Investigating Sensitivity of Step Size and Performance in True online TD(λ) for Different Values of λ

Kiarash Aghakasiri

RL II Course Project, University of Alberta, Edmonton, Canada

Abstract. In this project, I compare the performance of True online TD(λ) for five different values of λ and three different state representations. The comparison is from two perspectives. First, I compare them in terms of convergence. To compare the convergence I investigate the speed of convergence and the magnitude of error in the convergence point. The second comparison is checking the sensitivity of step size for different values of λ. First, I show that True online TD(λ) when λ=1 not only converges to a better point for deterministic environments but it is also faster than using lower values of λ. Secondly, using tile-coded features the algorithm seems to have less error than using state aggregation or binary-coded features after convergence. Finally, for higher values of λ, True online TD(λ) is more sensitive to the change in step size except in state aggregation which has the least sensitivity between the three representations in this project.

1 Introduction

The problem in all forward view algorithms is that the agent needs to wait until it gets the target, an n-step return, for the update. You can see the n-step return in equation 1. For large values of n, the state value update happens many steps after the agent has actually been into that state and it makes it delays the learning. Despite being slow we still want to use large values of n to reduce the bias caused by bootstrapping.

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) \tag{1}$$

A substitution for n-step return is λ-return which you can see its formula on equation 2. λ-return is the weighted average of all n-step returns. When λ is equal to zero the return is the same as one step return, and when λ is equal to 1 it is the same Monte Carlo return. We can choose the value of λ between zero and one to have almost the same effect as n-step returns. Forward view implementation of λ-return has the same disadvantages described above for n-step returns. The good thing with λ-return is that it can be implemented in backward view. In this view instead of waiting to get the n-step reward the agent carries a trace with itself, so at each state, the agent knows about the trajectory and updates are based on this trace. True online TD(λ) is a backward view algorithm which is the exact equivalent of the online λ-return algorithm which is the forward view version. Due to its importance, it is worth understanding the behaviour of this algorithm.

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t \tag{2}$$

True online TD(λ) has another parameter, λ, to tune. In this project, I investigate the performance of different values of λ. I also compare three different state representations for True online TD(λ). Thus, I answer the following questions:

- For true online TD(lambda): at what value of lambda is learning the fastest, and at what value of lambda is the converged error smaller?
- For true online TD(lambda): for which of the chosen state representations is the converged error smaller?
- For true online TD(lambda): at which λ value and for which state representation is more sensitive to the change of step size?

2 Methodology

In this section, I describe the environment, algorithm, and the hypotheses.

2.1 Environment Setting

A simple and commonly used environment for the task of prediction is the Chain environment. Because of the simplicity, I decided to use this environment in this work. The environment consists of 19 states and 2 actions in each of these states, Left and Right. Each state has a number corresponding to it and when the agent takes 'Right' action the next state's number will be one more than the previous one and when it takes 'Left' action the number will be one less than the previous state. The environment starts in the middle state (S_{10}) and all the transitions have a reward of 0 except two transitions: In the leftmost state (S_0) choosing Left action results in -1 reward and the rightmost state (S_{18}) choosing Right action results in +1 reward. In both cases, the environment terminates and the next episode starts from the middle state (S_{10}) again.

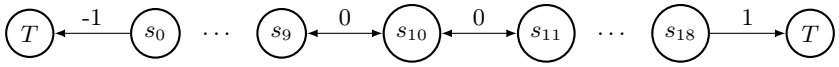

Fig. 1: Chain Environment

2.2 Algorithms

As I explained in the previous section True online TD(λ) is equal to the online version of the λ-return algorithm. But this statement is only true for linear function approximation. Algorithm 1 shows the pseudo-code of True online TD(λ).

The policy of the agent is completely random and in each state it has equal chance to choose between going left or right. I used three different state representations in this project which explained here:

- State-aggregation: I aggregated each group of three states as one. Since the environment has 19 states so it had 7 groups in total. Then the representation for each state is the one-hot vector corresponding to its group so each state was represented by 7 features.

- Tile-coding: I used 3 tilings and 4 tiles in each of the tilings so in total there were 12 tiles. Thus, each state was represented by 12 features corresponding to active tiles for that state.

– Binary-coding: In this representation, I turned state numbers into their binary equivalent and use each bit as a feature. As the environment has 19 states so it needed 5 bits to represent the states. Thus, each state was represented by 5 features.

