Temporal Difference Learning

Thursday, June 18, 2020 6:27 PM

Temporal Difference (TD) learning is a combination of Monte Carlo (MC) ideas and Dynamic Programming (DP) ideas
Policy evaluation is also called the prediction problem and finding an optimal policy (policy improvement) is also called the control problem
Constant-α MC:
$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$
Where $V(S_t)$ is the value of state S_t, G_t is the return following time t and α is a constant step-size parameter
Note that this must wait until the end of the episode to update the value since the return can't be known until then
One Step Temporal Difference TD(0):
$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
This can be considered as an extension of constant-α MC with the complete return replaced by a target that is the received reward and discounted estimate of the value of the next state (their current state when the update is performed)
Note that temporal difference methods update past states based on the current estimate of future states so they are bootstrapping methods
Comparison of MC, DP, and TD Methods:
Recall:
$v_\pi(s) \equiv E_\pi[G_t | S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$
MC methods are an estimate of the first form since the true expectation value is not known, only sampled experience
DP methods update an estimate of $v_\pi(S_{t+1})$ using a current estimate until it approaches the true value
TD methods estimate both the expectation value and using a current estimate so in a way they are a combination of MC and DP methods, combining the experience sampling of MC with the bootstrapping of DP methods
MC and TD methods are called sample updates since they use sample successor states and the reward obtained in reaching that sample state to compute a backed up value and update the original state accordingly
DP methods are called expected updates since they are based on a complete distribution of all possible successor states, not a single sample successor
TD Error:
$\delta_t \equiv R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
Note that this is the difference between an updated estimate of the value of a state $R_{t+1} + \gamma V(S_{t+1})$ (updated since it includes the reward obtained in reaching it) and the old estimate of the value of that state $V(S_{t+1})$
MC Error as Sum of TD Errors:
$$G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k$$
This assumes that $V(S_t)$ is not updated during the episode (the MC method is not incremental)
Action-Value Form:
$\delta_t \equiv R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$
TD methods usually converge faster than constant-α MC methods in practice
Markov Reward Process (MRP):

A Markov Decision Process with the actions neglected

Batch Updating:

Presenting an agent with a finite amount of experience (a batch) and calculating all increments, but not applying the increments to the value function until after the batch has been completely processed

Under batch updating constant-α MC and TD methods converge to different values

Batch constant-α MC methods converge to values that are sample averages of the actual returns experienced, which minimizes the mean-squared error from the actual returns in the training set

Batch TD(0) finds the maximum-likelihood estimate of the Markov process

In general batch TD(0) converges to the certainty-equivalence estimate

Note that without batch updating the methods do not achieve either the certainty-equivalence or minimum squared error estimates

Maximum-Likelihood Estimate:

The estimate of a parameter that has the value whose probability of generating observed data is the greatest

Certainty-Equivalence Estimate:

The estimate of the value function that would be exactly correct if the model was exactly correct

This is named the certainty-equivalence estimate since it is equivalent to assuming that the estimate of the underlying process is known with certainty rather than being approximated

Sarsa (State-Action-Reward-State-Action):

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

Where if S_{t+1} is terminal then $Q(S_{t+1}, A_{t+1})$ is defined to be 0

Note that this is an on-policy method that learns the values of state-action pairs in the same manner that TD(0) learns the values of states

The name for this method comes from making use of every element of the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

This converges to an optimal policy as long as all state-action pairs are visited an infinite number of times as the number of time steps goes to infinity and the policy converges to the greedy policy in the limit

One example that meets both criteria is using an ε-greedy policy with $\varepsilon = \frac{1}{t}$

Note that updates here cannot be performed until after the action in the next state is chosen (to get $Q(S_{t+1}, A_{t+1})$

Q-Learning:

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

Where $Q(terminal, :)$ is defined to be 0

This is an off-policy TD control algorithm

Note that this converges to q_* (the true action-value function) independent of the policy being followed (the behavior policy) although the behavior policy still affects which states are visited and updated

The behavior policy used is often ε-greedy

Expected Sarsa:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma E_\pi\left[Q(S_{t+1}, A_{t+1}) \middle| S_{t+1}\right] - Q(S_t, A_t)\right]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)$$

Note that these are equivalent forms, not different steps in the algorithm

The name comes from this algorithm moving deterministically in the same direction Sarsa moves in expectation

This consistently performs better than Sarsa and subsumes Q-learning

Maximization Bias:

The bias resulting from using the maximum of estimated values as the maximum of true values (if any estimate is higher than the maximum true value then the maximum of the estimates is biased positively)

Double Learning:

Learning two independent estimates of the values of actions then using one to determine the maximizing action and the other to estimate its value

Equations:

$A^* = \underset{a}{\operatorname{argmax}}\, Q_1(a)$

$Q(a) = Q_2(A^*)$

This is unbiased (i.e. $E[Q_2(A^*)] = q(A^*)$ where $q(A^*)$ is the true value)

Note that which estimate is 1 and which is 2 should be chosen randomly on each step

Double Q-Learning:

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

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

Only one of the above operations is done per step (determined perhaps randomly or another way that treats the two value functions symmetrically)

This is an off-policy approach and the behavior policy can use both action-value estimates in determining what action to take (perhaps an ε-greedy policy based on the sum of the two action-value estimates)

Note that double learning can be applied to other algorithms

Afterstate:

States evaluated after an agent has visited and left them (e.g. after an episode finishes)

Afterstate Value Functions:

Value functions over afterstates