Bayesian Reinforcement Learning

Vickie Ye and Alexandr Wang

1 Introduction

In reinforcement learning, the learning procedure consists of two parallel processes: estimating parameters of the surrounding environment and learning the optimal policy of highest long-term reward. A common way this is done is by learning a latent "value" function over the states and actions, which the agent uses to make policy decisions.

In this project, we examined two Bayesian approaches to learning the latent value function and policy. One way to do this is with a Bayesian model to estimate the parameters of the environment, as done in Strens (2000). This framework places Bayesian priors on the transition function and reward function of the environment and updates its posterior as it interacts with the environment.

Another Bayesian approach toward learning the value function uses Gaussian processes, as in Engel et al. (2005). This approach to value function approximation is nonparametric, and provides both a function estimate (the GP mean) and uncertainty (the GP covariance). The flexibility of this framework allows it to scale well to harder, higher-dimensional problems.

2 Bayesian MDP

MDPs are commonly used to learn the policy for the system with a set of states S, a set of actions A, a reward function $R(S, A)$, and a transition function $T(s, a, s') = P(X^{(t+1)} = s' | X^{(t)} = s, Y^{(t)} = a)$. To learn the optimal long-term-reward policy, we define a quality function with discount factor γ, $Q = \sum_{t=0}^{\infty} \gamma^t R^{(t)}$, which we approximate for each state-action pair as

$$Q(s, a) = \mathbb{E}[R(s, a)] + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$$

Thus, the quantities we need to estimate are the reward function $R(s, a)$ and transition probabilities $T(s, a, s')$ to make our updates to Q.

We define the transition distribution π for each state-action pair (s, a) as

$$\pi(s, a) = (T(s, a, s_0), ..., T(s, a, s_{N-1})),$$

where $\pi(s, a)_i = \mathbb{P}(s_i | s, a)$.

For our experiments we use a uniform Dirichlet prior with $\alpha_i = 1$. Our updated posterior $\pi(s, a)$ given the data is then

$$\pi^{(t)} \sim \text{Dirichlet}(\alpha^{(t)} | \mathbf{m}^{(t)}), \alpha_i^{(t)} = \alpha_i + m_i^{(t)}$$

where $\mathbf{m}^{(t)}$ represent the observed counts for the transitions. Then in estimating Q at each time step, we sampled $\pi(s, a)$ from our posterior.

We represent the reward for each state-action pair (s, a) as Gaussian- distributed with mean μ and precision τ. We use a $\text{Ga}(\beta, \rho)$ prior for τ and a $\mathcal{N}(\mu_0, c_0\tau)$ prior for μ. The updated posteriors are then

$$\tau \sim \text{Ga}\Big(\beta + \frac{k}{2}, \rho + \frac{1}{2}\sum_i (r_i - \bar{r})^2 + \frac{k c_0 (\bar{r} - \mu_0)^2}{2(n + c_0)}\Big),$$

$$\mu \sim \mathcal{N}\Big(\frac{k\bar{r} + c_0\mu_0}{k + c_0}, (k + c_0)\tau\Big)$$

for observed rewards $\mathbf{r} = \{r_i\}_{1 \le i \le k}$ for the state-action pair. Then to estimate Q at each time step, we sampled μ and τ, which we then used to sample R, for each (s, a). For comparison we also evaluated performance when using the modes of the posterior as estimates for $\pi(s, a)$ and $R(s, a)$.

2.1 Testing Problems

We used three toy problems to test our implementation. In the "Chain" problem (Figure 1 top), there are five states and two possible actions, with 0.2 probability slipping (performing the opposite action intended). The optimal policy is to explore to state 5, and stay there until slipping.

In the "Loop" problem (Figure 1 bottom), there are nine states and two possible actions, with no possibility of slipping. The optimal policy is to stay in the left loop and receive reward of 2 for every 5 actions.

In the "Maze" problem (Figure 2), the agent explores a maze to find flags. The agent can either move north, west, south, or east, and receives only its current position as its state. Each time the agent reaches the goal, it receives a reward equal to the number of flags collected and is transported back to the beginning of the maze. All other state-action pairs receive zero reward. In the maze problem, we also have a probability of slipping of 0.2.

