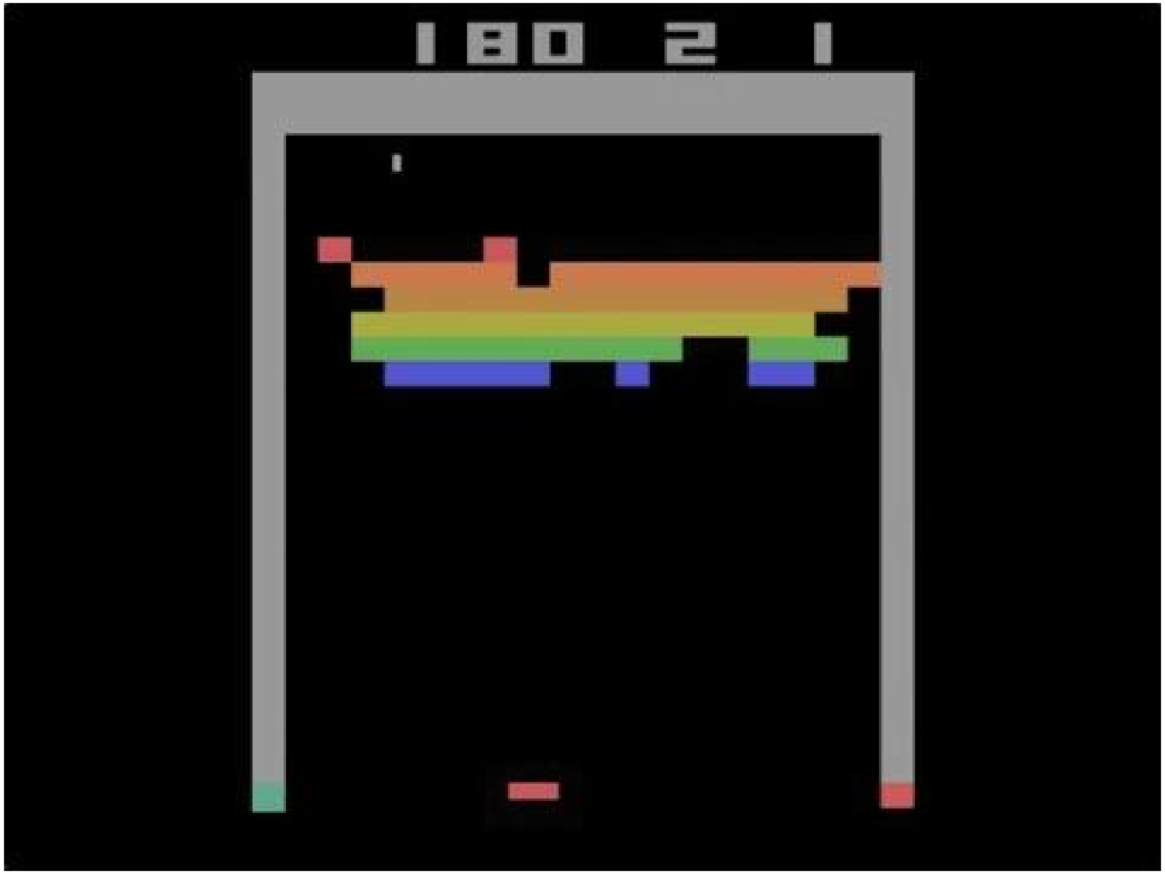
A Practical Introduction to Deep Reinforcement Learning

