Reinforcement Learning
Continuous Policy Iteration

Tabular Reinforcement Learning provides a nice framework but doesn't scale for large environment (i.e $\mathcal{S} \gg 1$) and doesn't provide solutions for real-worl problems where the data-space is continuous.

How can we scale the methods that we saw for tabular MDPs to large or continuous state spaces ? One solution is to use function approximators to extrapolate knowledge from discrete, sampled data. Parametric approximation would lead us to estimate approximates of the value functions:

$$\begin{aligned} \hat{V}(s,w) &\simeq V_\pi(s) \qquad \forall s \in \mathcal{S} \\ \hat{Q}(s,a,w) &\simeq Q_\pi(s,a) \quad \forall s,a \in \mathcal{S} \times \mathcal{A} \end{aligned} \tag{1}$$

The most commons parametric approximation are linear combination of engineered features:

$$\hat{V}(s,w) = w^T \phi(s) \tag{2}$$

and neural networks (or learned features). Unlike classical supervised learning, we will also focus on training methods that are suitable for non-stationary, non-i.i.d data (typically generated from environment-agent interactions).

1 Incremental methods

In online methods, we will consider gradient-descent like algorithms to estimate :

$$w^* = \underset{w}{\operatorname{argmin}} \left\{ J(w) = \frac{1}{2}\mathbb{E}_\pi \left[\left(V_\pi(s) - \hat{V}(s,w) \right)^2 \right] \right\} \tag{3}$$

and therefore updates will take the form:

$$\begin{aligned} \Delta w &\propto -\nabla_w J(w) \\ &= -\mathbb{E}_\pi \left[\left(V_\pi(s) - \hat{V}(s,w) \right) \nabla_w \hat{V}(s,w) \right] \end{aligned} \tag{4}$$

Of course, because the sequential distribution under π is unavailable, we proceed by SGD, sampling over trajectories:

$$\Delta w \propto \alpha \left(V_\pi(s) - \hat{V}(s,w)) \right) \nabla_w \hat{V}(s,w) \tag{5}$$

For a linear function approximator, this update writes :

$$\delta w = \alpha \left(V_\pi(s) - \hat{V}(s,w)) \right) \phi(s) \tag{6}$$

Remark : Of course, tabular features is a special of the linear function approximator: just pick $\phi(s) = \delta(s - s_i)$; and the weights to hold value for each state.

Since the correct value function $V_\pi(\cdot)$ is unavailable, we need to use proxy's for the update (5). As for the tabular case, we have several options (depending which expression of the expected return we want to approximate):

$$\begin{aligned} V_\pi(s) \quad &= \mathbb{E}_\pi \left[R_t | s_t = s \right] \\ &= \mathbb{E}_\pi \left[\sum_{i=0} \gamma^i r_{t+1+i} \right] \quad \gamma \in]0,1[\\ &\simeq r_{t+1} + \gamma V_\pi(s_{t+1}) \quad = R_t^{TD(0)} \end{aligned} \tag{7}$$

and adapt the update accordingly:

$$\Delta w \propto \alpha \left(G(s) - \hat{V}(s,w)) \right) \nabla_w \hat{V}(s,w) \tag{8}$$

with $G(s) = R^{MC}(s)$, $R^{TD}(s), \ldots$.

1.1 Monte-Carlo value approximation

Because Monte-Carlo like updates writes:

$$R_t^{MC} = r_{t+1} + \gamma r_{t+2} + \dots \gamma^T r_{T-t-1} \tag{9}$$

it provides an unbiased estimator of $V_\pi(s = s_t)$, and MC evaluation converges toward a **local optimum**. This holds for both non-linear and linear function approximation.

1.2 Temporal Difference value approximation

The TD target writes:

$$R_t^{TD(0)} = r_{t+1} + \gamma \hat{V}^\pi(s = s_{t+1}, w) \tag{10}$$

and produce (because of bootstrapping) a *biased* sample of the true value function approximation. We can however still apply our learning procedure, for instance on a linear function approximator:

$$\begin{aligned} \Delta w &= \alpha \left(r_{t+1} + \hat{V}(s_{t+1}, w) - \hat{V}(s_t) \right) \nabla_w \hat{V}(s = s_t, w) \\ &= \alpha \delta_t^{TD} \phi(s_t) \end{aligned} \tag{11}$$

which was proved to converge to a local optimizer in the linear case.