2.2 Results

For the chain, loop, and hard maze problems, we performed learning phases of 5000 time steps. For the easy maze problem, we performed learning phases of 2000 time steps. For the chain problem, because of slipping, the optimal behavior on average receives a total reward of 22000

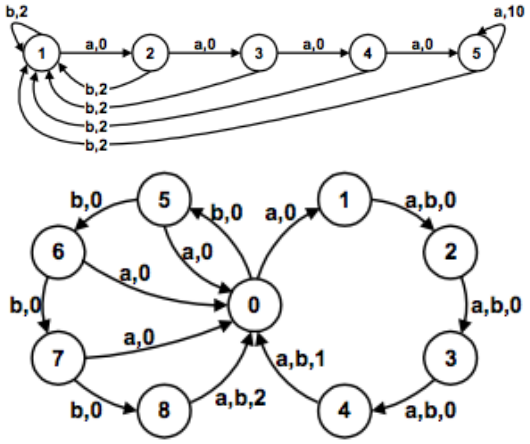

Figure 1: The "Chain" and "Loop" toy problems used in testing.

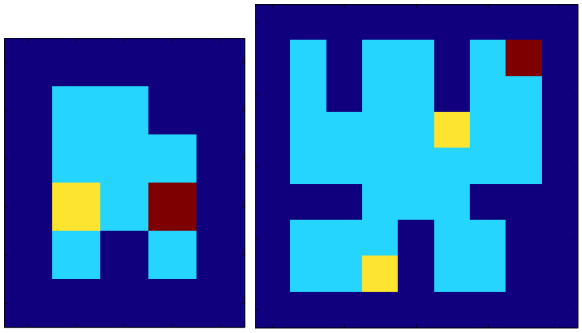

Figure 2: The "Maze" flag-collecting toy problem used in testing. Yellow squares are flags; red is the goal; all mazes start at the top left corner.

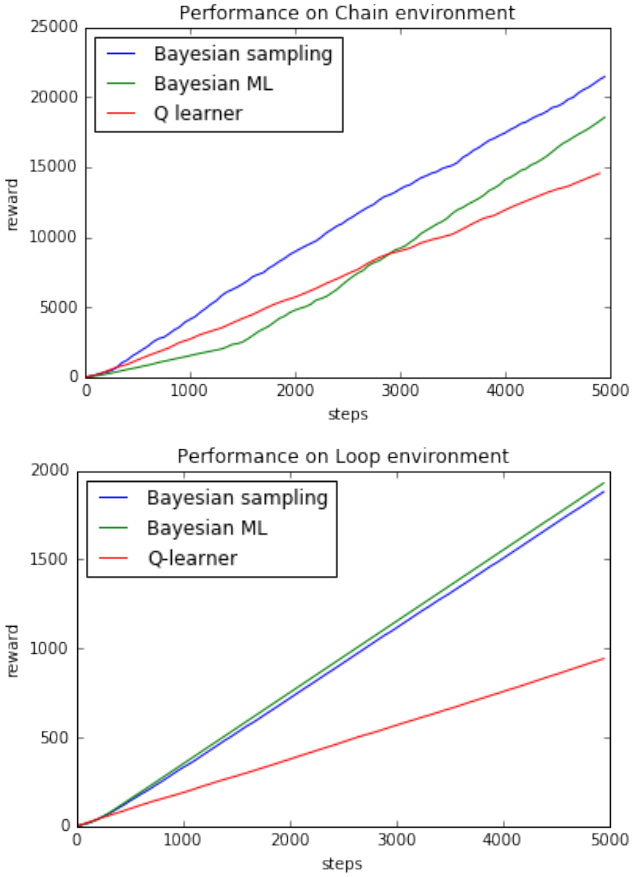

Figure 3: The performance of three learners in the "Chain" and "Loop" environments.

for 5000 steps. For the loop problem, the optimal behavior receives a total reward of 2000 for 5000 steps. In the maze problem, we estimate that each slip adds an extra step in on the optimal path, so the optimal reward for the easy maze is between 450 and 460 (with 2000 steps), and between 540 and 550 for the hard maze (with 5000 steps).

