Robot Off-policy Prediction: An empirical Comparison of Learning Algorithms

Banafsheh Rafiee, Sina Ghiassian, Adam White, and Richard S. Sutton

RLAI Lab, Department of Computing Science
University of Alberta
{rafiee,ghiassia,amw8,rsutton}@ualberta.ca

Abstract

We empirically compare off-policy algorithms for learning robot prediction tasks. We consider predictions that are made continually, in real time, and during the life of an agent living in a world. The first contribution of this work is to empirically compare off-policy algorithms and the second contribution is to discuss and address the methodological challenges that appear in comparative studies on robots.

The ability to make predictions is central to intelligence. Many researchers have argued that making predictions about a multitude of signals can benefit a learning system in many ways. Modayil and Sutton (2014) argued that predictions can be directly used to improve behavior. They proposed a method that was inspired by Pavlovian conditioning from psychology in which a learner produced a fixed response to an event as it learned to predict it. Jaderberg et al. (2016) showed that learning to predict a multitude of signals can help with learning good features that are useful for improving control performance. Ring (in preparation) proposed a mechanism solely in terms of predictions to build knowledge continually and layer-by-layer from a low-level understanding of the sensorimotor data to high-level concepts. Sutton and Barto (2018) argued that the ability to make predictions about many signals can help form a model of the environment and accelerate learning.

To maintain a system that is built upon predictions, we need reliable algorithms that can be run in real time and applied to the life of an agent living in a world. Many reliable algorithms have been developed in reinforcement learning (RL) that are suitable for such settings. In conventional RL, the predictions are limited to the reward signal. Sutton et al. (2011) proposed to generalize beyond rewards using general value functions (GVFs). Using GVFs, we can make predictions about a multitude of arbitrary signals conditioned on different ways of behaving and different ways of terminating. A GVF, similar to a value function, can be written as the expectation of a weighted sum of a signal of interest:

$$v_{\pi,\gamma,c}(s) = E\left[\sum_{k=0}^{\infty}\left(\prod_{j=1}^{k}\gamma(S_{t+j})\right)c(S_{t+k+1}) \;\middle|\; S_t = s, A_{t:\infty} \sim \pi\right]$$

where $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a target policy and specifies the probability of taking each action at each state. $\gamma : \mathcal{S} \to [0, 1]$ specifies the horizon of the prediction and we call it the horizon function. $c : \mathcal{S} \to \mathcal{R}$ is the signal of interest and is called the cumulant of the prediction.

While GVFs make it possible to formalize predictions about a multitude of arbitrary signals, we also like to be able to learn multiple predictions continually and during interaction with the world. To make lots of predictions each conditioned on an arbitrary target policy while behaving according to a single policy (behavior policy), we can use off-policy learning that is to learn about one policy while following another policy. Unfortunately, the classic RL algorithm, TD(λ), is unstable when combined with linear function approximation under off-policy training (Baird,1995). In the past couple of years,

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

many off-policy algorithms with convergence guarantees have been proposed. However, empirical comparisons of these algorithms in domains suitable for continual learning are missing in the field. In this work, we take a step toward a better understanding of how these algorithms work in practice.

We present two case studies in which we systematically compare several off-policy algorithms learning a robot prediction task formulated as a GVF. Our empirical comparison can be considered the most thorough comparison performed on robots including many algorithms and a wide range of parameters. To perform comparative studies on robots systematically, we had to consider simple tasks. In the first case study, we use a simple robot consisting of just two motors. In the second case study, we consider a more complicated task in which a robot wanders in a relatively large space trying to predict when it is going to bump into something.

1 Off-policy Algorithms

We consider several off-policy temporal-difference learning methods in addition to the classic TD(λ). Gradient-TD methods were the first to provide convergence guarantees with two weight vectors. These methods use stochastic gradient descent to minimize an objective function called the Mean Squared Projected Bellman Error (MSPBE). We consider GTD(λ) and GTD2(λ) (Sutton et al., 2009; Maei, 2011) from this family. It was known early on that TD(λ) can be superior to GTD(λ) in the on-policy case. This motivated the creation of another algorithm that we consider, called HTD(λ) (Hackman, 2012; White and White, 2016). HTD performs TD(λ)-like updates when data is sampled on-policy and GTD(λ)-like updates when data is sampled off-policy, while providing the same convergence guarantees as GTD(λ). Different derivations are possible if we consider saddle point formulation of the objective function. Examples of such derivations are proximal-GTD and proximal-GTD2 algorithms which we also consider in this study (Mahadevan et al., 2014; Liu et al., 2015).

