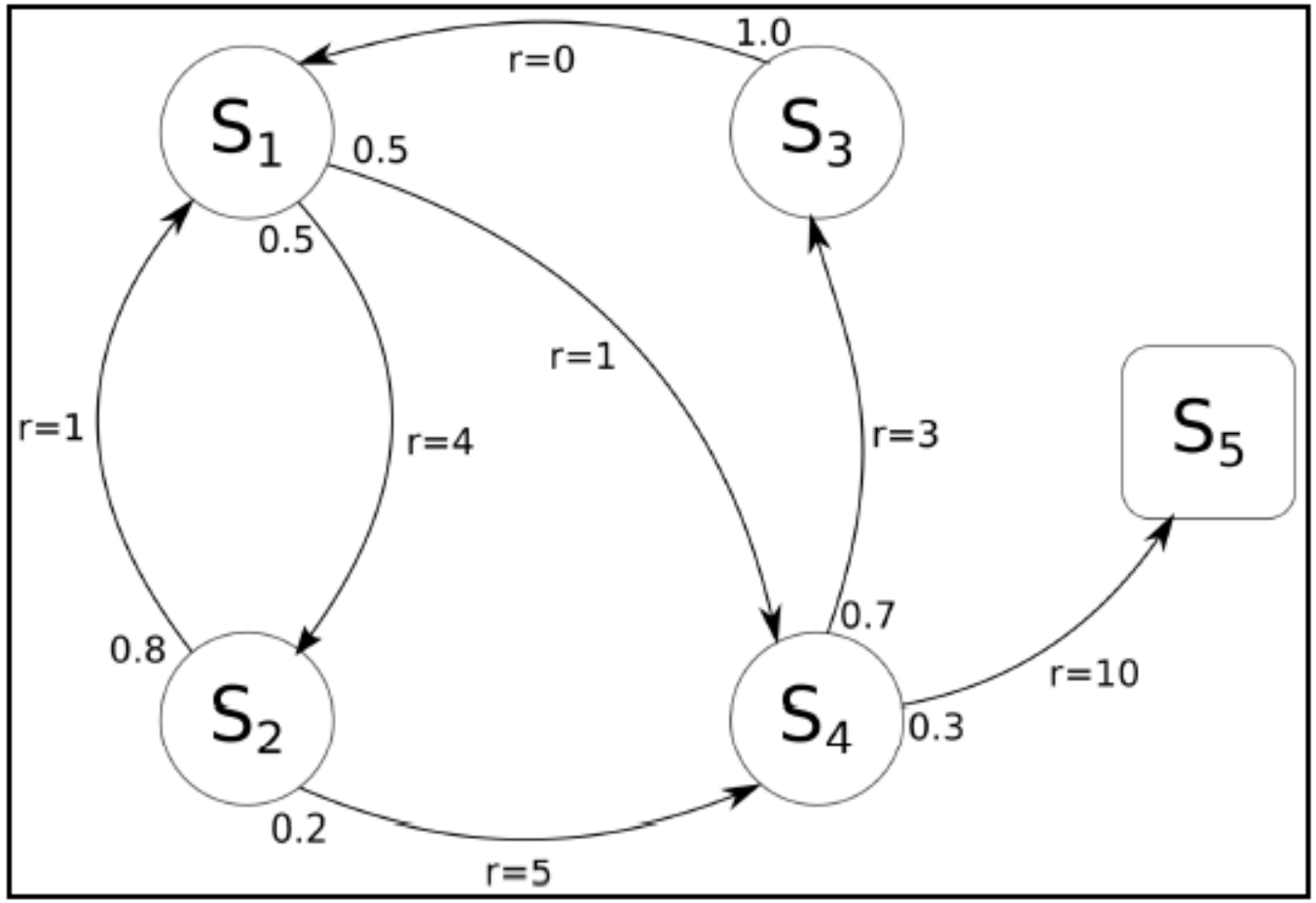
- **Value function**
  - Specifies what is good in the **long-run**
  - Amount of reward expected to **accumulate** over the future
  - Completely dependent on rewards
- **Model**
  - **Mimics** the environment's behavior
  - Used for **planning** course of action before it happens



[1,3]

MPD

- General framework for decision making in stochastic situations
- Formalization of the problem of learning a goal through iterations



S -> State space

A -> Action space

P -> Transition function

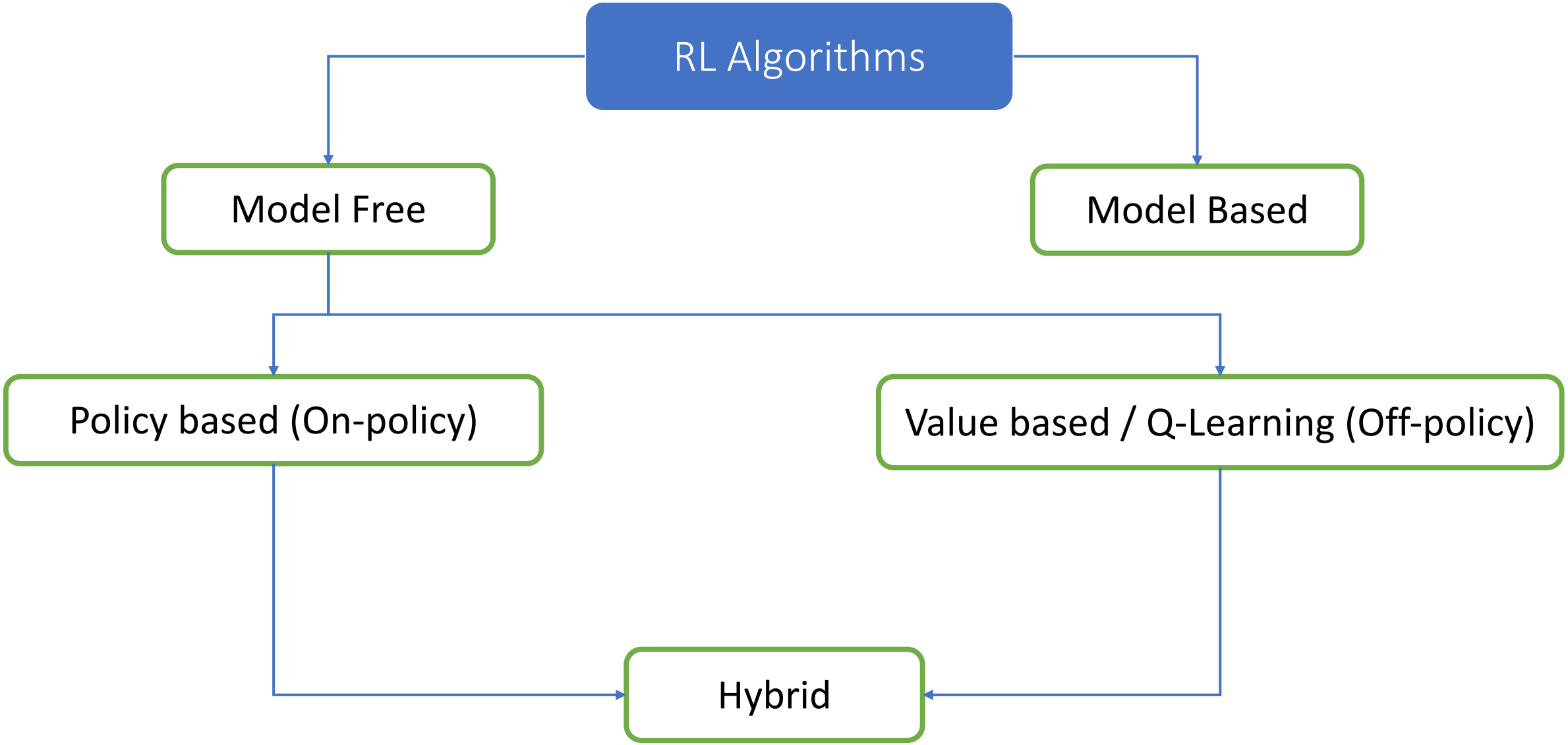
$P(s', s, a) = p(s' \mid s, a)$

R -> Reward function

Markov property: P and R are only determined by current state.

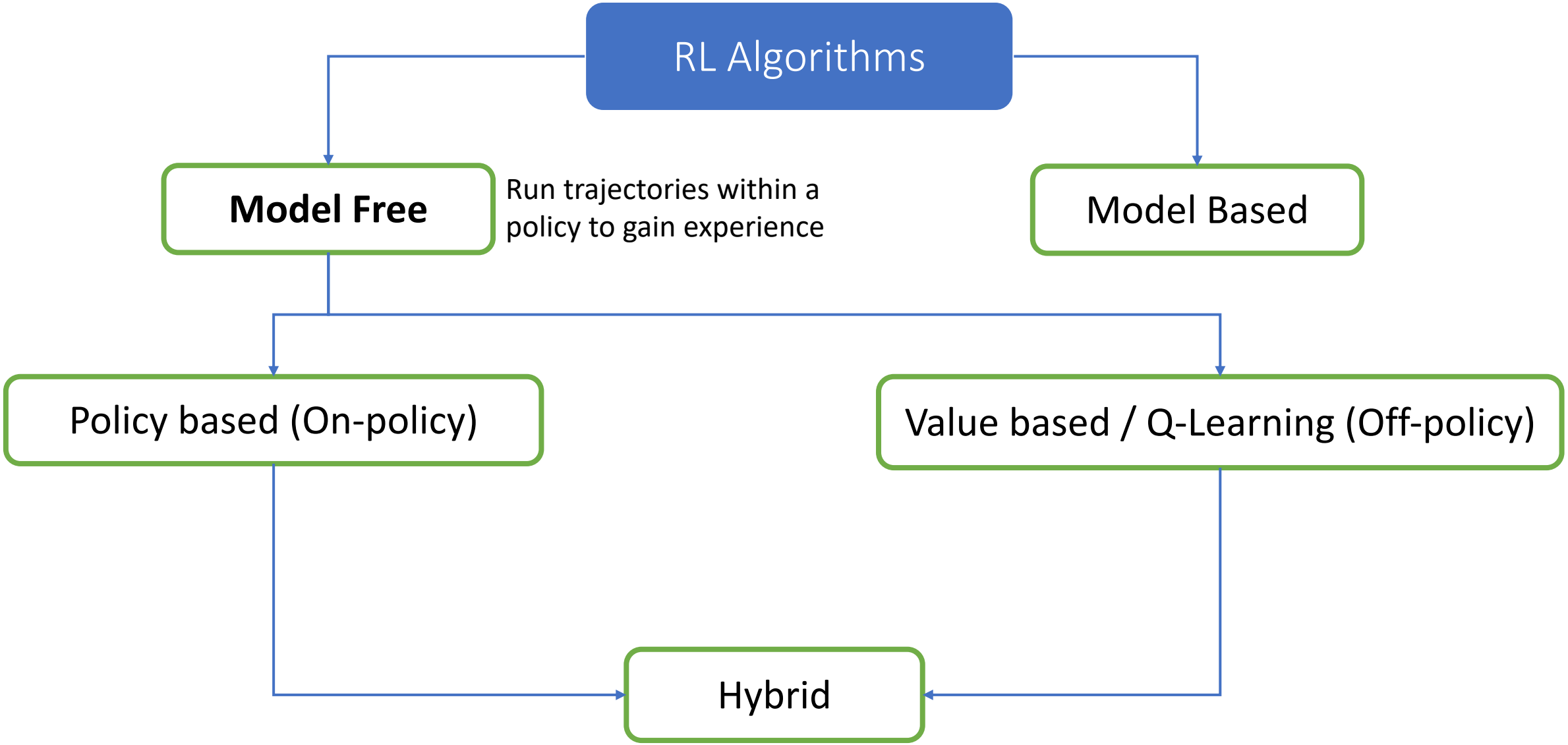
[2]

