Intro to Deep Q Networks

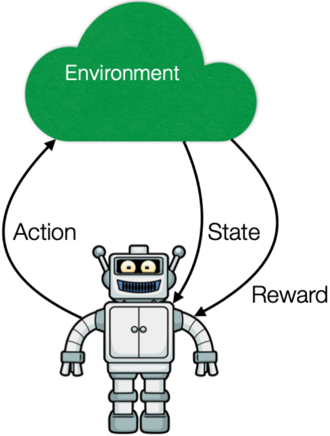
Jake Grigsby

University of Virginia

April 28, 2020

Agents and the Environment

Many problems in AI can be thought of as some autonomous agent interacting with an environment:



This concept is formalized as a **Markov Decision Process**...

Markov Decision Process

Definition

A Markov Decision Process **(MDP)** consists of:

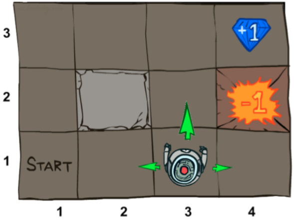
- $\mathcal{S}$, a set of states
- $\mathcal{A}$, a set of actions
- $\mathcal{R} \subseteq \mathbb{R}$, a set of rewards
- a dynamics function $p : \mathcal{S}x\mathcal{R}x\mathcal{S}x\mathcal{A} \to [0, 1]$

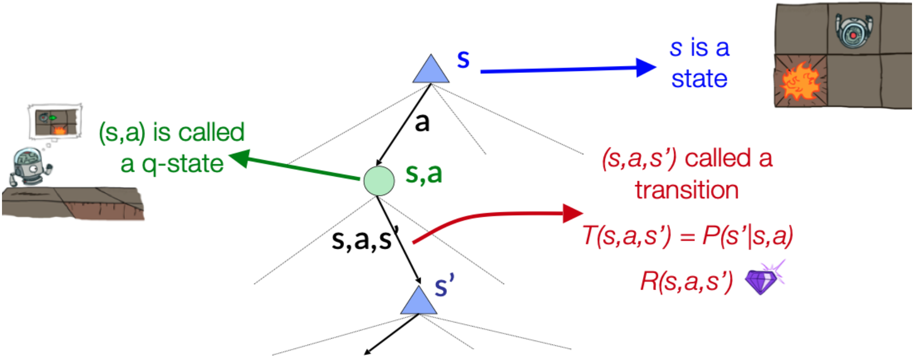
$$p(s', r|s, a) = Pr\{\mathcal{S}_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

It's common to break the dynamics function p up into a **Transition Function** $T(s, a, s') = \sum_{r \in \mathcal{R}} p(s', r|s, a)$, and a **Reward Function**

$$R(s, a) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$$

A Quick Example





The RL Problem

The goal of RL agents is to find a **policy**[1] $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the *expected discounted return*

$$\pi^* = \underset{\pi}{argmax} \mathop{\mathbb{E}}_{\tau \sim \pi} \left[\sum_{t=0}^{t=\infty} \gamma^t R_t \right]$$

where $\gamma \in [0, 1)$ is the *discount factor* that lets us deal with non-episodic tasks and τ is a *trajectory* (a sequence of states and actions that describe the agent's experience)

[1] Policies can also be stochastic, in which case they're written $\pi(a|s) : \mathcal{S} x \mathcal{A} \rightarrow [0, 1]$

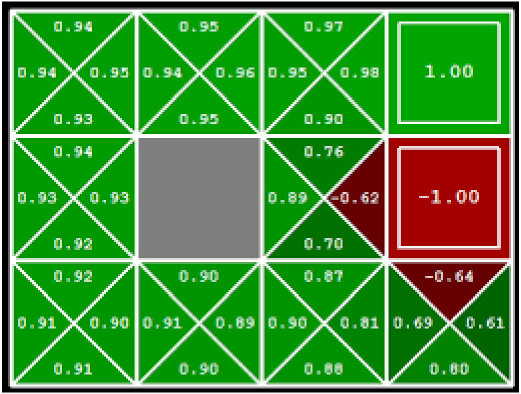
Simplifying Assumptions

We begin by making some assumptions about the task we are trying to solve:

1. The dynamics of the model $(p(s', r|s, a))$ are known
2. $|\mathcal{S}| \ll \infty$
3. $|\mathcal{A}| \ll \infty$

Generalized Policy Iteration

Solution: Policy Iteration Dynamic Programming



We'll skip these details because knowledge of dynamics is such a limiting assumption in our case. More info can be found in [4]

Simplifying Assumptions

1. ~~The dynamics of the model ($p(s', r|s, a)$) are known~~
2. $|\mathcal{S}| \ll \infty$
3. $|\mathcal{A}| \ll \infty$

What if the environment dynamics are unknown?

