Value Function Approximation

Weizi Li

Department of Computer Science
University of Memphis

Outline

- Introduction
- Incremental and Batch Methods
- Deep Q-Networks

Introduction

- So far, we use tabular representations, which allow exact solutions.
- Issue: large-scale RL problems (e.g., states, actions) cannot be fit into tables
  - Discrete state space: Game of Go has 10^{170} states (estimated atoms in the universe 10^{80})
  - Continuous state space

- Approach: use function approximation to achieve generalization
  - Assumption: nearby states should be similar (e.g., value functions)
- Goal: scale-up model-free methods for prediction and control

- Problem: difficult to store and compute value functions (V or Q) for large MDPs.
- Solution: parameterized function approximation

$$V^{\pi}(s) \approx \hat{V}(s, \mathbf{w})$$

or

$$Q^{\pi}(s, a) \approx \hat{Q}(s, a, \mathbf{w})$$

- Value function approximation options
  - Input: S; Output: $V(S)$
  - Input: S and A; Output: $Q(S, A)$
  - Input: S; Output: $Q(S, A_1), \ldots, Q(S, A_m)$
- Function approximation can also be used to represent a policy, e.g., policy gradient methods

- 1995: Gerald Tesauro solved Backgammon using TD + NN.
- 1996: “Neuro-Dynamic Programming” by Bertsekas and Tsitsiklis.
- 1996–1998: It’s proven that function approximator + off-policy control + bootstrapping can fail to converge
- 2015: DeepMind solved Atari, many breakthroughs since then

- Linear models (differentiable)
  - can work well if right features are given
- Neural networks (differentiable)
  - current mainstream
- Decision trees (non-differentiable)
- Deep RL: deep neural networks + RL

Incremental and Batch Methods

Supervised Learning with SGD

- Labeled data: $< s, V^{\pi}(s) >$, assume that an oracle provides $V^{\pi}(s)$
- Prediction: $\hat{V}(s, \mathbf{w})$
- Cost: $J(\mathbf{w}) = \mathbb{E}_{\pi}(V^{\pi}(s) - \hat{V}(s, \mathbf{w}))$
- SGD is usually adopted to update $\mathbf{w}$

- Linear model: $\hat{V}(s, \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w}$, where $\mathbf{x}(s) = (\mathbf{x}_1(s), \ldots, \mathbf{x}_n(s))^T$ represents features.
- Lookup table can be represented as a linear model

Using *table lookup features*

$$\mathbf{x}^{table}(S) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix}$$

Parameter vector $\mathbf{w}$ gives value of each individual state

$$\hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$

- We need labeled data $< s, V^{\pi}(s) >$, but we do not have an oracle to give us $V^{\pi}(s)$
- For MC: $V^{\pi}(s) \rightarrow G_t$ (return)
- For TD(0): $V^{\pi}(s) \rightarrow R_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w})$ (TD target)
- For TD(λ): $V^{\pi}(s) \rightarrow G_t^{\lambda}$ (λ-return)
- Similar replacements for $Q^{\pi}(s, a)$

- MC Control (after every episode)
  - policy evaluation: MC policy evaluation $Q \approx q_\pi$
  - policy improvement: ϵ-greedy
- TD Control (after every time step)
  - policy evaluation: Sarsa $Q \approx q_\pi$
  - policy improvement: ϵ-greedy
- Control with Function Approximator
  - policy evaluation: approximated policy evaluation $\hat{Q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$
  - policy improvement: ϵ-greedy

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

- "SBEED: Convergent Reinforcement Learning with Nonlinear Function Approximation" by Dai et al., 2018

- Incremental methods update $\mathbf{w}$ after every episode or time step, which it is not sample efficient, i.e., the episode or sample is discarded after the update
- Batch methods aim to find the best fitting value function by collecting $<$state, value$>$ pairs from the agent's experience
- The "value" can be return, TD target, λ-return, etc.

