# *Cart-Pole Swing-Up*

Anna Derzaev Naomi Deutsch Sarel Duanis Roni Zelenchuk

***Abstract*—This project proposes a strategy to learn a nonlinear control problem - to swing up an inverted pendulum balance it around the upright position. We successfully learn a controller for swinging and balancing in a simulation environment using Q-Learning with an exploration/exploitation policy, without any prior knowledge of the system at hand. We do however fail to learn a controller using Q-Learning with a linear function approximator and using deep Q-Learning.**

## I. Introduction

The Cart-Pole system is a classic benchmark for nonlinear control. It is an inherently unstable and underactuated mechanical system. The dynamics of this system is used to understand tasks involving the maintenance of balance, such as walking, control of rocket thrusters and self-balancing mechanical systems.

The system consists of a pole, which acts as an inverted pendulum, attached to a cart. The force applied to the cart can be controlled, and the goal is to swing the pole up and balance it around the upward position.

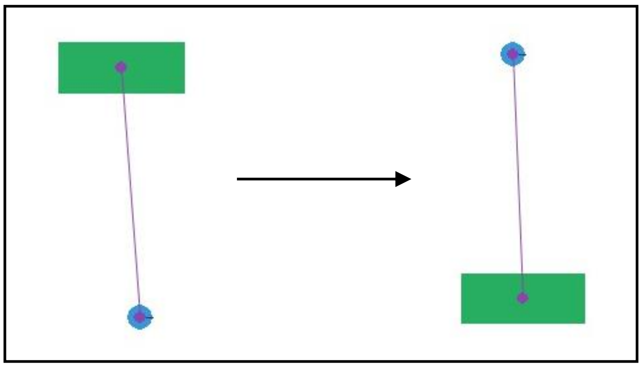


Figure 1: The objective of the project.

Our algorithms are applied and tested on a python simulation that was implemented by us. This simulation obeys the rules of physics and thus represents the problem as it is in the real world.

The goal of this paper is to solve the cart-pole swing-up problem using different algorithms in order to achieve the best outcome.

## II. Approaches

All our algorithms are based on reinforcement learning.

The state space consists of $(\theta, \dot{\theta})$ pairs that describe the angle of the pendulum and its angular velocity. Each state is independent of the others. In addition, we have defined three actions: left, right and no action. If the pole is downward at some angle range, the reward is -1. If the pole is upward in some angle range, the reward is 5. Else, the reward is -0.1 to encourage the algorithm to find the best solution. In order to prevent the pole from spinning uncontrollably at high speed and thus accumulate positive rewards infinite number of times, we subtract a value that is proportional to the pole's absolute horizontal velocity. This encourages the pole to stabilize within the angle range at which there is a positive reward while minimizing its horizontal velocity.

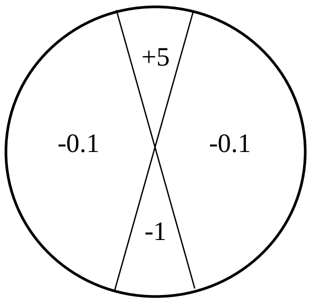


Figure 2. The reward as a function of the angle range.

It should be noted that the state space is very large. However, it is not continuous. Therefore, it is interesting to test whether this aspect has any impact on the results. Hence, we will propose two different approaches towards a solution: (1) Standard Q-Learning algorithms with a discrete state space, (2) Deep Q-Learning[1] and Linear Q-Learning[2] with a continuous state space.

In addition, we present two different approaches to the exploration vs. exploitation dilemma: (1) ε–Greedy, (2) SoftMax.

Finally, we define two different system settings: (1) limited movement space - we limit the cart movement by placing a wall on each side of the cart, (2) unlimited space - the cart can move along an infinite horizontal axis. This is a simpler problem than the first one since it has less predefined limitations.

## III. Methods

### A. *Q-Learning*

The Q(state, action) function is represented by a table of (s, a) pairs against Q-values. A state is represented by $s = (\theta, \dot{\theta})$ and an action is represented by $a \in \{-f, 0, f\}$, where f is the force applied to the cart.

The discretization of the state space is done by rounding off $\theta$ and $\dot{\theta}$ to one decimal place. That is, $\theta \in [0.0, 0.1, ..., 6.1, 2\pi]$ is limited to only 62 values. However, $\dot{\theta} \in [0.0, 0.1, ...]$ is theoretically infinite. But, above some angular velocity the Q-value is significantly low due to the subtraction of the pole's horizontal velocity from the reward. Hence, states with high angular velocities are rarely visited.

