# Learning to Invert a Pendulum

Jagennath Hari
jh7454@nyu.edu

***Abstract*—This project aims to learn a policy for an inverted pendulum model to make it perform a swing-up motion. Beyond the task of inverting a pendulum, the goal is also to understand how Q-learning works, its limitations and advantages.**

## I. Introduction

The entire project involves a pendulum, firstly the model of the pendulum having a mass $m$ and length $l$ in Figure 1 will give an insight to the problem. The term $u$ in Figure 1 denotes the torque given as control to the pendulum for varying $\theta$ *i.e.* its position.

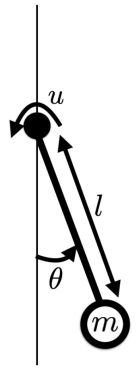



Fig. 1: Pendulum design

The goal will be start at $\theta = 0$ *i.e.* from the standing still non-inverted position to $\theta = \pi$ *i.e.* the inverted position. Figure 1 helps write the state vector (1) in terms of $\theta$(position) and its first order derivative $\omega$(velocity). The assumption of the term "swing" is when the bob returns to its starting position.

$$x = \begin{pmatrix} \theta \\ \omega \end{pmatrix} \tag{1}$$

Assuming discretization time $\Delta t$, the state (1) at time $t = n\Delta t$ is referred to as $x_n$. The next step is to introduce a discounted cost function (2a) which will penalize deviations from the inverted position but also encourage small velocities and control.

$$\sum_{i=0}^{\infty} \alpha^i g(x_i, u_i), \tag{2a}$$

$$g(x_i, u_i) = (\theta_i - \pi)^2 + 0.01\dot{\theta}_i^2 + 0.0001 \cdot u_i^2 \tag{2b}$$

(2b) can also be called the instantaneous cost having $\alpha^i = 0.99$ which gives rise to the discounted cost (2a).

For the rest of the report, the generic Q-learning algorithm using a temporal difference (TD) error and using a Q-table is explained in Section II. Section III has the results of Q-learning for the inverted pendulum problem. Finally, Section IV gives the conclusions of the experiment on the inverted pendulum problem.

## II. Q-learning with a Table

This section covers the theory behind the Q-learning algorithm using a table also commonly referred to as the Q-table.

Assuming the control $u$ can take only three possible values, and the range of both $\theta$ and $\omega$ are between $[0, 2\pi)$ and $[-6, 6]$ respectively having $50$ discretized states, the Q-table can be formulated.

Section II-A explains how to determine the cost and Section II-B gives an insight into Q-learning using a temporal difference error as control. Figure 2, the Q-table for the inverted pendulum problem is shown in Section II-C. Section II-D formulates a policy for obtaining control values. Finally, Section II-E has the pseudocode for Q-learning in Algorithim 1.

### A. Cost function

The cost can be computed by passing (1) and $u_i$ into cost function (2b), at discretized stage $i$, and this results in the instantaneous cost at stage $i$.

### B. Q-learning: off policy TD control

Q-learning is based on an an action-value function (3a), which can be further broken down (3b) in terms of the instantaneous cost (2b) and the optimal value function (3c) along with the discount factor (2a), the corresponding optimal policy for (3c) is given as (3d). The resulting TD error (3e) can be computed using (3a) and (3c). Then, 3a can be updated using (3e) and by selecting a step size/learning rate $\gamma$, the updated action-value function is (3f).

$$Q(x_t, u_t), \tag{3a}$$

$$Q(x_t, u_t) = g(x_t, u_t) + \alpha \min_u Q(x_{t+1}, u), \tag{3b}$$

$$J^*(x_t) = \min_u Q(x_t, u), \tag{3c}$$

$$\mu^*(x) = arg\min_u Q(x_t, u), \tag{3d}$$

$$\delta_t = g(x_t, u_t) + \min_u Q(x_{t+1}, u) - Q(x_t, u_t), \tag{3e}$$

$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \gamma\delta_t \tag{3f}$$

### C. Q-table

Using (3e), the resulting action-value function (3f) is stored as table. The previous assumptions of states and control in Section II gives the basic Q-table skeleton which will consist of $50$ rows, $50$ columns and $3$ channels for the inverted