Algorithm 1: True online TD(λ) for fixed policy π

z = 0
$V_{old} = 0$
S = initialize from the environment
for $e <$ *MaxEpisodes* **do**
  Choose action A according to π
  Take A and get (R, S', isTerminal) from environment
  if *not isTerminal* **then**
    V = $w^T X(S)$
    V' = $w^T X(S')$
    δ = R + γ V - V'
    z = γ λ z + (1 - α γ λ $z^T X(S))X(S)$
    w = w + α (δ + V - V_{old})$z - \alpha$(V - V_{old})$X(S)$
    $V_{old} = V'$
    S = S'
  else
    V = $w^T X(S)$
    δ = R + γ V
    z = γ λ z + (1 - α γ λ $z^T X(S))X(S)$
    w = w + α (δ + V - V_{old})$z - \alpha$(V - V_{old})$X(S)$
    z = 0
    $V_{old} = 0$

2.3 Hypotheses

Here are my hypothesis for each of the questions I introduced in the introduction.

In order to answer the question about using which value of λ the algorithm converges faster and for which one it converges to a better answer, I map them to n-step TD. We know that 1-step TD (λ = 0) has low variance but high bias because of bootstrapping. On the other hand, Monte Carlo (λ = 1) has a high variance but no bias as it doesn't bootstrap. Thus I hypothesize that when $\lambda = 0$ the algorithm converges faster, because of the low variance, but to a worse point, because of the high bias and on the contrary when $\lambda = 1$ it converges slower but to a better point.

The second question is about how well the algorithm converges for different state representations. For a representation to have a better convergence point, it has to cover more parts of true state value space. Here tile-coding has 12 features, state aggregation has 7 features and binary-coding has 5 features. Just base on the number of features, I hypothesize that the function approximation space with tile-coding is bigger and can cover more part of the true state value space. Thus, using tile-coding the algorithm should converge with less error.

The last question is about the sensitivity of the algorithm for different values of λ and state representations on the change of step size. When we have bigger λ we end up with bigger values in the eligibility vector. Since we update our weights using eligibility vector, higher values in the vector need a lower value for step size to be compensated. So, when we have a high step size there is a higher chance of divergence at bigger values of λ than lower values. Thus, my hypothesis is that at higher values of λ the algorithm is more sensitive to the change of step size.

Now I want to discuss the second part of the question which is the same thing but for different state representations. We know with our rule of thumb for step size, equation 3, that step size is inversely proportional to $E[XX^T]$. So, I want to calculate this expected value for each of the representations to answer the question. For tile coding, I know that this expectation is equal to the number of tilings which in my case is 3. For state aggregation, the expectation is equal to 1. For binary coding, I calculated this expectation using the stationary distribution of the policy and it is equal to 2.11 in this case. Based on these values, I hypothesize that tile coding is the most sensitive and state aggregation is the least to the change of step size.

$$\alpha \propto \frac{1}{\tau E[XX^T]} \tag{3}$$

3 Experiments

In this section I will explain how I implemented the experiments to answer the questions, then I will discuss the results one by one. I used the chain environment which I explained in section 2, with 19 states and the discount factor of 0.9 (γ). Implementation of the environment is based on the RLGlue framework. I used 5 different λ: [1, 0.9, 0.8, 0.4, 0]. I deliberately didn't choose them to be uniform because I had this prior knowledge that a small amount of difference for higher values of λ has a huge impact in terms of sensitivity to the step size (Sutton and Burto, 2018). The weight vector initialization is random and the random seed is equal to the run number. I ran each configuration many times and each run uses the same seed for all different configurations. For instance, if we are in run number three, all random initializations will use random seed 3.

In order to measure the performance, I chose mean square value error which shows the distance to the true state values and is the one that we truly want to minimize if we are able to. Calculating MSVE needs the true state values and the stationary distribution. You can see the definition of MSVE in equation 4. One way of calculating them is to write the Bellman equation and stationary distribution equation and solve them. An easier way is sampling which I used in the project. I initialized the environment for 10000 times and let the agent follows its policy and calculated the average return it got from each state and used it as an estimation of the true state values and also calculated the fraction of the time it spent on each state for the stationary distribution. You can see the true values used in my experiments in the table below.

