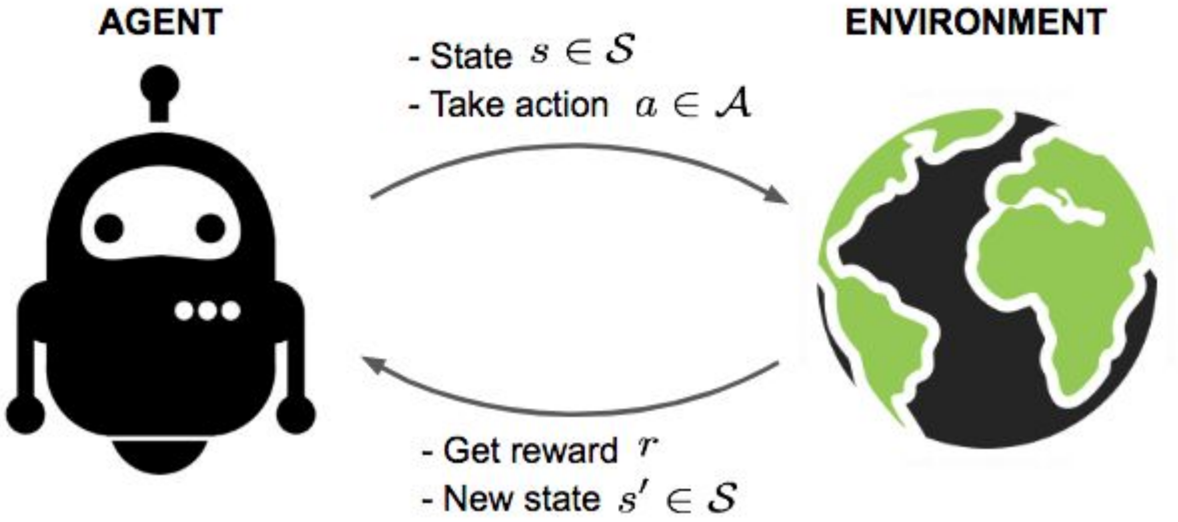
Deep Reinforcement Learning

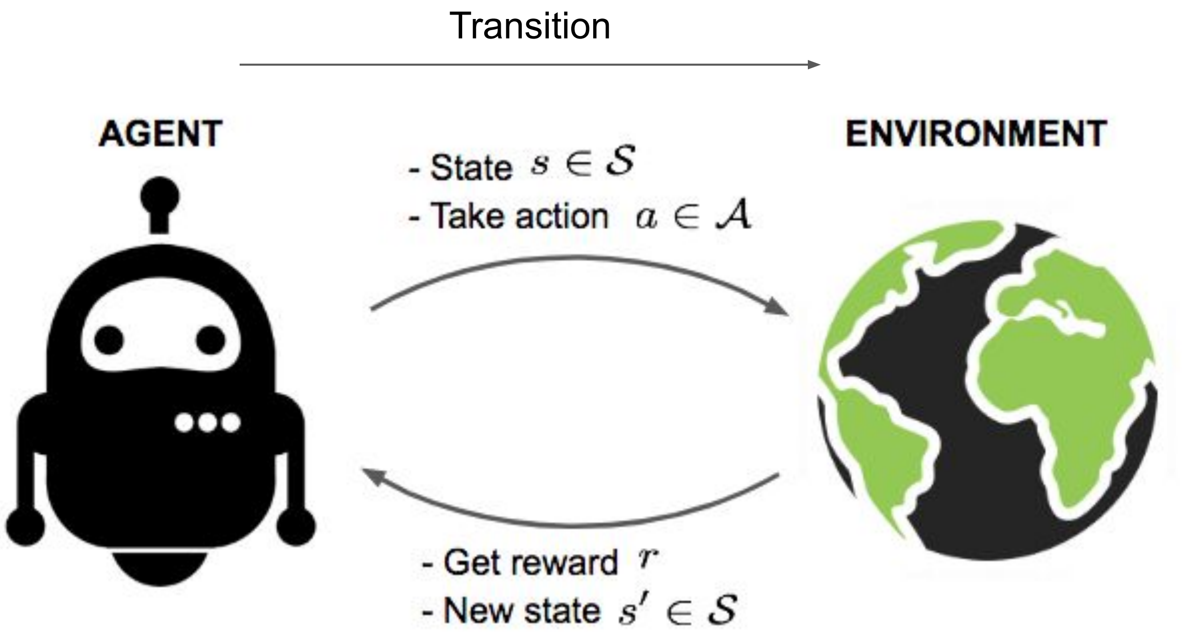
Colin Brust, Megan Finley, Daniel Olson

What is Reinforcement Learning?



https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

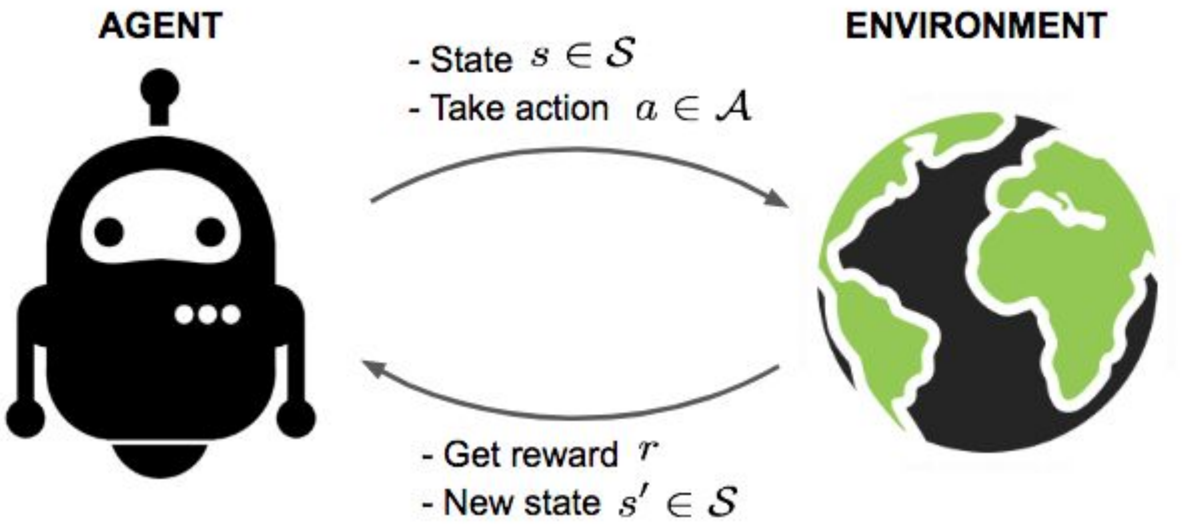
What is Reinforcement Learning?



