Kemal Alp Sezer
245996

Learning for Control: Project C

Introduction:

The goal of Project C is to develop and evaluate control policies using an actor-critic learning approach for minimizing the cumulative running cost function while dynamically updating the system's state. This project starts with a simpler 1D integrator system and advances to a more complex 2D integrator system. The optimal control problem is addressed using the CasADi solver to find the optimal control inputs and associated value functions, which are then used to train a neural network-based Critic.

The actor-critic methodology involves training two neural networks: the Critic network estimates the value function for a given state, while the Actor network determines the optimal control policy by minimizing the expected cost. This project evaluates the performance of the trained policies by comparing the results to the initial cost and analyzing the effectiveness of the learned control strategies.

The Cost Functions:

The running cost function, l(x,u), is composed of two parts: a control effort term f(u) and a state cost term f(x): l(x,u)=f(u)+f(x)

Two different cost functions are tested, both having a similar structure:

- f(u) is a convex quadratic term in the control input $0.5 \cdot u_k^{\,2}$, penalizing large control efforts.
- f(x) is a polynomial of the state x_k , designed to drive the system towards specific desired states or away from undesired states. The chosen polynomials create a complex, non-convex landscape for the state-related cost function.

The running cost proposed by the assignment:

$$l(x,u) = 0.5 \cdot u_k^{\,2} + (x_k - 1.9) \cdot (x_k - 1.0) \cdot (x_k - 0.6) \cdot (x_k + 0.5) \cdot (x_k + 1.2) \cdot (x_k + 2.1)$$

This polynomial has six roots and is not nonnegative, implying that the total cost can decrease over time (can be seen in figure 2).

The modified cost function:

$$l(x,u) = 0.5 \cdot u_k^{\,2} + (x_k - 2)^2 \cdot (x_k + 2)^2$$

This function is nonnegative and guides the agent towards the states $x_k = 2$ and $x_k = -2$, as the state polynomial function f(x) is minimized at these points (can be seen in figure 3).

Kemal Alp Sezer
245996

This loss function is used throughout the assignment (if not mentioned otherwise) as it can be easily interpreted.

Finding the Optimal Policy for the 1D Integrator System:

In the 1D integrator system, the state at the next time step is the current state plus the control input scaled by the time step Δt. The state update equation is given by:

$$x_{k+1} = x_k + \Delta t \cdot u_k$$

Here, x_k represents the current position of the agent, Δt is the time between consecutive actions, and u_k is the control input that determines the next position together with the current position acting as the velocity of the agent. The system operates with a time step Δt of 0.5 seconds and considers a control horizon of 10 steps, generating a trajectory over 5 seconds. Control inputs are constrained within the range [-1, 1] to limit the agent's speed.

Solving the Initial Optimal Control Problem:

The optimal control problem is solved using the CasADi solver for an initial state x, given the system dynamics and the cost function, to find a state value function V(x). For the 1D integrator, 50 initial states uniformly distributed between -3 and 3 are chosen. This interval is narrow because the roots of the polynomial state term are within this range, making the function convex outside this interval. For each initial state, the 10 optimal control inputs that minimize the total cost are stored along with the optimal total cost (value function). Using these inputs and initial states, the agent's trajectory can be computed, resulting in a dataset of 50 value functions for 50 different states used to train the critic network.

Preprocessing the Data:

The initial states and their corresponding value functions are scaled between the interval [-1, 1] using min-max normalization to ensure better convergence and stability. The dataset is then split into a training set (80%) and a validation set (20%) which are placed in **batches of size 10**.

Critic Network:

The critic network estimates the value function given the initial state. Given a horizon of 10 steps in our training data, the expected cost output from the critic provides the expected total cost over these steps.

