Neural Network Report: Actor-Critic Network for Cargo Lander

Group 13

1 Code Explanation

Our code implements an Actor-Critic network (ACNet) designed for reinforcement learning applications in the cargo lander environment. The neural network architecture is illustrated in the figure below.

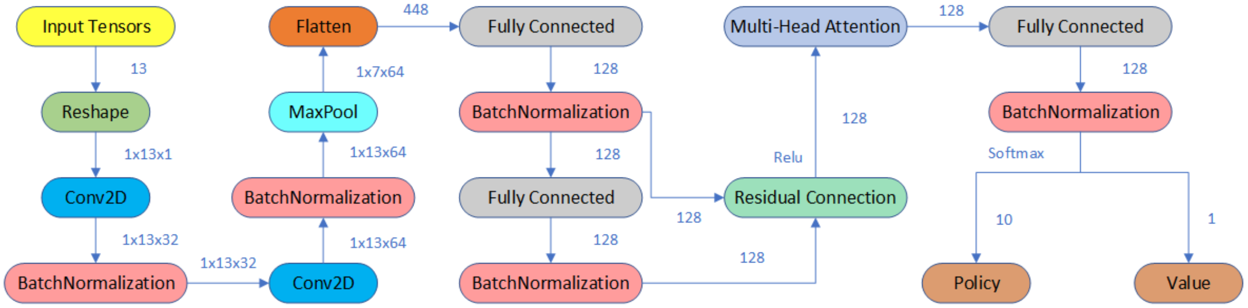


Figure 1: Neural Network Structure

1.1 Input Layer

The input data is a placeholder that receives information about the cargo lander's position, angles, linear and angular velocity in 3D space, and ground contact status. Key training parameters such as gradient clipping threshold and learning rate are also defined.

1.2 Hidden Layer

The `ACNet` class outlines the neural network architecture. Within the constructor, the input state is reshaped to accommodate the requirements of the convolutional layers. The network consists of two convolutional layers for feature extraction, each followed by batch normalization and ReLU activation functions to enhance training speed and stability. A max pooling layer follows the convolutional layers, and the extracted features are further processed through fully connected layers, incorporating residual connections to mitigate the vanishing gradient problem. The network also includes components for Proximal Policy Optimization (PPO) in the next phase, such as: Retaining old policy, Clipped policy loss, Advantage function, Entropy bonuses, Gradient clipping, and The Adam optimizer.

1.3 Output Layer

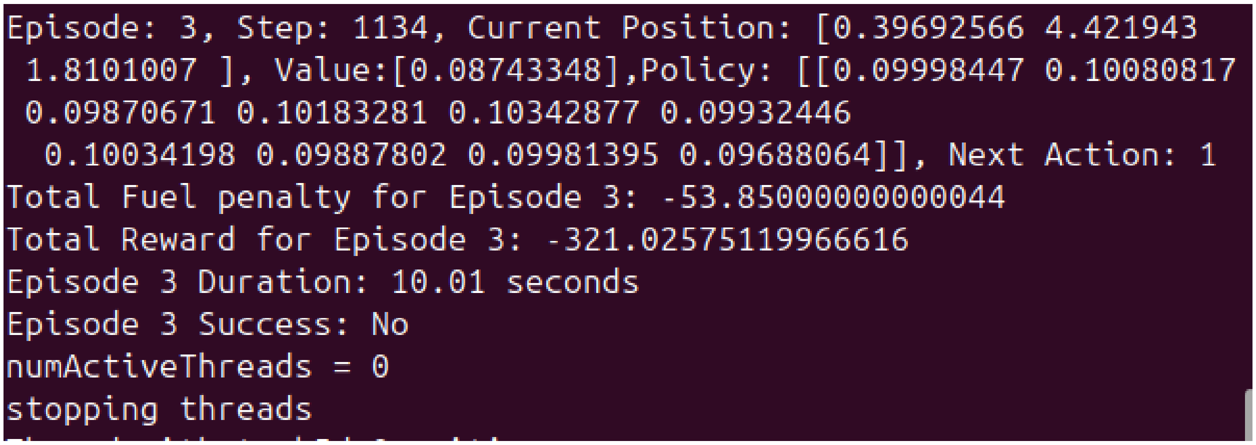
The network outputs two parts:

- The policy output: a ten-dimensional policy vector from a softmax function for action probabilities.
- The value output: expected returns in the current state.

During training, policy and value losses are calculated, and gradient clipping with the Adam optimization algorithm is used for weight updates.

1.4 Demo Code

The demo uses the pre-built environment and `ACNet` to simulate landing episodes. In each step, the network outputs a ten-dimensional policy vector and value estimate to determine the next action. The final output includes position, value estimates, policy distribution, chosen actions, fuel consumption, total reward, and landing success.



```
Episode: 3, Step: 1134, Current Position: [0.39692566 4.421943
 1.8101007 ], Value:[0.08743348],Policy: [[0.09998447 0.10080817
 0.09870671 0.10183281 0.10342877 0.09932446
  0.10034198 0.09887802 0.09981395 0.09688064]], Next Action: 1
Total Fuel penalty for Episode 3: -53.8500000000044
Total Reward for Episode 3: -321.02575119966616
Episode 3 Duration: 10.01 seconds
Episode 3 Success: No
numActiveThreads = 0
stopping threads
```

Figure 2: Demo Output

2 Why ACNet?

The Actor-Critic Network (ACNet) is a great fit for complex tasks like controlling a lunar lander because it combines action selection (actor) with action evaluation (critic) in one model. Unlike value-based methods such as DQN, which have difficulty with continuous action spaces, ACNet excels in these scenarios. It also improves stability over policy-only methods, which can have high variance and slower learning. With Proximal Policy Optimization (PPO), ACNet achieves stable training by balancing updates and using a clipped objective function, leading to faster learning and better performance. This combination is especially useful in high-dimensional environments like a lunar landing, where precise control is crucial.

3 Existing Codes/Libraries

In our project, we used TensorFlow to implement the neural network and leveraged an existing Actor-Critic model (`ACNet`) for reinforcement learning.

4 Reflections Learned

During the development of our neural network, we chose to use TensorFlow 1.x instead of TensorFlow 2.x. Because we want to make sure compatibility with some of the existing codebases and libraries that we intended to use, which were more straightforward to implement with TensorFlow 1.

In this report, we hope to illustrate the landing process more intuitively by an image capture of the pybullet simulation and a video recording 3 episodes. Diagrams explaining the neural network flow, all in latex format.