Robot Module 2: General Value Functions

Introduction

This module takes what was learned in setting up a testing infrastructure for a reinforcement learning (RL) robot in Module 1 [1] and begins initial implementation of common RL algorithms. The goal of this module is to begin implementation of General Value Functions (GVFs) to predict the sensorimotor stream of our robot. This will be done by implementing 3 learners, two on-policy, and one off-policy. The report is broken down into three sections. First I outline the setup of the experiments. Next I give the details of the experimental results. Finally I outline the conclusions from the experiments.

Experiment Setup

The setup for this experiment is based off of the robot constructed during Module 1 [1]. The goal for this module was to take the previously setup robot and implement three general value functions (GVFs), two on-policy and one off-policy. In the on-policy setting the target policy, what the agent is learning about, and the behavior policy, the policy the agent is using to chose its actions, are the same. [2] There are well-defined algorithms for on-policy learning using function approximation, and here TD(λ) [2] was used to learn the answers to the on-policy questions.

Off-policy GVFs have a different target policy than behavior policy. [2] This is a powerful paradigm, and important to a Horde system [3]. It allows a learning system to learn about many things from one stream of experience. This is especially important in robotic systems as we have only the one stream of data coming from the robot and need to make as best use of it as possible. Learning off-policy value functions with function approximation has been known to diverge at times, however recent algorithmic advances have much more robust convergence guarantees. [2] One such method is GTD(λ) [4], which was implemented to learn the answers to the off-policy question.

For this experiment the behavior policy was to have a Robotis Dynamixel AX-12A servo [5] rotate back and forth between an angle of -2 and 2. In order to iterate on both the development of the algorithms, and to be able to run sweeps over different parameters, a stream of data was recorded from the dynamixel. This allows for rapid testing, without having to have the robot running the entire time. This had the benefit of much faster data throughput for testing, as well as avoiding unnecessary wear on the robot.

In order to focus more on the algorithms themselves, a basic method of function approximation was used in which the feature space was broken down into evenly spaced bins. For all experiments run the feature set was the current angle of the servo, a float between -2 and 2, as well as the direction that the motor is going (No Movement, Clockwise, Counter-Clockwise). For the experiments in this module the agent was only running the clockwise or counter-clockwise actions and the no movement action was only included to keep continuity for future experiments.

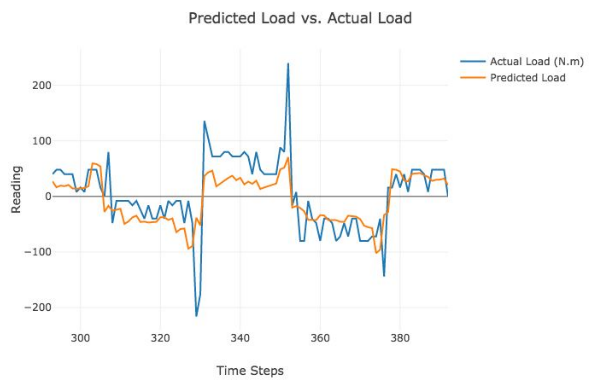
Results

In order to make sure that the algorithms were learning as expected, hand-coded verifiers were built to compare the predictions to. Sweeps were run over step sizes, lambda, and bin sizes to find the best combination for the agent to learn, and do act as a sanity check during development. The results of the sweeps were chosen based on the mean squared error of the agent's prediction compared to its verifier. For all of the sweeps a bin size of 11 for the function approximator was the best which allowed for some granularity in the angle, while not slowing with the need to "fill" a lot of the bins with learning. The best parameters are given below and results of the sweeps can be found in the repo for this project. [6]

On-policy question 1

The first question asked was: "What will the robot's load be at the next time step while following the current policy?". This is an on-policy question as the target and behavior policies are the same. In order to specify this, the cumulant at each step was the agent's load and gamma was set to 0 on all time steps. Thus the agent was trying to predict the one step cumulant value of the load. The verifier for this was simply the actual load at the next step. After running sweeps the best parameter values were 11 bins and alpha of 0.1.

We can see in Figure 1 that after only a few hundred time steps the learner was able to fairly accurately predict the regularities in the load. This prediction continued to improve until beyond times step 1000. At this point the agent reached close to its best accuracy in predicting the load of the dynamixel. One thing to notice is that there were regularities in the stream of data that were not able to be predicted, seen in the jittering on each of side of 0. There did not seem to be any predictable pattern for the learner to pick up on, instead its prediction was a smoothed out prediction of what generally the load was around that time. Additionally the learner was able to predict jumps to either side of the reading, corresponding to the robot changing directions, but the magnitude of the jumps was spararatic and thus the agent made a smooth generalized prediction of that jump.



