INTRO TO **REINFORCEMENT LEARNING**

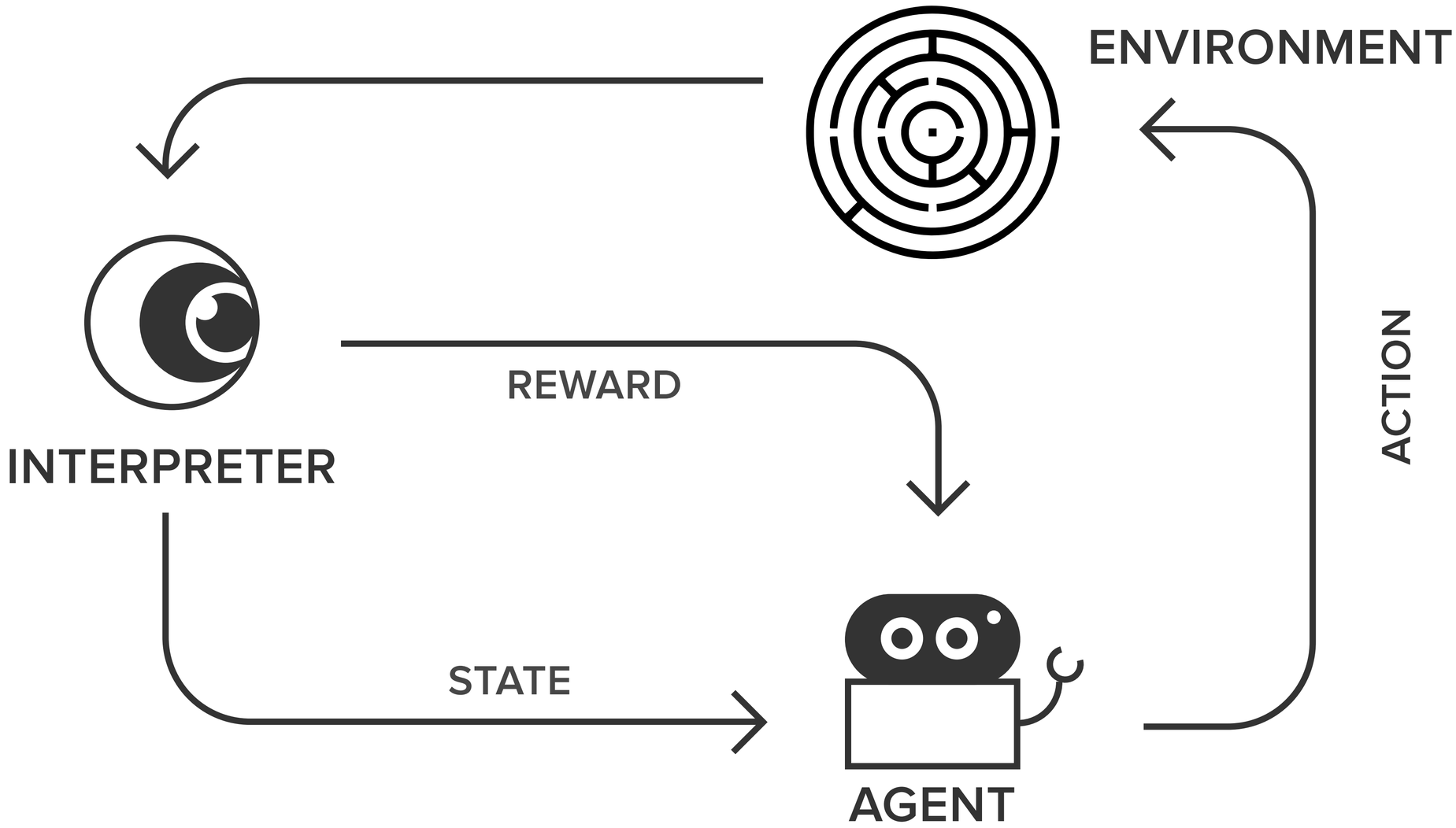
IVAN DIDUR

CTO & Co-Founder at **DataRoot Labs**

Lecturer at **DataRoot University**

OVERVIEW

WHAT IS REINFORCEMENT LEARNING?



