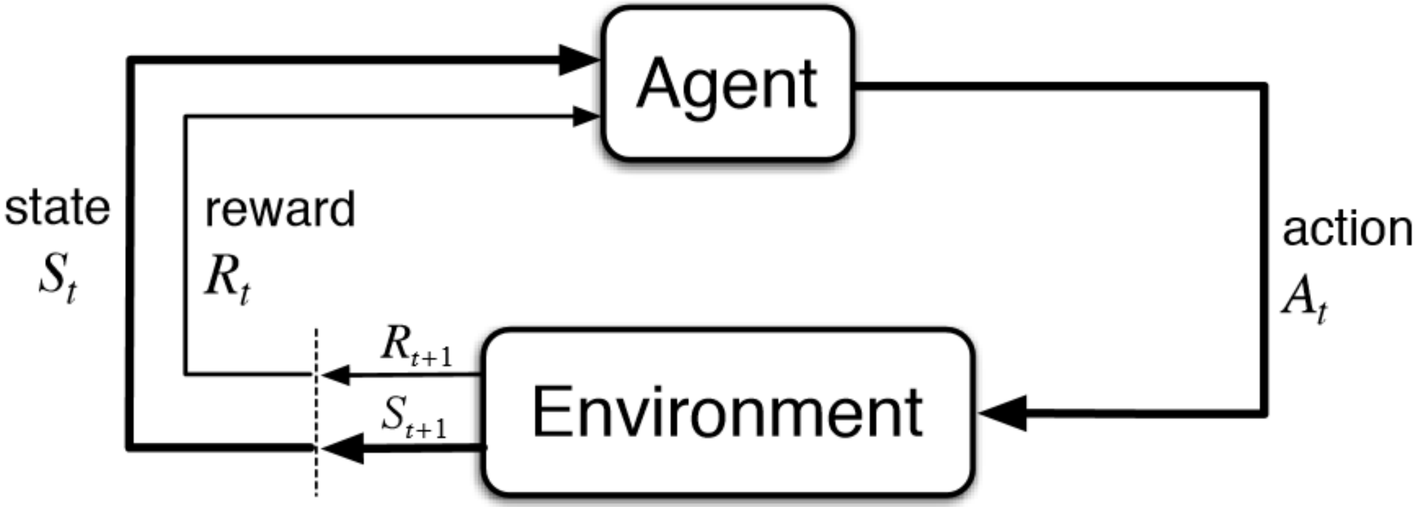
Reinforcement Learning

ICS/DATA 435 and ICS 635

Spring 2023

Reinforcement learning (RL)

Learning by experience (RL) vs learning by examples (supervised learning)



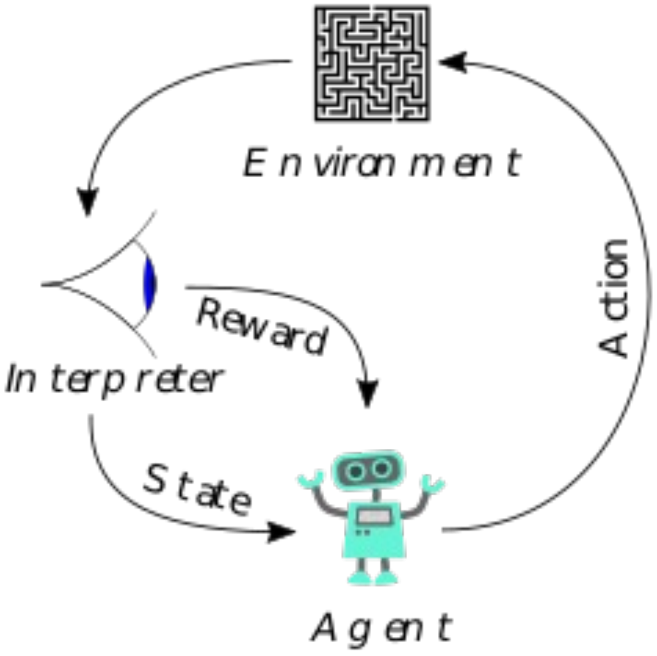
https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

When is RL used?

- Autonomous vehicles (cars, spacecraft, aircraft, satellites, submarines, …)
- Robotics
- Recommender systems
- AI for gaming
    - Usually as a “test bed” for more real-world ML
- Decision support (e.g., finance/trading)
- Artificial General Intelligence (AGI)
    - Maybe one day

Components of an RL system

- **Agent**: the RL actor (robot, car, etc)
- **States**: all possible configurations of the agent
- **Actions**: the possible actions of the agent
- **Reward**: feedback from the environment
- **Policy**: function returning new state given the current state and an action
- **Value**: function returning quantification of reward in the future given the current state and an action



Wikipedia

Types of environments

- Fully Observable (Chess) vs. Partially Observable (Poker)
- Single Agent (Atari) vs. Multi Agent (Self-Driving Cars)
- Deterministic (Chess) vs. Stochastic (Self-Driving Cars)
- Discrete (Chess) vs. Continuous (Robotic Navigation)

Self-driving cars

Agent:

States:

Actions:

Rewards:

Self-driving cars

Agent: the car

States: (Crash or No Crash, Lawful or Unlawful, distance from destination)

Actions: acceleration, deceleration, steering wheel angle change

Rewards: -10000 for Crash, +10 for No Crash, -100 for Unlawful, …

Targeted advertisement recommendation

Agent:

States:

Actions:

Rewards:

Targeted advertisement recommendation

Agent: the recommender system

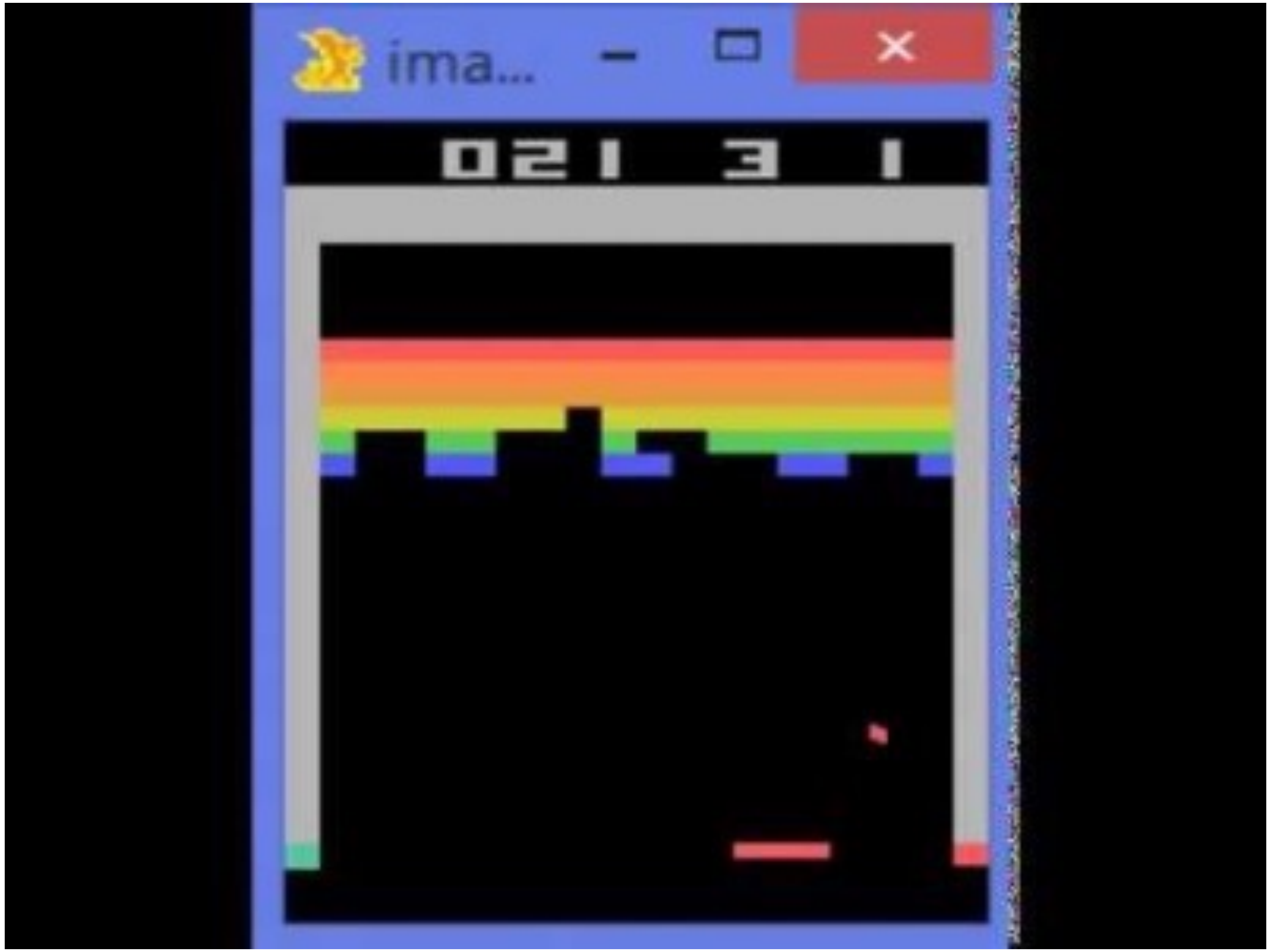
States:

Actions:

Rewards:

