

Chapter 1

Bayesian Q Learning

In an attempt to find a better balance between exploration and exploitation this thesis investigates the use of bayesian methods to allow for Thompson sampling. This chapter builds and compares bayesian methods in a linear model context before attempting to extend the most successful methods to neural networks.

1.1 Linear Q learning

In linear Q learning the goal is to create a regression model that maps the state and action to a Q-value, $Q(s, a)$. Let x_t denote the state and action at timestep t . X then denotes the design matrix containing these features and Q the vector of corresponding Q-values. The regression model for a single action can then be defined as

$$Q(X, a) = X\beta + \varepsilon \quad \text{where} \quad \varepsilon \sim N(0, \sigma^2)$$

with the response value defined as

$$Q(s, a) = r_t + \arg \max_{a'} Q(s', a'). \quad (1.1)$$

The ordinary least squares solution to the β coefficients can then be found using the normal equation which in matrix form is

$$\beta = [X^T X]^{-1} X^T Q$$

Given this model the agent can take an action by acting greedily over the models $Q(s, a)$ values in a given state. Since this purely an exploitation strategy, it is often coupled with the ε -greedy policy.

1.2 Bayesian Linear Q learning

To extend linear Q learning methods to follow a Thompson sampling policy a bayesian perspective is required. In a RL perspective the goal is to model the posterior

$$p(\theta|Q, \mathcal{F}) \propto p(Q|\theta, \mathcal{F})p(\theta)$$

$$p(Q) = \int p(Q|\theta, \mathcal{F})p(\theta)d\theta$$

where Q is a vector of all Q-values given the state X_t , θ denotes all parameters and \mathcal{F} denotes all previous transitions. In RL the value of interest are samples from $p(Q|\theta, \mathcal{F})$, not it's actual value.

The calculation of an arbitrary posterior is computationally heavy which is ill-suited to the already long running reinforcement learning methods. In order to keep computation costs low to start off this thesis will consider conjugate priors which have an analytical solution.

1.2.1 Normal Prior with Known noise

There are multiple ways to setup a bayesian regression model using conjugate priors. First consider the case used in Azizzadenesheli et al. (2019) which creates one model per action and assumes the noise variance is known. The known noise variance is then treated as a hyperparameter. In this case the posterior can be expressed as

$$p(\beta_a|Q_a, \sigma_{\varepsilon_a}, \mathcal{F}) \propto p(Q_a|\beta_a, \sigma_{\varepsilon_a}, \mathcal{F})p(\beta_a)$$

$$p(Q_a|\sigma_{\varepsilon_a}, \mathcal{F}) = \int p(Q_a|\beta_a, \sigma_{\varepsilon_a}, \mathcal{F})p(\beta_a)d\beta_a$$

In literature it is common to use a gaussian prior for β

$$p(\beta) = N(\mu, \sigma_{\varepsilon}\Lambda^{-1})$$

where Λ is the precision matrix. This results in the following posterior update

$$\Lambda_n = X^T X + \Lambda_0$$

$$\mu_n = \Lambda_n^{-1}(\Lambda_0 \mu_0 + X^T Q_a)$$

Note that when picking actions the sampled Q -value is used. However when calculating the target Q -value the MAP estimate of β is used instead. In this case the MAP estimate is μ .

1.2.2 Normal Prior with Unknown noise

To avoid the noise variance as a hyperparameter it can be included as an unknown parameter.

$$p(\beta_a, \sigma_{\varepsilon_a} | Q_a, \mathcal{F}) \propto p(Q_a | \beta_a, \sigma_{\varepsilon_a}, \mathcal{F}) p(\theta)$$

$$p(Q_a | \mathcal{F}) = \int p(Q_a | \beta_a, \sigma_{\varepsilon_a}, \mathcal{F}) p(\beta_a, \sigma_{\varepsilon_a}) d\beta_a d\sigma_{\varepsilon_a}$$