Network Architecture for Critic Network:

```
self.fc1 = nn.Linear(1, 128)
self.fc2 = nn.Linear(128, 128)
self.fc3 = nn.Linear(128, 128)
self.fc4 = nn.Linear(128, 64)
self.fc5 = nn.Linear(64, 1)
```

Kemal Alp Sezer
245996

The critic network is structured as a fully connected feedforward neural network:

- **Input Layer**: Single input neuron accepting the state/position.
- **Hidden Layers**: Four hidden layers using ReLU activation functions to mitigate the vanishing gradient problem, with the number of neurons increasing in the middle layers to enhance pattern recognition.
- **Output Layer**: Maps neurons from the fourth hidden layer to a single output neuron representing the cost of the given state under the optimal policy.

Training the Critic Network:

- The Mean Squared Error (MSE) Loss is used as the loss function for this regression problem.
- The Adam optimizer is used with a learning rate of 1e-4.
- The critic was trained for 300 epochs.
- The training lasted 9 seconds in total.

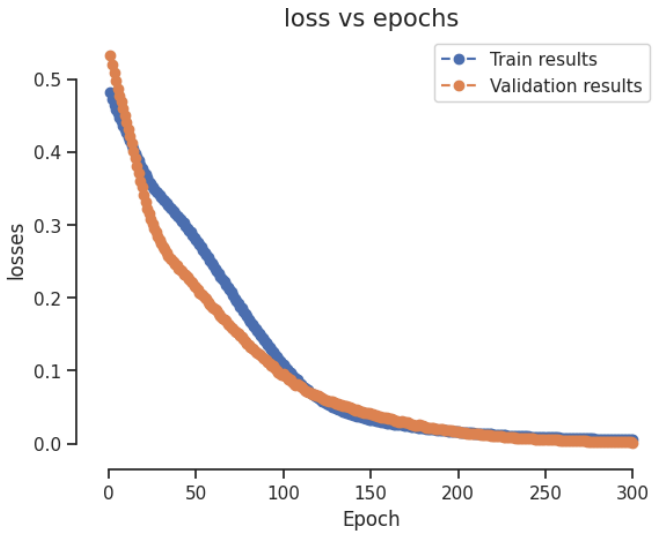

Fig 1: Training and validation losses of the Critic decreasing together with no indication of overfitting.

Kemal Alp Sezer
245996

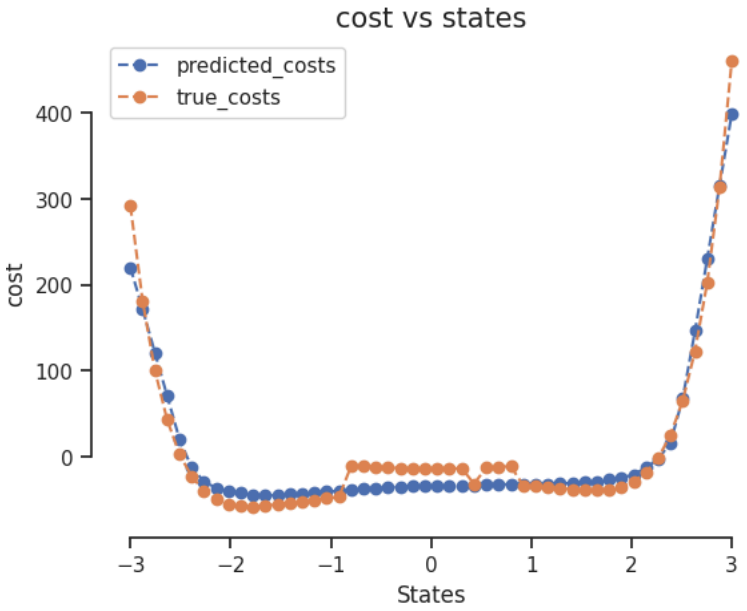

Fig 2: the value function graph based on the original running cost function corresponding to the initial states for a horizon of 10. The blue indicates the predictions of the critic while the orange indicates the optimal cost from the solver

