Reinforcement Learning in Modernity

Part I. Function approximation and Deep Q-networks

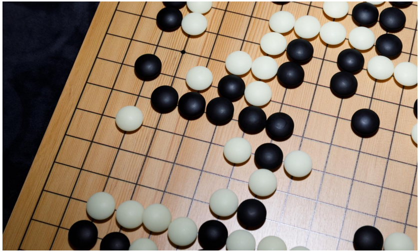
John D. Martin[α]

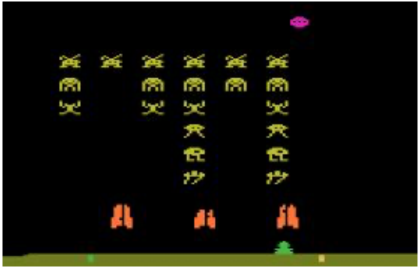
December 24, 2021

[α]University of Alberta
jmartin8@ualberta.ca

A Preview of What's to Come

- RL day consists of four lectures
  - ✓ 1. A First Glimpse at Reinforcement Learning
  - ✓ 2. Essentials of RL
  - ☞ 3. RL in Modernity
  - 4. Applications of RL
- This lecture will cover
  - The function approximation setting (John).
  - Case study: DQN (John).
  - Case study: PPO (Xutong Zhao).
  - Case study: AlphaGo (Xutong Zhao).
- The course is intended to prepare you for RL research.

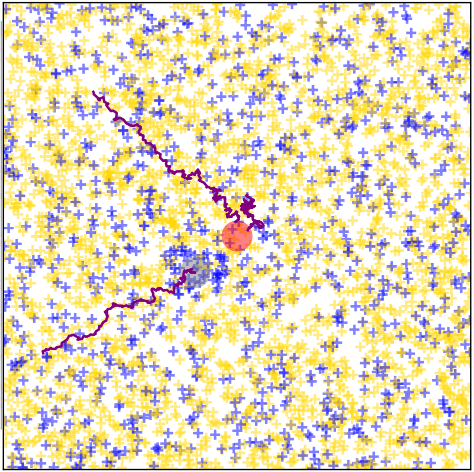
 

Interesting domains often involve a large number of observations, states, or actions.

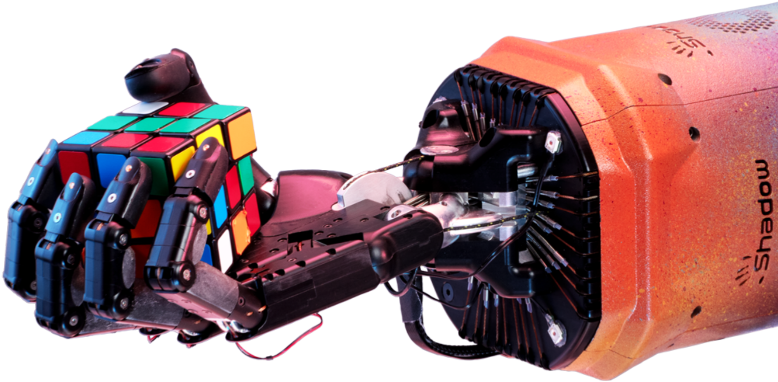
Stratospheric balloons can experience an enormous number of environment states.

Computer Go has 10^{170} states.