Rohan Taori and Brenton Chu

Hosted by Machine Learning @ Berkeley

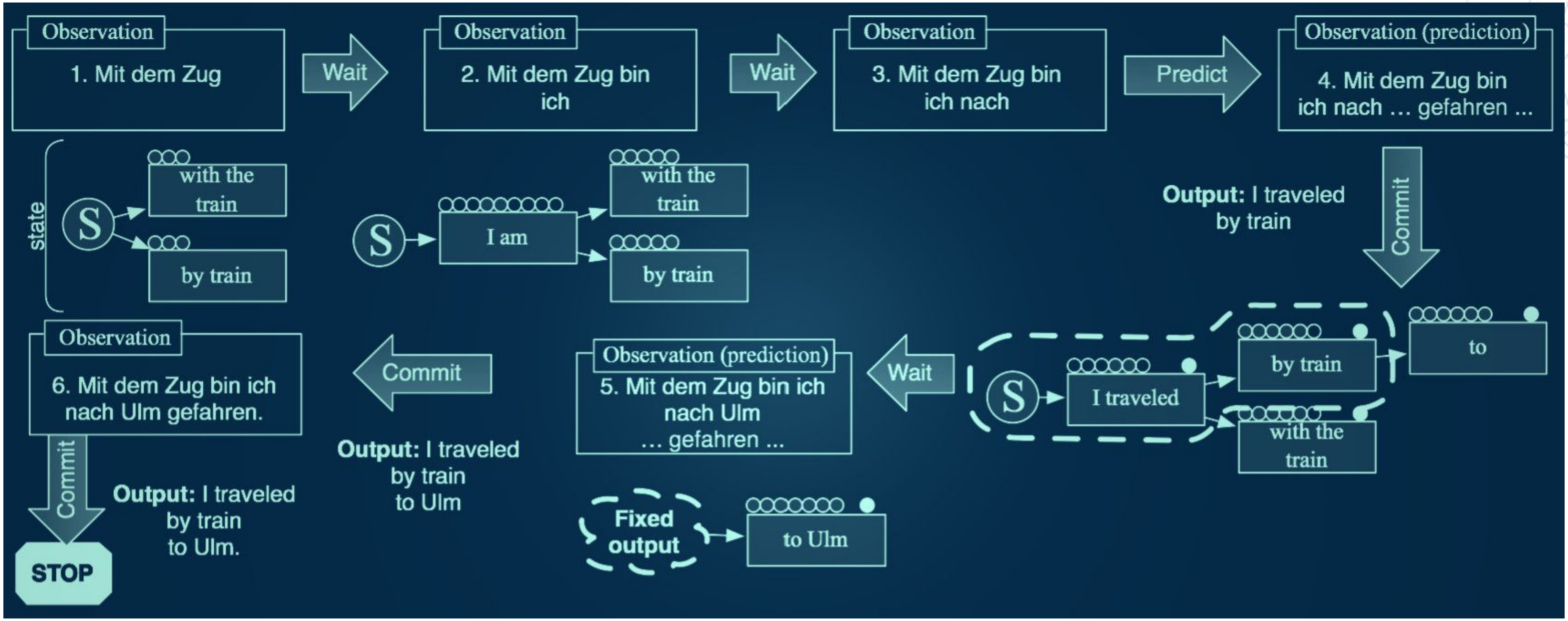
Video Example: DQN



Alpha Go

Video Example: Doom

Not just games! Image: Translation