https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

What is Reinforcement Learning?

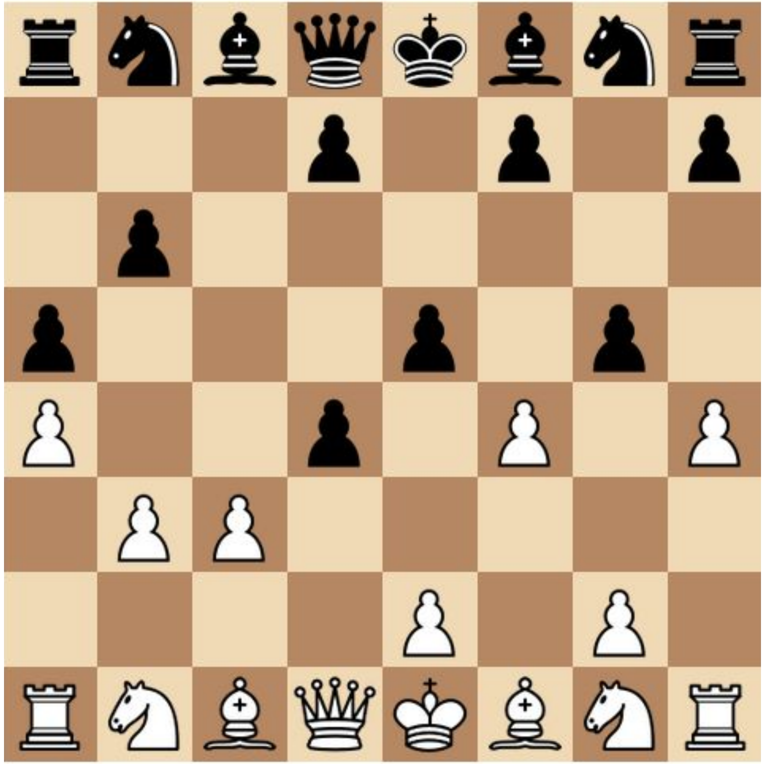
Transition



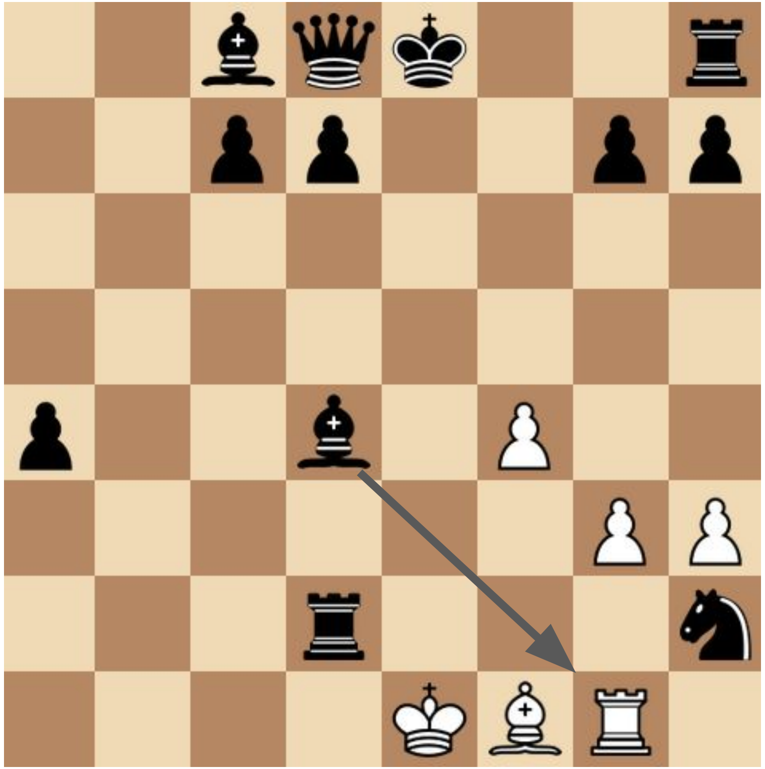
Reward

https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

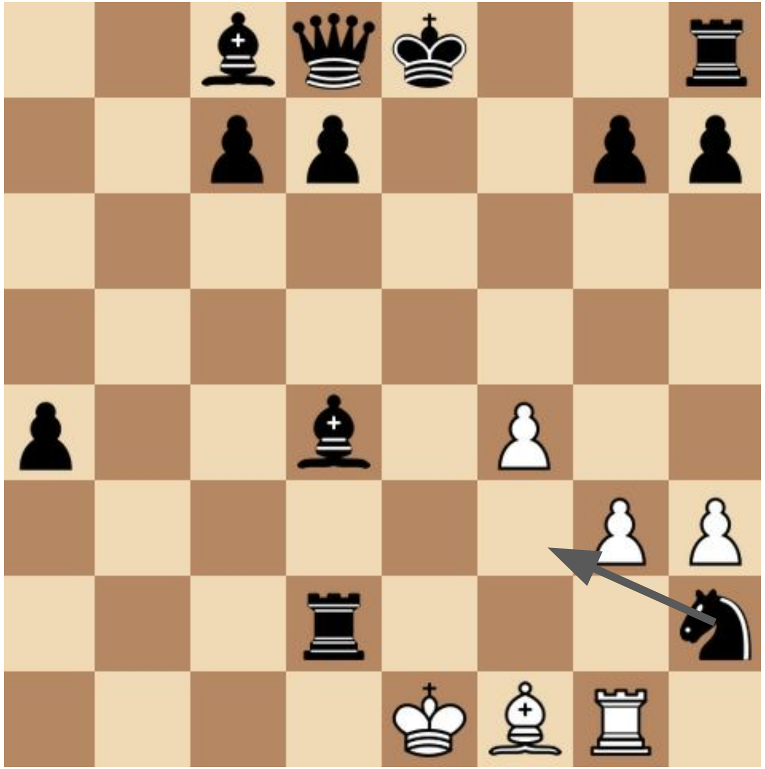
Goal: Maximize future reward



Exploration vs Exploitation Dilemma

