DAGGER and Friends

References:

1. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, Ross, Gordon & Bagnell (2010). DAGGER algorithm
4. SEARN in Practice Daume et al. (2006)

John Schulman
2015/10/5
CHAPTER 1. INTRODUCTION

Expert trajectory

Learned Policy

No data on how to recover

Data Mismatch Problem

Figure 1.1: Mismatch between the distribution of training and test inputs in a driving scenario.

Many state-of-the-art software systems that we use everyday. Systems based on supervised learning already translate our documents, recommend what we should read (Yue and Guestrin, 2011), watch (Toscher et al., 2009) or buy, read our handwriting (Daumé III et al., 2009) and filter spam from our emails (Weinberger et al., 2009), just to name a few. Many subfields of artificial intelligence, such as natural language processing (the understanding of natural language by computers) and computer vision (the understanding of visual input by computers), now deeply integrate machine learning.

Despite this widespread proliferation and success of machine learning in various fields and applications, machine learning has had a much more limited success when applied in control applications, e.g. learning to drive from demonstrations by human drivers. One of the main reason behind this limited success is that control problems exhibit fundamentally different issues that are not typically addressed by standard supervised learning techniques.

In particular, much of the theory and algorithms for supervised learning are based on the fundamental assumption that inputs/observations perceived by the predictor to make its predictions are independent and always coming from the same underlying distribution (Hastie et al., 2001). This ensures that after seeing enough training examples, we will be able to predict well on new examples (at least in expectation). However, this assumption is clearly violated in control tasks as these are inherently dynamic and sequential: one must perform a sequence of actions over time that have consequences on future inputs or observations of the system, to achieve a goal or successfully perform the task. As predicting actions to execute influence future inputs, this can lead to a large mismatch between the inputs observed under training demonstrations, and those observed during test executions of the learned behavior. This is illustrated schematically in Figure 1.1.

This problem has been observed in previous work. Pomerleau (1989), who trained a...
Data Mismatch Problem

Train:
- \((x, y) \sim D\)
- \(s \sim d_{\pi^*}\)

Test:
- \((x, y) \sim D\)
- \(s \sim d_{\pi}\)

Supervised learning

Supervised learning + control (NAIVE)
Compounding Errors

\[ E[\text{Total errors}] \leq \varepsilon(T + (T-1) + (T-2) + \ldots + 1) \propto \varepsilon T^2 \]
Forward Algorithm

Initialize $\pi_1, \pi_2, \ldots, \pi_T$ arbitrarily.

for $t = 1$ to $T$ do

Sample multiple $t$-step trajectories by executing the policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$, starting from initial states drawn from the initial state distribution.

Query expert for states encountered at time step $t$.

Get dataset $D = \{(s_t, \pi^*(s_t))\}$ of states, actions taken by expert at time step $t$.

Train classifier $\pi_t = \arg \min_{\pi \in \Pi} \sum_{(s,a) \in D} \ell(s, a, \pi)$.

end for

Return non-stationary policy $\hat{\pi}$, such that at time $t$ in state $s$, $\hat{\pi}(s, t) = \pi_t(s)$

$E[\text{total errors}] \leq \varepsilon T$
3.6. DATASET AGGREGATION: ITERATIVE INTERACTIVE LEARNING APPROACH

Figure 3.5: Depiction of the DAGGER procedure for imitation learning in a driving scenario.

Execute current policy and Query Expert

Steering from expert

New Policy

Aggregate Dataset

Supervised Learning

New Data

All previous data

Figure 3.6: Diagram of the DAGGER algorithm with a general online learner for imitation learning.

policies, with relatively few data points, may make many more mistakes and visit states that are irrelevant as the policy improves. We will typically use $\pi = 1$ so that we do not have to specify an initial policy $\hat{\pi}$ before getting data from the expert's behavior. Then we could choose $i = p i$ to have a probability of using the expert that decays exponentially as in SMILE and SEARN. The only requirement is that $\{i\}$ be a sequence such that $\sum_{N=1}^{\infty} i ! = 0$ as $N ! 1$. The simple, parameter-free version of the
**DAGGER**

Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\Pi$.

