CS230: Lecture 9
Deep Reinforcement Learning

Kian Katanforoosh
Menti code: 80 24 08

Today's outline

I. Motivation

II. Recycling is good: an introduction to RL

III. Deep Q-Networks

IV. Application of Deep Q-Network: Breakout (Atari)

V. Tips to train Deep Q-Network

VI. Advanced topics

I. Motivation

AlphaGo

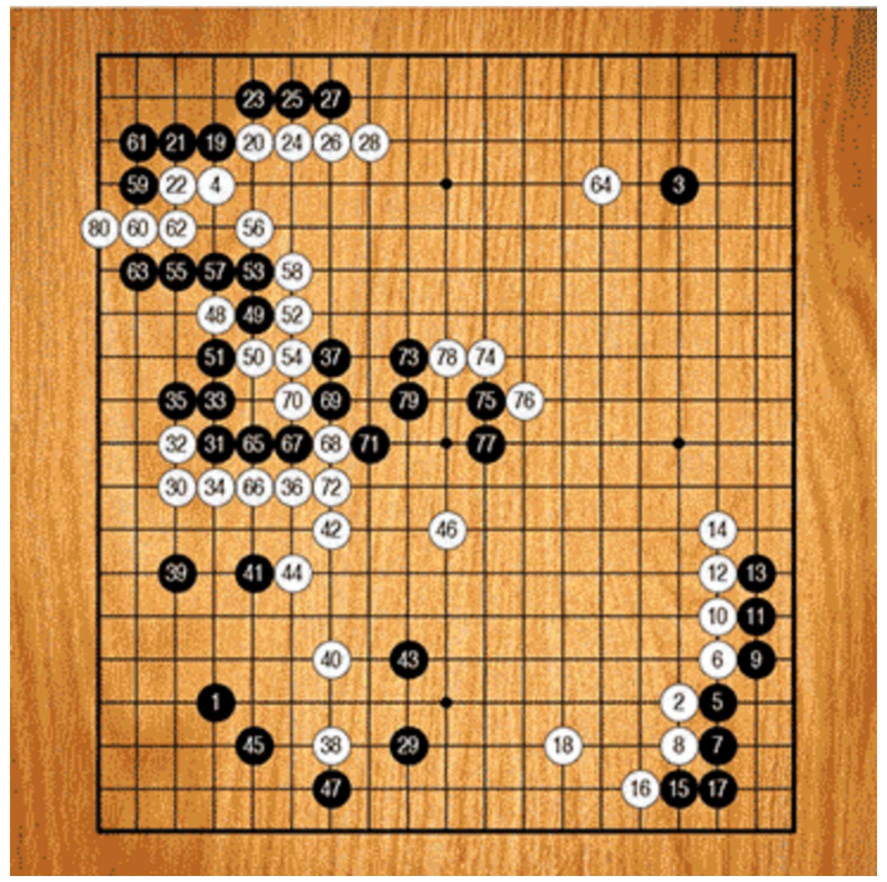
Human Level Control through Deep Reinforcement Learning

[Silver et al. (2017): Mastering the game of Go without human knowledge]
[Mnih et al. (2015): Human Level Control through Deep Reinforcement Learning]

I. Motivation

Why RL?
- **Delayed labels**
- **Making sequences of decisions**

What is RL?
- **Automatically learn to make good sequences of decision**



Source: https://deepmind.com/blog/alphago-zero-learning-scratch/

Examples of RL applications

Robotics

Games

Advertisement

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

Today's outline

I. Motivation

II. Recycling is good: an introduction to RL

III. Deep Q-Networks

IV. Application of Deep Q-Network: Breakout (Atari)

V. Tips to train Deep Q-Network

VI. Advanced topics

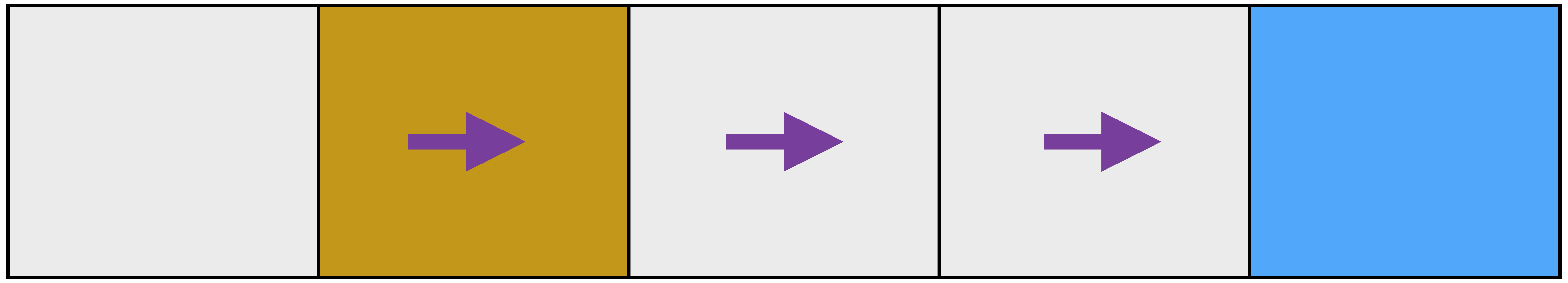
II. Recycling is good: an introduction to RL

Problem statement

*Define reward "**r**" in every state*

+2	0	0	+1	+10

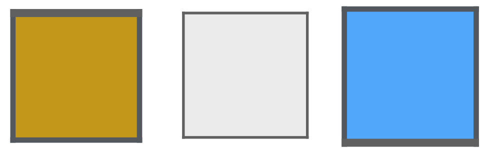
Best strategy to follow if $\gamma = 1$



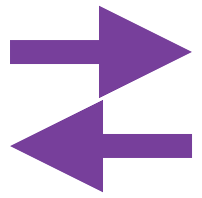
***Goal**: maximize the return (rewards)*

***Number of states**: 5*

***Types of states**:*



***Agent's Possible actions**:*



***Additional rule:** garbage collector coming in 3min, it takes 1min to move between states*

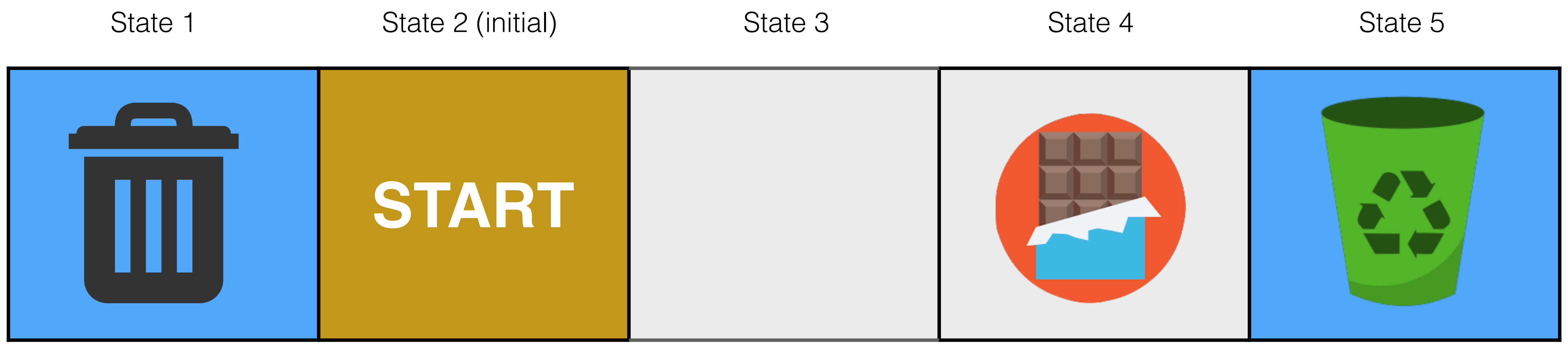
How to define the long-term return?

Discounted return $$R = \sum_{t=0} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$$

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement



*Define reward "**r**" in every state*

S1	S2	S3	S4	S5
+2	0	0	+1	+10

Assuming $\gamma = 0.9$

Discounted return $R = \sum_{t=0} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$

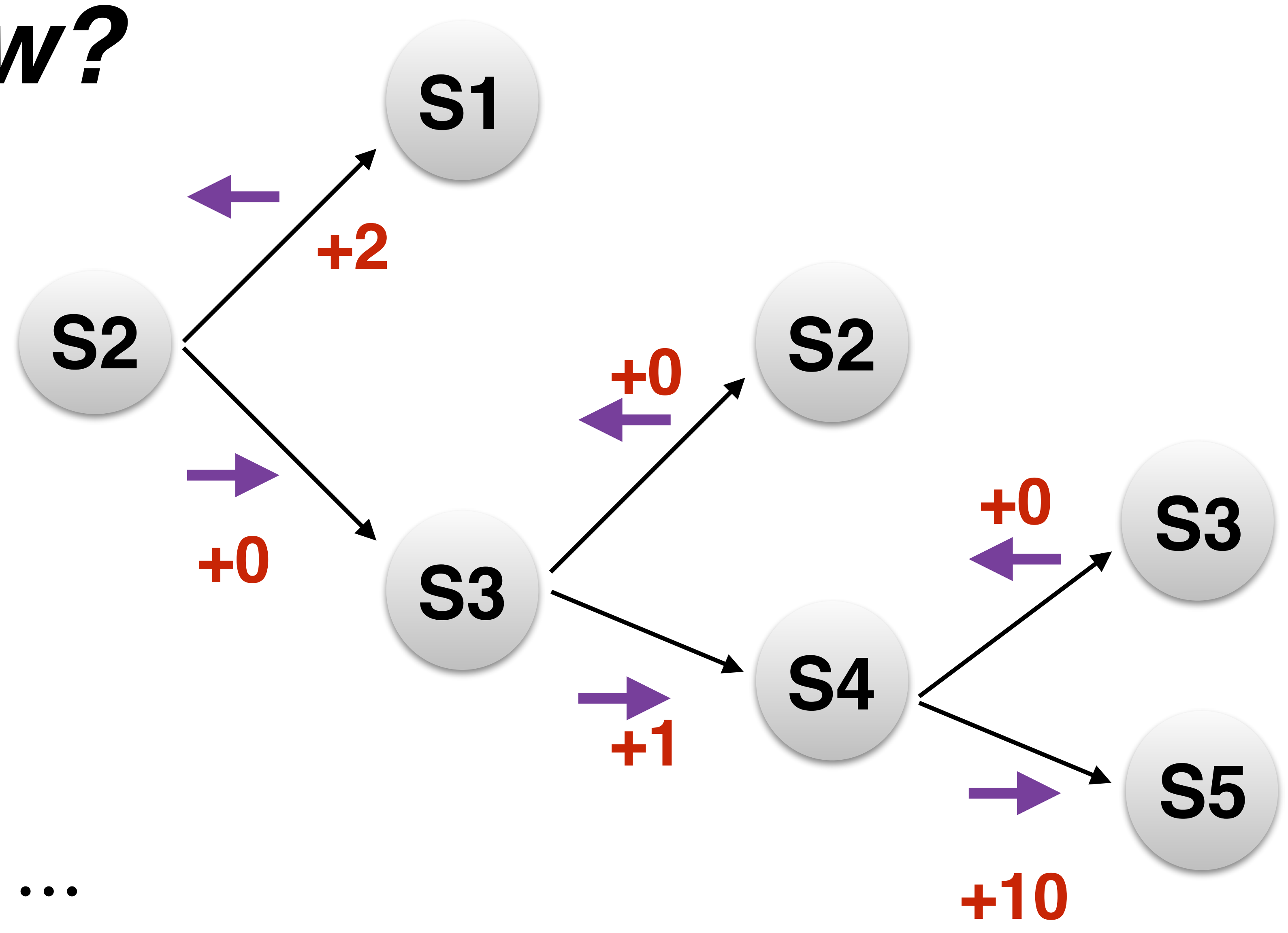
What do we want to learn?

how good is it to take action 1 in state 2

Q-table

$$\begin{array}{cc} & \#\text{actions} \\ Q = & \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix} \#\text{states} \end{array}$$

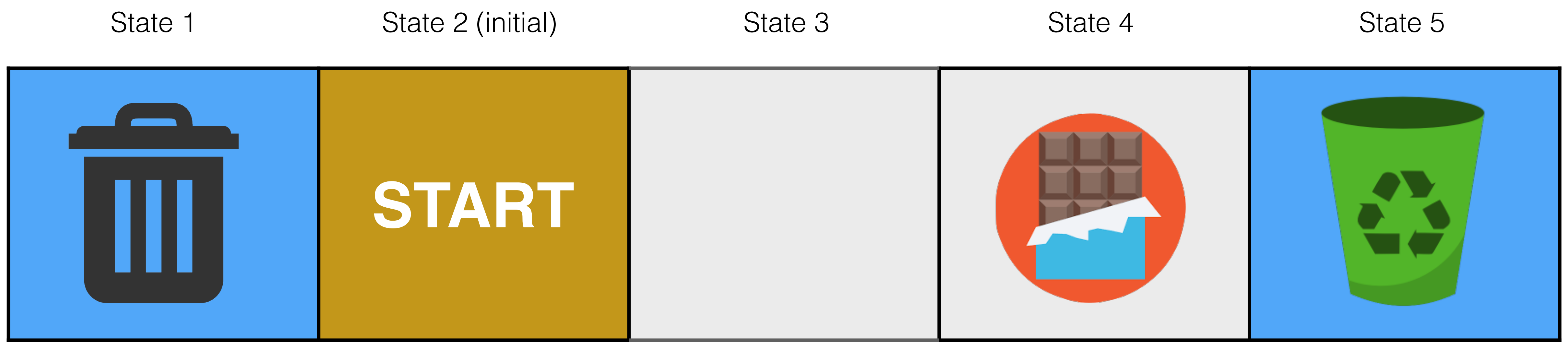
How?



Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement



*Define reward "**r**" in every state*

+2	0	0	+1	+10
S1	S2	S3	S4	S5

Assuming $\gamma = 0.9$

Discounted return $R = \sum_{t=0} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$

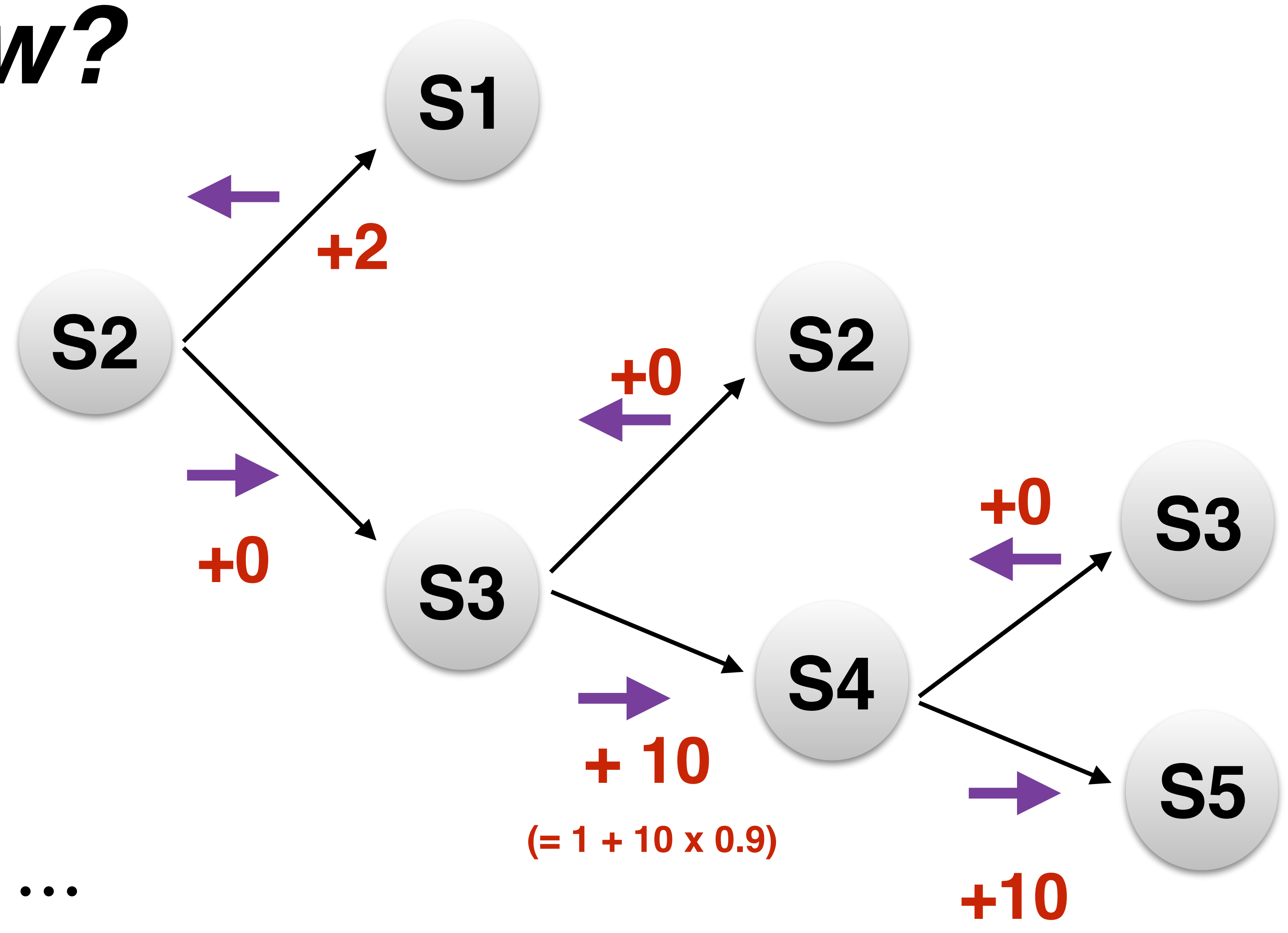
What do we want to learn?