For each experiment, we compared the performance of the policy learned with samples from the posteriors (Bayesian sampling), the policy learned with the modes of the posteriors (Bayesian ML), and the naive Q learner, which learns Q with dynamic programming ($Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(R + \gamma \max_{a'} Q(s,a')$). The comparisons are shown in Figure 3 and 4.

In all four environments, the Bayesian ML and sampling learners significantly outperform the naive Q learner. In the chain environment, Bayesian sampling finds the near-optimal policy and receives a total reward of 21458, compared with the 18550 from Bayesian ML and the 14548 from Q learning. The difference between Q and Bayesian models is obvious in the loop problem, in which both Bayesian sampling and ML achieved optimal performance by receiving 2 rewards for every 5 steps, while the Q received 1 reward for every 5 steps.

In the easy maze, Bayesian sampling achieved a near-optimal reward of 456, and both the Bayesian ML and sampling learners drastically outperform the Q learner. Because our environment provided so little feedback in the maze, the Q learner was unable to find and exploit the optimal path. The Bayesian learners were able to deduce the walls without rewards and exploit the optimal path.

In general the Bayesian sampling slightly outperformed Bayesian ML. We think this is because the added randomness from sampling our posteriors causes the agent explore and find the optimal policy quicker than just using the modes of the posteriors.

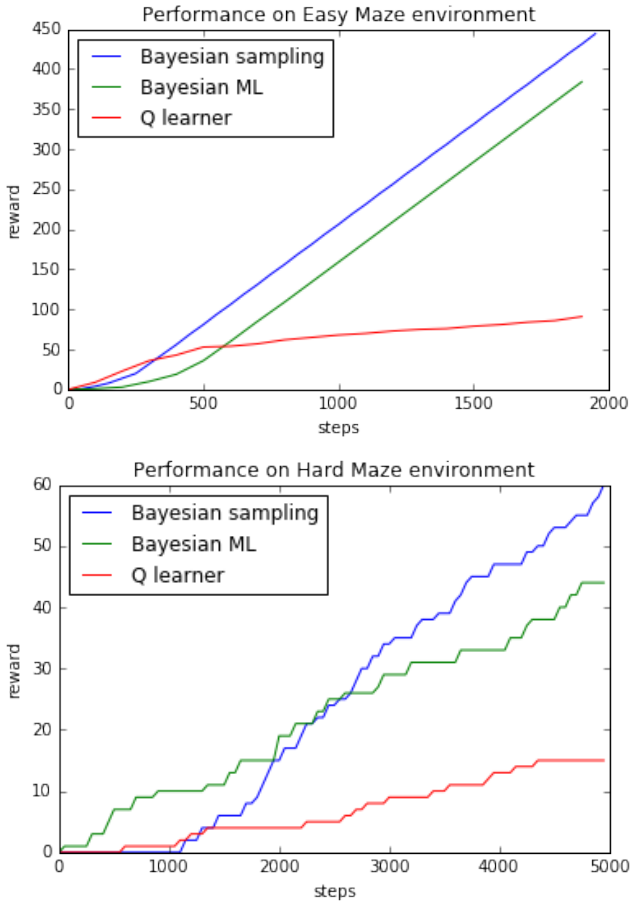

Figure 4: The performance of three learners in the easy and hard "Maze" environments. The Bayesian ML and sampling learners clearly outperform the naive Q learner, and find the optimal policy in the easy maze.

3 GPSARSA

In the GPSARSA framework, GPs are used to approximate the quality function Q. Similarly to GP regression, we put a GP prior over $Q \sim \mathcal{N}(0, k(\cdot,\cdot))$, where $\mathbb{E}[Q(x)] = 0$ and $\mathbb{E}[Q(x)Q(x')] = k(x,x')$, where $x, x' \in \mathcal{X} = S \times A$. The kernel $k(x,x')$ should reflect a similarity notion for the problem at hand.