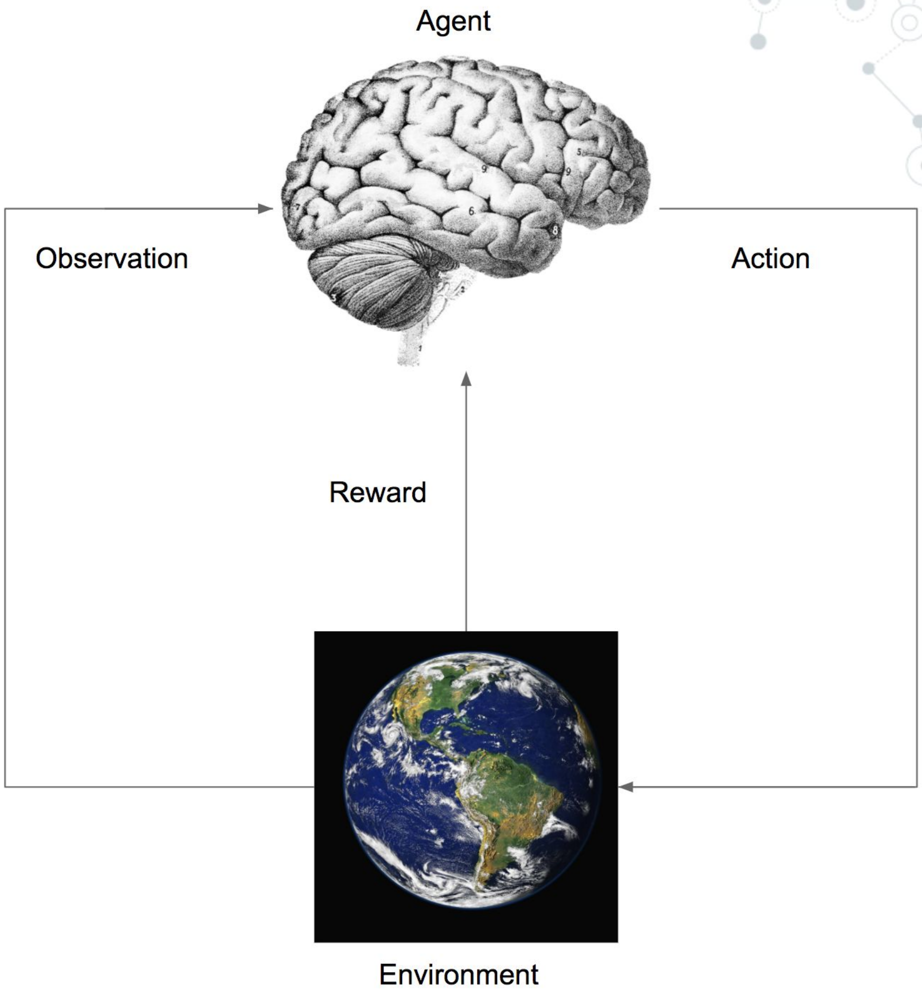
What is Reinforcement Learning?

Problem Setup

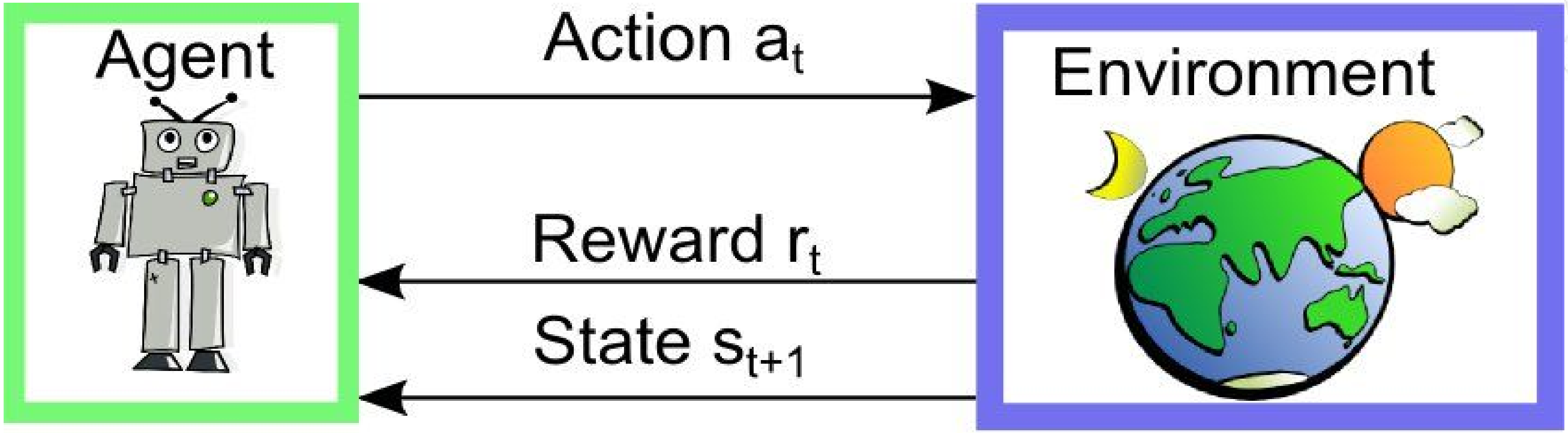
-Problem moves by time step

-At each step the agent sees a state and a reward.

