Table of Contents

List of Tables

List of Figures

Chapter 1

Conjugate Bayesian Linear 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 to investigate what models to use and what factors are important in a reinforcement learning setting.

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

$$Q(X, a) = X\beta + \varepsilon \quad \text{where} \quad \varepsilon \sim N(0, \sigma^2) \tag{1.1}$$

with the response value defined as

$$Q(s, a) = r_t + \arg\max_{a'} Q(s', a'). \tag{1.2}$$

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 is 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 Thompson sampling a bayesian perspective is required. To do this a prior distribution is placed over the regression parameters. Using bayes rule the posterior distribution of the parameters can be calculated and used to calculate the marginal distribution over Q.

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

Q is a vector of all Q-values given the state X_t, θ denotes all parameters and $\mathcal{F}$ denotes all previous transitions. Since this is only used for Thompson sampling the value of the integral is not of interest. Instead it is the samples from $p(Q|\theta, \mathcal{F})$ that will be used to drive exploration.

1.3 Conjugate Bayesian Linear Q learning

TODO:Emphasize the assumptions in this chapter

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

1.3.1 Gaussian Prior with Known noise

To start consider the model used in Azizzadenesheli et al. (2019). As in frequintist regression the noise term ε is a zero mean gaussian distribution with variance σ_ε. In Azizzadenesheli et al. (2019) σ_ε is assumed known. This is rarely the case, but even if it is unknown it can be treated as a hyperparameter that has to be tuned to the environment.

A regression model is created per action, each with the same noise variance σ_ε and with a prior over every regression coefficient. The posterior Q-value for each action is expressed as

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

A common conjugate prior for the coefficients is the gaussian distribution

$$p(\beta_a) = \mathrm{N}(\mu_a, \sigma_\varepsilon \Lambda_a^{-1})$$

where Λ_a is the precision matrix. With this choice of prior the posterior distribution of β_a is still gaussian with a closed form update for the distribution parameters. Given new state-action combinations X and target action-values Q^a the posterior can be updated using

$$\begin{aligned} \Lambda_{a_n} &= X^T X + \Lambda_{a_0} \\ \mu_{a_n} &= \Lambda_{a_n}^{-1}(\Lambda_{a_0}\mu_{a_0} + X^T Q_a). \end{aligned} \tag{1.3}$$

where the $_0$ denotes the prior and $_n$ denotes the posterior parameters. The development of these updates can be found in the appendix(TODO:reference). This model will be refered to as a bayesian normal model(BN).

With this setup actions can now be picked by Thompson sampling. For each action sample β values from its posterior distribution and a noise term from $N(0, \sigma_\varepsilon)$. These sample values are then used in the regression equation 1.1 to get a posterior sample from Q_a. Finally chose the action with the highest sampled Q-value.

When calculating the target Q-value Azizzadenesheli et al. (2019) does not use Q-value samples. Instead the MAP estimate of β is used which in this case is μ.

1.3.2 Propagating Variance

Using the MAP estimate means that the targets are calculated by

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

This does not correctly encorporate the target variance. To see why recall the definition of the Q-value

$$\begin{aligned} Q_t &= \mathbb{E}[G_t] = \mathbb{E}[r_t + r_{t+1} + \dots] \\ &= \mathbb{E}[r_t + Q_{t+1}] = \mathbb{E}[r_t + Q_{t+1}] \end{aligned}$$

This results in the regression problem $\mathbb{E}[r_t + Q_{t+1}] = X\beta_a$. However, since the expected reward is unknown this cannot be used. Instead one has access to the sample rewards from the environment. Asymptotically the mean of the samples approaches the expected value so this can be treated as a regression task with a noise term

$$\begin{aligned}\mathbb{E}[r_t + Q_{t+1}] &= X\beta_a \\ r_t + Q_{t+1} &= X\beta_a + \varepsilon\end{aligned}$$

where ε accounts for the difference between the sample and the mean, $r_t - \mathbb{E}[r_t] + Q_{t+1} - \mathbb{E}[Q_{t+1}]$. This implies that the target used must be a sample from the posterior of Q_t not its expected value $X\mu_a$ as used in Azizzadenesheli et al. (2019).

