CS 188: Artificial Intelligence

Special Topics: NLP/CV/RL

Instructor: Nicholas Tomlin

[Slides courtesy of Dan Klein, Abigail See, Greg Durrett, Yejin Choi, John DeNero, Eric Wallace, Kevin Lin, Fei-Fei Li, Sergey Levine, Pieter Abbeel, and many others]
What tasks do we care about?
Imitation Learning

$\mathbf{o}_t$  $\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$  $\mathbf{a}_t$

$\mathbf{o}_t$  $\mathbf{a}_t$  training data  supervised learning  $\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$
ALVINN: Autonomous Land Vehicle In a Neural Network
Distributional Drift

- Training trajectory
- $\pi_\theta$ expected trajectory
Modern Approach to Autonomous Driving
Avoiding Compounding Errors (Stability)
Avoiding Distributional Drift

\[ \pi_\theta \left( a_t | o_t \right) \]

\[ p_{\pi_\theta}(o_t) \]

\[ p_{\text{data}}(o_t) \]

can we make \( p_{\text{data}}(o_t) = p_{\pi_\theta}(o_t) \)?
can we make $p_{data}(o_t) = p_{\pi_\theta}(o_t)$?

idea: instead of being clever about $p_{\pi_\theta}(o_t)$, be clever about $p_{data}(o_t)$!

**DAgger: Dataset Aggregation**

goal: collect training data from $p_{\pi_\theta}(o_t)$ instead of $p_{data}(o_t)$

how? just run $\pi_\theta(a_t|o_t)$

but need labels $a_t$!

1. train $\pi_\theta(a_t|o_t)$ from human data $D = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ to get dataset $D_\pi = \{o_1, \ldots, o_M\}$
3. Ask human to label $D_\pi$ with actions $a_t$
4. Aggregate: $D \leftarrow D \cup D_\pi$
Reinforcement Learning

- generate samples (i.e. run the policy)
- improve the policy
- fit a model/estimate the return
<table>
<thead>
<tr>
<th>Known MDP: Offline Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
</tr>
<tr>
<td>Compute $V^<em>$, $Q^</em>$, $\pi^*$</td>
</tr>
<tr>
<td>Evaluate a fixed policy $\pi$</td>
</tr>
<tr>
<td><strong>Technique</strong></td>
</tr>
<tr>
<td>Value / policy iteration</td>
</tr>
<tr>
<td>Policy evaluation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unknown MDP: Model-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
</tr>
<tr>
<td>Compute $V^<em>$, $Q^</em>$, $\pi^*$</td>
</tr>
<tr>
<td>Evaluate a fixed policy $\pi$</td>
</tr>
<tr>
<td><strong>Technique</strong></td>
</tr>
<tr>
<td>VI/PI on approx. MDP</td>
</tr>
<tr>
<td>PE on approx. MDP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unknown MDP: Model-Free</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
</tr>
<tr>
<td>Compute $V^<em>$, $Q^</em>$, $\pi^*$</td>
</tr>
<tr>
<td>Evaluate a fixed policy $\pi$</td>
</tr>
<tr>
<td><strong>Technique</strong></td>
</tr>
<tr>
<td>Q-learning</td>
</tr>
<tr>
<td>Value Learning</td>
</tr>
</tbody>
</table>
Map of Reinforcement Learning

\[
\theta^* = \arg \max_{\theta} E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right]
\]

- Policy gradient: directly differentiate the above equation
- Value-based: estimate value function or Q-function of the optimal policy directly (but no explicit policy)
- Actor-critic: estimate value function or Q-function of the current policy, and use it to improve the policy
- Model-based RL: estimate the transition model, and then:
  - Use it for planning
  - Use it to improve a policy
Policy Gradient

- Generate samples (i.e. run the policy)
- Fit a model/estimate the return
- Evaluate returns
  \[ R_\tau = \sum_t r(s_t, a_t) \]
- Improve the policy
  \[ \theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(s_t, a_t)] \]
Value Function-Based Approaches

1. Generate samples (i.e., run the policy)
2. Fit a model or estimate the return
3. Improve the policy
4. Set $\pi(s) = \text{arg max}_a Q(s, a)$

$V(s)$ or $Q(s, a)$
Actor-Critic: Value Functions + Policy Gradients

- generate samples (i.e. run the policy)
- fit a model / estimate the return
- improve the policy
- fit $V(s)$ or $Q(s, a)$

\[
\theta \leftarrow \theta + \alpha \nabla_{\theta} E[Q(s_t, a_t)]
\]
Model-Based Reinforcement Learning

- Generate samples (i.e. run the policy)
- Fit a model / estimate the return
- Learn $p(s_{t+1}|s_t, a_t)$
- Improve the policy

(a few options)
Map of Reinforcement Learning

RL Algorithms

Model-Free RL
- Policy Optimization
  - Policy Gradient
  - A2C / A3C
  - PPO
  - TRPO

- Q-Learning
  - DDPG
  - TD3
  - SAC

- Learn the Model
  - DQN
  - C51
  - QR-DQN
  - HER

Model-Based RL
- Learn the Model
  - World Models
    - I2A
    - MBMF
    - MBVE
- Given the Model
  - AlphaZero
Why so many options?

- **Different tradeoffs:**
  - Sample efficiency
  - Stability and ease of use

- **Different assumptions:**
  - Stochastic or deterministic?
  - Continuous or discrete?
  - Episodic or infinite horizon?