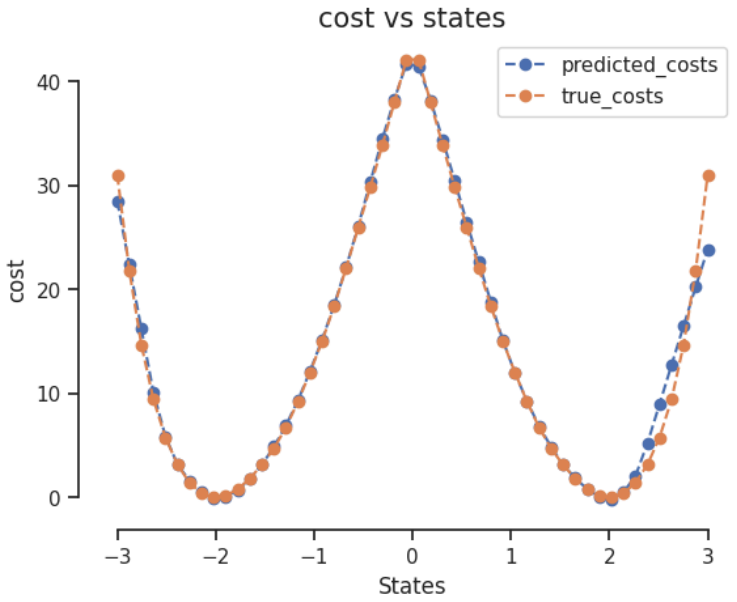

Fig 3: the value function graph based on the modified cost function corresponding to the initial states for a horizon of 10. The blue indicates the predictions of the critic while the orange indicates the optimal cost from the solver.

Actor Network:

After training the critic network, the actor network is set up to map states to actions, aiming to follow the optimal policy and minimize expected cost based on the current state.

Kemal Alp Sezer
245996

Network Architecture for Actor Network:

```
self.fc1 = nn.Linear(1, 256)
self.fc2 = nn.Linear(256, 256)
self.fc3 = nn.Linear(256, 256)
self.fc4 = nn.Linear(256, 256)
self.fc5 = nn.Linear(256, 128)
self.fc6 = nn.Linear(128, 128)
self.fc7 = nn.Linear(128, 64)
self.fc8 = nn.Linear(64, 1)
```

- **Input Layer**: Takes the current state as input.
- **Hidden Layer**: 7 hidden layers using a ReLU activation function.
- **Output Layer**: Uses a final linear layer followed by tanh activation function to produce continuous outputs in the range [-1, 1], adhering to constraints.

Training the Actor Network:

During training, the actor generates actions based on the current state using the feedforward network. The running cost is computed with respect to the current state and action. The next state is predicted using the 1D integrator dynamics, normalized, and fed into the critic network, which estimates the future cost. The loss function used is the sum of the running cost $l(x,u)$ and the expected future cost for the given action (action value function) $V(f(x,u))$.

- The Adam optimizer with a learning rate of 5e-5 is used.
- Trained for 1000 epochs.
- $\pi(x) = min_u\ l(x,u) + V(f(x,u))$ used as the loss function
- Training lasted 31 seconds.

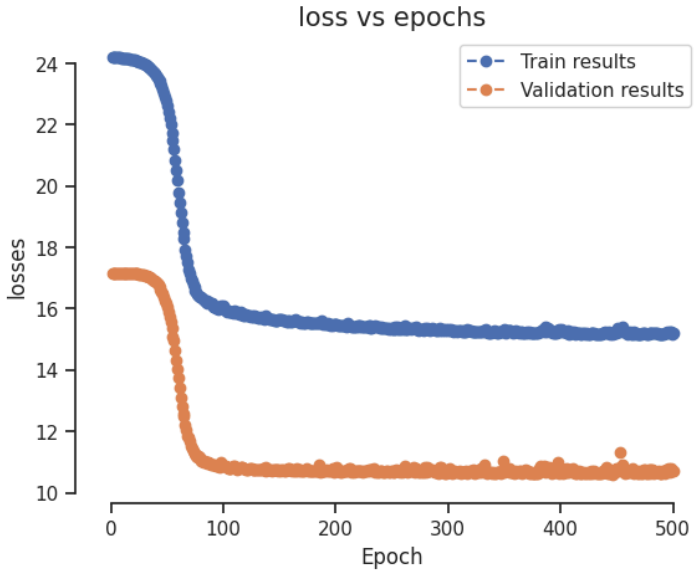

Fig 4: Training and validation losses of the Actor Network

Kemal Alp Sezer
245996

Results for 1D Integrator:

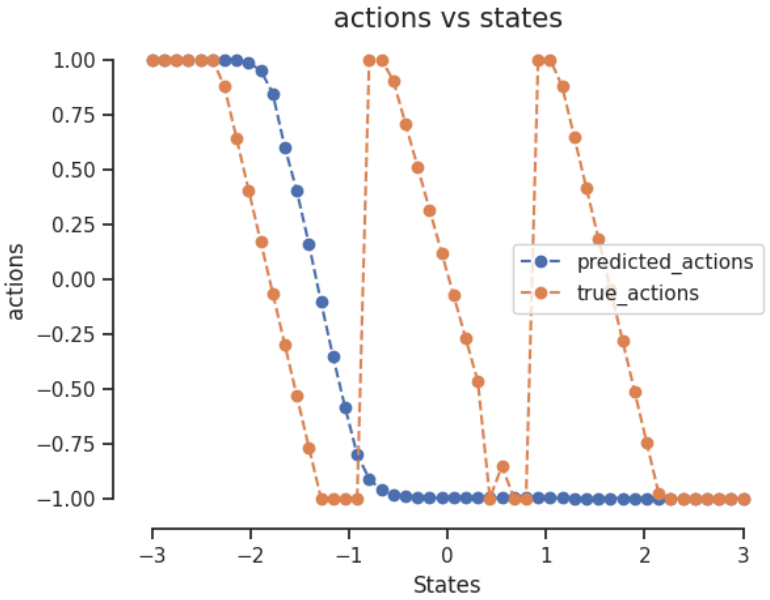

Fig 5: The graph of optimal policy, actions given a state, blue shows the actors predictions while orange shows the optimal action found from the solver. (for the original running cost function)

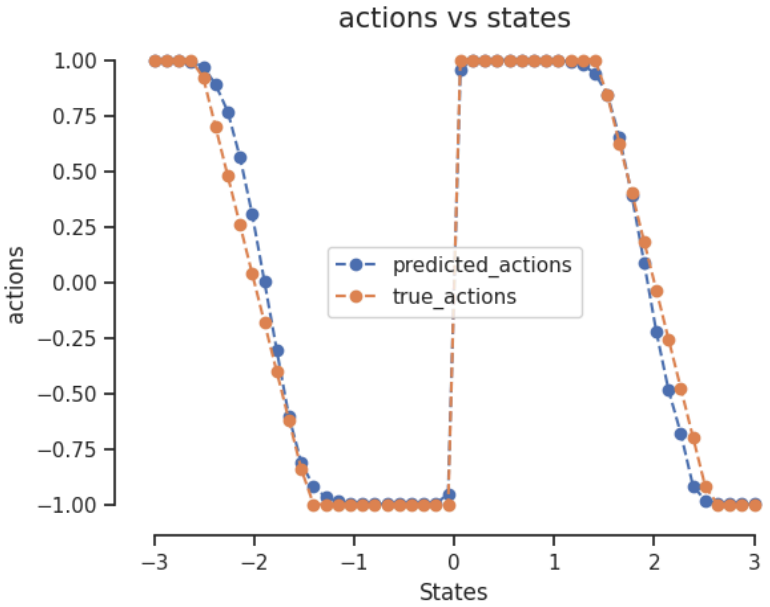

Fig 6: The graph of optimal policy, actions given a state, blue shows the actors predictions while orange shows the optimal action found from the solver. (for the modified cost function)

Kemal Alp Sezer
245996

Finding the Optimal Policy for the 2D Integrator System:

In the 2D integrator system an additional parameter is introduced generating a state vector:

$$p_{k+1} = p_k + \Delta t \cdot v_k + 0.5 \cdot u_k \cdot \Delta t^2$$

$$v_{k+1} = v_k + \Delta t \cdot u_k$$

Here, p_k represents the current position of the agent while v_k represents the velocity.