RL APPLICATIONS

Robots driven by reinforcement learning in factory

Space management in warehouse

Dynamic pricing to maximize revenue from products

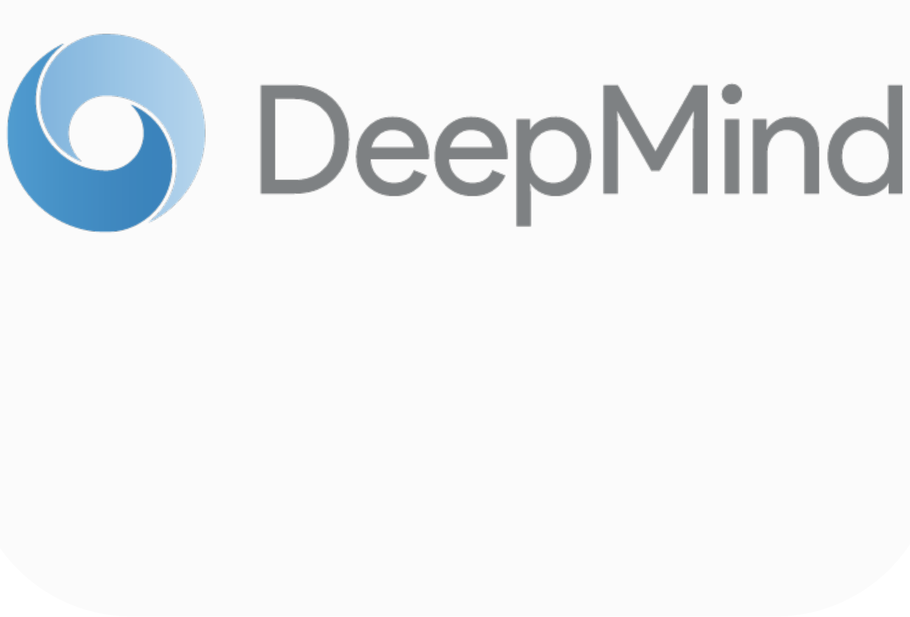
Customer delivery optimisation

eCommerce personalization

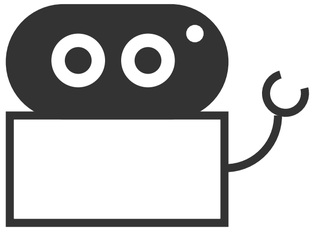
Financial investment decisions

Dynamic treatment (DTR) for finding effective treatments for patients.