One could also use the λ-return as another biased sample of the true value function. The update will write, for $\lambda < 1$:

$$\begin{aligned} \delta_t &= r_{t+1} + \gamma \hat{V}(s_{t+1}, w) - \hat{V}(s_t, w) \\ e_t &= \gamma \lambda e_{t-1} + \phi(s_t) \\ \Delta w &= \alpha \delta_t E_t \end{aligned} \tag{12}$$

the equivalent of the TD(0) backward view for the continuous case.

1.3 Control and convergence

As for tabular MDPs, we alternate action-value function evaluation:

$$\hat{Q}(s, a, w) \simeq Q_\pi(s, a) \quad \forall s, a \in \mathcal{S} \times \mathcal{A} \tag{13}$$

and policy improvement for control.

It has been showed that MC, TD(0) and TD(λ) converges for table look-up and linear function approximates when learning *on-policy*. For non-linear function approximation, only MC methods were proven to converge on-policy. If learning *off-policy*, there were proof of divergence for linear TD(0) and TD(λ).

2 Batch reinforcement learning

Gradient-descent is simple but not sample efficient. Batch methods aim at correcting this behavior by finding the best fitting value function. Consider the experience $\mathcal{D}$ of <state,value> pairs:

$$\mathcal{D} := \{\langle s_1, v_1^\pi \rangle, \dots, \langle s_T, v_T^\pi \rangle\} \tag{14}$$

and we wish to minimize:

$$\sum_{t=1}^{T} \left(V_t^\pi - \hat{V}(s_t, w) \right)^2 = \mathbb{E}_\mathcal{D} \left[\left(V_\pi(s) - \hat{V}(s, w) \right)^2 \right] \tag{15}$$

By repeating the procedure :

1. Sample $\langle s, V_\pi(s) \rangle \sim \mathcal{D}$
2. Apply SGD update:

$$\Delta w = \alpha \left(V_\pi(s) - \hat{V}(s, w) \right) \nabla_w \hat{V}(s, w) \tag{16}$$

we converge to a least-square solution. Of course, if $\hat{V}(\cdot, w)$ is linear in w, the least-square solution is analytically computed:

$$w_{LS}^* = \left(\sum_{t=1} \phi(s_t) \phi(s_t)^T \right)^{-1} \left(\sum_{t=1}^{T} V_\pi(s_t) \phi(s_t) \right) \tag{17}$$

Again, because we don't know $V_\pi(\cdot)$, we use either unbiased or biases samples from the current policy. For instance, for Least-Square TD(0) (or LSTD), we choose the target return to be:

$$\delta_t = r_{t+1} + \gamma \hat{V}(s_{t+1}, w) \tag{18}$$

and therefore the least-square solution writes:

$$w^*_{LSTD} = \left(\sum_{t=0}^{T} \phi(s_t) \left(\phi(s_t) - \gamma \phi(s_{t+1}) \right) \right)^{-1} \left(\sum t = 1^T \phi(s_t) r_{t+1} \right) \tag{19}$$

The same reasoning holds for control - i.e action value function approximation. However, when using policy iteration with off-policy learning, and non-linear approximators, learning used to be unstable and practitioners reported that converging required more advances techniques. Those techniques are *experience replay* and *target networks*.

2.1 Experience replay

Let us store the transition of the form $(s_t, a_t, s_{t+1}, r_{t+1})$ in a cyclic buffer, enabling an agent to sample from and train on previously observed data offline. This provides three main advantages:

- reduce the amount of interactions needed with the environment.
- reduce the variance of updates.
- by sampling from a large memory, the temporal correlations that can adversely affect RL are broken.

While the first DQN algorithm used uniform sampling in that buffer, recent implementations have shown the superiority of prioritized sampling (w.r.t TD erros).

2.2 Fixed targets

Instead of using the network we are currently optimizing to compute the target:

$$\delta_t = r + \gamma \max_{a'} Q(s_{t+1}, a', w) - Q(s_t, a, w) \tag{20}$$

we use a fixed network that gets updated more rarely, avoiding to use the fluctuating weights. Hence, with w^- being old frozen parameters:

$$\delta_t = r + \gamma \max_{a} Q(s', a', w^-) - Q(s, a, w) \tag{21}$$