S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18
-0.90	-0.80	-0.70	-0.60	-0.50	-0.40	-0.30	-0.20	-0.10	0	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90

Table 1: This table shows the true state values for each state used in this project. Numbers cut off after two floating points.

$$MSVE(w) = ||v_w - v_\pi^*||_\mu^2 \tag{4}$$

Now I will explain how I implemented the experiments. To compare the convergence for different λs, I used 5 different values and ran the algorithm 30 times for each λ. Then took the average of these 30 run times at each episode and drew learning curve plot for three different step sizes: 2^{-3}, 2^{-5}, 2^{-7}. You can see the plots in figure 2.

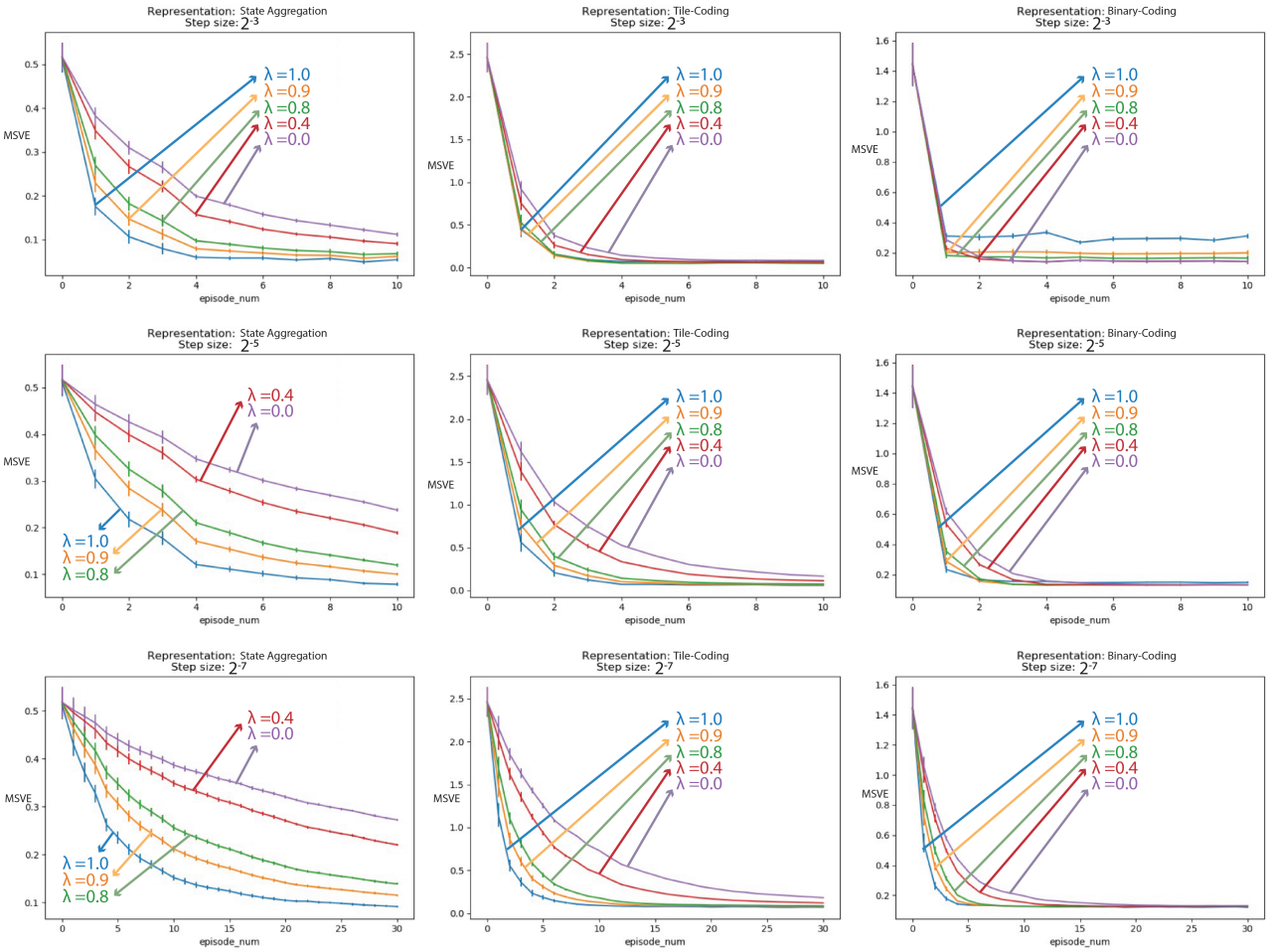

Fig. 2: The X-axis is the number of episodes and the Y-axis is MSVE. Y-axis scales vary for each representation. Left column is for state aggregation, the middle one is tile coding, and the right column is binary coding. First row is for step size = 2^{-3}, second row is for 2^{-5} and third row is for 2^{-7}. The error bars are smaller than the thickness of the lines for some points, so they might not be visible. The figure shows that for λ=1 the algorithm converged faster and had the best convergence point in this deterministic environment except with the highest step size and in binary-coded features, both at the same time, that it diverged.

