Reinforcement learning 101

Reinforcement learning (RL) considers the very general framework of an agent taking actions in an environment to accumulate rewards, which can be seen as a sequential decision-making problem. These rewards act as feedback from the environment and quantify the performance of the agent with regard to the task/environment.

Examples of RL task environments:
- A board game of Go (or chess), where RL agents have already defeated world champions (AlphaGo)
- An optical table for RL agents to design and optimize experiments

Reinforcement learning: quantization scenarios

While many quantum algorithms for (un)supervised learning have been proposed in the last decade, relatively few have been dealing with quantum reinforcement learning (QRL).

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Supervised learning</th>
<th>Unsupervised learning</th>
<th>Reinforcement learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learn</td>
<td>P(labels</td>
<td>data)</td>
<td>Structure in P(data)</td>
</tr>
<tr>
<td>Samples from</td>
<td>P(data, labels)</td>
<td>P(data)</td>
<td>P(history</td>
</tr>
</tbody>
</table>

Table 1: A comparison of the three paradigms of machine learning. The history of a reinforcement learning agent is a sequence of actions, rewards and what it has experienced.

The curse of dimensionality and its solutions

Traditional methods for RL rely on learning a merit function defined on the entire state-action space, which estimates the expected future reward of an agent when performing a particular action in a given state. When learned on the entire state-action space, an optimal behavior (or policy) for the agent can be derived by following the actions with the largest merit value in any encountered state.

- **Projective Simulation (PS)**: $h^{(n+1)}(x, a) = h^{(n)}(x, a) - \gamma_p h^{(n)}(x, a) - 1 + F$
- **Q-learning [2]**: $Q^{(n+1)}(x, a) = (1 - \alpha)Q^{(n)}(x, a) + \alpha \left[ r + \gamma \max \limits_{a'} Q^{(n)}(x', a') \right]$
- **Policy**: $\pi(a|s) = \frac{e^{H(s, a)}}{\sum_{a'} e^{H(s, a')}}$
- **Drawbacks**: Lack of generalization: not updated or updated only once in a state.
- **Deep Q-learning**: $\left[ |v| \times |a| \right]$ values

The most basic learning algorithms for RL (e.g., Q-learning and Projective Simulation below) rely on a table of values stored in memory to approximate the merit function. These stored values are updated using an update rule proper to the learning method, and according to the rewards collected by interacting with the environment when following a given policy. This policy is itself derived from the stored table through normalization of its columns.

In real-world environments, e.g., Go play or quantum experiments described above, the size of the state and action spaces is gigantic. But basic tabular methods update one value in the table at a time, leading to learning times at least as large as these spaces.

A common solution to this problem relies on function approximation models, such as artificial neural networks. These do not represent each value in the table separately but define a family of functions parameterized by a set of weights. Training then consists in finding the most suited function (i.e., weights) in this family.

Quantum enhancements for reinforcement learning

Quantum enhancements can be of two kinds:
- Quantum algorithms to speed-up sampling from the agent policy
- Quantum generalizations of function approximation models to gain a learning advantage

Interesting candidates are:

- Quantum algorithms (e.g., Quantum Boltzmann Machines) to speed up the sampling process.
- Quantum machines (e.g., Quantum Circuits) to approximate the function $h$.

References