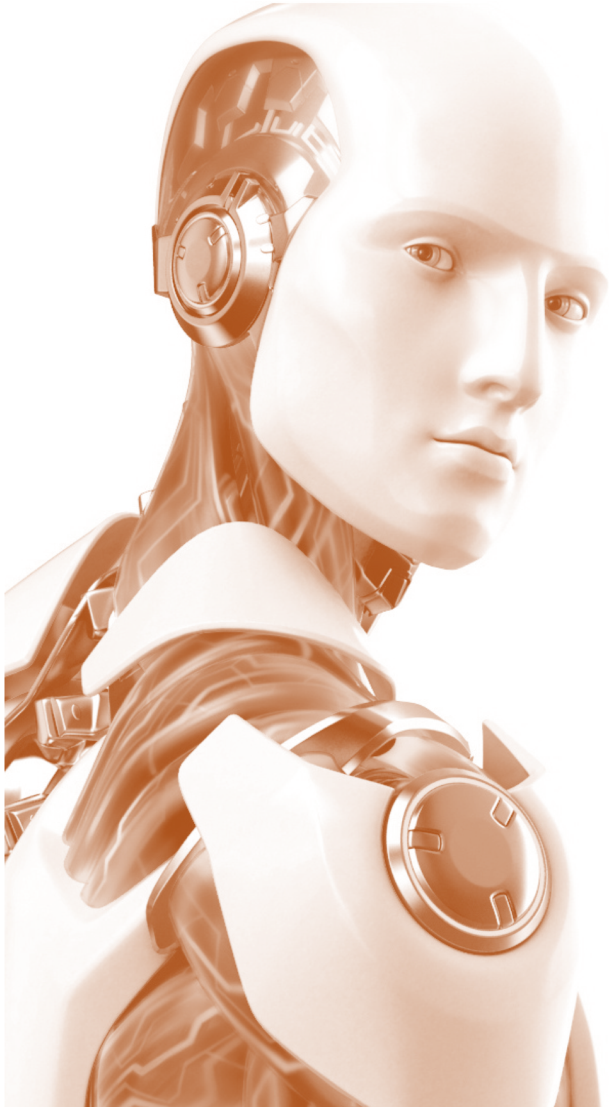


Introduction to AI

Lecture 16 - Machine Learning - Reinforcement Learning

Profa. Dra. Esther Luna Colombini
esther@ic.unicamp.br

Prof. Dr. Alexandre Simoes
alexandre.simoes@unesp.br

LaRoCS – Laboratory of Robotics and Cognitive Systems

LAROCS

UNICAMP

Advanced Institute for Artificial Intelligence

Summary

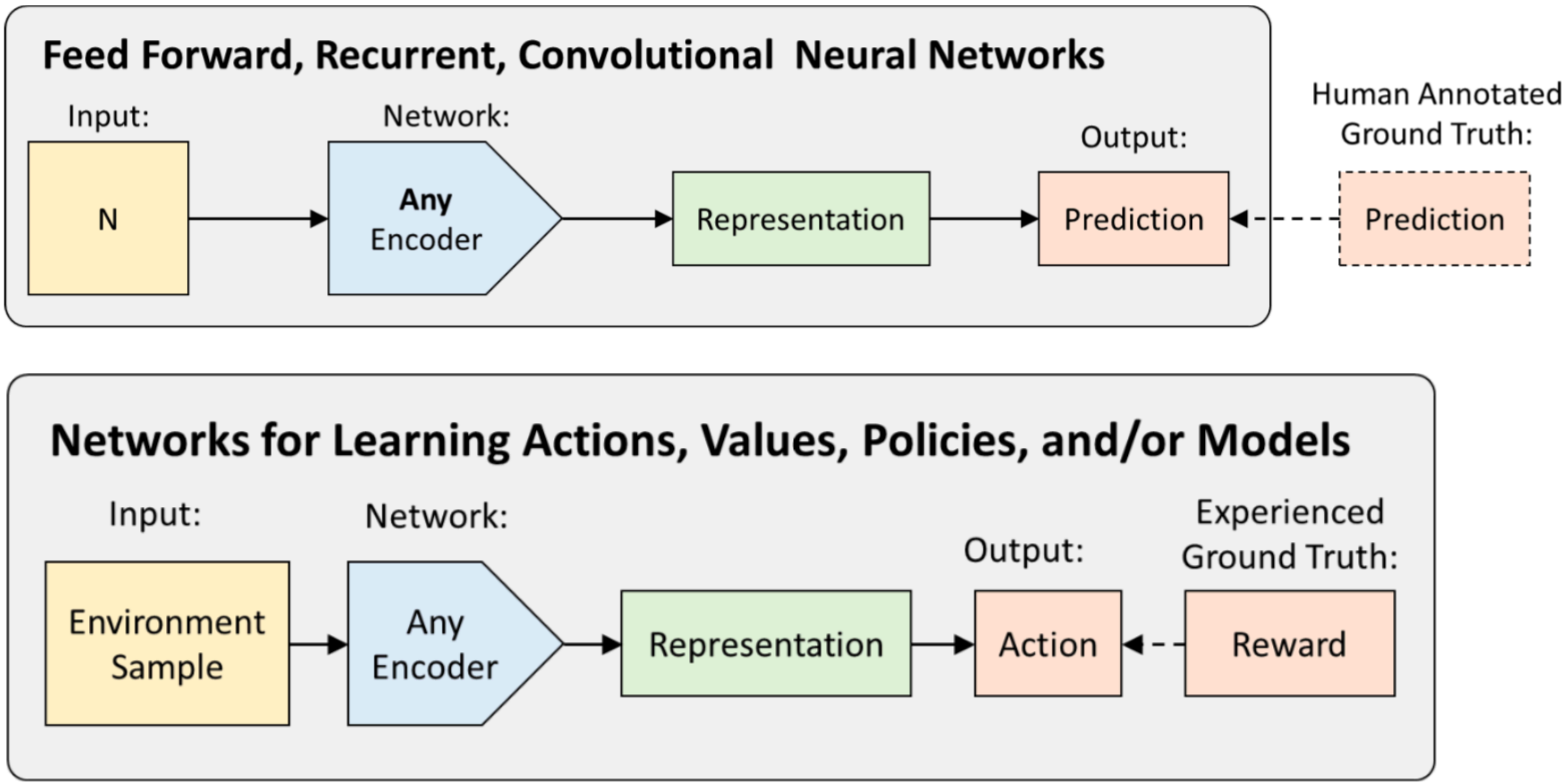
- Markov Decision Process
- Reinforcement Learning
- Q-learning
- Examples
- Exercise
- Deep RL

Types of Learning

- Supervised Learning
- Semi-Supervised Learning
- Unsupervised Learning
- Reinforcement Learning
  - It's all "supervised" by a loss function!

- Supervised learning is "teach by **example**":
  - Here's some examples, now learn patterns in these example.
- Reinforcement learning is "teach by **experience**":
  - Here's a world, now learn patterns by exploring it.

Types of Learning



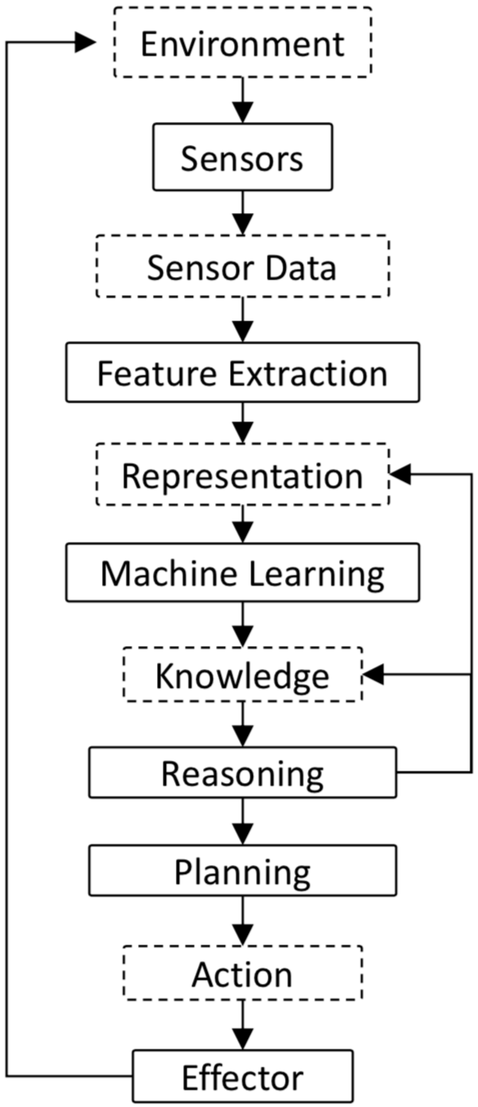
https://deeplearning.mit.edu

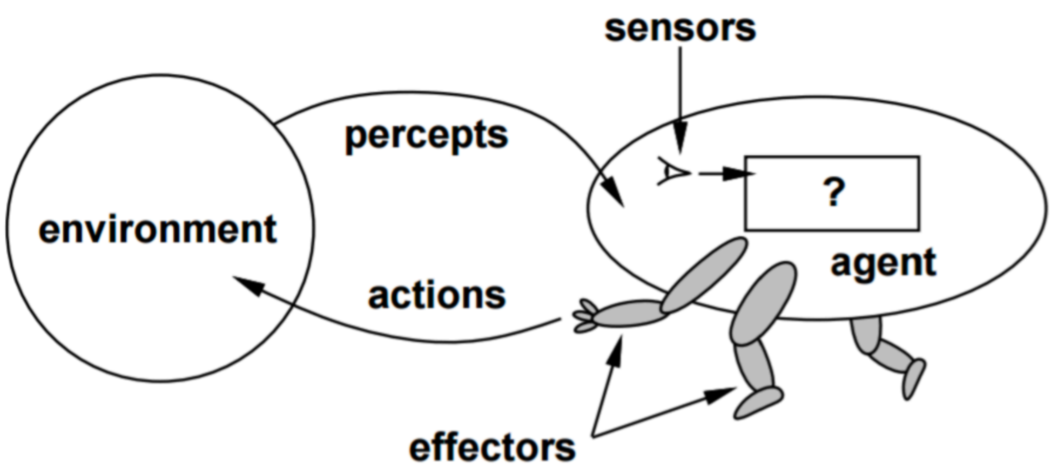
Reinforcement Learning: For what?

- Often the use of supervised learning is impractical
  - How to get correct training examples for a given situation? What if the environment is unknown?
  - Examples:
    - Child acquiring motor coordination
    - Robot interacting with an environment to achieve objective(s)
- Human appear to learn to walk through "very few examples" of trial and error. **How** is an open question...
  - Possible answers
    - **Hardware:** 230 million years of bipedal movement data.
    - **Imitation Learning:** Observation of other humans walking
    - **Algorithms:** Better than backpropagation and stochastic gradient descent

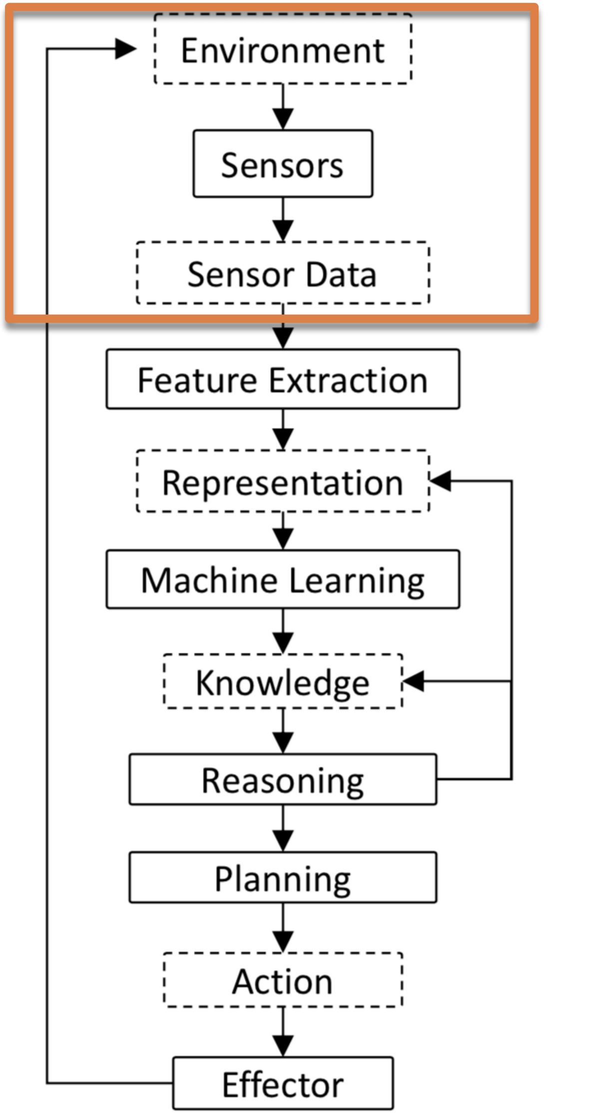
Reinforcement Learning: For what?

Reinforcement Learning: For what?

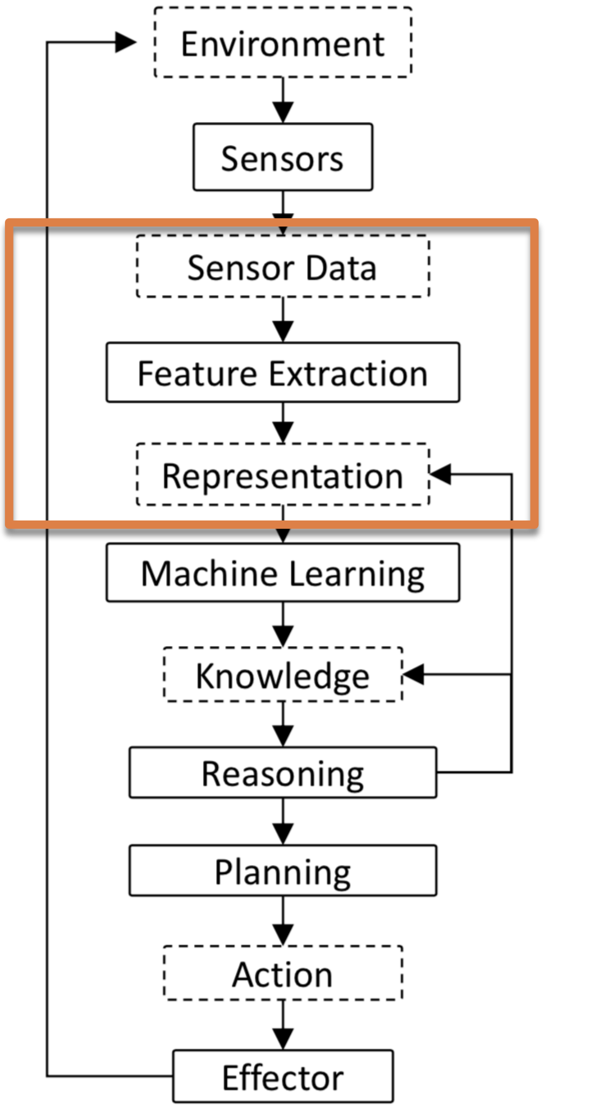


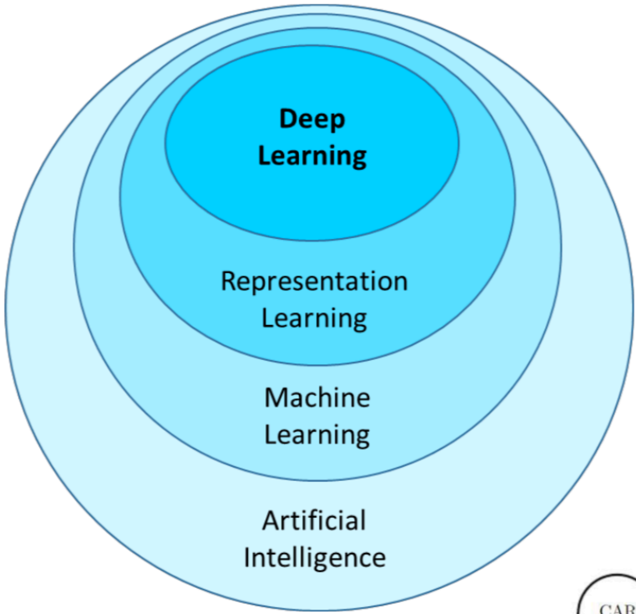


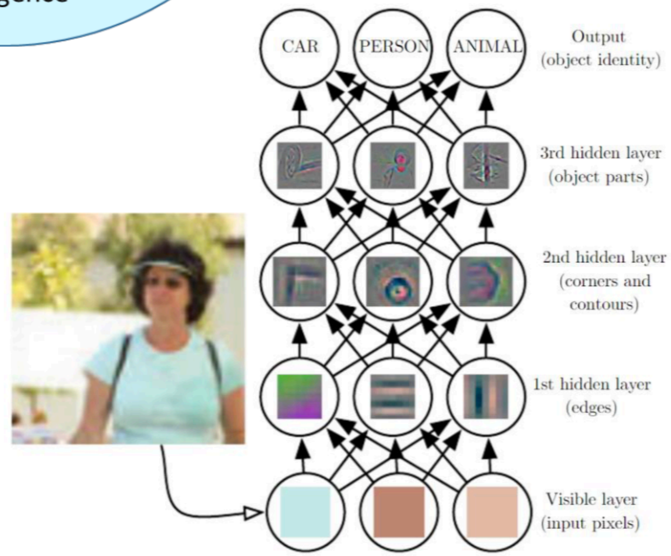
Reinforcement Learning: For what?



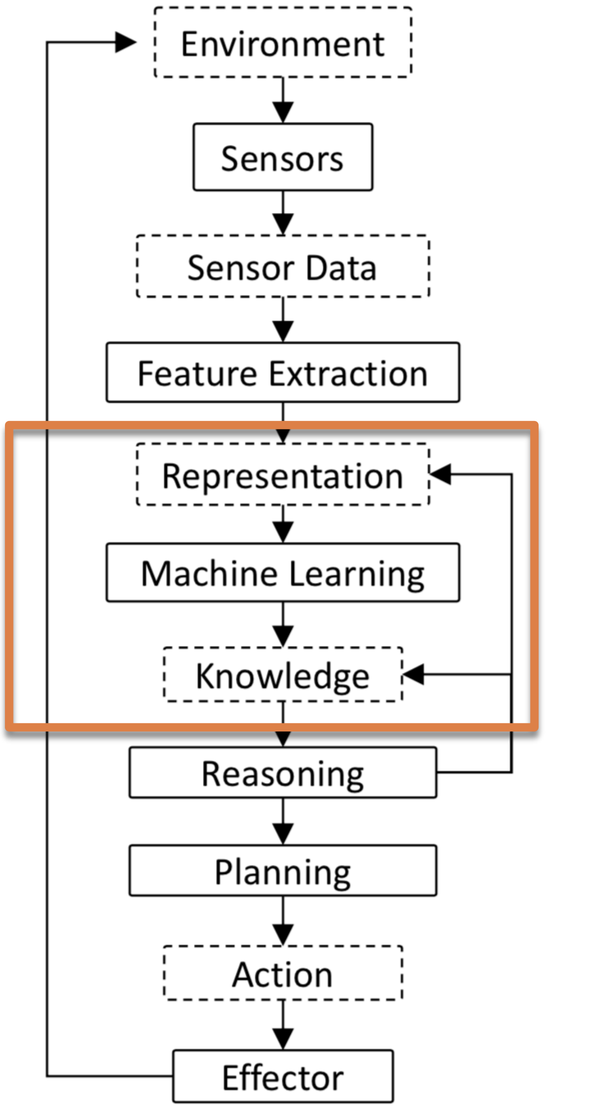
Reinforcement Learning: For what?

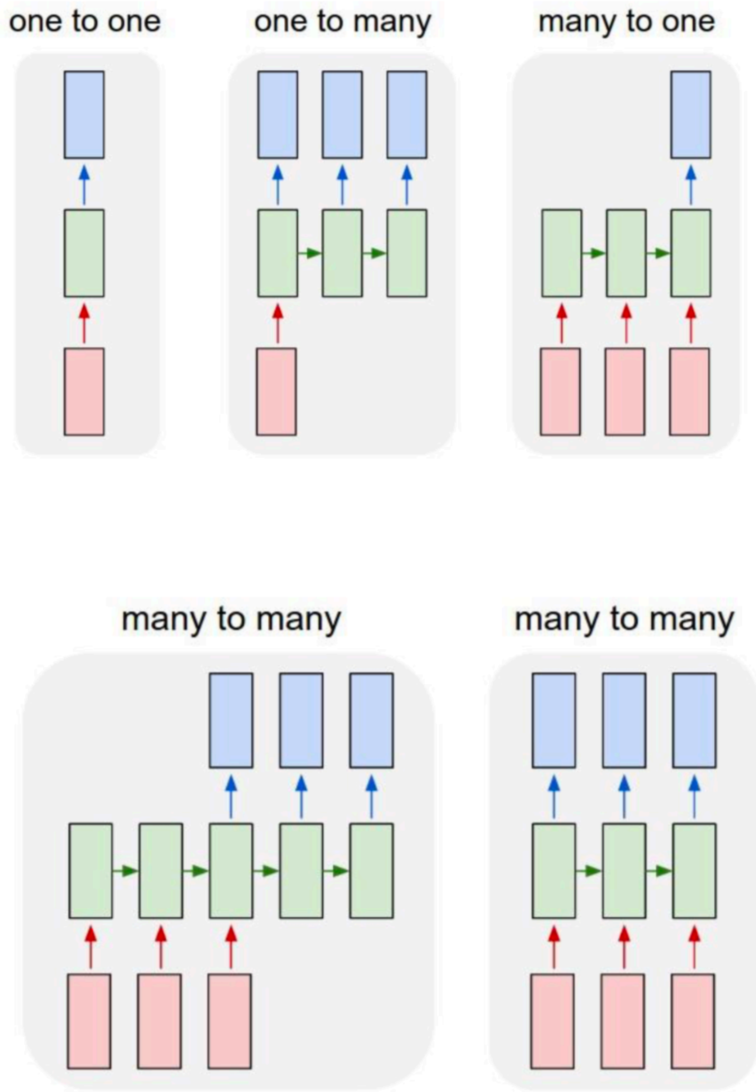






Reinforcement Learning: For what?





Reinforcement Learning: For what?