how good is it to take action 1 in state 2

Q-table

$$Q = \begin{array}{c} \text{\#actions} \\ \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix} \end{array} \text{\#states}$$

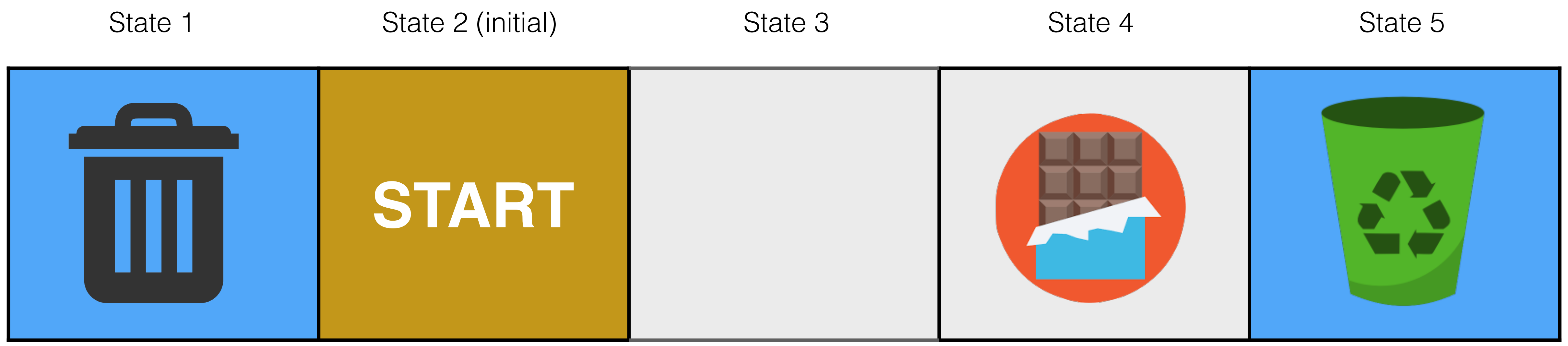
How?



Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement



Define reward "r" in every state

+2	0	0	+1	+10
S1	S2	S3	S4	S5

Assuming $\gamma = 0.9$

Discounted return $R = \sum_{t=0} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$

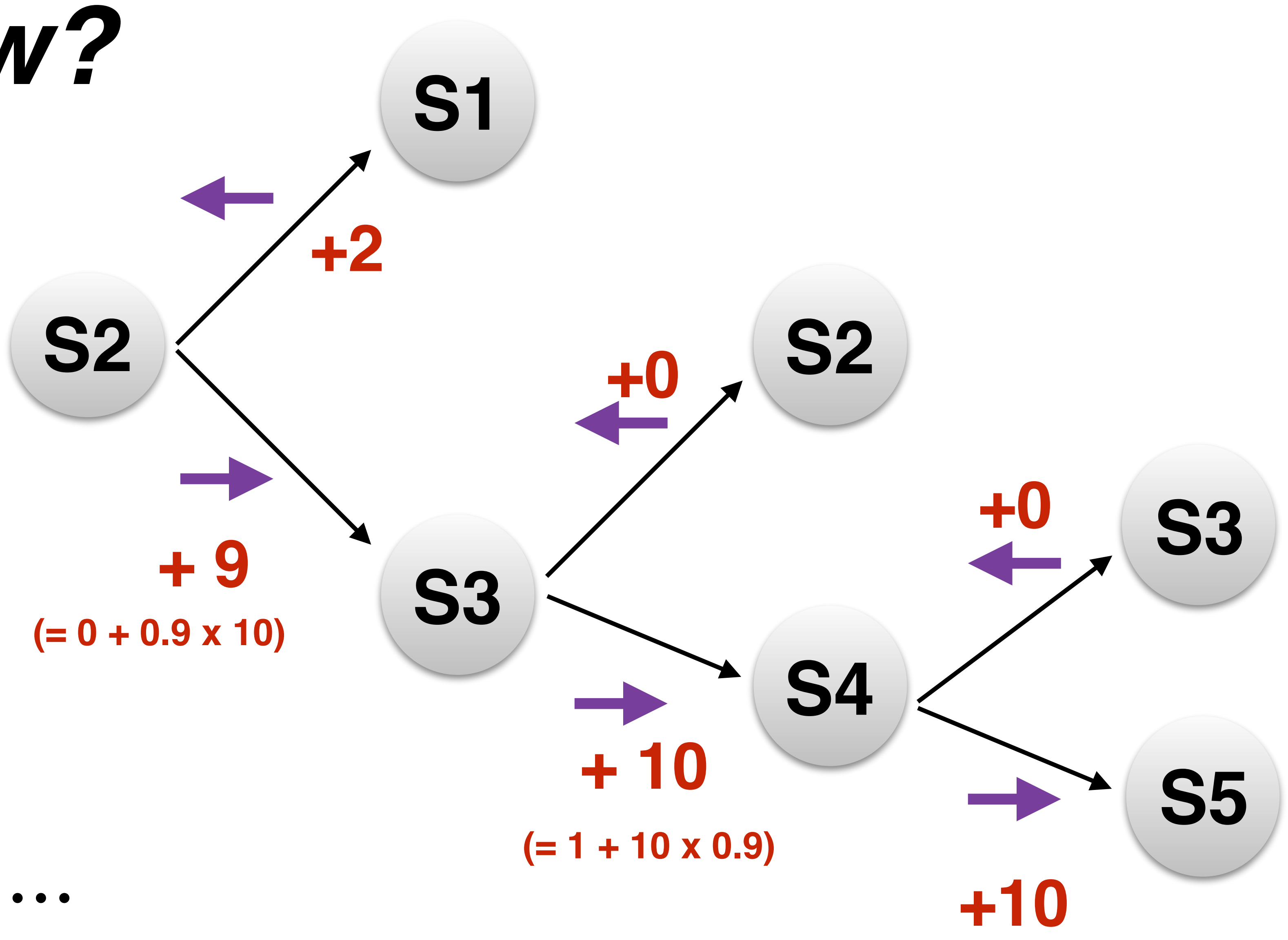
What do we want to learn?

how good is it to take action 1 in state 2

Q-table

$$\begin{array}{cc} & \#\text{actions} \\ Q = & \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix} \end{array} \#\text{states}$$

How?



Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement

State 1 | State 2 (initial) | State 3 | State 4 | State 5

*Define reward "**r**" in every state*

+2	0	0	+1	+10
S1	S2	S3	S4	S5

Assuming $\gamma = 0.9$

Discounted return $R = \sum_{t=0} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$

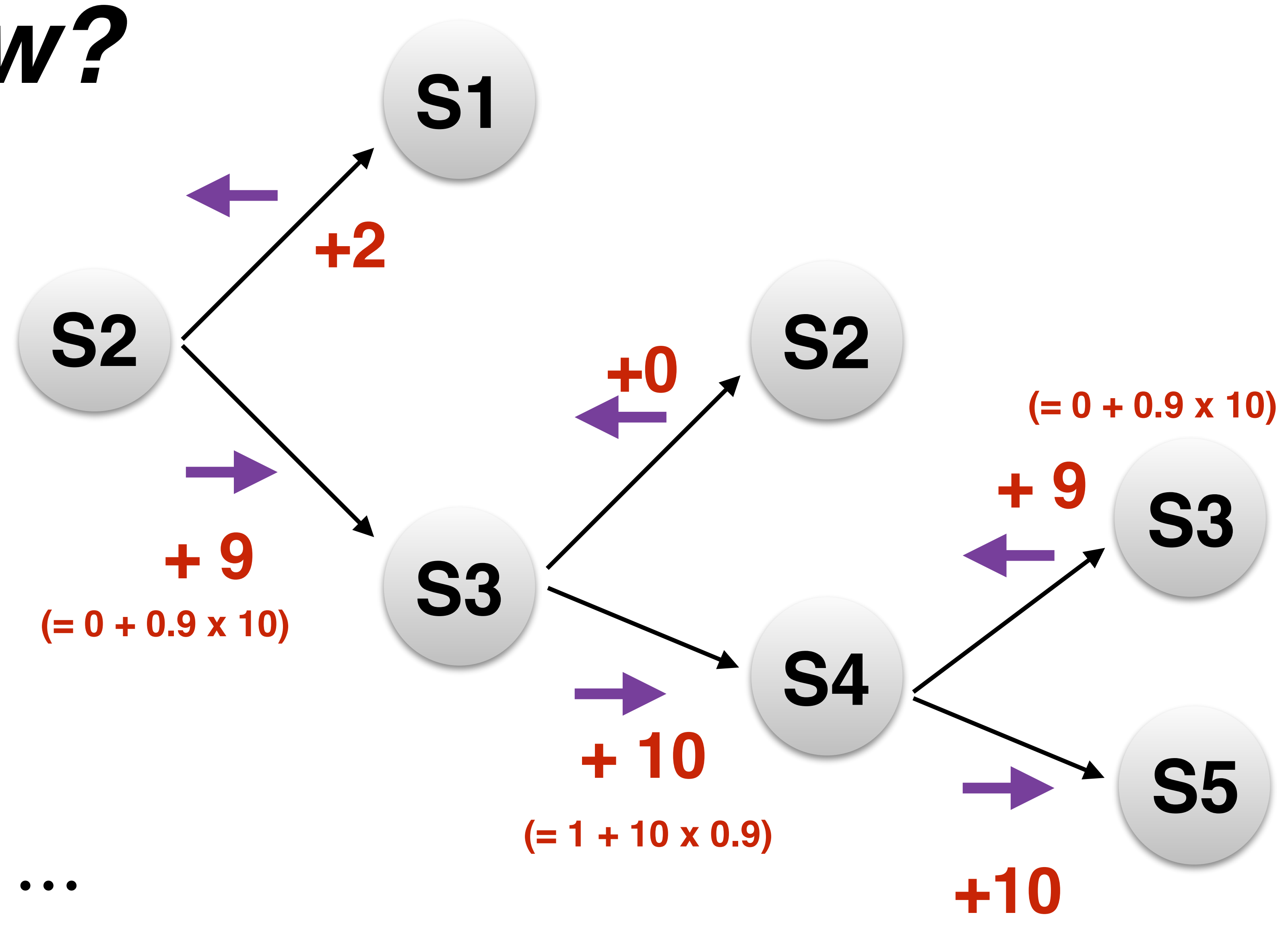
What do we want to learn?

how good is it to take action 1 in state 2

Q-table

$$Q = \begin{array}{c} \text{\#actions} \\ \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix} \end{array} \text{\#states}$$

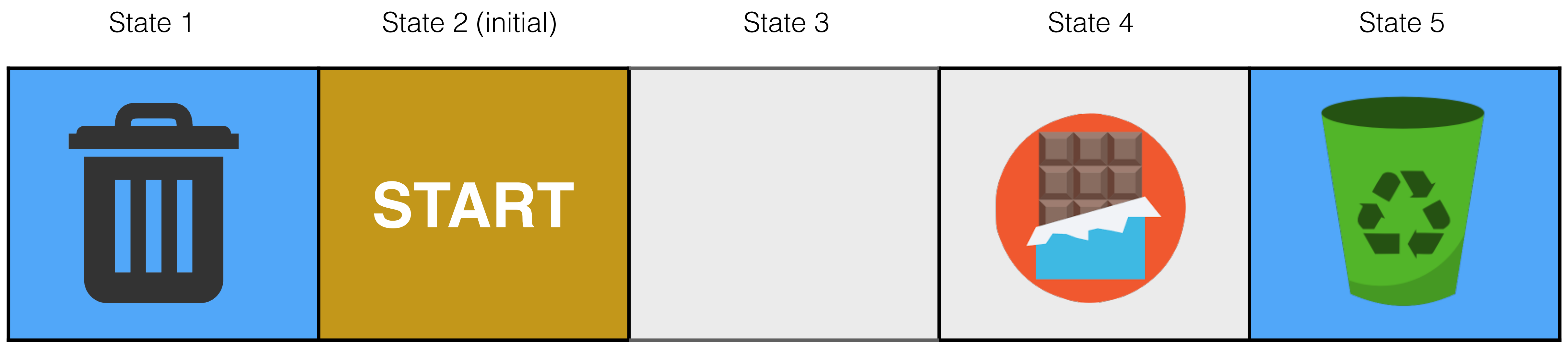
How?



Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement



Define reward "r" in every state

S1	S2	S3	S4	S5
+2	**0**	**0**	**+1**	**+10**

Assuming $\gamma = 0.9$

Discounted return $R = \sum_{t=0} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$

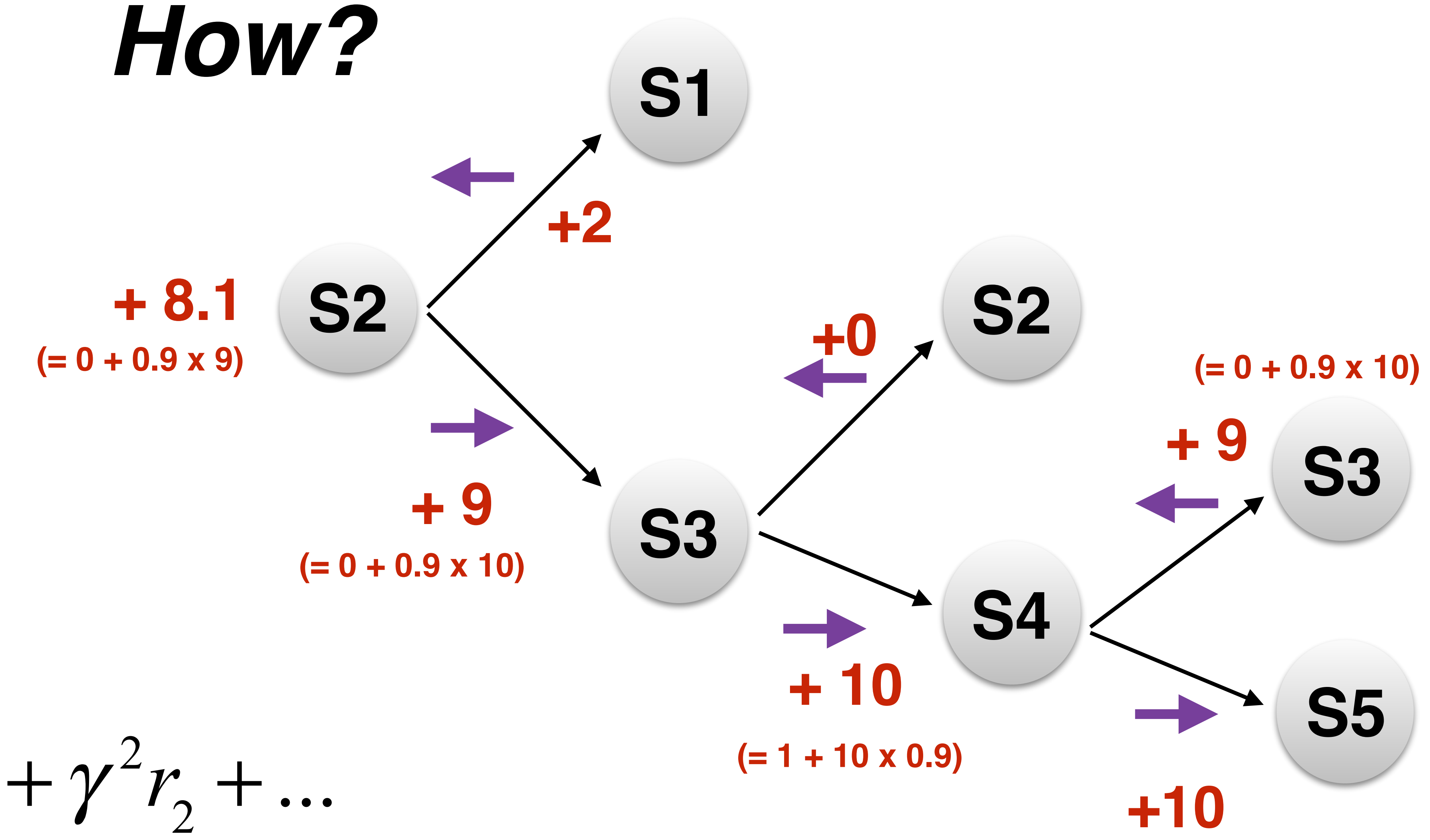
What do we want to learn?

how good is it to take action 1 in state 2

Q-table

$$Q = \begin{array}{c} \text{\#actions} \\ \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix} \end{array} \ \text{\#states}$$

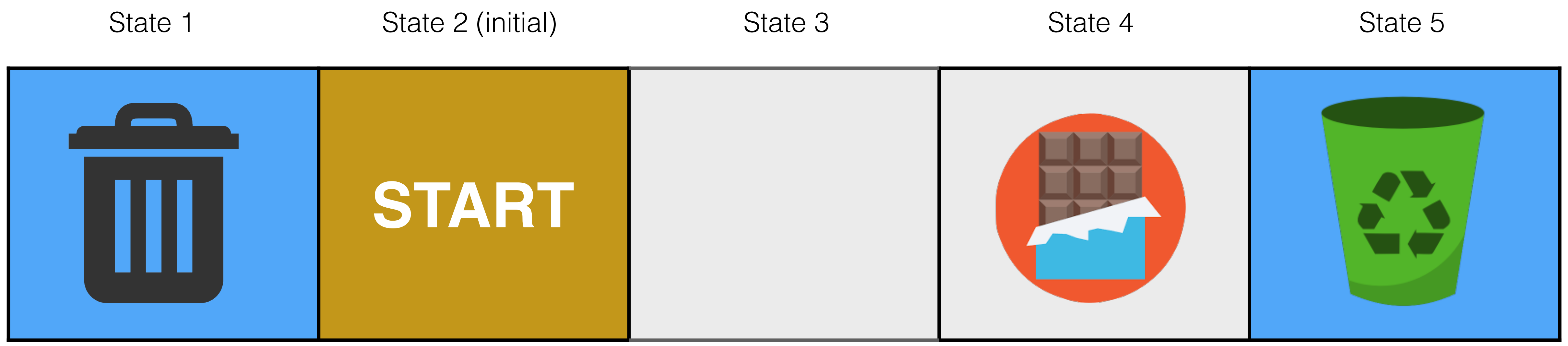
How?



Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement



Define reward "**r**" in every state

S1	S2	S3	S4	S5
+2	**0**	**0**	**+1**	**+10**

Assuming $\gamma = 0.9$

Discounted return $R = \sum_{t=0} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$

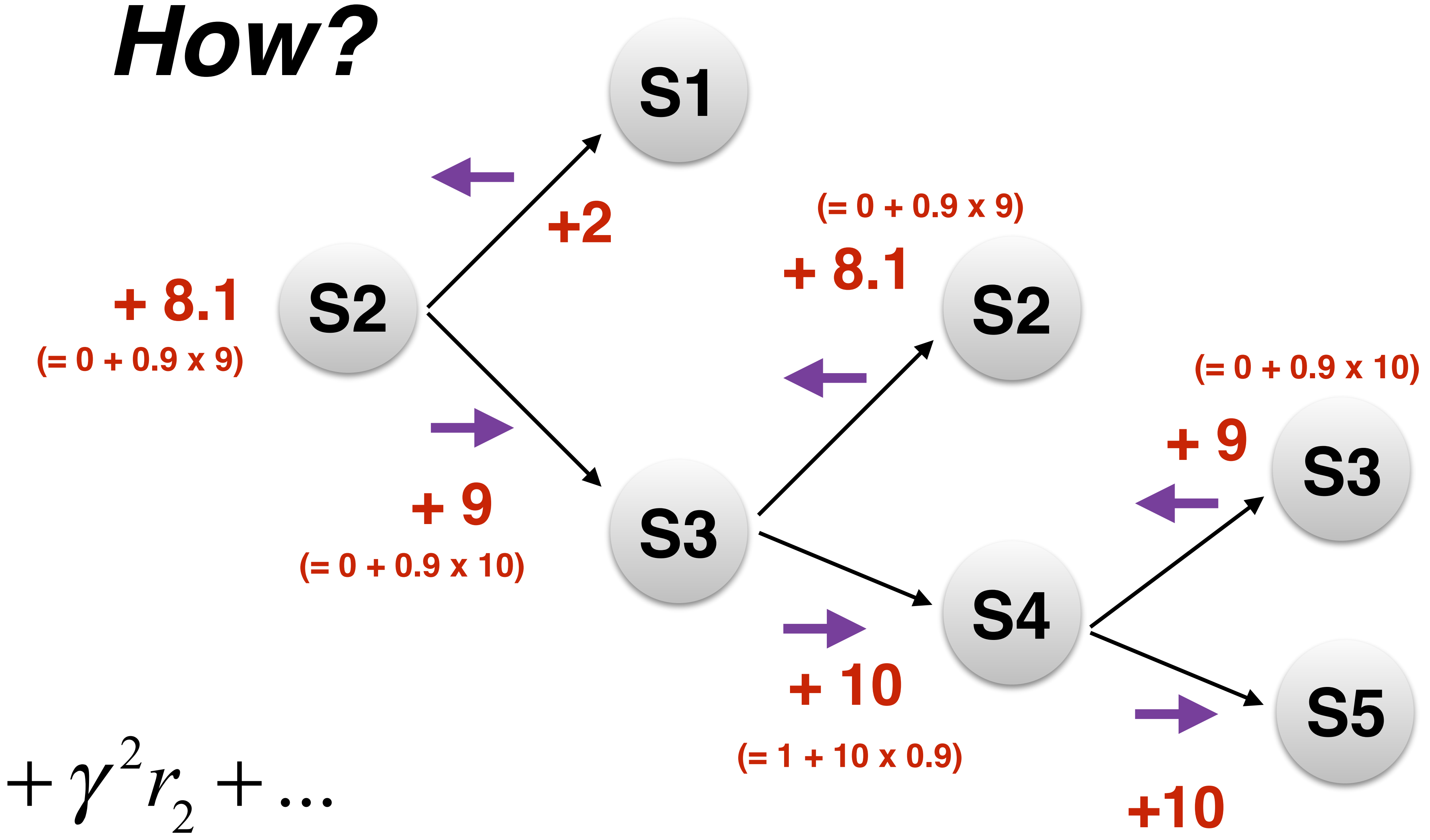
What do we want to learn?

how good is it to take action 1 in state 2

Q-table

$$\begin{array}{c} \text{\#actions} \\ Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix} \text{\#states} \end{array}$$

How?



Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement

*Define reward "**r**" in every state*

+2	**0**	**0**	**+1**	**+10**
S1	S2	S3	S4	S5

Assuming $\gamma = 0.9$

Discounted return $R = \sum_{t=0} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$

What do we want to learn?

how good is it to take action 1 in state 2

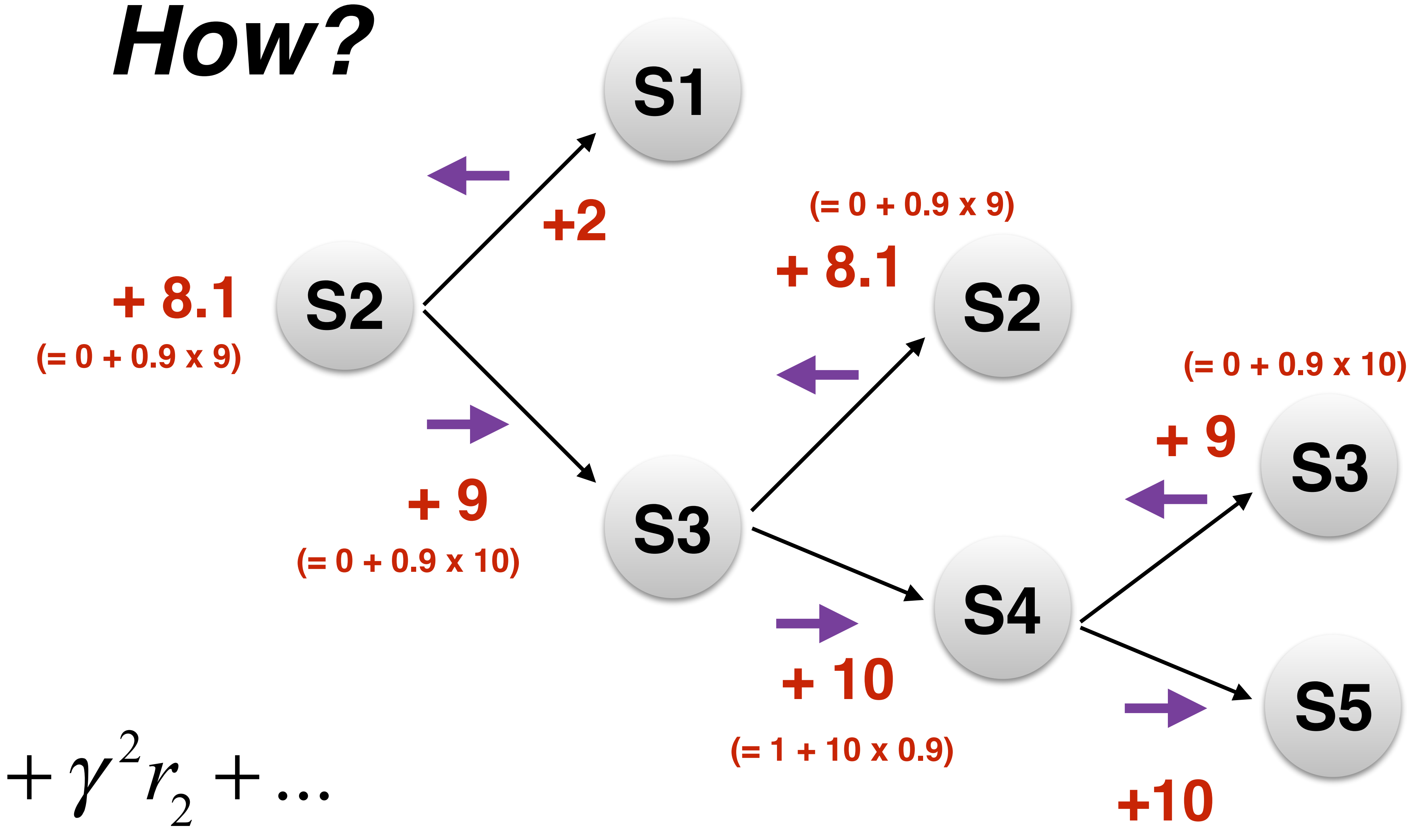
Q-table

#actions

$$Q = \begin{pmatrix} 0 & 0 \\ 2 & 9 \\ 8.1 & 10 \\ 9 & 10 \\ 0 & 0 \end{pmatrix}$$

#states

How?



Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement

State 1	State 2 (initial)	State 3	State 4	State 5

*Define reward "**r**" in every state*

+2	**0**	**0**	**+1**	**+10**

Best strategy to follow if $\gamma = 0.9$

When state and actions space are too big, this method has huge memory cost

Function telling us our best strategy

What do we want to learn?

how good is it to take action 1 in state 2

Q-table

#actions

$$Q = \begin{pmatrix} 0 & 0 \\ 2 & 9 \\ 8.1 & 10 \\ 9 & 10 \\ 0 & 0 \end{pmatrix}$$

#states

Bellman equation (optimality equation)

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

Policy $$\pi(s) = \arg\max_{a}(Q^*(s,a))$$

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

What we've learned so far:

- *Vocabulary: environment, agent, state, action, reward, total return, discount factor.*

- *Q-table: matrix of entries representing "how good is it to take action a in state s"*

- *Policy: function telling us what's the best strategy to adopt*

- *Bellman equation satisfied by the optimal Q-table*

Today's outline

I. Motivation

II. Recycling is good: an introduction to RL

III. Deep Q-Networks

IV. Application of Deep Q-Network: Breakout (Atari)

V. Tips to train Deep Q-Network

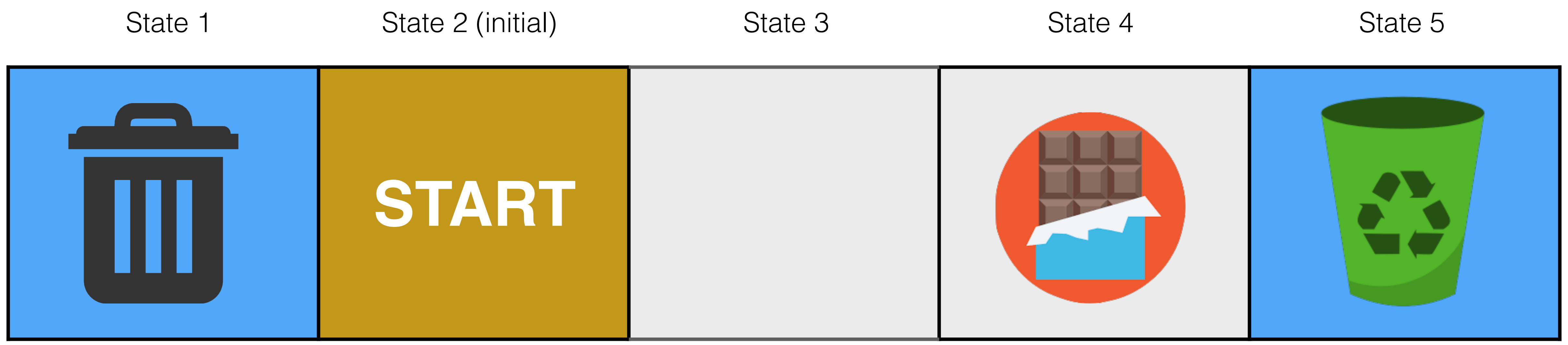
VI. Advanced topics

III. Deep Q-Networks

Main idea: *find a Q-function to replace the Q-table*

Neural Network

Problem statement



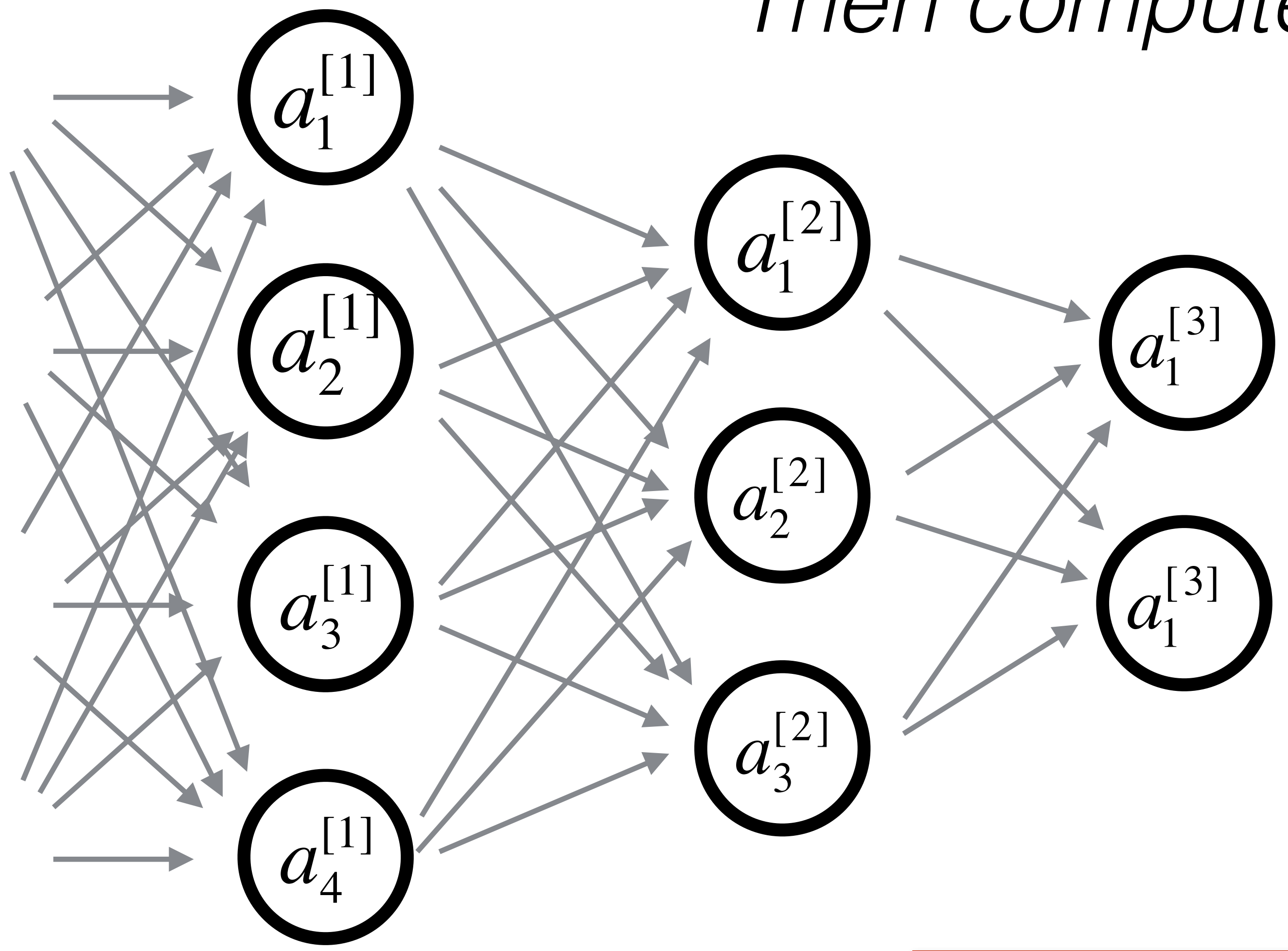
Q-table

#actions

$$Q = \begin{pmatrix} 0 & 0 \\ 2 & 9 \\ 8.1 & 10 \\ 9 & 10 \\ 0 & 0 \end{pmatrix}$$