-The agent uses that information to decide an action



Problem Setup



Reinforcement Learning Setup

Goal

Goal

- For every action the agent takes, it sees a reward and a new state

- We want the agent to **maximize** all the reward it can get

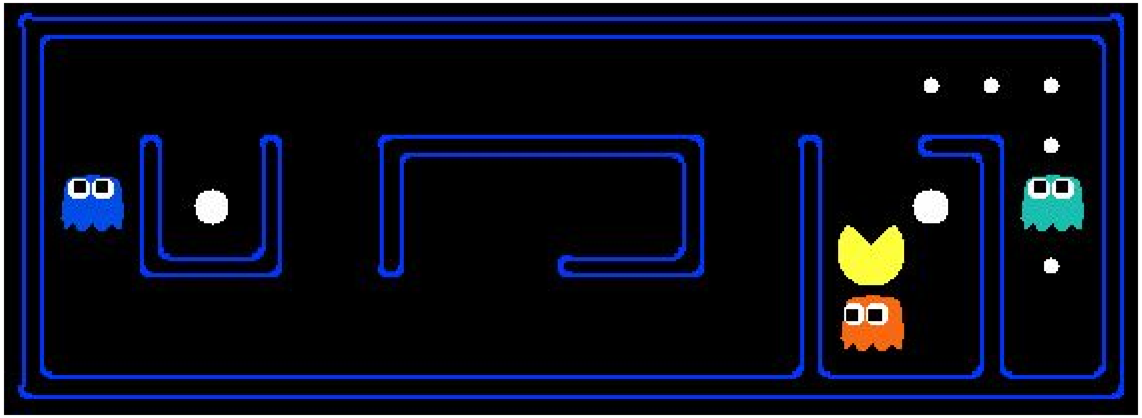
$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T$$

- let T go to infinity

- include a discount factor gamma to weight immediate rewards more than future ones

Goal

- Why the discount factor?



- sooner rewards are probably more useful than later ones

- helps our algorithms converge

The Value Function

- So now we know that we want to maximize expected future reward

- What can we do based on this?

- Let's try and see what our expected future reward from each state looks like! Let's call it the **value** of that state

$$V^{p}(s) = \sum_{k=1}^{+\infty} \gamma^{k-1} r_k = r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots$$

The Value Function

$$V^{p}(s) = \sum_{k=1}^{+\infty} \gamma^{k-1} r_k = r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots$$

- $\boldsymbol{\gamma}$ is a hyperparameter between 0 and 1 and is called the “discount”.

-$\mathbf{r_t}$ is the reward at timestep **t**

- Or in other words:

$$V^*(S) = \max_a \left[R(s,a) + \gamma \sum_{s' \in \mathbb{S}} p(s'|s,a) V^*(s') \right]$$

-The Value Function maps a value to a state, but it doesn’t tell us what action to take.

The Value Function

$$V^*(S) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^*(s') \right]$$

- Requires us to know the transition probabilities beforehand
- In other words, we must have a **model** of the transition dynamics of the system
-The Value Function maps a value to a state, but it doesn't tell us what action to take.
- Both of these are problems

Q-Values

-Solution: Estimate Q-Values instead! *Q(s,a)* returns the value of taking an action *a* in a state *s*.

-Now, it is easy to determine what action to take given a state.

$$\operatorname*{argmax}_{a} \{Q(s,a)\}$$

- Very similar recursive formula:

$$Q'(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \max_{\alpha} \{Q'(\mathcal{S}(s,a),\alpha)\}$$

Q-Learning

Initialise the Q' table with random values.

1. Choose an action a to perform in the current state, s.
2. Perform a and receive reward $\mathcal{R}(s, a)$.
3. Observe the new state, $\mathcal{S}(s, a)$.
4. Update:

$$Q'(s, a) \leftarrow \mathcal{R}(s, a) + \gamma \max_{\alpha} \{Q'(\mathcal{S}(s, a), \alpha)\}$$

5. If the next state is not terminal, go back to step 1.

-R(s,a) returns the reward of taking action *a* in state *s*

-S(s,a) returns the next state, *s'*, after taking action *a* in state *s*.

