An Empirical Study of Least-Squares Algorithms in Reinforcement Learning

Howard Huang HOWARD.HUANG@MAIL.MCGILL.CA
School of Computer Science
McGill University
Montreal, QC, H3A 2A7, Canada

Doina Precup DPRECUP@CS.MCGILL.CA
School of Computer Science
McGill University
Montreal, QC, H3A 2A7, Canada

Abstract

Due to its many successes, the TD(λ) class of algorithms is a popular choice when it comes to solving control problems. One such algorithm which was recently proposed is Q(σ), which unifies two existing TD solutions. This kind of unification is a popular technique for creating new families of algorithms, which include the original algorithms as special cases. Here, we will do the same by applying Q(σ) traces to LSTD(λ) to create a new class of algorithms we will call LSTDQ(σ). We then evaluate it against Q(σ), and against itself with different values of σ, all in the off-policy control setting. Our results show that with our formulation of LSTDQ(σ), there is little benefit to the additional flexibility. We also propose a small change to the algorithm for future work.

Keywords: Reinforcement Learning, least squares, eligibility traces, off-policy learning

1. Introduction

One of the main problems plaguing reinforcement learning is the high sample complexity. Some have attempted to tackle this issue through the credit assignment problem, wherein by properly assigning credit to the deserving transitions, the learning that comes with encountering a reward can be propagated further back in time. The standard solution to this credit assignment problem is the use of eligibility traces. SARSA(λ) and Tree-Backup are two algorithms employing these techniques. The Q(σ) algorithm, originally introduced by Sutton and Barto (2017), is a unification of SARSA(λ) and Tree-Backup. This brings about the existence of an entire family of new algorithms that lie between SARSA and TB, which would perform no worse (but ideally better) than the latter two. The works of De Asis et al. (2017) verified that this is the case for on-policy control with no eligibility traces on all of the tasks they tested.

Other attempts at improving sample complexity involve the reuse of data. Algorithms such as LSTD by Bradtke and Barto (1996) or LSQ by Lagoudakis et al. (2002) belong to the class of least squares algorithms, which preserve and reuse data rather than discarding them after one use, as is typically done with stochastic gradient descent. These works have also shown LSTD to have better sample complexity in the policy evaluation scenario than their SGD counterpart. LSQ was also shown to perform well on a simple control task, but with no comparison to Q-learning.

A unification of eligibility traces with LSTD has been proposed by Boyan (1999), dubbed LSTD(λ), which showed an improved performance when compared to SARSA(λ) on a policy evaluation task. It can also be used for action-value learning by selecting a set of features $\phi(s, a)$ which are a function of both the state and action.

With the goal of further improving sample complexity, we will be introducing an extension to LSTD(λ) to make use of the Q(σ) traces: LSTDQ(σ). This algorithm will be run against Q(σ) on three control problems from the OpenAI Gym library: FrozenLake-v0, CartPole-v0, and MountainCar-v0. Additionally, we will be testing the off-policy control scenario, in order to take advantage of Tree-Backup's off-policy nature.

2. LSTD(λ) with Q(σ) Traces

Q(σ) is an algorithm which uses a linear combination of the SARSA and Tree-Backup return as its own return. Namely, its n-step return is defined as

$$G_{t:t+n} = \sigma G^{\text{SARSA}}_{t:t+n} + (1-\sigma) G^{\text{TB}}_{t:t+n} \tag{1}$$

and its λ-return is

$$G^\lambda_t = Q(S_t, A_t) + \sum_{k=t}^{\infty} (\lambda\gamma)^{k-t} \delta_k \prod_{i=t+1}^{k} [(1-\sigma_i)\pi(A_i|S_i) + \sigma_i] \tag{2}$$

The update rule for Q(σ) using a linear function approximator, due to Sutton and Barto (2017), are

$$\delta_t = r_{t+1} + \gamma \left[\sigma Q(s_{t+1}, a_{t+1}) + (1-\sigma) \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s_{t+1})} Q(s_{t+1}, a) \right] - Q(s_t, a_t) \tag{3}$$

$$e_t = \lambda\gamma e_{t-1}[(1-\sigma_t)\pi(a_t|s_t) + \sigma_t] + \phi_t \tag{4}$$

$$\theta_t \leftarrow \theta_{t-1} + \alpha e_t \delta_t \tag{5}$$

where $Q(s, a) = \phi(s, a)^\intercal \theta$, with $\phi(s, a)$ being the feature vector describing taking action a at state s.

With that, we propose a new algorithm which extends LSTD(λ) to use Q(σ) traces. It can be simply derived by considering the sum of changes to θ_t in (5).

$$\begin{aligned}
\sum_t \alpha e_t \delta_t &= \sum_t \alpha e_t [r_{t+1} + \gamma \left[\sigma \phi^\intercal_{t+1}\theta + (1-\sigma)\bar{\phi}^\intercal_{t+1}\theta\right] - \phi^\intercal_t \theta] && (6)\\
&= \alpha \underbrace{\left[\sum_t e_t r_{t+1}\right]}_{=\mathbf{b}} - \alpha \underbrace{\left[\sum_t e_t [\phi_t - \gamma(\sigma\phi_{t+1} + (1-\sigma)\bar{\phi}_{t+1})]^\intercal\right]}_{=\mathbf{A}} \theta && (7)\\
&= \alpha(\mathbf{b} - \mathbf{A}\theta) && (8)
\end{aligned}$$

where $\bar{\phi}_t = \mathbb{E}_{a\sim\pi(\cdot|s_t)}\, \phi(s_t, a)$. Repeatedly applying the update $\theta \leftarrow \theta + \alpha(\mathbf{b} - \mathbf{A}\theta)$ leads to the fixed point $\theta = \mathbf{A}^{-1}\mathbf{b}$ as the solution. The resulting algorithm is presented in Algorithm 1. When $\sigma = 1$, this reduces to Boyan's LSTD(λ), and at $\sigma = 0$, gives LSTD with Tree-Backup traces.

Algorithm 1 LSTDQ(σ)

```
1:  $\theta \leftarrow \vec{0}$
2:  $e \leftarrow \vec{0}$
3:  $\mathbf{A} \leftarrow \mathbf{0}$
4:  $\mathbf{b} \leftarrow \vec{0}$
5:  for all $t$ do
6:      $e \leftarrow \lambda\gamma e[(1-\sigma_t)\pi(a_t|s_t) + \sigma_t] + \phi_t$
7:      if $s_t$ is not terminal then
8:          $\mathbf{A} \leftarrow \mathbf{A} + e[\phi_t - \gamma(\sigma\phi_{t+1} + (1-\sigma)\bar{\phi}_{t+1})]^\intercal$
9:          $\mathbf{b} \leftarrow r_{t+1}e$
10:     else
11:         $\mathbf{A} \leftarrow \mathbf{A} + e\phi_t^\intercal$
12:         $\mathbf{b} \leftarrow r_{t+1}e$
13:         $\theta \leftarrow \mathbf{A}^{-1}\mathbf{b}$
14:         $e \leftarrow \vec{0}$
```

3. Empirical Results

Q(σ) and LSTDQ(σ) were run on three environments from the OpenAI Gym library by Brockman et al. (2016): FrozenLake-v0, CartPole-v0, and MountainCar-v0. In terms of the features used, FrozenLake used a one-hot encoding of the state space, CartPole used the raw observations but with the two velocity values bounded by $f(x) = \tanh(x/10)$, and MountainCar used an RBF function approximator with the car position and velocity normalized to $[0, 1]$ and an array of 8×8 evenly spaced basis functions $\rho(x) = \exp(-r/0.01)$ covering this $[0, 1]^2$ space. ϵ-greedy policies were used for both the behaviour and target policies. A gridsearch was performed on each task over the following parameters: learning rate, the SARSA/TB weighting factor σ, the trace factor λ, ϵ on the behaviour policy, and ϵ on the target policy. The best-performing parameters were used in the comparison below.

The results of the experiments are shown in tables 1 and 2. In all cases, Q(σ) performs better with intermediate σ values, similar to the on-policy results of previous works. LSTDQ, however, tends toward the extremes.

	$\sigma = 0$	$\sigma = 0.25$	$\sigma = 0.5$	$\sigma = 0.75$	$\sigma = 1$
FrozenLake-v0	0.461	**0.551**	0.544	0.529	0.470
CartPole-v0	178.1	180.4	182.1	**183.5**	181.3
MountainCar-v0	-128.5	-125.6	-124.1	$\mathbf{-123.9}$	-124.9

Table 1: Mean reward per episode after running Q(σ) for 100 episodes.

	$\sigma = 0$	$\sigma = 0.25$	$\sigma = 0.5$	$\sigma = 0.75$	$\sigma = 1$
FrozenLake-v0	0.528	**0.530**	0.519	0.505	0.485
CartPole-v0	167.1	184.2	192.3	195.2	**196.8**
MountainCar-v0	**−129.2**	−130.7	−129.8	−135.1	−137.0

Table 2: Mean reward per episode after running LSTDQ(σ) for 100 episodes.

Using Q(σ) on FrozenLake, early learning favours a higher weighting on SARSA (σ closer to 1), while later learning favours a higher weighting on TB (σ closer to 0), but the extremes of pure SARSA and pure TB perform the overall worst. Meanwhile LSTDQ(σ) sees better performance as σ approaches 0, but with little variation with the different choices of σ.

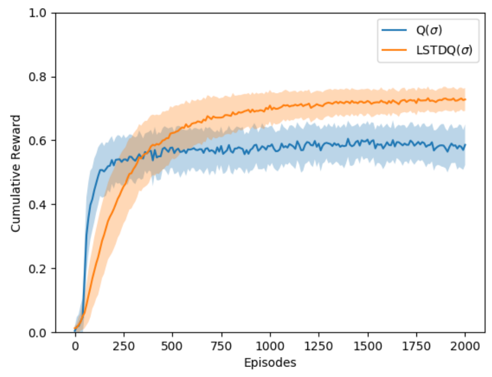
LSTDQ(σ) on CartPole is similar to its behaviour on FrozenLake in that there is a monotonic relationship between the cumulative reward and the choice of σ. However, in this case, the better performing algorithm was SARSA, with $\sigma = 1$. There is also a much larger difference in performance between Q(σ) and LSTDQ(σ) in this task. LSTDQ(σ) perfects that task almost instantaneously, with next to no performance variance past the first 1000 episodes, while SGD takes over 2000 episodes to reach the same average performance, and never truly converges.

In the case of MountainCar, Q(σ) outperforms LSTDQ(σ) over the course of the entire trial on average. If we look at the performance on individual trials, we see that every run with Q(σ) has the same behaviour: the average reward obtained jumps up to around -120, and fluctuates wildly around this average. LSTDQ(σ) on the other hand, has more consistent performance over time on good trials, where the average reward hovers around -110 with noticeably less fluctuation than Q(σ). However, there are also a few bad trials where the agent does not learn at all.

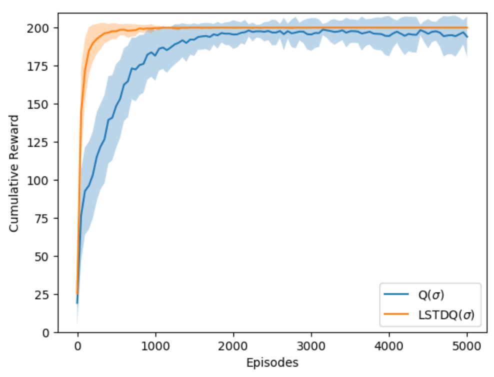
3.1 Sample Complexity

Between the three environments tested, only CartPole shows an improvement in using LSTDQ(σ) over Q(σ) when it comes to sample efficiency. The main property of CartPole that distinguishes it from FrozenLake and MountainCar is that a uniformly random policy can efficiently explore all of the state space, so that in this case, the data gathered during the first few episodes are just as useful as those that are gathered later on. This is further evidenced by the optimal parameters that were found through gridsearching. There is a clear trend in the data where high ϵ values in the behaviour policy are favoured over lower values.

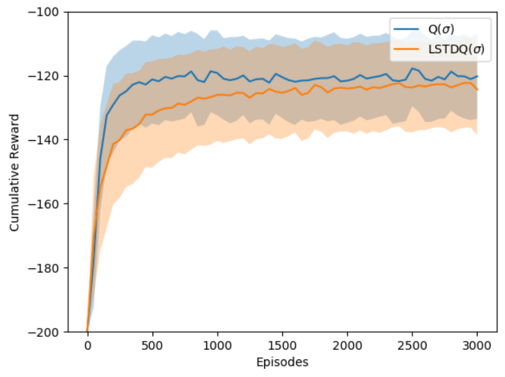
FrozenLake and MountainCar both require more exploration, and have to stumble upon the reward state by chance before they can start learning. Until a reward is encountered, LSTDQ(σ) will be gathering data from a random policy, which will continue to have an effect on the resulting policy for all future computations. In the case of Q(σ), the initial exploration affects the weights only as the data comes in, but since the data is discarded after being used for a single update, it ceases to have any significant influence beyond those updates. This could potentially be resolved with the addition of a decay hyperparameter, which would be used to reduce the influence of older data, similar to the LSQL algorithm described by Lagoudakis et al. (2002).



(a) FrozenLake



(b) CartPole



(c) MountainCar

Figure 1: Comparison of Q(σ) and LSTDQ(σ) on FrozenLake, CartPole, and MountainCar. The parameters chosen were those with the highest score, as measured by mean rewards obtained over the first 100 episodes.

3.2 Asymptotic Performance

Comparing the performance at the end of the trials, we see that LSTDQ(σ) generally matches or outperforms Q(σ) in terms of mean reward, as well as variance. The lower variance is unsurprising, as each data point used in LSTDQ(σ) are reused on each update and are all equally weighted, so the computed weights change less from one episode to the next and as more data is gathered. In the case of MountainCar, LSTDQ(σ)'s performance remains just below that of Q(σ), but not by a very significant margin ($p = 0.09$ in a two-tailed t-test over the last five episodes). It would appear that the agent is still slowly improving, so it is possible that had the experiment been run for longer, we would see LSTDQ(σ) catch up. Variance is also very close in the MountainCar scenario (std of 27.9 on LSTDQ(σ) and 25.4 on Q(σ)), but looking at the individual trials, we see that the good trials with LSTD are visibly lower variance than SGD, and the high variance of LSTDQ(σ) are due to the few failed trials where the agent completely fails to learn the task.

3.3 Robustness to Hyperparameter Choice

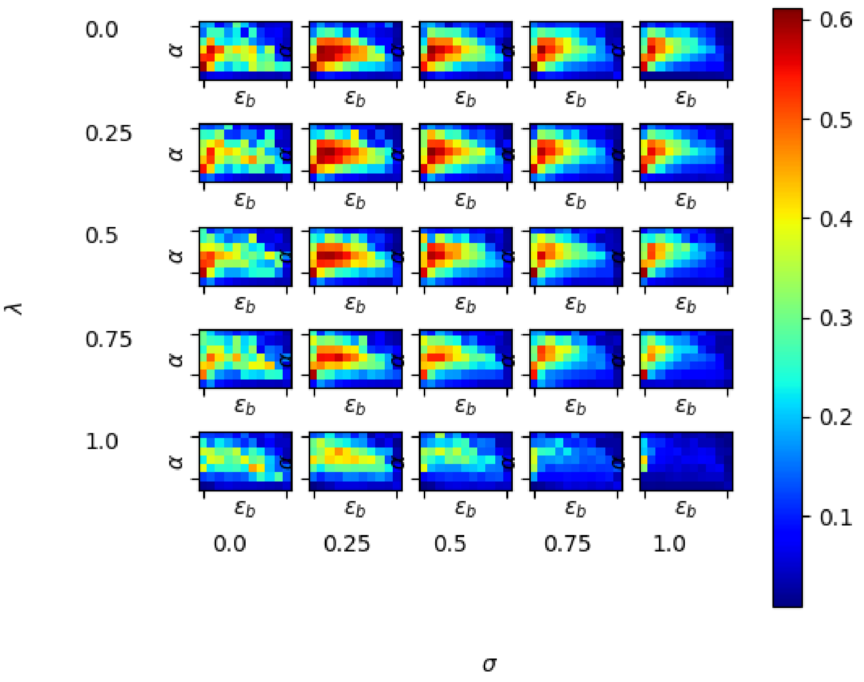


Figure 2: Performance as a function of hyperparameter choice in FrozenLake. Each of the plots in the 5×5 matrix represents a one of the combinations of λ and σ values. Within each plot, the horizontal axis represents the choice of behaviour policy ϵ, and the vertical axis represents the choice of learning rate. The values plotted represent the average rewards obtained per episode over the course of all trials.

Through a visual inspection of figure 2 for Q(σ), we can see that just like how intermediate σ values performed better in terms of mean reward, the same σ value also lead to a larger range of hyperparameters which also perform well. This behaviour was found in the CartPole and MountainCar task as well. And LSTDQ(σ), while favouring σ values of 1 or 0, also showed better robustness when the σ value with the best performance is selected.

4. Conclusion and Future Work

Although there is no clear-cut advantage to using LSTDQ(σ), we still see promising results, particularly in the case of CartPole, and the asymptotic behaviour on FrozenLake. Its failure in the MountainCar task and in the first few episodes of FrozenLake are likely due to the accumulation and reuse of data coming from a bad policy, while also trying to learn about a different bad policy. The application of a decaying factor on old data could help to resolve this issue. More work evaluating LSTDQ(σ) with this decay is in order.

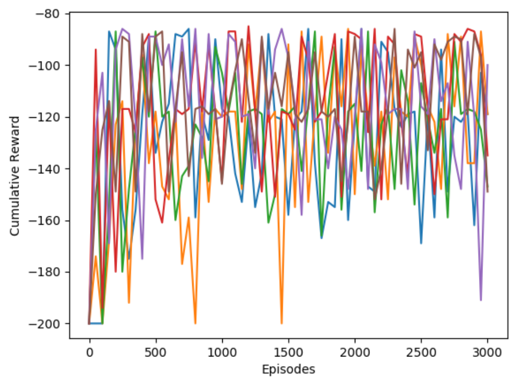
Appendix A. Additional Tables and Figures

	$\sigma = 0$	$\sigma = 0.25$	$\sigma = 0.5$	$\sigma = 0.75$	$\sigma = 1$
FrozenLake-v0	0.554	**0.623**	0.614	0.605	0.569
CartPole-v0	178.2	180.6	182.2	**183.6**	181.4
MountainCar-v0	−128.5	−125.6	−124.1	**−123.9**	−124.9

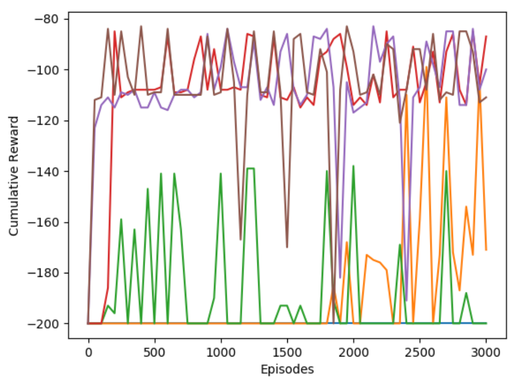
Table 3: Mean reward per episode after running Q(σ) for over 2000 episodes.

	$\sigma = 0$	$\sigma = 0.25$	$\sigma = 0.5$	$\sigma = 0.75$	$\sigma = 1$
FrozenLake-v0	**0.628**	0.624	0.614	0.605	0.591
CartPole-v0	167.1	184.3	192.4	195.2	**196.8**
MountainCar-v0	**−129.2**	−130.7	−129.8	−135.1	−137.0

Table 4: Mean reward per episode after running LSTDQ(σ) for over 2000 episodes.

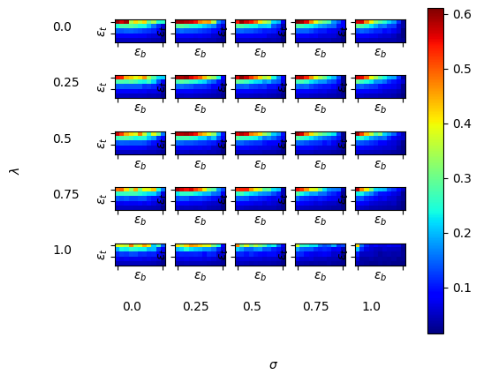


(a) Q(σ)

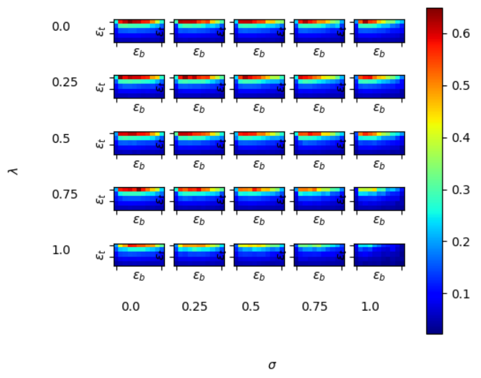


(b) LSTDQ(σ)

Figure 3: 3 best and worst performing trials on MountainCar. The parameters used are those which led to the best average performance over the first 100 episodes.

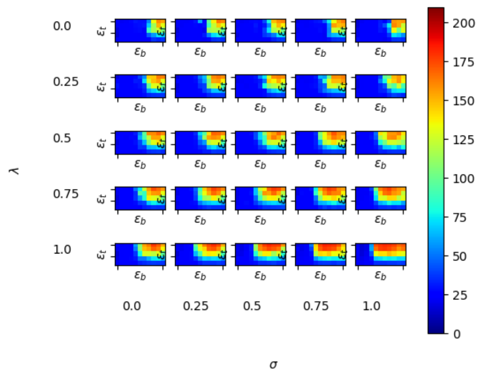


(a) Q(σ)

