

Deep Reinforcement Learning

Jay Urbain, PhD

Credits:

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

<https://arxiv.org/abs/1312.5602>

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529.

<https://daiwk.github.io/assets/dqn.pdf>

Andrew Ng, Kian Katanforoosh, Stanford

Topics

- Motivation
- RL Review
- Deep Q-Learning
- Application of Deep Q-Learning: Breakout (Atari)
- Training Deep Q-Networks
- Advanced topics

Demis Hassabis - Deep Mind's CEO

- “In the pursuit of the AGI ([Artificial General Intelligence](#)), we need to widen the domains in which our agents excel. Creating a program that solves a single game is no longer a challenge and it stands true even for the relatively complex games with enormous search spaces like Chess ([Deep Blue](#)) or Go ([Alpha Go](#)). The real challenge would be to create an agent that can solve multiple tasks.”
- *“We have a prototype of this - the human brain. We can tie our shoelaces, we can ride cycles and we can do physics with the same architecture. So we know this is possible.”*

Motivation – Amazing Results

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 den Driessche, Thore Graepel, Demis Hassabis.

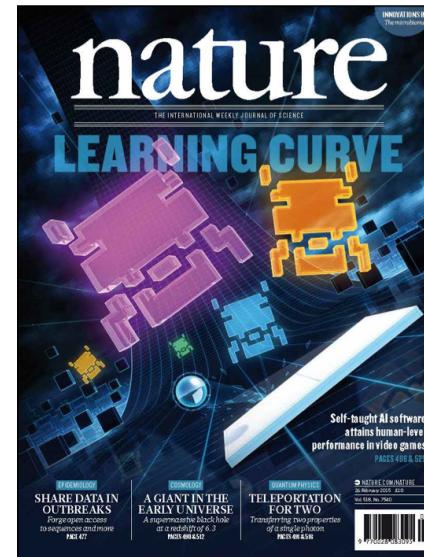
DeepMind, 5 New Street Square, London EC4A 3TW.

*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*.

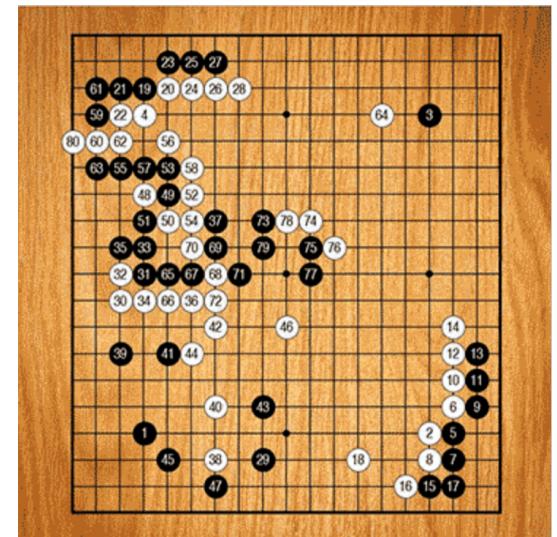
AlphaGo

[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**]



Motivation

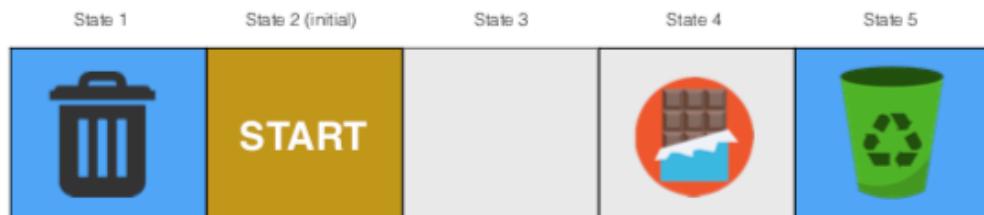
- *How would you solve Go with supervised learning?*
- **Issues:**
 - Ground truth for supervised learning is probably incorrectly defined.
 - Two many states in the game.
 - Unlikely to generalize.
- **Why RL?**
 - Delayed labels
Making sequences of decisions
- **What is RL?**
 - Automatically learn to make good sequences of decision



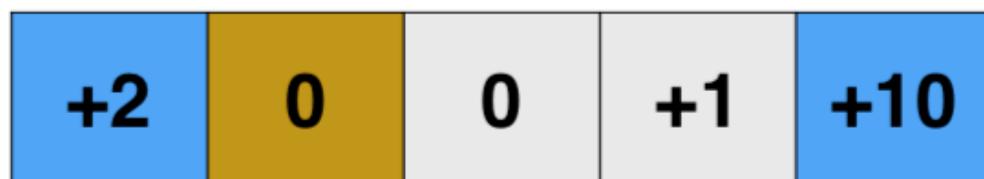
RL Q-Learning Review

Andrew Ng, Kian Katanforoosh, Stanford

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:

initial normal terminal

Agent's Possible actions: 

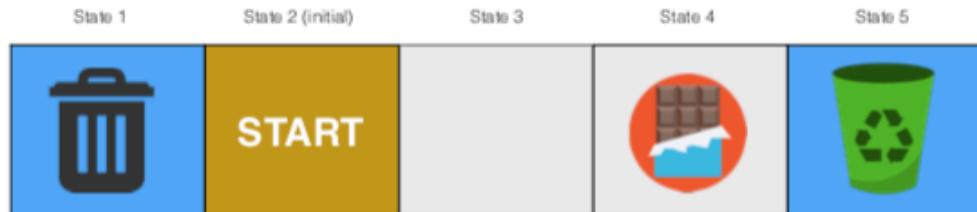
Additional rule: garbage collector coming in 3min, it takes 1min to move between states

How to define the long-term return?

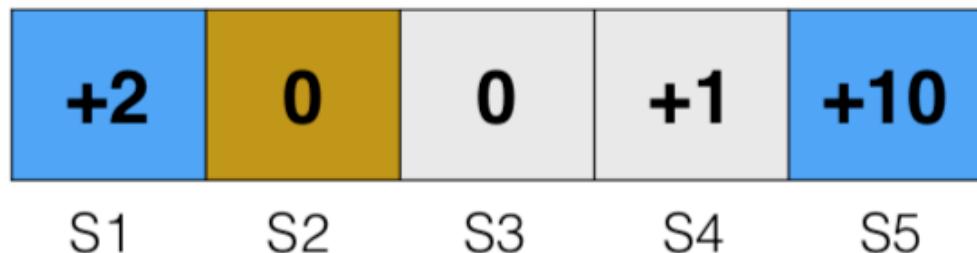
$$Discounted\ return \quad R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$$

RL

Problem statement



Define reward “ r ” in every state



Assuming $\gamma = 0.9$

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

What do we want to learn?