Q-Table Learning

-Maintain a huge table of Q(s, a) values

- One axis for all actions, one axis for all states

- Given current state s:

    - Take action a

    - Observe reward r and new state s’

    - Update Q(s, a)

$$Q\,(s, a) = Q(s, a) + \alpha\,[\, r + \gamma \max_{a'} Q\,(s', a') - Q\,(s, a)]$$

$$\begin{array}{rcc} & & \text{Action} \\ & \text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ R= & \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Shortcomings of Basic Q-Learning

-It does not work with continuous action spaces.

-It performs very poorly in very large state spaces.

-It requires a huge table as well for large state spaces.

-Tables do not scale well for larger problems in general.

-For example, for the DQN Atari setup we would need...

$$(84*84*4*256)*18 = 130056192 \text{ Values}$$

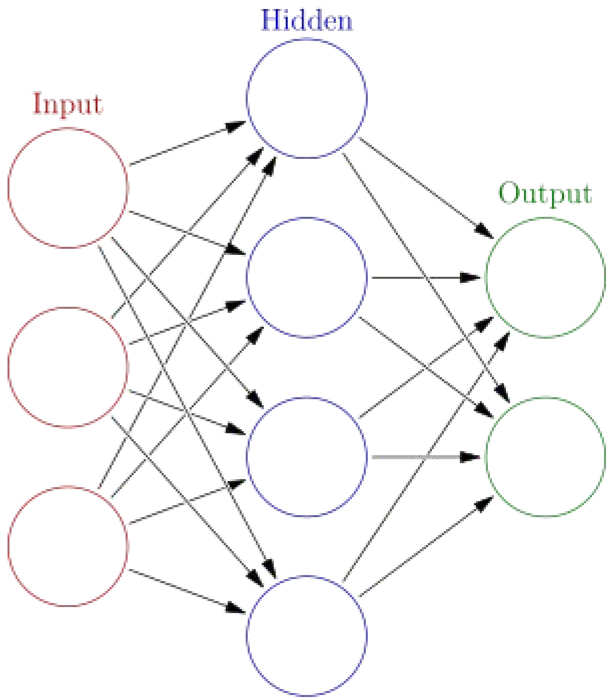
- 4 frames of 84x84 pixels, each with 256 color values, and 18 actions

Neural Networks!

On Neural Networks

-Neural Networks approximate a function given a sufficient number of inputs and outputs.

-We are trying to approximate the function Q(s,a).



On Neural Networks

-Neural Networks approximate a function given a sufficient number of inputs and outputs.

-We are trying to approximate the function Q(s,a).

$$loss = \Bigg(\underbrace{\overset{\text{Reward}}{\mathrm{r}} + \overset{\text{Decay Rate}}{\gamma} \max_{a`} \hat{Q}(s, a`)}_{\text{Target}} - \underbrace{Q(s, a)}_{\text{Prediction}}\Bigg)^2$$

-Comes from the Bellman formula

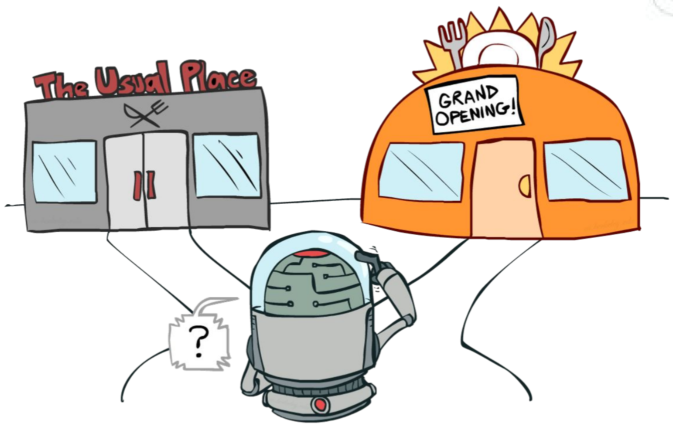
$$Q'(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \max_{\alpha} \{Q'(\mathcal{S}(s,a),\alpha)\}$$