The result of this is that the known noise model only includes 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."

Based on the above a better choice of target is

$$y = r_t + \max_a (X_{t+1}\beta + \varepsilon)$$

where β is sampled from its posterior and ε from the gaussian noise distribution. However the variance of β, as seen in equation 1.3, is independent of the target. One way to include a variance term that is dependent on the target is to include σ_ε as an unknown parameter.

1.3.3 Normal Prior with Unknown noise

Including σ_ε as an unknown parameter resuts in the new posterior

$$\begin{aligned}p(\beta_a, \sigma_{\varepsilon_a} | Q_a, \mathcal{F}) &\propto p(Q_a|\beta_a, \sigma_{\varepsilon_a}, \mathcal{F})p(\theta) \\ p(Q_a|\mathcal{F}) &= \int p(Q_a|\beta_a, \sigma_{\varepsilon_a}, \mathcal{F})p(\beta_a, \sigma_{\varepsilon_a}) d\beta_a d\sigma_{\varepsilon_a}.\end{aligned}$$

The conjugate priors for this setup are

$$\begin{aligned}p(\sigma^2) &= \text{InvGamma}(\alpha, b) \\ p(\beta|\sigma^2) &= \text{N}(\mu, \sigma^2\Sigma)\end{aligned}$$

with the posterior update

$$
\begin{aligned}
\Lambda_n &= (X^T X + \Lambda_0) \\
\mu_n &= \Lambda_n^{-1}(\Lambda_0 \mu_0 + X^T Q) \\
\alpha_n &= \alpha_0 + \frac{n}{2} \\
b_n &= b_0 + (Q^T Q + \mu^T \Lambda_0 \mu_0 - \mu_n^T \Lambda_n \mu_n)
\end{aligned}
$$

The development of these formulas can be found in the appendix (TODO:reference). This model will be refered to as the Bayesian Normal Inverse Gamma model(BNIG).

1.3.4 Testing Variance Propagation

To ensure that this method is propagating uncertainty consider a simple example from Osband et al. (2018). Consider a MPD with two states. The initial state allows only one action that deterministicly 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 $\gamma = 1$.

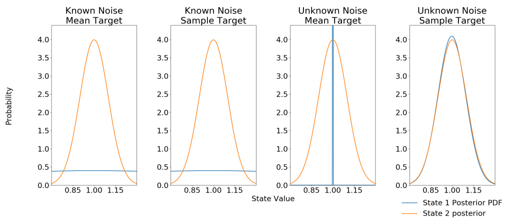
Both models were tested with both MAP and sample targets. The priors for the BN models were set to

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

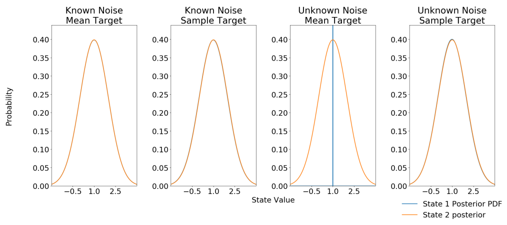
and the priors for the BNIG models were set to

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

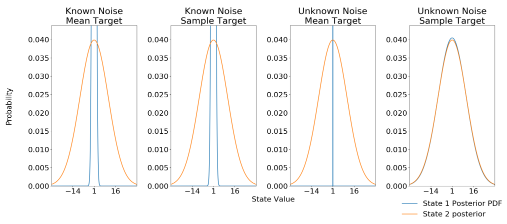
Three MDP's were set up with a known posterior of $N(1, 0.1), N(1, 1)$ and $N(1, 10)$ respectively. The results are seen in figure 1.1.