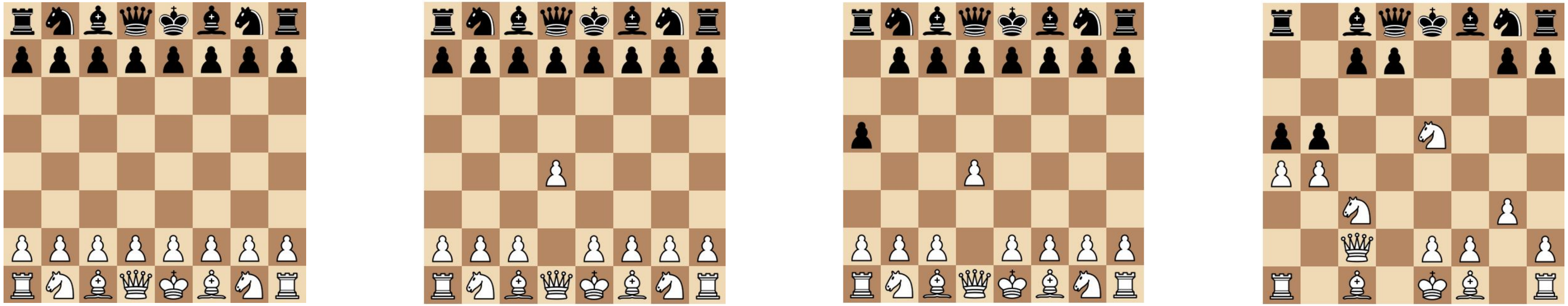


vs.



Reinforcement Learning Terminology

Term	Description
$s_t \in S$	The environment state at time t and the set of all states an agent can observe.

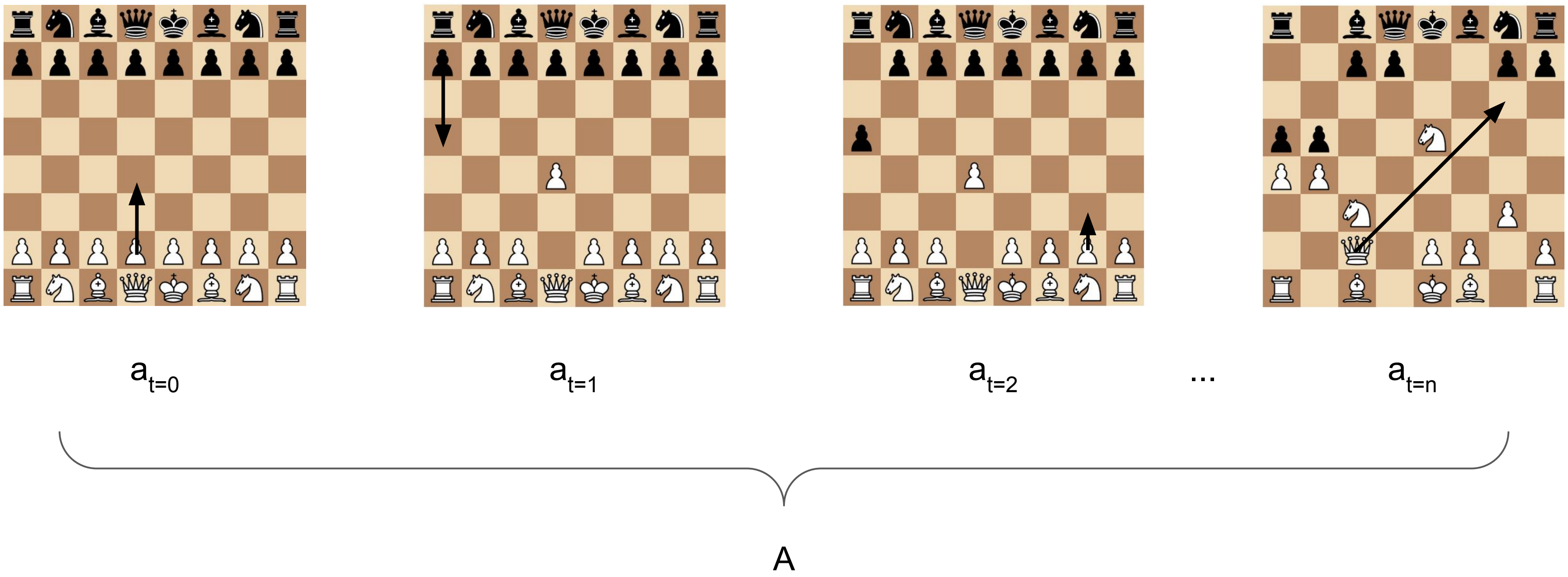


$s_{t=0}$ $s_{t=1}$ $s_{t=2}$ … $s_{t=n}$

$$\underbrace{s_{t=0} \quad s_{t=1} \quad s_{t=2} \quad \dots \quad s_{t=n}}_{S}$$

