Bio-inspired Intelligence project: Stabilizing inverted pendulum using SAC

Niels Stienen, 5595738

CONTENTS

NOMENCLATURE

Abbreviations

DDPG	Deep Deterministic Policy Gradient
MuJoCo	Multi-Joint dynamics with Contact
RL	Reinforcement Learning
SAC	Soft Actor Critic

Symbols

$\dot{\theta}$	angular pendulum velocity
$\dot{x}$	linear cart velocity
γ	discount factor
τ	smoothing coefficient
θ	pendulum angle
x	cart position

I. Introduction

In the field of control systems and robotics, stabilizing an inverted pendulum on a cart is a classic problem to test control algorithms due to its nonlinear dynamics and inherent instability. Additionally, several real-life problems can be simplified to this problem making it very applicable in varied settings. This problem, also known as the "cart-pole" problem, involves a pole mounted on a cart that moves along a horizontal track. The objective is to apply appropriate forces to the cart to keep the pole balanced upright.

This report will cover one method of doing so which is Reinforcement Learning (RL) using the Soft Actor-Critic (SAC) method. SAC is an off-policy RL algorithm that leverages both the policy gradient and Q-learning approaches to achieve stable and sample-efficient learning. The algorithm is based on the principle of entropy maximization, which encourages exploration by optimizing not only for the expected reward but also for the randomness of the policy. All code and images and videos resulting from the project are available on GitHub [1].

II. Simulation Environment

The environment used for implementing the inverted pendulum on a cart is MuJoCo [2] together with Gymnasium [3]. The inverted pendulum is one of the default environments available within Gymnasium based on the work done by Barto, Sutton, and Anderson [4].

A. Action Space

The agent in this environment takes a 1-element vector for actions which is the force applied on the cart in Newtons. The action space thus is continuous in the range $[-3, 3]$.

TABLE I: Action Space

Action	Min	Max	Symbol	Units
Force applied on the cart	-3	3	τ	Force (N)

B. Observation Space

The observation space is larger as there are four different observations returned. These observations are the position of the cart along the linear surface, the vertical angle of the pole on the cart, the linear velocity of the cart, and the angular velocity of the pole on the cart. These observations are all continuous in the real ($\mathbb{R}$) domain.

TABLE II: Observation Space

Observation	Min	Max	Symbol	Units
Position of the cart along the linear surface	$-\infty$	∞	x	position (m)
Vertical angle of the pole on the cart	$-\infty$	∞	θ	angle (rad)
Linear velocity of the cart	$-\infty$	∞	$\dot{x}$	velocity $\left(\frac{m}{s}\right)$
Angular velocity of the pole on the cart	$-\infty$	∞	$\dot{\theta}$	angular velocity $\left(\frac{rad}{s}\right)$

C. Rewards

The default reward function used is to give a reward of $+1$ for each timestep the pendulum stays upright within a certain angle limit. This however, leads to some undesired behaviour which is why some other reward function are defined as well which will be further compared in section IV Results.

The first reward function thus can be stated as:

$$r = \begin{cases} 1 & \text{if} \quad |\theta| < 0.2 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The reward function was changed to also penalize deviations from the origin such that it attempts to stabilize the pendulum around the origin giving the reward function:

$$r = \begin{cases} 1 - x^2 - \theta^2 & \text{if} \quad |\theta| < 0.2 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Finally, the reward function was changed to include the velocity terms in an attempt to reduce oscillations around the stabilizing equilibrium. The velocity terms are weighed less in the reward function as they are not as important compared to the position terms.

$$r = \begin{cases} 1 - x^2 - \theta^2 - \frac{\dot{x}^2}{10} - \frac{\dot{\theta}^2}{10} & \text{if} \quad |\theta| < 0.2 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

III. Methodology

To optimize the parameters, the following methodology was adopted. First using the default parameters the reward function was designed to give the desired results. Then the effect of the hyperparameters was analyzed to find a near-optimal set of parameters for the problem with the new reward function.

