ME5418 - Machine Learning in Robotics
Group 16

Ji-Dog quadrupedal al Robots Locomotion via Reinforcement Learning Neural Network Report

Group members:

Bai Yue (ID: A0304233U)

Ji Shuchen (ID: A0305152U)

Zhang Binlin (ID: A0304090U)

October 28, 2024

1 Project Introduction

In this project, we intend to train a quadrupedal al robot to perform locomotion using reinforcement learning. In the previous report, we finished an OpenAI Gym environment. And our second step is to set up a neural network for training. In this part, we use PyTorch since it is more user-friendly and has dynamic computational graphs that are convenient for debugging. Our neural network is based on MLP and LSTM. MLP is easy to implement and capable of training the quadrupedal al robot without vision. LSTM is chosen for its ability to selectively remember or forget information and robustness in handling dynamically changing time-series data, such as the gait variations in quadrupedal robot walking. In *Ji_dog_net.py*, we define *ActorCritic* class to realize the neural network designed for the Proximal Policy Optimization(PPO) reinforcement learning algorithm. Also, we attach *ji_dog_net.ipynb* to show the training process.

2 Code Explanation

This section describes the core components and execution flow of the PPO algorithm implemented in our project:

- *Class RolloutBuffer*: This class is a storage buffer that temporarily holds important data (states, actions, log probabilities, rewards, state values, and terminal flags) generated during an episode.
- *class ActorCritic*: This is the core neural network model used in the PPO algorithm, containing both actor and critic components for a reinforcement learning agent.
- *class PPO*: This class implements the PPO algorithm, using an Actor-Critic model to update the policy in a stable manner.

2.1 Actor Network and Critic Network

2.1.1 Actor Network

The actor network is used to determine the optimal policy which is the distribution of the probability of a certain action for the current state. The goal of the actor network is to learn a policy to maximize the cumulative reward of the agent and meanwhile maintain a restricted policy update magnitude.
As Figure 1 shows, our actor Network is composed of four fully connected layers and an LSTM layer. (1)Firstly, three fully connected layers sequentially map the state vector into 128, 256, and 128 dimensions. Each layer is followed by a ReLU activation function. (2)Then, an LSTM layer receives the 128-dimensional output as input from the last fully connected layer to capture temporal dependencies. (3)Last, a final fully connected layer outputs the action. (4)Plus, a Tanh activation function is applied to constrain the action within the range of -1 and 1. (5)Additionally, the action is sampled from a multivariate normal distribution, *dist = MultivariateNormal(action_mean, cov_mat)*, created using the network's mean and covariance.

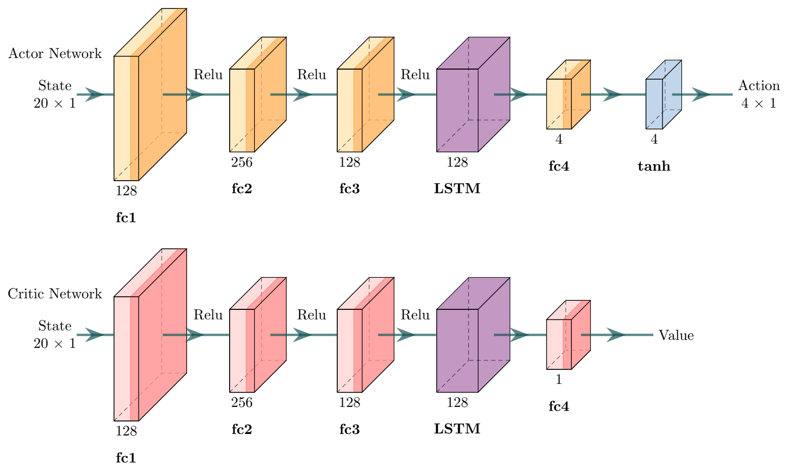

Figure 1: The Structure of the Actor-Critic Network

2.1.2 Critic Network

Similar to the actor, the critic network has three fully connected layers mapping the state input into dimensions of 128, 256, and 128, each followed by a ReLU activation. The LSTM layer, identical to the actor's LSTM, processes the 128-dimensional vector output from the fully connected layers, enabling the network to capture the temporal dependency of state values. What's more, different from the actor, the final fully connected layer produces a single scalar, representing the estimated value of the input state.