(a) $N(1, 0.1^2)$



(b) $N(1, 1^2)$



(c) $N(1, 10^2)$

Figure 1.1: Variance Propagation On 2 State Toy Example: The blue lines show the models Q-value posterior distribution while the orange lines show the target posterior distribution. Only the BNIG model with sample targets is able to correctly estimate the target in all cases.

The results summarized in figure 1.1 showed that all models were able to correctly estimate the mean. However the target variance was only correctly estimated by the the BNIG model with sample targets. The BNIG model with the MAP target leads to the correct mean but dramatically understimates the variance. This would be an expected result if the model is approximating $\mathbb{E}[Q]$ instead of Q. The BN model is only correct for both the mean and sample target if the hyperparameter ε is set to the correct variance. In an unknown and more complex environment this is unlikely to be possible. However with enough hyperparameter tuning one could argue that this can lead to good results which might explain the results achieved in Azizzadenesheli et al. (2019).

Based on these results further developments are focused on the BNIG model with sampled targets.

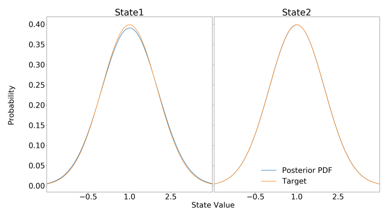
1.4 Variance Propagation

1.4.1 Over Multiple States

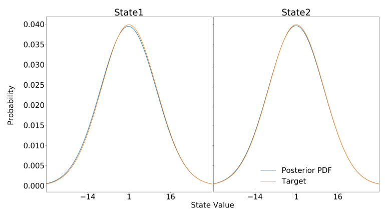
The setting above is a regular regression setting. In a RL setting the variance needs to be propagated to further states. To test that this is still the case with the BNIG model consider a modification to the environment where an extra state is placed between the initial and terminal state. This state has the same dynamics as earlier. It deterministically transitions to the next state with zero reward over the transition. The correct posterior for each state is then the target posterior in the terminal state.

The same priors as earlier are used on 4 different target posteriors. The results are summarised in figure 1.2.

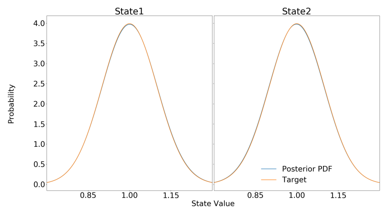
TODO:Add details around these experiments in the appendix



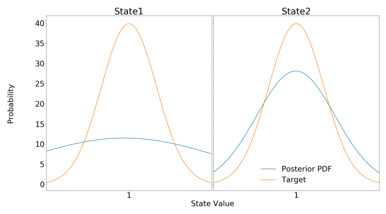
(a) $N(1,1^2)$



(b) $N(1,10^2)$



(c) $N(1,0.1^2)$



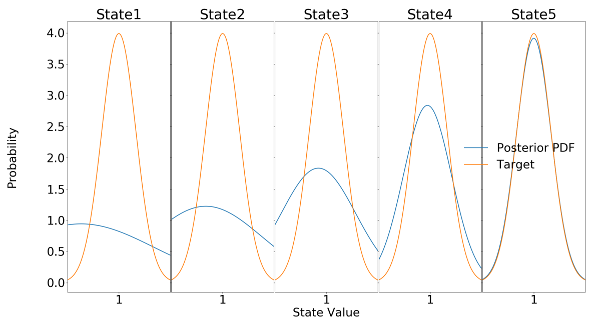
(d) $N(1,0.01^2)$

Figure 1.2: Variance Propagation On 3 State Toy Example: The models posterior estimate for the two first states are shown. The third state is the terminal state that returns a sample from the target distribution.

