Robust Q learning

Tue Herlau

November 16, 2022

1 Introduction

The work is an extension of [LBB+22]. The idea is fairly straight-forward. The Q-update which is robust within a KL distance of δ is given as a minimization problem. The implementation, however, really sucks.

We try to fix this by decompositing the value function. Let the temporal observations be:

$$x_0, x_1, \cdots \tag{1}$$

We define a state as a block of K of these:

$$s_t = x_{t-K+1:t} \tag{2}$$

This means that the transition probability of the states is partly deterministic as the first $K-1$ elements are simply copied into the last $K-1$ elements of the next state.

Let $\hat{s}_t = x_{t-K+1:t-1}$ and $s_t = s = \begin{bmatrix}\hat{s} & x\end{bmatrix}$ Then we parameterize the optimal value function $V(s)$ as a quadratic function:

$$V(s) \approx \hat{V}(s) = a(\hat{s}) + \frac{1}{2\sigma^v(\hat{s})^2}(\mu(\hat{s}) - x)^2 \tag{3}$$

We also assume that the transition dynamics is jointly Gaussian, viz.:

$$p_0(s'|s,a) = \mathcal{N}(x'|\mu^f(s,a), \sigma^f(s,a)^2) \tag{4}$$

1.1 Bellman operator

The bellman operator defined in the main reference is:

$$\mathcal{T}_\delta^{\text{rob}}(Q)(s,a) = r(s,a) + \gamma \sup_{\beta \geq 0} \left\{ -\beta \log \left(\mathbb{E}_{p^0_{s,a}} \left[e^{-\frac{1}{\beta} \max_b Q(s',b)} \right] \right) - \beta\delta \right\} \quad (5)$$

This parametrization now allows us to solve the integral and get an explicit expression for the minimization problem. The integral is a standard product of two Gaussians and it becomes:

$$\mathbb{E}_{p_0} \left[e^{\frac{-V(s')}{\beta}} \right] = e^{-\frac{a}{\beta}} \sqrt{2\pi\beta \left(\sigma^v\right)^2} \left[\frac{e^{\frac{-1}{2} \frac{\left(\mu^f - \mu^v\right)^2}{S_\beta^2}}}{\sqrt{2\pi S_\beta^2}} \right], \quad S_\beta^2 = \left(\sigma^f\right)^2 + \beta \left(\sigma^v\right)^2 \quad (6)$$

We plug it in and get that the quantity to be optimized is:

$$a - \beta \log(\sqrt{2\pi}\sigma^v) - \beta \frac{1}{2} \log \beta + \beta \frac{1}{2} \frac{\left(\mu^f - \mu^v\right)^2}{S_\beta^2} + \beta \log \sqrt{2\pi} + \beta \log S_\beta - \beta\delta \quad (7)$$

$$= a - \beta \frac{1}{2} \log \beta + \beta \frac{1}{2} \frac{\left(\mu^f - \mu^v\right)^2}{S_\beta^2} + \beta \log S_\beta - \beta(\delta + \log \sigma^v) \quad (8)$$

$$= a - \beta \log \left(\frac{\sqrt{\beta}\sigma^v}{S_\beta} \right) + \beta \frac{1}{2} \frac{\left(\mu^f - \mu^v\right)^2}{S_\beta^2} - \beta\delta \quad (9)$$

$$= a - \beta \log \left(\frac{\sqrt{\beta}\sigma^v}{\sqrt{(\sigma^2)^2 + \beta(\sigma^v)^2}} \right) + \beta \frac{1}{2} \frac{\left(\mu^f - \mu^v\right)^2}{S_\beta^2} - \beta\delta \quad (10)$$

$$= a + \frac{1}{2}\beta \log \left(1 + \frac{(\sigma^f)^2}{\beta(\sigma^v)^2} \right) + \beta \frac{1}{2} \frac{\left(\mu^f - \mu^v\right)^2}{(\sigma^f)^2 + \beta(\sigma^v)^2} - \beta\delta \quad (11)$$

$$(12)$$

(TODO: simplify that ridiculous mess). The questions is if we can help ourselves a bit in the optimization task by somehow showing this function has a single optimum or similar – I am not sure that is possible, but you never know. A step towards this is to differentiate it. vlg., lets assume it has

the form:

$$F(x) = x \log\left(1 + \frac{\lambda}{x}\right) + \frac{A}{\frac{\lambda}{x} + 1} - Bx \tag{13}$$

$$= x \log h(x) + \frac{A}{h(x)} - Bx \tag{14}$$

$$h(x) = 1 + \frac{\lambda}{x} \tag{15}$$

We differentiate to get:

$$F'(x) = \log h(x) + x\frac{h'(x)}{h(x)} + \frac{h'(x)}{h(x)}\left(\frac{-A}{h(x)}\right) - B \tag{16}$$

$$= \log h(x) + \frac{h'(x)}{h(x)}\left[\frac{h(x)x - A}{h(x)}\right] - B \tag{17}$$

$$\frac{h'(x)}{h(x)} = \frac{-\frac{\lambda}{x^2}}{1 + \frac{\lambda}{x}} = \frac{1}{x}\frac{-\lambda}{x + \lambda} \tag{18}$$

