Lecture 7: Imitation Learning in Large State Spaces

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2020

1With slides from Katerina Fragkiadaki and Pieter Abbeel
Experience replay in deep Q-learning (select all):

1. Involves using a bank of prior \((s,a,r,s')\) tuples and doing Q-learning updates using all the tuples in the bank
2. Always uses the most recent history of tuples
3. Reduces the data efficiency of DQN
4. Increases the computational cost
5. Not sure
Success in Atari has led to huge excitement in using deep neural networks to do value function approximation in RL

Some immediate improvements (many others!)

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
Last time: CNNs and Deep Reinforcement learning
This time: DRL and Imitation Learning in Large State Spaces
Next time: Policy Search
Double DQN

- Recall maximization bias challenge
  - Max of the estimated state-action values can be a biased estimate of the max
- Double Q-learning
Recall: Double Q-Learning

1: Initialize $Q_1(s, a)$ and $Q_2(s, a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
2: loop
3: Select $a_t$ using $\epsilon$-greedy $\pi(s) = \arg \max_a Q_1(s, a) + Q_2(s, a)$
4: Observe $(r_t, s_{t+1})$
5: if (with 0.5 probability) then
6: \[
Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha (r_t + Q_1(s_{t+1}, \arg \max_{a'} Q_2(s_{t+1}, a')) - Q_1(s_t, a_t))
\]
7: else
8: \[
Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha (r_t + Q_2(s_{t+1}, \arg \max_{a'} Q_1(s_{t+1}, a')) - Q_2(s_t, a_t))
\]
9: end if
10: $t = t + 1$
11: end loop
Success in Atari has led to huge excitement in using deep neural networks to do value function approximation in RL

Some immediate improvements (many others!)

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- **Prioritized Replay** (Prioritized Experience Replay, Schaul et al, ICLR 2016)
Check Your Understanding: Mars Rover Model-Free Policy Evaluation

\[ \pi(s) = a_1 \; \forall s, \; \gamma = 1. \] Any action from \( s_1 \) and \( s_7 \) terminates episode

Trajectory = \((s_3, a_1, 0, s_2, a_1, 0, s_2, a_1, 0, s_1, a_1, 1, \text{ terminal})\)

First visit MC estimate of \( V \) of each state? \([1 \; 1 \; 1 \; 0 \; 0 \; 0 \; 0]\)

TD estimate of all states (init at 0) with \( \alpha = 1 \) is \([1 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0]\)

Chose 2 "replay" backups to do. Which should we pick to get estimate closest to MC first visit estimate?

1. Doesn’t matter, any will yield the same
2. \((s_3, a_1, 0, s_2)\) then \((s_2, a_1, 0, s_1)\)
3. \((s_2, a_1, 0, s_1)\) then \((s_3, a_1, 0, s_2)\)
4. \((s_2, a_1, 0, s_1)\) then \((s_3, a_1, 0, s_2)\)
5. Not sure

<table>
<thead>
<tr>
<th></th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
<th>( s_4 )</th>
<th>( s_5 )</th>
<th>( s_6 )</th>
<th>( s_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R(s_1) = +1 )</td>
<td>( R(s_2) = 0 )</td>
<td>( R(s_3) = 0 )</td>
<td>( R(s_4) = 0 )</td>
<td>( R(s_5) = 0 )</td>
<td>( R(s_6) = 0 )</td>
<td>( R(s_7) = +10 )</td>
<td></td>
</tr>
</tbody>
</table>

*Okay Field Site*

*Fantastic Field Site*
Impact of Replay?

- In tabular TD-learning, **order** of replaying updates could help speed learning
- Repeating some updates seem to better propagate info than others
- Systematic ways to prioritize updates?
Potential Impact of Ordering Episodic Replay Updates

Oracle: picks \((s, a, r, s')\) tuple to replay that will minimize global loss

Exponential improvement in convergence
  - Number of updates needed to converge

Oracle is not a practical method but illustrates impact of ordering

**Figure:** Schaul, Quan, Antonoglou, Silver ICLR 2016
Prioritized Experience Replay

- Let $i$ be the index of the $i$-the tuple of experience $(s_i, a_i, r_i, s_{i+1})$
- Sample tuples for update using priority function
- Priority of a tuple $i$ is proportional to DQN error

$$p_i = \left| r + \gamma \max_{a'} Q(s_{i+1}, a'; \mathbf{w}^-) - Q(s_i, a_i; \mathbf{w}) \right|$$

- Update $p_i$ every update. $p_i$ for new tuples is set to 0
- One method\(^1\): proportional (stochastic prioritization)

