REINFORCEMENT LEARNING

……. AN INTRODUCTION

By Luke Ditria

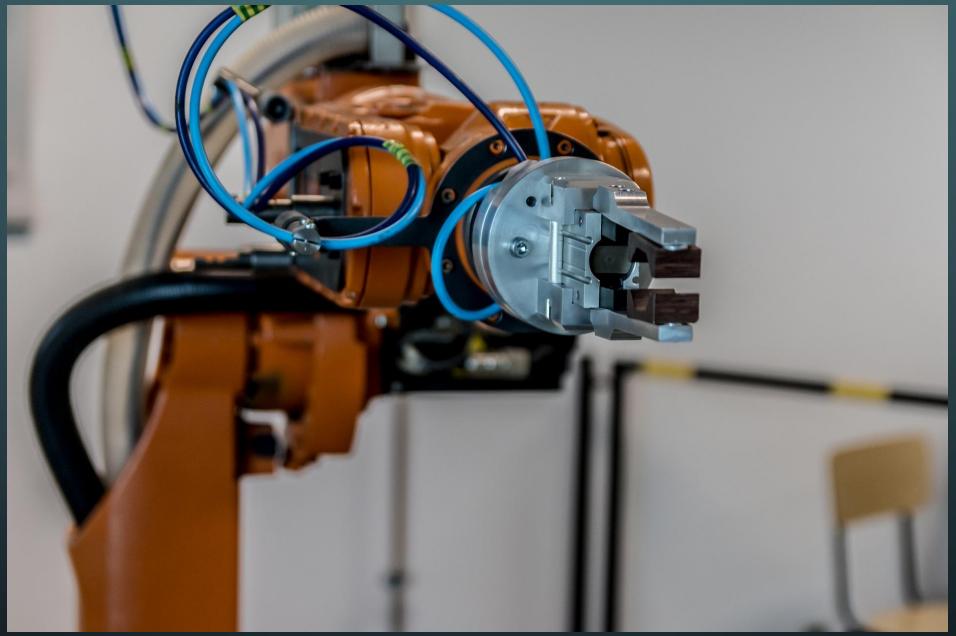
github.com/LukeDitria
youtube.com/@LukeDitria
in/lukeditria

BACKGROUND INFORMATION….

WHY RL?

- We want to use Machine Learning to come up with solutions to problems.
- To be able to solve our problems with ML we need to frame our problems in terms of some “cost” or “loss” that we want to minimize
- We need to know how to update our Machine Learning system in order to minimize the loss
    - What do we do if we don’t know how to update our system in order to minimize the cost?

EXAMPLE - PICK AND PLACE WITH A ROBOT ARM



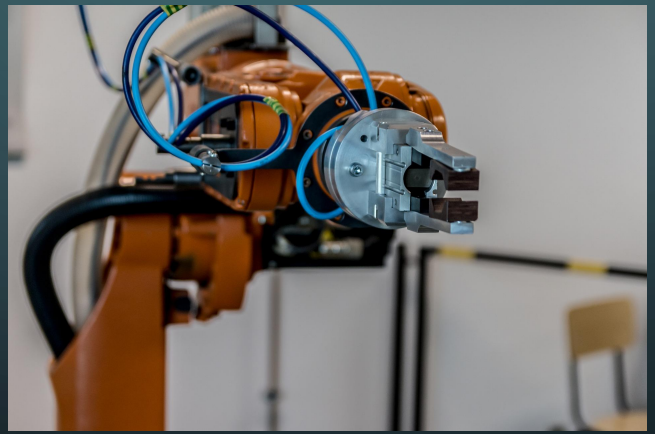
EXAMPLE - PICK AND PLACE WITH A ROBOT ARM

For simple situations we could use a camera and use CV or ML techniques to detect the object and using our model of the robot (forward kinematics), move the arm to the location of the object.

For more complicated situations we would need to take more things into consideration, eg other objects we need to avoid, moving target objects, distractions etc, the problem becomes harder to define!

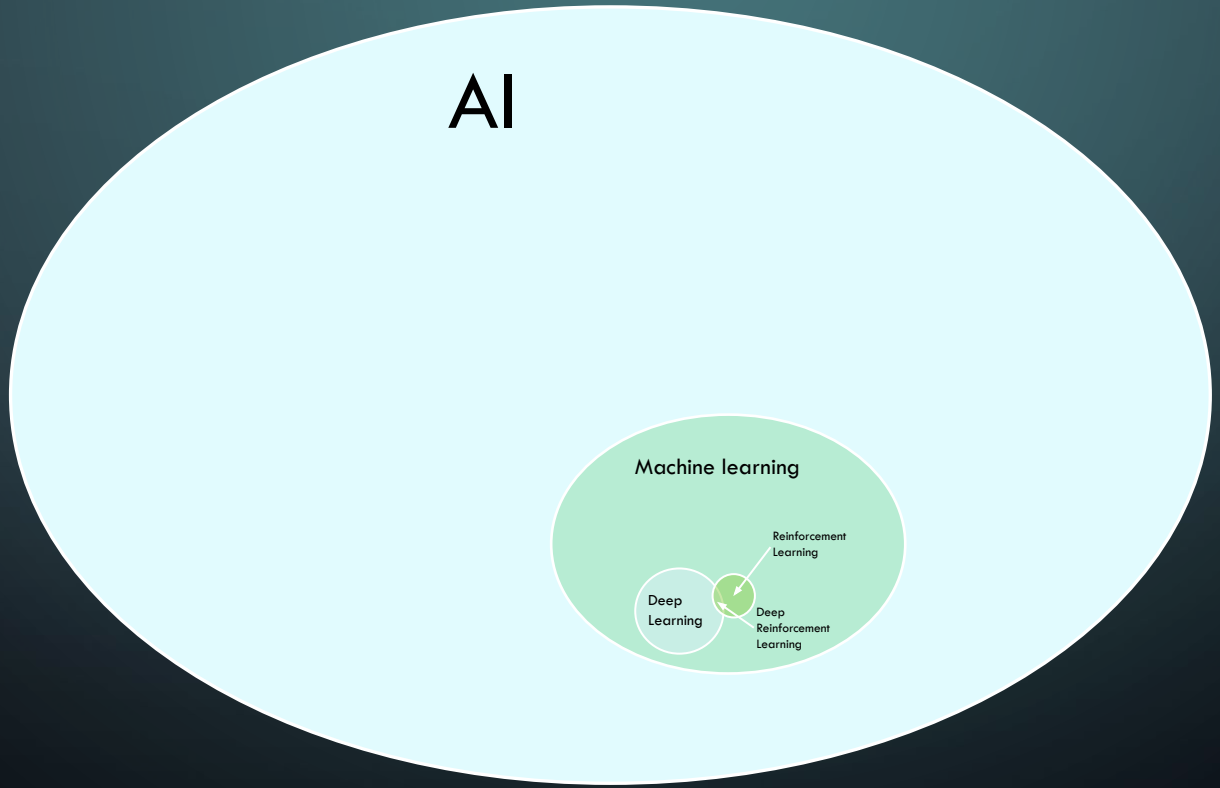
EXAMPLE - PICK AND PLACE WITH A ROBOT ARM

In situations like this we may want to use ML! But how do we frame this problem in terms of loss/cost and without knowing what actions we need to take how do we train our robot to act optimally in the environment?