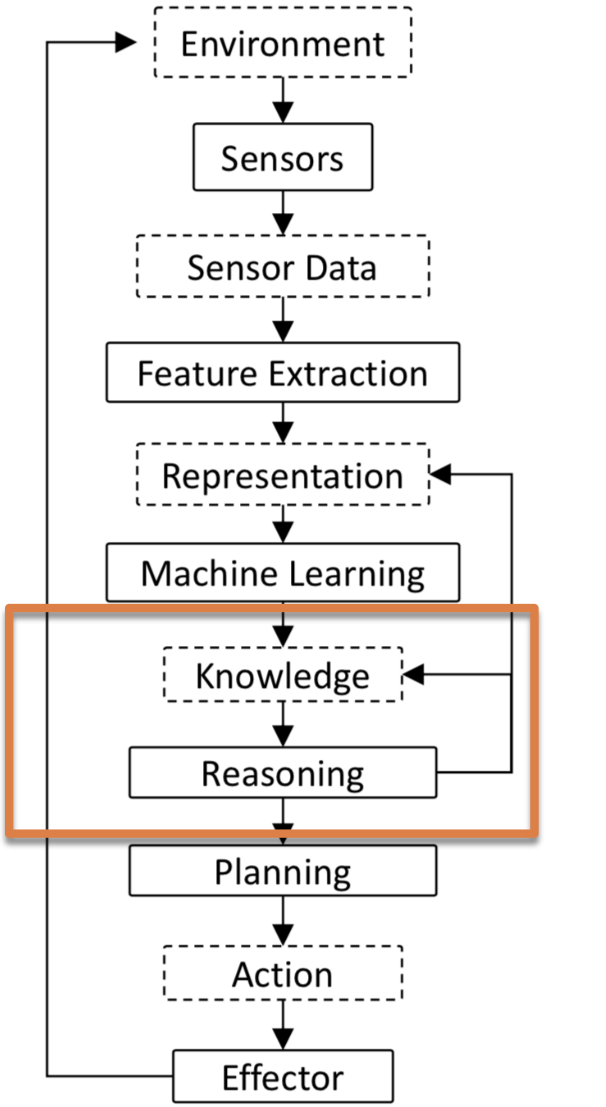
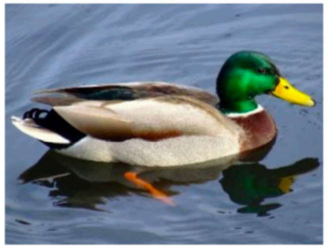
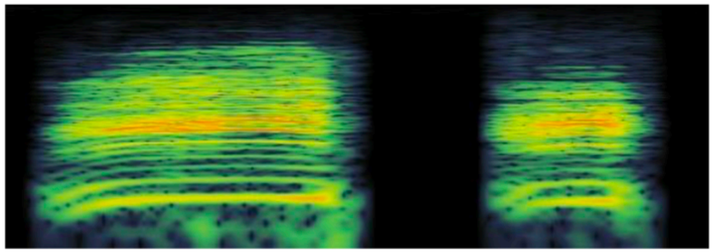


Image Recognition:
If it looks like a duck

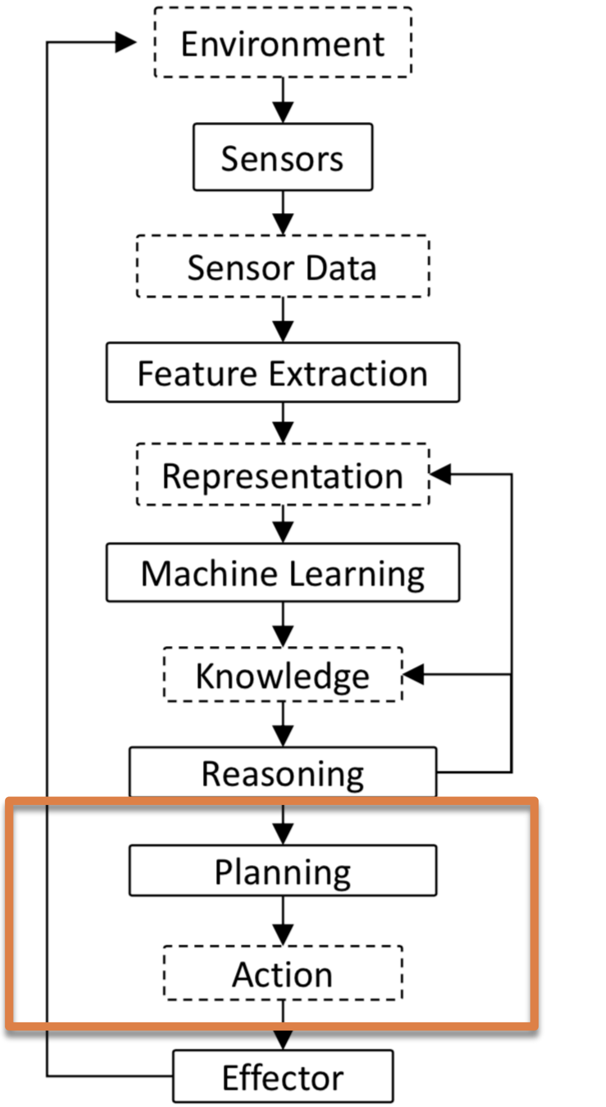


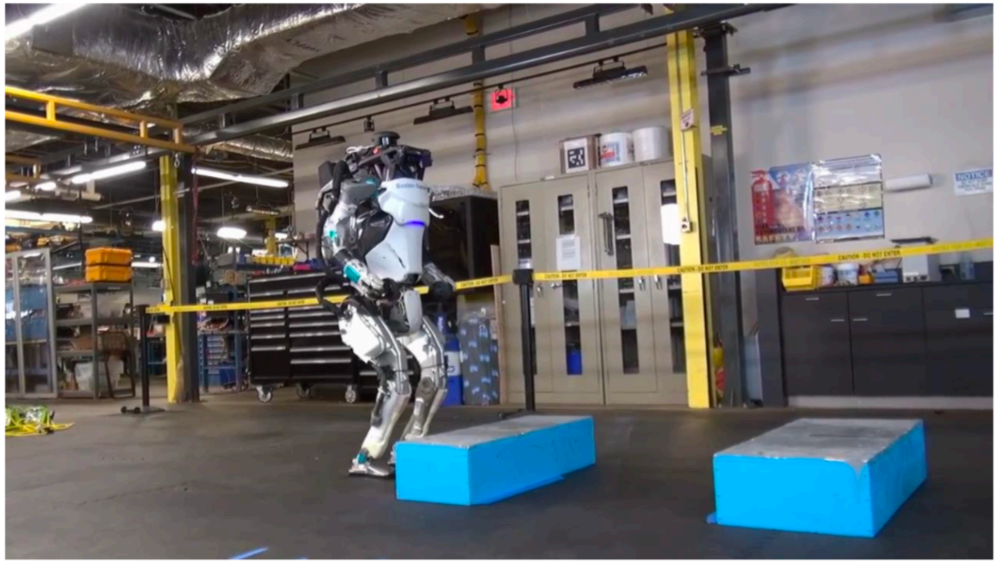
Audio Recognition:
Quacks like a duck

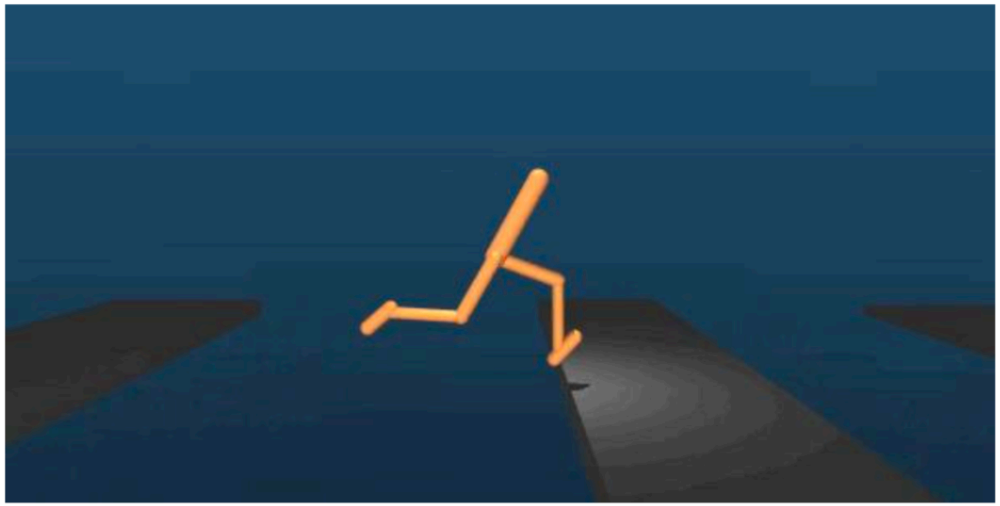


Activity Recognition:
Swims like a duck

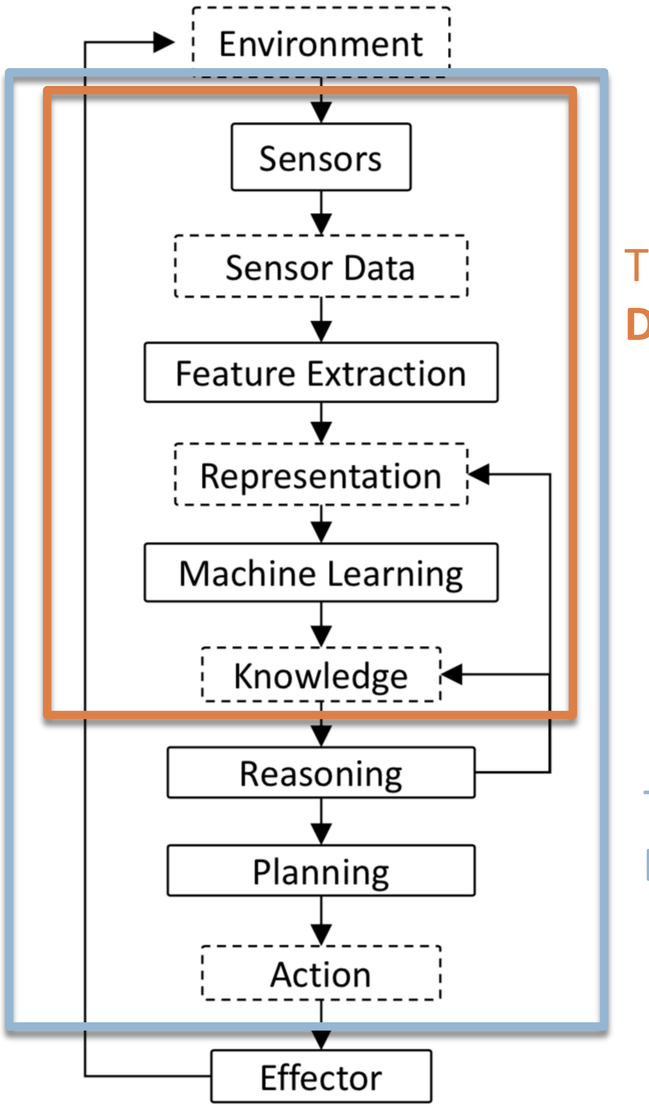
Reinforcement Learning: For what?







Reinforcement Learning: For what?



The promise of
Deep Learning

The promise of
Deep Reinforcement Learning

What is Reinforcement Learning?

- Premise:
  - At each **time** instant t, the agent is in a **state** s
  - In **state** s it performs **action** a and goes to **state** s'
  - The **state** is evaluated and gives a **reward** to the agent
  - Thus, the **action** a in state s has a **value** for the agent
  - If you choose correct
    - wins a reward (gains value)
  - if not
    - receives a punishment (loses value)
- Reinforcement learning:
  - Choose an action policy that maximizes the total rewards received by the agent

Environment and Actions

- Fully Observable (Chess) vs Partially Observable (Poker)
- Single Agent (Atari) vs Multi Agent (DeepTraffic)
- Deterministic (Cart Pole) vs Stochastic (DeepTraffic)
- Static (Chess) vs Dynamic (DeepTraffic)
- Discrete (Chess) vs Continuous (Cart Pole)

  - Note: Real-world environment might not technically be stochastic or partially-observable but might as well be treated as such due to their complexity.

Reinforcement Learning Assumptions

- Experiential learning
  - The world is the best model of itself
- RL is characterized by problems involving the concept of Autonomy
  - RL links AI concepts and Optimal Control
  - RL is applicable to real problems

Reinforcement Learning Assumptions

- Specify what to do, not how to do
  - This is done through the reward function
- Usually find the best end solutions
  - Based on current experiences, there are no assumptions of the programmer
- In short:
  - Less human time is needed to find a good solution
  - It is not necessary to define heuristics, techniques to solve the problem, etc.
  - Just set the learning system and let the system learn!

Applications

- Backgammon Game (10^{20} states)
  - Game Modeling:
    - Victory: +100
    - Defeat: - 100
  - Zero for other states of the game (delayed reward)
    - DELAYED REWARD -> leaves to reward at the end of a process
    - After 1 million matches against himself, he plays as well as the best human player
- Robots Football
    - Robot Soccer Brainstormers (RoboCup)
    - Team whose knowledge is obtained 100% by learning techniques by reinforcement
  - Computer Go (10^{170} states)

Markov Decision Processes - MDPs

- Formally, an MDP is given by:
  - A set of states, S = {s_1, s_2, …, s_n}
  - A set of actions, A = {a_1, a_2, …, a_m}
  - A Reward function, R: S:A:S → r

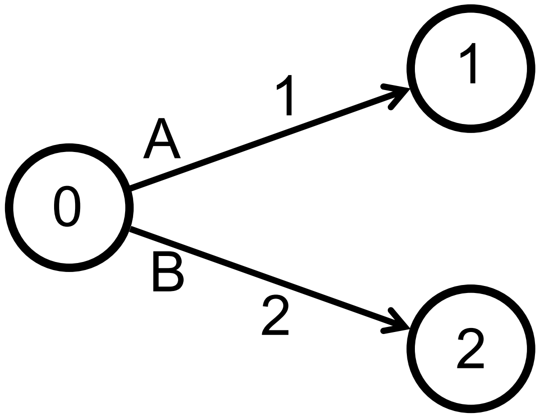
$$s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{n-1}, a_{n-1}, r_n, s_n$$

state
action
reward
Terminal state

- A state transition function, T: S,A → S
- We want to learn the policy p: S → A, that is, given states in S we have the best actions in A to be applied.
  - **Policy**: sequence of states and actions
- Markov property
  - Everything you need to make a decision is included in the status
  - There is no way to consult the past (previous states)

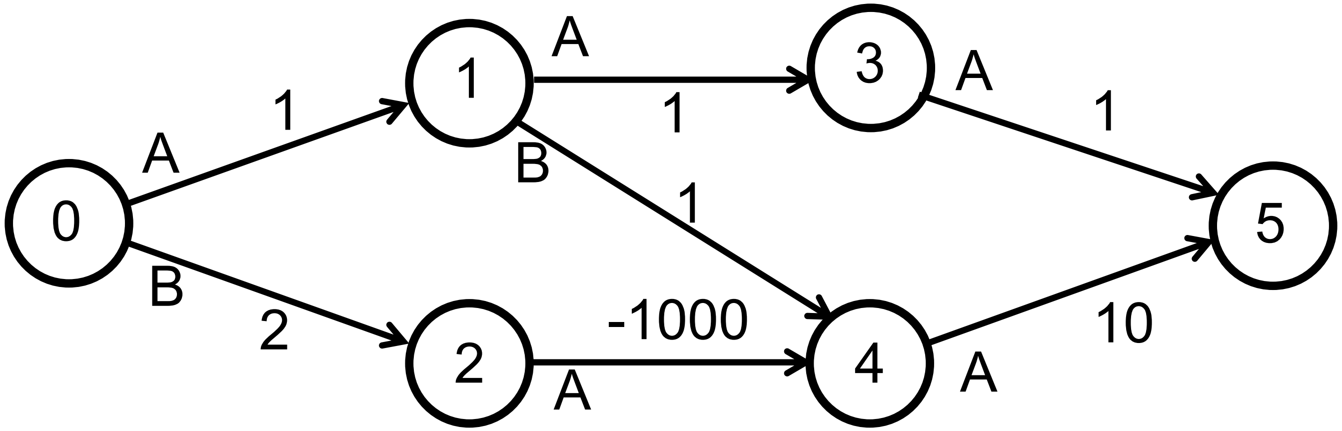
Making decisions

- With the reward set, what we need is to make a decision in each state:
  - Multiple actions (A and B)
  - Each action has a reward associated with it



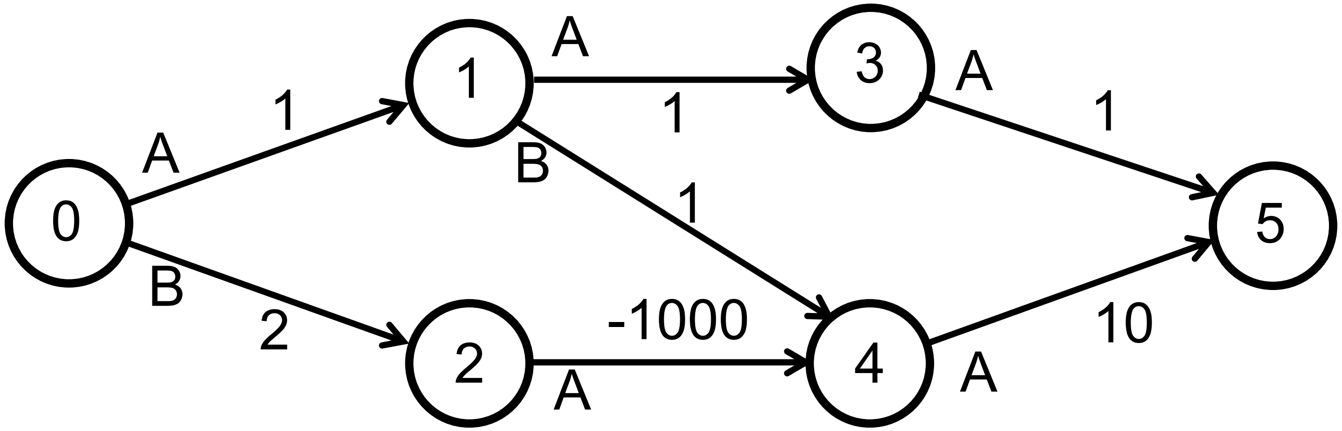
- The goal is to maximize the reward
- Just take the action with the highest reward for the current state

- We can generalize the previous example to multi-sequential decisions
  - Each decision affects the next decision
- This is formally modeled as Markov Decisive Process (PDM or MDP)

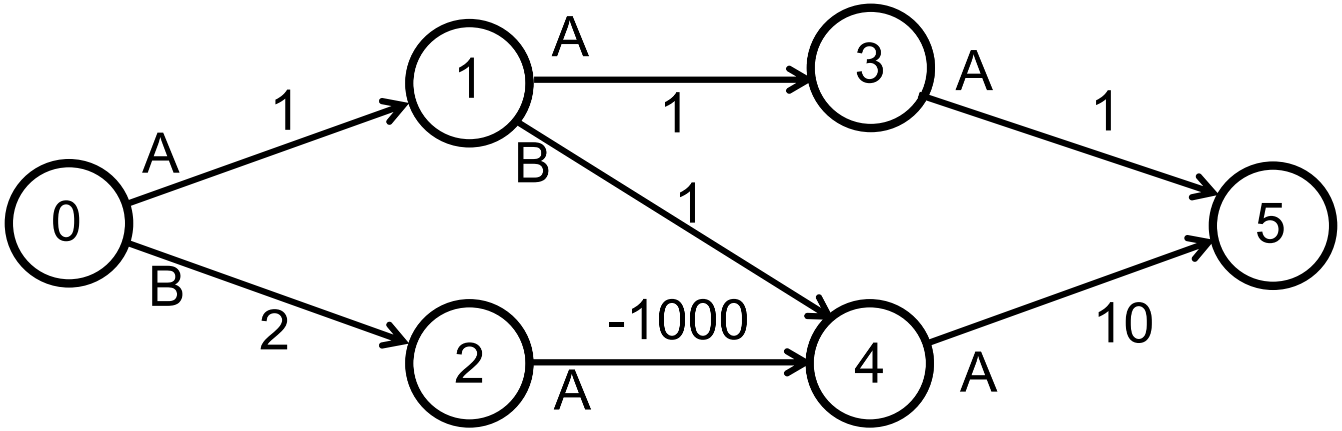


Policies

- There are 3 policies for the MDP below:
  - $0 \rightarrow 1 \rightarrow 3 \rightarrow 5$
  - $0 \rightarrow 1 \rightarrow 4 \rightarrow 5$
  - $0 \rightarrow 2 \rightarrow 4 \rightarrow 5$
- Which is better?

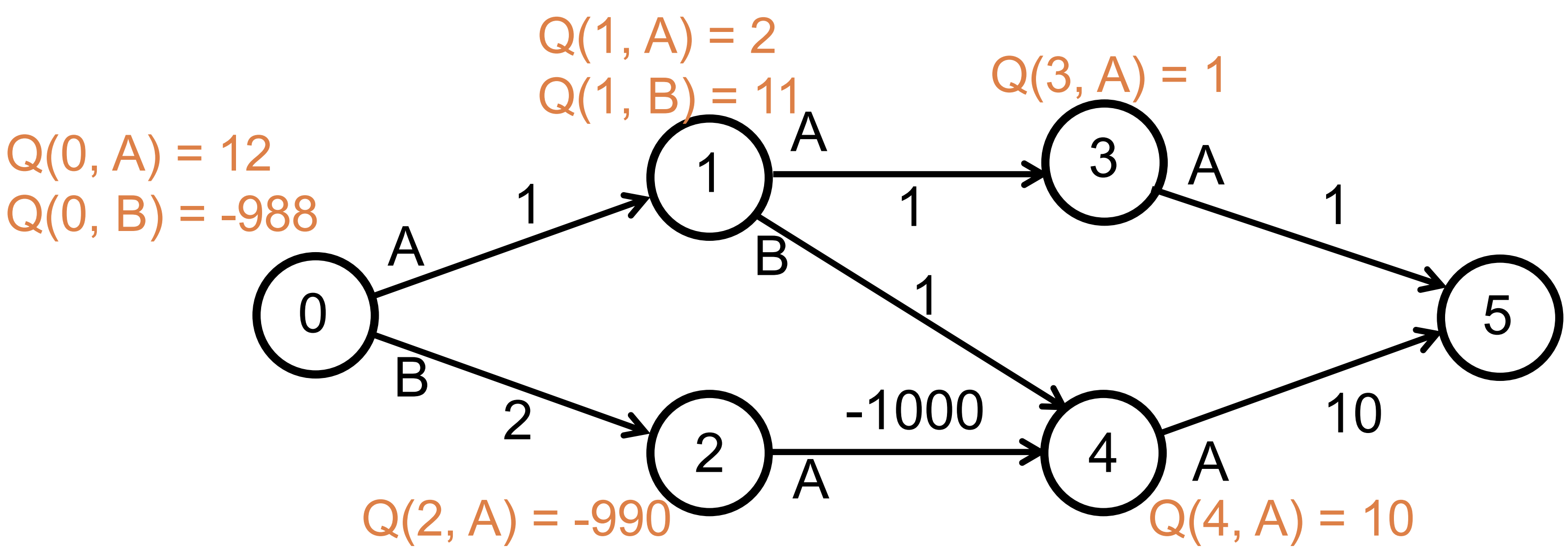


- Sort policies by rewards
  - $0 \to 1 \to 3 \to 5 = 1 + 1 + 1 = 3$
  - $0 \to 1 \to 4 \to 5 = 1 + 1 + 10 = 12$
  - $0 \to 2 \to 4 \to 5 = 2 - 1000 + 10 = -988$



State-Action Value

- We can set a value without specifying the policy
  - Specify the value of choosing action **a** from state **s**
  - This is the state-of-action quality function, **Q**



- $Q(s, a) = R(s, a, s') + \max_{a'} Q(s', a')$

s' is the next state

- Form:
  - Next reward + the best I can do from the next state, even if the policy is not followed
- If we have the value function, then finding the best policy is easy:
  - $p(s) = \text{argmax } Q(s, a)$
- argmax f(x) means the argument that makes f(x) maximum

Optimal Policy

- We are looking for the optimal policy: p*(s)
- This means that no policy generates a reward greater than p*

- Optimal policy defines optimal value functions:

$$Q^*(s,a) = R(s,a,s') + \max_{a'} Q^*(s',a')$$

- The easiest way to learn an optimal policy is to learn the optimal value function first!

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

Action-Value Functions

- We can introduce a term into the function to prevent high values from saturating the system and getting it into looping
- Called the temporary discount factor, γ

$$0 \leq \gamma \leq 1$$

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

- Interpretation:
  - Determines the importance of future rewards
    - If 0, the agent is considered "short-sighted" because it only considers current elements
    - If $\gamma >= 1$ and no final state, it will tend to infinity
    - To aim to reduce the influence of future expected reinforcements

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

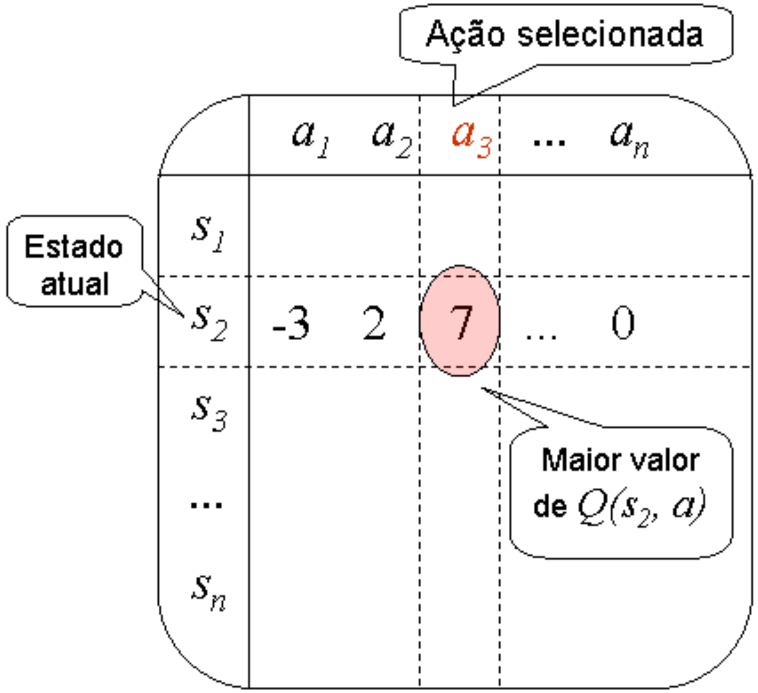
- If we have the model, an iterative algorithm by value or by policy can find the best policy
  - Dynamic Programming (applicable to problems in which the optimal solution can be computed from the optimum solution previously calculated and stored)
  - **Value Iteration**: uses dynamic programming to determine the value $V^*(s)$, or argmax, of each state $s \in S$ of the MDP, at each decision time.
  - In the case of infinite horizon (and stationary policy - with numerous properties that are unchanged in time) one can also use an algorithm called **policy iteration**, which makes a greedy search in the policy space. This algorithm is more efficient than value iteration.

