CS230: Lecture 9

Deep Reinforcement Learning

Kian Katanforoosh

Menti code: 21 90 15

Today's outline

I. Motivation

II. Recycling is good: an introduction to RL

III. Deep Q-Learning

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

V. Tips to train Deep Q-Network

VI. Advanced topics

I. Motivation

Mastering the Game of Go without Human Knowledge

David Silver*, Julian Schrittwieser*, Karen Simonyan*, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, Demis Hassabis.

DeepMind, 5 New Street Square, London EC4A 3TW.

*These authors contributed equally to this work.

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, *AlphaGo* became the first program to defeat a world champion in the game of Go. The tree search in *AlphaGo* evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here, we introduce an algorithm based solely on reinforcement learning, without human data, guidance, or domain knowledge beyond game rules. *AlphaGo* becomes its own teacher: a neural network is trained to predict *AlphaGo*'s own move selections and also the winner of *AlphaGo*'s games. This neural network improves the strength of tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program *AlphaGo Zero* achieved superhuman performance, winning 100-0 against the previously published, champion-defeating *AlphaGo*.

AlphaGo

Human Level Control through Deep Reinforcement Learning

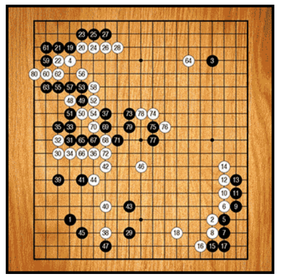
[Silver, Schrittwieser, Simonyan et al. (2017): Mastering the game of Go without human knowledge]

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

I. Motivation

How would you solve Go with classic supervised learning?

Input

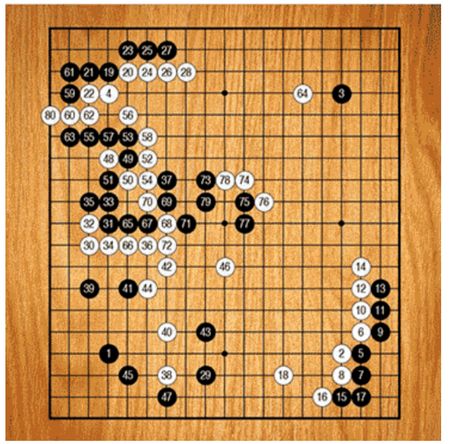


output class

The move of the professional player

issues:

- Ground truth probably wrongly defined.
- Two many states in this Game.
- We will likely not generalize.



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

Why RL?

- **Delayed labels**
- **Making sequences of decisions**

What is RL?

- **Automatically learn to make good sequences of decision**

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

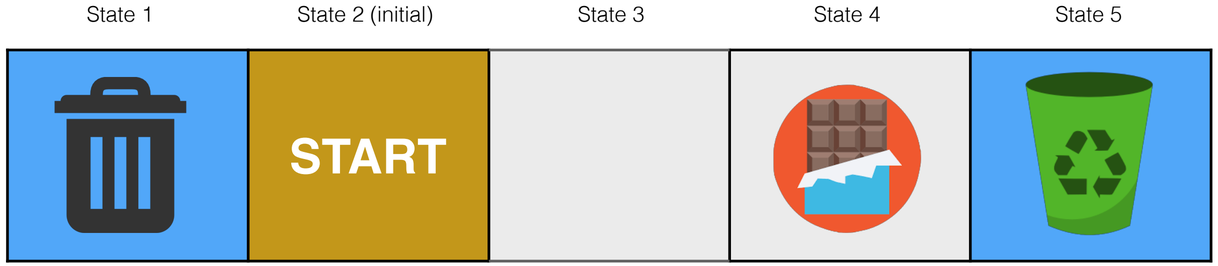
IV. Application of Deep Q-Network: Breakout (Atari)

V. Tips to train Deep Q-Network

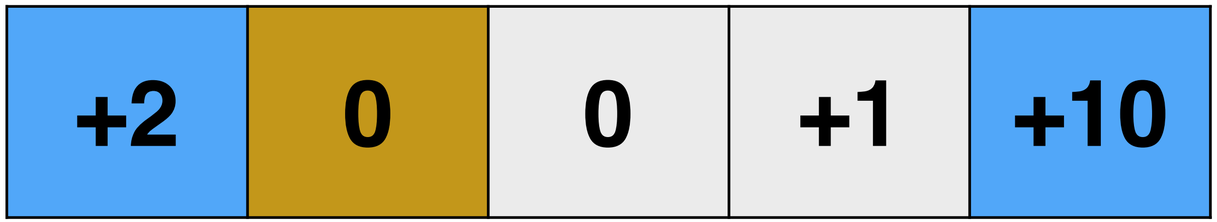
VI. Advanced topics

II. Recycling is good: an introduction to RL

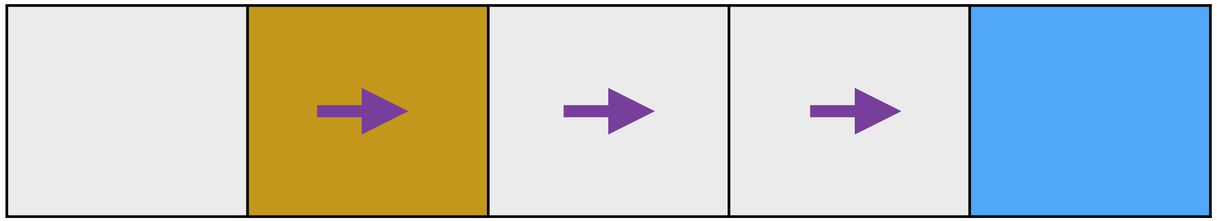
Problem statement



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



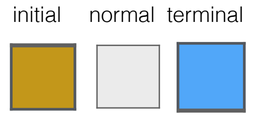
Best strategy to follow if $\gamma = 1$



***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*

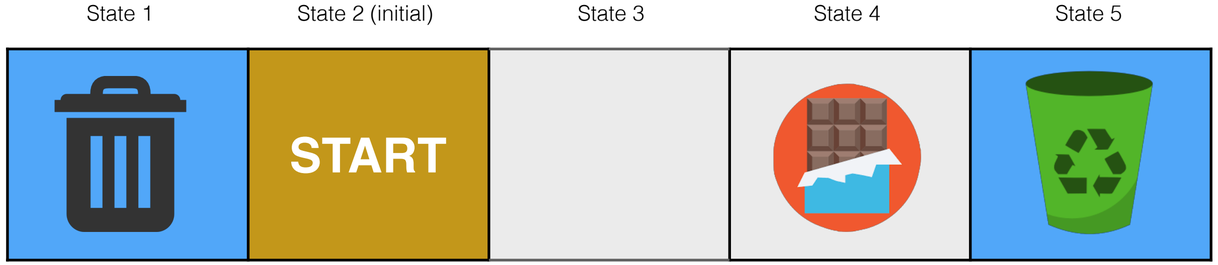
How to define the long-term return?

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

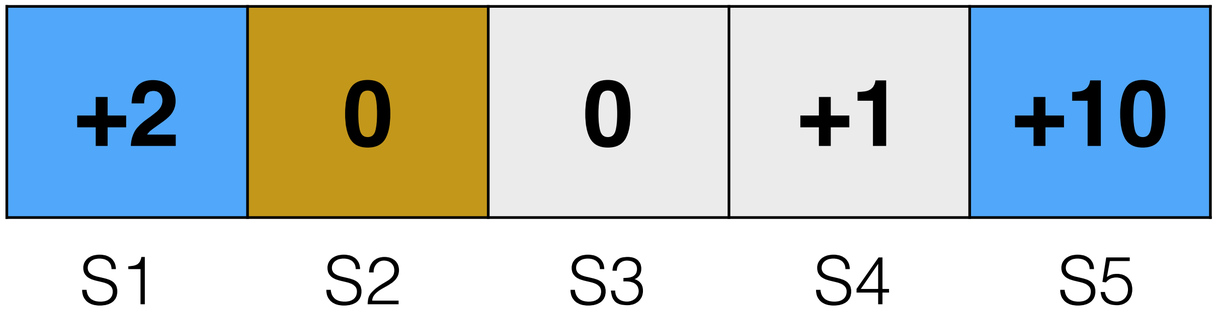
Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement



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



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

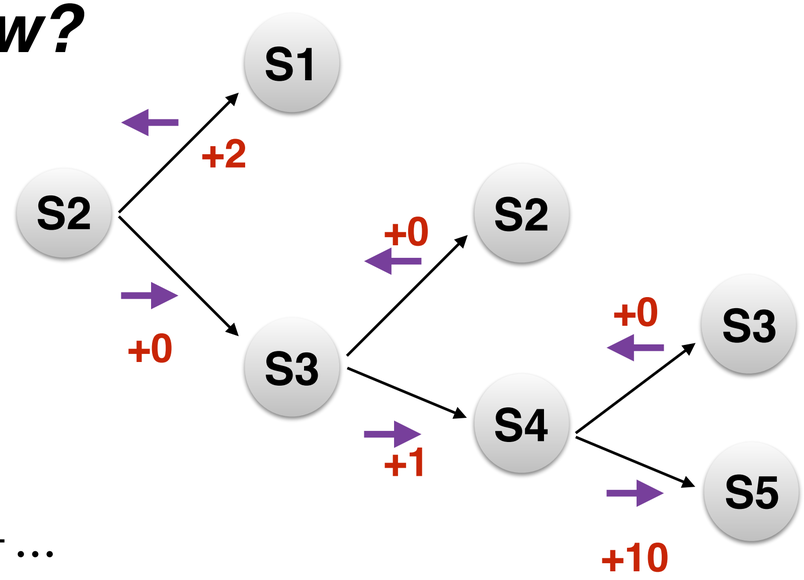
Q-table

$$Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix}$$

#actions

#states

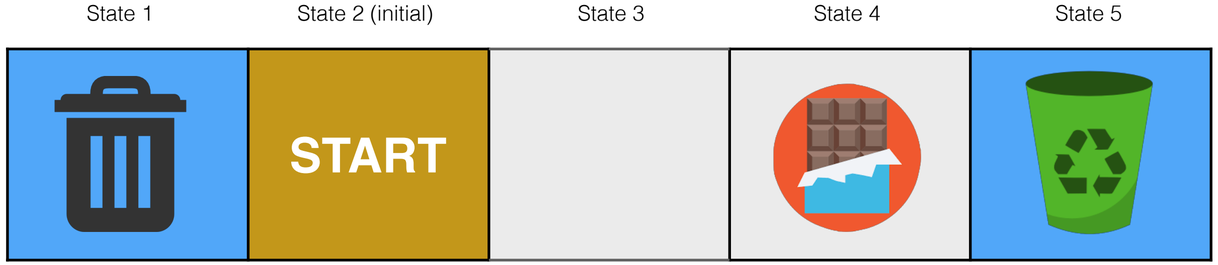
How?



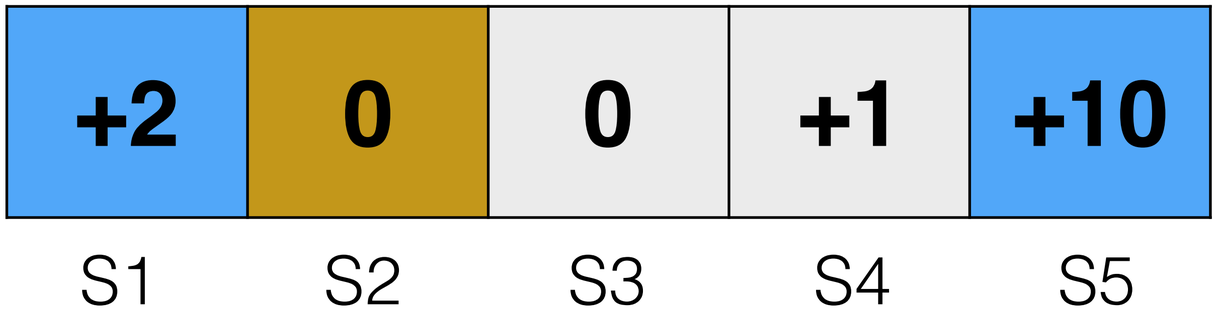
Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement



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



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

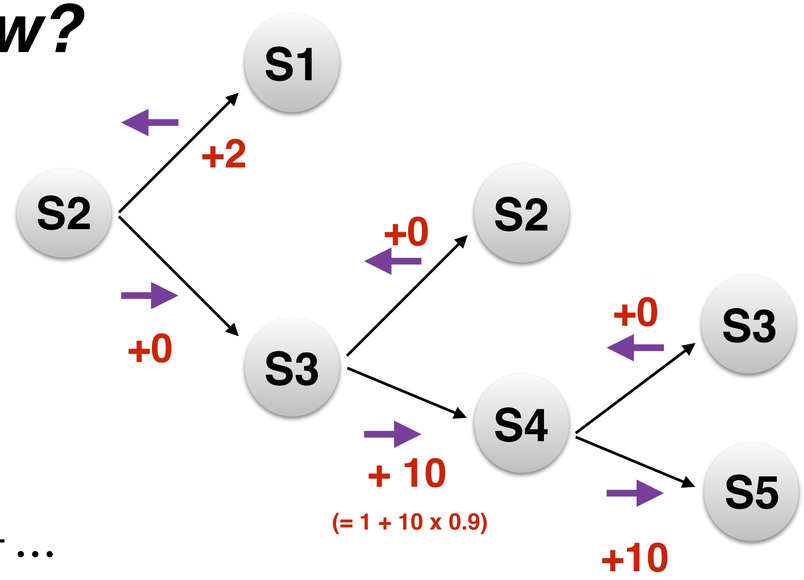
Q-table

$$Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix}$$

#actions

#states

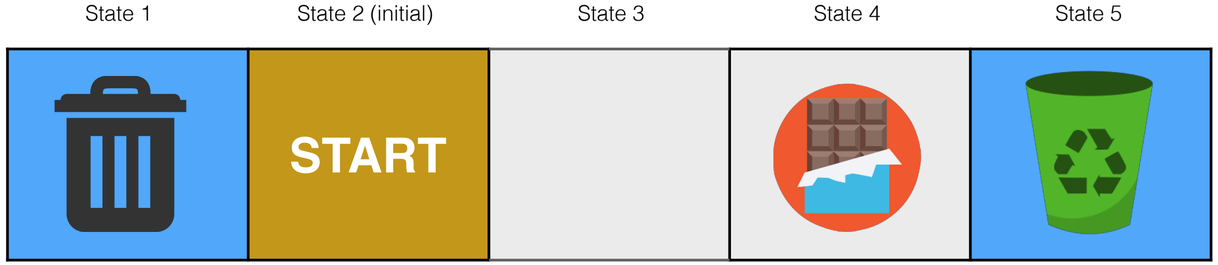
How?