Reinforcement Learning Terminology

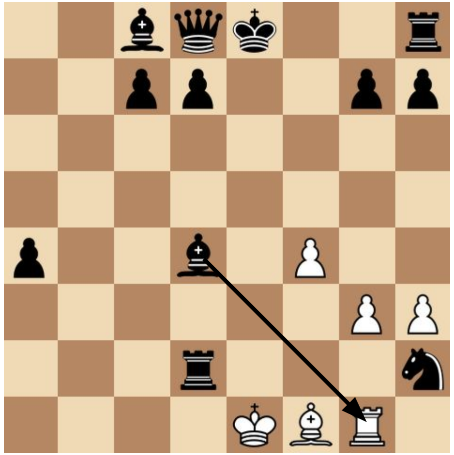
Term	Description
$a_t \in A$	The action an agent takes at time t and the set of all actions an agent can make.



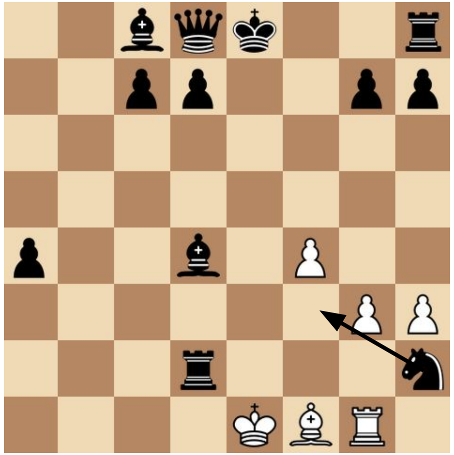
Reinforcement Learning Terminology

Term	Description
$s_t \in S$	The environment state at time t and the set of all states an agent can observe.
$a_t \in A$	The action an agent takes at time t and the set of all actions an agent can make.
$\pi(s, a, \theta)$	The policy that governs an agent's decision making process at a given time step, defined as $P(a_t = a \mid s_t = s, \theta)$.

Policy with untrained parameters

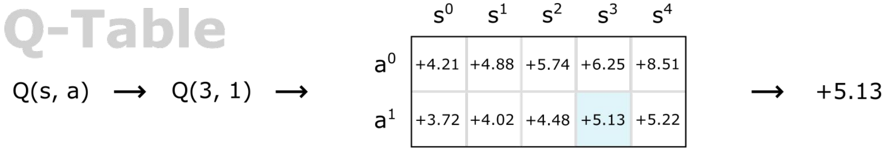


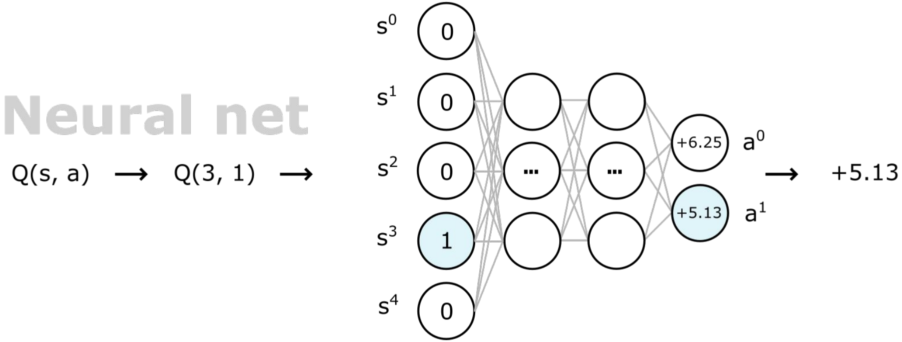
Policy with trained parameters



Reinforcement Learning Terminology

Term	Description
Q(s, a)	Function that predicts the value of an action within a given state (i.e. predicts the maximum future reward that an action will produce).
$Q^*(s, a)$	Function that predicts the optimal action for a given state, action and policy. $Q^*(s, a) = \max E\,[r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \ldots \mid s_t = s, a_t = a, \pi]$
γ	The discount factor applied to future rewards.





https://towardsdatascience.com/reinforcement-learning-tutorial-part-3-basic-deep-q-learning-186164c3bf4

Goal for RL

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \mid s_t = s, \pi\right], \tag{3.1}$$

From page 19

Goal for RL

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, \pi\right], \qquad (3.1)$$

Sum of future rewards

From page 19

Goal for RL

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \mid s_t = s, \pi\right], \tag{3.1}$$

Sum of future rewards | assuming we start at S and use policy π

From page 19

Goal for RL

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \mid s_t = s, \pi\right], \qquad (3.1)$$

Sum of future rewards | assuming we start at S and use policy π

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \mid s_t = s, a_t = a, \pi\right]. \qquad (3.3)$$

From page 19

Goal for RL

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, \pi\right], \tag{3.1}$$

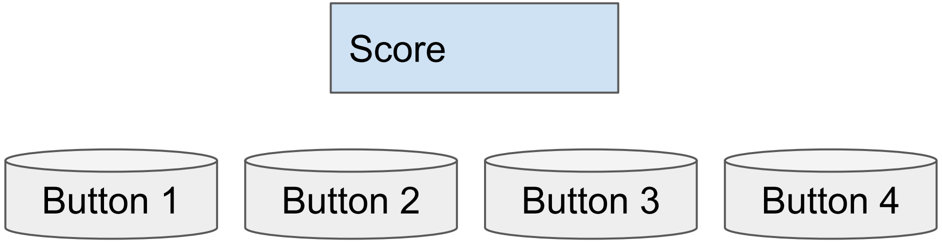
Sum of future rewards | assuming we start at S and use policy π