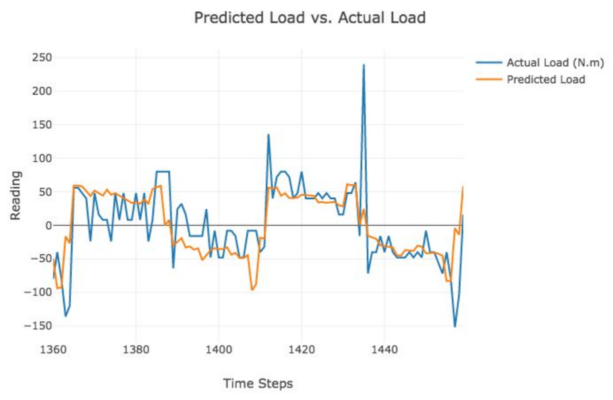


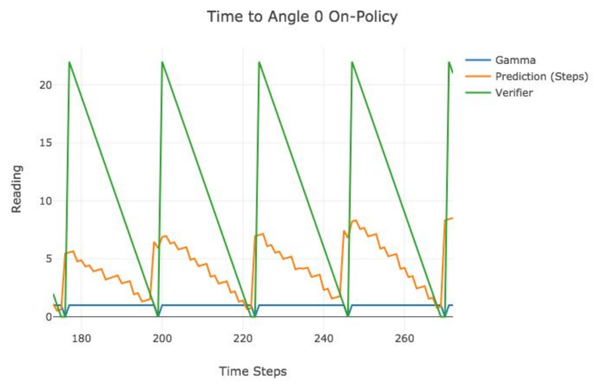
Figure 1. Left: Predictions after a few hundred time steps. Right: Predictions after over 1000 time steps. Note that the sign of the load, and prediction, corresponded to the direction the agent was rotating in. Thus the load prediction was in a direction.

On-policy question 2

The second question asked was: “How long will it take to return to an angle of 0 while following the current policy?”. As above this is an on-policy question. To specify this question we use a cumulant of 1 at each time step to accumulate the steps for the agent. To specify a termination at angle 0 we need to use a time conditional gamma, where gamma is 1 at each time step (since we want to count it), until it reaches angle 0 in which case gamma is set to 0 to terminate. After running sweeps the best parameters were 11 bins, alpha of 0.1, and lambda of 0.99.

In this case a hand-built verifier was made based on what the number of steps “should” be. The verifier was not perfectly accurate as there were times that the agent would move from one side to the other in slightly shorter or longer amounts of time. Additionally the loop time for the agent was not always the same leading to slight differences in the number of steps. Thus the verifier was not meant to be a perfect reflection of how many steps, but a very close approximation to compare the agent’s runtime too. You can see in Figure 2 that it takes the agent approximately 22 steps from the zero angle to move all the way to one extreme (either an angle of -2 or 2) and to return back. The verifier shows this by jumping up to 22 and then slowly returning back to 0 on each step.

As shown in Figure 2 the agent was able to learn to predict the number of steps that it would take to return to the zero angle based on the policy it was following. Because of the bin size the agent was not able to smoothly predict the results. Additionally binning did lead to some random jumps, and some areas where the agent was not able to accurately predict the number of steps, most noticeably near 0 where the 0 angle and the jump in steps were occasionally binned together.



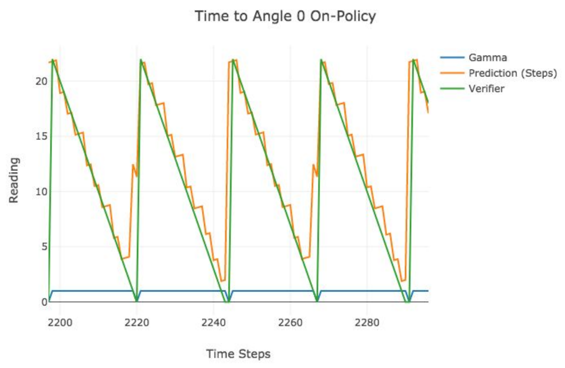


Figure 2: Left: Prediction after 200 time steps. We can see the agent building up it’s prediction and capturing the regularities of the system. Right: after 1500 time steps the agent has nearly completely captured the regularities and magnitude of the data stream.