Δt is the time between consecutive actions, and u_k is the control input that determines the acceleration of the agent affecting both the position and the velocity. The system operates with a time step Δt of 0.5 seconds and considers a control horizon of 10 steps, generating a trajectory over 5 seconds. Control inputs are constrained within the range [-1, 1] to limit the agent's acceleration.

Solving the Initial Optimal Control Problem:

The optimal control problem is solved using the CasADi solver for an initial state [p,v], given the system dynamics and the cost function, to find a state value function V(p,v). For the 2D integrator, the initial states are chosen from a uniform mesh grid by considering 20 points within the interval [-3,3] for both position and velocity resulting in 400 samples. For each initial state, the 10 optimal control inputs that minimize the total cost are stored along with the optimal total cost (value function). The same preprocessing steps were followed as the 1D integrator before feeding the data to the critic network.

Critic Network:

```
self.fc1 = nn.Linear(2, 128)
self.fc2 = nn.Linear(128, 128)
self.fc3 = nn.Linear(128, 128)
self.fc4 = nn.Linear(128, 64)
self.fc5 = nn.Linear(64, 64)
self.fc6 = nn.Linear(64, 32)
self.fc7 = nn.Linear(32, 1)
```

Network Architecture:

A few adjustments are made to the critic network,

- **Input Layer** was adjusted to accept 2 inputs position and velocity.
- **More hidden Layers** (6 in total) were added to capture the more complex relationship between the value function and the state vector.

Training the Critic Network:

- The Mean Squared Error (MSE) Loss is used.

- The Adam optimizer updates the network's weights starting with a learning rate of 1e-4. Training and validation losses were shown to decrease together with no indication of overfitting.
- The critic was trained for 500 epochs and the training lasted 55 seconds in total.

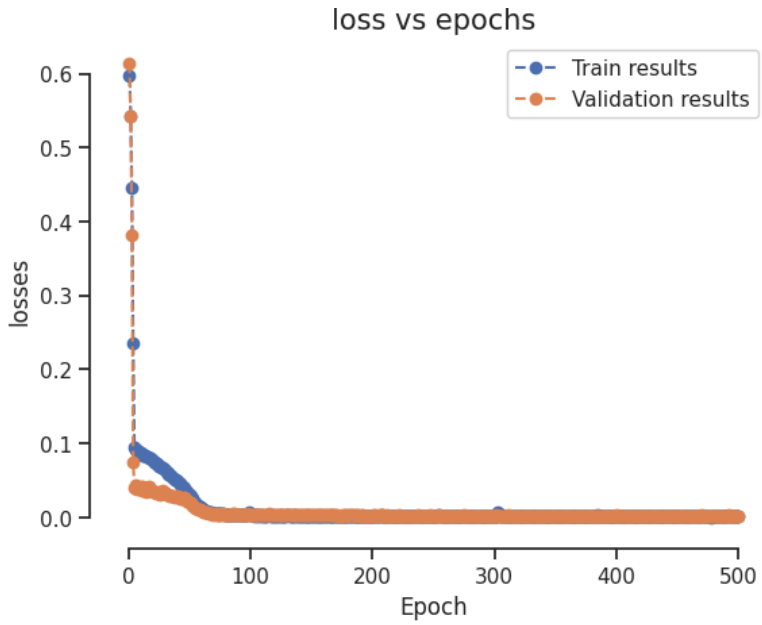

Fig 7: Training and validation losses of the Critic Network for the 2D integrator case