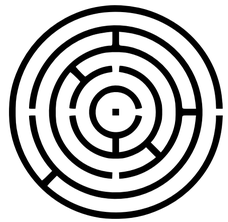
LEADERS IN RL RESEARCH



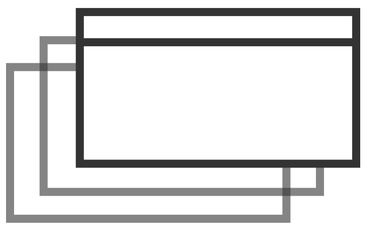
RL SYSTEM COMPONENTS



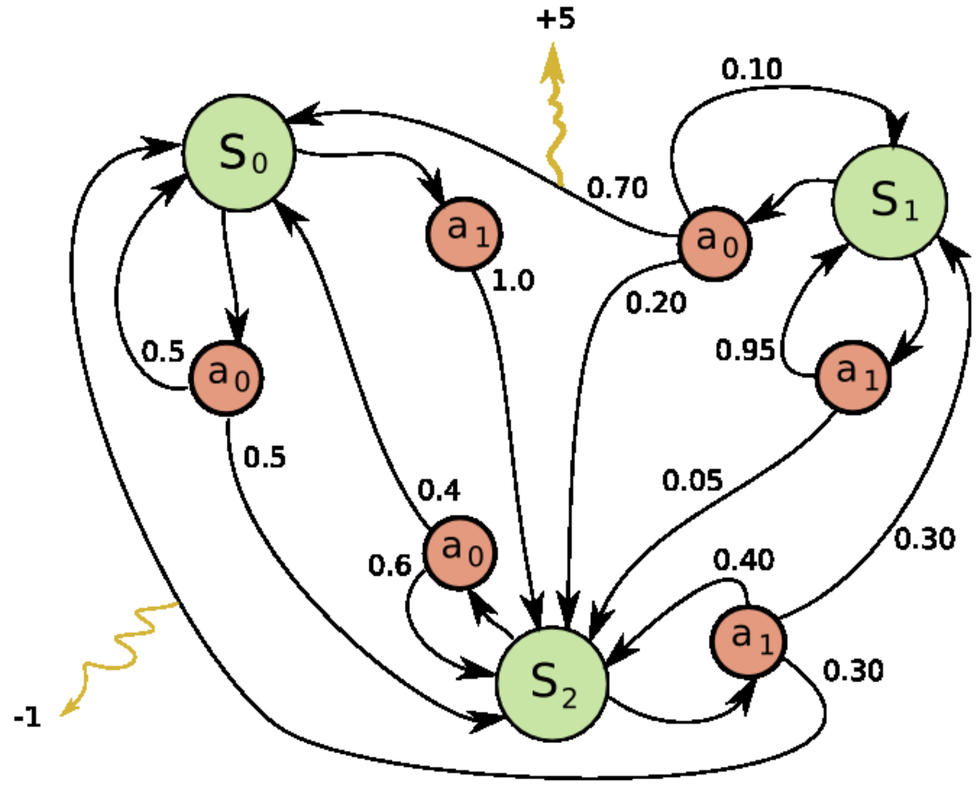
Agent



Environment



Episodes
& Terminal States



State, Action, Reward

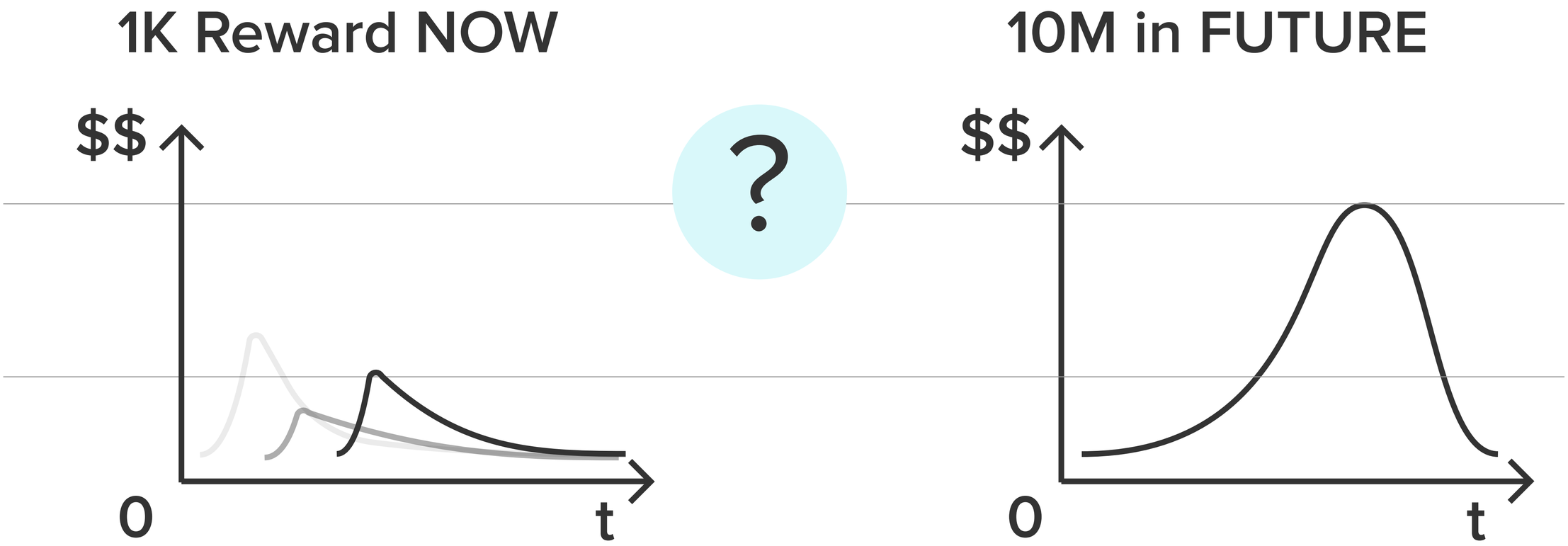
EXPLORE-EXPLOIT DILEMMA

?

EXPLORE

EXPLOIT

REWARD ENGINEERING & VALUE FUNCTION



REWARD ENGINEERING & VALUE FUNCTION

1K Reward NOW

Reward is IMMEDIATE

10M in FUTURE

Value is a meausure of POSSIBLE Future Rewards we may get from being in this state

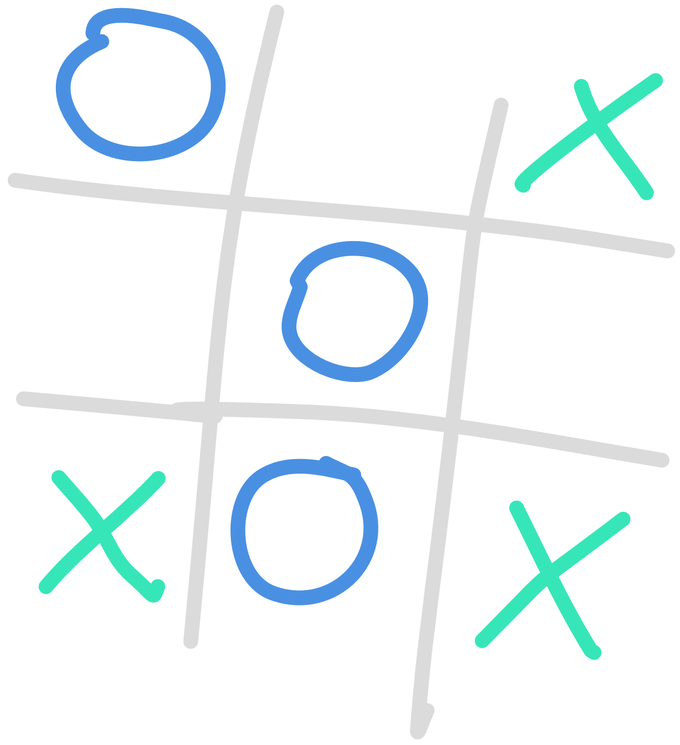
Naive Update Rule for Value Function

$$V(s) \leftarrow V(s) + \alpha(V(s') - V(s))$$

YOUR FIRST RL ALGORITHM

CLONE IT OUT:

https://goo.gl/81mFNM



EXECUTE:

rl/tic_tac_toe.py

MDP IN RL

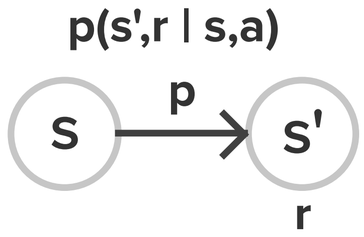
Any RL task with a set of states, actions, and rewards, that follows the Markov property, is an MDP

MDP is defined as the collection of a 5-tuples:

Set of states

Set of actions

Set of rewards



State-transition probabilities, reward probabilities

Discount factor

DISCOUNT FACTOR & RETURN

We would rather get $1K NOW than $10M in 10 years from now.

We would like to maximize total future reward.

The further we look into the future, the harder it is to predict.

Therefore, discount future rewards to make them "matter less".

We can then define the RETURN: $$G(t) = \sum_{\tau=0}^{\infty} \gamma^{\tau} R(t + \tau + 1)$$

POLICY (π)

Technically policy is not part of the MDP itself, but it, along with the value function, forms the solution.

Think of π as shorthand for the algorithm the agent is using to navigate the environment.

Can be represented as a distribution: $\pi(a \mid s)$

STATE-VALUE & ACTION-VALUE

State-value: V(s)

$$V_\pi(s) = E_\pi \left[G(t) \mid S_t = s \right]$$

Action-value: Q(s, a)

$$Q_\pi(s, a) = E_\pi \left[G(t) \mid S_t = s, A_t = a \right]$$

PREDICTION & CONTROL PROBLEMS

Prediction problem: Finding *V* or Q given a fixed policy.

Control problem: Typically use *Q* for learning the optimal policy.

Policy denoted by π, can be *deterministic* or *probabilistic*.

DYNAMIC PROGRAMMING

Pioneered by Richard Bellman, his equations for MDPs allow us to define the value function recursively:

$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \{r + \gamma V_\pi(s')\}$$

If you look carefully, this is actually a system of linear equations:

$$V(s_1) = a_{11} V(s_1) + a_{12} V(s_2) + \ldots + a_{1N} V(s_N)$$
$$V(s_2) = a_{21} V(s_1) + a_{22} V(s_2) + \ldots + a_{2N} V(s_N)$$
$$\ldots$$

Prediction problem: Iterative Policy Evaluation.

Control problem: Policy Iteration & Value Iteration.