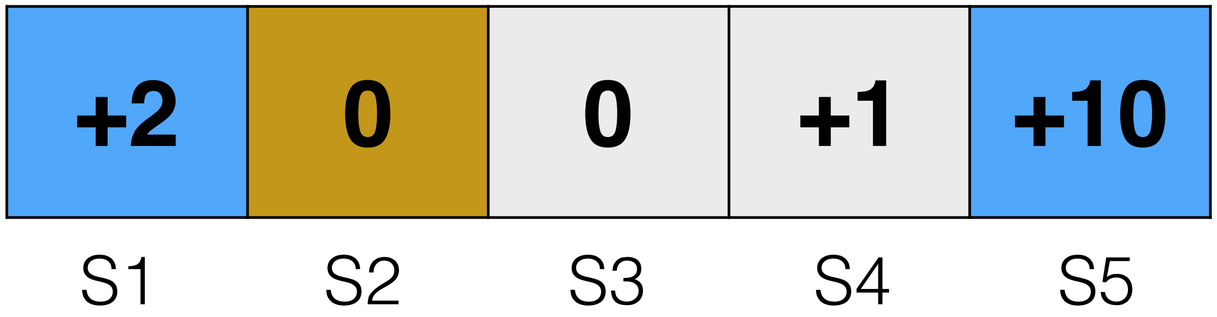
Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement



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



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

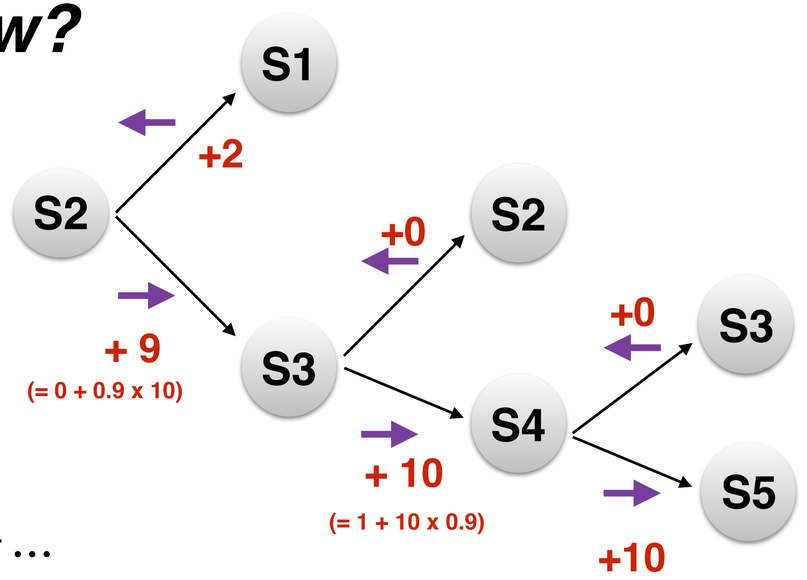
Q-table

$$Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix}$$

#actions

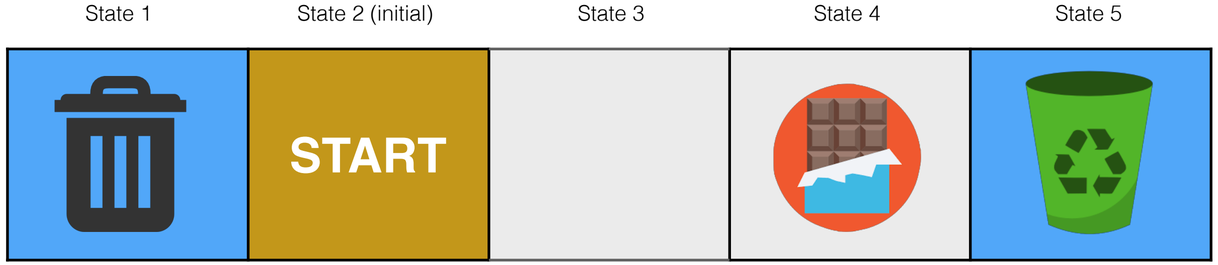
#states

How?

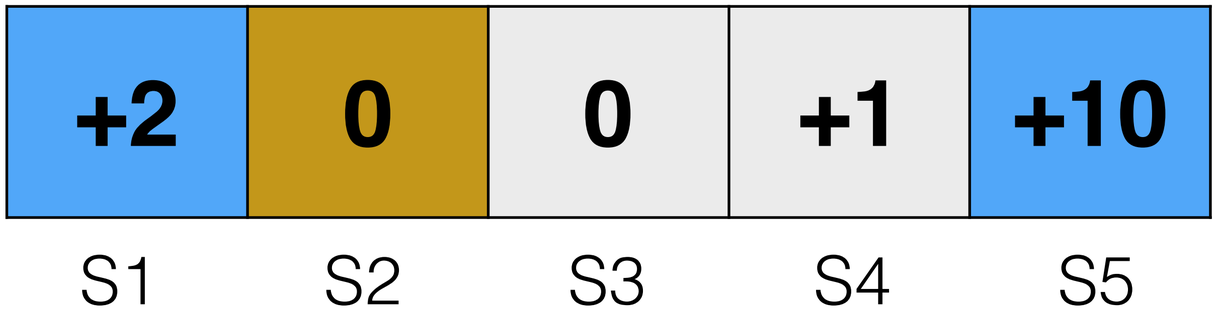


II. Recycling is good: an introduction to RL

Problem statement



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



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

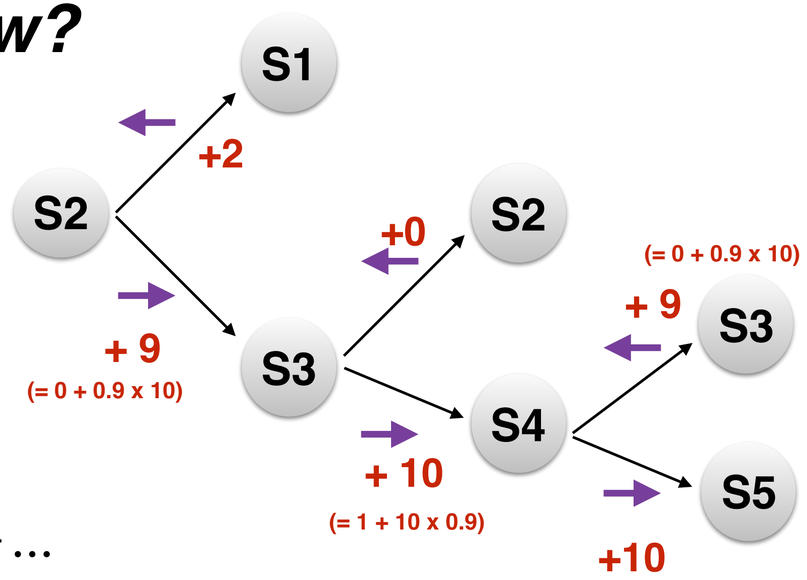
Q-table

$$Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix}$$

#actions

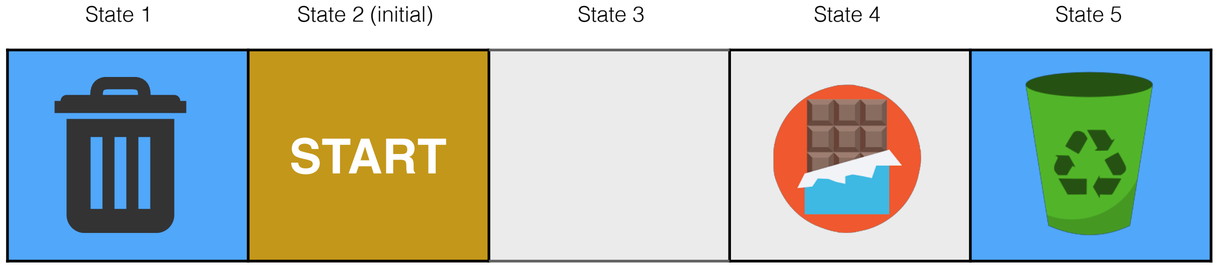
#states

How?

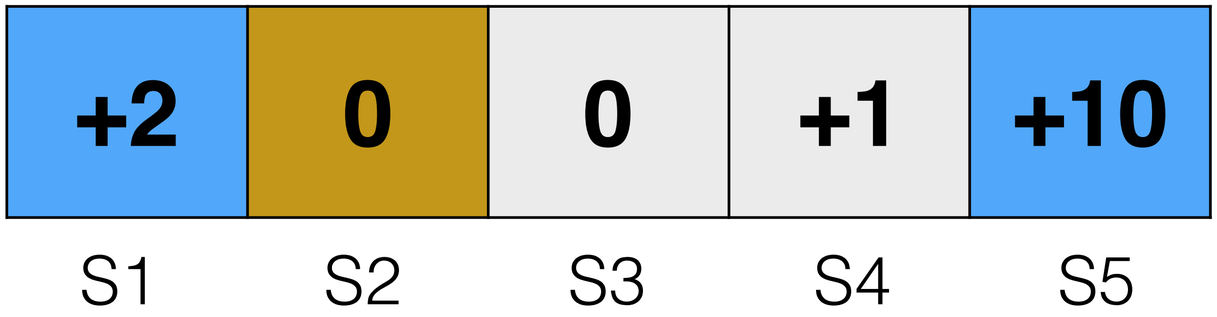


II. Recycling is good: an introduction to RL

Problem statement



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



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

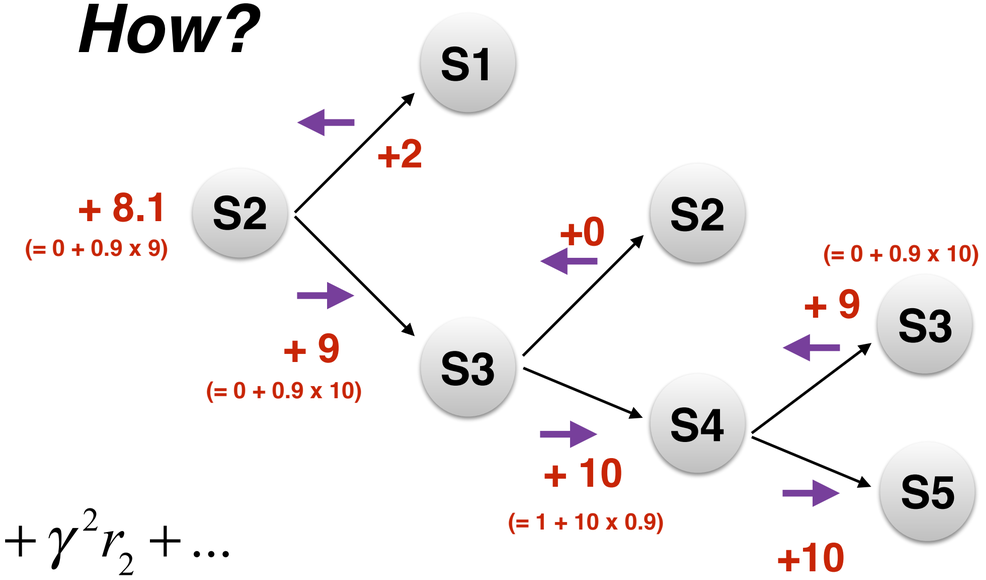
Q-table

$$Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix}$$

#actions

#states

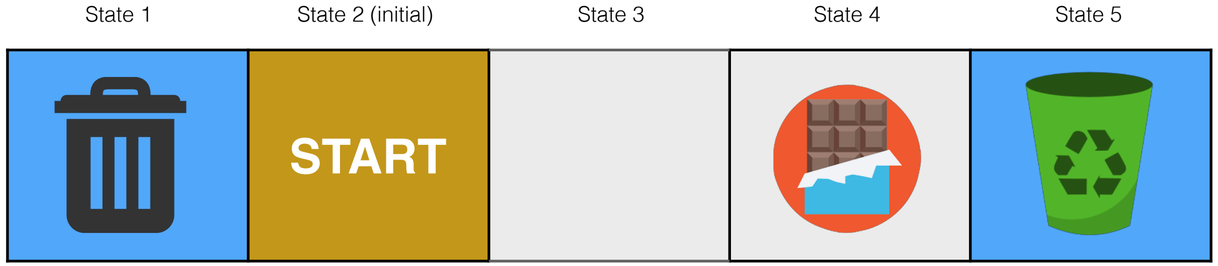
How?



