# Temporal Regularization for Control

**Raviteja Chunduru***
McGill University
raviteja.chunduru@mail.mcgill.ca

**Barleen Kaur***
McGill University
barleen.kaur@mail.mcgill.ca

**Surya Penmetsa***
McGill University
surya.penmetsa@mail.mcgill.ca

## Abstract

The stability of reinforcement learning algorithms is often affected by the variance of estimates during training. Temporal regularization is a technique which uses the values of past states in the trajectory, along with the values of future states, to estimate the value of the current state. Methods that have been previously proposed in temporal regularization are restricted to policy evaluation, where it has been shown that temporal regularization reduces variance and increases the rate of convergence. In this work, we extend temporal regularization to control. Specifically, we propose temporal regularization for SARSA, Q-learning and policy gradient methods and demonstrate the performance on several environments.

## 1 Introduction

Reinforcement algorithms have made remarkable progress in the recent past with the introduction of deep neural networks. However, these methods suffer due to the high variance in the estimates of values. One way to avoid this problem is through the use of regularization. Regularization, a widely applied technique in many reinforcement learning control algorithms, brings down the variance at the cost of introducing a small bias.

There are two common types of regularization applied in reinforcement learning: spatial regularization and temporal regularization. Spatial regularization is a result of function approximation where states in close spatial proximity have similar features, and in effect, similar values.

Temporal difference methods impose temporal regularization in that the value of current state is bootstrapped based on the values of future states along the trajectory. Taking this a step further, bootstrapping is done with the values of past states too. We exploit the information that the value function is coherent along the trajectory.

This allows for faster convergence and estimates with lower variance, while introducing small bias. The variance is lower because bootstrapping is based on values from the previous states in the trajectory that have already been updated and hence are more stable. The convergence is faster because a more stable and already updated value from the past is used along with uncertain future states which may not have been updated yet.

This algorithm can also be specially helpful in the cases where the agent is moving into a new trajectory which has never been explored before. Bootstrapping on the past allows the agent to still have reasonable estimates of the value function.

*Equal contribution

## 2 Related Work

Thodoroff et al. [1] show that using values of past states to regularize can be beneficial in the case of policy evaluation. We extend the ideas to the control setting. Temporal regularization is naturally employed when using temporal-difference learning, where the values of one state are bootstrapped with the values of the states in the past. However, these methods do not explicitly make use of the values of past states. Methods like frame-skipping employed in Mnih et al. [2] can be seen as a very strict way of enforcing temporal regularization because they force the agent to take the same action for a fixed number of consecutive frames $k$, which is set as a hyperparameter. The agent selects an action only after every $k$ frames. Braylan et al. [3] study the effects of frame-skip on atari games and show that a well chosen value for frame-skip leads to performance gains. Temporal regularization is similar in the sense that it also aims to enforce consistency in action selection, but it is a much more flexible and powerful method in the sense that a temporally coherent policy is learned only when required and not at the expense of episodic return, as could be the case in strictly enforcing temporal coherence through frame-skip. Also, frame-skip can be applied only to atari environments whereas temporal regularization is more generally applicable.

## 3 Technical Background

In this section, we introduce the concepts leading to our formulation of temporal regularization for the control case.

### 3.1 Temporal Regularization for Policy Evaluation

Thodoroff et al. [1] apply temporal regularization in the policy evaluation case and analyze the bias introduced.

#### 3.1.1 Markov Chains

A discrete-time Markov chain [4] is defined with: a discrete set of states $S$ and a transition matrix $P : S \times S \longrightarrow [0, 1]$ where each element $P_{ij} = P(j|i)$ i.e. the probability of moving from state $i$ to state $j$. Assume that the markov chain is ergodic with stationary distribution $\mu$. Then, if $P$ is irreducible, the chain satisfies detailed balance [5] if and only if

$$\mu_i P_{ij} = \mu_j P_{ji} \quad \forall\, i, j \in S \tag{1}$$

If the chain satisfies detailed balance, it is said to be reversible. A reversal markov chain $\tilde{P}$ of $P$ can defined as:

$$\tilde{P_{ij}} = \frac{\mu_i P_{ij}}{\mu_j} \tag{2}$$

$\tilde{P}$ can be seen as a markov chain with time running backwards and when $P$ is reversible, $\tilde{P} = P$.

#### 3.1.2 Discounted Markov Decision Process

A discounted MDP [6] is characterized by $S, A, R, P, \gamma$ where $S$ is the set of states, $A$ is the set of actions, $R$ is the reward function, $P : S \times A \times S \longrightarrow [0, 1]$ is the transition function and $\gamma$ is the discount factor. In the policy evaluation setting, the $P$ function is fixed since the policy does not change. The Bellman operator $T^\pi$ can be applied to the vector of value estimates $V$ to find the fixed point as follows:

$$T^\pi V = r + \gamma P^\pi V \tag{3}$$

#### 3.1.3 Temporal Regularization Formulation for Policy Evaluation

In matrix form, temporal regularization in policy evaluation is applied as follows:

$$T^\pi_\beta V_\beta = r + \gamma((1 - \beta)P^\pi V_\beta + \beta \tilde{P^\pi} V_\beta) \tag{4}$$

The operator $T^\pi_\beta$ is a contraction mapping and has a unique fixed point $V^\pi_\beta$, the proof of which can be found in [1]. Let $V^\pi$ be the fixed point from equation 3 and let $V^\pi_\beta$ be the fixed point from equation 4.

Then, $V^\pi = \sum_{i=0}^{\infty} \gamma^i P^i r$ and $V_\beta^\pi = \sum_{i=0}^{\infty} \gamma^i ((1-\beta)P + \beta\tilde{P})^i r$ and thus,

$$||V^\pi - V_\beta^\pi||_\infty = ||\sum_{i=0}^{\infty} \gamma^i (P^i - ((1-\beta)P + \beta\tilde{P})^i) r||_\infty \leq \sum_{i=0}^{\infty} \gamma^i ||(P^i - ((1-\beta)P + \beta\tilde{P})^i)||_\infty \quad (5)$$

The quantity on the right hand side of inequality 5 is bounded for $\gamma < 1$. Also, $P$ and $(1-\beta)P + \beta\tilde{P}$ have the same stationary distribution $\mu$, the proof of which can be found in [1]. This means that $||(P^i - ((1-\beta)P + \beta\tilde{P})^i)||_\infty$ converges to 0 in the limit, which ensures a stronger guarantee. When $P = \tilde{P}$, both of the fixed points are the same.