Off-policy question 1

The third question asked was: “If I were to go back to an angle of 0 from this current state how long would it take me?”. This is an off-policy question as the target policy, return to 0, is different from the behavior policy, go back and forth between -2 and 2. This means that only sometimes will the robot be taking an action that the agent can learn from. As mentioned previously off-policy methods with function approximation have been known to have divergence issues. One algorithm that has solved this is GTD(λ) [4], which was implemented here. After running sweeps the best parameters were 11 bins, alpha of 0.05, beta of 0.005, and lambda of 0.9. One hand-coded piece was that rho, had to be set to 1, even though with importance sampling it should have been 2. However if rho was set to 2 the learner diverged rapidly so it was set to 1 for this experiment.

The question specification was the same as the above, with a cumulant of 1 on every time step, with a state conditional gamma of 1 on every time step and 0 when the robot’s angle was 0. The verifier was again hand coded, but was different than above. At each time step the agent is not considering how long it will take to return following the current policy, instead it is considering how long it will take it it were to return to zero at that moment. You can see the result of this where the cumulant builds up the farther the agent is from zero until a max of 11 steps, and decreases the closer the agent is to 0. It does not have the spike in steps as soon as it leaves the 0 angle as it would if it were following the behavior policy. As above this verifier is not perfect, but a very close approximation of how many steps away the agent would be.

We can in Figure 3 that the agent was able to learn to fairly accurately predict the number of steps it would take to return to 0 after about 10,000 steps. This seems like a long time to learn and likely better parameter tuning, such as increasing alpha and then decreasing it over time would help. You can see it build up the middle section over time as it made learned its predictions. As above binning meant that there were jumps in the prediction rather than an exactly smooth prediction following the verifier. If further accuracy was needed a different function approximation method could be used to provide more granular results. Alternatively smaller bins could be used to provide more granularity, however during sweeps across bin sizes a higher number of bins ended up having a lower accuracy, which was verified by looking at the graphs. This often had to do with a bin not getting enough data to build up an effective prediction.