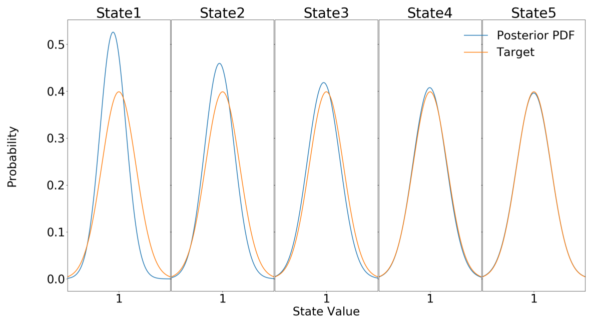
Figure 1.2 shows that for larger variance targets the propagation correctly updates the state 1 variance. However the low variance targets results in an overestimation over the variance in state 1. Running this experiment for more iterations does lead to a better approximation implying that the problem lies in the the convergence rate for different posteriors.

One possible reason for this is that the posterior representing the 0.01 standard deviation posterior is more sensitive to small changes in in its parameters than the larger variance posteriors. In other words small changes in the parameters for a distribution with low variance leads to large changes in the variance of the posterior. Since the learning is happening in an online fashion the first estimates of the posterior will likely have large error. This effect is amplified for state 1 since it is training on the large error state 2 posterior. (TODO:: Confirm/source that this is actually the case)

By extending the toy example to even more states as in figure 1.3 one can see that this problem increases the further the variance needs to be propagated. Even large variance targets will fail to correctly propagate given enough states.



(a) 6 States with $N(0.1, 1^2)$



(b) 6 States with $N(1, 1^2)$ target

Figure 1.3: 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 errors closer to the initial state.

This issue will be refered to as the speed of propagation. It encompasses the problem of quickly propagating variance estimates from downstream states back to states which are far from the environments reward.

TODO:: Define speed of propagation. Maybe a seperate subsection that plots how the error changes with length of chain. How do we measure the error between two distributions?

KL Divergence?

1.4.2 Speed of propagation

Inorder to improve the speed of propagation insight is required into what causes the issue. In this section two causes and methods for counteracting them are discussed. However these are not necessarily the only two causes of slow propagation.

TODO:These argumentations are weak and require sources or ”proof”.

The first cause is best explained using an example. Consider the 3 state toy example. State 1 can only converge to the correct distribution once State 2 has converged. If more states are added, State 1 will converge once all the states between it and the terminal state have converged. The longer the chain is, the more iterations are required for state 1 to converge. This issue is the same issue faced with the expected Q value in regular RL which is dealt with through a bias-variance trade-off (TODO:Include n-step in theory). Similarly extending the 1-step update to an n-step update one will increase the speed of variance propagtation with the downside of introducing more variance to the σ_ε estimate.

The second cause is a result of no longer using a step size when updating the Q-values using bayesian updates. In regular Q-learning the step size can be viewed as the weighting of new data relative to the current model. This effect is also found in the bayesian setting in the posterior update that combines the prior and the new data. However, in regular Q-learning the step size also leads to the model forgetting old data(Sutton and Barto (2018)TODO:find this page). This does not happen in the bayesian regression setting that has been used.

In the bayesian regression setting described the prior is always the previous posterior. In simple terms the model assumes all data to be equally important. Recalling that the prior used is the previous posterior, a prior based on many datapoints will have a bigger impact on the posterior than a single new datapoint. This is a problem since a reinforcement learning problem is almost always non-stationary due to changes in policy. With this weighting scheme the new data points which are more relevant to the current target are weighted the same as a datapoint collected based on the initial priors.

Counteracting this effect while retaining the bayesian regression model is not trivial. The solution used in Azizzadenesheli et al. (2019) is to define a hyperparameter T_{bayes} and train a new bayesian regression model from scratch using targets from the old model every T_{bayes} steps. However this is a computationally heavy step and can greatly increase the run time of the algorithm for problems with many predictors.

To avoid this a concept called exponential forgetting is used. This is first mentioned in Dearden et al. (1998) but with no reference and no explanation to what it is. (TODO:write down best source). The method reduces the impact of previous data by exponentially decaying old data.