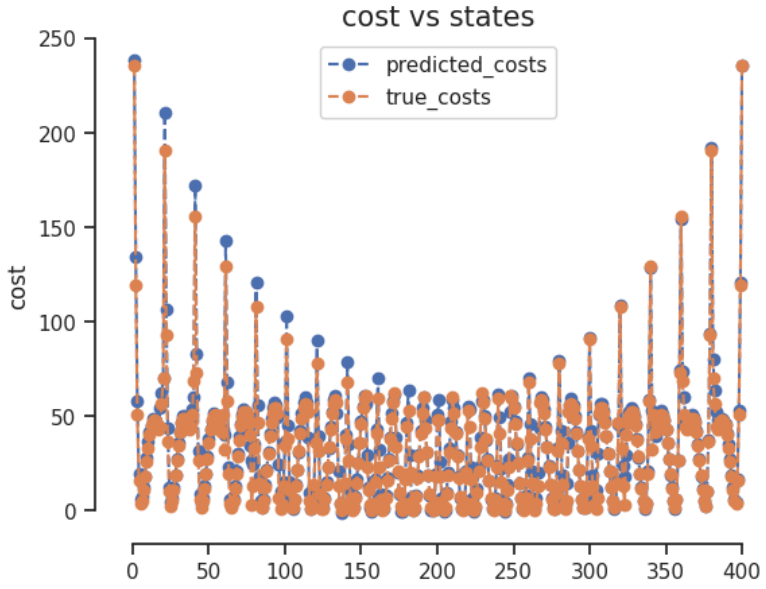

Fig 8: A 2D illustration of the value function graph with the x axis showing the enumerated 400 states instead of the p and v values just to show the overlap. The blue indicates the predictions of the critic while the orange indicates the optimal cost from the solver

Kemal Alp Sezer
245996

Actor Network:

After training the critic network, the actor network is set up to map states to actions, aiming to follow the optimal policy and minimize expected cost based on the current state.

Network Architecture for Actor Network:

```
self.fc1 = nn.Linear(2, 256)
self.fc2 = nn.Linear(256, 256)
self.fc3 = nn.Linear(256, 512)
self.fc4 = nn.Linear(512, 512)
self.fc5 = nn.Linear(512, 256)
self.fc6 = nn.Linear(256, 128)
self.fc7 = nn.Linear(128, 64)
self.fc8 = nn.Linear(64, 1)
```

- **More neurons** were added to capture the more complex relationship between the value function and the state vector.
- **Number of hidden layers** were kept the same.

Training the Actor Network:

- The Adam optimizer with a learning rate of 5e-5 is used.
- Trained for 1000 epochs.
- $\pi(x) = min_u \; l(x,u) + V(f(x,u))$ used as the loss function (where $x = [p, v]$).
- The training lasted 343 seconds.

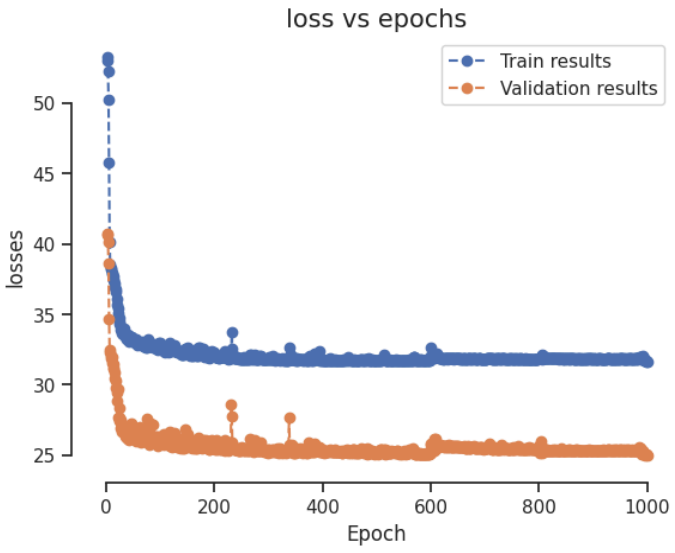

Fig 9: Training and validation losses of the Actor Network

Kemal Alp Sezer
245996

Results for 2D Integrator:

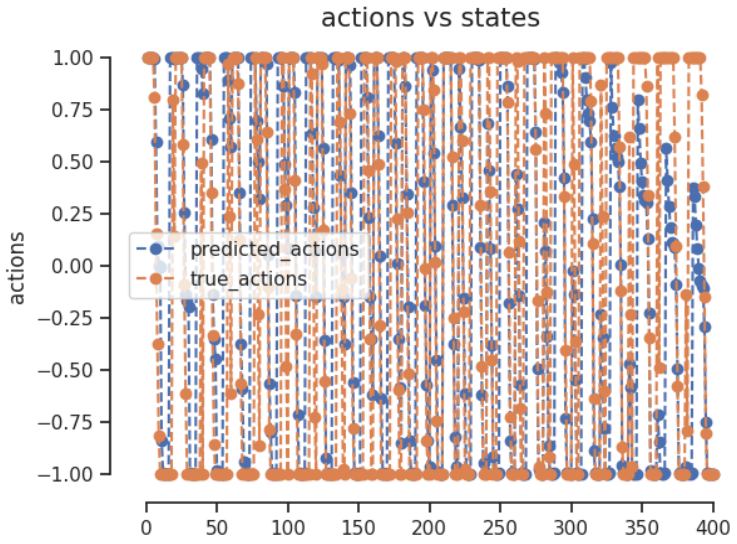

Fig 10: Another enumeration of states vs actions this time showing how well the solver's actions overlap with the actor's estimates given the state.

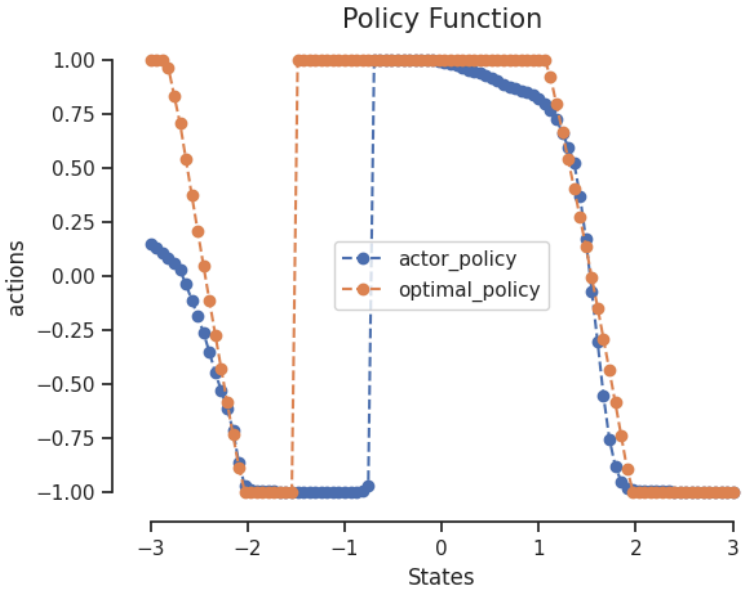

Fig 11: The policy function given a constant velocity (of 0.5 in this case) and position vs action. The actor's policy and the optimal policy found by the solver can be seen.

Kemal Alp Sezer
245996

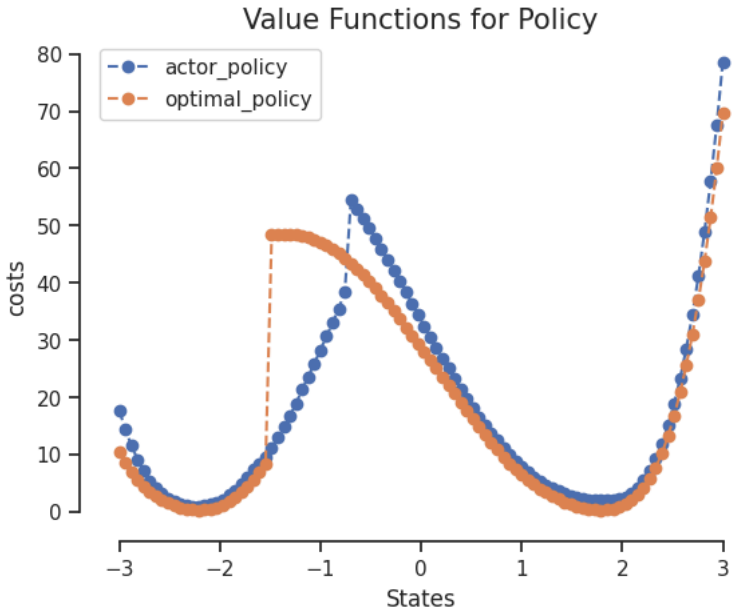

Fig 12: The optimal value function given a constant velocity (of 0.5 in this case) and position vs expected cost. The value function for the actor's policy and the optimal policy found by the solver can be seen.