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right]. \tag{3.3}$$

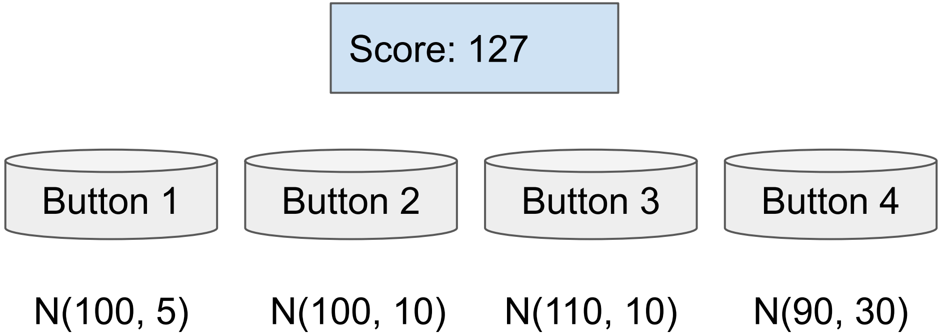
Sum of future rewards | assuming we: start at S,
use policy π,
Selected action a

From page 19

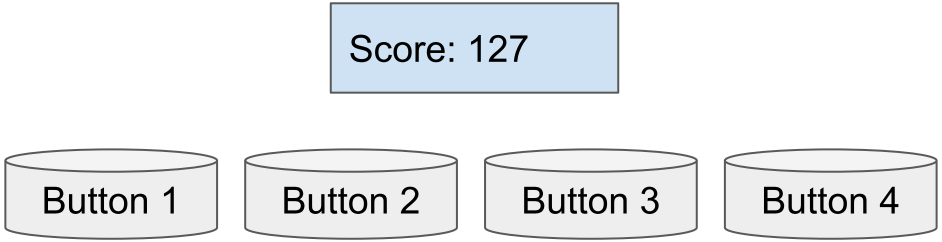
N-armed Bandit



N-armed Bandit



N-armed Bandit



N(100, 5) N(100, 10) N(110, 10) N(90, 30)

Q Table T=0 (mean reward)			
Button 1	Button 2	Button 3	Button 4
0	0	0	0

N-armed Bandit

Score: 127

Button 1 | Button 2 | Button 3 | Button 4

N(100, 5) | N(100, 10) | N(110, 10) | N(90, 30)

Q Table T=4 (mean reward)			
Button 1	Button 2	Button 3	Button 4
111	99	98	72

Policy: Greedy (select best action)

N-armed Bandit

Score: 127

Button 1	Button 2	Button 3	Button 4
N(100, 5)	N(100, 10)	N(110, 10)	N(90, 30)

Q Table T=4 (mean reward)			
Button 1	Button 2	Button 3	Button 4
111	99	98	72

Policy: Greedy (select best action)

N-armed Bandit

Score: 127

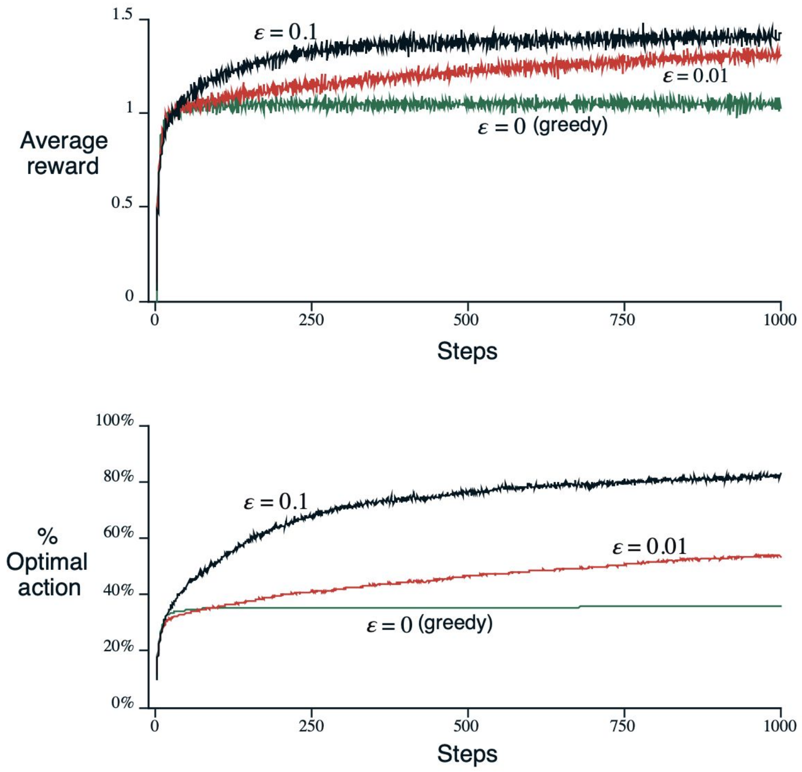
Button 1	Button 2	Button 3	Button 4
N(100, 5)	N(100, 10)	N(110, 10)	N(90, 30)

Q Table T=1000 (mean reward)			
Button 1	Button 2	Button 3	Button 4
100	99	98	72

Policy: Greedy (select best action)