Emphatic-TD methods are the first family to provide convergence guarantees under off-policy training with only one learned weight vector (Sutton et al., 2016). We study the original ETD(λ) (Sutton et al., 2016) as well as a slightly modified version of it, ETD(λ, β) (Hallak et al., 2016). ETD(λ, β) has an extra parameter, β, that works as a bias-variance trade-off parameter.

Off-policy learning can suffer from high variance introduced by the importance sampling ratios. There exist methods that control the effect of importance sampling ratio on updates, not allowing high values of it to affect the update. One such algorithm for learning action values is ABQ(ζ) introduced by Mahmood et al. (2017). This algorithm also uses gradient-based TD updates, achieving stability under off-policy training. In this work, we use a variant of ABQ(ζ) that is used for learning state values, called ABTD(ζ). We also use a simpler version of ABTD(ζ) without gradient corrections.

2 The Dynamixel Case Study

For this case study, we constructed a simple robot, called the Dynamixel robot, consisting of two Dynamixel AX-12 motors. The Dynamixel prediction task is to learn how soon one of the motors reaches a particular target angle, given that it is going back and forth between two limiting angles. The limiting angles have the values of 0 and 1.5 radians and at each time step, the motor moves for 0.05 radians. See Figure 1. We formulated this prediction task as a single GVF. The target policy was to move back and forth between the two limiting angles. The horizon function returned 0.9 unless the distance between the current angle and the target angle was less than 0.05 in which case it returned 0. The cumulant function returned 1 when the distance between the current angle and the target angle was less than 0.05; otherwise, it returned 0.

To approximate this GVF, we used a linear function with a tile coded representation. To construct the feature vector, we used the angle and velocity of the motor. Angle is between -0.1 and 1.6. Velocity is 1 whenever the difference between the current angle and the previous angle is positive and has a value of -1 otherwise. To produce the features, the angle was tile coded using 8 tilings each with 4 tiles resulting in a binary vector of size 8×4. The final feature vector was of size $2 \times 8 \times 4$ where each of the 2 parts corresponded to one of the values of velocity.

We applied the algorithms discussed in Section 1 to this prediction task. We made many instances of them by considering a wide range of parameters including step sizes and trace parameter λ. For

details about the parameters see Appendix A. The behavior policy was to move in the same direction as target policy with probability 0.9 and to move in the opposite direction with probability 0.1. We generated 30 runs of data. Each run consisted of 20000 steps. Learning occurred offline.

Evaluating the learning algorithms on a robot can be challenging. In this domain, like all other robot domains, we do not have access to the true value function. Therefore, we cannot evaluate the algorithms by looking at the difference between the estimated and true values. To evaluate the algorithms, we sampled some time steps following the behavior policy and calculated the return corresponding to those time steps by following the target policy prior to learning. Sampled time steps and returnes formed our evaluation data. During the learning time, we computed the difference between the estimated predictions and the returns and computed the average over the evaluation data. We denote our performance measure by $\widehat{\text{MSRE}}(\mathbf{w})$ because it is an estimation of the mean squared return error (MSRE). For more detail about the performance measure see Appendix B.

The learning curves and parameter studies of the asymptotic performance over step-size for all the algorithms for the case of $\lambda = 0$ are shown in Figure 1. We report the results for the case of $\lambda = 0$ because in this case, the difference between the algorithms was more evident (see Appendix C for results with $\lambda = 0.95$). All the other parameters (e.g., GTD's second step-size) were set to values resulting in the lowest asymptotic performance. To estimate the asymptotic performance, we computed the average of error over the last 0.0025 percent of each run and averaged over 30 runs.

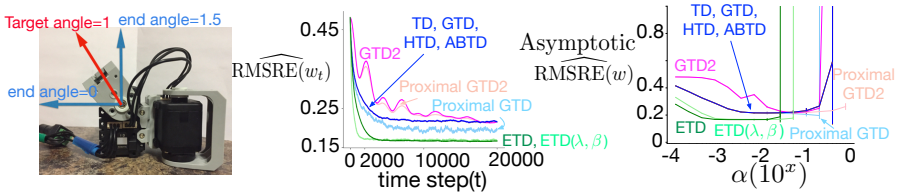

Figure 1: The Dynamixel robot, learning curves, and parameter studies for the case of $\lambda = 0$

According to the learning curves, ETD and ETD(λ, β) substantially outperformed the other algorithms in terms of asymptotic performance and speed with ETD(λ, β) being the fastest. Proximal GTD achieved the next best level of asymptotic error. However, it achieved its best performance for one specific parameter setting whereas ETD and ETD(λ, β) achieved theirs for a wide range of step-sizes according to the parameter studies.