### 3.2 Temporal Regularization for Control

Usually, TD methods bootstrap from only values of future states which might be noisy in the initial phase of learning the policy. On the other hand, Temporal regularization methods take into account the value of future states as well as previous states as the target for performing updates. Learning from recently updated past states helps in making more informed decisions which leads to faster convergence of temporally regularized methods. Intuitively, temporal regularization in control case encourages the agent to perform similar actions in temporally closer states along a trajectory. This is achieved by regularizing $q(s_t, a_t = a)$ with $q(s_{t-1}, a_{t-1} = a)$ as part of its target as shown in step 8 of Algorithm 1. This convex combination of future state values and past state values is parameterized by $\beta$: the higher the value of $\beta$, the more weight we give to the Q-values of past states when regularizing. Now, this past state q-values could either be value of only the immediately previous state or previous n states or an exponential averaging [8] of all the previous states along a trajectory. In the case where $\beta = 0$, we have the conventional TD target and in the case where $\lambda = 0$, we regularize with only the immediately previous q-value from the past.

In the case of policy gradient methods, the formulation is different. We augment the objective function, which the agent tries to maximize, with a regularization term to penalize policy changes between consecutive states in the trajectory. The policy change is quantified by a distance measure between policy at consecutive time steps. This distance measure could be L2 distance in the discrete action case and KL-divergence in the continuous action space. This regularization term is parameterized by $\lambda$ which dictates how much importance is given to policy coherence between consecutive states.

#### 3.2.1 Temporal Regularization for SARSA

In order to regularize with past values from the very beginning of the trajectory, a vector $p$ is used to store all of the q-values which could be needed to potentially regularize with. The vector has the same size as the action space. This vector is updated after every time step. When updating $q(s_t, a_t)$, only the relevant element indexed by action $a_t$ from vector $p$ is required for the update. After updating the q-value, the element in $p$ indexed by $a_t$ is updated with a convex combination of the current value and the q-value that was just updated. The other values in $p$ are decayed by $\lambda$. This can be seen in lines 8 and 9 in Algorithm 1.

#### 3.2.2 Temporal Regularization for Q-learning

Since we require only one element of the $p$ vector whenever an update is being performed, only that element needs to be pushed into the replay buffer when implementing q-learning with a deep q-network. Thus, the added space complexity is negligible. This is similar also in the case of double q-learning where two networks $Q_1$ and $Q_2$ are used in every time step: one to find the greedy next action and the other to find the q-value of the next state using this greedy next action. The network which was used to select the greedy next action is updated. Here also a single $p$ vector is used, and is updated similarly in every time step regardless of which network between $Q_1$ and $Q_2$ is chosen to be updated. Algorithm 2 and Algorithm 3 show temporal regularization in q-learning and double q-learning respectively.

#### 3.2.3 Temporal Regularization for Policy Gradient Methods

Temporal coherence is enforced by adding an additional cost term that penalizes the change in policy along the trajectory. The pseudo code in Algorithm 4 shows how this can be implemented on top of a

**Algorithm 1** SARSA with Temporal Regularization

Input: $\alpha, \gamma, \beta, \lambda$
Initialize $\mathbf{Q_1}, \mathbf{p} = \mathbf{Q_1}$
Choose $a_t \sim \epsilon$-greedy
**for all** steps **do**
  Take action $a_t$, observe reward $r(s_t, a_t)$ and next state $s_{t+1}$
  Choose $a_{t+1} \sim \epsilon$-greedy
  $Q_{sarsa} = Q(s_{t+1}, a_{t+1})$
  $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r(s_t, a_t) + \gamma((1-\beta)Q_{sarsa} + \beta\mathbf{p}(a_t)) - Q(s_t, a_t))$
  $\mathbf{p} = (1-\lambda)\mathbf{Q_t} + \lambda\mathbf{p}$
  $s_t = s_{t+1}, a_t = a_{t+1}$
**end for**

**Algorithm 2** Q-learning with Temporal Regularization

Input: $\alpha, \gamma, \beta, \lambda$
Initialize $\mathbf{Q_1}, \mathbf{p} = \mathbf{Q_1}$
**for all** steps **do**
  Choose $a_t \sim \epsilon$-greedy
  Take action $a_t$, observe reward $r(s_t, a_t)$ and next state $s_{t+1}$
  $Q_{max} = max_a Q(s_{t+1}, a)$
  $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r(s_t, a_t) + \gamma((1-\beta)Q_{max} + \beta\mathbf{p}(a_t)) - Q(s_t, a_t))$
  $\mathbf{p} = (1-\lambda)\mathbf{Q_t} + \lambda\mathbf{p}$
  $s_t = s_{t+1}$
**end for**

vanilla A2C method (a variation of the vanilla A3C [7]). More specifics on the exact parameters used have been specified in the experimental section. We used Pytorch for the implementation, and hence did not have to calculate the gradient of the objective function ourselves.

**Algorithm 4** Policy Gradient with Policy Smoothing

Input: $\pi_\theta, \alpha, \gamma, \lambda$
**for all** steps **do**
  Generate an *episode* $s_0, a_0, r_1, \ldots s_{t-1}, a_{t-1}, r_t$ following $\pi_\theta$
  $\pi_\theta(a_{-1}|s_{-1}) = \pi_\theta(a_0|s_0)$
  **for all** transition t in episode **do**
    $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
    $\theta = \theta + \alpha\gamma^t \nabla_\theta [G_t \log \pi_\theta(a_t|s_t) - \lambda \mathbf{D}(\pi_\theta(a_t|s_t), \pi_\theta(a_{t-1}|s_{t-1}))]$
  **end for**
**end for**

#### 3.2.4 Bias

Temporal regularization introduces bias because we bootstrap using the q-value of past state-action pairs from the trajectory. In expectation, the updates for SARSA temporal regularization are performed as follows,

$$Q(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t-1} \sim \pi}[r(s_t, a_t) + \gamma((1-\beta)Q(s_{t+1}, a_{t+1}) + \beta Q(s_{t-1}, a_t))] \quad (6)$$

for some action $a_t$ taken at $s_t$. The update equation for vanilla version of temporal difference is given as follows,

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r(s_t, a_t) + \gamma((1-\beta)Q(s_{t+1}, a_{t+1}) + \beta Q(s_{t-1}, a_t)) - Q(s_t, a_t)] \quad (7)$$

**Algorithm 3** Double Q learning with Temporal Regularization

Input: $\alpha, \gamma, \beta, \lambda$
Initialize $\mathbf{Q_1}, \mathbf{Q_2}, \mathbf{p} = \mathbf{Q_1}$
**for all** steps **do**
  Choose $a_t \sim \epsilon$-greedy$(\mathbf{Q_1} + \mathbf{Q_2})$
  Take action $a_t$, observe reward $r(s_t, a_t)$ and next state $s_{t+1}$
  With 0.5 probability:
    $Q_{target} = r(s_t, a_t) + \gamma\big((1-\beta)Q_2(s_{t+1}, \text{argmax}_a\, Q_1(s_{t+1}, a)) + \beta\mathbf{p}(a_t)\big)$
    $Q_1(s_t, a_t) = Q_1(s_t, a_t) + \alpha\big(Q_{target} - Q_1(s_t, a_t)\big)$
  else:
    $Q_{target} = r(s_t, a_t) + \gamma\big((1-\beta)Q_1(s_{t+1}, \text{argmax}_a\, Q_2(s_{t+1}, a)) + \beta\mathbf{p}(a_t)\big)$
    $Q_2(s_t, a_t) = Q_2(s_t, a_t) + \alpha\big(Q_{target} - Q_2(s_t, a_t)\big)$
  $\mathbf{p} = (1-\lambda)\mathbf{Q_t} + \lambda\mathbf{p}$
  $s_t = s_{t+1}$
**end for**

