Intro to (Practical) Reinforcement Learning

Dec 2019 Second Nepal Winter School in AI

Course Format

- 4 x 45min with 3min breaks in between
- Afterwards 1h lab session with quiz + python fun
- Ask questions anytime, interrupt me!

Goal

- Understand basic concepts around RL (S, A, T, R, γ)
- Understand basic algorithms (policy iteration, SARSA, etc.)
- Be able to use DRL at a grad student level*
- NOT: understand SotA algorithms / create new SotA

* i.e. be able to download somebody else's algorithm and run it on your task

Outline

Part 1 - Intro & MDPs
(Examples, Markov stuff)

Part 2 - RL for Evaluation
(Policy Evaluation, TD(0))

Part 3 - Model-Free RL for Control
(Q learning / SARSA)

Part 4 - Practical RL
(OpenAI Gym, SotA algorithms, etc.)

Part 1 - Introduction + MDP

Why RL tho?

Task (e.g. is this image a cat?)

Supervised Learning

Answer:
Yes, it’s a cat

No, it’s a rabbit
(Ground Truth)

MSE/BCE

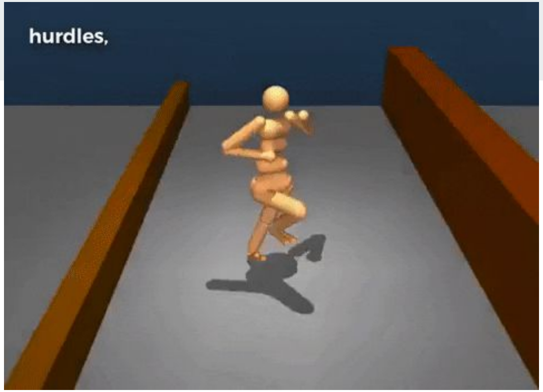
Reinforcement Learning

Answer:
Yes, it’s a cat

That answer
was incorrect

Policy loss

Why RL tho?



Task (e.g. make this robot stand up & walk)

Supervised Learning

Answer: Some movement

Lol, IDK!

MSE/BCE

Reinforcement Learning

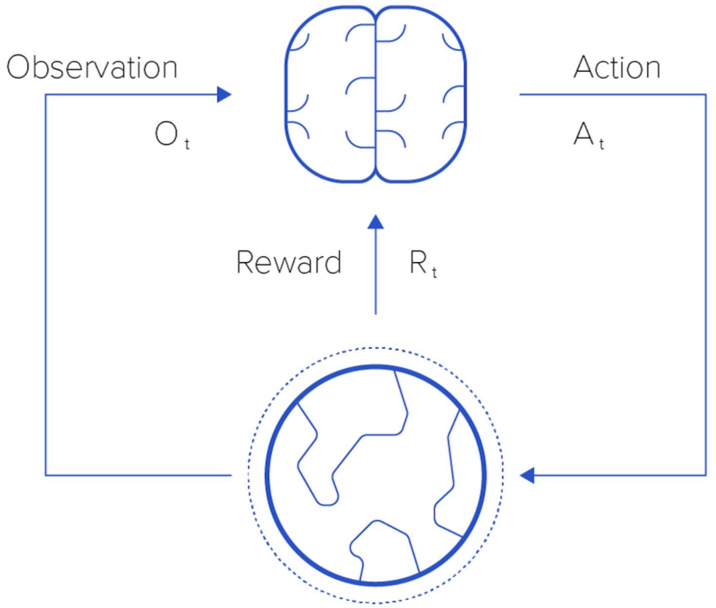
Answer: Some movement

Warmer, warmer!

Policy loss

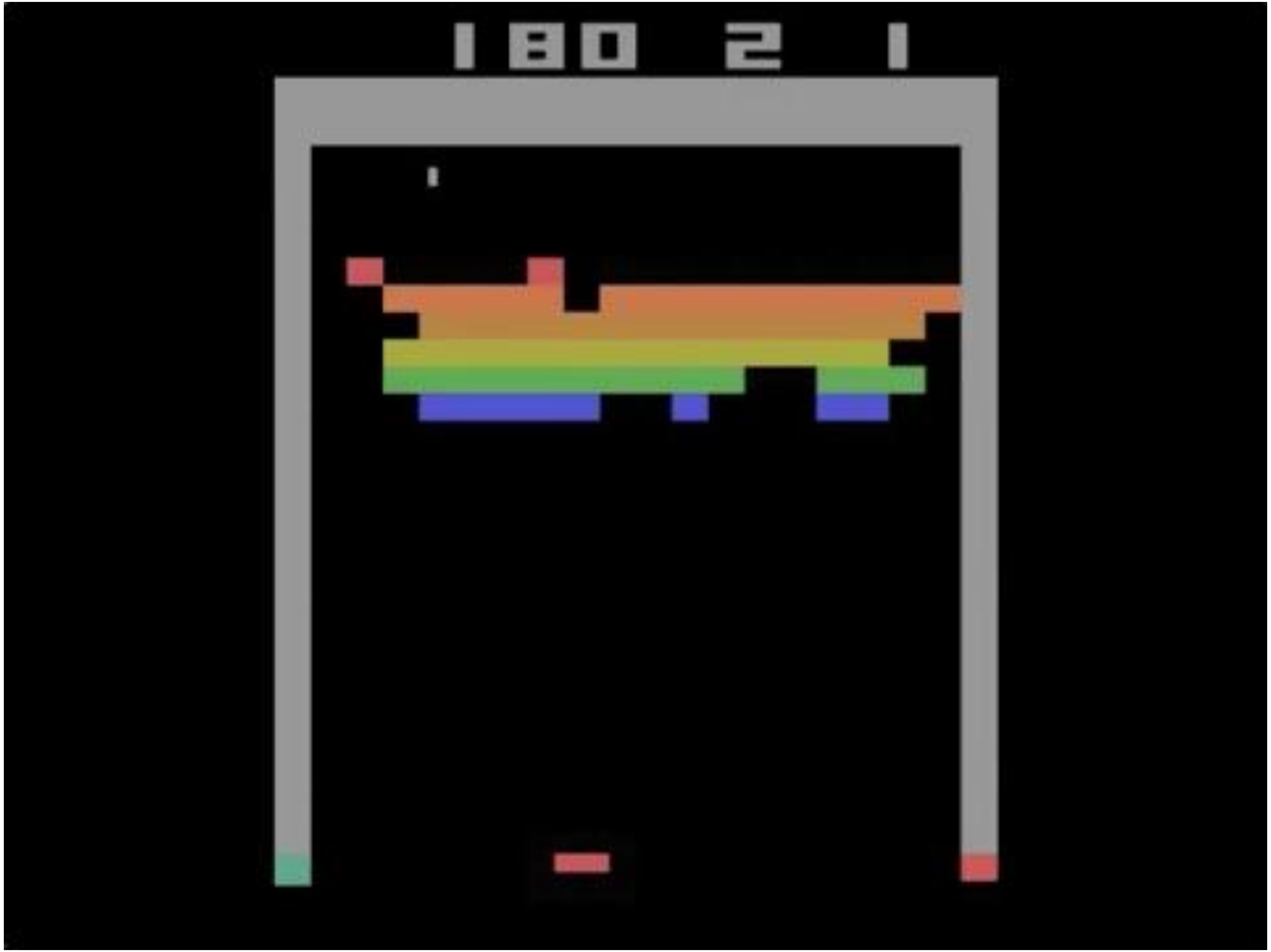
Source: Deepmind, https://www.youtube.com/watch?v=gn4nRCC9TwQ

RL Loopdy Loop

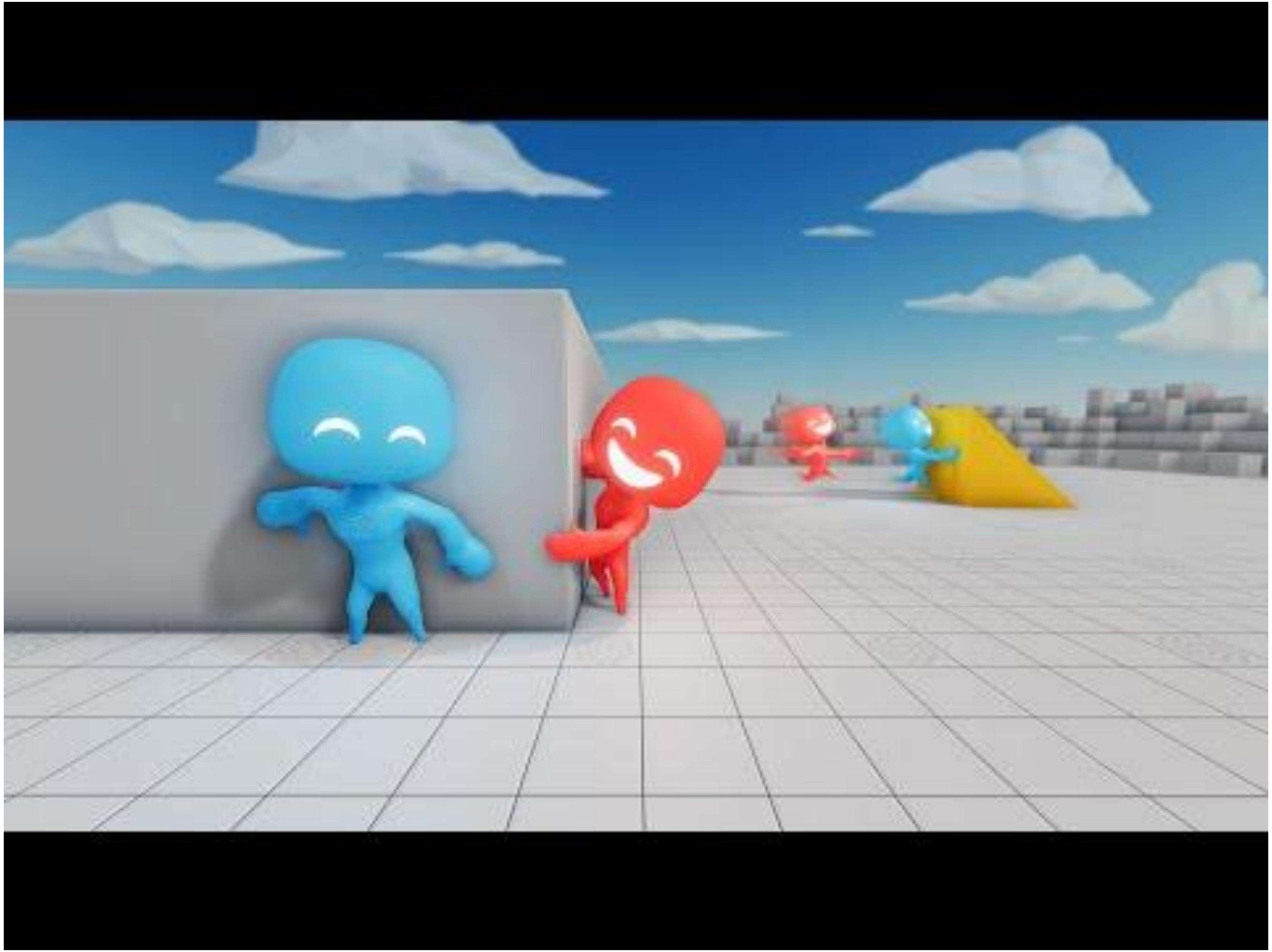


What else can RL do?