Exploration vs. Exploitation

-Exploring = less reward, but understand environment better

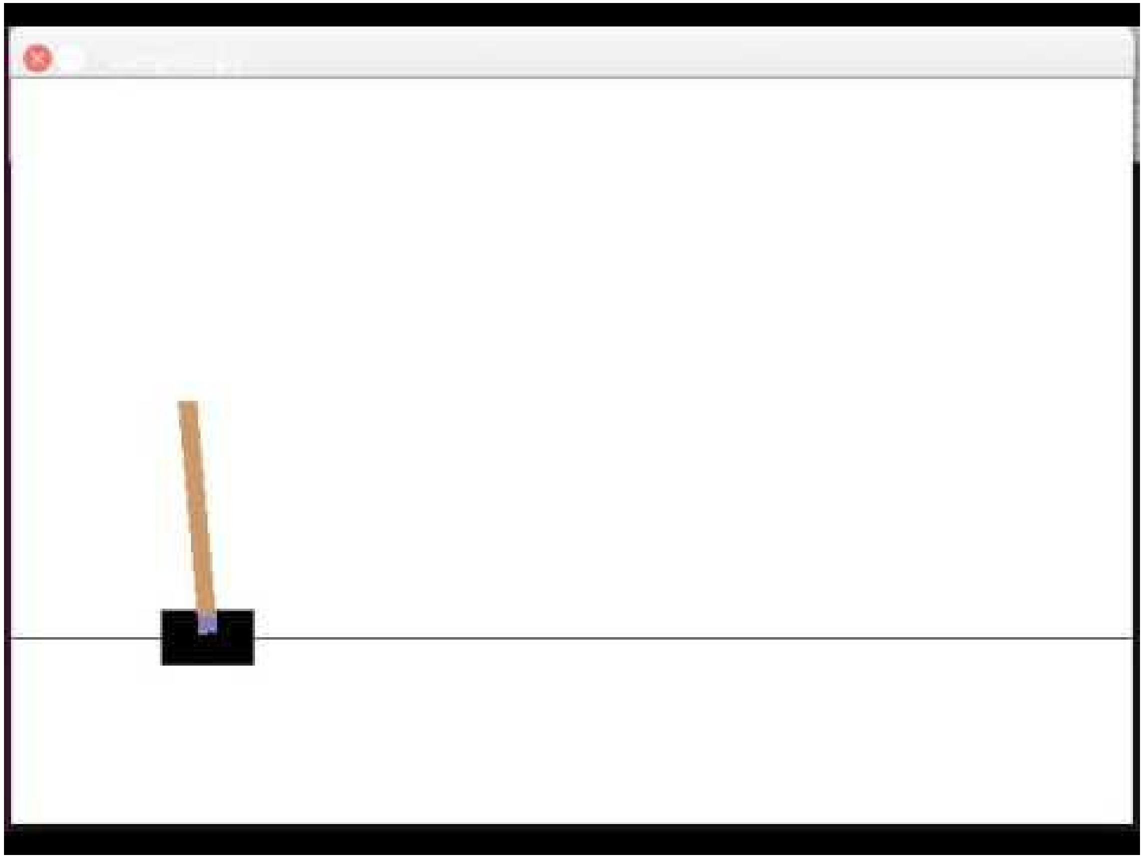
-Exploit = more reward, may be missing out on an important part of environment

- ε-greedy action selection: We take a random action with probability ε and decrease ε over time.



Coding It All Up

Introducing the environment: Cartpole



Experience Replay

-The big innovation!

-We to store the SARS in a buffer and then randomly sample to train the network.

-Similar to how animals/humans recall previous experiences while learning. (https://www.nature.com/articles/nature14236)

Target Networks

-Another big step!

-Use a slightly outdated version of the network

- Only after N episodes, accumulate the gradients and update the network

- Can be implemented with 2 networks - old and new

- New network's weights transfer over after N episodes

- Most common now to use exponentially weighted average

Drawbacks of Q learning

-Doesn’t work with continuous actions spaces

- Need (s, a) pairs but can’t enumerate continuous actions!

