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 5 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 investigated 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 showed that λ=1 not only converges to a better point for deterministic environments but it is also faster. Secondly, tile-coding seems to have less error than state aggregation and binary-coding after convergence. Finally, higher values of λ are 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 good solution is backward view. In backward view the target changes to λ-return which you can see its formula on equation 2. 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:

- Which λ learns faster and which one has less error at its convergence point?
- Which of the chosen state representations will converge to a point with a less error?
- Which λ and state representation show more change in the error by 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.

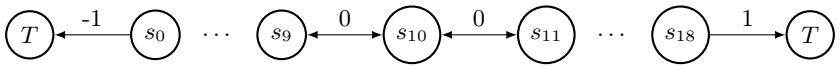

Fig. 1: Chain Environment

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

The second question is about how different state representations converge. 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, tile-coding should have less error.

The last question is about the sensitivity of 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 for bigger λ values to diverge than lower values. Thus, my hypothesis is higher values of λ are 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. Base 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 about how I implemented the experiments to answer the questions, then I will discuss about the results one by one. I used the chain environment which I explained in section 2, with 19 states and the discounting 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 (Sutton and Burto, 2018). The weight vector initialization is random and the random seed is equal to the number of run times. I ran each configuration many times each run time has the same seed for all different configurations. For instance, if it is the third time that same feature representation with the same λ is running so the random seed will be 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.

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

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.

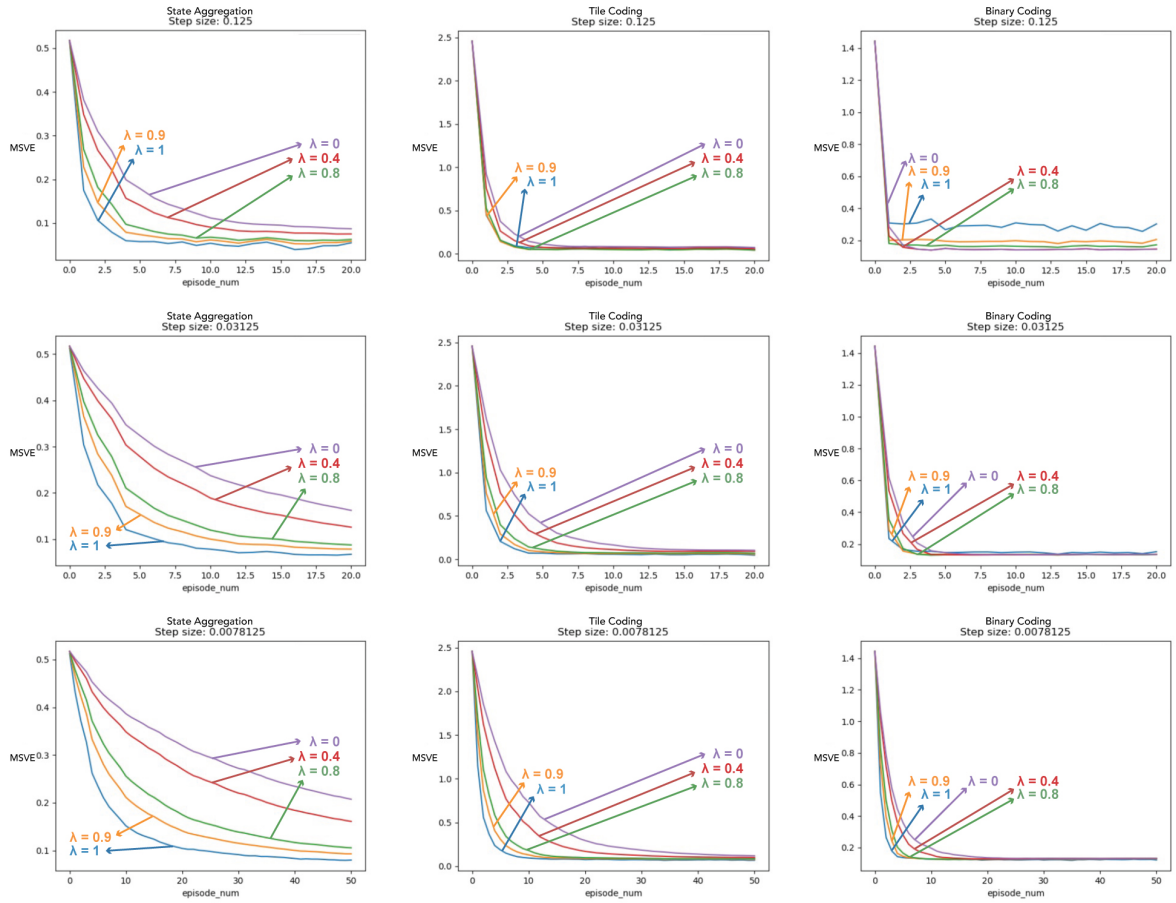

Fig. 2: The X-axis is number of episode and the Y-axis is MSVE. Left column is for state aggregation, the middle one is tile coding, and the right column is binary coding. First row is for step size = 2^{-3}, second row is for 2^{-5} and third row is for 2^{-7}. The figure shows that λ=1 is faster and has best convergence point in this deterministic environment except with high step sizes that it diverges.

From the above graphs, we can see in all three representations and step sizes that λ=1 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 λ=1 also converged faster which contradicts my hypothesis. I hypothesized that λ=1 has a higher variance so it will converge later than the others. 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 λ=1 not only converged to a better point but also 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 1: This table shows the variance of different configurations over 20 runs and 50 episodes. We can see that bigger λs has lower variance in this deterministic environment which means they learn 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 λ=1 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).

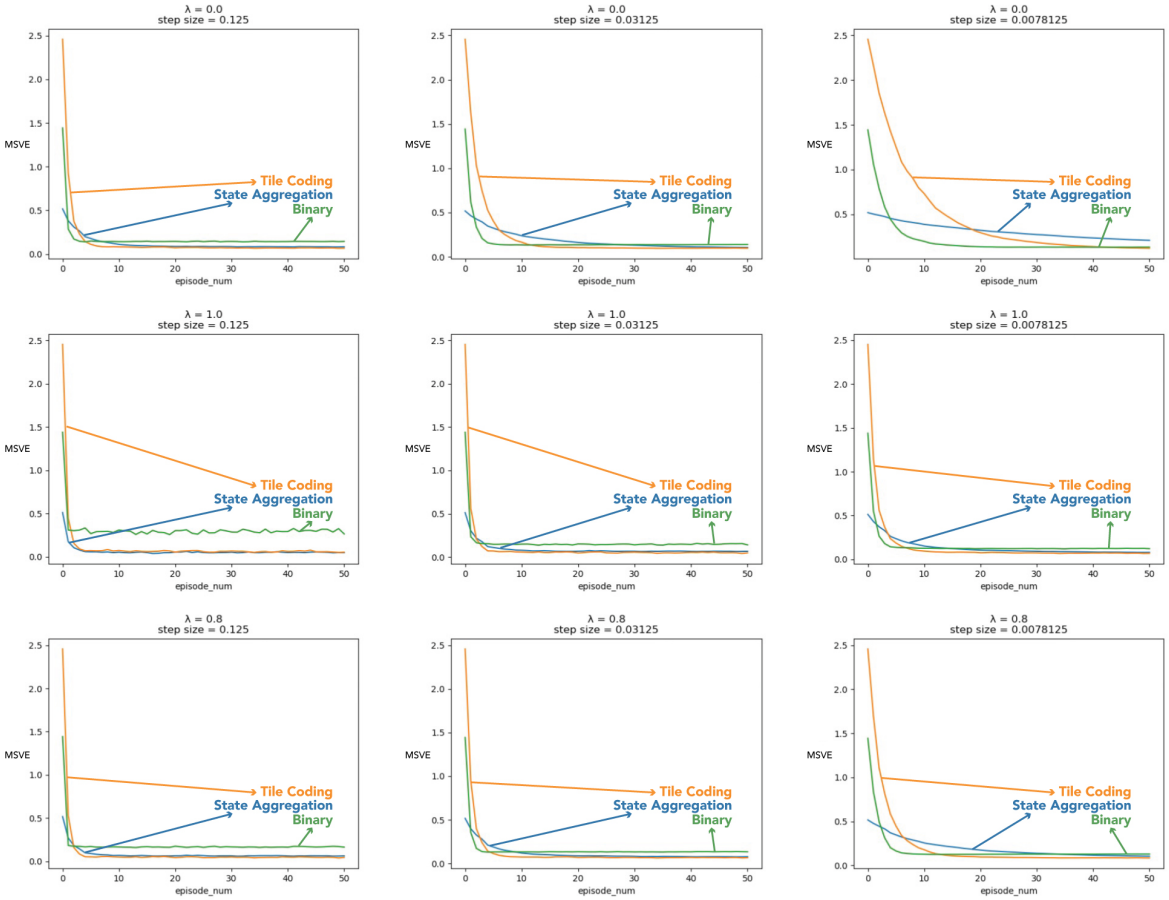

Fig. 3: The X-axis is number of episode and the Y-axis is MSVE. Step size is 2^{-3},2^{-5}, and 2^{-7} in left, middle, and right column respectively. is equal to 0, 1, and 0.8 in the first, middle, and the third row respectively. As figure shows, tile-coding converges to a better point that state aggregation and binary coding.

The third hypothesis was about the sensitivity of 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 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.

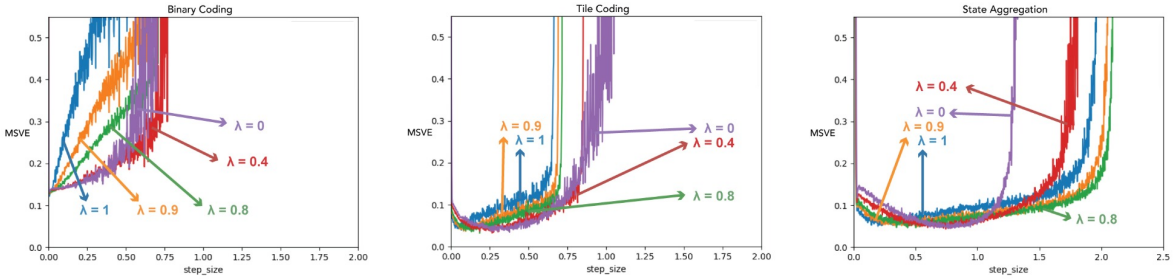

Fig. 4: In all plots, the X-axis is the step size and Y-axis is the average error over the last 5 episodes (episodes 15 to 20). The state representations are binary-coding, tile-coding, and state aggregation from left to right. As we can see in the figures, state aggregation has the highest tolerance 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.

4 Conclusion

To summarize, I investigated the performance of True online TD(λ) algorithms for different values of λ and 3 different state representation, in convergence and sensitivity. The results showed that in a deterministic environment like Chain, λ=1 converges faster and to a point with a less error unless the step size is too high. λ=1 is more sensitive to the change of step size at least in two of the representations so for large step sizes it will diverge. Moreover, in my experiments, I saw that tile coding converged to a better point than state aggregation and binary coding. I also observed that state aggregation has the lowest sensitivity for the change of step size between these three representations.

References

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