Deep Q Network (DQN)

John Tan Chong Min

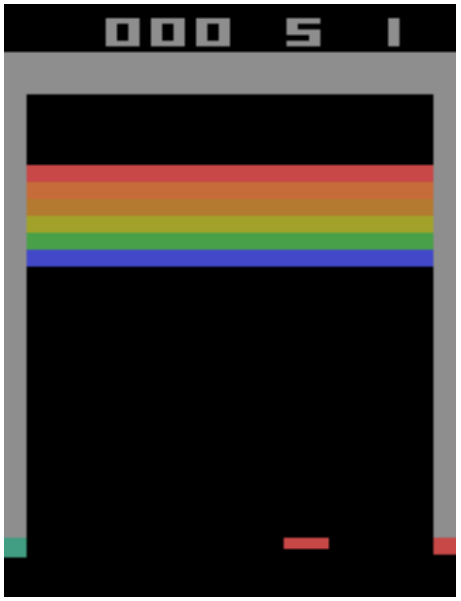
Background

- Written by Mnih et. al. (2013) at Google DeepMind

- Motivation:
  - Creating agents that learn from high-dimensional sensory input that can perform well on general scenarios
  - Use neural networks to do functional approximation
  - Same algorithm used for multiple environments – step towards general AI

Introduction

- https://www.deepmind.com/blog/deep-reinforcement-learning