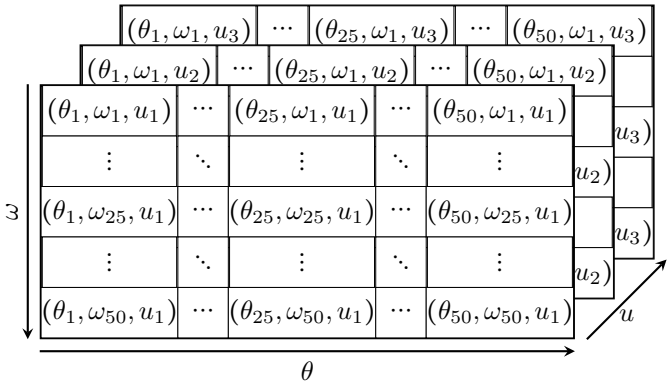



Fig. 2: Q-table

pendulum problem. Moreover, since (1) has 2 elements it is expanded accordingly having depth channels(tensor).

Every element in Figure 2 represents the update action-value function (3f), and this stored in a table form is called the Q-table.

### D. $\epsilon$-greedy policy

There is a possibility of missing a better actions/states if the optimal guess (3d) is always chosen. Likewise, choosing random actions might cause a bad guess for (3a). To overcome this limitation the $\epsilon$-greedy policy is introduced.

$$u_t = \begin{cases} \textit{random action}, & \textit{if probability } < \epsilon \\ \textit{arg } \min_u Q(x_t, u) & \textit{else} \end{cases} \tag{4}$$

The $\epsilon$-greedy policy (4) allows the use of both optimal guess and random action, the value of $\epsilon$ can be changed accordingly to increase or decrease the randomness.

### E. Q-learning algorithm

Initially the learning rate $\gamma$ and $\epsilon$ are chosen. An empty Q-table is created as shown in Figure 2 for every (1) and $u$(control). Loops are performed for each episode and each step in the episode, after initializing the initial state. Using (4), (2b), (3c), (3d), and (3e) the (3f) is updated.

**Algorithm 1** Q-learning

```
Initialize hyper-parameters γ ∈ [0, 1], ϵ and α
Initialize Q(x, u) for all states and controls   ▷ Q-table Table in Figure 2
for episode in episodes do
    Initialize state x_0
    for step in episode do
        Choose a control using ϵ-greedy policy from Q ▷ Section II-D, (4)
        Observe x_{t+1}
        Compute g(x_t, μ(x_t))   ▷ instantaneous cost (2b)
        Compute TD-error using α   ▷ TD-error (3e)
        Update Q(x_t, u_t)   ▷ Action-value update (3f)
    end for
end for
```

Using Algorithm 1 the updated (3c) and (3d) can be computed from the updated Q-table in Figure 2.

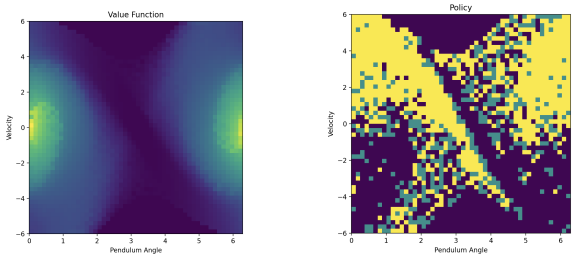
## III. Results on Inverted Pendulum

This section cover the implementation of Algorithm 1 using Figure 2 in python for the inverted pendulum problem. For each experiment *step* is 100 in Algorithm 1. Similarly, look up tables were created for $\theta \in [0, 2\pi)$ and $\omega \in [-6, 6]$. Figure 2, the Q-table is initialized as a numpy array having 50 rows, 50 columns and 3 channels. To understand the learning process moving averages for (2b) and (3e) were plotted having a window size of 100.

Section III-A covers the results when $u \in \{-4, 0, 4\}$, while Section III-B is for $u \in \{-5, 0, 5\}$. Section III-C compares the graphs of Section III-A and Section III-B. Lastly, Section III-D describes the effect of hyper-parameters during the learning process.

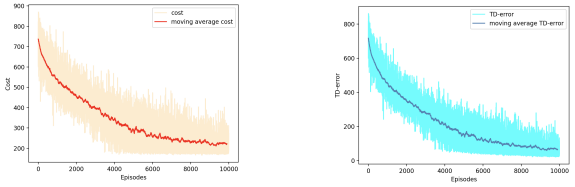
### A. Inverting the pendulum when $u \in \{-4, 0, 4\}$

The first step in Algorithm 1 is initializing the hyper-parameters, so $\alpha$ in (2a) is 0.99, $\gamma$ in (3f) is 0.1 and $\epsilon$ in (4) is 0.1. The rest of the algorithm is carried out the same way as depicted in Algorithm 1.