#states

$$s = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$



$\rightarrow Q(s,\leftarrow)$

$\rightarrow Q(s,\rightarrow)$

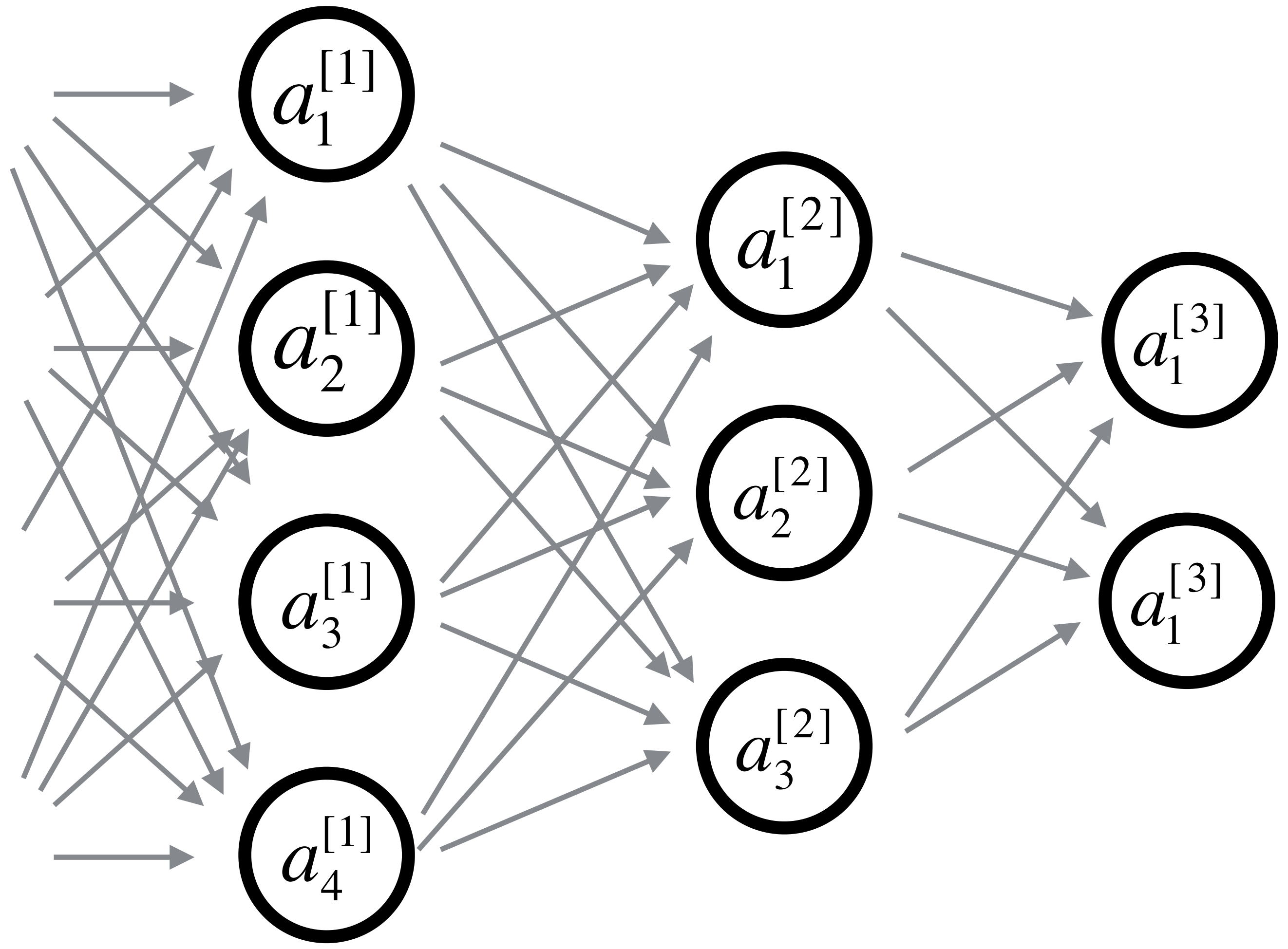
Then compute loss, backpropagate.

How to compute the loss?

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

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

$$s = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$



$\to Q(s,\leftarrow)$

$\to Q(s,\rightarrow)$

Loss function

$$L = (y - Q(s,\leftarrow))^2$$

Target value

Case: $\boxed{Q(s,\leftarrow) > Q(s,\rightarrow)}$

$$y = r_{\leftarrow} + \gamma \max_{a'}(Q(s_{\leftarrow}^{next}, a'))$$

Hold fixed for backprop

Immediate reward for taking action $\leftarrow$ in state s

Discounted maximum future reward when you are in state $s_{\leftarrow}^{next}$

Case: $\boxed{Q(s,\leftarrow) < Q(s,\rightarrow)}$

$$y = r_{\rightarrow} + \gamma \max_{a'}(Q(s_{\rightarrow}^{next}, a'))$$

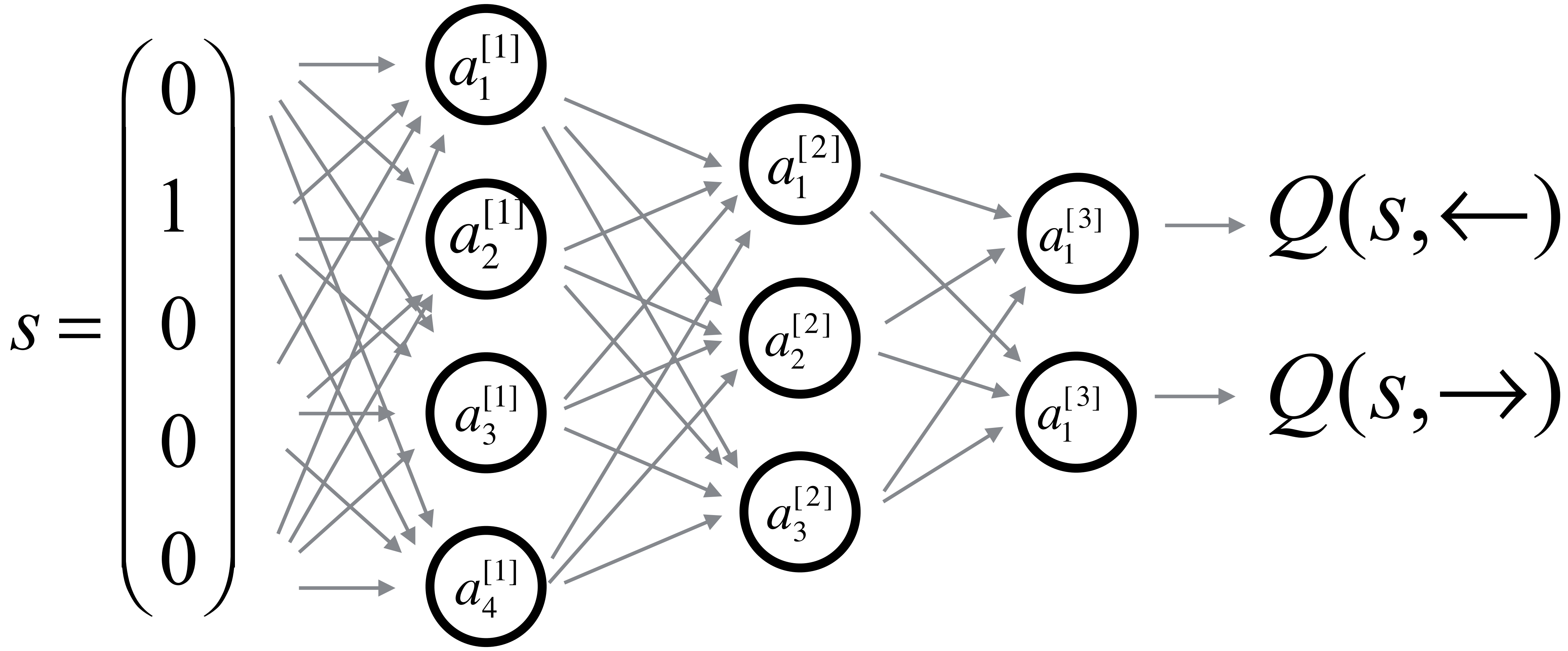
Hold fixed for backprop

Immediate Reward for taking action $\rightarrow$ in state s

Discounted maximum future reward when you are in state $s_{\rightarrow}^{next}$

[Francisco S. Melo: Convergence of Q-learning: a simple proof]

III. Deep Q-Networks



Loss function (regression)

$$L = (y - Q(s,\rightarrow))^2$$

Target value

Case: $\boxed{Q(s,\leftarrow) > Q(s,\rightarrow)}$

$$y = r_{\leftarrow} + \gamma \max_{a'}(Q(s_{\leftarrow}^{next}, a'))$$

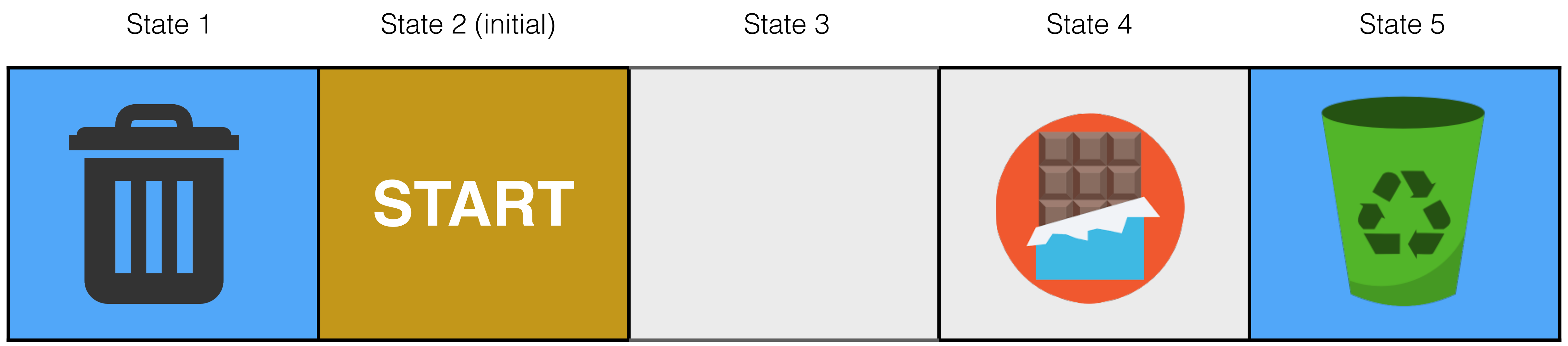
Case: $\boxed{Q(s,\leftarrow) < Q(s,\rightarrow)}$

$$y = r_{\rightarrow} + \gamma \max_{a'}(Q(s_{\rightarrow}^{next}, a'))$$

Backpropagation Compute $\frac{\partial L}{\partial W}$ and update W using stochastic gradient descent

Recap'

DQN Implementation:



- *Initialize your Q-network parameters*
- *Loop over episodes:*
    - *Start from initial state s*
    - *Loop over time-steps:*
        - *Forward propagate s in the Q-network*
        - *Execute action a (that has the maximum Q(s,a) output of Q-network)*
        - *Observe rewards r and next state s'*
        - *Compute targets y by forward propagating state s' in the Q-network, then compute loss.*
        - *Update parameters with gradient descent*

Today's outline

I. Motivation

II. Recycling is good: an introduction to RL

III. Deep Q-Networks

IV. Application of Deep Q-Network: Breakout (Atari)

V. Tips to train Deep Q-Network

VI. Advanced topics

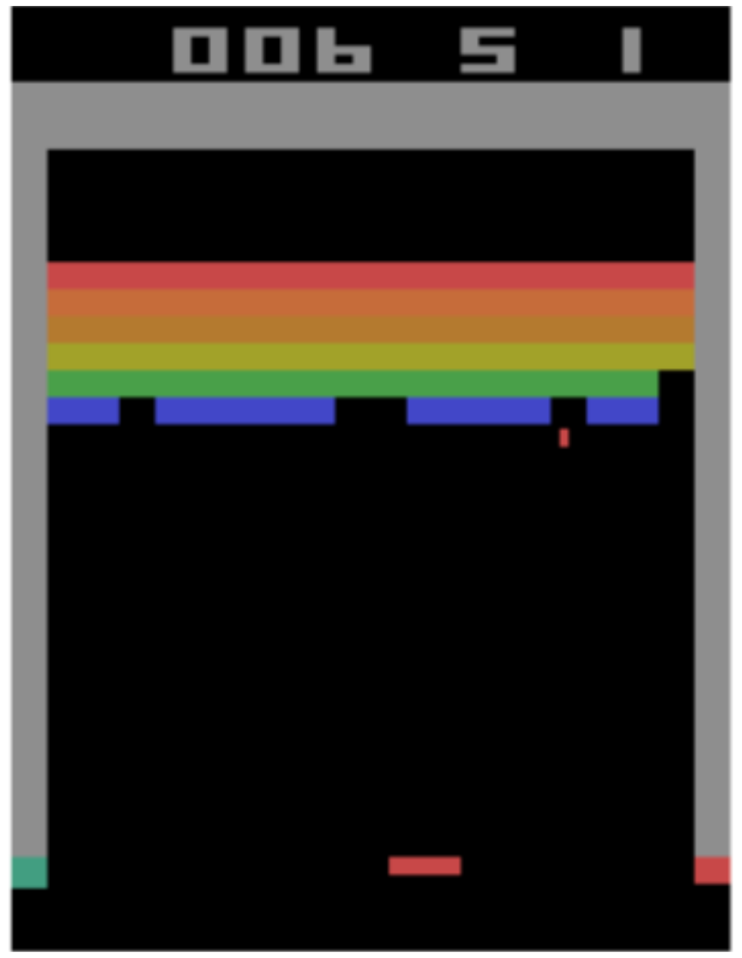
IV. Deep Q-Networks application: Breakout (Atari)

Goal: play breakout, i.e. destroy all the bricks.

Demo

input of Q-network

s =



Output of Q-network

Q-values

$$\begin{pmatrix} Q(s,\leftarrow) \\ Q(s,\rightarrow) \\ Q(s,-) \end{pmatrix}$$

Would that work?

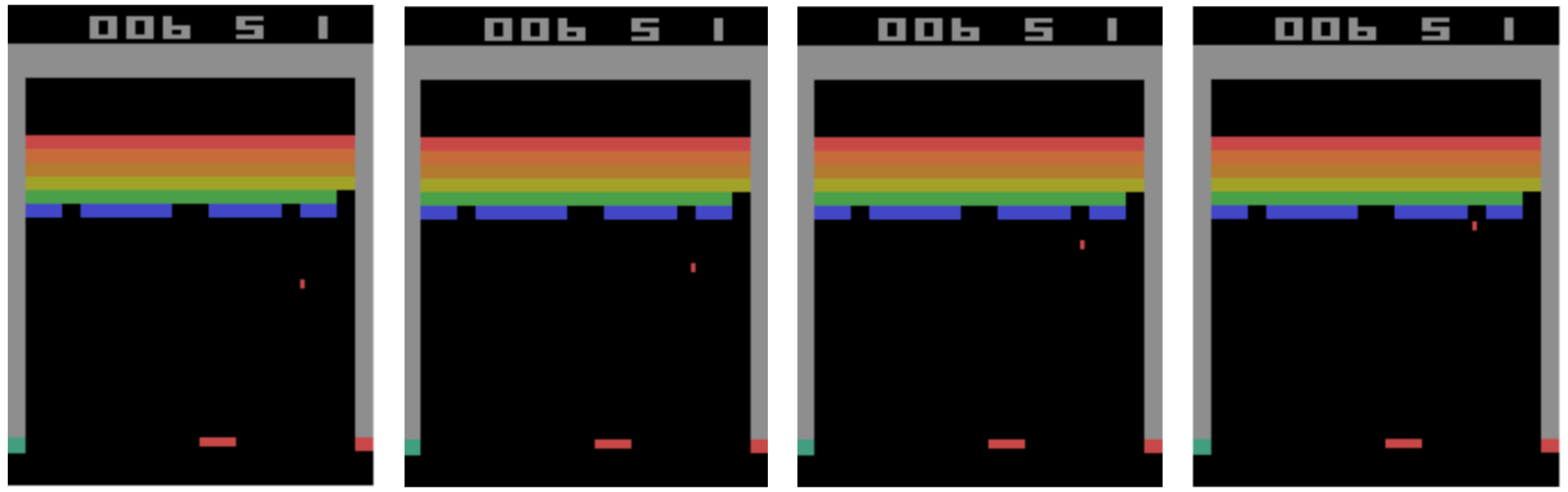
IV. Deep Q-Networks application: Breakout (Atari)

Goal: play breakout, i.e. destroy all the bricks.