Categorizing RL Algorithms



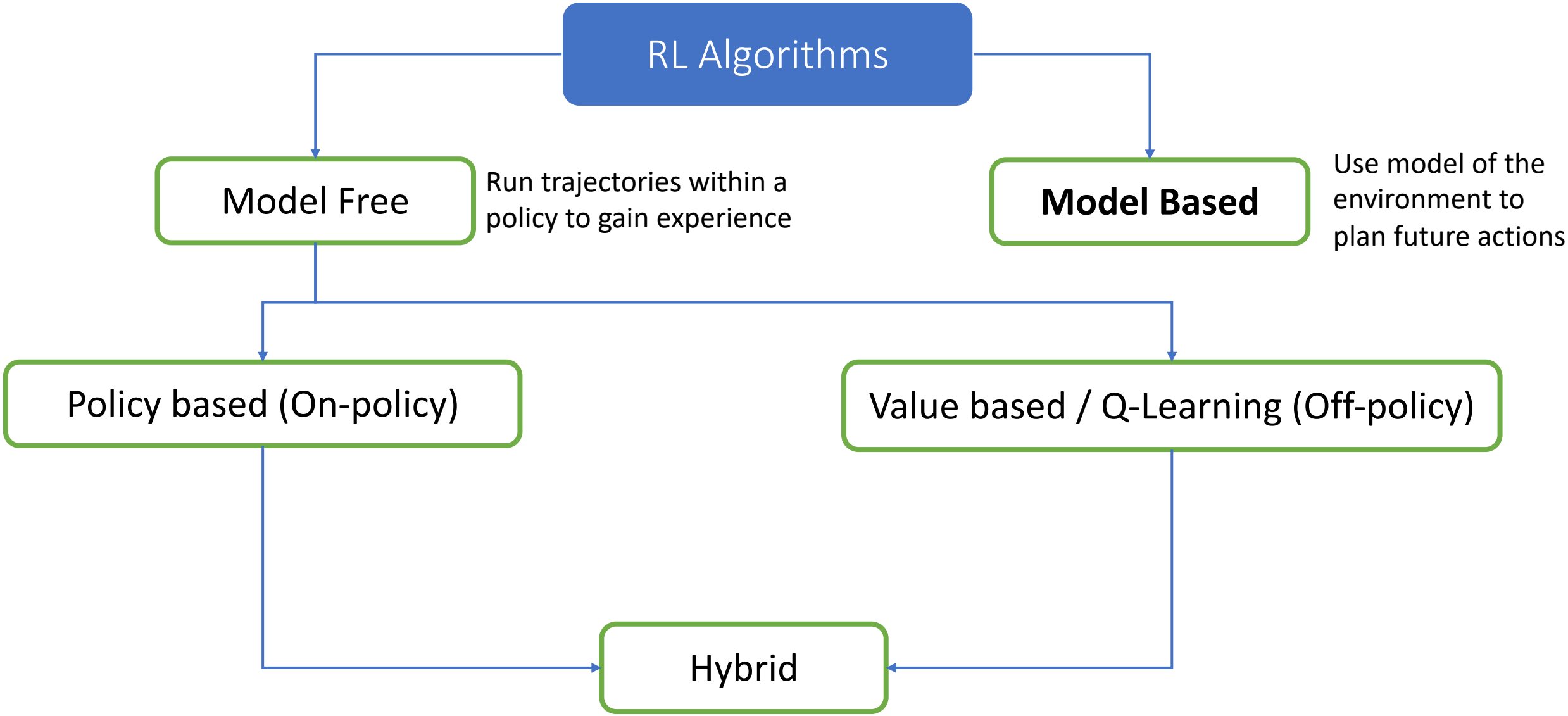
[2]

Categorizing RL Algorithms



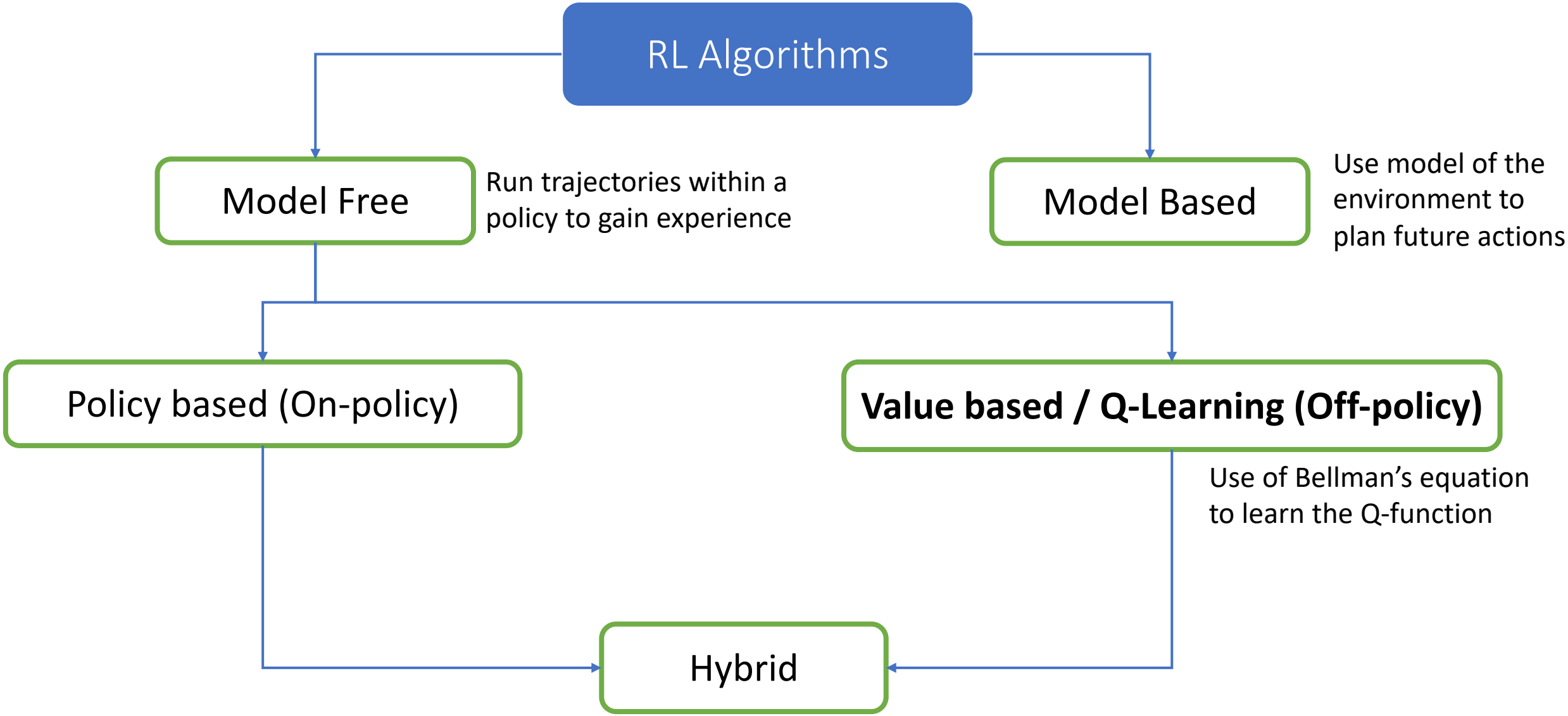
[2]

Categorizing RL Algorithms



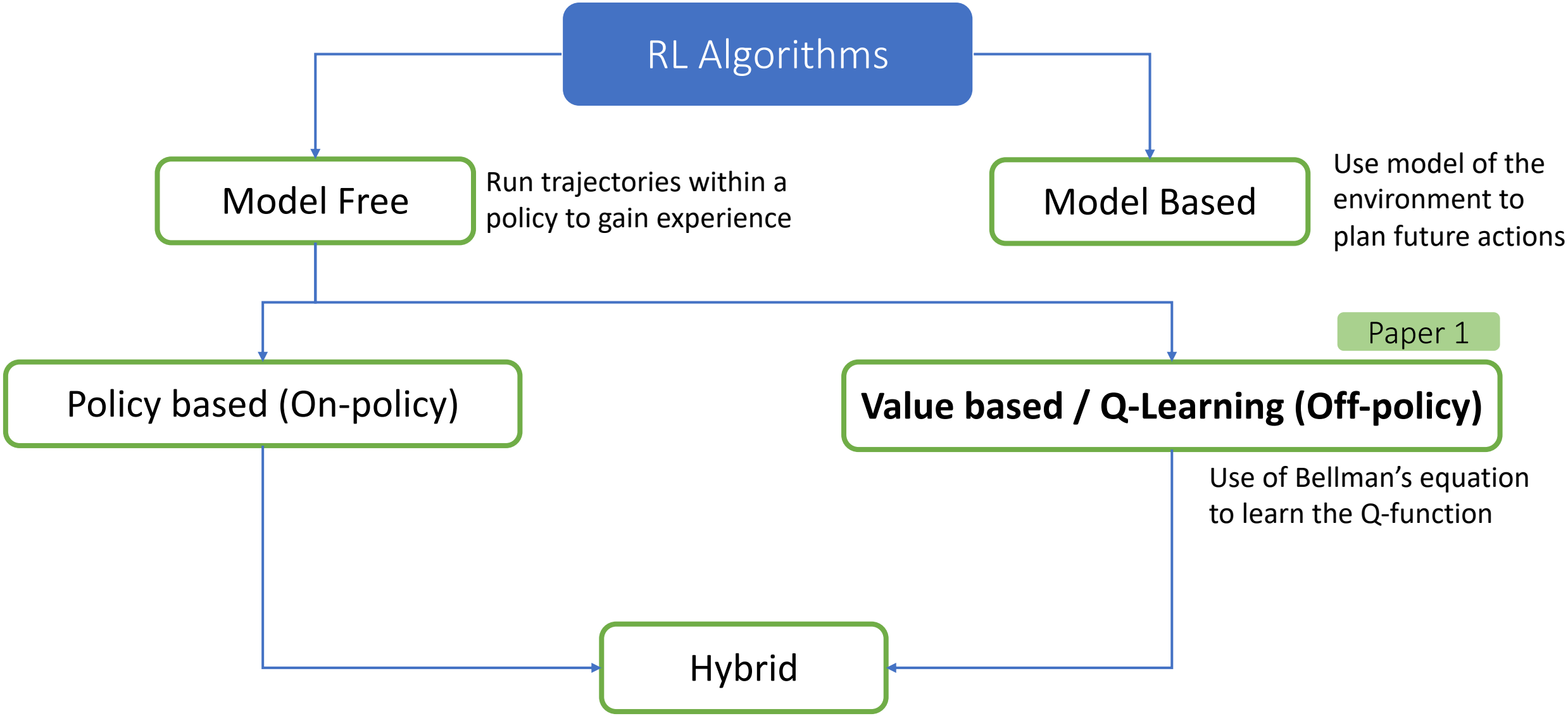
[2]

Categorizing RL Algorithms



[2]

Categorizing RL Algorithms



[2]

Bellman's Equation & Q-function

¿What is the **value** of the **state**?

Basic Bellman's equation: $V(s) = max_a(\mathrm{R(s, a)} + \gamma' \mathrm{V(s')})$

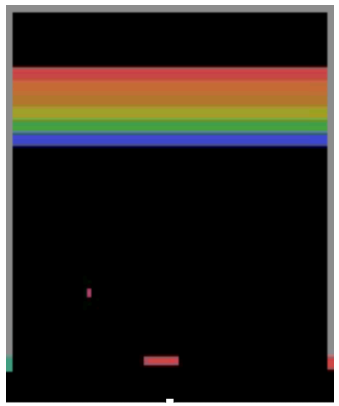
Q- function: Action-value function which obeys Bellman's equation

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right]$$

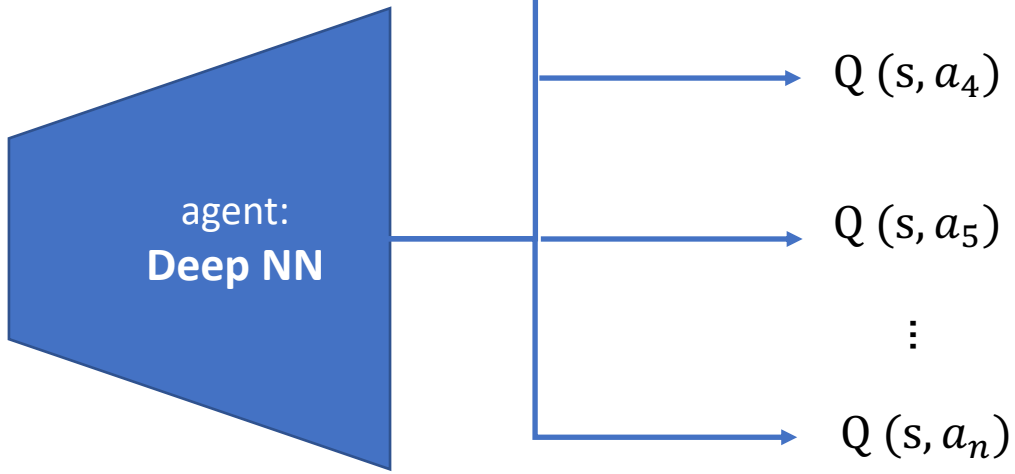
[2]

Deep-Q Networks (DQN)

- It might be hard to determine which pair of state and action *(s,a)* has a higher Q-value.
  - The agent can be a Deep NN to help us solve this problem!



state (s)



CAP: the temporal credit assignment problem

Which actions lead to a certain outcome?