From the above graphs, we can see in all three representations and step sizes that when λ=1 it converged to a better point except in binary representation with biggest step size which I will explain

the reason later. So the first part of my hypothesis is correct but at λ=1 the algorithm also converged faster which contradicts my hypothesis. I hypothesized that for λ=1 the algorithm has a higher variance so it will converge later than the other values of λ. Here our environment is deterministic so there shouldn't be much variance in the data. I think determinism of the environment is the reason my hypothesis was wrong. As λ=1 has higher values in the trace vector and variance is low so at λ=1 not only it converged to a better point but also it converged faster.

Now, to have a good measurement of their speed of convergence in my experiment, I decided to calculate the variance for each of them. So, I calculated the variance of error over the 30 run times at each episode and then took the average for all the 50 episodes using the three different step sizes step size (2^{-7}, 2^{-5}, 2^{-3}). Below you can see the table of the variances. As I expected for two lower step sizes higher λs cause lower variance. But for the bigger step size (2^{-3}) in binary coding, this sequence somehow got interrupted. I think for this step size and higher λ values it couldn't converge (it keeps going back and forth in error). The divergence for bigger step size gets us to sensitivity to the change of step size that I will investigate later on.

step size $= 2^{-7}$	$\lambda = 1$	$\lambda = 0.9$	$\lambda = 0.8$	$\lambda = 0.4$	$\lambda = 0$
State aggregation	0.0020	0.0028	0.0036	0.0067	0.0090
Tile coding	0.0345	0.0411	0.0472	0.0671	0.0861
Binary coding	0.0143	0.0166	0.0186	0.0235	0.0257

step size $= 2^{-5}$	$\lambda = 1$	$\lambda = 0.9$	$\lambda = 0.8$	$\lambda = 0.4$	$\lambda = 0$
State aggregation	0.0013	0.0015	0.0016	0.0023	0.0030
Tile coding	0.0262	0.0274	0.0293	0.0381	0.0456
Binary coding	0.0122	0.0123	0.0127	0.0145	0.0154

step size $= 2^{-3}$	$\lambda = 1$	$\lambda = 0.9$	$\lambda = 0.8$	$\lambda = 0.4$	$\lambda = 0$
State aggregation	0.0016	0.0014	0.0013	0.0014	0.0015
Tile coding	0.0238	0.0240	0.0241	0.0256	0.0278
Binary coding	0.0187	0.0122	0.0117	0.0121	0.0126

Table 2: This table shows the average of variances of different configurations over 30 runs and 50 episodes. We can see that the algorithm with bigger values of λ, had lower variance in this deterministic environment which means they learned faster.

To compare different representations, this time I used three different λs (0, 0.8, 1.0) and drew the plots for the same λ but different representations at each plot. You can see the results in figure 3. Different representations have different starting points in figure 3. The reason for this difference is although the weight initialization is the same in all representations, similar weights result in different state values. Having different state values means different errors. As you can see in figure 3 that in all these cases tile coding has the lowest error as I expected and binary coding has the highest error. Only in $\lambda = 0$ and step size $= 2^{-7}$, binary coding showed a lower error than state aggregation which is because the step size is too small for state aggregation representation to converge in 50 episodes. When I checked it with 100 episodes state aggregation reached a better point (I didn't include this

graph for the sake of compactness).