### B. *Exploration vs. Exploitation*

- ε–Greedy: the amount of exploration is globally controlled by a parameter, ε, that determines the randomness in action selection. One advantage of the ε–Greedy is the fact that no memorization of exploration specific data is required, which makes the method particularly interesting for very large state spaces. At each time step, the agent selects a random action with a fixed probability,

$0 \leq \varepsilon \leq 1$, instead of selecting greedily one of the learned optimal actions with respect to the Q-function.

- SoftMax: In contrast to ε–Greedy, SoftMax bias exploration towards promising actions. The action selection methods grade action probabilities by estimated values. We used the Gibbs distribution as our SoftMax function over the Q-values of the actions:

$$\pi(a|s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{e^{\sum_{a' \in \mathcal{A}} \frac{Q(s,a')}{\tau}}}$$

### C. *Linear Q-Learning*

In this case the state space is continuous. So, we cannot implement a regular Q-Learning algorithm. Instead, we use a linear function approximation. That is, we approximate the true action-value function $q_\pi$ by a linear combination of n state-action features:

$$q_\pi(s,a) \approx \sum_{j=1}^{n} w_j x_j(s,a) \;=\; x(s,a)^T w = q'(s,\ a,\ w)$$

where $x(s, a) \in R_n$ is the feature vector. Given features x, our goal is to choose the parameter $w \in R_n$ such that the mean-squared error between q' and $q_\pi$ is minimized. Doing so using stochastic gradient descent yields the update rule:

$$\Delta w \;=\; \alpha(r` + \gamma max_{a`} q'(s`,\ a`,\ w) \;- q'(s,\ a,\ w))x(s,a)$$
$$w \longleftarrow w + \Delta w$$

where r` is the immediate reward associated with taking action a from state s, and s` is the successor state.

Let $s = [\theta_0, \theta_1, \dot{\theta}_0, \dot{\theta}_1]$, where $\theta_0$ is the horizontal position of the cart and $\theta_1$ is the angle of the pole. We define the future vector $x = [f_1, f_2, f_3, f_4]$, where normalAngle(φ) is the difference between φ and its closest multiple of 2π.

| | |
|---|---|
| $f_1$ = normalAngle($\theta_1$) | $f_2 = \dot{\theta}_1$ |
| $f_3 = a \cdot \theta_1$ | $f_4 = a \cdot \dot{\theta}_1$ |

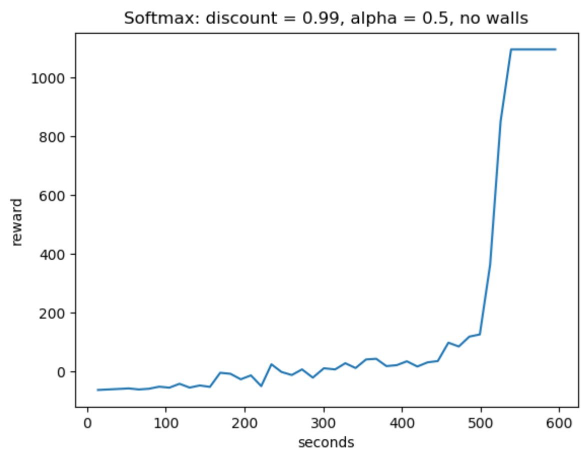
### D. *Deep Q-Learning*

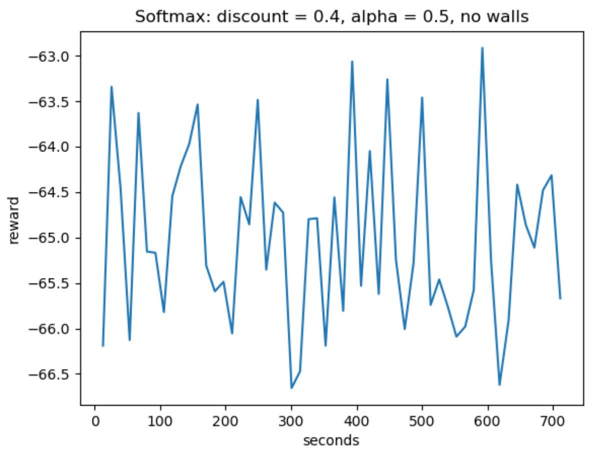
We implemented the neural network that is described in paper [1]. The paper proposes a neural network-based reinforcement learning controller that is able to learn control policies in a highly data efficient manner. This allows to apply reinforcement learning directly to real plants; neither a transition model nor a simulation model of the plant is needed for training. The only training information provided to the controller are transition experiences collected from interactions with the real plant. By storing these transition experiences explicitly, they can be reconsidered for updating the neural Q-function in every training step. This results in a stable learning process of a neural Q-value function.