Source: Stanford, https://www.youtube.com/watch?v=0JL04JJjocc



Source: Deepmind, https://www.youtube.com/watch?v=TmPfTpjtdgg



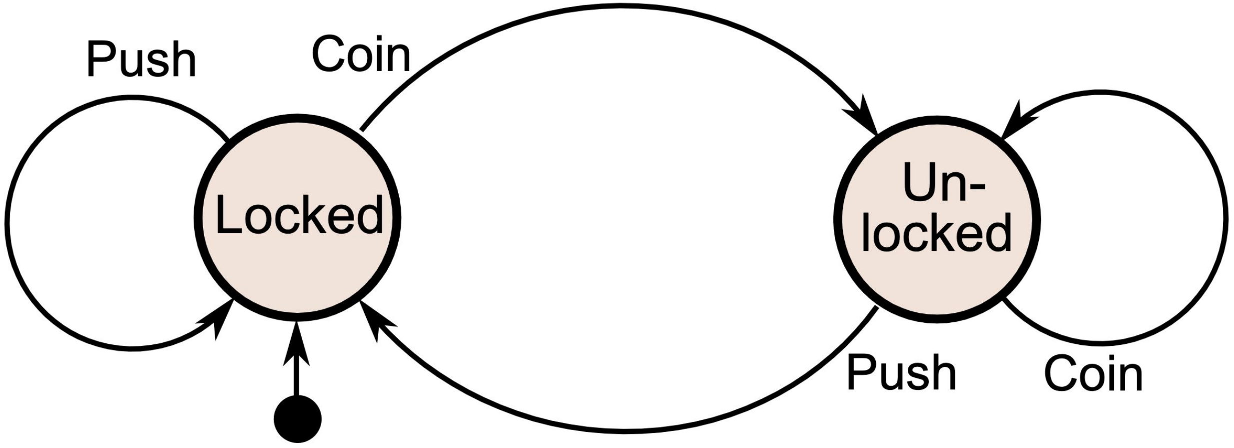
Source: OpenAI, https://www.youtube.com/watch?v=kopoLzvh5jY

Resources

- Youtube, “RL Course by David Silver”, 11 lectures x 1h30: https://www.youtube.com/playlist?list=PLzuuYNsE1EZAXYR4FJ75jcJseBmo4KQ9-
- Youtube, Abbeel & Klein, http://ai.berkeley.edu/lecture_videos.html
- Sutton & Barto “Reinforcement Learning: An Introduction”: http://incompleteideas.net/book/bookdraft2017nov5.pdf
- Spinning Up in DRL: https://spinningup.openai.com/en/latest/

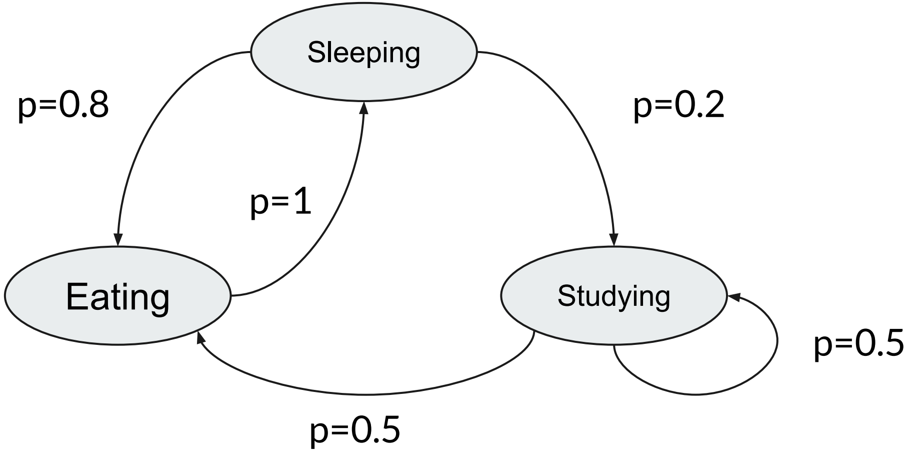
Some slides/formulas were copied from the above

Finite State Machine

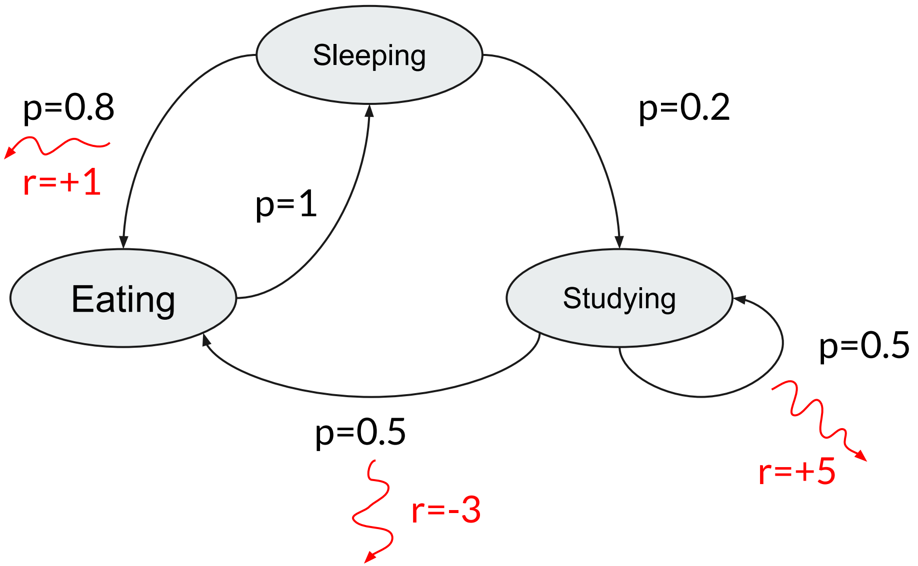


Source: Wikipedia,
https://en.wikipedia.org/wiki/Finite-state_machine?oldformat=true

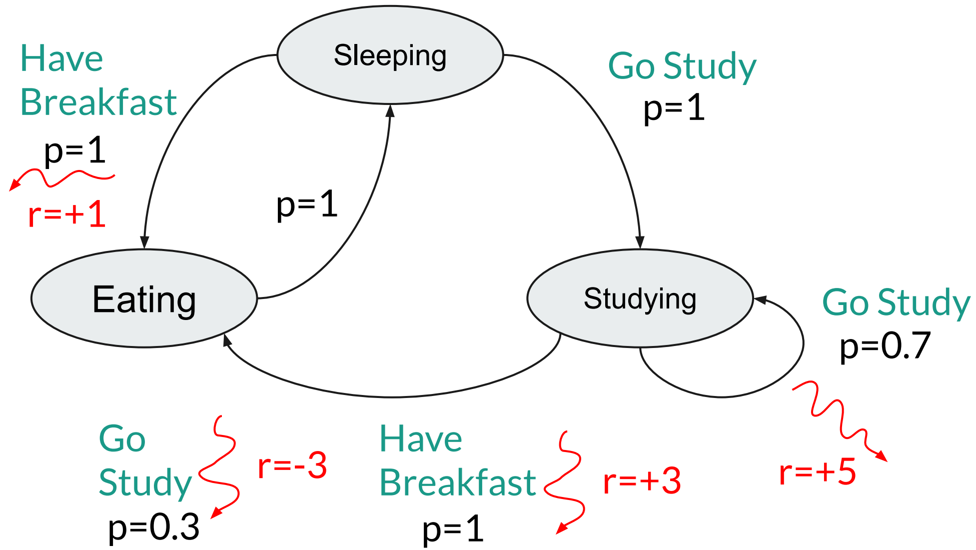
Markov Chain



Markov Reward Process

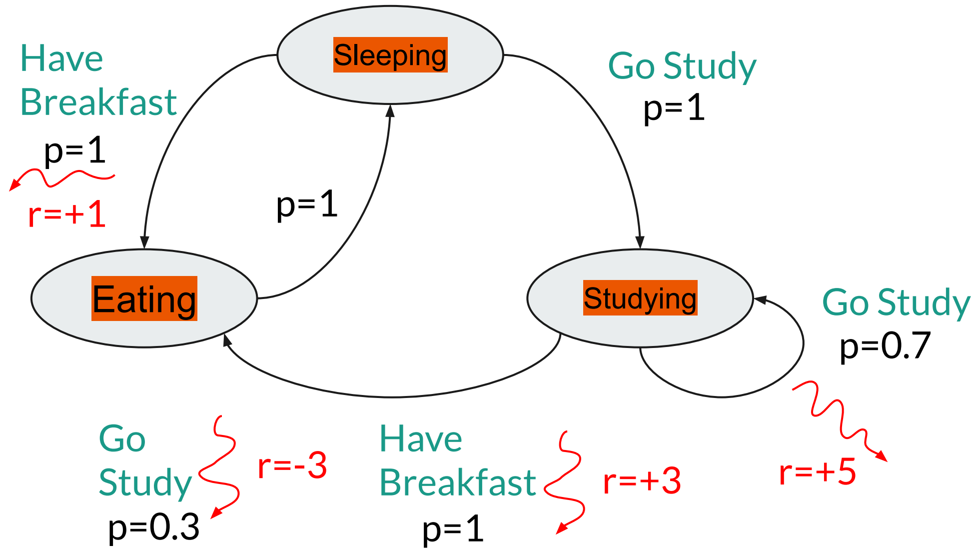


Markov Decision Process



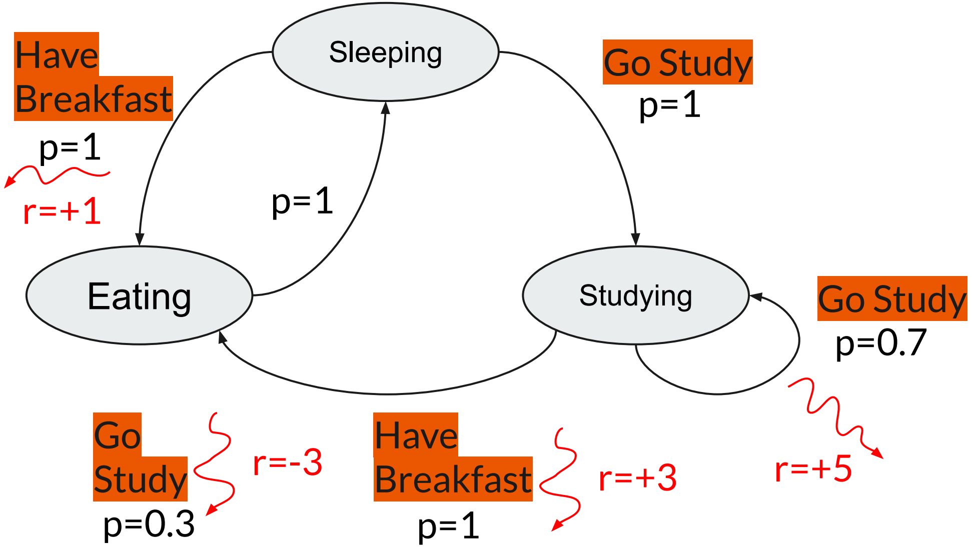
Markov Decision Process

- States (S)