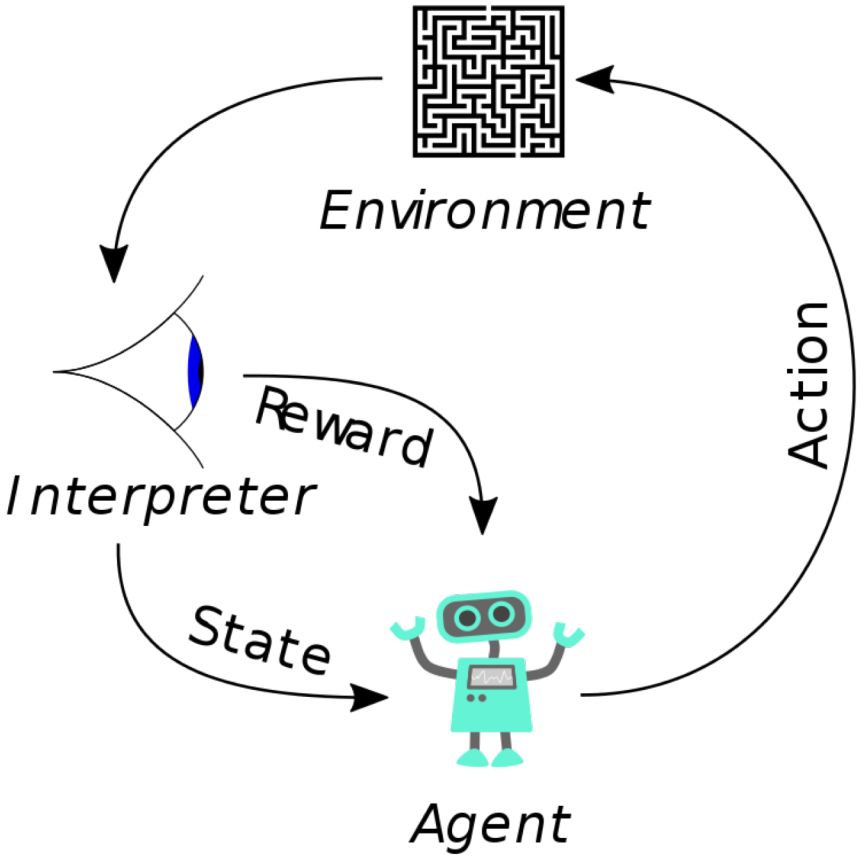
Breakout

Space Invaders

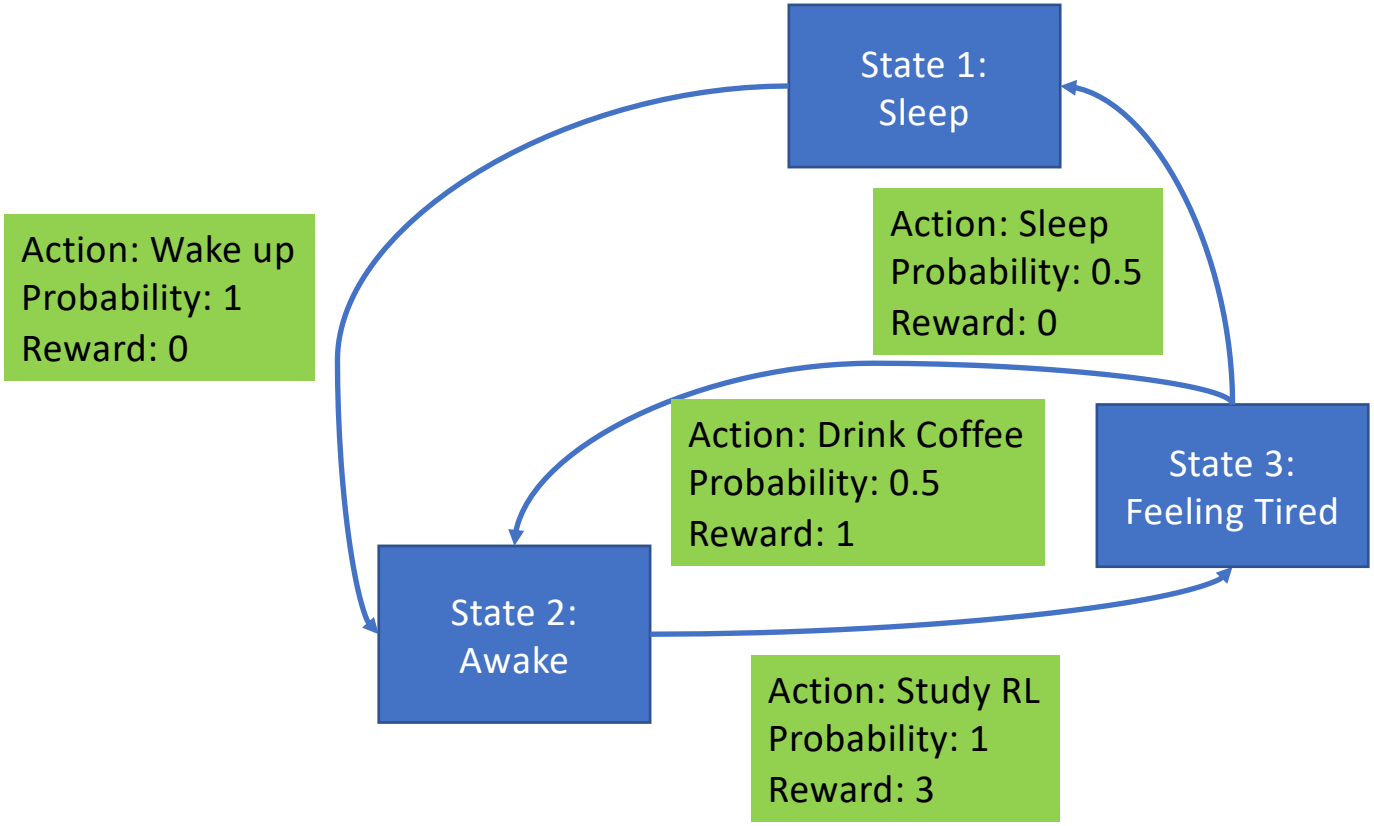
Recap: Reinforcement Learning (RL)

- For a Reinforcement Learning (RL) agent, we have
  - Environment $\mathcal{E}$
  - State $s \in S$
  - Action $a \in A$
  - Reward $r \in R$
  - Transition Function $P(\cdot\,|s, a)$
  - Discount factor γ

- Interaction with environment can be modelled as Markov Decision Process (MDP)



Recap: Markov Decision Process (MDP)