The algorithm used is the Soft Actor Critic algorithm [5] which optimizes a stochastic policy in an off-policy way, creating a middle-ground between stochastic policy optimization and DDPG-style approaches. A central feature of SAC is entropy regularization. The policy is trained to maximize a trade-off between the expected return and entropy. Here, entropy is a measure of randomness in the policy. The use of entropy is similar to the trade-off between exploration and exploitation. Increasing entropy results in more exploration as there is more randomness in the policy. This can help to prevent the policy from converging to a bad local optimum prematurely.

To explain Soft Actor Critic first the use of entropy in the reinforcement learning needs to be shown. The agent gets a bonus reward at each time step proportional to the entropy ($H = E[-logP(x)]$) of the policy at that timestep. The policy update equation then becomes:

$$\pi^* = \arg\max_{\pi} \underset{\tau \sim \pi}{E} \left[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t))) \right] \tag{4}$$

where $\alpha > 0$ is the coefficient that determines how heavily the entropy is weighed. The updated Bellman equation then becomes:

$$Q^{\pi}(s, a) = \underset{s' \sim P, a' \sim \pi}{E} [R(s, a, s') + \gamma (Q^{\pi}(s', a') + \alpha H(\pi(\cdot|s')))] \tag{5}$$

The loss functions for the Q-networks used in SAC are:

$$L(\phi_i, D) = \underset{(s,a,r,s',d) \sim D}{E} \left[\left(Q_{\phi_i}(s, a) - y(r, s', d) \right)^2 \right] \tag{6}$$

where the target $y(r, s', d)$ is given by

$$y(r, s', d) = r + \gamma(1 - d) \left(\min_{j=1,2} Q_{\phi_j}(s', \tilde{a}') - \alpha \ln \pi_{\theta}(\tilde{a}'|s') \right), \tilde{a}' \sim \pi_{\theta}(\cdot|s') \tag{7}$$

To optimize the policy samples are then obtained using a squashed Gaussian policy:

$$\tilde{a}_\theta(s, \xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s)\xi\right), \xi \sim \mathcal{N}(0, I) \tag{8}$$

Figure 1 contains the pseudocode of the specific implementation used in the project code.

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \Big(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|\, s\right) \Big),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.
      Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$

    end for
  end if
until convergence

Fig. 1: Pseudo code implementation of the SAC algorithm [6]

IV. Results

The results section is split into four parts, first, the different reward functions are trained and evaluated, the influence of how long the model is trained is evaluated, the hyperparameters are tuned, and finally, the model is trained with the optimized set of parameters to analyze the performance of the model. All models trained as well as videos how each model performed are available on GitHub [1].

A. *Reward function*

Each reward function has been trained for 100000 timesteps after which the results are compared. As can be seen in Figure 4 all reward functions increase at a similar rate indicating that in all cases the pendulum stays upright for a longer duration. When looking at the actual observations of a run for each reward function, it becomes clear that there are some differences. Figure 2 shows that both the default reward and the position-based reward deviate from the starting position and eventually the pendulum falls over terminating the run. Training these reward functions for longer the pendulum stays upright longer but the position keeps increasing over time for the default reward function. For the position-based reward function longer training results in constant overshooting of the starting position.

Based on these results the reward function that incorporates both the position and velocity observations is used for the latter training and parameter tuning.