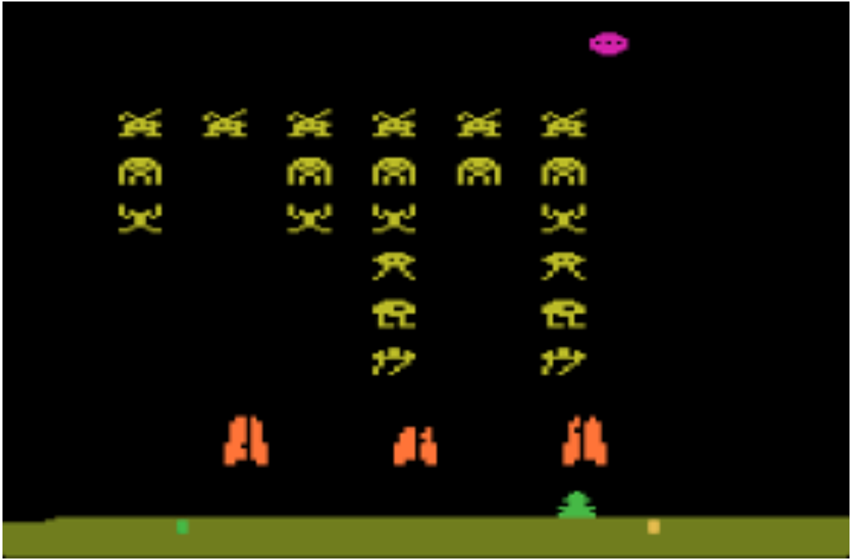


Martin and Modayil (2021).

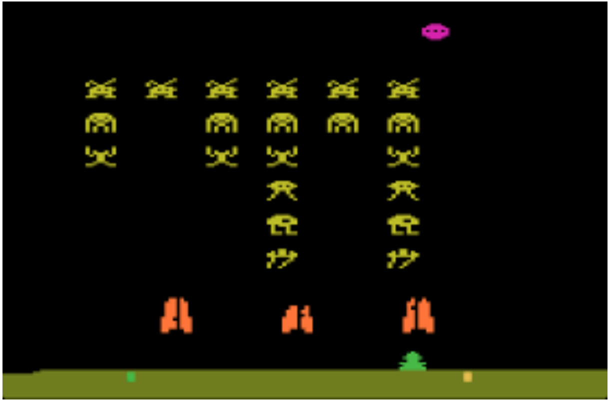
A simulated frog's eye contains $2^{4000} \approx 10^{1204}$ observations.



Manipulating a Rubik's cube involves precision control, with many actions.

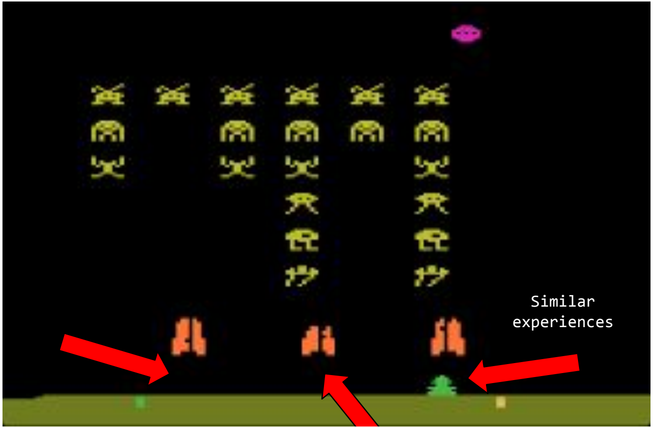


Playing Atari involves high-dimensional observations.



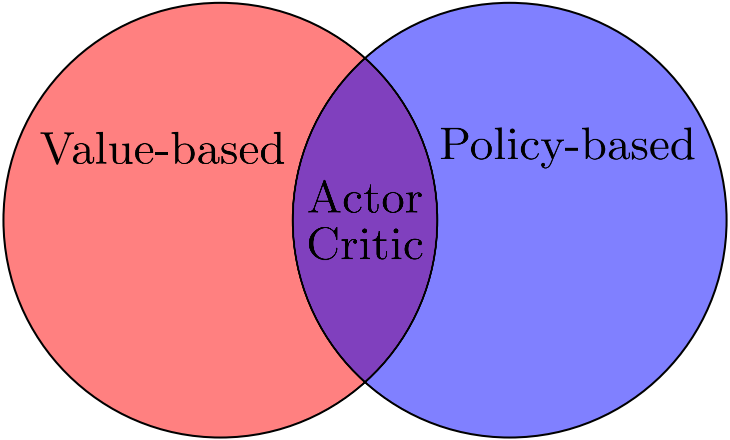
The problem with tabular RL

- Tables contain a value for every state or state-action pair.
- In large-scale domains, this representation becomes intractable.
- The learner should be able to *generalize* between experiences.