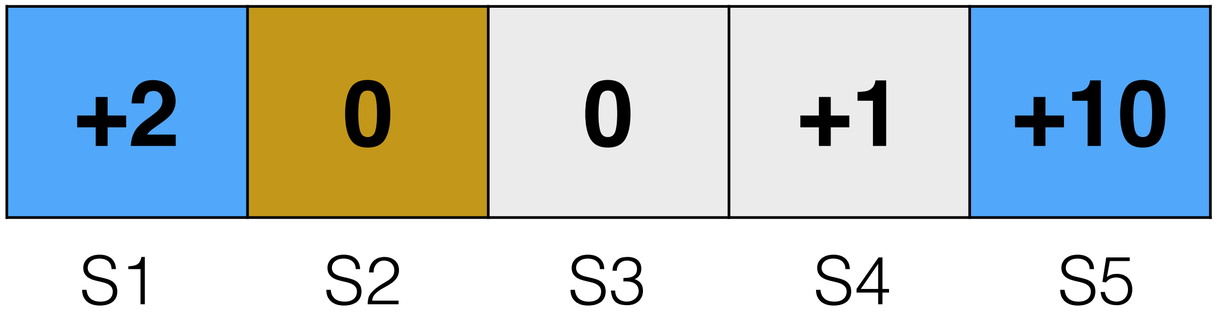
Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement



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



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

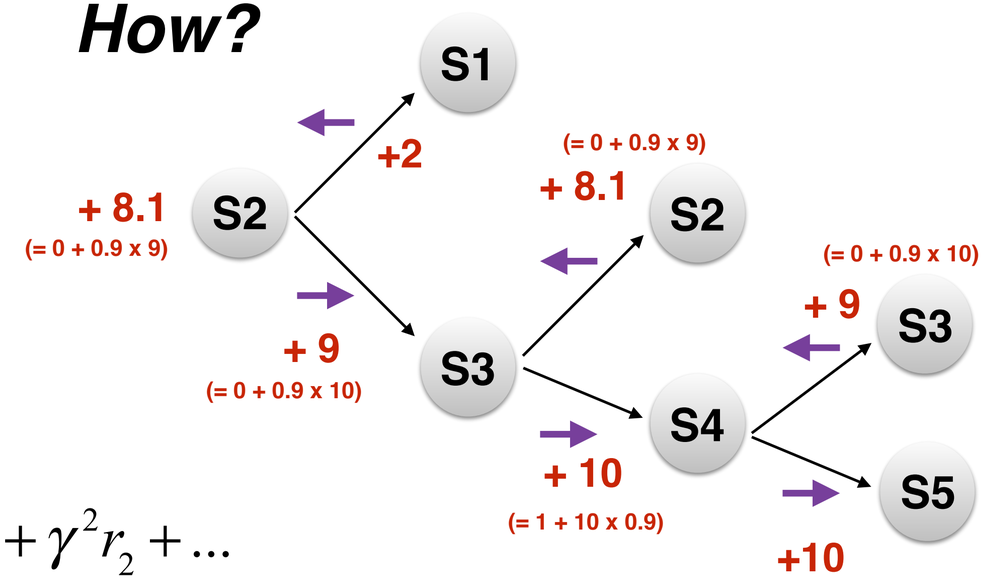
Q-table

$$Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix}$$

#actions

#states

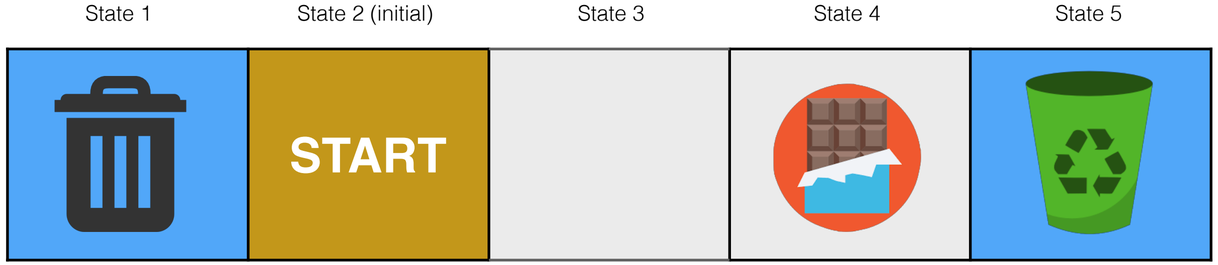
How?



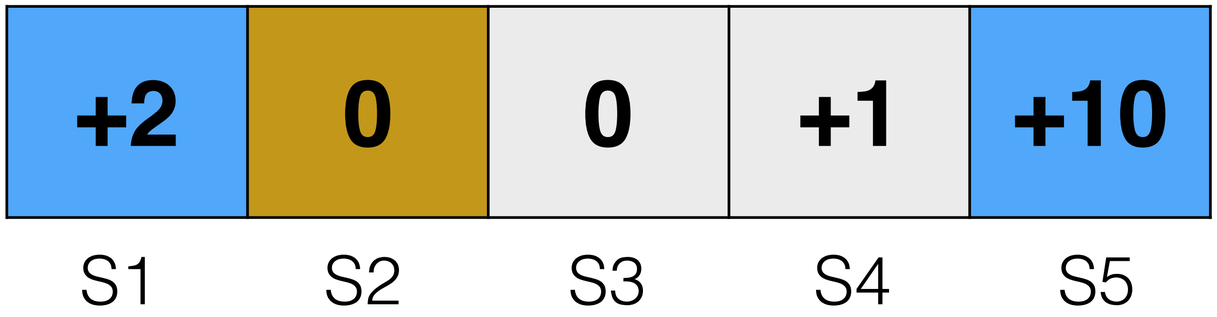
Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

Problem statement



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



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

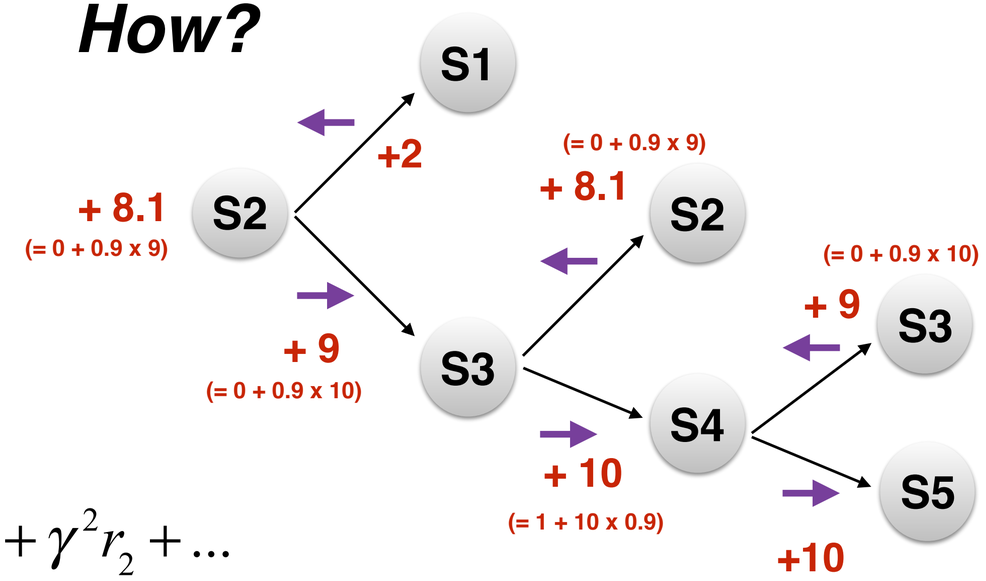
Q-table

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

#actions

#states

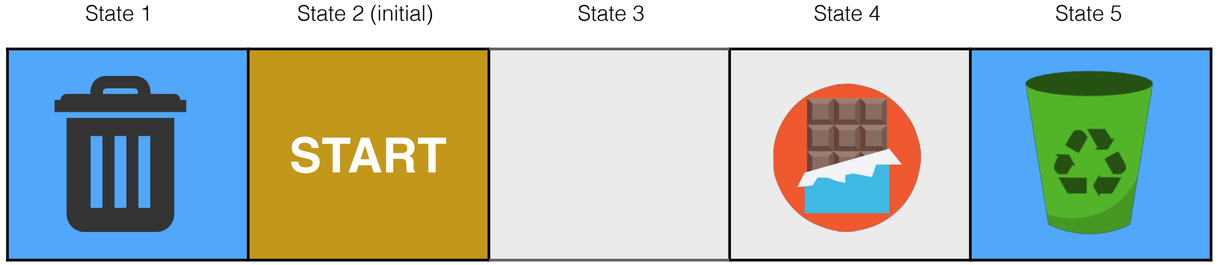
How?



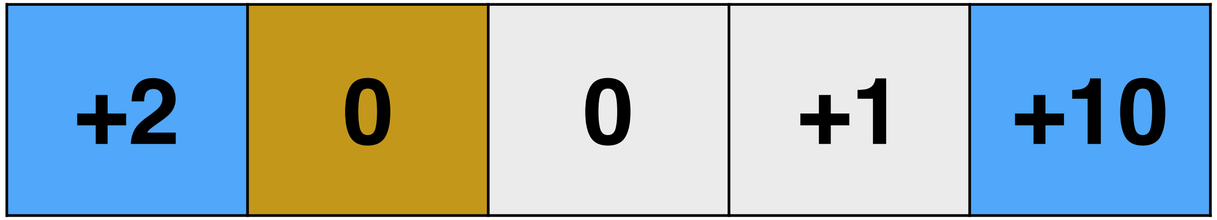
Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

II. Recycling is good: an introduction to RL

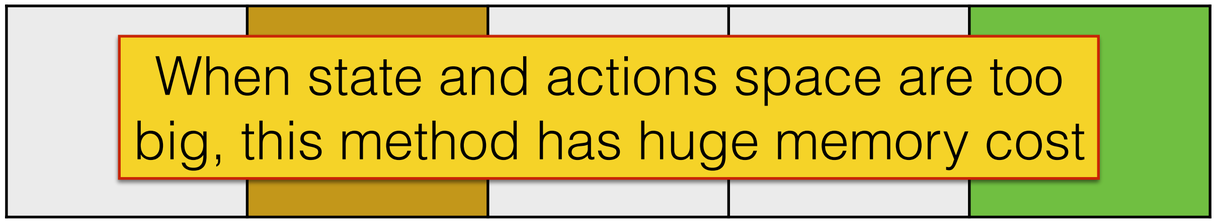
Problem statement



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



Best strategy to follow if $\gamma = 0.9$



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

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

#actions

#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-Learning

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

V. Tips to train Deep Q-Network

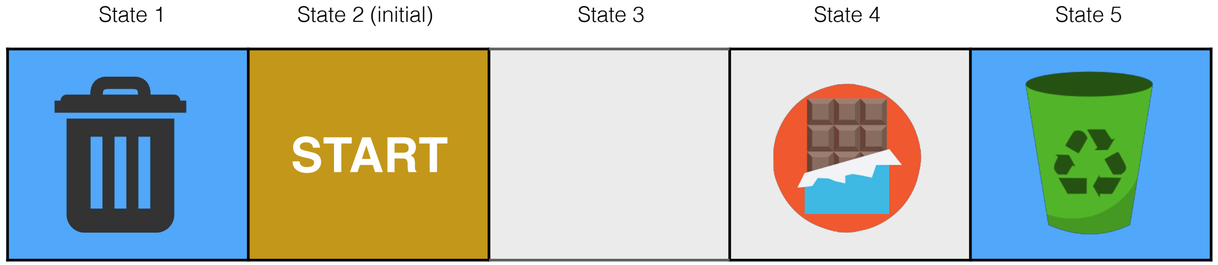
VI. Advanced topics

III. Deep Q-Learning

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

Neural Network

Problem statement



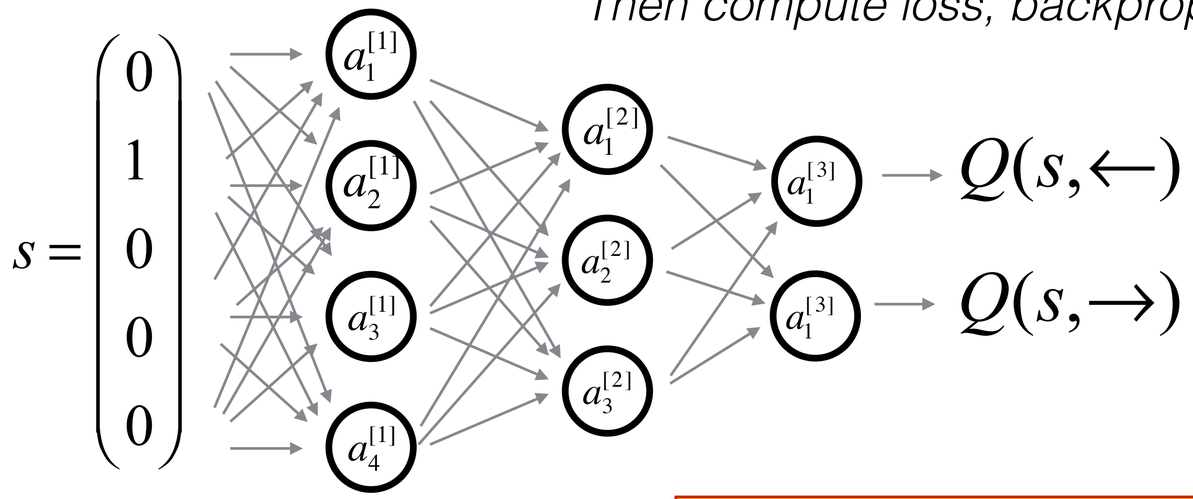
Q-table

#actions

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

#states

Then compute loss, backpropagate.

