Offline Policy-search in Bayesian Reinforcement Learning

Castronovo Michael

University of Liège, Belgium
Advisor: Damien Ernst

15th March 2017
Contents

- Introduction
- Problem Statement
- Offline Prior-based Policy-search (OPPS)
- Artificial Neural Networks for BRL (ANN-BRL)
- Benchmarking for BRL
- Conclusion
Introduction

What is Reinforcement Learning (RL)?
A sequential decision-making process where an agent observes an environment, collects data and reacts appropriately.

Example: Train a Dog with Food Rewards

- Context: Markov-decision process (MDP)
- Single trajectory (= only 1 try)
- Discounted rewards (= early decisions are more important)
- Infinite horizon (= the number of decisions is infinite)
The Exploration / Exploitation dilemma (E/E dilemma)

An agent has two objectives:

- Increase its knowledge of the environment
- Maximise its short-term rewards

⇒ Find a compromise to avoid suboptimal long-term behaviour

In this work, we assume that

- The reward function is known
  (= agent knows if an action is good or bad)
- The transition function is unknown
  (= agent does not know how actions modify the environment)
Reasonable assumption:
Transition function is not unknown, but is instead uncertain:
  ⇒ We have some prior knowledge about it
  ⇒ This setting is called Bayesian Reinforcement Learning

What is Bayesian Reinforcement Learning (BRL)?
A Reinforcement Learning problem where we assume some prior knowledge is available on start in the form of a MDP distribution.
Intuitively...

A process that allows to simulate decision-making problems similar to the one we expect to face.

Example:
A robot has to find the exit of an unknown maze.

→ Perform simulations on other mazes beforehand
→ Learn an algorithm based on those experiences (e.g.: Wall follower)
Contents

• Introduction
• Problem Statement
• Offline Prior-based Policy-search (OPPS)
• Artificial Neural Networks for BRL (ANN-BRL)
• Benchmarking for BRL
• Conclusion
Let $M = (X, U, x_0, f_M(\cdot), \rho_M(\cdot), \gamma)$ be a given unknown MDP, where

- $X = \{x^{(1)}, \ldots, x^{(n_X)}\}$ denotes its finite state space
- $U = \{u^{(1)}, \ldots, u^{(n_U)}\}$ denotes its finite action space
- $x_0^M$ denotes its initial state.
- $x' \sim f_M(x, u)$ denotes the next state when performing action $u$ in state $x$
- $r_t = \rho_M(x_t, u_t, x_{t+1}) \in [R_{\min}, R_{\max}]$ denotes an instantaneous deterministic, bounded reward
- $\gamma \in [0, 1]$ denotes its discount factor

Let $h_t = (x_0^M, u_0, r_0, x_1, \cdots, x_{t-1}, u_{t-1}, r_{t-1}, x_t)$ denote the history observed until time $t$. 
An E/E strategy is a stochastic policy $\pi$ that, given the current history $h_t$ returns an action $u_t$:

$$u_t \sim \pi(h_t)$$

The expected return of a given E/E strategy $\pi$ on MDP $M$:

$$J_{\pi}^M = \mathbb{E}_M \left[ \sum_t \gamma^t r_t \right]$$

where

$$x_0 = x_0^M$$

$$x_{t+1} \sim f_M(x_t, u_t)$$

$$r_t = \rho_M(x_t, u_t, x_{t+1})$$
**RL (no prior distribution)**

We want to find a high-performance E/E strategy $\pi^*_M$ for a given MDP $M$:

$$\pi^*_M \in \arg \max_{\pi} J^\pi_M$$

**BRL (prior distribution $p^0_M(\cdot)$)**

A prior distribution defines a distribution over each uncertain component of $\mathcal{M}$ ($f_M(\cdot)$ in our case).

Given a prior distribution $p^0_M(\cdot)$, the goal is to find a policy $\pi^*$, called *Bayes optimal*:

$$\pi^* = \arg \max_{\pi} \widehat{J}^\pi_{p^0_M(\cdot)}$$

where

$$\widehat{J}^\pi_{p^0_M(\cdot)} = \mathbb{E}_{M \sim p^0_M(\cdot)} J^\pi_M$$
Contents

• Introduction
• Problem Statement
• Offline Prior-based Policy-search (OPPS)
• Artificial Neural Networks for BRL (ANN-BRL)
• Benchmarking for BRL
• Conclusion
Offline Prior-based Policy-search (OPPS)

1. Define a rich set of E/E strategies:
   → Build a large set of $N$ formulas
   → Build a formula-based strategy for each formula of this set