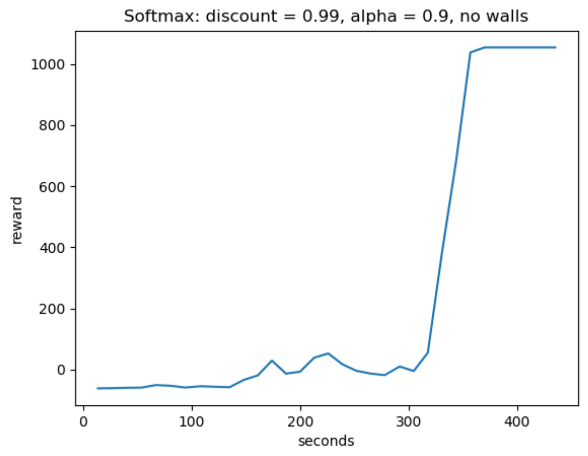
## IV. Results

### A. *Q-Learning*

For best results, we examined different values of the discount factor γ, learning rate α and ε. We measured the quality of the parameters based on the "learning process" - the reward as a function of time. Every 100 episodes, all the rewards over these episodes were summed and divided by 100.

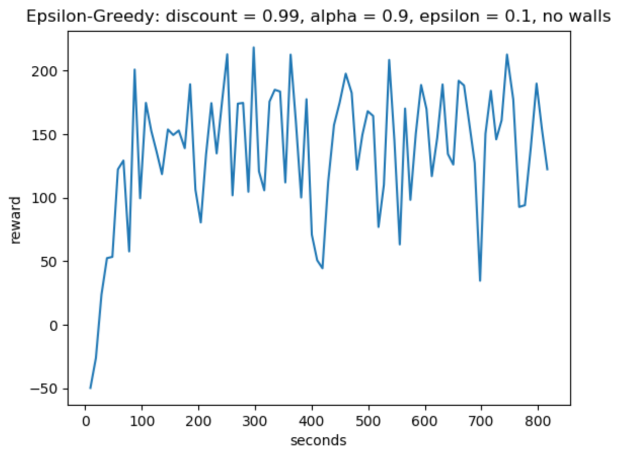


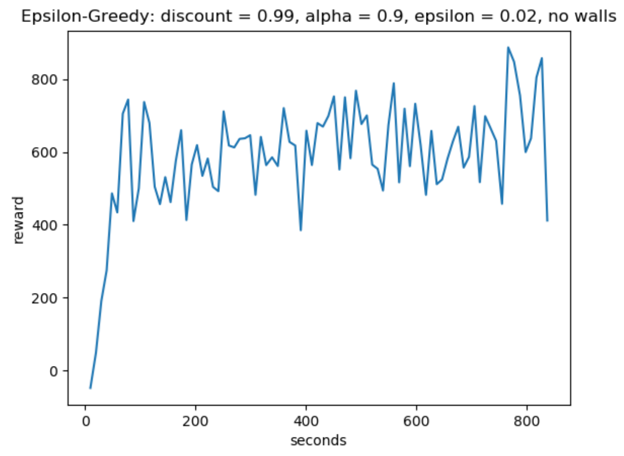


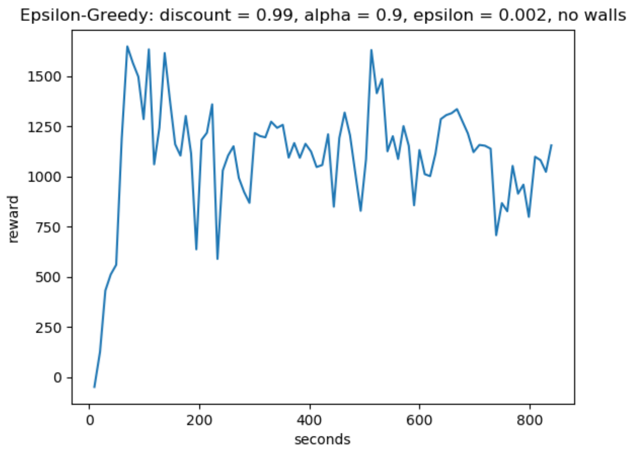


The discount factor determines the importance of future rewards. A factor of 0 will make the agent "myopic" (or short-sighted) by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. We notice that the learning process is much better with $\gamma \approx 1$ since we are more interested in future rewards, i.e. when the pole is upward, than in rewards where the position of the pole is near the starting position.