The Bellman operator for the above update with transition probability $P(.|s_t, a_t) \in (0, 1)$ can be written as,

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma((1-\beta)P(s_{t+1}|s_t, a_t)Q(s_{t+1}, a_{t+1}) + \beta P(s_t|s_{t-1}, a_{t-1})Q(s_{t-1}, a_t)) \tag{8}$$

Note that by setting $\beta = 0$ we fall back to the traditional temporal difference method. But for any $\beta > 0$ we would be introducing some bias due to the bootstrapping from the past. This bias could be either helpful or harmful depending upon the MDP. For instance, in an MDP where there are cliffs or sharp changes in rewards the bias could be problematic. In other words, bootstrapping from a previous action value to the current value may result in sub-optimal policies due to uneven reward distribution. However, bootstrapping from the past could be useful in the case of MDPs where the rewards are smooth and has a constant value everywhere as the Q values of the past and the present are close (by a factor of $\gamma$ since rewards are constant and smooth). In the sections further, we demonstrate the effectiveness of bootstrapping from the past. Specifically, we show that speed of convergence is significantly better in an MDP with smooth and constant rewards when we bootstrap from the past.

## 4 Experiments

In this section, we describe the experiments performed to verify the effectiveness of this approach.

### 4.1 Toy Example

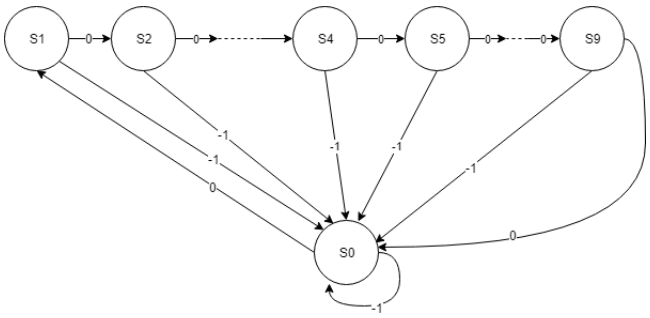


Figure 1: State diagram of the toy example which demonstrates the effectiveness of temporal regularization long the trajectory

This an example that we used to demonstrate the effectiveness of temporal regularization along the trajectory, and more specifically the importance of bootstrapping based on the past.

This is a deterministic environment that has ten states (namely S0, S1, ..., S10) and you have two actions to pick from at each state. The first action takes you along the chain with no reward while the other drops you back to S0 with a negative reward of -1. The optimal action would be to always pick the first action so that you keep moving along the chain without incurring any negative rewards. The full diagram of this environments state transitions with rewards are shown in 1.

We setup initial values of states in special ways, which we believe are equivalent to common scenarios encountered while using function approximation. We show the learning curves against the vanilla version of SARSA on three different configurations to show the effectiveness of this approach. All the experiments have been average over 10 runs. Updates were performed only on states initialized to non-optimal values.

In the first configuration, we initialized all states to their optimal values except states S4 and S5. We believe this is equivalent to a case where the function approximator has bad estimates for specific states. We compare the results of SARSA with and without the temporal regularization. With temporal regularization, we show in Fig 2(a) that the proposed approach converges faster. This happens because temporal regularization bootstraps based on past state value (which are accurate) and future estimates (which are inaccurate), as opposed to vanilla SARSA that bootstraps only based on inaccurate future estimates.

In the second configuration, we initialized more successive states in the trajectory to random values. We believe this is a common case encountered, specially while using function approximation– where some states along the trajectory (in this case four) have bad estimates. Even in this case, we show that in Fig 2 (b) converges faster. This time, the convergence speeds between the algorithms is more prominent. Fig 2 (c) shows when 6 consecutive states are initialized randomly (namely S4 to S9) and this shows the speeds changing even more. This shows the effectiveness of the approach even if all future states along the trajectory have uncertain values.

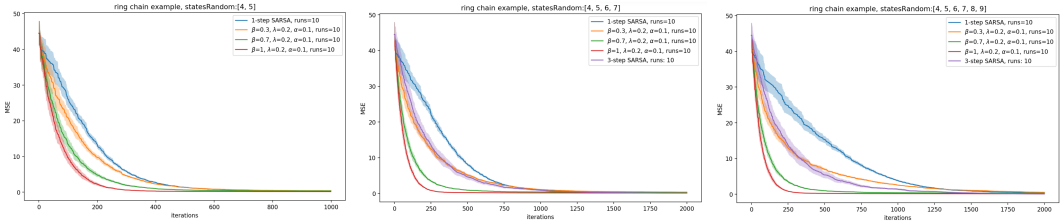


Figure 2: Results on the toy example comparing the convergence rates of various algorithms. Note: the y-axis is the MSE of S4.

### 4.2 Q-learning

The implementation of Q-learning was done in Pytorch. Adam was used an as optimizer with a learning rate of $10^{-3}$ for non-atari environments and $10^{-4}$ for atari games. We ran the experiments for 250,000 frames for non-atari and 1.7 million frames for atari environments. We performed hyper-parameter tuning of $\beta$ and $\lambda$ using grid search on with values $\beta \in \{0, 0.1, 0.2, 0.3, 0.7\}$, $\lambda \in \{0.1, 0.2, 0.3\}$ and $\beta \in \{0, 0.1, 0.2, 0.3\}$, $\lambda \in \{0.2\}$ for non-atari and atari environments respectively which resulted in a total of 15 experiments in the non-atari case and 4 experiments in the atari case per seed. We averaged the mean episodic return of the past 20 episodes over 5 seeds for non-atari as shown in Figure 3. For atari, we could only generate results for one seed due to time and computational constraints as shown in the last row of Figure 3, and we did not plot the action change frequency in this case. We observe that intermediate values of $\beta = \{0.1, 0.2\}$ perform better than the vanilla TD method which is in line with the motivation of getting informed updates due to the inclusion of values of past states in the target. We also observe that too much regularization is bad for learning as the updates would be highly biased towards the values of past states. The intuition behind finding the optimal intermediate beta is similar to finding the best $\lambda$ in TD-$\lambda$ methods. Along with tracking mean episodic return, we also tracked the frequency of action changes for every 2000 frames (non-atari) and averaged over 5 seeds. Since temporal regularization forces similar actions to be taken in temporally closer states, when using high $\beta = 0.7$, the frequency of action change along a trajectory is minimum as shown in Figure 3.