The conjugate priors for this setup are

$$p(\sigma^2) = \text{InvGamma}(\alpha, b)$$

$$p(\beta | \sigma^2) = \text{N}(\mu, \sigma^2 \Sigma)$$

with the posterior update

$$\Lambda_n = (X^T X + \Lambda_0)$$

$$\mu_n = \Lambda_n^{-1}(\Lambda_0 \mu_0 + X^T Q)$$

$$\alpha_n = \alpha_0 + \frac{n}{2}$$

$$b_n = b_0 + (Q^T Q + \mu^T \Lambda_0 \mu_0 - \mu_n^T \Lambda_n \mu_n)$$

Once again μ , the MAP estimate of β is used to calculate the target Q -value.

TODO:: These results are outdated

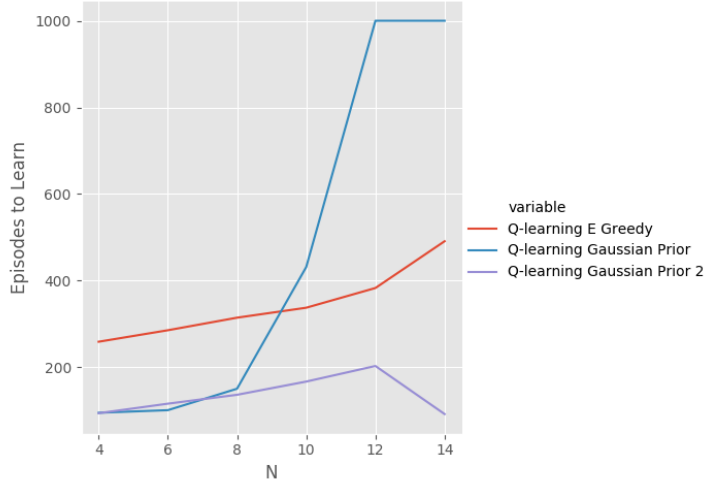


Figure 1.1: WIP: Performance on chain environment Gaussian Prior 2 includes the noise as a parameter

1.2.3 Propagating Uncertainty

One possible issue with the two above methods is training using the MAP estimate of β . Using the MAP estimate means that the targets come from the following process:

$$y = R + \max_a X_{t+1} \mu_a.$$

Though this does incorporate the variance in the reward process through R it does not convey the variance in the Q-value estimate of the next state. Even in a deterministic environment the policy shifts during training mean that there is an uncertainty in the Q-value of the next state. Quoting Moerland et al. (2017), "...repeatedly visiting a state-action pair should not makes us certain about its value if we are still uncertain about what to do next."

One possible method to include this uncertainty is to sample the β posterior when calculating the target value.

TODO:Here is where I want stronger argumentation of why this should work (other than intuition).

TODO:These results are outdated

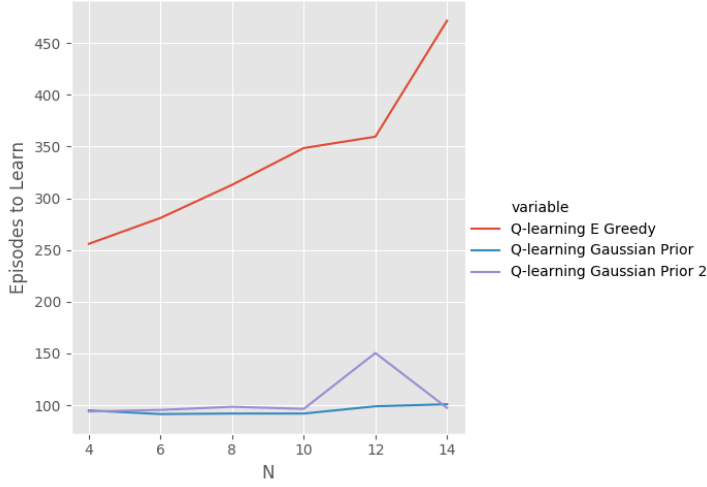


Figure 1.2: WIP: Performance on chain environment Gaussian Prior 2 includes the noise as a parameter

1.2.4 Investigating propagation

To ensure that these methods are propagating uncertainty we can consider a simple example from Osband et al. (2018). Consider a MPD with two states. The initial state allows only one action that deterministically leads to state 2 with no reward. State 2 is a terminal state with a known posterior distribution.