**for** $i = 1$ **to** $N$ **do**

- Let $\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_i$.
- Sample $T$-step trajectories using $\pi_i$.
- Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_i$ and actions given by expert.
- Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$.
- Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$ (or use online learner to get $\hat{\pi}_{i+1}$ given new data $\mathcal{D}_i$).

**end for**

**Return** best $\hat{\pi}_i$ on validation.
Initialize $D \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\Pi$.

for $i = 1$ to $N$ do

Let $\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_i$.
Sample $T$-step trajectories using $\pi_i$.
Get dataset $D_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_i$ and actions given by expert.
Aggregate datasets: $D \leftarrow D \cup D_i$.
Train classifier $\hat{\pi}_{i+1}$ on $D$ (or use online learner to get $\hat{\pi}_{i+1}$ given new data $D_i$).

end for

Return best $\hat{\pi}_i$ on validation.

A*(s,a) for all actions a

i.e., minimize $\sum_n A^*(s_n, \pi(s_n))$
Empirical Demonstrations
Online Learning + Regret

• Learn from a stream of data, might be non-stationary or adversarial

• At $n^{th}$ step, algorithm chooses $\pi_n$, receives loss $L_n(\pi_n)$

• Want to minimize $\sum_n L_n(\pi_n)$

• Regret: $\sum_n L_n(\pi_n) - \min_{\pi} \sum_n L_n(\pi)$

• e.g. for convex $L$ with online gradient descent, one can show that total regret $\sim \sqrt{T}$

AGGREGATE: Theory

\[ \eta(\pi) - \eta(\pi^*) = \mathbb{E}_\tau;\pi \left[ \sum_{t=1}^{T} A^\pi(s_t, a_t) \right] \]

\[ L_n(\pi) := \mathbb{E}_\tau;\pi_n \left[ \sum_{t=1}^{T} A^{\pi^*}(s_t, \pi(s_t)) \right] \]

Suboptimality of n\textsuperscript{th} policy: \[ L_n(\pi_n) \]
AGGREGATE (β=0)

- At $n^{th}$ step, sample trajectories using $\pi_n$
  - suboptimality is $L_n(\pi_n)$
- Update policy based on new data to get $\pi_{n+1}$
  - e.g., take $\pi_{n+1} = \arg\min_\pi \sum_n L_n(\pi)$
AGGREGVATE ($\beta=0$)

- Now, consider $\pi$, obtained by randomly sampling $n$ in $\{1,2,\ldots,N\}$

$$\eta(\bar{\pi}) - \eta(\pi^*) = \frac{1}{N} \sum_{n=1}^{N} L_n(\pi_n)$$

- $\Rightarrow$ Suboptimality is bounded by regret of learning algorithm
AGGREGVATE ($\beta=0$)

- Sample trajectories
Application to Atari

Deep Learning for Real-Time Atari Game Play
Using Offline Monte-Carlo Tree Search Planning

Xiaoxiao Guo
Computer Science and Eng.
University of Michigan
guoxiao@umich.edu

Satinder Singh
Computer Science and Eng.
University of Michigan
baveja@umich.edu

Honglak Lee
Computer Science and Eng.
University of Michigan
honglak@umich.edu

Richard Lewis
Department of Psychology
University of Michigan
rickl@umich.edu

Xiaoshi Wang
Computer Science and Eng.
University of Michigan
xiaoshiw@umich.edu

NIPS 2014
Monte Carlo Tree Search (UCT) + ConvNet Policy/Classifier


Application to Atari

Figure 2: Visualization of the first-layer features learned from Seaquest. (Left) visualization of four first-layer filters; each filter covers four frames, showing the spatio-temporal template. (Middle) a captured screen. (Right) gray-scale version of the input screen which is fed into the CNN. Four filters were color-coded and visualized as dotted bounding boxes at the locations where they get activated. This figure is best viewed in color.

Figure 3: Visualization of the second-layer features learned from Seaquest.

Learned Features from Convolutional Layers.

We provide visualizations of the learned filters in order to gain insight on what the CNN learns. Specifically, we apply the “optimal stimuli” method [7, 14] to visualize the features CNN learned after training. The method picks the input image patches that generate the greatest responses after convolution with the trained filters. We select 8 ⇴ 8 ⇴ 4 input patches to visualize the first convolutional layer features and 20 ⇴ 20 ⇴ 4 to visualize the second convolutional layer filters. Note that these patch sizes correspond to receptive field sizes of the learned features in each layer.

