AsyncQVI: Asynchronous Parallel Q-value Iteration for Markov Decision Processes

Yibo Zeng
Columbia University, IEOR

Fei Feng
UCLA, Math

Wotao Yin
UCLA, Math
Markov Decision Process

A framework for **Reinforcement Learning**.

- \( M := (S, A, p, r, \gamma) \).
- Transition kernel \( p(s_{t+1} | s_t, a_t) \) & reward function \( r(s_t, a_t, s_{t+1}) \)

- A **policy** \( \pi: S \rightarrow A \)
- We execute \( \pi \) to obtain a trajectory: \( s_0, a_0, r_0, s_1, a_1, r_1 \ldots \)

- **State-value function:**

\[
V^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, \pi \right]
\]

- **Goal:** find a policy that maximizes the value function \( V^\pi \).
Recent Successes and Time Demands in RL

[DeepMind 2015]
Human-level performance on **Atari 2600**. Trained for **38 days** of game experience

[DeepMind 2017]
**AlphaGo Zero.** Trained for **40 days** to surpass all old versions.

[OpenAI 2019]
Defeating **Dota 2** world champion. Trained for **180 days**.

Parallel computing accelerates training.

[Minh et al. 2016]

Fei Feng @ UCLA MATH
Parallel Computing

Sync-parallel
- probably long idle time;
- little tolerant to communication glitches;
- keeps information consistent.

Async-parallel
- saves idle time;
- more tolerant to communication glitches;
- easier to incorporate new agents;
- information is delayed or inconsistent.

Our goal: a theoretically justified async-parallel algorithm for MDPs.
Review the Bellman Operator

• Initialize a table $Q_0$ of size $S \times A$.

• Bellman Operator (Q-value iteration):

$$Q_{t+1}(s, a) = T^B(Q_t) = E_{s' \sim p(\cdot | s, a)}[r(s, a) + \gamma \max_{a'} Q_t(s', a')]$$  \hspace{1cm} (1)

• The fixed point $Q^*$ can induce an optimal policy [Sutton & Barto, 1999].

$T^B$ is friendly for parallel design:

- A nice structure: linear (expectation) + simple nonlinear (max).
- A nice convergence property: $\gamma$-contraction under $|| \cdot ||_\infty$. 

Fei Feng @ UCLA MATH
How we build AsyncQVI.

1. The original Q-value iteration:

$$Q_{s,a}(t+1) = \sum_{s'} p_{ss'}^a (r_{ss'}^a + \gamma \max_{a'} Q_{s',a'}(t)), \quad \forall (s,a) \in S \times A.$$

2. Revise to a coordinate-update fashion:

$$Q_{s,a}(t+1) = \begin{cases} 
\sum_{s'} p_{ss'}^a (r_{ss'}^a + \gamma \max_{a'} Q_{s',a'}(t)), & (s,a) = (s_{t+1}, a_{t+1}); \\
Q_{s,a}(t), & (s,a) \neq (s_{t+1}, a_{t+1}).
\end{cases}$$

3. Implement in an asynchronous parallel manner:

$$Q_{s,a}(t+1) = \begin{cases} 
\sum_{s'} p_{ss'}^a (r_{ss'}^a + \gamma \max_{a'} \hat{Q}_{s',a'}), & (s,a) = (s_{t+1}, a_{t+1}); \\
Q_{s,a}(t), & (s,a) \neq (s_{t+1}, a_{t+1}).
\end{cases}$$

4. Approximate with Samples:

$$Q_{s,a}(t+1) = \begin{cases} 
\frac{1}{K} \sum_{k=1}^K (r_k + \gamma \max_{a'} \hat{Q}_{s_k,a'}(t)) - c, & (s,a) = (s_{t+1}, a_{t+1}); \\
Q_{s,a}(t), & (s,a) \neq (s_{t+1}, a_{t+1}).
\end{cases}$$

Update ALL entries per iteration. Time complexity per iteration: $O(S^2 A)$. 

Fei Feng @ UCLA MATH
How we build AsyncQVI.