- **Different things are easy or hard in different settings:**
  - Easy to represent the policy?
  - Easy to represent the model?
Comparison: Efficiency

- Sample efficiency = how many samples we need to get a good policy
- Most important question: is the algorithm on-policy or off-policy?
  - On-policy: each time the policy is changed, need to generate new samples
  - Off-policy: able to improve the policy without generating new samples
Comparison: Stability and Ease of Use

- Value function fitting:
  - At best, minimizes error of fit ("Bellman error")
  - At worst, doesn’t optimize anything (often no guarantees with deep RL)

- Model-based RL:
  - Model minimizes error of fit (will converge)
  - No guarantee that better model = better policy

- Policy gradient:
  - The only approach that actually performs gradient descent on the true objective
  - In practice, often the least efficient!
Comparison: Assumptions

- **Common assumption #1: full observability**
  - Generally assumed by value function fitting methods
  - Can be mitigated by adding recurrence

- **Common assumption #2: episodic learning**
  - Often assumed by pure policy gradient methods
  - Assumed by some model-based RL methods

- **Common assumption #3: continuity or smoothness**
  - Assumed by some continuous value function learning methods
  - Often assumed by some model-based RL methods
Model-Free RL: Q-Learning
**Recall: Approximate Q-Learning**

\[
Q(s, a) = w_1f_1(s, a) + w_2f_2(s, a) + \ldots + w_nf_n(s, a)
\]

- **Q-learning with linear Q-functions:**
  
  \[
  \text{transition} = (s, a, r, s')
  \]
  
  \[
  \text{difference} = \left[ r + \gamma \max_{a'} Q(s', a') \right] - Q(s, a)
  \]
  
  \[
  Q(s, a) \leftarrow Q(s, a) + \alpha \text{[difference]}
  \]
  
  \[
  w_i \leftarrow w_i + \alpha \text{[difference]} f_i(s, a)
  \]

  **Intuitive interpretation:**
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state’s features

  **Can perform update based on a single sample, or with multiple**
Model-Free RL: DQN

Algorithm 1 Deep Q-learning with Experience Replay

1. Initialize replay memory $\mathcal{D}$ to capacity $N$
2. Initialize action-value function $Q$ with random weights

for episode = 1, $M$ do

1. Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

for $t = 1, T$ do

1. With probability $\epsilon$ select a random action $a_t$
2. Otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
3. Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
4. Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
5. Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
6. Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
7. Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
8. Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

Q-function is represented as a CNN
Model-Free RL: REINFORCE

REINFORCE algorithm:
1. sample \{\tau^i\} from \pi_\theta(a_t|s_t) (run the policy)
2. \nabla_\theta J(\theta) \approx \sum_i \left( \sum_t \nabla_\theta \log \pi_\theta(a^i_t|s^i_t) \right) \left( \sum_t r(s^i_t, a^i_t) \right)
3. \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)

- Inefficient: run the policy to get trajectories and then throw them away
- Gradient computations may be noisy (high variance)
- Practical considerations with batch sizes, learning rates, and optimizers
Model-Based RL: World Models
Model-Based RL: AlphaZero

- Learn both a policy and value network via self-play (reward of +1/-1 comes from end of game)
- Transition function is known: we can do explicit planning
- Use Monte Carlo tree search (MCTS) to choose actions based on the current value function
What does AlphaGo Zero Learn? (Tomlin, et al. 2022)

<table>
<thead>
<tr>
<th>Concept</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eye</td>
<td>Surrounded empty space</td>
</tr>
<tr>
<td>Wall</td>
<td>Sequence of stones in a row</td>
</tr>
<tr>
<td>Ladder</td>
<td>Zig-zag capturing race</td>
</tr>
<tr>
<td>Pincer</td>
<td>Attack on a corner approach</td>
</tr>
<tr>
<td>Joseki</td>
<td>Fixed local sequence of moves</td>
</tr>
<tr>
<td>Sente</td>
<td>Initiative</td>
</tr>
<tr>
<td>Hane</td>
<td>Move that “reaches around”</td>
</tr>
<tr>
<td>Up</td>
<td>Toward the center of the board</td>
</tr>
<tr>
<td>Aji</td>
<td>Possibilities left in a position</td>
</tr>
<tr>
<td>Ko</td>
<td>Repeated capture sequence</td>
</tr>
<tr>
<td>Shape</td>
<td>Quality of stone arrangement</td>
</tr>
<tr>
<td>Gote</td>
<td>Loss of initiative</td>
</tr>
<tr>
<td>Moyo</td>
<td>Sphere of influence</td>
</tr>
<tr>
<td>Aji</td>
<td>Possibilities left in a position</td>
</tr>
<tr>
<td>Atari</td>
<td>Threat to capture</td>
</tr>
</tbody>
</table>
Extracting Concepts from Game States

Bad shape. If white wants to defend it should be solid at c8, leaving no weaknesses or sente moves for black.

Dataset Statistics:
- 10K annotated games (19x19)
- Approximately 458K comments
- Additional data from unplayed variations (ignored in this work)
Game-Playing Agents

- **Agent #1: Imitation Learning**
  - Following CNN architecture from Clark and Storkey (2014)
  - Trained on 228K human games
  - Played against real humans on Online Go Server (OGS) and received a rating of 1K

- **Agent #2: Reinforcement Learning**
  - Pre-trained ELF OpenGo [Tian, et al. 2019]
  - Open-source equivalent of AlphaGo Zero [Silver, et al. 2017]
  - Better than all human players: ELO 5000+
Key finding: human-level concepts are predictable from the intermediate representations of both models.

Additionally: some concepts appear in early layers, and others in later layers (i.e., different levels of abstraction).

Long-term goal: how do we get human-interpretable explanations of models which exceed human capacity?