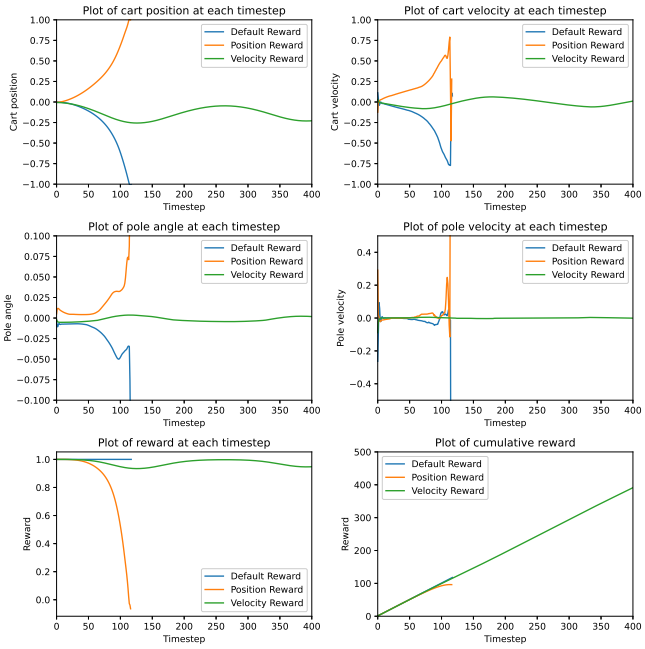

Fig. 2: Comparison of training results for the different reward functions used.

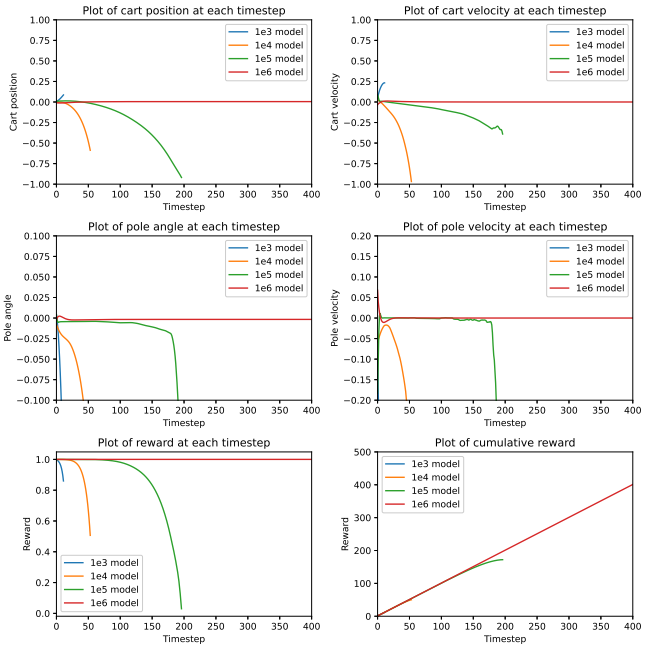

Fig. 3: Comparison of performance for different training lengths.

B. *Timesteps*

The model is trained several times for a varying number of steps. The model significantly improves with a longer training time until converging after training for one million steps or longer. Since the velocity results are already close to desired with the model that is trained for 100000 steps it might be possible to decrease the needed training time by slightly adjusting the weights on the velocity and position components in the reward function. For this assignment however, the reward function is kept the same and the parameters will be tuned using a model that is trained for 100000 steps as a trade-off between training time and performance needs to be made.