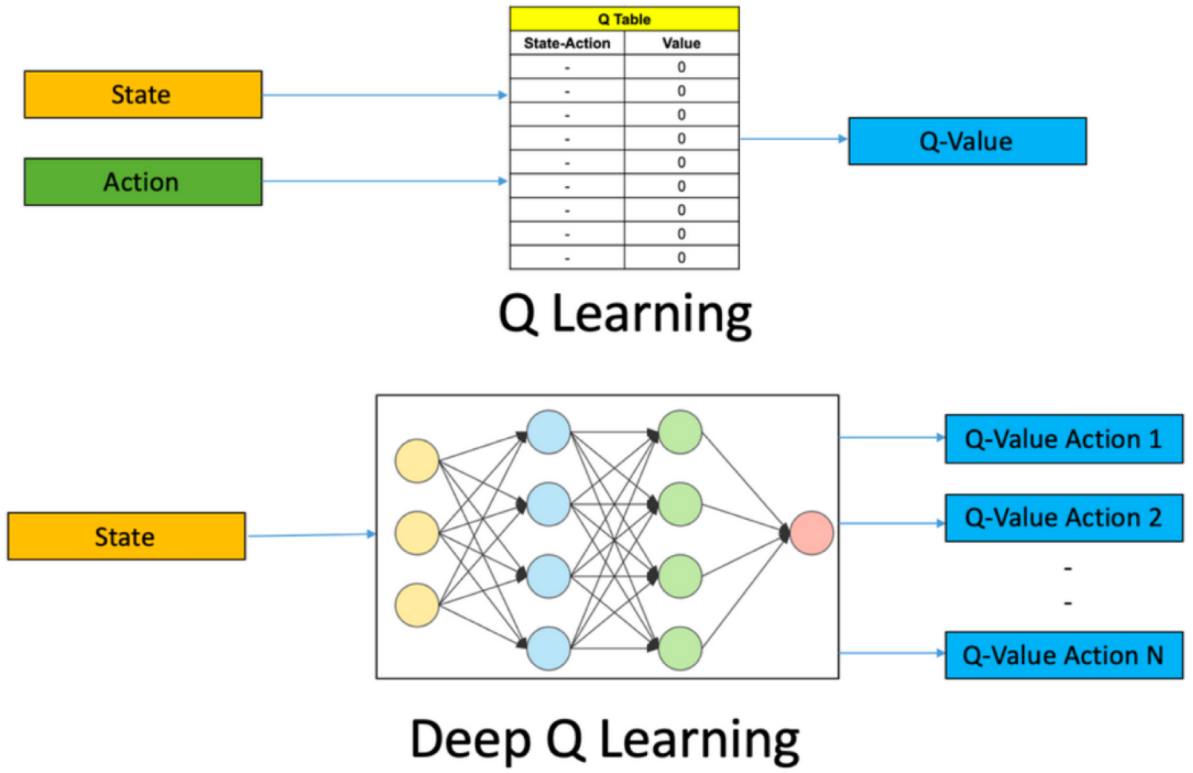


MDP states are memory-free.

Knowing the state itself is sufficient to do planning, no need for past state histories.

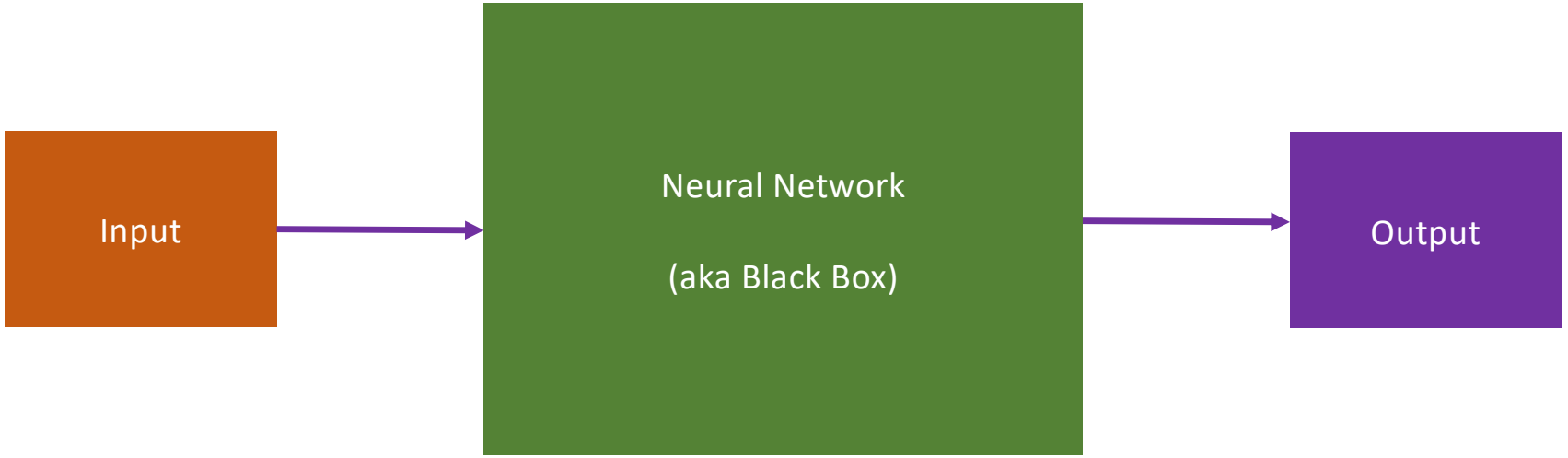
Question: What if your game/environment requires a history of past states?