Value Methods

Definition

Value methods attempt to learn the optimal *Q Function*

$$Q^*(s,a) = \max_{\pi} \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{k=\infty} \gamma^k R_{t+k+1} \mid \mathcal{S}_t = s, \mathcal{A}_t = a\right]$$

Why? Because given $Q^*(s,a)$, the optimal policy can easily be computed by

$$\pi^*(s) = \underset{a}{argmax}\, Q^*(s,a)$$

Value Methods: Temporal Difference

- Randomly initialize Q(s, a) and use interactions with the environment as a sample to update this 'bootstrap'
- Updates based on the Bellman Equation:

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{s'}\left[r(s, a) + \gamma \mathop{\mathbb{E}}_{a' \sim \pi}\left[Q^{\pi}(s', a')\right]\right]$$

Value Methods: Temporal Difference

- Randomly initialize Q(s, a) and use interactions with the environment as a sample to update this 'bootstrap'
- Updates based on the Bellman Equation:

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{s'}\left[r(s,a) + \gamma \mathop{\mathbb{E}}_{a'\sim\pi}\left[Q^{\pi}(s',a')\right]\right]$$

- We minimize the *Bellman Error*:

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

Value Methods: Temporal Difference

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

```
Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        Q(S, A) ← Q(S, A) + α[R + γ max_a Q(S', a) − Q(S, A)]
        S ← S'
    until S is terminal
```

Figure: Q-Learning Pseudo-code [4]

A Quick Note on Exploration vs. Exploitation

- At each time step, the agent must choose between "exploiting" the action it currently thinks has the best return and "exploring" alternatives to learn more about them.
- Most convergence guarantees assume state *coverage*
  - Every state will be visited an infinite number of times in an infinite number of timesteps.
  - This can be acheived by enforcing:

$$\pi(a|s) > 0, \forall s \in \mathcal{S}$$

A Quick Note on Exploration vs. Exploitation

- The simplest way to do this is to make an existing policy ϵ-greedy:

$$\pi'(s) = \begin{cases} \pi(s) & \text{with probability } (1-\epsilon); \\ \pi_{random}(s) & \text{with probability } \epsilon; \end{cases}$$

A Quick Note on Exploration vs. Exploitation

- The simplest way to do this is to make an existing policy ϵ-greedy:

$$\pi'(s) = \begin{cases} \pi(s) & \text{with probability } (1-\epsilon); \\ \pi_{random}(s) & \text{with probability } \epsilon; \end{cases}$$

- This can be thought of as injecting noise into the action space
- All of the agent's we'll be talking about use this general approach, but there is a lot of interesting work on motivating agents to explore efficiently. [3] [5]

Simplifying Assumptions

1. ~~The dynamics of the model ($p(s', r|s, a)$) are known~~
2. $\cancel{|\mathcal{S}| \ll \infty}$
3. $|\mathcal{A}| \ll \infty$

What if the state space is too large for dynamic programming?

Tasks with Large State Spaces

Example: Video Games

- Pixel input makes $S = \mathbb{N}_{256}^{HxWxC}$
- Atari 2600 games make up one of the most popular benchmarks in modern RL.

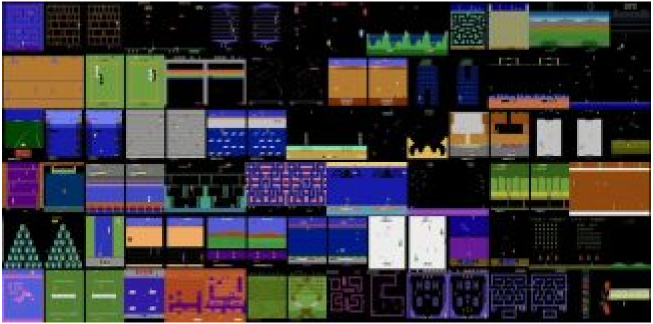
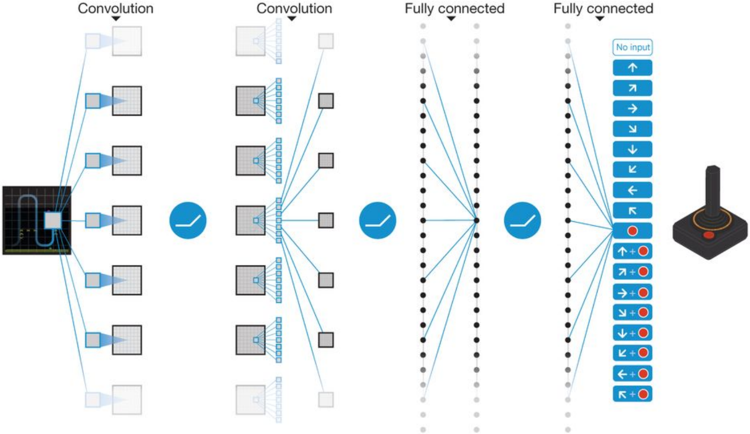


Figure: Games in the Arcade Learning Environment [1] benchmark

Deep Q Networks (DQN)

- Paramaterize Q with a neural network that can learn to recognize patterns between similar states.