SARSA refers to the interaction model where x (state-action) is evaluated, a reward is received, and the new x' is evaluated. We can thus formulate the reward model as

$$R(x^{(t)}, x^{(t+1)}) = Q(x^{(t)}) - \gamma Q(x^{(t+1)}) + N(x^{(t)}, x^{(t+1)})$$

where $N(x,x') = \Delta Q(x) - \gamma \Delta Q(x')$, $\Delta Q \sim \mathcal{N}(0, \Sigma)$, with $\Sigma(x,x') = \delta(x-x')\sigma^2(x)$. We can define, for some time t, the processes

$$R_t = (R(x^{(1)}, ..., R(x^{(t)}))^T$$
$$Q_t = (Q(x^{(1)}, ..., Q(x^{(t)}))^T$$
$$N_t = (N(x^{(1)}, ..., N(x^{(t)}))^T$$

and the vectors and matrices

$$\mathbf{k_t}(x) = (k(x^{(1)}, x), ..., k(x^{(t)}, x))^T$$
$$\mathbf{K_t}(x) = (\mathbf{k_t}(x^{(1)}), ..., \mathbf{k_t}(x^{(t)}))^T$$
$$\mathbf{\Sigma_t}(x) = \mathrm{diag}(\sigma_1^2, ..., \sigma_t^2)^T.$$

$$\mathbf{H_t} = \begin{bmatrix} 1 & -\gamma & 0 & \dots & 0 \\ 0 & 1 & -\gamma & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \dots & 1 & -\gamma \end{bmatrix}$$

and rewrite our reward model as

$$R_{t-1} = H_t Q_t + N_{t-1} \tag{1}$$

Then the standard results on jointly Gaussian random variables gives the posterior on Q given R_{t-1} as

$$Q(x)|R_{t-1} \sim \mathcal{N}\big(v_t(x), p_t(x)\big), \tag{2}$$

where

$$v_t(x) = \mathbf{k_t}(x)^T \mathbf{H_t}^T \mathbf{Q_t} R_{t-1},$$
$$p_t(x) = k(x,x) - \mathbf{k_t}(x)^T \mathbf{H_t}^T \mathbf{Q_t} \mathbf{H_t} \mathbf{k_t}(x),$$

with $\mathbf{Q_t} = (\mathbf{H_t K_t H_t}^T + \mathbf{\Sigma_{t-1}})^{-1}$

3.1 Sparsification

To reduce the computation required to compute v_t and p_t, Engel et al. (2005) uses a sparsification method that represents $k(\cdot,\cdot)$ as the inner product of the Hilbert space $\mathcal{H}$, i.e. $k(x,x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$. We can then reduce computation by maintaining a dictionary of $\{\tilde{\phi}(x_i)\}$, with which we can approximate $\phi(x_{(i)})$ by linear projection:

$$\phi(x_i) = \sum_j a_{ij} \tilde{\phi}(x_j) + d$$

Every x where d is greater than a precision threshold ν is then added to our dictionary, and our estimates of the GP mean and covariance can be rewritten as

$$\tilde{v}_t(x) = \tilde{\mathbf{k}}_\mathbf{t}(x)^T \tilde{\mathbf{H}}_\mathbf{t}^T \tilde{\mathbf{Q}}_\mathbf{t} R_{t-1},$$
$$\tilde{p}_t(x) = k(x,x) - \tilde{\mathbf{k}}_\mathbf{t}(x)^T \tilde{\mathbf{C}}_\mathbf{t} \tilde{\mathbf{k}}_\mathbf{t}(x).$$

More details on how $\tilde{\mathbf{k}}_\mathbf{t}, \tilde{\mathbf{H}}_\mathbf{t}, \tilde{\mathbf{Q}}_\mathbf{t}, \tilde{\mathbf{C}}_\mathbf{t}$ are calculated from the dictionary are in Engel et al. (2005).

3.2 Experiment and Results

We tested this framework on larger mazes and compared its performance with the Bayesian MDP framework in Strens (2000). In this problem, states are defined in integer coordinates (x, y), and actions can be one of 8 choices of neighbors. We note that this framework allows continuous state and action spaces, but to reduce complexity of testing and implementation, we tested on the "hard Maze" in Figure 2 and the "very hard Maze" in Figure 6. In these problems, however, the agent had twice as many action choices.