Demo

input of Q-network

s =



↓

Preprocessing

↓

$$\phi(s)$$

- Convert to grayscale
- Reduce dimensions (h,w)
- History (4 frames)

Output of Q-network

Q-values

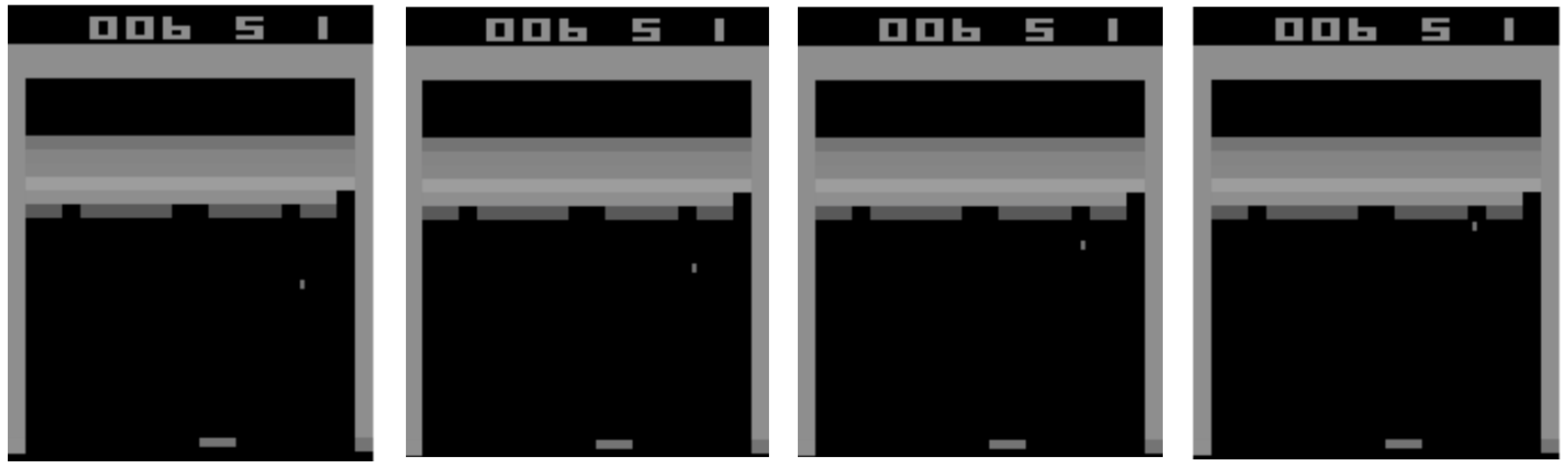
$$\begin{pmatrix} Q(s,\leftarrow) \\ Q(s,\rightarrow) \\ Q(s,-) \end{pmatrix}$$

What is done in preprocessing?

https://www.youtube.com/watch?v=V1eYniJ0Rnk

IV. Deep Q-Networks application: Breakout (Atari)

input of Q-network

$\phi(s) =$ 

Deep Q-network architecture?

$\phi(s) \rightarrow$ CONV | ReLU $\rightarrow$ CONV | ReLU $\rightarrow$ CONV | ReLU $\rightarrow$ FC (RELU) | FC (LINEAR) $\rightarrow$ $\begin{pmatrix} Q(s,\leftarrow) \\ Q(s,\rightarrow) \\ Q(s,-) \end{pmatrix}$

Recap' (+ preprocessing + terminal state)

Some training challenges:

- Keep track of terminal step
- Experience replay
- Epsilon greedy action choice (Exploration / Exploitation tradeoff)

DQN Implementation:

- *Initialize your Q-network parameters*
- *Loop over episodes:*
  - *Start from initial state ~~s~~* $\phi(s)$
  - *Loop over time-steps:*
    - *Forward propagate ~~s~~* $\phi(s)$ *in the Q-network*
    - *Execute action a (that has the maximum Q(~~s~~* $\phi(s)$*,a) output of Q-network)*
    - *Observe rewards r and next state s'*
    - ***Use s' to create*** $\phi(s')$
    - *Compute targets y by forward propagating state ~~s'~~* $\phi(s')$ *in the Q-network, then compute loss.*
    - *Update parameters with gradient descent*

Recap' (+ preprocessing + terminal state)

Some training challenges:
- Keep track of terminal step
- Experience replay
- Epsilon greedy action choice (Exploration / Exploitation tradeoff)

DQN Implementation:

- *Initialize your Q-network parameters*
- *Loop over episodes:*
    - *Start from initial state ~~s~~* $\phi(s)$
    - *Loop over time-steps:*
        - *Forward propagate ~~s~~* $\phi(s)$ *in the Q-network*
        - *Execute action a (that has the maximum Q(~~s~~* $\phi(s)$*,a) output of Q-network)*
        - *Observe rewards r and next state s'*
        - ***Use s' to create*** $\phi(s')$
        - *Compute targets y by forward propagating state ~~s'~~* $\phi(s')$ *in the Q-network, then compute loss.*
        - *Update parameters with gradient descent*

Recap' (+ preprocessing + terminal state)

DQN Implementation:

- *Initialize your Q-network parameters*
- *Loop over episodes:*
  - *Start from initial state ~~s~~* $\phi(s)$
  - ***Create a boolean to detect terminal states: terminal = False***
  - *Loop over time-steps:*
    - *Forward propagate ~~s~~* $\phi(s)$ *in the Q-network*
    - *Execute action a (that has the maximum Q(~~s~~* $\phi(s)$*,a) output of Q-network)*
    - *Observe rewards r and next state s'*
    - *Use s' to create* $\phi(s')$
    - ***Check if s' is a terminal state.*** *Compute targets y by forward propagating state ~~s~~* $\phi(s')$ *in the Q-network, then compute loss.*
    - *Update parameters with gradient descent*

$$\begin{cases} if \quad terminal = False \quad : \quad y = r + \gamma \max_{a'}(Q(s',a')) \\ if \quad terminal = True \quad : \quad y = r \qquad (break) \end{cases}$$

Some training challenges:
- Keep track of terminal step
- Experience replay
- Epsilon greedy action choice (Exploration / Exploitation tradeoff)

IV - DQN training challenges

Experience replay

1 experience (leads to one iteration of gradient descent)

Current method is to start from initial state s and follow:

E1 $\phi(s) \rightarrow a \rightarrow r \rightarrow \phi(s')$

E2 $\phi(s') \rightarrow a' \rightarrow r' \rightarrow \phi(s'')$

E3 $\phi(s'') \rightarrow a'' \rightarrow r'' \rightarrow \phi(s''')$

$\ldots$

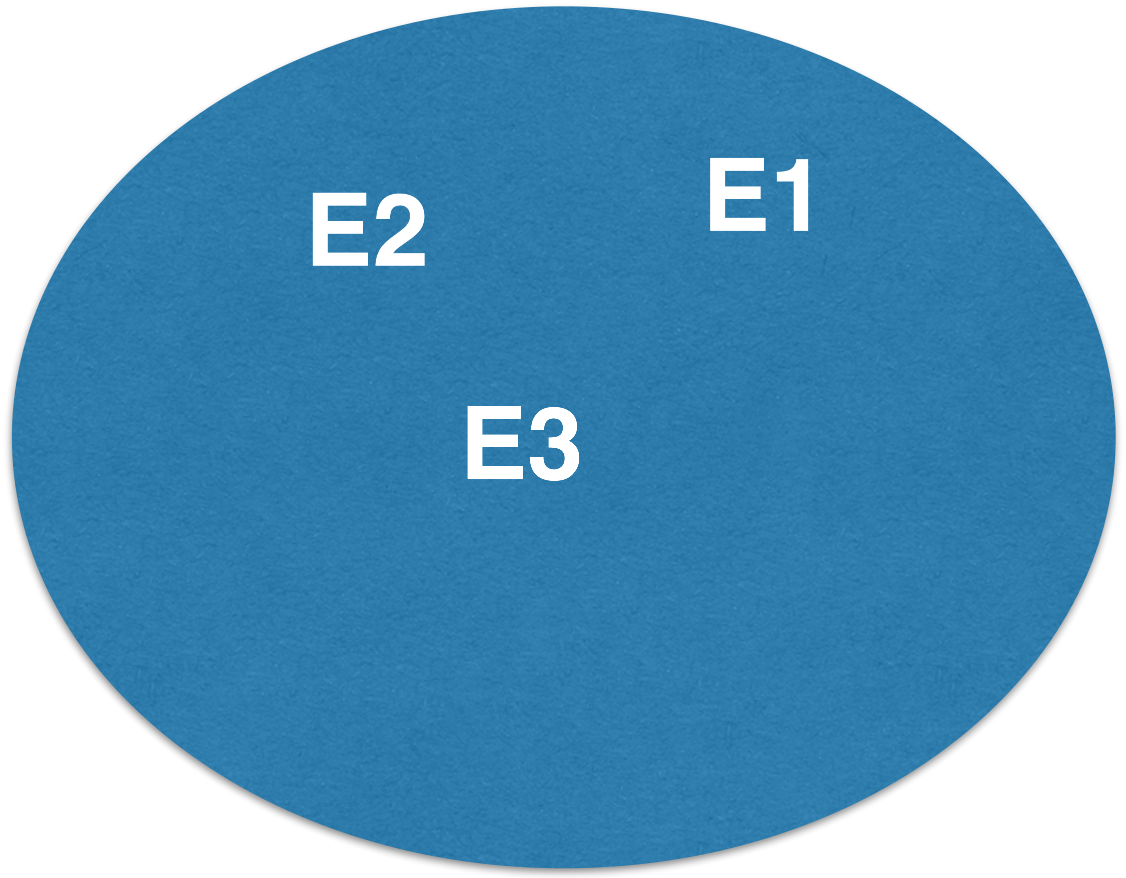
Experience Replay

E1

E2

E3

$\ldots$



Replay memory (D)

Training: E1 ⟶ E2 ⟶ E3

Training: E1 ⟶ sample(E1, E2) ⟶ sample(E1, E2, E3) ⟶ sample(E1, E2, E3, E4) ⟶ $\cdots$

Can be used with mini batch gradient descent

Advantages of experience replay?

Recap' (+ experience replay)

DQN Implementation:

- *Initialize your Q-network parameters*
- ***Initialize replay memory D***
- *Loop over episodes:*
  - *Start from initial state* $\phi(s)$
  - *Create a boolean to detect terminal states: terminal = False*
  - *Loop over time-steps:*
    - *Forward propagate* $\phi(s)$ *in the Q-network*
    - *Execute action a (that has the maximum Q(*$\phi(s)$*,a) output of Q-network)*
    - *Observe rewards r and next state s'*
    - *Use s' to create* $\phi(s')$
    - ***Add experience*** $(\phi(s),a,r,\phi(s'))$ ***to replay memory (D)***
    - ***Sample random mini-batch of transitions from D***
    - *Check if s' is a terminal state. Compute targets y by forward propagating state* $\phi(s')$ *in the Q-network, then compute loss.*
    - *Update parameters with gradient descent*

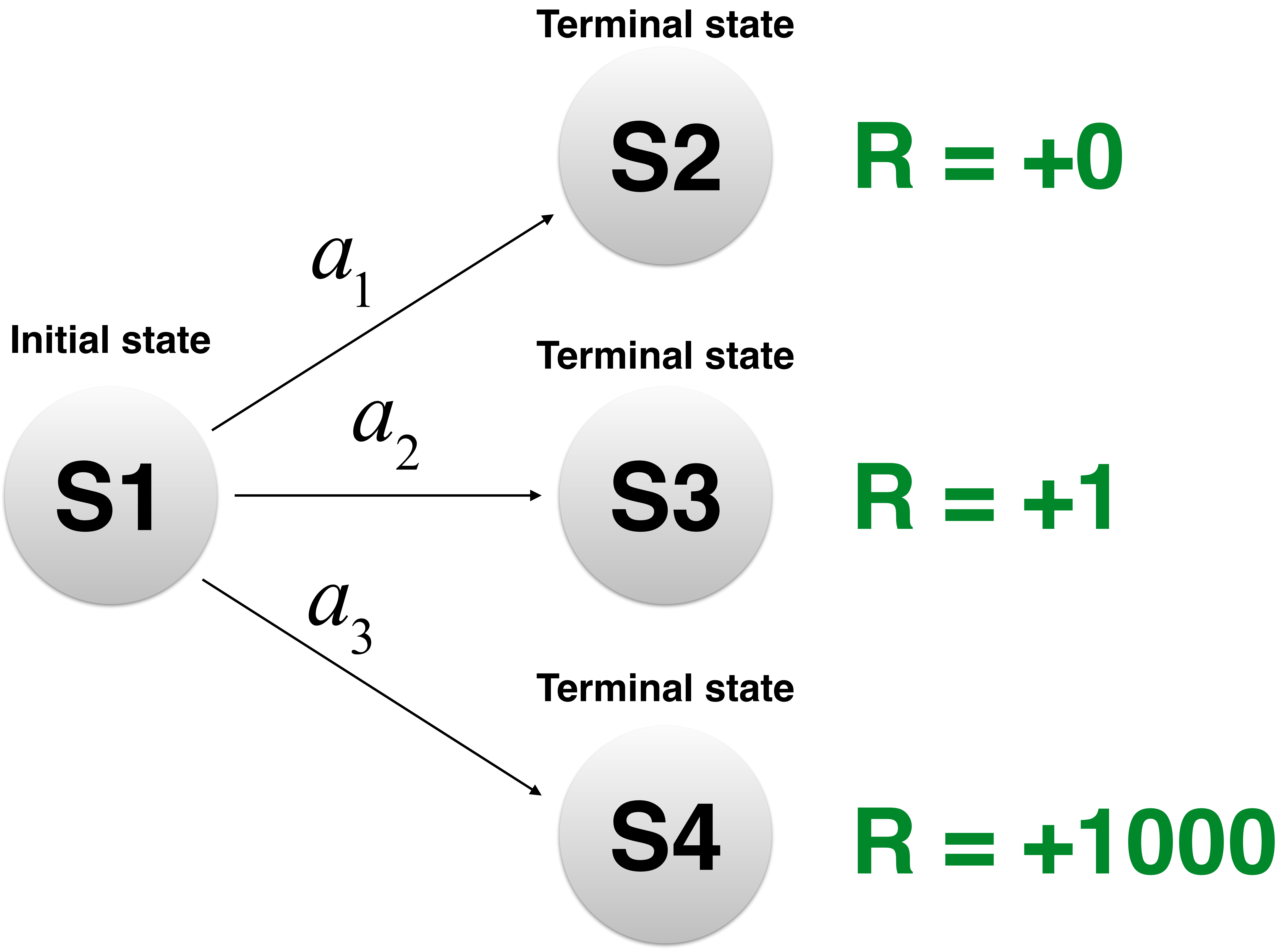
Some training challenges:
- Keep track of terminal step
- Experience replay
- Epsilon greedy action choice (Exploration / Exploitation tradeoff)

The transition resulting from this is added to D, and will not always be used in this iteration's update!