1. The original Q-value iteration:
   \[ Q_{s,a}(t+1) = \sum_{s'} p_{ss'}^a (r_{ss'}^a + \gamma \max_{a'} Q_{s',a'}(t)), \quad \forall (s, a) \in S \times A. \]

2. Revise to a coordinate-update fashion:
   \[ Q_{s,a}(t+1) = \begin{cases} \sum_{s'} p_{ss'}^a (r_{ss'}^a + \gamma \max_{a'} Q_{s',a'}(t)), & (s, a) = (s_{t+1}, a_{t+1}); \\ Q_{s,a}(t), & (s, a) \neq (s_{t+1}, a_{t+1}). \end{cases} \]

3. Implement in an asynchronous parallel manner:
   \[ Q_{s,a}(t+1) = \begin{cases} \sum_{s'} p_{ss'}^a (r_{ss'}^a + \gamma \max_{a'} Q_{s',a'}(t)), & (s, a) = (s_{t+1}, a_{t+1}); \\ Q_{s,a}(t), & (s, a) \neq (s_{t+1}, a_{t+1}). \end{cases} \]

4. Approximate with Samples:
   \[ Q_{s,a}(t+1) = \begin{cases} \frac{1}{K} \sum_{k=1}^{K} (r_k + \gamma \max_{a'} Q_{s_k',a'}(t)) - c, & (s, a) = (s_{t+1}, a_{t+1}); \\ Q_{s,a}(t), & (s, a) \neq (s_{t+1}, a_{t+1}) \end{cases} \]

Update ONE entry per iteration.
Time complexity per iteration: \( O(S) \).
How we build AsyncQVI.

1. The original Q-value iteration:

\[ Q_{s,a}(t+1) = \sum_{s'} p_{ss'}^{a} \left( r_{ss'}^{a} + \gamma \max_{a'} Q_{s',a'}(t) \right), \quad \forall (s, a) \in S \times A. \]

2. Revise to a coordinate-update fashion:

\[ Q_{s,a}(t+1) = \begin{cases} \sum_{s'} p_{ss'}^{a} \left( r_{ss'}^{a} + \gamma \max_{a'} Q_{s',a'}(t) \right), & (s, a) = (s_{t+1}, a_{t+1}); \\ Q_{s,a}(t), & (s, a) \neq (s_{t+1}, a_{t+1}). \end{cases} \]

3. Implement in an asynchronous parallel manner:

\[ Q_{s,a}(t+1) = \begin{cases} \sum_{s'} p_{ss'}^{a} \left( r_{ss'}^{a} + \gamma \max_{a'} Q_{s',a'}(t) \right), & (s, a) = (s_{t+1}, a_{t+1}); \\ Q_{s,a}(t), & (s, a) \neq (s_{t+1}, a_{t+1}). \end{cases} \]

4. Approximate with Samples:

\[ Q_{s,a}(t+1) = \begin{cases} \frac{1}{K} \sum_{k=1}^{K} \left( r_{k} + \gamma \max_{a'} Q_{s_{k},a'}(t) \right) - c, & (s, a) = (s_{t+1}, a_{t+1}); \\ Q_{s,a}(t), & (s, a) \neq (s_{t+1}, a_{t+1}). \end{cases} \]

Parallel run with \( N \) agents.
Scatter computing load.
Asynchronous causes information delay.

Fei Feng @ UCLA MATH
How we build AsyncQVI.

1. **The original Q-value iteration:**
   \[
   Q_{s,a}(t+1) = \sum_{s'} p_{ss'}^a (r_{ss'}^a + \gamma \max_{a'} Q_{s',a'}(t)), \quad \forall (s,a) \in S \times A.
   \]

2. **Revise to a coordinate-update fashion:**
   \[
   Q_{s,a}(t+1) = \begin{cases} 
   \sum_{s'} p_{ss'}^a (r_{ss'}^a + \gamma \max_{a'} Q_{s',a'}(t)), & (s,a) = (s_{t+1}, a_{t+1}); \\
   Q_{s,a}(t), & (s,a) \neq (s_{t+1}, a_{t+1}).
   \end{cases}
   \]