If a RL method properly propagates uncertainty the posterior distribution of state 1 should match state 2 as long as $\gamma = 1$. To test the following setup was used The priors for the known noise models were set to

$$\begin{aligned}\beta &\sim N(0, 10^3) \\ \sigma^2 &= 1\end{aligned}$$

and the priors for the unknown noise models were set to

$$\begin{aligned}\beta &\sim N(0, 10^3) \\ \sigma^2 &\sim \text{InvGamma}(1, 1).\end{aligned}$$

Three MDP's were set up with a known posterior of $N(1, 0.1)$, $N(1, 1)$ and $N(1, 10)$ respectively. The results are seen in figure 1.3.

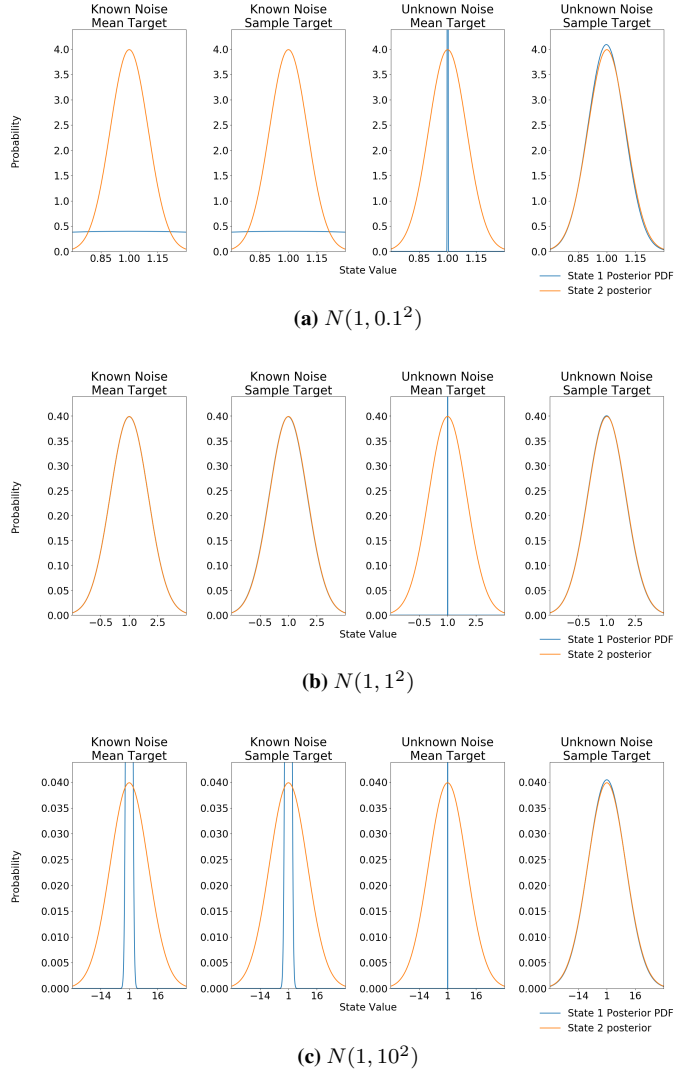


Figure 1.3: Variance Propagation On 2 State Toy Example: It is clear that the known noise models result in a normal distribution around the correct mean but with the assumed noise level for variance. The unknown noise model using the mean target leads to a variance that heads towards 0. The only model that is able to correctly model the posterior is the unknown noise model with sampled targets.