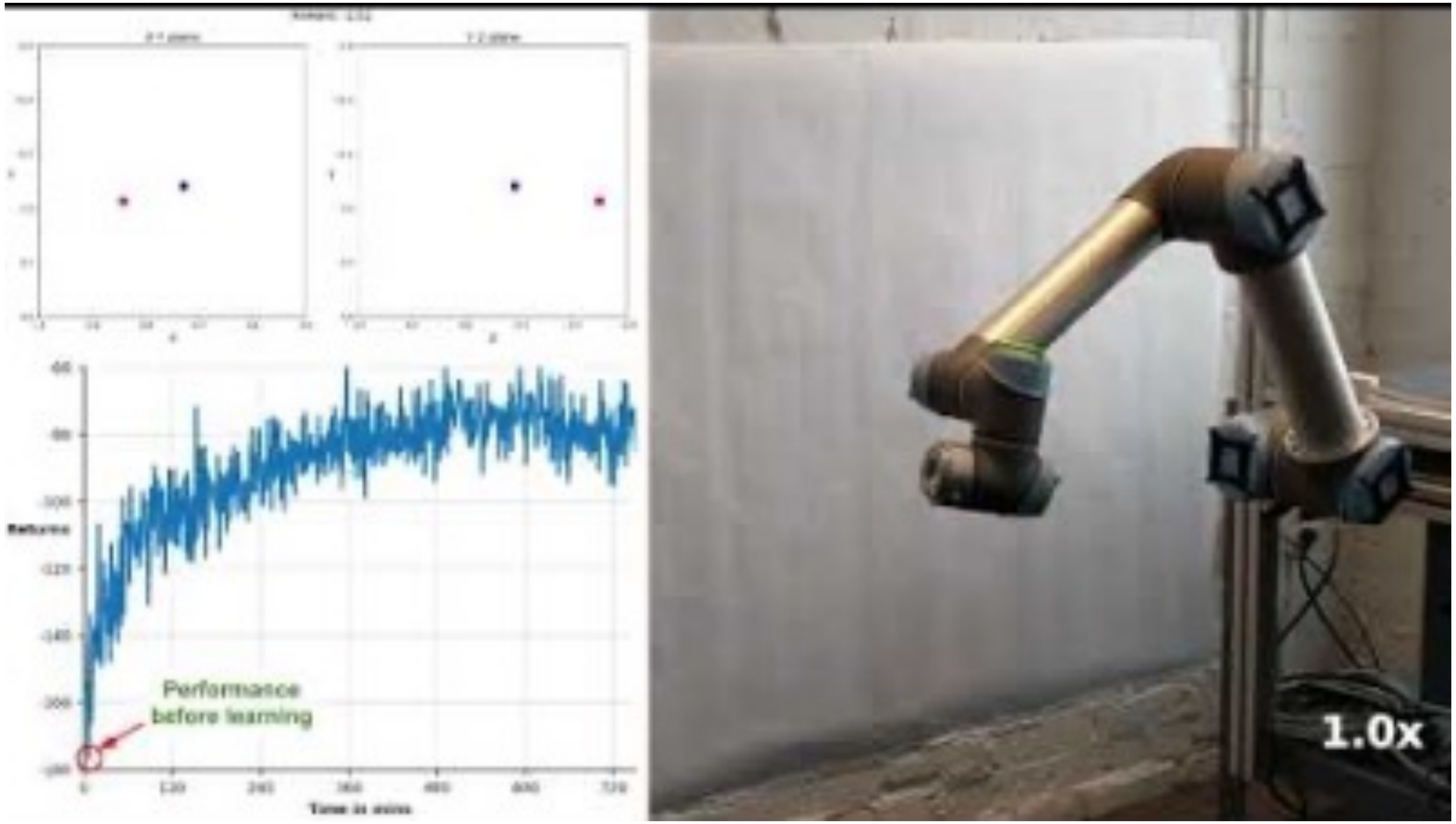
Examples of Reinforcement Learning

Simple Games

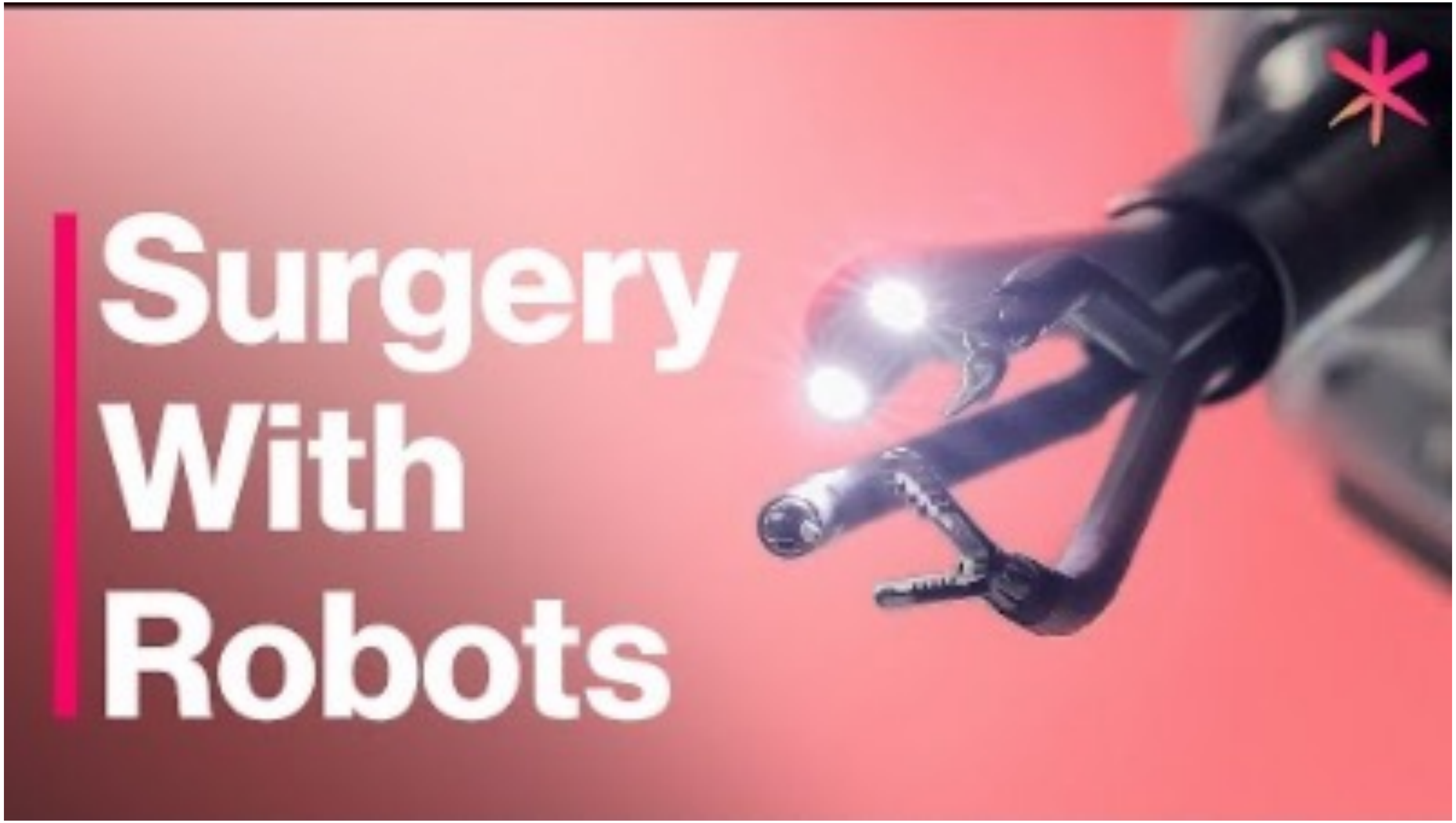


Complex Games

Robotics


Returns
0
120
240
360
480
600
720
Time in mins
Performance before learning
1.0x

Including but not limited to Surgical Robotics



Autonomous Vehicles

“Classical” Reinforcement Learning

Ultimate goal: maximize reward

Future Reward:

$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n$$

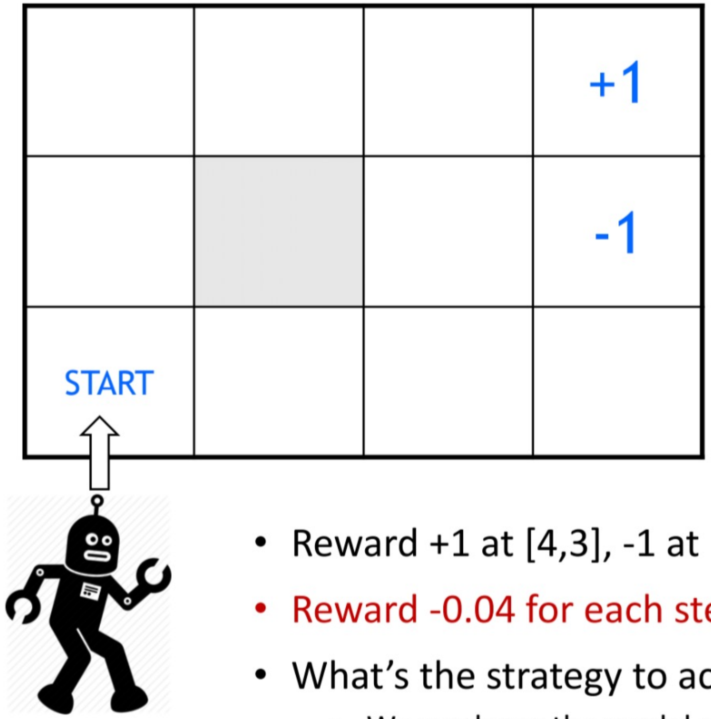
Discounted Future Reward:

$$R_t = r_t + \underset{\uparrow}{\gamma} r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n$$

Discount Factor: How much weight do we give to rewards in the future?
Higher γ → Incorporate more of the future reward

Lex Fridman, MIT 6.S091: Introduction to Deep Reinforcement Learning

Rewards are defined by humans: Demonstrative example

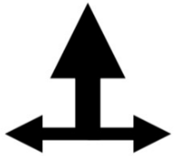


actions: UP, DOWN, LEFT, RIGHT

(Stochastic) model of the world:

Action: **UP**

80%	move UP
10%	move LEFT
10%	move RIGHT

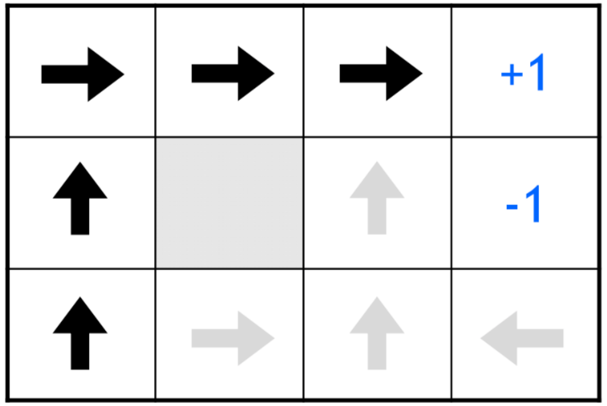


- Reward +1 at [4,3], -1 at [4,2]
- Reward -0.04 for each step
- What’s the strategy to achieve max reward?
  - We can learn the model and plan
  - We can learn the value of (action, state) pairs and act greed/non-greedy
  - We can learn the policy directly while sampling from it

Rewards are defined by humans: Demonstrative example

Optimal Policy for a Deterministic World

Reward: **-0.04** for each step



actions: UP, DOWN, LEFT, RIGHT

When actions are deterministic:

UP

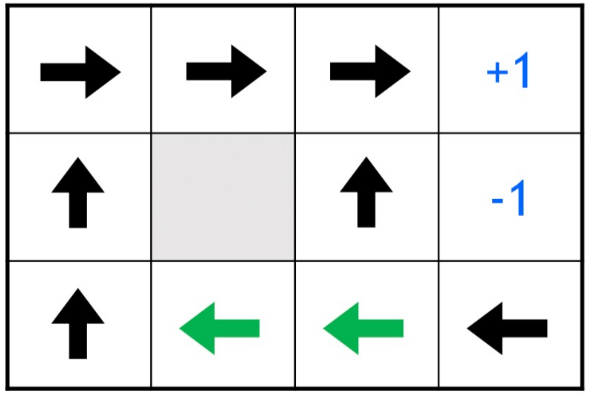
100%	move UP
0%	move LEFT
0%	move RIGHT

Policy: Shortest path.