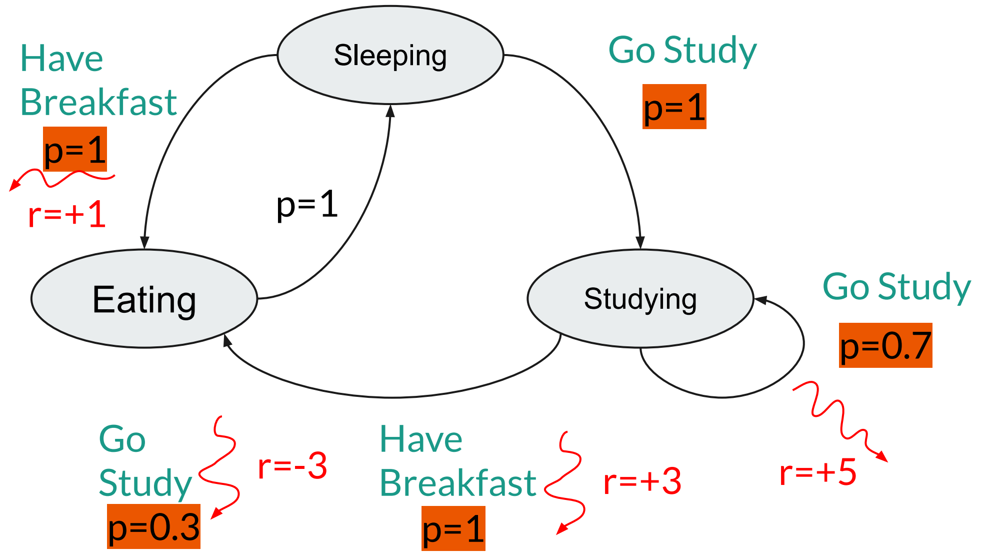
Markov Decision Process

- States (S)
- Actions (A)



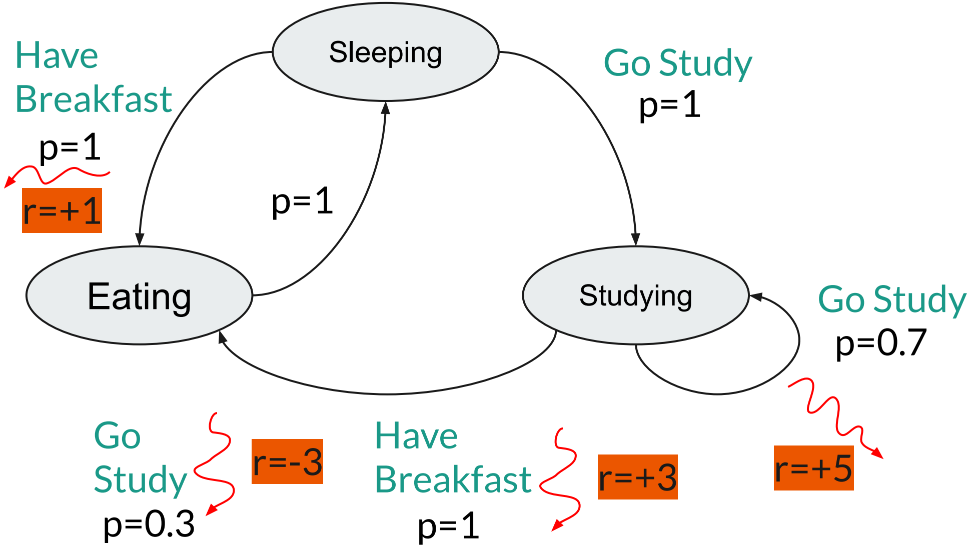
Markov Decision Process

- States (S)
- Actions (A)
- Transition Probabilities (T/P)



Markov Decision Process

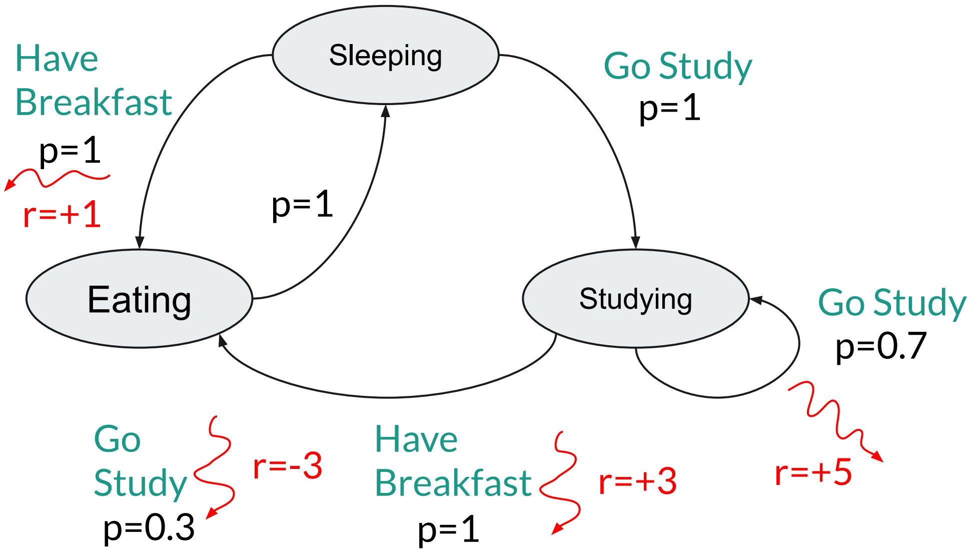
- States (S)
- Actions (A)
- Transition Probabilities (T/P)
- Rewards (R)



Markov Decision Process

- States (S)
- Actions (A)
- Transition Probabilities (T/P)
- Rewards (R)
- Discount Factor (γ)

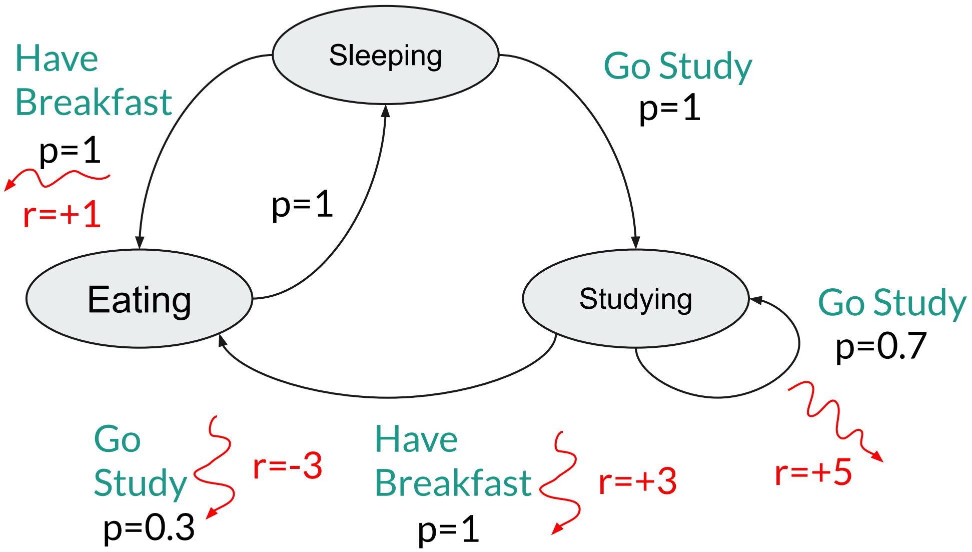
$$(S, A, T, R, \gamma)$$



Markov Decision Process

- States (S)
- Actions (A)
- Transition Probabilities (T/P)
- Rewards (R)
- Discount Factor (γ)

$$(S, A, T, R, \gamma)$$

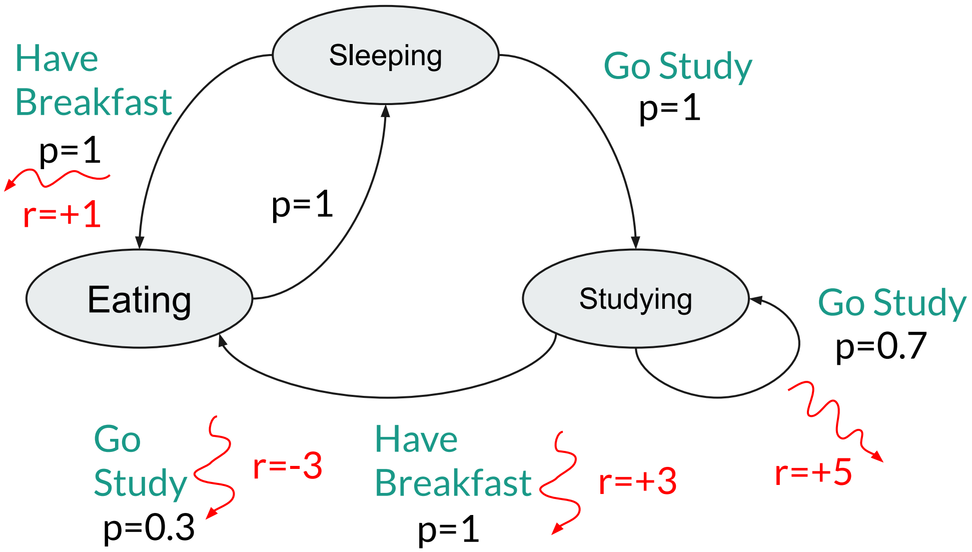


$\pi(s) \rightarrow a$

Markov Decision Process

- States (S)
- Actions (A)
- Transition Probabilities (T/P)
- Rewards (R)
- Discount Factor (γ)