(a) (3c) when $u \in \{-4, 0, 4\}$ (b) (3d) when $u \in \{-4, 0, 4\}$

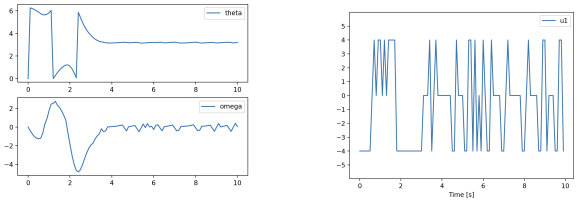
Fig. 3: Final (3c)(left) and 3d(right) after 10000 episodes




(a) (2b) when $u \in \{-4, 0, 4\}$ (b) (3e) when $u \in \{-4, 0, 4\}$

Fig. 4: Learning curves (2b)(left) and 3e(right) after 10000 episodes

Figure 3a and Figure 3b shows the final (3c) and (3d) after Algorithm 1 was iterated for 10000 episodes. Figure 4a depicts the change in cost (2b) while training for 10000 episodes, as observable it is converging after 4000 episodes. Similarly, Figure 4b indicates (3e) also converging around 4000 episodes. (1) and $u$, in Figure 5a and Figure 5b indicate that after 2 swings the pendulum achieved the desired inverted (1) at $t$ = 3.



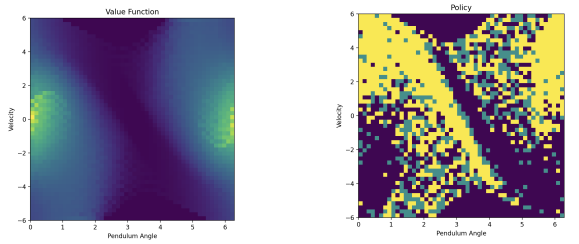

(a) Reaching desired (1) at $t = 3$

(b) Change in control to reach desired (1) in 2 swing(s)

Fig. 5: Evolution of (1)(left) and $u$(right) for inverted position

### B. Inverting the pendulum when $u \in \{-5, 0, 5\}$

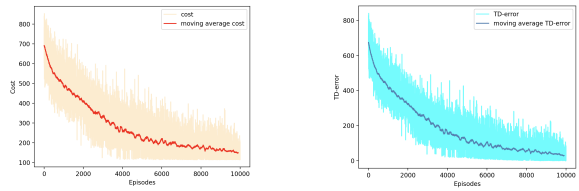
The hyper-parameters, $\alpha$ in (2a) is 0.99, $\gamma$ in (3f) is 0.1 and $\epsilon$ in (4) is 0.1 are initialized and the rest of the algorithm is carried out the same way as depicted in Algorithm 1.




(a) (3c) when $u \in \{-5, 0, 5\}$

(b) (3d) when $u \in \{-5, 0, 5\}$

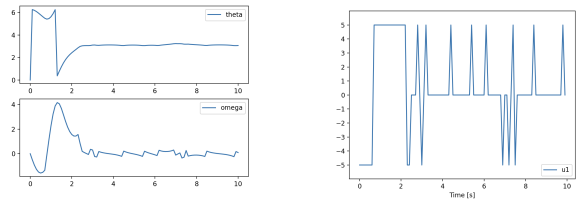
Fig. 6: Final (3c)(left) and 3d(right) after 5000 episodes




(a) (2b) when $u \in \{-5, 0, 5\}$

(b) (3e) when $u \in \{-5, 0, 5\}$

Fig. 7: Learning curves (2b)(left) and 3e(right) after 10000 episodes




(a) Reaching desired (1) at $t = 2$

(b) Change in control to reach desired (1) in 1 swing(s)