Update using sampled transitions

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

Exploration vs. Exploitation



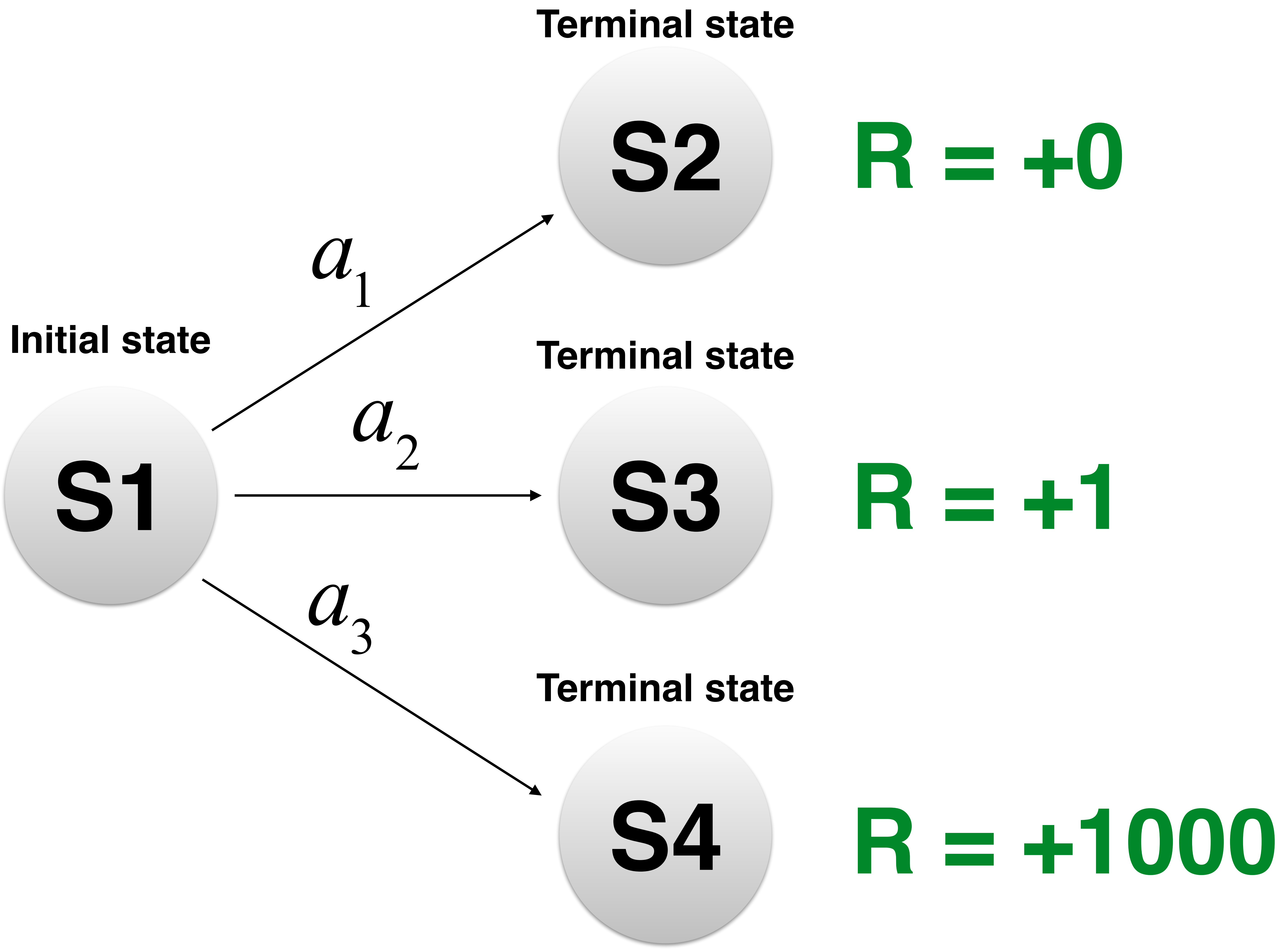
Just after initializing the Q-network, we get:

$$Q(S1, a_1) = 0.5$$

$$Q(S1, a_2) = 0.4$$

$$Q(S1, a_3) = 0.3$$

Exploration vs. Exploitation



Just after initializing the Q-network, we get:

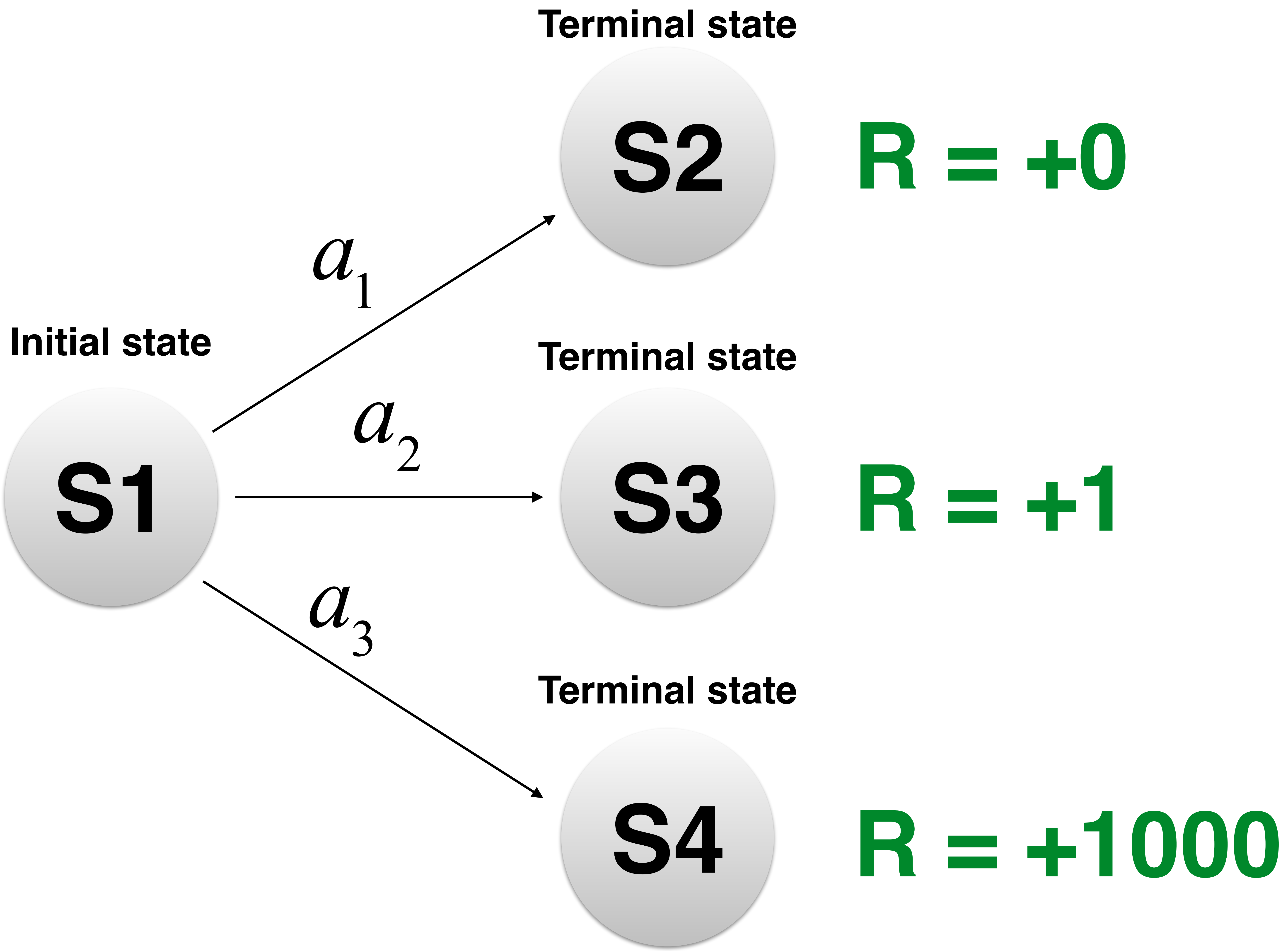
$$Q(S1, a_1) = \cancel{0.5} \ 0$$

$$Q(S1, a_2) = 0.4$$

$$Q(S1, a_3) = 0.3$$

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

Exploration vs. Exploitation



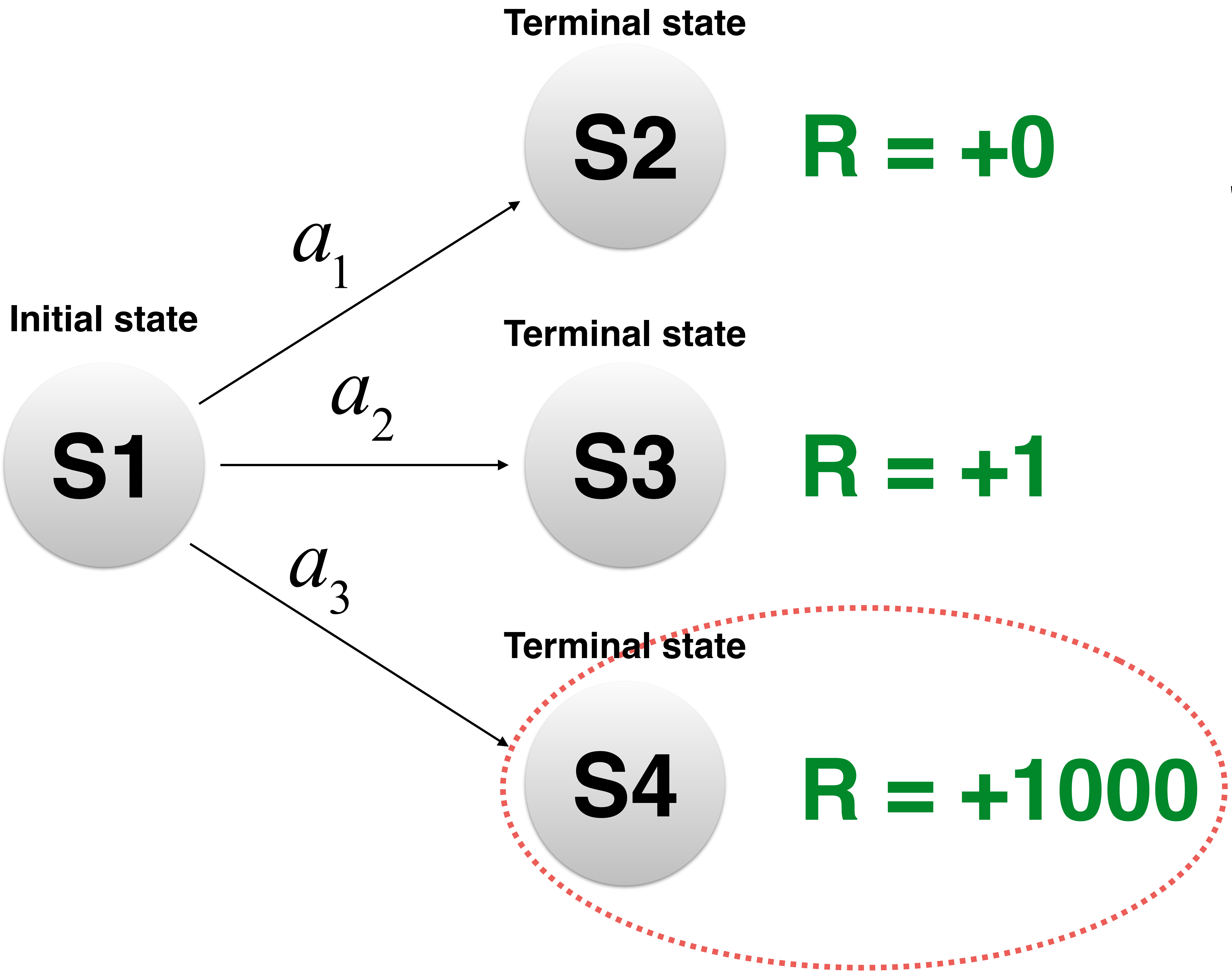
Just after initializing the Q-network, we get:

$$Q(S1, a_1) = \cancel{0.5} \quad 0$$

$$Q(S1, a_2) = \cancel{0.4} \quad 1$$

$$Q(S1, a_3) = 0.3$$

Exploration vs. Exploitation



Just after initializing the Q-network, we get:

$$Q(S1,a_1) = \cancel{0.5} \quad 0$$

$$Q(S1,a_2) = \cancel{0.4} \quad 1$$

$$Q(S1,a_3) = 0.3$$

Will never be visited, because Q(S1,a3) < Q(S1,a2)

Recap' (+ epsilon greedy action)

DQN Implementation:

- *Initialize your Q-network parameters*
- *Initialize replay memory D*

- *Loop over episodes:*

    - *Start from initial state* $\phi(s)$

    - *Create a boolean to detect terminal states: terminal = False*

    - *Loop over time-steps:*

        - ***With probability epsilon, take random action a.***
        - ***Otherwise:***
            - *Forward propagate* $\phi(s)$ *in the Q-network*
            - *Execute action a (that has the maximum Q(*$\phi(s)$*,a) output of Q-network).*

        - *Observe rewards r and next state s'*

        - *Use s' to create* $\phi(s')$

        - *Add experience* $(\phi(s), a, r, \phi(s'))$ *to replay memory (D)*

        - *Sample random mini-batch of transitions from D*

        - *Check if s' is a terminal state. Compute targets y by forward propagating state* $\phi(s')$ *in the Q-network, then compute loss.*

        - *Update parameters with gradient descent*

Overall recap'

DQN Implementation:

- Preprocessing
- Detect terminal state
- Experience replay
- Epsilon greedy action

- *Initialize your Q-network parameters*
- ***Initialize replay memory D***

- *Loop over episodes:*
    - *Start from initial state* $\phi(s)$
    - ***Create a boolean to detect terminal states: terminal = False***
    - *Loop over time-steps:*
        - ***With probability epsilon, take random action a.***
        - ***Otherwise:***
            - *Forward propagate* $\phi(s)$ *in the Q-network*
            - *Execute action a (that has the maximum Q(*$\phi(s)$*,a) output of Q-network).*
        - *Observe rewards r and next state s'*
        - ***Use s' to create*** $\phi(s')$
        - ***Add experience*** $(\phi(s),a,r,\phi(s'))$ ***to replay memory (D)***
        - ***Sample random mini-batch of transitions from D***
        - ***Check if s' is a terminal state****. Compute targets y by forward propagating state* $\phi(s')$ *in the Q-network, then compute loss.*
        - *Update parameters with gradient descent*

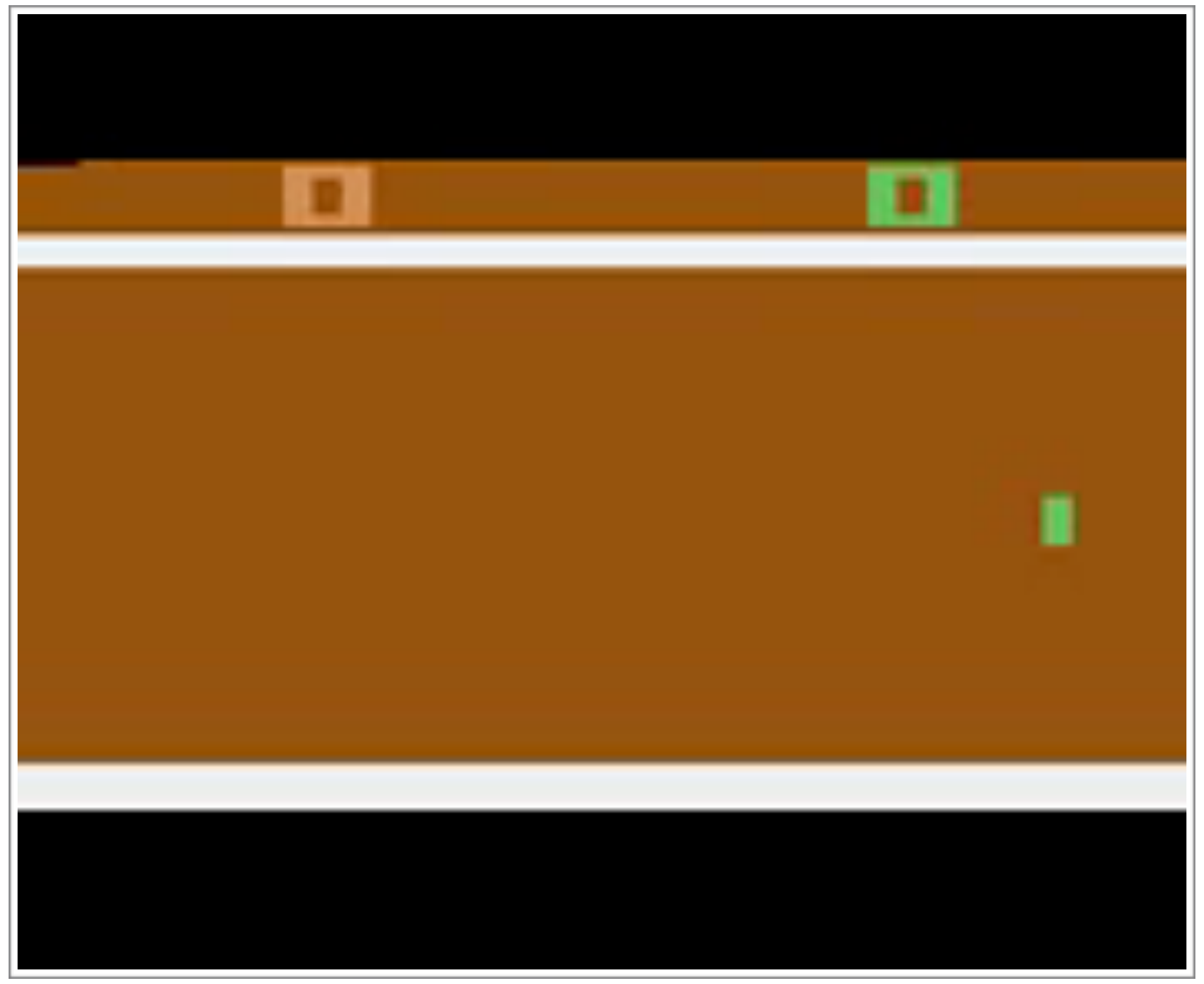
Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