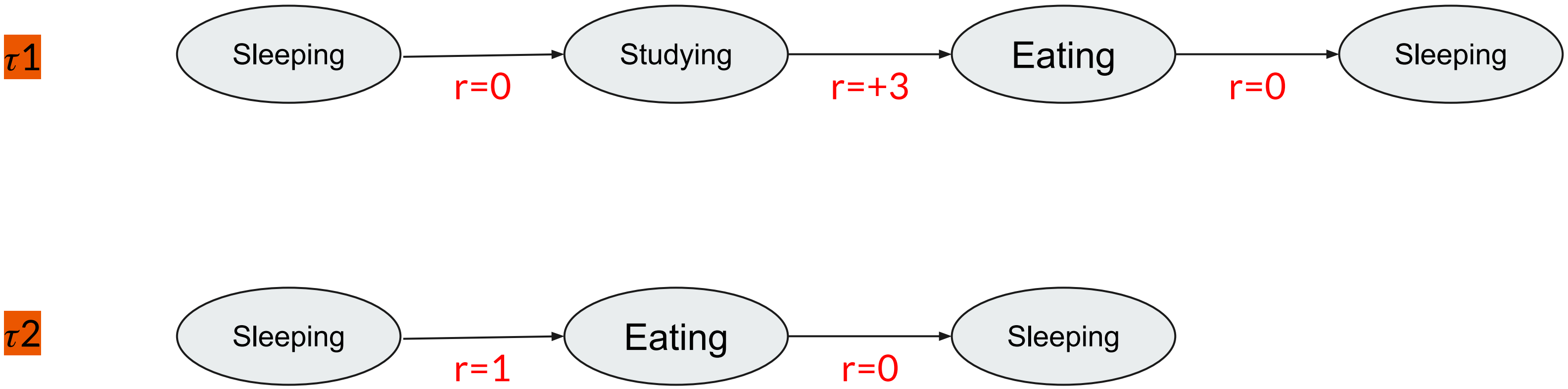
$$(S, A, T, R, \gamma)$$



$\pi(s) \rightarrow a$

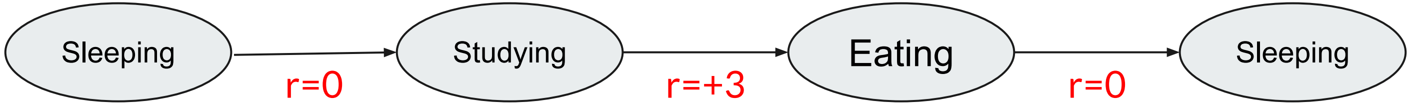
Deterministic or stochastic environment?

Discount Factor & Return



$$G_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Discount Factor & Return



If γ=1: G = 1*0 + 1*3 + 1*0 = 3

If γ=1: G = 1*1 + 1*0 = 1

Source (formula): David Silver reinforcement learning lecture series, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

$$G_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Discount Factor & Return

$$\text{Sleeping} \xrightarrow{r=0} \text{Studying} \xrightarrow{r=+3} \text{Eating} \xrightarrow{r=0} \text{Sleeping}$$

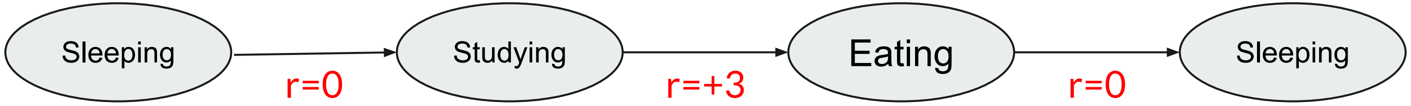
If γ=0: G = ???

$$\text{Sleeping} \xrightarrow{r=1} \text{Eating} \xrightarrow{r=0} \text{Sleeping}$$

If γ=0: G = ???

$$G_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Discount Factor & Return



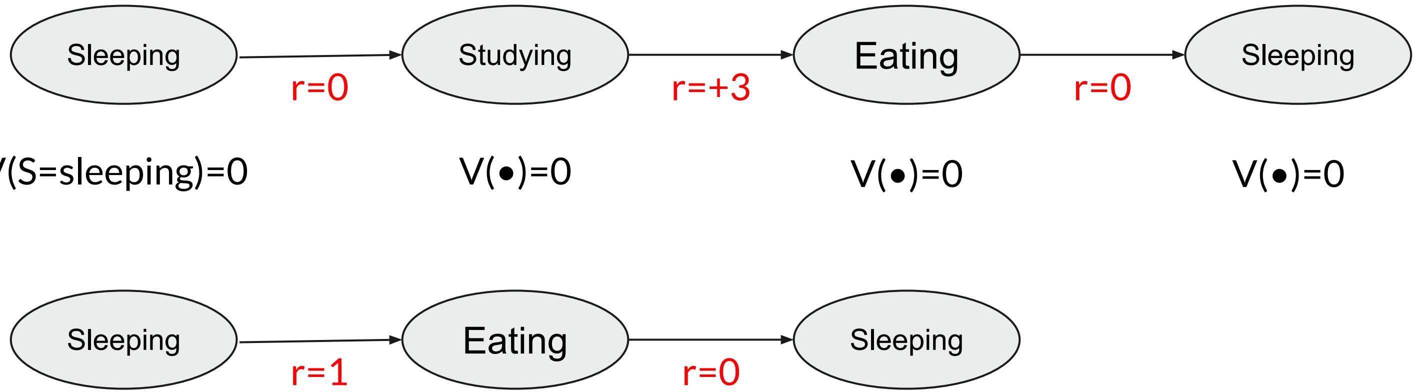
If γ=0: G = (0^0)*0 + (0^1)*3 + (0^2)*0 = 0

If γ=0: G = (0^0)*1 + (0^1)*0 = 1

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s] = \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s]$$

"How good is a state?"

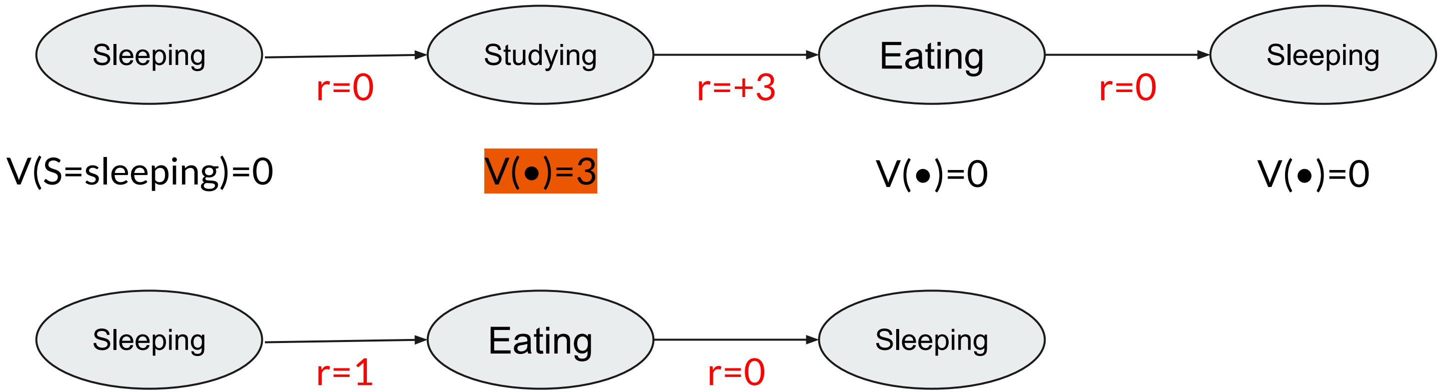
Value Function



Source (formula): David Silver reinforcement learning lecture series, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

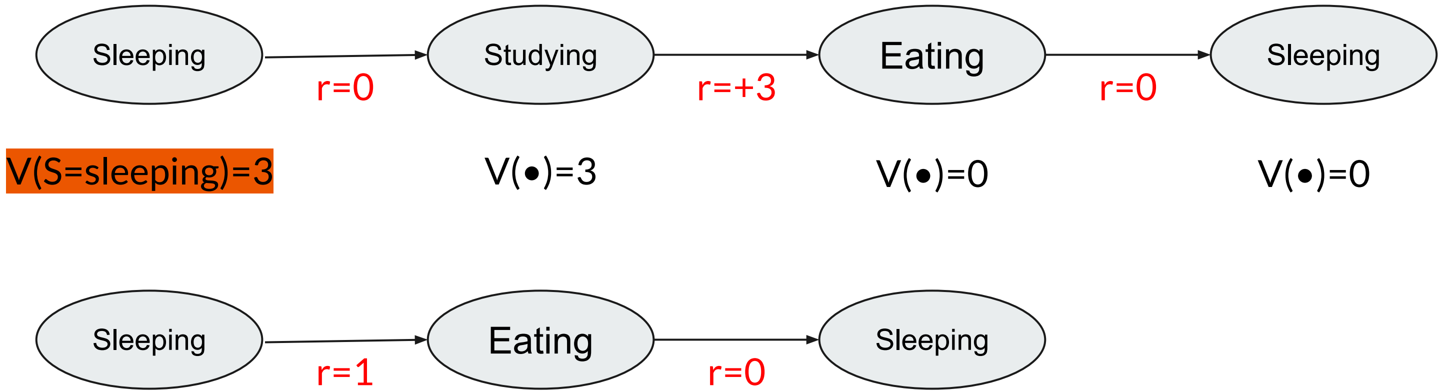
$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s] = \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s]$$

Value Function



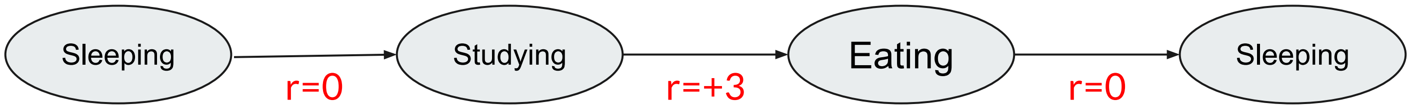
$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s] = \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s]$$

Value Function



$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s] = \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s]$$

