On-policy Monte Carlo control

evaluating $Q_\pi(s,a)$

ϵ - greedy π

$$\pi(s) = \begin{cases} \arg\max_a Q(s,a) & \text{w.p. } 1-\epsilon \\ \text{random} & \epsilon \end{cases}$$

Algorithm 4 On-policy Monte Carlo control

```
1:  Initialise $Q$ and $\pi$ arbitrarily
2:  $Returns(s, a) \leftarrow$ empty list $\forall s \in \mathcal{S}, a \in \mathcal{A}$
3:  **repeat**
4:      **for** $s \in \mathcal{S}$ and $a \in \mathcal{A}$ **do**
5:          Generate an episode using $\epsilon$-greedy $\pi$ starting with $\underline{s, a}$
6:          **for** $(s, a)$ in the episode **do**
7:              $Returns(s, a) \leftarrow$ append return following $s, a$
8:              $Q(s, a) = average(Returns(s, a))$
9:          **end for**
10:         **for** $\boxed{s}$ in the episode **do**
11:             $\underline{\pi(s) = \arg\max_a Q(s, a)}$
12:         **end for**
13:     **end for**
14: **until** convergence
```

evaluate & improve policy
$Q^\pi(s,a)$ $\pi(s)$

Monte Carlo off-policy methods

In off-policy methods we have two policies

target policy
: the policy being learned

- The target policy is the greedy policy with respect to Q.

behaviour policy
: the policy that generates behaviour

- The behaviour policy must have a non-zero probability of selecting all actions that might be selected by the target policy (coverage).
- To insure this we require the behaviour policy to be soft (i.e., that it select all actions in all states with non-zero probability)
- The behaviour policy μ can be anything, but in order to assure convergence of π to the optimal policy, an infinite number of returns must be obtained for each pair of state and action.

Off-policy Monte Carlo control

Algorithm 5 Off-policy Monte Carlo control

```
 1: Initialise $Q$ arbitrarily, $C(s,a) = 0 \ \forall s \in \mathcal{S}, a \in \mathcal{A}$ $\pi \leftarrow$ greedy
    with respect to $Q$
 2: repeat
 3:     Generate an episode $[s_0, a_0, \ldots, a_{T-1}, s_T]$ using soft policy $\mu$
 4:     $R \leftarrow 0, W \leftarrow 1$
 5:     for $t = T$ down-to 0 do
 6:         $R \leftarrow \gamma R + r_{t+1}$
 7:         $C(s_t, a_t) \leftarrow C(s_t, a_t) + W$
 8:         $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{w}{C(s_t,a_t)} (R - Q(s_t, a_t))$
 9:         $\pi(s) = \arg\max_a Q(s, a)$
10:         if  $a_t \neq \pi(s_t)$ then
11:             Exit for loop
12:         end if
13:         $W \leftarrow W \frac{1}{\mu(a_t,s_t)}$
14:     end for
15: until convergence
```

Recall Bellman Equation:

$$Q^{\mu}(s,a) = \mathbb{E}[r_{t+1}|S_t=s, a_t=a] + \sum_{s',r} V^{\mu}(s')\, p(s',r|s,a)$$
$$= \mathbb{E}[r_{t+1}|S_t=s, a_t=a] + \gamma \sum_{s',r} p(s',r|s,a) \sum_{a'\in A} \mu(a'|s')\, Q^{\mu}(s',a')$$

Now π is greedy w.r.t Q, therefore

$$Q^{\pi}(s,a) = \mathbb{E}[r_{t+1}|S_t=s, a_t=a] + \gamma \sum_{s',r} p(s',r|s,a) \max_{a'\in A} Q^{\pi}(s',a')$$

```
9:    $\pi(s) = \arg\max_a Q(s,a)$
```

line 9 of the code ensures that actions from $t \to T$ are the same for the behaviour policy μ and the greedy policy π.