3. **Implement in an asynchronous parallel manner:**
   \[
   Q_{s,a}(t+1) = \begin{cases} 
   \sum_{s'} p_{ss'}^a (r_{ss'}^a + \gamma \max_{a'} Q_{s',a'}(t)), & (s,a) = (s_{t+1}, a_{t+1}); \\
   Q_{s,a}(t), & (s,a) \neq (s_{t+1}, a_{t+1}).
   \end{cases}
   \]

4. **Approximate with Samples:**
   \[
   Q_{s,a}(t+1) = \begin{cases} 
   \frac{1}{K} \sum_{k=1}^K (r_k + \gamma \max_{a'} Q_{s',a'}(k)) - c, & (s,a) = (s_{t+1}, a_{t+1}); \\
   Q_{s,a}(t), & (s,a) \neq (s_{t+1}, a_{t+1}).
   \end{cases}
   \]

---

**Generative Model (GM):**
- Input: any \((s,a)\)
- Output: \((s',r) \sim p(\cdot|s,a)\)

Stochastic approximation with a generative model. Scatter sampling load.
AsyncQVI

N computing agent continuously and asynchronously do:

- Selection can be random or cyclic.
- Memory complexity: $O(S)$;
- The copying can be done in a less frequent fashion;
- The overhead of updating lock is negligible;
- Parallel speedup: N times faster (ideally).
Theoretical Guarantee

Assumptions [partial asynchronism]:
1. Delay is uniformly bounded by $B_1$;
2. The time interval between consecutive updates for each entry is bounded by $B_2$.

The sample complexity of single-thread methods with a generative model is:

$$\tilde{O}\left(\frac{B_1 + B_2}{(1 - \gamma)^5 \epsilon^2} \log \left(\frac{1}{\delta}\right)\right).$$

[Azar et al. 2013, Agarwal et al. 2019]

If $B_1 + B_2 = O(SA)$, our sample complexity is near-optimal.
## Related Works

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Setting</th>
<th>Async-parallel</th>
<th>Sample</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Totally Async QVI(^2)</td>
<td>Full knowledge</td>
<td>Unbddd delay</td>
<td>N/A</td>
<td>(\mathcal{O}(SA))</td>
</tr>
<tr>
<td>Partially Async QVI(^3)</td>
<td>Full knowledge</td>
<td>Bdd delay</td>
<td>N/A</td>
<td>(\mathcal{O}(SA))</td>
</tr>
<tr>
<td>Async Q-learning(^4)</td>
<td>RL</td>
<td>Unbddd delay</td>
<td>N/A</td>
<td>(\mathcal{O}(SA))</td>
</tr>
<tr>
<td>VRVI(^5)</td>
<td>Generative model</td>
<td>(\times)</td>
<td>(\tilde{O}\left(\frac{SA}{(1-\gamma)^4\varepsilon^2}\log\left(\frac{1}{\delta}\right)\right))</td>
<td>(\mathcal{O}(SA))</td>
</tr>
<tr>
<td>VRQVI(^6)</td>
<td>Generative model</td>
<td>(\times)</td>
<td>(\tilde{O}\left(\frac{SA}{(1-\gamma)^5\varepsilon^2}\log\left(\frac{1}{\delta}\right)\right))</td>
<td>(\mathcal{O}(SA))</td>
</tr>
<tr>
<td>AsyncQVI</td>
<td>Generative model</td>
<td>Bdd delay</td>
<td>(\tilde{O}\left(\frac{SA}{(1-\gamma)^5\varepsilon^2}\log\left(\frac{1}{\delta}\right)\right))</td>
<td>(\mathcal{O}(S))</td>
</tr>
</tbody>
</table>

\(^2\) Bertsekas and John N Tsitsiklis 1989.  
\(^3\) Bertsekas and John N Tsitsiklis 1989.  

AsyncQVI trades a few more samples for less running time and memory.
Numerical Experiment

- 100*100 grid world;
- Noises in transition;
- A big reward at the target;
- Minor traveling costs.

Parallel algorithms are run with 20 threads.
Conclusion:

• We propose an async-parallel algorithm for MDPs with a near-optimal sample complexity.
• AsyncQVI trades a little more samples for less time and memory.
• AsyncQVI has linear parallel speedup empirically.

Future work:

• Involve exploration.
• Consider function approximation.

Thank you!