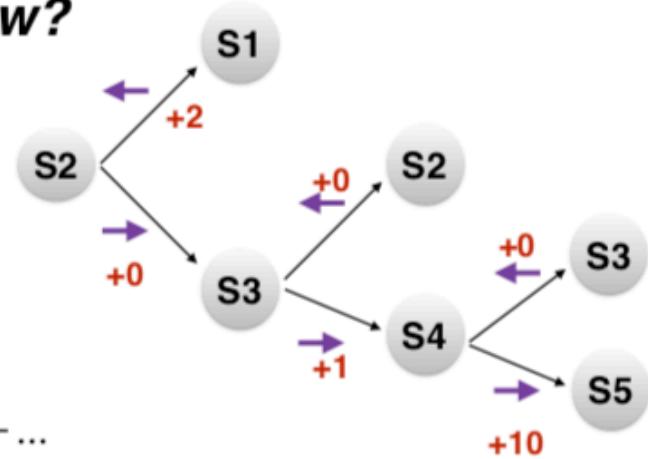
how good is it to take action 1 in state 2

Q-table

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

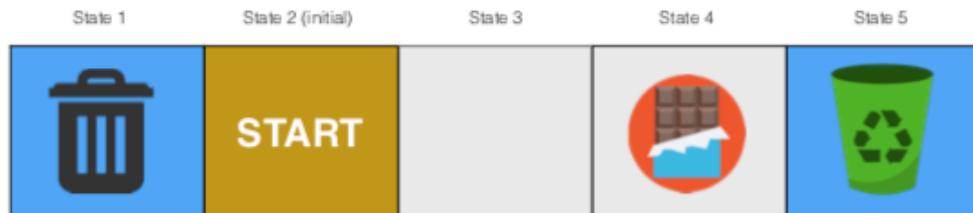
#actions #states

How?

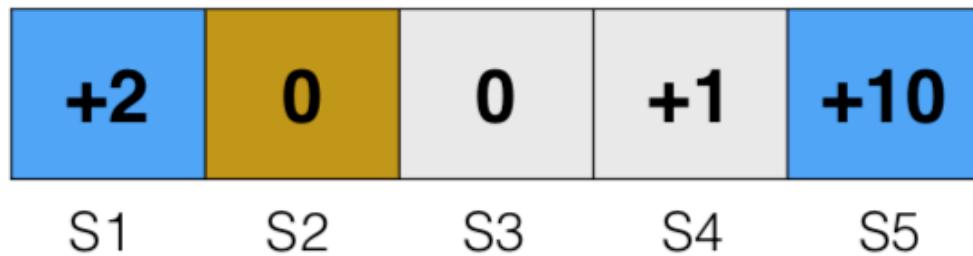


RL

Problem statement



Define reward “**r**” in every state



Assuming $\gamma = 0.9$

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

What do we want to learn?

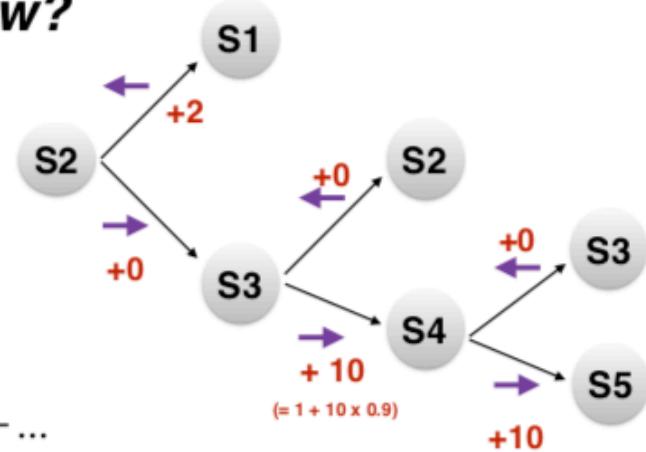
how good is it to take action 1 in state 2

Q-table

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

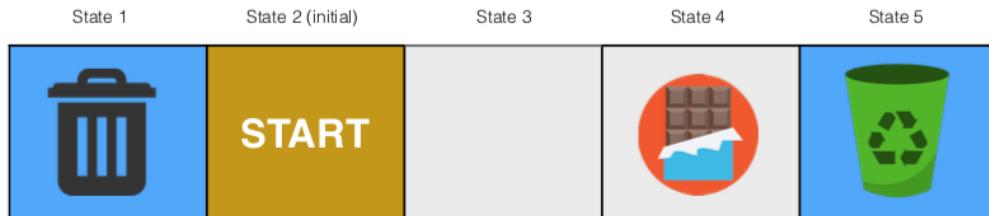
#actions #states

How?

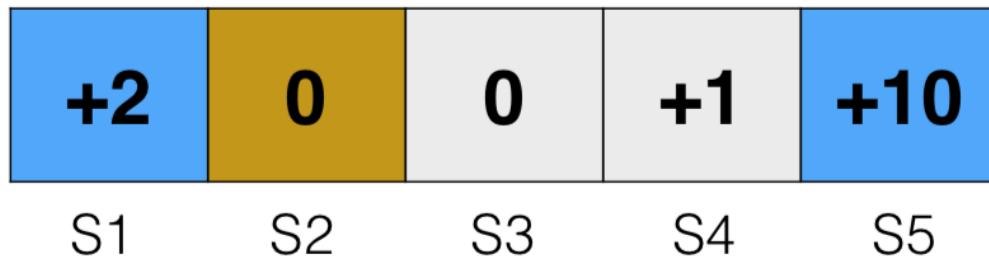


RL

Problem statement



Define reward “ r ” in every state



Assuming $\gamma = 0.9$

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

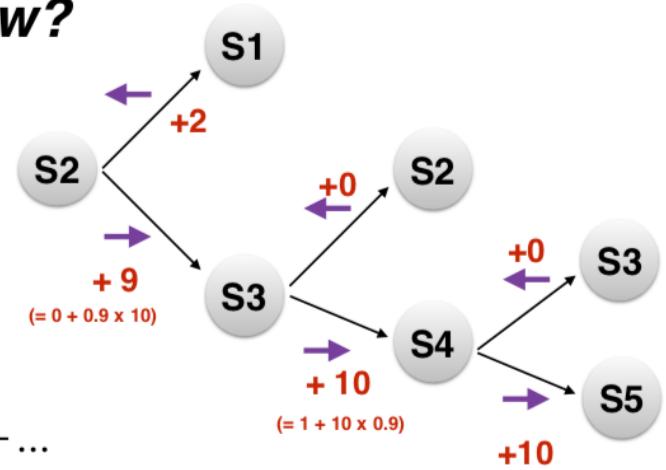
What do we want to learn?