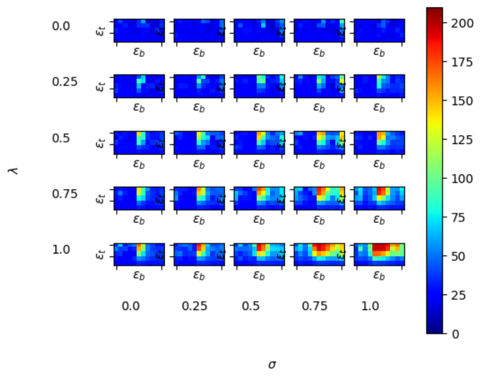


(b) LSTDQ(σ)

Figure 4: Average reward per episode as a function of hyperparameter choice in FrozenLake.

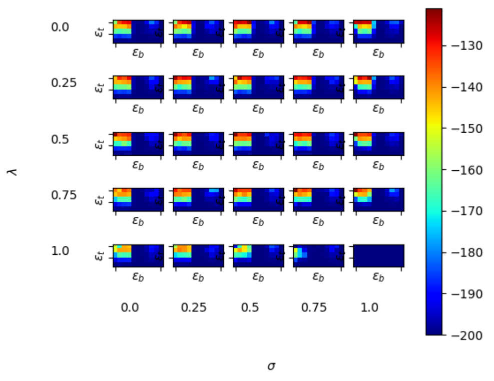


(a) Q(σ)

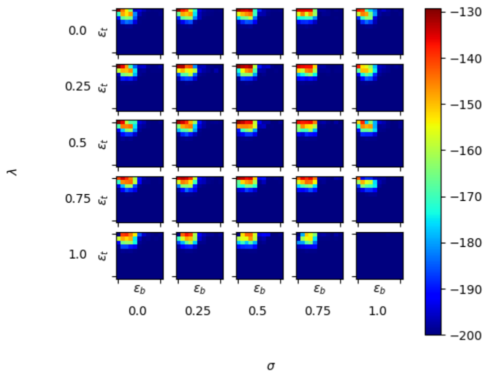


(b) LSTDQ(σ)

Figure 5: Average reward per episode as a function of hyperparameter choice in Cartpole.



(a) Q(σ)



(b) LSTDQ(σ)

Figure 6: Average reward per episode as a function of hyperparameter choice in Mountain-Car.

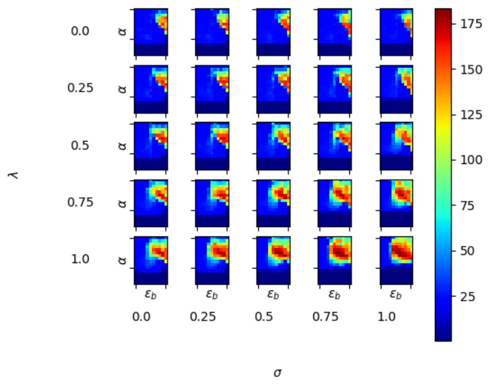


Figure 7: Average reward per episode as a function of hyperparameter choice in Cartpole.

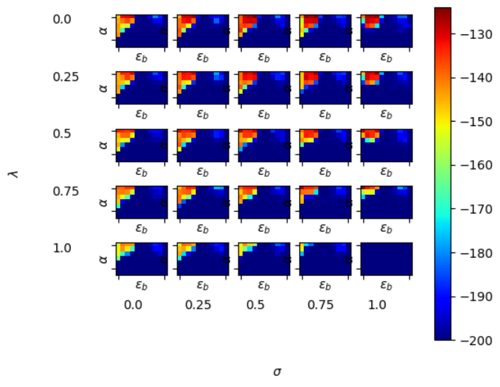


Figure 8: Average reward per episode as a function of hyperparameter choice in Mountain-Car.

References

Justin A Boyan. Least-squares temporal difference learning. In *ICML*, pages 49–56, 1999.

Steven J Bradtke and Andrew G Barto. Linear least-squares algorithms for temporal difference learning. *Machine learning*, 22(1-3):33–57, 1996.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Kristopher De Asis, J Fernando Hernandez-Garcia, G Zacharias Holland, and Richard S Sutton. Multi-step reinforcement learning: A unifying algorithm. *arXiv preprint arXiv:1703.01327*, 2017.

Michail G Lagoudakis, Ronald Parr, and Michael L Littman. Least-squares methods in reinforcement learning for control. In *Hellenic Conference on Artificial Intelligence*, pages 249–260. Springer, 2002.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction (2nd ed.)*, volume 1. MIT press Cambridge, 2017.