From Tabular Q-Learning to Deep Q-Learning

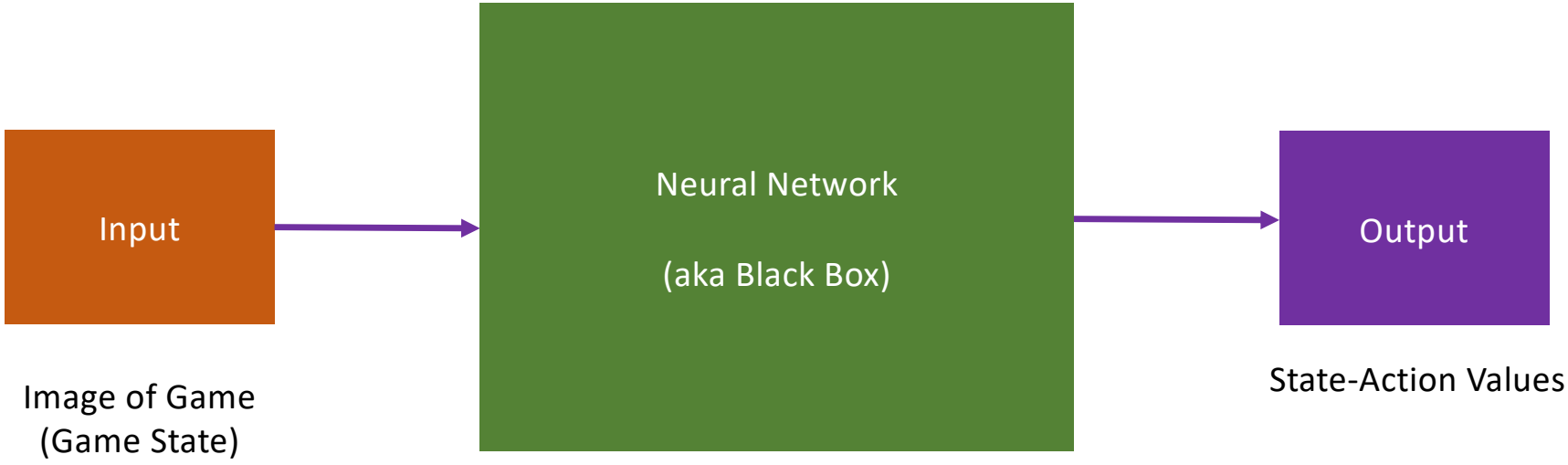


Extracted from: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

Neural Network

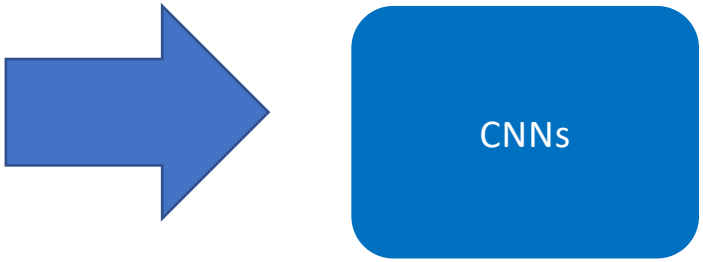


Neural Network for DQN

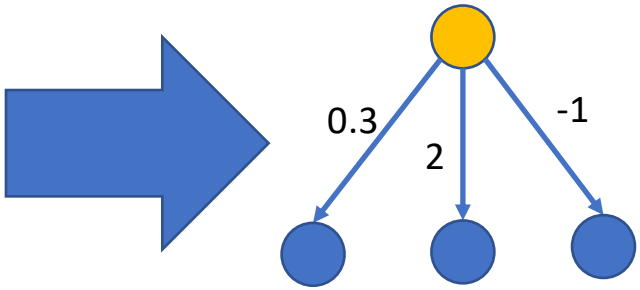


Neural Network for DQN

Image of Game
(Game State)