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

\(^1\)See paper for details and an alternative
Exercise: Prioritized Replay

- Let $i$ be the index of the $i$-th tuple of experience $(s_i, a_i, r_i, s_{i+1})$
- Sample tuples for update using priority function
- Priority of a tuple $i$ is proportional to DQN error

$$p_i = \left| r + \gamma \max_{a'} Q(s_{i+1}, a'; w^-) - Q(s_i, a_i; w) \right|$$

- Update $p_i$ every update. $p_i$ for new tuples is set to 0
- One method\(^1\): proportional (stochastic prioritization)

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

- $\alpha = 0$ yields what rule for selecting among existing tuples?
  - Selects randomly
  - Selects the one with the highest priority
  - It depends on the priorities of the tuples
  - Not Sure
Performance of Prioritized Replay vs Double DQN

Figure: Schaul, Quan, Antonoglou, Silver ICLR 2016
Success in Atari has led to huge excitement in using deep neural networks to do value function approximation in RL

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
Intuition: Features need to accurately represent value may be different than those needed to specify difference in actions.

E.g.
- Game score may help accurately predict $V(s)$
- But not necessarily in indicating relative action values $Q(s, a_1)$ vs $Q(s, a_2)$

Advantage function (Baird 1993)

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$
Dueling DQN

DQN

Q(s,a1)
Q(s,a2)
...

Dueling DQN

V(s)

Q(s,a1)
Q(s,a2)
...

A(s,a1)
A(s,a2)
...

Wang et.al., ICML, 2016
Check Understanding: Unique?

- Advantage function

\[ A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \]

- For a given advantage function, is there a unique \( Q \) and \( V \)?
  1. Yes
  2. No
  3. Not sure
Uniqueness

- Advantage function

\[ A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \]

- Not unique

- Option 1: Force \( A(s, a) = 0 \) if \( a \) is action taken

\[ \hat{Q}(s, a; \mathbf{w}) = \hat{V}(s; \mathbf{w}) + \left( \hat{A}(s, a; \mathbf{w}) - \max_{a' \in \mathcal{A}} \hat{A}(s, a'; \mathbf{w}) \right) \]

- Option 2: Use mean as baseline (more stable)

\[ \hat{Q}(s, a; \mathbf{w}) = \hat{V}(s; \mathbf{w}) + \left( \hat{A}(s, a; \mathbf{w}) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} \hat{A}(s, a'; \mathbf{w}) \right) \]
Dueling DQN V.S. Double DQN with Prioritized Replay

Figure: Wang et al, ICML 2016
DQN is more reliable on some Atari tasks than others. Pong is a reliable task: if it doesn’t achieve good scores, something is wrong.

- Large replay buffers improve robustness of DQN, and memory efficiency is key.
  - Use uint8 images, don’t duplicate data.
- Be patient. DQN converges slowly—for ATARI it’s often necessary to wait for 10-40M frames (couple of hours to a day of training on GPU) to see results significantly better than random policy.
- In our Stanford class: Debug implementation on small test environment.
Try Huber loss on Bellman error

\[ L(x) = \begin{cases} 
\frac{x^2}{2} & \text{if } |x| \leq \delta \\
\delta |x| - \frac{\delta^2}{2} & \text{otherwise}
\end{cases} \]
Practical Tips for DQN on Atari (from J. Schulman) cont.

- Try Huber loss on Bellman error
  \[ L(x) = \begin{cases} 
  \frac{x^2}{2} & \text{if } |x| \leq \delta \\
  \delta |x| - \frac{\delta^2}{2} & \text{otherwise}
  \end{cases} \]

- Consider trying Double DQN—significant improvement from small code change in Tensorflow.
- To test out your data pre-processing, try your own skills at navigating the environment based on processed frames
- Always run at least two different seeds when experimenting
- Learning rate scheduling is beneficial. Try high learning rates in initial exploration period
- Try non-standard exploration schedules
Recap: Deep Model-free RL, 3 of the Big Ideas

- Double DQN: (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
DNN are very expressive function approximators
Can use to represent the Q function and do MC or TD style methods
Should be able to implement DQN (assignment 2)
Be able to list a few extensions that help performance beyond DQN
We want RL Algorithms that Perform

- Optimization
- Delayed consequences
- Exploration
- Generalization
- And do it all statistically and computationally efficiently
We will discuss efficient exploration in more depth later in the class.

But exist hardness results that, if learning in a generic MDP, can require large number of samples to learn a good policy.

Alternate idea: use structure and additional knowledge to help constrain and speed reinforcement learning.