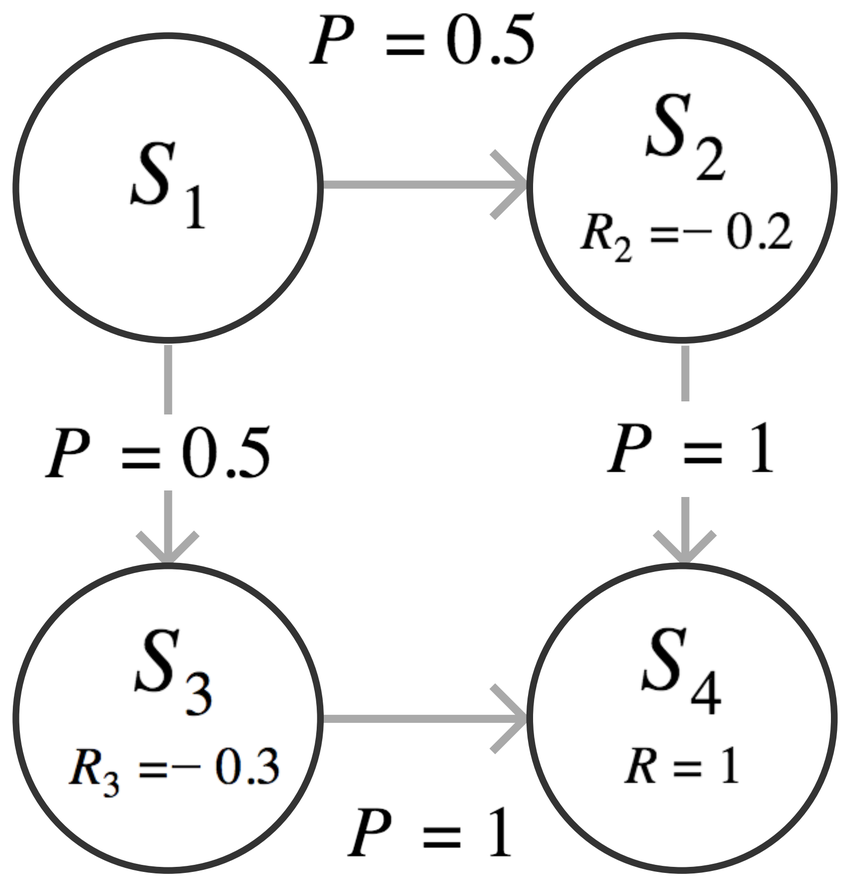
BELLMAN EQUATION EXAMPLE

$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \{r + \gamma V_\pi(s')\}$$

$\gamma = 0.9$ – Discount Factor

$V(S_4) = 0$ – Terminal

$$V(S_3) = p(S_3|S_4) * (R_4 + 0.9 * V(S_4))$$
$$= 1 * (1 + 0.9 * 0) = 1$$



BELLMAN EQUATION EXAMPLE

$$V(S_2) = 1, \quad V(S_3) = 1$$

$$V(S_1) = p(S_2|S_1)[R_2 + \gamma V(S_2)]$$

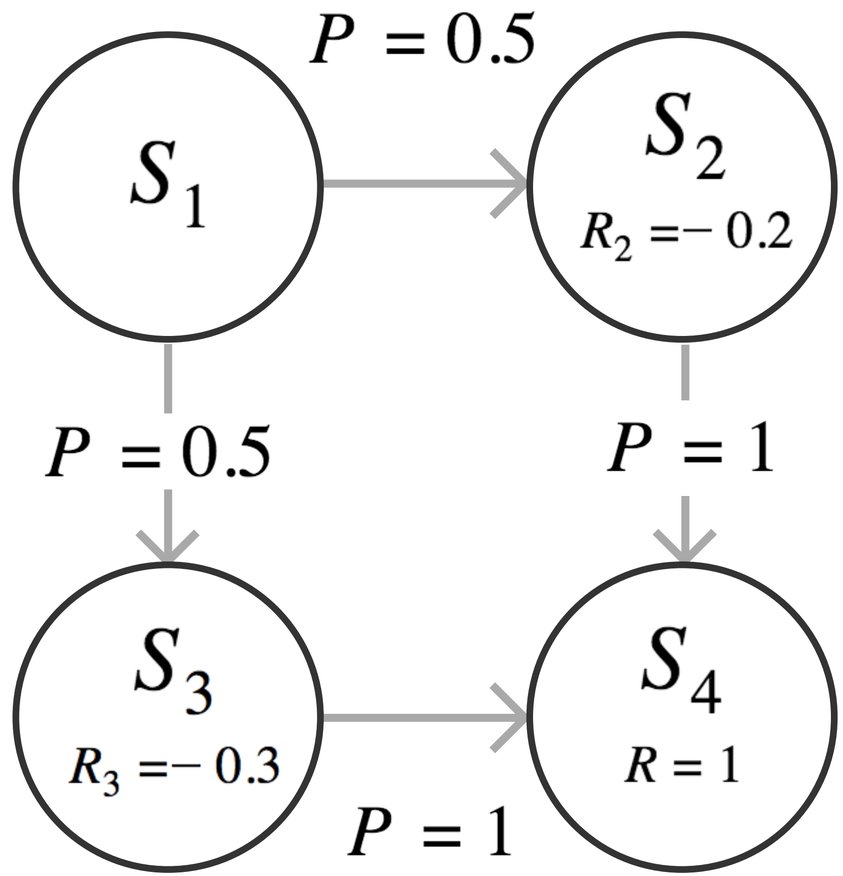
$$+ p(S_3|S_1)[R_3 + \gamma V(S_3)]$$

$$= 0.5(-0.2 + 0.9 * 1) +$$

$$0.5(-0.3 + 0.9 * 1)$$

$$= 0.5 * (-0.2 - 0.3) + 0.9$$

$$= -0.25 + 0.9 = 0.65$$



DYNAMIC PROGRAMMING CONS.

Lays the groundwork, but is not very practical

We need to loop through all states on every iteration

State space may be very large or infinite

Requires us to know p(s',r | s,a)

Calculating that can be infeasible

Doesn't learn from experience

MC and TD learning do, no model of the environment needed

MONTE CARLO (MC)

Unlike DP, MC is all about learning from experience

Expected values can be approximated by sample means

Plays a bunch of episodes, gather returns, average them

$$V(s) = E[G(t) \mid S(t) = s] \approx \frac{1}{N} \sum_{i=1}^{N} G_{i,s}$$

Prediction problem: Using sample means.

Control problem: Similar to Policy Iteration.

MONTE CARLO CONS.