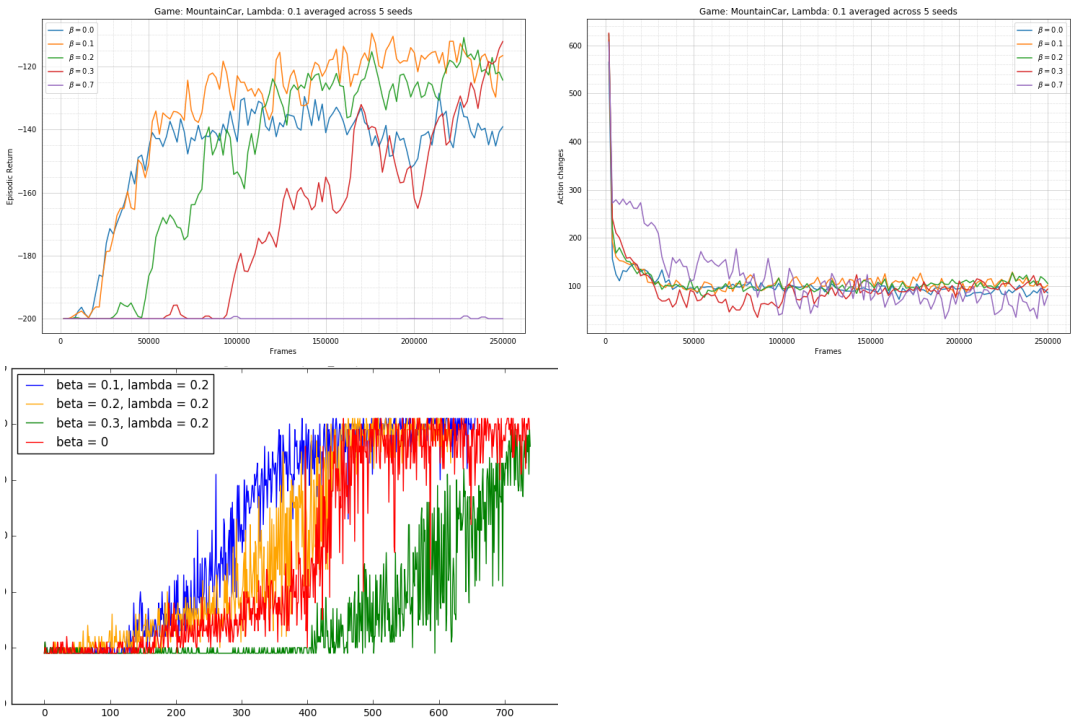


Figure 3: DQN results on Mountain Car (row 1) and Pong (row 2) for different values of $\beta$ and best $\lambda = 0.1$ for row 1 and $\lambda = 0.2$ for row 2. The left image in each row shows the average episodic return and the right image in each row shows the frequency of action changes.

### 4.3 Double Q-learning

For non-atari environments, the hyperparameter tuning and experimental setup for Double DQN [9] was done in exactly the same way as in the DQN experiments. For atari environments, we tried $\beta \in \{0, 0.1, 0.2, 0.3\}$ and $\lambda \in \{0.1\}$. We averaged the mean episodic return of past 20 episodes over 3 seeds for atari as shown in Figure 4. We observe that the clear distinction among different $\beta$ values is much more evident in case of DQN over Double DQN. Since Double DQN already reduces overestimation of Q-values over vanilla Q-learning, we believe that the effect of temporal regularization is not as clearly distinguishable in Double Q-learning as in Q-learning. However, it can be easily seen that an intermediate value of $\beta = 0.1, 0.2$ worked best in both atari and non-atari cases as shown in Figure 4. We also tracked the count of action changes for every 7000 frames for Pong and averaged over 3 seeds. Surprisingly, $\beta = 0$ leads to minimum action changes as per Pong results in Figure 4. We suspect the reason could be that temporal regularization penalizes action changes only when possible but not at the expense of gaining more return.

### 4.4 Policy gradient approach

The algorithm has been implemented on top of A2C and has been tested on the CartPole-v1 environment. RMSprop was used as the optimizer for the neural network with a learning rate of $7 * 10^{-4}$, RMSprop epsilon of $10^{-5}$, RMSprop alpha of $0.99$. The algorithm has been tested for $\lambda \in [0, 0.01, 0.05]$. Here, $\lambda = 0$ represents the vanilla case. The results of how the mean rewards change while training and how often the agent changes it's actions are shown in 5. These results are averaged over five runs of 3 million timesteps each.

It can be observed that while the proposed method doesn't perform much better than vanilla A2C, it is interesting to see that we have no compromise in performance while strongly regularizing the actions (see 5(a)). The formulation needs to be tested more extensively on more environments for stronger evidence on the effectiveness of this approach in the policy gradient setting. We plan to do this in future work.


Figure 4: Double-DQN results on Mountain Car (row 1) and Pong (row 2) for different values of $\beta$ and best $\lambda$. The left image in each row shows the average episodic return and the right image in each row shows the frequency of action changes.

It can be seen in 5(b) that the frequency of action change is much more for the vanilla A2C approach than the models which include regularization. This is consistent with our understanding of the algorithm, that when we penalize changing the policy too often, it is reflected in the agent taking actions that don't change very often.

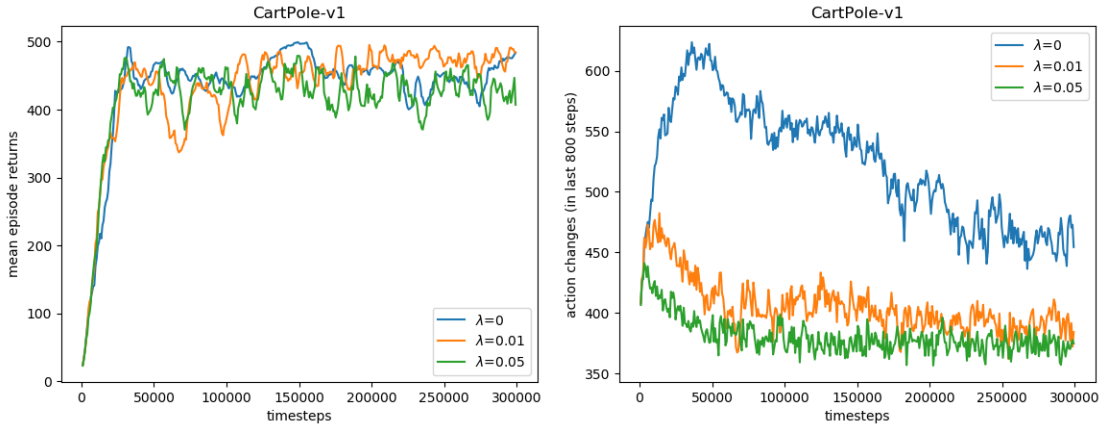


Figure 5: Results of doing temporal regularization in vanilla A2C method. (a) shows the mean episode reward while it's training, (b) shows number of times actions have been changed in the last 800 steps

## 5 Discussions and Conclusion

In this report, we show empirically the benefits of temporal regularization along the trajectory. We have proposed formulations on how this can be implemented in both the q value case and the policy regularization case. In case of q values, we implemented the formulation on tabular SARSA, Q learning and double Q learning and tested them on specific examples and presented the results. The results show that temporal regularization leads to much faster convergence to the optimal value. In the policy regularization case, we show the results on the CartPole environment that we can regularize the actions significantly without sacrificing the performance. We believe that this approach has good potential for reducing the high variance problem in reinforcement learning. This approach can be improved further by decreasing the value of beta using a linear or exponential schedule. This idea has not been explored in this project, but may be included in future work.