how good is it to take action 1 in state 2

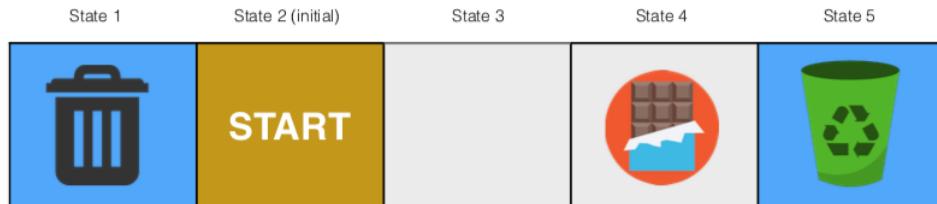
Q-table

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

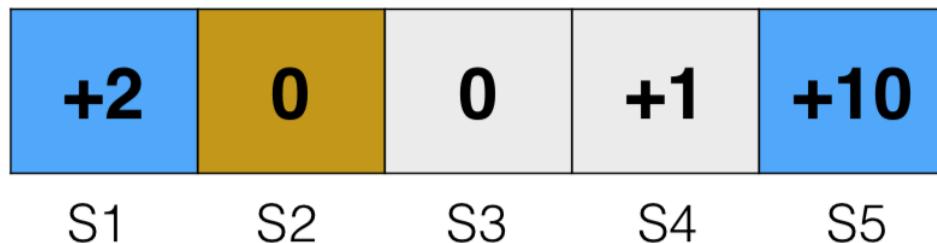
How?



Problem statement



Define reward “ r ” in every state



Assuming $\gamma = 0.9$

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

What do we want to learn?

how good is it to take action 1 in state 2

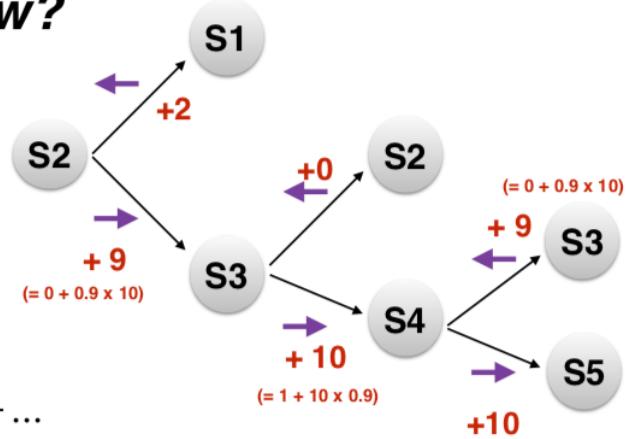
Q-table

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

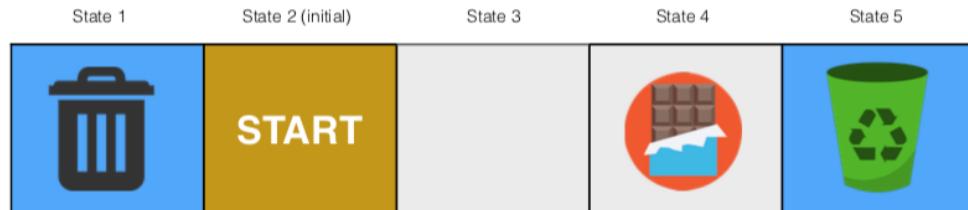
#actions

#states

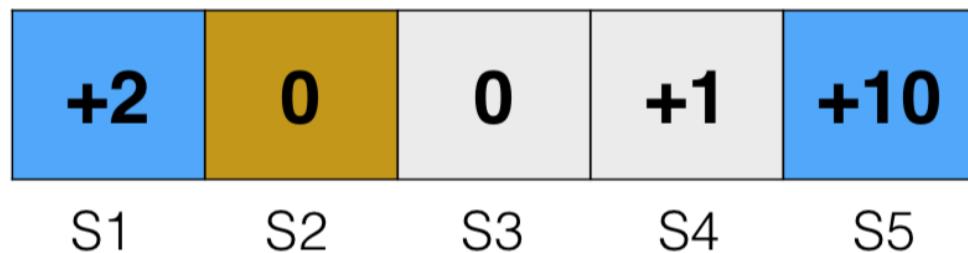
How?



Problem statement



Define reward “ r ” in every state



Assuming $\gamma = 0.9$

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

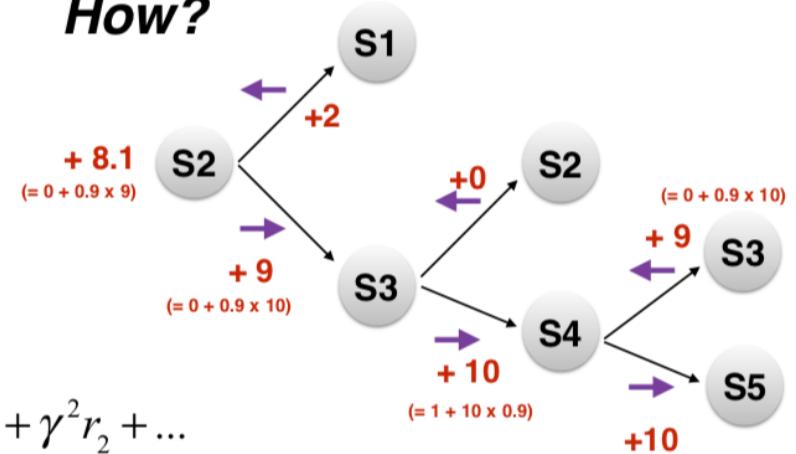
What do we want to learn?

how good is it to take action 1 in state 2

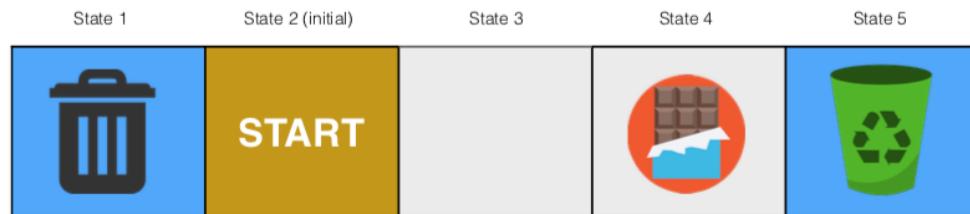
Q-table

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

How?



