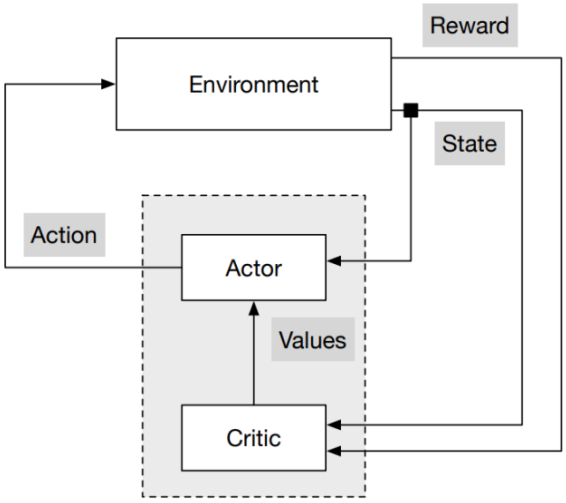
CSE 574 Reinforcement Learning
Assignment – 3 Report

Advantage Actor Critic Algorithm:

The Advantage Actor-Critic (A2C) algorithm is a reinforcement learning that combines policy gradients and value-based methods. In this method we parameterize the policy instead of relying on calculation of action-value or state-value to dictate the behavior. Actor takes the action based on the actions suggested by the critic.



Actor-Crtic Architecture

Advantage Actor-Critic model maintains two neural networks:

- Actor Network: The actor network takes in the current state as input and outputs a probability distribution over possible action. It learns to select actions that maximize expected rewards.
- Critic Network: The critic network estimates the value function, which represents the expected cumulative reward from a given state. The value function in the case of A2C model is an advantage function defined by *A(s, a)*. The network learns to evaluate how good or bad the current state is and directs the actor network in a way to optimize its actions. The advantage function is an addition to the vanilla Actor-Crtic model which helps reducing the variance of the policy gradient.

A2C updates the actor network using policy gradients, which guides the network to increase the probability of actions that lead to higher rewards. The algorithm updates the critic network by minimizing the mean squared error between the predicted value and the observed (or estimated) value of states. Both networks are updated using stochastic gradient ascent.

Actor-Critic vs Value based approximation:

In value-based approximation techniques like DQN we rely on approximating the state-value or action value functions by parameterizing the value function, that is, we try to find the value function based on a neural network. They are effective in environments with discrete action spaces.

However, in Actor Critic models we directly estimate the policy by parameterizing it. We try to obtain the policy without relying on value-function to direct the actions of our agent. They combine the benefits of both value-function approximation and policy gradient to get to the solution.

Part – 1 Implementing A2C model to solve Grid World environment:

Grid-World Environment

Description: The environment is a deterministic grid world based on Door dash delivery agent who must deliver food to the customer. On his way the agent is met with obstacles and rewards. The objective is to deliver the food to the customer in the least possible time while maximizing rewards.

Actions available: The agent can move in four directions; Up, Down, Left and Right

Rewards available: There are several types of rewards/ penalties. The below picture gives an idea of the reward function used in the environment.

```
-------------------------------
| 0.0 | |0.01 | |0.05 | | 0.1 |
-------------------------------
| O=o | |0.02 | |0.15 | | 0.2 |
-------------------------------
|-0.01| | 0.1 | | 0.2 | | 0.3 |
-------------------------------
|0.01 | |0.15 | | 0.3 | | 1.0 |
-------------------------------
```

Visualization of grid world

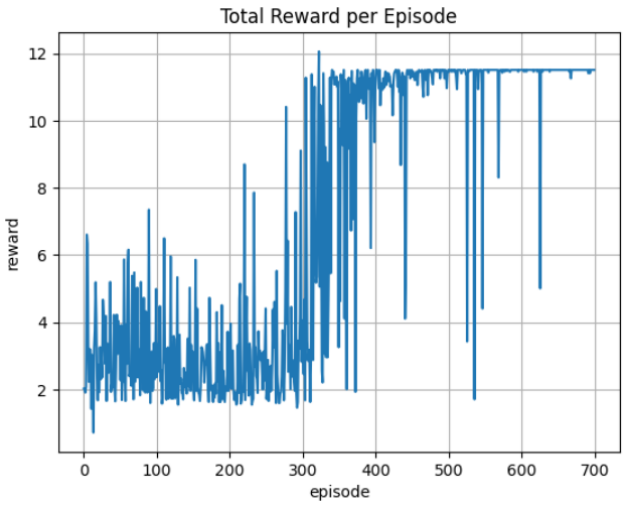
The above image is a simple visualization of an agent in the grid environment. Here the agent is represented by a car symbol("O=o"). The rewards/ penalties are represented by constant values. The goal state is at the bottom right corner of the grid.

It is worth noting that the environment is modelled in a way that rewards stay alive at their respective positions even if the agent attains the reward at some point during the episode.

States available: There are 16 states in which the agent can be in.

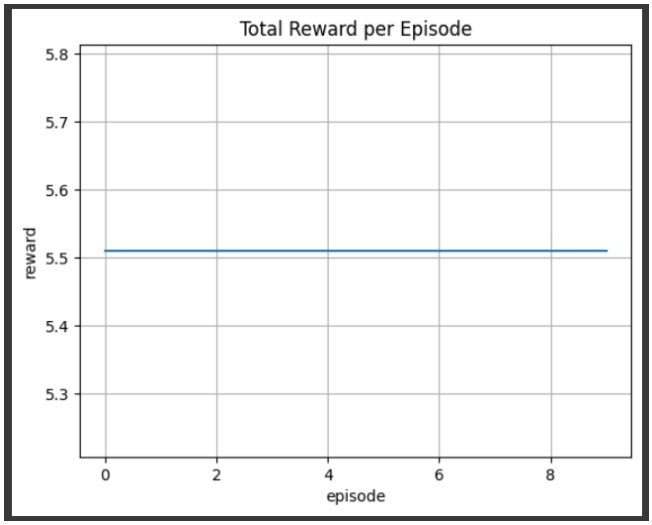
Termination: The episode is terminated if the agent reaches the goal state, or it exceeds the maximum allowed time steps.