The code has been made available at `https://github.com/barleen-kaur/temporal_regularization_control`. Each of the implementations are in a different branch in the repository.

## 6 Future Work

As part of future work, we plan to run DQN, Double DQN on more atari environments to test the effectiveness of the algorithm more extensively. Testing on more complex environments like atari could lead to more noticeable performance improvements. We would like to explore the case of policy gradient further too by looking into why the present formulation of temporal regularization doesn't give a notable improvement over baselines yet. We want to run the experiments on environments like mujoco because we believe this approach might be a much better fit there.

We also plan to theoretically prove convergence of our proposed formulation and show bounds for bias. We have shown the effectiveness of policy regularization in the tabular case with a toy example. We additionally want to make a more convincing case by showing results on an example that uses linear function approximator.

#### Acknowledgments

We are thankful to Nishanth V Anand for his constant guidance and support.

## References

[1] Pierre Thodoroff, Audrey Durand, Joelle Pineau, Doina Precup *Temporal Regularization in Markov Decision Process*. Montréal, Canada, NeurIPS 2018

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Daan Wierstra, Alex Graves, Ioannis Antonoglou, Martin Riedmiller *Playing Atari with Deep Reinforcement Learning*. NIPS Deep Learning Workshop, 2013

[3] Alex Braylan, Mark Hollenbeck, Elliot Meyerson, Risto Miikkulainen *Frame Skip Is a Powerful Parameter for Learning to Play Atari*. Learning for General Competency in Video Games, AAAI Workshop, 2015

[4] D. A. Levin and Y. Peres *Markov chains and mixing times*. American Mathematical Soc., 2008

[5] J. G. Kemeny and J. L. Snell. *Finite markov chains*. undergraduate texts in mathematics. 1976.

[6] M. L. Puterman *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu *Asynchronous Methods for Deep Reinforcement Learning*. CoRR 2016

[8] E. S. Gardner *Exponential smoothing: The state of the art—part ii*. International journal of forecasting, 22(4):637–666, 2006

[9] E. S. Gardner *Deep Reinforcement Learning with Double Q-learning*. Hado van Hasselt, Arthur Guez, David Silver CoRR 2016

# 7 Appendix

## 7.1 DQN results on other environments

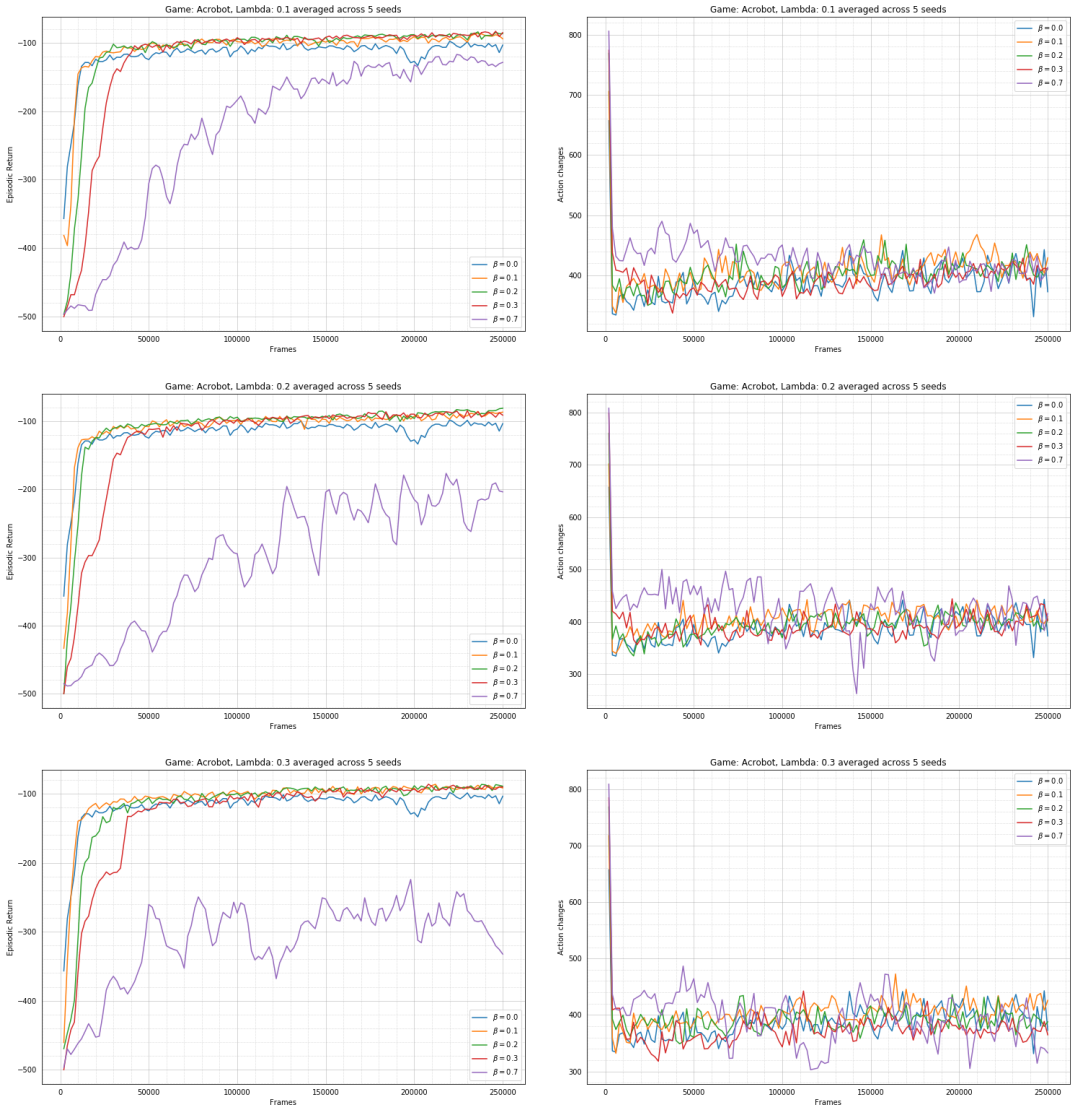


Figure 7: DQN results on Acrobot for different values of $\beta$ and $\lambda$. The left image in each row shows the average episodic return and the right image in each row shows the frequency of action changes.