In Figure 2, we show four first-layer filters of the CNN trained from Seaquest for the UCTtoClassification agent. Specifically, each filter covers four frames of 8 ⇴ 8 pixels, which can be viewed as a spatio-temporal template that captures specific patterns and their temporal changes. We also show an example screen capture and visualize where the filters get activated in the gray-scale version of the image (which is the actual input to the CNN model). The visualization suggests that the first-layer filters capture “object-part” patterns and their temporal movements.

Figure 3 visualizes the four second-layer features via the optimal stimulus method, where each row corresponds to a filter. We can see that the second-layer features capture bigger spatial patterns (often covering beyond the size of individual objects), while encoding interactions between objects, such as two enemies moving together, and a submarine moving along a direction. Overall, these qualitative results suggest that the CNN learns relevant patterns useful for game playing.

Visualization of Learned Policy.

Here we present visualizations of the policy learned by the UCT-to-Classification agent with the aim of illustrating both what it does well and what it does not. Figure 4 shows the policy learned by UCT-to-Classification to destroy nearby enemies. The CNN changes the action from “Fire” to “Down+Fire” at time step 70 when the enemies first show up at the right columns of the screen, which will move the submarine to the same horizontal position of the closest enemy. At time step 75, the submarine is at the horizontal position of the closest enemy and...

cool finding — low level filters show game objects
Application to Atari

<table>
<thead>
<tr>
<th>Agent</th>
<th>B.Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q*bert</th>
<th>Seaquest</th>
<th>S.Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>4092</td>
<td>168</td>
<td>470</td>
<td>20</td>
<td>1952</td>
<td>1705</td>
<td>581</td>
</tr>
<tr>
<td>-best</td>
<td>5184</td>
<td>225</td>
<td>661</td>
<td>21</td>
<td>4500</td>
<td>1740</td>
<td>1075</td>
</tr>
<tr>
<td>UCC</td>
<td>5342 (20)</td>
<td>175 (5.63)</td>
<td>558 (14)</td>
<td>19 (0.3)</td>
<td>11574 (44)</td>
<td>2273 (23)</td>
<td>672 (5.3)</td>
</tr>
<tr>
<td>-best</td>
<td>10514</td>
<td>351</td>
<td>942</td>
<td>21</td>
<td>29725</td>
<td>5100</td>
<td>1200</td>
</tr>
<tr>
<td>-greedy</td>
<td>5676</td>
<td>269</td>
<td>692</td>
<td>21</td>
<td>19890</td>
<td>2760</td>
<td>680</td>
</tr>
<tr>
<td>UCC-I</td>
<td>5388 (4.6)</td>
<td>215 (6.69)</td>
<td>601 (11)</td>
<td>19 (0.14)</td>
<td>13189 (35.3)</td>
<td>2701 (6.09)</td>
<td>670 (4.24)</td>
</tr>
<tr>
<td>-best</td>
<td>10732</td>
<td>413</td>
<td>1026</td>
<td>21</td>
<td>29900</td>
<td>6100</td>
<td>910</td>
</tr>
<tr>
<td>-greedy</td>
<td>5702</td>
<td>380</td>
<td>741</td>
<td>21</td>
<td>20025</td>
<td>2995</td>
<td>692</td>
</tr>
<tr>
<td>UCR</td>
<td>2405 (12)</td>
<td>143 (6.7)</td>
<td>566 (10.2)</td>
<td>19 (0.3)</td>
<td>12755 (40.7)</td>
<td>1024 (13.8)</td>
<td>441 (8.1)</td>
</tr>
</tbody>
</table>

Table 2: Performance (game scores) of the off-line UCT game playing agent.

<table>
<thead>
<tr>
<th>Agent</th>
<th>B.Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q*bert</th>
<th>Seaquest</th>
<th>S.Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCT</td>
<td>7233</td>
<td>406</td>
<td>788</td>
<td>21</td>
<td>18850</td>
<td>3257</td>
<td>2354</td>
</tr>
</tbody>
</table>

but... 800 games * 1000 actions/game * 10000 rollouts/action * 300 steps/rollout = 2.4e12 steps