SSID: MSFTGUEST PWD: msevent242zc

WELCOME AND RECAP

- This is our third session on reinforcement learning
- Sessions I and II focused on some code samples and terminology using python notebooks mimicking the contextual bandit problem (finding the best slot machine to play from a collection of slot machines).
- Tonight's session presents a conceptual foundation for some key principles used in larger scale and more complex learning scenarios
- Prior code and presentations can be found at:
  - https://github.com/jimwill3/NY-AZML-Meetup/tree/CNTK/RL
- In June we will take a first look at an Azure Reinforcement Learning Service called "Personalizer"

https://aischool.microsoft.com/en-us/machine-learning/learning-paths (free self-paced training)

Markov Decision Processes

Michael H. Evangelista - michael@powerof.to

2019-05-23

Guest WiFi: MSFTGUEST -> Event Code -> “msevent242zc”

Jupyter Notebook:
https://notebooks.azure.com/mhe500/projects/gridworld

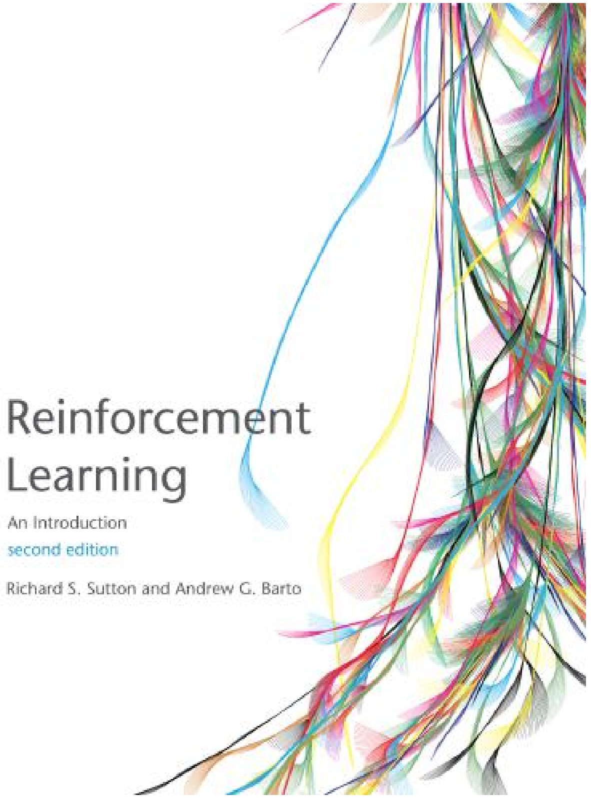
(Optional if you want to follow along)

What We'll Cover

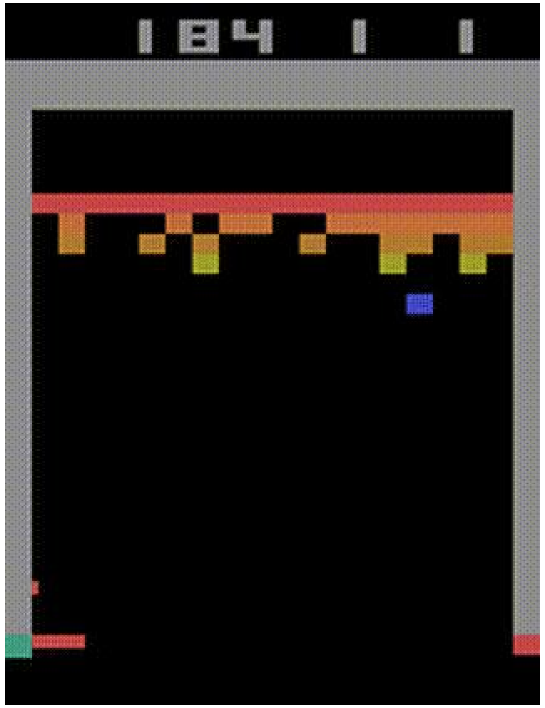
- Reinforcement learning overview
- MDP basics & describing a simple game as an MDP
- Finding the best actions for a given MDP using Value Iteration
- Value Iteration implementation in Python, solving the simple game
- Extending to more complex problems (like Atari games)

Reinforcement Learning

"Reinforcement learning ***is learning what to do****— how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by* ***trying them****. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that,* ***all subsequent rewards****. These two characteristics—* ***trial-and-error search*** *and* ***delayed reward****—are the two most important distinguishing features of reinforcement learning."*



Sutton, Richard S., and Andrew G. Barto. "Reinforcement Learning: an Introduction." *Reinforcement Learning: an Introduction*, The MIT Press., 2018, pp. 1–2.
Available as free PDF at http://incompleteideas.net/book/the-book-2nd.html



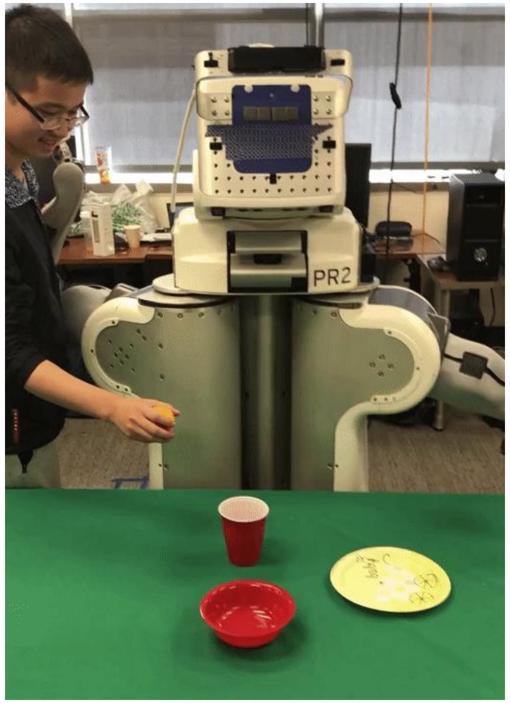


Image Credits:

- https://stable-baselines.readthedocs.io/en/master/guide/examples.html
- Tianhe Yu and Chelsea Finn (https://bair.berkeley.edu/blog/2018/06/28/daml/)
- Jason Peng (https://bair.berkeley.edu/blog/2018/04/10/virtual-stuntman/)

Why Bother with This?
Let's get to the CNTK/Tensorflow/MXNet code!

Soft Actor-Critic:
Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Tuomas Haarnoja[1] Aurick Zhou[1] Pieter Abbeel[1] Sergey Levine[1]

Abstract

Model-free deep reinforcement learning (RL) al-

of these methods in real-world domains has been hampered by two major challenges. First, model-free deep RL methods are notoriously expensive in terms of their sample com-

3. Preliminaries

We first introduce notation and summarize the standard and maximum entropy reinforcement learning frameworks.

3.1. Notation

We address policy learning in continuous action spaces. We consider an infinite-horizon Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are continuous, and the unknown state transition probability $p: \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty)$ represents the probability density of the next state $\mathbf{s}_{t+1} \in \mathcal{S}$ given the current state $\mathbf{s}_t \in \mathcal{S}$ and action $\mathbf{a}_t \in \mathcal{A}$. The environment emits a bounded reward $r: \mathcal{S} \times \mathcal{A} \rightarrow [r_{\min}, r_{\max}]$ on each transition. We will use $\rho_\pi(\mathbf{s}_t)$ and $\rho_\pi(\mathbf{s}_t, \mathbf{a}_t)$ to denote the state and state-action marginals of

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou
Daan Wierstra Martin Riedmiller
DeepMind Technologies
{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future

We consider tasks in which an agent interacts with an environment $\mathcal{E}$, in this case the Atari emulator, in a sequence of actions, observations and rewards. At each time-step the agent selects an action a_t from the set of legal game actions, $\mathcal{A} = \{1, \ldots, K\}$. The action is passed to the emulator and modifies its internal state and the game score. In general $\mathcal{E}$ may be stochastic. The emulator's internal state is not observed by the agent; instead it observes an image $x_t \in \mathbb{R}^d$ from the emulator, which is a vector of raw pixel values representing the current screen. In addition it receives a reward r_t representing the change in game score. Note that in general the game score may depend on the whole prior sequence of actions and observations; feedback about an action may only be received after many thousands of time-steps have elapsed.

Since the agent only observes images of the current screen, the task is partially observed and many emulator states are perceptually aliased, i.e. it is impossible to fully understand the current situation from only the current screen x_t. We therefore consider sequences of actions and observations, $s_t = x_1, a_1, x_2, ..., a_{t-1}, x_t$, and learn game strategies that depend upon these sequences. All sequences in the emulator are assumed to terminate in a finite number of time-steps. This formalism gives rise to a large but finite Markov decision process (MDP) in which each sequence is a distinct state. As a result, we can apply standard reinforcement learning methods for MDPs, simply by using the complete sequence s_t as the state representation at time t.

The goal of the agent is to interact with the emulator by selecting actions in a way that maximises future rewards. We make the standard assumption that future rewards are discounted by a factor of γ per time-step, and define the future discounted *return* at time t as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where T is the time-step at which the game terminates. We define the optimal action-value function $Q^*(s, a)$ as the maximum expected return achievable by following any strategy, after seeing some sequence s and then taking some action a, $Q^*(s, a) = \max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$, where π is a policy mapping sequences to actions (or distributions over actions).