Figure 6: DQN results on LunarLander for different values of $\beta$ and $\lambda$. The left image in each row shows the average episodic return and the right image in each row shows the frequency of action changes.

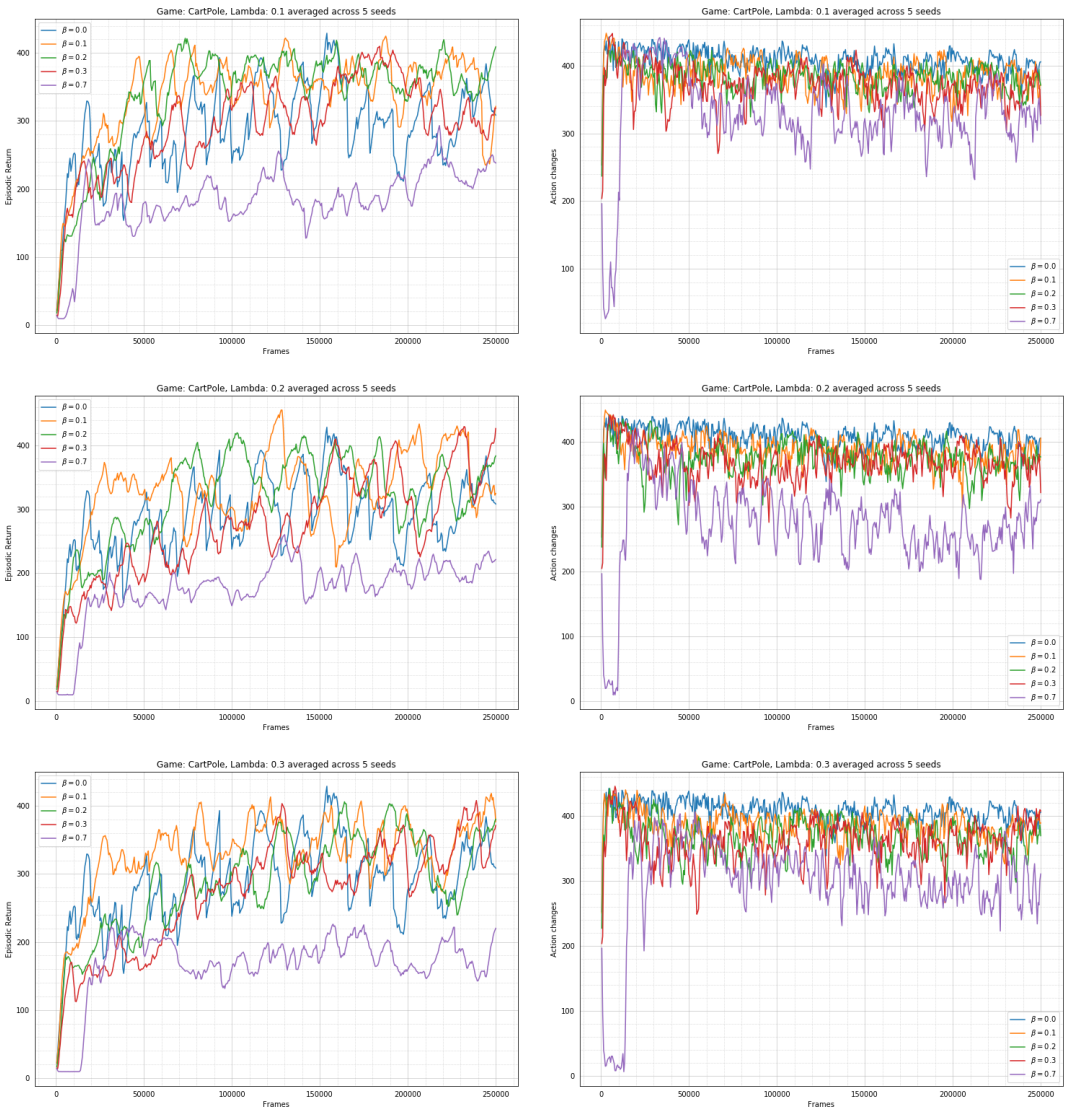


Figure 8: DQN results on CartPole for different values of $\beta$ and $\lambda$. The left image in each row shows the average episodic return and the right image in each row shows the frequency of action changes.