Value Function

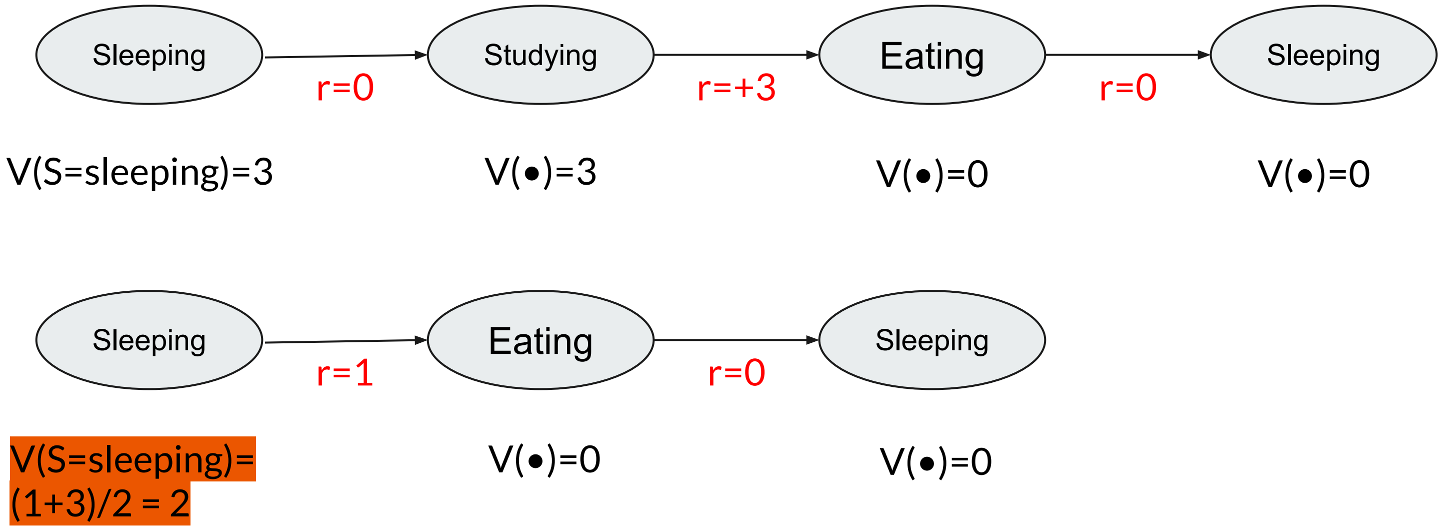


V(S=sleeping)=3 V(•)=3 V(•)=0 V(•)=0

V(S=sleeping)=3 V(•)=0 V(•)=0

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s] = \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s]$$

Value Function



Bellman Equation

$$\boxed{V^{\pi}(s)} = \boxed{R(s, \pi(s))} + \gamma \boxed{\sum_{s'}} \boxed{T(s, \pi(s), s')} \boxed{V^{\pi}(s')}$$

Value of s

Immediate reward

Probability of
reaching s’
from s under π

Value of s’

Source (formula): Guillermo Garcia-Hernando,
https://drive.google.com/open?id=1kVfE4NWXWFdWo3U3sEK_oQezoTtFqfXS

Part 2 - Model-based RL

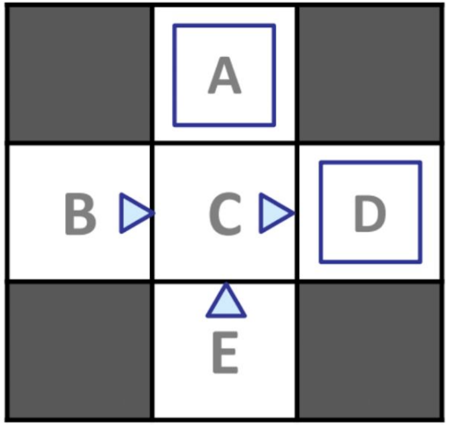
Model-based RL Idea

1. Build (or have) a model of the environment dynamics: transitions (T), rewards (R)
2. “Solve” the environment: via planning or value estimation
3. …
4. Profit

Learning the Model

T & R can be learned by averaging observations from trajectories.

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$$\hat{T}(s, a, s')$$

T(B, east, C) = $\boxed{}$
T(C, east, D) = $\boxed{}$
T(C, east, A) = $\boxed{}$
...

$$\hat{R}(s, a, s')$$

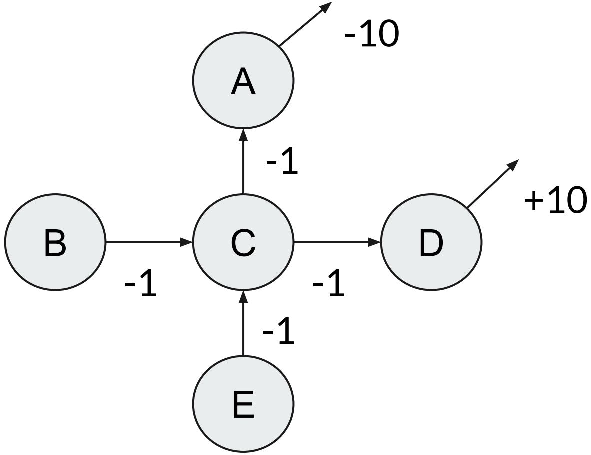
R(B, east, C) = $\boxed{}$
R(C, east, D) = $\boxed{}$
R(D, exit, x) = $\boxed{}$
...

Policy Evaluation

(Goal: get value function everywhere)

1. Initialize VF everywhere at 0
2. Iterate over all states; update their value with the reward from all reachable states + their current value function
3. GOTO 2

Policy Evaluation

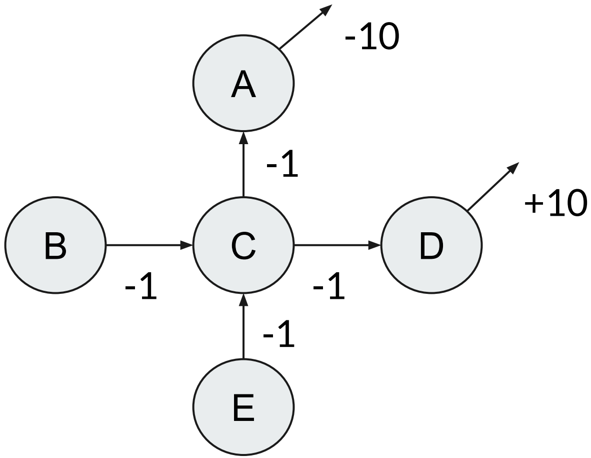


	0	
0	0	0
	0	

Assume T(•) = 1, γ = 1,
uniform random policy

Iteration 0

Policy Evaluation

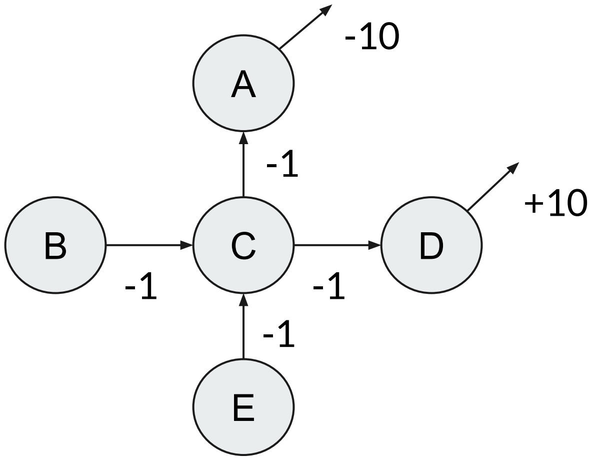


Assume T(•) = 1, γ = 1,
uniform random policy

	-10	
-1	-1	+10
	-1	

Iteration 1

Policy Evaluation

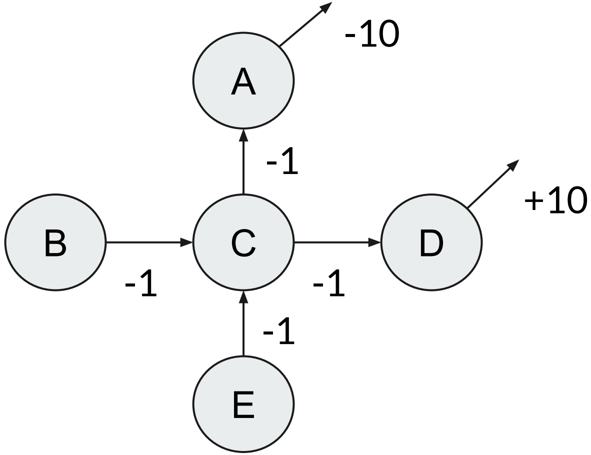


Assume T(•) = 1, γ = 1,
uniform random policy

	-10	
-1+(-1)	½ (-1-10)+½ (-1+10) = -1	+10
	-1+(-1)	

Iteration 2

Policy Evaluation

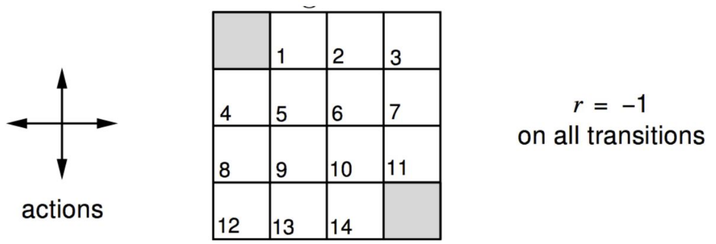


Assume T(•) = 1, γ = 1,
uniform random policy

	-10	
-2	-1	+10
	-2	

Iteration 2

And another one



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, \ldots, 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

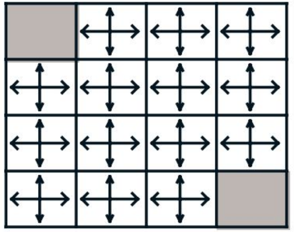
$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

Source: David Silver reinforcement learning lecture series, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

v_k for the Random Policy | Greedy Policy w.r.t. v_k

$k = 0$

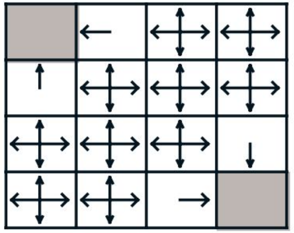
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

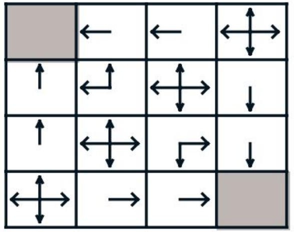


¼ * (-1 + 0) +
¼ * (-1 -1) +
¼ * (-1 -1) +
¼ * (-1 -1)

Chance of going there under current policy * (Reward of transition + V(s'))

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

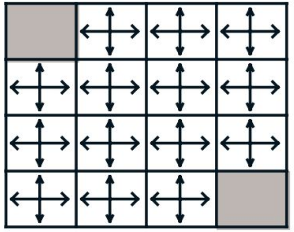


Source: David Silver reinforcement learning lecture series, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

v_k for the Random Policy	Greedy Policy w.r.t. v_k

$k = 0$

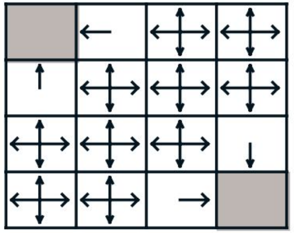
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

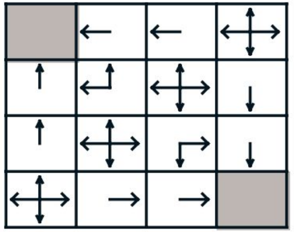
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



