Implementing GQ($\lambda$) for RL in Soar

Mitchell Keith Bloch

University of Michigan
2260 Hayward Street
Ann Arbor, MI. 48109-2121
bazald@umich.edu

June 4, 2015
Why GQ(\(\lambda\))?

- It supports off-policy learning well and sometimes we care less about agent performance during training than agent performance after training.
- GQ(\(\lambda\)) converges despite irreversible actions and other difficulties approaching the training goal.
  - Imagine a robotic arm that is likely to knock over a tower of blocks just before achieving the goal configuration.
- It’s modern and the RL community thinks we should be using it.
On-Policy vs Off-Policy

From Sutton & Barto:

![Diagram showing optimal and safe paths in an environment with rewards and episodes for Sarsa and Q-learning](image)

Reward per episode vs Episodes for Sarsa and Q-learning.
Temporal Difference Methods—Simple

A value function, $Q(s, a)$, can explicitly store estimates of return

- **On-policy—Sarsa:**
  \[
  \delta \leftarrow r_t + \gamma Q_t(s', a') - Q_t(s, a)
  \]

- **Off-policy—Q-learning:**
  \[
  \delta \leftarrow r_t + \gamma \max_{a^*} Q_t(s', a^*) - Q_t(s, a)
  \]

Then for both:

\[
Q_{t+1}(s, a) \leftarrow Q_t(s, a) + \alpha \delta
\]
Temporal Difference Methods—Add Eligibility Traces

- **On-policy—Sarsa(\(\lambda\))**:
  \[
  \delta_t \leftarrow r_t + \gamma Q_t(s', a') - Q_t(s, a)
  \]

- **Off-policy—Q(\(\lambda\))**:
  \[
  \delta_t \leftarrow r_t + \gamma \max_{a^*} Q_t(s', a^*) - Q_t(s, a)
  \]

Then for both, \(\forall s, \forall a\):

- \[e_t(s, a) \leftarrow \lambda e_{t-1}(s, a) + \phi(s, a)\]
- \[Q_{t+1}(s, a) \leftarrow Q_t(s, a) + \alpha \delta_t e_t(s, a)\]
Using a weight vector to represent values increases generality

\[ Q(s, a) = \sum_{i=1}^{n} \theta_t(i) \phi_{s,a}(i) \]

For both Sarsa($\lambda$) and Q($\lambda$), given $\delta_t$, $\forall i$:

\[ e_t(i) \leftarrow \lambda e_{t-1}(i) + \frac{\phi_{s,a}(i)}{\sum_{i=1}^{n} \phi_{s,a}(i)} \]

\[ \theta_{t+1}(i) \leftarrow \theta_t(i) + \alpha \delta_t e_t(i) \]
Implementation of Function Approximation with Eligibility Traces (Soar 9.4)

- Store a list of eligible weights and currently active weights
- Every step:
  1. Loop through current weights to calculate $\delta_t$ and increase $e_t$
  2. Loop through $e_t$, applying $\delta_t$
  3. Decay the list of eligible weights, $e_t$
  4. If learning off-policy and choosing a non-greedy action, clear $e_t$
Temporal Difference Methods—GQ(\(\lambda\))

Big idea: guarantee convergence using a second weight vector

New requirements:
- \(w(i)\) – a secondary set of learned weights
- \(\eta\) – a secondary learning rate / step-size parameter
- \(\rho\) – importance sampling ratio
- \(I(s, a)\) – an interest function for hierarchical RL
Temporal Difference Methods—GQ(\(\lambda\))

\(w(i)\) – a secondary set of learned weights
\(\eta\) – a secondary learning rate / step-size parameter

- Ordinary Q(\(\lambda\)) can diverge
- Roughly, encourage \(\theta(i)\) to change in a consistent direction
- \(\eta\) affects the learning rate of \(w(i)\) only
$\rho_t = \frac{\pi(S_t,A_t)}{b(S_t,A_t)}$ – importance sampling ratio

- $Q(\lambda)$ requires eligibility to be explicitly cleared before exploration
- $\rho$ provides a generalization of that clearing
- Typically, $\rho_t > 1$ for greedy actions, so not a substitute for decay

$\forall i : e_t(i) \leftarrow \rho_t e_t(i)$ – incomplete (see next slide)
Temporal Difference Methods—GQ($\lambda$)

$I(s, a)$ – an interest function for hierarchical RL

- 1 for all values for flat RL
- 1 for initiating states in HRL
- 0 for non-initiating state in HRL
- Focuses learning on the states in which decisions are made

$$\forall i : e_t(i) \leftarrow \rho_t e_t(i) + I\phi_t(i)$$
What was necessary to add GQ(λ) to Soar?

- Provide a user-controlled step-size-parameter
- Add a second weight to each RL-Rule
- Calculate $\rho$, $I$, and a couple more intermediate variables
- Use $\rho$ instead of explicitly clearing traces
- Subtract off new GQ(λ) terms from current and next RL-rules
Cliff Walking

50 runs of 50 episodes, for a total of 2500 episodes

<table>
<thead>
<tr>
<th>Temporal Difference Method</th>
<th>Total Steps</th>
<th>Times Goal Reached</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sarsa($\lambda$)</td>
<td>72764</td>
<td>2093</td>
</tr>
<tr>
<td>On-Policy GQ($\lambda$)</td>
<td>72932</td>
<td>2083</td>
</tr>
<tr>
<td>Q($\lambda$)</td>
<td>72787</td>
<td>2096</td>
</tr>
<tr>
<td>Off-Policy GQ($\lambda$)</td>
<td>73124</td>
<td>2074</td>
</tr>
</tbody>
</table>
Nuggets:
- GQ($\lambda$) is now available for Soar agents to use in 9.5.
- Convergence should be guaranteed for stable environments.
- It appears to work well.

Coal:
- Should use a lower learning rate (Be aware!)
- step-size-parameter is another parameter to tune
- Computational cost is marginally higher.
- Second set of weights lost when reloading rules, like $e(i)$
- Performance is not guaranteed to dominate Sarsa($\lambda$) or Q($\lambda$).
- The goal is a convergence guarantee.
- This implementation could use additional testing and code review.