2. Search for the best E/E strategy in average, according to the given MDP distribution:
   → Formalise this problem as an $N$-armed bandit problem
1. Define a rich set of E/E strategies

Let $\mathbb{F}^K$ be the discrete set of formulas of size at most $K$. A formula of size $K$ is obtained by combining $K$ elements among:

- **Variables:**
  \[ \hat{Q}^t_1(x, u), \hat{Q}^t_2(x, u), \hat{Q}^t_3(x, u) \]

- **Operators:**
  \[ +, -, \times, /, | \cdot |, \frac{1}{\cdot}, \sqrt{\cdot}, \min(\cdot, \cdot), \max(\cdot, \cdot) \]

**Examples:**

- Formula of size 2: \[ F(x, u) = | \hat{Q}^t_1(x, u) | \]
- Formula of size 4: \[ F(x, u) = \hat{Q}^t_3(x, u) - | \hat{Q}^t_1(x, u) | \]

To each formula $F \in \mathbb{F}^K$, we associate a formula-based strategy $\pi_F$, defined as follows:

\[
\pi_F(h_t) \in \arg\max_{u \in U} F(x_t, u)
\]
Problems:

- \( F^K \) is too large
  \(|F^5| \simeq 300,000 \) formulas for 3 variables and 11 operators

- Formulas of \( F^K \) are redundant
  (= different formulas can define the same policy)

Examples:

1. \( Q^t_1(x, u) \) and \( Q^t_1(x, u) - Q^t_3(x, u) + Q^t_3(x, u) \)
2. \( Q^t_1(x, u) \) and \( \sqrt{Q^t_1(x, u)} \)

Solution:

⇒ Reduce \( F^K \)
Reduction process

→ Partition $\mathbb{F}^K$ into equivalence classes, two formulas being equivalent if and only if they lead to the same policy

→ Retrieve the formula of minimal length of each class into a set $\bar{\mathbb{F}}^K$

Example:
$|\bar{\mathbb{F}}^5| \simeq 3,000$ while $|\mathbb{F}^5| \simeq 300,000$

Computing $\bar{\mathbb{F}}^K$ may be expensive. We instead use an efficient heuristic approach to compute a good approximation of this set.
2. Search for the best E/E strategy in average

A naive approach based on Monte-Carlo simulations (= evaluating all strategies) is time-inefficient, even after the reduction of the set of formulas.

Problem:
In order to discriminate between the formulas, we need to compute an accurate estimation of $\hat{J}_{P_{\mathcal{M}}}^{\pi_0}(\cdot)$ for each formula, which requires a large number of simulations.

Solution:
Distribute the computational resources efficiently.
⇒ Formalise this problem as a multi-armed bandit problem and use a well-studied algorithm to solve it.
What is a multi-armed bandit problem?

A reinforcement learning problem where the agent is facing bandit machines and has to identify the one providing the highest reward in average with a given number of tries.
Formalisation

Formalise this research as a $N$-armed bandit problem.

- To each formula $F_n \in \mathcal{F}^K$ ($n \in \{1, \ldots, N\}$), we associate an arm.
- Pulling the arm $n$ consists in randomly drawing a MDP $M$ according to $p^0_M(\cdot)$, and perform a single simulation of policy $\pi_{F_n}$ on $M$.
- The reward associated to arm $n$ is the observed discounted return of $\pi_{F_n}$ on $M$.

$\Rightarrow$ This defines a multi-armed bandit problem for which many algorithms have been proposed (e.g.: UCB1, UCB-V, KL-UCB, ...).
Learning Exploration/Exploitation in Reinforcement Learning

M. Castronovo, F. Maes, R. Fonteneau & D. Ernst (EWRL 2012, 8 pages)

BAMCP versus OPPS: an Empirical Comparison

M. Castronovo, D. Ernst & R. Fonteneau (BENELEARN 2014, 8 pages)
Contents

• Introduction
• Problem Statement
• Offline Prior-based Policy-search (OPPS)
• Artificial Neural Networks for BRL (ANN-BRL)
• Benchmarking for BRL
• Conclusion
Artificial Neural Networks for BRL (ANN-BRL)

We exploit an analogy between decision-making and classification problems.

A reinforcement learning problem consists in finding a policy $\pi$ which associates an action $u \in U$ to any history $h$.

A multi-class classification problem consists in finding a rule $C(\cdot)$ which associates a class $c \in \{1, \ldots, K\}$ to any vector $v \in \mathbb{R}^n$ ($n \in \mathbb{N}$).

