Implementing GQ(λ) for RL in Soar

Mitchell Keith Bloch

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

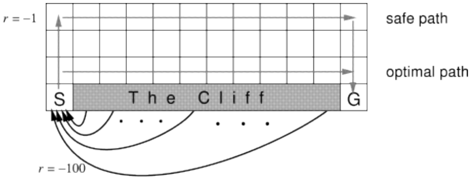
June 4, 2015

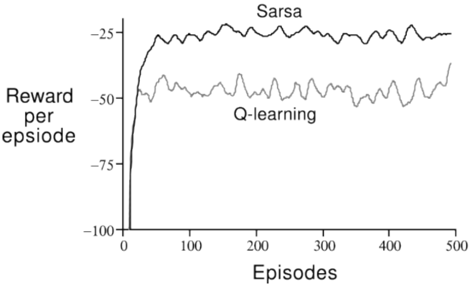
Why GQ(λ)?

- It supports off-policy learning well and sometimes we care less about agent performance during training than agent performance after training.
- GQ(λ) 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:





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(λ):

$$\delta_t \leftarrow r_t + \gamma Q_t(s', a') - Q_t(s, a)$$

- Off-policy—Q(λ):

$$\delta_t \leftarrow r_t + \gamma \max_{a^*} Q_t(s', a^*) - Q_t(s, a))$$

Then for both, $\forall s, \forall a$:

$$e_t(s, a) \leftarrow \lambda e_{t-1}(s, a) + \phi(s, a)$$

$$Q_{t+1}(s, a) \leftarrow Q_t(s, a) + \alpha \delta_t e_t(s, a)$$

TD Methods—Add Linear Function Approximation

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(λ) and Q(λ), 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 δ_t and increase e_t
    2. Loop through e_t, applying δ_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(λ)

Big idea: guarantee convergence using a second weight vector

New requirements:

- $w(i)$ – a secondary set of learned weights
- η – a secondary learning rate / step-size parameter
- ρ – importance sampling ratio
- $I(s, a)$ – an interest function for hierarchical RL

Temporal Difference Methods—GQ(λ)

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

- Ordinary Q(λ) can diverge
- Roughly, encourage $\theta(i)$ to change in a consistent direction
- η affects the learning rate of $w(i)$ only

Temporal Difference Methods—GQ(λ)

$\rho_t = \frac{\pi(S_t,A_t)}{b(S_t,A_t)}$ – importance sampling ratio

- Q(λ) requires eligibility to be explicitly cleared before exploration
- ρ 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(λ)

$I(s, a)$ – 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

$$\forall i : e_t(i) \leftarrow \rho_t e_t(i) + I\phi_t(i)$$

Implementing GQ(λ) in Soar 9.5

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 ρ, I, and a couple more intermediate variables
- Use ρ 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

Temporal Difference Method	Total Steps	Times Goal Reached
Sarsa(λ)	72764	2093
On-Policy GQ(λ)	72932	2083
Q(λ)	72787	2096
Off-Policy GQ(λ)	73124	2074

Nuggets and Coal

Nuggets:

- GQ(λ) 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(λ) or Q(λ). The goal is a convergence guarantee.
- This implementation could use additional testing and code review.