So therefore

$$F'(x) = \log h(x) - B - \frac{\lambda}{x + \lambda}\left[\frac{xh(x) - A}{xh(x)}\right] \tag{19}$$

$$= \log h(x) - B - \frac{\lambda}{x + \lambda}\left[\frac{x + \lambda - A}{x + \lambda}\right] \tag{20}$$

So we can now start to bound this. We get that

$$F'(x) \leq \frac{\lambda}{x} - B - \frac{\lambda}{x + \lambda}\left[\frac{x + \lambda - A}{x + \lambda}\right] \tag{21}$$

Solving for $F'(x) \leq 0$ can be done by reducing this to a 3rd degree polynomial and solve for the smallest root. Less exactly, we can also just ignore higher-order terms in x and solve to get:

$$0 < \frac{\lambda}{x} - B - \frac{\lambda}{x + \lambda} + \frac{A}{(x + \lambda)x} \tag{22}$$

$$x > \frac{1}{2B}\sqrt{(\lambda B)^2 + 4B(\lambda^2 + \lambda A)} - \frac{\lambda}{2} \tag{23}$$

This can be simplified obviously (TODO). The lower bound can be obtained by simply dropping terms to get:

$$F'(x) > \log h(x) - B - 1 \tag{24}$$

$$x \leq \frac{\lambda}{e^{B+1} - 1} \tag{25}$$

(TODO: Much better lower bound can be obtained by back-substituting upper and lower bounds into expression for F' and solve again for lower bound. This will give explicit A-dependency.

This gets us an interval for x within which to search for a root using the bisection method:

$$x \in \left[\frac{\lambda}{e^{B+1} - 1} \; ; \; \frac{1}{2B} \sqrt{(\lambda B)^2 + 4B(\lambda^2 + \lambda A)} - \frac{\lambda}{2} \right] \tag{26}$$

2 Generalization:

Suppose the dynamics of the model is given by:

$$x_{k+1} = f(x_k, a_k, L\epsilon_k) \tag{27}$$

$$w_k \sim P_W(\cdot) \tag{28}$$

Where L is a linear operator (let's see). We then define the state as:

$$s_k \equiv \begin{bmatrix} x_k \\ x_{k-1} \\ a_{k-1} \\ L\epsilon_k \end{bmatrix} \tag{29}$$

This definition contains redundant information. The update rule is then

$$s_{k+1} = \bar{f}(s_k, a_k, L\epsilon_k) = \begin{bmatrix} f(x_k, a_k, L\epsilon_k) \\ x_k \\ a_k \\ L\epsilon_k \end{bmatrix} \tag{30}$$

Then the hard part of the update rule is:

$$\mathbb{E}_{p^0_{s,a}} \left[e^{-\frac{V(s')}{\beta}} \right] \tag{31}$$

Where p_0 is the dynamics of the (unperturbed) version of the system. To compute the expectation, we will assume a special form of the value function namely:

$$V(s_{k+1}) \approx \hat{V}(s_{k+1}) = V^{(1)}(x_k, a_k) + V^{(2)}(x_k, a_k)^\top w_k + \frac{1}{2} w_k^\top V^{(3)}(x_k, a_k) w_k \tag{32}$$

We will also assume the transition dynamics takes the form:

$$p^0_{s,a} \approx \hat{p}^0_{s,a}(s'|s, a) = V^{(1)}(x_k, a_k) + V^{(2)}(x_k, a_k)^\top w_k + \frac{1}{2} w_k^\top V^{(3)}(x_k, a_k) w_k \tag{33}$$

$$\tag{34}$$

Then the transition function is assumed to be:

$$x_{k+1} \approx \bar{f}(x_k, a_k, w_k) = f^{(1)}(x_k, a_k) + f^{(2)}(x_k, a_k)^\top w_k + \frac{1}{2} w_k^\top f^{(3)}(x_k, a_k) w_k \tag{35}$$

The expectation can then still be computed exactly, and the coefficients above can be learned using regression (even local linear?).