First, when comparing the performance on the "hard Maze" (Figure 5), the GPSARSA learner performed significantly better than the Bayesian learners we implemented in section 2. In 5000 time steps, the GPSARSA learner achieved a reward of 148, compared to less than 60 for all other learners. The optimal path achieves 2 every 19 steps, so the GPSARSA learner is operating significantly closer to the optimal of 210 in 2000 steps. On the "very hard Maze", the GPSARSA learner again significantly outperformed the Bayesian learners, but none were operating close to optimal (8 points every 24 steps).

We believe the better performance is due to an improved exploration ability when compared to the Bayesian learners. In particular, the Bayesian learners seemed to do a poorer job in learning the uncertainty of its reward function. The GPSARSA learner, on the other hand, seemed to explore the state space much better, and had much better estimates on the uncertainty of its reward. In addition, the GPSARSA learner executed significantly faster than the Bayesian learners; the 20,000 step test took 30 seconds for the GPSARSA learner, but 10 minutes for the Bayesian learners.

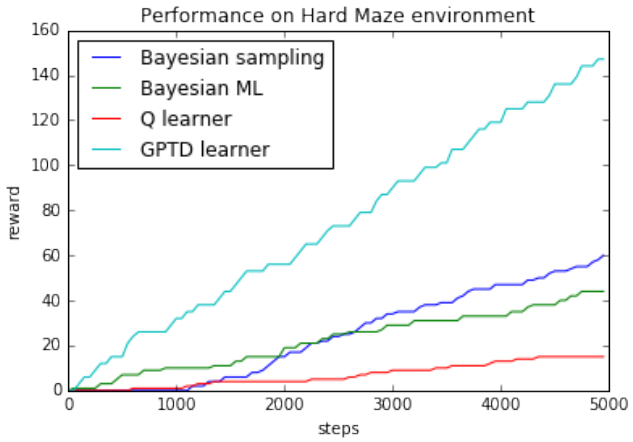

Figure 5: Comparison of GPTD learner against other learners in the hard "Maze" environment.

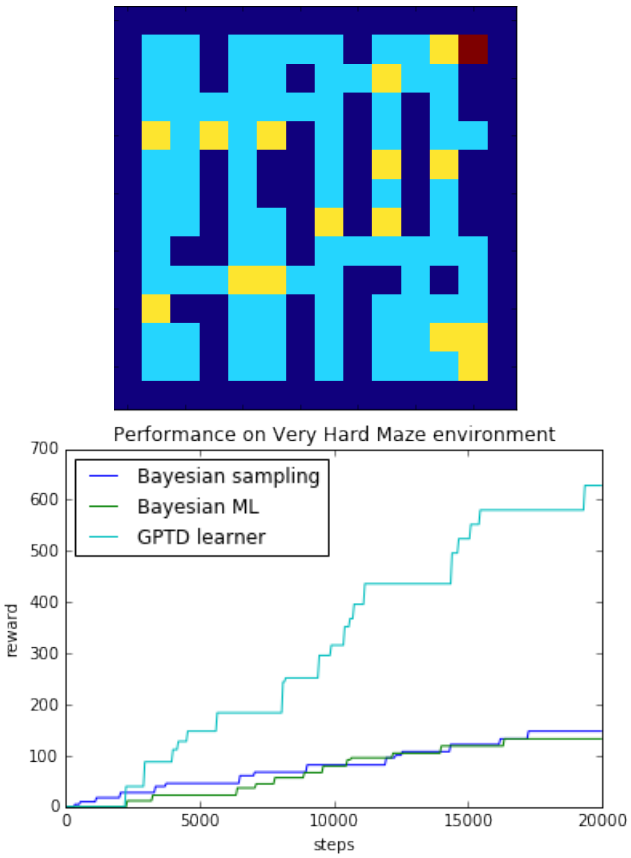

Figure 6: Pictured above is the very hard "Maze" environment. Pictured below is the performance of GPTD learner against the Bayesian learners on very hard "Maze" environment.

4 Source Code

All of our source code is accessible at `https://github.com/alexandrwang/6882project`.

References

Engel, Y., Mannor, S., and Meir, R. (2005). Reinforcement learning with gaussian processes. In *International Conference on Machine Learning*.

Strens, M. (2000). A bayesian framework for reinforcement learning. In *International Conference on Machine Learning*.