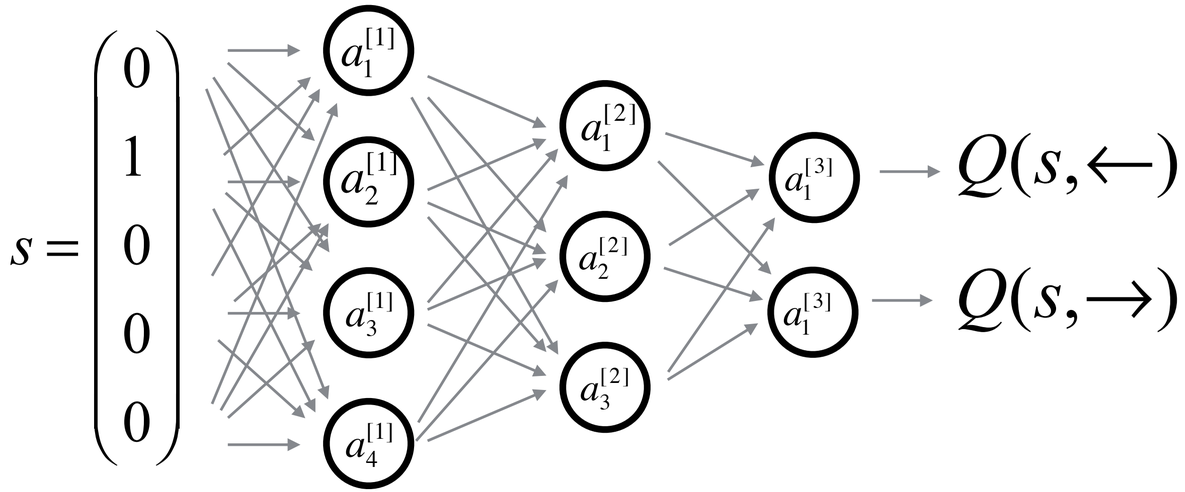


How to compute the loss?

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

III. Deep Q-Learning

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



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 ← 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

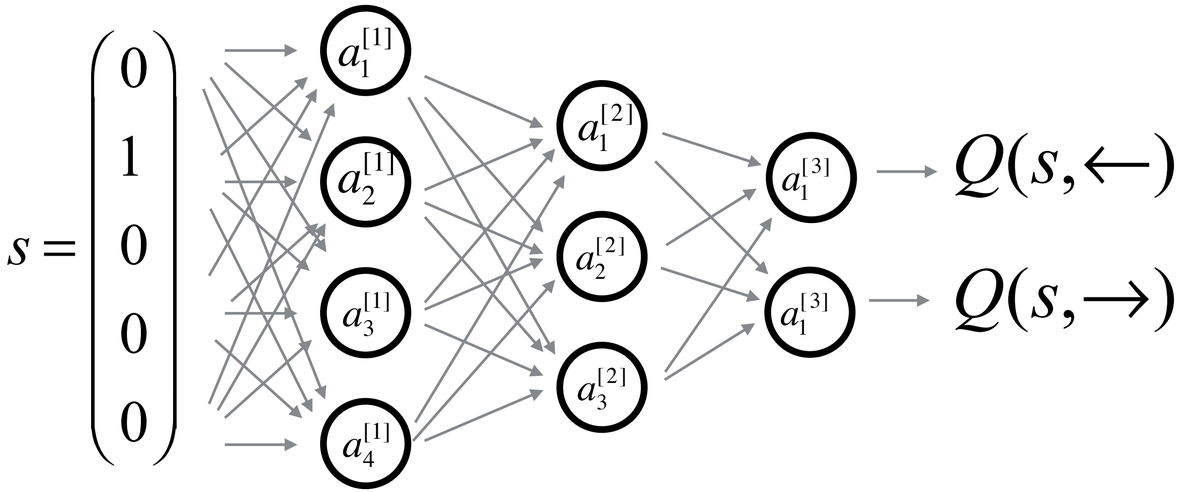
Immediate Reward for taking action → in state s

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

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

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

III. Deep Q-Learning



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

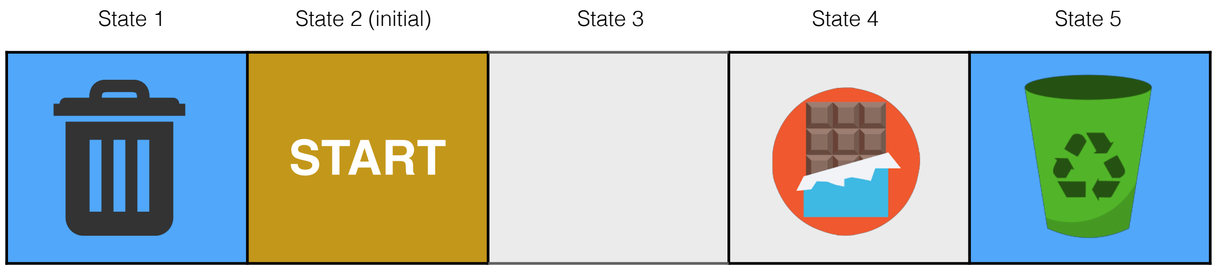
$$y = r_{\rightarrow} + \gamma \max_{a'}(Q(s_{\rightarrow}^{next}, a'))$$

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

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

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 reward 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

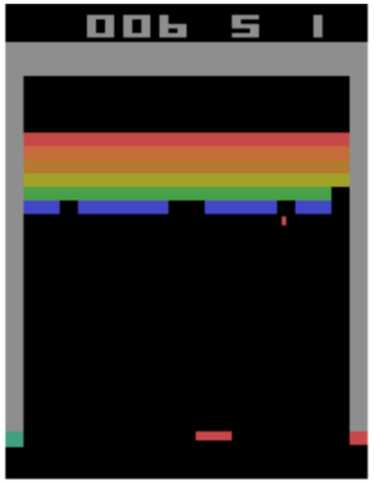
IV. Deep Q-Learning application: Breakout (Atari)

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?

[Video credits to Two minute papers: Google DeepMind's Deep Q-learning playing Atari Breakout https://www.youtube.com/watch?v=V1eYniJ0Rnk]

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

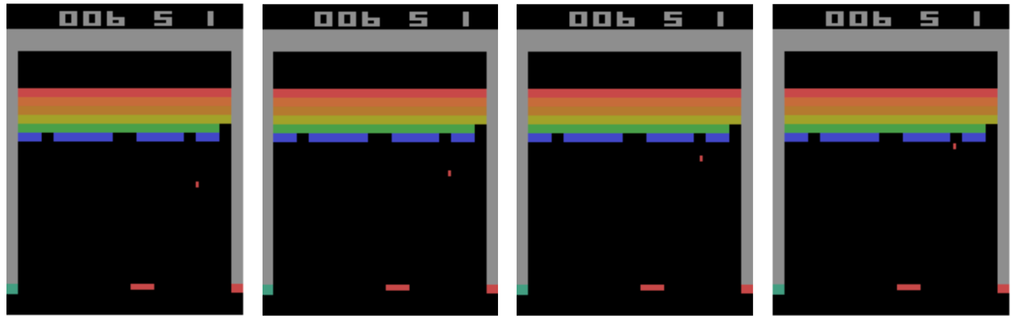
IV. Deep Q-Learning application: Breakout (Atari)

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

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

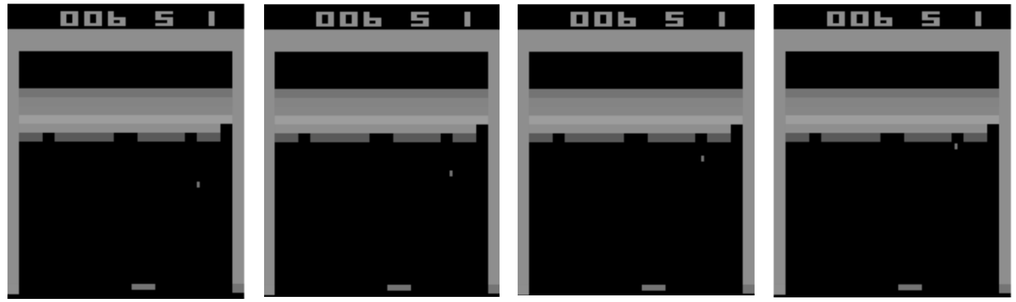
What is done in preprocessing?

[Video credits to Two minute papers: Google DeepMind's Deep Q-learning playing Atari Breakout https://www.youtube.com/watch?v=V1eYniJ0Rnk]

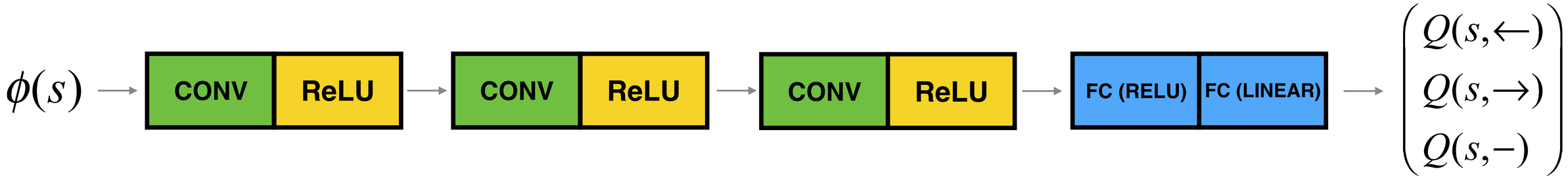
[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

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

input of Q-network

$\phi(s) =$ 

Deep Q-network architecture?



[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

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:* $\phi(s)$
    - *Forward propagate s in the Q-network*
    - *Execute action a (that has the maximum Q(s,a) output of Q-network)* $\phi(s)$
    - *Observe reward r and next state s'*
    - ***Use s' to create*** $\phi(s')$
    - *Compute targets y by forward propagating state s' in the Q-network, then compute loss.* $\phi(s')$
    - *Update parameters with gradient descent*

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

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:* $\phi(s)$
    - *Forward propagate s in the Q-network*
    - *Execute action a (that has the maximum Q(s,a) output of Q-network)* $\phi(s)$
    - *Observe reward r and next state s'*
    - ***Use s' to create*** $\phi(s')$
    - *Compute targets y by forward propagating state s' in the Q-network, then compute loss.* $\phi(s')$
    - *Update parameters with gradient descent*

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

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* ~~$\phi(s)$~~ *in the Q-network*
    - *Execute action a (that has the maximum Q(*~~$\phi(s)$~~*,a) output of Q-network)*
    - *Observe reward 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* ~~$\phi(s')$~~ *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)

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

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

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

• • •

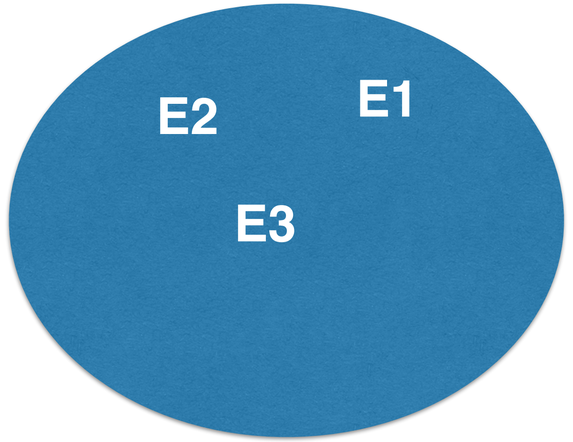
Experience Replay

E1

E2

E3

…



Replay memory (D)

Training: E1 → E2 → E3

Training: E1 → sample(E1, E2) → sample(E1, E2, E3) → sample(E1, E2, E3, E4) → …

Can be used with mini batch gradient descent

Advantages of experience replay?