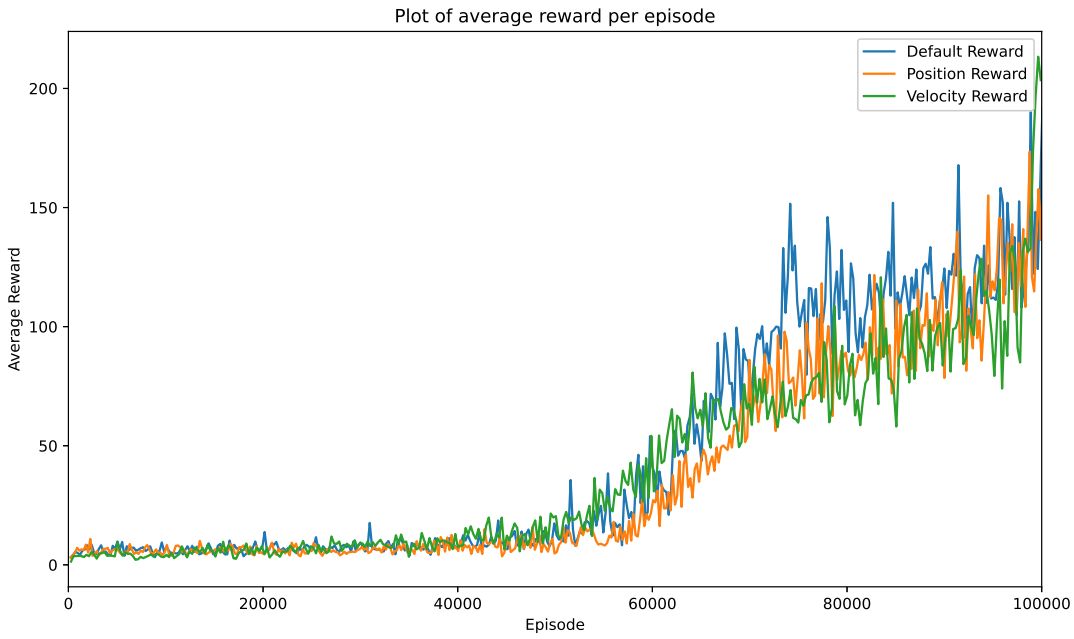

Fig. 4: Comparison of the average reward during the learning process.

C. *Parameter tuning*

The parameters investigated are τ, γ, and the learning rate. τ is the target smoothing coefficient used in the soft update rule, determining how fast the target networks are updated. γ is the discount factor in the policy update equation responsible for how important the long-term rewards are compared to the short-term rewards. The learning rate controls the step size of the gradient update and is a trade-off between stability and learning speed.

As seen in Figure 5 the discount factor should be close to one; otherwise, the agent overshoots in its correction and the pendulum falls down. This issue is prevented by prioritizing the long-term rewards although some small drift from the initial position occurs.

For the smoothing coefficient the optimal value found is $\tau = 0.01$. Using larger values causes the updates to occur too fast resulting in unstable behaviour seen by the oscillations as seen in Figure 6.

For the learning rate, the optimal value is found to be in the range $0.0003 \leq lr_{opt} < 0.0004$ as outside of this range the learned model has significantly worse performance for equal training time. Figure 7 shows that a faster learning rate results in oscillating behaviour in the stabilizing of the pendulum while a slower learning rate was not sufficient to learn to stabilize the pendulum within the training steps used.

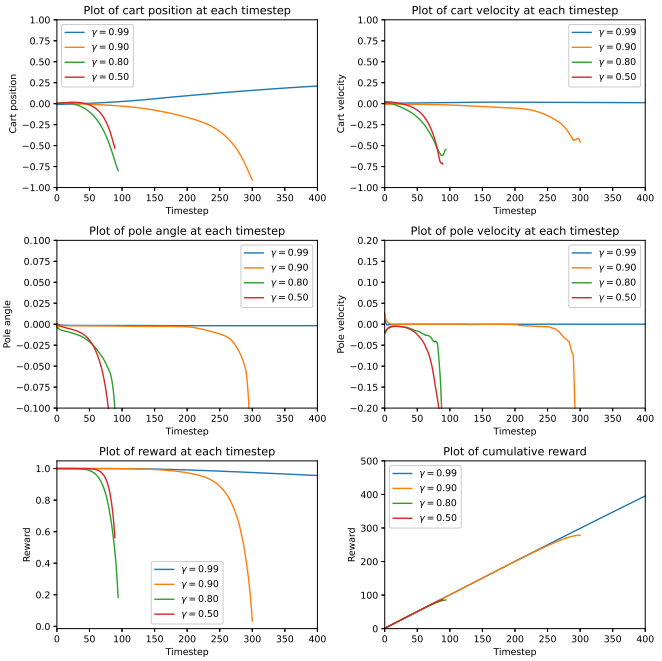

Fig. 5: Comparison of performance for different discount factors.