N-armed Bandit - Greedy ε

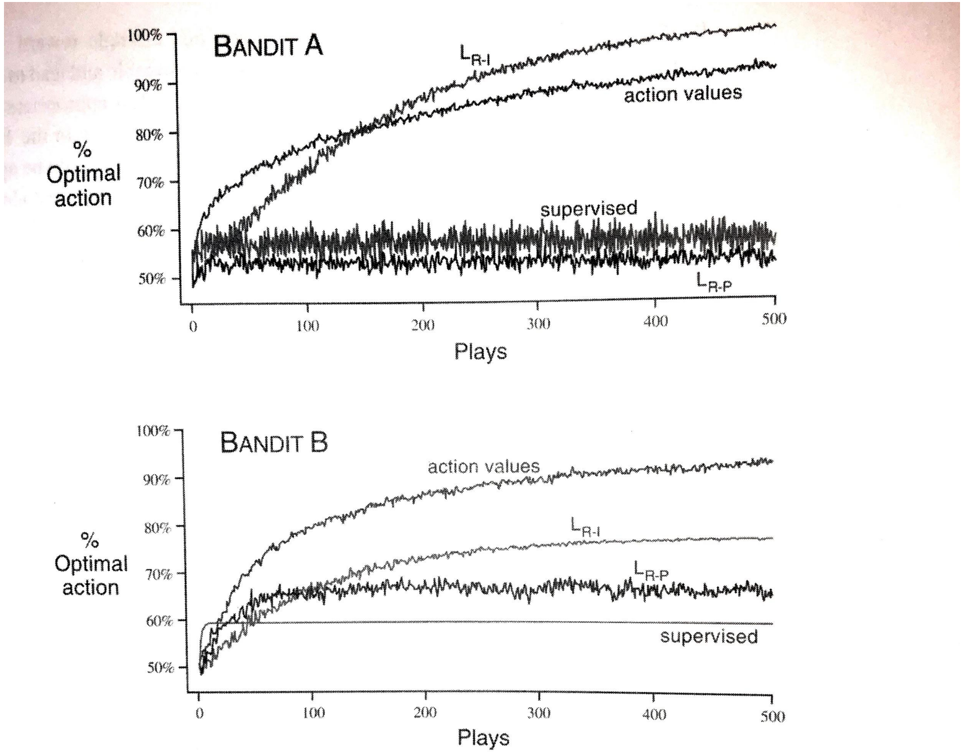
1.0 - ε chance to select best
ε chance to select random



From Reinforcement Learning by Sutton and Barto pg. 29

N-armed Bandit - Lr-p Lr-i

Lr-p (linear reward-penalty)



From Reinforcement Learning by Sutton and Barto pg. 35

N-armed Bandit - Lr-p Lr-i

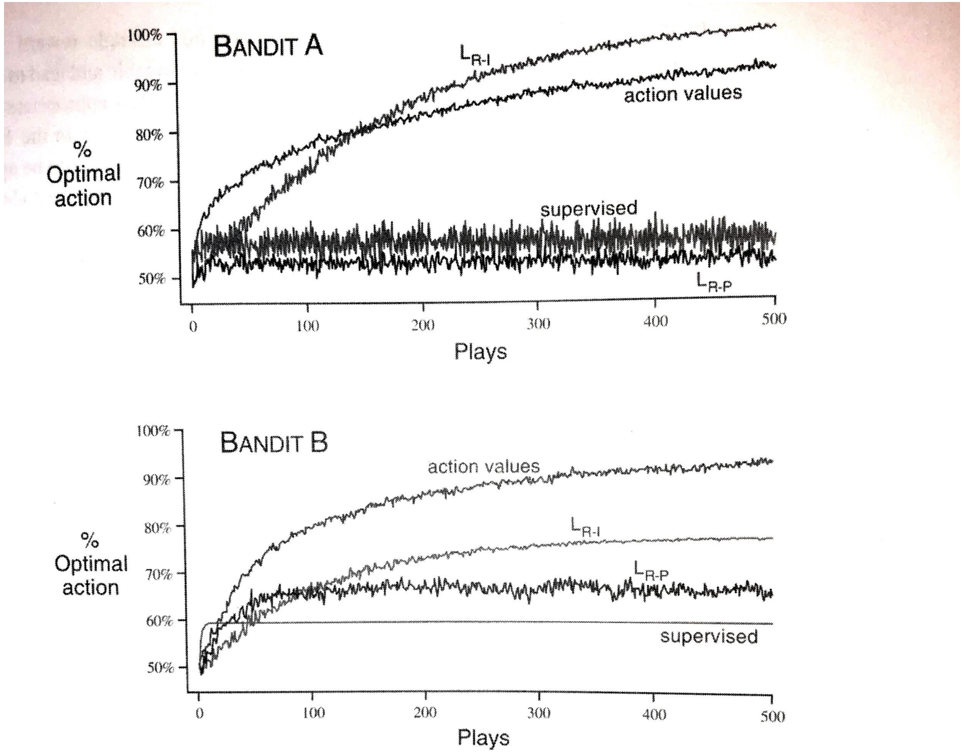
Lr-p (linear reward-penalty)

Success

$$\pi_{t+1}(a) = \pi_t(a) + \alpha(1.0 - \pi_t(a))$$

Failure

$$\pi_{t+1}(a) = \pi_t(a) - \alpha(1.0 - \pi_t(a))$$



From Reinforcement Learning by Sutton and Barto pg. 35

N-armed Bandit - Lr-p Lr-i

Lr-p (linear reward-penalty)