The learning rate determines to what extent newly acquired information overrides old information. A factor of 0 makes the agent learn nothing (exclusively exploiting prior knowledge), while a factor of 1 makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities). We notice that the learning process is much better with $\alpha = 0.9$, since our main concern is to keep the pole stable once it is upward, but we also want to swing it up. That is, we are more interested in recent rewards when the pole is near the goal state, but we need to keep learning how to properly swing the pole.







As for $\varepsilon$, we notice that higher values means that the agent explores with higher probability, even when the reward is good enough or even the best one. So, the rewards we get are accordingly - higher rewards are accomplished with little exploration and much exploitation. Once the best action is found, we want to persist that action, so we need to 'simulate' less exploration.

After choosing the best values for the parameters, we created a graphical representation of the maximum Q-value for each state. We call this representation a heatmap. The heatmap was created in order to comfortably track the Q-table.

At the beginning of the training session most states' Q-values are close to zero and the negative values are around the initial state ($-\pi/2$, 0), as shown in figure 3.1. Later, higher values are placed around ($\pi/2$, 0), the goal state, and lower values around ($-\pi/2$, 0), as shown in figure 3.2. At the end of the training session there are high values around the goal state and a trail of states connects it to the initial state, as shown in figure 3.3. This indicates that the agent has learned how to swing up the pole and stabilize it through these specific states.

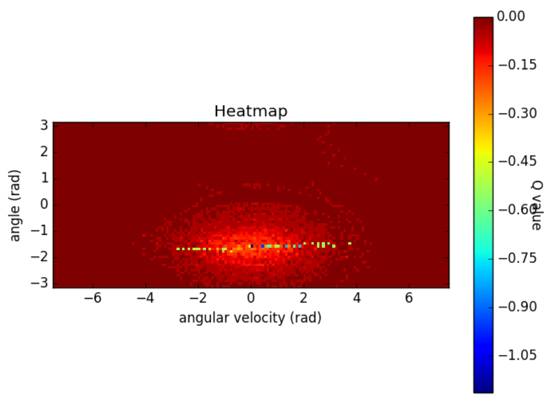


Figure 3.1: Heatmap at the beginning of the training.

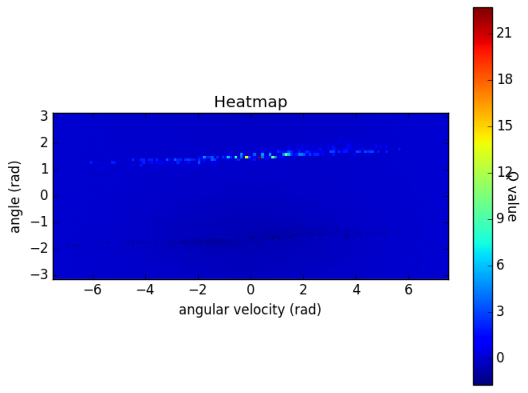


Figure 3.2: Heatmap while the training.