Deep Q Networks (DQN)

- Train this network to minimize the *Mean Squared Bellman Error* (MSBE)

$$BE(s, a, r, s', d) = (r + \gamma(1 - d)\max_{a'} Q_{\theta'}(s', a')) - Q_{\theta}(s, a)$$

Deep Q Networks (DQN)

- Train this network to minimize the *Mean Squared Bellman Error* (MSBE)

$$BE(s, a, r, s', d) = (r + \gamma(1 - d) \max_{a'} Q_{\theta'}(s', a')) - Q_\theta(s, a)$$

- Kind of like supervised deep learning!
  - One important difference:
    - The data distribution depends on the parameters (far from i.i.d)

Deep Q Networks (DQN)

DQNs [2] use a couple tricks to make this more like supervised learning:

1. Create a *replay buffer* $\mathcal{R}$ to store transitions (s, a, r, s', d)
   - Randomly sample from this buffer at each training step.
2. Create a *target network* to generate the the bellman error targets.
   - This is a duplicate of the original network that is not trained but is updated with fresh params every ~ 10000 steps.

The original DQN was able to learn superhuman policies on many games with dense reward signals!

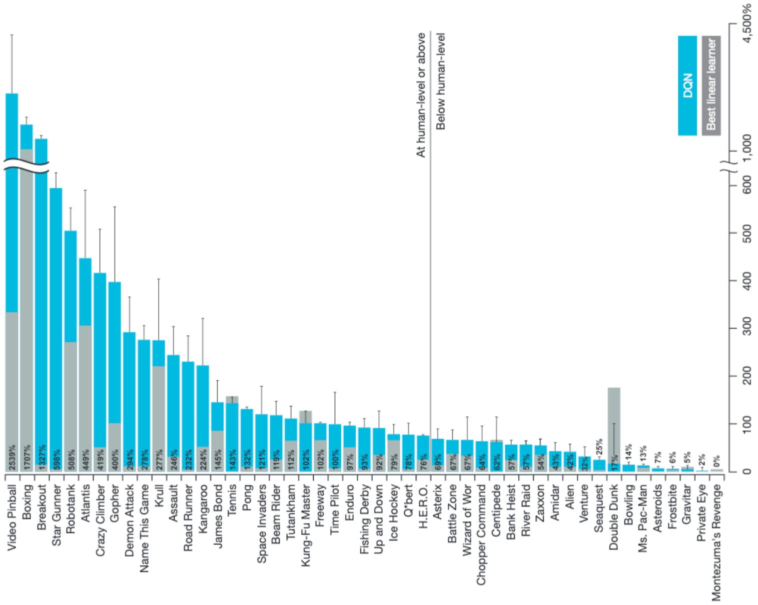
Deep Q Networks (DQN)

Algorithm 1: deep Q-learning with experience replay.

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**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
    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
    Every C steps reset $\hat{Q} = Q$
  End For
End For

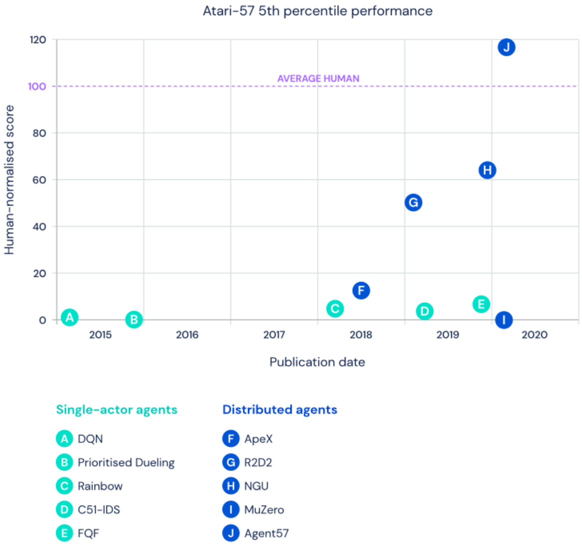
Figure: [2]

Results



DQN
Best linear learner
At human-level or above
Below human-level
4,500%
1,000
600
500
400
300
200
100
0
Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%

Since 2015...


Atari-57 5th percentile performance
Human-normalised score
120
100
80
60
40
20
0
AVERAGE HUMAN
A
B
C
D
E
F
G
H
I
J
2015
2016
2017
2018
2019
2020
Publication date
Single-actor agents
A DQN
B Prioritised Dueling
C Rainbow
D C51-IDS
E FQF
Distributed agents
F ApeX
G R2D2
H NGU
I MuZero
J Agent57

References I

Marc G Bellemare et al. “The arcade learning environment: An evaluation platform for general agents”. In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.

Yuri Burda et al. “Large-scale study of curiosity-driven learning”. In: *arXiv preprint arXiv:1808.04355* (2018).

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Adrià Puigdomènech Badia et al. “Never Give Up: Learning Directed Exploration Strategies”. In: *arXiv preprint arXiv:2002.06038* (2020).