Rewards are defined by humans: Demonstrative example

Optimal Policy for a Stochastic World

Reward: **-0.04** for each step



actions: UP, DOWN, LEFT, RIGHT

When actions are stochastic:

UP

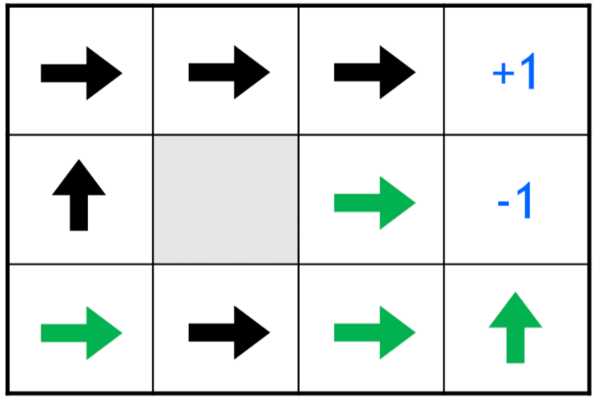
80%	move UP
10%	move LEFT
10%	move RIGHT

Policy: Shortest path. Avoid -UP around -1 square.

Rewards are defined by humans: Demonstrative example

Optimal Policy for a Stochastic World

Reward: **-2** for each step



actions: UP, DOWN, LEFT, RIGHT

When actions are stochastic:

UP

80%	move UP
10%	move LEFT
10%	move RIGHT

Policy: Shortest path.

Rewards are defined by humans: Demonstrative example

Optimal Policy for a Stochastic World

Reward: **-0.1** for each step

→	→	→	+1
↑	▨	↑	-1
↑	→	↑	←

More urgent

Reward: **-0.04** for each step

→	→	→	+1
↑	▨	↑	-1
↑	←	←	←

Less urgent

Rewards are defined by humans: Demonstrative example

Optimal Policy for a Stochastic World

Reward: **+0.01** for each step

↓	←	←	+1
↓		←	-1
←	←	←	↓

actions: UP, DOWN, LEFT, RIGHT

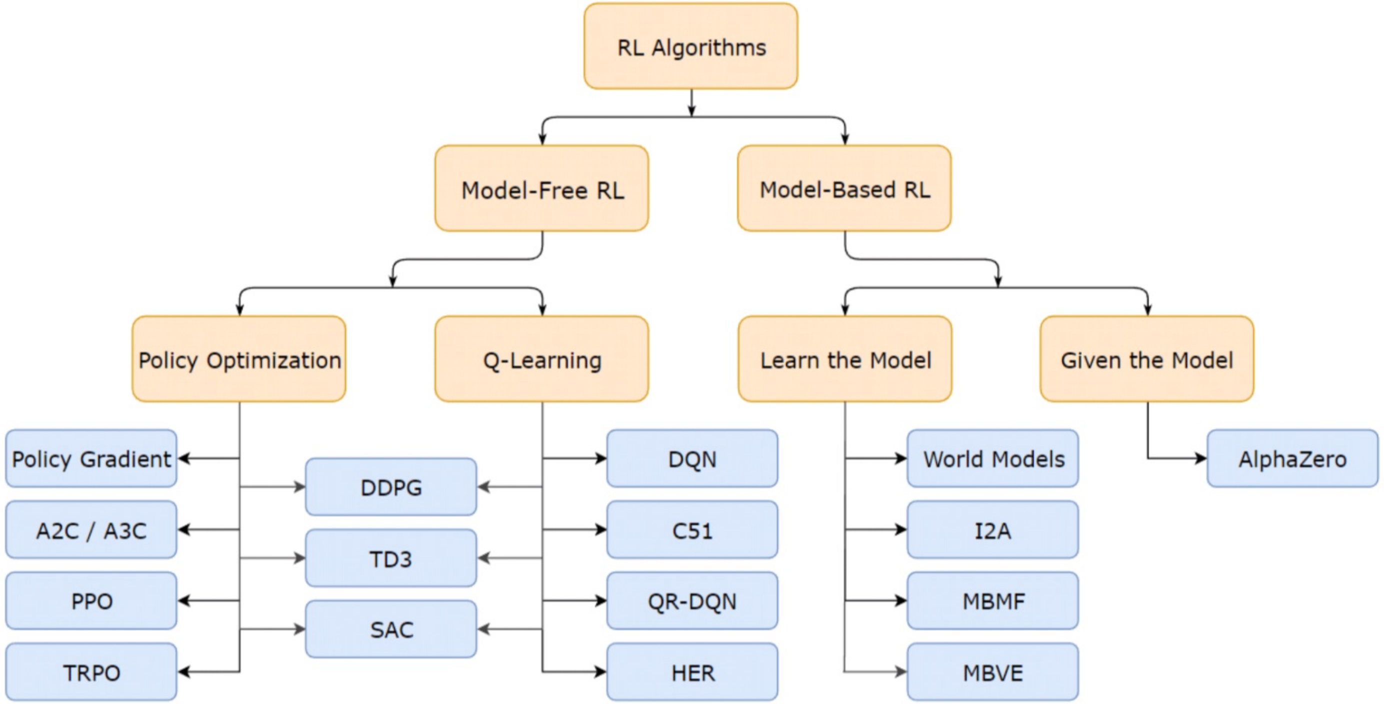
When actions are stochastic:

UP

80%	move UP
10%	move LEFT
10%	move RIGHT

Policy: Longest path.

Many types of RL methods



https://spinningup.openai.com/

Markov Decision Process (MDP)

(S, A, R, P)

S: states

A: actions

R: reward function, R(s, s’, a)

P: transition function, P(s, s’, a) = $P(s_{t+1}=s' \mid s_t = s, a_t = a)$

Policies and Values

Policy function (usually denoted as π): $\pi(a|s)$ is the probability of taking action *a* when currently in state *s*

Value function: $V_{\pi}(s)$ is the expected total return when starting in state *s* and following policy π thereafter

Bellman Equation: Dynamic Programming

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s') \right)$$

Bellman Equation : Dynamic Programming

$$\underbrace{V(s)}_{\text{Expected return (value) of the current state } s} = \overbrace{\max_{a}}^{\text{Maximum value of any possible action } a} \left(\underbrace{R(s,a)}_{\text{Reward of taking action } a \text{ at state } s} + \overbrace{\gamma}^{\text{Discount Factor}} \underbrace{\sum_{s'} P(s,a,s')}_{\text{Weight each of the possible next state } s' \text{ by the probability of ending up at that state}} \overbrace{V(s')}^{\text{Value of the next state } s'} \right)$$

Brute Force Solution

- For each possible policy, sample returns while following it
- Choose the policy with the largest expected return

Brute Force Solution

- For each possible policy, sample returns while following it
- Choose the policy with the largest expected return

Issues:

- The number of policies can be large or infinite
- Variance of returns can be large, requiring large number of samples to estimate the return of any given policy

Model-based vs Model-free RL

- Model-based: explicitly learn the transition function P and the reward function R
- Model-free: learn a policy directly without necessarily understanding the world via P and R

Model-based RL: Value Iteration

Iteratively update the state-value function.