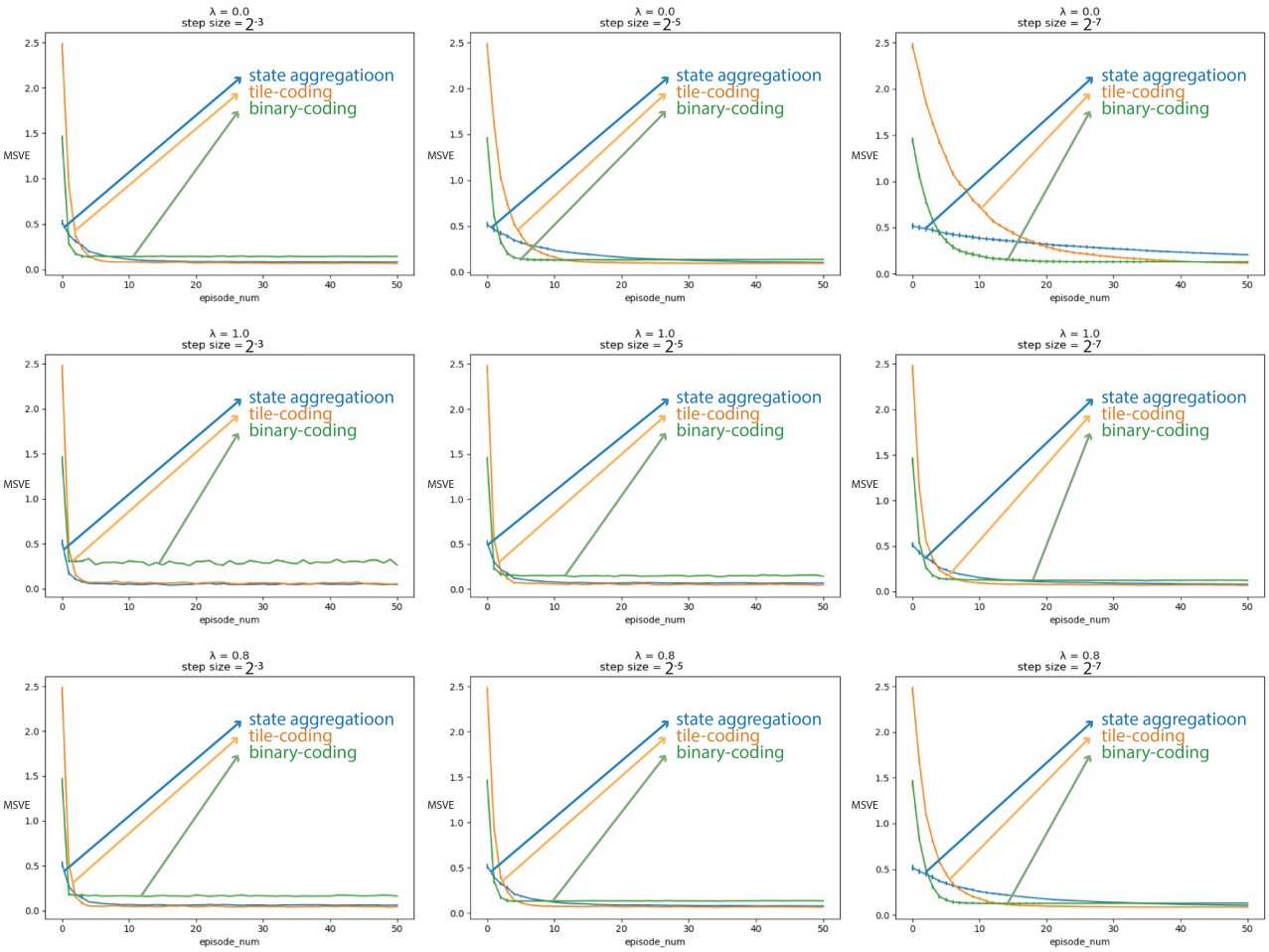

Fig. 3: The X-axis is number of episode and the Y-axis is MSVE. Step size is 2^{-3},2^{-5}, and 2^{-7} in left, middle, and right column respectively. λ is equal to 0, 1, and 0.8 in the first, middle, and the third row respectively. The error bars are smaller than the thickness of the lines for most of the points, so they might not be visible. As the figure shows, using tile-coding the algorithm converged to a better point than state aggregation and binary coding.

The third hypothesis was about the sensitivity of the algorithm at different λs and state representations to the change of step size. To investigate this, I ran the algorithm with 5 different values of λ and 3 different state representations. For this experiment, I chose this range of step sizes (1250 different step size): $\frac{1}{1000}, \frac{3}{1000}, \frac{5}{1000}, \ldots, \frac{2499}{1000}$. For each of them I ran the algorithm for 20 episodes, and then took the average error over the last 5 episodes. I ran each of them for 30 times and then took the average over the all runs at each point for step sizes. The reason I chose 20 to be the maximum number of episodes was I saw in figures 2 and 3 that even for step size $= 2^{-7}$ the algorithm almost converged after 20 episodes. The results are shown in figure 4, the reason that the plots look noisy is

that the offset that step sizes are changing is very small (0.002).