The optimal action-value function obeys an important identity known as the *Bellman equation*. This

Under review as a conference paper at ICLR 2017

RL^2: FAST REINFORCEMENT LEARNING VIA SLOW REINFORCEMENT LEARNING

Yan Duan[†‡], John Schulman[†‡], Xi Chen[†‡], Peter L. Bartlett[†], Ilya Sutskever[‡], Pieter Abbeel[†‡]
[†] UC Berkeley, Department of Electrical Engineering and Computer Science
[‡] OpenAI
{rocky,joschu,peter}@openai.com, peter@berkeley.edu, {ilyasu,pieter}

ABSTRACT

Deep reinforcement learning (deep RL) has been successful in learning sophisticated behaviors automatically; however, the learning process requires a huge number of trials. In contrast, animals can learn new tasks in just a few trials, benefiting from their prior knowledge about the world. This paper seeks to bridge this gap. Rather than designing a "fast" reinforcement learning algorithm, we propose to represent it as a recurrent neural network (RNN) and learn it from data. In our proposed method, RL^2, the algorithm is encoded in the weights of the RNN, which are learned slowly through a general-purpose ("slow") RL algorithm. The RNN receives all information a typical RL algorithm would receive, including observations, actions, rewards, and termination flags; and it retains its state across episodes in a given Markov Decision Process (MDP). The activations of the RNN store the state of the "fast" RL algorithm on the current (previously unseen) MDP. We evaluate RL^2 experimentally on both small-scale and large-scale problems.

Hindsight Experience Replay

Marcin Andrychowicz*, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel[†], Wojciech Zaremba[†]
OpenAI

Abstract

Dealing with sparse rewards is one of the biggest challenges in Reinforcement

which was achieved in the episode.

2 Background

In this section we introduce reinforcement learning formalism used in the paper as well as RL algorithms we use in our experiments.

2.1 Reinforcement Learning

We consider the standard reinforcement learning formalism consisting of an agent interacting with an environment. To simplify the exposition we assume that the environment is fully observable. An environment is described by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a distribution of initial states $p(s_0)$, a reward function $r: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, transition probabilities $p(s_{t+1}|s_t, a_t)$, and a discount factor $\gamma \in [0, 1]$.

A deterministic policy is a mapping from states to actions: $\pi: \mathcal{S} \rightarrow \mathcal{A}$. Every episode starts with sampling an initial state s_0. At every timestep t the agent produces an action based on the current state: $a_t = \pi(s_t)$. Then it gets the reward $r_t = r(s_t, a_t)$ and the environment's new state is sampled from the distribution $p(\cdot|s_t, a_t)$. A discounted sum of future rewards is called a *return*: $R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$. The agent's goal is to maximize its expected return $\mathbb{E}_{s_0}[R_0|s_0]$. The Q-function or action-value function is defined as $Q^\pi(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t]$.

The Agent-Environment Interface

1. Agent receives state info S_t from Environment at time t
2. Agent chooses action A_t
3. Environment transitions to and emits S_{t+1}
4. Environment emits numerical reward R_{t+1}
5. $t = t + 1$
6. Repeat (until task ends or forever)

Agent's goal: Maximize Return (cumulative reward from time t and continuing):

$$G_t = \sum_{k=0..\infty} \gamma^k R_{t+k+1}$$

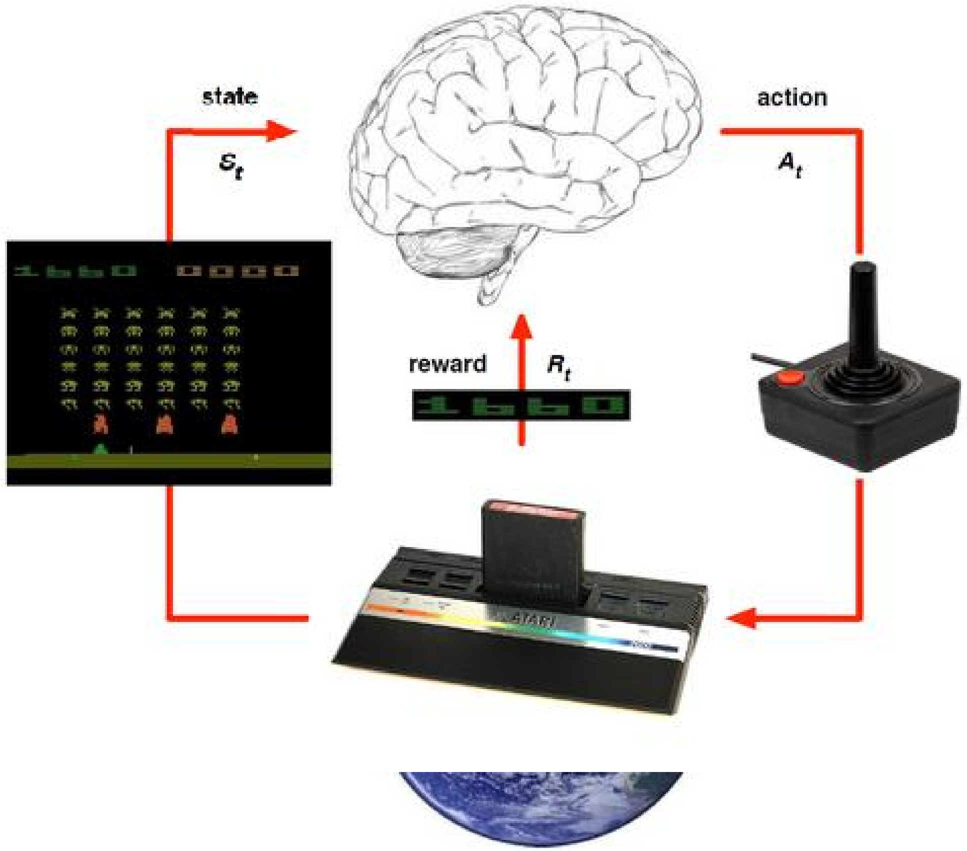


Image Credit: Silver, David. "Lecture 1: Introduction to Reinforcement Learning." Advanced Topics 2015 (COMPM050/COMPGI13). http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html.
Note: Atari image slightly modified to indicate state for simplicity.

Markov Decision Process

- Defines sequential decision making problem
- Possible environment states
- Allowed actions
- *Dynamics* – how an action leads from one state to another and how the agent is rewarded

Andrey Markov (1856-1922)

Simple MDP Illustration

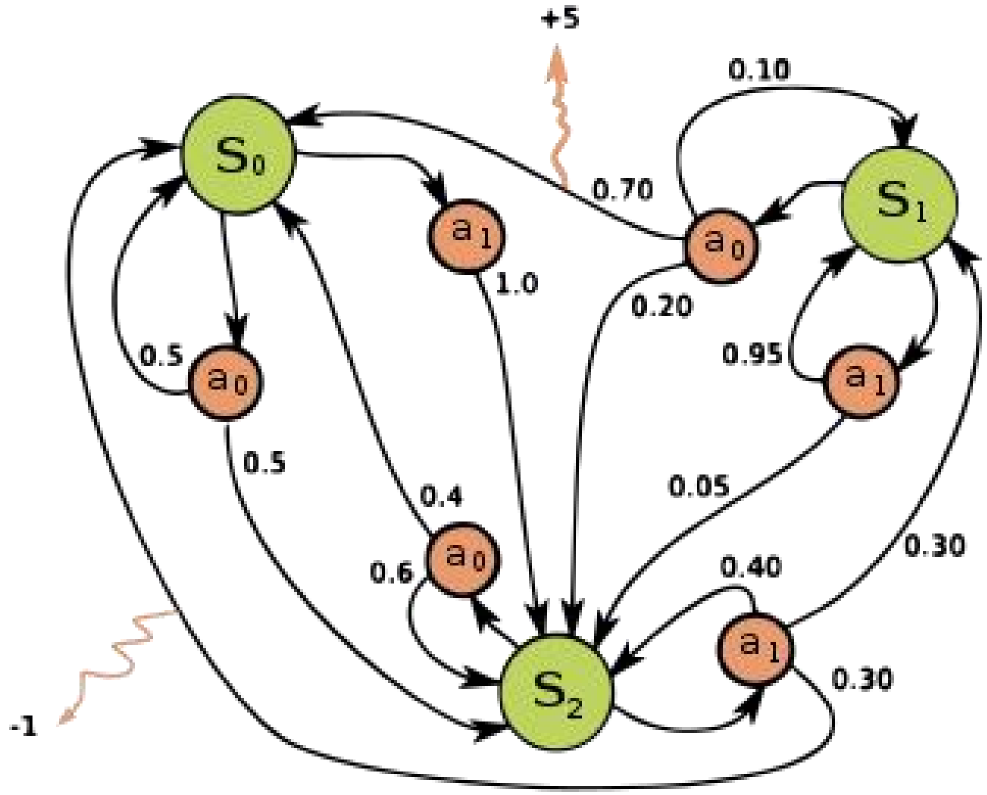


Image Credit: Wikipedia user waldoalveraz. "Markov Decision Process." Licensed under CC BY-SA 4.0.