Fig. 8: Evolution of (1)(left) and $u$(right) for inverted position

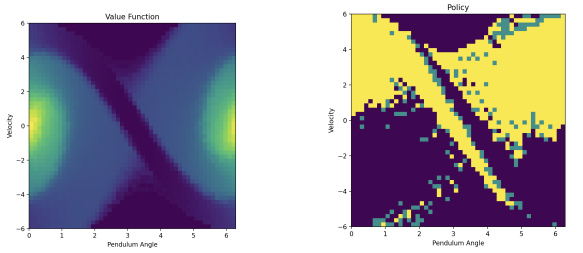
Figure 6a and Figure 6b shows the final (3c) and (3d) after Algorithm 1 was iterated for 10000 episodes. Figure 7a depicts the change in cost (2b) while training for 10000 episodes, as observable it is converging after 5000 episodes. Similarly, Figure 7b indicates (3e) also converging around 5000 episodes. (1) and $u$, in Figure 8a and Figure 8b indicate that after 1 swing the pendulum achieved the desired inverted (1) at $t = 2$.

### C. Comparing III-A and III-B

It can be observed that when $u \in \{-5, 0, 5\}$ the controller achieves the desired (1) in a smaller duration while lowering the number of swings when compared to control values when $u \in \{-4, 0, 4\}$. When comparing Figures 3b and 6b and Figures 3a and 6a the differences are subtle but by observing under the hood, the (3d) are slightly more diverse when $u \in \{-5, 0, 5\}$. As for the (3c) the difference is extremely low but there are more costly regions indicated when $u \in \{-5, 0, 5\}$.

### D. Effect of $\gamma$ and $\epsilon$

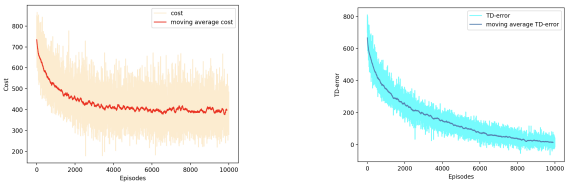
Changing the hyper-parameters can improve the learning process. By keeping $u \in \{-4, 0, 4\}$ as a constant the change can be observed. Section III-D1 analyses the result of having $\gamma$ at 0.2 while keeping $\epsilon$ at 0.5, similarly Section III-D2 shows the effect of further increasing both $\gamma$ and $\epsilon$ to 0.75 and 0.65 respectively. Section III-D3 compares the results of Section III-D1 and III-D2.




(a) (3c) when $\gamma = 0.2$ and $\epsilon = 0.5$

(b) (3d) when $u \in \{-4, 0, 4\}$

Fig. 9: Final (3c)(left) and 3d(right) after 5000 episodes

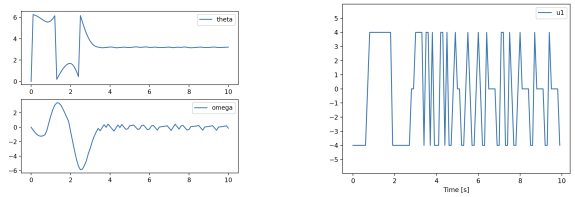



(a) (2b) when $u \in \{-4, 0, 4\}$

(b) (3e) when $u \in \{-4, 0, 4\}$

Fig. 10: Learning curves (2b)(left) and 3e(right) after 10000 episodes

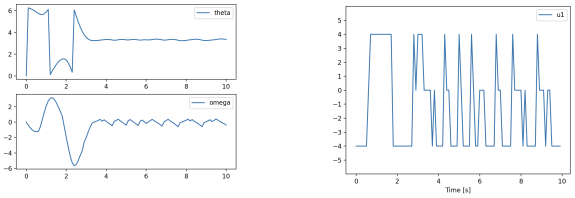
*1) $\gamma = 0.2$ and $\epsilon = 0.5$:* Figure 9a and Figure 9b shows the final (3c) and (3d) after Algorithm 1 was iterated for 10000 episodes. Figure 10a depicts the change in cost (2b) while training for 10000 episodes, as observable it is converging after 2000 episodes. Similarly, Figure 10b indicates (3e) also converging at the end of 10000 episodes. (1) and $u$, in Figure




(a) Reaching desired (1) at $t = 3$

(b) Change in control to reach desired (1) in 2 swing(s)

Fig. 11: Evolution of (1)(left) and $u$(right) for inverted position



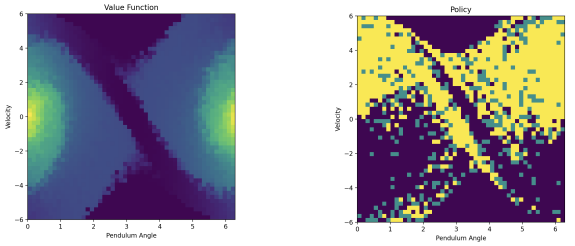

(a) Reaching desired (1) at $t = 3$

(b) Change in control to reach desired (1) in 2 swing(s)

Fig. 14: Evolution of (1)(left) and $u$(right) for inverted position

11a and Figure 11b indicate that after 2 swings the pendulum achieved the desired inverted (1) at $t = 3$.

