MACHINE INTELLIGENCE COMMUNITY

Summary of Reinforcement Learning: An Introduction Chapter 6: Temporal-Difference Learning

Justin Chen* chenjus@bu.edu
Tyrone Hou* tyroneh@bu.edu

January 18, 2018

Introduction

- **Temporal-difference (TD) learning** - Combination of Monte Carlo and dynamic programming
  - Learn directly from expectations without a model like Monte Carlo methods
  - Update estimates based on other learned estimates without waiting for a final outcome (bootstrap)
- **Goal**:
  - How to apply temporal-difference learning
- **Prediction Problem (Policy Evaluation)** - Estimating the value function v_π for a given policy π
- **Control Problem (Find Optimal Policy)** - DP, TD, and Monte Carlo all use variation of generalized policy iteration
- DP, TD, and Monte Carlo methods differ at prediction problem

6.1 Prediction

- TD methods update estimate V of v_π each time step of episode using R_{t+1} and estimate of $V(S_{t+1})$
- **TD(0) (one-step TD)** - When the update target for TD learning is $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow +\alpha\big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big]$$

Tabular TD(0) for estimating v_π
Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., V(s)=0, for all $s \in S^+$)
Repeat (for each episode):
Initialize S
Repeat (for each episode):
$A \leftarrow action given by \pi for S$
Take action A, observe R, S'
$V(s) \leftarrow V(s) + \alpha[R + \gamma V(S') - V(S)]$
$S \leftarrow S'$
until S is terminal TD(λ) denotes **n-step TD methods**

*These authors contributed equally to this work

- **constant-αMC** - Every-visit Monte Carlo method with a constant step-size α for nonstationary environments
- Monte Carlo method update is G_t

$$V(S_t) \leftarrow V(S_t) + \alpha\big[G_t - V(S_t)\big]$$

  - Only updates at the end of the episode

- **Bootstrapping** - When estimates are based partially on existing estimates e.g.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \end{aligned}$$

- Methods and their targets:
  - Monte Carlo methods: $\mathbb{E}_\pi[G_t|S_t = s]$
  - Dynamic Programming: $\mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$
  - Temporal-Difference: $R_{t+1} + \gamma V(S_{t+1})$
- TD methods combine sampling of Monte Carlo with bootstrapping of DP
- **Sample Updates** - Updates that look ahead at next possible states or state-action pairs and use estimated look ahead state or reward information to update its current state or state-action pair value
  - Unlike DP expected updates, sample updates use only one sample successor instead of a complete distribution of all possible successors.
- **TD Error** - Difference between estimated value of S_t and $R_{t+1} + \gamma V(S_{t+1})$, $\delta \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
  - Represents the error at the current timestep
  - V is an array of values
- *If the value array V does not change during the entire episode* then the **Monte Carlo error** can be expressed as:

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+1} - V(S_{t+2})) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \ldots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-1}(G_T - V(S_t)) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \ldots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-1}(0 - 0) \\ &= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k \end{aligned}$$

6.2 Advantages of TD Prediction Methods

- Does not require model of environment, reward, state-transition probability
- Can be implemented on-line and incrementally to do updates *during* interaction with environment. For long episodes this speeds up learning considerably.
- Learns from all state transitions regardless of whether the action is exploratory or exploitative. (Some MC methods only learn from greedy actions, slowing learning)

- Importantly, still converges to v_π under certain stochastic approximation conditions on the step-size parameter α

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

- *Which converges faster?* This is an open question, but in practice TD methods have been found to converge faster than MC methods with constant *alpha*.

6.3 Optimality of TD(0)

- What happens if we do not have an abundance of experience?
- If the step size parameter is less than 1, we can present the same experience multiple times to improve the value estimate. Typically we batch the data to do updates
- **batch** - group/subset of training data where all updates in the batch occur simultaneously
- **batch updating** - updating that occurs using the sum of all updates for each episode/experience in the batch
- Constant α MC methods and TD methods actually converge to different estimates due to differences in how they estimate expected return during updates
- **Monte Carlo**
  - Estimates expected returns using sample returns G_k whose expected value is v_π.
  - Value Estimate converges to *sample average* of returns.
  - Minimizes the mean squared error on training set
- TD
  - Models expected return as $R_t + \gamma v_\pi(s')$.
  - Implicitly computes *maximum-likelihood estimate* for an MDP. This is the parameter value that are most likely to have generated the observed data.
  - For an MDP, the maximum likelihood estimate model computes transition probabilities from state A to state B as the fraction of transitions from A that led to B.
  - **certainty-equivalence estimate** - Value function estimate computed as if the estimated model of the underlying process (the MDP) were correct. Does not account for errors in the model.
  - TD(0) in general converges to the certainty-equivalence estimate.
- TD(0) can be done in an *off-policy* manner using importance sampling, where we let the update rule using ordinary importance sampling at time step t be

$$V_{t+1}(s) = V_t(s) + \alpha \rho_{t:T(t)-1}[R_{t+1} + \gamma V_t(s') - V_t(s)]$$

