Applying Variance Reduction Methods to Policy Evaluation for Off-Policy Setting

Farnaz Kohankhaki and Kiarash Aghakasiri

ML Course Project, University of Alberta, Edmonton, Canada

Abstract. One of the main problems in reinforcement learning is computing the value function of each state. This problem is harder to solve when the number of states is large and the agent is behaving off-policy. In this case, we use gradient TD methods that are compatible with off-policy learning and function approximation. These methods try to minimize mean squared projected Bellman error or MSPBE. One challenge in minimizing the MSPBE is its high variance. Stochastic variance reduction methods or SVRG are general methods that help variance reduction. Our goal is to apply SVRG and SAGA, two commonly used variance reduction methods, to reduce the variance of GTD2 and TDC methods. We succeed to reduce the variance (one implementation of variance) for TDC but couldn't achieve the same results for GTD2.

Keywords: Policy Evaluation · Off-Policy · Variance Reduction · Function Approximation.

1 Introduction

Temporal-difference or TD methods are powerful techniques in solving reinforcement learning or RL problems. TD learning can learn from an incomplete sequence of events without waiting for the outcome. In TD learning the current prediction is updated based on the next prediction. [3]

In many RL problems, the number of states is very large. Therefore, it is not practical to compute the value of each state individually(like tabular methods). To solve this problem we use function approximation; we do not compute value functions exactly and instead use approximation techniques. Function approximation allows us to approximate the value of unseen states from observed data. More specifically, the states are mapped to feature vectors that have considerably fewer components than the number of states.

The policy being learned about by the agent is called the target policy, and the policy used to generate behavior is called the behavior policy. In off-policy learning, we seek to learn a value function for a target policy, given data due to a different behavior policy. Freeing the behavior policy from the target policy allows a greater variety of exploration strategies to be used.

TD methods with function approximation have a severe stability problem. Particularly, this stability problem arises when we seek the following four desirable algorithmic features: (1) TD learning, (2) function approximation, (3) off-policy learning, (4) linear complexity both in terms of memory and per-time-step computation.

Gradient-TD methods are based on stochastic gradient descent in a Bellman error objective function. It has been proven that they converge in general settings, including off-policy learning with unrestricted features, and nonlinear function approximation.

In off-policy settings, the objective function that we can consider is mean squared projected Bellman error or MSPBE which we want to minimize in our project. Now, we will go through the math to find how can we minimize MSPBE in linear value function setting with parameter W. First we need to write MSPBE (1) and try to simplify the equation.

$$\overline{PBE} = \|\pi\bar{\delta}\|_\mu^2 = (\pi\bar{\delta})^T D(\pi\bar{\delta}) \tag{1}$$

If we write down the formulas and do some simplification, after some steps we will reach 2.

$$\overline{PBE} = (X^T D\bar{\delta}_w)^T (X^T DX)^{-1}(X^T D\bar{\delta}_w) \tag{2}$$

To minimize $\overline{PBE}$ using Gradient Descent methods we need to take the gradient of $\overline{PBE}$ with respect to w. The equations are shown in 3.

$$\nabla\overline{PBE} = 2\nabla(X^T D\bar{\delta}_w)^T (X^T DX)^{-1}(X^T D\bar{\delta}_w) \tag{3}$$

The next step is to calculate these three factors. The last factor of the gradient can be written as in 4:

$$X^T D\bar{\delta}_w = \mathbb{E}[\rho_t \delta_t x_t] \tag{4}$$

The first factor can also be derived using 4. The formula is shown in 5:

$$\nabla(X^T D\bar{\delta}_w)^T = \nabla\,\mathbb{E}[\rho_t \delta_t x_t]^T = \mathbb{E}[\rho_t \nabla\delta_t^T x_t^T] = \mathbb{E}[\rho_t(\gamma x_{t+1} - x_t)x_t^T] \tag{5}$$

Calculating the middle factor is a little bit different. You can see the formula in 6.

$$X^T DX = \mathbb{E}[x_t x_t^T] \Rightarrow (X^T DX)^{-1} = \mathbb{E}[x_t x_t^T]^{-1} \tag{6}$$

Now we can write the whole formula for the gradient of MSPBE, substituting all the above equations (4, 5, 6) in 3.

$$\nabla\overline{PBE} = 2\,\mathbb{E}[\rho_t(\gamma x_{t+1} - x_t)x_t^T]\mathbb{E}[x_t x_t^T]^{-1}\,\mathbb{E}[\rho_t \delta_t x_t] \tag{7}$$

At this point, we need to calculate these expectations to calculate the gradient. But as the first and third expectations are not independent we cannot use samples because this will give us a bias estimation. Thus, we need to calculate it using a different approach than using a sample for them. Another approach is to calculate each expectation separately and then multiply all of them. The problem with this approach is that it's too computationally complex so we don't want to use this approach either. A more efficient idea is to compute the first two expectations and stored them and then use sampling for the third expectation which is in order of quadratic complexity. In our project, We used two algorithms called GTD(λ) and GTD2 which are based on this idea.[4]

Now we will go through a brief explanation of these two algorithms we mentioned above. The equations here came from [5]. First, we will start with GTD(λ). We have three equations for that.

$$\begin{aligned} e_t &= \rho_t(e_{t-1}\gamma\lambda + x(S_t)) \\ w_{t+1} &= w_t + \alpha(\delta_t e_t - \gamma(1-\lambda)x(S_{t+1})(e_t^T h_t)) \\ h_{t+1} &= h_t + \beta(\delta_t e_t - (h_t^T x(S_t))x(S_t)) \end{aligned} \tag{8}$$

In 8, w is our primary variable and h is our secondary variable. Below are the equations for GTD2 algorithm and the same thing applies for primary and secondary variables in 9 as well.

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma w_t^T x(S_{t+1}) - w_t^T x(S_t) \\ w_{t+1} &= w_t + \alpha(x(S_t) - \gamma x(S_t))(x(S_t)^T h_t) \\ h_{t+1} &= h_t + \beta(\delta_t - h_t^T x(S_t))x(S_t) \end{aligned} \tag{9}$$

The problem with these two methods is that they usually have a high variance, so it will take a long time for them to converge. So, the next step is to use variance reduction methods for GTD(λ) and GTD2 algorithms. In this regard, we want to use SVRG and SAGA which are well-known variance reduction methods in general and want to apply them for policy evaluation as [2]. Now, we will explain a brief description of these variance reduction methods. Assume we have a parameter θ and we want to minimize a loss, let's call it $\mathcal{L}$. Using gradient descent we need to calculate the gradient of the loss with respect to our parameter and then update the parameter like below:

$$\theta = \theta - \eta \nabla_\theta \mathcal{L} \tag{10}$$

In SVRG and SAGA method we don't update our parameter with only one sample of the gradient. In fact, we need to have two samples and the expectation like this:

$$\theta = \theta - \eta(\nabla_\theta^1 \mathcal{L} + \mathbb{E}[\nabla_\theta \mathcal{L}] - \nabla_\theta^2 \mathcal{L}) \tag{11}$$

To find an estimate for the expectation value we used a buffer. At each update in SVRG, we randomly select two points from that buffer and calculate the mean value of the buffer to update the weights. SAGA is very similar to SVRG, the only difference is that we use moving mean methods in SAGA which is less computationally expensive.

2 Methodology

In this section, we will describe the environments, the algorithms, the hyperparameters we chose and the results.

2.1 Environment Setting

For our experiments, we chose a simple MDP, called Chain environment. The environment consists of a predefined number of states (in our experiments used 5 states) and at each state, the agent has two actions to choose between, LEFT or RIGHT. If the agent chooses right in the rightmost state it will result in $+1$ reward and if the agent chooses left in the leftmost state it will result in -1 reward and in these two cases, the episode terminates. All the other transitions have 0 reward. The environment is shown in 1.

$$\begin{tikzcd}
T & s_0 \arrow[l, "-1"'] \arrow[r, leftrightarrow, "0"] & s_1 \arrow[r, leftrightarrow, "0"] & s_2 \arrow[r, leftrightarrow, "0"] & s_3 \arrow[r, leftrightarrow, "0"] & s_4 \arrow[r, "1"] & T
\end{tikzcd}$$

Fig. 1: Chain Environment

2.2 Algorithms

In our experiment first we implemented the two methods we mentioned in the previous section ($GTD(\lambda)$ and $GTD2$), then we applied SVRG these two methods. In this section, we will explain the implementation techniques and present the algorithms.

First we start with $GTD(\lambda)$ and GTD2.

Algorithm 1 GTD(λ), GTD2 Algorithms

```
while Max Epochs do
    while Not Terminated do
        Action = behavior policy(CurrentState)
        Reward, NewState = env.step(Action)
        if algorithm == GTD($\lambda$) then
            Calculate e, w, h with equation 8
        end
        if algorithm == GTD2 then
            Calculate e, w, h with equation 9
        end
    end
end
```

The next thing we mentioned in the previous chapter was applying SVRG to these two methods. Here we will explain the algorithm for adding SVRG.

Algorithm 2 GTD(λ) and GTD2 with SVRG

```
buffer_h = empty
buffer_w = empty
while Max Epochs do
    while Not Terminated do
        Action = behavior policy(CurrentState)
        Reward, NewState = env.step(Action)
        if buffers are full then
            for Range of SamplingRate do
                $\nabla_w^1, \nabla_w^2, \overline{\nabla_w}$ = two samples from buffer_w and the mean of the buffer_w
                $\nabla_h^1, \nabla_h^2, \overline{\nabla_h}$ = two samples from buffer_h and the mean of the buffer_h
                update w, h with equation 11
            end
        else
            if algorithm == GTD($\lambda$) then
                Append $\nabla_w$ to buffer_w with 8
                Append $\nabla_h$ to buffer_h with 8
            end
            if algorithm == GTD2 then
                Append $\nabla_w$ to buffer_w with 9
                Append $\nabla_h$ to buffer_h with 9
            end
        end
    end
end
```

3 Experiments

3.1 Setting

We have run GTD(λ) with λ=0 called TDC and GTD2 methods. Then we applied SVRG and SAGA to them. For the sake of fairness, we created our experiments as follows:

• Weight Initialization: Each of these methods has a secondary and primary variable (h and w). Here we initialize both of them as an array of all zeros.

• Experiance Replay: Because the policy we have here is random, each time our agent interacts with the environment for a while, it will result in a different sequence of rewards and next states. Thus, we used a similar sequence of experience, created by interacting with the environment once, for all these algorithms.

• Fine-Tuning: Each of these methods may act well in a different set of parameters so we cannot use the same parameter set for all of them. Here, we fine-tuned each of them separately by 5 run times using the following list of parameters:

– $\alpha = [2^{-5}, 2^{-7}, 2^{-9}, 2^{-11}]$
– $\beta = [2^{-1}, 2^{-3}, 2^{-5}, 2^{-7}, 2^{-9}, 2^{-11}]$

In the next section, we present the results of running all these 6 different methods after fine-tuning and using experience replay.

3.2 Results

Hyperparameters for each of these algorithms are shown in Table 1.

	α	β	γ	**Buffer Size**	**Sampling Rate**
TDC	0.0078125	0.0078125	0.9	-	-
GTD2	0.03125	0.5	0.9	-	-
TDC+SVRG	0.00048828125	0.00048828125	0.9	100	10
GTD2+SVRG	0.001953125	0.0078125	0.9	100	10
TDC+SAGA	0.03125	0.5	0.9	100	10
GTD2+SAGA	0.03125	0.5	0.9	100	10

Table 1: Hyperparameters

Figures 2-7 show the MSPBE result over epochs for each of the methods. It can be derived that SVRG reduces the variance of the MSPBE error.

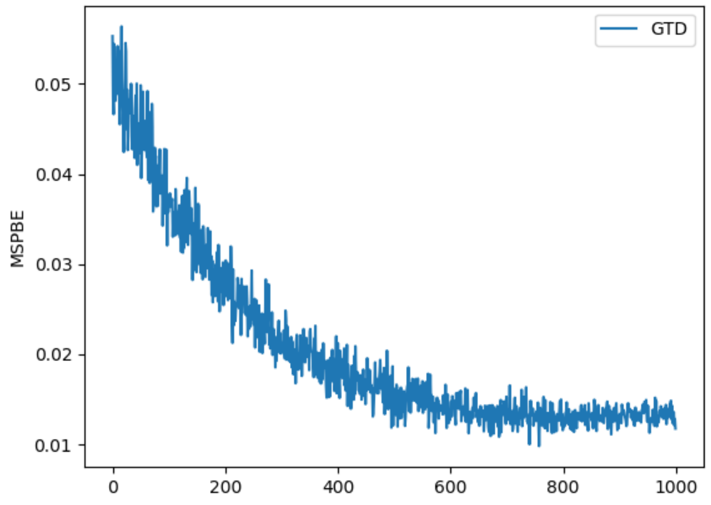

Fig. 2: MSPBE for TDC

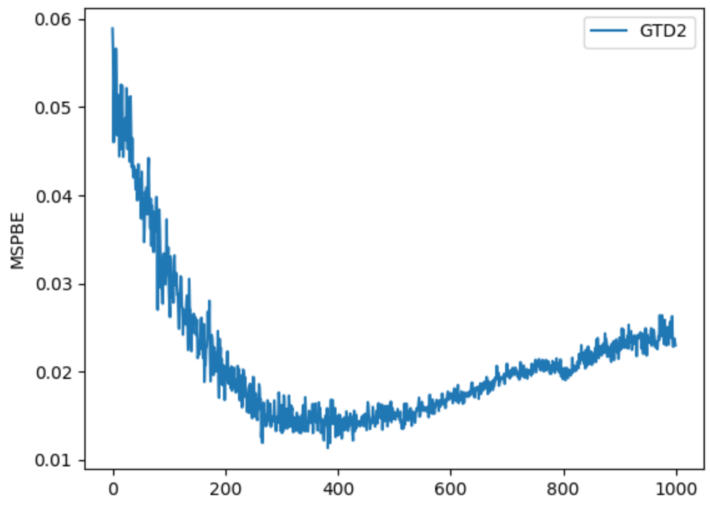

Fig. 3: MSPBE for GTD2

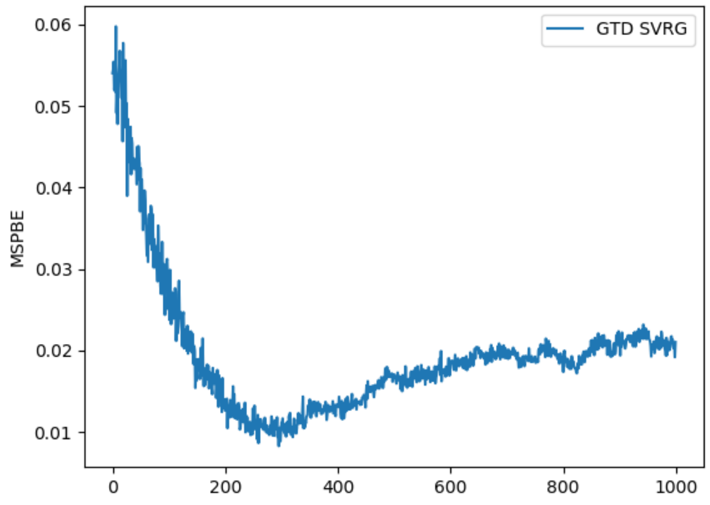

Fig. 4: MSPBE for TDC with SVRG

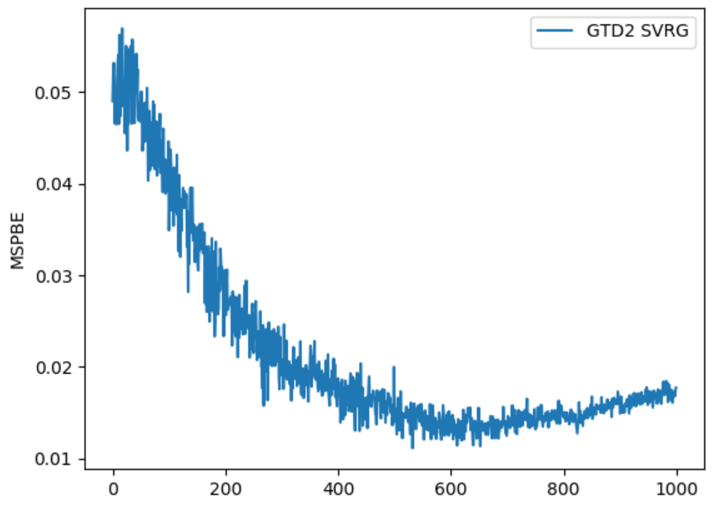

Fig. 5: MSPBE for GTD2 with SVRG

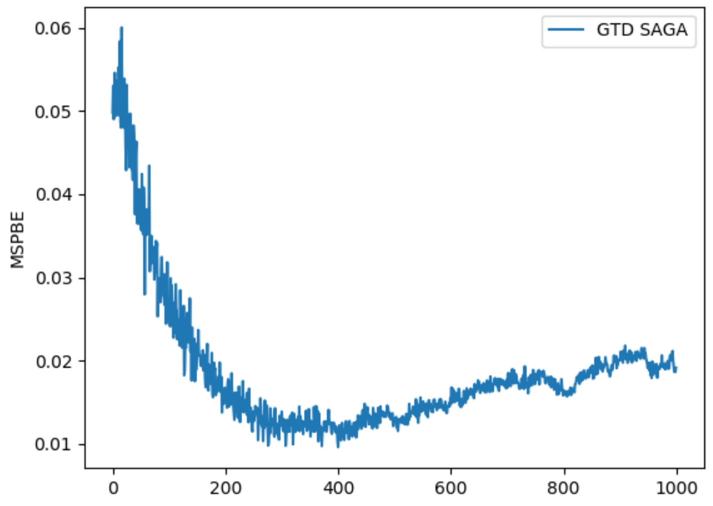

Fig. 6: MSPBE for TDC with SAGA

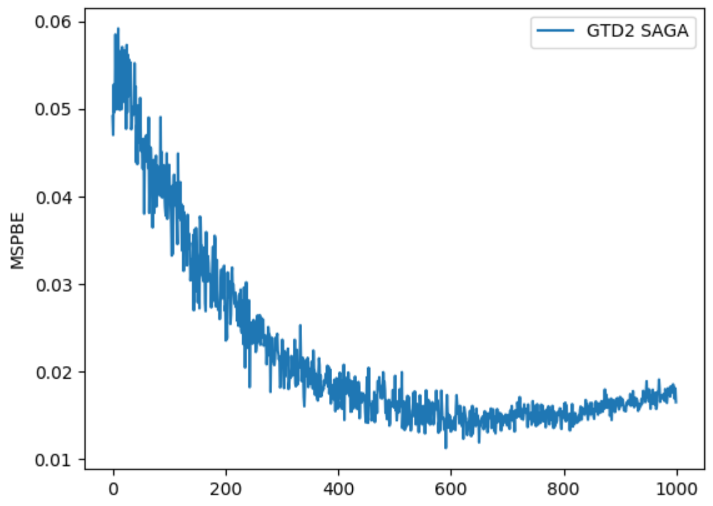

Fig. 7: MSPBE for GTD2 with SAGA

3.3 Evaluation

Our goal is to reduce the variance of TDC and GTD2 using SVRG and SAGA. Now, we will compare the variance and area under the curve (AUC) of MSPBE for 1000 epochs using the same experience for all these six methods. In table 2 you can see the average of these values for 35 run times for each of the methods.

	TDC	**GTD2**	**TDC+SVRG**	**GTD2+SVRG**	**TDC+SAGA**	**GTD2+SAGA**
AUC	20.57852566	21.24050861	18.87458555	22.24311276	19.03480063	21.95619784
Variance	1.05892386e-04	6.43749959e-05	7.33803157e-05	1.16460709e-04	6.79855361e-05	1.08155409e-04

Table 2: Mean value of the variance and AUC of MSPBE

As you can see from table 2, unfortunately, the variance reduction approaches didn't work very well for GTD2 but they are working fine for TDC, which means TDC+SVRG and TDC+SAGA has less variance and also less AUC than the pure TDC itself. The next step is to find out if this difference is meaningful or just happened by chance. Thus, we need statistical significance metrics to check this.

To do this, we used independent t-test as a statistical metric. The independent t-test, also called the two sample t-test, independent-samples t-test or student's t-test, is an inferential statistical test that determines whether there is a statistically significant difference between the means in two unrelated groups.

T-test has three assumptions. The first assumption is that the scale of measurement applied to the data collected follows a continuous or ordinal scale. The second is that the data is collected from a representative, randomly selected portion of the total population. The third assumption is the data, when plotted, results in a normal distribution, bell-shaped distribution curve. The first two assumptions are valid in our experiments and in figures 8 and 9 you can see that the third assumption is also satisfied. Hence, we can use t-test for AUC and variance.

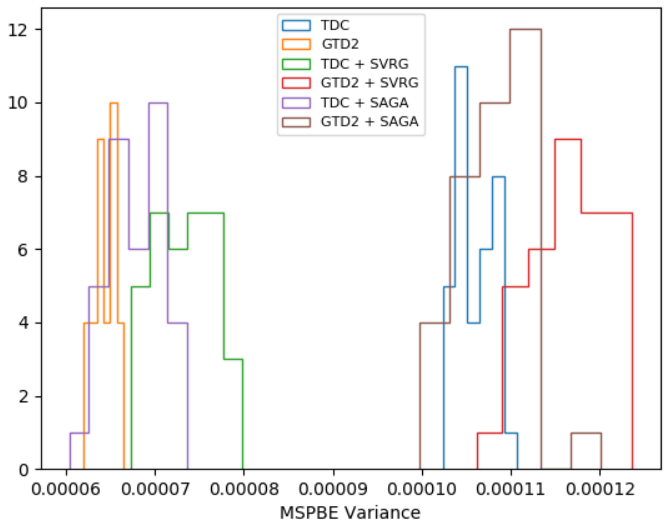

Fig. 8: Variance of MSPBE for the six methods

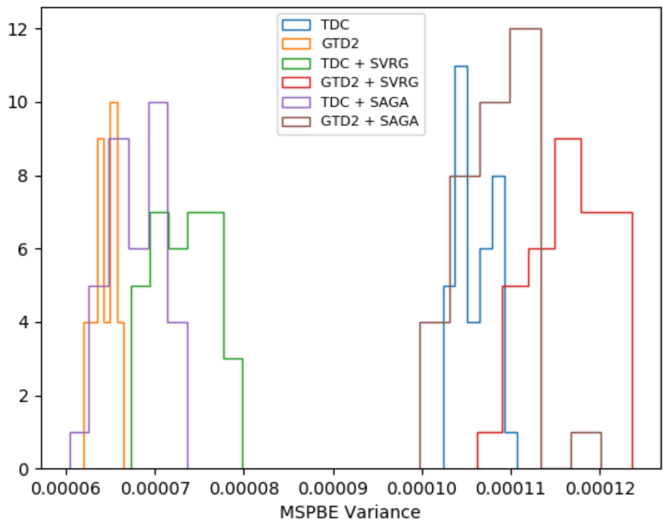

Fig. 9: AUC for the six methods

For TDC and TDC+SVRG, and TDC and TDC+SAGA, we calculated t-value for both variance and AUC based on formula 12 where $\overline{X_i}$, S_i, and N_i are mean, standard deviation, and the number of samples for method i respectively. Absolute value of the calculated t-values are shown in table 3. We then compared each of the t-values with the critical value computed from t table[1]. In order to find the critical value, we go row no degree_of_freedom equation 13 navigate to $t_{.975}$ column where we have alpha value of 0.05 for 2 tailed test. We chose α 0.05 to in order to get 95% confidence level. The critical value is 1.6449 for this data. We can see that t-value for each combination is greater than critical value. Therefore, we can confidently say that the difference between the results was not accidentally.

$$t - value = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} \tag{12}$$

$$degree\_of\_freedom = \frac{(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2})^2}{\frac{1}{N_1 - 1}(\frac{s_1^2}{N_1})^2 + \frac{1}{N_2 - 1}(\frac{s_2^2}{N_2})^2} \tag{13}$$

	TDC+SVRG	**TDC+SAGA**
TDC	47.8856	59.1077

Table 3: t-values of t-test for variance of MSBPE

We repeated the same procedure for the AUC of MSBPE. Table 4 shows t-values of t-test($\alpha = 0.05$)for TDC and TDC+SVRG, and TDC and TDC+SAGA. The critical value is 1.6450 for this data. Therefore, again we can conclude that the difference between the results was not accidentally.

	TDC+SVRG	**TDC+SAGA**
TDC	25.0303	21.4717

Table 4: t-values of t-test for AUC of MSBPE

As we mentioned before, SVRG and SAGA didn't improve GTD2. To make sure about that, we calculate the t-test for GTD2 and GTD2+SVRG, and GTD2 and GTD2+SAGA pairs. The results for AUC and variance are shown in table 5 and 6 respectively. As you can see from the tables the t-value is higher than the critical value so we can conclude that SVRG and SAGA didn't improve pure GTD2 at least with how we calculated the variance.

	GTD2+SVRG	**GTD2+SAGA**
GTD2	67.6532	59.4170

Table 5: t-values of t-test for variance of MSBPE

	GTD2+SVRG	**GTD2+SAGA**
GTD2	14.7449	11.3013

Table 6: t-values of t-test for AUC of MSBPE

4 Conclusion

To give a summary of our research, we applied variance reduction methods (SVRG and SAGA) to two policy evaluation methods, TDC and GTD2. In the result section, we showed that they didn't work as we expected for GTD2 but they worked well for TDC. Then we applied a statistical significance test to be sure about the correctness of the results that we got. Thus, we can conclude that TDC+SVRG and TDC+SAGA have less variance and less area under the error curve (MSPBE) than TDC and in general, it is a better algorithm comparing to pure TDC.

5 Future Works

There are at least three different approaches to calculate the variance that we know of, so the next step is to calculate these variances and check all these methods on them as well. After that, we can run them with harder features such as Neural Networks which are commonly used in real problems.

References

1. t table. https://www.sjsu.edu/faculty/gerstman/StatPrimer/t-table.pdf, accessed: 2019-12-5
2. Du, S.S., Chen, J., Li, L., Xiao, L., Zhou, D.: Stochastic variance reduction methods for policy evaluation. CoRR **abs/1702.07944** (2017), http://arxiv.org/abs/1702.07944
3. Maei, H.R.: Gradient temporal-difference learning algorithms (2011)
4. Sutton, R.S., Maei, H.R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., Wiewiora, E.: Fast gradient-descent methods for temporal-difference learning with linear function approximation. In: Proceedings of the 26th Annual International Conference on Machine Learning. pp. 993–1000. ACM (2009)
5. White, A., et al.: Developing a predictive approach to knowledge (2015)