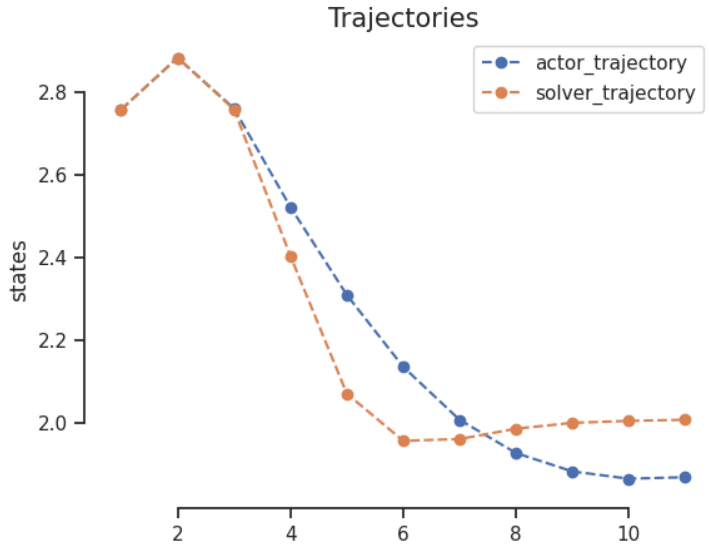

Fig 13: The position vs time step (10 time steps + the initial state) can be seen for the 2D integrator for a sample. Although there are some deviations, the trajectories approach the same position ($p_T = 2$) where the modified cost is minimized.

We generally see the tendency to approach one of the two minimas ($p_T = 2$) or ($p_T = -2$) over time which can be seen by all the results. The Actors policy sometimes

can aim to reach the minima that is further away instead of the one that is closer. This does not increase the cost very much in general but is suboptimal. The critic in general can estimate the value function accurately within the given interval even with a smaller number of points.

Conclusion:

In summary, Project C successfully demonstrates the application of actor-critic learning for control in both 1D and 2D integrator systems. The methodology effectively minimizes the cumulative running cost by leveraging neural networks to approximate the value function and the optimal policy. The 1D integrator results show how the Critic network can predict the cost associated with various initial states, and the Actor network can derive control inputs to minimize these costs. The extension to the 2D integrator system introduces additional complexity by incorporating both position and velocity into the control framework, reflecting more realistic scenarios.

The use of actor-critic learning in this project highlights the potential of this approach for solving optimal control problems, offering valuable insights into the dynamics of the system and the effectiveness of learned policies. Future work could explore further refinements to the network architectures, incorporate more complex cost functions, and apply the methodology to additional real-world systems to validate and enhance the robustness of the learned control strategies.

Discussion:

A possible improvement to the project can be optimizing the structure of the actor and critic for the running cost function provided by the assignment which is harder to fit. The number of samples can be increased so that the critic can estimate the value function better as the critic's prediction deviates from the original value function which may be causing it to see a certain state better than it actually is and we actually see a tendency for it to move to that certain spot.

Another point is on the extension of this project. The initial thought starting this project was to include a second dimension to the problem, a second position and velocity so that the agent can move in 2D but if were to consider these two dimensions independently following our current formulation there would be no need to do so as the current solution can be used separately for x,y axes if we are to restrict ax and ay within the interval [-1,1] but if we were to link the two axes by a different constraint such as $a_x^{\,2} + a_y^{\,2} = 1$, then we would have a different problem to solve. The initial problem can be extended to a different one for any dimension using this formulation while keeping in mind the exponential growth in the number samples required.