3 Question: Can we use an embedding?

Assume a new generative model of the form:

$$x_{k+1} = f(h(x_k, a_k) + w_k) \tag{36}$$

Where h is an embedding and w_k is assumed to be $\mathcal{N}(0, I)$. Let $g = f^{-1}$ so that

$$g(x_{k+1}) = h(x_k, a_k) + w_k. \tag{37}$$

Then it holds that

$$p(x_{k+1}|x_k, a_k)dx_{k+1} = \mathcal{N}(w_k = g(x_{k+1}) - h(x_k, a_k); 0, I)|J_g(x_{k+1})|dw_k \tag{38}$$

We also assume that x_{k-1}, a_{k-1} and w_{k-1} are part of the state s_k. Therefore we can introduce the approximation (assuming W is a second degree polynomial):

$$V(s_{k+1}) \approx \hat{V}(s_{k+1}) = \hat{V}(x_k, a_k, w_k) = W_{h(x_k,a_k)}(w_k; \Phi). \tag{39}$$

3.0.1 Defining objectives

When learning the objectives, we train Q by minimizing the specific objective given in the paper (viz., using the simulator) as:

$$Q(s,a) = ... \tag{40}$$

So you could just as well train V directly and use the simulator to define Q... but this way is more elegant (or is it?). Robust learning requires V internally (to get 2nd degree polyal). Or does it?

The Q-learning objective satisfy the Bellman equation

$$Q(s_k,a_k) = \inf_{\mathrm{KL}(p|p_0)<\delta} \left\{ r(s_k,a_k) + \gamma \max_b Q(s_{k+1},b) \right\} \tag{41}$$

Re-writing this, and assuming for simplicity reward only depends on state/action, we get that for any constant C:

$$Q^{\mathrm{rob}}(s_k,a_k) = r(s_k,a_k) + \sup_{\beta>0} \left\{ -\beta \log \mathbb{E}\left[e^{-\frac{\max_b Q^{\mathrm{rob}}(s_{k+1},b)}{\beta}} \right] + \delta\beta \right\} \tag{42}$$

$$= r(s_k,a_k) + C + \sup_{\beta>0} \left\{ -\beta \log \mathbb{E}\left[e^{-\frac{V^{\mathrm{rob}}(s_{k+1})-C}{\beta}} \right] + \delta\beta \right\} \tag{43}$$

$$V^{\mathrm{rob}}(s_k) = \max_b Q^{\mathrm{rob}}(s_k,b) \tag{44}$$

for any constant C (probably a bad idea to have such a constant). Then to proceed, we need to find a way to compute the above expectation exactly and update. Let's say we divide this into two parts. The first computes this expectation (and updates). The second considers how to fit V^{rob}. For each V^{rob}, we can compute the target easily. Then perform the update as a regression problem (batched) which is trained jointly with o. IOW, we get that:

$$V^{\mathrm{rob}}(s_k) = \max_b Q^{\mathrm{rob}}(s_k,b) \tag{45}$$

$$\approx \hat{V}^{\mathrm{rob}}(s_k) \tag{46}$$

$$= C_0(h(x_{k-1},a_{k-1})) + C_1(h(x_{k-1},a_{k-1}))w_k + \frac{1}{2} w_k^\top C_2(h(x_{k-1},a_{k-1}))w_k \tag{47}$$

$$w_k = g(x_k) - h(x_{k-1},a_{k-1}) \tag{48}$$

Train this using gradient descent.

What happens if you use the Q-learning function and try to integrate? Let each coordinate be a 2nd degree polynomial.

V to minimize the recursion:

is a second-order approximation of V only depending on the latent state $h(x_k, a_k)$. We can learn the latent embedding by minimizing a loss over both V and over g (issue: g most well-defined. Possibly fix h insofar as second-order approx is concerned and use that ϵ automatically normal? Alternatively use that V must satisfy, before optimization, that:

$$V(s_{k+1}) = \max a_k \mathbb{E}\left[r(x_k, a_k) + V(x_{k+1})\right] \tag{49}$$

Then it holds that

$$J_g(x_{k+1}) \tag{50}$$

that $h_k = h(x_k, a_k, w_k)$ is a latent embedding of some sort and the next state is given by:

$$x_{k+1} = o(h_k) \tag{51}$$

Given embedding, we can write the value function as a quadratic in embedding parameter?

In other word the expectation is with respect to

$$0 \tag{52}$$

However, we can re-write this to be:

$$p(s'|s, a)$$

We will then assume a special parametrization of the value function in the form that:

$$V(s') \approx V^{(1)}(s) + V^{(2)}(s)L\epsilon + (L\epsilon)^\top V^{(3)}(s)(L\epsilon) \tag{53}$$

There are some questions about whether this is a rigorous construction or not. The three matrices need to be learned using a neural network; this is done by simply regressing

References

[LBB+22] Zijian Liu, Qinxun Bai, Jose Blanchet, Perry Dong, Wei Xu, Zhengqing Zhou, and Zhengyuan Zhou. Distributionally robust q-learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 13623–13643. PMLR, 17–23 Jul 2022.