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

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 reward 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*

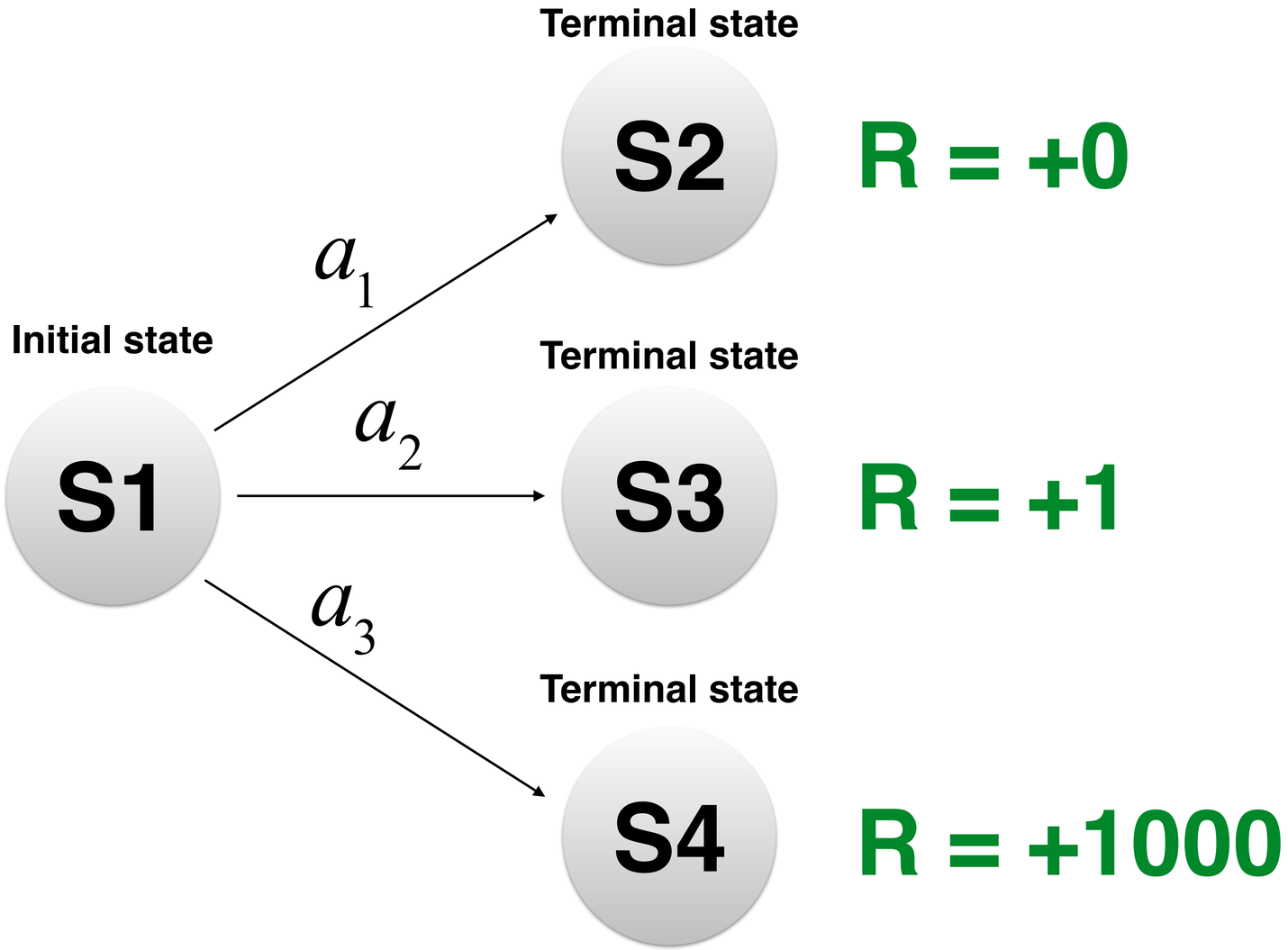
Some training challenges:
- Keep track of terminal step
- Experience replay
- Epsilon greedy action choice (Exploration / Exploitation tradeoff)

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

Update using sampled transitions

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

Exploration vs. Exploitation



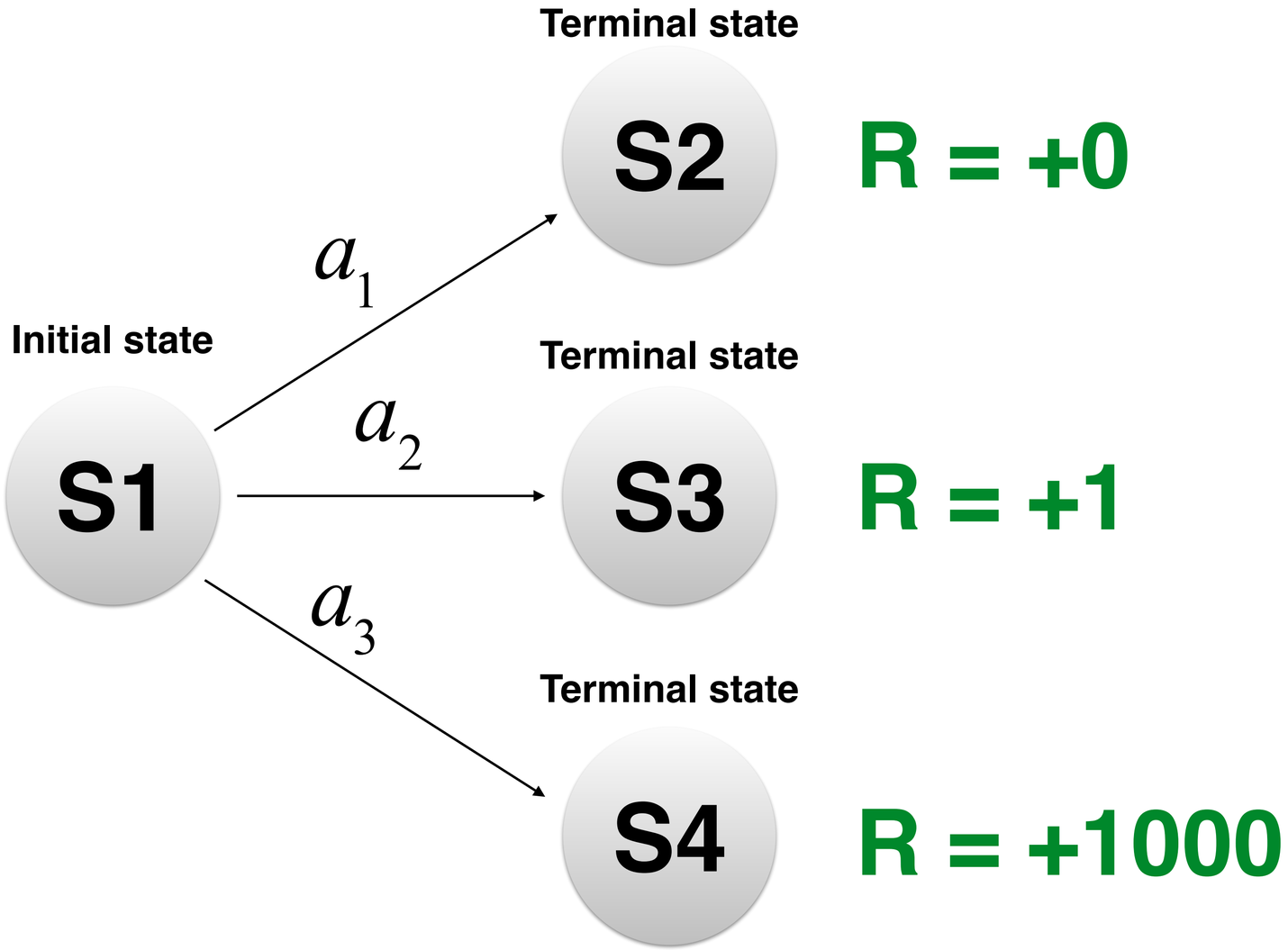
Just after initializing the Q-network, we get:

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

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

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

Exploration vs. Exploitation



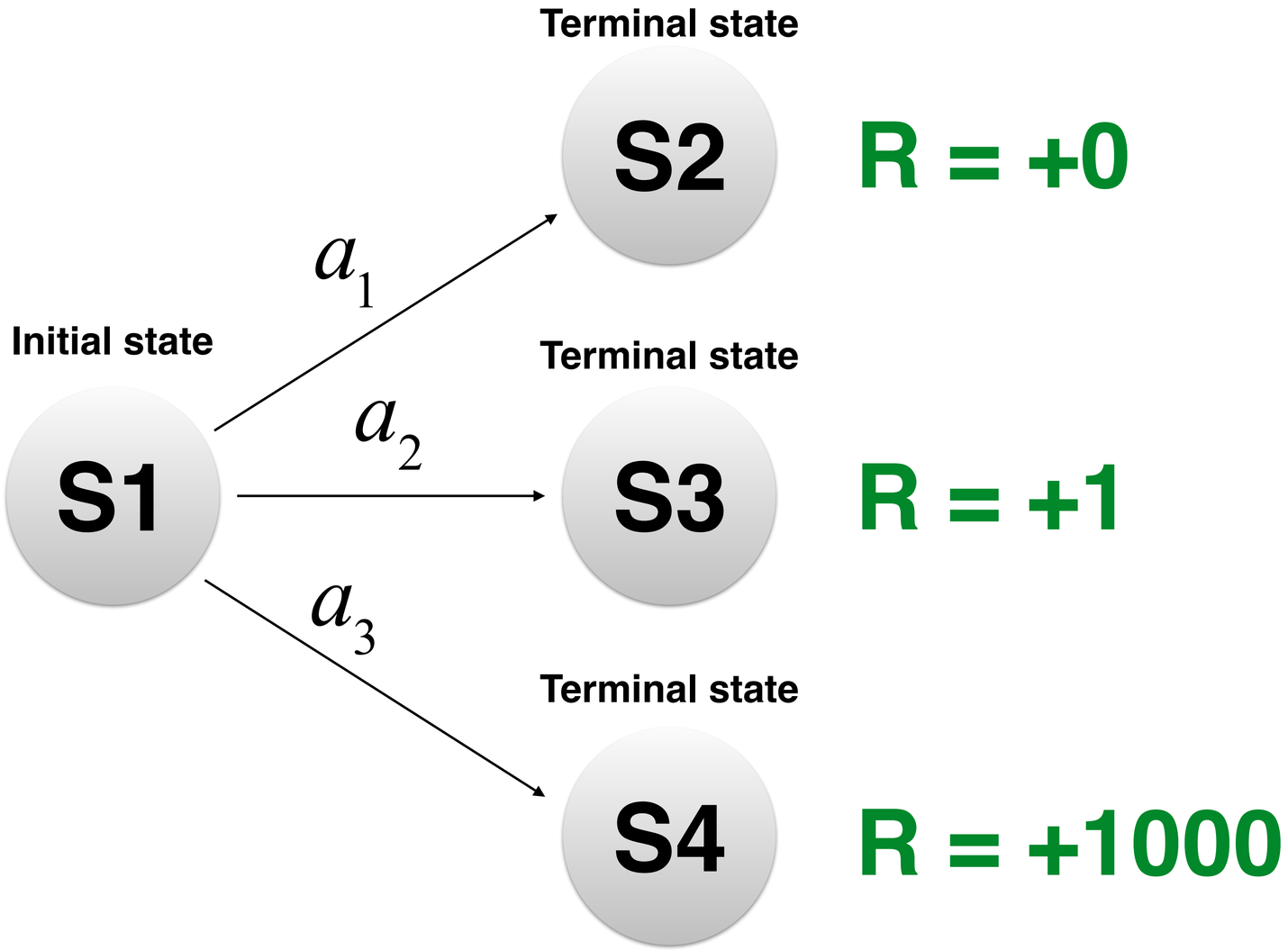
Just after initializing the Q-network, we get:

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

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

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

Exploration vs. Exploitation



Just after initializing the Q-network, we get:

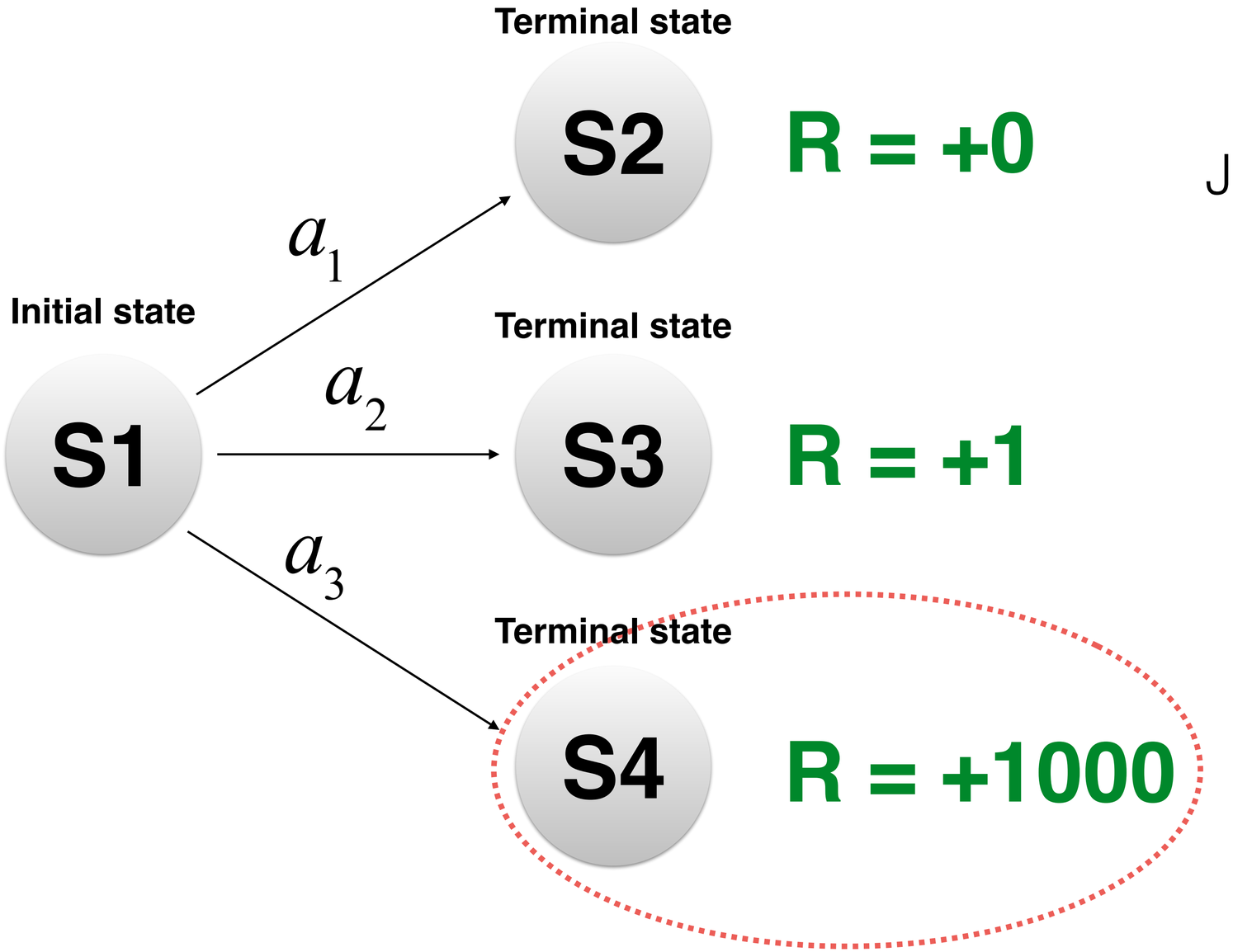
$$Q(S1, a_1) = 0.5 \quad 0$$

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

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

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

Exploration vs. Exploitation



Just after initializing the Q-network, we get:

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

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

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

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

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

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 reward 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*

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

Overall recap'

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*

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

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

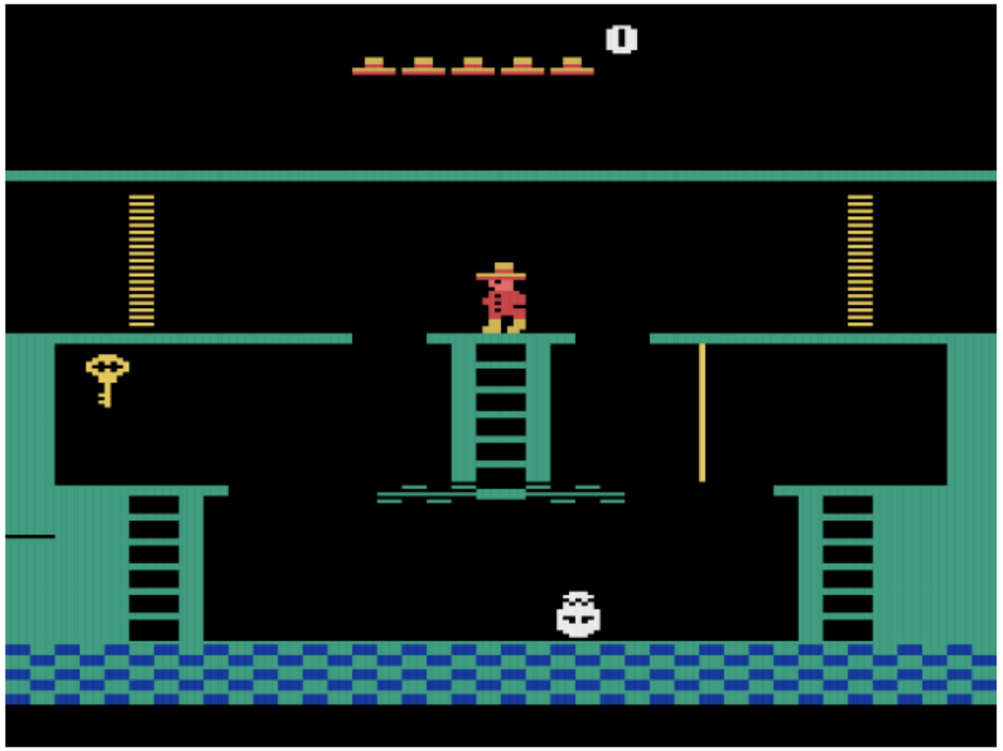
Results

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

[Credits: DeepMind, DQN Breakout - https://www.youtube.com/watch?v=TmPfTpjtdgg]

Difference between with and without human knowledge

Imitation learning

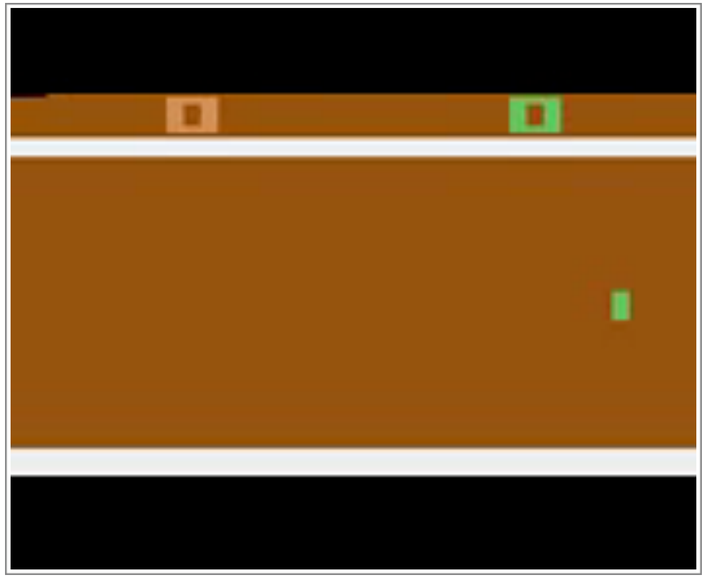


[Source: Bellemare et al. (2016): Unifying Count-Based Exploration and Intrinsic Motivation]

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

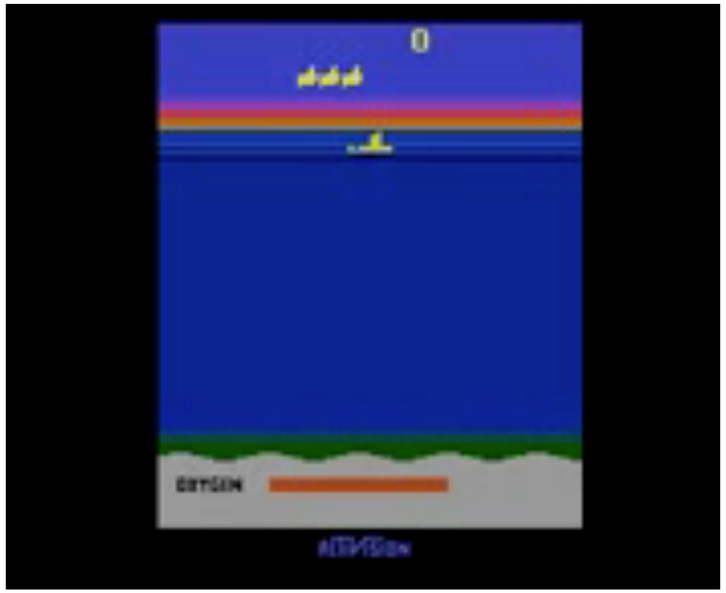
Other Atari games

Pong



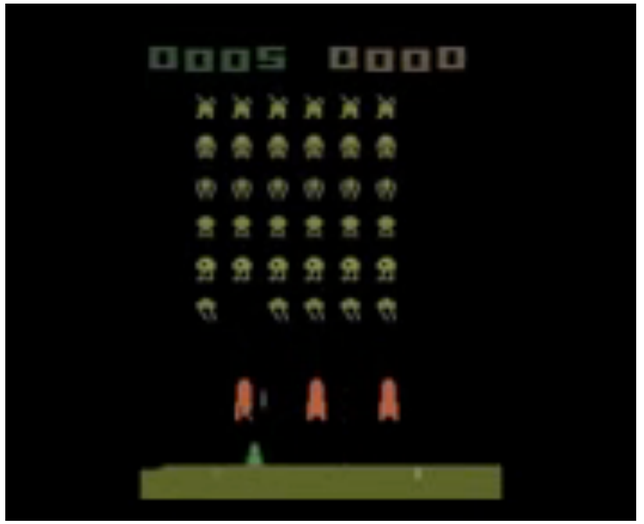
[Chia-Hsuan Lee, Atari Seaquest Double DQN Agent - https://www.youtube.com/watch?v=NirMkC5uvWU]

SeaQuest



[moooopan, Deep Q-Network Plays Atari 2600 Pong - https://www.youtube.com/watch?v=p88R2_3yWPA]

Space Invaders



[DeepMind: DQN SPACE INVADERS - https://www.youtube.com/watch?v=W2CAghUiofY&t=2s]

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

Today's outline

I. Motivation

II. Recycling is good: an introduction to RL

III. Deep Q-Networks

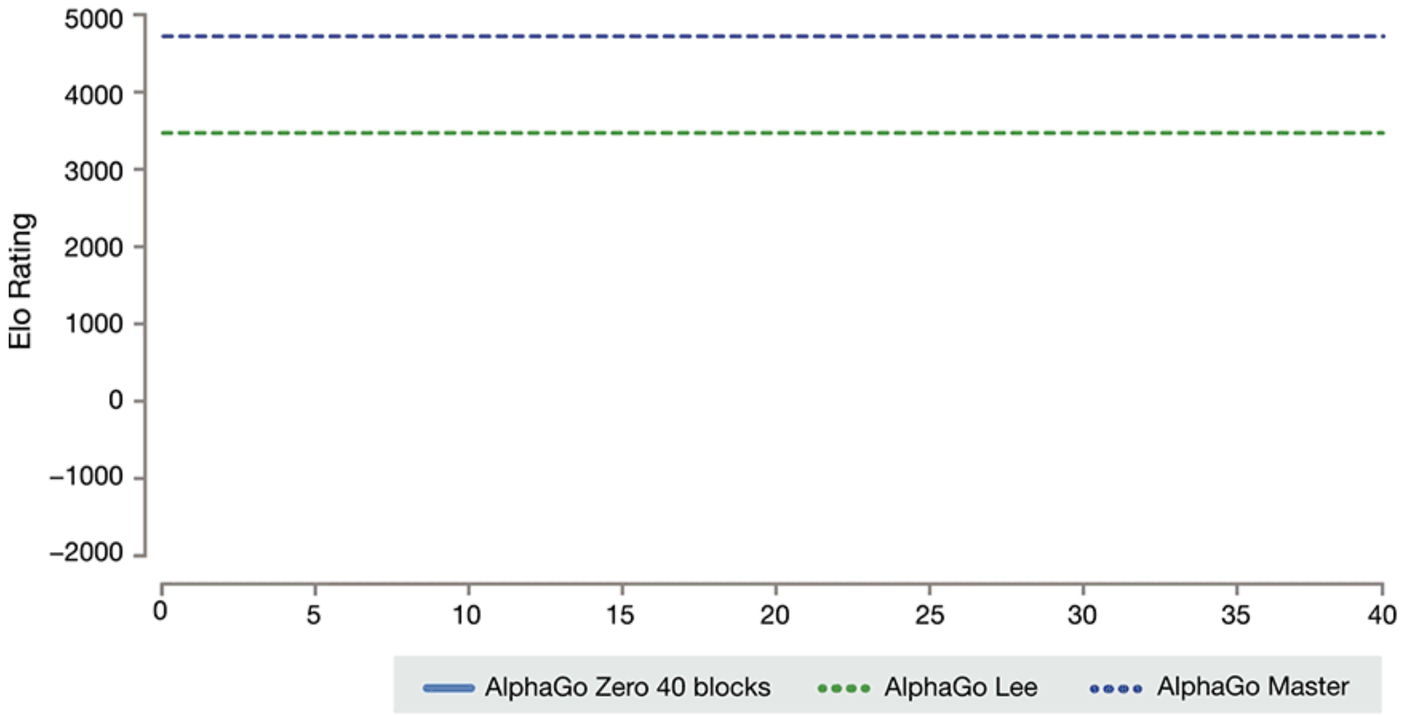
IV. Application of Deep Q-Network: Breakout (Atari)