Based on these results focus is placed on the unknown noise model with sampled targets. Now consider a modification to the environment where an extra state is placed between the initial and terminal state. This state has the same dynamics meaning it deterministic transitions to the terminal state with zero reward over the transition. This allows an investigation over how well variance is propagated through multiple states.

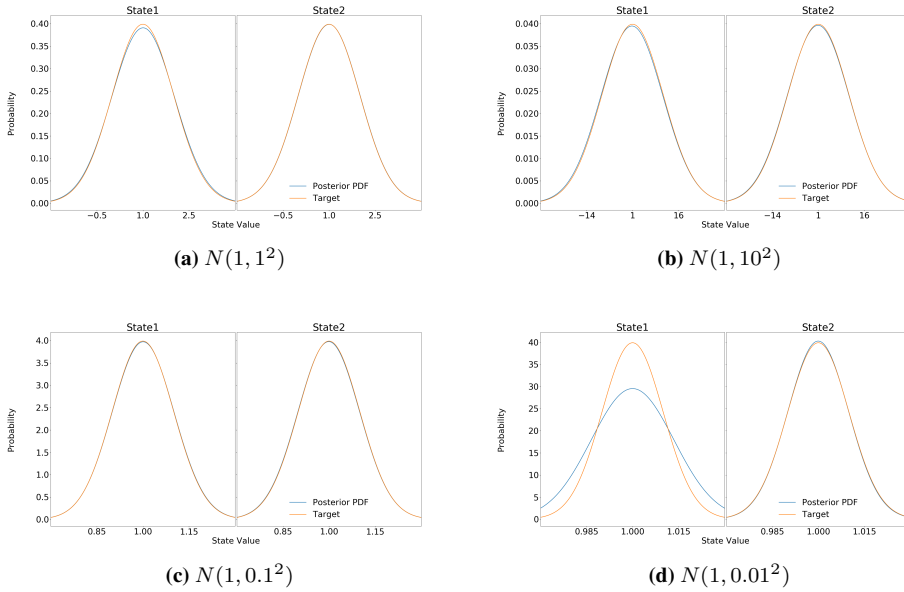
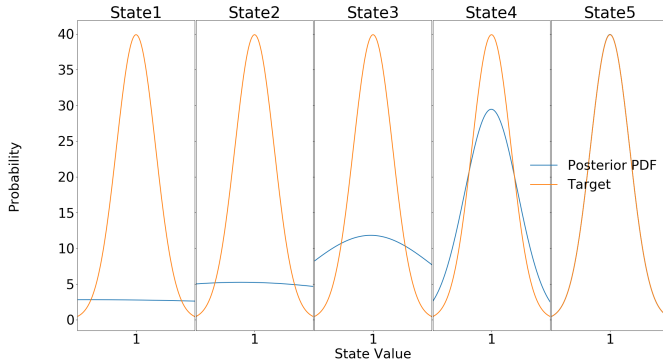
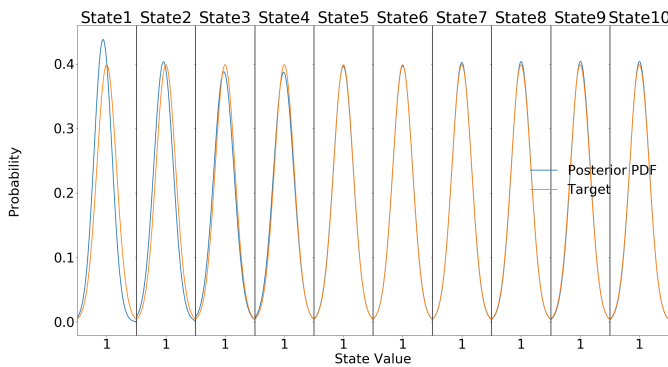


Figure 1.4: Variance Propagation On 3 State Toy Example: For larger variance targets the propagation correctly updates the state 1 variance. However the low variance targets results in an over-estimation over the variance which worsens further away from the known posterior.