The problem with tabular RL

- Tables contain a value for every state or state-action pair.
- In large-scale domains, this representation becomes intractable.
- The learner should be able to *generalize* between experiences.



Where does function approximation apply in RL?

- Approximate value function $\hat{v}_\pi(\mathbf{x}; \mathbf{w}) \approx v_\pi(s)$.
- Policy: $\pi(\mathbf{x}; \boldsymbol{\theta})$
- Part I focuses on value approximation.

Agent State

- *Agent state* $\mathbf{x}(s)$ is the learner's internal representation of environment state.
- Agent state encodes patterns of the observation stream.
- Useful agent states are often more complex than raw observations.

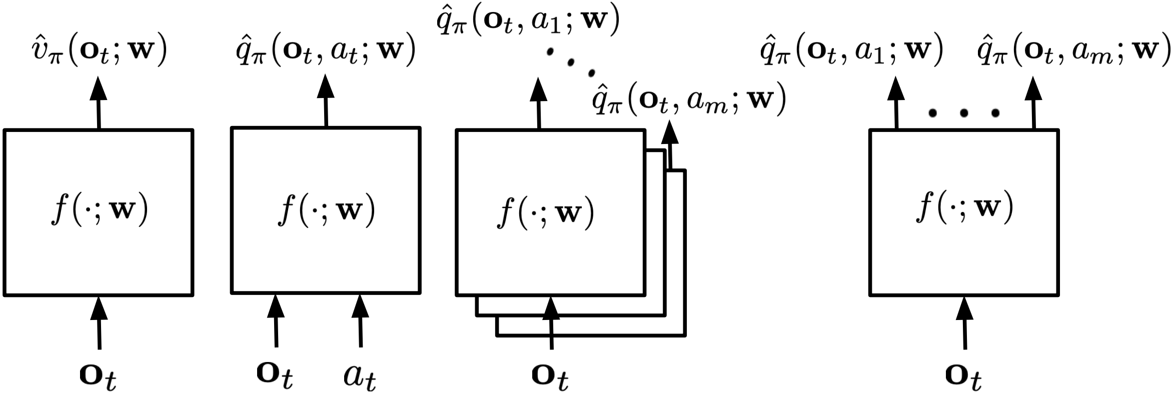
$$\hat{v}(\mathbf{x};\mathbf{w}) \triangleq \mathbf{w}^\top \mathbf{x}(s),$$
$$\hat{v}(\mathbf{x};\mathbf{w}) \approx v(s)$$

Agent state and feature vectors

- In a linear architecture, the agent state is a vector of *features*.
- The approximate value function is a weighted sum of features.
- There are many choices for features.

Examples of features

- Constant: $\mathbf{x}(s) = 1$
- Tabular: $\mathbf{x}(s) = (I(s = 1), I(s = 2), \cdots, I(s = d))^\top$
- Linear: $\mathbf{x}(s) = (O_1, O_2, \cdots, O_k)^\top$ for k-dimensional observations.
- Aggregation: Binary features that indicate occupancy of some support set.
- Fourier features: $\mathbf{x}(s) = \sum_{i=1}^{n} e^{\frac{2\pi j}{T}\mathbf{o}i}$.
- Neural network: $\mathbf{x}(s) = h_\ell \circ h_{\ell-1} \circ \cdots \circ h_1(\mathbf{o})$ for ℓ hidden layers.



Types of approximate value function architectures

- Monolithic architecture: one parametric function for $\hat{v}$ or $\hat{q}$.
- Stacked architecture: one parametric function for each action.
- Hybrid architecture: monolithic features and stacked final layers.