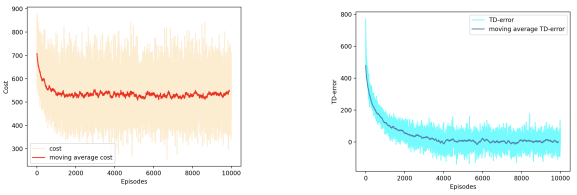
When Figure 3a and Figure 9a are compared along with Figure 3b and Figure 9b, the latter plots indicate more efficient learning process as it able converge faster due to the increase in $\gamma$, though the variance has increased in (2b) due to the increase in (4) which increased the randomness as seen in Figures 10a and Figure 4a. The Figures 4b and 10b indicate lower variance in the latter plot due to high values of (4) and $\gamma$, this can be interpreted as the training process is continuously minimizing the (3e). The direct comparison between Figure 5a and Figure 11a, and Figure 5b and Figure 11b indicate that the new controller achieves the desired (1) in a similar duration and with the same number of swings.




(a) (3c) when $\gamma = 0.75$ and $\epsilon = 0.65$

(b) (3d) when $u \in \{-4, 0, 4\}$

Fig. 12: Final (3c)(left) and 3d(right) after 5000 episodes




(a) (2b) when $u \in \{-4, 0, 4\}$

(b) (3e) when $u \in \{-4, 0, 4\}$

Fig. 13: Learning curves (2b)(left) and 3e(right) after 10000 episodes

*2) $\gamma = 0.75$ and $\epsilon = 0.65$:* Figure 12a and Figure 12b shows the final (3c) and (3d) after Algorithm 1 was iterated for 10000 episodes. Figure 13a depicts the change in cost (2b) while training for 10000 episodes, as observable it is converging after 2000 episodes. Similarly, Figure 13b indicates (3e) also converging at 2000 episodes. Similarly, (1) and $u$, in Figure 14a and Figure 14b indicate that after 2 swings the pendulum achieved the desired inverted (1) at $t = 3$.

When Figure 3a and Figure 12a are compared along with Figure 3b and Figure 12b, the latter plots indicate more diverse learning process as it able converge faster due to the increase in $\gamma$, though the variance has increased in (2b) due to the increase in (4) which increased the randomness as seen in Figures 13a and Figure 4a. The Figures 4b and 13b indicate lower variance in the latter plot due to high values of (4) and $\gamma$, this can be interpreted as the training process is continuously minimizing the (3e). The direct comparison between Figure 5a and Figure 14a, and Figure 5b and Figure 14b indicate that the new controller achieves the desired (1) in a similar duration and with the same number of swings.

*3) Comparing III-D1 and III-D2:* Comparing Figure 10a and Figure 13a it as be observed that the variance increases due to the increase in (4) which is causing an increase in randomness in the latter plot, this leads to the learning process trying more random actions rather than the optimal control. Though subtle it can observed in higher cost at each episode as the increase in randomness is causing actions which leads away from the desired (1).

Figure 10b and Figure 13b helps grasp of the influence of $\gamma$, higher the value causes faster convergence as often the case in classical machine learning, hence the (3e) gets minimized faster when comparing Figures 13b with the Figures 4b, 7b and 10b.

## IV. Conclusion

The critical factors which influence the controller to achieve and maintain the inverted state are (4) and $\gamma$ in (3f). An intermediate value($\alpha$, $\epsilon$) must be chosen to achieve the desired state, while having high values of $\epsilon$ cause the controller to overshoot the desired state, counter intuitively low values causes the designed controller unable to reach the desired state. Increasing the $\gamma$ causes the learning process to converge faster.

The range of $u$(control) does not really effect the controller much as the learning process is able accommodate different ranges, though a higher influences the controller to achieve the desired (1) in a smaller duration while lowering the number of swings. Overall each of the designed controller was able to accomplish the given task, prompting Q-learning to be a viable method when designing controllers.