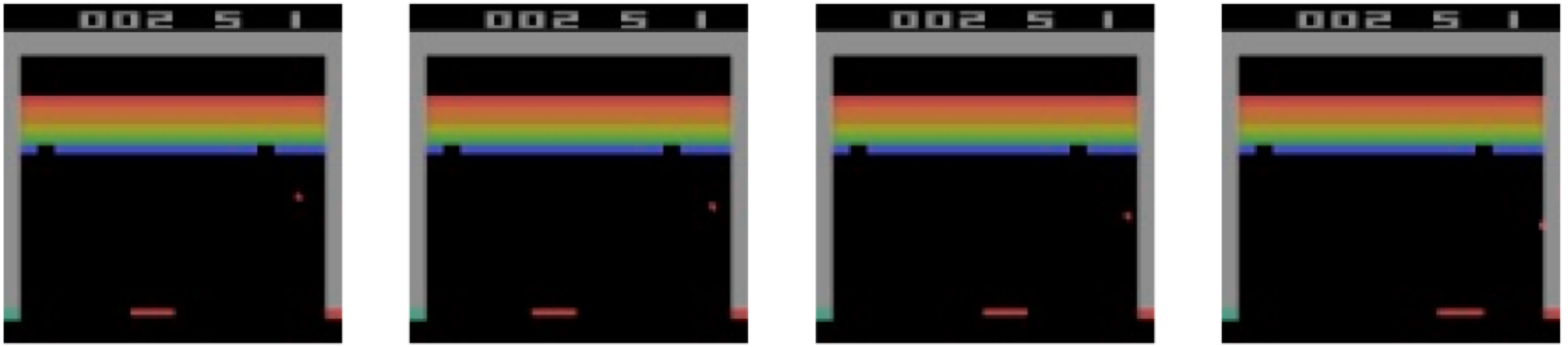


Neural Network



State-Action
Values

State, Action, Reward

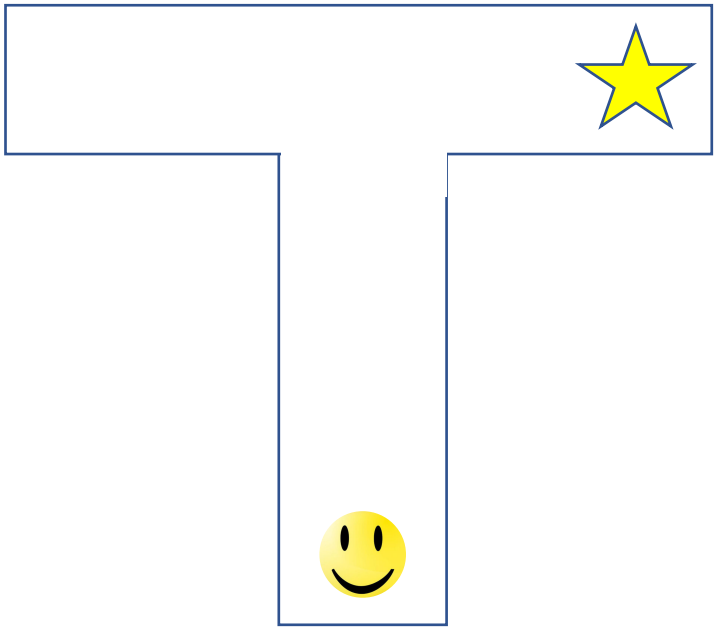


State:
4 frames of pixels
84 x 84 pixels x 4 frames

Action:
Left / Right

Reward:
Total Score

Intuition of Q-learning



Q-Learning

- Based off Temporal Difference (TD) Learning

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)}^{\text{temporal difference}}$$