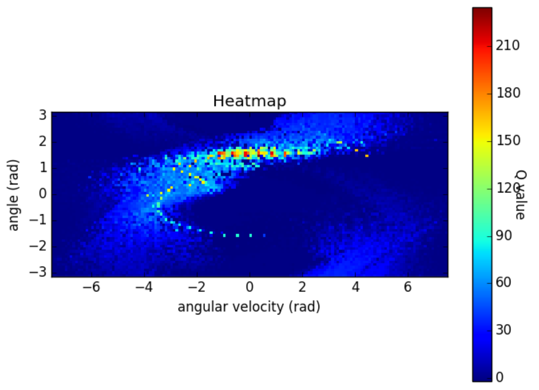
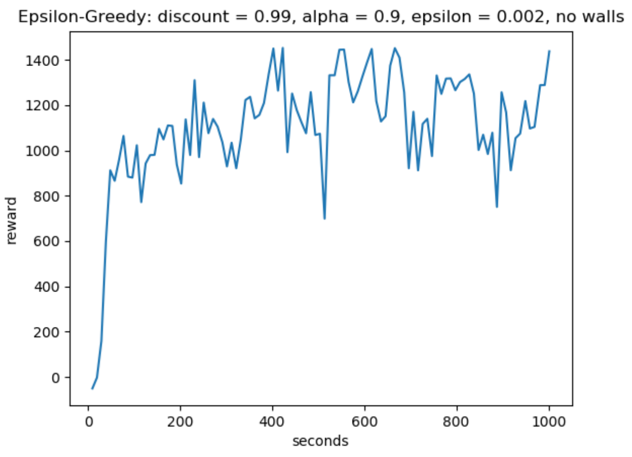
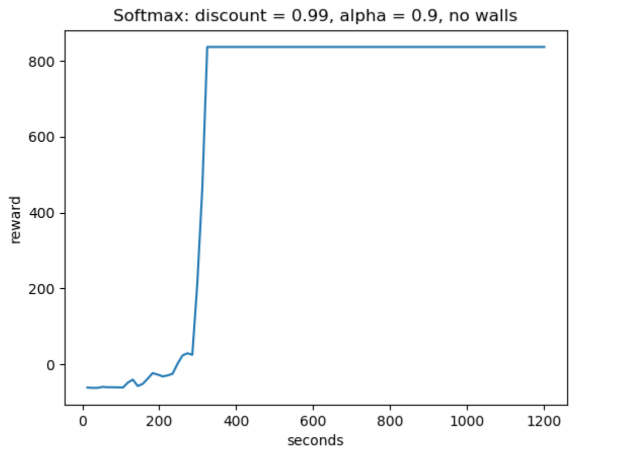


Figure 3.3: Heatmap at the end of the training.

### B. ε-Greedy vs. SoftMax

The following graphs describe the reward as a function of time of both algorithm with unbounded space:



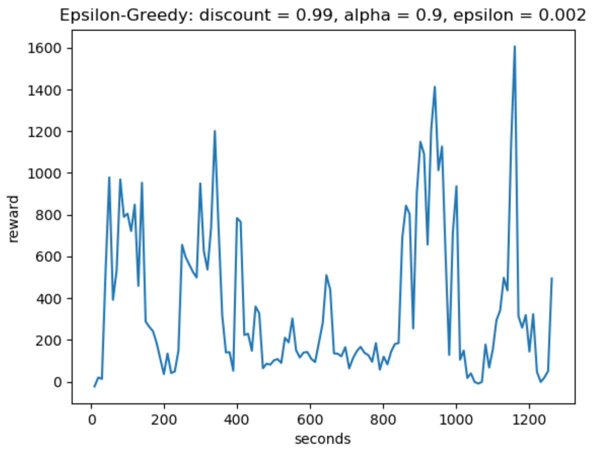


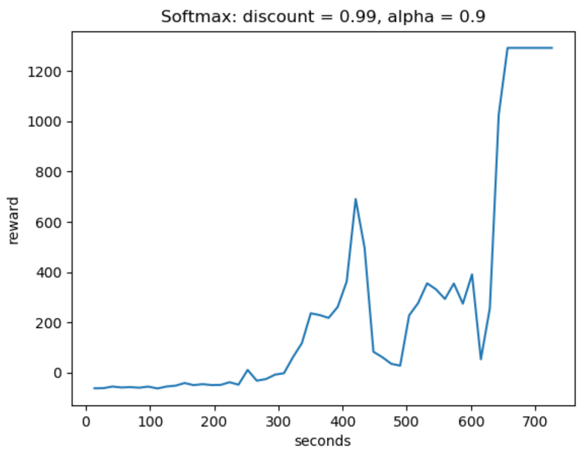
In addition, we measured how long it took for each model to swing the pole up and how long was the pole stable (an average of 5 tries).

- ε-Greedy: swing up took 2.848 seconds and keeping the pole stable lasted 18.992 seconds.
- SoftMax: swing up took 2.426 seconds and keeping the pole stable lasted 18.924 seconds.

It is clear that ε-Greedy reaches higher rewards in a shorter amount of time. It is also clear that unlike ε-Greedy, SoftMax stops exploring at some point and is therefore converges to a fixed reward. As for the performance of both algorithms, SoftMax is slightly better at the swing up. After the training process, it always chooses the best actions that maximizes the reward, i.e. leads to the goal state, as opposed to ε-Greedy that keeps exploring even when the agent already knows the best actions that lead to the goal state.