- Simple case: single action = single outcome
- Real case: sequence of actions = outcome

- **Goal**: determine the contribution of each action to the final cumulative reward.
  - Which action should I give more credit to?
  - Will switching the order of previous actions result in a similar outcome?
  - Are the final actions more relevant than the previous ones?

Actions: Dribble, right forward step, left forwqrd step, dribble, left backward step, right backward step, shoot = **score**

[1]

CAP: the structural credit assignment problem

The problem has a structure with multiple elements that contribute to the final outcome.

- Which **element** should I give more credit to?
- Should I **blame** a single element for a negative reward?

Player 1 shoots,
Player 2 is ready for rebound,
Player 3 blocks defense, etc

[1]

Reward function

- Must acknowledge CAP in **episodic** as well as **continuing** tasks.
- Incorporates a **discount rate** for future rewards.

discount rate: determines present value of future rewards

$0 \leq \gamma \leq 1$

expected return

reward sequence

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

timestep

[1]

Effect of γ on expected return

Myopic agent: only concerened with immediate rewards

Farsighted agent: takes future rewards into account

discount rate: determines present value of future rewards

$0 \leq \gamma \leq 1$

expected return

reward sequence

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

timestep

[1]

Exploration vs. exploitation?

Explore the environment so I can discover better paths

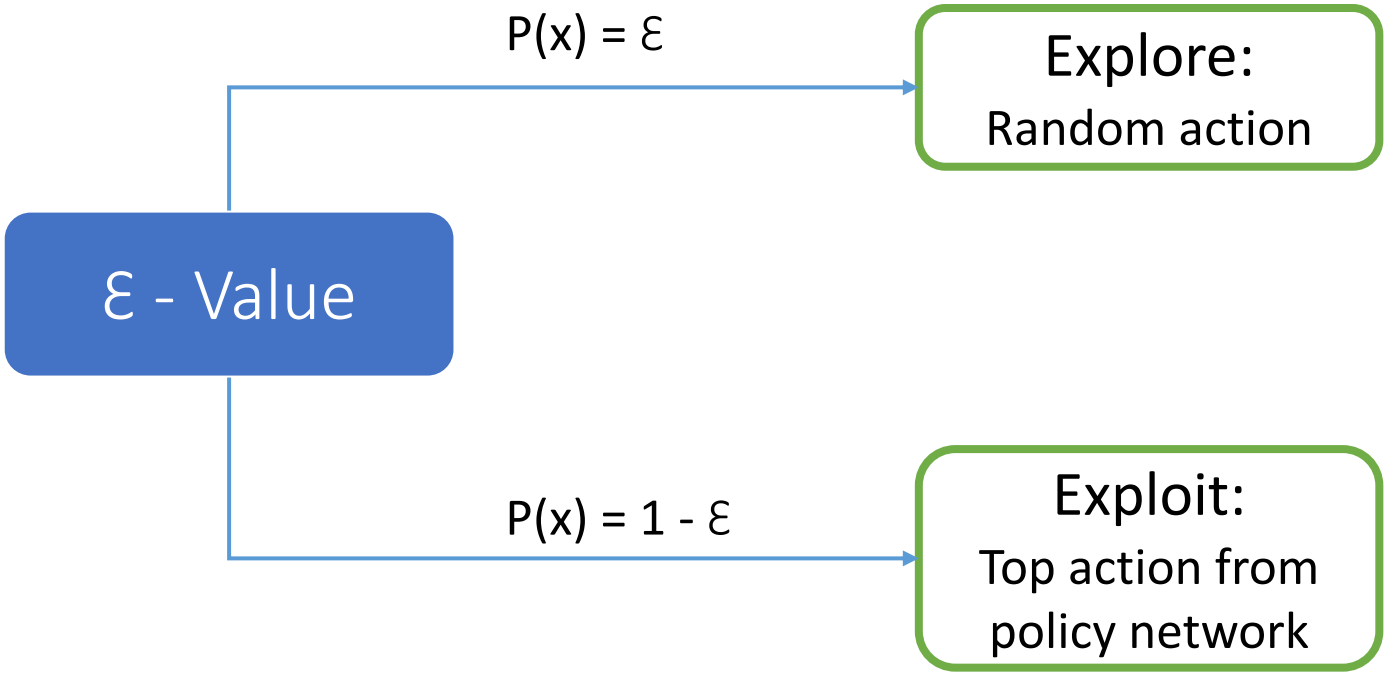
- Try new things to get more information

Stick to what I already know that works and secure points

- What we already know

[2]

Ɛ-greedy exploration

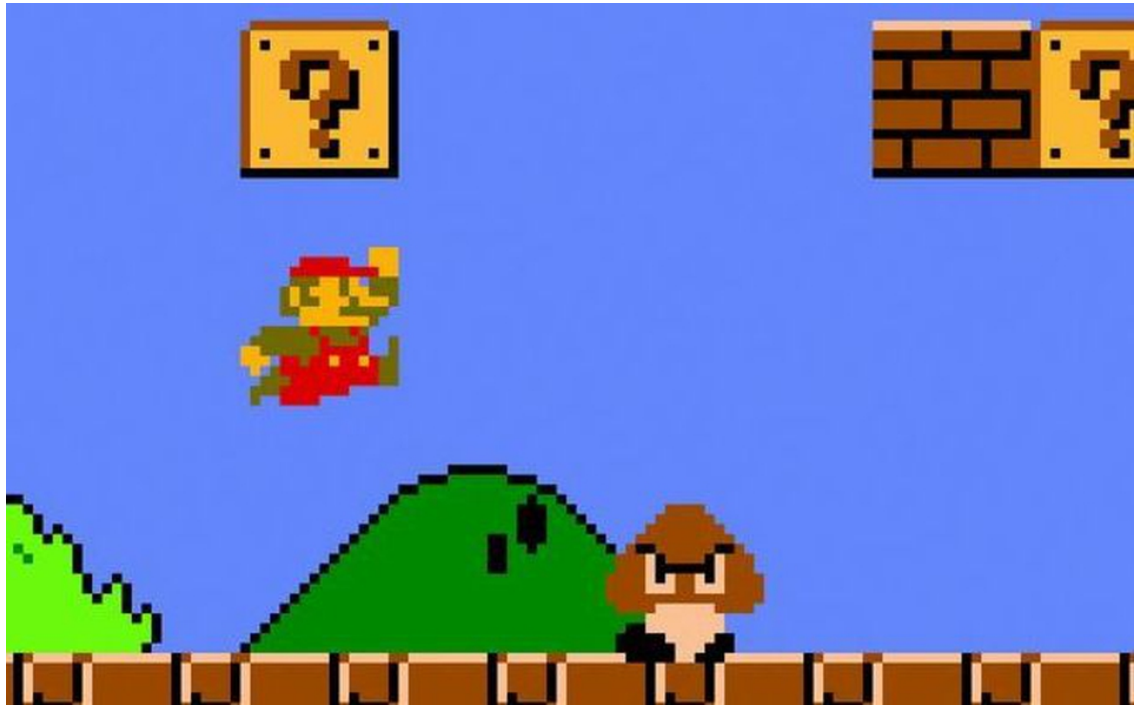


- Probability of **Ɛ** to take a random action
- Incentive your agent to learn about new things in your environment

[2]

Curiosity driven exploration

Paper: “Curiosity driven exploration by self supervised prediction”

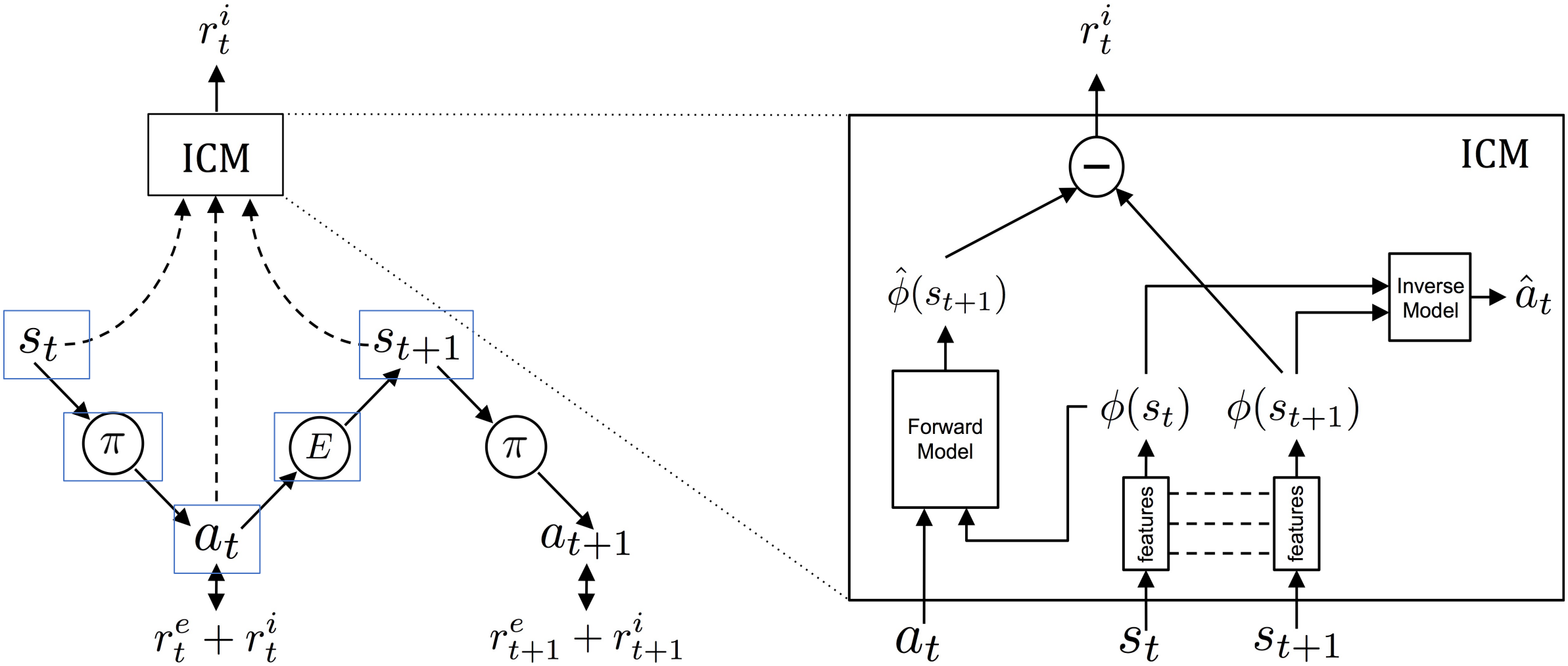


- Tackle the reward sparse problem.
- Get your algorithm to explore new environments by itself.
- We want Mario to be curious about the environment!