3 The Collision Case Study

For this case study, we used a Kobuki. The robot wanders in a wooden pen and tries to learn how soon it will bump into something if it goes forward. See Figure 3. The sensors available for the learning system to answer this question are the Kobuki's camera and two bump sensors. We formulated this task as a single GVF. The target policy was to pick the forward action in all states. The horizon function returned zero whenever the robot bumped into something and returned 0.97 otherwise. The cumulant function returned a binary value that became 1 whenever either of the bump sensors was on.

To approximate this GVF we used a linear function with tile coded features. To construct the feature vector we used the tile coding software publicly available on Richard Sutton's website. The input to tile coding was 50 RGB pixels randomly selected from the camera, represented as a vector of size 150. We tile coded each vector element separately using 8 tilings each with 4 tiles. (The software uses a hash table. The feature vector size is the same as the size of the hash table – 9600 in our case.)

We applied all aforementioned algorithms to this prediction task. We made many instances of each by considering a wide range of parameters (see Appendix A for details). We used a behavior policy which moved the robot forward with probability 0.9 and made it turn left with probability 0.1.

The evaluation process was similar to that of the Dynamixel case study. However, instead of collecting the evaluation and learning data separately, we collected all the data at the same time. The data collection process consisted of three phases: learning-data collection, excursion, and recovery. See Figure 2. In the learning-data collection phase, the agent followed the behavior policy and the stream

of observations and actions was stored to be used for learning offline. In the excursion phase, the agent switched from the behavior policy to the target policy to compute the return for the state at which the switching happened. The probability of starting an excursion at each time step of the learning-data collection phase was 0.01. At the end of each excursion, we recorded the return and the observations for the state from which the excursion was started. This information was used later for evaluation. After the excursion, the agent entered the recovery phase where it followed the behavior policy for a while, to come back to the distribution of the behavior policy. Following this procedure, we collected 30 runs of learning and evaluation data. Each run contained 150 excursions. Therefore, each run contained 150 streams of learning data and an evaluation data with 150 samples.

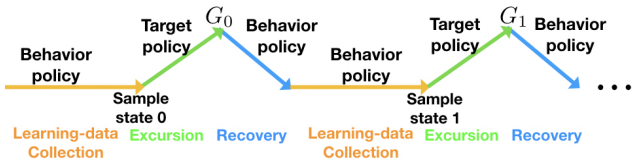

Figure 2: The process of collecting the learning and evaluation data for the collision prediction task.

After collecting the learning and evaluation data, we did learning and evaluation offline. For each run, we concatenated the 150 streams of learning data and applied the algorithms to it to learn the predictions. To evaluate the learned predictions, we computed the difference between the learned predictions and returns and got the average over the evaluation data collected from the 150 excursions.

Parameter studies of the asymptotic performance over the step-size for the case of $\lambda = 0$ is shown in Figure 3 (See Appendix D for $\lambda = 0.95$ results). We report the results for $\lambda = 0$ because $\lambda > 0$ is computationally challenging with neural networks and performance with $\lambda = 0$ may be of more interest for large-scale continual learning. All the parameters were set to values resulting in the lowest asymptotic performance. To estimate the asymptotic performance, we computed the average of the error for the last 100 time steps of each run and averaged over 30 runs.

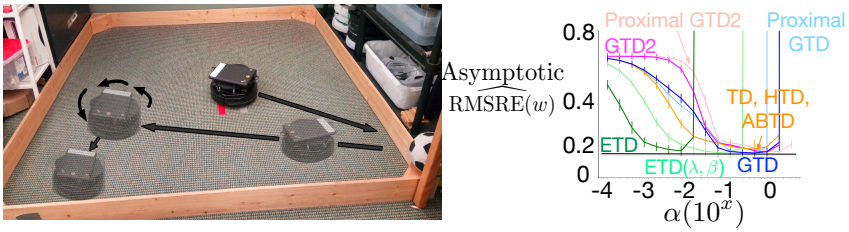

Figure 3: The Kobuki wandering in the pen and the parameter studies for the case of $\lambda = 0$

According to Figure 3, all the algorithms performed similarly in terms of asymptotic performance with ETD(λ, β), GTD(λ), and proximal GTD(λ) reaching a lower level of asymptotic error (more runs would be needed to establish significance). ETD converged with much smaller values of step-size, but still performed well. The β parameter of ETD(λ, β) helped improve the sensitivity of ETD(λ, β) to the step-size over conventional ETD. TD, HTD, and ABTD had the best sensitivity to step-size; however, they converged to a higher level of error.