Requires many episodes

Doesn’t work for continuous tasks (e.g. no episodes – never ends)

MC can leave many states unexplored

Requires full control over environment, not always feasible

TEMPORAL DIFFERENCE LEARNING (TDL)

MC: Sample returns based on an episode & need to wait until entire episode is over

TDL: Estimate returns based on current value function estimate

Can learn during the episode, can be used for long or non-episodlc tasks

Only need to wait until *t+1* to update value for state at time *t* so does TRUE online learning

Instead of using G, we use $r + \gamma V(s')$

Prediction problem: $V(s_t) = V(s_t) + \alpha[r + \gamma V(s_{t+1}) - V(s_t)]$

Control problem: Sarsa, Q-Learning

SARSA

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma Q(s',a') - Q(s,a)]$$

$$a' = argmax_a\{\ Q(s',a)\ \}$$

Q-LEARNING

Off-policy algorithm, doesn't matter what action we take next, the target remains the same since it's a max

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma max_{a'} Q(s',a') - Q(s,a)\right]$$

APPROXIMATION METHODS

We know that supervised learning can be used for function approximation

We want to estimate *V* or *Q*

$$Input : x = \varphi(s),\ Target : G$$

Since reward is a real number, so is *G*,

therefore we do regression, and the

appropriate loss is squared error

$$E = \sum_{n=1}^{N} \left(G_n - \hat{V}\left(\varphi(s_n); \theta\right)\right)^2$$

