Utility of accelerated temporal difference methods over gradient based optimizers

Dhawal Gupta [1]

Abstract

Temporal Difference (TD) learning approaches are currently optimized in a manner where we don't use the actual gradient of the objective (i.e. a semi gradient) and hence often end up using different optimization paradigms like TD(λ) and LSTD. In past gradient-based stochastic optimizers have shown success in optimizing objective functions and in this paper we wish to study the use of gradient-based optimizer like Adam on semi gradient problems like the Mean Squared Projected Bellman Error (MSPBE), i.e. policy evaluation. We compare the two approaches on two environments along with parameter sensitivity tests, to judge how they fare across one another.

1. Introduction

Reinforcement Learning problems can be broken down into two subproblems, where the first problems deals with policy evaluation and the second part deals with policy optimization, and the interplay between these two processes is termed as policy iteration. TD methods work on the principle of bootstrapping on the current estimates of the value function and the reward from the environment, allowing the agent to learn while interacting with the environment. This approach is termed as online, offering crucial benefits over the previous methods which relied on delaying the learning part to the end of the episode. The current methods for performing the TD updates lie at two opposite ends of a spectrum. On one side, we have TD (λ) methods (Sutton, 1988) which are designed to be computationally frugal. But these methods are not very data efficient and suffer from high parameter sensitivity (i.e. needs to have a carefully curated learning rate for optimal performance). On the other end, we have Least Squares TD (LSTD) methods (Boyan, 2002), which are highly data-efficient, but computationally expensive to calculate, i.e. quadratic in the feature size. The following work (Pan et al., 2016) introduces a class of methods called Accelerated GradientTD (ATD) that tries to strike a balance between the two extremes. ATD is a sub-quadratic method which builds an estimate of the second-order gradient (quasi second-order) improving the parameter sensitivity of the step size, along with being comparable in data-efficiency to LSTD.

Adam is a gradient-based stochastic optimizer which maintains a running second and first-order moments of the gradients of the parameters (Kingma & Ba, 2014), along with bias correction due to prior's. This method has shown significant improvement in convergence performance of learning algorithms when compared to say conventional Stochastic Gradient Descent. But the performance of gradient-based optimizers on semi gradient problems remain untested and this is what we try to study in this paper.

We need to find out if developing methods, especially for Reinforcement Learning, has any utility over just using conventional gradient optimizer. There is a low amount of consensus in the community regarding the usage of different standards, and this paper tries to investigate one of those issues.

1.1. Structure of the paper

The paper is broken down into seven sections, where Section 1 deals with introducing the reader to the problem and the topic. Section 2 deals with describing the problem in detail and explain the semi gradients in TD methods. Section 3 deals with implementation details of the project. In section4 we enumerate the experimental details. Section 5 I present the results for our experiments and give my analysis on that. Section 6 gives conclusion and a direction for future work to do.

2. Semi Gradient TD Methods

TD methods have been extensively studied for tabular functions and converge to the optimal value function estimates for that condition, but in systems which constitute of large

[*]Equal contribution [1]Department of Computing Science, University of Alberta, Alberta, Canada. Correspondence to: Dhawal Gupta <dhawal@ualberta.ca>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

state spaces, it becomes restrictive, in terms of memory and computation to use tabular methods and hence we try to use function approximation, on a set of features which represent different states, often written as $\mathbf{X}$, where $\mathbf{X} \in \Re^{|\mathbf{S}| \times \mathbf{d}}$, where $|S|$ is the size of state space and d is the size of the feature set. In table methods, all the value estimations are decoupled from one another, which allows us to approximate the exact value of the states. Still, in the case of function approximation, there is a spatial property updating the value of one state will change the value of many other states as well. This property is desirable in the terms that in a good representation of states, state closer to each other will have similar values, but having a bad representation can also degrade our learning. This paper will consider the simple case of linear function approximation where we will try to estimate the weight $\mathbf{w} \in \Re^{\mathbf{d}}$ vector such that $\mathbf{Xw}$ is as close to the $\mathbf{V}$, where $\mathbf{V} \in \Re^{|\mathbf{S}|}$ is the real value function.

There are two approaches to solving policy evaluation using TD methods differing in their space and time complexity. One is a stochastic approach of TD(λ) techniques and other is the direct approximation of a linear system using the LSTD method. ATD offers us the best of both worlds being data efficient as well quasi second order allows it to have a trade-off between both. There are other methods also which are quasi second-order, i.e. iLSTD (**?**), tLSTD (**?**), but for the sake of completeness, ATD seems to perform reasonably better than the other algorithms.

In our study, we will be comparing methods for linear function approximation where $\mathbf{w}$ will represent the weight of our model and $\mathbf{x_t}$ represent our representation of the state at time t. $\mathbf{x(s)} \doteq (\mathbf{x_1(s)}, \mathbf{x_2(s)}, ..., \mathbf{x_d(s)})^{\mathbf{T}}$, with the same size as the weight vector $\mathbf{w}$.

$$(s, \mathbf{w}) \doteq \mathbf{w^T x(s)} \doteq \sum_{\mathbf{i=1}}^{\mathbf{d}} \mathbf{w_i x_i(s)} \tag{1}$$

For simplicity we will consider TD(0), so when taking in consideration the TD Error where we want to say minimize teh squared TD erro.

$$\delta_t^2 = (r_t + \gamma \mathbf{w_t^T x_{t+1}} - \mathbf{w_t^T x_t})^{\mathbf{2}} \tag{2}$$

To minimize this we take the gradient of the above expression and do a gradient descent on the weights (a stochastic approximation approach). So in this our target is also dependent on the current estimates of the weights, but while taking the gradient we ignore the weights from the next state and take the gradient only of the current state estimate, hence making it not the true gradient and being termed as a semi gradient.

$$\nabla_{\mathbf{w_t}} = -2(r_t + \gamma \mathbf{w_t^T x_{t+1}} - \mathbf{w_t^T x_t})\mathbf{x_t} \tag{3}$$

This makes our gradient descent update as follows, and we absorb the factor of 2 in α:

$$\mathbf{w}_{t+1} = \mathbf{w_t} + \mathbf{alpha}(\mathbf{r_t} + \gamma \mathbf{w_t^T x_{t+1}} - \mathbf{w_t^T x_t})\mathbf{x_t} (4)$$

The above update ii

3. Implementation Details

I have done all the implementation from scratch, including the environments TD algorithms and Adam optimizer (borrowed from the assignment). I have made a repository, i.e. `TDSuite`.

3.1. Accelerated TD Method

I followed the ATD paper (Pan et al., 2016) to implement the optimization process, with subtle differences. In the paper, they use incremental SVD to approximate a low-rank Hessian, which essentially makes their methods subquadratic. The paper treats the rank of the Hessian as a hyper parameter. For the sake of clarity, in my implementation, I have made use of the `inv` function that is available in the `numpy.linalg` package. This makes our approach quadratic, but the purpose of our study here is to judge the quality of solutions produced by gradient and TD approaches; hence we will ignore that for now. Effectively there are no hyperparameters for our algorithm.

3.2. Environments

I will investigate the differences between two sets of environments which are Markov Reward Processes (MRP), i.e. Random Walk and Boyan Chain. Both of these implementations are general enough to be flexible with the number of states that we need to estimate. A linear function approximator can determine both of the environment value functions, the exact details of the environment are provided in Section 4.

3.3. TD Suite

There is a collection of other methods like TD(λ) and Monte Carlo (MC) available in the codebase which can be interfaced with the environment. Currently it includes the following implementations:

- TD(λ)
- LSTD
- ATD
- Monte Carlo
- Adam
- RMSProp
- Boyan Chain
- Random Walk

4. Experimental Setup

I have made use of two environment test beds i.e. Random Walk (Sutton, 1988) and the Boyan Chain (Boyan, 2002).

4.1. Random Walk

Random walk is a Markov Reward Process (MRP), where all episodes start from the center state C, and then proceed either left or right by on one state on each step, with equal probability. Episodes terminated either on the extreme ends (left or right). When an episodes terminates on the right a reward of +1 occurs , all others are zero. The true values of boyan chain with with n states (normally odd is as follows $1/n+1, 2/n+1, \ldots, n/n+1$. This value corresponds to the probability of reaching the right end from that state. Fig. 4.1 shows the environment in consideration.

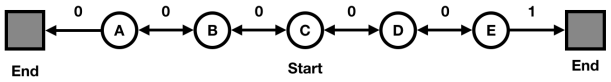

Figure 1. Random Walk adapted from (Sutton & Barto, 2018)

4.2. Boyan Chain

Represented in the (Boyan, 2002), it is also an MRP , where according to the paper it consisted of 13 states , each of which is represented by four state features. The Figure. 4.2 aptly summarizes the environment below. The optimal

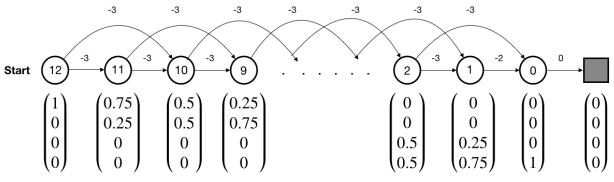

Figure 2. A 13-state Markov chain. In states 2–12, each outgoing arc is taken with probability 0.5. For value function approximation, each state is represented by four features, as follows: the representations for states 12, 8, 4, and 0 are, respectively, [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], and [0, 0, 0, 1]; and the representations for the other states are obtained by linearly interpolating between these. Borrowed from (Boyan, 2002)

weights for the above system are as follows(-24, -16, -8, 0).

4.3. Experiments

I have used Adam that I had implemented in Assignment 2, and I am comparing ADAM using different metrics. I am running each of learners for 20,000 episodes, where episodes length are stochastic. Each metric for a hyperparameter is averaged over 30 runs. The hyperparmeters that I have optimized over as follows :

- Learning Rate : $\{2^0, 2^{-1}, 2^{-2}, \ldots, 2^{-17}\}$
- Beta 1 : $\{0.8, 0.85, 0.9, 0.95, 0.99, 0.999\}$
- Beta 2 : $\{0.8, 0.85, 0.9, 0.95, 0.99, 0.999\}$

This gives a total number of 648 hyperparameters. In all these experiments we consider the mean squared error between the optimal weights and the estimated weights, which we will write as RVE, which in short represents a mean squared value error (As our original environments value functions can be approximated by a linear function, both errors are directly proportional) .

$$RVE(\mathbf{w}) = \sqrt{(\mathbf{w}^* - \mathbf{w})^2} \tag{5}$$

Metrics for algorithm performance : This paper studies the performance of Adam with respect to two performance metrics listed as follows :

- **Final Performance** : In this regime we try to optimise the final performance which entails the average value errors of the last 200 episodes in our different runs of optimization.
- **Area Under Curve** : In this regime we optimize to minimize the area under the RVE curve. The area under curve is normalized by the number of episodes in the curve.

We also conduct experiments to study the sensitivity of different parameters to see how flexible Adam is in terms of bad hyper parameter choices. This analysis has been highly inspired by the following work (Ghiassian et al., 2018).

5. Results

I will be presenting the comparison of ATD and ADAM, where ADAM's hyper parameters are optimized for the above mentioned metric. Remember that ATD doesn't have any hyper-parameters to choose from at this point.

5.1. Boyan Chain

Best RVE reached for ATD is as follows : 5.7702×10^{-3} (averaged over 30 runs) Best RVE reached for ATD : 5.5983×10^{-4} (unaveraged)

5.1.1. Final Performance

Best set of hyperparameters

- Learning Rate : 2^{-11}
- Beta 1 : 0.95
- Beta 2 : 0.999

Best RVE reached ADAM : 6.5589×10^{-2} (averaged over 30 runs)
Best RVE reached ADAM : 3.4602×10^{-3} (unaveraged)
Figure 3 compares the learning performance for ADAM vs ATD optimized for final performance. As we can see that to improve the final performance, we tradeoff fast convergence to choose a low learning rate allowing us to converge to a small value. But still ATD offers us a substantially improved performance, also we can notice that the performance for ADAM plateaus at a level above the error of ATD, whereas the mean error of ATD still increases, maybe allowing a smaller step size would have allowed for convergence of the two solutions.

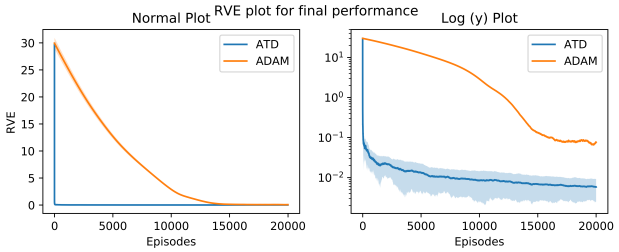

Figure 3. This graph shows the learning performance optimized for final performance in the case of ADAM and ATD. Figure on the left has a linear y scale, whereas figure on the right has a log scale Environment : **Boyan**.

Figure 4 Shows the sensitivity of different hyperparameters of adam with respect to the RVE optimized for final performance. There is clearly a powered increase in the RVE based on the learning rate but over all this seems pretty robust to the hyperparamters, with slightly better performance for the best set. So the most sensitive parameter as we suspected will be the learning rate in this case.

5.1.2. Area Under Curve (AUC)

Best set of hyperparameters

- Learning Rate : 2^{-6}
- Beta 1 : 0.85
- Beta 2 : 0.999

Best RVE reached ADAM : 3.4516×10^{-1} (averaged over 30 runs)
Best RVE reached ADAM : 1.1034×10^{-2} (unaveraged)

Figure 5 compares the learning performance for ADAM vs ATD optimized for area under the error curve (auc) As we can see that to improve the auc, we tradeoff lower convergence to choose a fast convergence. The error achieved is significantly higher than the final performance (Figure 3) curve as well as the ATD learning curve.

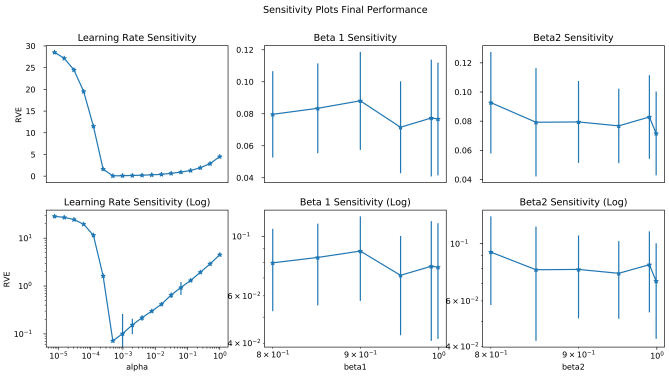

Figure 4. This graph shows the learning performance optimized for final performance in the case of ADAM and ATD. Figure on the left has a linear y scale, whereas figure on the right has a log scale. Environment : **Boyan**

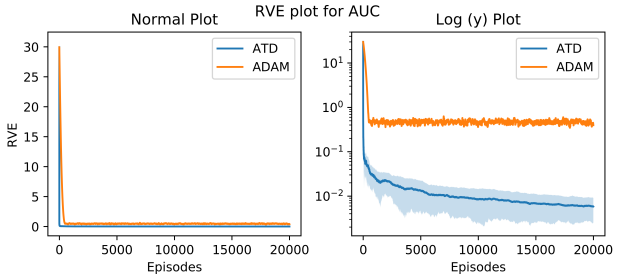

Figure 5. This graph shows the learning performance optimized for area under curve in the case of ADAM and ATD. Figure on the left has a linear y scale, whereas figure on the right has a log scale. Environment : **Boyan**

Figure 6 Shows the sensitivity of different hyperparameters of ADAM with respect to the RVE optimized for auc. Again in this case the learning rate seems to be quite sensitive this time favouring larger values. Surprisingly, β_1 seems be really bad near 1 otherwise its remain pretty much constant, vis-a-vis for β_2, where the performance varies across different values of β_2.

As we can see from Figure 7, that most of the parameter configurations perfom sub optimally, this can partly be because of slow learning rates not getting the time to converge to a goof weight vector.

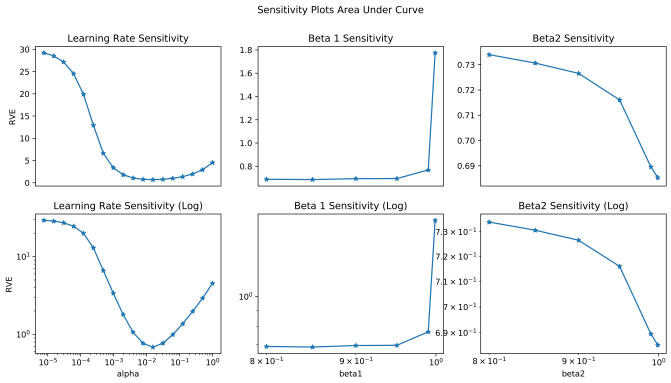

Figure 6. This graph shows the learning performance optimized for final performance in the case of ADAM and ATD. Figure on the left has a linear y scale, whereas figure on the right has a log scale. Environment : **Boyan**

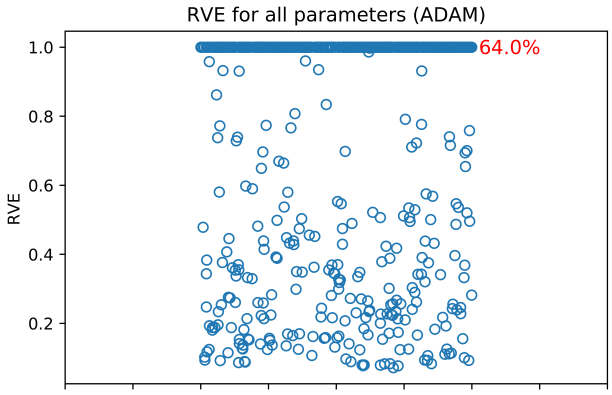

Figure 7. RVE for all set of parameter configurations for **Boyan**

5.2. Random Walk

Best RVE reached for ATD is as follows : 8.1818×10^{-3} (averaged over 30 runs)
Best RVE reached for ATD : 2.8324×10^{-17} (unaveraged)

5.2.1. Final Performance

Best set of hyperparameters

- Learning Rate : 2^{-14}
- Beta 1 : 0.999
- Beta 2 : 0.999

Best RVE reached ADAM : 1.6419×10^{-2} (averaged over 30 runs)
Best RVE reached ADAM : 3.6240×10^{-3} (unaveraged)

Figure 8 compares the learning performance for ADAM vs ATD optimized for final performance. In this environement ADAM seem to converge pretty fast and seems to asymptotically reach the performance of ATD. Although longer runs would be required to actually achieve this. ATD seems to plateau around 15K episodes. It seems that both cases are possible where the ADAM plateaus above ATD or reaches the same error as ATD.

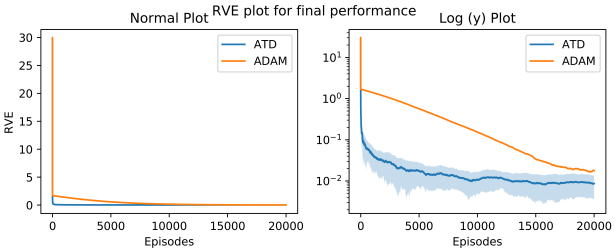

Figure 8. This graph shows the learning performance optimized for final performance in the case of ADAM and ATD. Figure on the left has a linear y scale, whereas figure on the right has a log scale Environment : **Random Walk**.

Figure 9 Shows the sensitivity of different hyperparameters of ADAMs with respect to the RVE optimized for final performance. ADAM as usual is sensitive to the learning rate but as opposed to Boyan, it is able to achieve lower error even for lower learning rates. Also it seems for this environment that β_2 is quite sensitive. This analysis tells us that the performance of ADAM is actually very dependent on the environment. But similar to Boyan the β_1 is not that sensitive.

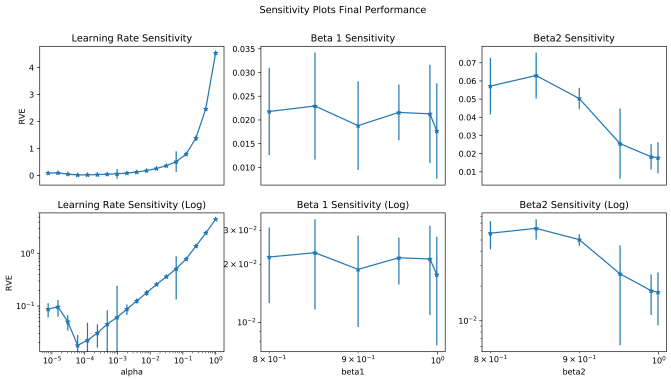

Figure 9. This graph shows the learning performance optimized for final performance in the case of ADAM and ATD. Figure on the left has a linear y scale, whereas figure on the right has a log scale. Environment : **Random Walk**

5.2.2. AREA UNDER CURVE (AUC)

Best set of hyperparameters

- Learning Rate : 2^{-10}
- Beta 1 : 0.999
- Beta 2 : 0.999

Best RVE reached ADAM : 5.3774×10^{-2} (averaged over 30 runs)
Best RVE reached ADAM : 8.6473×10^{-3} (unaveraged)

Figure 10 compares the learning performance for ADAM vs ATD optimized for area under the error curve (auc). As we can see that to improve the auc, we tradeoff lower convergence to choose a fast convergence. The error achieved is significantly higher than the final performance (Figure 8) curve as well as the ATD learning curve. The learning curve pleateaus a higher error level.

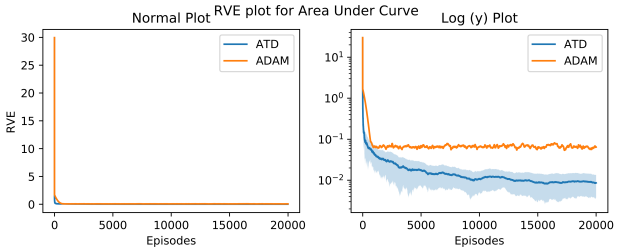

Figure 10. This graph shows the learning performance optimized for area under curve in the case of ADAM and ATD. Figure on the left has a linear y scale, whereas figure on the right has a log scale. Environment : **Random Walk**

Figure 11 Shows the sensitivity of different hyperparameters of ADAM with respect to the RVE optimized for AUC. Again in this case the learning rate seems to be quite sensitive this time favouring smaller values. Also both the parameters i.e. β_1 and β_2 are really sensitive for performance, more than the Boyan Chain.

As we can see from Figure 12, more number of parameter configurations perform better when compared to the Boyan Chain. Here 39% of the parameters got a RVE of more than 1 on their final weights.

6. Conclusion & Future Work

From the above analysis we can easily see that ADAM is really sensitive to its hyper-parameters,specially the learning rate. Also the sensitivity of hyper-parameters depends on different environments, which is something that we want to avoid in our algorithms. One interesting thing would be to see if ADAM converges to a biased solution when compared to say a solution by the ATD method. There a completely

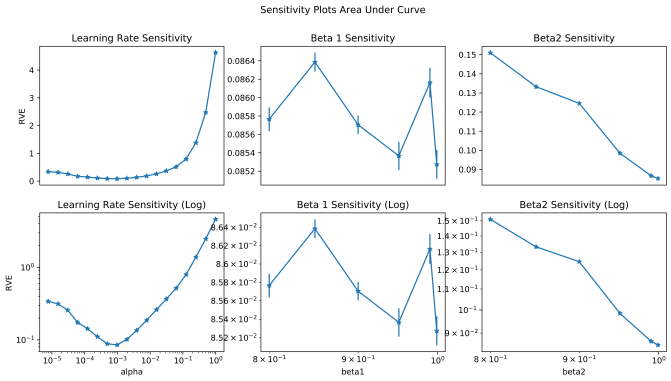

Figure 11. This graph shows the learning performance optimized for final performance in the case of ADAM and ATD. Figure on the left has a linear y scale, whereas figure on the right has a log scale. Environment : **Random Walk**

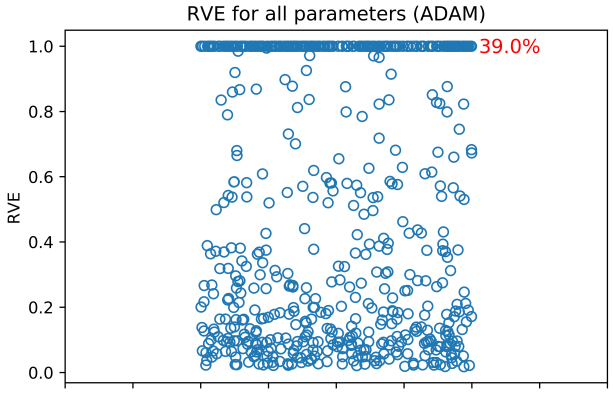

Figure 12. RVE for all set of parameter configurations for **Random Walk**

different class of methods know as Gradient

Future Work : One possible line of study could have been to test GTD and GTD2 methods with ADAM, as they more closely resemble true gradients and should perform better. Also it would be interesting to see if TD on ADAM would converge to a biased solution when compared to ATD (which is unlikely). Another possible direction is to prove convergence of ATD for the stochastic sampling case. (**?**)

Acknowledgements Huge thanks to Dr. Martha White and Yangchen Pan for their continuous guidance on navigating the problems and possible ways of studying it. I would also like to thank TA's of ML course (CMPUT 566) specially Matthew Schlegel for helping me out with running experiments on Compute Canada.

References

Boyan, J. A. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, Nov 2002. ISSN 1573-0565. doi: 10.1023/A:1017936530646. URL https://doi.org/10.1023/A:1017936530646.

Ghiassian, S., Patterson, A., White, M., Sutton, R. S., and White, A. Online off-policy prediction. *CoRR*, abs/1811.02597, 2018. URL http://arxiv.org/abs/1811.02597.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2014.

Pan, Y., White, A. M., and White, M. Accelerated gradient temporal difference learning. *CoRR*, abs/1611.09328, 2016. URL http://arxiv.org/abs/1611.09328.

Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. A Bradford Book, USA, 2018. ISBN 0262039249, 9780262039246.