Repeat until convergence:

$$V_{i+1}(s) := \max_{a} \left\{ \sum_{s'} P_a(s, s') \left(R_a(s, s') + \gamma V_i(s') \right) \right\}$$

Act by choosing the best action in a state

Model-based RL: Value Iteration

Iteratively update the state-value function.

Repeat until convergence:

$$V_{i+1}(s) := \max_a \left\{ \sum_{s'} P_a(s, s') \left(R_a(s, s') + \gamma V_i(s') \right) \right\}$$

Keep exploring the environment while updating the Value function as you go

Act by choosing the best action in a state

Model-based RL: Policy Iteration

Start with an arbitrary policy **π**, then iteratively update the policy.

Repeat until convergence (two-step process):

$$V(s) := \sum_{s'} P_{\pi(s)}(s, s') \left(R_{\pi(s)}(s, s') + \gamma V(s')\right)$$

$$\pi(s) := \mathrm{argmax}_a \left\{ \sum_{s'} P_a(s, s') \left(R_a(s, s') + \gamma V(s')\right) \right\}$$

Act by sampling the policy

Model-based RL: Policy Iteration

Start with an arbitrary policy **π**, then iteratively update the policy.

Repeat until convergence (two-step process):

$$V(s) := \sum_{s'} P_{\pi(s)}(s,s')\left(R_{\pi(s)}(s,s') + \gamma V(s')\right)$$

Keep Policy fixed and update Value function until it converges

$$\pi(s) := \operatorname{argmax}_a \left\{ \sum_{s'} P_a(s,s')\left(R_a(s,s') + \gamma V(s')\right) \right\}$$

Find the best actions at each state using one step lookahead

Act by sampling the policy

Model-free RL: Q-Learning

- Q(s, a) is the expected return starting at state s, taking action a, then thereafter following policy π

- Best possible future return when performing action a in state s

- Q function is the Action-Value function for policy π

- In general: model-free RL involves predicting the value function of a certain policy **without having a concrete model of the environment**

Q-learning

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

Compute Q function for all state-action pairs

After Q function is learned, act by selecting a for current state which has the highest Q value

Q-learning

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)}^{\text{temporal difference}}$$

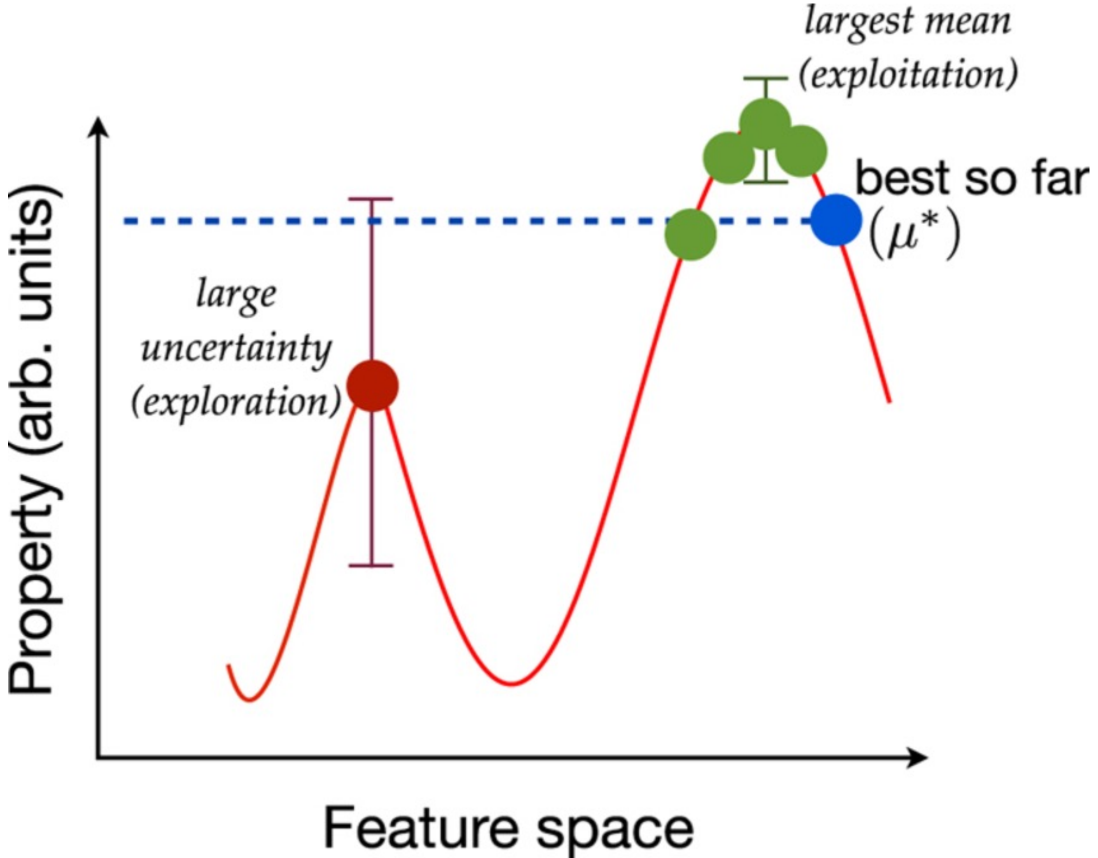
We will see the specifics of how to implement Q-learning in our final class coding notebook

We will see the specifics of how to implement Q-learning in our final class coding notebook

(But it is literally just implementing the Q-learning equation and choosing the best action accordingly)

Fundamental tradeoff (not specific to Reinforcement Learning): exploration vs. exploitation

- Exploration: sample the global search space; helps avoid local optima
- Exploitation: maximize with a promising local region of the search space to fully optimize the solution



Epsilon (ε) Greedy Approach

- With probability ε, select a random action
- With probability 1-ε, choose the best action
- ε is a hyperparameter which defines the Exploration-Exploitation tradeoff

