Multi-step RL: Unifying Algorithm

Kirill Bobyrev

November 16, 2017

Plan

Results

Introducing algorithm $Q(\sigma)$ with following properties

- Model-free
- Can be trained both off- and on-policy
- Uses *n*-step backup rule
- Generalizes a huge number of existing algorithms while subsuming them as special cases
- Performs better than base algorithms given a reasonable choice of hyperparameters

Classic RL algorithms landscape

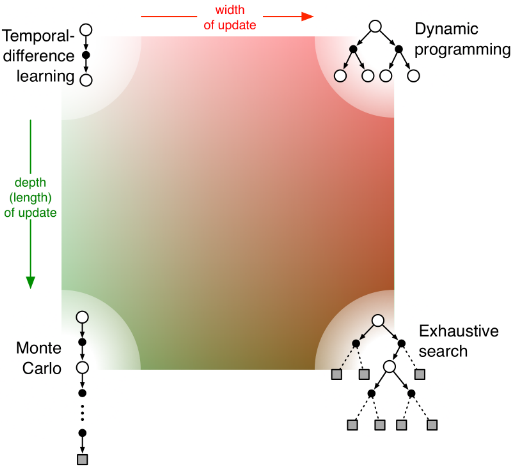

Figure: Four families of RL algorithms

Monte Carlo methods

- Sample many episodes
- MC every-visit backup: $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$
- Does not need environment model

TD methods

- Combines Monte Carlo and Dynamic Programming
- Does not need environment model
- Uses bootstrapping for updates
- Sample many steps instead of methods
- One-step TD backup: $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} - \gamma V(S_{t+1}) - V(S_t)]$

From 1-step to *n*-step

Define multi-step return for TD:
$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$
Using this multi-step return:

- Monte Carlo backup uses $G_{t:T}$
- One-step TD backup uses $G_{t:t+1}$

n-step TD: $V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t)]$
$Q(\sigma)$ is based on *n*-step Sarsa and *n*-step Tree Backup

Backups: From one-step TD to MC

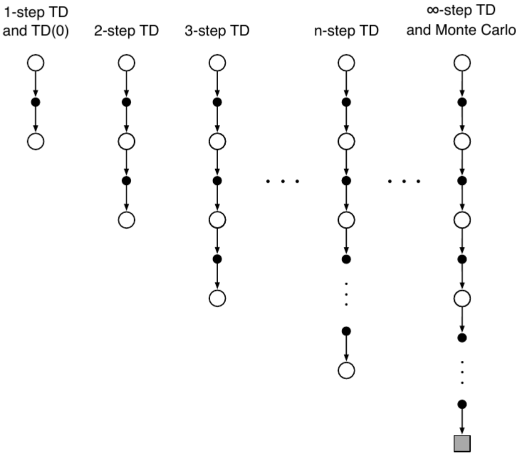

Figure: Backup schemes comparison: from 1 step in TD(0) to ∞ steps in Monte Carlo methods

n-step Sarsa

Transition from one-step Sarsa to *n*-step version is straightforward:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

The update rule is transformed as follows:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)$$

Generalizing Expected Sarsa would modify the error definition by a small margin:

$$\delta_t^S = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

$$\delta_t^{ES} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a) - Q_{t-1}(S_t, A_t)$$

Therefore we obtain Expected Sarsa return:

$$G_{t:t+n} \doteq R_{t+1} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a|S_{t+n}) Q_{t+n-1}(S_{t+n}, A_{t+n})$$

Sarsa off-policy learning

- π is greedy policy for the current action-value-function estimate
- μ is exploratory policy, perhaps ε-greedy

Goal: learn value function for π while following μ
Solution: use importance sampling ratio and modify update rule

$$\rho_{t:t+n} = \prod_{k=t}^{\min t+n, T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}$$

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha\rho_{t+1:t+n-1}[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$

n-step Sarsa and Expected Sarsa backup mechanisms

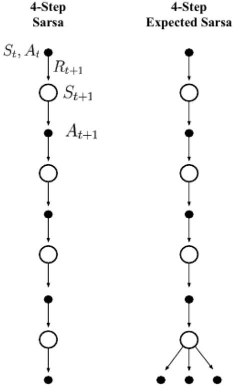

Figure: Backup schemes for Sarsa and Expected Sarsa with $n = 4$. The last step of Expected Sarsa makes an update based on pure expectation rather than on full sampling.