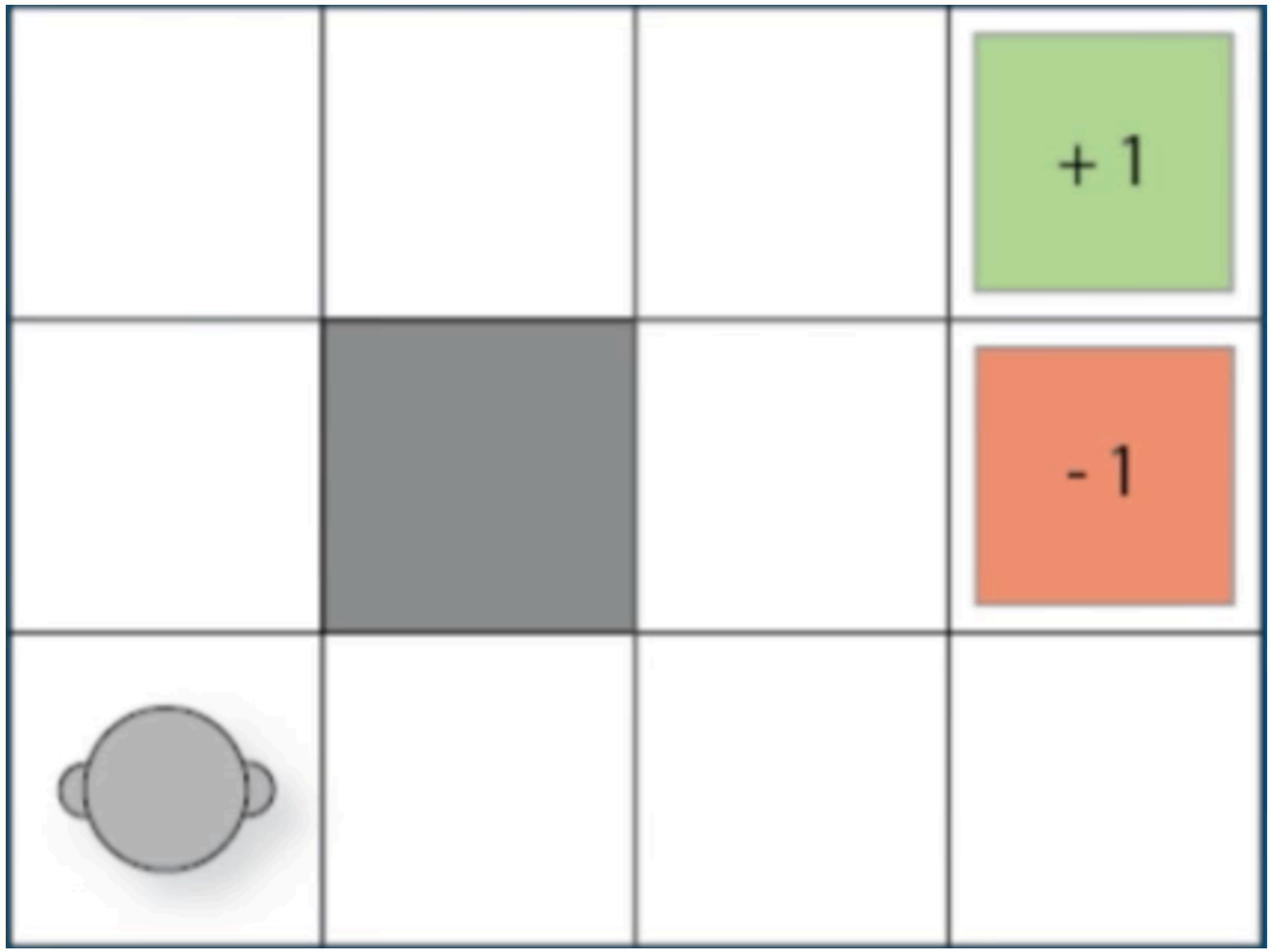
We need a differentiable model

so we can do gradient descent

$$\hat{V}\left(\varphi(s_n); \theta\right) = \theta^T \varphi(s_n)$$

$$\theta = \theta - \alpha \partial E / \partial \theta = \theta + \alpha (G - \hat{V}) \partial V / \partial \theta$$

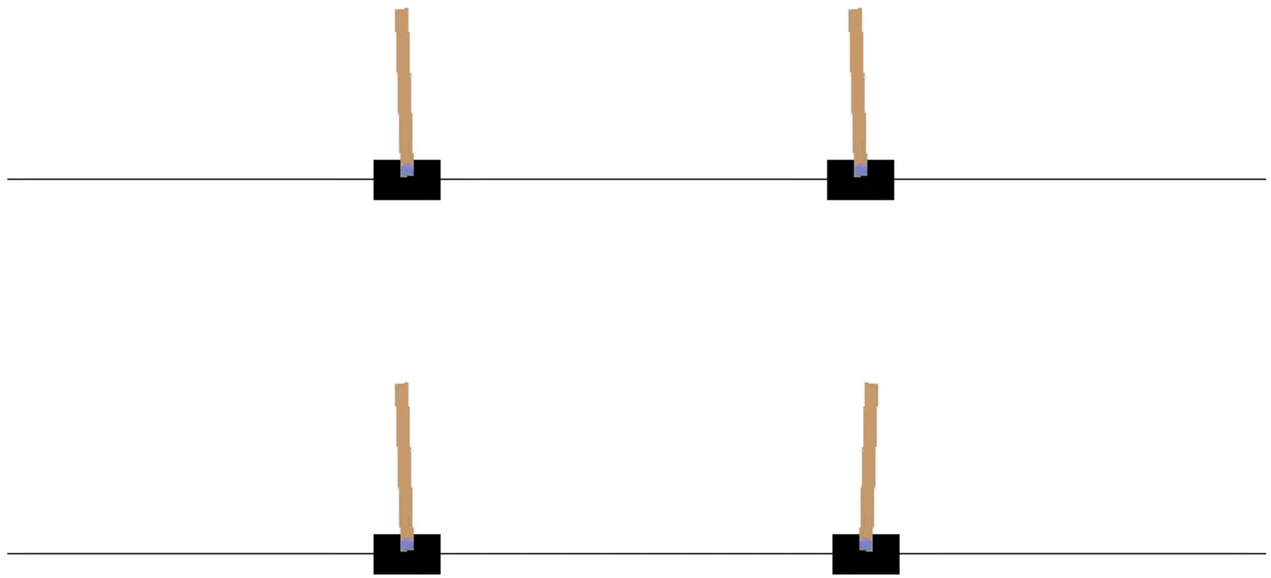
H/T: CHECK ALL METHODS ON GRID WORLD



EXECUTE:

rl/grid_world.py

OPENAI GYM: CART POLE



EXECUTE:

rl2/gym_tutorial.py

rl2/cartpole/

DEEP Q-NETWORKS (DQN)

All we need to know is how to make differentiable models

Use TF or Theano for automatic gradients

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

Deep neural network being used for Q-Learning, e.g. a function approximator for *Q(s, a)*.

Each output node corresponds to a different action *a*

EXPERIENCE REPLAY

Alluded to this multiple times in the past

Can think of MC as a "tiny" step toward experience replay – b/c we obtain *(s,a,G)* triples to train with all at once at the end of an episode.

Here, we use a **replay buffer** [Fancy name for a list that stores *(s,a,r,s')* 4-tuples]