State vs. Observation

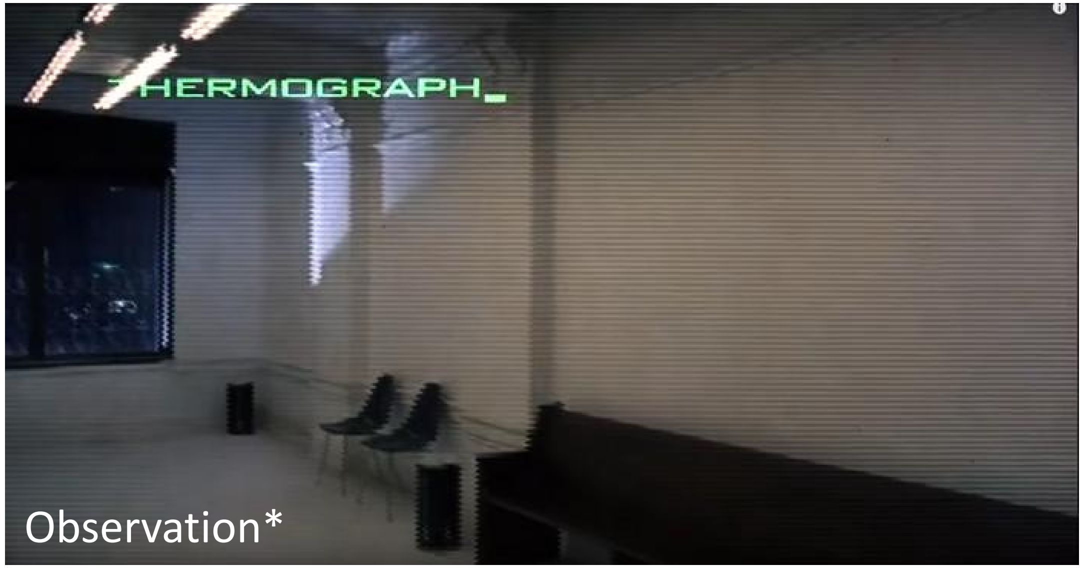
State

- A Markov state s “fully characterizes” the future (Markov Property)
- No matter what happened before s, the expected outcome (rewards, transitions) are the same going forward
- More simply: “the past doesn't affect the future”

Observation

- What you can “see”
- Different states may produce the same observation
- If you don't know (“see”) true state, your problem is Partially Observed MDP (POMDP)
- **Not dealing with POMDP today (assume we know true state)**

Agent



Observation*

* From human POV. Luckily, for Robocop this is an MDP, not POMDP!

State

Choose Best Action

Image Source: https://www.youtube.com/watch?v=Md14H_qD8iQ

Robot Grid World

- Robot can move to any adjacent square
- Environment is stochastic (meaning, with some probability the robot may "slip" and go to the left/right instead of forward in desired direction)
- Attempt to move to the rock or past the edge (whether intentionally or by slip) leaves you in current square
- Get the diamond for a score of +1 (win the game)
- Fall into the fiery pit, get a score of –1 (and die)

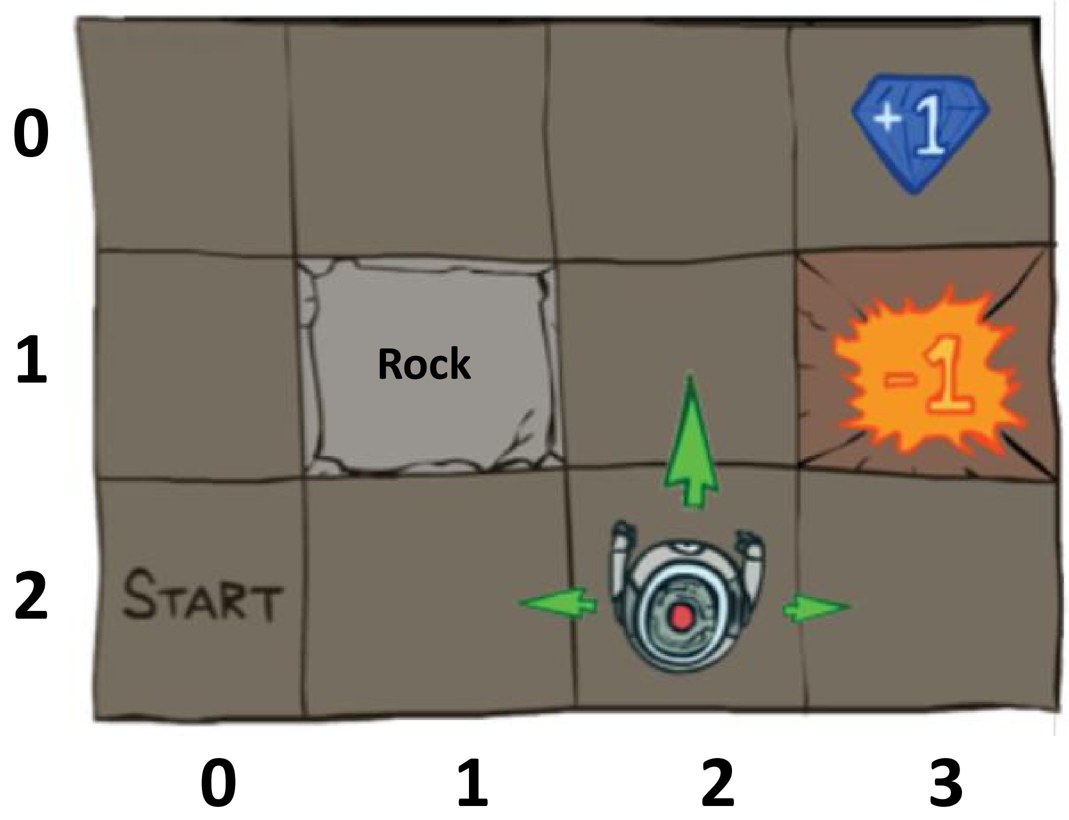


Image Credit: Abbeel, Pieter (2017). *Deep Reinforcement Learning Lecture 1: Motivation + Overview + MDPs + Exact Solution Methods*. https://sites.google.com/view/deep-rl-bootcamp/lectures

Grid World as an MDP

- Set of States - S
- Set of Actions - A
- Transition Probabity Function - $p(s' \mid s, a)$
  - Probability of ending up in state s', when taking action a from state s.
- Reward Function - $r(s, a, s')$
  - Reward gained when action a causes transition from state s to state s'.
- Initial State - s_0
- Time Horizon - H (can be ∞)
- Discount Factor - $\gamma \in [0, 1]$

