Kernel-Based Reinforcement Learning

Tianyu Li

March 17, 2017

1 Preliminary and notations

Given a MDP{S, A, $P_{\cdot}(\cdot|\cdot)$, $r(\cdot,\cdot,\cdot)$, α}, where S is a set of possible states, A is the set of actions, $P_a(s_t+1 = s|s_t = s')$ describes the probability of transferring from s' to s provided the action a, $r(s_t+1 = s, s_t = s, a)$ gives the corresponding reward for the above behaviour, and α is the discount factor. For this MDP, the (optimal) value function and action value function can be written as:

$$J_t^*(s') = \max_{a\in A} E(r(s, s', a) + \alpha J_{t+1}^*(s)|s_t = s', a_t = a) \tag{1}$$

$$Q_{t,a}^*(s') = E(r(s, s', a) + \alpha J_{t+1}^*(s)|s_t = s', a_t = a) \tag{2}$$

Here we can tell the relationship between $J_t^*(s')$ and $Q_{t,a}^*(s')$. Define $\Gamma_a(x)$ as the function $E(r(s, s', a) + \alpha x|s_t = s', a_t = a)$, then we can see this function as a mapping from $J_{t+1}^*(s)$ to $Q_{t,a}^*(s')$, that is $Q_{t,a}^* = \Gamma_a J_{t+1}^*$. Note for equation (1) we have the Bellman operator[2], which is $J_t^* = \Lambda J_{t+1}^*$ for the finite case and for infinite case, we have $J^* = \Lambda J^*$ as the solution is the fix point.

2 Introducing Kernel

The original expansion of the Γ_a operator can be written as:

$$E(r(s, s', a)+\alpha J_{t+1}^*(s)|s_t = s', a_t = a) = \sum_s P_a(s|s')[r(s, s', a)+\alpha J_{t+1}^*(s)] \tag{3}$$

The above formation works very well if we are in the discrete case. However, if now we are facing continuous case, i.e. $s \in \mathbb{R}^d$. Then an intuitive way of calculate value function and action value function would be try to discretize s. For example, assume $s \in [0, 1]^d$, then in order to discretize the interval, we would like to perform partition of the space. Let $B_1, B_2, ..., B_N \in [0, 1]^d$, then we will have:

$$\begin{aligned} P_a(s_{t+1} = s|s_t = s') &= P(s_{t+1} \in B_s|s_t \in B_{s'}, a_t = a) \\ &= \frac{N(s_{t+1} \in B_s, s_t \in B_{s'}, a_t = a)}{N(s_t \in B_{s'}, a_t = a)} \end{aligned} \tag{4}$$

N in the above equation is merely the count of each corresponding situation appears in the sample, as the probability can be estimated through empirical frequency. However, for continuous case, we cannot split the space into arbitrary small subspace. Indeed, the finer we cut the space the less bias we will have in our estimation, but not only this will require huge amount of data, we also will face the problem of overfitting. Also in [3], they pointed out that a piecewise constant table estimates are typically inferior to "smoothing" method.

What if we replace each discrete distribution with continuous ones, i.e. Gaussian distribution, although this will involve estimation of parameter: mean, if the variance is given. The intuition will be modeling $P(s_{t+1}, s_t|a_t = a)$ as a two-variant Gaussian, and $P(s_t)$ as a uni-variant Gaussian, then we will be able to compute the conditional distribution. And then we can apply maximum likelihood estimation to solve the mean for each of the Gaussian and then we do this recursively in a EM fashion. Will this be problematic?

Yes, This will be problematic. First, we do not have sufficient control of the samples that we will encounter in the future. This is due the probability we first have is biased and thus the policy induced by this probability is also biased, which will lead to insufficient data regarding to the optimal policy, therefore the probability might converge poorly.

Now instead of looking at the piecewise probability, we just look at the "neighborhood" of s, measure the similarity between them and sum over the neighbourhood's value. Denote S^a as the set of every state pair that got transferred through the action a. Mathmatically we have: $S^a = \{(s_1, s_2)|s \in S, (s_1, a, s_2) \in samples\}$, where S is the set of states. Now we can rewrite the approximation of Γ_a as the following equation:

$$\hat{\Gamma_a} J(s) = \sum_{(s_1,s_2)\in S^a} \Phi_{S^a}(s_1, s)[r(s_2, s_1, a) + \alpha J(s_2)] \tag{5}$$