Rather than keeping track of the entire dataset used one can consider the terms used to update the posterior seen in equation 1.4. The terms X^TX, X^Ty, y^Ty and n are all proportional to the number of predictors rather than the number of datapoints. The terms can then be exponentially decayed by using the updates

$$(X^TX)^{(k+1)} = \alpha X^TX^{(k)} + x^Tx \tag{1.4}$$

$$(X^Ty)^{(k+1)} = \alpha X^Ty^{(k)} + x^Ty \tag{1.5}$$

$$(y^Ty)^{(k+1)} = \alpha y^Ty^{(k)} + y^Ty \tag{1.6}$$

$$n^{(k+1)} = \alpha n^{(k)} + b \tag{1.7}$$

where α is the decay rate and the b is the batch size.

1.5 Deep Exploration

Osband et al. (2018)

1.6 Performance on Linear RL Problem

Up to now the model has only been tested on simple problems with known posteriors and no active decision making. To test how well this generalizes to a more realistic RL environment consider a variant on the corridor environment introduced in the theory section in figure **??**.

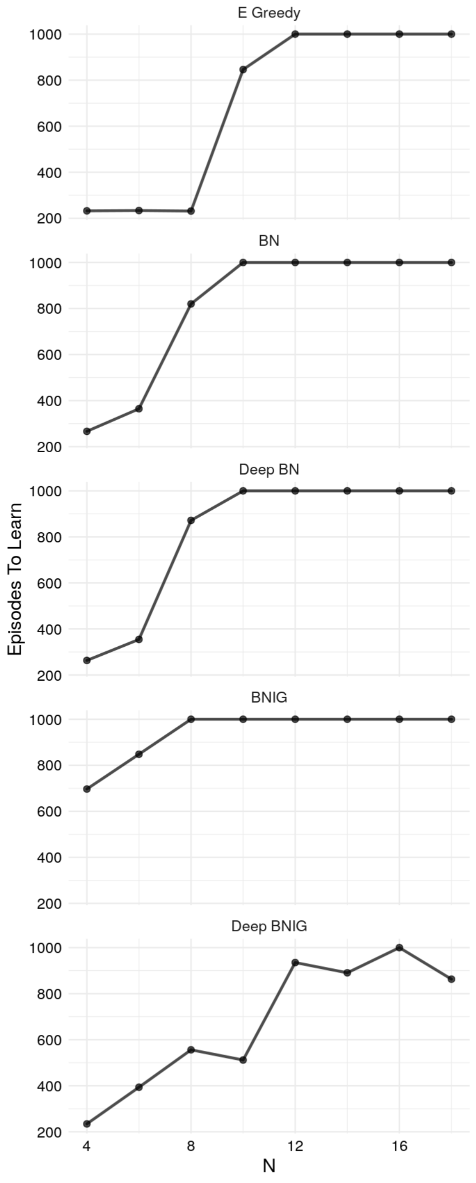
Consider a chain of length N where the goal is to move from the initial state on the left side of the corridor to the final state on the right side of the corridor in N steps. A reward of 1 is given in a final state, moving left gives a reward of $\frac{1}{10N}$ and otherwise the agent recieves no reward.