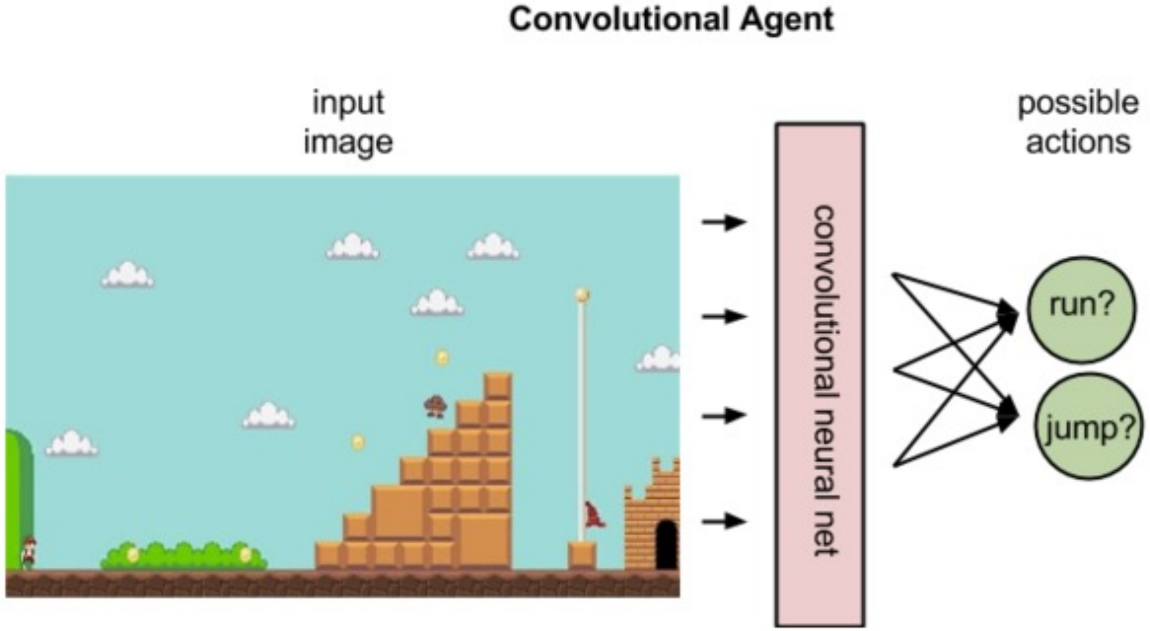
Modern Reinforcement Learning: Deep Reinforcement Learning

Deep RL

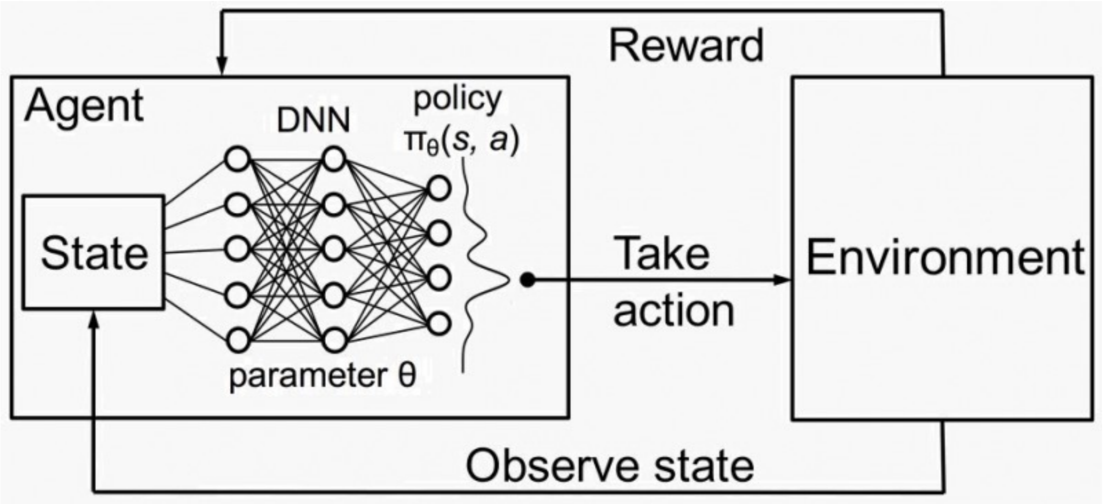
Replace parts of the RL process with DL

Deep RL

Representation matters!

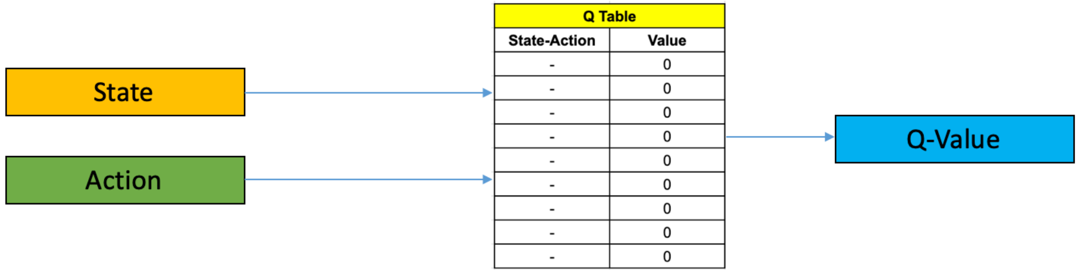


https://wiki.pathmind.com/deep-reinforcement-learning



https://medium.com/@vishnuvijayanpv/deep-reinforcement-learning-value-functions-dqn-actor-critic-method-backpropagation-through-83a277d8c38d

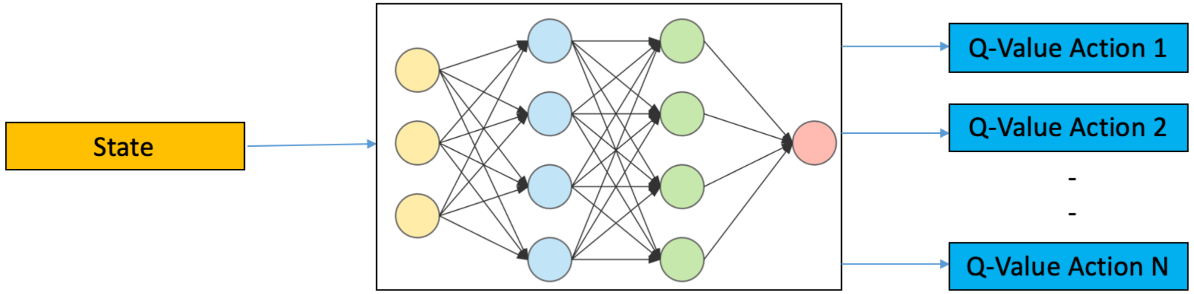
Deep Q-learning



Q Table	
State-Action	Value
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

Q Learning

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \bigg)}^{\text{temporal difference}}$$



Deep Q Learning

https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

Loss Function for a Q Network: Mean Square Error

$$\overbrace{\left(\underbrace{\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}}}_{\text{“y\_true”}} - \underbrace{\underbrace{Q(s_t, a_t)}_{\text{old value}}}_{\text{“y\_pred”}}\right)}^{\text{temporal difference}}{}^{2}$$

Some cool examples of multiple interacting neural networks for reinforcement learning

“World Models”: Predict how the environment works and run simulations with that model