4 Conclusions

Off-policy learning and GVFs have been recognized as key algorithmic techniques for artificial intelligence and continual learning, making it possible to continually learn about many signals while interacting with the world. In this work, we empirically compared several off-policy algorithms for learning GVFs on robot domains. We also developed an evaluation methodology that was successful and applicable to a relatively complicated robot domain. Our results suggest that several RL algorithms can be useful for learning predictions represented as GVFs, within the computational and sample complexity constraints of simple robots. Based on our results, ETD and ETD(λ, β) are good choices for learning robot prediction tasks.

References

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings*, pp. 30–37.

Hackman, L. (2012). *Faster Gradient-TD Algorithms*. MSc thesis, University of Alberta, Edmonton.

Hallak, A., Tamar, A., Munos, R., Mannor, S. (2016). Generalized emphatic temporal difference learning: Bias- variance analysis. In *AAAI*, pp. 1631–1637.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. ArXiv:1611.05397.

Liu B, Liu J, Ghavamzadeh M, Mahadevan S, Petrik M (2015). Finite-Sample Analysis of Proximal Gradient TD Algorithms. In *Proceedings of the 31st International Conference on Uncertainty in Artificial Intelligence (UAI-2015)*, pp. 504–513. AUAI Press Corvallis, Oregon.

Maei, H. R. (2011). *Gradient Temporal-Difference Learning Algorithms*. PhD thesis, University of Alberta, Edmonton.

Mahadevan, S., Liu, B., Thomas, P., Dabney, W., Giguere, S., Jacek, N., Gemp, I., Liu, J. (2014). Proximal reinforcement learning: A new theory of sequential decision making in primal-dual spaces. ArXiv:1405.6757.

Mahmood, A. R., Yu, H., Sutton, R. S. (2017). Multi-step off-policy learning without importance sampling ratios. ArXiv:1702.03006.

Modayil, J., Sutton, R. S. (2014, July). Prediction driven behavior: Learning predictions that drive fixed responses. In *The AAAI-14 Workshop on Artificial Intelligence and Robotics, Quebec City, Quebec, Canada*.

Ring, M. B. (in preperation). Representing Knowledge as Predictions (and State as Knowledge).

Sutton, R. S., Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. 2nd edition in preparation.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, Cs., Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 993–1000, ACM.

Sutton, R. S., Mahmood, A. R., White, M. (2016). An emphatic approach to the problem of off-policy temporal-difference learning. *Journal of Machine Learning Research 17*(73):1–29.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., Precup, D. (2011, May). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *Proceeding of the tenth International Conference on Autonomous Agents and Multiagent Systems*, pp. 761–768, Taipei, Taiwan.

Adam, A., White, M. (2016, May). Investigating practical linear temporal difference learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pp. 494–502, International Foundation for Autonomous Agents and Multiagent Systems.

Appendices

Appendix A: Experimental Details

For both the Dynamixel case study and the collision case study, we tried a wide range of parameters for all the algorithms. The step-size was in the form $\frac{\alpha}{\text{number of tilings}}$. For the Dynamixel case study α was a number in $\{0.1 \times 2^x | x = -10, -9, ..., 3\}$ and in the collision case study α was a number in $\{0.1 \times 2^x | x = -10, -9, ..., 5\}$. For the gradient based algorithms the second step-size was in the following form: $\alpha_w = \eta \times \frac{\alpha}{\text{number of tilings}}$ where $\eta \in \{2^x | x = -10, -12, ..., 2\}$. The ζ parameter for the ABTD algorithm and the trace parameter λ for all the other algorithms was a number in $\{0, 0.2, 0.4, 0.6, 0.8, 0.9, 0.95, 1\}$. The β parameter for ETD(λ, β) for the Dynamixel case study was a number in $\{0, 0.2, 0.4, 0.6, 0.8, 0.9, 1\}$ and for the collision case study $\beta \in \{0, 0.2, 0.4, 0.6, 0.8, 0.9, 0.97, 1\}$. The TD($\lambda$) and ETD($\lambda$) algorithms are the special cases of ETD(λ, β) when $\beta = 0$ and $\beta = \gamma$ respectively.