TODO:Add regret to theory section

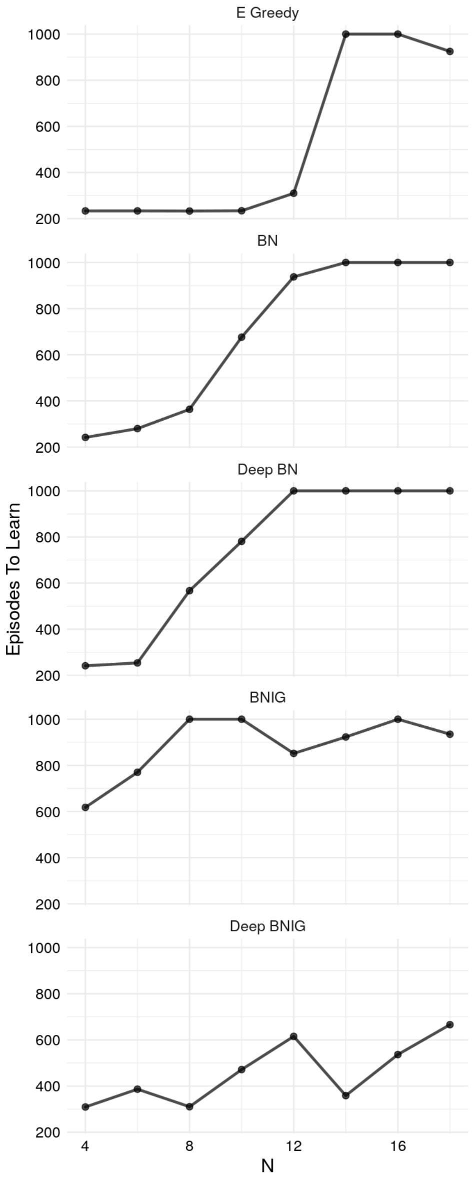
All the methods discussed so far were tested on this enviroment with different chain lengths. This includes the BN and BNIG model with both regular and deep exploration. In addition a linear Q-learning method following an ε-greedy exploration scheme was tested. The hyperparameters for each model were found by manually testning different combinations. The priors that gave good propagation for the toy examples were used as a starting point. A table of the hyperparameters can be found in the appendinx in table TODO:.

As in Osband et al. (2018) models were compared by the average number of episodes required to "learn" the enivorment. In this case the enivorment is considered learned when the running average regret of the last 100 episodes drops below 0.1.

TODO:These results are based on 10 attempts per length, this should be run for 50-100 attempts for nicer plots.



(a) 1 step update



(b) 3 step update

Figure 1.4: Linear Model Performance on Corridor Environment: (**a**) shows that the only method to out perform is the Deep BNIG model. Increasing to a 3-step update in (**b**) improves the performance of all models but the largest increase in performance seems to be Deep BNIG.

The key result from figure 1.4 is that the ε-greedy method outperforms all methods apart from the BNIG model with deep exploration. However there are a few interesting differences in the results between the bayesian models and their exploration methods.

The regular BNIG model performs worse than the regular BN model. One reason for this could be the constant noise term allowing for exploration even when the model is certain about it's current estimate. If the BNIG model quickly learns that a left moving policy gives a reward, the variance for the left action might approach zero before it gets enough data showing that a right moving policy gives a higher reward. A BN model will result in the same issue, but the small additional variance might provide enough exploration to manage a slightly longer corridor. An improvement to this would be to have the variance term σ_a^2 be dependent on the state.

Adding deep exploration has seemingly no effect on the BN model but leads to far better results on the BNIG model. This could be due to the BN model not propagating variance quickly leading to a low variance model. Once the variance is low the difference between sampled action-value functions is also small so sampling each step versus sampling each episode will lead to similar actions.

Finally, increasing from a 1-step to a 3-step update leads to better performance across all models. However for all models apart from deep BNIG the improvement is increasing the longest learnable corridor by around 2-4 states. In deep BNIG's case the improvment is an increase of atleast 8 states and possibly more. Once again this is probably due to the BNIG model being the only one actually propagating uncertainty. By increasing the step update in the BNIG model one is not only decreasing the bias of the expected value but also decreasing the bias of the variance. In other words the n-state update not only improves the value estimation but also improves the exploration, resulting in a larger improvment than in other methods.

1.7 Temp? State Dependent Inverse Gamma Distribution

TODO:This might not be worth including but I'm writing it down so we can discuss if it's worth working on.

TODO:I originally worked on this to stabalize beta (which no longer is a problem) but realized it could be used with a model. I've tested a bit on corridor with a linear model with OK results, but need to test more to actually conclude that this works. But I wont work more on this until we have discussed this and after I've come further on my thesis.