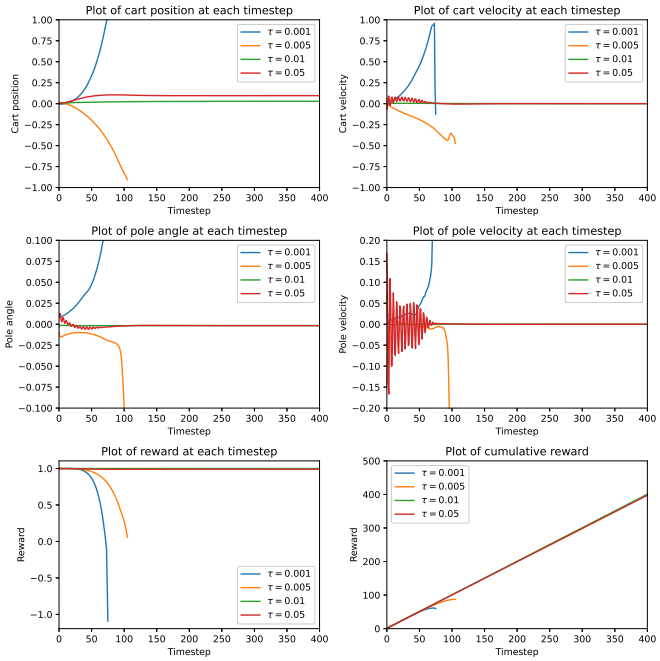

Fig. 6: Comparison of performance for different smoothing coefficients.

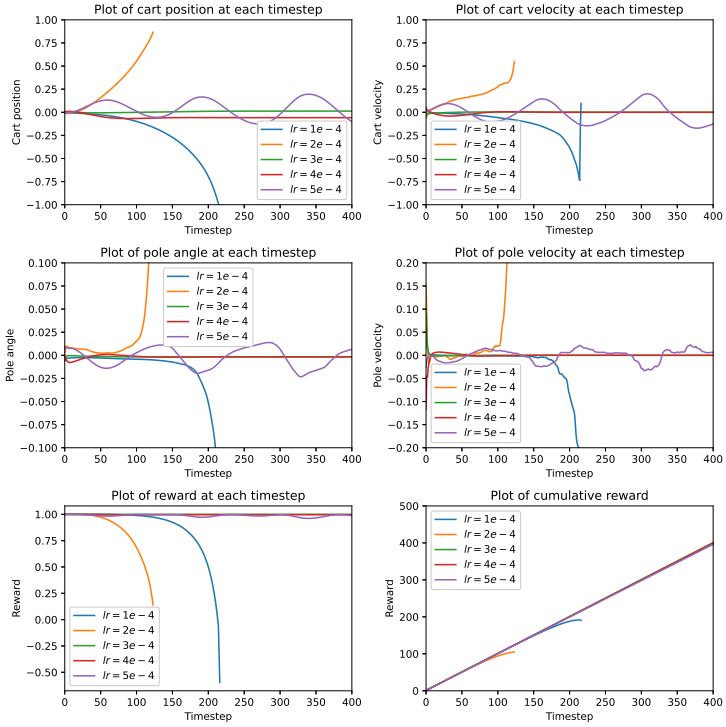

Fig. 7: Comparison of performance for different learning rates.

With the optimal parameters found from the above analysis, a new model was trained for a longer duration to evaluate the final performance of the model. Figure 8 shows that for varying random initial conditions, the model can stabilize the inverted pendulum and settle in the unstable equilibrium. Figure 9 shows the average reward per episode during the training process. Around 700000 episodes the average reward spikes up significantly indicating that the model has learned to stabilize the pendulum in each starting configuration for the duration of a simulation which is 1000 timesteps.