Results

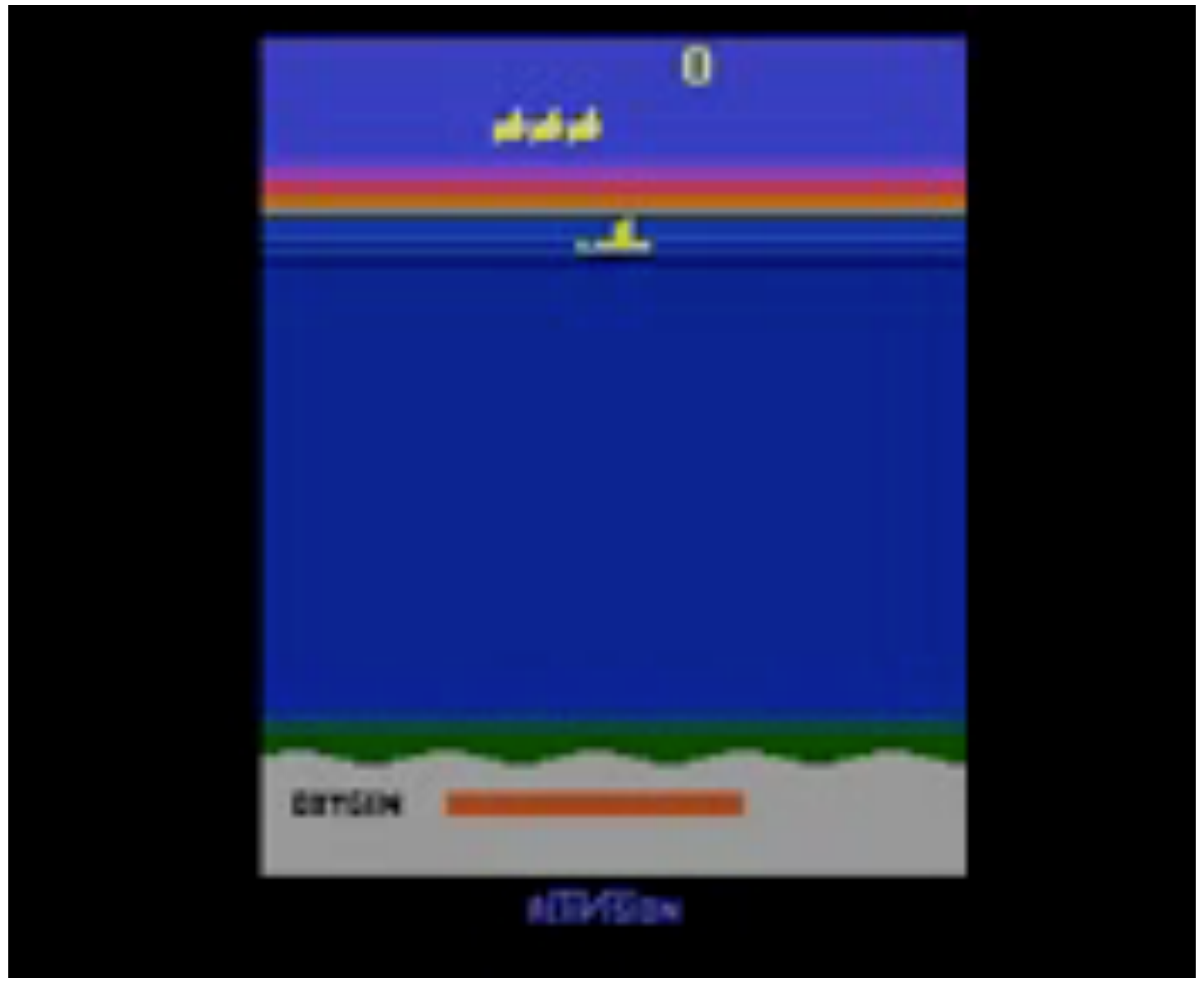
[https://www.youtube.com/watch?v=TmPfTpjtdgg]

Other Atari games

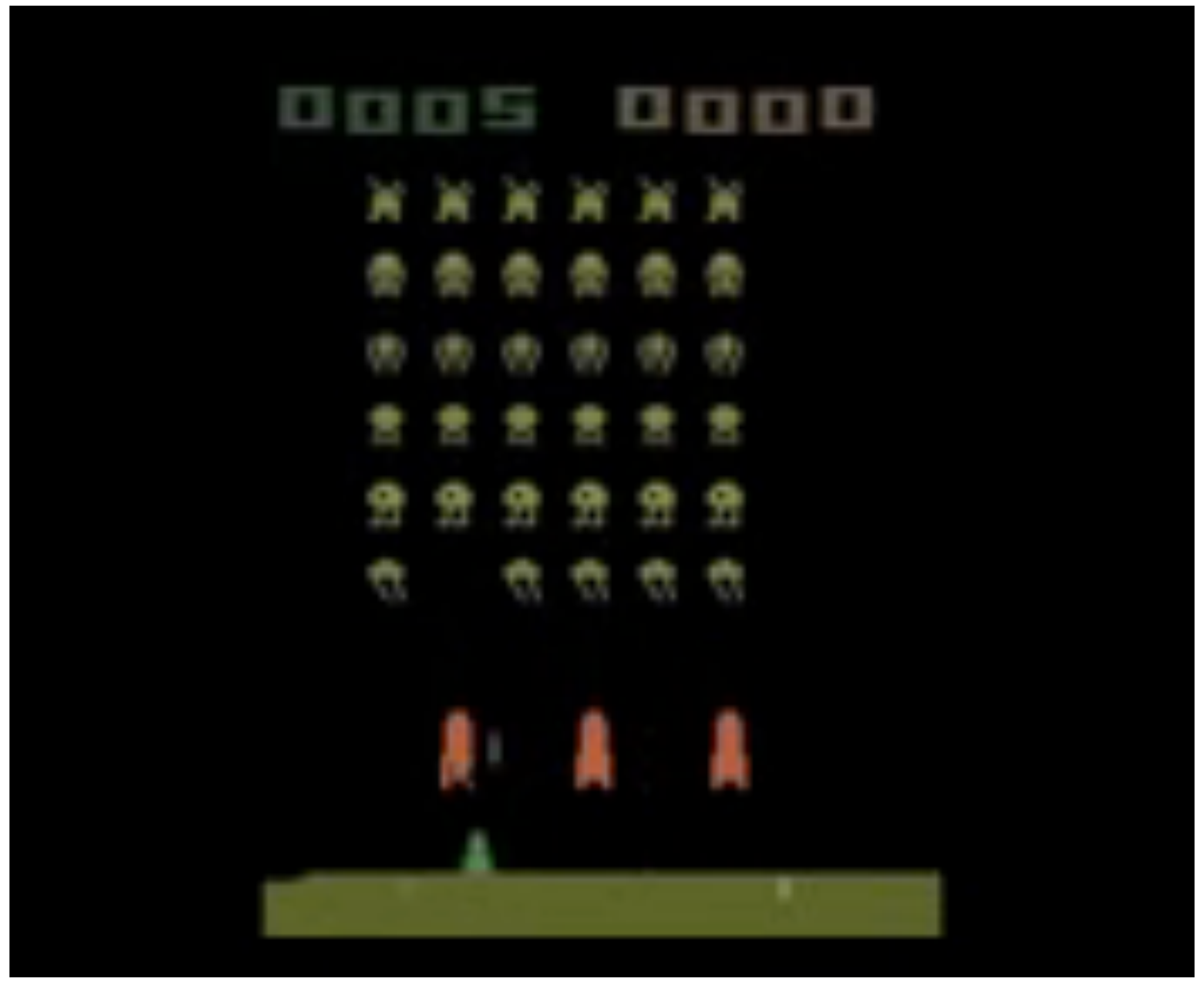
Pong



SeaQuest



Space Invaders



[https://www.youtube.com/watch?v=NirMkC5uvWU]
[https://www.youtube.com/watch?v=p88R2_3yWPA]
[https://www.youtube.com/watch?v=W2CAghUiofY&t=2s]

Today's outline

I. Motivation

II. Recycling is good: an introduction to RL

III. Deep Q-Networks

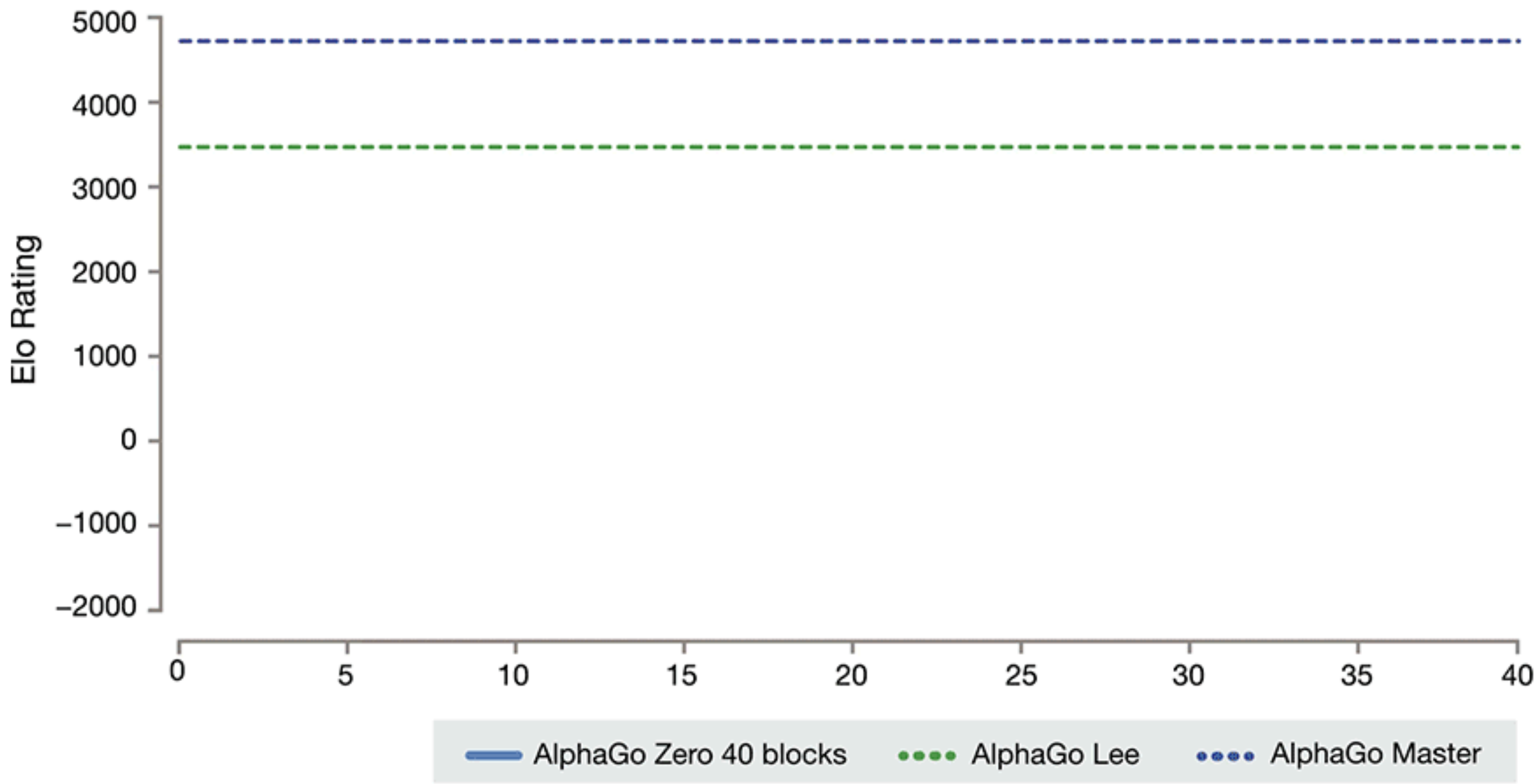
IV. Application of Deep Q-Network: Breakout (Atari)

V. Tips to train Deep Q-Network

VI. Advanced topics

VI - Advanced topics

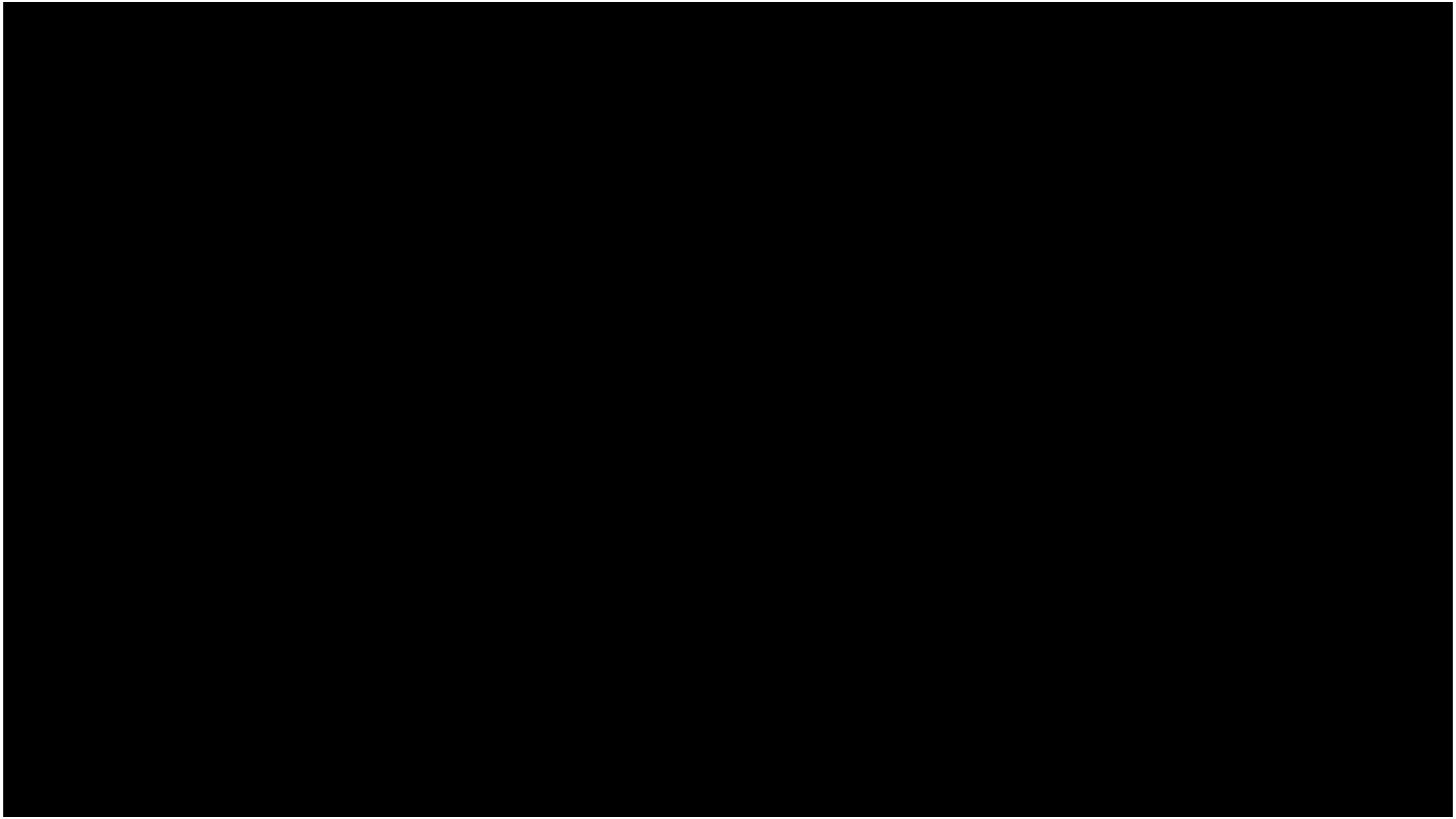
Alpha Go



[DeepMind Blog]

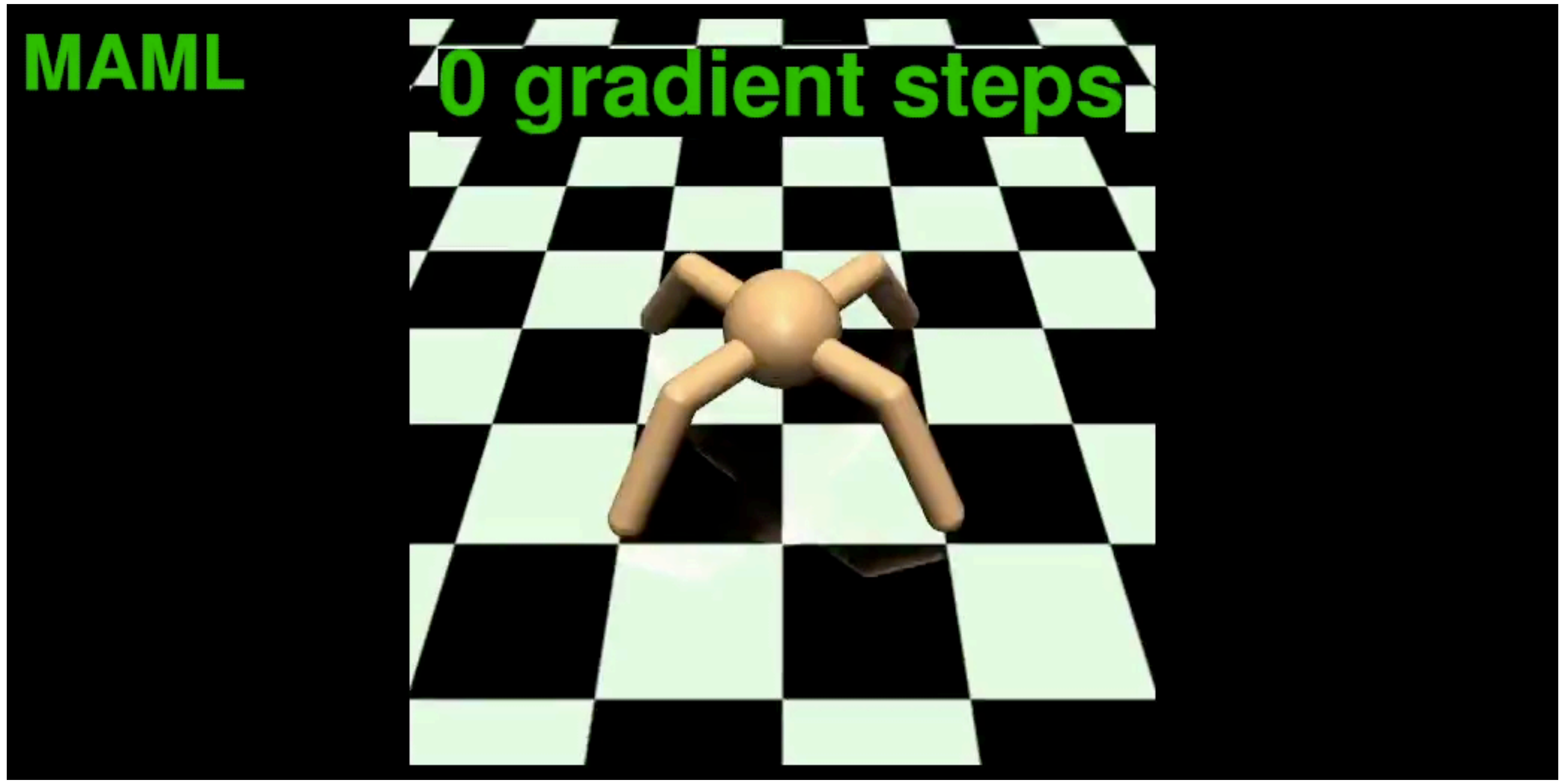
[Silver et al. (2017): Mastering the game of Go without human knowledge]

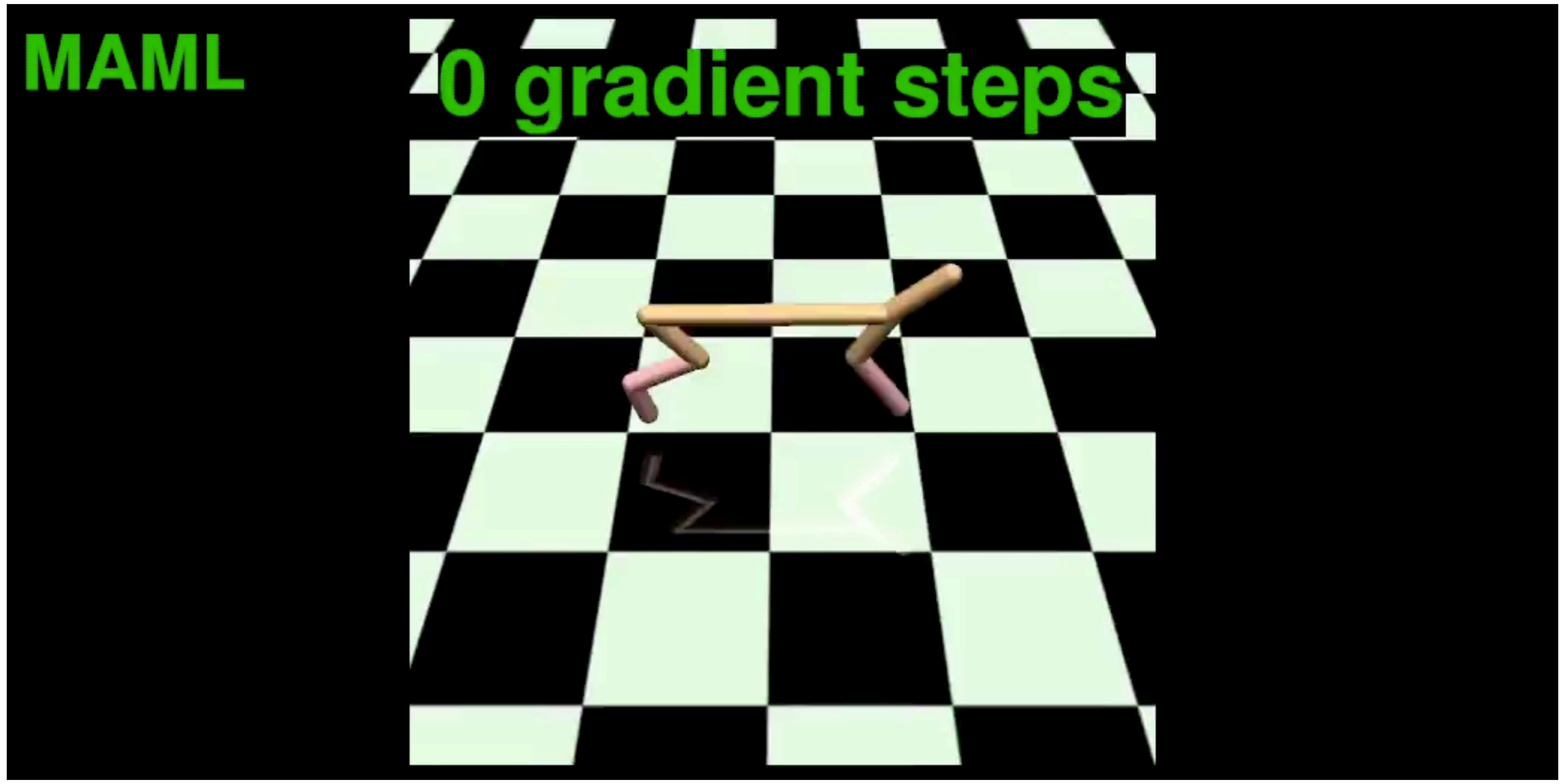
VI - Advanced topics Competitive self-play



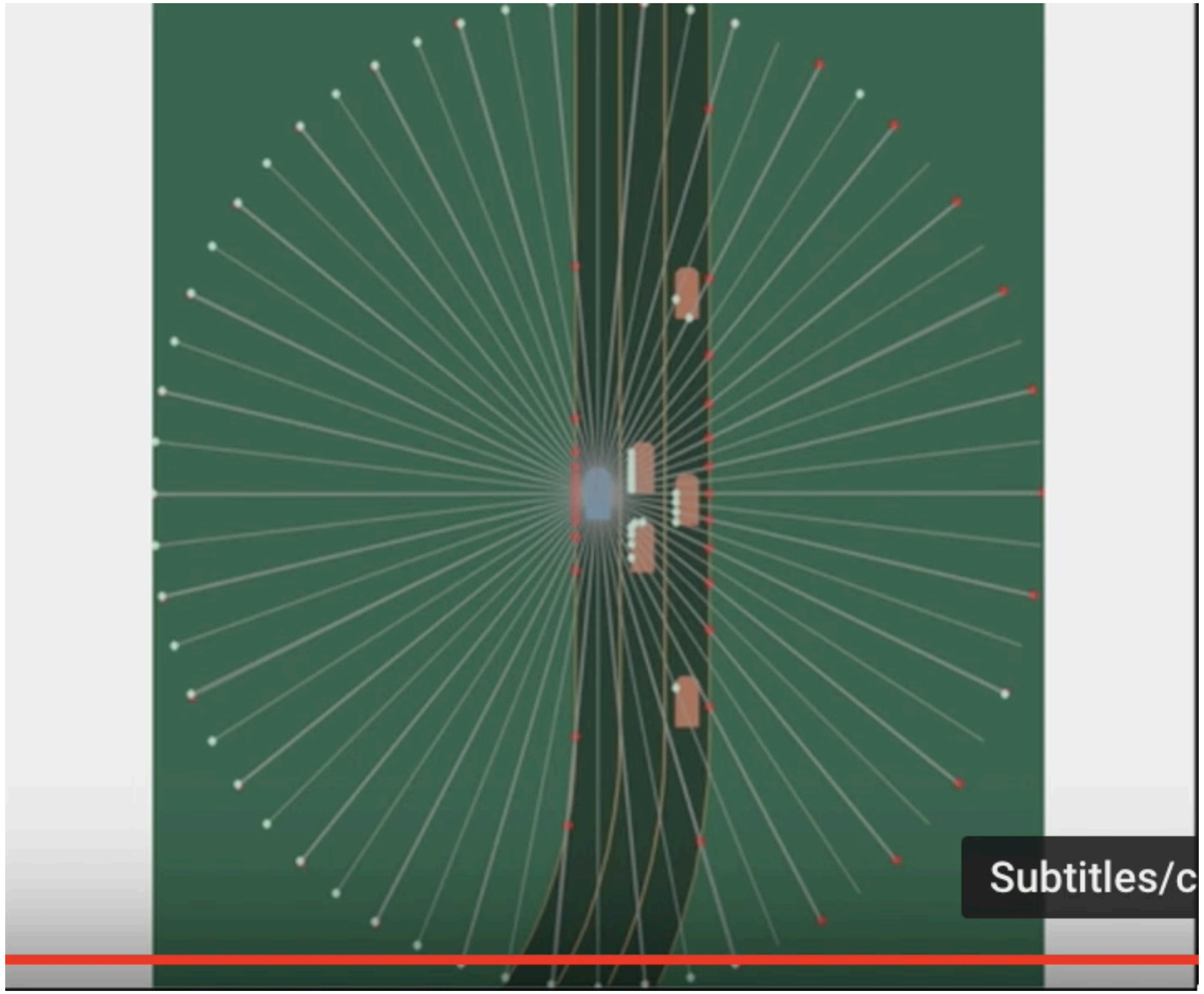
[Bansal et al. (2017): Emergent Complexity via multi-agent competition]
[OpenAI Blog: Competitive self-play]

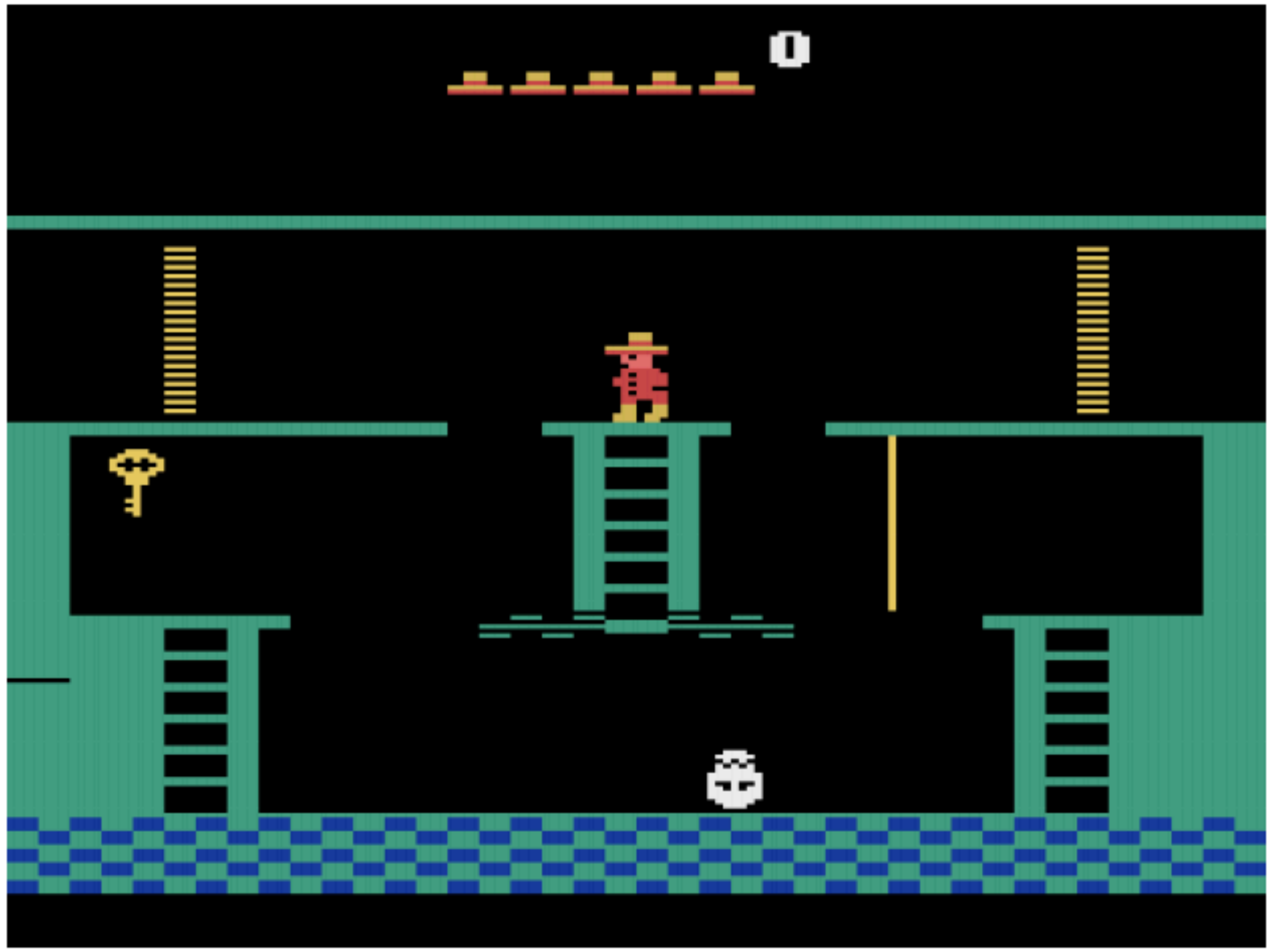
Meta learning





[Finn et al. (2017): Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks]



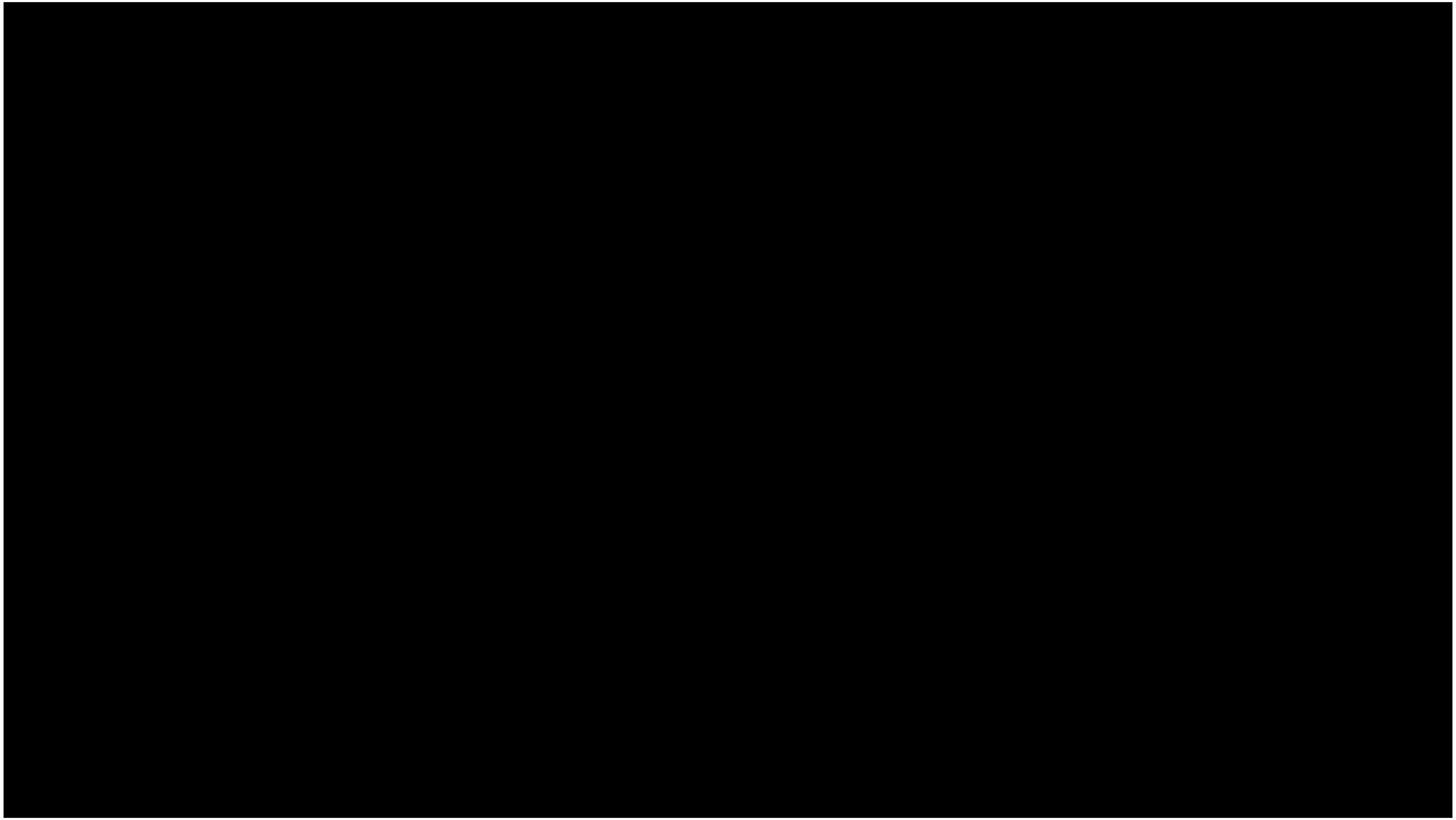


[Source: Bellemare et al. (2016): Unifying Count-Based Exploration and Intrinsic Motivation]

[Ho et al. (2016): Generative Adversarial Imitation Learning]

VI - Advanced topics

Auxiliary task



Announcements

For Tuesday 06/05, 9am:

This Friday:

- **TA Sections:**
  - **How to have a great final project write-up.**
    - **Advices on: How to write a great report.**
    - **Advices on: How to build a super poster.**
    - **Advices on: Final project grading criteria.**
  - **Going through examples of great projects and why they were great.**
  - **Small competitive quiz in section.**