The variance of a state should be dependent on the state, which is not the case with the BNIG model. To fix this the inverse gamma distribution parameters should be dependent on the state. With some linear algebra it is possible to rewrite the b term in terms of the next state variance and temporal difference. First recall the parameter updates

$$a_n = a_0 + \frac{n}{2} \tag{1.8}$$
$$b_n = b_0 + \frac{1}{2}(Q^T Q - \mu_n^T X^T X \mu_n) \tag{1.9}$$

where some terms are removed as a prior mean of 0 is used.

Denote y as the target, $\hat{Q}$ as the action-value of the next state and Q the action-value of the current state. The target is then distributed by $N(r + \hat{Q}, \sigma_{a'}^2)$. Using the moments of a normal distribution one finds

$$\begin{aligned}
\mathbb{E}[Q^T Q] &= n\sigma_{a'}^2 + (r + \hat{Q})^T (r + \hat{Q}) \\
\mathbb{E}[b_n] &= b_0 + \frac{1}{2}\left(n\sigma^2 + (r + \hat{Q})^T (r + \hat{Q}) - Q^T Q\right) \\
&= b_0 + \frac{1}{2}\left(n\sigma_{a'}^2 + (r + \hat{Q} - Q)^T (r + \hat{Q} + Q)\right) \\
&= b_0 + \frac{1}{2}\left(n\sigma_{a'}^2 + \delta^T (r + \hat{Q} + Q)\right) \\
&= b_0 + \frac{1}{2}\left(n\sigma_{a'}^2 + 2\delta^T y - \delta^T \delta\right)
\end{aligned} \tag{1.10}$$

where δ is the temporal difference and the last line reduces the number arguments required for the calculation. This leaves an equation that is dependent on the variance of the next state and the temporal difference error. With this setup σ_a^2 can be replaced by a model $\sigma_a^2 = X\beta_{\sigma_a}$ that trains on targets $\sigma_{a'}^2 + 2\delta^T y - \delta^T \delta$.

Chapter 2

Neural Linear Bayesian Regression Model

A major drawback of the methods discussed in chapter 1 is that they are all linear models. These cannot perform well in environments with complex non-linear relationships between state-action pairs and Q-values without significant feature engineering. Recent developments in the RL field focus on deep RL(Mnih et al., 2015, 2016; Silver et al., 2017) where neural networks are used as encode these relationships allowing succesfull results on complex games. As such it it would be beneficial to be able to combine the methods from chapter 1 with more complex models. This chapter attempts this and tests the new model on multiple complex environments with comparisons to other popular deep RL methods.

2.1 Combining Bayesian Q-learning with Neural Networks

2.1.1 Neural Linear Model

Without significant feature engineering a linear model cannot generalize to more complex non-linear relationships between state-action pairs and their Q-values. However using bayesian methods with non-linear models can be difficult and computationally heavy. Riquelme et al. (2018) compared a large array of bayesian models on a set of bandit environments. They found that accurate complex models often performed worse than simpler approximate methods. The suggested reason for this is that complex models require more data and training to acheive reasonable variance estimates. Since RL is an online task this can lead to miscalibrated variance early on in the training process that leads to worse results.

Emperically Riquelme et al. (2018) finds what they coin as a neural linear model to work best. The model consists of using a neural network as a basis function that is used as the covariates to a linear bayesian regression model. 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. The error in the bayesian regressions point estimates is backpropagated through the neural network to learn a useful basis function. Note that this means the bayesian regression no longer incorporates all the uncertainty since the above assumes no uncertainty in the $\phi(X)$ encoding. Riquelme et al. (2018) suggests that error that comes with this assumption is counteracted by the models stable unceraintty estimates.

This setup allows the application of the methods discussed in chapter 1 in more complex environments. It is also this method Azizzadenesheli et al. (2019) follows in their application of the BN model to more complex models.

