8 Scaling Up: Value and Policy Approximations

8.1 Overview

When the state and action spaces are large, the previous methods 'table-based' methods, that accounted for each state and/or action separately, are not feasible. One therefore needs to resort to approximate methods.

In essence, large problems impose several difficulties, involving:

- Computation.
- Representation.
- Learning: generalization and exploration.

These issues are common to other AI /ML domains. Many heuristic and approximation approaches have been developed to deal with these issues. These include (in the AI jargon):

- State abstraction (aggregation, feature-based mappings,....).
- Temporal abstraction (different time scales, high-level actions).
- Problem decomposition, hierarchical methods.

In the context of MDPs and RL, one can distinguish between

- Approximate Dynamic Programming (ADP): Focus on planning
- (Approximate) RL: Focus on learning.

This distinction is not strict, and both the terms and the relevant methods are used interchangeably.

In this chapter we focus on function approximation methods for the value function. In general, function approximation addresses both issues of *representation* and learning *generalization*.

8.2 Function Approximation

RL involves several 'large' functions that may need to be approoximated:

- Value functions: $V^\pi(s)$, $V^*(s)$,$Q(s,a)$...
- Policies: $\pi(s)$, $\pi(a|s)$.
- Model: $p(s'|s,a)$, $r(s,a)$.

Function approximation methods can be classified into:

- Parametric methods: $F(x) \approx f_\theta(x)$, and parametric family, with $\theta \in \Theta$ is a parameter vector of fixed dimension.
- Non-parametric methods: $F(x) \approx f_n(x)$, with f_n depending on the size n of the data. (E.g.: Kernel methods, lazy learning + nearest-neighbor).

The basic approximation problem is to minimize some distance metric between $f(x)$ and $F_\theta(x)$, for example

$$\min_{\theta\in\Theta} \int \|F(x) - f_\theta(x)\| .$$

In the learning context, we are given only noisy samples $(x_k, y_k)_{i=1}^n$ with $y_k \approx F_k(x)$ (i.e., $y_k = F(x_k) + n_k$). This is called the *regression problem* in statistics. We can then try to minimize the empirical risk

$$\min_{\theta\in\Theta} \sum_k \|y_k - f_\theta(x_k)\|^2 .$$

However, we must beware of *overfitting* the parameters to the data. A common approach is to add a regularization term: E.g. $\lambda \int (F''_\theta(x))^2$, or simply $\lambda\|\theta\|^\alpha$, $\alpha = 1$ or 2. The parameters θ can be tuned (learned) using gradient methods, or more advanced optimization schemes. See the basic Learning Systems course.

In RL the problem is usually more complicated. We are not directly given the values y_k (e.g., of the value function or the policy), but rather need to estimate those from the date.

Approximation Architectures: The functional form of the parametric family $\{f_\theta\}$ is sometimes called the approximation architecture. Common choices include:

Nonlinear Neural Networks (feedforward, others...).

Linear function Architectures: Here ,

$$f_\theta(x) = \sum_{j=1}^{m} \theta_j \phi_j(x) \equiv \theta^T \phi(s)\,.$$

where (θ_j) are the *weights*, and (ϕ_j) are the *basis functions*. Some examples for basis functions

1. Harmonic functions (Fourier series).
2. Polynomials / piecewize polynomial approximations (splines)
3. Radial basis functions (RBFs).
4. Piecewise constant functions (state aggregation).
5. 'Tile coding'.
6. Features.

For RL, linear architectures are currently the best understood in terms of analysis, and provide more predictable results. A basic problem is how to choose the basis functions. Also, a linear representation is often not the most compact one.

8.3 Policy Evaluation

Consider the first problem of evaluating the value function $V^\pi(s)$ (or similarly $Q^\pi(s,a)$) for a given policy. This is the simplest RL problem.

8.3.1 Simple Monte Carlo

Let $\{v_\theta(s)\}$ be a parametric family of approximating value functions over the state space S, with parameter θ.

As already discussed for basic RL, it is possible to obtain noisy samples v_k of $V^\pi(s_k)$ for various visited states (s_k), by applying the policy π to the process.

We now fact the standard regression problem of fitting $v_\theta(s)$ to the data (s_k, v_k). This can be done using gradient or Newton methods, in a batch or sequential (sample-by-sample) fashion.

For example, consider the least-squares problem:

$$\sum_s (v_\theta(s_k) - v_k)^2$$

For the linear architecture $v_\theta(s) = \theta^T \phi(s)$, we may obtain the solution explicitly as the solution of the linear equation

$$(\sum_k \phi(s_k)\phi(s_k)^T)\theta = \sum_k \phi(s_k) v_k \,.$$

8.3.2 Temporal-Difference Methods

Here we adjust the TD(λ) algorithm to function approximation, in a gradient-based framework. Recall that $V = V^\pi$ satisfies

$$V(s) = \sum_{a,s'} \pi(a|s)\big(r(s,a) + \gamma p(s'|s,a)V(s)\big)$$

Known model: Suppose the model is know (ADP problem). We may attempt to minimize the squared difference

$$\sum_s w(s)\Big(v_\theta(s) - \sum_{a,s'} \pi(a|s)\big(r(s,a) + \gamma v_\theta(s')\big)\Big)^2$$

over θ. Here $(w(s))$ are some weights – perhaps the stationary state distribution $p^\pi(s)$ under π.
If S is large, we may instead minimize the above sum over a finite sample (s_k) of states. These can reasonably be obtained by observing a trajectory (s_k) under π, leading to

$$\min_\theta \sum_{s_k} \Big(v_\theta(s_k) - \sum_{a,s'} \pi(a|s_k)\big(r(s_k,a) + \gamma p(s'|s_k,a) v_\theta(s')\big)\Big)^2$$