- What happens if we do not have full MDP?
  - That is, we need to learn the associated rewards
  - Well .. We know about states and actions
  - We do not know about the system model (transition function) or the reward function
  - We can learn from experience and carry out actions to generate such experiences.
- This is the main goal of reinforcement learning...

Learning the value function

- We still want to learn the value function
  - We are forced to approach it interactively
  - Based on the experiences of the world
- Let's talk about one of the main algorithms:
  - Q-learning
  - Off-policy x On-policy
    - An off-policy learns the optimal policy value regardless of the agent's actions, as well as Q-learning
    - An on-policy learns the value of the policy being carried out by the agent, including the exploitation steps (SARSA)
    - This distinction disappears if the policy followed is greedy

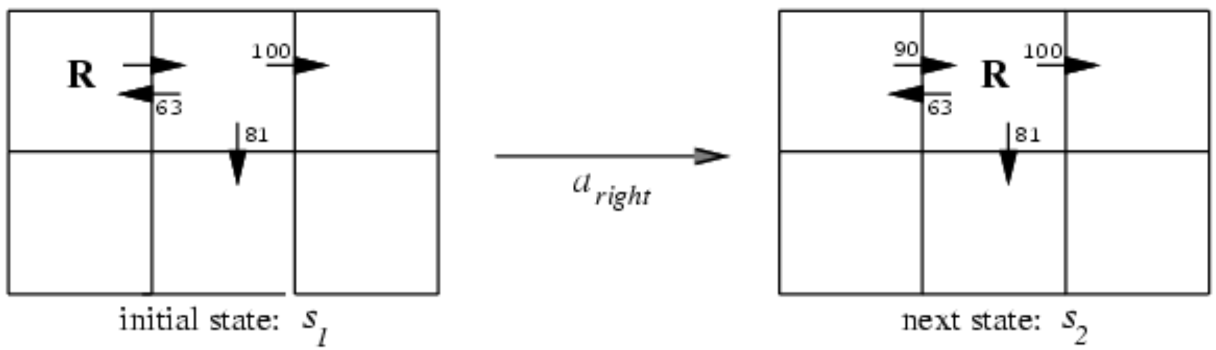
- Learning algorithm to compute optimal Q function (value of actions)
- $Q^*(s) = \text{argmax}_a[Q(s,a)]$



This table is usually huge and takes up a lot of memory!

- $Q^*(s_t,a_t) = r(s_t,a_t) + \gamma \max_{a'} [Q(s_{t+1},a')]$

- Q(s_t) is updated after observing the s_{t+1} state and the reward received
  - Consider that we are in s_1 and that we want to perform the aright action. How to update Q (s_1, a_{right})? Use reward 0 and γ=0.9



- $Q(s_1,a_{right}) = r + \gamma \max_{a'} Q(s2,a')$
  $= 0 + 0.9 \max\{63,81,100\}$
  $= 90$

- Q-learning approximation, iteratively, the state-action value function, Q
  - We will not estimate MDP directly
  - Learn the function value and policy simultaneously
- Maintains the estimate of Q (s, a) in a table
  - Updates these estimates as you add more experience
  - The estimate does not depend on the operating policy

Q-Learning Algorithm

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

Reward

Old value

Estimate of the best future value

Old value

error

- The change in Q value to perform action a in state s is the difference between the real reward ($r + \max_{a'}Q(s', a')$) and the expected reward ($Q(s, a)$)
- We can think of this as a type of PD control or the error of the output of an NN that takes the system in the direction of the correct Q.

Q-Learning Algorithm

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

Temporal discount factor

- To aim to reduce the influence of future expected reinforcements. As see before:
  - If 0, the agent is considered "short-sighted" because it only considers current elements
  - If $\gamma >= 1$ and no final state, it will tend to infinity

Q-Learning Algorithm

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

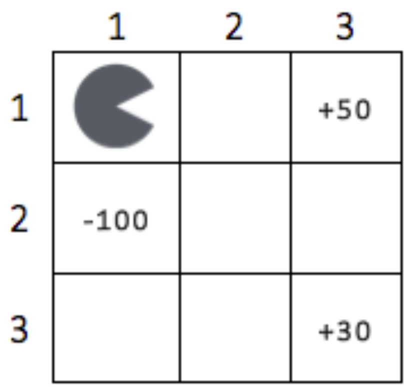
Learning rate

- $0 \leq \alpha \leq 1$ is the rate of learning
- α indicates how much new information is relevant
  - A value of 0 will prevent the agent from learning
  - A value of 1 will make it learn only with the latest information
  - If the problem is deterministic, the factor must be 1
  - If the problem is stochastic, the value must be less than 1

Q-Learning Algorithm

- Initialize Q (s, a) for small random values, $\forall$s, a
- Observe state s
- Choose an action, a, and execute
- Observe next state, s ', and reward of s', r
- $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$
- Go back to 2

- Consider the grid-world below and an agent who is trying to learn the optimal policy.



- The possible actions are:
  - D (right), E (left), N (north) and S (south).

Q-Learning Algorithm

- The Q table has been initialized with the following values:

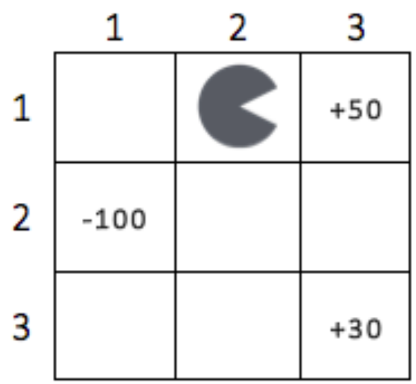
	D	E	N	S
1,1	0,4	0,2	0,2	0,3
1,2	0,4	0,2	0,1	0,2
1,3	0,2	0,1	0,2	0,3
2,1	0,2	0,1	0,5	0,1
2,2	0,3	0,4	0,1	0,2
2,3	0,2	0,5	0,1	0,2
3,1	0,4	0,2	0,4	0,2
3,2	0,1	0,2	0,3	0,2
3,3	0,2	0,2	0,3	0,1

- Reinforcements (positive and negative) will be given only in the indicated regions. Assume $\gamma = 1$ and $\alpha = 0.5$ for all calculations. Consider the agent in the initial position indicated.
- Perform 5 greedy actions in sequence, performing the required updates on Table Q.
- Consider: $Q(s_t, a_t) = (1 - \alpha)\, Q(s_t, a_t) + \alpha\, (r_t + \gamma \max Q(s_{t+1}, a'))$

Q-Learning Algorithm

- Action 1:

	D	E	N	S
1,1	0,4	0,2	0,2	0,3
1,2	0,4	0,2	0,1	0,2
1,3	0,2	0,1	0,2	0,3
2,1	0,2	0,1	0,5	0,1
2,2	0,3	0,4	0,1	0,2
2,3	0,2	0,5	0,1	0,2
3,1	0,4	0,2	0,4	0,2
3,2	0,1	0,2	0,3	0,2
3,3	0,2	0,2	0,3	0,1



- Which action to perform in state 1,1?
  - According to greedy policy, the one with the highest value of Q.
    - Hence, action D.
  - Updating the value
    - $Q(s_{1,1}, a_D) = (1 - \alpha)\ Q(s_{1,1}, a_D) + \alpha(rt + Q(s_{1,2}, a_D)\ )$
    - $Q(s_{1,1}, a_D) = (1 - 0{,}5)*(0{,}4) + 0{,}5(0 + 1*0{,}4) = 0{,}4$

Q-Learning Algorithm

- Action 2:

	D	E	N	S
1,1	0,4	0,2	0,2	0,3
1,2	0,4	0,2	0,1	0,2
1,3	0,2	0,1	0,2	0,3
2,1	0,2	0,1	0,5	0,1
2,2	0,3	0,4	0,1	0,2
2,3	0,2	0,5	0,1	0,2
3,1	0,4	0,2	0,4	0,2
3,2	0,1	0,2	0,3	0,2
3,3	0,2	0,2	0,3	0,1

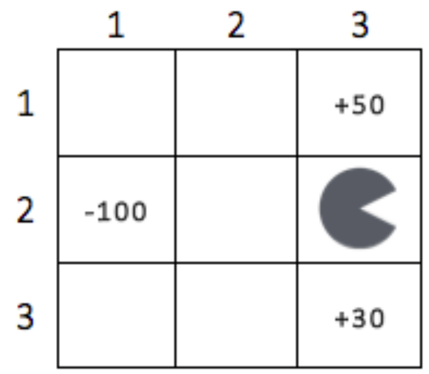
	1	2	3
1			+50
2	-100		
3			+30

- Which action to execute in state 1,2?
  - According to greedy policy, the one with the highest value of Q.
    - Hence, action D.
  - Updating the value
    - $Q(s_{1,2}, a_D) = (1 - \alpha)\, Q(s_{1,2}, a_D) + \alpha(r_t + Q(s_{1,3}, a_S))$
    - $Q(s_{1,2}, a_D) = (1 - 0,5)*(0,4) + 0,5(50 + 1*0,3) = 25,35$

Q-Learning Algorithm

- Action 3:

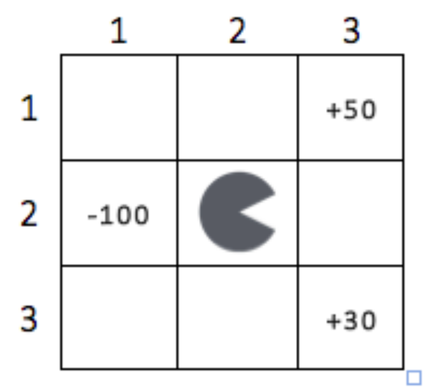
	D	E	N	S
1,1	0,4	0,2	0,2	0,3
1,2	25,35	0,2	0,1	0,2
1,3	0,2	0,1	0,2	0,3
2,1	0,2	0,1	0,5	0,1
2,2	0,3	0,4	0,1	0,2
2,3	0,2	0,5	0,1	0,2
3,1	0,4	0,2	0,4	0,2
3,2	0,1	0,2	0,3	0,2
3,3	0,2	0,2	0,3	0,1



- Which action to execute in state 1,3?
  - According to greedy policy, the one with the highest value of Q.
    - Hence, action S.
  - Updating the value
    - $Q(s_{1,3}, a_S) = (1 - \alpha)\ Q(s_{1,3}, a_S) + \alpha(r_t + Q(s_{2,3}, a_E))$
    - $Q(s_{1,3}, a_S) = (1 - 0{,}5)*(0{,}3) + 0{,}5(0 + 1*0{,}5) = 0{,}4$

- Action 4:

	D	E	N	S
1,1	0,4	0,2	0,2	0,3
1,2	25,35	0,2	0,1	0,2
1,3	0,2	0,1	0,2	0,4
2,1	0,2	0,1	0,5	0,1
2,2	0,3	0,4	0,1	0,2
2,3	0,2	0,5	0,1	0,2
3,1	0,4	0,2	0,4	0,2
3,2	0,1	0,2	0,3	0,2
3,3	0,2	0,2	0,3	0,1

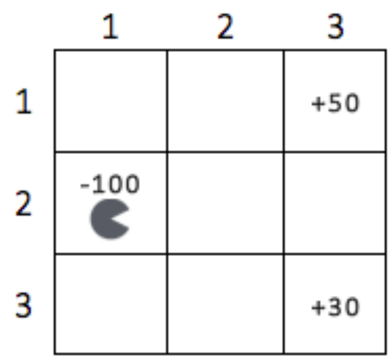


- What action to perform in state 2,3?
  - According to greedy policy, the one with the highest value of Q.
    - Therefore, action E.
  - Updating the value
    - $Q(s_{2,3}, a_E) = (1 - \alpha)\, Q(s_{2,3}, a_E) + \alpha(r_t + Q(s_{2,2}, a_E))$
    - $Q(s_{2,3}, a_E) = (1 - 0,5)*(0,5) + 0,5(0 + 1*0,4) = 0,45$

Q-Learning Algorithm

- Action 5:

	D	E	N	S
1,1	0,4	0,2	0,2	0,3
1,2	25,35	0,2	0,1	0,2
1,3	0,2	0,1	0,2	0,4
2,1	0,2	0,1	0,5	0,1
2,2	0,3	0,4	0,1	0,2
2,3	0,2	0,45	0,1	0,2
3,1	0,4	0,2	0,4	0,2
3,2	0,1	0,2	0,3	0,2
3,3	0,2	0,2	0,3	0,1

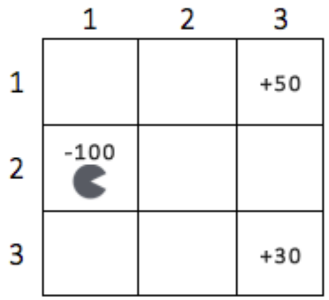


- What action to perform in state 1,3?
  - According to greedy policy, the one with the highest value of Q.
    - Hence, action E.
  - Updating the value
    - $Q(s_{2,2}, a_E) = (1 - \alpha)\, Q(s_{2,2}, a_E) + \alpha(r_t + Q(s_{2,1}, a_N))$
    - $Q(s_{2,2}, a_E) = (1 - 0{,}5)*(0{,}4) + 0{,}5(-100 + 1*0{,}5) = -49{,}55$

Q-Learning Algorithm

	D	E	N	S
1,1	0,4	0,2	0,2	0,3
1,2	25,35	0,2	0,1	0,2
1,3	0,2	0,1	0,2	0,4
2,1	0,2	0,1	0,5	0,1
2,2	0,3	-49,55	0,1	0,2
2,3	0,2	0,45	0,1	0,2
3,1	0,4	0,2	0,4	0,2
3,2	0,1	0,2	0,3	0,2
3,3	0,2	0,2	0,3	0,1

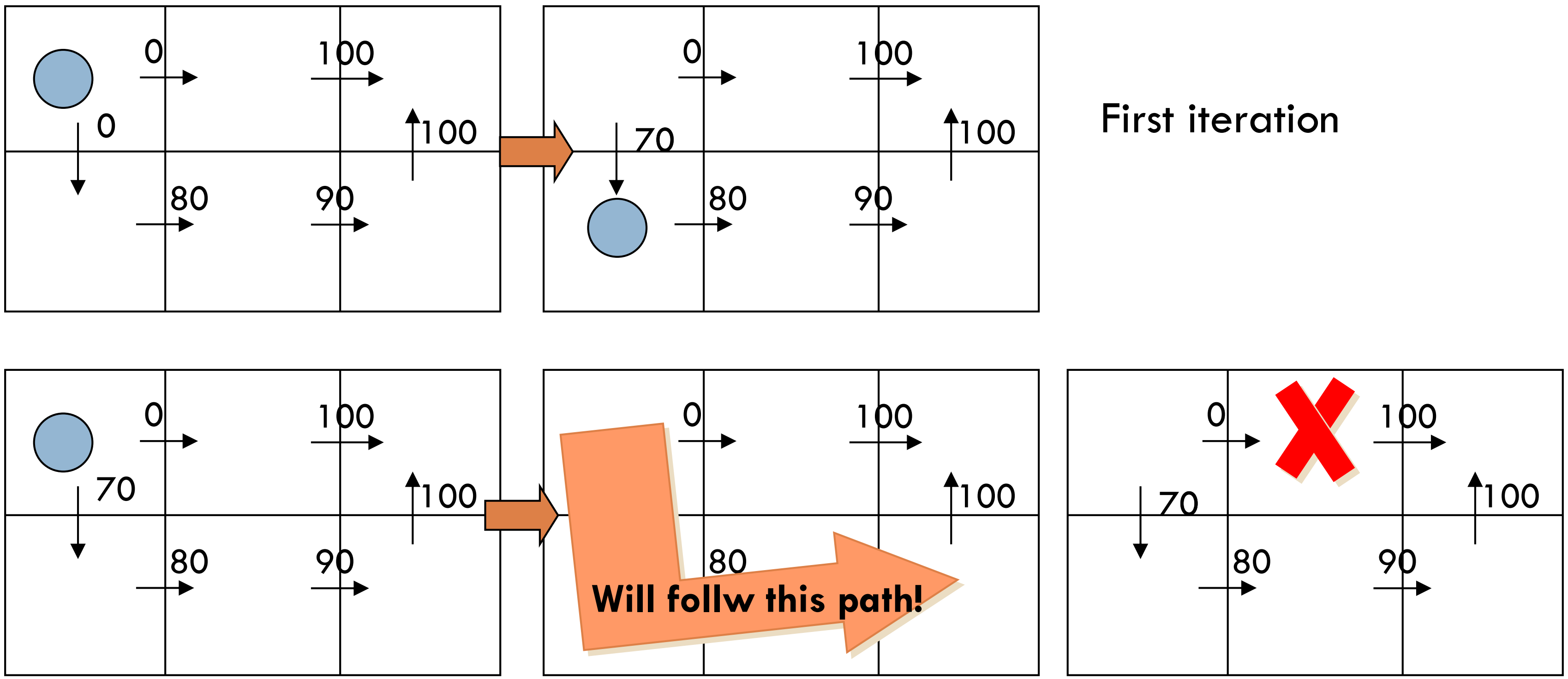
- Final state of the agent: 2,1



- Table Q has been updated for the 5 actions performed in the respective states.
- What did the agent learn?
  - Going to the right when it is 1,2 is good as it will receive high reinforcement
  - Going to the left when it is 2,2 is bad because it will receive negative reinforcement value
- This information is now embedded in the Q table
  - The value Q of action D in state 1,2 is high -> it tends to be chosen
  - The Q value of the E action in the 2,2 state is low -> it tends to be discarded

Dilemma

- If I always choose the Maximum value for Q, one can fall into a trap! Each step r = -10



Dilemma: Explore x Exploit

- To exploit
  - Choose the action that currently has the highest value Q (s, a)
- To explore
  - Choose a random action so that its Q (s, a) value is updated
- Dilemma:
  - Given that I have learned that Q (s, a) is worth 100, it is worth trying to perform the action a’ if Q (s, a') for now is 20?
- It depends on the environment, the number of actions already taken and the number of shares remaining

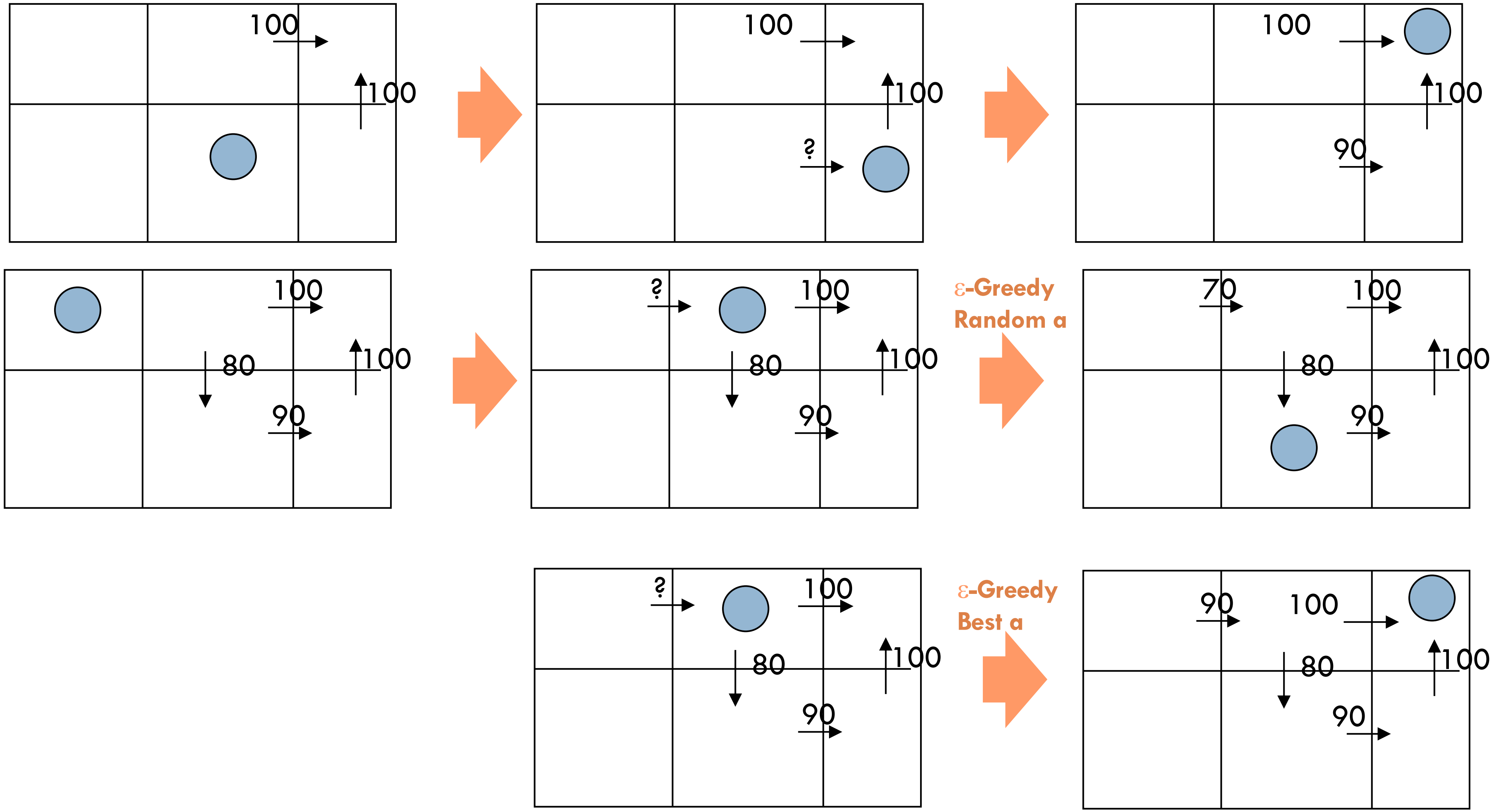
ε-Greedy

- Formula to solve the Dilemma:
  - ε-Greedy: random exploration
    - Given a random q

$$\pi(s_t) = \begin{cases} a_{random} & \text{if } q \leq \varepsilon \\ \arg\max_a Q(s_t, a_t) & \text{otherwise} \end{cases}$$

- The system will choose a random action if q <= ε or will choose the highest reward action if q> ε
- It is hoped, with this, that with many iterations the optimal solution (optimal policy)

ε-Greedy



Dense Rewards

- Dense rewards
  - Reinforcements other than zero are given to intermediate states
  - The opposite of delayed reward
  - Can reduce the complexity of learning
  - Lead to faster convergence

Considerations

- RL will solve many of your problems, however:
  - Needs a lot of training
- Choosing random actions can be dangerous and time consuming, but the system may not converge if they are not chosen
  - It can take a lot of time to learn
- Not all problems fit into MDP format
  - The algorithm finds the optimal solution (theoretically proved) in infinite iterations
  - That is, sometimes we have to settle for sub-optimal solutions