Problem statement



Define reward “ r ” in every state

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

Assuming $\gamma = 0.9$

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

What do we want to learn?

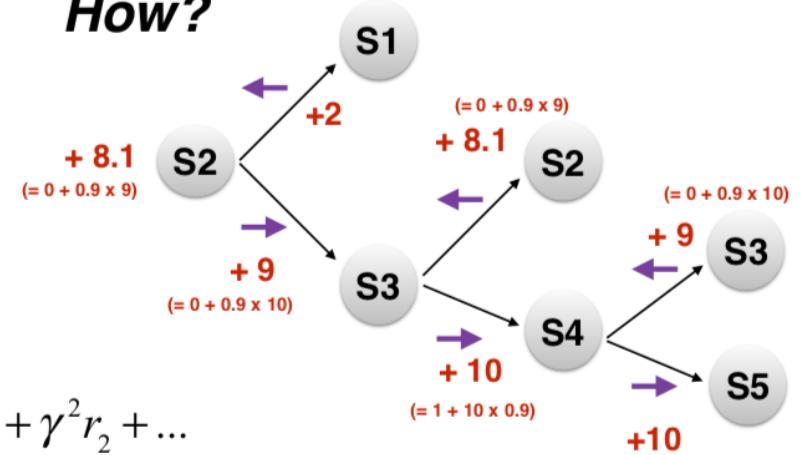
how good is it to take action 1 in state 2

Q-table

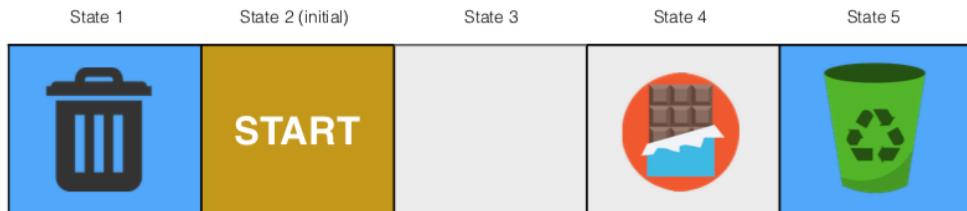
$$Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix}$$

#actions
↑
#states

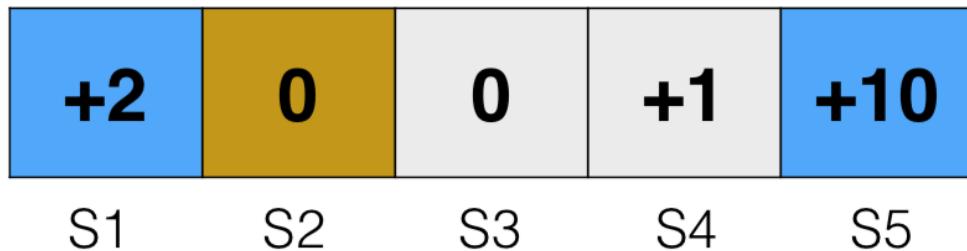
How?



Problem statement



Define reward “ r ” in every state



Assuming $\gamma = 0.9$

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

What do we want to learn?

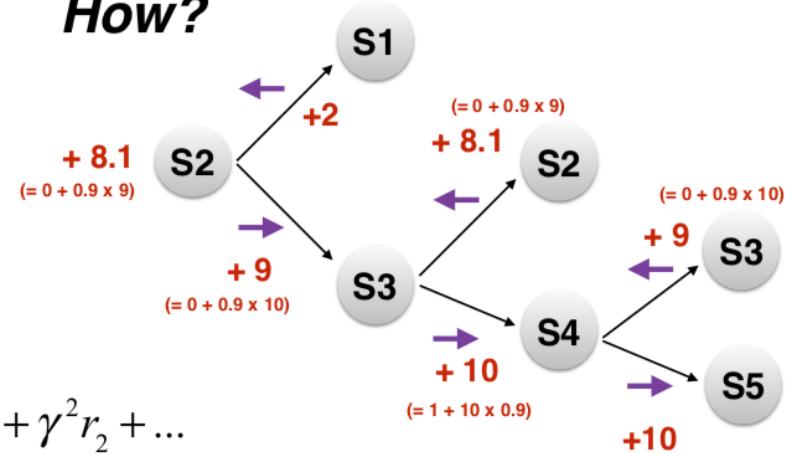
how good is it to take action 1 in state 2

Q-table

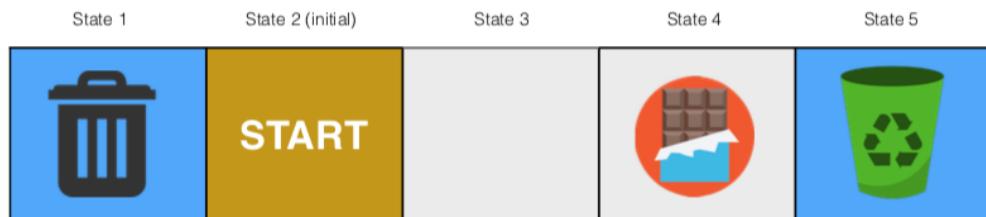
#actions	
0	0
2	9
8.1	10
9	10
0	0

#states

How?



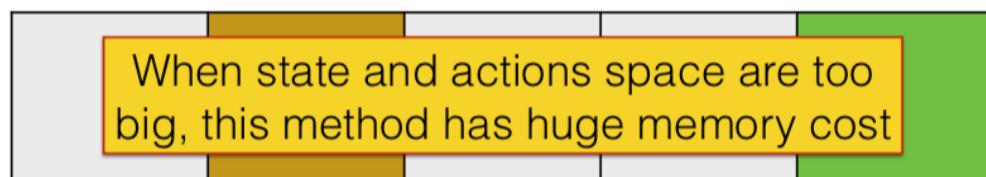
Problem statement



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

$$Q = \begin{pmatrix} & \xleftarrow{\text{\#actions}} \\ \xdownarrow{\text{\#states}} & \end{pmatrix}$$
$$\begin{matrix} 0 & 0 \\ 2 & 9 \\ 8.1 & 10 \\ 9 & 10 \\ 0 & 0 \end{matrix}$$

