Temporal-Difference Learning (TD) is the integration of Monte Carlo methods and Dynamic Programming, combining the advantages of both learning algorithms. TD can learn directly through experiences without any knowledges of the environment just like Monte Carlo, and TD stores the learning values for future episodes just like Dynamic Programming. Though, there are some diversity between them. For example, Monte Carlo requires to update the estimated value at the end of episode, while TD updates the estimated value right at the next time step. The updating of TD raises a question to consider: Since TD updates values based on a previous prediction, without the final result, does TD maintains accuracy to compute the optimal answer? The answer is yes, that introduces to a branch of TD called "TD (0)". TD (0) has the following update rule:

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

where $R_{t+1} + \gamma V(S_{t+1})$ denotes the target of updating, coming from the next run. This equation is proved to be converging to v_π.

TD can also be used for control problems, using the idea of Generalized Policy Iteration from Dynamic Programming. Instead of evaluating state values, action value is computed. An on-policy approach is with the algorithm "SARSA", defined as changing the above equation from V(S) into Q(S, A). The new equation uses every element in the following set $\{ S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1} \}$ and that is the origin of the name coming from. An off-policy approach is with the algorithm "Q-Learning", defined as changing $\gamma Q(S_{t+1}, A_{t+1})$ from SARSA into $\gamma \max_a Q(S_{t+1}, a)$. Q-Learning ignores policy and directly computes q_* with the action-value function. Another off-policy algorithm is expected SARSA, where an expected value of $Q(S_{t+1}, A_{t+1})$ is used instead; by doing so, the constraint of small α value from SARSA is lifted.

However, both SARSA and Q-Learning form a positive bias, since the maximum values in estimation are positive, if the true value is 0, such bias exists. To avoid such bias, one could use Double Q-Learning. Double Q-Learning divides one time step into two. In terms of algorithm, this replaces:

$Q(S_t, A_t) \mathrel{+}= \alpha[R_{t+1} + \underline{\gamma \max_a Q(S_{t+1}, a)} - V(S_t)]$ into $\underline{\gamma Q_2(S_{t+1}, \operatorname{argmax}_a Q(S_{t+1}, a))}$

with all other Q as Q_1. This update rule is used for winning cases. For losing cases, we could simply switch Q_1 with Q_2 vice versa.