Therefore, we know that

$$Q^{u}(S_t=s, a_t=a \mid a_{t+1}=a') = \overbrace{\mathbb{E}[r_{t+1}|S_t=s, a_t=a]}^{A}$$
$$+ \overbrace{\gamma \sum_{s',r} p(s',r|s,a)\; \mu(a_{t+1}=a'|S_{t+1}=s')\, Q^{u}(s',a')}^{B}$$

$$Q^{\pi}(S_t=s, a_t=a \mid a_{t+1}=a') = \overbrace{\mathbb{E}[r_{t+1}|S_t=s, a_t=a, a_{t+1}=a')}^{A}$$
$$+ \underbrace{\gamma \sum_{s',r} p(s',r|s,a)\, Q^{\pi}(s',a')}_{C}$$

Thus,

```
8:    $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \frac{W}{C(s_t,a_t)}(R - Q(s_t,a_t))$

13:   $W \leftarrow W \frac{1}{\mu(a_t,s_t)}$
```

Note that R is the current estimation of return under policy μ, which should be rescaled by $\mu(a_t, S_t)$ for an estimation of value function under π.

Summary

- **Model-based** vs **model-free** methods
- The Monte Carlo methods learn value functions and optimal policies from experience in the form of sample episodes.
- They do not require the model of the environment and can be learned directly in the interaction with the environment of in simulation.
- They simply average many returns for each state-action pair.
- **On-policy** vs **off-policy** methods.

.

Temporal-Difference Methods

In this lecture...

Introduction to temporal-difference learning

SARSA: On-policy TD control

Q-learning: Off-policy TD control

Planning and learning with tabular methods

Temporal-difference (TD) learning

Temporal-difference methods are similar to

Dynammic programming update estimates based in part on other learned estimates, without waiting for the final outcome (they bootstrap)

Monte Carlo methods learn directly from raw experience without a model of the environment's dynamics

TD prediction

$V(S_t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \cdots -$

$V(S_{t+1}) = r_{t+2} + \gamma \cdot r_{t+3} + \gamma^2 r_{t+4} \cdots -$

$r_{t+1} = V(S_t) - \gamma V(S_{t+1})$

- TD methods only wait until the next time step to update the value estimates.
- At time $t+1$ they immediately form a target and make an update using the observed reward r_{t+1} and the current estimate $V(s_{t+1})$.

what r_{t+1} should be if value fn is correct

backup :
$$V(s_t) \leftarrow V(s_t) + \alpha\left(r_{t+1} + \boxed{\gamma V(s_{t+1}) - V(s_t)}\right),$$

where $\alpha > 0$ is a step-size parameter.

- Note that this is similar to the MC update except that it takes place at every step.
- Similar to DP methods, the TD method bases its update in part on an existing estimate – a bootstrapping method.

TD error

TD error arises in various forms through-out reinforcement learning

$$\boxed{\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)}$$

The TD error at each time is the error in the estimate made at that time. Because the TD error at step t depends on the next state and next reward, it is not actually available until step $t+1$. Updating the value function with the TD-error is called a **backup**. The TD error is related to the Bellman equation.

$$V(S_t) \leftarrow V(S_t) + \alpha \, \delta_t$$

$$\delta_t = r_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

SARSA: On-policy TD control

state - action - reward - state - action

- TD prediction for control ie action-selection
- A generalised policy iteration method
- Balances between exploration and exploitation
- Learns tabular Q-function

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right)$$

This update is done after every transition from a non-terminal state s_t. If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1})$ is defined as zero. This rule uses every element of the quintuple of events, $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, hence the name.

SARSA: On-policy TD control

ε-greedy: $a = \arg\max_{a \in A} \hat{Q}(s,a)$ w.p. $1-\varepsilon$
random action w.p ε

Algorithm 1 SARSA

```
1:  Initialise $Q$ arbitrarily, $Q(terminal, \cdot) = 0$
2:  repeat
3:      Initialize s
4:      Choose $a$ $\epsilon$-greedily
5:      repeat
6:          Take action $a$, observe $r$, $s'$
7:          Choose $a'$ $\epsilon$-greedily
8:          $Q(s,a) \leftarrow Q(s,a) + \alpha\,(r + \gamma Q(s',a') - Q(s,a))$
9:          $s \leftarrow s'$, $a \leftarrow a'$
10:     until $s$ is terminal
11: until convergence
```

no policy is needed!
ϵ-greedily choose a'

1-step roll-out

ϵ-greedy π

Properties of SARSA

- SARSA is an **on-policy** algorithm which means that while learning the optimal policy it uses the current estimate of the optimal policy to generate the behaviour.
- SARSA converges to an **optimal policy** as long as all state-action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy ($\epsilon = \frac{1}{t}$).