We can see from the plots that in tile-coding and binary-coding, higher values of λ have a lower range of step sizes with a low error. This means that higher values of λ are more sensitive which confirms my hypothesis. On the contrary, for state aggregation, there ordering appears to be different for reasons that are not clear.

Comparing different representations, we can see that state aggregation (right plot) has the widest range of step sizes with a low error which means it is the least sensitive to the change of step size, which is the same as the hypothesis. I hypothesized that binary-coding should be less sensitive than tile-coding which is not true according to figure 4. I think one problem with binary-coding is that it is not normalized. Also, it probably cannot cover much from the true state space as it has only 5 features, so it doesn't have an acceptable error for any step size. In this project I only used one configuration of tile-coding and state aggregation so the comparison between the representations is not as rigorous as it could be. Because of this, I emphasized the comparison between the values of λ, not the representations.

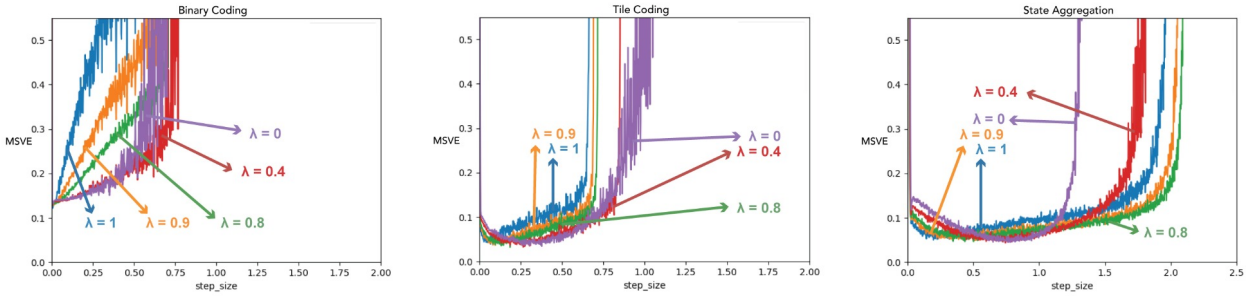

Fig. 4: In all plots, the X-axis is the step size and Y-axis is the average error over the last 5 episodes (episodes 15 to 20). The state representations are binary-coding, tile-coding, and state aggregation from left to right. As we can see in the figures, state aggregation has the highest tolerance for step size changing, it works with higher step sizes as well. Also we can see as λ gets bigger it gets more sensitive to the change of step size in tile-coding and binary-coding representation.

Since the standard error was too small and plots in figure 4 were noisy the error bar wouldn't be visible at all. Hence, I am reporting the average of standard error over all step sizes for each configuration in the below table.

	Binary-Coding	**Tile-Coding**	**State Aggregation**
$\lambda = 1$	0.0112	0.0232	0.0124
$\lambda = 0.9$	0.0120	0.0275	0.0068
$\lambda = 0.8$	0.0125	0.0303	0.0070
$\lambda = 0.4$	0.0133	0.0369	0.0229
$\lambda = 0$	0.0134	0.0405	0.0083

Table 3: This table shows the average of standard error over step sizes for 5 different values of λ and 3 different state representations for 30 runs.

4 Conclusion

To summarize, I investigated the performance of True online TD(λ) algorithms for different values of λ and three different state representation, in convergence and sensitivity. Based on the results, my first conclusion is that in a deterministic environment like Chain, when λ=1 and with right step size, True online TD(λ) not only converges to a point with less error but also it converges faster in comparison with lower values of λ. Moreover, when λ=1 the algorithm is more sensitive to the change of step size and it will diverge with lower step sizes. On the other hand, using lower values for λ, the algorithm is more stable for higher step sizes. Furthermore, tile-coding is a better state representation than binary-coding and state aggregation in terms of error at the convergence point. Lastly, using state aggregation as the representation rather than tile-coding or binary-coding reduces the sensitivity of True online TD(λ) to the change in step size. So, with state aggregation, we can be less worried about tuning the step size. Since the experiments only used one configuration of these representations, the last two conclusions may not be as strong as the first two and need a more thorough investigation.

References

Sutton, R. S., Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT press.