Examples

- Examples of RL applications in robotics

https://www.youtube.com/watch?v=2iNrJx6IDEo

Examples

- Examples of RL applications in robotics

https://www.youtube.com/watch?v=W_gxLKSsSIE

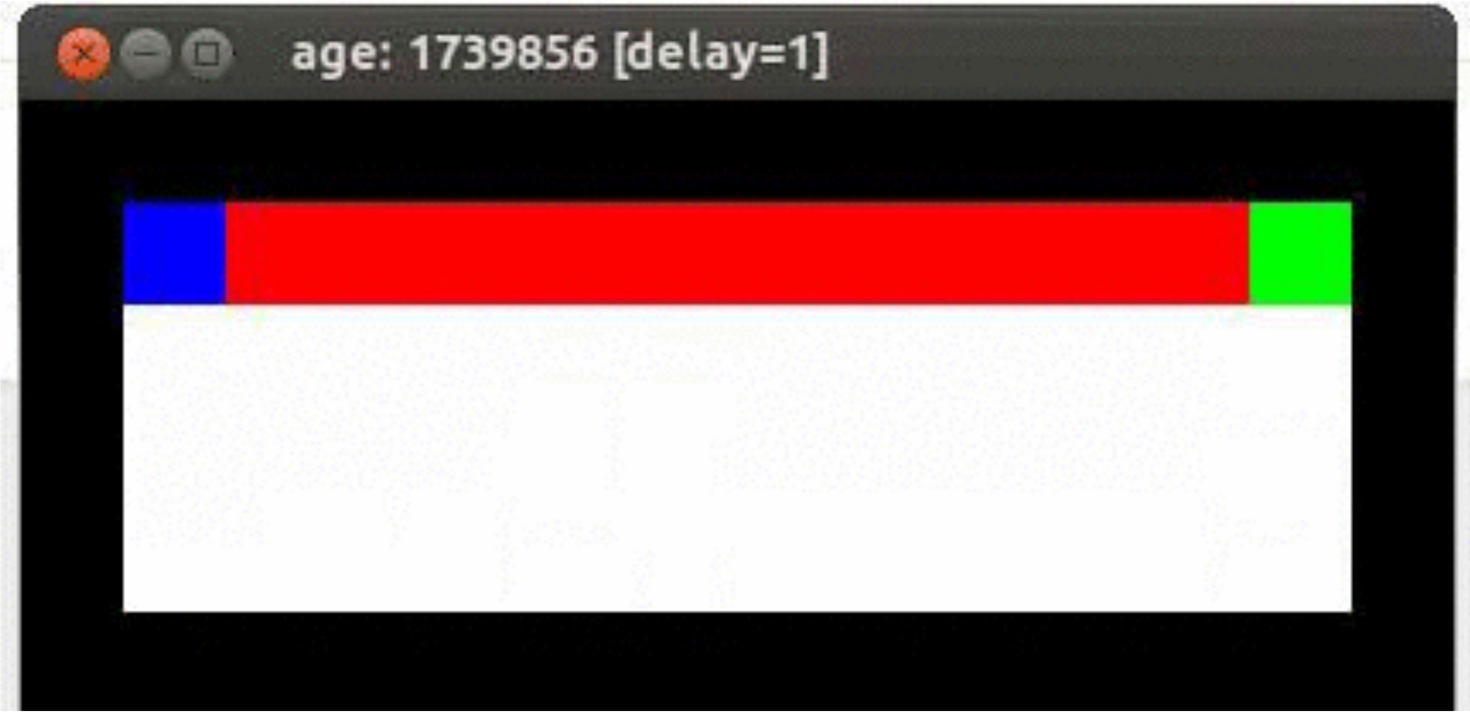
- SARSA (State-action-reward-state-action)
  - State s, performs action a, falls in state s'
    - In Q-learning, it is assumed that the best possible action will be taken from the state s'
    - In SARSA, this value will be the value of the actual share that was executed
    - That is, the value will only be updated when the new action choice occurs, based on the defined control policy
    - That is:
      - The Q-learning policy update is Q (s, a) = r(s) + alfa * max (Q(s'))
      - The SARSA policy update is Q (s, a) = r(s) + alfa * Q (s', a')

- Cliff example
  - Each action = -1
  - Hang on the cliff = -100
  - Reach target = 0
  - epsilon=0.1
  - alpha=0.1
  - gamma=0.9

  - https://studywolf.wordpress.com

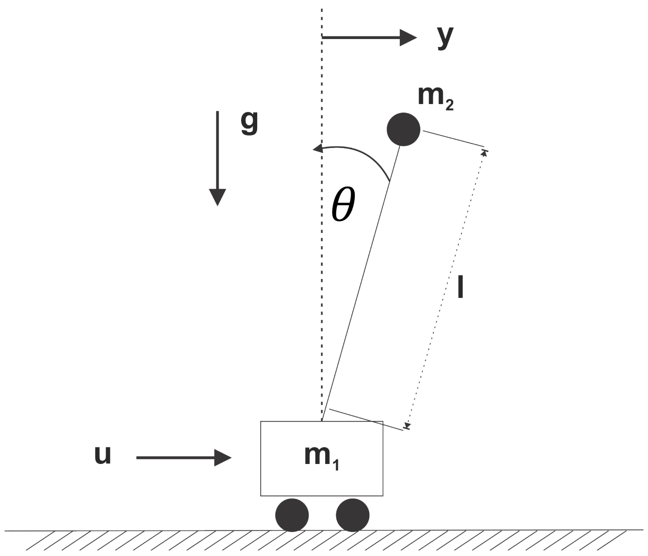
- Q-learning

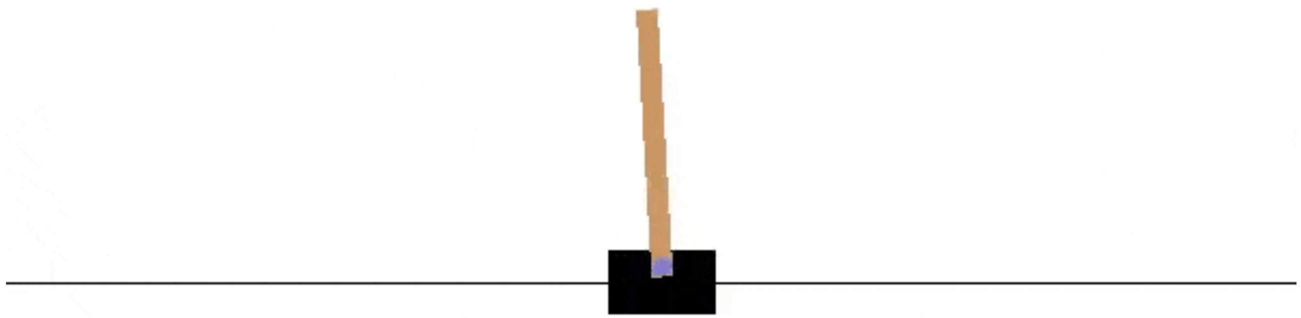
- SARSA



Examples of RL

Cart-pole balancing - traininng



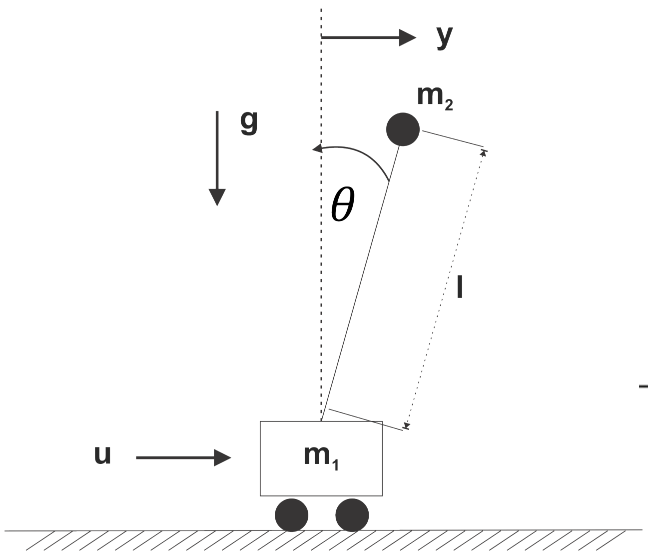


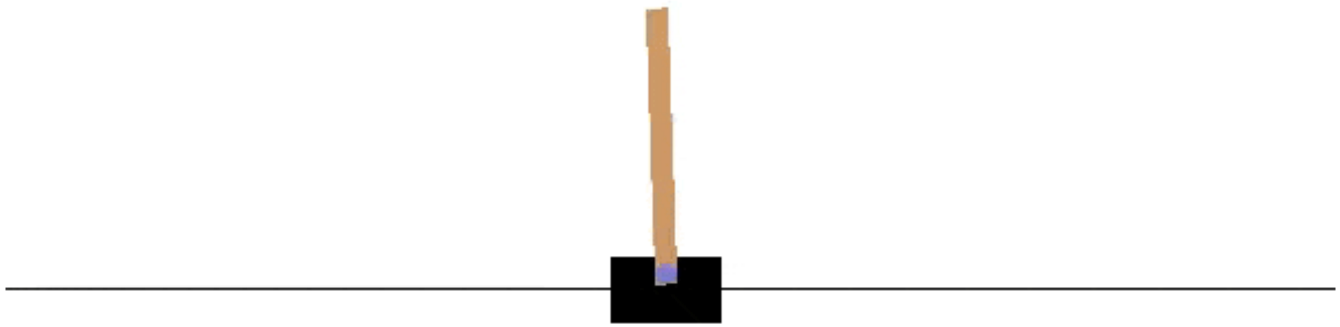
https://medium.com/@tuzzer/cart-pole-balancing-with-q-learning-b54c6068d947

- **Goal:** Balance the pole on top of a moving cart
- **State:** Pole angle, angular speed. Cart position, horizontal velocity
- **Actions:** horizontal force to the cart
- **Reward:** 1 at each time step if the pole is upright

Examples of RL

Cart-pole balancing – learned policy



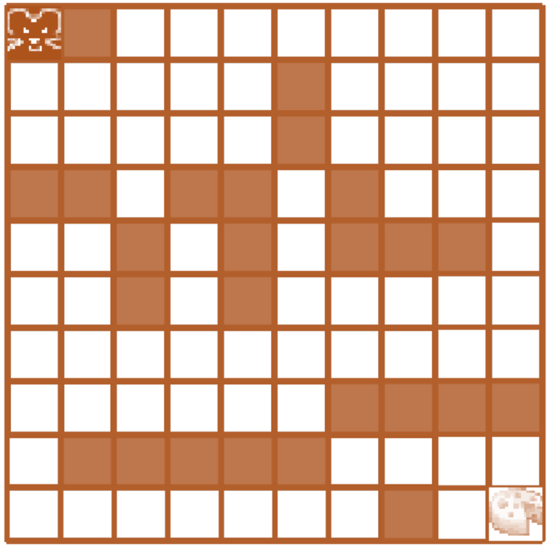


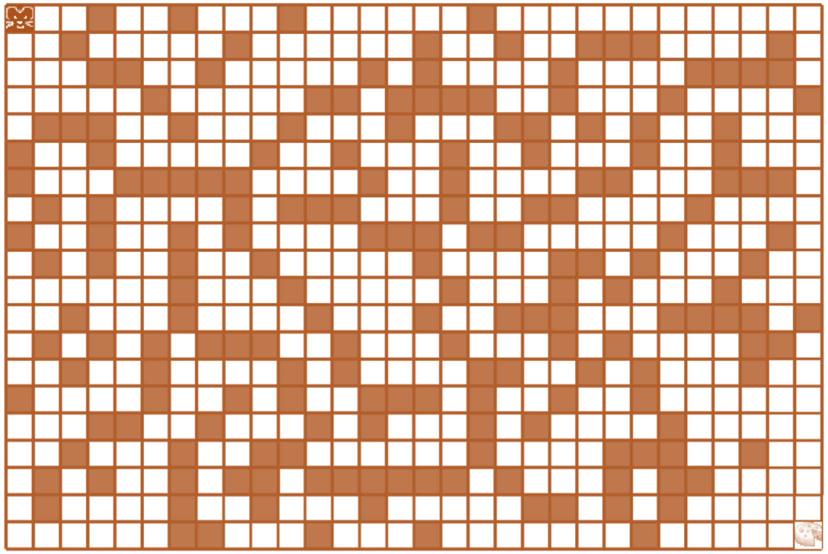
https://medium.com/@tuzzer/cart-pole-balancing-with-q-learning-b54c6068d947

- **Goal:** Balance the pole on top of a moving cart
- **State:** Pole angle, angular speed. Cart position, horizontal velocity
- **Actions:** horizontal force to the cart
- **Reward:** 1 at each time step if the pole is upright

Examples of RL

Maze-solving



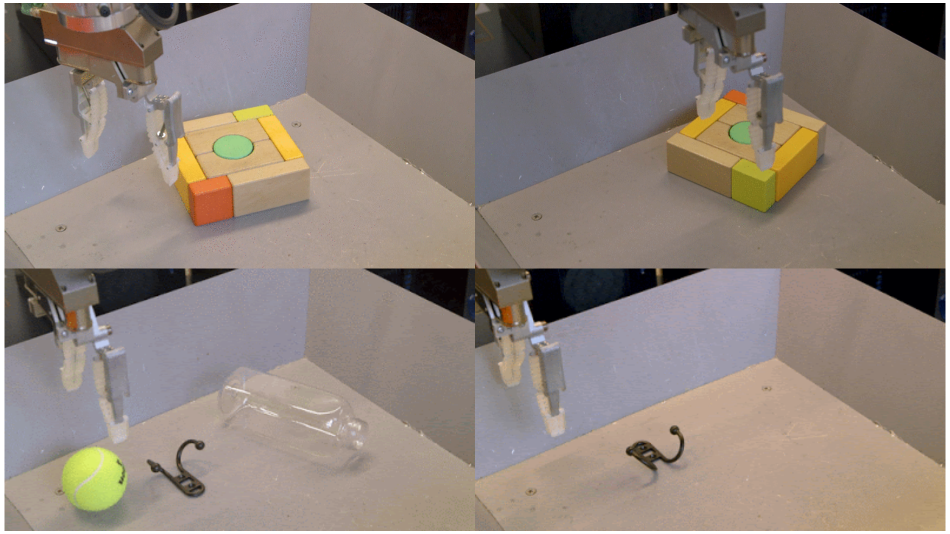


https://www.samyzaf.com/ML/rl/qmaze.html

- **Goal:** To get the cheese while avoiding collision
- **State**: Grid with cells that can be: occupied, free, target, visited
- **Actions**: left, up, right, down
- **Reward:**
  - 1 when the rat hits the cheese cell
  - -0.04 for each move from one cell to an adjacent cell
  - -0.8 for an attempt to move outside the maze boundaries
  - -0.75 when hit a blocked cell (dark-orange cell)
  - -0.25 points for any move to a cell which he has already visited
- **Stop criteria:** when the total reward hits -0.5 * maze.size

Examples of RL

Grasping Objects with Robotic Arm



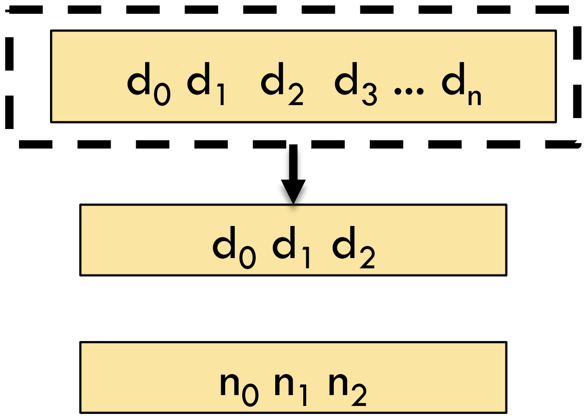
https://ai.googleblog.com/2018/06/scalable-deep-reinforcement-learning.html

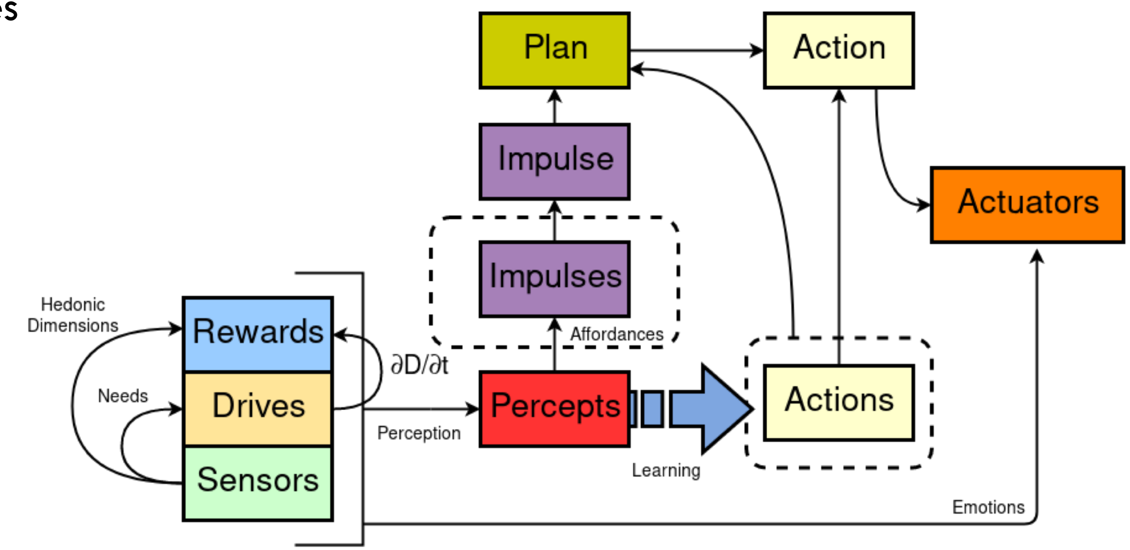
- **Goal:** Pick an object of different shapes
- **State:** Raw pixels from camera
- **Actions:** Move arm. Grasp
- **Reward:** Positive when pickup is successful

Examples of RL

Human Life

Homeostasis is a reference state for drives





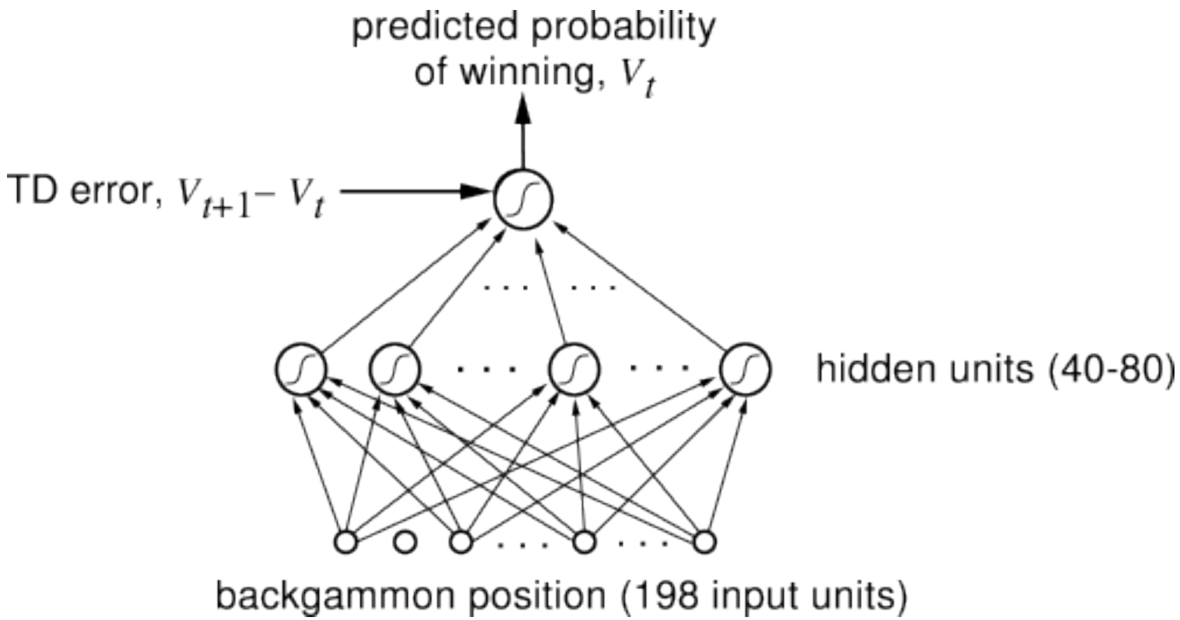
- **Goal:** To satisfy one' needs
- **State:** Sight. Hearing. Taste. Smell. Touch. Level on unsatisfaction of needs (drives)
- **Actions:** Think. Move.
- **Reward:** Homeostasis of needs?

Deep RL

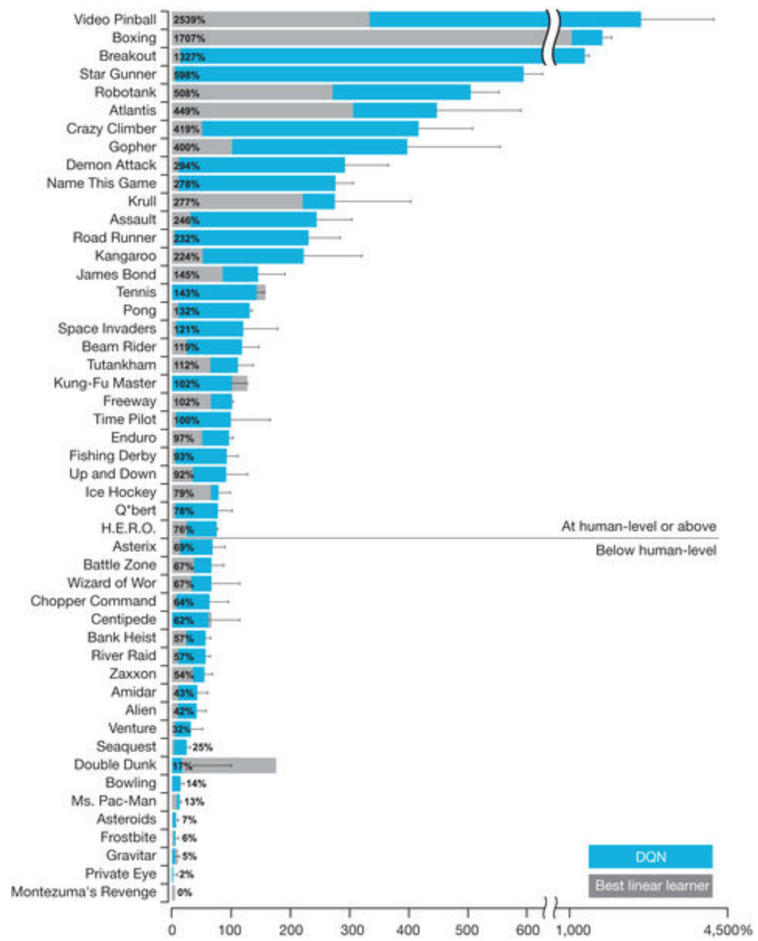
Deep Reinforcement Learning

- **RL** it's problematic for coping with large state-spaces and continuous values
- To help solving this problem, we could work with function approximators
- Any kind of function approximators may be employed in RL, however, neural nets are achieving best results
- **DRL**: Reinforcement learning that uses neural networks to approximate functions from complex data inputs
  - Reinforcement learning becoming tractable
  - No output examples needed
  - Able to derive robust control policies
  - Much lower custom handcrafted tuning
  - Possible to bypass human intuition
  - Able to cope with high dimensional inputs
  - DRL allows the development of control without explicit models
  - Recent theoretical and empirical improvements

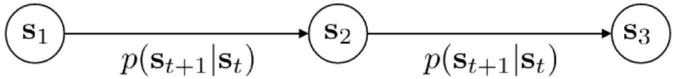
- NN for control goes back to the 90's
  - The thesis of Lin, 93 already discussed:
    - Experience replay and more
- TD-Gammon (Tesauro, 1992)
  - It stopped improving after about 1,500,000 games (auto-play) using 80 hidden units

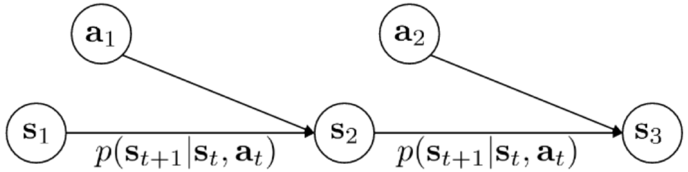


- But… playing backgammon does not look as cool as playing Atari!
- DeepMind breakthrough
  - Human-level control through deep reinforcement learning
- Posted in Nature in 2015
- Okay … the computer time is not the same as the time limited by the mechanical factor … but …