Appendix B: Performance Measure

To evaluate the algorithms, in both case studies, we computed an estimation of mean square return error (MSRE) that is the difference between the estimated value and the return, squared and averaged over all states:

$$\text{MSRE}(\mathbf{w}) = \sum_{s \in S} d_b(s) E[(\mathbf{w}^T \mathbf{x}(S_t) - G_t)^2 | S_t = s]$$

where $d_b(s)$ denotes the probability of state s under the behavior policy. w and $x(S_t)$ are respectively the weight vector and the feature vector and their dot product produces the estimation of value of S_t. G_t is the return at time step t.

To compute an estimation of MSRE, we collected some evaluation data consisting of observations sampled from the behavior policy and returns corresponding to those observations computed by following the target policy. Using the evaluation data, we computed the following expression and got its root:

$$\widehat{\text{MSRE}}(\mathbf{w}) = \frac{1}{|D|} \sum_{(s,G) \in D} (\mathbf{w}^T \mathbf{x}(s) - G)^2 \tag{1}$$

where D is the evaluation data.

For both the Dynamixel case study and the collision case study we computed an estimation of MSRE using Equation 1. However, in the Dynamixel case study, we collected the learning and evaluation data separately whereas in the collision case study, we collected them at the same time. For the Dynamixel case study, to collect samples from the behavior policy, we followed the behavior policy for a random number of time steps coming from the uniform distribution of $U(70, 140)$. After sampling a specific time step, we followed the target policy for 70 time steps to compute the return corresponding to that time step. The reason that we used number 70 is that it takes about 60 steps for the motor to complete a cycle following the target policy; therefore, 70 seemed to be a big enough number to be used for this prediction task. Following this procedure multiple times, we sampled 40 time steps and their corresponding return.

For the collision case study, we collected the learning and evaluation data using one process. As we discussed in Section 3, this process included three phases: learning-data collection, excursion, and recovery. The sample time steps were recorded at the beginning of the excursions and the return corresponding to them were recorded at the end of the excursions. We collected an evaluation data consisting 150 samples for each run.

Appendix C: Results for the Dynamixel Case Study for the Case of $\lambda = 0.95$

The learning curves and parameter studies of the asymptotic performance over step-size for the all the algorithms for the case of $\lambda = 0.95$ is shown in Figure 4. According to this figure, all the algorithms performed similarly both in terms of asymptotic performance and sensitivity to step-size.

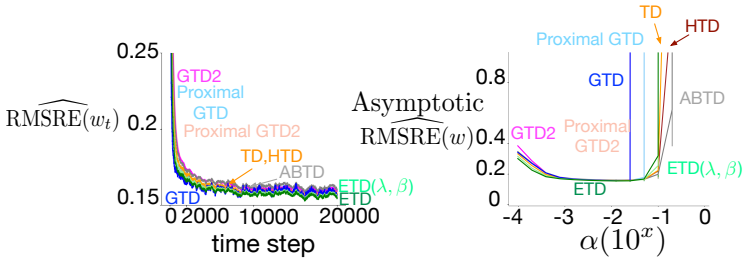

Figure 4: The learning curves (left) and parameter studies of the asymptotic performance over the step-size (right) for the case of $\lambda = 0.95$ for the Dynamixel case study.

Appendix D: Results for the Collision Case Study for the Case of $\lambda = 0.95$

The parameter studies of the asymptotic performance over step-size for the all the algorithms for the case of $\lambda = 0.95$ is shown in Figure 5. According to this figure, all the algorithms performed similarly in terms of asymptotic performance except for GTD2 which reached a higher level of error. ABTD had the best parameter sensitivity to step-size and converged for higher values of step-size.

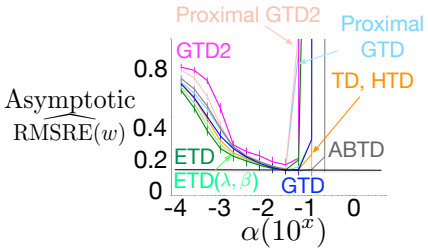

Figure 5: The learning curves (left) and parameter studies of the asymptotic performance over the step-size (right) for the case of $\lambda = 0.95$ for the collision case study.