$k = 3$

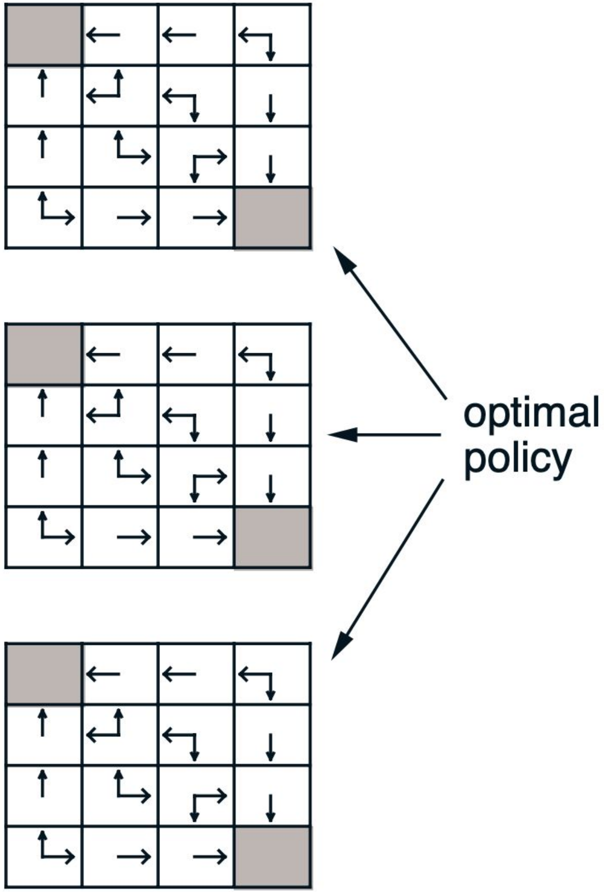
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

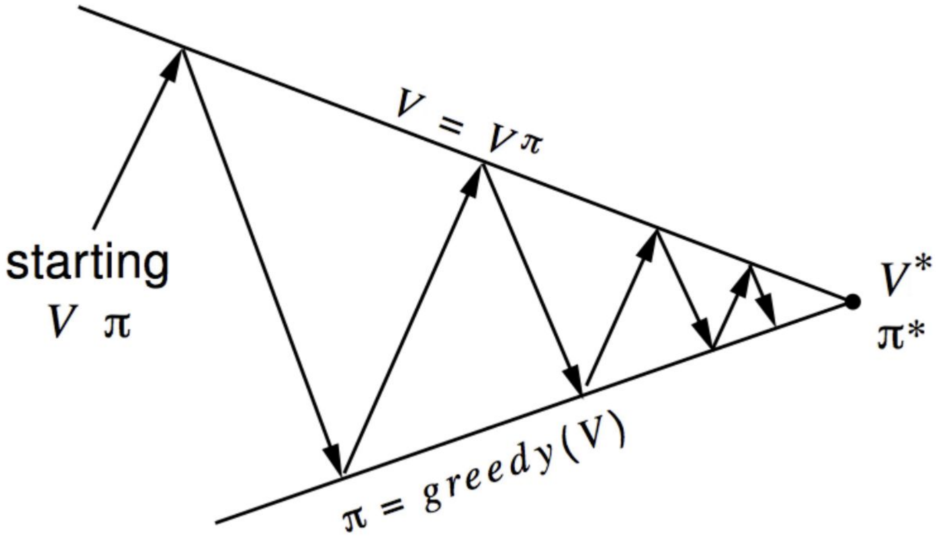
$k = 10$

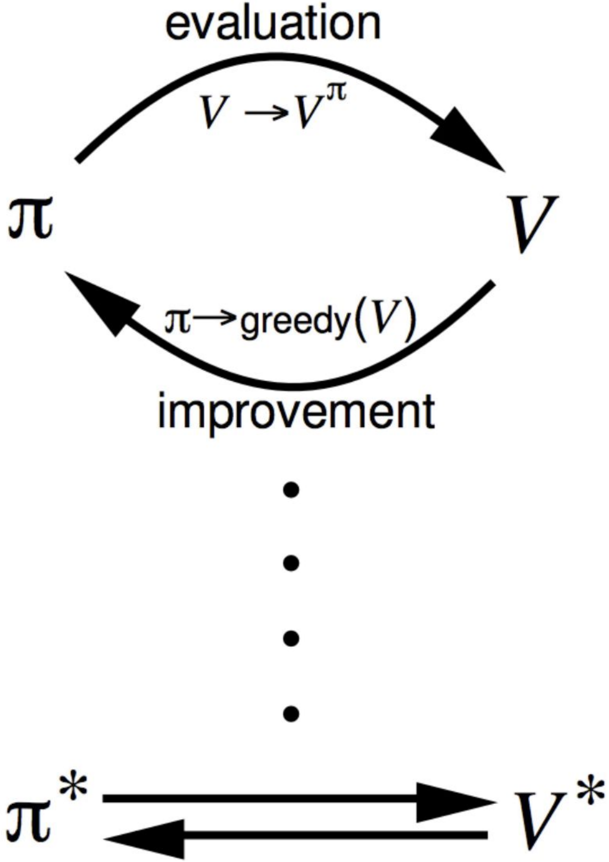
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0







Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

Source: David Silver reinforcement learning lecture series, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

Problems:

- Costly (synchronous update of all states + every state vs. every accessible state)
- Need transition function

$\longrightarrow$ how about asynchronous updates as we go? (MC/TD(0))

Part 2 - Model-free RL

Monte-Carlo Policy Evaluation

Idea:

- When an episode is over, store actual mean return (G) for each state
- Update value function to approximate this G for each state

Learning Rate

$$V(S_t) \leftarrow V(S_t) + \boxed{\alpha}(G_t - V(S_t))$$

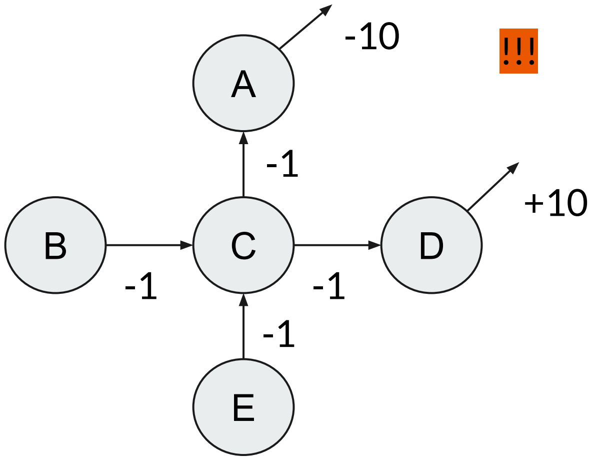
Source: David Silver reinforcement learning lecture series,
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

Incremental Updates

New value = old value + learning rate * (measurement - old value)

If measurement == old value, then no change,
otherwise small increase/decrease

MCPE



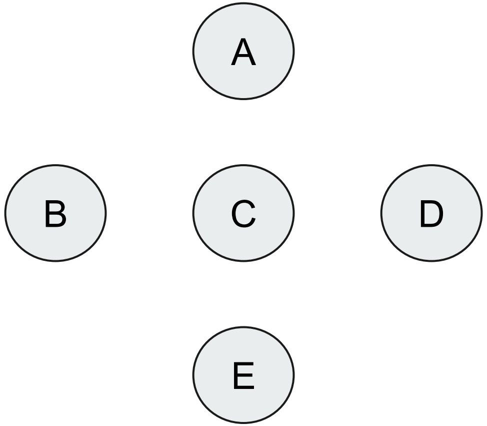
Assume $\gamma = 1$, $\alpha = 0.1$

	0	
0	0	0
	0	

Iteration 0

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

MCPE



	0	
0	0	0
	0	

Trajectory 1

B→C, -1
C→D, -1
D→x,+10

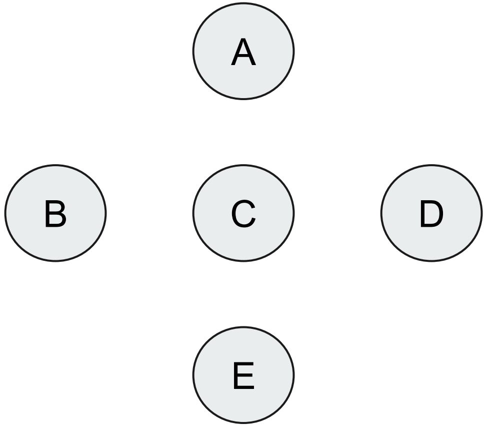
G(B): +8
G(C): +9
G(D): +10

Assume γ = 1, α = 0.1

Iteration 0

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

MCPE



	0	
0+0.1*8 =.8	0+0.1*9 =.9	0+0.1*10 =1
	0	

Trajectory 1

B→C, -1
C→D, -1
D→x,+10

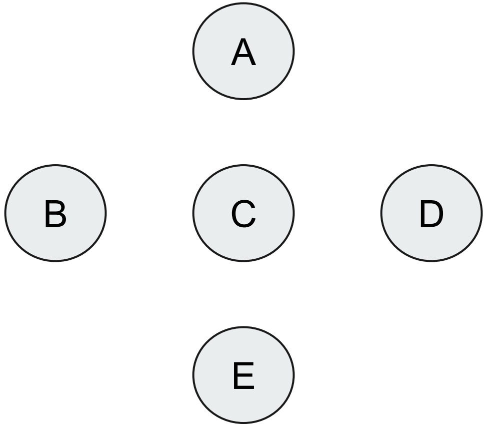
G(B): +8
G(C): +9
G(D): +10

Assume γ = 1, α = 0.1

Iteration 1

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

MCPE



	0	
.8	.9	1
	0	

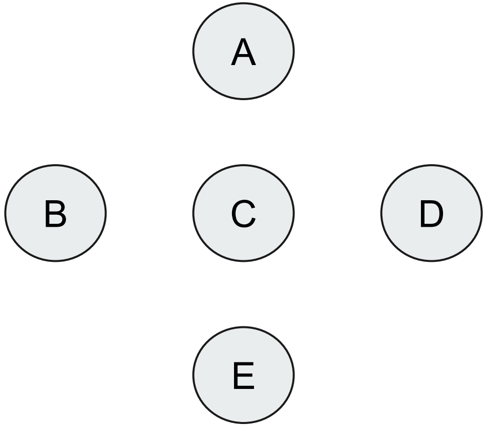
Trajectory 1	Trajectory 2
B→C, -1	B→C, -1
C→D, -1	C→A, -1
D→x,+10	A→x,-10
G(B): +8	G(B): -12
G(C): +9	G(C): -11
G(D): +10	G(A): -10

Assume $\gamma = 1$, $\alpha = 0.1$

Iteration 1

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$

MCPE



Assume γ = 1, α = 0.1

	0+.1*-10 =-.1	
.8+.1* (-12-.8) = -0.48	.9+.1* (-11-.9) = -0.29	1
	0	

Iteration 2

Trajectory 1	Trajectory 2
B→C, -1 C→D, -1 D→x,+10	B→C, -1 C→A, -1 A→x,-10
G(B): +8 G(C): +9 G(D): +10	G(B): -12 G(C): -11 G(A): -10

Temporal Difference Learning

Idea:

- Same thing but we don’t wait for the episode’s end
- Use single step (reward+V(s’) to update → single step = TD(0)

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

$$\downarrow$$

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$

Source: David Silver reinforcement learning lecture series,
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

Part 3 - RL for Control

Summary so far

- Learned about MDP/(S,A,T,R,γ)
- Policy Eval learns value func. (given T,R,π)
- Monte-Carlo Policy Eval learns value func (given π)
- Temporal Difference Learning learns value func (given π)
- Can use greedy π if we have T, but what if we don't?