$$G_t = \sum_{k=0..\boldsymbol{H}} \boldsymbol{\gamma}^k R_{t+k+1}$$

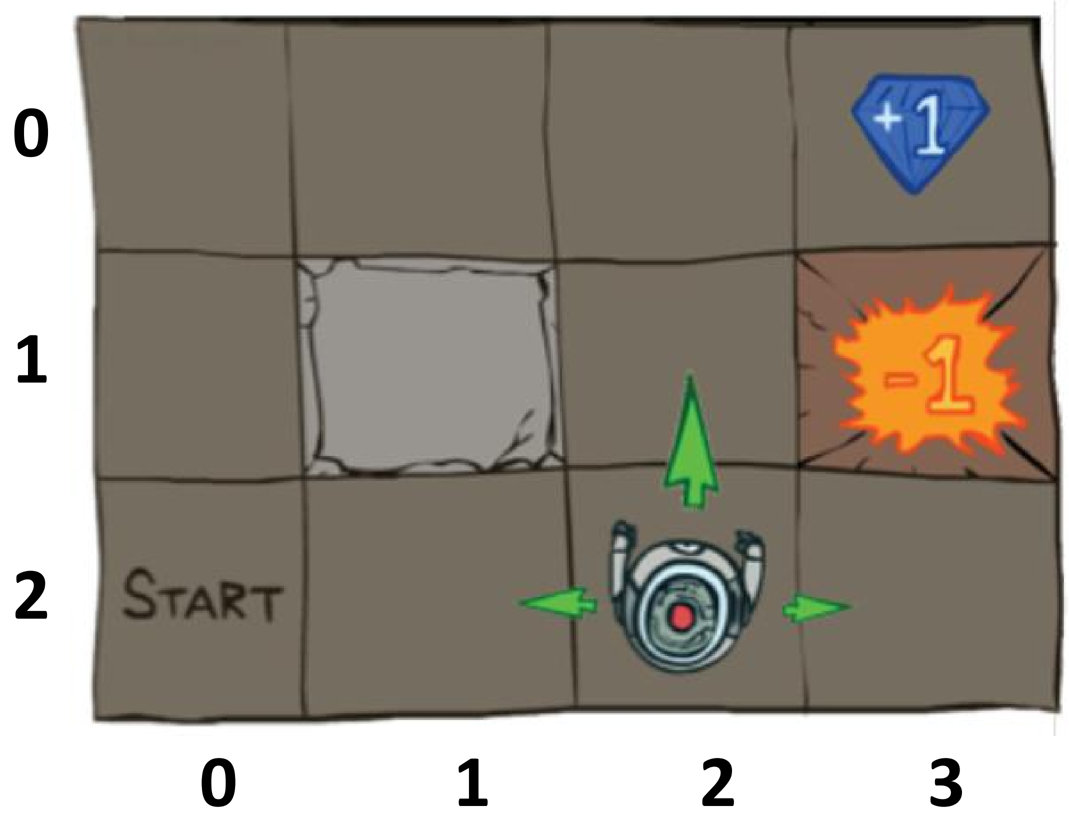


Image Credit: Abbeel, 2017

Exercise 1: Grid World Transition Probabilities

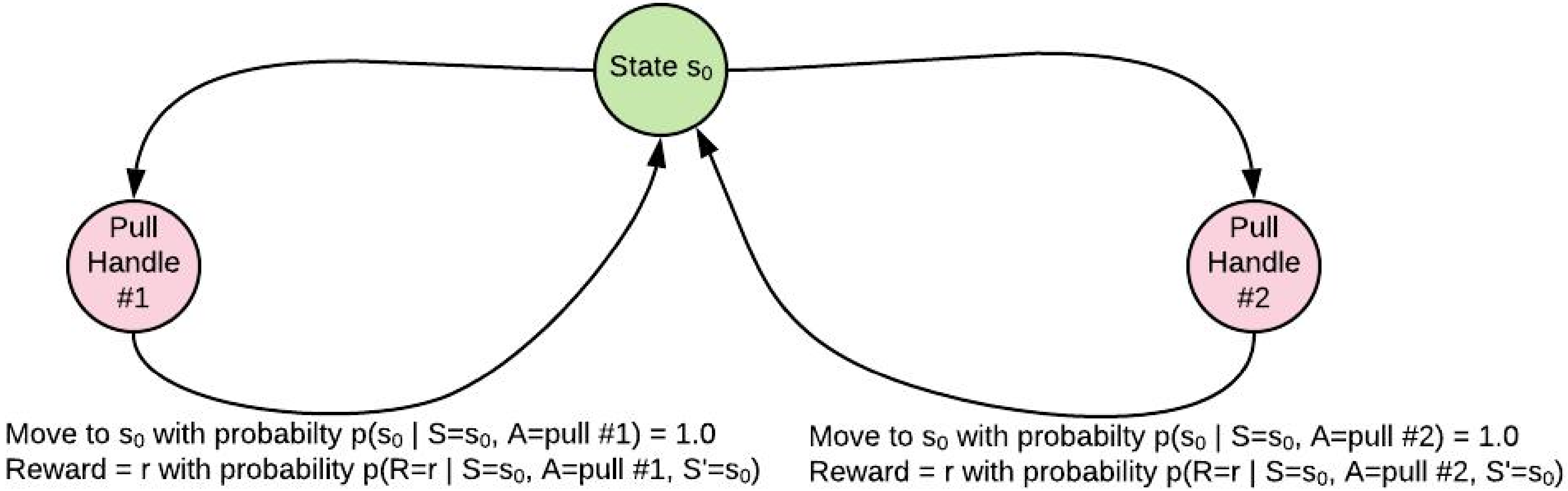
s'	$p(\ s' \mid s{=}(2,2),\ a{=}\text{up},\ slip{=}0.20)$
(0,0)	0
(0,1)	0
(0,2)	0
(0,3)	0
(1,0)	0
(1,1)	0
(1,2)	0.80
(1,3)	0
(2,0)	0
(2,1)	0.10
(2,2)	0
(2,3)	0.10

	0	1	2	3
0	(0,0)	(0,1)	(0,2)	(0,3) R=+1
1	(0,1)	Rock	(1,2)	(1,3) R=-1
2	(0,2) Start	(2,1)	(2,2)	(2,3)

Attempt to move UP from here

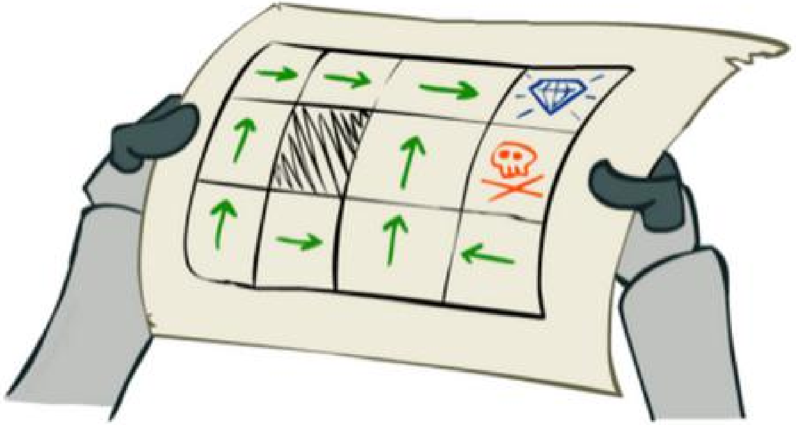
Remember Bandits (from sessions 1 & 2)?

- A bandit is a one-state MDP...
- ... with stochastic (random) rewards



Solving the MDP

Find a policy, π^*, that yields the best return:



π

- $\pi^* = \text{argmax}_\pi \, E_\pi[G_t \mid \pi, s_0]$
- $\quad = \text{argmax}_\pi \, E_\pi[\sum_{t=0..H} \gamma^t \, r(s_t, a_t, s_{t+1}) \mid \pi, s_0]$

Wait – what the heck does this mean?

- A policy, π, defines an agent's behavior
- π^* is the optimal policy. When followed, π^* should, in expectation (i.e., on average), produce the highest possible return starting from any state

Image Credit: Abbeel, 2017

The State- and Action-Value Function Definitions

State-Value Function under policy π:

Intuition: Expected return starting in state s and following policy π thereafter.

$$\boldsymbol{v_\pi(s)} = \mathrm{E}_\pi[G_t \mid S_t = s]$$

$$= \mathrm{E}_\pi[\textstyle\sum_{k=0..\infty} \gamma^k R_{t+k+1} \mid S_t = s] \qquad \text{(By definition of } G_t\text{)}$$

Action-Value Function under policy π:

Intuition: Expected return starting in state s, taking action a as the first step, then following policy π thereafter.

$$\boldsymbol{q_\pi(s, a)} = \mathrm{E}_\pi[G_t \mid S_t = s, A_t = a]$$

$$= \mathrm{E}_\pi[\textstyle\sum_{k=0..\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a] \qquad \text{(By definition of } G_t\text{)}$$

***Calculating $q_\pi(s)$ from $v_\pi(s)$*:**

Intuition: If we know v_π we can easily get q_π.

$$q_\pi(s, a) = \mathrm{E}_\pi[\textstyle\sum_{k=0..\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a]$$

$$= \mathrm{E}_\pi[R_{t+1} + \gamma \textstyle\sum_{k=1..\infty} \gamma^{k-1} R_{t+k+1} \mid S_t = s, A_t = a] \qquad \text{(Split summation)}$$

$$= \mathrm{E}_\pi[R_{t+1} + \gamma v_\pi(s') \mid S_t = s, A_t = a] \qquad \text{(By definition of } v_\pi(s')\text{)}$$

$$= \textstyle\sum_{s'} p(s' \mid s, a)\, [r(s, a, s') + \gamma v_\pi(s')] \qquad \text{(By definition of expectation \& } R_{t+1}\text{)}$$

<u>Optimal</u> State- and Action-Value Function

State-Value Function under <u>optimal policy</u> π^*:

Intuition: Expected return starting in state s and following the <u>optimal policy</u> π^* thereafter.

$$\boldsymbol{v^*(s)} = \max_{\boldsymbol{\pi}} \mathrm{E}_{\pi}[G_t \mid S_t{=}s]$$
$$= \max_{\boldsymbol{\pi}} \mathrm{E}_{\pi}[\textstyle\sum_{k=0..\infty} \gamma^k R_{t+k+1} \mid S_t{=}s]$$

Action-Value Function under <u>optimal policy</u> π^*:

Intuition: Expected return starting in state s, <u>taking action a as the first step</u>, then following the <u>optimal policy</u> π^* thereafter. <u>**If we know $q^*(s, a)$, we effectively have our policy**</u>.

$$\boldsymbol{q^*(s, a)} = \max_{\boldsymbol{\pi}} \mathrm{E}_{\pi}[G_t \mid S_t{=}s, A_t{=}a]$$
$$= \max_{\boldsymbol{\pi}} \mathrm{E}_{\pi}[\textstyle\sum_{k=0..\infty} \gamma^k R_{t+k+1} \mid S_t{=}s, A_t{=}a]$$

Optimal State- and Action-Value Function

Same As Prior Slide

Just Maximizing

Over all Possible Policies

State-Value Function under optimal policy π^*:

Intuition: Expected return starting in state s and following the optimal policy π^* thereafter.

$$\boldsymbol{v^*(s)} = \max_{\boldsymbol{\pi}} \mathrm{E}_\pi[G_t \mid S_t{=}s]$$
$$= \max_{\boldsymbol{\pi}} \mathrm{E}_\pi[\textstyle\sum_{k=0..\infty} \gamma^k R_{t+k+1} \mid S_t{=}s]$$

Key: If we can find v*(s), we can get q*(s) and problem solved!