⇒ Formalise a BRL problem as a classification problem in order to use any classification algorithms such as Artificial Neural Networks.
1. Generate a training dataset:
   → Perform simulations on MDPs drawn from $p^0_\mathcal{M}(\cdot)$
   → For each encountered history, recommend an action
   → Reprocess each history $h$ into a vector of fixed size
     ⇒ Extract a fixed set of features (= variables for OPPS)

2. Train ANNs:
   ⇒ Use a boosting algorithm
1. Generate a training dataset

In order to generate a trajectory, we need a policy:

- A random policy?
  
  **Con:** Lack of histories for late decisions

- An optimal policy? ($f_M(\cdot)$ is known for $M \sim p^0_M(\cdot)$)
  
  **Con:** Lack of histories for early decisions

$\Rightarrow$ Why not both?

Let $\pi^{(i)}$ be an $\epsilon$-Optimal policy used for drawing trajectory $i$ (on a total of $n$ trajectories).

$$\text{For } \epsilon = \frac{i}{n} : \pi^{(i)}(h_t) = u^* \text{ with probability } 1 - \epsilon$$

and is drawn randomly in $U$ else.
To each history $h_0^{(1)}, \ldots, h_{T-1}^{(1)}, \ldots, h_0^{(n)}, \ldots, h_{T-1}^{(n)}$ observed during the simulations, we associate a label to each action:

- 1 if we recommend the action
- $-1$ else

**Example:**

$U = \{u^{(1)}, u^{(2)}, u^{(3)}\}$ : $h_0^{(1)} \leftrightarrow (-1, 1, -1)$

$\Rightarrow$ We recommend action $u^{(2)}$

We recommend actions which are optimal w.r.t. $M$ ($f_M(\cdot)$ is known for $M \sim p^0_M(\cdot)$).
Reprocess of all histories in order to feed the ANNs with vectors of fixed size.

⇒ Extract a fixed set of $N$ features: $\phi_{ht} = [\phi_{ht}^{(1)}, \ldots, \phi_{ht}^{(N)}]$

We considered two types of features:

- **Q-Values:**
  \[
  \phi_{ht} = [Q_{ht}(x_t, u^{(1)}), \ldots, Q_{ht}(x_t, u^{(nU)})]
  \]

- **Transition counters:**
  \[
  \phi_{ht} = [C_{ht}(< x^{(1)}, u^{(1)}, x^{(1)}>), \ldots, \\
  C_{ht}(< x^{(nX)}, u^{(nU)}, x^{(nX)}>)]
  \]
2. Train ANNs

Adaboost algorithm:

1. Associate a weight to each sample of the training dataset
2. Train a weak classifier on the weighted training dataset
3. Increase the weights of the samples misclassified by the combined weak classifiers trained previously
4. Repeat from Step 2

Problems

- Adaboost only addresses two-class classification problems (reminder: we have one class for each action) ⇒ Use SAMME algorithm instead
- Backpropagation does not take the weights of the samples into account ⇒ Use a re-sampling algorithm for the training dataset
Approximate Bayes Optimal Policy Search using NNs

M. Castronovo, V. François-Lavet, R. Fonteneau, D. Ernst & A. Couëtoux (ICAART 2017, 13 pages)
Contents

- Introduction
- Problem Statement
- Offline Prior-based Policy-search (OPPS)
- Artificial Neural Networks for BRL (ANN-BRL)
- Benchmarking for BRL
- Conclusion
Benchmarking for BRL

Bayesian literature

Compare the performance of each algorithm on well-chosen MDPs with several prior distributions.

Our benchmark

Compare the performance of each algorithm on a distribution of MDPs using a (possibly) different distribution as prior knowledge.

Prior distribution = Test distribution ⇒ Accurate case
Prior distribution ≠ Test distribution ⇒ Inaccurate case

Additionally, computation times of each algorithm is part of our comparison criteria.
Motivations:

⇒ No selection bias
   (= good on a single MDP \neq good on a distribution of MDPs)
⇒ Accurate case evaluates generalisation capabilities
⇒ Inaccurate case evaluates robustness capabilities
⇒ Real-life applications are subject to time constraints
   (= computation times cannot be overlooked)
The Experimental Protocol

An experiment consists in evaluating the performances of several algorithms on a test distribution $p_M(\cdot)$ when trained on a prior distribution $p_0^M(\cdot)$.