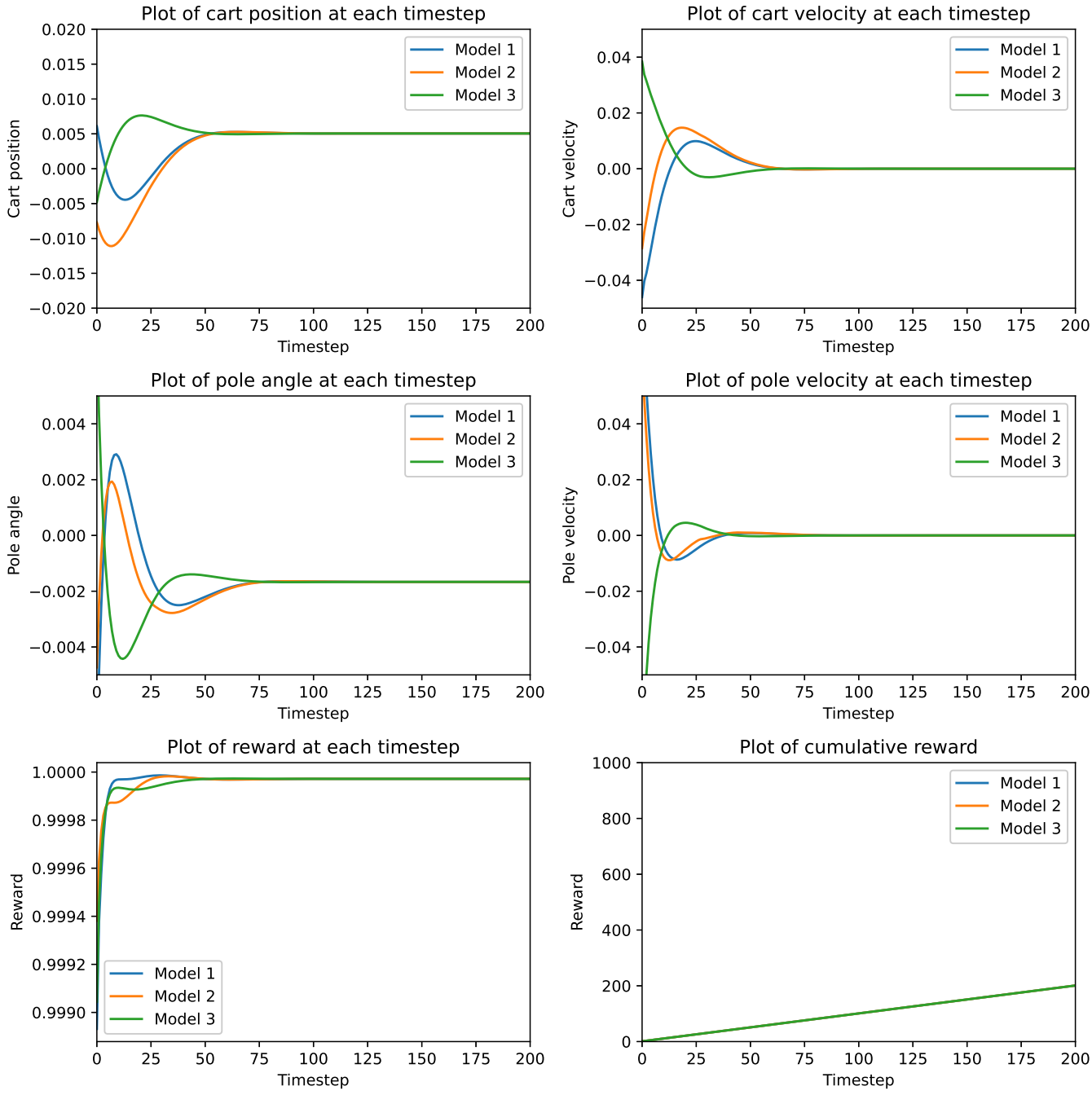

Fig. 8: Comparison of different runs using the trained final model. All random initial positions converge to a stabilized inverted pendulum at the origin.