Action-Value Function under optimal policy π^*:

Intuition: Expected return starting in state s, taking action a as the first step, then following the optimal policy π^* thereafter. **If we know $q^*(s, a)$, we effectively have our policy**.

$$\boldsymbol{q^*(s, a)} = \max_{\boldsymbol{\pi}} \mathrm{E}_\pi[G_t \mid S_t{=}s, A_t{=}a]$$
$$= \max_{\boldsymbol{\pi}} \mathrm{E}_\pi[\textstyle\sum_{k=0..\infty} \gamma^k R_{t+k+1} \mid S_t{=}s, A_t{=}a]$$

Exercise 2.1: Find $v^*(s)$

Remember:

$$\begin{aligned} v^*(s) &= \max_\pi E_\pi[G_t \mid S_t=s] \\ &= \max_\pi E_\pi[\textstyle\sum_{k=0..\infty} \gamma^k R_{t+k+1} \mid S_t=s] \\ &= \max_\pi E_\pi[\textstyle\sum_{k=0..\infty} \gamma^k r(s_{t+k}, a_{t+k}, s_{t+k+1}) \mid S_t=s] \end{aligned}$$

	γ = 1 (no discounting) Slip = 0 (deterministic dynamics)
$v^*(0,3) =$	1
$v^*(0,2) =$	1
$v^*(0,1) =$	1
$v^*(2,0) =$	1
$v^*(1,3) =$	-1

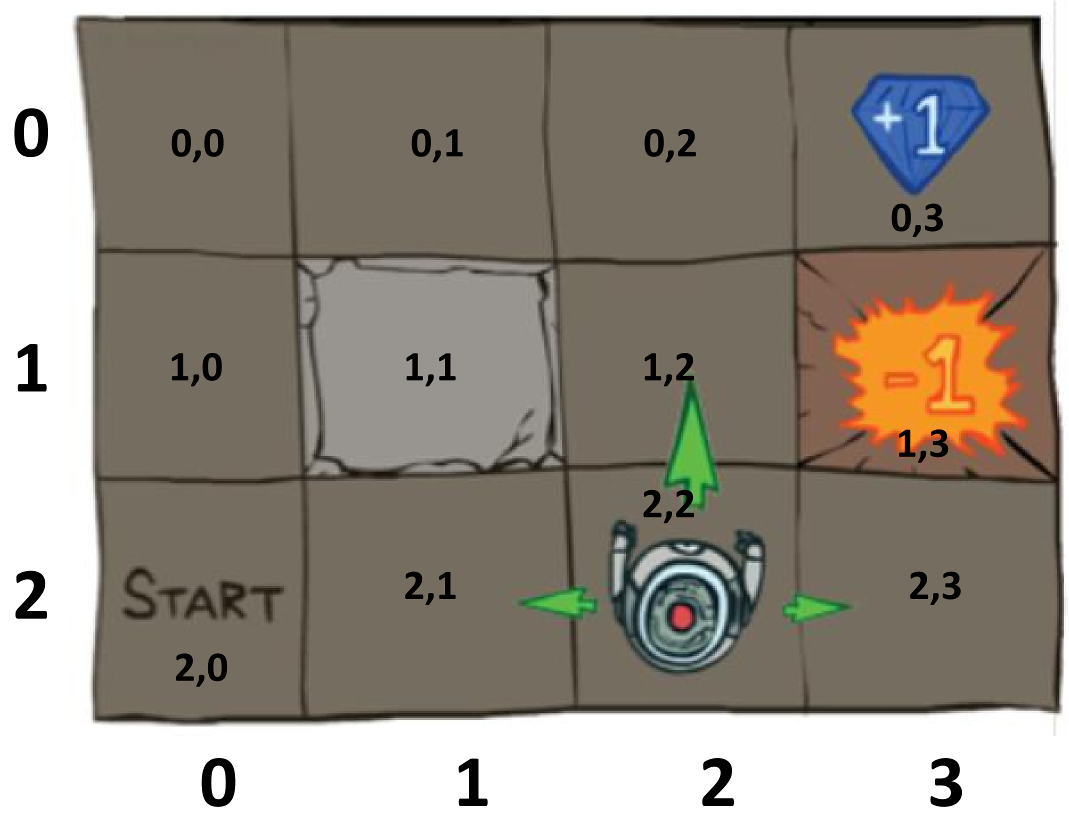


Image Credit: Abbeel, 2017

Exercise 2.2: Find $v^*(s)$

Remember:

$$\begin{aligned} \boldsymbol{v^*(s)} &= \max_{\boldsymbol{\pi}} \mathrm{E}_\pi[G_t \mid S_t{=}s] \\ &= \max_{\boldsymbol{\pi}} \mathrm{E}_\pi[\textstyle\sum_{k=0..\infty} \gamma^k R_{t+k+1} \mid S_t{=}s] \\ &= \max_{\boldsymbol{\pi}} \mathrm{E}_\pi[\textstyle\sum_{k=0..\infty} \gamma^k r(s_{t+k}, a_{t+k}, s_{t+\mathrm{k}+1}) \mid S_t{=}s] \end{aligned}$$

	γ = 0.9 **Slip = 0 (deterministic dynamics)**
$v^*(0,3) =$	1
$v^*(0,2) =$	0.9 × 1 = 0.9
$v^*(0,1) =$	0.9 × 0.9 × 1 = 0.81 ...or... 0.9 × $v^*(0,2)$ = 0.81
$v^*(2,0) =$	0.9 × 0.9 × 0.9 × 0.9 × 0.9 × 1 = 0.59 ...or... 0.9 × max($v^*(1,0)$, $v^*(2,1)$) = 0.59
$v^*(1,3) =$	-1

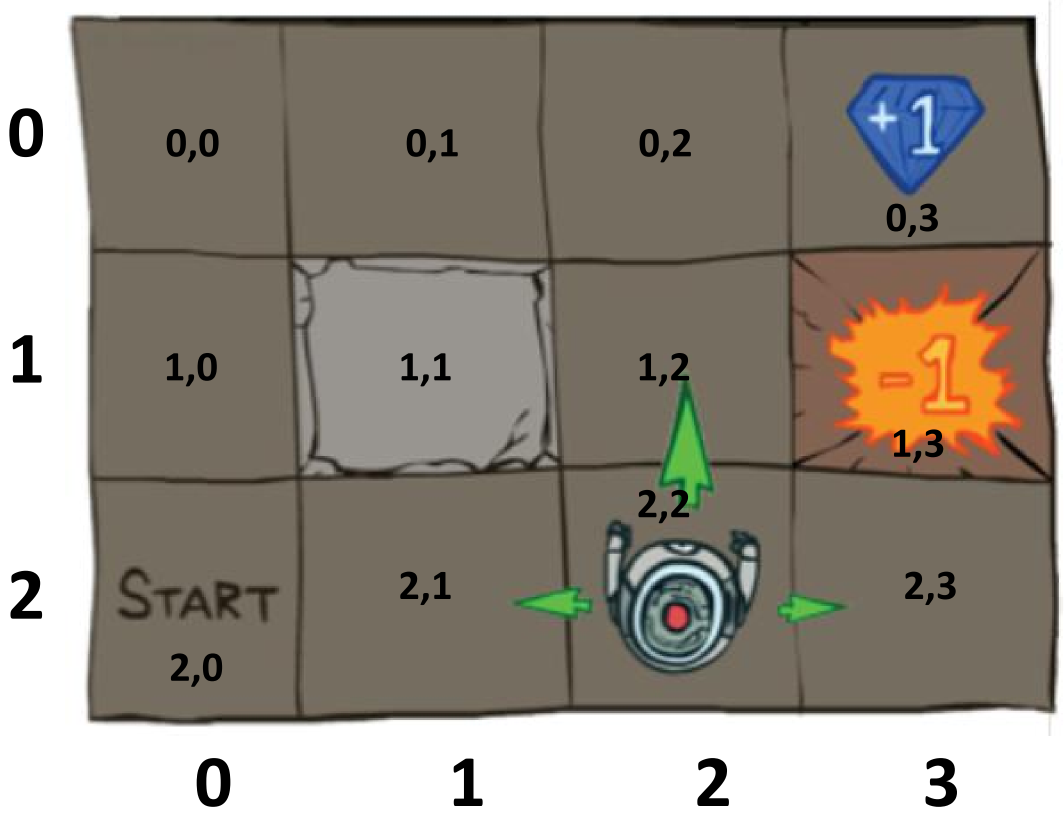


Image Credit: Abbeel, 2017

Exercise 2.3: Find $v^*(s)$

Remember:

$$\begin{aligned} \boldsymbol{v^*(s)} &= \max_\pi \mathrm{E}_\pi[G_t \mid S_t{=}s] \\ &= \max_\pi \mathrm{E}_\pi[\textstyle\sum_{k=0..\infty} \gamma^k R_{t+k+1} \mid S_t{=}s] \\ &= \max_\pi \mathrm{E}_\pi[\textstyle\sum_{k=0..\infty} \gamma^k r(s_{t+k}, a_{t+k}, s_{t+k+1}) \mid S_t{=}s] \end{aligned}$$

