Model-Based Learning with Dyna-Q for Continuous Problems

Fré Haver, Laura Ruis, Mirthe van Diepen and Lois van Vliet Reinforcement Learning, UvA

Introduction

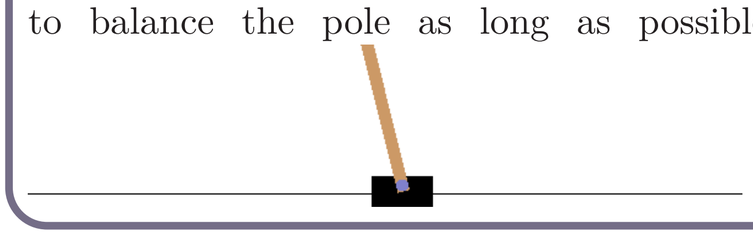
Dyna-Q [1] is an instance of the Dyna architecture [2] introduced by Sutton to make more efficient use of interactions with an environment by learning a model. It combines Q-learning [3] from real and simulated experience. The latter is obtained by learning a model of the environment which is then used for n-step planning. The action-value function is learned with one-step *tabular* Q-learning, which means it can only be used for problems with a *finite number of states*. In this work we adjust Dyna-Q in two different ways to be able to handle infinitely large state spaces. Firstly, the state space can be approximated with a tabular method by *binning values*. Secondly, a continuous function approximator, like a *neural network*, can be learned to model the environment and the action-value function.

Research Question

Can we adjust **Dyna-Q** to work for **non-tabular problems**, where the state space is infinitely large, and **utilize its learned model** to **find a better policy with limited data** than model-free methods?

Environment: Cartpole

The cartpole environment is a task where a cart is moving along a track while keeping a pole from falling over. It is a continuing task where the goal is to balance the pole as long as possible.



Experimental Setup

All experiments are run 5 times and the average result with the variance is reported.

Models:

- *DQN*: neural baseline - Q network
- *Deep Dyna-Q*: Q network, Model network
- *Tabular Dyna-Q*: Q table, Model table
  Binning of states:
  Cart position: $[-2.40, 2.40]$
  Cart velocity: $[-1.50, 1.50]$
  Pole position: $[-0.21, 0.21]$
  Pole velocity: $[-1.50, 1.50]$
  Each of the 4 dimensions of the state space are binned into 10 equal bins. Together with the edge values this gives 12^4 states.

Parameters:

- Neural models: learning rate 0.001, hidden size 128, batch size 64, $\epsilon = 1$ to 0.05 in evenly spaced steps.
- Tabular Dyna-Q: $\epsilon = 0.2$, $\alpha = 0.5, \gamma = 0.8$

Conclusion

Tabular Dyna-Q did not perform well and this wasn't due to the deterministic assumption of the environment. Deep Dyna-Q did outperform the baseline for a small number of interactions, so we can conclude that we successfully adjusted the framework to work for non-tabular problems. The Deep Dyna-Q framework can learn a better policy with limited data than the model-free DQN.

References

[1] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[2] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991.

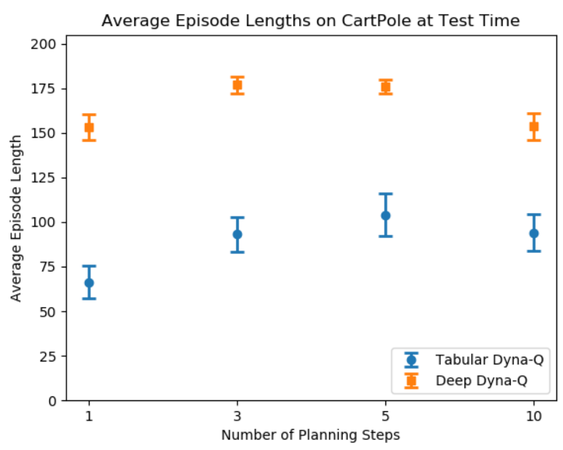
[3] Christopher Watkins. Learning from delayed rewards. 01 1989.

Discussion

In preliminary experiments Tabular Dyna-Q did not perform well, and we suspected this was due to the deterministic assumption of the environment. Even though Cartpole is a deterministic environment when continuous, this seizes to be the case when states are binned. After a particular action, it's possible for a state to stay in the same bin or move to an adjacent bin, which violates the deterministic assumption. To improve the model we instead learned probabilities of next states with maximum likelihood. This gave the following results:
Deterministic Tabular Dyna-Q: mean episode length 87.06 (19.54 - 10 run variance)
Stochastic Tabular Dyna-Q: mean episode length 94.30 (20.61 - 10 run variance)
The stochastic model does improve the average episode length, although it's not significant. Nonetheless, the experiments are run with the stochastic model, since the binned environment is not deterministic.

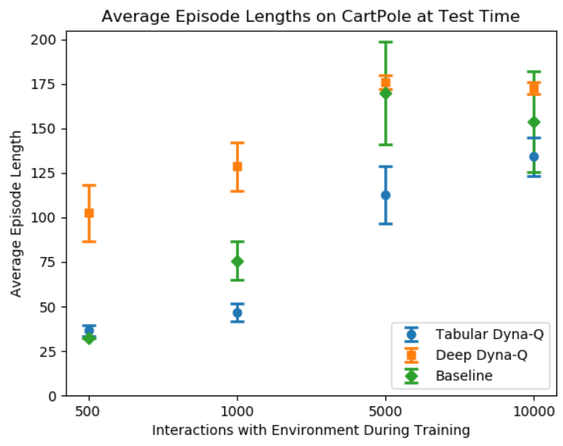
Results

Below, the average episode length of 100 episodes by the models at test time are reported. These results are obtained after the models are trained for a certain number of interactions or updates, as indicated on the x-axis. The performance of Tabular Dyna-Q, Deep Dyna-Q and the baseline are shown together with its 5-run variance.



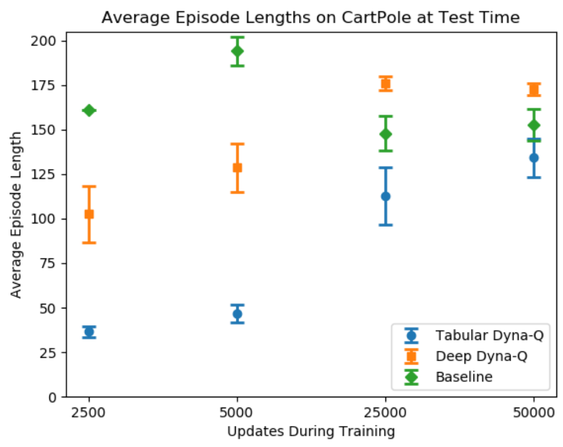
Number of planning steps

- As the number of planning steps increases, so does the average episode length, but for too many planning steps (10) the performance goes down.
- As the number of planning steps increases, decrease in variance for Deep Dyna-Q.
- 5 planning steps performs best, so this is selected for the following experiments.



Number of interactions with the environment during training

- Deep Dyna-Q performs well with low number of interactions and outperforms the model-free method. It can utilize its model for extra updates.
- Baseline performs bad with low number of interactions. As the number of interactions increase, baseline may outperform Deep Dyna-Q.



Number of updates on the Q-function during training

- Baseline performs better with lower number of updates, which is as expected because it uses a real data point for each update, whereas Dyna-Q uses its own learned model for a large part of them.
- Both Dyna-Q methods improve as number of updates increases. The baseline seems to overfit for large number of updates.