Where Φ_{S^a} is the function that measure the similarity of two states. The intuition behind this equation is like K-nearest neighbour. First we obtain S^a, which is the state pair acquired through action a, from sample. Then for each pair (s_1, s_2) we compute the corresponding utility, then in order to compute the value function, we will sum over all these utility using the weights we obtained through Φ_{S^a}, which measure the similarity of the actual input state s and the "neighbours" s_1. This can be illustrated through the following graph:

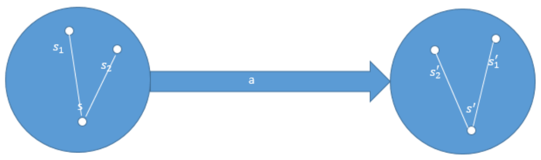

Figure 1: Illustration for equation (5), the two circles represent S_a and the string represents the corresponding weights

Would it be better to not use the entire S^a, instead, we set up a threshold $\tilde{T}$, then we should have: $\Phi_{S^a}(s_1, s) = \begin{cases} \Phi_{S^a}(s_1, s) & if \Phi_{S^a}(s_1, s) > \tilde{T} \\ 0 & otherwise \end{cases}$

Yes, this would help. However, later we will introduce the bandwidth parameter and it will have a similar effect with the above setting. In this way we can adjust the bias-variance trade-off, i.e. if we are looking into all the pairs in S^a, then we will have a relatively high variance.

As we treat Φ as a similarity function, then naturally we will consider it as a Kernel function. Thus we change our notation Φ to $K_{S^a,b}(s_1, s)$, where b is the bandwidth parameter and it controls the degree of smoothing and therefore bias-variance trade-off. In order to prevent the recursive function (5) from exploding, we need to normalize this kernel function.

$$K_{S^a,b}(s_1, s) = \frac{k(\frac{\|s_1 - s\|}{b})}{\sum_{(s_1,s_2)\in S^a} k(\frac{\|s_1 - s\|}{b})} \tag{6}$$

Then to solve the value function, similar to value iteration, we can iterate backwards from the terminal condition $J^*(s) = R(s)$.

3 Bandwidth selection and bias-variance trade-off

The above setting of kernel-based RL expand the discrete parametric setting of traditional MDP into a continuous and non-parametric one. This is rather nice, however, one leftover for the above kernel is how to set the bandwidth parameter. Intuitively, as the weight is the inverse of $\frac{\|s_1 - s\|}{b}$, so if we increase b then the result will be very smooth, however this will allow very different states-pair has similar weight as the closely related state-pair. Nevertheless, if you decrease b the difference of the weight will be drastic, resulting in a rough distribution. Mathematically to see this, we can decompose the error term $\hat{\Gamma_a} J - \Gamma_a J$ into a bias term $E(\hat{\Gamma_a} J(s)) - \Gamma_a J(s)$ and the variance term $\Gamma_a \hat{J}(s) - E(\hat{\Gamma_a} J(s))$.

Note the bias term describes how far is the expectation of the estimated value away from the true value, while variance describes how far is any estimated value away from the expectation. Here we can see the opposite effect of bandwidth on these two terms, by decreasing the bandwidth we essentially reduce the size of S^a leading the bias term smaller as the neighbours are now closer to s, while increase the variance term as do not have enough data and vice versa.

This could be a headache, because if we set the bandwidth parameter as a hyper parameter to train, then due to the complexity of training a MDP, this could be very expensive. Luckily, it has been proven in [1], if the number of the data is big enough, we can choose the bandwidth parameter as arbitrary small value. Intuitively, if the sample size is large enough, then given a state s, it is very likely that it has already appeared before, then just use a few of its neighbours could give us good estimation. This also gives us a hint on how to adjust the bandwidth when training with Monte Carlo or TD. If we are using Monte Carlo, then the bandwidth should be roughly chosen based on the length of the episode, i.e. if the episode if long enough, we can use smaller value of the bandwidth. While for TD, because we increase the sample size as we go, so the bandwidth should be set as a decaying function.

4 Reference

[1] Ormoneit, D., Sen, . (2002). Kernel-based reinforcement learning. Machine learning, 49(2-3), 161-178.
[2] Bellman, R. (1956). Dynamic programming and Lagrange multipliers. Proceedings of the National Academy of Sciences, 42(10), 767-769.
[3] Fan, J., Gijbels, I. (1996). Local polynomial modelling and its applications: monographs on statistics and applied probability 66 (Vol. 66). CRC Press.