Today: Imitation learning

Later:
- Policy search (can encode domain knowledge in the form of the policy class used)
- Strategic exploration
- Incorporating human help (in the form of teaching, reward specification, action specification, . . . )
Class Structure

- Last time: CNNs and Deep Reinforcement learning
- **This time:** Imitation Learning with Large State Spaces
- Next time: Policy Search
Reinforcement Learning: Learning policies guided by (often sparse) rewards (e.g. win the game or not)

- Good: simple, cheap form of supervision
- Bad: High sample complexity

Where is it most successful?

- In simulation where data is cheap and parallelization is easy
- Harder when:
  - Execution of actions is slow
  - Very expensive or not tolerable to fail
  - Want to be safe
Reward Shaping

Rewards that are **dense in time** closely guide the agent. How can we supply these rewards?

- **Manually design them**: often brittle
- **Implicitly specify them through demonstrations**

---

Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain, Silver et al. 2010
Examples

- Simulated highway driving [Abbeel and Ng, ICML 2004; Syed and Schapire, NIPS 2007; Majumdar et al., RSS 2017]
- Parking lot navigation [Abbeel, Dolgov, Ng, and Thrun, IROS 2008]
Learning from Demonstrations

- Expert provides a set of **demonstration trajectories**: sequences of states and actions.

- Imitation learning is useful when it is easier for the expert to demonstrate the desired behavior rather than:
  - Specifying a reward that would generate such behavior,
  - Specifying the desired policy directly.
Problem Setup

- **Input:**
  - State space, action space
  - Transition model \( P(s' \mid s, a) \)
  - No reward function \( R \)
  - Set of one or more teacher’s demonstrations \((s_0, a_0, s_1, s_0, \ldots)\)
    (actions drawn from teacher’s policy \( \pi^* \))

- **Behavioral Cloning:**
  - Can we directly learn the teacher’s policy using supervised learning?

- **Inverse RL:**
  - Can we recover \( R \)?

- **Apprenticeship learning via Inverse RL:**
  - Can we use \( R \) to generate a good policy?
Table of Contents

1 Behavioral Cloning

2 Inverse Reinforcement Learning
Behavioral Cloning

- Formulate problem as a standard machine learning problem:
  - Fix a policy class (e.g. neural network, decision tree, etc.)
  - Estimate a policy from training examples \((s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots\)
- Two notable success stories:
  - Pomerleau, NIPS 1989: ALVINN
  - Summut et al., ICML 1992: Learning to fly in flight simulator
Supervised learning assumes iid. \((s, a)\) pairs and ignores temporal structure. Independent in time errors:

Error at time \(t\) with probability \(\epsilon\)

\[\mathbb{E}[\text{Total errors}] \leq \epsilon T\]
Problem: Compounding Errors

Data distribution mismatch!
In supervised learning, \((x, y) \sim D\) during train and test. In MDPs:

- Train: \(s_t \sim D_{\pi^*}\)
- Test: \(s_t \sim D_{\pi_\theta}\)

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, Ross et al. 2011
Problem: Compounding Errors

Error at time $t$ with probability $\epsilon$

$$\mathbb{E}[\text{Total errors}] \leq \epsilon (T + (T - 1) + (T - 2) \ldots + 1) \propto \epsilon T^2$$

---

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, Ross et al. 2011
DAGGER: Dataset Aggregation

Initialize $D \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\Pi$.
\begin{itemize}
  \item for $i = 1$ to $N$ do
    \begin{itemize}
      \item Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
      \item Sample $T$-step trajectories using $\pi_i$.
      \item Get dataset $D_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_i$ and actions given by expert.
      \item Aggregate datasets: $D \leftarrow D \cup D_i$.
      \item Train classifier $\hat{\pi}_{i+1}$ on $D$.
    \end{itemize}
  \end{itemize}
Return best $\hat{\pi}_i$ on validation.

- Idea: Get more labels of the expert action along the path taken by the policy computed by behavior cloning
- Obtains a stationary deterministic policy with good performance under its induced state distribution
- Key limitation?
Table of Contents

1  Behavioral Cloning

2  Inverse Reinforcement Learning
Feature Based Reward Function

- Given state space, action space, transition model $P(s' | s, a)$
- No reward function $R$
- Set of one or more teacher’s demonstrations $(s_0, a_0, s_1, s_0, ...)$ (actions drawn from teacher’s policy $\pi^*$)
- Goal: infer the reward function $R$
- Assume that the teacher’s policy is optimal. What can be inferred about $R$?
Check Your Understanding: Feature Based Reward Function

- Given state space, action space, transition model $P(s' \mid s, a)$
- No reward function $R$
- Set of one or more teacher’s demonstrations $(s_0, a_0, s_1, s_0, \ldots)$ (actions drawn from teacher’s policy $\pi^*$)
- Goal: infer the reward function $R$
- Assume that the teacher’s policy is optimal.