6.4 Sarsa: On-policy TD Control

- generalized policy iteration, with TD methods for policy evaluation
- Since we may not have access to a model, we learn *action values* in a manner similar to TD(0) above
- Since we use $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ in the update rule below, the algorithm is called **Sarsa**
- **Sarsa update rule**

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

- **Sarsa backup diagram**

Sarsa (on-policy TD control) for estimating $Q \approx q*$
For all $s \in \mathcal{S}, a \in mathcal A$, initialize $Q(s,a)$ arbitrarily, and Q(*terminal-state*, $\cdot$) $= 0$
Let π be a policy derived from Q (e.g. e-greedy)
Repeat (for each episode):
Initialize S
$A \leftarrow pi(S)$
Repeat (for each episode step):
Take action A, observe R, S'
$A' \leftarrow S'$
$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
$S \leftarrow S'; A \leftarrow A';$
until S is terminal

- **Convergence** - Sarsa converges to $q*$ as long as 1) all state-action pairs are visited infinitely (e.g. with ϵ-greedy or ϵ-soft policies) and 2) the policy converges in the limit to *greedy*.
- Another advantage of TD learning over Monte Carlo is that they never get stuck without terminating. Because Monte Carlo methods learn *between* episodes, if they ever reach an infinite state loop, they are less likely to break out, even with ϵ-greedy methods. TD methods will quickly learn a particular route/action sequence is bad, and switch to something else.

6.5 Q-learning: Off-policy TD Control

- **Q-learning** - An **off-policy** TD control method
- **Q-learning update rule**

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \underset{a}{max} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

- The target value function Q approximates q_* regardless of the behavior policy generating episodes
- **Convergence** - Q-learning converges to q_* given that all state-action pairs are visited infinitely and the same conditions for step size parameter as TD (see above)

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$
For all $s \in \mathcal{S}, a \in mathcal A$, initialize $Q(s,a)$ arbitrarily, and Q(*terminal-state*, $\cdot$) $= 0$
Repeat (for each episode):
Initialize S
repeat (for each step of episode):
Choose A from S using policy derived from Q (e.g., ϵ-greedy)
Take action A, observe R, S'
$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \underset{a}{max} Q(S',a) - Q(S,A)]$ $S \leftarrow S'$ until S is terminal

- Learning the greedy policy can actually be less optimal if the greedy path is "riskier", since ϵ-greedy action selection can result in actions that produce a large negative reward.

6.6 Expected Sarsa

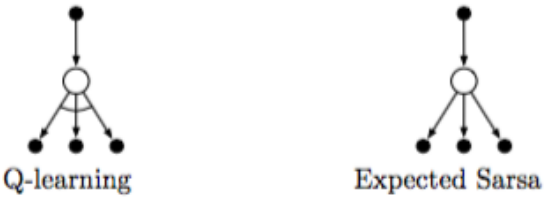
- **Expected Sarsa** is like Q-learning, except that instead of the maximum over actions, it uses the *expected value* according the probability of taking each action giving the policy.
- **Expected Sarsa update rule**

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_t + \gamma \mathbf{E}[Q(S_{t+1}, A_{t+1}) | S_{t+1}] - Q(S_t, A_t) \Big] \\ &\leftarrow Q(S_t, A_t) + \alpha \Big[R_t + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big] \end{aligned}$$

- Expected Sarsa moves *deterministically* in the direction Sarsa moves *in expectation*.
- Because it does an expected update, expected Sarsa requires more computation, but has less variance than Sarsa.
- Expected Sarsa can be done on-policy or off-policy. In the off-policy case, we simply use a different behavior policy to generate episodes, then do an expected update on target policy π.
- Q-learning is a *special case* of off-policy expected Sarsa where the target policy is greedy.

