When to use parametric models in reinforcement learning?

Hado van Hasselt, Matteo Hessel, John Aslanides
DeepMind
Motivation / prior work

Model Based Reinforcement Learning for Atari

Łukasz Kaiser*1 Mohammad Babaeizadeh *2,3 Piotr Miłoś*4,5 Błażej Osiński*4,5,3 Roy H Campbell2 Konrad Czechowski4 Dumitru Erhan1 Chelsea Finn1 Piotr Kozakowski4 Sergey Levine1 Afroz Mohiuddin1 Ryan Sepassi1 George Tucker1 Henryk Michalewski*4,5
Motivation / prior work
Motivation / prior work

Performance (=score) after 100,000 steps (=400,000 frames)

Baseline: Rainbow DQN
Question

Why does the parametric model perform better than replay?
Models and planning

A model is a function:

\[ r, s' = m(s, a) \]

We can use models to plan: spend more compute to improve prediction & policies.

We can also plan with experience replay:

\[ r_{n+1}, s_{n+1} = \text{replay}(s_n, a_n) \]

- Experience replay is similar to a non-parametric model
- But we can only query it at observed state action pairs \((s_n, a_n), n < t\).
Replay and models, properties

Typically models use less memory and more compute than replay.

But what about data efficiency & performance?
Algorithms

\[
\text{for iteration } \in \{1, 2, \ldots, K\} \text{ do}
\]
\[
\quad \text{for interaction } \in \{1, 2, \ldots, M\} \text{ do}
\quad \quad \text{Generate action: } a \leftarrow \pi(s)
\quad \quad \text{Generate reward, next state: } r, s' \leftarrow \mathcal{E}(a)
\quad \quad m, d \leftarrow \text{UPDATEMODEL}(s, a, r, s')
\quad \quad \pi, v \leftarrow \text{UPDATEAGENT}(s, a, r, s')
\quad \quad \text{Update current state: } s \leftarrow s'
\quad \text{end for}
\]
\[
\text{for planning step } \in \{1, 2, \ldots, P\} \text{ do}
\quad \text{Generate state, action } \tilde{s}, \tilde{a} \leftarrow d
\quad \quad \text{Generate reward, next state: } \tilde{r}, \tilde{s}' \leftarrow m(\tilde{s}, \tilde{a})
\quad \quad \pi, v \leftarrow \text{UPDATEAGENT}(\tilde{s}, \tilde{a}, \tilde{r}, \tilde{s}')
\quad \text{end for}
\]
## Algorithms

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Iterations (K)</th>
<th>Real steps per iteration (M)</th>
<th>Planned steps per iteration (P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimPLe</td>
<td>16</td>
<td>6400</td>
<td>800,000</td>
</tr>
<tr>
<td>Rainbow DQN</td>
<td>12,500,000</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>Data-efficient Rainbow DQN (new)</td>
<td>100,000</td>
<td>1</td>
<td>32</td>
</tr>
</tbody>
</table>
### Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Total real experience ($K \times M$)</th>
<th>Total planned experience ($K \times P$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimPLe</td>
<td>100,000 (400K frames)</td>
<td>15,200,000</td>
</tr>
<tr>
<td>Rainbow DQN</td>
<td>50,000,000 (200M frames)</td>
<td>400,000,000</td>
</tr>
<tr>
<td>Data-efficient Rainbow DQN (new)</td>
<td>100,000 (400K frames)</td>
<td>3,200,000</td>
</tr>
</tbody>
</table>
Algorithms
When do models help performance?
What if learning the model is easy?
Surprising instabilities

- Even with **perfect models** learning can be unstable
- This happens surprisingly easily!
- Related to the **deadly triad**:
  - the state sampling distribution $d$ and the model $m$ may mismatch, even if the model is perfect.
How can we best use learnt models?
Forward planning for credit assignment

The maze

Scalability

Total steps (log. scale)

Updates per real step
Backward planning for credit assignment
Forward planning for behaviour
Conclusions

1. **Replay** can be used for **planning**

2. There are different ways to use models:
   a. **Forward** planning for **credit assignment**
   b. **Forward** planning for **immediate behaviour**
   c. **Backward** planning for **credit assignment**

3. **b.** and **c.** might be better than **a.**
Thank you

https://arxiv.org/abs/1906.05243
Related work on forward planning for credit assignment

“Dyna, an integrated architecture for learning, planning, and reacting”
← argues for combining learning with forward planning for credit assignment with a learnt model

“The Effect of Planning Shape on Dyna-style Planning in High-dimensional State Spaces”
Zach Holland, Eric Talvitie, Mike Bowling (2018)
← shows that it is important how to use your planning budget (e.g., short vs long rollouts)

“Model-Based Reinforcement Learning for Atari”
Lukasz Kaiser et al. (2019)
← introduces the SimPLe algorithm we compare against
Related work on forward planning for behaviour

Essentially: lots of work on model-predictive control, e.g.:


“Model predictive control: Recent developments and future promise” D. Q. Mayne (2014)

Related work on backward planning for credit assignment

Comparatively less...? (More common: plan backwards from a real or imagined goal.)

E.g., doesn’t show up in:

Note: in dynamic programming and replay **forward and backward planning are equivalent**.
The only distinction is then which states to update, i.e., prioritized sweeping:
“Prioritized sweeping: Reinforcement learning with less data and less time” Moore, Atkeson (1993)
“Efficient Learning and Planning Within the Dyna Framework” Peng, Williams (1993)

The idea of using learnt models to plan backwards for credit assignment is perhaps underexplored
## Rainbow DQN hyperparameters changes

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>canonical</th>
<th>data-efficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training frames</td>
<td>200,000,000</td>
<td>400,000</td>
</tr>
<tr>
<td>Min replay size for sampling</td>
<td>20,000</td>
<td>1600</td>
</tr>
<tr>
<td>Memory size</td>
<td>1,000,000 steps</td>
<td>unbounded</td>
</tr>
<tr>
<td>Replay period every</td>
<td>4 steps</td>
<td>1 steps</td>
</tr>
<tr>
<td>Multi-step return length</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Q network: channels</td>
<td>32, 64, 64</td>
<td>32, 64</td>
</tr>
<tr>
<td>Q network: filter size</td>
<td>$8 \times 8, 4 \times 4, 3 \times 3$</td>
<td>$5 \times 5, 5 \times 5$</td>
</tr>
<tr>
<td>Q network: stride</td>
<td>4, 2, 1</td>
<td>5, 5</td>
</tr>
<tr>
<td>Q network: hidden units</td>
<td>512</td>
<td>256</td>
</tr>
<tr>
<td>Optimizer: learning rate</td>
<td>0.00000625</td>
<td>0.00001</td>
</tr>
</tbody>
</table>