*Bellman equation
(optimality equation)*

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

Policy $\pi(s) = \arg \max_a (Q^*(s, a))$

Summary so far

- *Vocabulary: environment, agent, state, action, reward, total return,*
- *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 P*

Problems:

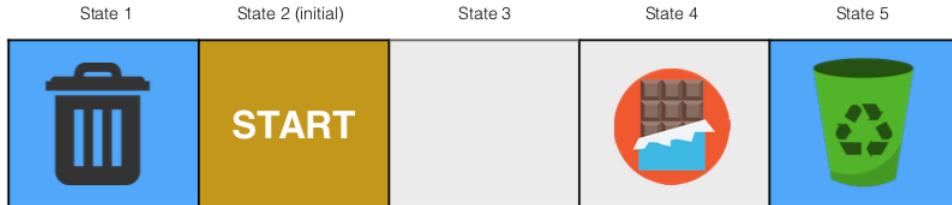
- Complex problems have intractable number of states
- Tabular data does not generalize to unseen examples

Deep Q-Learning

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

Problem statement

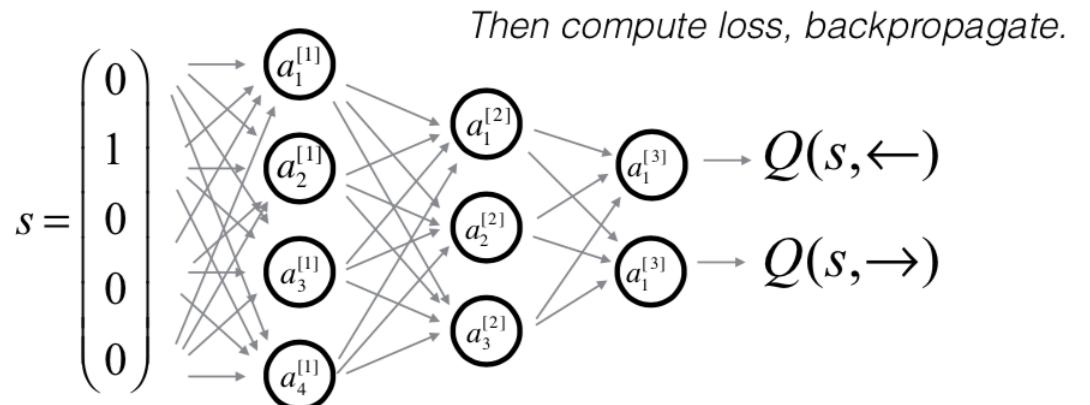
Neural Network



Q-table

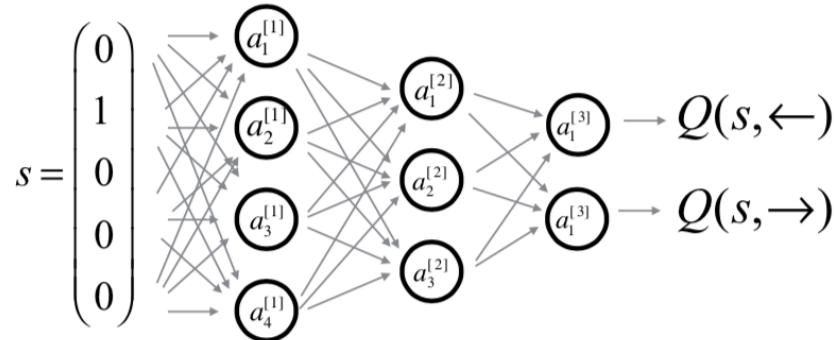
$$Q = \begin{pmatrix} 0 & 0 \\ 2 & 9 \\ 8.1 & 10 \\ 9 & 10 \\ 0 & 0 \end{pmatrix}$$

#actions #states



How to compute the loss?

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



Target value

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

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

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}

Loss function

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

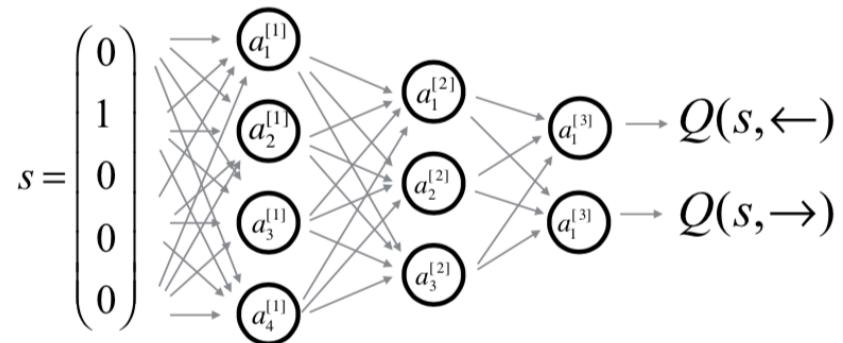
Case: $Q(s, \leftarrow) < Q(s, \rightarrow)$

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

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}



Target value

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

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

Loss function (regression)

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

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

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

Backpropagation

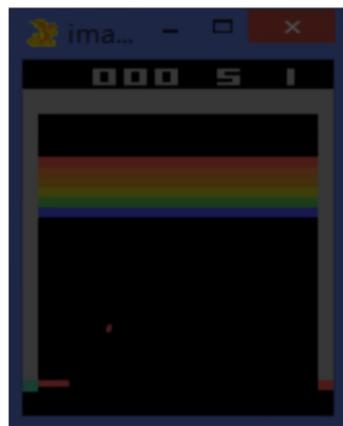
Compute $\frac{\partial L}{\partial W}$ and update W using stochastic gradient descent

DQN Implementation:

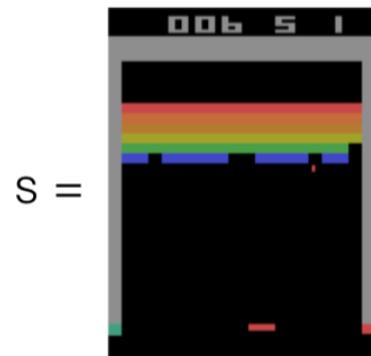
- Initialize your Q-network parameters
- Loop over episodes:
 - Start from initial state s
 - Loop over time-steps:
 - Forward propagate s in the Q-network
 - Execute action a (that has the maximum $Q(s,a)$ output of Q-network)
 - Observe reward r and next state s'
 - Compute targets y by forward propagating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

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

Demo



input of Q-network



Output of Q-network

Q-values

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

Will this work?