Learning: Given a sample (s_k, a_k, r_k, s'_k), with (a_k, r_k, s'_k) obtained under the policy π for each s_k,, we can similarly attempt to minimize the empirical squared difference:

$$\min_\theta \sum_k \left(v_\theta(s_k) - (r_k + \gamma v_\theta(s'_k))\right)^2 \equiv \min_\theta \sum_k d_k^2 \,,$$

where d_k is the familiar temporal difference.

Partial Gradient Learning: Note that θ appears in two terms inside the square. Taking the gradient with respect to both leads to a complicated scheme. Instead, we treat $y_k \triangleq r_k + \gamma v_\theta(s'_k)$ as measurement, to which $v_\theta(s_k)$ should be fitted. Thus, with $d_k = y_k - v_\theta(s_k)$ serving as the error, we obtain

$$\nabla_\theta(d_k^2) = -d_k \nabla_\theta v_\theta(s_k)\,,$$

and the gradient update rule

$$\theta_{k+1} = \theta_k - \frac{\alpha}{2}\nabla_\theta = \theta_k + \alpha \nabla_\theta v_\theta(s_k) d_k\,.$$

Here α is the gain, which can be fixed or time-varying. This is the analogue of the familiar TD(0) algorithm.

$TD(0)$ with linear function approximation: Suppose now that $v_\theta(s) = \theta^T \phi(s)$. Here we obtain,

$$\theta_{k+1} = \theta_k + \alpha \phi(s_k) d_k\,,$$

or in scalar form,

$$w_{k+1}(j) = w_k(j) + \alpha \phi_j(s_k) d_k\,,$$

We can view $\phi_j(s_k)$ as a measure for the relevance of ϕ_j (hence $\theta(j)$) to state s_k.

$TD(\lambda)$ with linear function approximation: Recall the TD(λ) algorithm in the tabular case:

$$\begin{aligned}\hat{V}_{k+1}(s) &= \hat{V}_{k+1}(s) + \alpha z_k d_k \\ z_k(s) &= \lambda\gamma z_{k-1}(s) + 1\{s_k = 1\}\,.\end{aligned}$$

Similar considerations lead to the following algorithm with linear function approximation:

$$\begin{aligned}\theta(k+1) &= \theta(k) + \alpha z_k d_k \\ z_k &= \lambda\gamma z_{k-1} + \phi(s_k)\,.\end{aligned}$$

Convergence of TD(λ) [Tsitsiklis and van Roy '97]: Suppose the Markov chain induced by π is irreducible (all states communicate), and let sequence (s_k) be sampled according to π. Then $\theta_k \to \theta^*(\lambda)$ (w.p. 1), where $\theta^*(\lambda)$ is defined as follows.

$\lambda = 1$ (pure Monte-Carlo): $\theta^*(1)$ is the solution of

$$\min_w \sum_s p^\pi(s)\big(v_\theta(s) - V^\pi(s)\big)^2 .$$

$0 \le \lambda < 1$: $w^*(\lambda)$ is the solution of $A(\lambda)w = b(\lambda)$, where

$$A(\lambda) = E\big(z_k\big(\phi(s_k) - \gamma\phi(s_{k+1})\big)^T\big) = \dots$$
$$b(\lambda) = E\big(z_k r(s_k, a_k)\big)\big) = \dots$$

where the expectation is taken with respect to the stationary distribution under π.

We note that $\theta^*(\lambda)$ is generally biased with respect to the "optimal" value $\theta(1)$, but convergence may be faster.

Further note that the requirement that (s_k) be sampled sequentially according to π is essential here: otherwise convergence can fail. This is in contrast to the tabular case!

Least Squares Schemes: LSTD(λ): The solution $w^*(\lambda)$ of TD(λ) can be arrived at more quickly using the following scheme. Define

$$A_n = \frac{1}{n}\sum_{k=1}^{n} z_k\big(\phi(s_k) - \gamma\phi(s_{k-1})\big)^T$$
$$b_n = \frac{1}{n}\sum_{k=1}^{n} z_k r_k$$
$$\theta_n = A_n^{-1} b_n$$

with z_k defined as above. Then, by the Law of Large Numbers, $\theta_n \to \theta^*(\lambda)$ (w.p. 1).

Comments:

- LSTD(λ) typically converges much faster than TD(λ).
- LSTD(λ) makes full use of the data: It does not 'forget' earlier measurements.
- When $\phi(s) = e_s$ (tabular case), LSTD(λ) coincides with a model-based scheme (show that for LSTD(0)).
- TD(λ) requires $m = \dim(\theta)$ operations per sample. LSTD(λ) requires $O(m^2)$ operations per sample (to compute A_n), plus $O(m^3)$ operations for matrix inversion (when needed). Alternatively, the θ_n can be calculated recursively, using RLS formulae, which requires $O(m^2)$ operations per sample.