V. Tips to train Deep Q-Network

VI. Advanced topics

VI - Advanced topics

Alpha Go

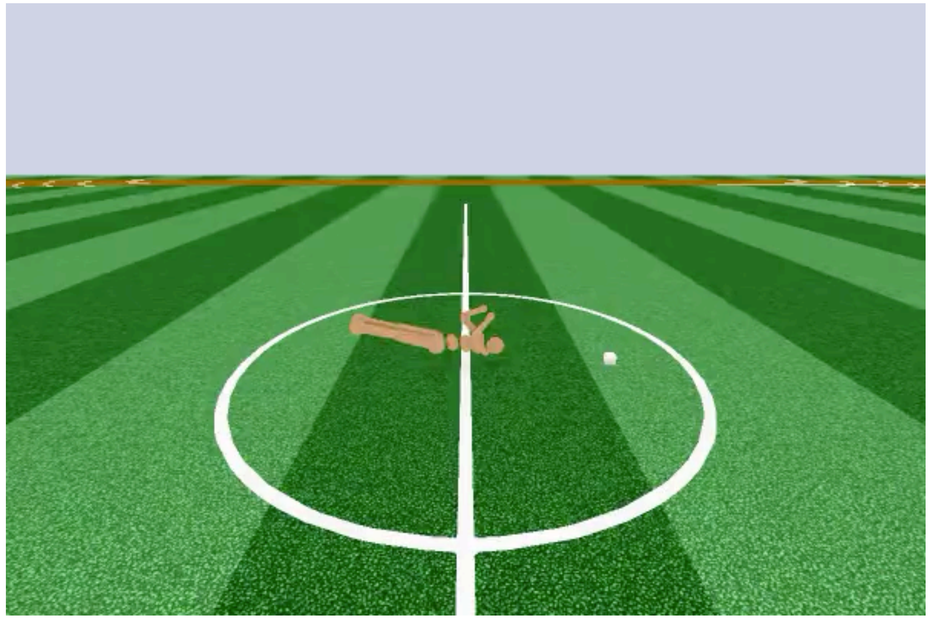


[DeepMind Blog]

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

Policy Gradient Methods

PPO



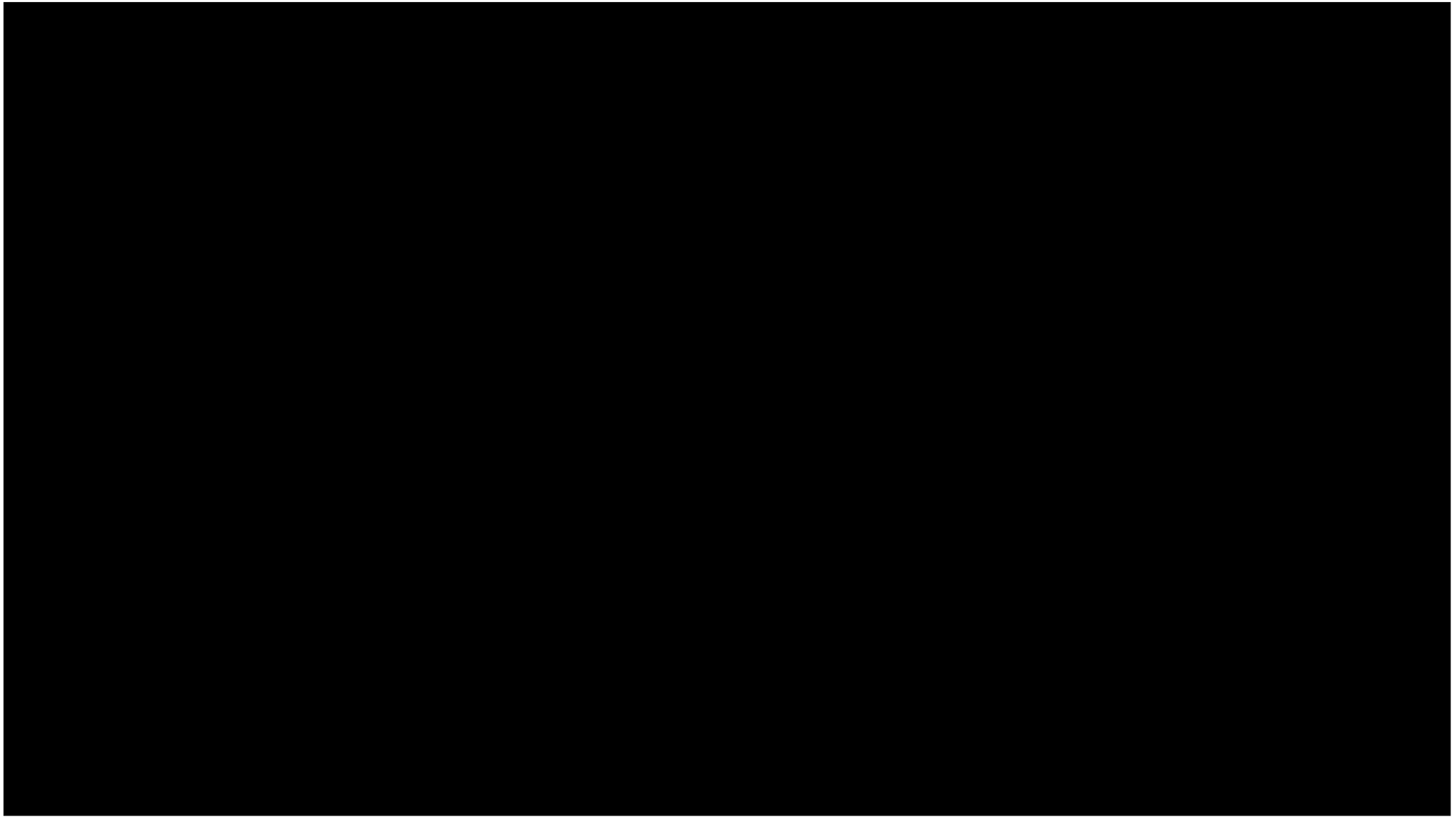
[Open AI Blog]

TRPO

[TRPO]

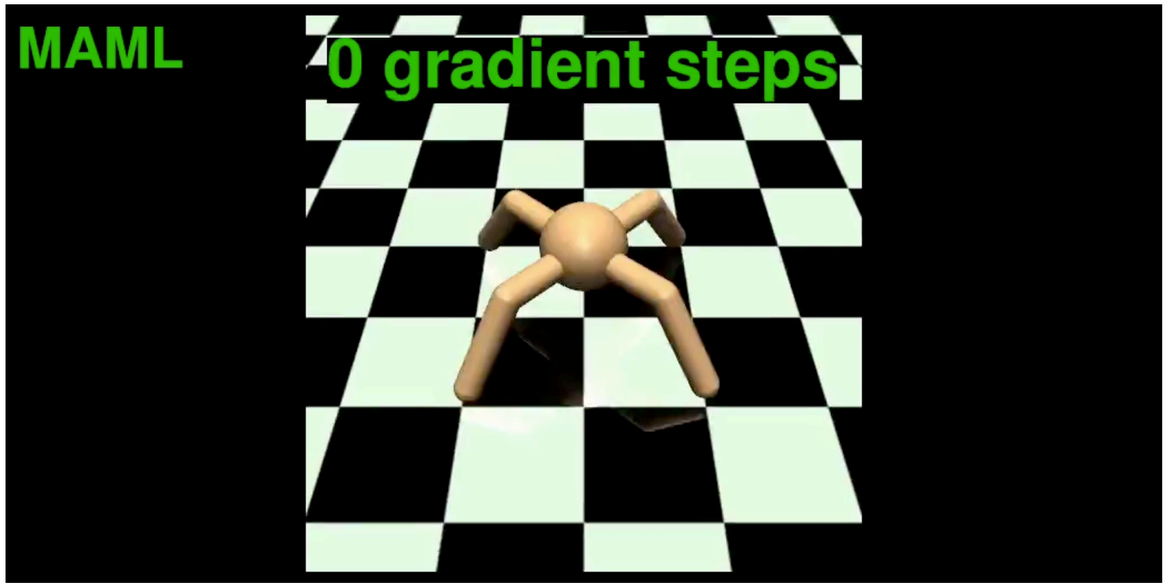
[Schulman et al. (2017): Trust Region Policy Optimization]

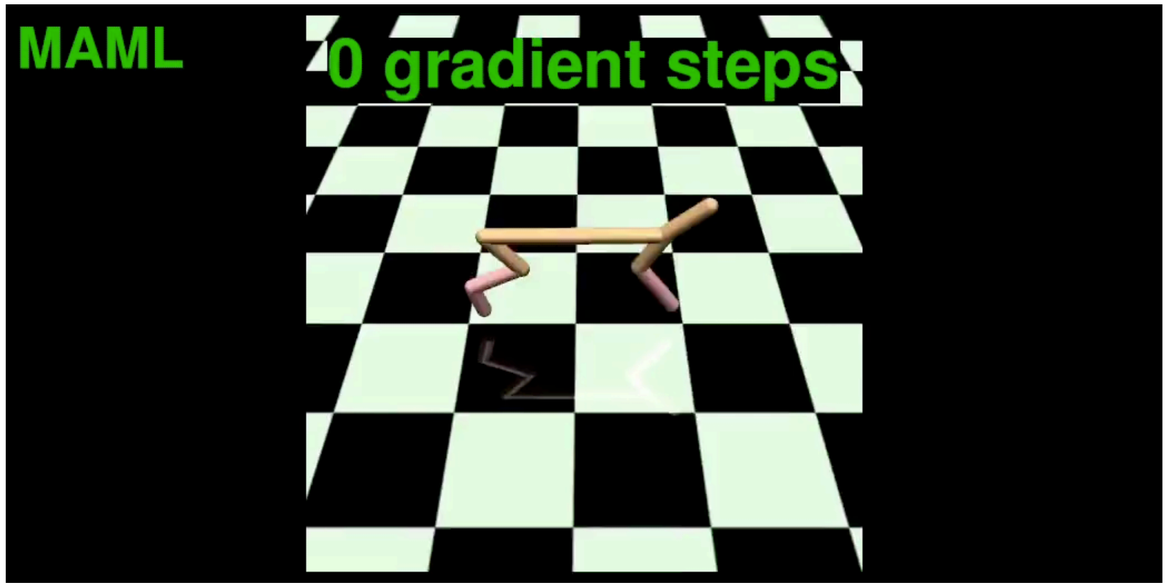
[Schulman et al. (2017): Proximal Policy Optimization]