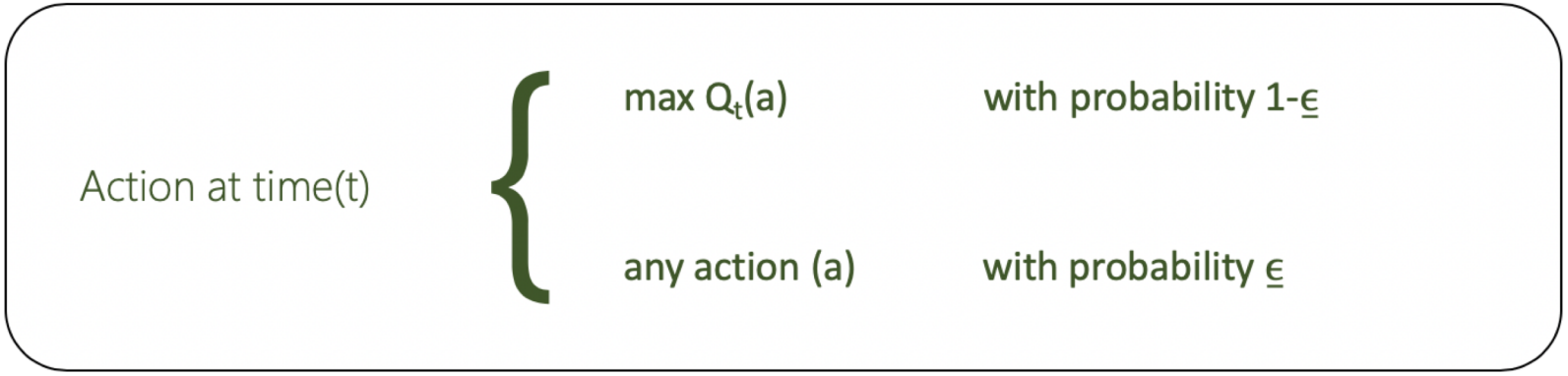
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

Extracted from: https://en.wikipedia.org/wiki/Q-learning

How to generate experiences?

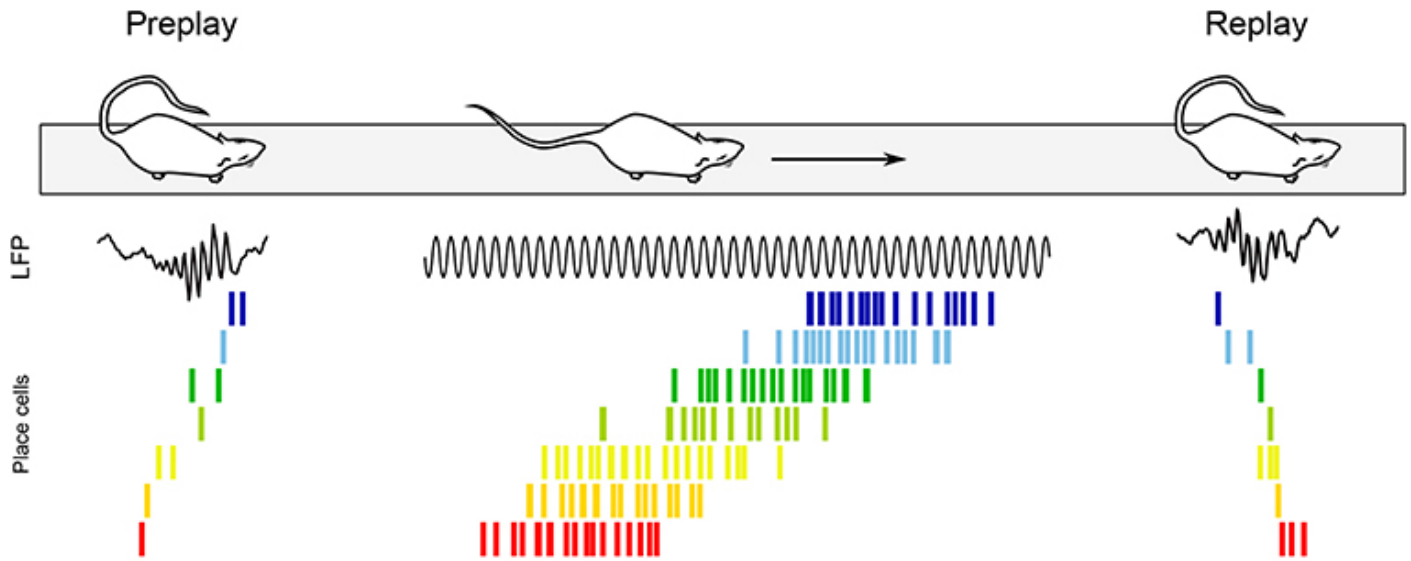
- One way is to generate all possible state-action pairs

- Another way is to generate common state-action pairs
  - Behaviour policy: Epsilon-greedy



How to learn from experience?