→ Q function to the rescue

Q Learning

Similar to value function, but also taking actions into consideration:

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

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

Source: David Silver reinforcement learning lecture series, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

Q Learning Policy

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ q_\pi(s, a)$$

At each state, check the Q value of all the actions; Pick action with highest Q

Source: David Silver reinforcement learning lecture series, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

How to learn Q?

- Use Monte-Carlo algorithm ("Monte-Carlo Q Learning")
- Use Temporal Difference algorithm ("Sarsa") - same as TD(0) but with Q function instead of value function

Monte-Carlo Q Learning

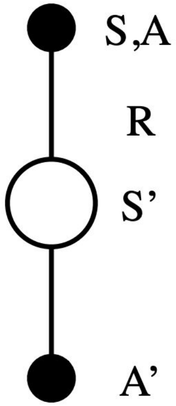
- For each state S_t and action A_t in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}\left(G_t - Q(S_t, A_t)\right)$$

Simple counter (start at 0)

Sarsa - State-action-reward-state-action



$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma Q(S',A') - Q(S,A) \right)$$

Source: David Silver reinforcement learning lecture series,
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

BUT: exploration

"Behind one door is tenure - behind the other is flipping burgers at McDonald's."

- There are two doors in front of you.
- You open the left door and get reward 0
  $V(left) = 0$
- You open the right door and get reward $+1$
  $V(right) = +1$
- You open the right door and get reward $+3$
  $V(right) = +2$
- You open the right door and get reward $+2$
  $V(right) = +2$

$$\vdots$$

- Are you sure you've chosen the best door?

Source: David Silver reinforcement learning lecture series, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

ϵ-Greedy Exploration

- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

You can decrease ϵ slowly as you go

Source: David Silver reinforcement learning lecture series, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html,

How to Neural-network this?

- Q function: a neural network, in: observation, action, out: scalar float
- π: testing each action + picking highest Q value
- “Actor-Critic” = Q network + policy network

Continuous actions:

- $Q(s,a) = Q(s, \pi(s)) \longrightarrow$ backprop through Q into both nets

Part 4 - Practical RL

OpenAI gym

- De-facto standard RL environment(s)
- Contains variety of tasks (text adventures, Atari games, robotic tasks...)

→ show https://gym.openai.com/envs/#classic_control

- Only tasks, no learning algorithms
- Homogenous API

```python
import gym

env = gym.make("Pendulum-v0")

obs = env.reset()

# optional
env.render()

done = False

while not done:
    # in practice the action comes from your policy
    action = env.action_space.sample()

    obs, rew, done, misc = env.step(action)

    # optional
    env.render()

```

```python
obs, rew, done, misc = env.step(action)
```

Observation as
List, Tuple, Numpy array

Ex: img,
`np.array((128,128,3), dtype=np.uint8)`

Ex: robot joints + velocities
`list(0.4, 1.0, -0.3, 0.0)`

Scalar float

Ex: +10
Ex: -0.001

Bool

Ex: True
Ex: False

Dictionary

Ex: `{}`

Ex:
`{“success”: True, “steps”: 420}`

Ex:
`{“reward_pos”: 69, “reward_vel”: 1, “reward_rules”: -10.4}`

```python
import gym

env = gym.make("Pendulum-v0")

policy = Policy() # <-- not part of Gym
replay_buf = ReplayBuffer() # <-- also not part of Gym

while True:

    obs = env.reset()
    done = False

    while not done:

        action = policy.select_action(obs)

        new_obs, rew, done, misc = env.step(action)

        replay_buf.add((obs, action, new_obs, rew, done))

    replay_batch = replay_buf.sample()
    policy.train(replay_batch)

```

Find the error!

Best Practices

- Normalize observations & actions to be in [-1,1] or [0,1]:

  `np.array(-100, 5, 30) → np.array(-1, 0.05, .3)`

  (normalize by max/range or by mean/std)

Best Practices

- Normalize observations & actions to be in [-1,1] or [0,1]:

  `np.array(-100, 5, 30) → np.array(-1, 0.05, .3)`

  (normalize by max/range or by mean/std)

- Limit/scale rewards

  -100 on failure, +1 on success, -0.00001 per step → -5, +1, -0.01

Best Practices

- Normalize observations & actions to be in [-1,1] or [0,1]:

  `np.array(-100, 5, 30) → np.array(-1, 0.05, .3)`

  (normalize by max/range or by mean/std)

- Limit/scale rewards

  -100 on failure, +1 on success, -0.00001 per step → -5, +1, -0.01

- Make sure environment is markovian

Best Practices

- Normalize observations & actions to be in [-1,1] or [0,1]:

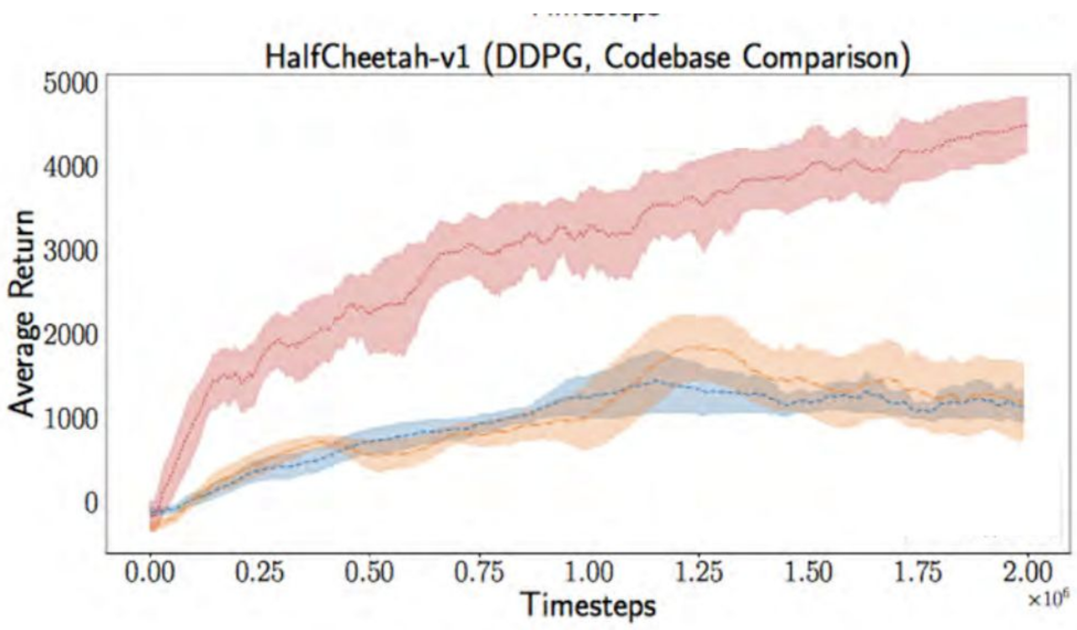
  `np.array(-100, 5, 30) → np.array(-1, 0.05, .3)`

  (normalize by max/range or by mean/std)
- Limit/scale rewards

  -100 on failure, +1 on success, -0.00001 per step → -5, +1, -0.01
- Make sure environment is markovian

And
LOTS OF SEEDS

Reproducibility



Source: Joelle Pineau's NeurIPS 2018 talk,
https://media.neurips.cc/Conferences/NIPS2018/Slides/jpineau-NeurIPS-dec18-fb.pdf

Markov property

Example:

Flying a helicopter

Observation:
`(position_xyz, velocity_xyz)`

Markov property

Example:

Flying a helicopter

Observation:
`(position_xyz, velocity_xyz)`

Better:
```
(position_xyz, velocity_xyz,
rotation_quat, goal_xyz)
```

Markov property

Example:

Flying a helicopter

Observation:
`(position_xyz, velocity_xyz)`

Better:
`(position_xyz, velocity_xyz, rotation_quat, goal_xyz)`

Example:

Driving a car

Observation:
`Image, (256, 256, 3)`

Markov property

Example:

Flying a helicopter

Observation:
`(position_xyz, velocity_xyz)`

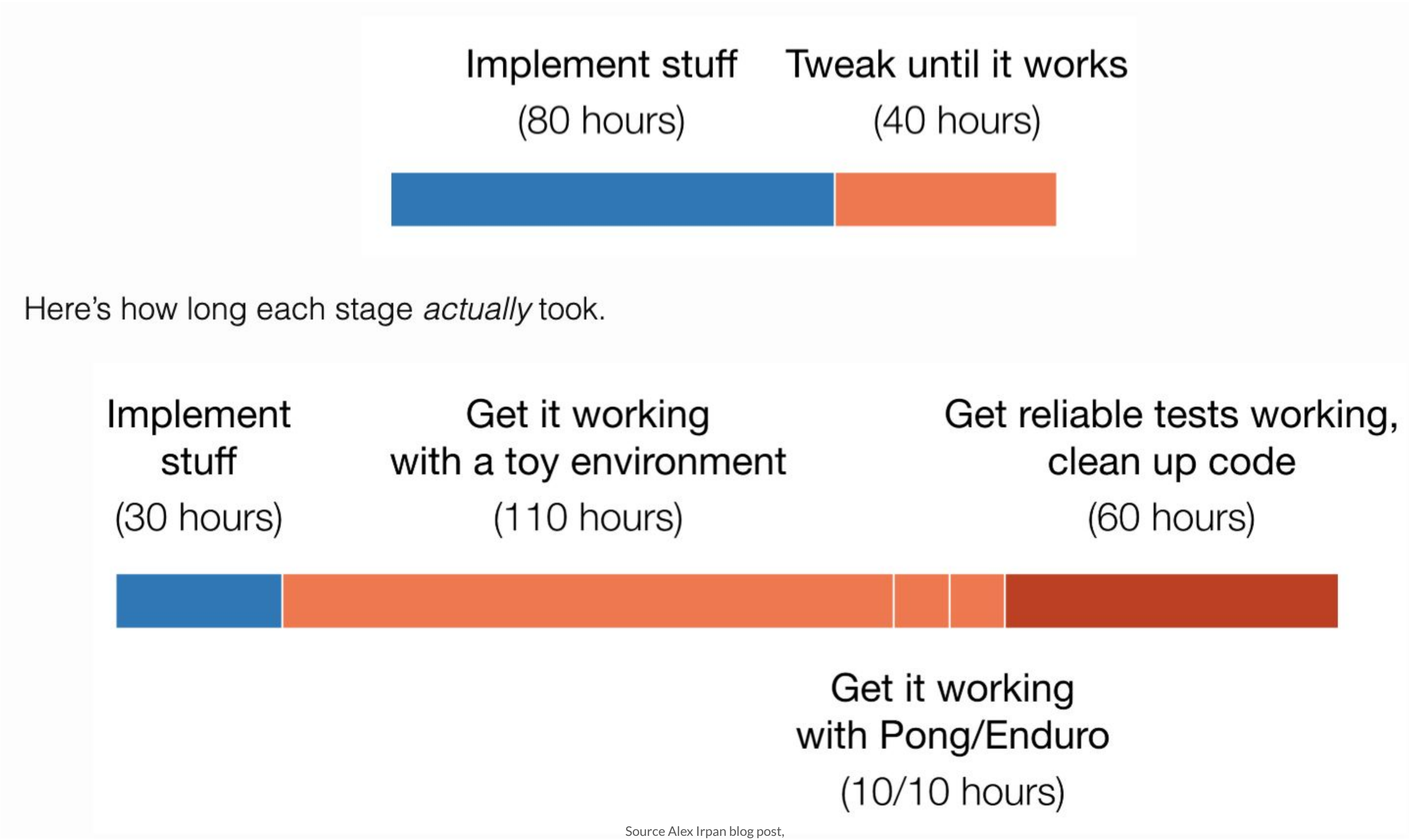
Better:
`(position_xyz, velocity_xyz, rotation_quat, goal_xyz)`