- Given value function approximation $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- And *experience* $\mathcal{D}$ consisting of $\langle$*state*, *value*$\rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Which parameters $\mathbf{w}$ give the *best fitting* value fn $\hat{v}(s, \mathbf{w})$?
- Least squares algorithms find parameter vector $\mathbf{w}$ minimising sum-squared error between $\hat{v}(s_t, \mathbf{w})$ and target values v_t^π,

$$\begin{aligned} LS(\mathbf{w}) &= \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2 \\ &= \mathbb{E}_\mathcal{D}\left[(v^\pi - \hat{v}(s, \mathbf{w}))^2\right] \end{aligned}$$

Given experience consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

Repeat:

1. Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

2. Apply stochastic gradient descent update

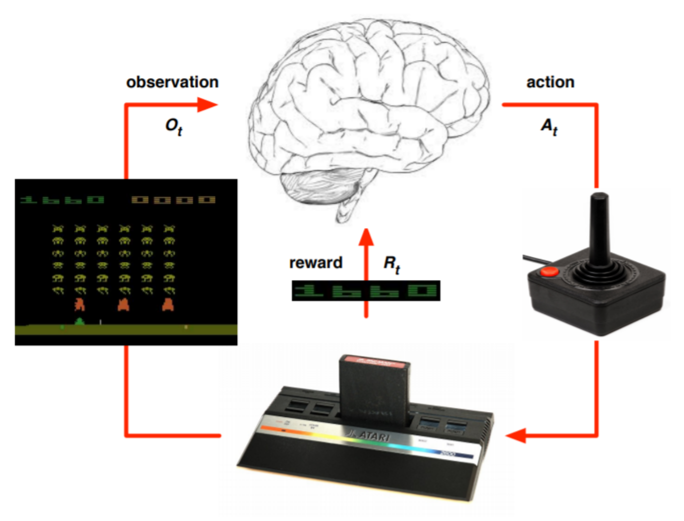
$$\Delta \mathbf{w} = \alpha (v^\pi - \hat{v}(s, \mathbf{w})) \nabla_\mathbf{w} \hat{v}(s, \mathbf{w})$$

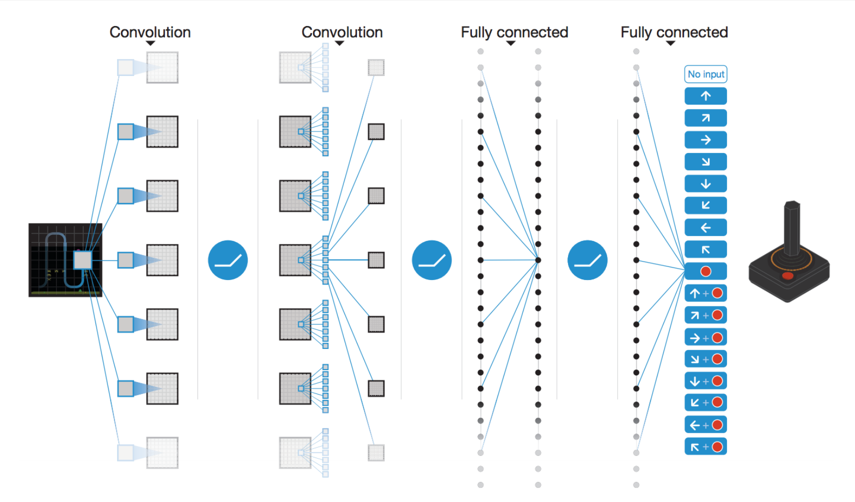
Converges to least squares solution

$$\mathbf{w}^\pi = \underset{\mathbf{w}}{\operatorname{argmin}}\ LS(\mathbf{w})$$