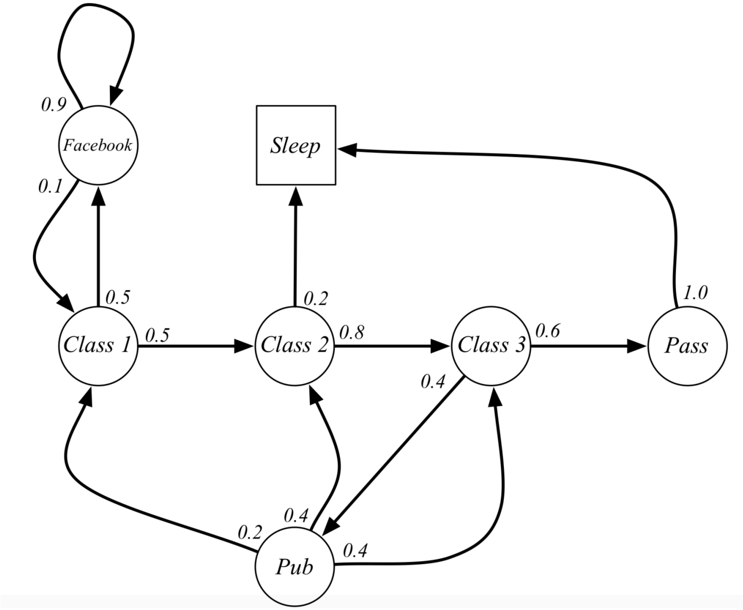
MC x MDP



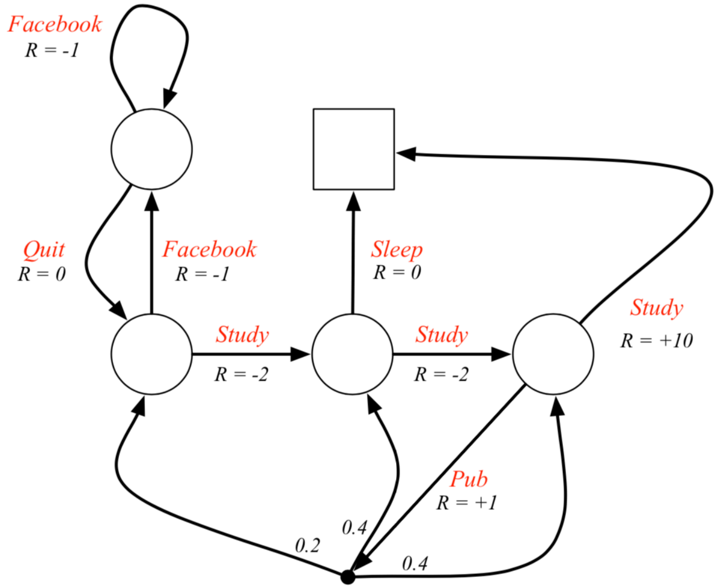


$$\mathcal{M}(\mathcal{S}, \mathcal{P})$$

$$\mathcal{M}(\mathcal{S}, \mathcal{A}, \mathcal{P})$$



MC



MDP

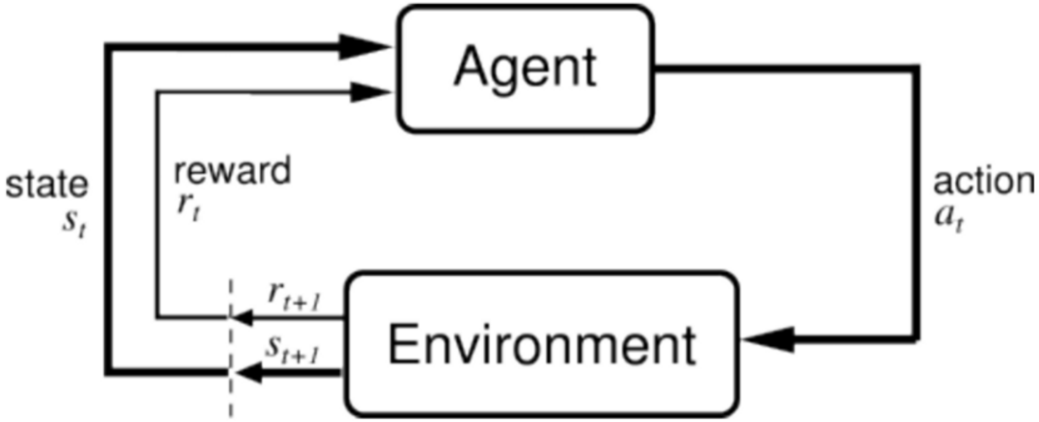
- Markov property

$$\mathcal{P}(s_{t+1}|s_t) = P(s_{t+1}|s_0, s_1, \cdots, s_{t-1}, s_t)$$

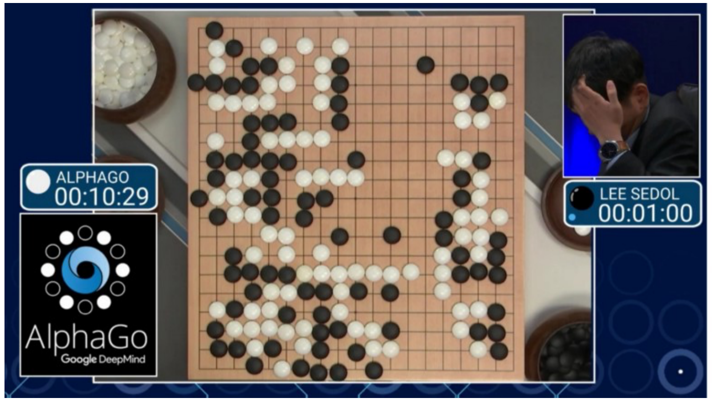
$$\mathcal{P}(s_{t+1}|s_t) = P(s_{t+1}|s_0, a_0, s_1, a_1, \cdots, s_{t-1}, s_t, a_t)$$

Reinforcement Learning

Problem Formulation



- **State** in MDP can be represented as raw images
- An **action** can be a move in a chess game or moving a robotic arm or a joystick
- For a GO game, the **reward** is very sparse: 1 if we win or -1 if we lose.



- RL problem may be formalized as MDPs:
  - Partially observable Markov decision process (POMDP) is a generalization of a Markov decision process (MDP)
  - Control problems are essentially concerned with continuous MDPs. POMDP requires the inclusion of O:
    - the observation space
- Reinforcement Learning is an optimization problem for the policy
- What to approach?
    - Policies (select the next action)
    - Value functions (measures the quality of actions or state-action pairs)
  - **Value function** x **Reinforcement**
    - The **value function** measures the quality of the state over time.
    - **Reinforcement** is immediate

- S is the state space, or the finite set of states in the environment
- A is the action space, the finite set of actions that an agent can execute
- $P(s_{t+1}, r_t \mid s_t, a_t)$ is the transition operator. It specifies the probability that the environment will emit reward r_t and transit to state s_{t+1} for each state s_t and action a_t
  - The transition function is the system dynamics. It predicts the next state after taking action. It is called the **model** which plays a major role when we discuss Model-based RL
- r_t is the reward signal at a given instant t, as r ∈ R
- ρ_0 is the initial state probability distribution
- $\gamma \in [0,1]$, is the discount rate, used to adjust the ratio between the contribution of recent rewards and past rewards.

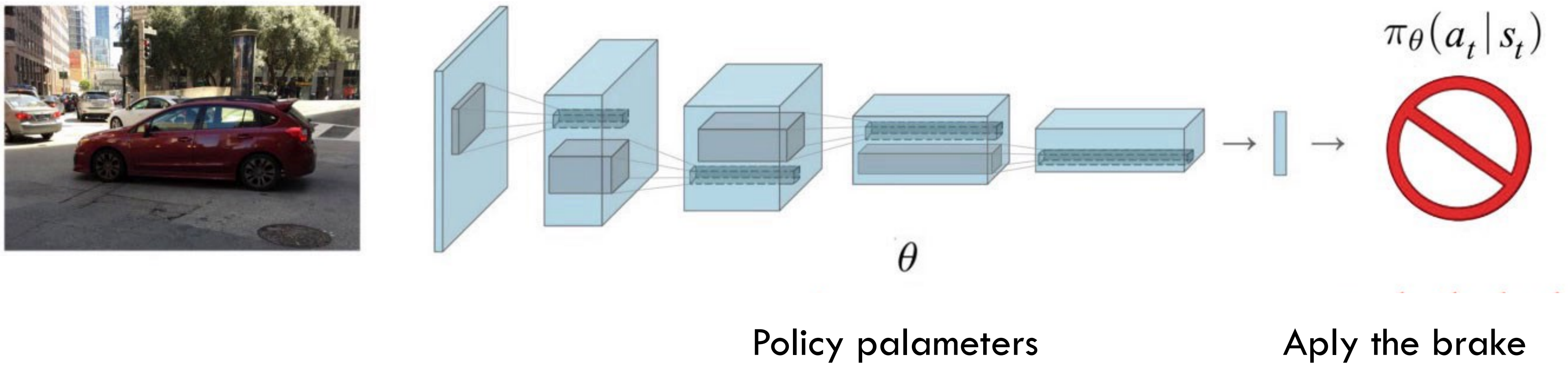
Agent-environment interaction

- Return and Discounted Return
  - Episodic approach of Reinforcement Learning:
    - $R = r_0 + r_1 + r_2 + \cdots + r_{T-1} = \sum_{t=0}^{T-1} r_t$
  - However, when no terminal state is naturally given or we desire a weighting of instantaneous rewards:
    - $\eta_\pi = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$

- **Policies**
  - describe the behavior of an agent and may be deterministic or stochastic
  - $\pi(a|s)$ maps states to probabilities of selecting each possible action at a given state
  - It can be **deterministic** $\pi(s)$, which directly maps an state s to a determined action a
    - $\pi: S \rightarrow A$
  - It can be **stochastic** $\pi(a|s)$, which given a state s, each action a ∈ A(s) has an associated probability to be chosen
    - $\pi: SxA \rightarrow [0,1], \forall s \in S \; (\sum_{a \in A} \pi(s,a) = 1)$
- **Value Functions**
  - Function that maps states s or state-action pairs (s,a) to a real number
  - Interpreted as the measure of how good it is for the agent to be at a given state or how good it is to perform a given action in a given state.

Agent-environment interaction

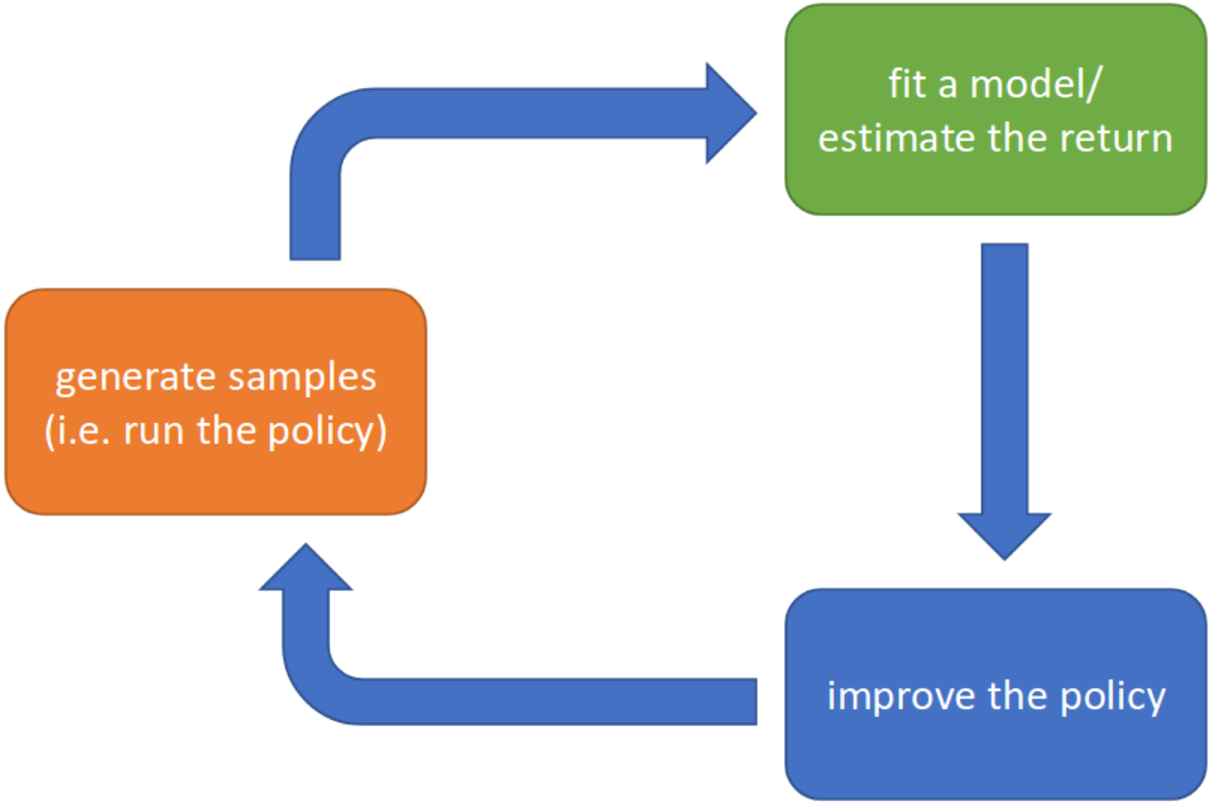
Policy



- **Value function based algorithms**

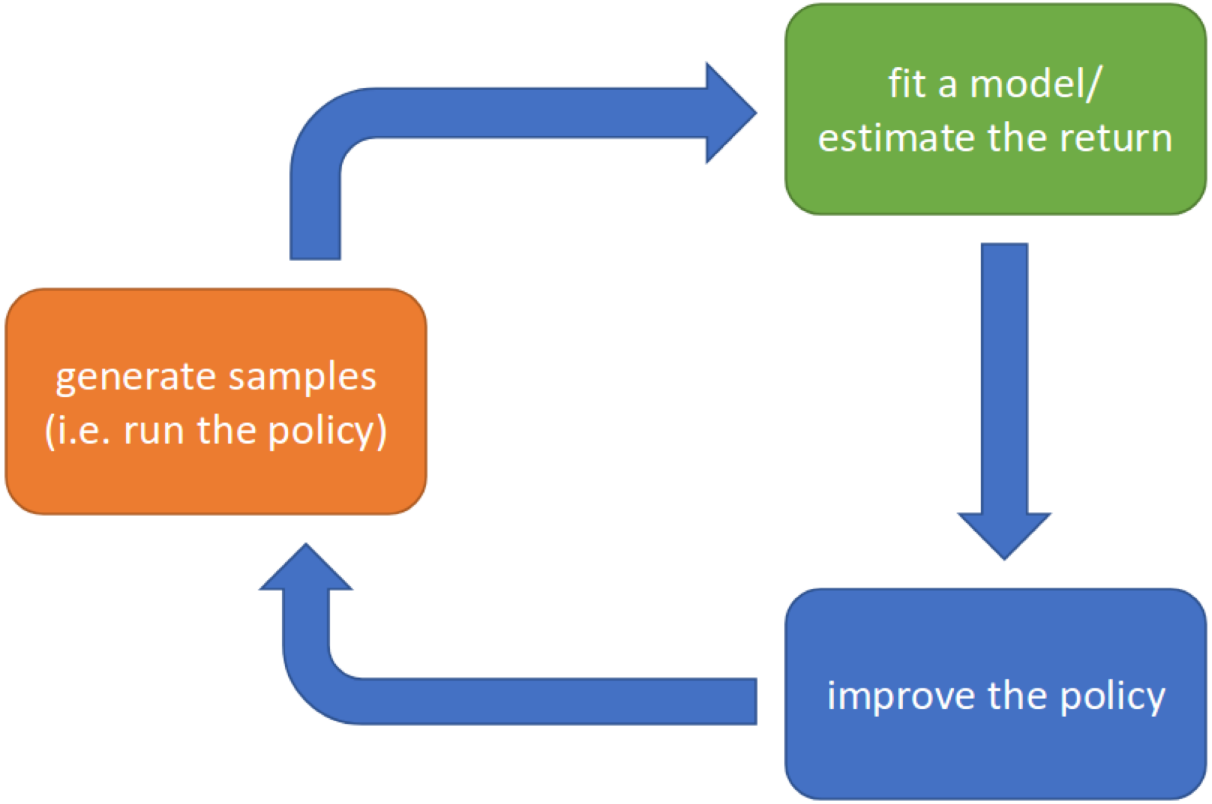
- Value Functions
  - **V(s)**
    - $V_\pi \; (s_t) = \mathbb{E}_\pi \, [\eta_t | s_t]$
    - $V_\pi \; (s_t) = \mathbb{E}_\pi \, [\sum_{k=0}^{\infty} \gamma^k \, r_{t+k} | s_t]$
  - **Q(s,a)**
    - $Q_\pi \, (s_t, a_t) = \mathbb{E}_\pi \, [\eta_t | s_t, a_t]$
    - $Q_\pi \, (s_t, a_t) = \mathbb{E}_\pi \, [\sum_{k=0}^{\infty} \gamma^k \, r_{t+k} | s_t, a_t]$

RL Algorithms



S. Levine, "Lecture 4: Reinforcement learning introduction - cs 294-112 at uc berkeley: Deep reinforcement learning." [Online]. Available: http://rail.eecs.berkeley.edu/deeprlcourse/

RL Algorithms: Value function based



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

$$\pi^*(s) = argmaxa \quad Q(s,a)$$

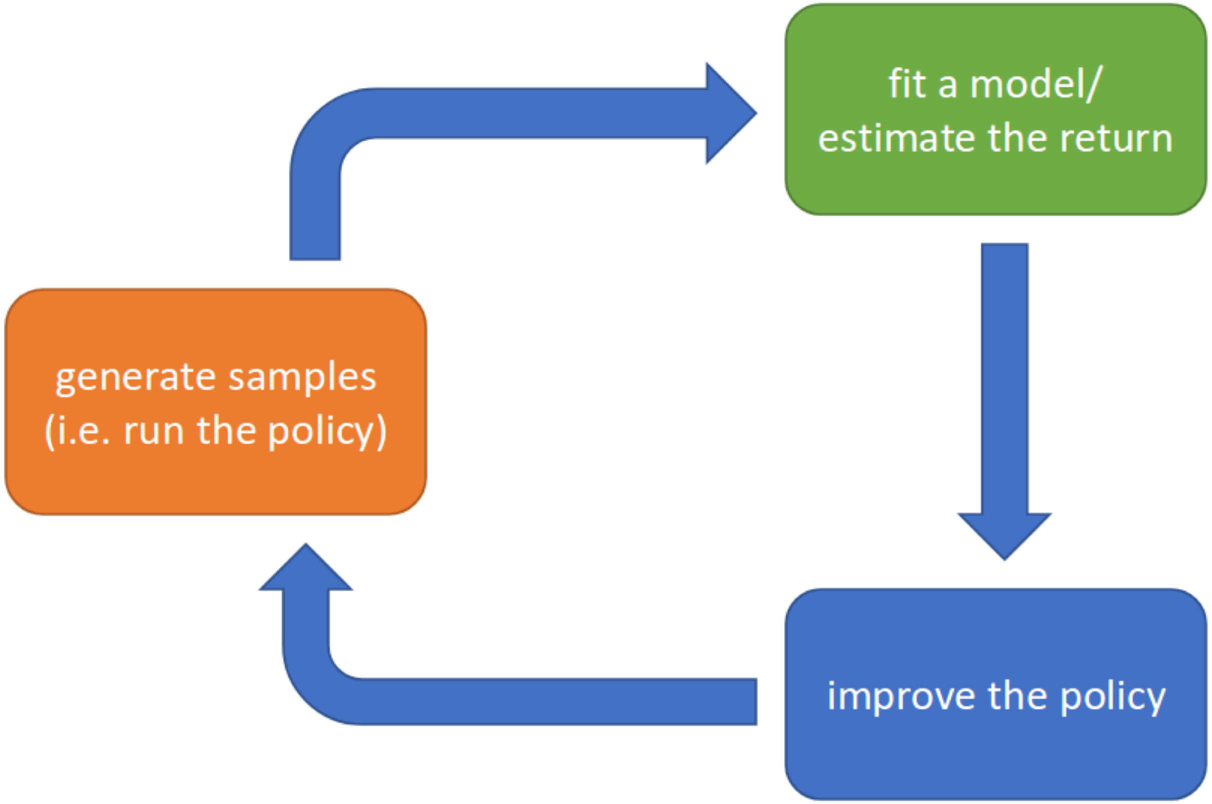
S. Levine, "Lecture 4: Reinforcement learning introduction - cs 294-112 at uc berkeley: Deep reinforcement learning." [Online]. Available: http://rail.eecs.berkeley.edu/deeprlcourse/

Policy optimization based algorithms

- Policies are often represented as a parameterized function π_θ, typically encoded by an Artificial Neural Network, where θ is the net parameter set
- We can optimize the policy through gradient-based optimization or gradient-free methods (e.g.: AG)
- Generally, Stochastic Gradient Ascent or one of its variations is used to optimize an objective function of the form
  - $\mathcal{L}^{PG}(\theta) = \widehat{\mathbb{E}_t}\,[log_{\pi_\theta}(a_t|s_t)[\sum_t r(s_t, a_t)]]$

maximizing the likelihood of taking that action in that state

RL Algorithms: Policy optimization based algorithms



Evaluate returns

$R = \sum_{t=0}^{T-1} r_t$

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E} \left[\sum_t r_t \right]$$

S. Levine, "Lecture 4: Reinforcement learning introduction - cs 294-112 at uc berkeley: Deep reinforcement learning." [Online]. Available: http://rail.eecs.berkeley.edu/deeprlcourse/

- **Policy optimization based algorithms**
  - The regular ("vanilla") policy gradients are susceptible to high variance when the objective function considers simply the "reward to go"
  - To reduce the variance of policy gradients, without introducing bias to the model, is to use an alternative objective function with a baseline b
    - $\mathcal{L}^{PG}(\theta) = \widehat{\mathbb{E}_t}\left[log_{\pi_\theta}(a_t|s_t)[\sum_t r(s_t, a_t) - b]\right]$
    - $b = \frac{1}{N}\sum_{i=1}^{N} r_i$

- **Actor-Critic**
  - Improves the choice of baseline
  - An actor-critic algorithm consists of a policy gradient method that works in association with a value estimator $\widehat{V_{\pi}}(s)$.
- The actor is the policy that infers the best actions to take, while the critic is the component that bootstraps the evaluation of the current policy
- Concept of advantage function A:
  - $A_{\pi} = Q_{\pi}(s_t, a_t)$- $V_{\pi}(s_t)$

- Policy gradients rely on a stochastic gradient ascent, or other first order optimization technique, to maximize some performance measure $\eta(\theta)$. The policy π_θ is, commonly, a deep or shallow neural network
- One of the most frequently used gradient estimator has the form:
  - $\widehat{g} = \widehat{\mathbb{E}_t}\left[log_{\pi_\theta}(a_t|s_t)\widehat{A_t}\right]$
- Derived from the object function:
  - $\mathcal{L}^{PG}(\theta) = \widehat{\mathbb{E}_t}\left[log_{\pi_\theta}(a_t|s_t)\widehat{A_t}\right]$

Deep Reinforcement Learning

Taxonomy

Better
Sample Efficient

Less
Sample Efficient

Model-based
(100 time steps)

Off-policy
Q-learning
(1 M time steps)

Actor-critic

On-policy
Policy Gradient
(10 M time steps)

Evolutionary/
gradient-free
(100 M time steps)

- Learn the model of the world, then plan using the model
- Update model often
- Re-plan often

Value-based

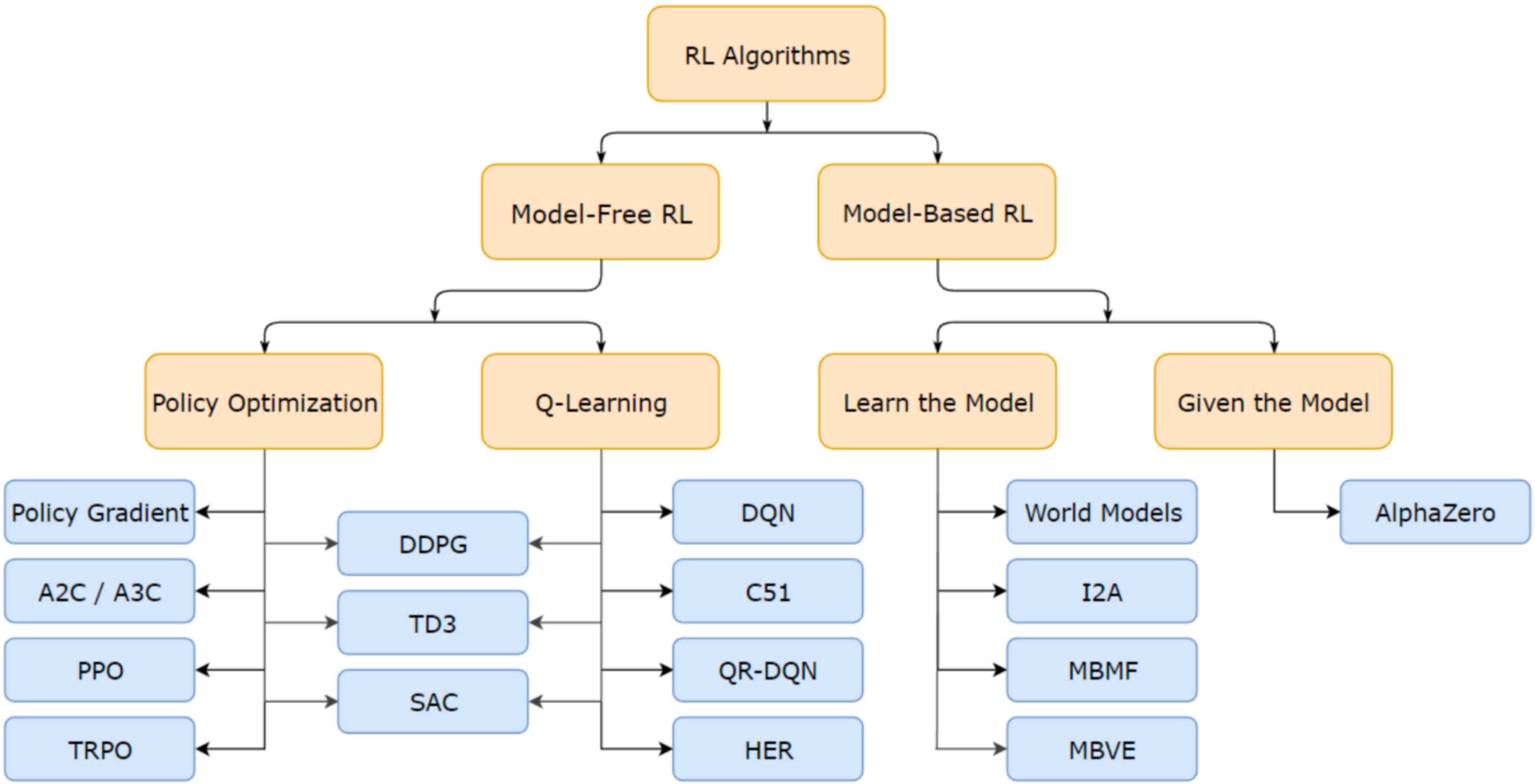
- Learn the state or state-action value
- Act by choosing best action in state
- Exploration is a necessary add-on

Policy-based

- Learn the stochastic policy function that maps state to action
- Act by sampling policy
- Exploration is baked in

Deep Reinforcement Learning

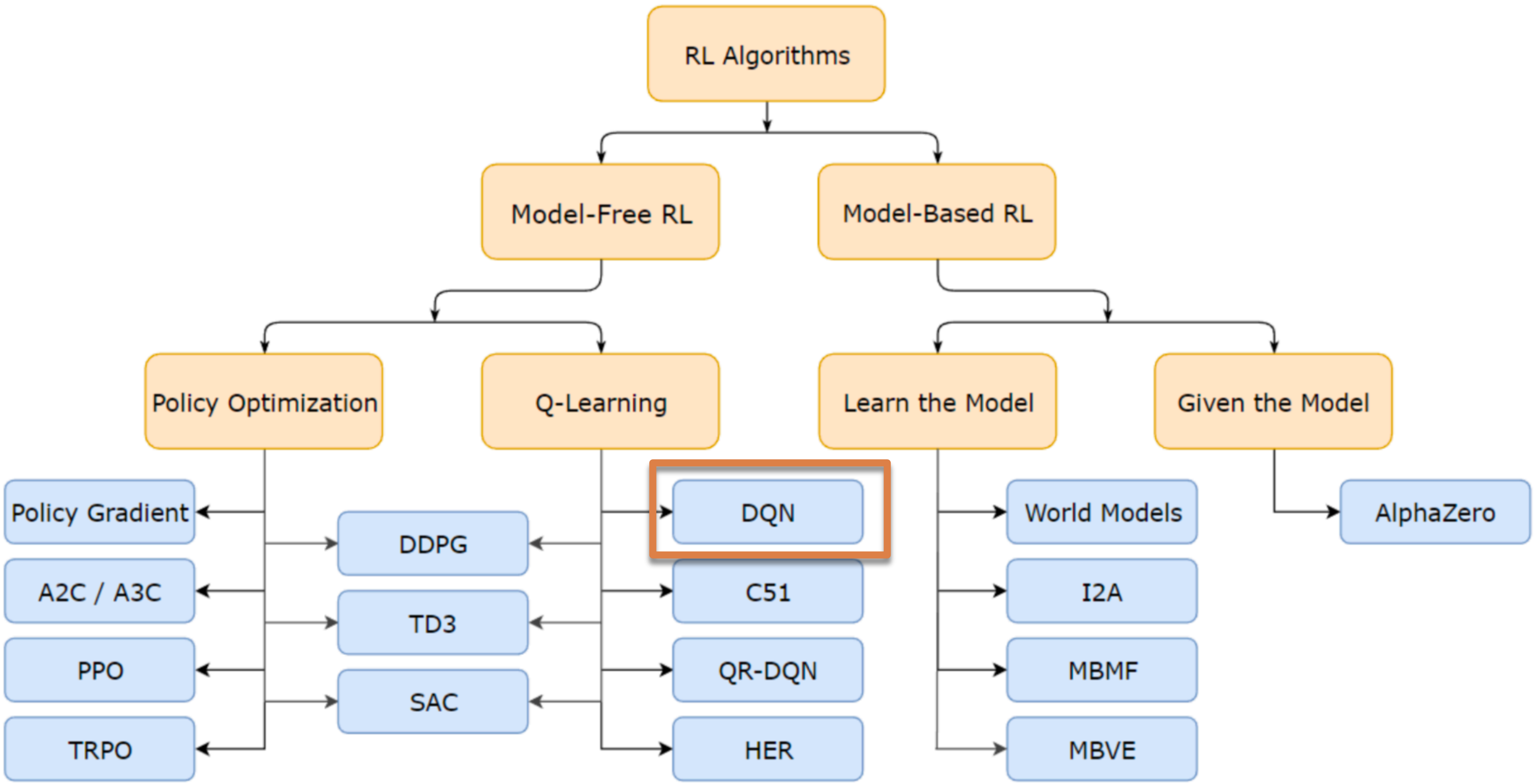
Taxonomy



Link: https://spinningup.openai.com

Deep Reinforcement Learning

Taxonomy

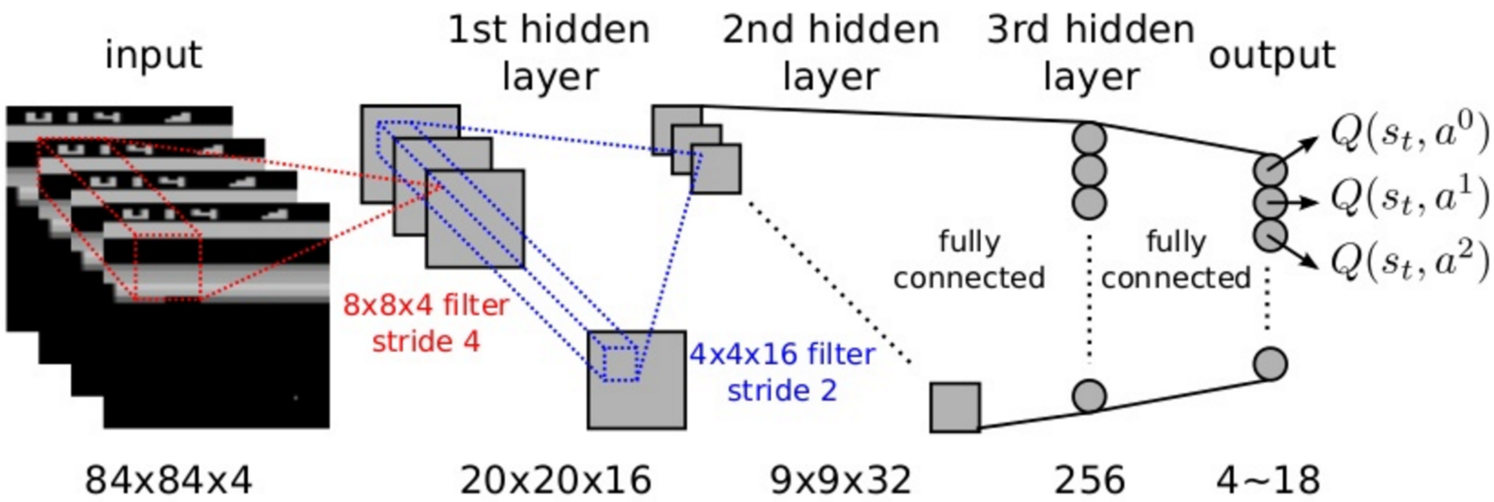


Link: https://spinningup.openai.com

Deep Reinforcement Learning

DQN

- DQN
  - End-to-end learning of Q(s,a) values from the pixels s
  - The input state s is a stack of pixels from the last 4 frames
  - The output is Q (s, a) for the 18 joystick / button positions
  - The reward is the change in the score for this step



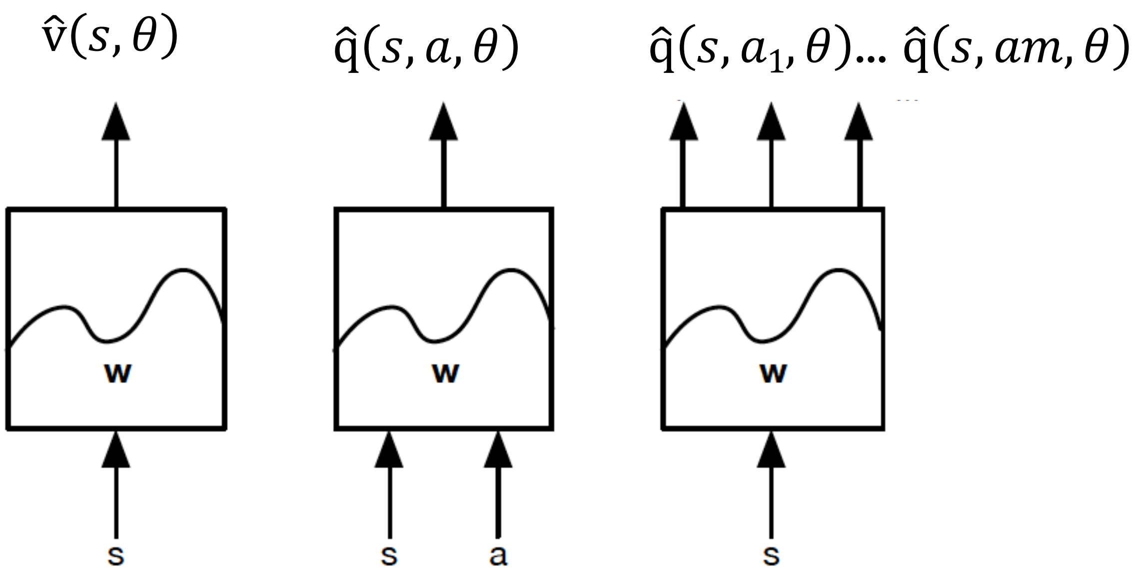
Learning rate   Discount factor

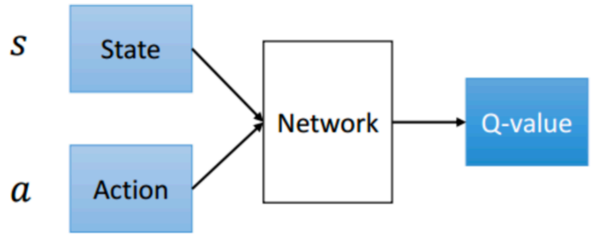
$$Q_{t+1}(s_t, a_t) = Q_t\ (s_t, a_t) + \alpha(R_{t+1} + \gamma max_a Q_t\ (s_{t+1}, a\ \ ) - Q_t\ (s_t, a_t))$$

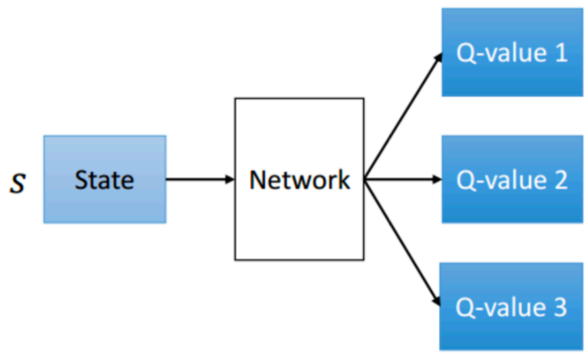
New state   Old state   Reward

DQN

- Use a neural network to approximate the Q-function:
  - $Q(s,a;\theta) \approx Q^*(s,a)$

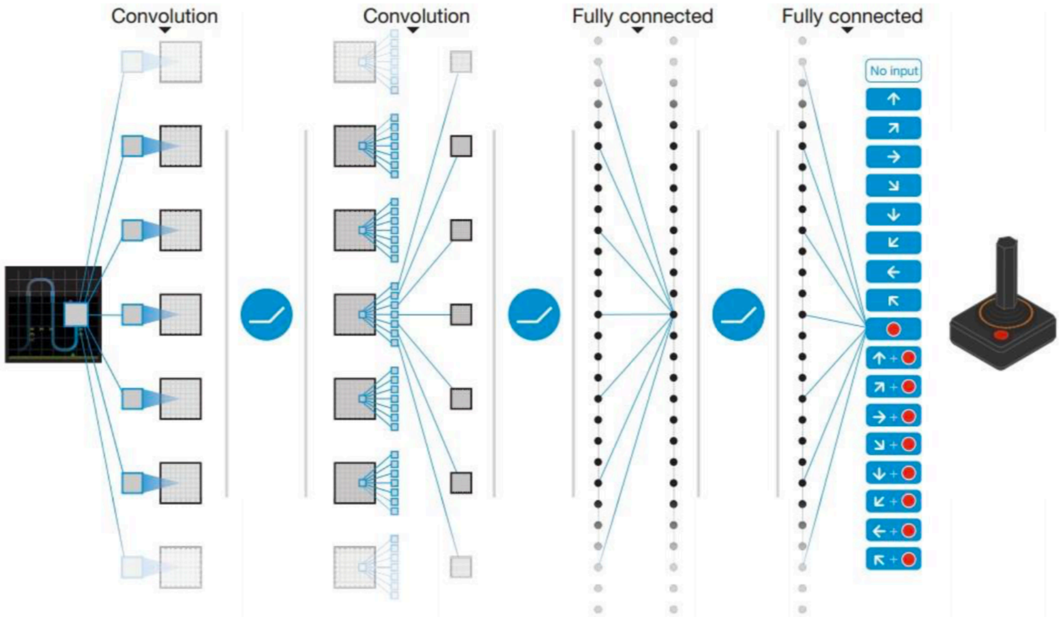






- DQN (With Experience replay)
  - Take action a_t according to ε-greedy policy
  - Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
  - Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
  - Compute Q-learning target
  - Optimize MSE between Q-network and Q-learning targets
    - $\mathcal{L} = \mathbb{E}_{\widehat{s,a,r,s'}\sim\mathcal{D}} [(r + \gamma max_{a'} Q(s', a') - Q(s, a))^2]$
    - Using variant of SGD
    - $\Delta\theta = \alpha[\underbrace{[(\overbrace{R + \gamma max_{a'} \hat{Q}(s', a')}^{\text{Target}}) - \overbrace{\hat{Q}(s, a)}^{\text{Prediction}}}_{\text{TD Error}}]\underbrace{\nabla \hat{Q}(s, a)}_{\text{Gradient of our current prediction}}$

- DQN for Atari



Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

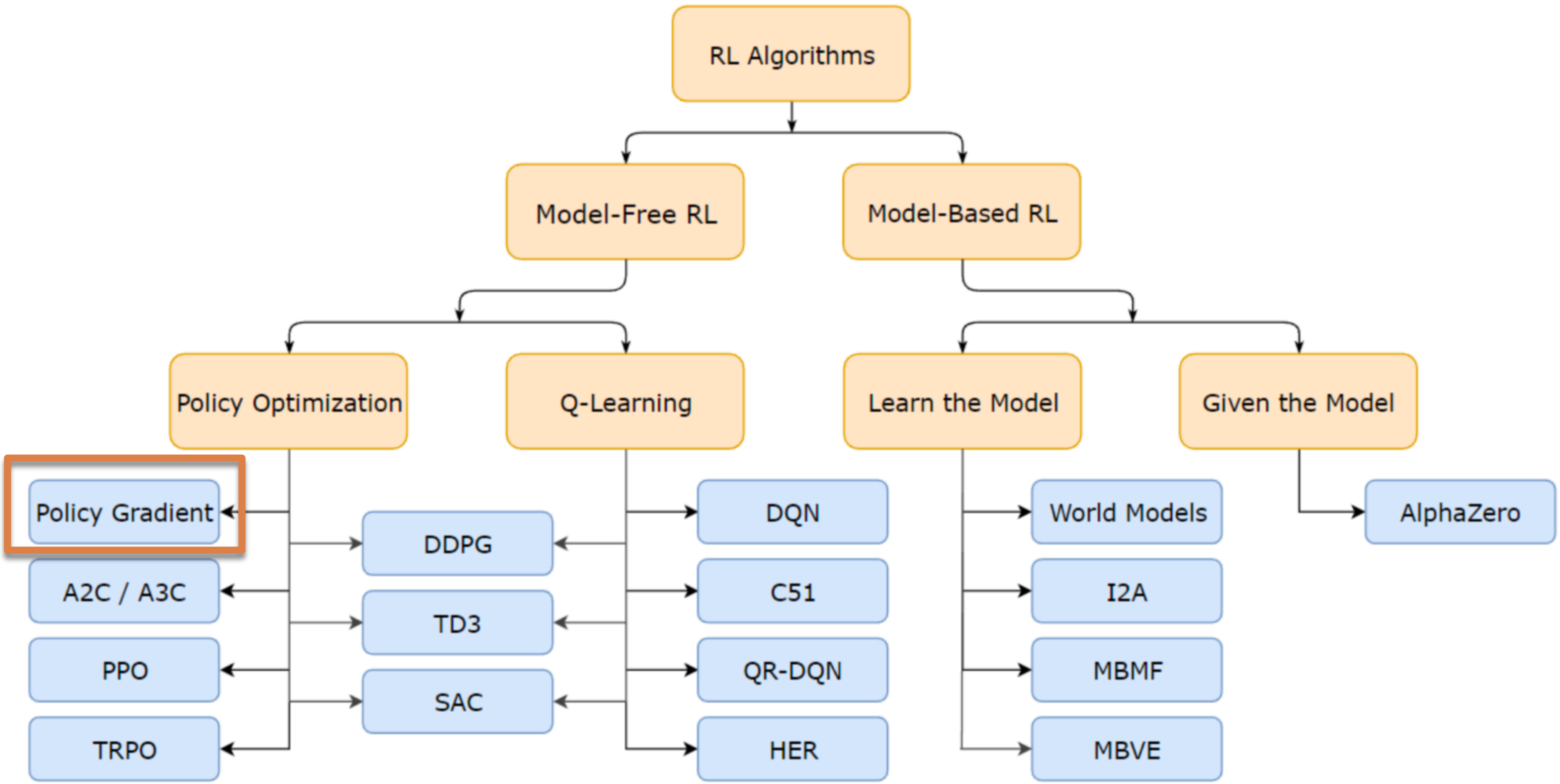
- DQN tricks
  - Experience Replay
    - Stores experiences (actions, state transitions, and rewards) and creates mini-batches from them for the training process
  - Fixed Target Network
    - Error calculation includes the target function depends on network parameters and thus changes quickly. Updating it only every 1,000 steps increases stability of training process
- Variations
  - Dueling DQN (DDQN)
  - Decompose **Q(s,a)**
    - **V(s)**: the value of being at that state
    - **A(s,a)**: the advantage of taking action ***a*** in state **s** versus all other possible actions at that state
  - Use two streams:
    - one that estimates the state value V(s)
    - one that estimates the advantage for each action A(s,a)
  - Useful for states where action choice does not affect Q(s,a)

Deep Reinforcement Learning

Deep Reinforcement Learning

Deep Reinforcement Learning

Taxonomy



Link: https://spinningup.openai.com

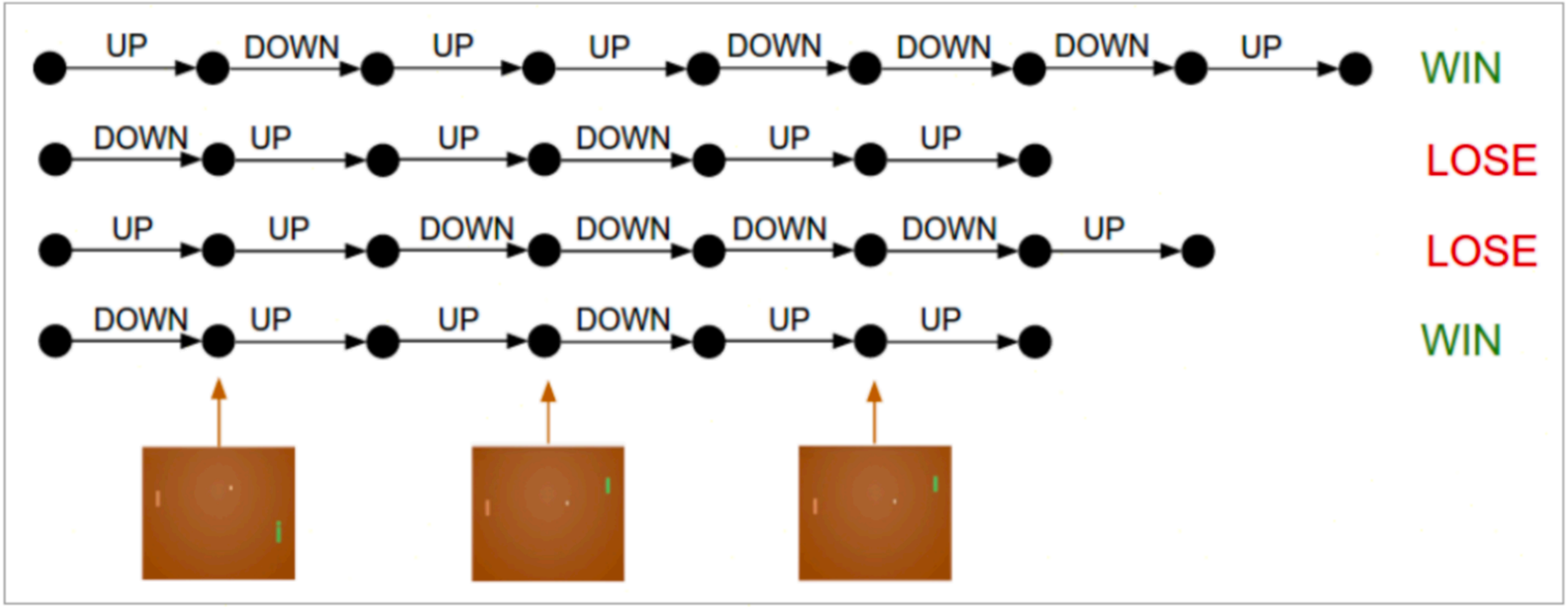
- Before, the policy was to use the best action
- But ... and if it is simpler to represent the policy?
- Value Based
  - Learned value function
  - Implied Policy
    - For example, ε-greedy
- **DQN (off-policy):** Approximate Q and infer optimal policy
- **PG (on-policy):** Directly optimize policy space
- Policy-based
  - No function value
  - Policy Learned

- Policy Gradient
  - Adjust the policy to make it better
  - We will directly adjust the policy
  - Let's see our experience and adjust following the gradient
- Benefits:
  - Better Convergence Properties
  - Effective with high-dimensional or continuous action spaces
  - Can learn stochastic policies
- Methods
  - Finite differences
  - Monte-Carlos
  - Actor-critic

Deep Reinforcement Learning

Policy Gradient

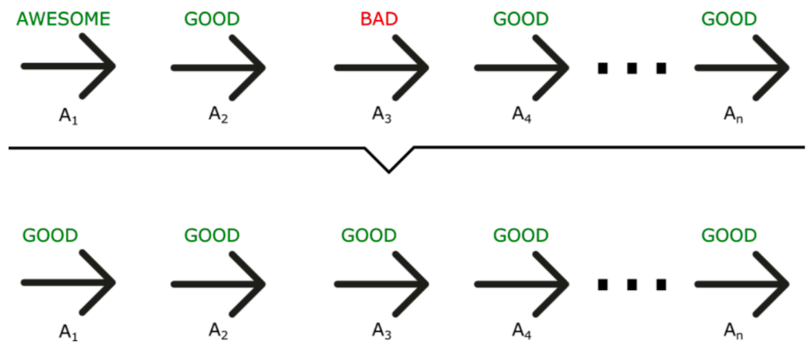
Policy Gradients: Run a policy for a while. See what actions led to high rewards. Increase their probability.



- **REINFORCE:** Policy gradient that increases probability of good actions and decreases probability of bad action:
  - $\nabla \mathbb{E}_t[R_t] = \mathbb{E}\left[\nabla_\theta \;\; log P(a) R_t\right]$

- Pros vs DQN:
  - **Messy World:** If Q function is too complex to be learned, DQN may fail miserably, while PG will still learn a good policy
  - **Speed:** Faster convergence
  - **Stochastic Policies:** Capable of learning stochastic policies - DQN can't
  - **Continuous actions:** Much easier to model continuous action space
- Cons vs DQN:
  - **Data:** Sample inefficient (needs more data)
  - **Stability:** Less stable during training process
  - Poor **credit assignment** to (state, action) pairs for delayed rewards

- Pros vs DQN:
  - **Messy World:** If Q function is too complex to be learned, DQN may fail miserably, while PG will still learn a good policy
  - **Speed:** Faster convergence
  - **Stochastic Policies:** Capable of learning stochastic policies - DQN can't
  - **Continuous actions:** Much easier to model continuous action space
- Cons vs DQN:
  - **Data:** Sample inefficient (needs more data)
  - **Stability:** Less stable during training process
  - Poor **credit assignment** to (state, action) pairs for delayed rewards
- **Problem with REINFORCE:**
  - Calculating the reward at the end, means all the actions will be averaged as good because the total reward was high

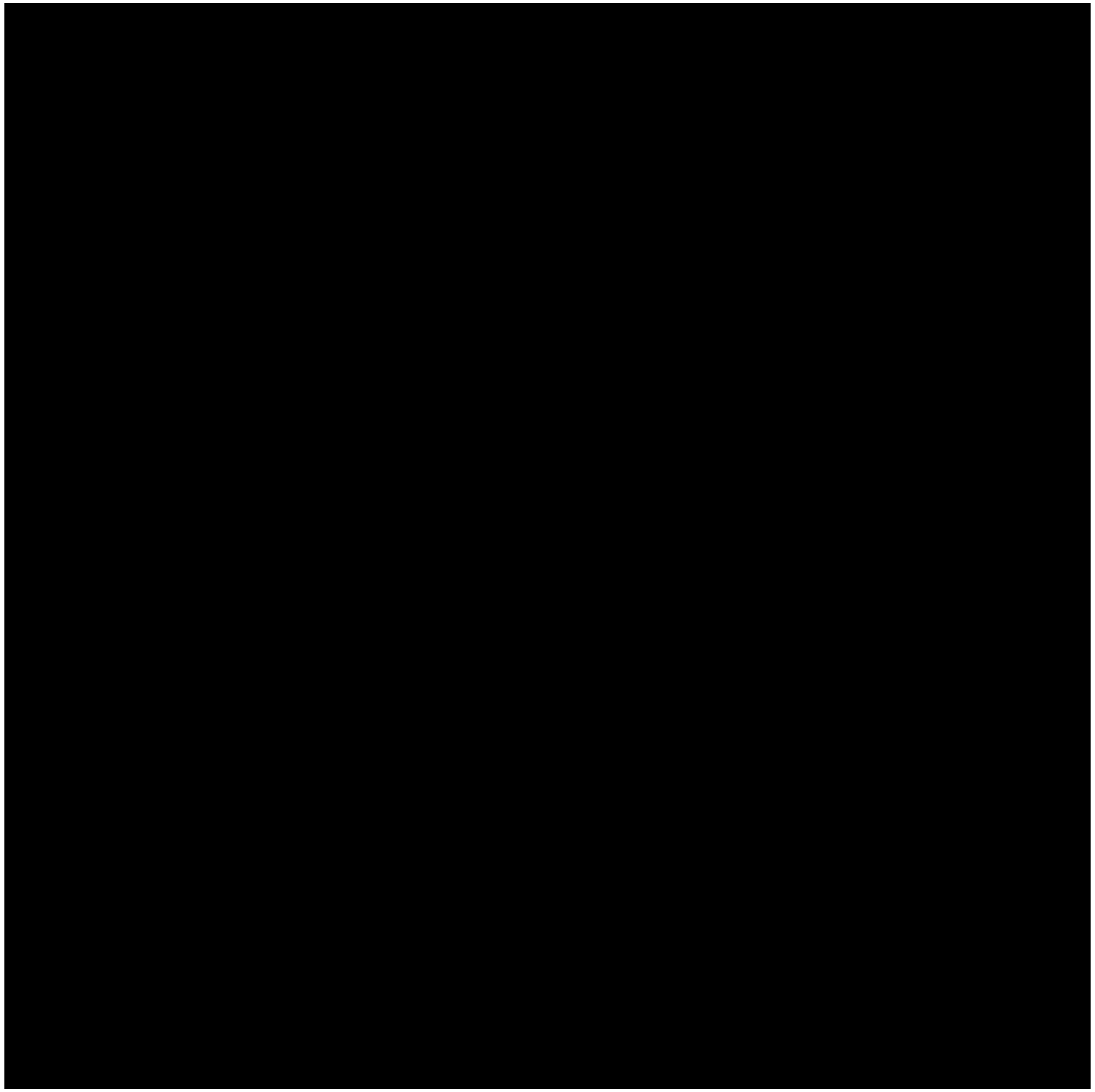


- Policy gradient
  - More refined methods:
    - Basic idea in on-policy optimization
      - Avoid taking bad actions that collapse the training performance.
    - TRPO
    - PPO

PPO

Deep Reinforcement Learning

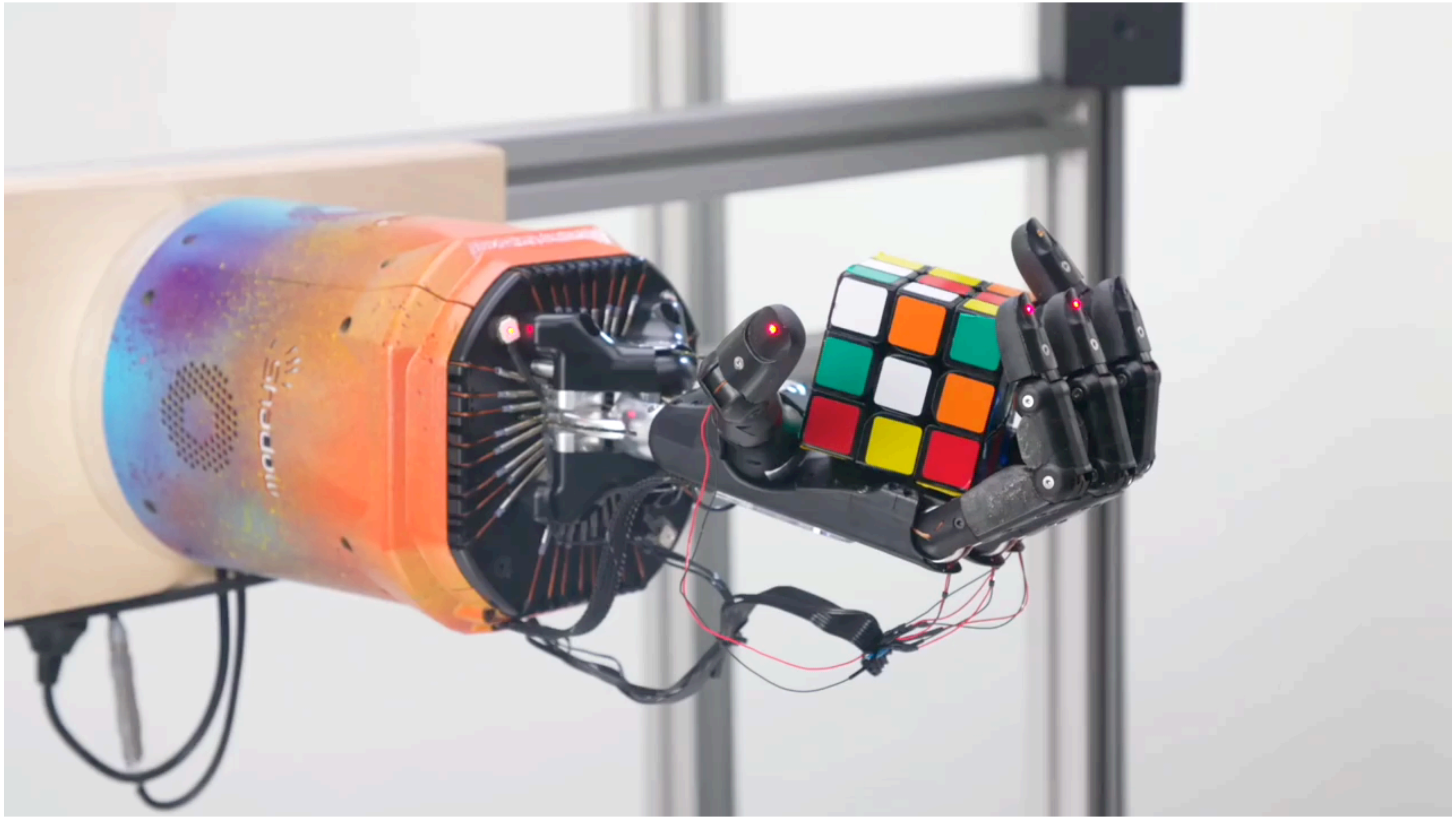
PPO



PPO

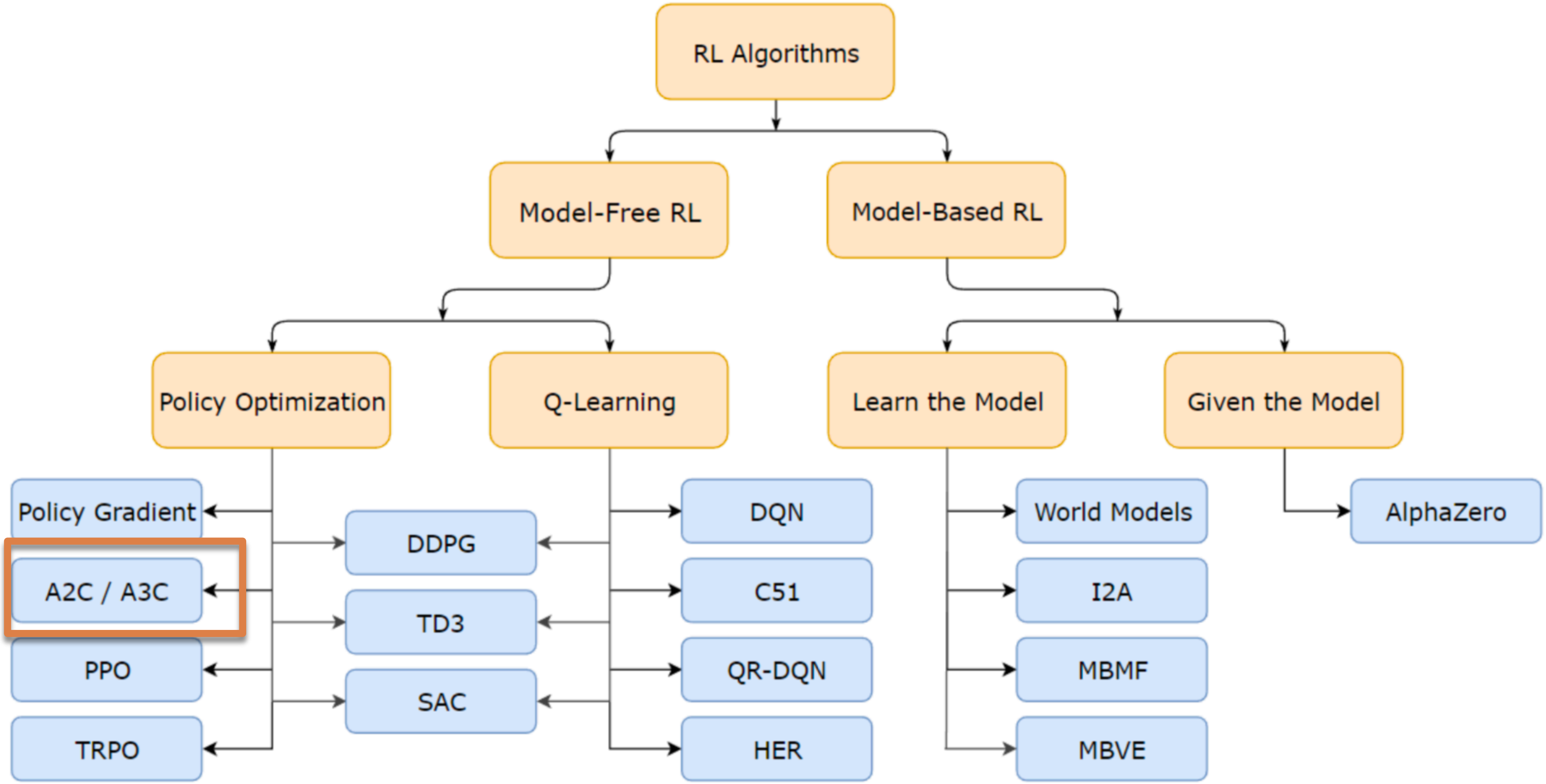
Deep Reinforcement Learning

- PPO



Deep Reinforcement Learning

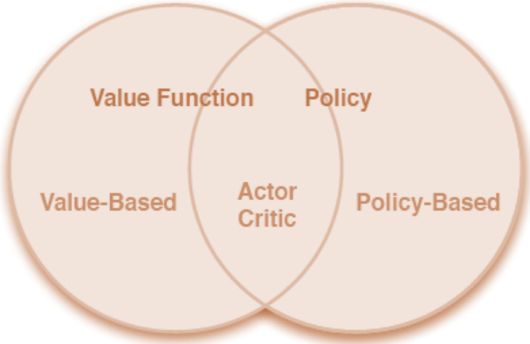
Taxonomy



Link: https://spinningup.openai.com

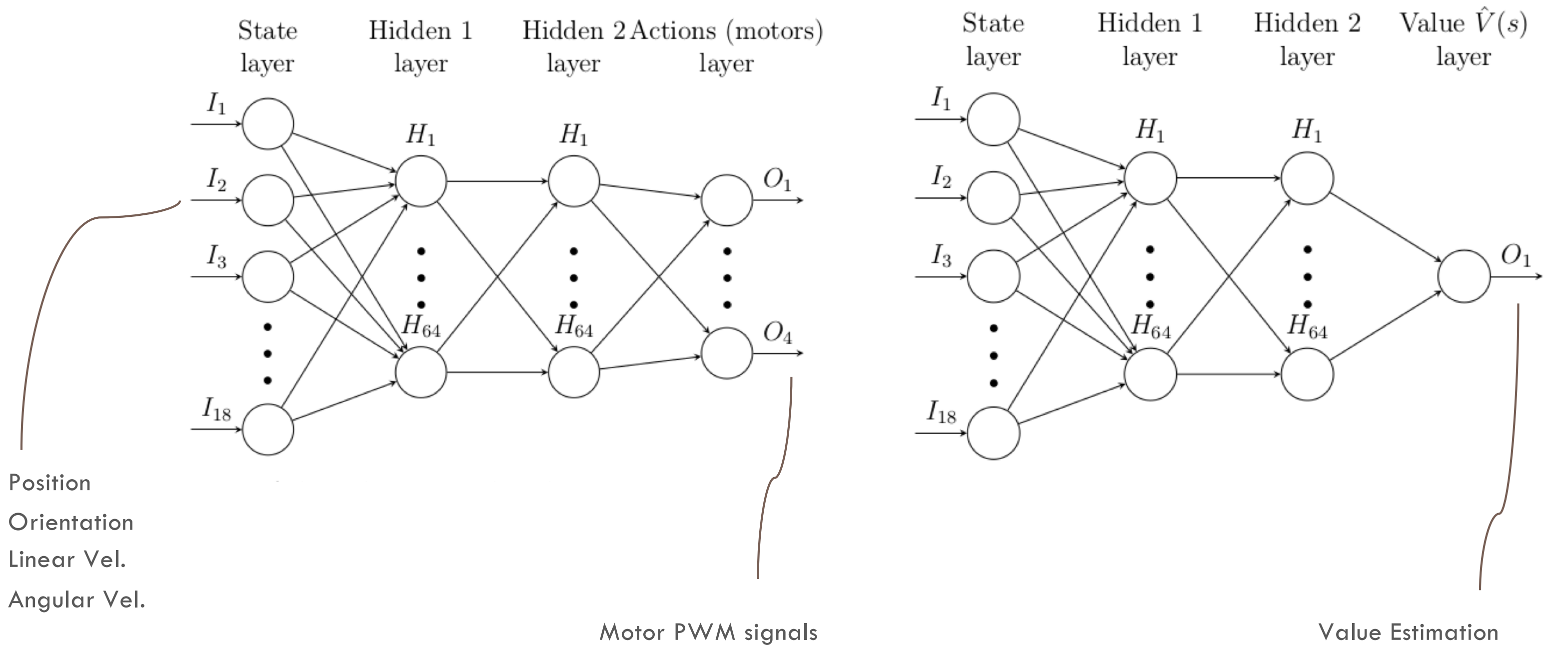
A2C

- Can we combine the best of policy-based and value-based?
- Yes. Advantage Actor-Critic (A2C)
- Combine DQN (value-based) and REINFC (policy-based)



  - Two neural networks (Actor and Critic):
    - **Actor** is policy-based: Samples the action from a policy
    - **Critic** is value-based: Measures how good the chosen action is
    - $\Delta\theta \quad = \alpha\nabla_\theta log\pi(S_t, A_t; \theta)\ \cancel{R_t}$ Traditional Policy Update
    - $\Delta\theta \quad = \alpha\nabla_\theta log\pi(S_t, A_t; \theta)\ \boxed{Q(S_t, A_t)}$ New Policy Update
  - Update at each time step - temporal difference (TD) learning

- Actor-critic PPO



- Actor-critic PPO

Reward Signal $$r_t(s) = r_{alive} - 1.2\,\|\epsilon_t(s)\|$$

Where $$\epsilon_t(s) = \sqrt{\xi^2_{target_{(t)}} - \xi^2_{quad_{(t)}}}$$

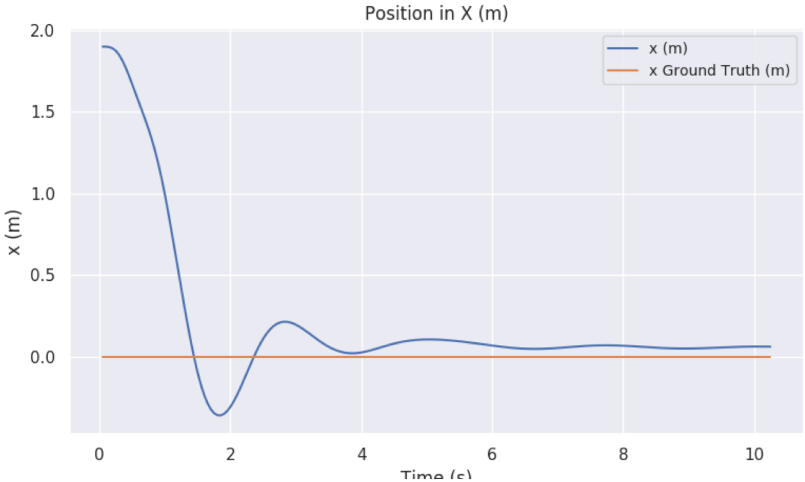
$$\epsilon_t(s) = \sqrt{(x_{target_{(t)}} - x_{quad_{(t)}})^2 + (y_{target_{(t)}} - y_{quad_{(t)}})^2 + (z_{target_{(t)}} - z_{quad_{(t)}})^2}$$

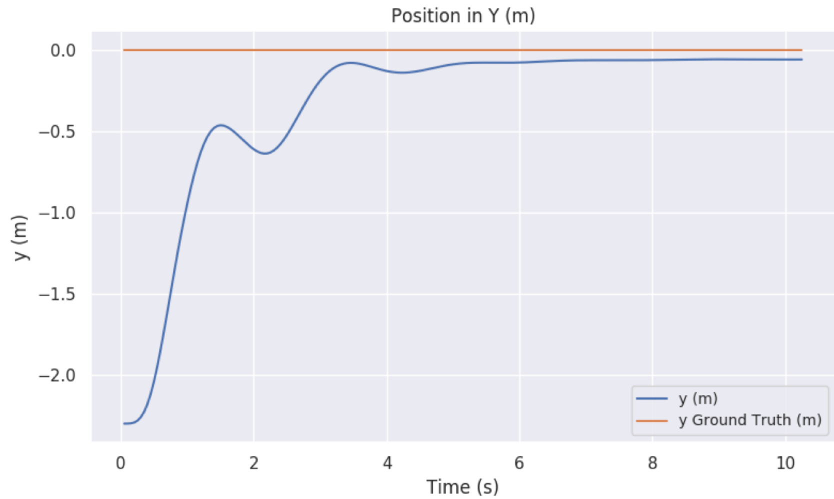
$$r_{alive} = 4.0$$

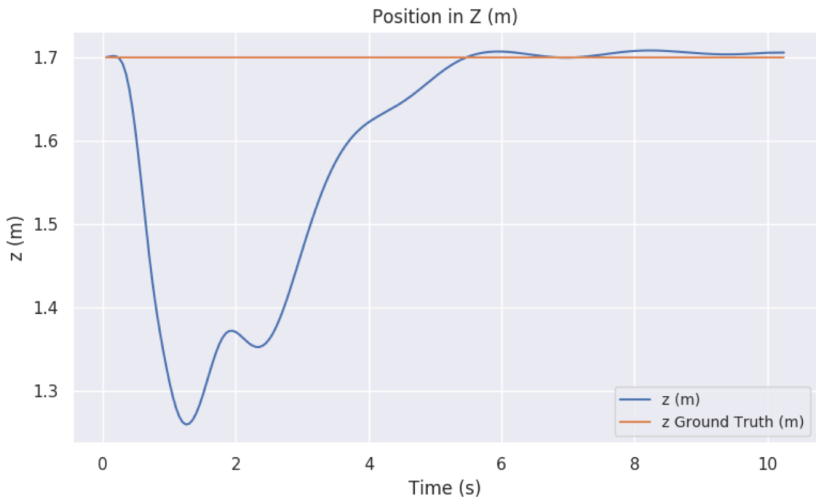
Quadrotor is bounded to a 3.2m radius sphere