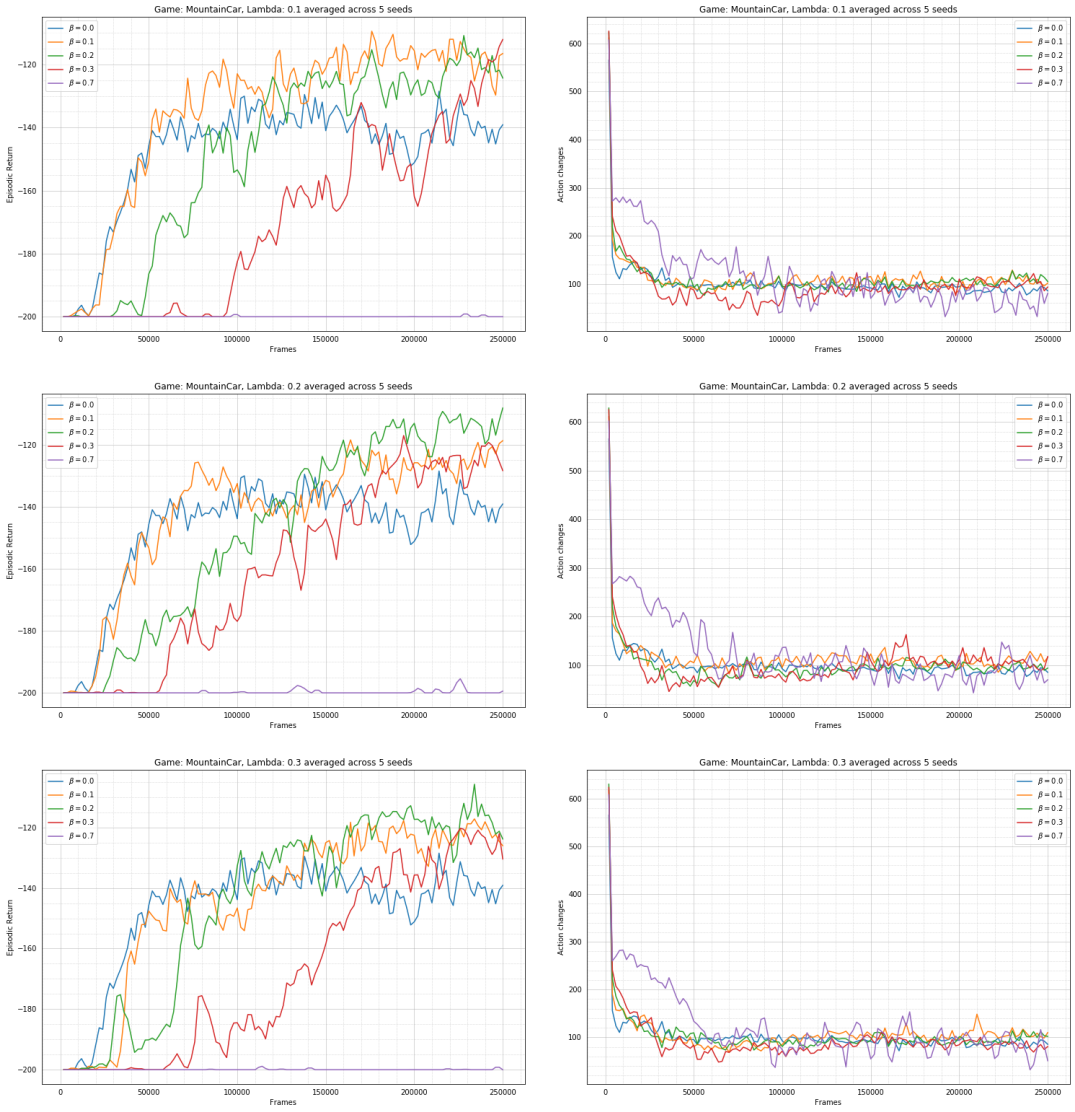


Figure 9: DQN results on Mountain Car for different values of $\beta$ and $\lambda$. The left image in each row shows the average episodic return and the right image in each row shows the frequency of action changes.

## 7.2 Double DQN results on other environments

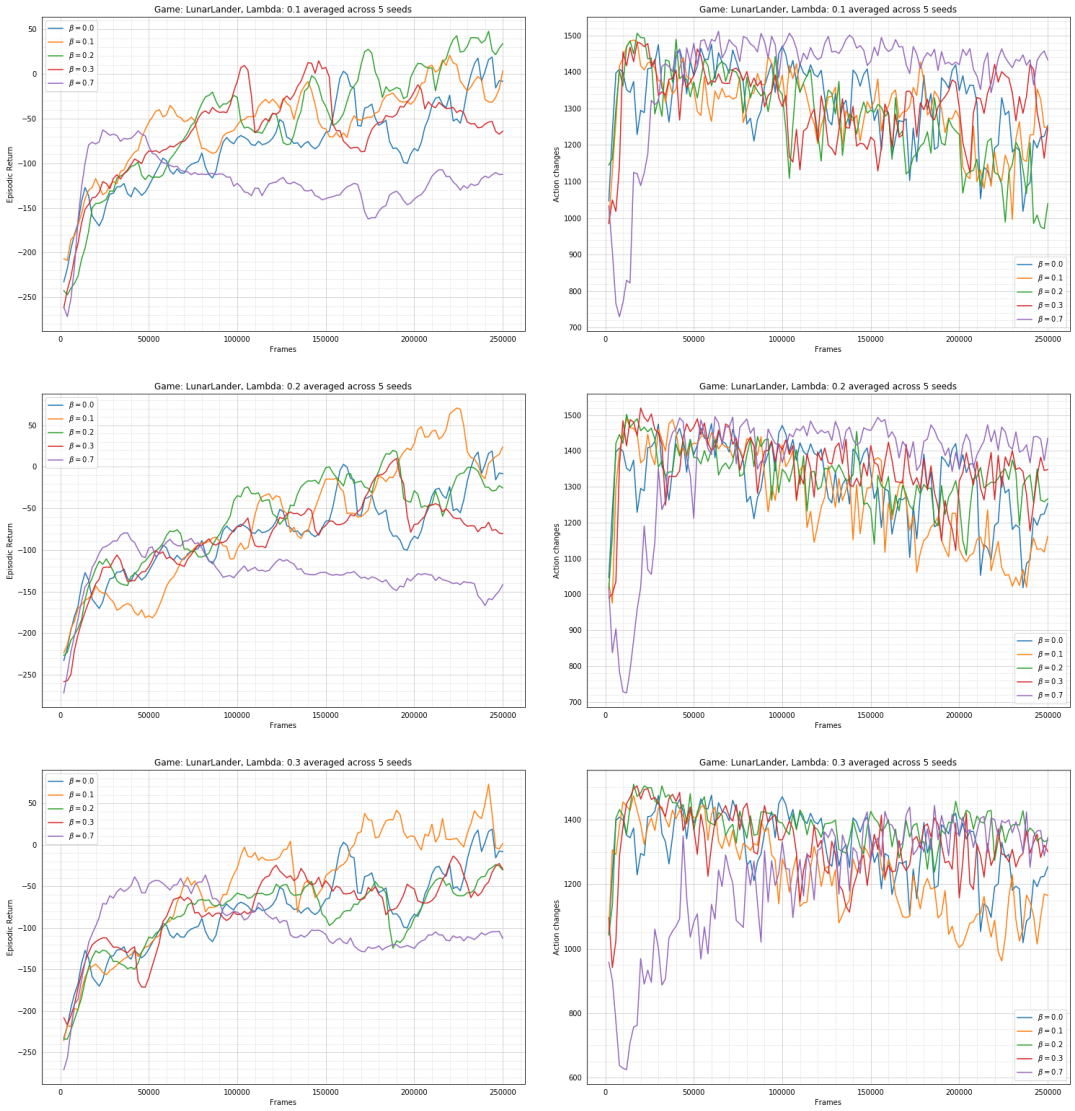


Figure 10: Double DQN results on LunarLander for different values of $\beta$ and $\lambda$. The left image in each row shows the average episodic return and the right image in each row shows the frequency of action changes.