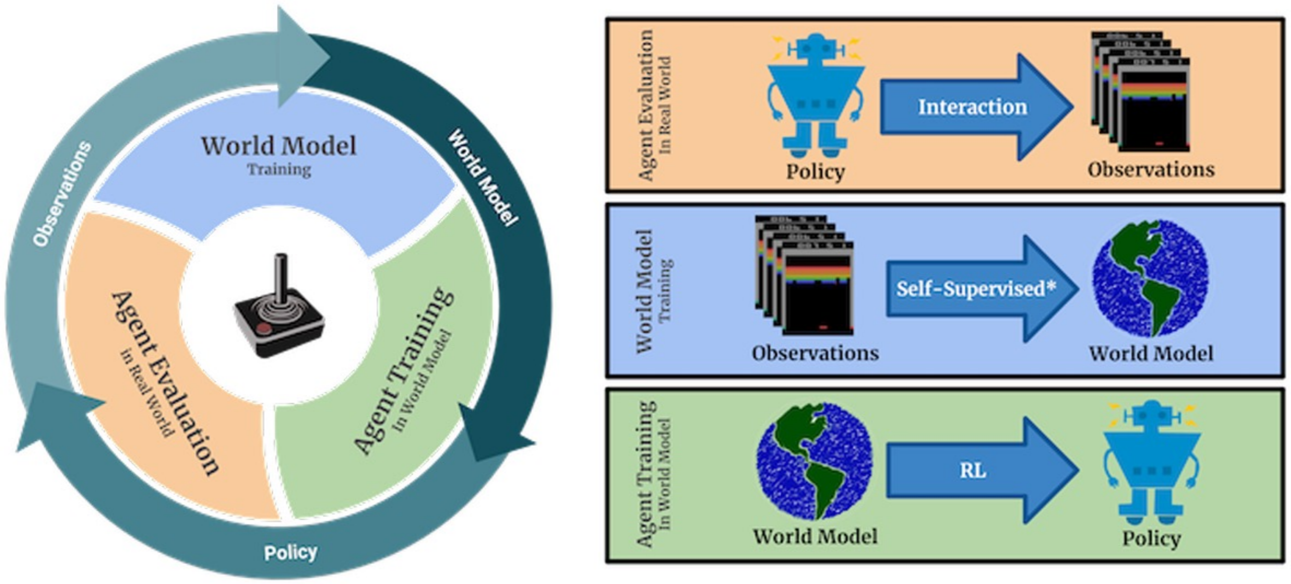
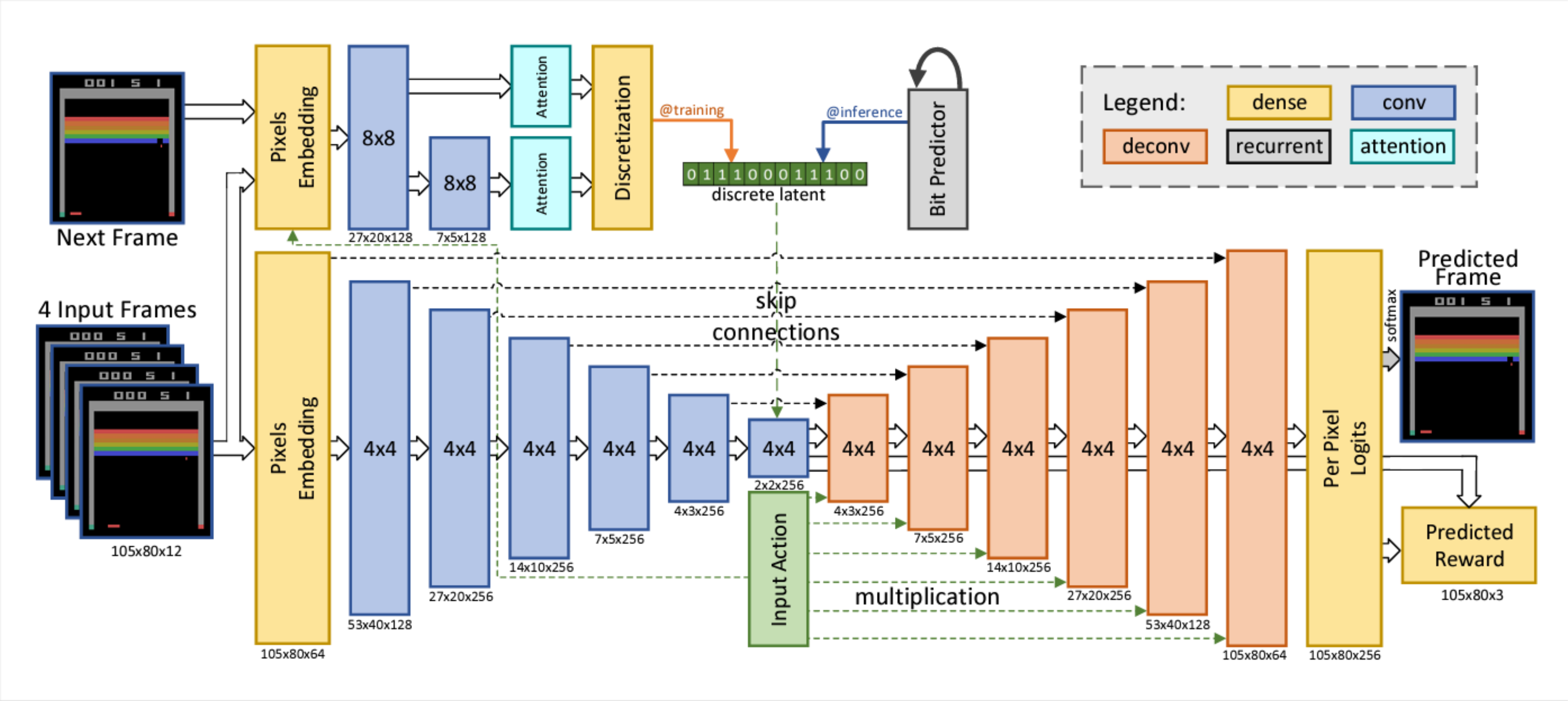


Figure 1: Main loop of SimPLe. 1) the agent starts interacting with the real environment following the latest policy (initialized to random). 2) the collected observations will be used to train (update) the current world model. 3) the agent updates the policy by acting inside the world model. The new policy will be evaluated to measure the performance of the agent as well as collecting more data (back to 1). Note that world model training is self-supervised for the observed states and supervised for the reward.

Kaiser et al. “Model-based reinforcement learning for Atari.” ICLR 2020.

“World Models” : Predict how the environment works and run simulations with that model



Kaiser et al. “Model-based reinforcement learning for Atari.” ICLR 2020.

“Self Models”: agent can take actions that result in the highest loss

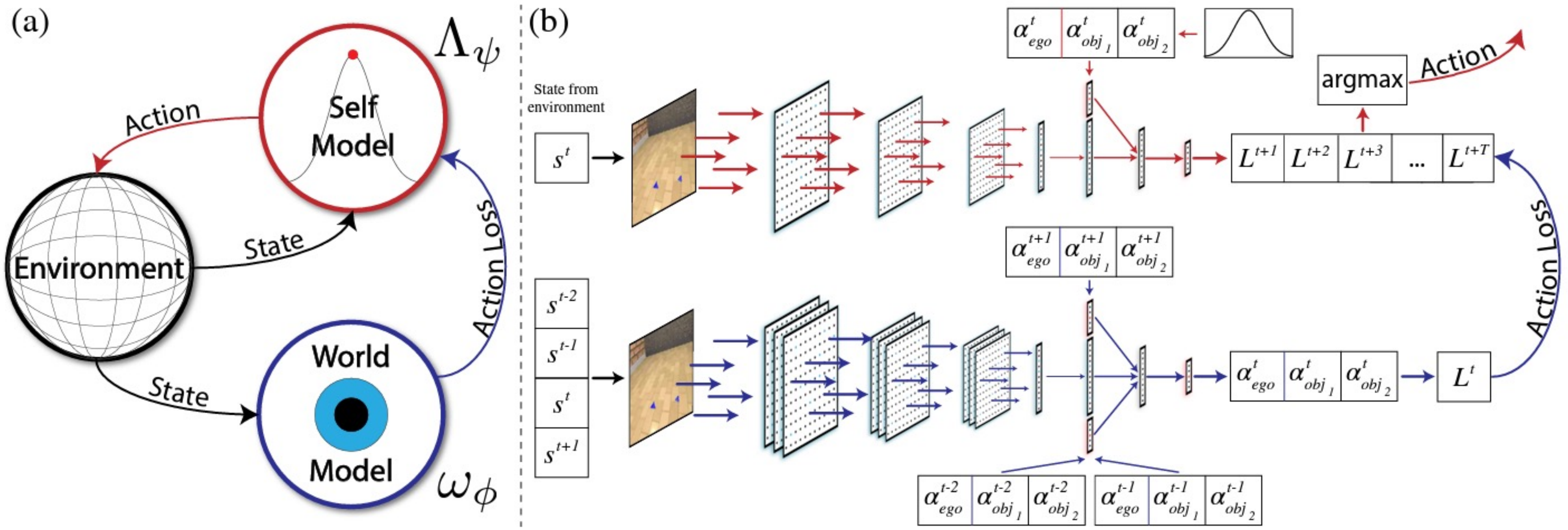
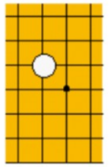