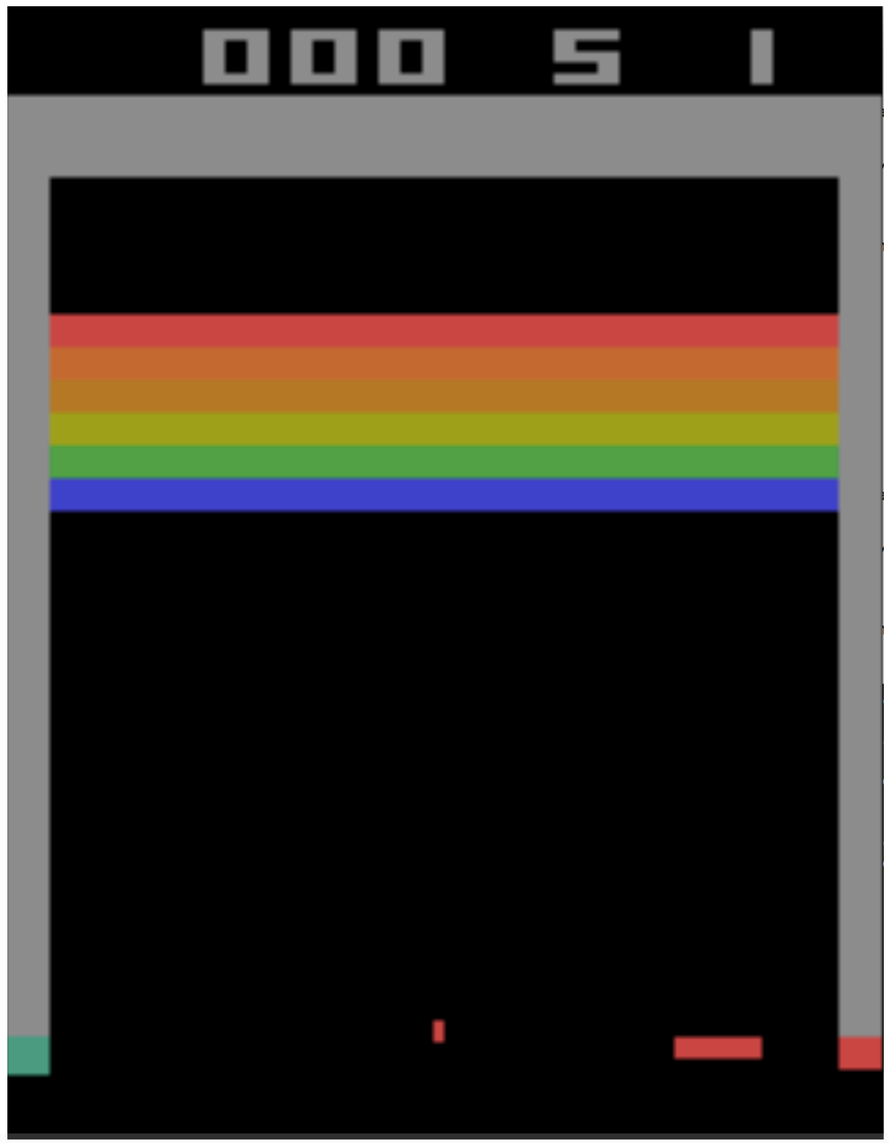
TO TRAIN

Do NOT do SGD wrt – most recent TD error

INSTEAD: sample a random mini-batch from the buffer, do GD wrt the batch

Buffer is a queue: FIFO (first-in first-out)

OPENAI GYM: ATARI



EXECUTE:

rl2/atari/dqn_tf.py

OPENAI GYM: RETRO

https://contest.openai.com/

REFERENCES

OpenAI Research

https://openai.com/research/

OpenAI Retro Contest

https://contest.openai.com

Gotta Learn Fast: A New Benchmark for Generalization in RL

https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/retro-contest/gotta_learn_fast_report.pdf

DeepMind: Deep Reinforcement Learning

https://deepmind.com/blog/deep-reinforcement-learning/

Playing Atari with Deep Reinforcement Learning

https://arxiv.org/pdf/1312.5602.pdf

Variational Deep Q Network

https://arxiv.org/pdf/1711.11225.pdf

Hindsight Experience Replay

https://arxiv.org/pdf/1707.01495.pdf

REFERENCES

Few-Shot Learning with Metric-Agnostic Conditional Embeddings

https://arxiv.org/pdf/1802.04376.pdf

Matching Networks for One Shot Learning

https://arxiv.org/pdf/1606.04080.pdf

REVIEW INSIGHT: Deep Learning

http://pages.cs.wisc.edu/~dyer/cs540/handouts/deep-learning-nature2015.pdf

Improving Information Extraction by Acquiring External Evidence with Reinforcement Learning

https://arxiv.org/pdf/1603.07954.pdf

MIT: Reinforcement Learning: An Introduction

http://incompleteideas.net/book/bookdraft2017nov5.pdf

Berkeley CS294 Lecture Videos: Deep Reinforcement Learning

http://rll.berkeley.edu/deeprlcourse/

A Brief Survey of Deep Reinforcement Learning

https://arxiv.org/pdf/1708.05866.pdf

CONTACTS

fb.com/didur.ivan

linkedin.com/in/ivan-didur/

didur@datarootlabs.com

---

fb.com/datarootlabs

fb.com/dataroot.university

FUNDAMENTAL
AI PARTNER FOR
NEXT-GENERATION
BUSINESSES

DataRoot Labs

ADVANCED
DATA SCIENCE
& **ENGINEERING**
COURSES

DataRoot
University