Tree Backup: off-policy without Importance Sampling

Tree Backup is the multi-step generalization of Expected Sarsa:

$$G_{t:t+1} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a) = \delta_t^{ES} + Q_{t-1}(S_{t+1}, a)$$

Hence *n*-step return of Tree Backup is a sum of TD errors:

$$\begin{aligned} G_{t:t+n} &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_t(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n} \\ &= \delta_t^{ES} + Q_{t-1}(S_t, A_t) - \gamma \pi(A_{t+1}|S_{t+1}) Q_t(S_{t+1}, A_{t+1}) + \\ &+ \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n} \\ &= Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min t+n-1, T-1} \delta_k^{ES} \prod_{i=t+1}^{k} \gamma \pi(A_i|S_i) \end{aligned}$$

Tree Backup is obtained after taking update rule from *n*-step Sarsa:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha [G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$

Tree Backup mechanism

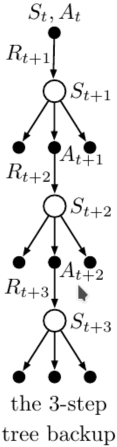

Figure: Tree Backup calculates the expected return of each step using δ_t^{ES} at each timestep t, i.e. using pure expectation rather than full sampling

Relation to other algorithms

Two families of multi-step algorithms:

- Algorithms that backup their actions and samples (Sarsa and Expected Sarsa)
- Algorithms that consider an expectation over all actions in their backup (Expected Sarsa and Tree Backup)

New parameter $\sigma \in [0, 1]$, which controls the degree of sampling at each step of the backup, unifies both families

Details

Error modification:

$$\begin{aligned}\delta_t^\sigma &= \sigma_{t+1}\delta_t^S + (1-\sigma_{t+1})\delta_t^{ES} \\ &= R_{t+1} + \gamma[\sigma_{t+1}Q_t(S_{t+1}, A_{t+1}) + (1-\sigma_{t+1})V_{t+1}] - Q_{t-1}(S_t, A_t)\end{aligned}$$

Resulting return:

$$G_t^{(n)} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min t+n-1, T-1} \delta_k^\sigma \prod_{i=t+1}^{k} \gamma[(1-\sigma_i)\pi(A_i|S) + \sigma_i]$$

Backup comparisons

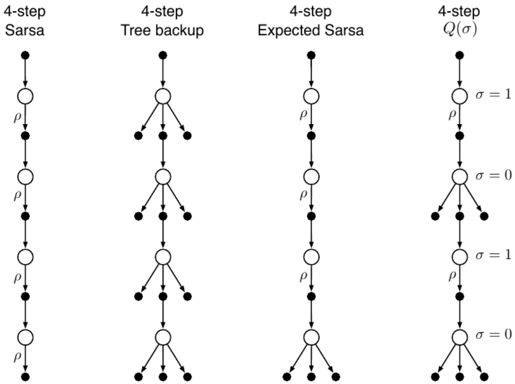

Figure: Backup mechanisms for the presented atomic multi-step algorithms with $n = 4$. $Q(\sigma)$ combines backup techniques of the previous algorithms. ρ indicates that corresponding transition requires importance sampling in off-policy learning.

$Q(\sigma)$ off-policy learning

- Both base algorithms can be trained off-policy
- $Q(\sigma)$ importance sampling for off-policy learning combines the off-policy learning ideas for base algorithms

Learning policy π while following μ can be achieved via introducing appropriate importance ratio:

$$\rho_{t+1:t+n} = \prod_{k=t+1}^{\min t+n-1, T-1} \left(\sigma_k \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} + 1 - \sigma_k\right)$$