Q-learning: Off-Policy TD Control

In Q-learning the learned action-value function, Q, directly approximates the optimal action-value function, independent of the policy being followed.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$$

This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs: all that is required for correct convergence is that all pairs continue to be updated.

Q-learning: Off-policy TD control

Algorithm 2 Q-learning

```
1:  Initialise $Q$ arbitrarily, $Q(terminal, \cdot) = 0$
2:  repeat
3:      Initialize s
4:      repeat
5:          Choose $a$ $\epsilon$-greedily
6:          Take action $a$, observe $r, s'$
7:          $Q(s,a) \leftarrow Q(s,a) + \alpha\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
8:          $s \leftarrow s'$
9:      until $s$ is terminal
10: until convergence
```

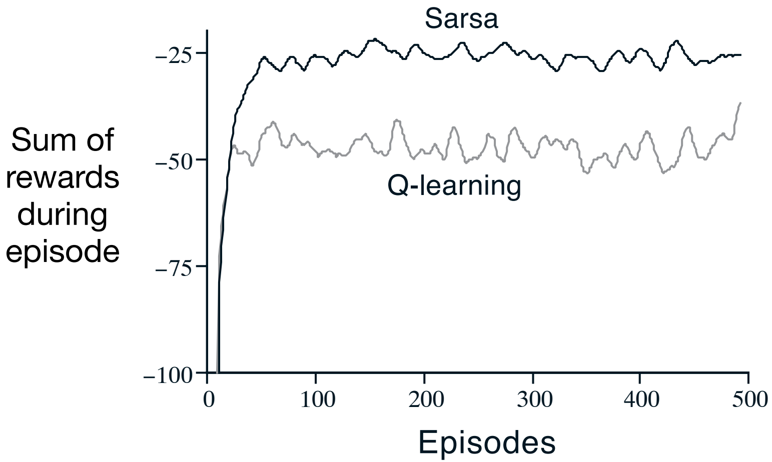
w.p. $1-\epsilon$ $a = \arg\max_a Q(s,a)$

w.p ϵ random a.

s' is generated.

SARSA vs Q-learning

Comparison of the SARSA and the Q-learning algorithm on the cliff-walking task (a variant of grid-world). The results show the advantage of on-policy methods during the learning process.



Expected Sarsa

- An alternative to taking a random action and using the estimate of the Q-function for that action in TD-error (as in SARSA) is to use the expected value of the Q-function.

$$\begin{aligned}Q(s_t, a_t) &\leftarrow Q(s_t, \overset{\text{random action}}{a_t}) + \alpha\left(E[Q(s_{t+1}, a_{t+1}) \mid s_{t+1}] - Q(s_t, a_t)\right) \\ &= Q(s_t, a_t) + \\ &\quad \alpha\left(r_{t+1} + \gamma \sum_{a'} \pi(a'|s_{t+1}) Q(s_{t+1}, a') - Q(s_t, a_t)\right)\end{aligned}$$

- Although computationally more complex, this method has a lower variance.
- Generally performs better and it can be either on-policy or off-policy.

Summary

- Prediction: the value function must accurately reflect the policy
- Improvement: the policy must improve locally (eg ϵ-greedy) with respect to the current value function
- SARSA is an on-policy TD method works for tabular setting
- Q-learning is an off-policy TD method works for tabular setting
- Expected SARSA can be either an on-policy or an off-policy method
- They can be applied on-line, with a minimal amount of computation, to learn from interaction with an environment