One reason the 0.01 standard deviation posterior is harder to approximate than the 1 sd is that the sensitivity of the posterior to the α and β parameters. Small changes in these for a distribution with low variance leads to large changes (TODO:: Confirm that this is actually the case)



(a) 6 States with $N(0.01, 1^2)$



(b) 11 States with $N(1, 1^2)$ target

Figure 1.5: Failure of variance propagation over many states: (a) shows that the error in estimation close to the terminal state leads to failure in the estimation of the posterior of the initial states. In (b) the seemingly correct estimation close to the terminal state still does not prevent some errors from occurring closer to the initial state.

1.2.5 Temporary Stability is (maybe) not a problem, propagation is just slow

Sensitivity of the Q-value

With better priors stability doesn't seem to be a problem. I was using a suggested weak prior from Gelman which was $\alpha = 0.001$ and $\beta = 0.001$ which leads to inf values in the Scipy library. Essentially But I'm leaving this here cause the reasoning behind this does cause issues. In addition I was using the wrong Invgamma parameterization. This probably caused some instability as well.

Slow propagation

A problem is the "speed" of propagation. The issues shown in figure 1.5 are dealt with by just training for a longer time. However, the amount of training required for good results seems to grow exponentially with the number of states to propagate over.

Keep in mind that the next state which is used as the current states target. I think the slow propagation is due to the wrong target values used to train the bayesian regression before the next states posterior is correct. The further away from the actual environment return, the longer time it takes before the next states posterior is correct which means the model is trained on more bad data.

I think retraining the model from scratch at certain intervals fixes the second part of the issue. To speed up the time it takes for the uncertainty to propagate we can also consider using n-step updates.

1.3 Bayesian Deep Q Network

TODO:this section assumes DQN has been explained

The predominant issue with bayesian methods in deep reinforcement learning is using bayesian methods with neural networks. This thesis will address the linear layer method (**TODO:**Actual name for method), a simple and computationally efficient method that comes at the cost of accuracy.

The final layer in a DQN is a linear layer. Since bayesian regression is also a linear combination one can replace the final layer with a bayesian regression model per action. This is equivalent to rewriting the regression task to

$$Q = \phi(X)\beta + \varepsilon \quad \text{where} \quad \varepsilon \sim N(0, \sigma^2)$$

where $\phi(X)$ is the neural networks output given an input X . Note that this means the bayesian regression no longer incorporates all the uncertainty since the above assumes no uncertainty in the $\phi(X)$ encoding.

Training the model now needs to be split into two processes. Firstly the bayesian regression is trained using the posterior update shown above. The neural network is trained using a similar loss function as the DQN. However the networks Q-value estimate is replaced by the MAP estimate of β resulting in

$$\theta = \theta - \alpha \nabla_{\theta} (Q_t - [\mu_n^T \phi_{\theta}(x_t)])^2.$$

Note that these do not have to happen sequentially. In Azizzadenesheli et al. (2019) and this implementation the bayesian regression is updated less often than the neural network.

Finally to deal with the fact that reinforcement learning is a non-stationary problem the the bayesian regression is trained for scratch each time it is updated.

Bibliography

Azizzadenesheli, K., Brunskill, E., Anandkumar, A., 2019. Efficient exploration through bayesian deep q-networks. CoRR abs/1802.04412.

URL <http://arxiv.org/abs/1802.04412>

Moerland, T. M., Broekens, J., Jonker, C. M., 2017. Efficient exploration with double uncertain value networks. CoRR abs/1711.10789.

URL <http://arxiv.org/abs/1711.10789>

Osband, I., Aslanides, J., Cassirer, A., 2018. Randomized prior functions for deep reinforcement learning, 8617–8629.

URL <http://papers.nips.cc/paper/8080-randomized-prior-functions-for-deep-reinforcement-learning.pdf>