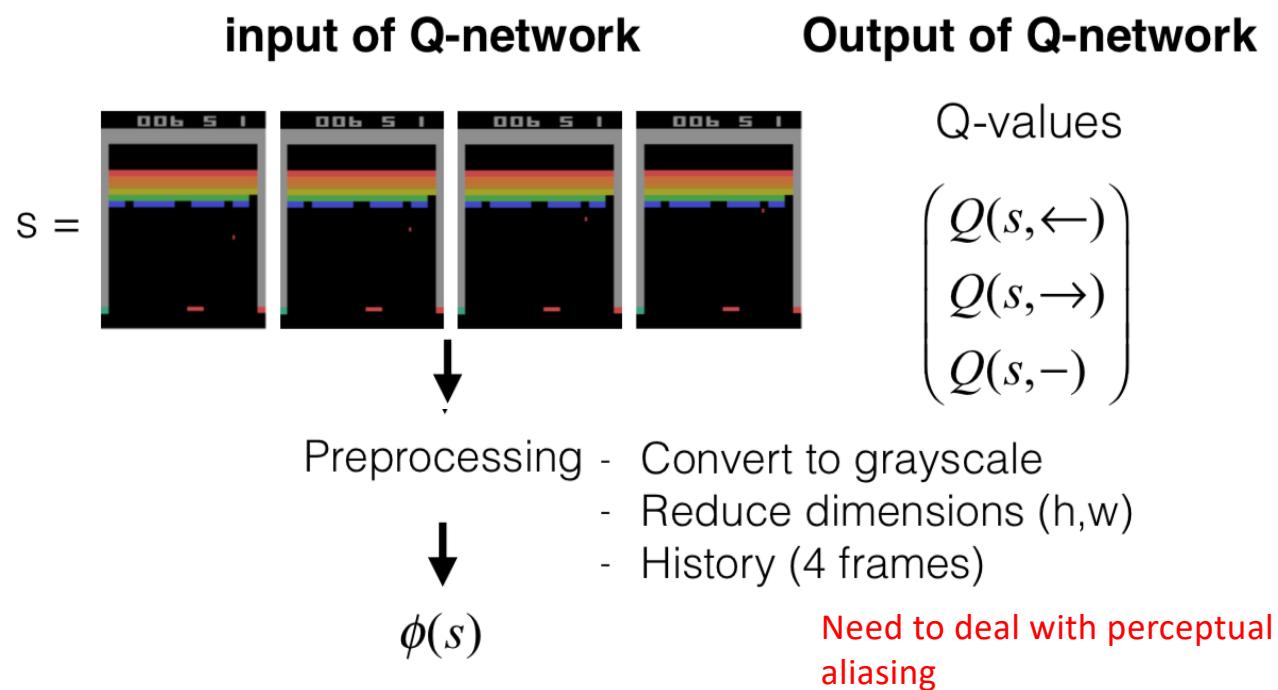
[Two minute papers: Google DeepMind's Deep Q-learning playing Atari Breakout

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

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

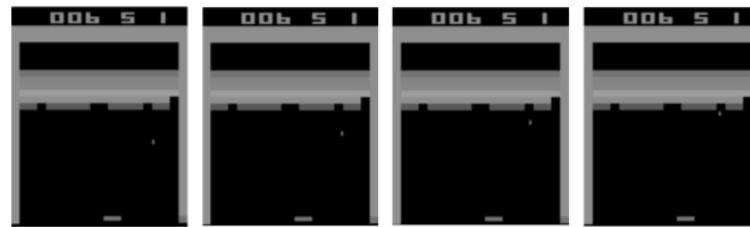
Goal: play breakout, i.e. destroy all the bricks.

Demo

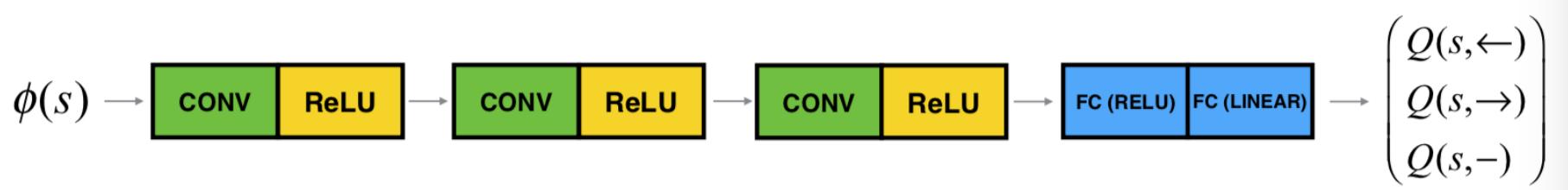


input of Q-network

$$\phi(s) =$$



Deep Q-network architecture?



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

$$\phi(s)$$

- Execute action a (that has the maximum $Q(\cancel{s}, a)$ output of Q-network)

- Observe reward r and next state s'

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

Some training challenges:

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

- Compute targets y by forward propagating state s' in the Q-network, then compute loss.

$$\phi(s')$$

- Update parameters with gradient descent

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 in the Q-network $\phi(s)$

- Execute action a (that has the maximum $Q(s, a)$ output of Q-network)

- Observe reward r and next state s'

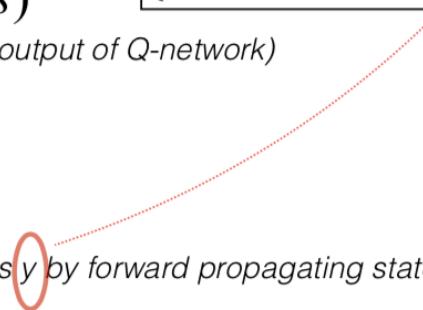
- Use s' to create $\phi(s')$

- Check if s' is a terminal state. Compute targets y by forward propagating state 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)

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


$\phi(s')$



Experience replay

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

1 experience (leads to one iteration of gradient descent)

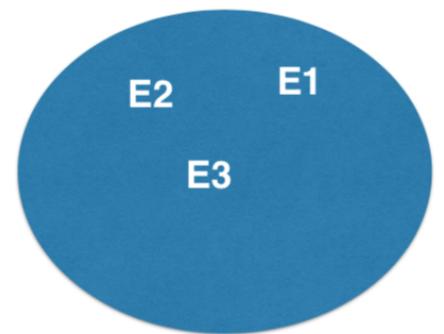
Experience Replay

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

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.