- Convergence in Q networks is finicky
  - Various methods to get around this:
    - replay buffer, target networks, double q-learning, n-step q learning, distributed simultaneous updates (A3C)

Papers to Read

Where to go from here?

-There are some very simple and effective improvements on the code that we just wrote.

-Here are some easy papers you can read and implement with the code we just wrote!

Policy-Based Methods

-We just went over a Value-Based Method

-Learn Policy (function mapping states to action) directly instead of Q-Values in the neural network.

DDPG:

https://arxiv.org/abs/1509.02971

A3C:

https://arxiv.org/abs/1602.01783

PPO:

https://arxiv.org/abs/1707.06347

Deep Reinforcement Learning with Double Q-learning

https://arxiv.org/abs/1509.06461

-September 2015

-Target Network to estimate Q-values, use online network for actions

-Prevent overestimation of state values

- Since in the Bellman equation, each state uses the max of the next state

-Repeated max operations can lead to Q value overestimation

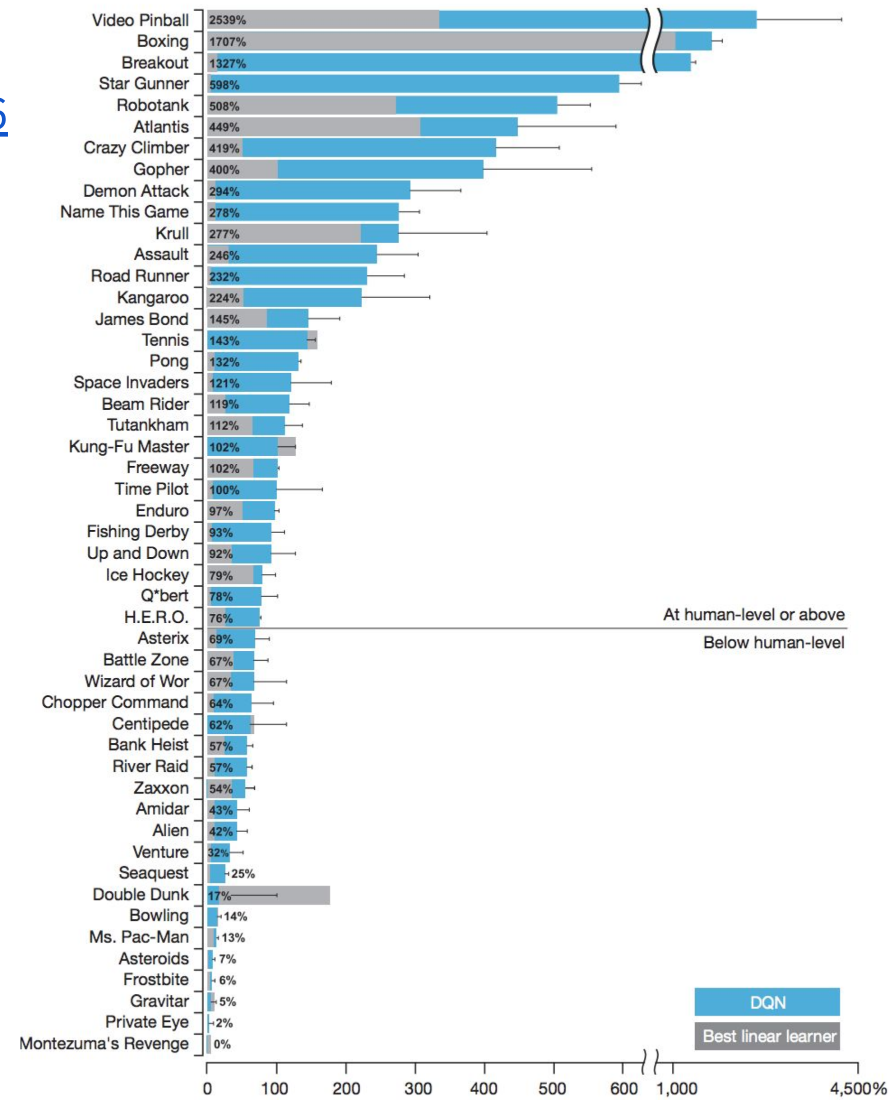
Human-level control through deep reinforcement learning

https://www.nature.com/articles/nature14236

-Nature, 2014 - rise of Deep RL

-Introduces DQN.

-Adds a “target network” to our implementation



Parameter Space Noise for Exploration

https://arxiv.org/abs/1706.01905

-June, 2017

-Inject noise into parameters /weights of network.

-Better than \eps-greedy exploration

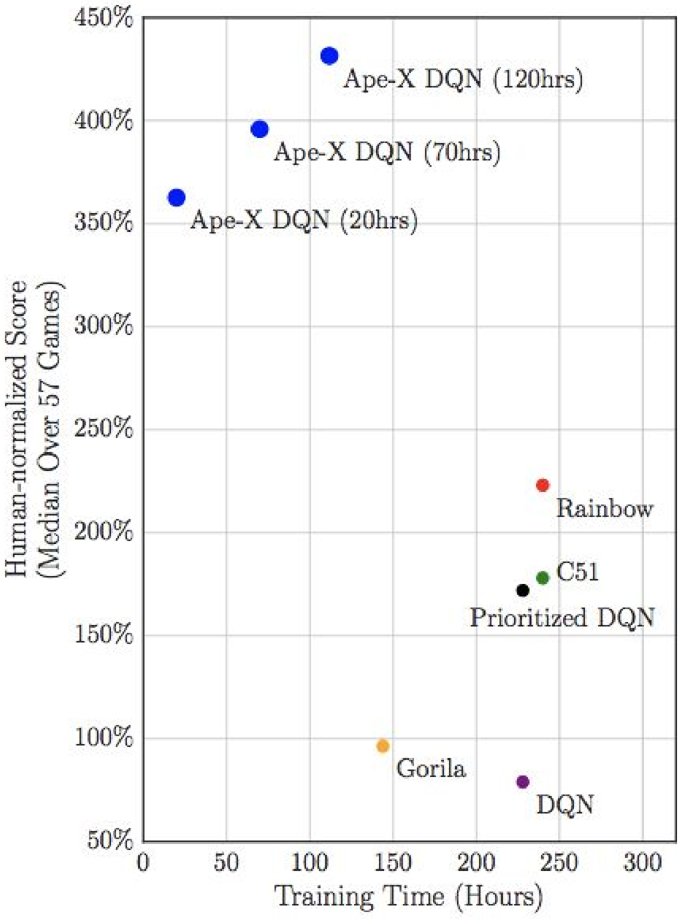
Distributed Prioritized Experience Replay

https://openreview.net/pdf?id=H1Dy---0Z

-Very new paper

-Large number of threads running the game at the same time to update the experience buffer.

-Uses "prioritized experience replay" which samples from the experience buffer based on how much it "learns" from the sample.



Active Areas

-Hierarchical Reinforcement Learning

Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation.

https://arxiv.org/abs/1604.06057

-Model-Based Reinforcement Learning

Learning model-based planning from scratch

https://arxiv.org/abs/1707.06170

-Improved Exploration

Curiosity-driven Exploration by Self-supervised Prediction

https://arxiv.org/abs/1705.05363

-Benchmarks and Environments

StarCraft II: A New Challenge for Reinforcement Learning

https://arxiv.org/pdf/1708.04782.pdf

Other Active Areas

-Multi-Agent RL

Multi-agent Reinforcement Learning in Sequential Social Dilemmas

https://storage.googleapis.com/deepmind-media/papers/multi-agent-rl-in-ssd.pdf

-Memory and Attention

Control of Memory, Active Perception, and Action in Minecraft

https://arxiv.org/abs/1605.09128

-Transfer Learning / K-Shot Learning

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

https://arxiv.org/abs/1703.03400

-Competitive Self-Play

Emergent Complexity via Multi-Agent Competition

https://arxiv.org/abs/1710.03748

-And a lot more!

Take-Aways

-Deep RL isn't that complicated!

-There are tons of papers!