Case Study: Deep Q-Network

LETTER

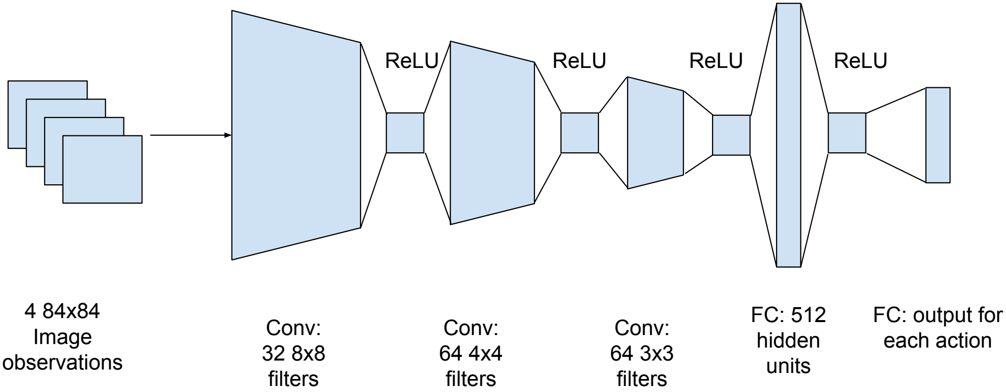
doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

Deep Q-Network (DQN)

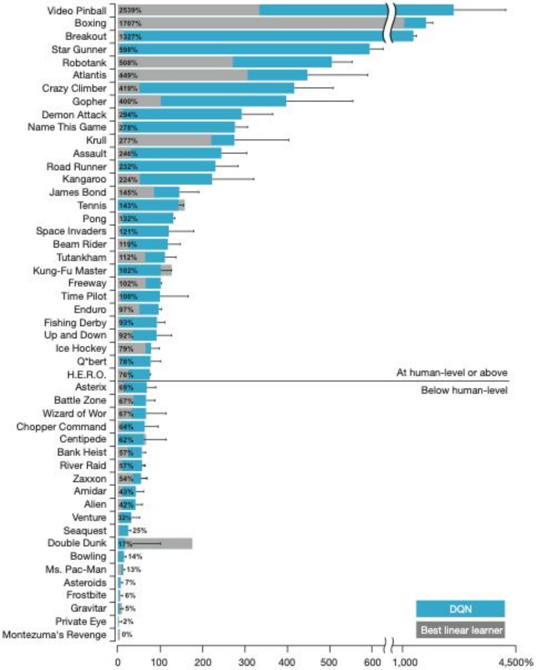
- Learning system observes a stream of images.
- Objective: learn a good approximate action-value function for control.
- Architecture uses a stacked representation of shared convolutional layers.


ReLU
ReLU
ReLU
ReLU
4 84x84
Image
observations
Conv:
32 8x8
filters
Conv:
64 4x4
filters
Conv:
64 3x3
filters
FC: 512
hidden
units
FC: output for
each action

```python
class NatureDQNNetwork(nn.Module):
  """The convolutional network used to compute the agent's Q-values."""
  num_actions: int
  inputs_preprocessed: bool = False

  @nn.compact
  def __call__(self, x):
    initializer = nn.initializers.xavier_uniform()
    if not self.inputs_preprocessed:
      x = preprocess_atari_inputs(x)
    x = nn.Conv(features=32, kernel_size=(8, 8), strides=(4, 4),
                kernel_init=initializer)(x)
    x = nn.relu(x)
    x = nn.Conv(features=64, kernel_size=(4, 4), strides=(2, 2),
                kernel_init=initializer)(x)
    x = nn.relu(x)
    x = nn.Conv(features=64, kernel_size=(3, 3), strides=(1, 1),
                kernel_init=initializer)(x)
    x = nn.relu(x)
    x = x.reshape((-1))  # flatten
    x = nn.Dense(features=512, kernel_init=initializer)(x)
    x = nn.relu(x)
    q_values = nn.Dense(features=self.num_actions,
                        kernel_init=initializer)(x)
    return atari_lib.DQNNetworkType(q_values)
```