One algorithm $\rightarrow$ several agents (we test several configurations)

We draw $N$ MDPs $M^{(1)}, \ldots, M^{(N)}$ from the test distribution $p_M(\cdot)$ in advance, and we evaluate the agents as follows:

$\rightarrow$ Build policy $\pi$ offline w.r.t. $p_0^M(\cdot)$

$\rightarrow$ For each sampled MDP $M^{(i)}$, compute estimate $\bar{J}_M^{\pi(i)}$ of $J_M^{\pi(i)}$

$\rightarrow$ Use these values to compute estimate $\bar{J}_{p_M}^{\pi(\cdot)}$ of $J_{p_M}^{\pi(\cdot)}$
Estimate $J^\pi_M$:
Truncate each trajectory after $T$ steps:

$$
T = \left\lceil \log\left( \frac{\eta \times (1 - \gamma)}{R_{max}} \right) / \log \gamma \right\rceil
$$

$$
J^\pi_M \approx \tilde{J}^\pi_M = \sum_{t}^{T} r_t \gamma^t
$$

where $\eta$ denotes the accuracy of our estimate.
Estimate $\tilde{J}_p^\pi(\cdot)$:

We compute $\mu_\pi = \tilde{J}_p^\pi(\cdot)$ and $\sigma_\pi$, the empirical mean and standard deviation of the results observed on the $N$ MDPs drawn from $p_\mathcal{M}(\cdot)$.

The statistical confidence interval at 95% for $\tilde{J}_p^\pi(\cdot)$ is computed as:

$$
\tilde{J}_p^\pi(\cdot) \approx \tilde{J}_p^\pi(\cdot) = \frac{1}{N} \sum_{1 \leq i \leq N} \tilde{J}_p^M(i)
$$

$$
\tilde{J}_p^\pi(\cdot) \in \left[ \tilde{J}_p^\pi(\cdot) - \frac{2\sigma_\pi}{\sqrt{N}}; \tilde{J}_p^\pi(\cdot) + \frac{2\sigma_\pi}{\sqrt{N}} \right]
$$
Time constraints

We want to classify algorithms based on their time performance.

More precisely, we want to identify the best algorithm(s) with respect to:

1. Offline computation time constraint
2. Online computation time constraint

We filter the agents depending on the time constraints:

- Agents not satisfying the time constraints are discarded
- For each algorithm, we select the best agent in average
- We build the list of agents whose performances are not statistically different than the best one observed ($Z$-test)
Experiments

GC - Generalised Chain

GDL - Generalised Double-loop

Grid

$\text{GC}(n_x = 5, n_U = 3); \text{GDL}(n_x = 9, n_U = 2); \text{Grid}(n_x = 25, n_U = 4)$
Simple algorithms

- Random
- $\epsilon$-Greedy
- Soft-Max

State-of-the-art BRL algorithms

- BAMCP
- BFS3
- SBOSS
- BEB

Our algorithms

- OPPS-DS
- ANN-BRL
Results

Figure: Best algorithms w.r.t offline/online periods (accurate case)
<table>
<thead>
<tr>
<th>Agent</th>
<th>Score on GC</th>
<th>Score on GDL</th>
<th>Score on Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>31.12 ± 0.90</td>
<td>2.79 ± 0.07</td>
<td>0.22 ± 0.06</td>
</tr>
<tr>
<td>e-Greedy</td>
<td>40.62 ± 1.55</td>
<td>3.05 ± 0.07</td>
<td>6.90 ± 0.31</td>
</tr>
<tr>
<td>Soft-Max</td>
<td>34.73 ± 1.74</td>
<td>2.79 ± 0.10</td>
<td>0.00 ± 0.00</td>
</tr>
<tr>
<td>BAMCP</td>
<td>35.56 ± 1.27</td>
<td><strong>3.11 ± 0.07</strong></td>
<td>6.43 ± 0.30</td>
</tr>
<tr>
<td>BFS3</td>
<td>39.84 ± 1.74</td>
<td>2.90 ± 0.07</td>
<td>3.46 ± 0.23</td>
</tr>
<tr>
<td>SBOSS</td>
<td>35.90 ± 1.89</td>
<td>2.81 ± 0.10</td>
<td>4.50 ± 0.33</td>
</tr>
<tr>
<td>BEB</td>
<td>41.72 ± 1.63</td>
<td>3.09 ± 0.07</td>
<td>6.76 ± 0.30</td>
</tr>
<tr>
<td>OPPS-DS</td>
<td><strong>42.47 ± 1.91</strong></td>
<td>3.10 ± 0.07</td>
<td><strong>7.03 ± 0.30</strong></td>
</tr>
<tr>
<td>ANN-BRL (Q)</td>
<td>42.01 ± 1.80</td>
<td><strong>3.11 ± 0.08</strong></td>
<td>6.15 ± 0.31</td>
</tr>
<tr>
<td>ANN-BRL (C)</td>
<td>35.95 ± 1.90</td>
<td>2.81 ± 0.09</td>
<td>4.09 ± 0.31</td>
</tr>
</tbody>
</table>