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

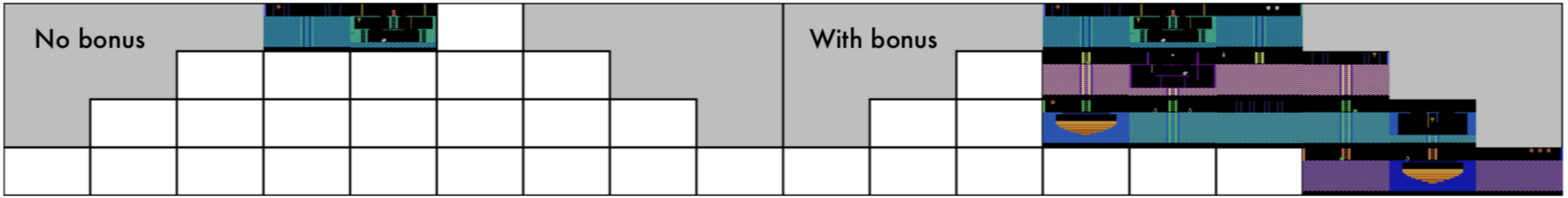
Meta learning





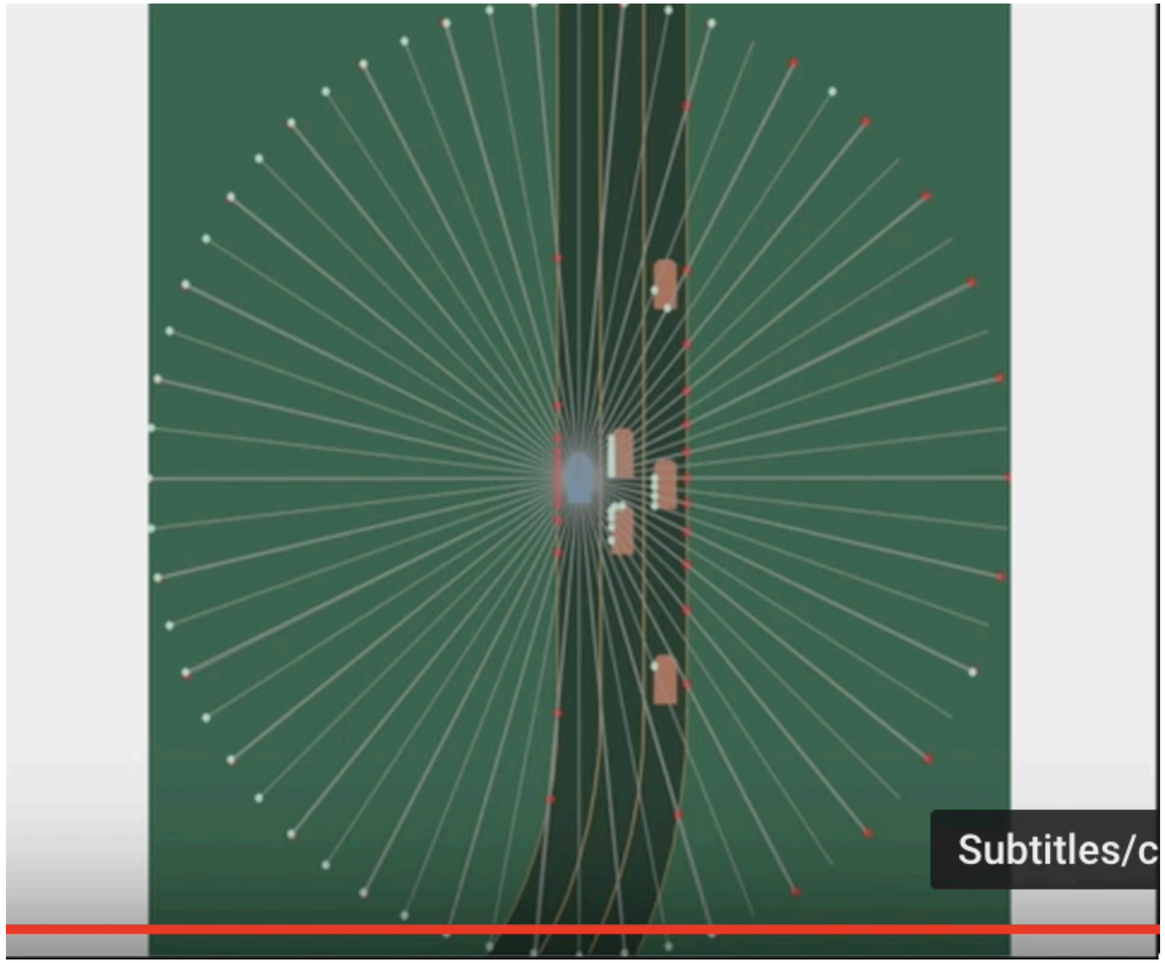
[Finn et al. (2017): Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks]

Exploration in Reinforcement Learning



[Bellemare et al. (2017):Unifying Count-Based Exploration and Intrinsic Motivation]

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri



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

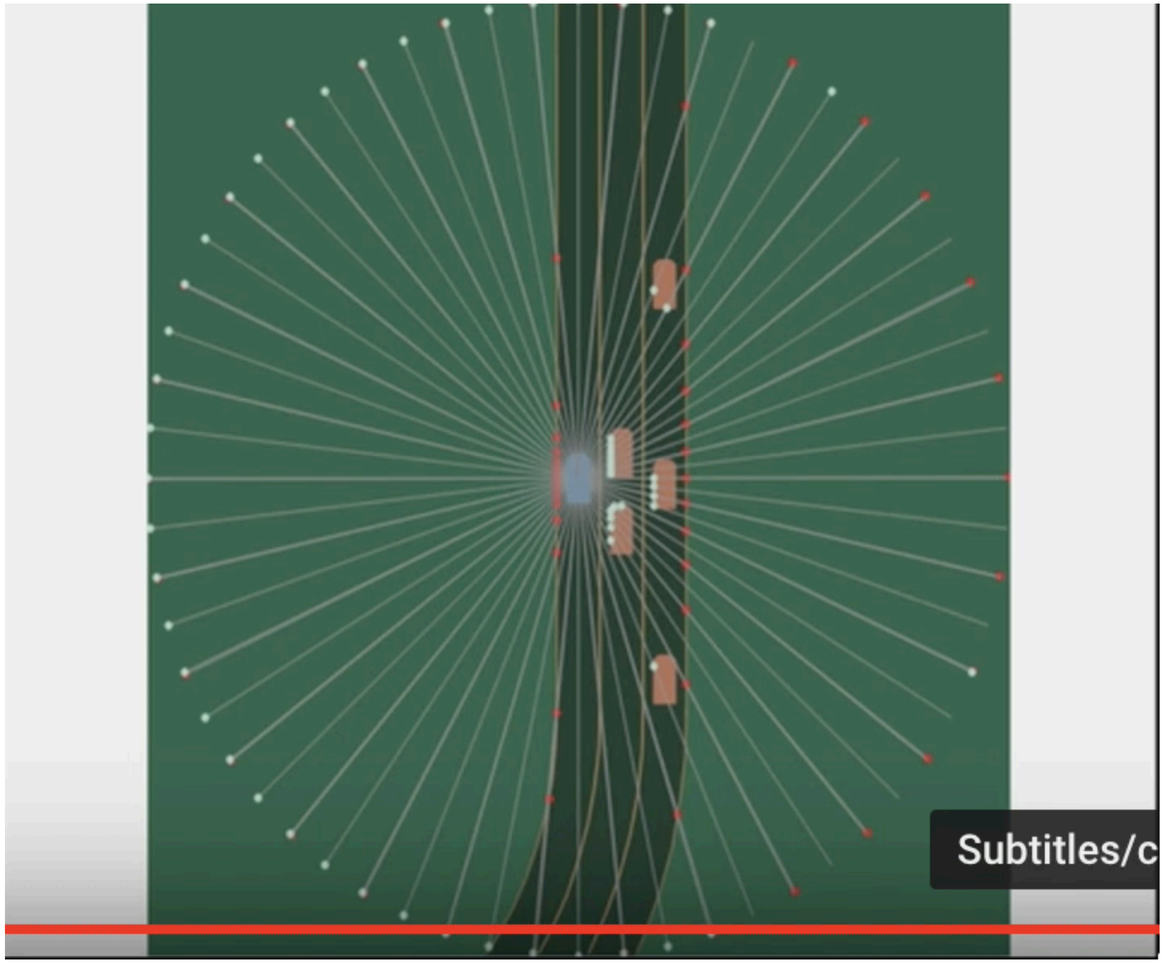
Announcements

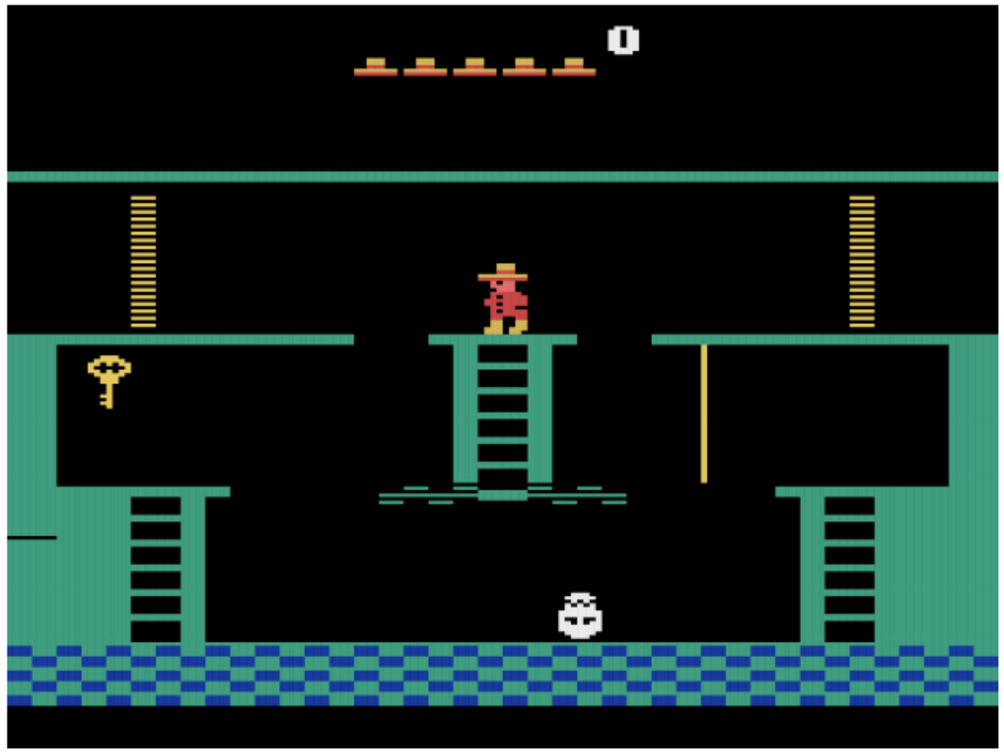
For Wednesday 12/05, 11am:

- **No assignment this week, but work on your projects and meet your mentors!**

This Friday:

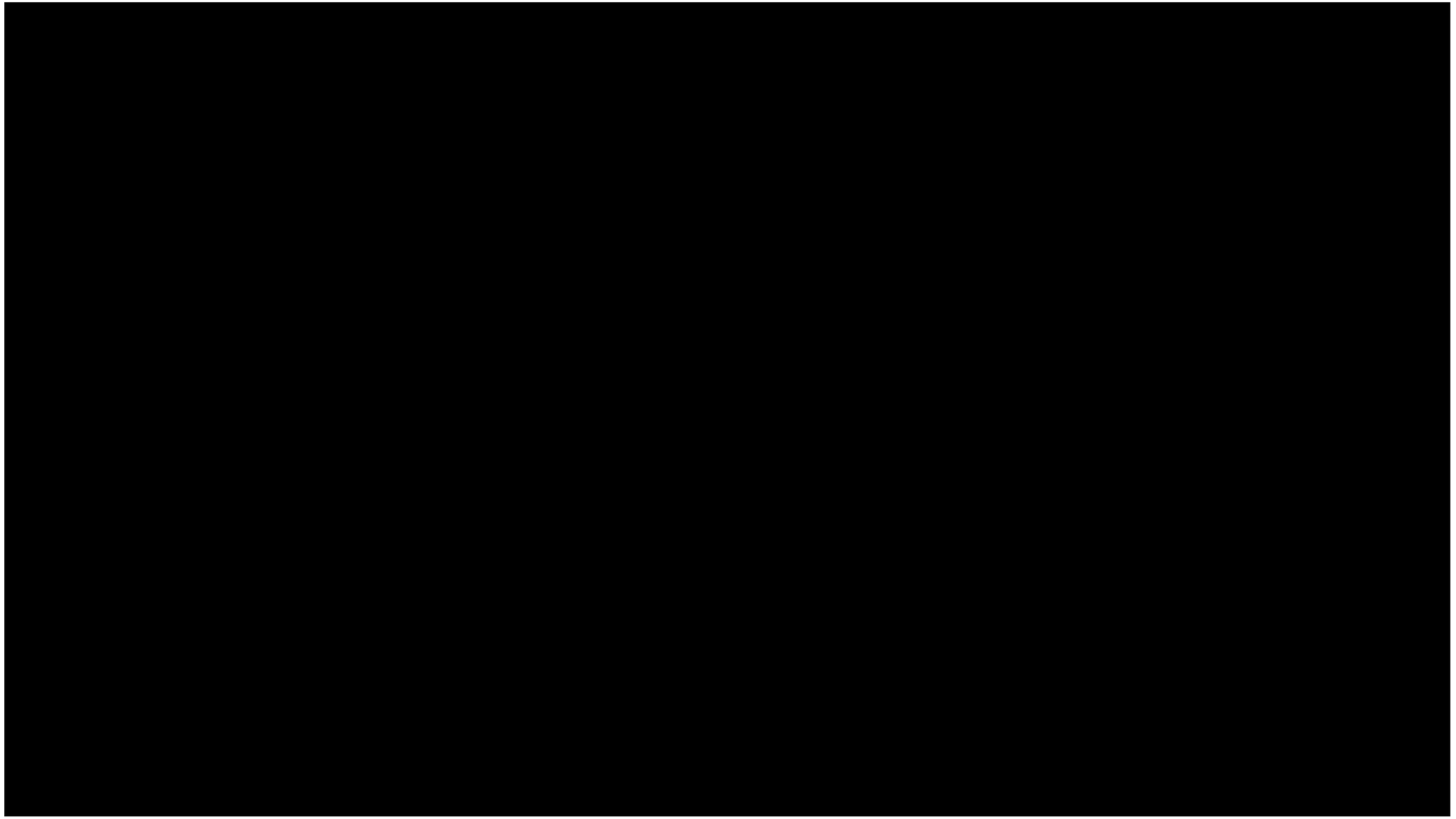
- **TA Sections: reading research papers.**





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

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



Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri