Applicability of Momentum in the Methods of Temporal Learning

Dhawal Gupta*

March 6, 2020

Abstract

Reinforcement Learning (RL) and Supervised Learning (SL) are two forefronts of machine learning. RL relies heavily on the methods of temporal difference learning. SL community offers various adaptive gradients methods to speed and improve the convergence properties of their algorithms. Hence a natural question arises that, whether we can utilize the techniques from SL community for RL methods. This paper provides initial steps in those directions by comparing the applicability of Momentum in RL updates, by comparing Momentum on TD(0) to TD(λ) on a set of environments and showing initial results that Momentum instead of improving performance might hinder the learning in a given set of environments.

1 Introduction

Temporal Difference (TD) (Chapter 6, Sutton, 2018) learning methods have been a standard go-to method in the Reinforcement Learning community be it for prediction or control. They use bootstrapping, which allows the agent to start learning immediately instead of delaying the learning to the very end of the episode, unlike the Monte Carlo Methods (Chapter 5, Sutton, 2018). Though this approach is a boon, also suffers from various problems like biasness of the results due to bootstrapping, which often hinders the learning. TD(λ) was introduced as a bridge between the Monte Carlo methods and the TD(0) algorithm, offering a bias-variance tradeoff with the use of λ parameter. Specifically, the backward view of the algorithm helps us reduce the credit assignment problem (with linear function approximation), by projecting back the TD Errors (δ) to the previous states as well. Effectively TD learning tries to solve the Reinforcement Learning problem by adapting it to the setting of the Supervised Learning (SL) (Sutton, 1988), but differs in various aspects to the specification of problem for e.g. Non Stationarity of the objective, and non i.i.d. sampling of samples. Nonetheless, SL community has developed several accelerated and stable variants of Stochastic Gradient Descent (SGD) that can increase the learning potential manifold, some common ones for e.g. are ADAM (Kingma, 2017), RMSProp (Tieleman, 2012) etc. Momentum (Rumelhart, 1986) is one of the widely used modification on top of SGD which also forms a sub-part of algorithms like ADAM. The most touted feature of Momentum is that it helps with reducing the variance in gradient updates, by maintaining a short term memory (a feature of TDλ) also.

This paper tries to do an initial study on the applicability of accelerated gradients methods from SL to the case of Reinforcement Learning as opposed to using and developing separate acceleration methods like Accelerate Gradient TD (ATD) (Pan, 2017). Mainly I will be comparing the TD(λ) method to Momentum applied on the TD(0) update, to see if we can achieve similar effects with Momentum that TD(λ) effectively provides us on a given set of environments. This paper aims at providing an initial study of interaction fo hyperparameters of both the methods.

2 Hypothesis

Hypothesis

> **Momentum on TD(0) can perform worse than TD(λ), and actually hinder performance on some environments possibly dues to biasing of updates.**

Momentum with TD(0) (refereed to as Momentum(0)) and TD(λ), differ in single hyperparameters and try to maintain a short term memory of the updates to modify and help performance, TD(λ) maintains a trace vector of the previous features (i.e. $\mathbf{x}_t$). In contrast, Momentum maintains a trace of the previous updates, i.e. ($\delta_t \mathbf{x}_t$). I believe that because in Momentum we are tracing the TD Error δ along with the features, this will make our update more biased because of the older more biased TD Errors. This is particularly a problem with Non Stationary environments. β the tracing parameter in Momentum(0) will allow us to tune this bias.

*dhawal@ualberta.ca

3 The RL Setting and Background

We will be using an Episodic Markov Reward Process (MRP) defined as $\langle \mathcal{S}, \mathcal{S}^+ P, r, \gamma \rangle$. An MRP differs from an MDP in the sense that it does need actions, and the states have an automatic transition between themselves, more of way to say an MDP with a fixed policy. Here $\mathcal{S}$ defines the state space of the environment, $\mathcal{S}^+$ represents the states of environment including the terminal states, P is the transition probability between different states defined as $P : \mathcal{S} \times \mathcal{S}^+ \to [0, 1]$, which can also be written as a matrix i.e. $\mathbf{P} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$ and r is the reward function defined as $r : \mathcal{S} \to \mathbb{R}$, which might not be deterministic and can be defined as $r(s) = \mathbb{E}[R_{t+1} | S_t = s]$, where R_{t+1} and S_t are random variables representing the reward and states at time t and $t + 1$. The γ is the discount factor that is used to discount the future value of states. The return at time t is denoted by G_t and defined as follows :

$$G_t \doteq \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

3.1 Problem of Prediction

In RL we have broadly two kinds of problems known as Prediction and Control. We will be dealing with only prediction wherein we have to estimate the expected return, that we get from each state, which is succinctly called as a **value function** :

$$v(s) \doteq \mathbb{E}[G_t | S_t = s]$$

For our study we will be using a linear function approximator for our value function which is defined as follows :

$$\hat{v}(S_t, \mathbf{w}_t) \doteq \mathbf{w}_t^T \mathbf{x}_t$$

Where $\mathbf{x}_t \in \mathbb{R}^d$ is the feature representation of state at time t, and $\mathbf{w}_t$ represents the our parameter estimates at time t. All the vectors that we will be considering will be column vectors. One more thing to note is that all the states that belong to the set of terminal states have a value of 0 i.e.

$$v(s) \doteq 0, \forall s \in \mathcal{S}^+ - \mathcal{S}$$

3.2 Sampling

Sampling refers to the order in which the agent sees the sample that the agent has to learn from. In our case the sample constitutes of $\langle s, r, s' \rangle$. Where s and s' refers to the state and next state and r refers to the reward received between the transition. There are mainly two types of sampling methods that we will consider in this project, which have been explained below in brief.

3.2.1 Markov Sampling

Markov Sampling is the most common form of sampling in Reinforcement Learning, where the samples are drawn as the agent works in the environment, i.e. along with the trajectory of the environment. Hence our transition sample can also be written as $\langle S_t, R_{t+1}, S_{t+1} \rangle$, where t denote the current time that we are in the trajectory. Hence the samples that our agents will see are highly correlated from each other. This kind of sampling is of the essence in the TD(λ) algorithm, where the trace vector is highly dependent on the trajectory of the episodes that the agent sees. These kind of samplings are natural when the agent learns while following episodes.

3.2.2 Independent and Identically Distributed Sampling

Independent and Identically Distributed samples, also known as I.I.D. samples. This applies for sample transitions which are uncorrelated with each other. This kind of sampling is one of the underlying assumptions in the Supervised Learning community for algorithms like SGD, Momentum, ADAM etc.

For the case of RL, this sampling can be simulated in multiple ways, 2 of which I am mentioning very briefly.

- **Exact Sampling** : For small environments which are exactly solvable we can easily calculate the stationary distribution for a given policy denoted by $\mu(s), s \in \mathcal{S}$ i.e. $\boldsymbol{\mu} \in \mathbb{R}^{|S|}$. So knowing the transition dynamics and exact reward function we can sample $s \sim \mu(S)$, in and the reward and next states as $s' \sim P(s)$, $r \sim R(s, s')$, to get the transition $\langle s, r, s' \rangle$
- **Repay Buffer**: By maintaining a replay buffer of the samples seen, we can simulate the effect of doing I.I.D. sampling by uniformly sampling from the stored transitions. For big enough replay buffers we can say that the samples are sampled in an I.I.D. fashion.

4 Objectives and Solution Methods

Here I will be using the term optimizers a little loosely to refer to different optimization strategies and algorithms, and also their practical understanding. Optimizers are algorithmic steps to minimize an objective function, where the objective function directly or indirectly relates to the goal that we want our agent to achieve. E.g. in the case of prediction We want our agent to predict the correct value of states under a policy, so an example objective function can the error between the predicted value and the actual value. We will be representing our value function estimate for state at time t with weight vector $\mathbf{w}_t$ as

4.1 Objective : MSPBE

MSPBE stands for Mean Squared Projected Bellman Error (Maei, 2011), which is used as an objective for Gradient TD algorithms, which are actual gradient based methods. The optimal point of this objective coincides with the TD fixed point i.e. $\mathbb{E}[\delta_t \mathbf{x}_t] = 0$
MSPBE is given by :

$$\text{MSPBE}(\mathbf{w}) = (\mathbf{b} - \mathbf{A}\mathbf{w})^T \mathbf{C}^{-1} (\mathbf{b} - \mathbf{A}\mathbf{w}) \tag{1}$$

Where $\mathbf{A}, \mathbf{b}, \mathbf{C}$ are defined as :

$$\begin{aligned} \mathbf{A} &\doteq \mathbf{X}^T \mathbf{D} (\mathbf{I} - \gamma \mathbf{P}) \mathbf{X} \\ \mathbf{b} &\doteq \mathbf{X}^T \mathbf{D} \mathbf{R} \\ \mathbf{C} &\doteq \mathbf{X}^T \mathbf{D} \mathbf{X} \end{aligned}$$

Here , $\mathbf{X} \in \mathbb{R}^{|\mathcal{S}| \times d}$, is the feature matrix for all states, $\mathbf{D} \in \mathbb{R}^{|\mathcal{S}|}$, $\mathbf{D} \doteq \text{diag}(\boldsymbol{\mu})$, diagonal matrix containing the stationary distribution probabilities for states on diagonal. $\mathbf{R} \in \mathbb{R}^{|\mathcal{S}| \times 1}$, where $\mathbf{R}(s)$ is the expected one step reward from that state.

4.2 TD(λ)

TD(λ) (Chapter 12, Sutton, 2018), is a policy evaluation / prediction algorithm which gives a balance between the TD(0) a full bootstrap algorithm and Monte Carlo Methods. Monte Carlo methods are unbiased estimator of the value function whereas they suffer from the problem of delaying all the computation to the end of episode. In the case of TD(0), they are very nice in terms of distributing the computation evenly between the episode, whereas they can be highly biased because of the bootstrapping term (though they are guaranteed to converge under the on policy linear function approximation case). TD(λ) provides a congenial backward view which allows the computation to be distributed across the whole episode and also allows to trade-off the biasness based on the trace parameter.

The update equation for TD (λ) is given as

$$\begin{aligned} \mathbf{z}_t &\doteq \gamma \lambda \mathbf{z}_{t-1} + \nabla \hat{v}(\mathcal{S}_t, \mathbf{w}_t) \\ \delta_t &\doteq R_{t+1} + \gamma \hat{v}(\mathcal{S}_{t+1}, \mathbf{w}_t) - \hat{v}(\mathcal{S}_t, \mathbf{w}_t) \\ \mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t \end{aligned}$$

Where the $\mathbf{z}_{-1} = 0$, and $0 \leq t \leq T$. In the case of linear function approximation we get $\nabla \hat{v}(S_t, \mathbf{w}_t) = \mathbf{x}_t$, where $\mathbf{x}_t$ is the feature representation of state at time t, α is the learning rate of the agent. λ takes values between $[0, 1]$.

4.3 Momentum

Momentum is a first order gradient based method which maintain an expectation of the gradient by maintain a exponential moving average of the past observed gradients. Momentum differs from TD(λ), where TD(λ) maintains a trace of $\nabla \hat{v}$, but momentum momentum maintains a moving average of the gradient of the objective function. We will be applying momentum on the semi gradient of TD(0) algorithms, equations for which are given below[1].

$$\begin{aligned} \mathbf{m}_t &\doteq \beta \mathbf{m}_{t-1} + (1 - \beta) \mathbf{g}_t \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha \mathbf{m}_t \end{aligned}$$

[1]There are multiple variants of Momentum, but I have chosen the following variant because that is what has been proposed in several existing works like ADAM (Kingma, 2017), with convergence proofs. All the variants have effectively the same update, just differs by a scalar factor.

Where $\mathbf{m}_{-1} = 0$ For the TD learning the gradient for time t is

$$\mathbf{g}_t = \delta_t \mathbf{z}_t$$
$$\mathbf{g}_t = \delta_t \mathbf{x}_t \text{ for TD(0)}$$

For completeness we will be referring to the Momentum on top of TD(0) as Momentum(0), because we will not be maintaining traces for the case of Momentum. Similarly we can extend this definition to Momentum applied on TD(λ) to be referred to as **Momentum(λ)**, which will comprise of two parameters i.e. β and λ.

5 Experimental Setup

The experiments I have performed are centred around the problem of prediction in Reinforcement Learning. Hence I will be comparing the accuracy of prediction of the two methods based on a metric on a given environment.

5.1 Environment

I have conducted my experiments on 2 different MRP's namely as Random Walk and Boyans Chain.

5.1.1 Random Walk

Random Walk is used as described in the (Example 6.2, Sutton, 2018), with the 9 state variant and equiprobable policy. The environment consists of 9 states and 2 terminal states. The environment is arranged in the form of an elongated chain, as shown in 1. Each run in the environment is episodic, and the agent always starts in the middle state (i.e. 4). The agent has an equal probability of either moving left or moving right by one step at a given time. The episode ends either when the agent takes a left action from the left-most state (0), in which case it gets a reward of 0, or when it takes a right action from the rightmost state (i.e. 8), in which case it gets a reward of +1. Because this is an episodic environment, we will be using a $\gamma = 1$ value for all our experiments. Figure 1 also contains the feature representations that were used for the set of experiments.

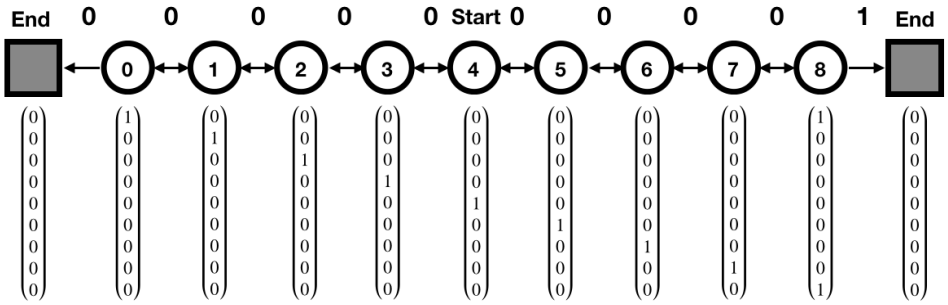

Figure 1: The 9 state random walk

5.1.2 Boyan's Chain

Boyan's Chain (Boyan, 1999) is a 13 state MRP used initially for prediction. See Figure 2 for the chain configuration, feature representation and reward structure. There are 13 states and 1 terminal state; the agent starts in the left-most states and transition towards the right by 1 or 2 states with equal probability except for the 2 rightmost states. Here also I had used a $\gamma = 1$

5.2 Error Metric

I had used the Mean Square Projected Bellman Error (MSPBE) as a way to measure error over different runs of the agent and compare respective performances. The reason that I have used MSPBE is that in the case of given environments the MSPBE solution and the Mean Squared Value Error (MSVE) solution coincide as the problem is entirely solvable with given linear function approximation and representation.

Equations for MSPBE are given Eq. 1, here I will discuss how I calculated the same, taking the example of the Random Walk environment. So according the equation mentioned above for MSPBE, we need the following entities to actually calculate it : $\mathbf{X}, \mathbf{D}, \mathbf{P}, \mathbf{R}$ and $\mathbf{w}_t$

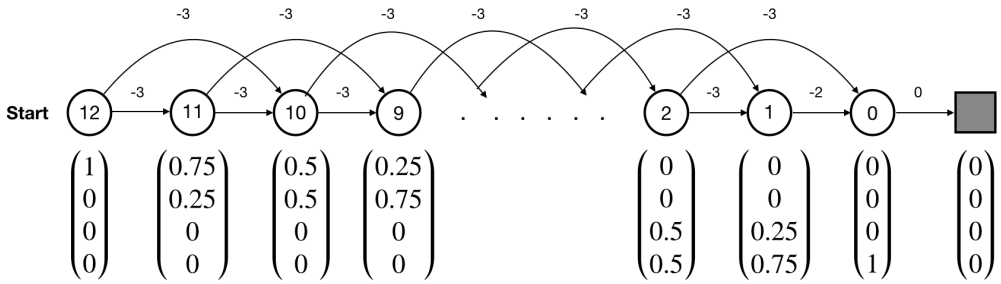

Figure 2: Boyan's Chain

- **X** : This the feature matrix which is quite easy to make, we have all the features for all the states and need to list them in a single matrix, we will put the terminal state feature as the last entry (and treat both the terminal states as same). hence for our case, we would have $\mathbf{X} \in \mathbb{R}^{10\times 9}$
- **P** : Based on a given policy it is straightforward to make the transition matrix, we need to take care of the transitions from a normal state to terminal states, that is why again we expand the **P** by one size (and remember there will be no out transitions from terminal states). Hence the $\mathbf{P} \in \mathbb{R}^{10\times 10}$.
- **D** : This is just a diagonal matrix of the stationary distribution. Now to calculate the stationary distribution, there are several methods, but I am going to list the easiest one. So first we need to change the **P** to be of size 9, and call this as $\hat{\mathbf{P}}$. Then make the transition from all the states penultimate to the terminal states to go to the starting state, i.e. 4, e.g. $\hat{\mathbf{P}}(8,4) = 0.5$ etc. This way, we connect the whole chain into a single continuous domain. Then exponentiate the whole matrix to a big power say 1000. Now take the mean of each column which corresponds to the stationary distribution of that state number. This will give us the vector $\boldsymbol{\mu}$ of size 9, extend it with one more value of 0 for the terminal state, and make it into a diagonal matrix.

$$\begin{aligned}\hat{\mathbf{P}}' &= \hat{\mathbf{P}}^{1000}\\ \mu(s_i) &= \text{mean}(\hat{\mathbf{P}}'_{:,i}), \forall s_i \in \mathcal{S}\\ \mu(terminal) &= 0\\ \mathbf{D} &= \text{diag}(\boldsymbol{\mu})\end{aligned}$$

- **R** : This is pretty straightforward, we have the full dynamics of the system, multiply the corresponding one-step rewards dues to each action with the action's probability, for each of the states (except for terminal state, value for that will be 0).

Having all these matrices, for a given $\mathbf{w}_t$ we can easily calculate the MSPBE using Equation 1.

5.3 Agents

Here I am going to discuss the different implementational details of the agents and experimental details that I followed while running them.

- **TD(λ)** : Implementation of TD(λ) is pretty straightforward and the agent is meant to work in the Markovian sampling case. Remember to reset the trace vector to zero after the termination of an episode. For completeness sake I also ran experiments with I.I.D. sampling (**Exact**) for TD(λ). The two hyperparameters in the agent are α and λ respectively.
- **Momentum(0)** : Momentum tries to maintain a running expectation of the semi gradient TD i.e . $\delta_t \mathbf{x}_t$. Momentum has been proposed for the I.I.D. sampling case, but I have conducted experiments in both versions of samplings. For the case of I.I.D., I have used the **Exact** technique as we already have all the required approximations for it. One thing to keep in mind is that I did not reset the expected gradient vector, i.e. $\mathbf{m}_t$ in the case of Momentum even if we receive transitions to a terminal state. The hyperparameters for Momentum(0) are α and β respectively.

5.4 Experimental Runs

5.4.1 Hyperparameters

For the TD(λ) algorithm, I swept over the learning rates and the λ parameters. Learning rates were swept between 2^{-2} and 2^{-14} with a progression of 2^{-1} and λ between 0 and 1 with a progression of 0.1. Similarly, for the case of Momentum, the

same set of learning rates were swept, and the β was swept over the same range and values as λ for TD(λ).

5.4.2 Runs & Analysis

Each parameter configuration ran for 3000 time steps averaged over 200 runs, weights initialized to all zeros. I am going to use the last step performance (**END**) for comparing different methods. Last step performance refers to measuring the error in the learning process for the last 10% steps. The **END** performance will help us judge better the end stable point performance of the algorithm for the best hyperparameters.

I will be presenting 4 different kinds of plots in this report; the first plot will show the learning curve for each variant and sampling of the algorithm for the best set of hyperparameters for that variant. The second curve will show the sensitivity of the agent's to their learning rate. The third plot will show the sensitivity of the agents with respect to the second hyperparameter, i.e. λ for TD(λ) and β for Momentum(0), to study the effect of these parameters. The fourth plot will be a waterfall plot which has been used in previous works (Ghiassian, 2018) to compare the general performance of all set of hyperparameters. In this plot, each point refers to a hyperparameter configuration the x coordinate is random, whereas they coordinate shows the error of that hyperparameter configuration.

6 Results & Discussions

We begin the investigation by comparing Momentum(0) and TD(λ), in the general Markovian Sampling case for the Random Walk environment.

6.1 Markov Sampling - Random Walk

In Figure 3, TD(λ), outperforms Momentum(0) for most of its hyperparameters settings on λ. This can be seen with Momentum(0) performing better than TD(λ) for smaller α. The Momentum(0) performance is bounded by the performance of $\beta = 0$ which effectively translates to TD(0). With the increase in β, the performance drops ever so slightly, i.e. it is largely really not sensitive to the value of β. From the waterfall plot, it is again evident that most of the hyperparameter configurations achieve lower performance when compared to TD(λ), particularly because of insensitivity to β. The result also provides an initial validation that the Momentum(0) update is more biased as evident from the leftmost plot; we perform further analysis below to check this claim. TD(λ) results are as expected, an intermediate value of λ performs best.

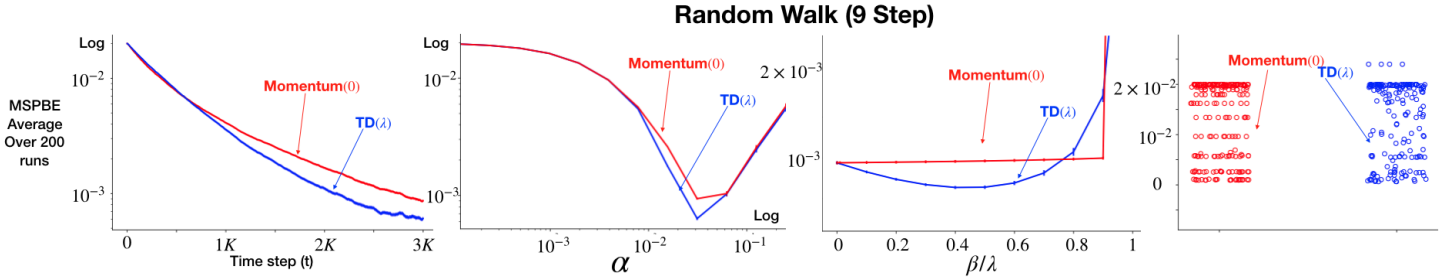

Figure 3: Compares the MSPBE of TD(λ) vs Momentum(0), with Markov sampling on the 9 state Random walk. In the first 3 curves (left, middle left, middle right), the free parameters are optimized to minimize **END**. The left Figure shows the learning curve on a log scale for better differentiation. The 2 middle figures exhibit the α and (β, λ) hyperparameter sensitivity for both the methods. The rightmost Figure lists the **END** error for all possible parameter configurations with the diverged values sitting at the top separated from the rest. All results are averaged over 200 independent runs. Error bars indicate ± 1 standard error.

As discussed above, Momentum(0) looks more biased than TD(λ) and fits my hypothesis. However, I hypothesized that the bias was due to the δ trace of Momentum (which means increasing β should degrade the performance) as evident from initial results. However, one can argue that Momentum was designed to work for I.I.D. sampling and there can be some evident bias because of the Markov Sampling in our first set of experiments. To validate my hypothesis further, I conducted a set of experiments based on Exact I.I.D. sampling to factor out the bias due to sampling, results for which are shown in Figure 4.

6.2 Markov and I.I.D. sampling - Random Walk

Figure 4 depicts that I.I.D. sampling does not help in improving the performance of Momentum(0), which is pretty counter-intuitive as it as designed to work on I.I.D. sampling. The Middle Right figure ever so slightly tells us that I.I.D. sampling is actually worse. I.I.D. sampling, as expected, breaks the backward view for TD(λ) and degrades the performance for increasing

λ due to irrelevant correlation. The waterfall plots again validate the above points by showing TD(λ) performing better for most of the parameters.

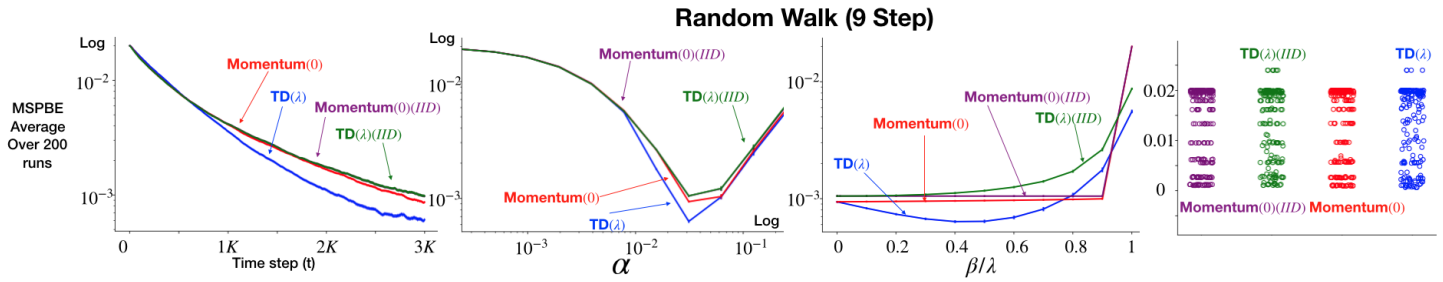

Figure 4: Compares the MSPBE of TD(λ) vs Momentum(0), on Markov and I.I.D. sampling for the 9 state Random Walk. In the first 3 curves (left, middle left, middle right), the free parameters are optimized to minimize **END**. The left Figure shows the learning curve on a log scale for better differentiation. The 2 middle figures exhibit the α and (β, λ) hyperparameter sensitivity for all 4 methods. The rightmost Figure lists the **END** error for all possible parameter configurations with the diverged values sitting at the top separated from the rest. All results are averaged over 200 independent runs. Error bars indicate ± 1 standard error.

The above result takes us a step further in actually saying that Momentum(0) is more biased with the increase in β. One experiment that can help us study this further is comparing the performance for very long runtime, and to see how Momentum applied to TD(λ), performs and how β and λ interact. The experiments in Figure 5 provide us with some insights.

6.3 Asymptotic Runs

Figure 5 (left) shows an asymptotic run with 20K time steps and bolsters the point that Momentum(0) is more biased than TD(λ). Also seeing the performance of Momentum on TD(λ), Momentum strictly does not help improve the performance of any value of λ; instead, it always hinders the learning in the case of Random Walk as evident in the (middle right, right) Figure.

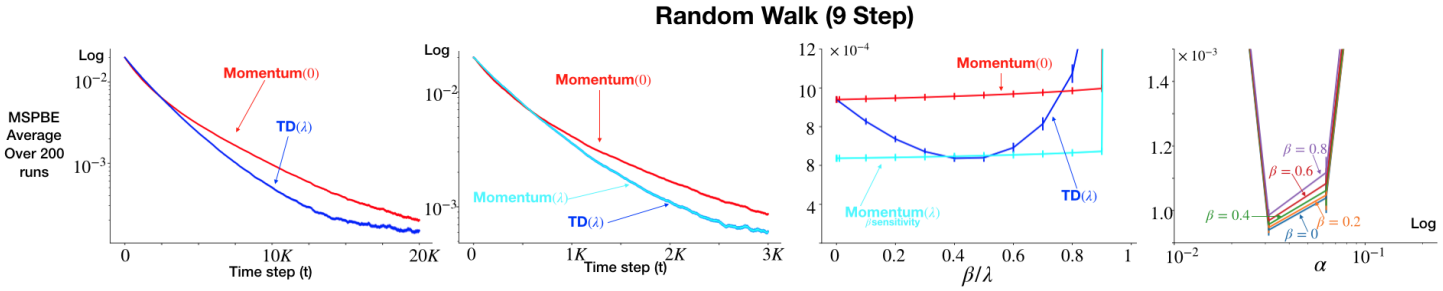

Figure 5: (Left) Comparing the asymptotic performance of TD(λ) with Momentum(0) on 9 state Random Walk for 20K time steps. (MIddle Left), The learning curves for Momentum(0) vs TD(λ), and Momentum(λ). (Middle Right) Showing the sensitivity of Momentum(λ) to the β parameter of Momentum. (Right) Showing the affect of β on the performance of Momentum(0), plotted for different learning rates. All results are averaged over 200 independent runs. Error bars indicate ± 1 standard error.

Testing on one environment provides little validation for the hypothesis, testing on further environments will be helpful, to do this I conducted additional experiments on the Boyan's Chain and Figure 6 illustrates the result of the same.

6.4 Boyans Chain - Markov Sampling

Figure 6, borrows structure heavily from figure 3, and shows the results Momentum(0) vs TD(λ) on Boyan's Chain, and environment where there is some extent of generalization. The results bolster the results that we got in the case of Random Walk, with $\beta = 0$ again performing the best for Momentum(0), showing that again Momentum hinders the performance. One more thing surprising is the layering of the errors in the waterfall plot (right) for the case of Momentum.

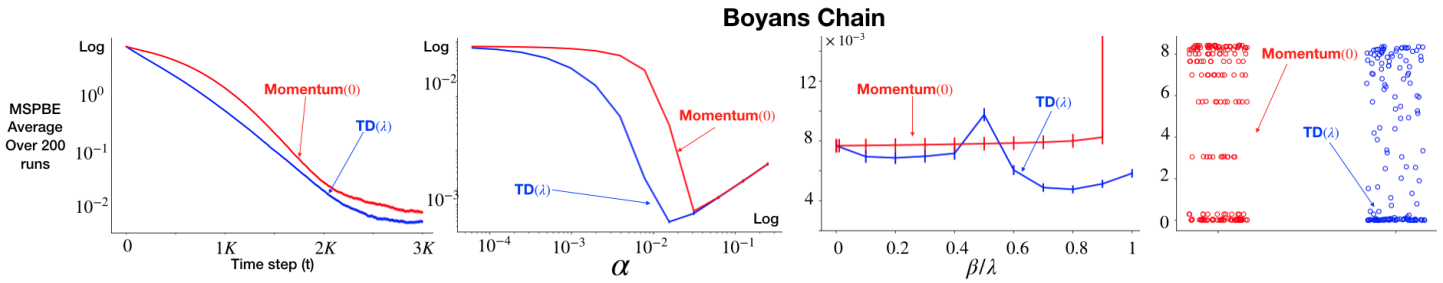

Figure 6: Comparing the MSPBE of TD(λ) vs Momentum(0), with Markov sampling on Boyan's Chain. In the first 3 curves (left, middle left, middle right), the free parameters are optimized to minimize the **END**. The left Figure shows the learning curves on a log scale for better differentiation. The 2 middle figures exhibit the α and (β, λ) hyperparameter sensitivity of both the methods. The rightmost Figure lists the **END** error for all possible parameter configurations with the diverged values sitting at the top separated from the rest. All results are averaged over 200 independent runs. Error bars indicate ± 1 standard error.

7 Conclusions

Herein I an initial set of experiments and results to test my hypothesis. Based on my experiments on 2 environments, my hypothesis seems to hold. We can see that indeed Momentum does not help us improve the convergence speed or the solution quality even when compared to the base case of not applying Momentum showing that it strictly hurts the performance. The experiments also allow us to validate that the bias stated is not exactly due to the sampling bias, which leaves TD error's bias as one of the major possibilities. The fact that performance degrades with increasing bias also points to the fact that accumulating gradients (i.e. δ) can, in turn, makes the update more biased and final solution less favourable. The thing that can be of concern is that the above two environments that were chosen have very low variance in the terms that most of the transitions are deterministic. Maybe testing on environments which are more stochastic can show some benefits of Momentum.

References

Boyan, J. A. (1999, June). Least-squares temporal difference learning. In ICML (pp. 49-56).

Ghiassian, S., Patterson, A., White, M., Sutton, R. S., & White, A. (2018). Online Off-policy Prediction. ArXiv:1811.02597

Kingma, D. P., & Ba, J. (2017). Adam: A Method for Stochastic Optimization. ArXiv:1412.6980

Maei, H. R. (2011, Fall). Gradient Temporal-Difference Learning Algorithms. Ph.D. Thesis, University of Alberta, Edmonton

Pan, Y., White, A., & White, M. (2017). Accelerated Gradient Temporal Difference Learning. ArXiv:1611.09328

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533–536

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

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (Second edition). The MIT Press.

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26-31.