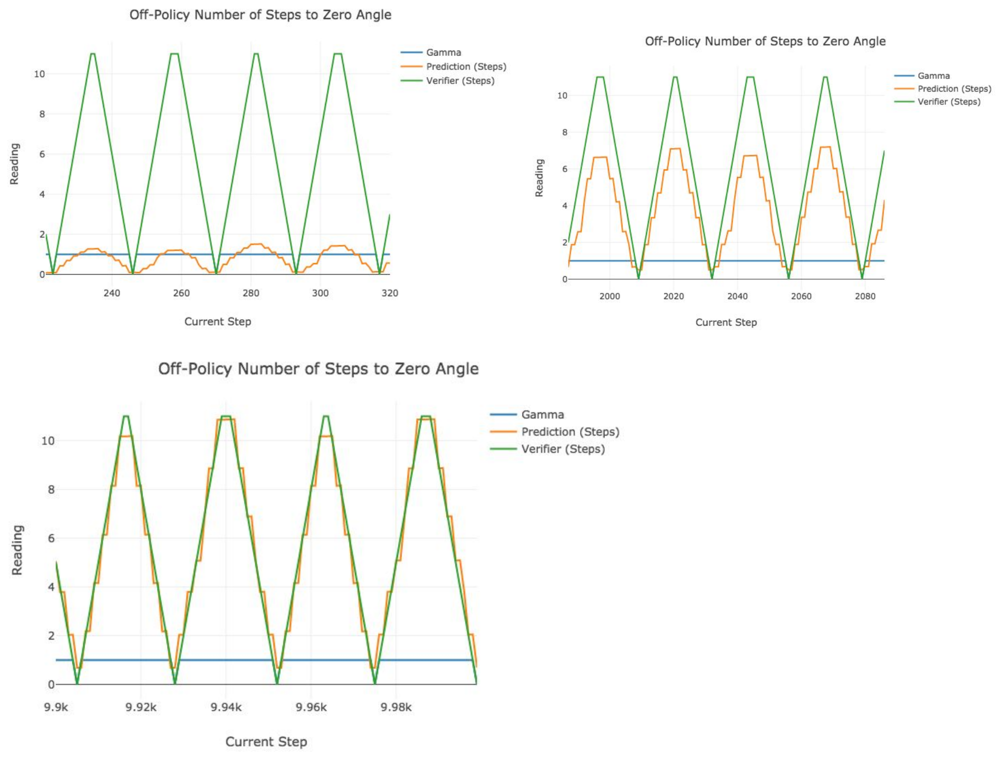


Figure 3: Top Left: initial learning after a few hundred steps. The agent is already learning the general regularities of the stream. Top Right: The agent is building up it's predictions with more and more data, although slowly given it's smaller amount of data and low learning rates. Bottom Left: After about 10,000 steps the agent is quite accurately predicting the time it would take to get back to 0 at any moment in time.

Dashboard

A running dashboard of results was built to keep track of the robot during live operation. This allowed for observation of the robot learning to debug and follow the progress of the learners. It also showed the difference in the learning rates, especially the difference between the on-policy and off-policy learners. Figure 4 shows the ongoing operation of the robot. A short video of the robot and the dashboard in action can be seen online. [7]

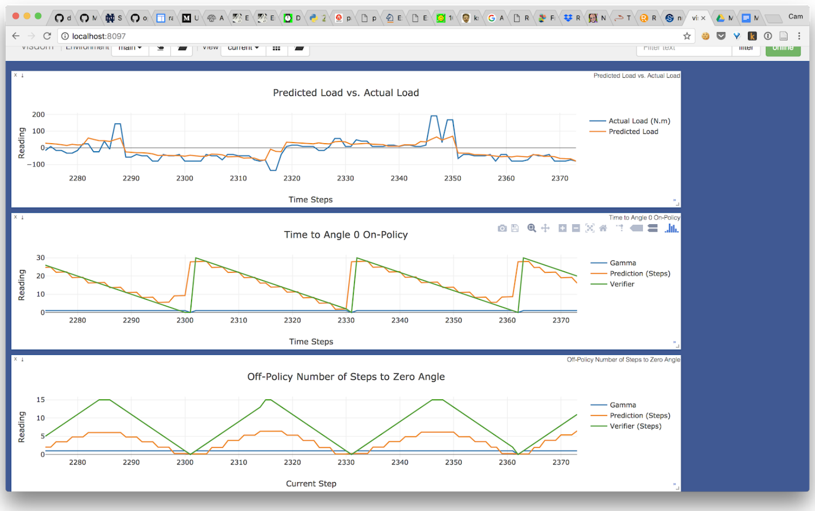


Figure 4: Real-time dashboard of learning robot.

Conclusion

This module showed that a robot's sensorimotor stream can be accurately predicted by General Value Functions. The learners were able to both update their predictions, and graph the results, in real time without slowing the operation of the robot. Further work could be done to improve the predictions: one of the main areas was in the function approximation. Since binning does not generalize across bins it can lead to jagged jumps in predictions. For many cases this is likely good enough granularity, and the simplicity of binning allows for it to be an effective function approximation method. However if further granularity is needed either a very large amount of data is needed to "fill" a higher number of buckets, or a function approximation method that provides granularity and generalizes effectively should be used. Future experiments will likely include some sort of tile coding, and I'm looking at using recently worked on Kanerva coding methods [8] to compare multiple function approximation results.

Code for this module can be found at https://github.com/camlinke/607/tree/master/module2
Video of the learning system in action can be seen at:
https://www.youtube.com/watch?v=wmorda5MDUk&feature=youtu.be

References

1. https://github.com/camlinke/607/tree/master/module1
2. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: an introduction. Cambridge, MA: The MIT Press.
3. R.S. Sutton, J. Modayil, M. Delp, T. Degris, P.M. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems, 2011 Adam White, Joseph Modayil, and Richard S. Sutton.

4. Adam White, Joseph Modayil, and Richard S. Sutton. Scaling life-long off-policy learning. CoRR, abs/1206.6262, 2012
5. http://www.robotis.us/ax-12a/
6. https://github.com/camlinke/607/tree/master/module2
7. https://www.youtube.com/watch?v=wmorda5MDUk&feature=youtu.be
8. Gautham Vasan, Patrick M. Pilarski, "Learning from demonstration: Teaching a myoelectric prosthesis with an intact limb via reinforcement learning", Rehabilitation Robotics (ICORR) 2017 International Conference on, pp. 1457-1464, 2017, ISSN 1945-7901.