(a)
Action
Self Model
Λ_ψ
State
Environment
Action Loss
State
World Model
ω_ϕ
(b)
State from environment
s^t
α^t_{ego} $\alpha^t_{obj_1}$ $\alpha^t_{obj_2}$
argmax
Action
L^{t+1} L^{t+2} L^{t+3} ... L^{t+T}
α^{t+1}_{ego} $\alpha^{t+1}_{obj_1}$ $\alpha^{t+1}_{obj_2}$
s^{t-2}
s^{t-1}
s^t
s^{t+1}
α^t_{ego} $\alpha^t_{obj_1}$ $\alpha^t_{obj_2}$
L^t
Action Loss
α^{t-2}_{ego} $\alpha^{t-2}_{obj_1}$ $\alpha^{t-2}_{obj_2}$
α^{t-1}_{ego} $\alpha^{t-1}_{obj_1}$ $\alpha^{t-1}_{obj_2}$

Figure 2: **Intrinsically-motivated self-aware agent architecture.** The world-model (blue) solves a dynamics prediction problem. Simultaneously a self-model (red) seeks to predict the world-model’s loss. Actions are chosen to antagonize the world-model, leading to novel and surprising events in the environment (black). (a) Environment-agent loop. (b) Agent information flow.

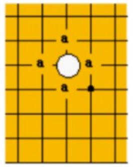
Haber et al. “Learning to Play With Intrinsically-Motivated, Self-Aware Agents.” NeurIPS 2018.

Applications of Deep RL

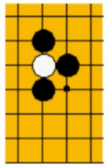
Go game



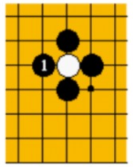
Dia. 3



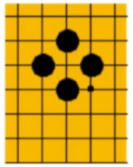
Dia. 4
liberties



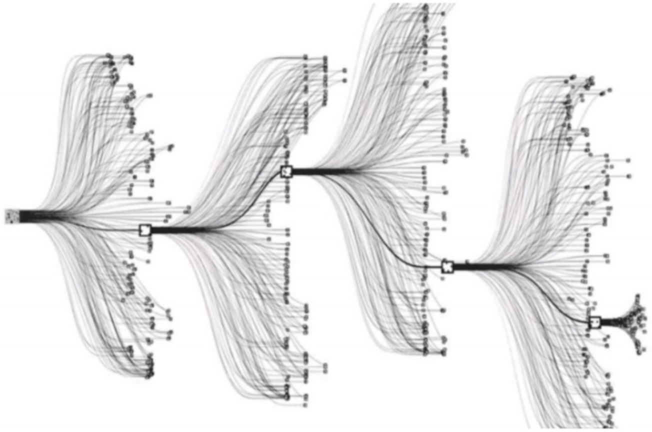
Dia. 5
atari

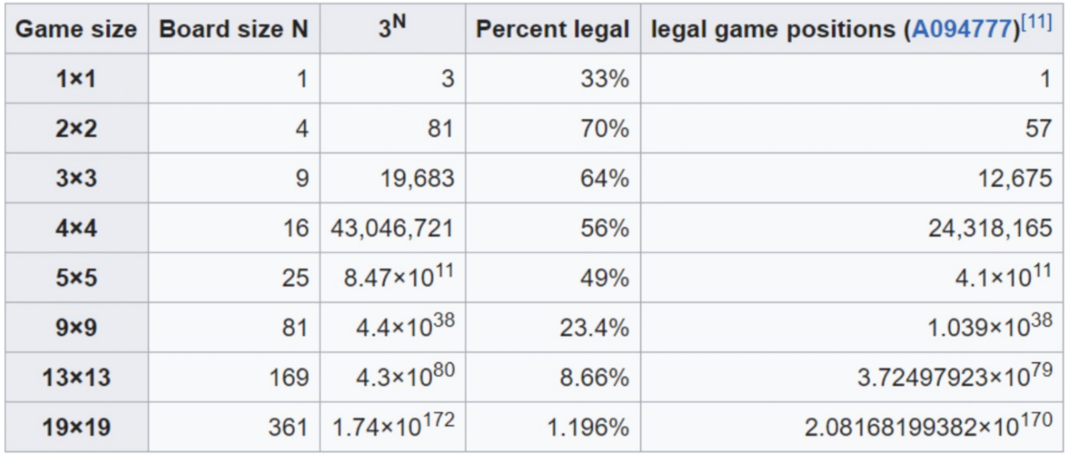


Dia. 6
capture



Dia. 7
result





Game size	Board size N	3^N	Percent legal	legal game positions (A094777)[11]
1×1	1	3	33%	1
2×2	4	81	70%	57
3×3	9	19,683	64%	12,675
4×4	16	43,046,721	56%	24,318,165
5×5	25	8.47×10^{11}	49%	4.1×10^{11}
9×9	81	4.4×10^{38}	23.4%	1.039×10^{38}
13×13	169	4.3×10^{80}	8.66%	3.72497923×10^{79}
19×19	361	1.74×10^{172}	1.196%	$2.08168199382\times10^{170}$

Wikipedia

AlphaGO

Netflix

AlphaGO

AlphaGO

- Originally trained by observing historical games by Go experts
- Later, was trained to play against itself using separate instances of itself

- Wikipedia: “As of 2016, AlphaGo's algorithm uses a combination of machine learning and tree search techniques, combined with extensive training, both from human and computer play. It uses Monte Carlo tree search, guided by a "**value network**" and a "**policy network**," both implemented using **deep neural network** technology. A limited amount of game-specific feature detection pre-processing (for example, to highlight whether a move matches a nakade pattern) is applied to the input before it is sent to the neural networks. The networks are **convolutional neural networks** with 12 layers, trained by **reinforcement learning**.”

AlphaStar

Alpha Fold

- Can accurately predict 3D models of protein structures
- Has the potential to accelerate research in every field of biology

Implementing RL in Python