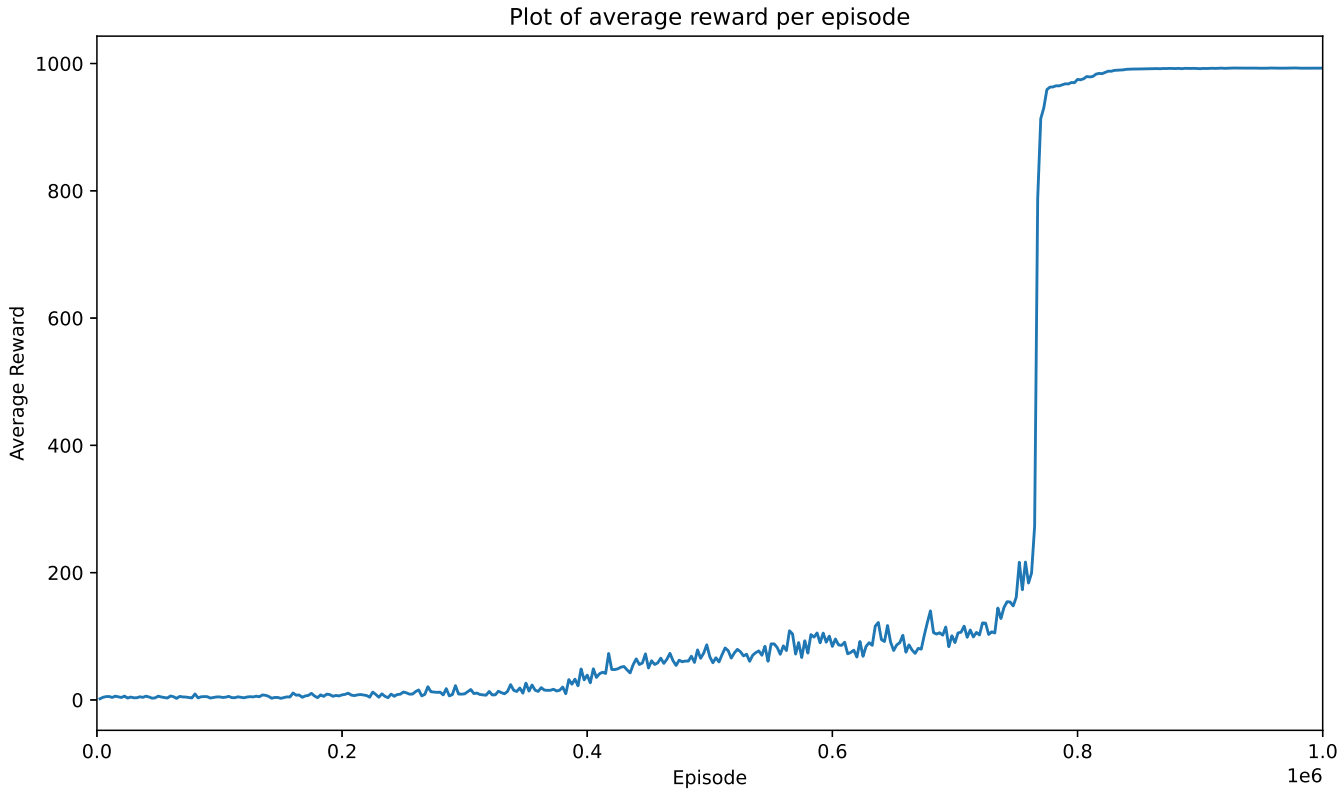

Fig. 9: Comparison of performance for different learning rates.

V. Conclusion

The Soft Actor-Critic algorithm was successful in learning to stabilize the inverted pendulum. The learned strategy was to quickly give a control input to push the inverted pendulum in the correct direction and get it to stop moving as quickly as possible. This behavior arises from the penalty factors added to the reward function. In the original environment, the pendulum would stay upright but the cart would keep moving at a constant velocity. By penalizing both the position and velocity the learned model stabilizes the pendulum as close to the unstable equilibrium as possible. In the real world due to the addition of friction, this would result in the pendulum staying upright without any additional control input needed as long as no additional disturbances are acting on the system.

References

[1] N. Stienen, "Soft actor critic for inverted pendulum," 2024. [Online]. Available: https://github.com/StienenNiels/BioInspired_Intelligence_AE4350

[2] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[3] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.

[4] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018. [Online]. Available: https://arxiv.org/abs/1801.01290

[6] "Soft actor-critic — spinning up documentation." [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/sac.html