	γ = 0.9 Slip = 0.2
$v^*(0,3)$ =	1
$v^*(0,2)$ =	# 80% chance of success moving right... 0.8 × 0.9 × 1 + # 10% chance of slip left, bounce off top edge 0.1 × 0.9 × $v^*(0,2)$ + # 10% chance of slip right.. 0.1 × 0.9 × $v^*(1,2)$ = ???
$v^*(0,1)$ =	I dunno, let Value Iteration figure it out
$v^*(2,0)$ =	I dunno, let Value Iteration figure it out
$v^*(1,3)$ =	-1

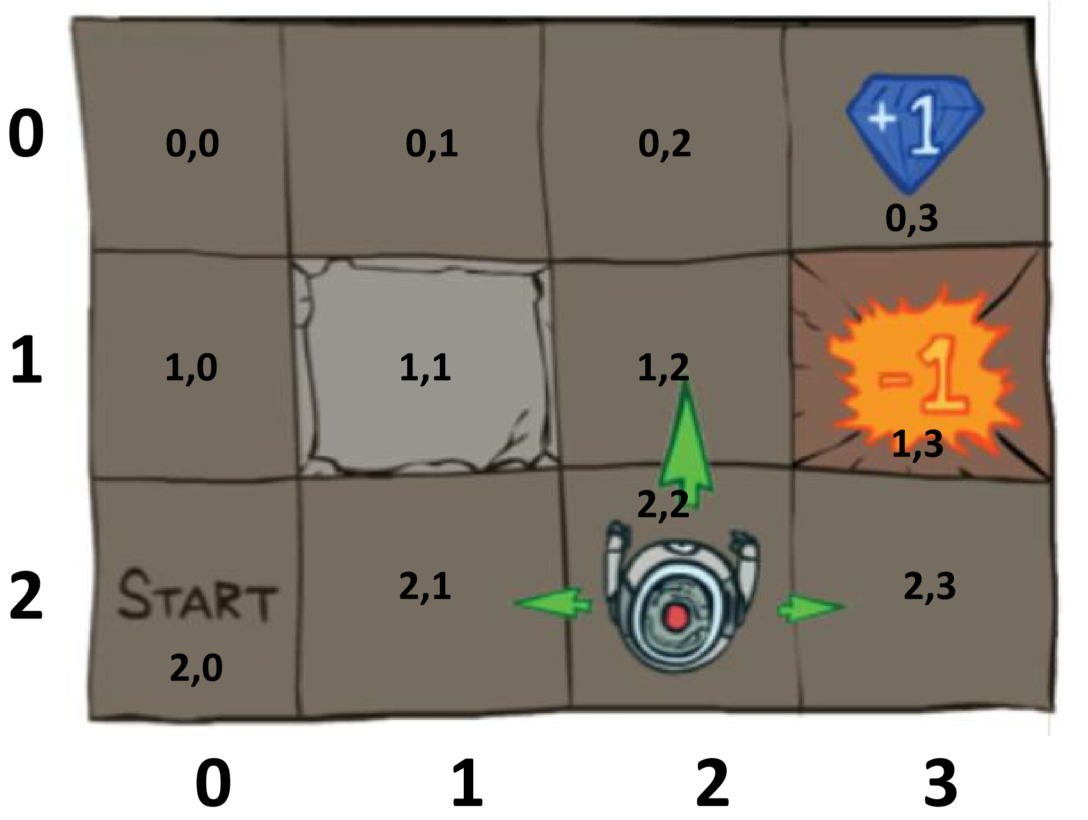


Image Credit: Abbeel, 2017

Value Iteration Algorithm

$V_0^*[s] \leftarrow 0 \ \forall s, k \leftarrow 0$

Repeat until convergence:

For each state s in S:

$V_k^*[s] \leftarrow \max_a \sum_{s'} p(s' \mid s, a)\,[r(s, a, s') + \gamma V_{k-1}^*[s']]$

$k \leftarrow k+1$

$\pi(\mathrm{s}) \leftarrow \mathrm{argmax}_a\, Q_K(s, a) = \mathrm{argmax}_a \sum_{s'} p(s' \mid s, a)\,[r(s, a, s') + \gamma V_K^*[s']]$

Bellman Optimality Equation*

Intuition:

V_k*[s] is best possible value of state s if I can take at most k steps. For for each state s we ask:

- For each possible action a I can take from here (state s)...
- ...in what states could I possibly end up (s')? And with what probability?
- ...how much immediate reward would I get for landing in sate s'?
- ...using my current table of values for V_{k-1}, how much is that next state s' worth (after discounting by γ)?

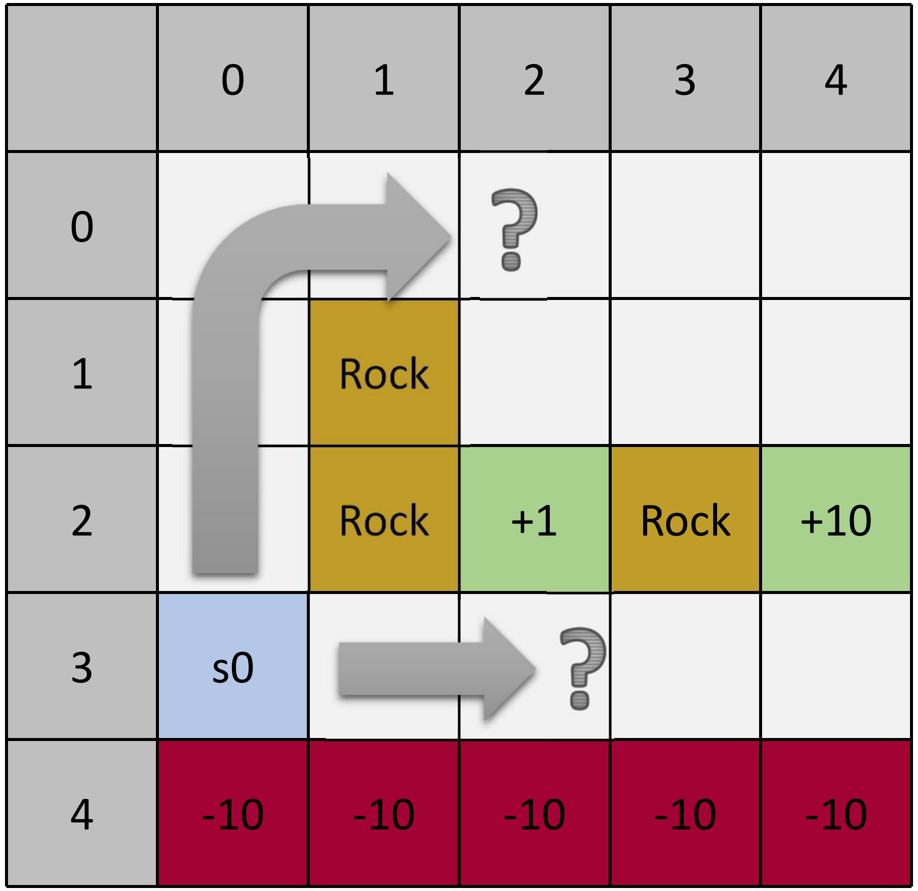
The highest value across all the action possiblities is new estimate of V*[s].

* Technically, this isn't *exactly* the BOE becuase we're relating iteration k to iteration k-1, but it's derived from BOE and is basically same thing.

Exercises: Value Iteration Notebook

https://notebooks.azure.com/mhe500/projects/gridworld

Exercise 3: Effect of Gamma and Stochasticity ("Slip Chance")



- When will the agent prefer the lower path (go near the cliff)?
- The upper path (stay away from the cliff)?
- When will the agent prefer the +1 reward? +10?

Now We Can Solve Atari Games!