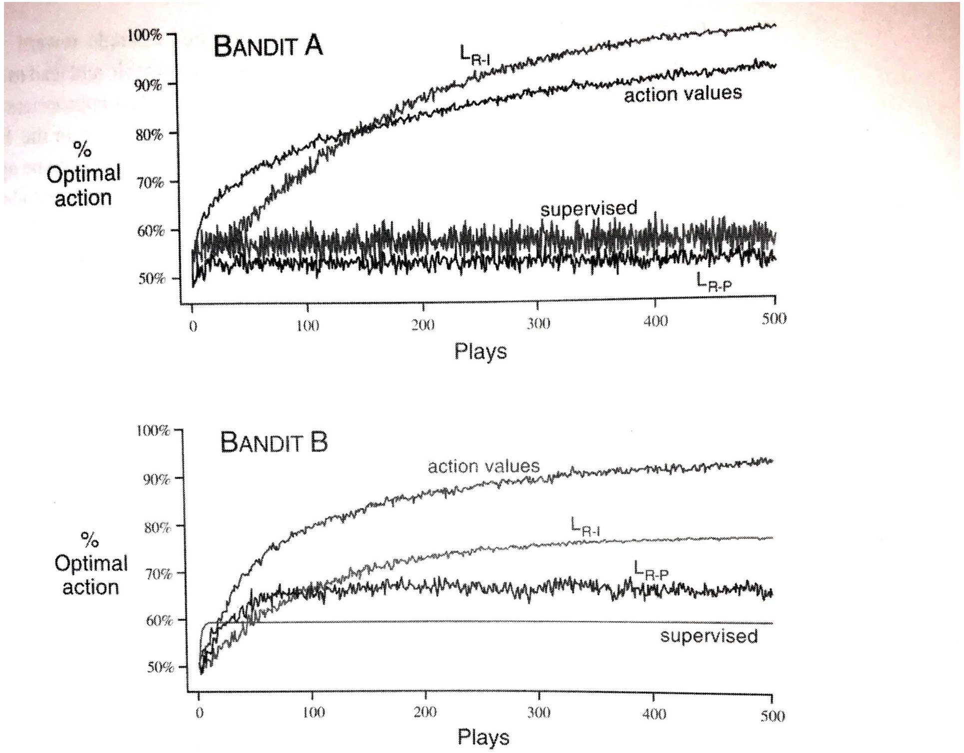
Success

$$\pi_{t+1}(a) = \pi_t(a) + \alpha(1.0 - \pi_t(a))$$

Failure

$$\pi_{t+1}(a) = \pi_t(a) - \alpha(1.0 - \pi_t(a))$$

Lr-i (linear reward-inaction)



From Reinforcement Learning by Sutton and Barto pg. 35

N-armed Bandit - Lr-p Lr-i

Lr-p (linear reward-penalty)

Success

$$\pi_{t+1}(a) = \pi_t(a) + \alpha(1.0 - \pi_t(a))$$

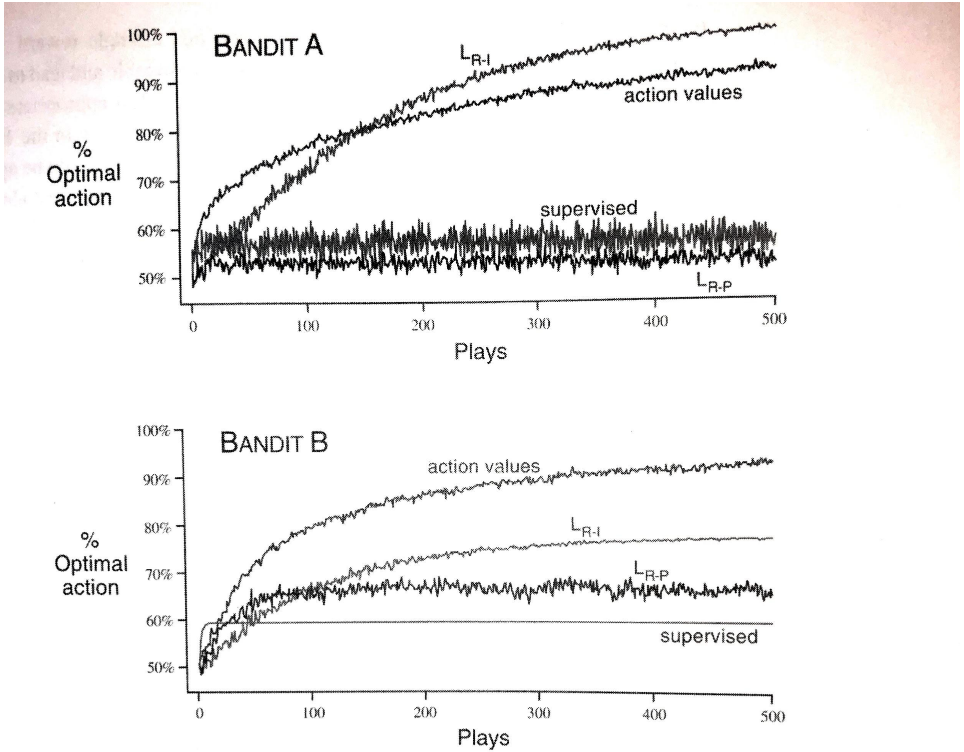
Failure

$$\pi_{t+1}(a) = \pi_t(a) - \alpha(1.0 - \pi_t(a))$$

Lr-i (linear reward-inaction)

Success

$$\pi_{t+1}(a) = \pi_t(a) + \alpha(1.0 - \pi_t(a))$$



From Reinforcement Learning by Sutton and Barto pg. 35

Deep Reinforcement Learning

Deep Reinforcement Learning

- Policy learning

Deep Reinforcement Learning

- Policy learning
- Q learning (DQN)

Parameter update for DQN

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k), \tag{4.3}$$

From page 26

Parameter update for DQN

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k), \tag{4.3}$$

From page 26

Parameter update for DQN

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k), \tag{4.3}$$

$$\mathrm{L}_{DQN} = \left(Q(s, a; \theta_k) - Y_k^Q\right)^2. \tag{4.4}$$

From page 26

Parameter update for DQN

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k), \tag{4.3}$$

$$\mathrm{L}_{DQN} = \left(Q(s, a; \theta_k) - Y_k^Q \right)^2. \tag{4.4}$$

$$\boldsymbol{\theta}_{\boldsymbol{k+1}} = \theta_k + \alpha \left(Y_k^Q - Q(s, a; \theta_k) \right) \nabla_{\theta_k} Q(s, a; \theta_k), \tag{4.5}$$

From page 26

RL Loss Function for DQN

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k), \tag{4.3}$$

$$\mathrm{L}_{DQN} = \left(Q(s, a; \theta_k) - Y_k^Q \right)^2 . \tag{4.4}$$

$$\theta_{k+1} = \theta_k + \alpha \left(Y_k^Q - Q(s, a; \theta_k) \right) \nabla_{\theta_k} Q(s, a; \theta_k), \tag{4.5}$$