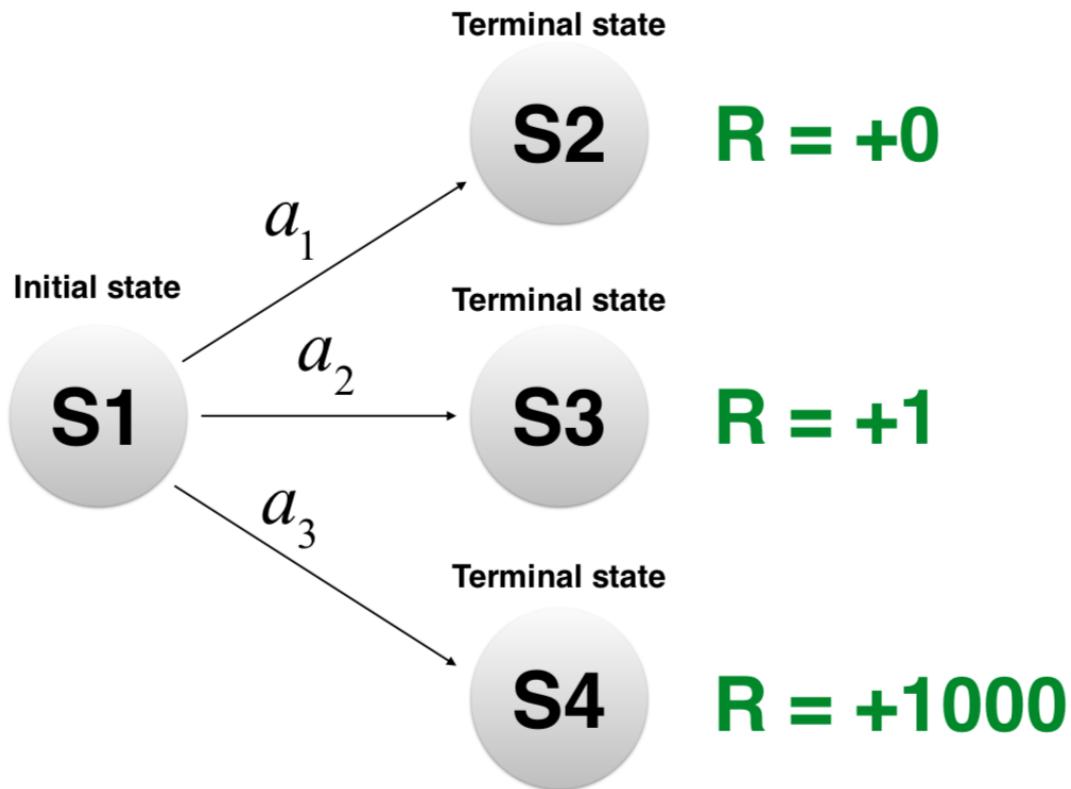
- Update parameters with gradient descent

Update using
sampled
transitions

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!

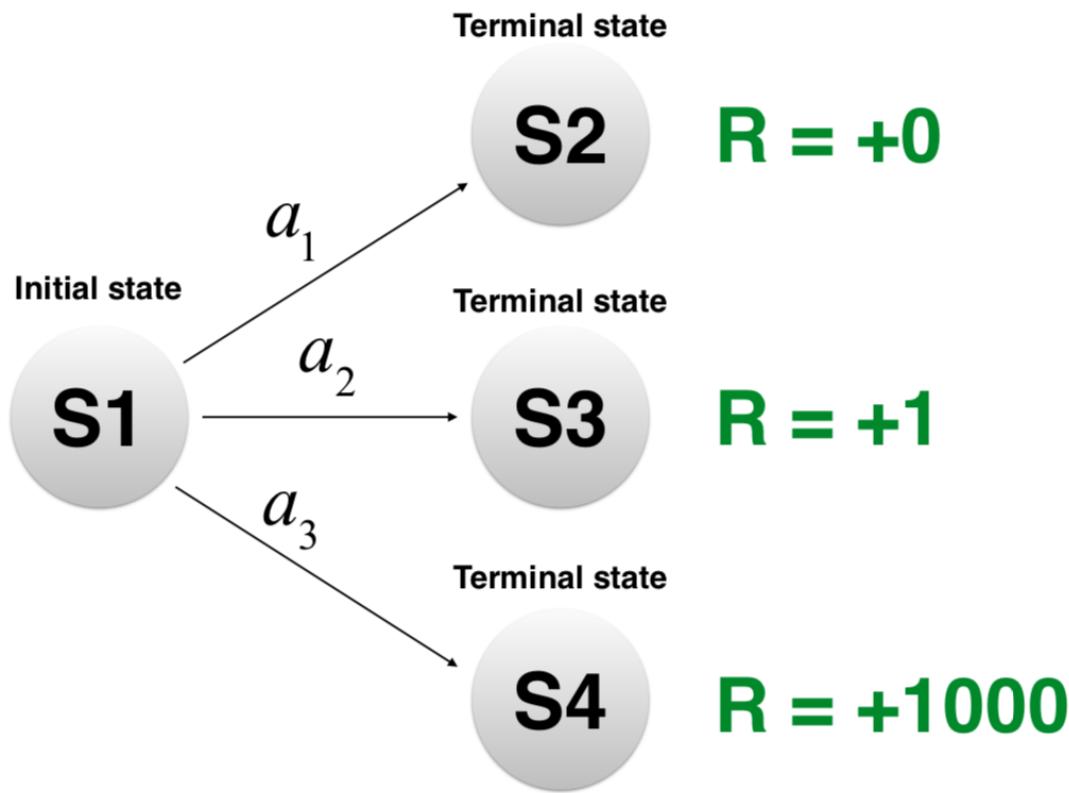


Just after initializing the Q-network, we get:

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

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

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

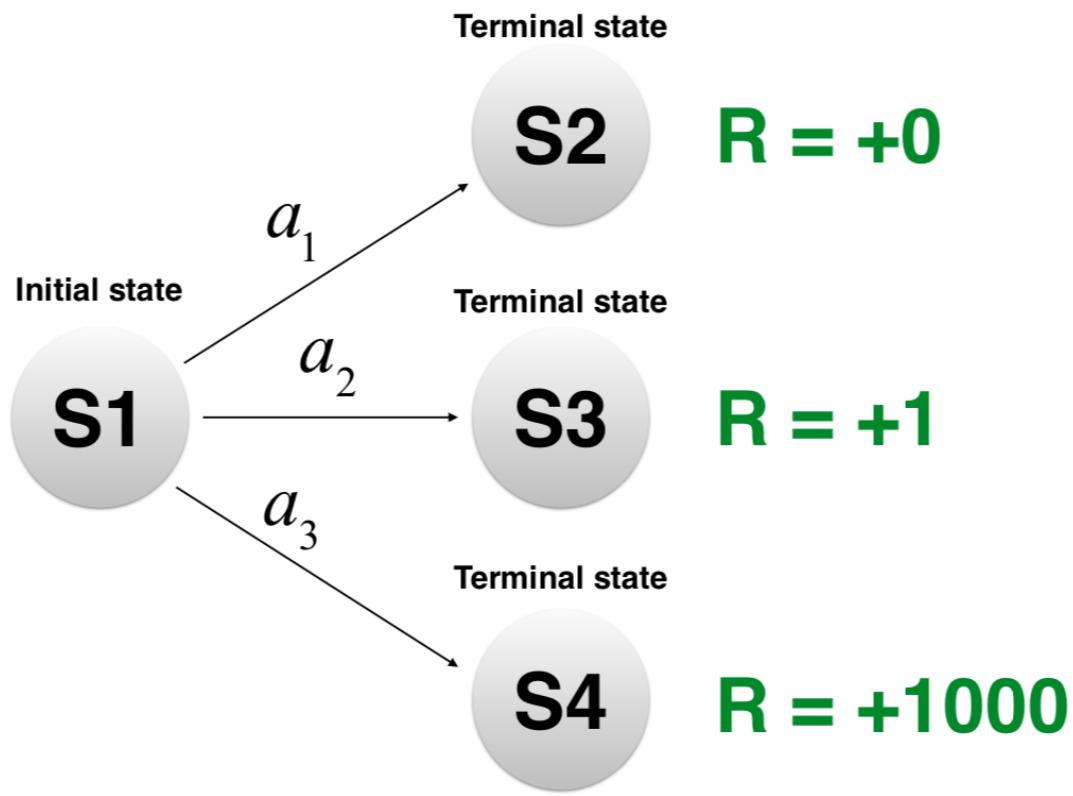


Just after initializing the Q-network, we get:

$$Q(S1, a_1) = 0 \cancel{.5} \quad 0$$

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

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

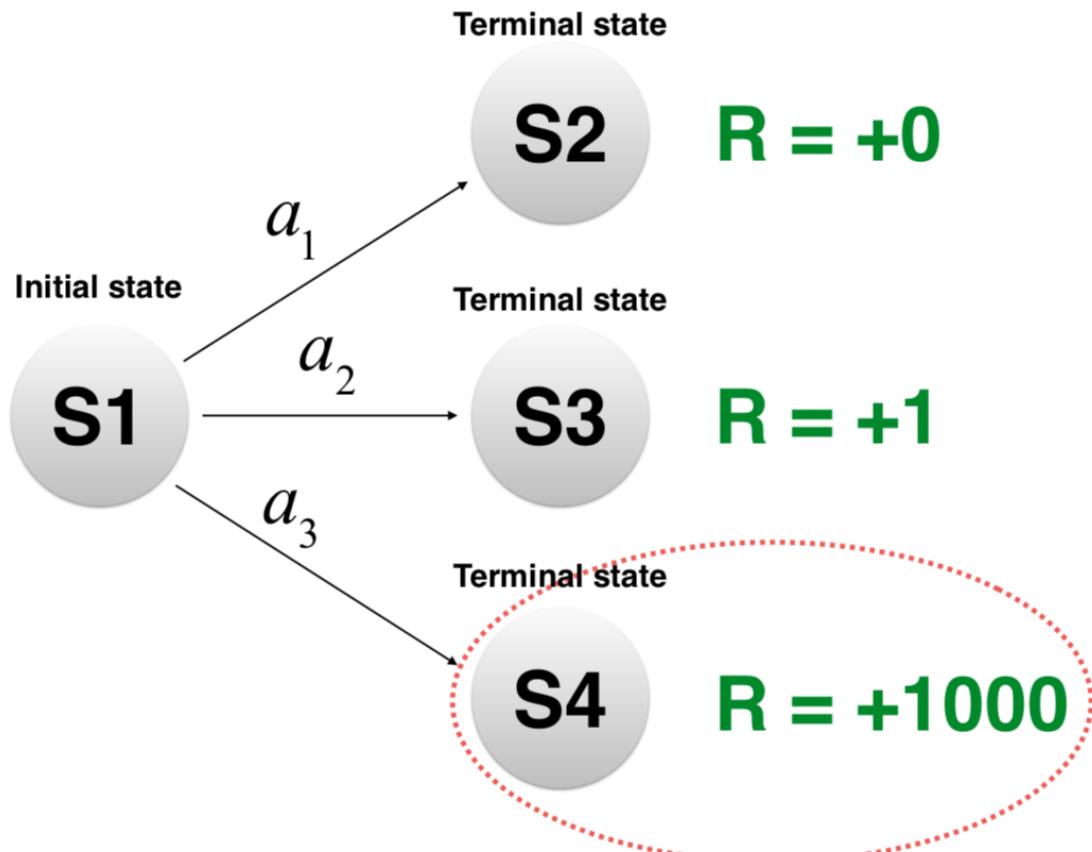


Just after initializing the Q-network, we get:

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

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

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



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
 $Q(S1, a_3) < Q(S1, a_2)$

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

DQN Implementation:

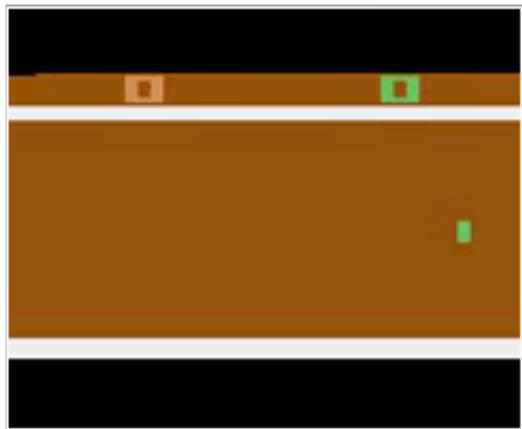
- Initialize your Q-network parameters
- **Initialize replay memory D**
- Loop over episodes:
 - Start from initial state $\phi(s)$
 - **Create a boolean to detect terminal states: terminal = False**
 - Loop over time-steps:
 - **With probability epsilon, take random action a.**
 - **Otherwise:**
 - 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**

Demo



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>]