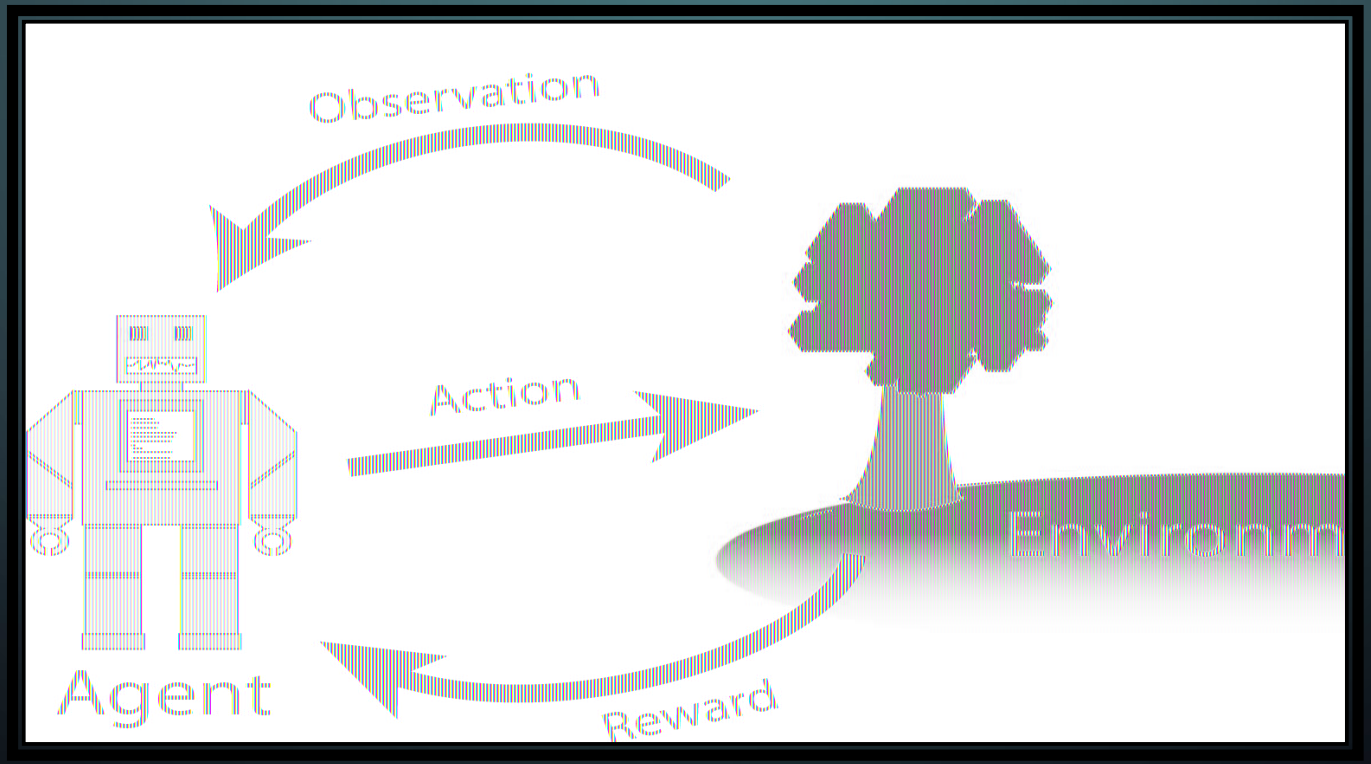


REINFORCEMENT LEARNING

WHERE ARE WE?


AI
Machine learning
Reinforcement Learning
Deep Learning
Deep Reinforcement Learning

THE REINFORCEMENT LEARNING PROBLEM



ASPECTS OF RL

- Environment
  - The medium in which we wish to interact with, could be…..
    - A video game
    - A simulation
    - The “real world”
    - A website
    - A virtual machine operating Ubuntu

ASPECTS OF RL

- Observation/State
  - Any information extracted from the environment at the current timestep, could be….
    - Camera frames
    - Frames from a computer game (or just game information like number of lives left etc)
    - Sensor information (temperature, light, wheel encoders, distance)
    - Audio
    - Text

ASPECTS OF RL

- Agent
  - The thing acting in the environment, could be….
    - A piece of software
    - A robot

ASPECTS OF RL

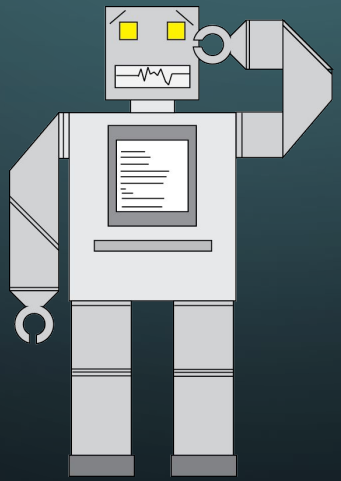
- Task/Goal
  - What we want the agent to do in the environment
    - Get the max score in a video game
    - Stack blocks with a robot arm as quickly as possible
    - Perform our online shopping

ASPECTS OF RL

- Reward
  - The means by which we encourage certain actions/behaviors in certain states
    - Positive reward encourage actions
    - Negative rewards discourage actions
  - It “defines” the task/goal
    - Very hard to come up with rewards that “align” the agents goals with out own…..

THE REINFORCEMENT LEARNING PROBLEM

How do we use observations to determine what actions we need to take in the environment in order to maximize the amount of reward?



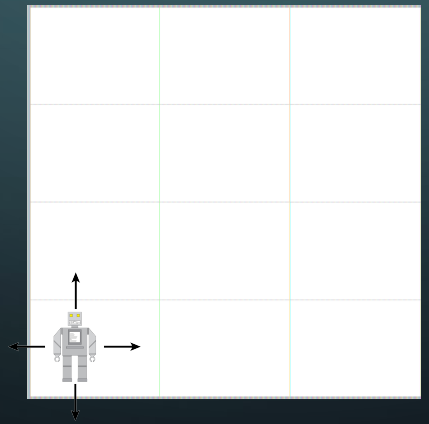
POLICIES AND VALUE

- Policy (denoted as pi - π)
  - How we decide what action to take
- Value
  - How “good” is our current situation

CHARACTERIZING OUR WORLD

We will start off with the simplest possible representation of an environment, a "Grid World"

Grid Worlds usually have a discrete set of states and actions with relatively simple state transitions

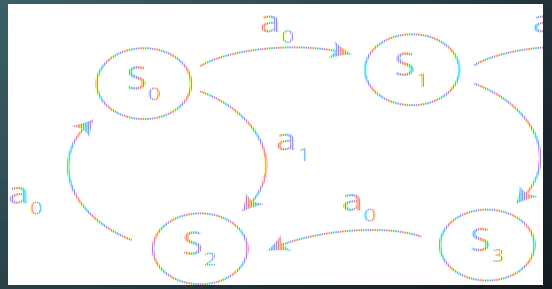


CHARACTERIZING OUR WORLD

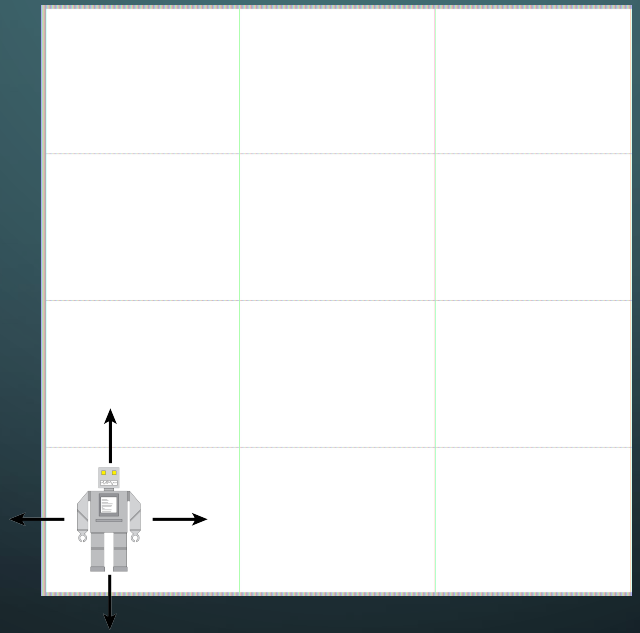
- A single interaction in our Grid World is call a “trajectory” or a “rollout”
  - The “trajectory” of our agent or the “rollout” of our policy
- There is a starting state
  - Either a constant location or randomized
- There are terminal states
  - States that signal the end of the current trajectory/rollout
    - Usually a reward state or “timeout”

TRAJECTORIES AS A MARKOV DECISION PROCESS (MDP)

- Framework for a basic decision making process, characterized by a 4-tuple (S, A, R, P)
  - S – Entire state space
  - A – Set of possible actions
  - R – Reward, either immediate or future
  - P – The probability an action will take you from one state to another



FIRST EXAMPLE, A DETERMINISTIC GRID WORLD

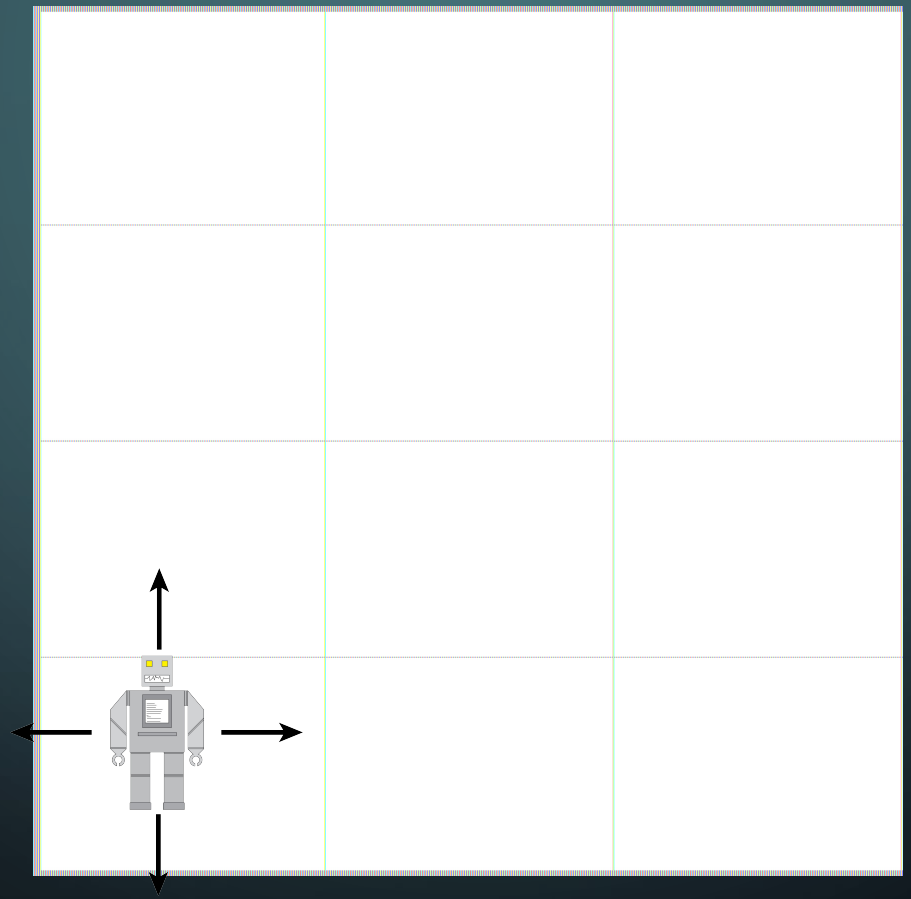


STATE TRANSITION TABLE/TRANSITION MATRIX

State/action	a_0	a_1	a_2	a_3
S_0	S_0	S_2	S_1	S_1
S_1	S_1	S_1	S_0	S_3
S_2	S_2	S_0	S_0	S_1
S_3	S_3	S_2	S_3	S_0

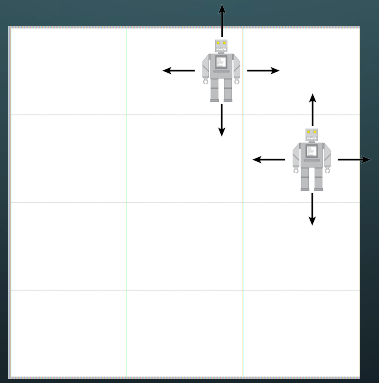
+

1

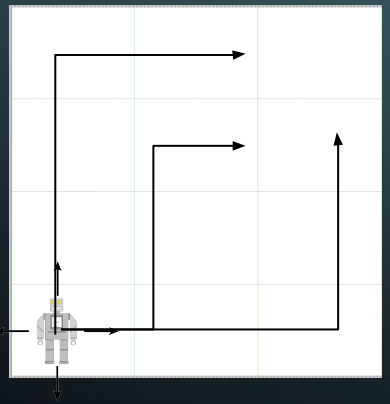


INTERACTING WITH OUR ENVIRONMENT

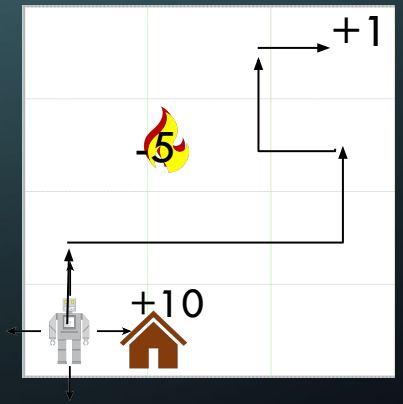
- We got a reward, so now what? Do we just take the same set of actions over and over again?
  - No! We want to ensure the trajectory we take is optimal



- Could we just calculate the distance from the start state to the reward state and find the minimum path?
  - No! We may have a "God-like" view of the environment, but our agent does not!
  - All our agent knows is that a certain set of actions got a certain reward, it knows nothing else about the task/environment!



Dangers or other rewards may be hiding from our agent!

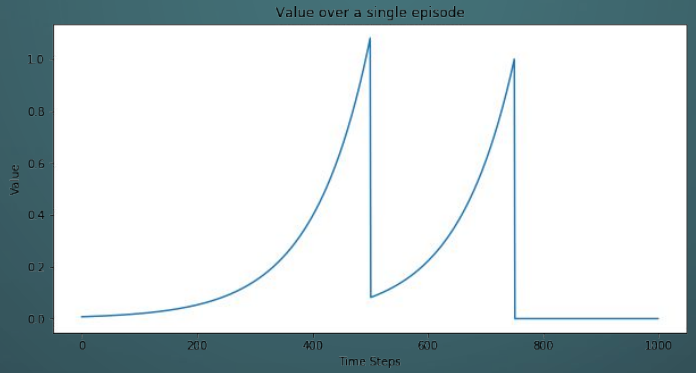


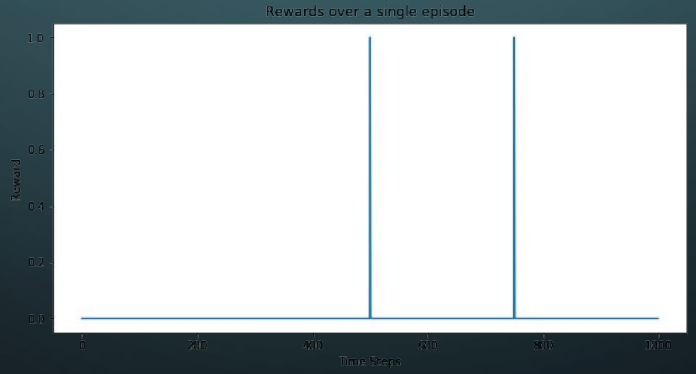
CUMULATIVE DISCOUNTED REWARDS

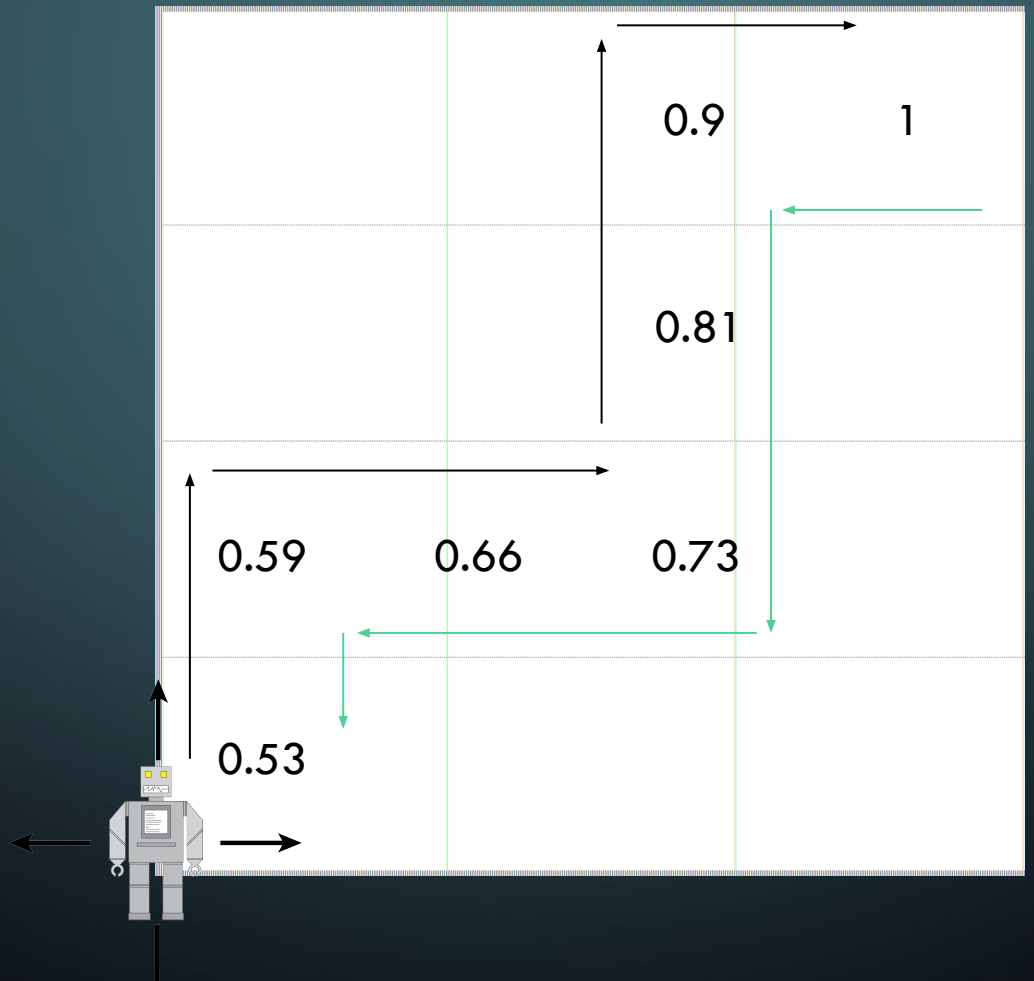
- We can propagate our reward backwards in time along the path that we took
- Also called the “Returns”
  - Rewards “returned” from the rollout
- Discounted with discount rate “gamma”
  - Gamma = 1, no discounting!
  - Gamma = 0, “returns” only in reward state

$$R_t = \sum_{l=0}^{T} \gamma^l r_{t+l}$$

REWARD DECAY

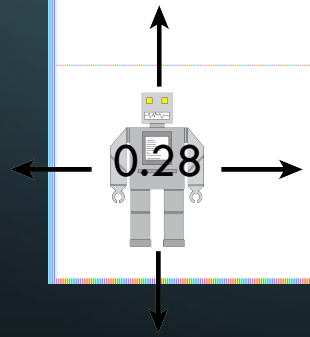






$\gamma = 0.9$

0.39	0.43	0.9	
0.35	0.48	0.81	
0.31	0.53	0.73	
0.28	0.59	0.66	



+

1

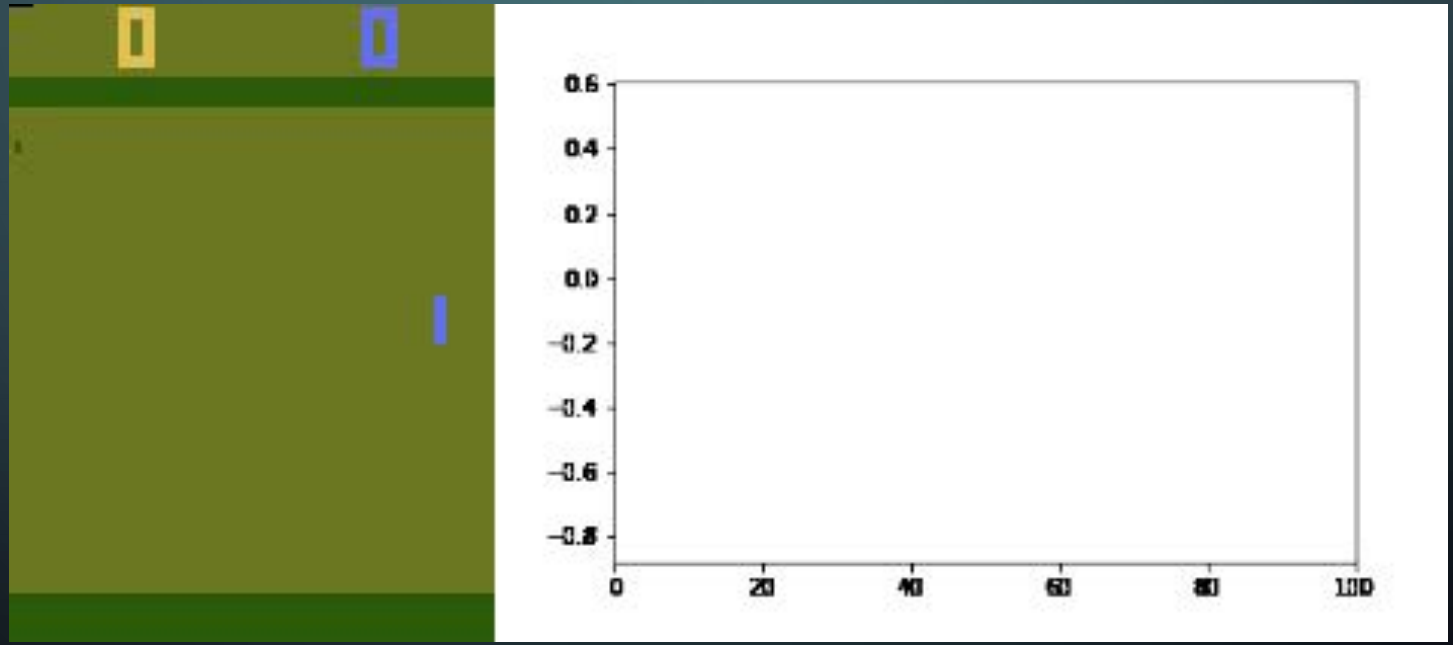
$\gamma = 0.9$

STATE VALUE

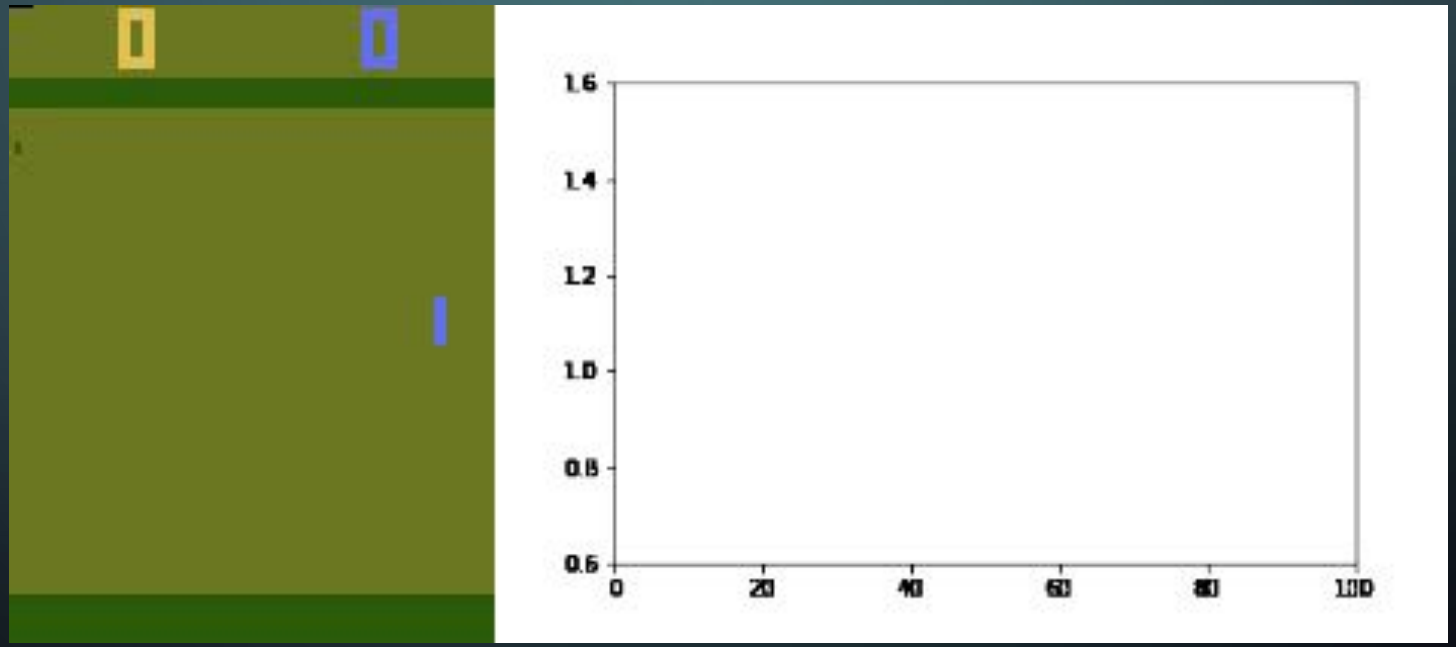
- Can't just use the Returns of their own to determine how good a State is as it depends on the actions taken after you leave that State
- So we can average over MANY different trajectories and get the Expectation (mean) of the Returns, which we call the Value
- Note: The Value of a given State is dependent on the Policy!

$$V_\pi(s_t) = \mathbb{E}_\pi[\sum_{l=0}^{T} \gamma^l r_{t+l}]$$

VALUE FUNCTION FOR A “BEGINNER”



VALUE FUNCTION FOR A “PRO”

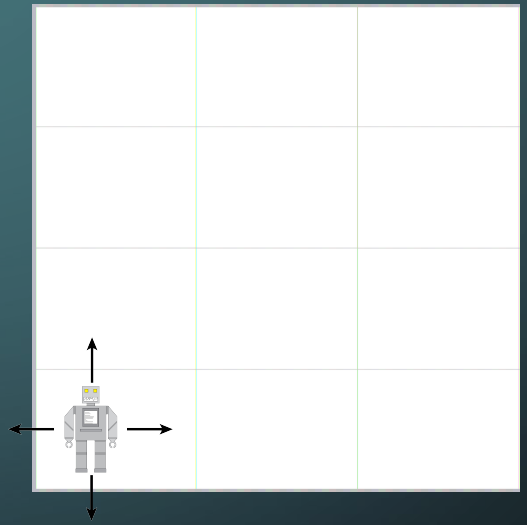


VALUE LOOK-UP TABLE

- We can store the Value of each state in a “Look-Up” Table

	S_0	S_1	S_2	...
$V(s)$	$V(S_0)$	$V(S_1)$	$V(S_2)$	

CODE EXAMPLE



VALUE CALCULATION PROBLEMS

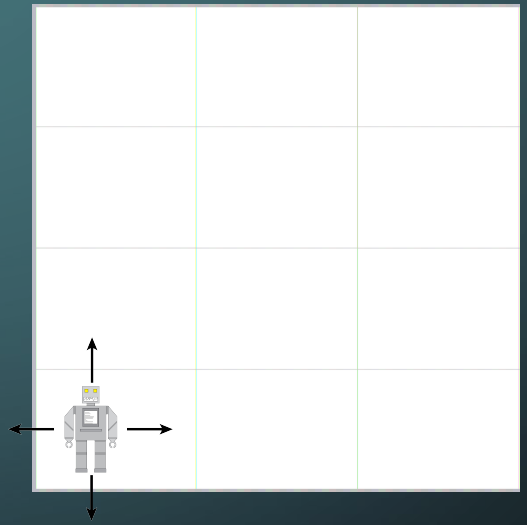
- Simply averaging the Returns over many rollouts does not work very well, as the Rewards in our environment are “sparce” and how long it takes to reach a reward state (if at all) can vary a lot
  - This introduces a lot of noise into our Value
- Having to keep track of the returns for every rollout is also wasteful
  - Can’t we just update our Value Look-Up table as we go?

MONTE CARLO UPDATE

- Calculates the difference between the current trajectories Returns and the Values
- Updates the Value function by adding a small amount of difference to the Value
  - Update rate – alpha (similar to a learning rate) determines how much of the difference we use
- This is an example of "Value iteration"
  - We will converge to the "real" Value after an "infinite" number of rollouts

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$$

CODE EXAMPLE



MONTE CARLO UPDATE PROBLEMS

- MC updates can still be quite noisy if the variance of the Returns is high
  - This might mean we need to use a small alpha, convergence will therefore be slow
- It only works if the trajectory is of finite length

TEMPORAL-DIFFERENCE (TD)

- Temporal-Difference updates help negate the effects of a “noisy” policy
- We do not calculate the Returns and instead only use the current value function and the reward
  - We estimate the returns with the value function!
  - No need for Returns means “infinite” trajectories are possible

$$V(s_t) \leftarrow V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)]$$

UNPACKING THE TEMPORAL-DIFFERENCE (TD) UPDATE

The Returns for the current State can be written as:

$$R(s_t) = r_t + \gamma R(s_{t+1})$$

We can Estimate the Returns at the next time-step with the average Returns at that time-step, aka the Value:

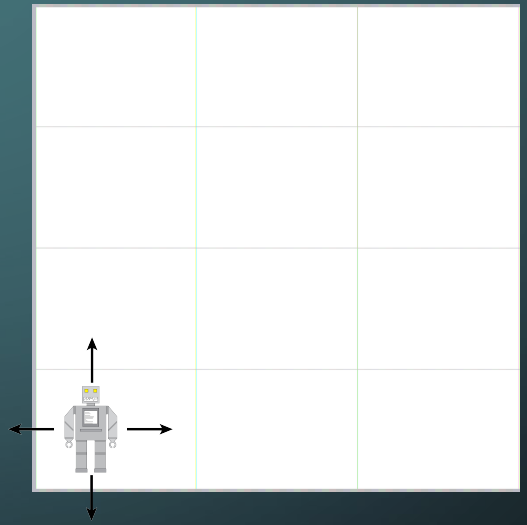
$$R(s_t) = r_t + \gamma V(s_{t+1})$$

Plug this into the Monte Carlo Update = TD update:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \quad \text{-MC}$$

$$V(s_t) \leftarrow V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)] \quad \text{-TD}$$

CODE EXAMPLE



EXPLORATION VS EXPLOITATION

- Up until now our policy during training has been random, this allows us to explore the environment by perform as many different trajectories as possible!
- But this does not seem very optimal…. Why can't we use our Value function during training and exploit the knowledge we already have?
- If we ONLY use our Value function we will only ever take the same steps.
- So why don't we combine random exploration AND exploit he knowledge we already have!

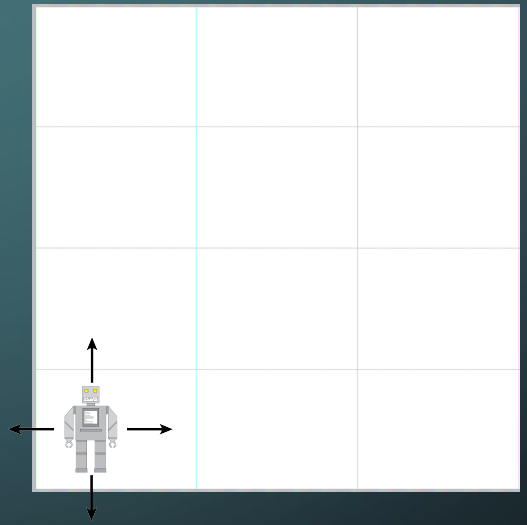
EPSILON GREEDY

- Lets use our Value Loop-Up table during training, but only sometime!
- Define a variable called “epsilon” (between 0 and 1)
  - This defines the probability of taking a random action
- We either take a random action or use our Value to determine what action to take!

$$Z \sim Uniform(0,1) \qquad \varepsilon \in (0,1)$$

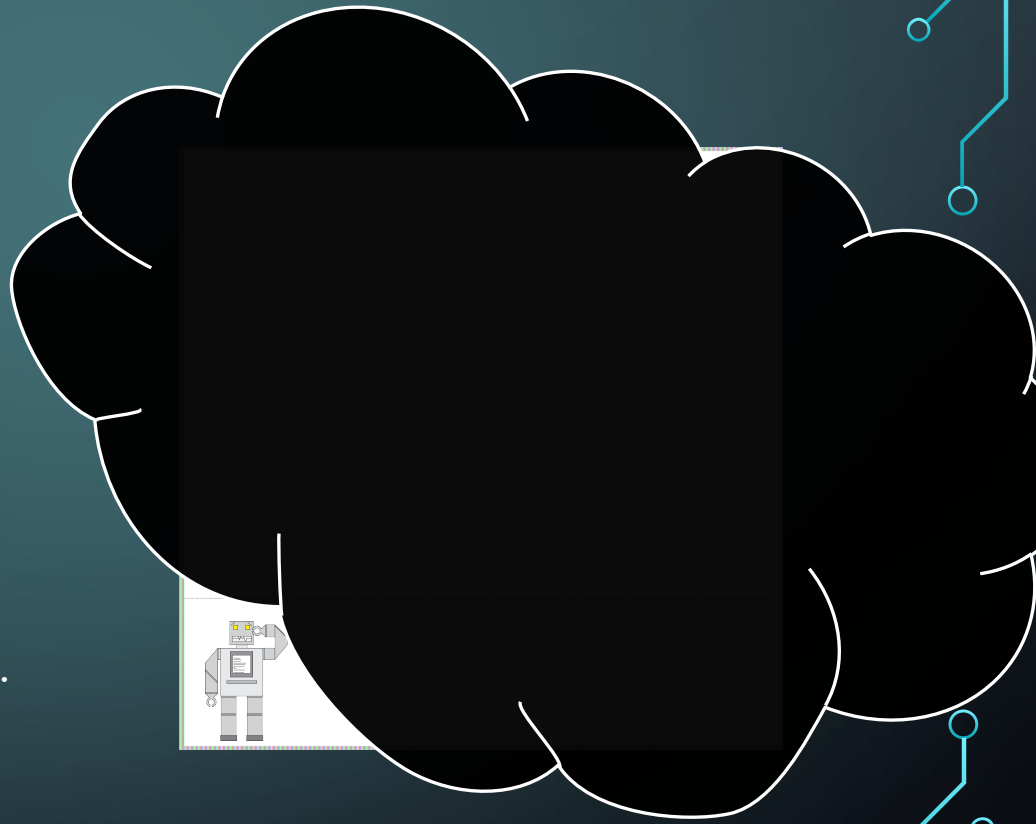
$$\pi(s_t) = \begin{cases} argmax_a V(s_{t+1}) & z \geq \varepsilon \\ random(a) & z < \varepsilon \end{cases}$$

CODE EXAMPLES



MODEL FREE LEARNING

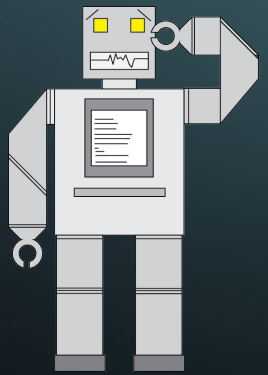
WE'VE BEEN MAKING A BIG ASSUMPTION……

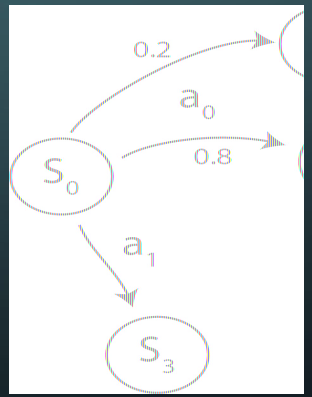


STOCHASTIC ENVIRONMENTS AND MODEL FREE LEANING

- Up until now we have not only assumed that the "P" in our MDP was always 1.0 and we have assumed that we KNOW what State (or States) a given action can take us into from where we are.

- That is, we not only assumed that a given action will ALWAYS take us into a given state, but that we also KNEW exactly what states we COULD get into from our current state.





Q-LEARNING

What we need is a “Value” measure that also takes into consideration what action we plan on taking. We replace the “Value” of a state with the “Quality” of each action, for a given state. Therefore each action in a given state is given it own Value, this means we are no longer dependant on a model of the environment in each state we simple take the action with the highest value (known as the Q value).

Q-LEARNING

We can learn this “Q-value” using TD learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \times maxQ(s_{t+1}, a) - Q(s_t, a_t)]$$

We update the current Q-value by assuming we will take the action with the highest Q-value at the next timestep. The Q-value can be thought of as the value of an action in a state, assuming that after taking that action we will always behave optimally (according to our current Q-values). Therefore the Q-value tries to represent the action-value of a completely optimal agent.

Q-LEARNING

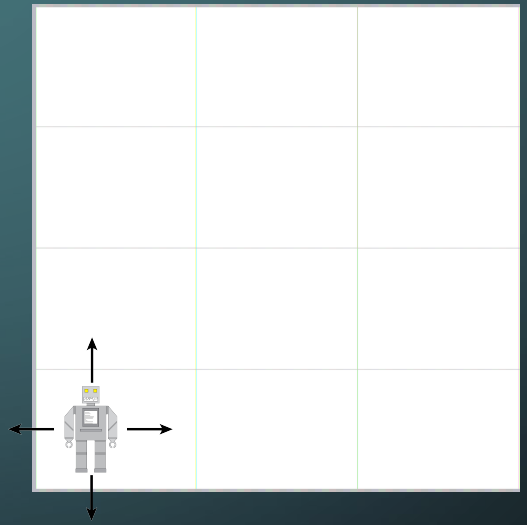
$Q(a, s)$	a_0	a_1	a_2	...
s_0				
s_1				
s_2				
s_3				
s_4				
⋮				
s_m				

EPSILON GREEDY

$$Z \sim Uniform(0,1) \qquad \varepsilon \in (0,1)$$

$$\pi(s_t) = \begin{cases} argmax_a Q(s_t) & z \geq \varepsilon \\ random(a) & z < \varepsilon \end{cases}$$

CODE EXAMPLE



OTHER RL METHODS…

Q-Learning is an example of a “Value based” RL method, that is, we try to learn the associated Value of either a State or the Value of an Action in a State. We use a mix of random actions and greedy actions during training. The “greedy” actions come from our interpretation of the Value of Actions/States. The algorithm itself does not directly give us an action to take, the “Policy” is determined by us using the learned Values.

Can we do something else?

POLICY BASED LEARNING

Policy based methods try to directly learn the policy, at every State we want to be given the action to take.

Policy based methods work by learning a probability distribution over possible actions, sampling an action from this distribution and then either increasing or decreasing the probability of taking that action, depending on performance.

$$P_{log}(a_t, s_t) \leftarrow P_{log}(a_t, s_t) + \alpha A_t$$

ACTION PROBABILITY TABLE

Every State has it's own independent distribution over the actions.

For a discrete set of actions, every action for a given State is given a probability based on how good it is to take that action, all the action probabilities for a given state should add up to 1.

We usually work with log probabilities or logits (inverse of a sigmoid) as they are not bound between 0 and 1

$P(a, s)$	a_0	a_1	a_2	.....	
s_0	$P(a_0, s_0)$	$P(a_1, s_0)$	$P(a_2, s_0)$		P
s_1					
s_2					
s_3					
s_4					
⋮					
s_m					

POLICY BASED LEARNING – GRID WORLD UPDATE

Here we see the most basic update for a Grid World setting.

We update the current log probability with the "Advantage" A_t, where A_t is either a positive or negative value, either increasing or decreasing the probability of taking that action in the given State.