- Now we can calculate q^* for Atari 2600 games
- Just check $q^*(s)$ to find the best action during game play
- **Will this work?**

$Q\Big[$ 

$, \text{Go Up}\Big] = +50$

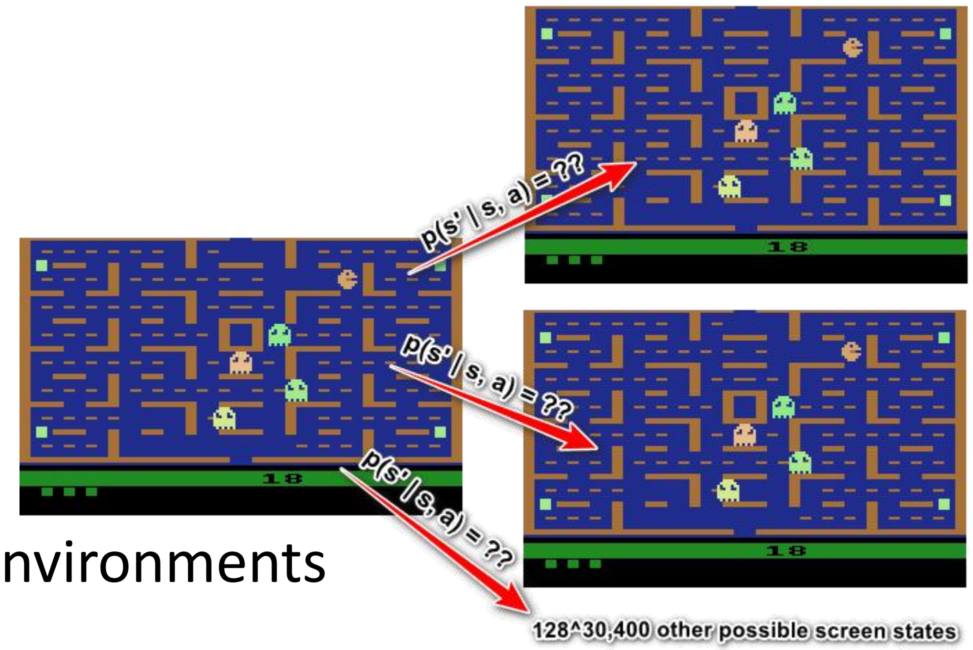
$Q\Big[$ 

$, \text{Go Right}\Big] = \text{-}100$

... other state, action Q values ...

Limitations of Value Iteration

1. Not appropraite for large state spaces
    - Atari VCS has 160x192 pixels, 128 colors (NTSC version)*
    - $128^{(160*192)} = 128^{30,400}$ screen states
    - Too big to hold $V[s]$ or $Q[s, a]$
    - Too big to iterate over all states



2. Requires full knowledge of dynamics
    - No need for trial-and-error
    - Assumption not reasonable for complex environments

*Due to limitations and technical details of Atari VCS graphics implementation, this is an approximation.

Other Approaches

Addresses unkown dynamics problem

	Rely on __Known__ Dynamics	*Use __Samples__ to Estimate Dynamics (Transitions & Rewards)*
Tabular Methods Store Full Table of $V^*[s]$ or $Q^*[s, a]$	**Exact Methods** • Value Iteration • Policy Iteration	**Monte Carlo** (learn from sampled full episodes) **Temporal Difference Methods** (learn from individually sampled transitions) • TD(0) / TD(λ) (learn $V^\pi[s]$ on-policy from (s, a, r, s') samples) • Sarsa (learn $Q^*[s, a]$ on-policy from (s, a, r, s', a) samples) • Q-Learning (learn $Q^*[s, a]$ off-policy from (s, a, r, s') samples)
Non-Tabluar Function Approximators Approximate $q_*(s, a)$ with a non-tabluar function approximator (e.g., neural network)		**Deep Q-Learning** • Train neural network to approximate to $q_*(s, a)$ off-policy • First method to solve Atari 2600 Games (Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. Retrieved from http://arxiv.org/abs/1312.5602) (cited > 999 times)

Addresses large state space problem

Tabular Q-Learning

$Q[s, a] \leftarrow 0 \ \forall s$

Repeat:

Choose action a using some exploration policy (e.g., ε-greedy policy)

Execute a in the environment to get next state s' and reward R

$target \leftarrow R + \gamma \max_{a'} Q[s', a']$ # $Q[s', a']$ considered 0 if episode ended (why?)

$Q[s, a] \leftarrow Q[s, a] + \alpha\,(target - Q[s, a])$ TD Error (δ_t)

$S \leftarrow S'$

Intuition/Explanation:

- ε-greedy means, with ε probability take a random action, else act according to your policy.
- We collect many state transition samples by running ε-greedy policy and use *actual received reward* plus *next state's (discounted) best action value* (from current $Q[...]$ values) as estimate of $Q[s, a]$
- If $Q[s, a]$ differs from our just-sampled estimate, adjust $Q[s, a]$ by adding a fraction (α) of the difference. Eventually, with enough samples, $Q[s, a]$ will converge to $q^*(s, a)$.
- **No use of dynamics (transition probabilities) & no “sweep” (foreach) over all states.**

Deep Q-Learning Intuition

~~$Q[s, a] \leftarrow 0$ for all s in S~~

Replace Q table with neural network Q_Θ estimates $q(s, a)$.

While not conveged:

Choose action a using some exploration policy (e.g., ε-greedy)

Execute a in the environment to get next state s' and reward R

$target \leftarrow R + \gamma \max_{a'} Q_\Theta(s', a')$ # $Q_\Theta(s', a')$ considered 0 if episode ended (why?)

~~$Q[s, a] \leftarrow Q[s, a] + \alpha\,(target - Q[s, a])$~~

Train Q_Θ on batches of X=(s, a), Y=target using “supervised learning” with MSE loss.

$S \leftarrow S'$

Intuition:

- Transform pixels into lower-dimensional learned internal representation using convolutional network (CNN)
- Replace the Q table with a neural network
- Gather and store experience, just like before
- Use the experience to generate target Q values, just like before
- Train the neural network as you would with supervised learning (what's odd here?)

Pong $q^*(s, a)$-Values

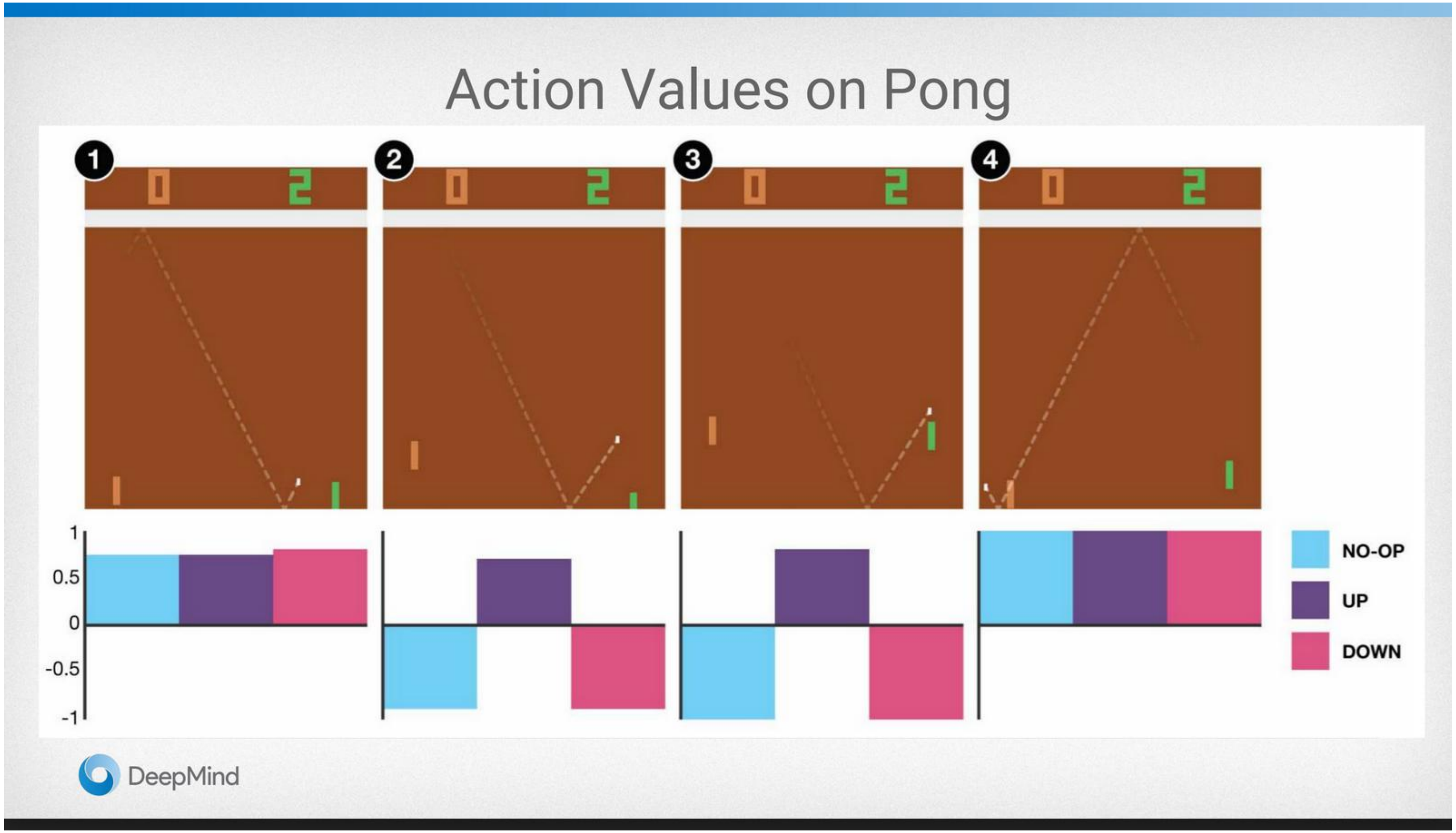
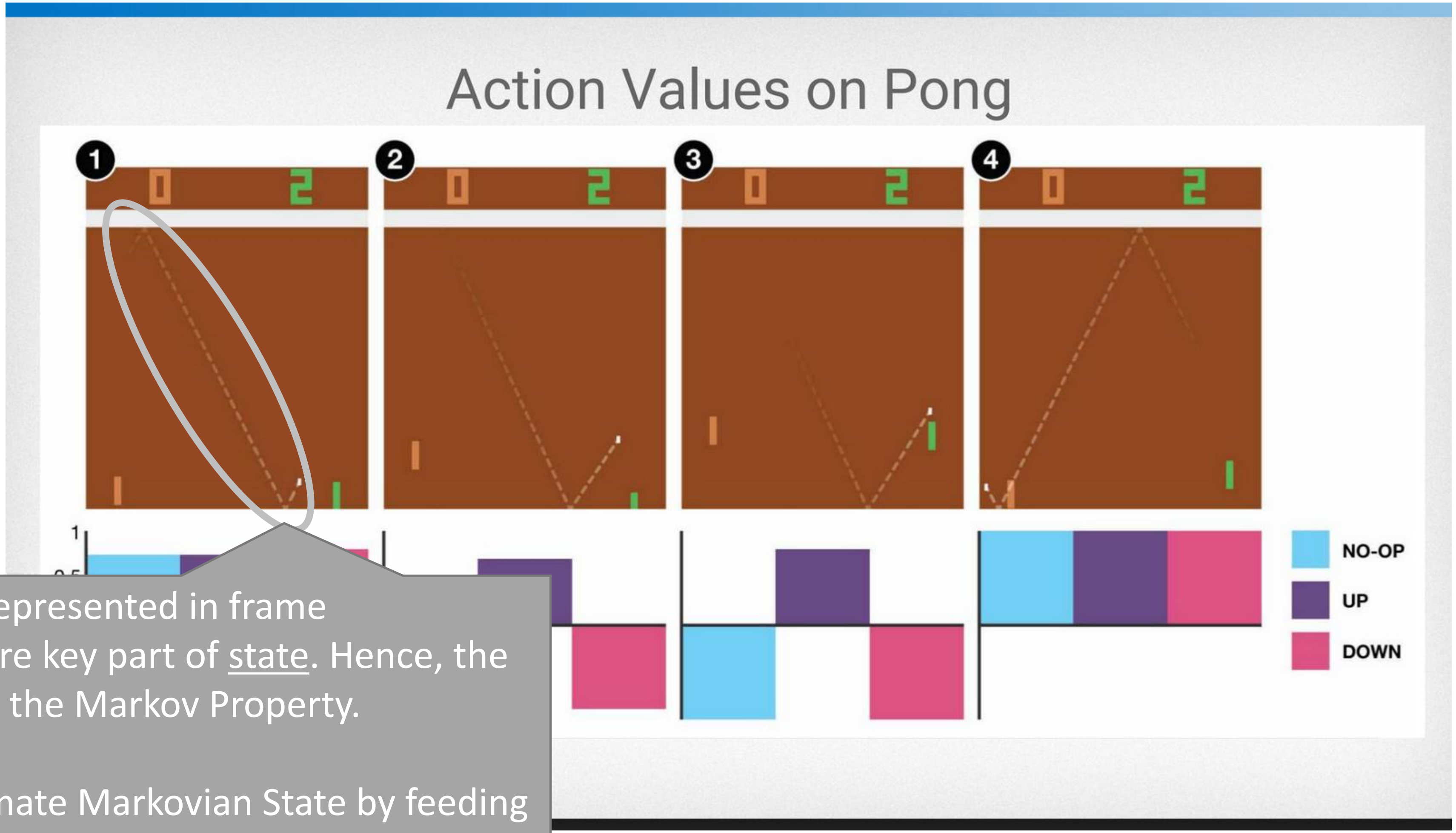