1. There is a single unique $R$ that makes teacher’s policy optimal
2. There are many possible $R$ that makes teacher’s policy optimal
3. It depends on the MDP
4. Not sure
Recall linear value function approximation

Similarly, here consider when reward is linear over features

\[ R(s) = \mathbf{w}^T \mathbf{x}(s) \text{ where } \mathbf{w} \in \mathbb{R}^n, \mathbf{x} : S \rightarrow \mathbb{R}^n \]

Goal: identify the weight vector \( \mathbf{w} \) given a set of demonstrations

The resulting value function for a policy \( \pi \) can be expressed as

\[
V^\pi = \mathbb{E}_{s \sim \pi} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \right]
\]
Recall linear value function approximation

Similarly, here consider when reward is linear over features

\[ R(s) = w^T x(s) \text{ where } w \in \mathbb{R}^n, x : S \rightarrow \mathbb{R}^n \]

Goal: identify the weight vector \( w \) given a set of demonstrations

The resulting value function for a policy \( \pi \) can be expressed as

\[
V^\pi = \mathbb{E}_{s \sim \pi} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi \right]
= \mathbb{E}_{s \sim \pi} \left[ \sum_{t=0}^{\infty} \gamma^t w^T x(s_t) \mid \pi \right]
= \mathbb{E}_{s \sim \pi} \left[ \sum_{t=0}^{\infty} \gamma^t x(s_t) \mid \pi \right]
= w^T \mu(\pi)
\]

where \( \mu(\pi)(s) \) is defined as the discounted weighted frequency of state features under policy \( \pi \).
Relating Frequencies to Optimality

- Assume $R(s) = w^T x(s)$ where $w \in \mathbb{R}^n$, $x : S \to \mathbb{R}^n$
- Goal: identify the weight vector $w$ given a set of demonstrations
- $V^\pi = \mathbb{E}_{s \sim \pi} \left[ \sum_{t=0}^{\infty} \gamma^t R^*(s_t) \mid \pi \right] = w^T \mu(\pi)$ where
  $\mu(\pi)(s) = \text{discounted weighted frequency of state } s \text{ under policy } \pi$.

$$V^* \geq V^\pi$$
Recall linear value function approximation.

Similarly, here consider when reward is linear over features

\[ R(s) = w^T x(s) \] where \( w \in \mathbb{R}^n, x : S \rightarrow \mathbb{R}^n \)

Goal: identify the weight vector \( w \) given a set of demonstrations

The resulting value function for a policy \( \pi \) can be expressed as

\[ V^\pi = w^T \mu(\pi) \]

\( \mu(\pi)(s) = \) discounted weighted frequency of state \( s \) under policy \( \pi \).

\[
\mathbb{E}_{s \sim \pi^*} \left[ \sum_{t=0}^{\infty} \gamma^t R^*(s_t) \mid \pi^* \right] = V^* \geq V^\pi = \mathbb{E}_{s \sim \pi} \left[ \sum_{t=0}^{\infty} \gamma^t R^*(s_t) \mid \pi \right] \quad \forall \pi
\]

Therefore if the expert’s demonstrations are from the optimal policy, to identify \( w \) it is sufficient to find \( w^* \) such that

\[ w^*^T \mu(\pi^*) \geq w^*^T \mu(\pi), \forall \pi \neq \pi^* \]
Feature Matching

- Want to find a reward function such that the expert policy outperforms other policies.
- For a policy $\pi$ to be guaranteed to perform as well as the expert policy $\pi^*$, sufficient if its discounted summed feature expectations match the expert’s policy [Abbeel & Ng, 2004].
- More precisely, if
  \[ \| \mu(\pi) - \mu(\pi^*) \|_1 \leq \epsilon \]
  then for all $w$ with $\| w \|_\infty \leq 1$:
  \[ | w^T \mu(\pi) - w^T \mu(\pi^*) | \leq \epsilon \]
There is an infinite number of reward functions with the same optimal policy.

There are infinitely many stochastic policies that can match feature counts.

Which one should be chosen?
Many different approaches

Two of the key papers are:
  - Maximum Entropy Inverse Reinforcement Learning (Ziebart et al. AAAI 2008)
  - Generative adversarial imitation learning (Ho and Ermon, NeurIPS 2016)
Imitation learning can greatly reduce the amount of data need to learn a good policy.

Challenges remain and one exciting area is combining inverse RL / learning from demonstration and online reinforcement learning.

For a look into some of the theory between imitation learning and RL, see Sun, Venkatraman, Gordon, Boots, Bagnell (ICML 2017).
Class Structure

- Last time: CNNs and Deep Reinforcement learning
- This time: DRL and Imitation Learning in Large State Spaces
- Next time: Policy Search