**Table**: Best algorithms w.r.t Performance (accurate case)
Figure: Best algorithms w.r.t offline/online periods (inaccurate case)
<table>
<thead>
<tr>
<th>Agent</th>
<th>Score on GC</th>
<th>Score on GDL</th>
<th>Score on Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>31.67 ± 1.05</td>
<td>2.76 ± 0.08</td>
<td>0.23 ± 0.06</td>
</tr>
<tr>
<td>e-Greedy</td>
<td>37.69 ± 1.75</td>
<td>2.88 ± 0.07</td>
<td>0.63 ± 0.09</td>
</tr>
<tr>
<td>Soft-Max</td>
<td>34.75 ± 1.64</td>
<td>2.76 ± 0.10</td>
<td>0.00 ± 0.00</td>
</tr>
<tr>
<td>BAMCP</td>
<td>33.87 ± 1.26</td>
<td>2.85 ± 0.07</td>
<td>0.51 ± 0.09</td>
</tr>
<tr>
<td>BFS3</td>
<td>36.87 ± 1.82</td>
<td>2.85 ± 0.07</td>
<td>0.42 ± 0.09</td>
</tr>
<tr>
<td>SBOSS</td>
<td>38.77 ± 1.89</td>
<td>2.86 ± 0.07</td>
<td>0.29 ± 0.07</td>
</tr>
<tr>
<td>BEB</td>
<td>38.34 ± 1.62</td>
<td>2.88 ± 0.07</td>
<td>0.29 ± 0.05</td>
</tr>
<tr>
<td>OPPS-DS</td>
<td>39.29 ± 1.71</td>
<td>2.99 ± 0.08</td>
<td>1.09 ± 0.17</td>
</tr>
<tr>
<td>ANN-BRL (Q)</td>
<td>38.76 ± 1.71</td>
<td>2.92 ± 0.07</td>
<td>4.29 ± 0.22</td>
</tr>
<tr>
<td>ANN-BRL (C)</td>
<td>36.30 ± 1.82</td>
<td>2.84 ± 0.08</td>
<td>0.91 ± 0.15</td>
</tr>
</tbody>
</table>

**Table:** Best algorithms w.r.t Performance (inaccurate case)
BAMCP versus OPPS: an Empirical Comparison
M. Castronovo, D. Ernst & R. Fonteneau (BENELEARN 2014, 8 pages)

Benchmarking for Bayesian Reinforcement Learning
Contents

• Introduction
• Problem Statement
• Offline Prior-based Policy-search (OPPS)
• Artificial Neural Networks for BRL (ANN-BRL)
• Benchmarking for BRL
• Conclusion
Conclusion

Summary

1. **Algorithms:**
   - Offline Prior-based Policy-search (OPPS)
   - Artificial Neural Networks for BRL (ANN-BRL)

2. **New BRL benchmark**

3. **An open-source library**
BBRL: Benchmarking tools for Bayesian Reinforcement Learning

BBRL is a C++ open-source library used to compare Bayesian reinforcement learning algorithms.

BBRL (Benchmarks tools for Bayesian Reinforcement Learning) is a C++ open-source library for Bayesian reinforcement learning (discrete state/action spaces). We decided to develop these tools by gathering as many agents and experiments as possible in order to provide a unified framework for Bayesian reinforcement learning.

Problem Statement

Reinforcement Learning is a field of Machine Learning, which aims to learn how to behave in different circumstances based on previous experiences. We consider an agent interacting with an environment. At each time-step, the agent is in a given state and has to perform an action. In return, the environment sends a reward and moves the agent to another state. The goal of the agent is to optimise the rewards collected by taking good decisions, based on its current state and what has happened during the previous interactions.

The environment is generally represented by a Markov Decision Process (MDP), which is defined as follows:

- The state space, which is the set of all possible states in which an agent can be.
- The action space, which is the set of all possible actions that an agent can perform.
- The transition function, which is the function defining how to move an agent from one state to another based on the current state and the action it chose to perform.
- The reward function, which is the function defining the reward to send based on the current state and the action it chose to perform.

https://github.com/mcastron/BBRL/
Future work

- OPPS
  - Feature selection (PCA)
  - Continuous formula space

- ANN-BRL
  - Extension to high-dimensional problems
  - Replace ANNs by other ML algorithms (e.g.: SVMs, decision trees)

- BRL Benchmark
  - Design new distributions to identify specific characteristics

- Flexible BRL algorithm
  - Design an algorithm to exploit both offline and online phases
Thanks for your attention!