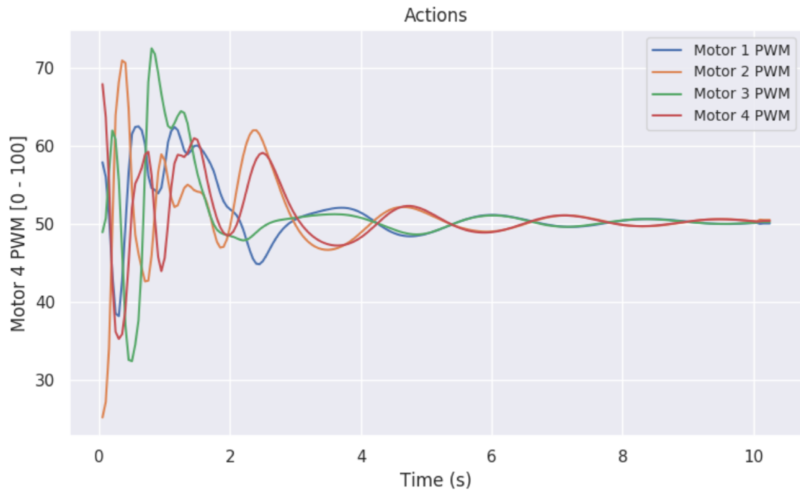
DRL for Quadrutor Control

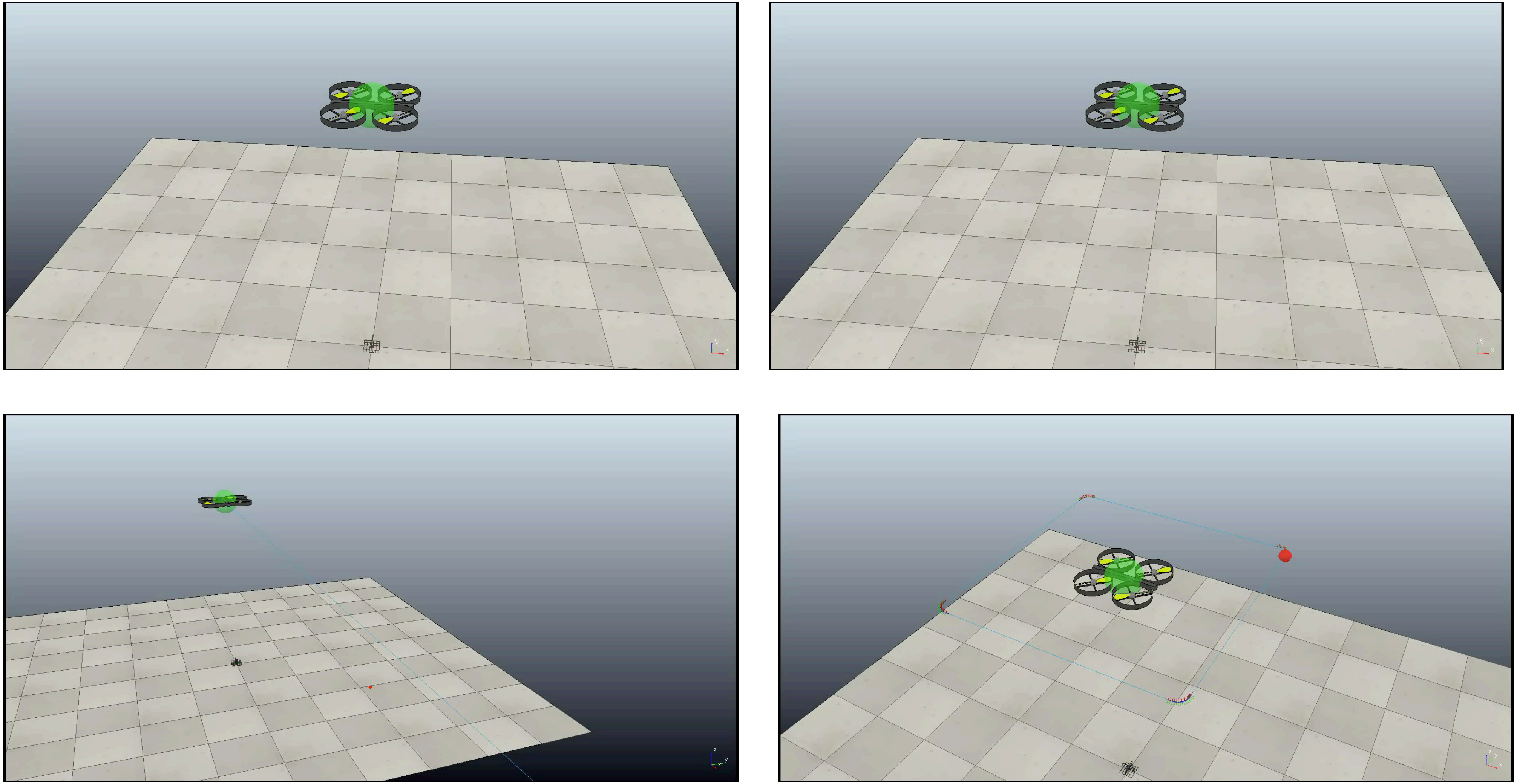
□ Actor-critic PPO



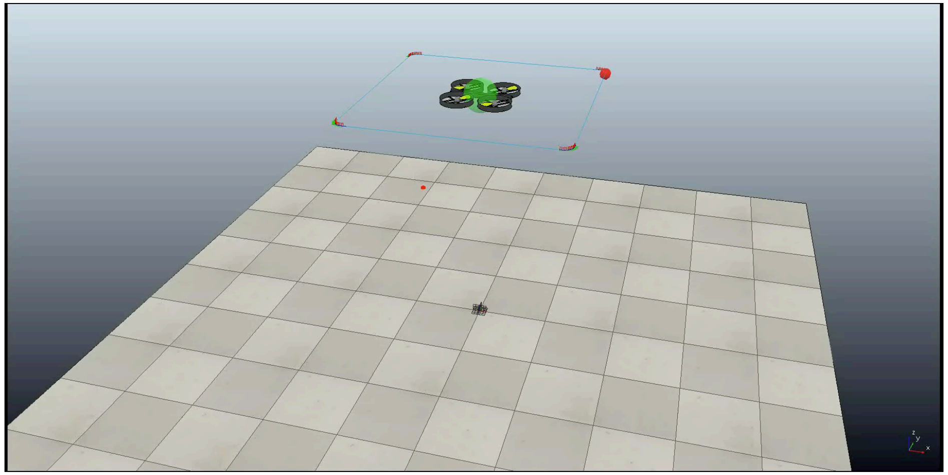


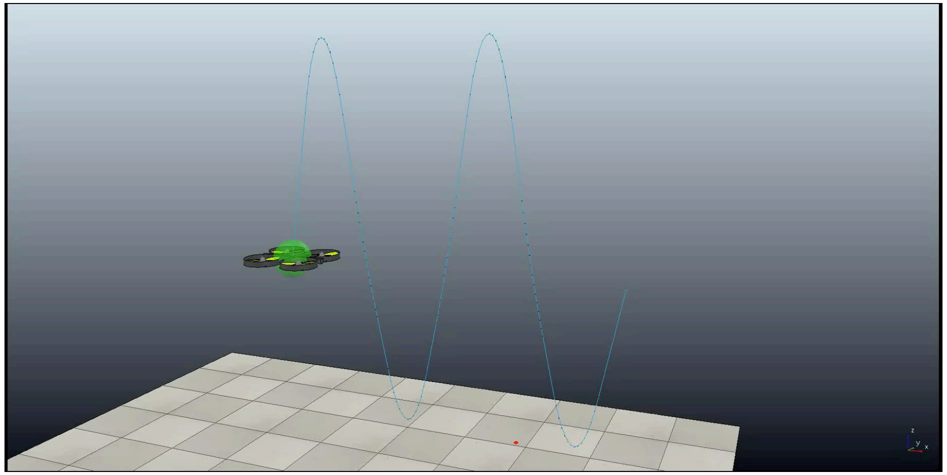


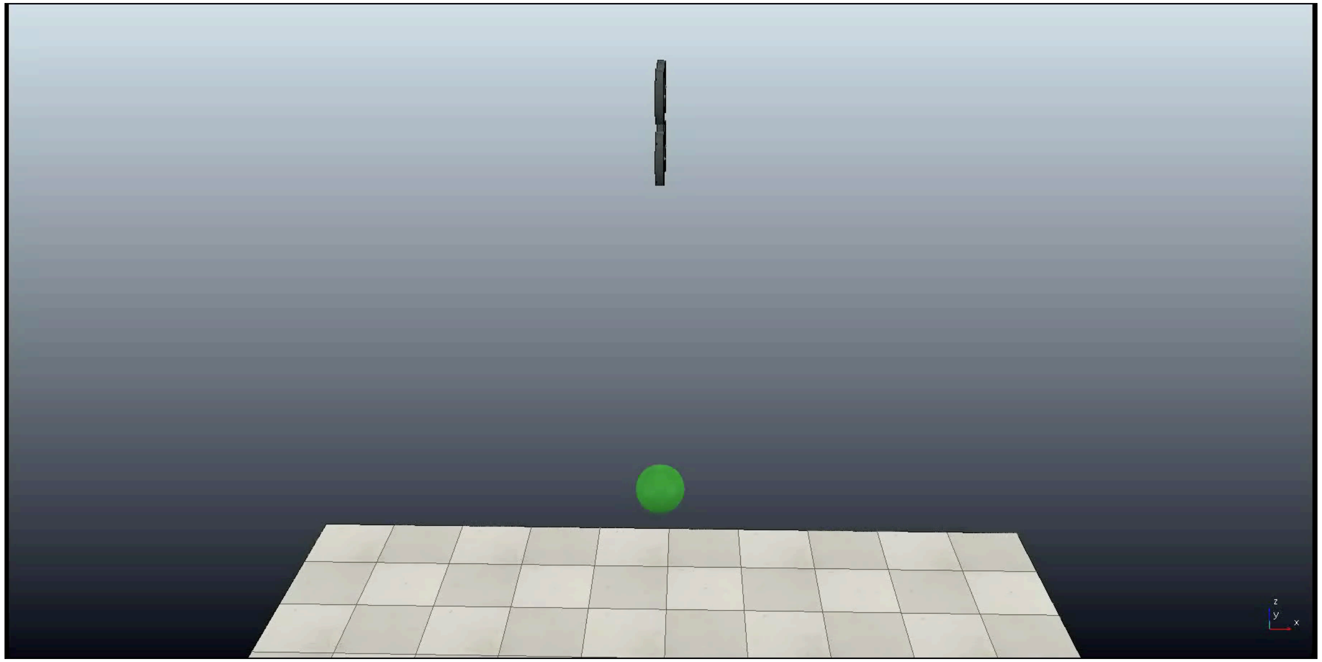




DRL for Quadrutor Control



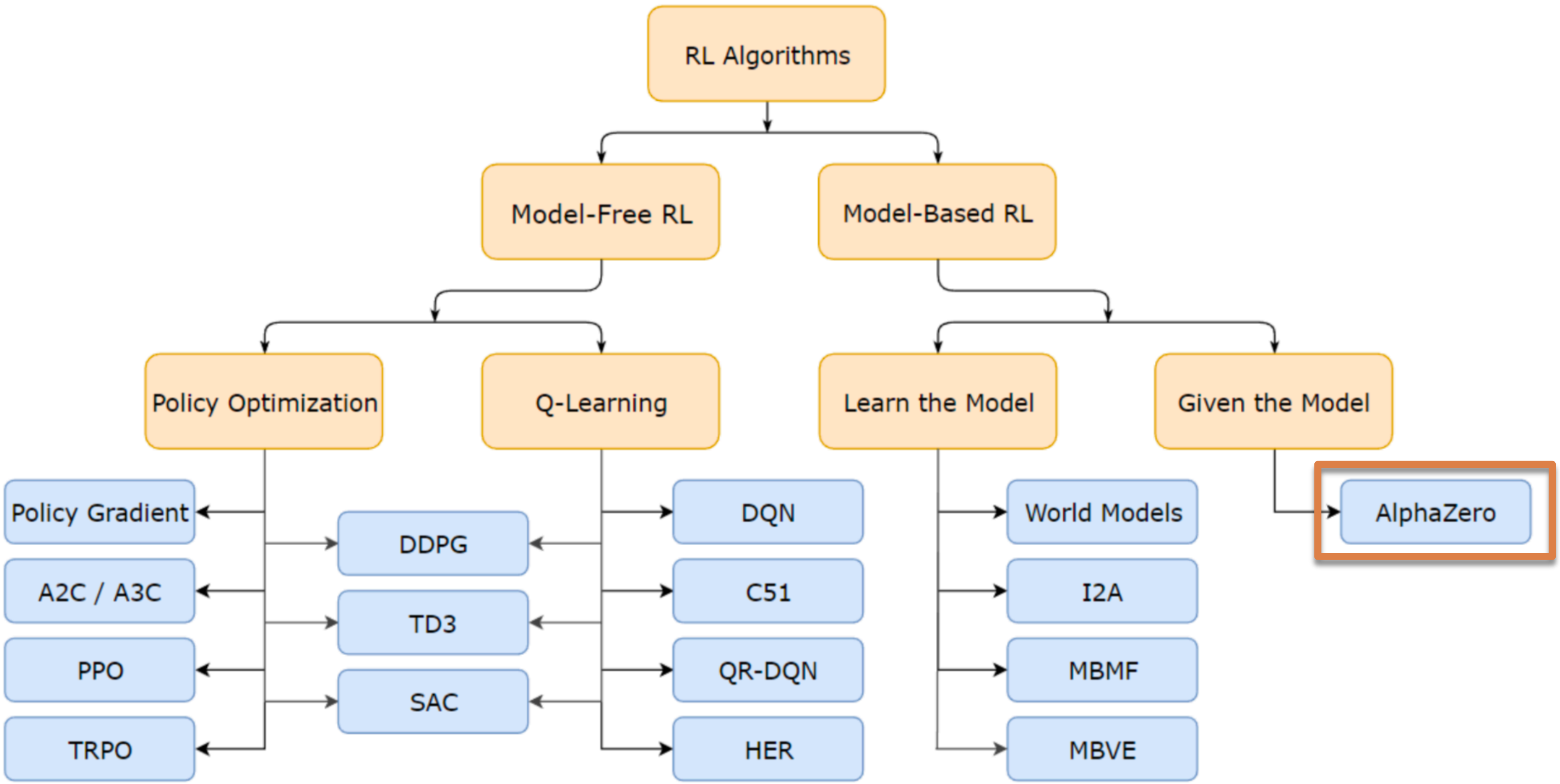




Deep Reinforcement Learning

Deep Reinforcement Learning

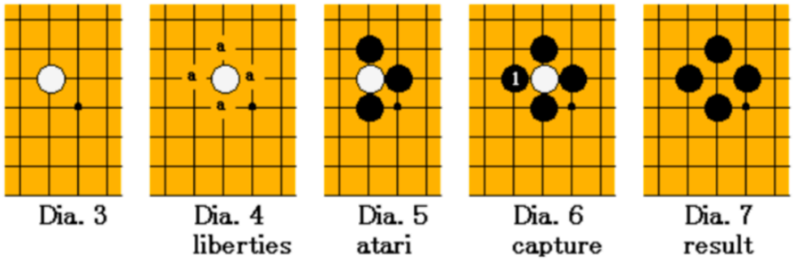
Taxonomy

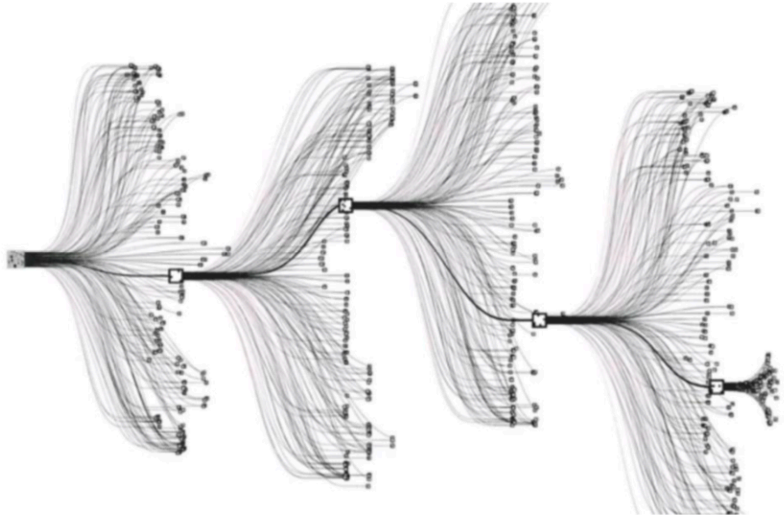


Link: https://spinningup.openai.com

Deep Reinforcement Learning: Model-based

The GO game

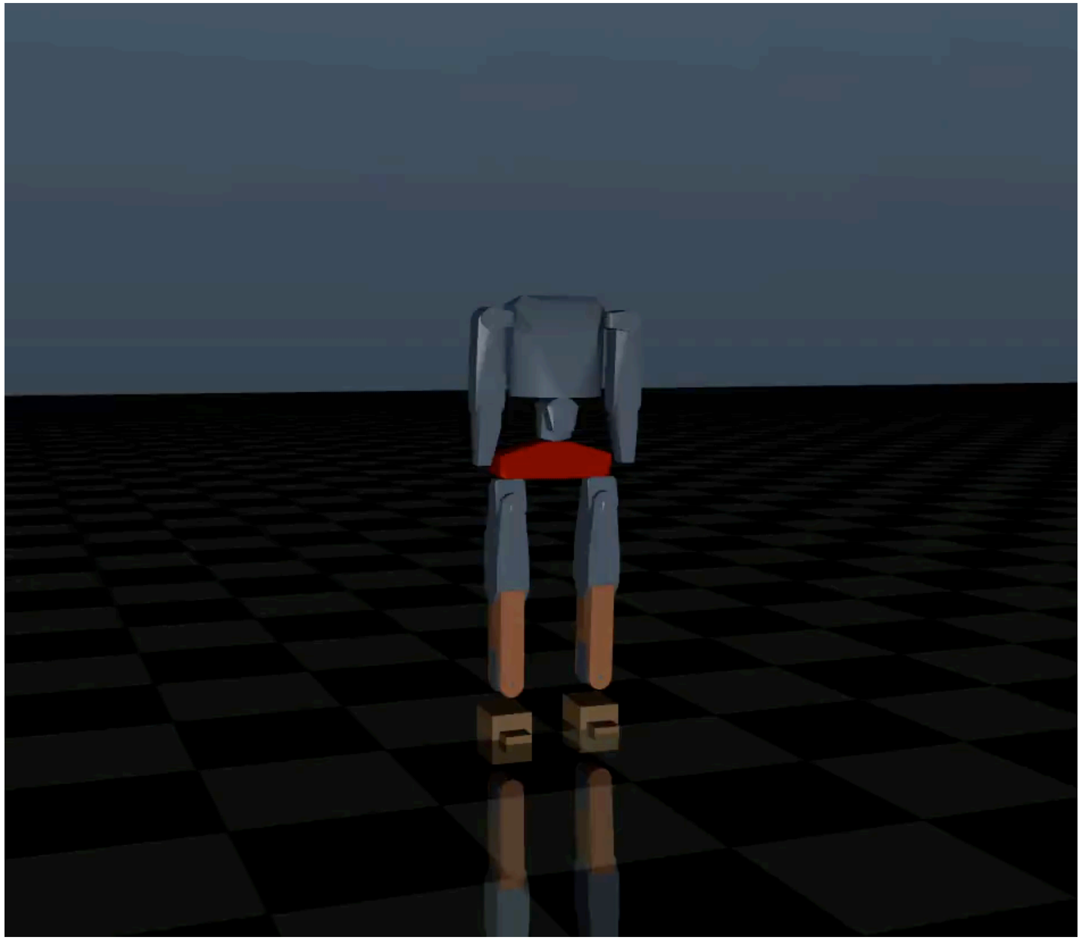




Game size	Board size N	3^N	Percent legal	legal game positions (A094777)[11]
1×1	1	3	33%	1
2×2	4	81	70%	57
3×3	9	19,683	64%	12,675
4×4	16	43,046,721	56%	24,318,165
5×5	25	8.47×10^{11}	49%	4.1×10^{11}
9×9	81	4.4×10^{38}	23.4%	1.039×10^{38}
13×13	169	4.3×10^{80}	8.66%	$3.72497923 \times 10^{79}$
19×19	361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

The GO game

- Video
  - Model-based Marta



DRL: transfer learning

- It does not always start from scratch
- Uses "low cost" simulation experience to learn real world skills
- Allows the agent to act effectively in an environment that has not seen before
  - TL: Using the experience of a set of tasks for faster learning and/or better performance in a new task

DRL: transfer learning

- A broad notion of "task":
  - varied objectives (reward)
  - robots (can affect state, action and dynamics)
  - varied environments (can affect observation space, dynamics, reward)
- Often, we will make assumptions about what will change between tasks

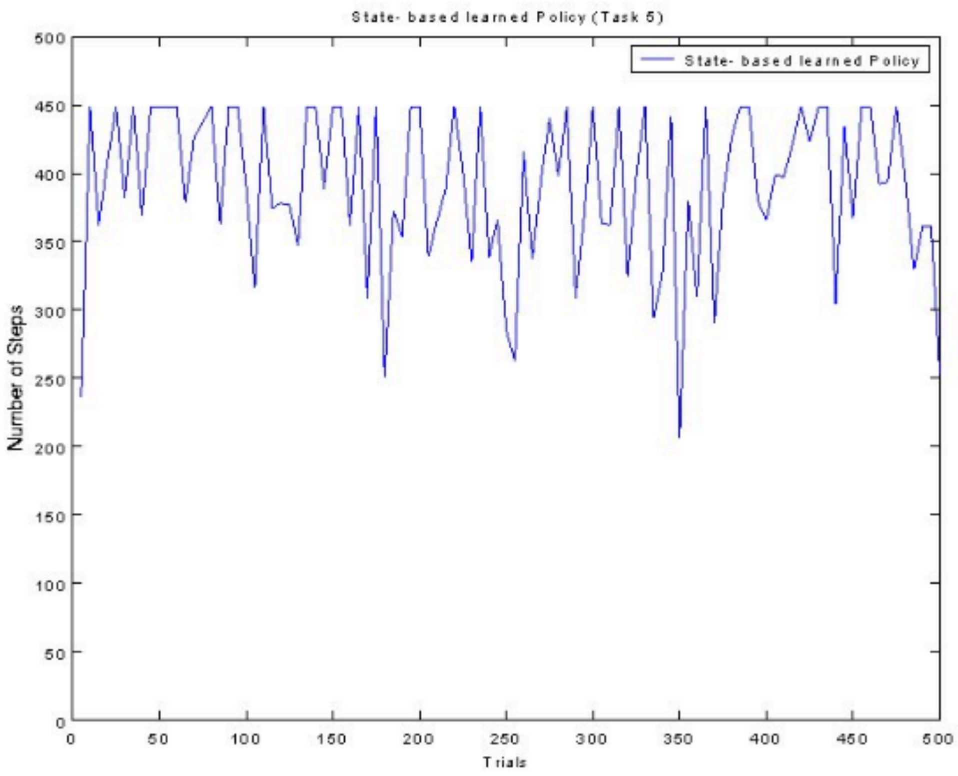
Deep Reinforcement Learning

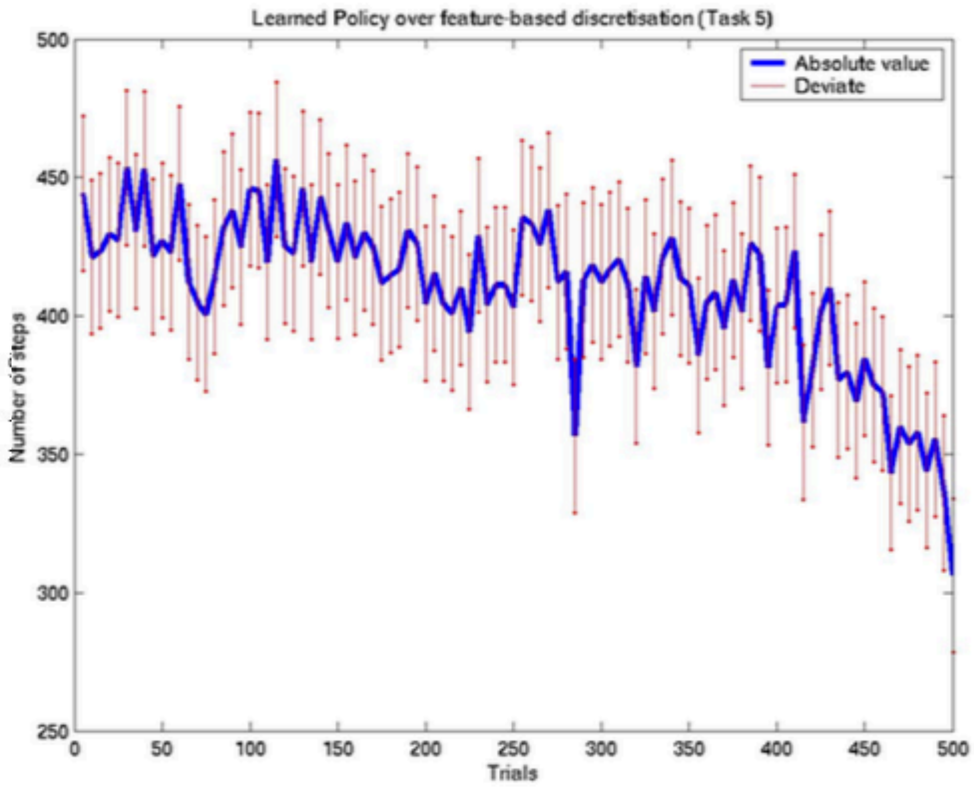
- Other approaches (with a kind of supervision)
  - Imitation Learning
    - Humans are able to do this early on
    - 8 months - mimics simple actions and expressions
    - 18 months - imitates delayed actions with multiple steps
    - 36 months - mimics actions with multiple steps
    - Imitation of the result of the action
    - Inferring intentions
  - Inverse RL
    - Behavior examples
    - Infer the reinforcement
    - Usually uses information from the expert, but, in the limit, could learn from a flawed system
    - Requires a similar body scheme
  - Prediction
    - There is a reference model

Exercise

- Model an MDP and an AR algorithm appropriate to the problem of a robot that has two IR sensors, which returns readings of {0,1} m and 4 Sonars, which returns readings of {0,5} m. The robot aims to walk as much as possible in an environment without hitting the walls. Possible actions are:
  - Walk forward
  - Walk backwards
  - Rotate 10°
  - Rotate -10°

- Non-Convergence x Convergence





Lecture 17

- Reading:
  - RUSSELL, S. NORVIG, P. Artificial Intelligence. 3a edição. Chapter 21.
  - BARTO, A., SUTTON, R. Reinforcement Learning: An Introduction. Second Edition. Freely Available at: https://drive.google.com/file/d/1xeUDVGWGUUv1-ccUMAZHJLej2C7aAFWY/view?usp=sharing

References

Lecture 17

- BARTO, A., SUTTON, R. Reinforcement Learning: An Introduction. Second Edition.
- MURPHY, R. R. Introduction to AI robotics. MIT Press, 2002.
- Lex Fridman, MIT Deep Learning Course, MIT, 2019.
- DUDEK, G.; JENKIN, M. Computational Principles of mobile robotics. Cambridge Press, 2000.
- ROMERO, R. A. F.; PRESTES, E.; OSÓRIO, F.; WOLF, D. (Orgs) Robótica móvel. LTC, 2014.
- BROOKS, R. Intelligence without representation. Artificial Intelligence, 47:139-159, 1991.
- RUSSEL, S. NORVIG, P. Artificial Intelligence: a modern approach. Prentice Hall, 2002.
- BRATKO, I. PROLOG: programming for artificial intelligence. Addison Wesley, 2nd edition, 1990.

This material is part of the Machine Learning Course

By Esther Colombini and Alexandre Simões