From page 26

Parameter update for DQN

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k), \tag{4.3}$$

$$\mathrm{L}_{DQN} = \left(Q(s, a; \theta_k) - Y_k^Q \right)^2 . \tag{4.4}$$

$$\theta_{k+1} = \theta_k + \alpha \left(Y_k^Q - Q(s, a; \theta_k) \right) \nabla_{\theta_k} Q(s, a; \theta_k), \tag{4.5}$$

From page 26

Parameter update for DQN

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k), \tag{4.3}$$

$$\mathrm{L}_{DQN} = \left(Q(s, a; \theta_k) - Y_k^Q \right)^2 . \tag{4.4}$$

$$\theta_{k+1} = \theta_k + \alpha \left(Y_k^Q - Q(s, a; \theta_k) \right) \nabla_{\theta_k} Q(s, a; \theta_k), \tag{4.5}$$

From page 26

Human-Level Control Through Deep Reinforcement Learning

The paper demonstrates how a deep neural network was trained to develop a deep Q-network to learn policies from sensory inputs

The tasks of interest for the study are those in which the agent interacts with an environment through a sequence of observations, actions, and rewards

The goal of the agent is to maximize its future reward

Experiment

Agent: game player

Environment: Atari Games

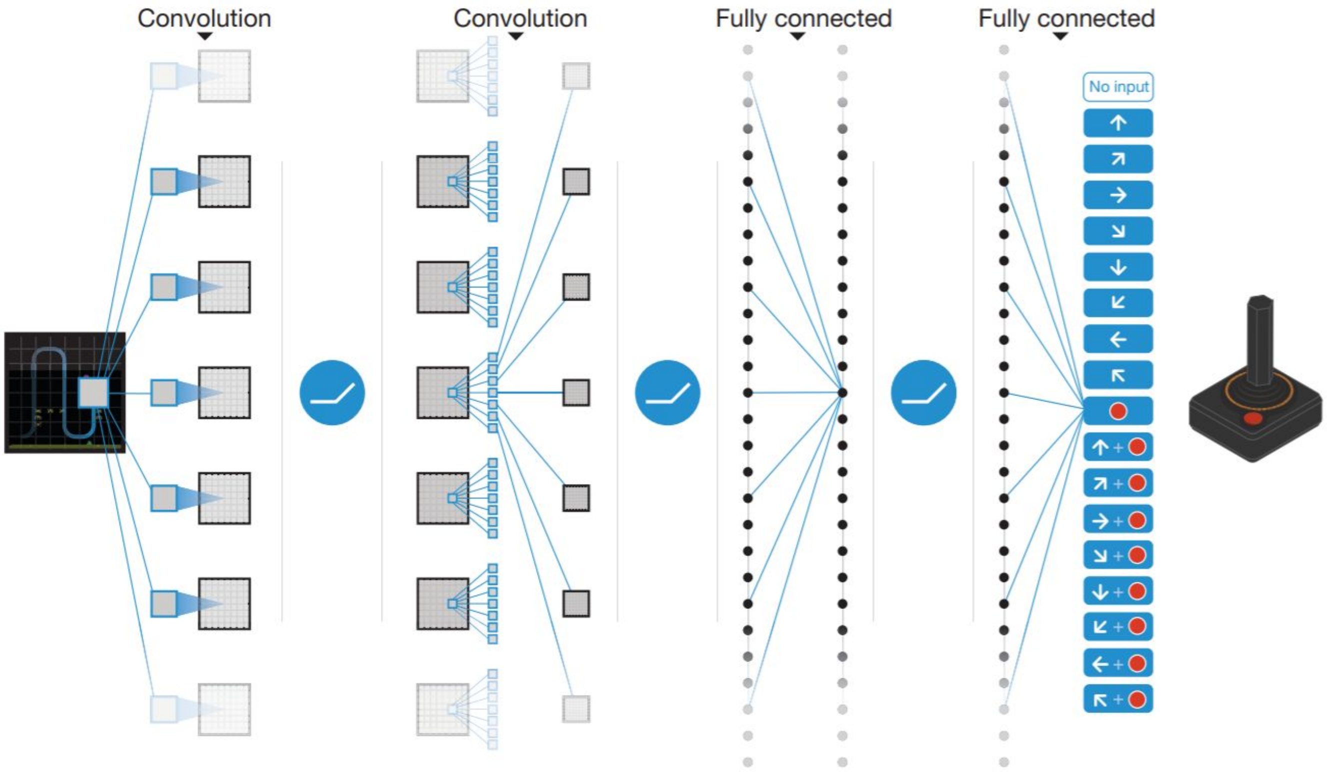
Actions: moves pertaining to game play

States: current position of player in the game and score

Reward: change in game score

Result: successfully trained model to learn to play 49 games just as good if not better than a professional game tester

The Architecture



Replay Memory

Saves sequences of state action pairs at time t with the corresponding reward as well as state at time t+1 to be used to test the current parameters

Allows for data efficiency by averaging the behavior distribution over previous states. This avoids oscillations and divergence in parameters

Random sampling of replay memory breaks up correlation in consecutive moves

Helpful because future actions may be dependent on actions taken many timesteps back

Problems with Q-Learning

Reinforcement learning is known to be unstable

Caused by correlations present in the sequence of observations

Repeated, small updates to Q may significantly change the policy which changes the data distribution and therefore correlations between actions and targets

Solved by updating parameters every k timesteps instead of every timestep

The Action-Value Function: Q

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \middle| s_t = s,\, a_t = a,\, \pi\right]$$

The Loss Function

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i) \right)^2 \right]$$

The Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialization
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode $= 1, M$ **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

Choose Action
With probability ε select a random action a_t
otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Update Replay Memory
Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Update Parameters and Compute Target
Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$$

Gradient Descent
Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

Clone/Update Q Function
Every C steps reset $\hat{Q} = Q$

End For

End For