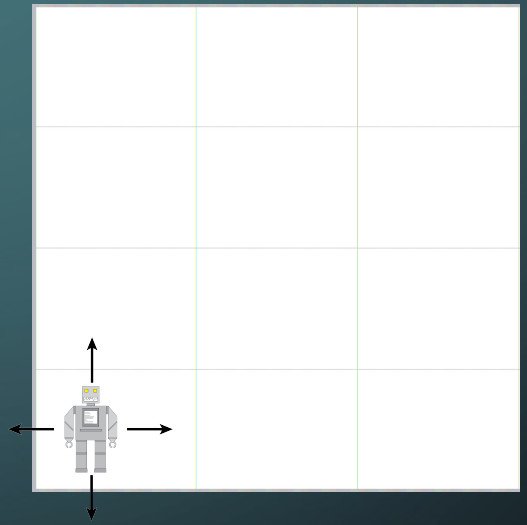
$$P_{log}(a_t, s_t) \leftarrow P_{log}(a_t, s_t) + \alpha A_t$$

REINFORCE – BASIC POLICY BASED METHOD

- In REINFORCE, the value of A_t is just the Returns, for always positive rewards:
  - If we took an action in a given State and the Returns were low, we increase the probability of taking that action a little bit
  - If we took an action in a given State and the Returns were high, we increase the probability of taking that action a lot
  - If there the Returns are 0 we do not change the probability

$$P_{log}(a_t, s_t) \leftarrow P_{log}(a_t, s_t) + \alpha R_t$$

CODE EXAMPLE



ACTOR-CRITIC, MIXING VALUE AND POLICY

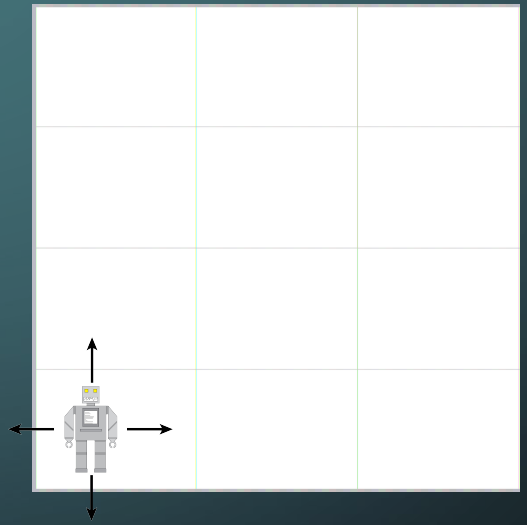
With Actor-Critic methods we also learn a Value for each State and the Advantage becomes:

$$A_t = R_t - V_t$$

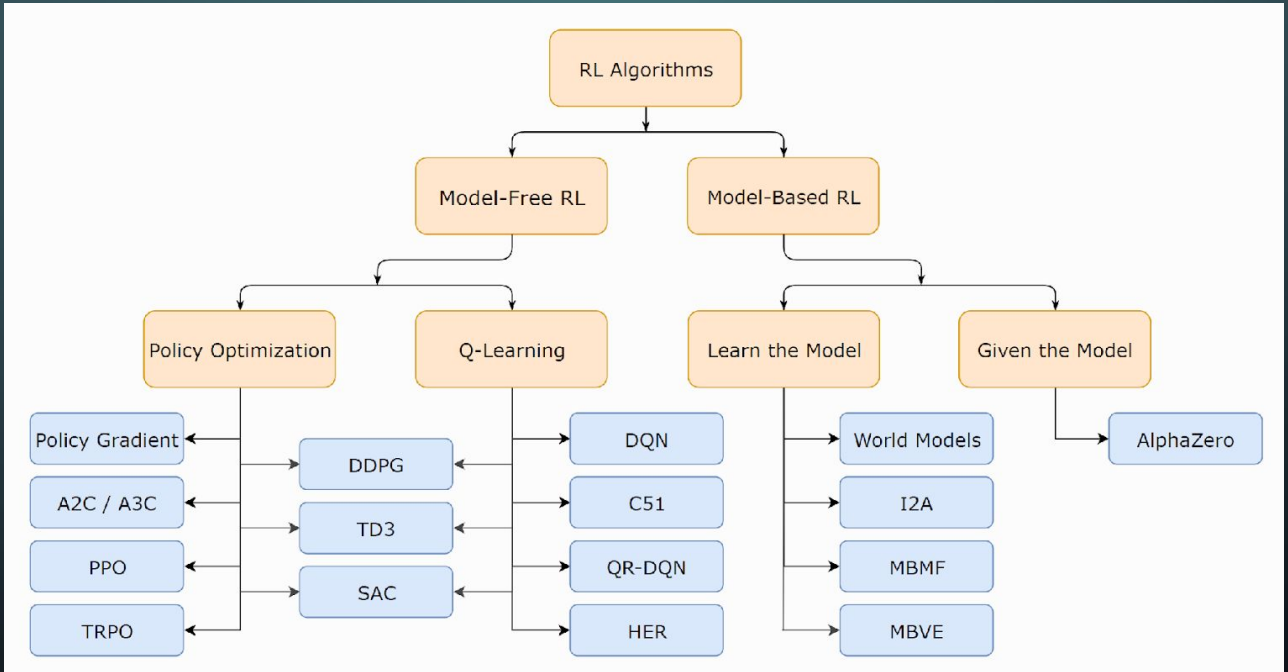
If the Agent performed better than average (Value), the Advantage is positive, if it performed worse than average, the Advantage is negative.

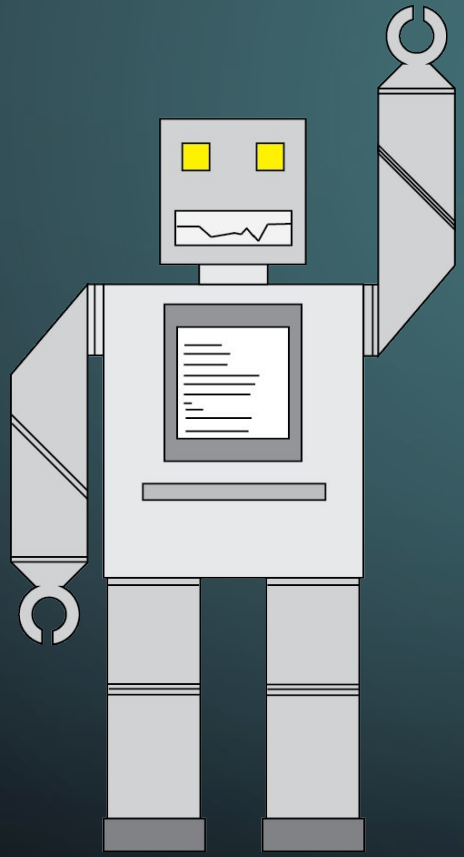
$$P_{log}(a_t, s_t) \leftarrow P_{log}(a_t, s_t) + \alpha A_t$$

CODE EXAMPLE



COMING UP…..


RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero



THANK YOU!