[4]

Curiosity driven exploration

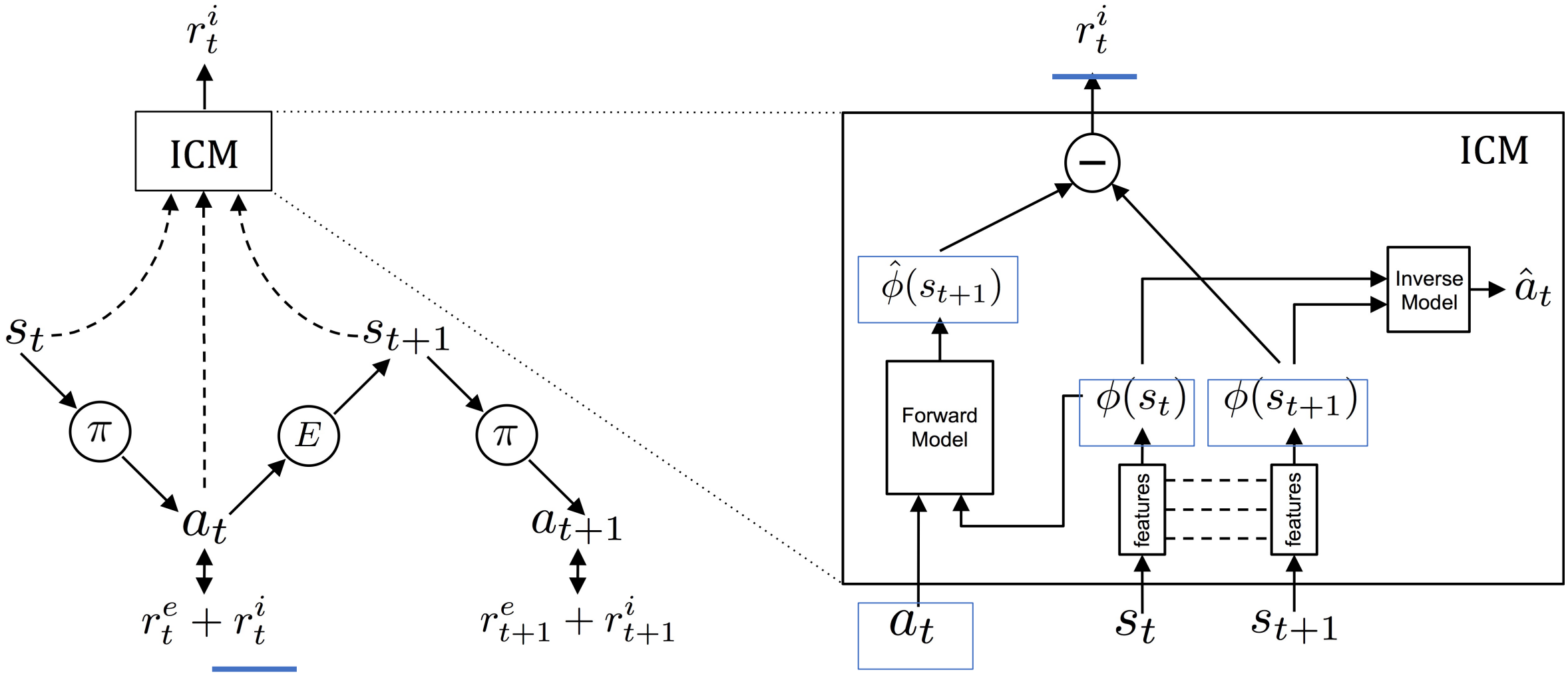
Intrinsic curiosity model (ICM)



[4]

Curiosity driven exploration

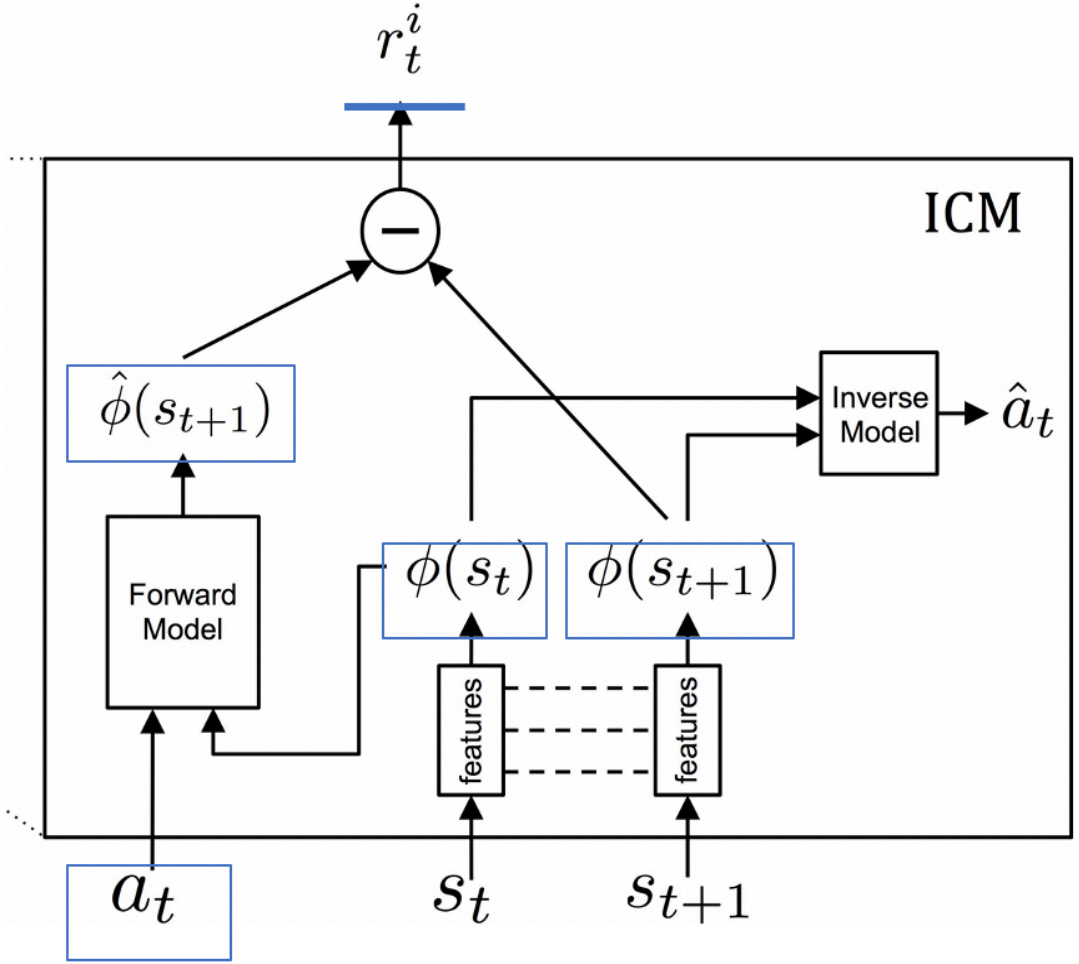
Forward model: Predict the next state, if I'm wrong I'll get a reward.



[4]

Curiosity driven exploration

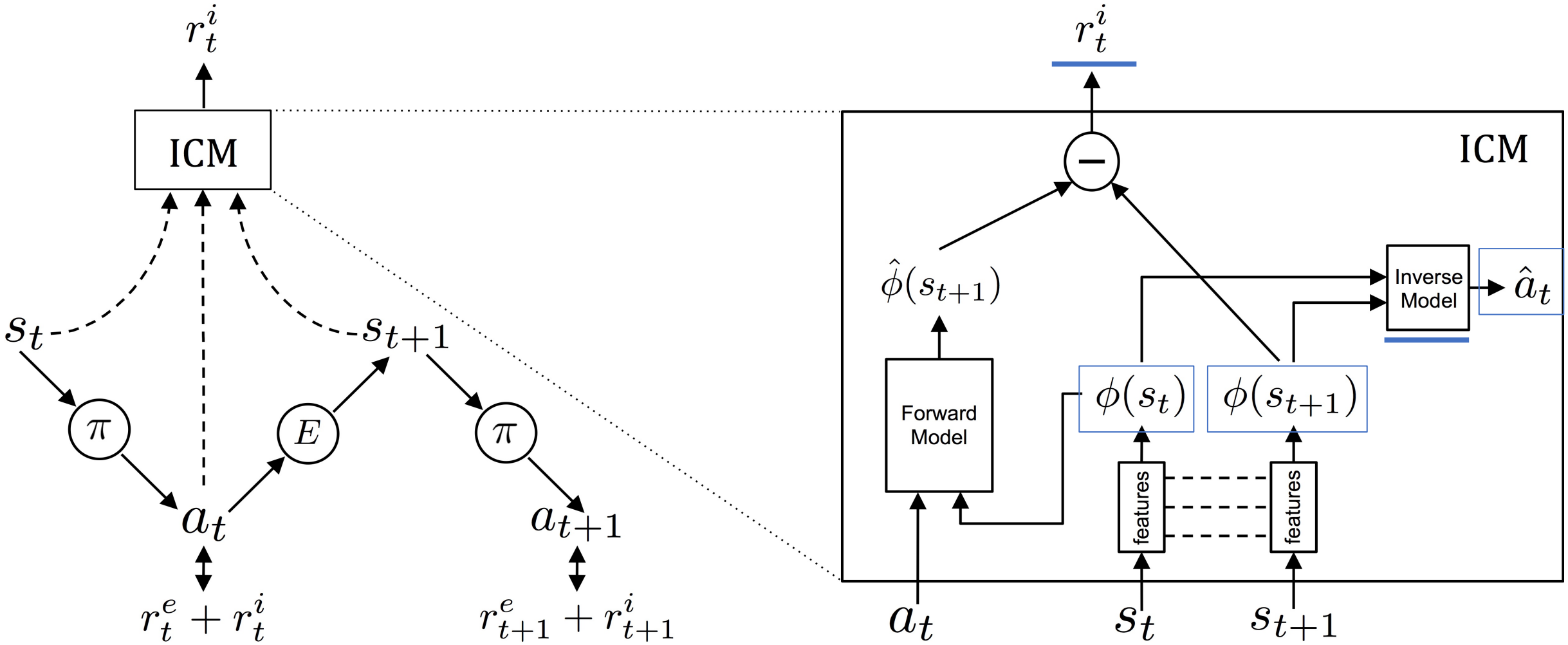
Forward model: Predict the next state, if I'm wrong I'll get a reward.



[4]

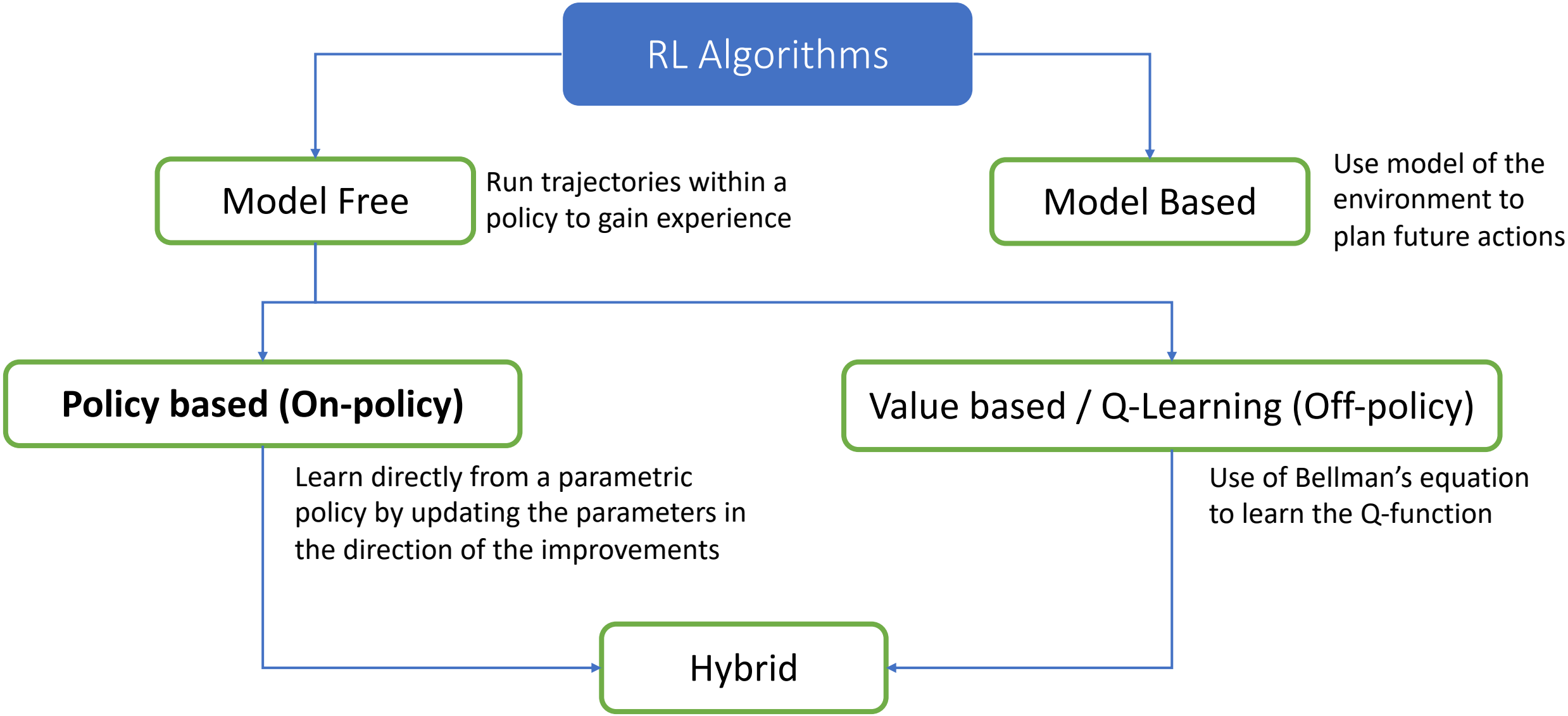
Curiosity driven exploration

Inference model: Predict action token to get from one state to another.



[4]

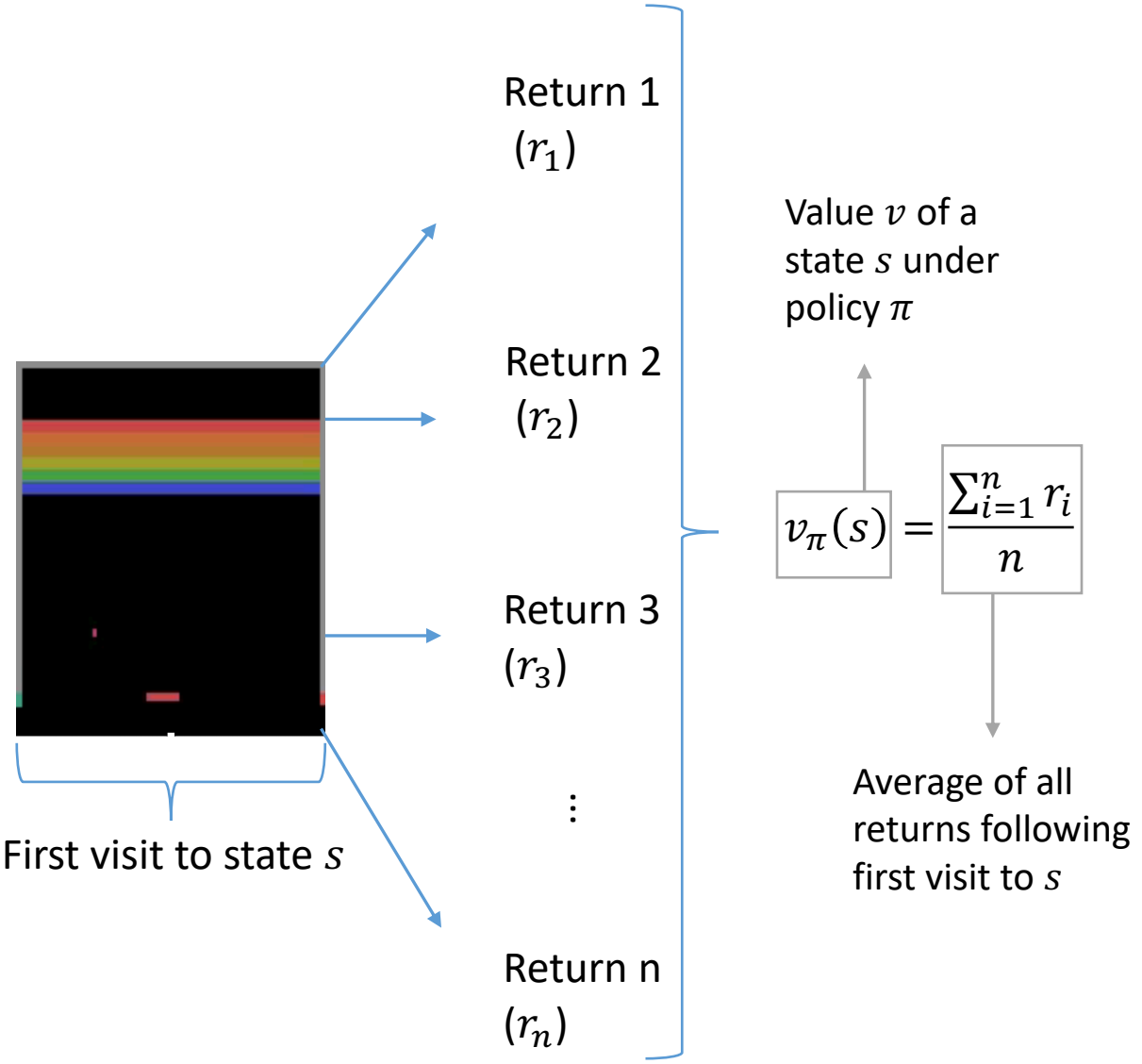
Categorizing RL Algorithms



[2]

First-visit MC method

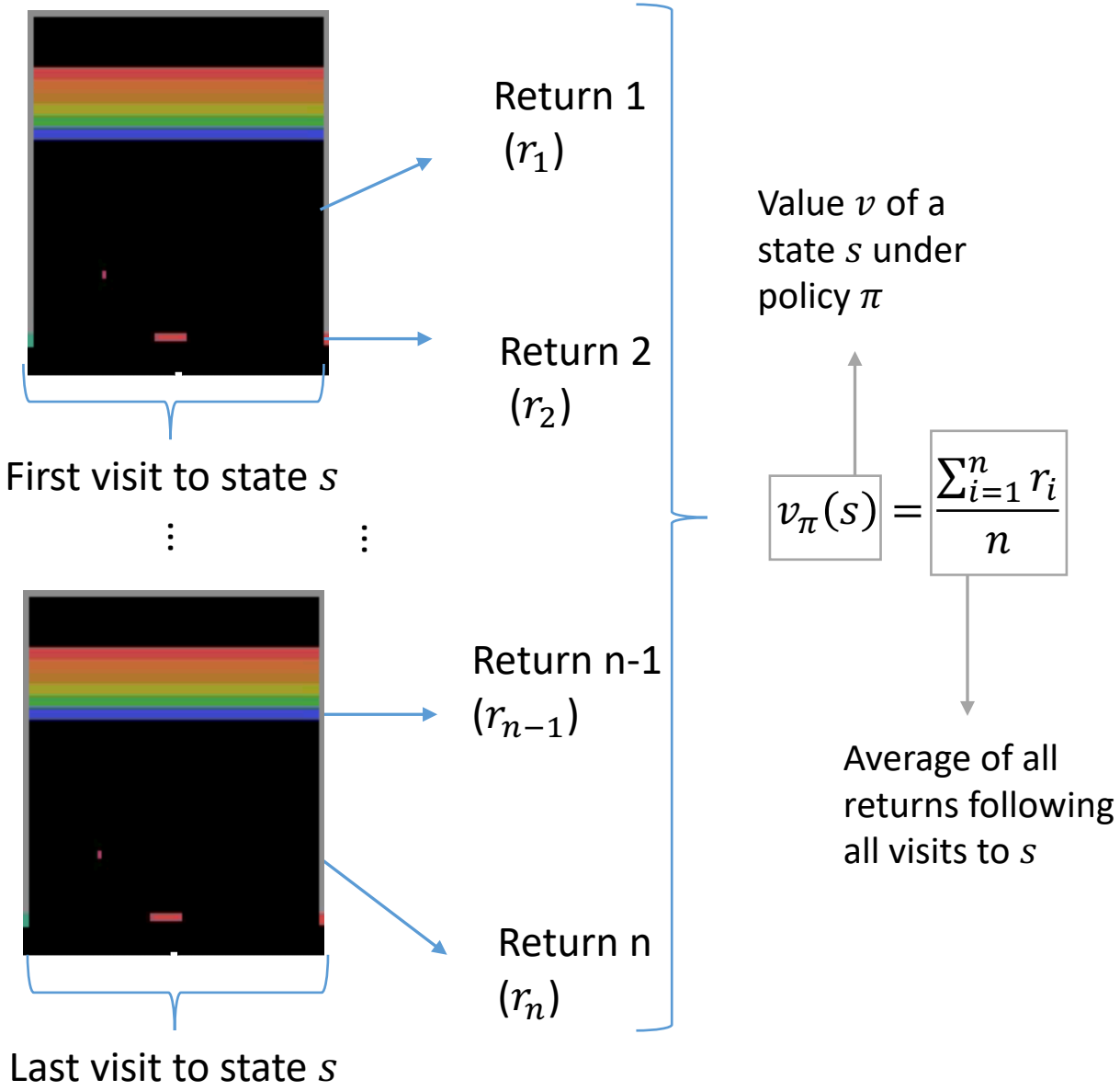
- Doesn't assume complete knowledge of the environment.
- *Learns* value functions from return rather than *computes* value from known MDP.
  - Averages complete returns
- Visit: occurrence of a certain state in an episode
  - First-visit: first occurrence of state s in the espisode



[1]

Every-visit MC method

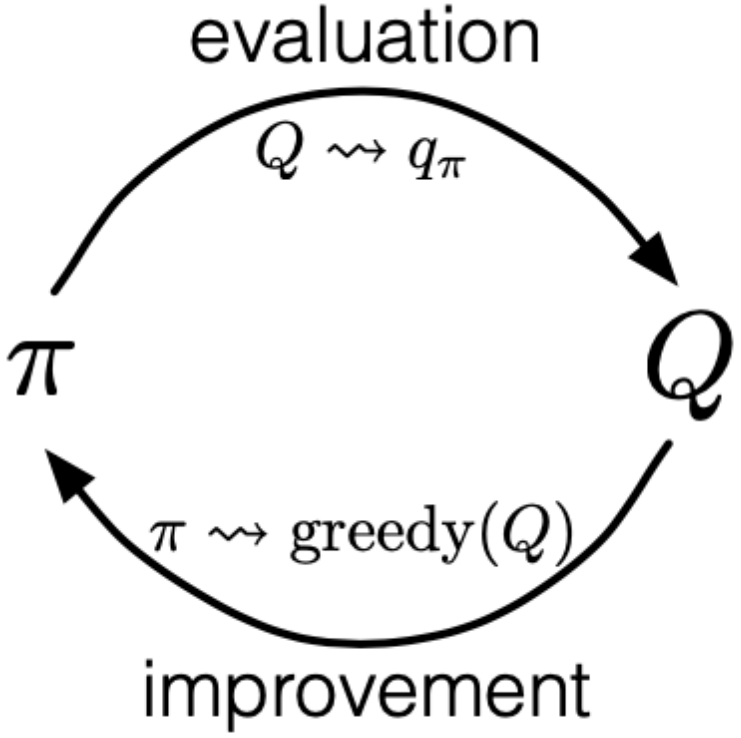
- Averages returns following **all** visits to s in a certain episode
- Extends more naturally to function approximation.
  - Converges to $v_\pi(s)$
- Estimate for each state s is independent of others.
  - Drawback: relies on full trajectory.
  - No bootstrapping!



[1]

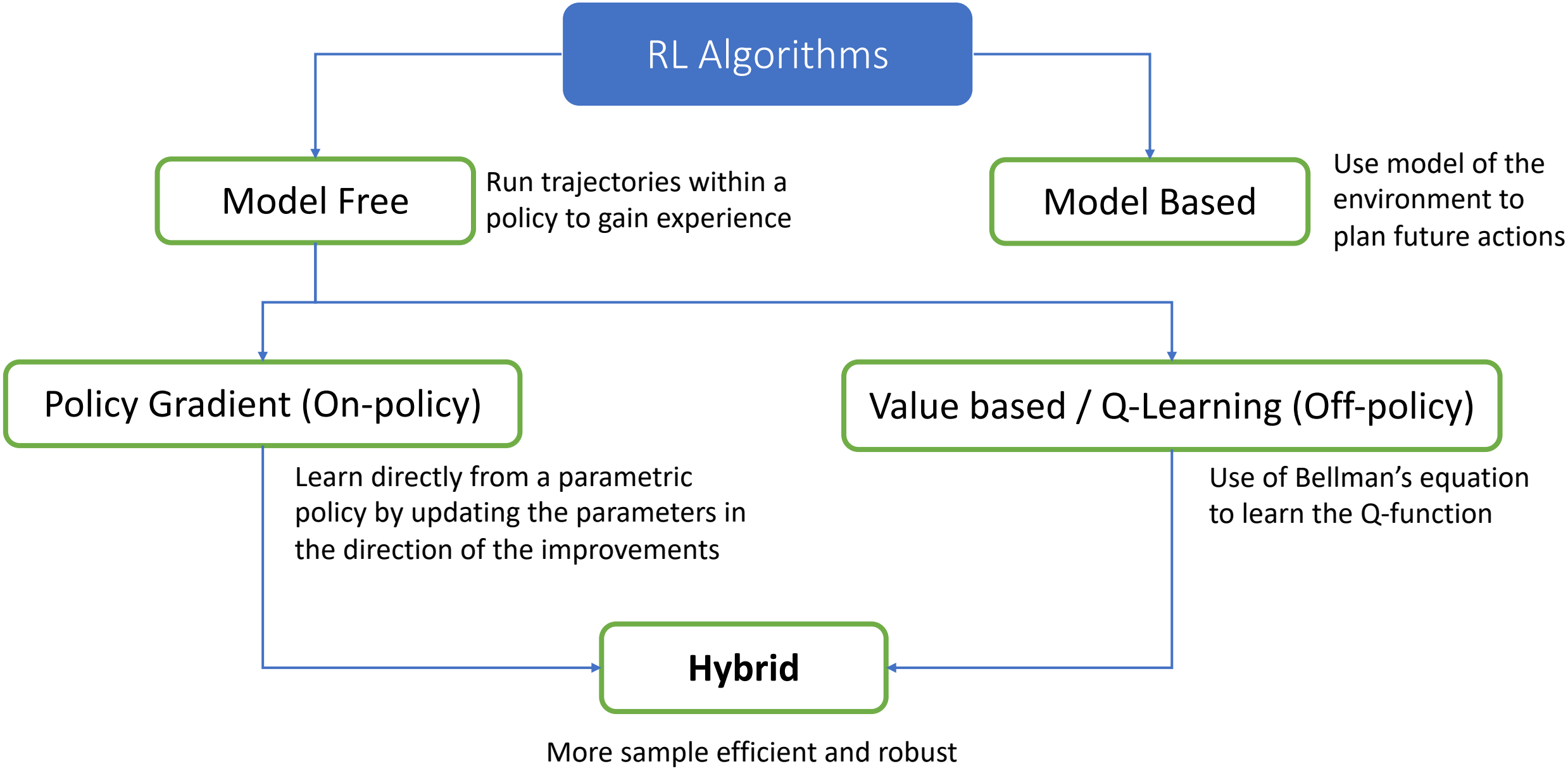
Monte Carlo control

- Proceed according to generalized policy iteration (GPI)
  - **Evaluation**: value function (Q) is altered to more closely approximate the value function for the current policy (q_π)
  - **Improvement**: Policy (π) is repeatedly improved with respect to the current value function (greedy (Q))



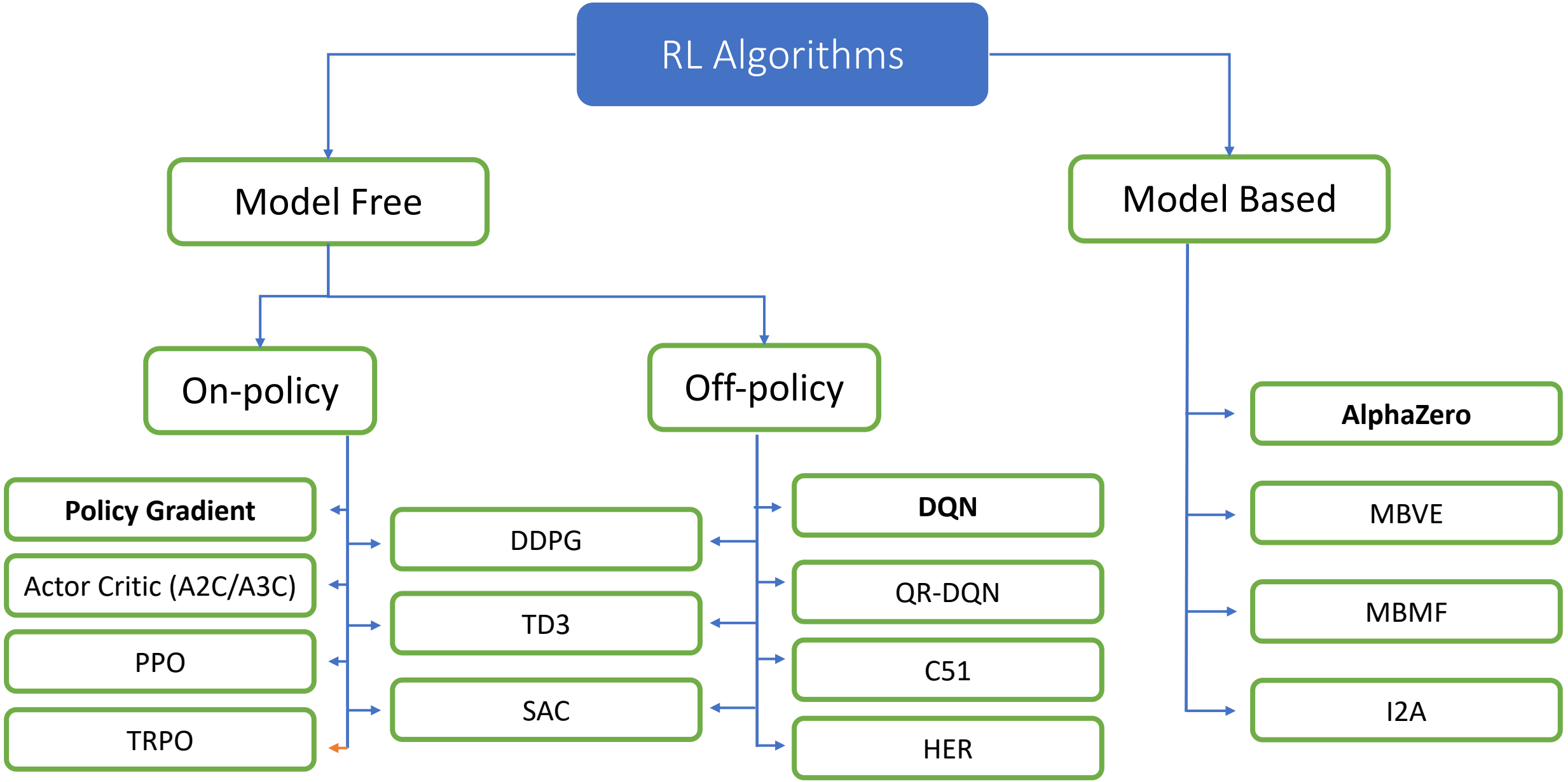
[1]

Categorizing RL Algorithms



[2]

Categorizing RL Algorithms



[10]

Break – 10 min

Kahoot!

Choose any nickname!

Paper 1: Playing Atari with Deep Reinforcement Learning

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Paper 1 (2013)

Paper 1: Playing Atari with Deep Reinforcement Learning

Deep Q-Learning

- **Input:** Raw pixels
- **Output:** Value function with future reward
- **Model- free and off-policy**

What we have learned

- Experience replay mechanism
- Discounted factor γ
- Q – optimal value function
- Ɛ-greedy

Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

Paper 1: Playing Atari with Deep Reinforcement Learning

Algorithm 1 Deep Q-learning with Experience Replay

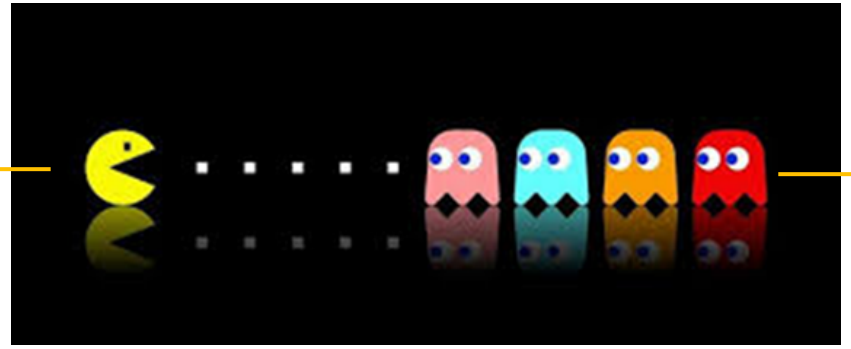
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

[5]

Paper 1: Playing Atari with Deep Reinforcement Learning

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.

DeepMind ←  → Humans

[5]

Paper 2: Discovering Reinforcement Learning Algorithms

Discovering Reinforcement Learning Algorithms

Junhyuk Oh **Matteo Hessel** **Wojciech M. Czarnecki** **Zhongwen Xu**

Hado van Hasselt **Satinder Singh** **David Silver**

Paper 2 (2020)

[11]

Meta-Learning

- Learning how to learn!
  - What should the agent predict?
  - How to improve policy by using those predictions.

Learned Policy Gradient (LPG)

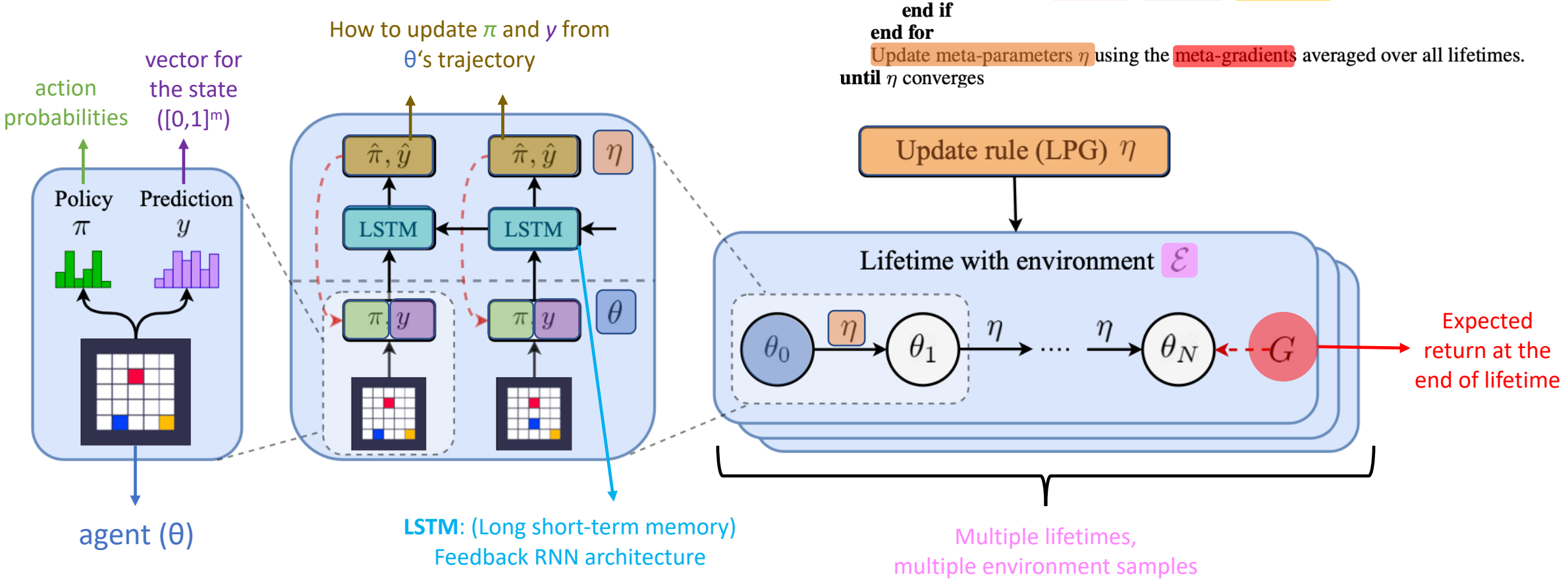
Meta-learner (update rule) decides what predictions should be made whithout relying on the value outputs.

Can this data-driven algorithm compete with traditional value-based approaches?

LPG algorithm

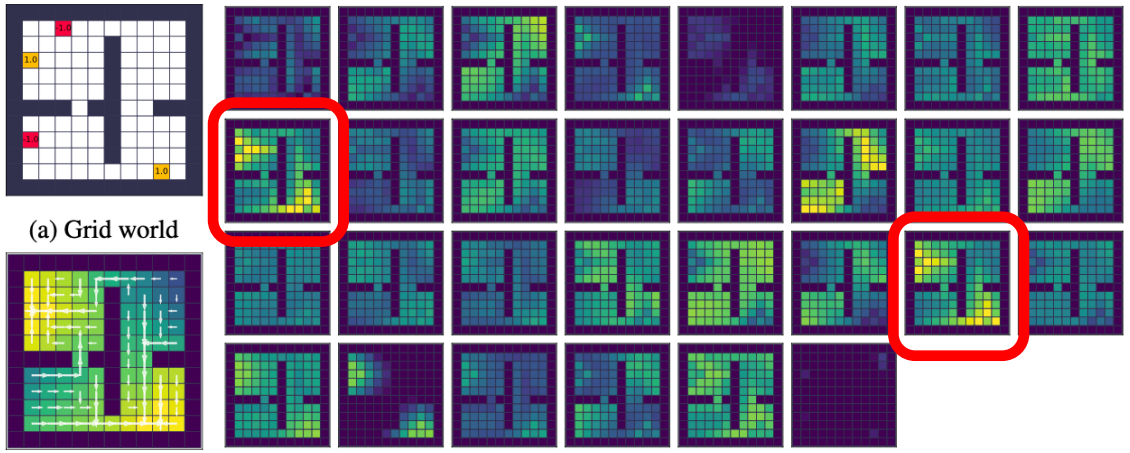
Input: $p(\mathcal{E})$: Environment distribution, $p(\theta_0)$: Initial agent parameter distribution
Initialise meta-parameters η and hyperparameter sampling distribution $p(\alpha|\mathcal{E})$
Sample batch of environment-agent-hyperparameters $\{\mathcal{E} \sim p(\mathcal{E}), \theta \sim p(\theta_0), \alpha \sim p(\alpha|\mathcal{E})\}_i$
repeat
  for all lifetimes $\{\mathcal{E}, \theta, \alpha\}_i$ **do**
    Update parameters θ using η and α for K times
    Compute meta-gradient
    if lifetime ended **then**
      Update hyperparameter sampling distribution $p(\alpha|\mathcal{E})$
      Reset lifetime $\mathcal{E} \sim p(\mathcal{E})$, $\theta \sim p(\theta_0)$, $\alpha \sim p(\alpha|\mathcal{E})$
    end if
  end for
  Update meta-parameters η using the meta-gradients averaged over all lifetimes.
until η converges

Optimizer, LR, γ, etc.

