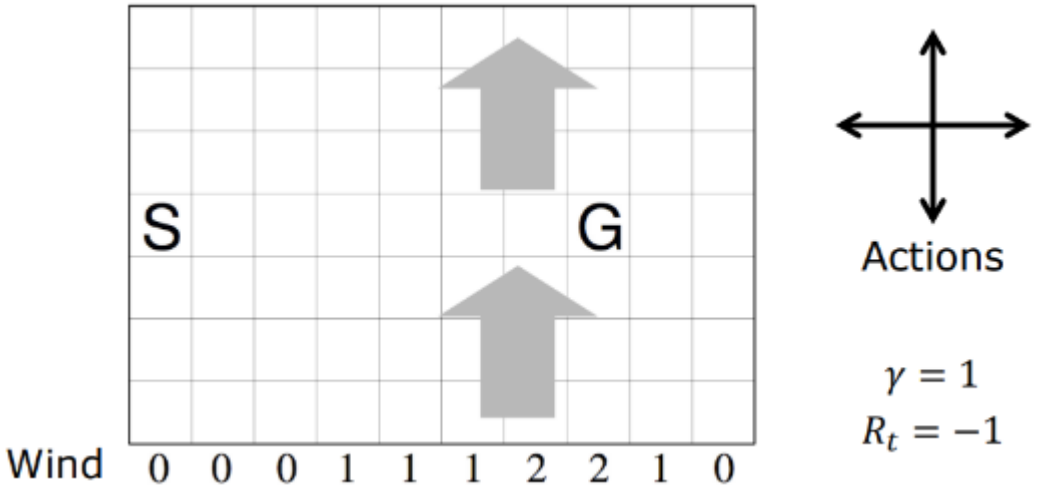
Temporal Difference Learning

Last family of algorithm to solve policy evaluations and policy improvement. We are in a similar case than Monte Carlo. Differently from Dynamic Programming we assume that we learn from experience. It tries to solve the needing of observing entire episodes. To compute the empirical averages we need a series of returns and each reach the end solution(two drawbacks: can be applied only to episodic task, if I am not in terminal state and cannot say that I can return and even if I am in an episodic situation finding the way of return ca be very difficult).



Windy table where the goal is to reach G in the least number of steps. The wind value represent how much I will be moved up in the environment in that column. With Monte Carlo at the start I would use a random policy but that would imply that I cannot reach G denying the learning process. With Monte Carlo the learning at the state will be very bad.

Temporal-Difference Policy Evaluation – TD(0)

Combine the DP idea that you divide the problem in immediate reward and the discounted value function in another state (bootstrapping)

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

$$G_t \approx R_{t+1} + \gamma V(S_{t+1})$$

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

Temporal-Difference Error (δ_t)

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

We use a sample average on the return but in this case G is not a complete return as it is decomposed in immediate return and discounted value.
TD can learn before knowing the final outcome

- TD can learn after every step
- MC must wait until end of episode before return is known
  TD can learn without the final outcome
- TD can learn from incomplete sequences
- MC can only learn form complete sequences
  TD works in continuing (non–terminating) environments MC only works for episodic (terminating) environments

- MC target has lower bias
  - MC return $R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_{t+T}$ is an **unbiased** estimate of $V_\pi(S_t)$
  - TD target $R_{t+1} + \gamma V(S_{t+1})$ is a **biased** estimate of $V_\pi(S_t)$, as $V(S_{t+1}) \neq V_\pi(S_{t+1})$
- TD target has lower variance
  - Return depends on many random actions, transitions, rewards
  - TD target depends on one random action, transition, reward
- Overall
  - MC works well with function approximation and it is not very sensitive to initial values
  - TD has problem with function approximation and it is more sensitive to initial values

Bootstrapping: update involves an estimate

- MC does not bootstrap(doesn't require the decomposition of the value function, not using Bellman equation, works on non Markovian problems)
- DP bootstraps
- TD bootstraps(When I train the algorithm my knowledge of my current value function I decompose it in two parts: immediate rewards and discounted value functions)
  Sampling: update does not involve an expected value

- MC samples
- DP does not sample(Need to know the 1-step ahead procedure)
- TD samples

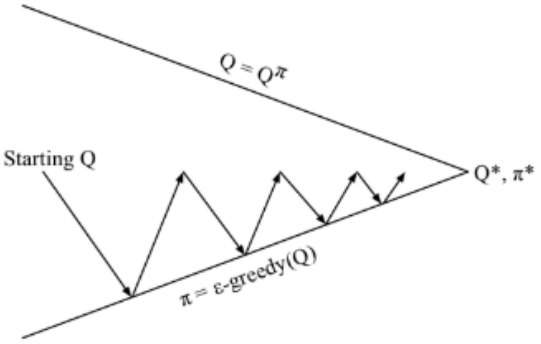
SARSA

It is the equivalent of Monte Carlo policy iteration in Temporal Difference Learning.

On–Policy Control with SARSA

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

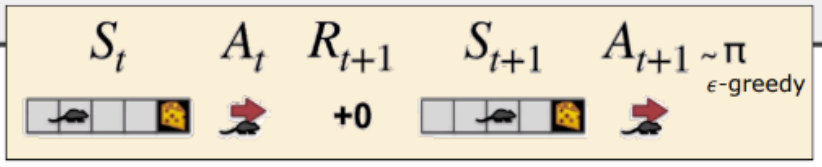
- Evaluation
  - SARSA
- Improvement
  - ε-Greedy Policy Improvement



The idea is to leverage the idea of decomposing the expected return as immediate rewards plus discounted value function in the next time value. I take in consideration current value and the TD error(target minus current Q). SARSA derives from the elements in the update rule

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma Q(S', A') - Q(S, A)\right]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal



SARSA is an on policy algorithm(the policy that I'm using to generate data is the same that I am trying to learn) and the policy is an ϵ-greedy policy(still a good policy to resolve the problem with a non policy approach).

Q-Learning

Q-Learning, as Value Iteration in DP, is based on Bellman Optimality Equation

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

$$Q^*(s, a) = \sum_{s', r} p(s', r|s, a) \left(r + \gamma \max_{a'} Q^*(s', a') \right)$$

Use Bellman Optimality Equation to create the policy. The next action will depend on the policy I am using to explore the problem.

Q-Learning: Off-Policy TD Control

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A) \right]$
    $S \leftarrow S'$
  until S is terminal

The update rule now the best action according to my knowledge in the next state.
The exploration is now free from the goodness of my results as they are independent but still lead to an optimal policy. Remind to not use random policy for exploration, try to make it limited and gives priority to the most promising moves.
SARSA is better than Q-Learning when there are problems where the movement is equal in each part of the track than a limited patch where the movement is really penalised.
SARSA might take into account the worst case scenario in its analysis.

Eligibility traces

Instead of choosing TD(0) or Monte Carl I try to compute all the possible definition of target and average them. In practice by introducing the eligibility traces parameter that is waiting all the

observed return while I am learning.