The following graphs describe the reward as a function of time of both algorithm with bounded space, i.e. with a wall on each side of the cart:





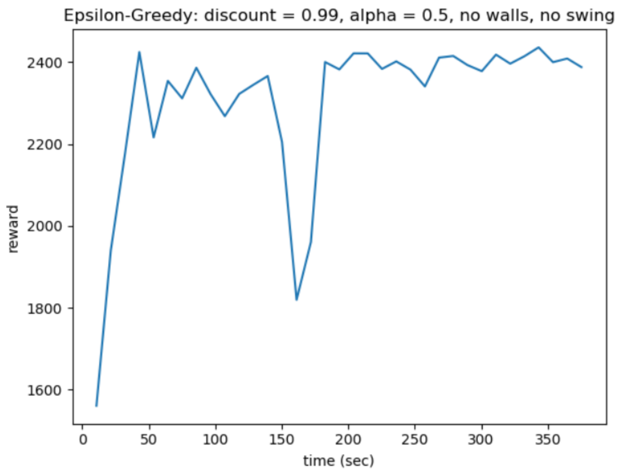
Though this problem may be harder, the constraint of the space does not affect much on the resolutions. However, there is a difference between ε-Greedy and SoftMax. It is much harder for ε-Greedy to keep the pole stable, and it takes more time to train it to get good results.

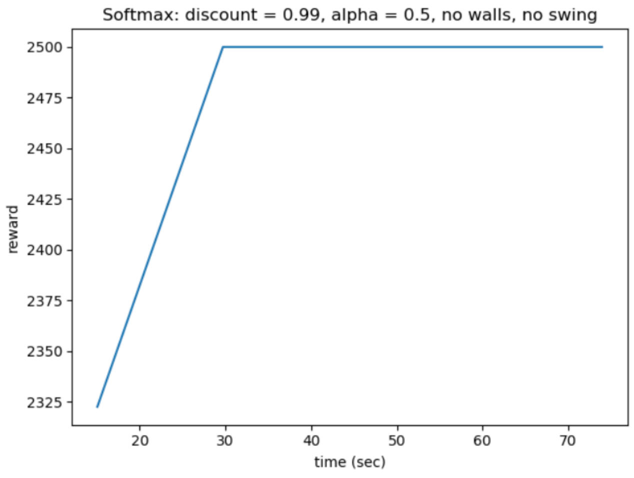
The state space stays the same as before and the lack of movability of the cart does not seem to really challenge the agents. However, this setting of the problem is more realistic since real life control problems have a finite space. So, it is a very positive outcome for us.

### C. Deep Q-Learning vs. Q-Learning

Unfortunately, our neural network implementation does not seem to actually learn any reasonable policy. It should be noted that it is the first time we try to implement such network, i.e. with a 'history' component for training, rather than examples that are known in advance. None the less, we are still interested in comparing between this method and the ones we implemented. So, we will use an existing code[3] for the comparison. The only difference is that the pole's initial position will be upwards and thus the agent only needs to balance it without the swing.

We ran our Q-Learning algorithms and the neural network (training step = 1850, Adam optimizer, learning rate = 0.001, loss cross entropy) for 6 minutes each (or until convergence in the SoftMax case). Both methods gave the same results. The agents were able to learn how to stabilize the pole and after the training process the pole did not fell even once (out of 10 tests).





We conclude that this aspect of the problem, whether the state space is discrete or continuous, does not affect the resolution of the problem.

### D. *Linear Q-Learning*

Like the neural network, here too we were unable to learn any reasonable policy.

## V. Conclusions

We conclude that the SoftMax policy is much better than ε-Greedy. SoftMax is much stable and promising than ε-Greedy that tends to often stumble.

These learning algorithms and policies are as good as a neural network for the predefined settings we examined and are much simpler. As for other settings, it is hard for us to answer how good are our models versus deep learning. But, it is clear that other approaches need to be taken in order to solve related but harder problems.

As for further work, one can think how to adjust our algorithms for the double inverted pendulum problem. We tried to use the same methods for this problem, i.e. ($\theta_0$, $\theta_1$, $\dot\theta_0$, $\dot\theta_1$) for the state space, but the state space is too big for this approach.

## References

[1] Martin Riedmiller, "Neural Reinforcement Learning to Swing-up and Balance A Real Pole".

[2] Fredrik Gustafsson, "Control of Inverted Double Pendulum using Reinforcement Learning".

[3] Neural network, https://gist.github.com/AhmetHamzaEmra/e7fdff82af99b58f4f91dd5df9460e7f.

Cart, https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py.