Code from Dopamine research framework (Castro et al., 2018)



Video Pinball
Boxing
Breakout
Star Gunner
Robotank
Atlantis
Crazy Climber
Gopher
Demon Attack
Name This Game
Krull
Assault
Road Runner
Kangaroo
James Bond
Tennis
Pong
Space Invaders
Beam Rider
Tutankham
Kung-Fu Master
Freeway
Time Pilot
Enduro
Fishing Derby
Up and Down
Ice Hockey
Q*bert
H.E.R.O.
Asterix
Battle Zone
Wizard of Wor
Chopper Command
Centipede
Bank Heist
River Raid
Zaxxon
Amidar
Alien
Venture
Seaquest
Double Dunk
Bowling
Ms. Pac-Man
Asteroids
Frostbite
Gravitar
Private Eye
Montezuma's Revenge
2539%
1707%
1327%
598%
508%
449%
419%
400%
294%
278%
277%
246%
232%
224%
145%
143%
132%
121%
110%
112%
102%
102%
100%
97%
93%
92%
79%
78%
76%
69%
67%
67%
64%
62%
57%
57%
54%
43%
42%
32%
25%
17%
14%
13%
7%
6%
5%
2%
0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

$$L(\mathbf{w}) = \mathbf{E}[(R + \gamma \max_{a' \in \mathcal{A}} q(S', a'; \boldsymbol{\tau}) - q(S, A; \mathbf{w}))^2],$$

$$S, A, S' \sim \mu(\cdot)$$

DQN Loss

- Layer weights are denoted by $\mathbf{w}$.
- *Target network* has weights $\boldsymbol{\tau}$.
- Approximate loss with an empirical expectation of i.i.d. experience.

$$L(\mathbf{w}) = \mathbf{E}[(R + \gamma \max_{a' \in \mathcal{A}} q(S', a'; \boldsymbol{\tau}) - q(S, A; \mathbf{w}))^2],$$

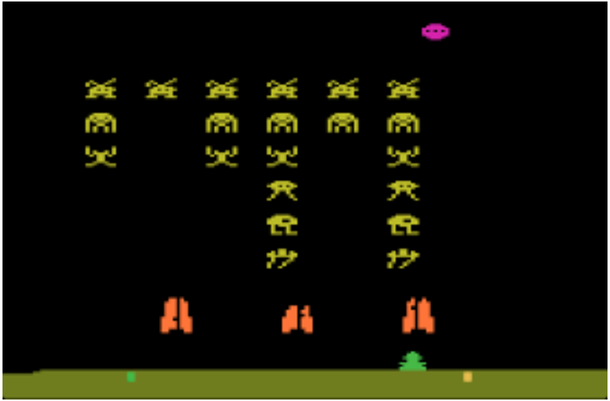
$$S, A, S' \sim \mu(\cdot)$$

$$\hat{L}(\mathbf{w}) \triangleq \frac{1}{n} \sum_{i=1}^{n} (r_i + \gamma \max_{a' \in \mathcal{A}} q(s'_i, a'; \boldsymbol{\tau}) - q(s_i, a_i; \mathbf{w}))^2],$$

$$s_i, r_i, s'_i \sim \mathcal{U}(\mathcal{D})$$

DQN Loss

- Layer weights are denoted by $\mathbf{w}$.
- *Target network* has weights $\boldsymbol{\tau}$.
- Approximate loss with an empirical expectation of i.i.d. experience.



Distinguishing features

- Draws i.i.d experience from a batch to remove serial correlation (Lin 1993).
- Computes targets from an equivalent network updated at a slower rate.
- Rewards are clipped to ± 1.

[DQN video link]