2.1.2 Bayesian DQN Models

Based on the results of Riquelme et al. (2018) this thesis attempts to combine neural networks and the BNIG model through a neural linear setup. To start off a summary of the archicture used in Azizzadenesheli et al. (2019), called the BDQN, is provided. This is used as a base which will modified to fit with the BNIG model and thus allow for better variance propagation.

BDQN Model

The BDQN architecture starts with the same archicture as the standard DQN architecture(Mnih et al., 2015). The final layer of a DQN is a linear layer which means it can be replaced by any linear model. Azizzadenesheli et al. (2019) replaces this with a BN model. This model is trained using the posterior updates described in chapter 1 in equation 1.3. The neural network is trained using the loss function

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

The only difference to a regular neural network is that the networks output estimate is replaced by the MAP estimate of β. One could replace the MAP estimate by samples from the posterior Q. However as shown in the linear case this should have no effect on the final estimate but does have a higher computational cost than simply using that MAP estimate.

These two training processes do not have to happen sequentially. In Azizzadenesheli et al. (2019) the neural network is updated as frequently as in the original DQN implementation, while the bayesian regression trained from scratch every 10,000 steps on either 100,000 datapoints or the entire replay buffer if it contains less. This is done to handle the non-stationarity of the task.

BNIG DQN model

The downside retraining the bayesian regression is that this is computational heavy, especially considering that the final layer in the neural network consists of 512 neurons, meaning the update requires matrix arithmetic with a 512 by 512 matrix. On top of this using a BNIG model requires the target Q-values to be sampled from the posterior. This means every 10,000 steps 100,000 samples must be calculated which requires matrix arithmetic of the same magnitude. Instead this thesis considers the exponential forgetting method. However implementing this requires some extra considerations to ensure that the model remains stable.

Recall that the classic DQN has one online network that is updated each training step and one target network that is updated occaisonally to match the online network. Mnih et al. (2013) found that using this target network to calculate the regression target helped stabalize the algorithm.

With exponential forgetting the bayesian regression is trained continuosly. However, using the online bayesian regression method to calculate targets based on the output from the target network will lead to instability. The regression model can train on thosands of datapoints from the online network between network syncs. Using different network encodings with the same bayesian regression will lead to different results which will increases the loss. An increased loss leads to larger network changes which amplifies the effect leading to instability.

To deal with this issue the same setup that is used for the networks is used for the regression models. Two bayesian regression models are created, one online and one target. The target model is used to calculate the target action-values while the online model is used for decision making. The target model is then updated to the parameters of the online model when the target network is updated. This decreases the loss and resulted in a stable network.

With this approach the only required change to transform this to a BNIG setup is to swap out the bayesian model used in the BDQN with the BNIG model and sample action-values from the posterior for both the online and target models.

2.2 BNIG DQN Results

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`, `arXiv:1802.04412v2`.

Dearden, R., Friedman, N., Russell, S., 1998. Bayesian q-learning, in: Aaai/iaai, pp. 761–768.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. CoRR abs/1602.01783. URL: `http://arxiv.org/abs/1602.01783`, `arXiv:1602.01783`.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A., 2013. Playing atari with deep reinforcement learning. CoRR abs/1312.5602. `arXiv:1312.5602`.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533. doi:`10.1038/nature14236`.

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`, `arXiv:1711.10789`.

Osband, I., Aslanides, J., Cassirer, A., 2018. Randomized prior functions for deep reinforcement learning , 8617–8629URL: `http://papers.nips.cc/paper/8080-randomized-prior-functions-for-deep-reinforcement-learning.pdf`.

Riquelme, C., Tucker, G., Snoek, J., 2018. Deep Bayesian Bandits Showdown: An Empirical Comparison of Bayesian Deep Networks for Thompson Sampling. arXiv e-prints .

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al., 2017. Mastering the game of go without human knowledge. Nature 550, 354–359. doi:10.1038/nature24270.

Sutton, R.S., Barto, A., 2018. Reinforcement learning: an introduction. The MIT Press.