Image Credit: Mnih, Volodymyr (2017). *Deep Q-Networks (2017 Deep RL Bootcamp)*. https://sites.google.com/view/deep-rl-bootcamp/lectures

Atari Frames are **<u>Not</u>** Markovian States (why not?)



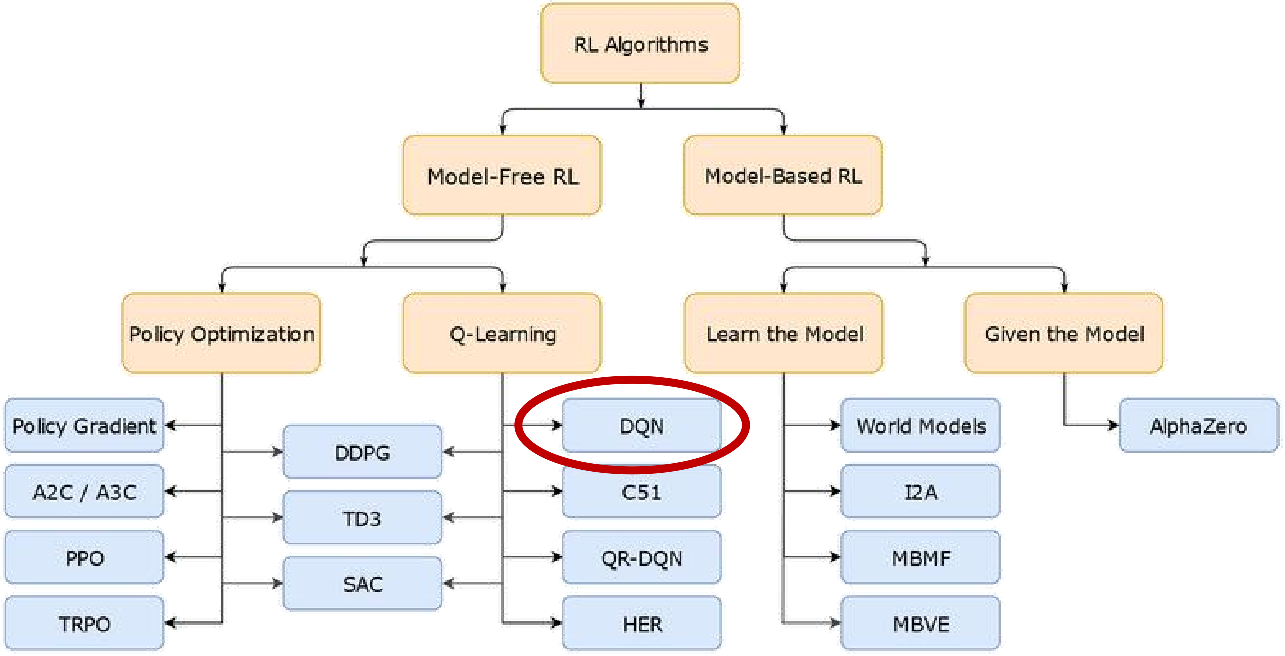
Velocities are not represented in frame <u>observations</u>, but are key part of <u>state</u>. Hence, the frame doesn't have the Markov Property.

Mnih et al approximate Markovian State by feeding multiple successive frames into DQN.

Image Credit: Mnih, Volodymyr (2017). *Deep Q-Networks (2017 Deep RL Bootcamp)*. <u>https://sites.google.com/view/deep-rl-bootcamp/lectures</u>

More Notes on DQN

- Prior slides are *simplified gist &* omit many key contributions of Mnih et al to making DQN work on Atari games
- DQN not the only approach to playing games
- More recent gaming advances (e.g., Starcraft, DotA) using different algorithms (variations of Policy Gradient)



A non-exhaustive, but useful taxonomy of algorithms in modern RL.

Image Credit: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Summary

- Agent-Environment Loop defines how agent & environment interact
- Markov Decision Processes formally define an environment/problem
- MDP assumes state known (else, POMDP)
- MDP can be solved (find v^*, q^*) through Value Iteration if state space is small enough and dynamics are known
- Sample-based methods (Monte Carlo, TD, Q-Learning) can approximatly solve the MDP without knowing dynamics if state space is small enough
- Neural Networks can be used to replace Q table becuase they generalize and propvide estimate Q value for any state (including unseen states)

Resources

- **Free Online RL Courses**
  - *OpenAI RL Bootcamp* (https://sites.google.com/view/deep-rl-bootcamp/lectures)
    - Lecture 1 – Pieter Abbeel on MDPs and Exact Solution Methods
    - Lecture 2 – Rocky (Yan) Duan on Temporal Difference (TD) Methods
    - Lecture 3 – Vlad Mnih on Deep Q-Learning / Atari (see also http://arxiv.org/abs/1312.5602)
  - *David Silver 2015 RL Course* (heavier foundations focus (e.g., MDPs, etc. Based heavily on earlier edition of Sutton & Barto) (http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html)
  - *Sergey Levine Berkeley Fall 2018 CS294-112* (heavier DNN, robotics, and current trends focus) (http://rail.eecs.berkeley.edu/deeprlcourse/)
- **Books**
  - Sutton, Richard S.., and Andrew G.. Barto. *Reinforcement Learning: an Introduction*. The MIT Press., 2018. Available as free PDF at http://incompleteideas.net/book/the-book-2nd.html
  - Géron, Aurélien. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly, 2018. (See RL chapter for DQN explanation; Note: don't purchase old edition; wait for 2019 second ed. covering Tensorflow 2.0)

Thank you

(And thank you Microsoft for hosting)