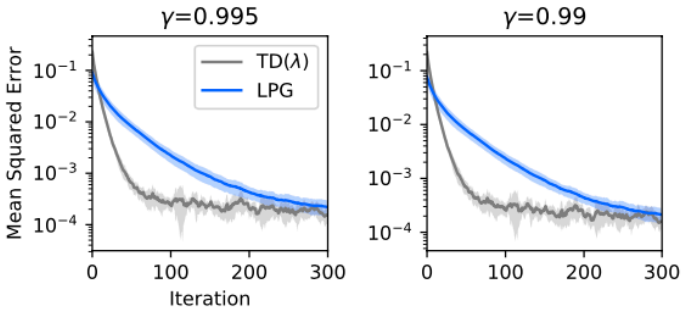


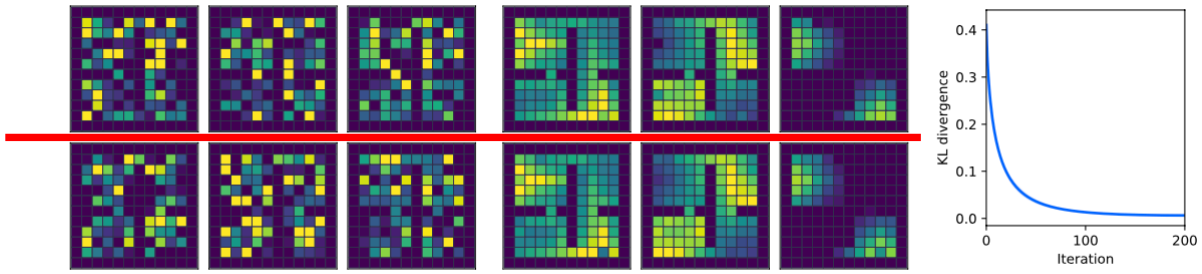
Predictions

- What do they look like?
- Do they capture true values?
  - Sometimes even better than TD(λ).
  - Dependent on hyperparameters.
- Do they always converge?
  - Not guaranteed, but usually they do.



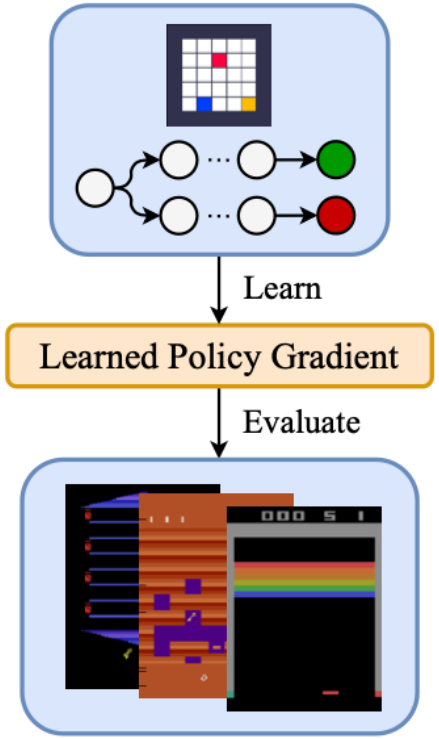
(a) Grid world (b) Policy and value (c) Predictions (y)

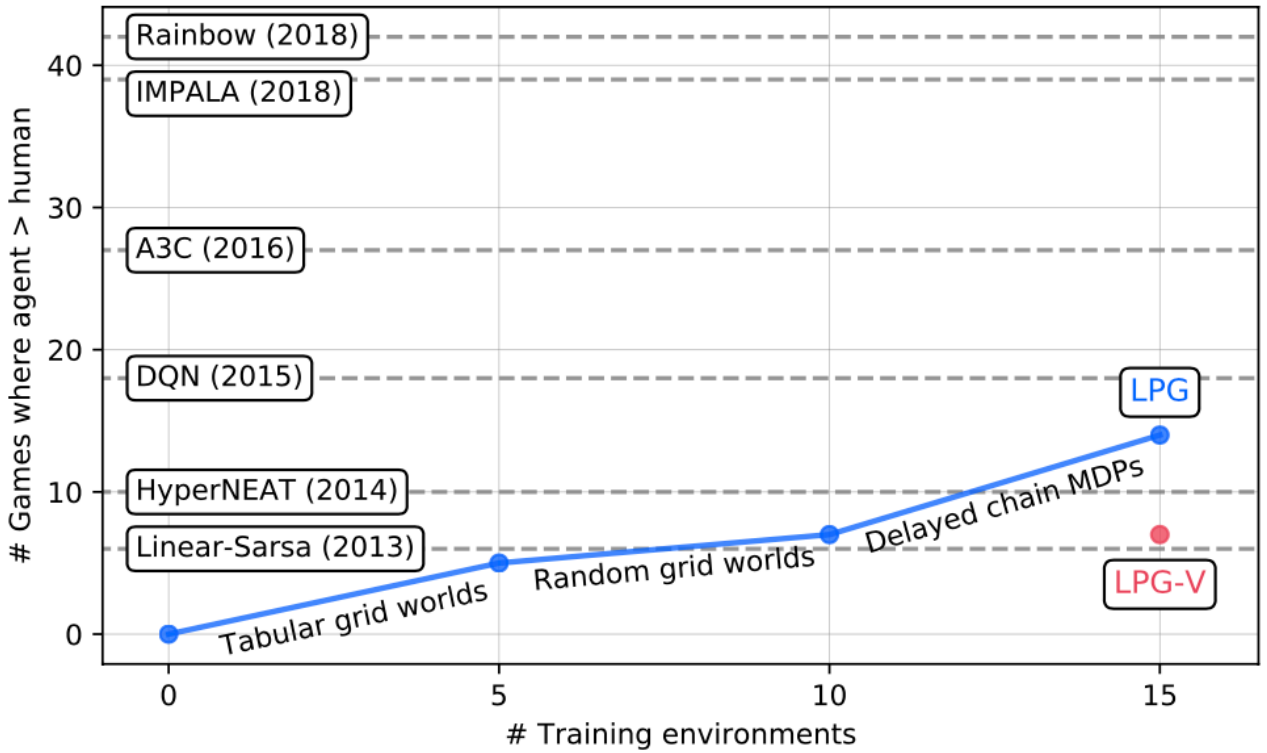


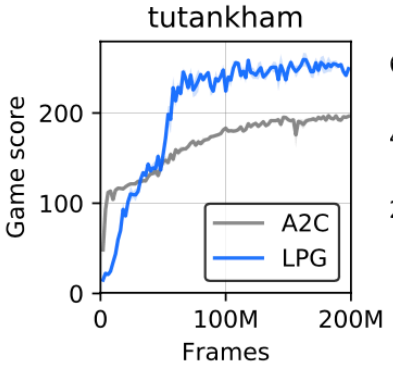


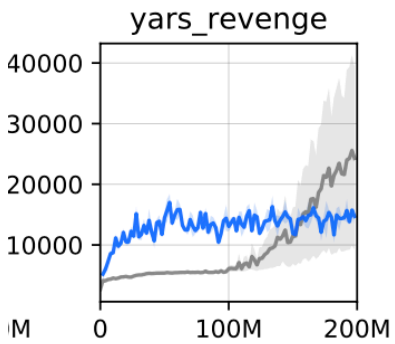
(a) Initial predictions (b) After convergence (c) KL divergence

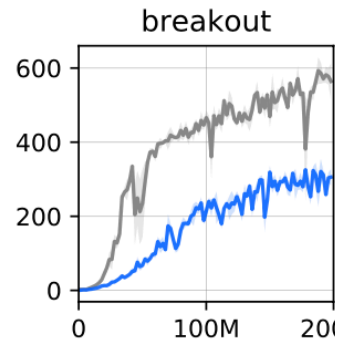
LPG vs. others











[11]

Broader Impact

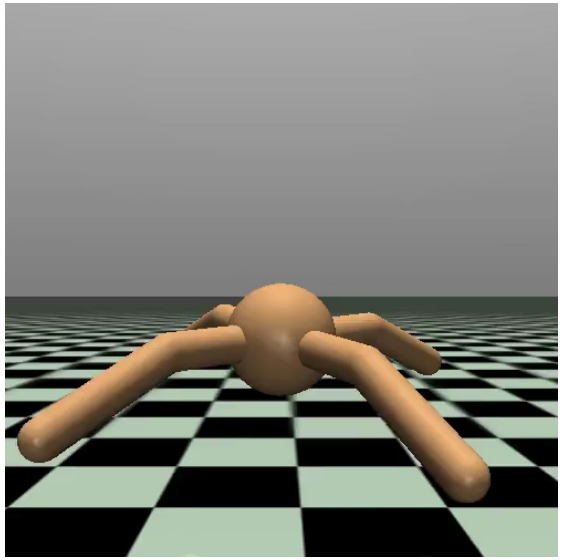
"Potential to **dramatically accelerate** the process of **discovering new RL algorithms** by **automating** the process of discovery in a data-driven way. If the proposed research direction succeeds, this could **shift the research paradigm** from manually developing RL algorithms to building a proper set of environments so that the resulting algorithm is efficient."

GYM Tutorial

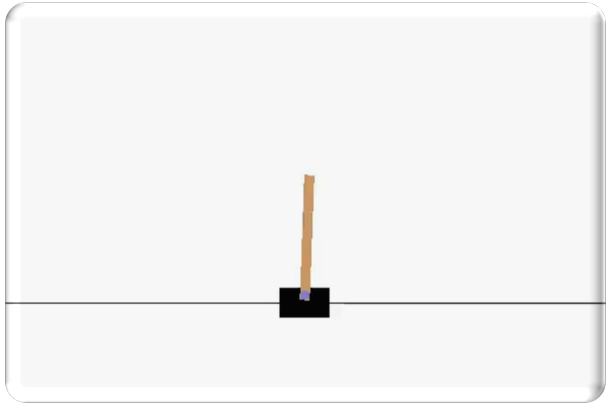
-> Open source interface for reinforcement leaning tasks

-> Toolkit for developing and comparing reinforcement learning algorithms

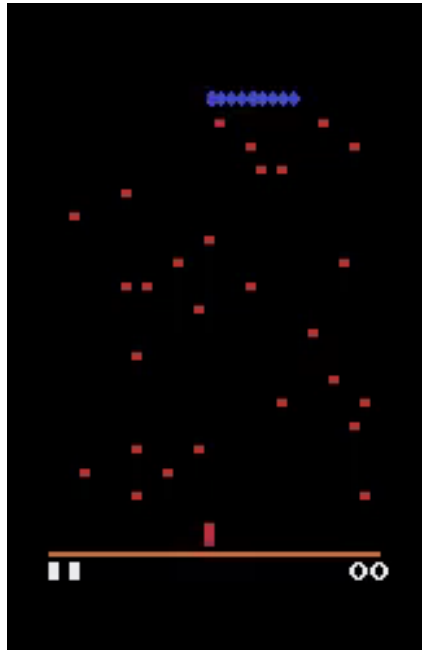
-> All kind of **environments** available