Example:

Driving a car

Observation:
`Image, (256, 256, 3)`

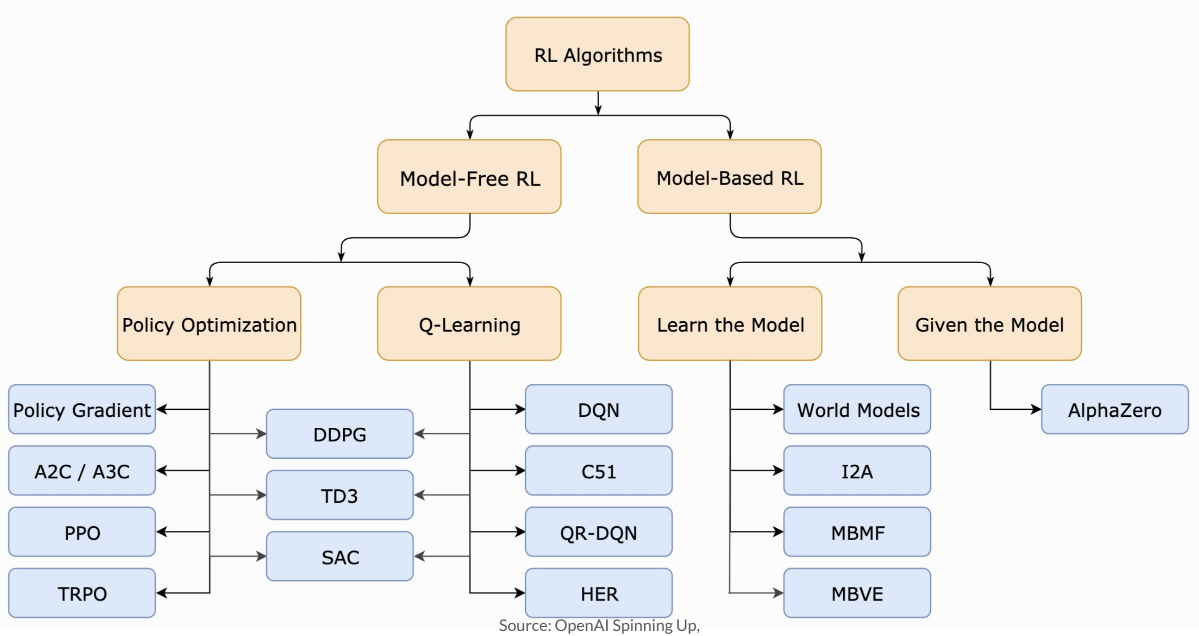
Better:
Image stack (last 4 images) + depth images
`(4, 256, 256, 3) + (4, 256, 256, 1)`



Source Alex Irpan blog post, https://www.alexirpan.com/2018/02/14/rl-hard.html,

Sourc: Openai ,https://www.youtube.com/watch?v=tlOIHko8ySg

State-of-the-Art DRL algos

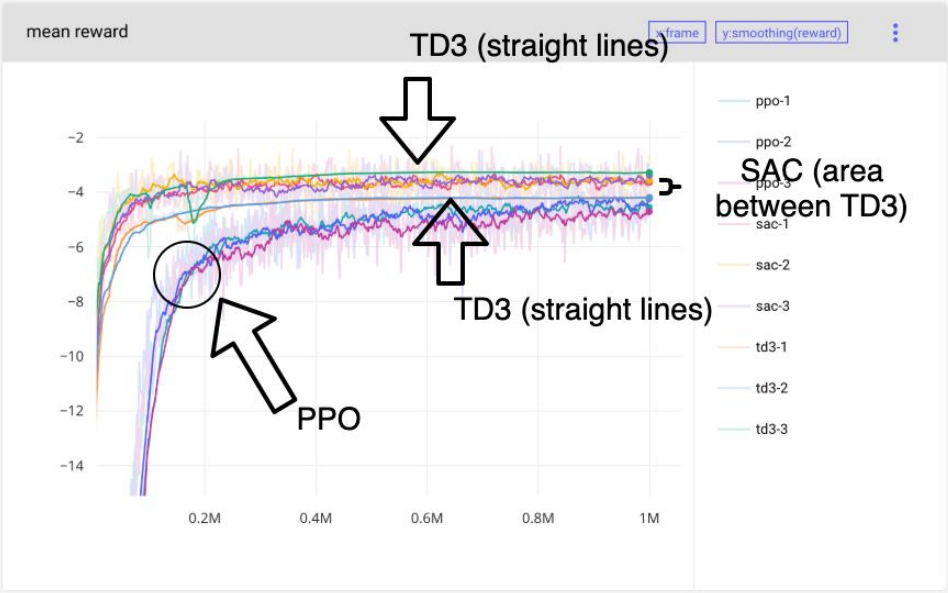


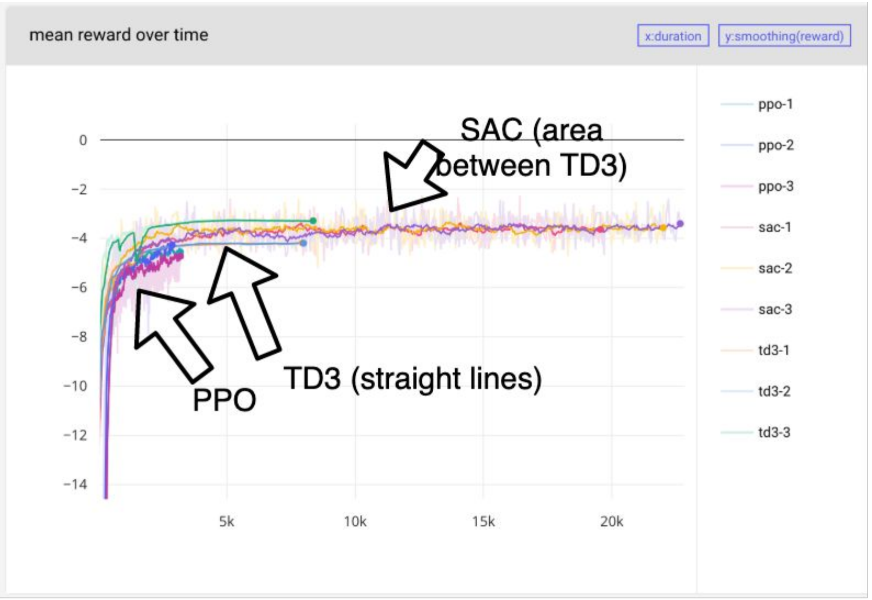
Source: OpenAI Spinning Up,
https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

How to pick?

9/10 times: PPO works (discrete or continuous actions)
+ few HP adjustments (episodes, stacked frames, hidden rep size)

Otherwise, try SAC (slow but stable) or TD3 (fast, easy, sensitive to seed)

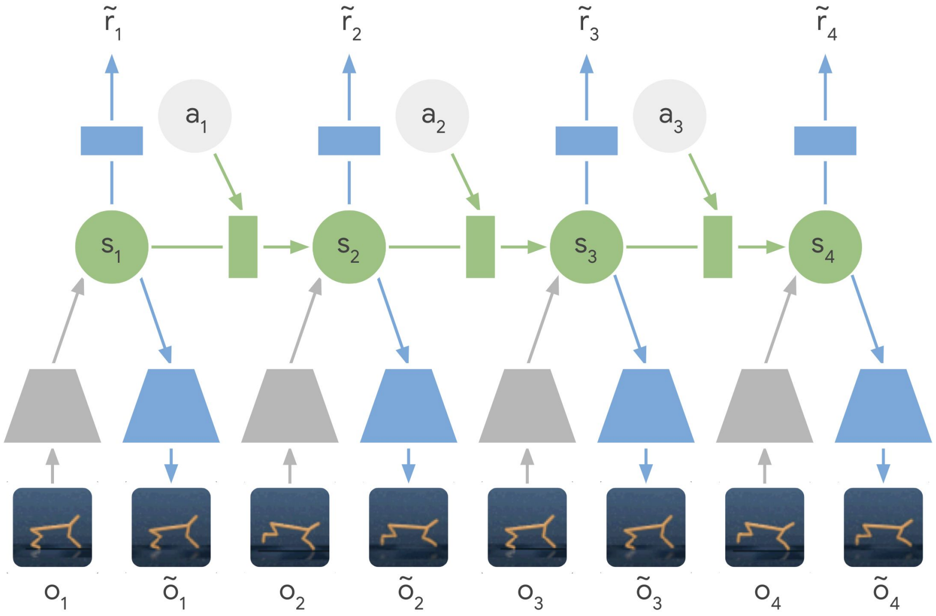




X axis: steps in the environment
(every time the “env.step()” function is called)

X axis: compute time in seconds

Bonus: Modern Planning via PlaNet



Source: PlaNet paper, https://arxiv.org/pdf/1811.04551.pdf

Bonus: Imitation Learning

Naive: behavior cloning

- Can be used as initialization for RL policy
- But overfits to training data

Better: DAgger / Deeply AggreVaTeD, see
http://videolectures.net/DLRLsummerschool2018_daume_imitation_learning/

Thanks, questions?