Algorithm description

Initialize $S_0 \neq S_T$; select A_0 according to $\pi(.|S_0)$
Store S_0, A_0 and $Q(S_0, A_0)$
for $t \leftarrow 0, T + n - 1$ **do**
  if $t < T$ **then**
    Take action A_t, get R, observe and store S_{t+1}
    if S_{t+1} is S_T **then**
      Store $\delta_t^\sigma = R - Q(S_t, A_t)$
    else
      Calculate and store δ_t^σ
      Select and store A_{t+1} according to $\pi(.|S_{t+1})$
      Store $Q(S_{t+1}, A_{t+1})$, σ_{t+1}, $\pi(A_{t+1}|S_{t+1})$
      Store $\pi(A_{t+1}, S_{t+1})$ as π_{t+1}
    end if
  end if
  if $t \geq n$ **then**
    Calculate and store $G_t^{(n)}$
    Perform backup: $Q(S_t, A_t) \leftarrow S(_t, A_t) + \alpha[G_t^{(n)} - Q(S_t, A_t)]$
  end if
end for

Calculate $G_t^{(n)}$

$Z = 1$
$G = Q_t$
for $k = \tau, \dots, \min(\tau + n - 1, T - 1)$ **do**
    $G = G + Z\delta_k$
    $Z = \gamma Z[(1 - \sigma_{k+1})\pi_{k+1} + \sigma_{k+1}]$
end for

Comparing Sarsa, Tree Backup, $Q(0.5)$ and dynamic σ performance

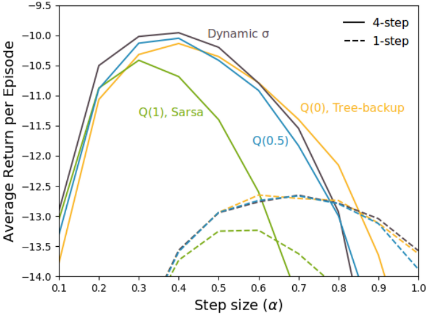

Figure: Stochastic Windy Gridworld environment experiments. Results of the algorithms are averaged after 100 returns. $Q(\sigma)$ performed the best overall.

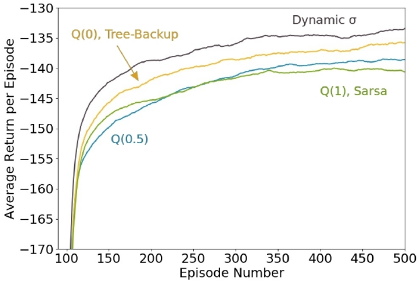

Figure: Mountain cliff results. The plot shows that dynamic σ choosing strategy showed the best overall performance.

Room for improvements

- Current definition of $Q(\sigma)$ is limited to the atomic multi-step case without eligibility traces, but it can be extended to use eligibility traces and compound backups
- Performance of $Q(\sigma)$ was only evaluated on on-policy problems, doing experiments with off-policy problems might be interesting

Better σ choosing

A trivial example of such strategies:

- Constant $\sigma = C$
- Altering $\sigma(t) = 1, 0, 1, 0, \ldots = [t \mod 2 = 0]$
- Random $\sigma(t) \sim \mathcal{U}[0, 1]$
- Decreasing or increasing over t (between 0 and 1)

Other schemes for dynamically varying σ could be investigated, for example a function of state, recently observed rewards or some measure of the learning progress.

Synopsis

Today we

- saw how n-step algorithms are derived from MC and one-step TD methods
- revised through Sarsa and Tree Backup
- were introduced to the $Q(\sigma)$ algorithm in MDP setting
- outlined the relation of presented algorithm to other n-step methods
- gained some intuition about algorithm structure
- compared $Q(\sigma)$ performance to its base algorithms
- thought about how the algorithm can be improved in the future

References

Kristopher De Asis, J. Fernando Hernandez-Garcia, G. Zacharias Holland, Richard S. Sutton.
Multi-step Reinforcement Learning: A Unifying Algorithm.
arXiv, 3 Mar 2017.

Richard S. Sutton, Andrew G. Barto.
Reinforcement Learning: An Introduction.
MIT Press, Cambridge, MA, 5 November 2017, 2nd Edition Complete Draft.

Materials

Presentation, code and other materials are available in the GitHub repository