6.7 Maximization Bias and Double Learning

- Implicit maximum over estimated values as an estimate for the maximum value can lead to significant positive bias
- **Maximization Bias** - Maximum value over estimator with significant variance can bias to a value that is greater than the actual value
  - Consider a situation where the true values of all actions are zero, but $q(s, a)$ estimate has some variance such that the range of it's estimates are $[\mu - \sigma, \mu + \sigma]$, where $\mu = 0$ then the maximum estimated value is σ instead of zero.
  - This can cause the agent to select suboptimal actions which can make prolong learning optimal behavior
  - Algorithms such as Double Q-Learning are unaffected by this
- Are there algorithms that avoid maximization bias?
- Problem is caused by using the same samples for both determining the maximizing action and estimating that actions value
  - Could address this by separating both functionalities
  - Learn two independent estimates, $Q_1(a)$ and $Q_2(a)$, of the true value $q(a) \forall a \in \mathcal{A}$
  - $Q_1(a)$ determines the maximizing action $A^* = \underset{a}{argmax} Q_1(a)$
  - Q_2 estimate the value of the maximizing action $Q_2(\underset{a}{argmax} Q_1(a))$
  - Q_2 is unbiased because $\mathbb{E}[Q_2(A^*)] = q(A^*)$
  - **Double Learning** - Compute either $Q_2(\underset{a}{argmax} Q_1(a))$ or $Q_1(\underset{a}{argmax} Q_2(a))$ each iteration
    - Doubles memory requirement
    - Same computation per step as typical approach
  - Estimates learned with double learning can be used as values in the usual setting
  - Double learning can be applied to MDPs



Backup diagrams for Q-learning and expected Sarsa

- Double learning variations: **Double Sarsa**, **Double Expected Sarsa**, and **Double Q-learning**

Double Q-learning
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}$, arbitrarily
Initialize $Q_1(terminal-state,\dot{)} = Q_2(terminal,\dot{)} = 0$
Repeat (for each episode):
Initialize S
Repeat (for each step of episode):
Choose A and S using policy derived from Q_1 and Q_2 (e.g., ϵ-greedy in $Q_1 + Q_2$)
Take action A, observe R, S'
With 0.5 probability:

$$Q_1(S_t,A_t) \leftarrow Q_1(S_t,A_t) + \alpha\big[R_{t+1} + \gamma Q_2(S_{t+1}, \underset{a}{argmax} Q_1(S_{t+1},a)) - Q_1(S_t,A_t)\big]$$

$else$:

$$Q_2(S_t,A_t) \leftarrow Q_2(S_t,A_t) + \alpha\big[R_{t+1} + \gamma Q_2(S_{t+1}, \underset{a}{argmax} Q_2(S_{t+1},a)) - Q_2(S_t,A_t)\big]$$

$S \leftarrow S'$
until S is terminal

6.8 Games, Afterstates, and Other Special Cases

- Consider a tic-tac-toe game framed as an RL problem as in chapter 1, where the states are possible board states and actions are placing checkers.
- We can estimate the value of states using something like the action-value estimations:

$$V(S_t) = V(S_t) + \alpha[V(S_{t+1}) - V(S_t)]$$

- Because we backup the value of state *after* having taking an action to reach a new state, we call this an **afterstate value function**, and the states looked at (S_t in the above case) are actually called *afterstates*.
- The advantages of afterstates over action values is that many actions may result in the same afterstate. These actions would have the same value as the afterstate, and would be redundantly stored in Q.
- However learning from any actions leading to the same afterstate can be *aggregated* in an after-state value function.
- Afterstates occur in any scenario where actions lead to *redundant states*, and their values can be estimated with GPI.

6.9 Summary

- temporal-difference (TD) learning is an alternative to Monte Carlo methods for estimating value functions. Works inside of GPI framework.

- Prediction problem - using GPI's value function to accurately predict returns for the current policy
- Difficult to maintain sufficient exploration as GPI's other policy attempts to improve w.r.t. current value function
- Sarsa maintain sufficient exploration
- Q-learning does not address maintaining sufficient exploration
- TD methods are able to perform *incremental, on-line learning* that updates through experience generated by an environment.
- They can be extended to take into account multiple steps into the future and use function approximation methods.
- TD methods are *general methods* for making long-time predictions about dynamical systems.

References

Sutton, Richard S., and Andrew G. Barto. "Temporal-Difference Learning." Reinforcement Learning: An Introduction, The MIT Press, 2018, pp. 97-114.