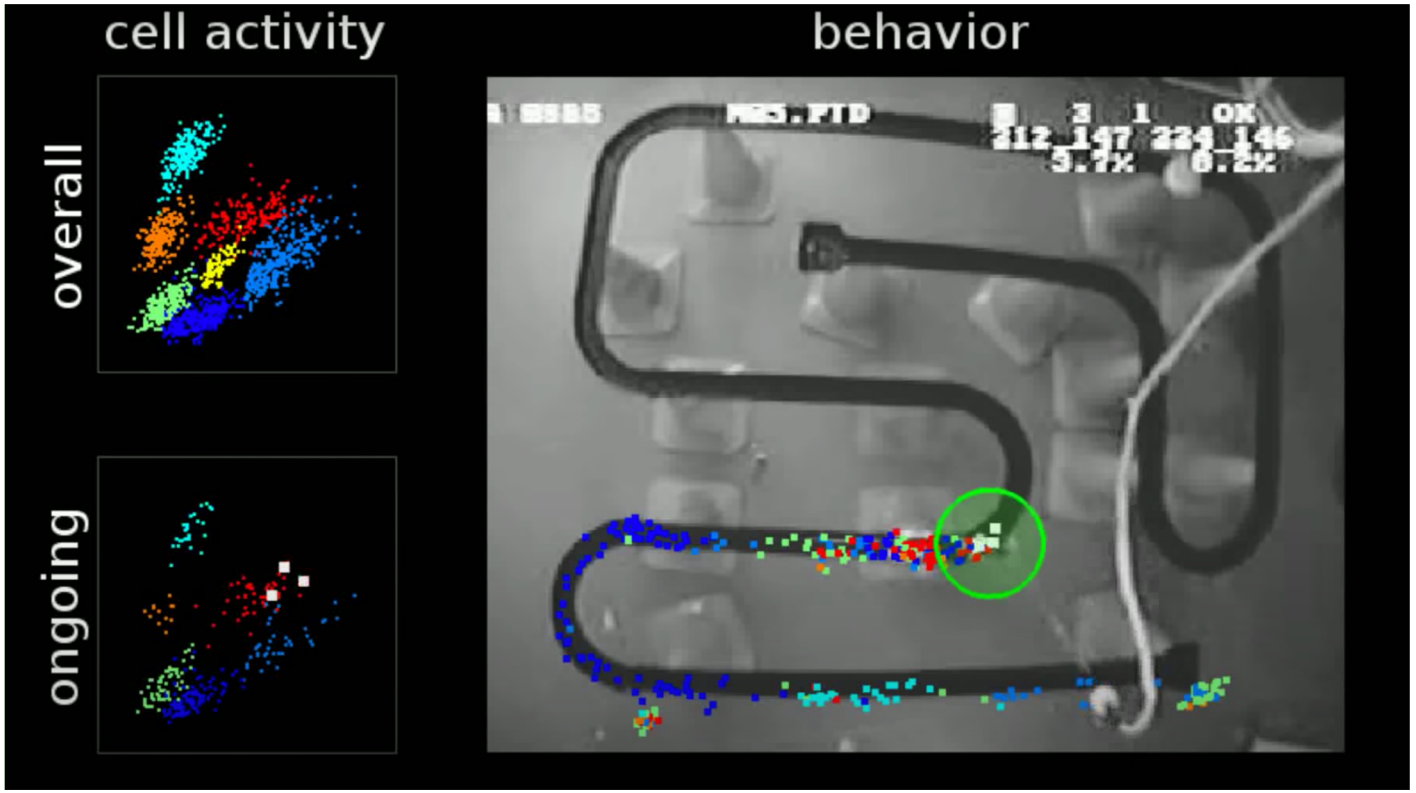
- Learn randomly?
- Rats learn from their daily experiences backwards (hippocampal replay)



Extracted from: Drieu et. al. Hippocampal Sequences During Exploration: Mechanisms and Functions, Frontiers in Cellular Neuroscience (2019)

Video on Hippocampal Replay

- https://www.youtube.com/watch?v=lfNVv0A8QvI



Q-Value: State-Action Value

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') \middle| s, a \right]$$

- Q value measures how good an action *a* is at state *s* (the higher the better)
- Every state *s* and action *a* has an optimal Q-value *Q*(s, a)*
- The update of optimal Q-value is done iteratively taking into account next time step *s'* and *a'* given current state *s* and action *a*
  - Known as Bellman Equation