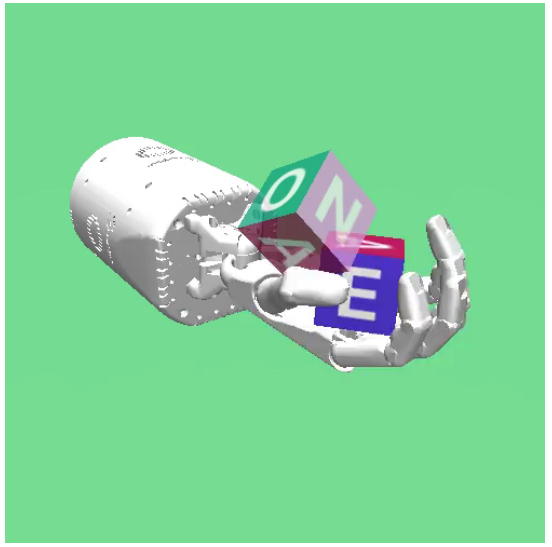
MuJoCo



Classic control



Atari



Robotics

[7]

GYM Tutorial + Understanding RL code

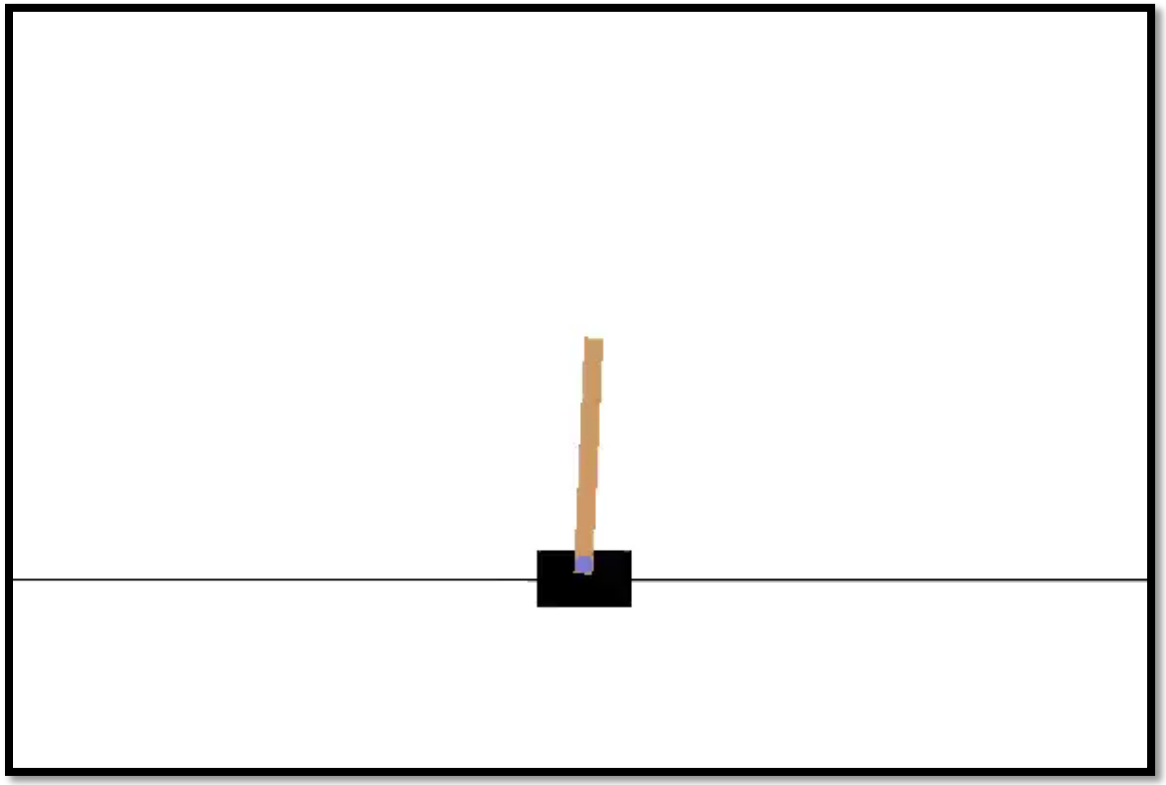
Let’s start with something easy

```
import gym
env = gym.make('CartPole-v0')
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
env.close()
```

GYM Tutorial + Understanding RL code

You can run the code on your local computer. Results should look like this!

```
[-0.061586   -0.75893141  0.05793238  1.15547541]
[-0.07676463 -0.95475889  0.08104189  1.46574644]
[-0.0958598  -1.15077434  0.11035682  1.78260485]
[-0.11887529 -0.95705275  0.14600892  1.5261692 ]
[-0.13801635 -0.7639636   0.1765323   1.28239155]
[-0.15329562 -0.57147373  0.20218013  1.04977545]
Episode finished after 14 timesteps
[-0.02786724  0.00361763 -0.03938967 -0.01611184]
[-0.02779488 -0.19091794 -0.03971191  0.26388759]
[-0.03161324  0.00474768 -0.03443415 -0.04105167]
```



GYM Tutorial + Understanding RL code

Select your environment

```
env = gym.make('CartPole-v1')
env.seed(args.seed)
torch.manual_seed(args.seed)
```

Function for selecting action

```
def select_action(state):
    state = torch.from_numpy(state).float().unsqueeze(0)
    probs = policy(state)
    m = Categorical(probs)
    action = m.sample()
    policy.saved_log_probs.append(m.log_prob(action))
    return action.item()
```

Policy class

```
class Policy(nn.Module):
    def __init__(self):
        super(Policy, self).__init__()
        self.affine1 = nn.Linear(4, 128)
        self.dropout = nn.Dropout(p=0.6)
        self.affine2 = nn.Linear(128, 2)

        self.saved_log_probs = []
        self.rewards = []

    def forward(self, x):
        x = self.affine1(x)
        x = self.dropout(x)
        x = F.relu(x)
        action_scores = self.affine2(x)
        return F.softmax(action_scores, dim=1)


policy = Policy()
optimizer = optim.Adam(policy.parameters(), lr=1e-2)
eps = np.finfo(np.float32).eps.item()
```

[6]

GYM Tutorial + Understanding RL code

Main code for
Car Pole tutorial

```
def main():
    running_reward = 10
    #frames = []
    for i_episode in count(1):
        state, ep_reward = env.reset(), 0
        for t in range(1, 10000):  # Don't infinite loop while learning
            action = select_action(state)
            state, reward, done, _ = env.step(action)
            if args.render:
                env.render()
            #frames.append(env.render(mode="rgb_array"))
            policy.rewards.append(reward)
            ep_reward += reward
            if done:
                break

        running_reward = 0.05 * ep_reward + (1 - 0.05) * running_reward
        finish_episode()
        if i_episode % args.log_interval == 0:
            print('Episode {}\tLast reward: {:.2f}\tAverage reward: {:.2f}'.format(
                  i_episode, ep_reward, running_reward))
        if running_reward > env.spec.reward_threshold:
            print("Solved! Running reward is now {} and "
                  "the last episode runs to {} time steps!".format(running_reward, t))
            break
```

[6]

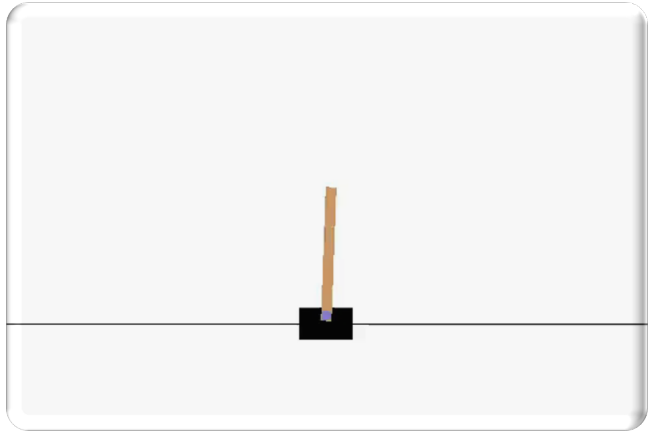
GYM Tutorial + Understanding RL code

You should see something like this!

```
Episode 480     Last reward: 500.00     Average reward: 406.31
Episode 490     Last reward: 500.00     Average reward: 435.96
Episode 500     Last reward: 63.00      Average reward: 419.28
Episode 510     Last reward: 17.00      Average reward: 323.39
Episode 520     Last reward: 175.00     Average reward: 261.57
Episode 530     Last reward: 217.00     Average reward: 240.43
Episode 540     Last reward: 63.00      Average reward: 237.48
Episode 550     Last reward: 347.00     Average reward: 311.83
Episode 560     Last reward: 500.00     Average reward: 374.02
Episode 570     Last reward: 500.00     Average reward: 424.57
Episode 580     Last reward: 500.00     Average reward: 454.84
Episode 590     Last reward: 500.00     Average reward: 472.96
Solved! Running reward is now 475.5955691464257 and the last episode runs to 500 time steps!
(rl) iramos@bcv001:~/reinforcement-learning$
```

→ Try to maximize your reward

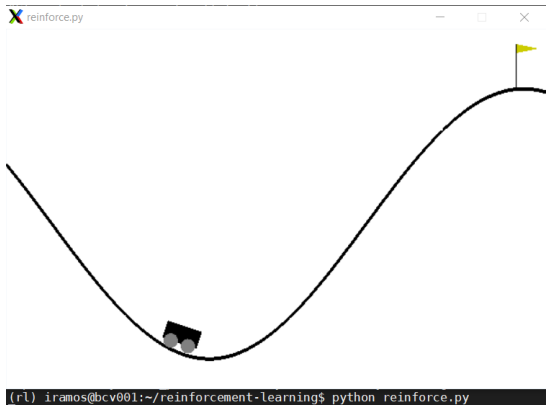
Try running the code on MobaXterm or your local computer to see the CartPole

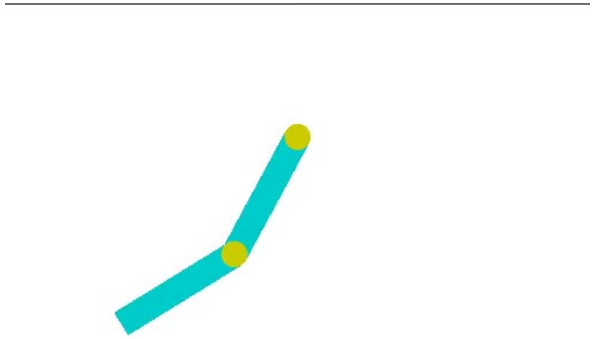


[6]

Homework

- (2 points) Explore and understand the code. Change hyperparameters and try to change the Policy class by adding or modifying layers. Show your results in an organized table and discuss them. What do you think lead to your best result? Which changes caused the most variation?
- (1 point) For two of your best results, graph Episodes vs Average Reward. Was the graph what you were expecting? Analyze.
- (2 points) Try a different environment! Choose between Acrobot-v1 and MountainCar-v0. What changed compared to the previous environment? How well did the agent perform in your chosen environment?
- **Bonus:** Investigate about RL environment packages other than OpenAI-Gym. How might they be useful for different tasks?





Abbreviation glossary

RL: Reinforcement Learning

MDP: Markov decision process

CAP: credit assignment problem

ICM: intrinsic curiosity model

DP: dynamic programming

TD: temporal difference learning

MC: Monte Carlo

GPI: Generalized policy iteration

DQN: Deep-Q Networks

LPG: Learned Policy Gradient

References

Textbook references:

[1] “Reinforcement Learning an introduction” Sutton R.S, Barto A.G. Adaptive Computation and Machine Learning

[2] “Reinforcement Learning algorithms with Python” Lonza A. Packt Birmingham - Mumbai

Other references:

[3] Uc-Cetina, Víctor. (2013). A Novel Reinforcement Learning Architecture for Continuous State and Action Spaces. Advances in Artificial Intelligence. 2013. 10.1155/2013/492852.

[4] Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 16-17)

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[6] Chintala et. Al. Pytorch Examples (2020) Reinforcement learning. GitHub link: https://github.com/pytorch/examples.git

[7] GYM (2020), Getting started with Gym. https://gym.openai.com/docs/

[8] Alexander Amini and Ava Soleimany (2020) MIT 6.S191: Introduction to Deep Learning IntroToDeepLearning.com

[9] Atari (2020). About Us: Game and Hardware Timeline. https://www.atari.com/about-us/

[10] OpenAI (2018). A Taxonomy of RL Algorithms. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

[11] Oh, J., Hessel, M., Czarnecki, W. M., Xu, Z., van Hasselt, H., Singh, S., & Silver, D. (2020). Discovering Reinforcement Learning Algorithms. arXiv preprint arXiv:2007.08794.

Reinforcement Learning (RL)

Advanced Machine Learning – IBIO Uniandes

September 7, 2020

By: Daniela Tamayo & Isabella Ramos

Tutors: Laura Daza, Catalina Gómez