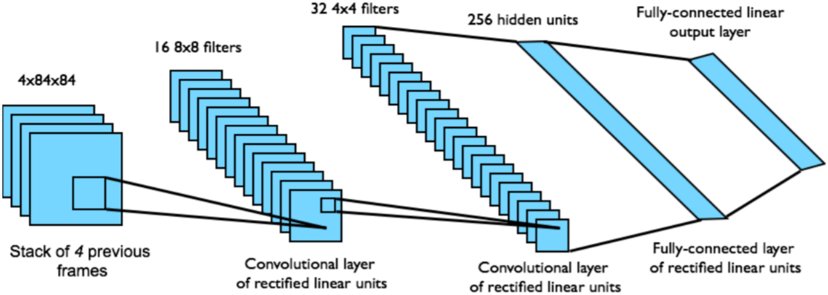
Deep Q-Networks

- “Human-level control through deep reinforcement learning” by Mnih et al, 2015


observation
O_t
action
A_t
reward
R_t
ATARI


Convolution
Convolution
Fully connected
Fully connected
No input

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

- Recall: function approximator + off-policy control + bootstrapping can fail to converge
- Two issues are potentially causing the problem:
  - correlations between samples (solution: concatenate multiple frames as input)
  - non-stationary targets

Recall Q-learning

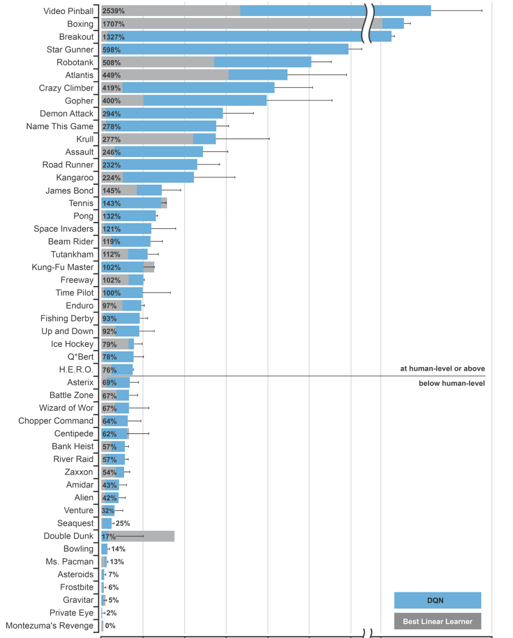
Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$;
    until S is terminal

- DQN uses *experience replay* (keep memory D) and *fixed Q-targets* (keep two sets of parameters w^- and w).
- Compared to Q-learning, we replace the "original $Q(S, A)$ update" and do the following
  - Store transitions (S, A, R, S') in D and sample mini-batch of transitions from D
  - Compute loss using the sampled mini-batch

$$L_i(w_i) = \mathbb{E}_{S,A,R,S' \sim D_i}\left[\left(R + \gamma max_{a'} Q(S', A'; w_i^-) - Q(S, A; w_i)\right)^2\right]$$

  - Using $L_i(w_i)$ to adjust w, and after certain steps $w^- \leftarrow w$

DQN on Atari Games



	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99

- Double DQN: "Deep Reinforcement Learning with Double Q-Learning" by Van Hasselt et al., AAAI 2016
- Prioritized Replay: "Prioritized Experience Replay" by Schaul et al., ICLR 2016
- Dueling DQN (best paper ICML 2016): "Dueling Network Architectures for Deep Reinforcement Learning" by Wang et al., ICML 2016

- DQN is more reliable on some Atari games than others. Pong is a reliable one: if doesn't achieve good scores, something is wrong
- Large replay buffers improve robustness of DQN
- DQN converges slowly: for Atari it's often necessary to wait for 10–40 M frames (couple of hours to a day of training on GPU) to outperform random policy
- Try DQN on small test environment first prior to Atari
- Try Double DQN: significant improvement from small code change
- Try large learning rates in initial exploration period

- DQN does not explore very well
- "Unifying Count-Based Exploration and Intrinsic Motivation" by Bellemare et al., 2016
- "Deep Q-learning from Demonstrations" by Hester et al., 2018. (Demo)
- "First return, then explore" by Ecoffet et al., 2021