Learning Q-Values

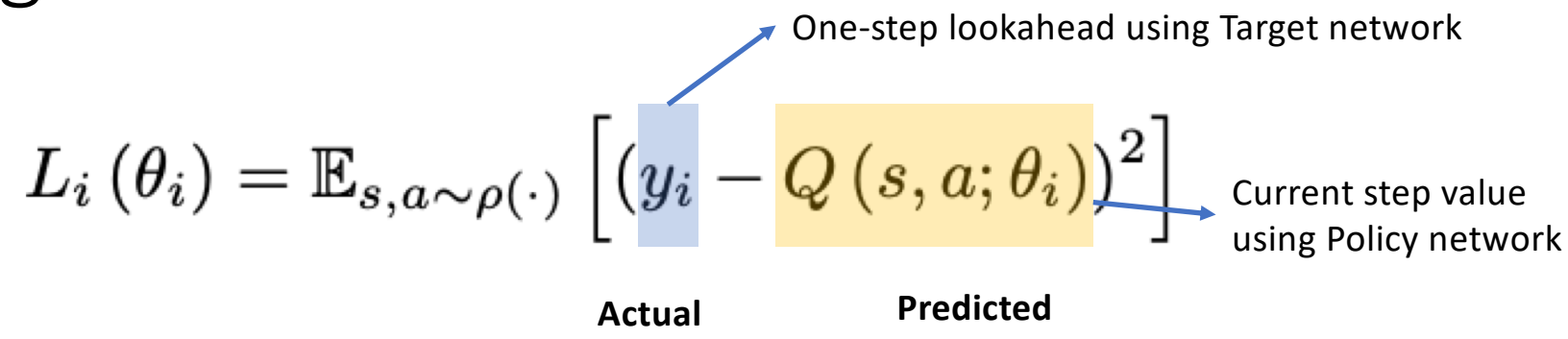
$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

Actual **Predicted**

- To learn, network minimizes Mean Squared Error between **Predicted** and **Actual** Q-values

- To prevent correlation between consecutive samples, training is done via a **replay buffer** which samples transitions randomly

Learning Q-Values



$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})|s, a\right]$$

- Q-value function approximated by two neural networks
  - **Policy** (trainable)
  - **Target** (fixed) – helps mitigate issues of moving target destabilizing training!

- After some iterations, Target network updated to Policy network's weights

Visualizing Q-values

Figure 3: The leftmost plot shows the predicted value function for a 30 frame segment of the game Seaquest. The three screenshots correspond to the frames labeled by A, B, and C respectively.

Choosing the Optimal Action

$$a^* = \max_a Q(s, a; \theta)$$

- After Q-value is learnt, optimal action a* is action that gives maximal Q-value using policy network

- Key insight of DQN:
  - Q-value can be approximated by a neural network
  - Exponential increase of possible state space

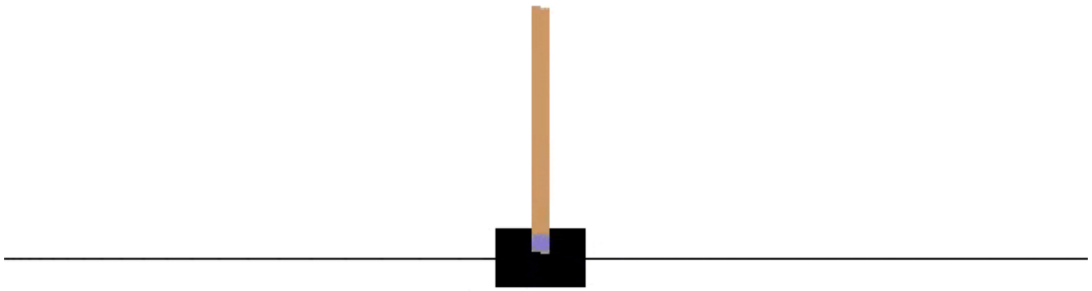
Improving DQN

- Incorporate Planning
  - DQN is model-free: No planning done when choosing actions
  - Can utilize learnt Q values to choose actions to explore-exploit
  - Similar to MCTS

- Apply it to continuous action spaces
  - Use a policy network to choose actions rather than doing argmax of next state values
  - Policy Gradient methods

How hard is it for an AI to do this?

Practical Time!



Cart Pole: Balance a pole on a cart for as long as possible

State space:
Cart position, velocity
Pole angle, velocity

Action:
Left or Right

Reward:
+1 for every timestep pole is balanced

Jupyter Notebook

DQN on Cart Pole