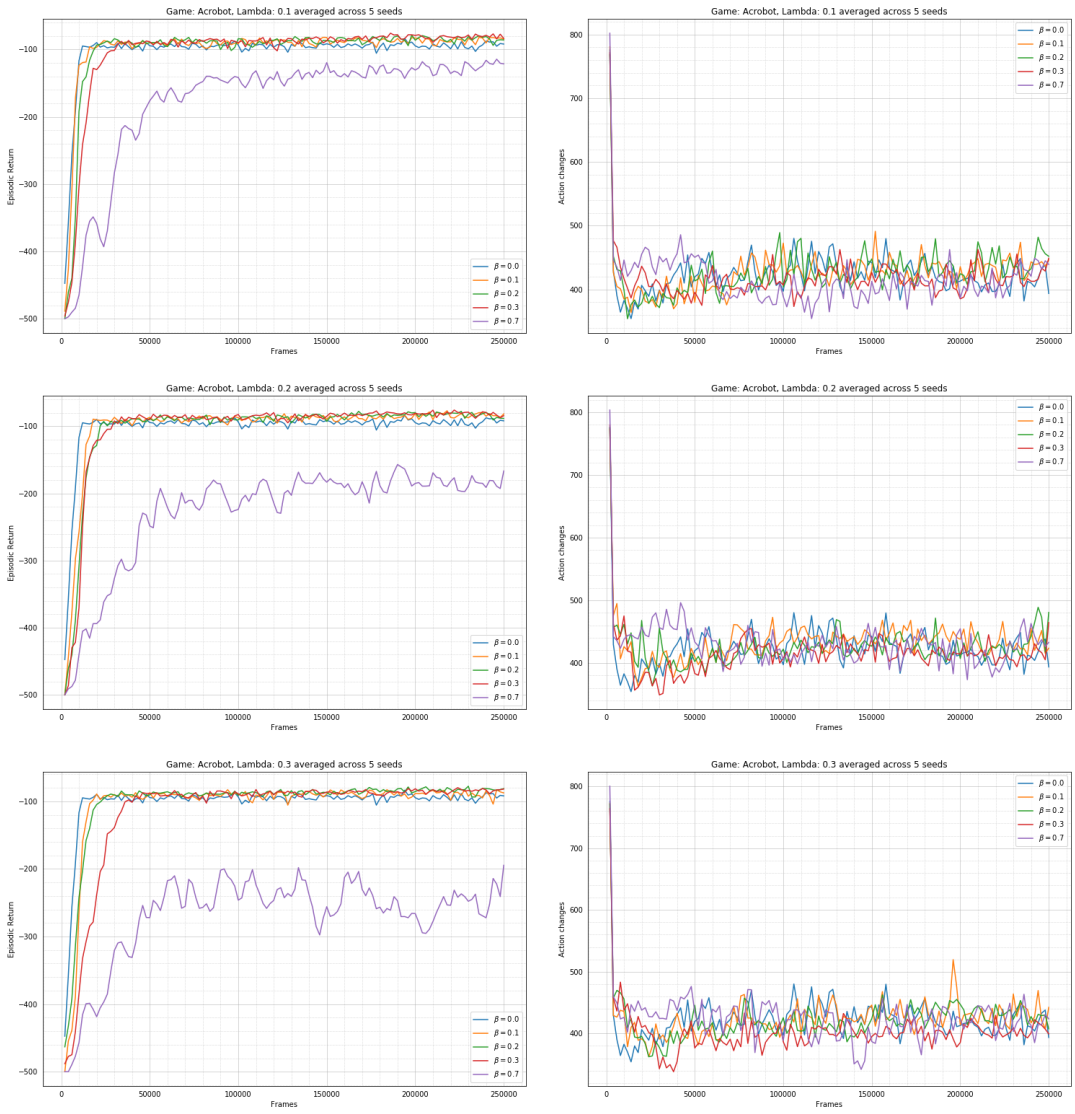


Figure 11: Double DQN results on Acrobot for different values of $\beta$ and $\lambda$. The left image in each row shows the average episodic return and the right image in each row shows the frequency of action changes.

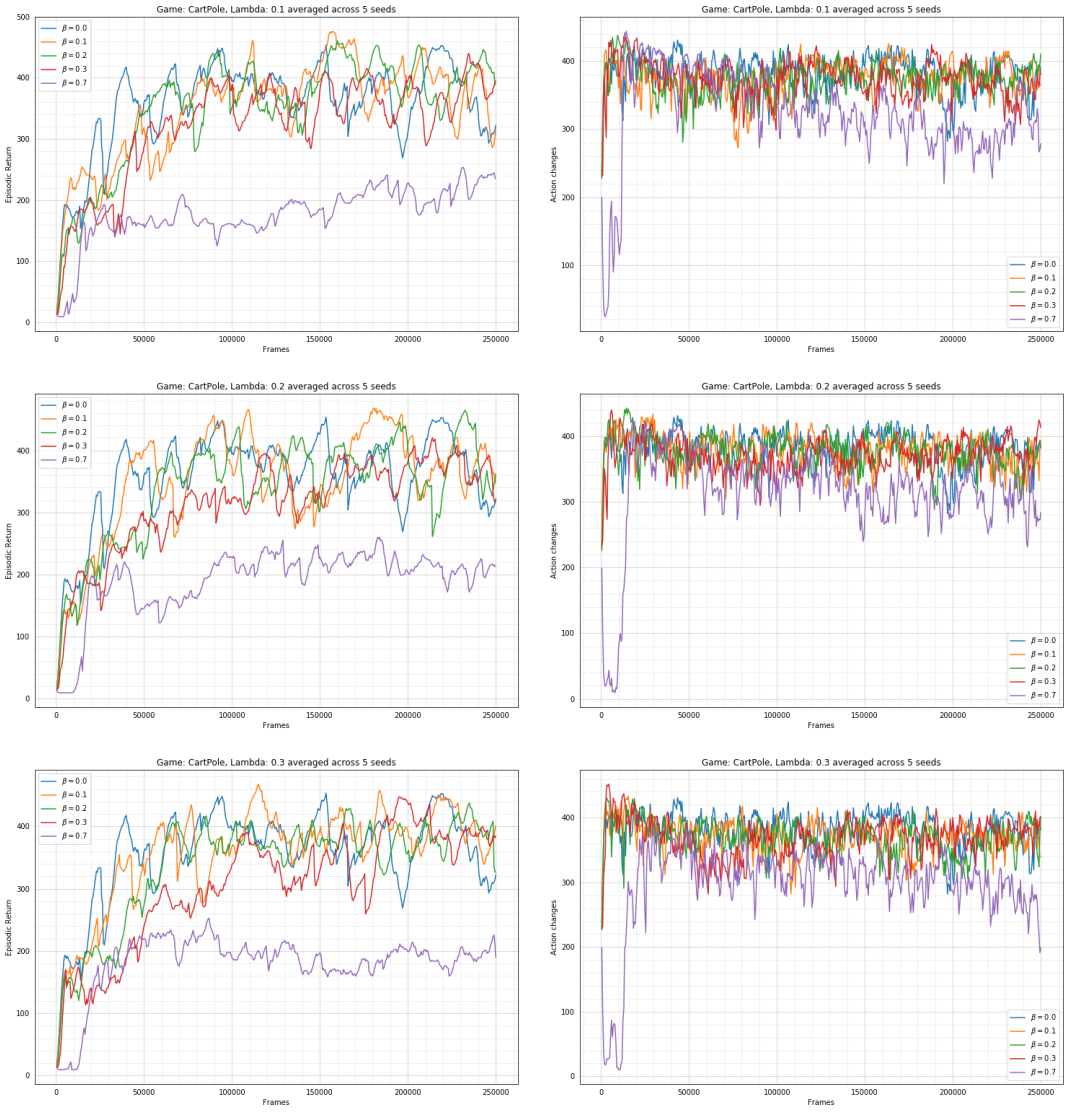


Figure 11: Double DQN results on CartPole for different values of $\beta$ and $\lambda$. The left image in each row shows the average episodic return and the right image in each row shows the frequency of action changes.