2.1.3 Key Functions

- *act()*: Selects an action based on the current state and returns the action, log probability, and updated hidden state.

- *evaluate()*: Evaluates the value of a given action, the log probability of the action, and the entropy, which are used to calculate loss and update the policy.
- *evaluate_critic()*: Estimates the value of a given state.
- *compute_loss()*: Computes the policy loss, value loss, and entropy bonus according to PPO's Clipped Objective to update the model parameters.

2.2 Network Training

First of all, we establish the simulation environment and set the state dimension, action dimension and PPO initialization parameters. Then, The agent will go through 500 episodes to learn to perform locomotion properly. At the beginning of each episode, the environment is reset to obtain the initial state, and both the actor and critic LSTM hidden states are initialized to zeros, preparing the network for a new sequence. Within each episode, a time step is limited to 100 steps. As it is shown in Figure 2, the actor network is used to choose an action based on the current state. The critic is used to evaluate the value based on the current state and action. Then, the chosen action is executed in the environment using *env.step(action)*, which returns the next state, reward, done flag. After each episode, PPO's policy gradient optimization is applied based on collected data, incrementally training the Actor-Critic neural network to improve its performance.

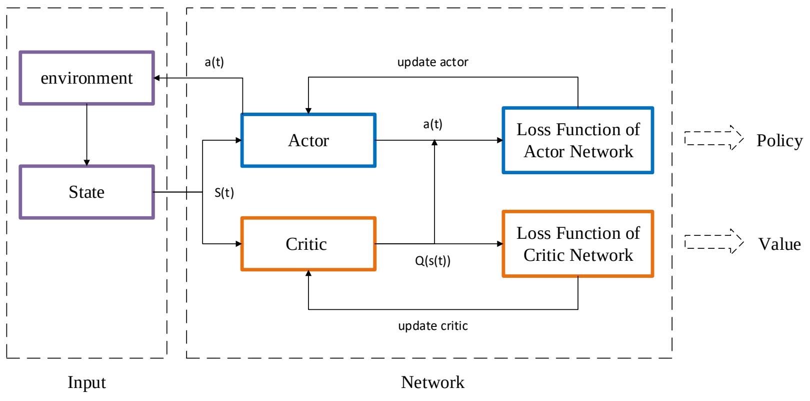

Figure 2: Network Training Based on PPO

- *Key Parameters*: *gamma*: Discount factor for rewards; *eps_clip*: Clipping parameter to control policy updates; *K_epochs*: Number of epochs to iterate over the experience buffer.
- *Key Methods*: *select_action()*: Selects an action based on the current policy and stores it in the buffer; *update()*: After each episode, it updates the actor-critic network by minimizing the PPO loss using stored experience.

3 Existing Codes/Libraries

3.1 Existing Codes

We use our customized simulation environment *Ji_dog_env* built based on Isaac Sim to interact with the simulation environment, providing the PPO with necessary interface information such as states, rewards, and actions in reinforcement learning and also enabling it to optimize the strategy with simulation feedback data.

3.2 Libraries

PyTorch: an open-source deep-learning library for building and training neural networks. In our code, its neural network module *torch.nn* and probability distribution module *torch.distributions* are used to implement the actor-critic network in PPO.

4 Reflections/Lessons Learned

4.1 Reflections

As our current task does not rely on visual input, we opt to build the Actor and Critic networks using fully connected layers and include an LSTM layer to capture temporal dependencies. This design allows the network to learn smooth, coordinated movement patterns, facilitating a natural gait for the quadrupedal robot. The use of ReLU activation functions in the intermediate layers strikes a balance between stability and gradient flow, while the Tanh activation function in the Actor output layer keeps the actions within a controllable range, which is crucial for stable robot locomotion.

4.2 Lessons Learned

For future improvements, we plan to replace the existing LSTM with the minLSTM approach discussed in the literature. This update will make the network architecture more streamlined and efficient, as minLSTM eliminates dependencies between hidden states, removes output range constraints, and ensures time-independent scaling of outputs, resulting in faster parallel training and reduced parameter requirements.