Results: With a discount factor of 0.99 and default learning rates of 0.001 for both actor and critic networks, the model reaches a sub-optimal solution. The model is run for 700 episodes at a maximum of 40 timesteps per episode.



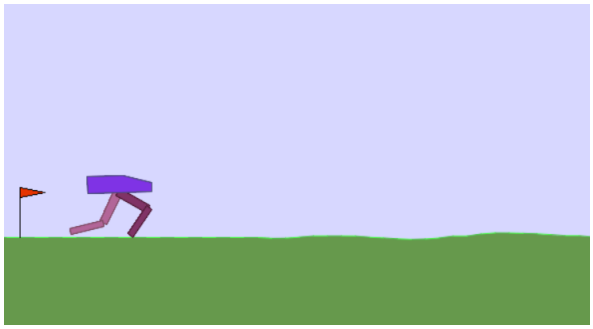
Rewards per episode during training

Between 300th and 400th episodes, the model reaches a good level of reward rate, rapidly learning the optimal policy. After which, it stays the same on average.



Rewards per episode during evaluation

Part – 2.1 Solving Bipedal Walker Environment using A2C model:



Rendering of bipolar walker

Description: The environment consists of a 4-joint walker robot. In this version the terrain is slightly uneven. To solve the normal version, the agent needs to get 300 points in 1600 timesteps.

It is worth noting that this environment has a continuous action space. The action is a vector of 4 values, each between -1 and 1.

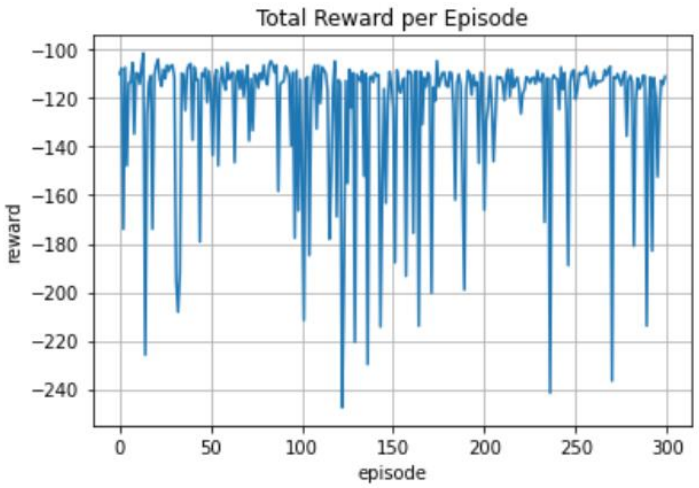
Action Space: Actions are motor speed values in the [-1, 1] range for each of the 4 joints at both hips and knees.

Observation Space: State consists of hull angle speed, angular velocity, horizontal speed, vertical speed, position of joints and joints angular speed, legs contact with ground, and 10 lidar rangefinder measurements.

Rewards: Reward is given for moving forward, totaling 300+ points up to the far end. If the robot falls, it obtains a reward of -100. Applying motor torque costs a small number of points.

Termination: The episode will terminate if the hull gets in contact with the ground or if the walker exceeds the right end of the terrain length.

Results:

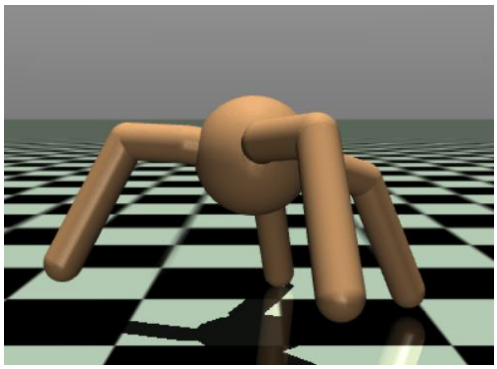


Model rewards during training.

The model is trained for 300 episodes, each with a maximum of 1600 timesteps. The discount factor is 0.99 and we use Adam optimizer with a default learning rate of 0.001. The agent fails to learn the optimum policy and struggles to perform better over time.

Model rewards during evaluation

Part – 2.2 Solving MuJoCo Ant Environment using A2C model:



Rendering of MuJoco Ant

Description

The agent is an ant which is modeled as a 3D robot consisting of one torso (free rotational body) with four legs attached to it with each leg having two body parts. The goal is to coordinate the four legs to move in the forward (right) direction by applying torques on the eight hinges connecting the two body parts of each leg and the torso (nine body parts and eight hinges).

Additionally, the environment has a continuous action space and hence we would have to change the architecture of our neural network to handle continuous action space.

Action Space: The action space is of type Box(-1, 1, (8,), float32). An action represents the torques applied at the hinge joints.

Observation Space: Observations consist of positional values of different body parts of the ant, followed by the velocities of those individual parts (their derivatives) with all the positions ordered before all the velocities. It is defined as Box(-Inf, Inf, (27,), float64)

Rewards:

The reward consists of three parts:

- Healthy_reward: Every timestep that the ant is healthy (see definition in section "Episode Termination"), it gets a reward of fixed value healthy_reward
- forward_reward: A reward for moving forward which is measured as (x-coordinate before action - x-coordinate after action)/dt. dt is the time between actions and is dependent on the frame_skip parameter (default is 5), where the frametime is 0.01 -

making the default dt = 5 * 0.01 = 0.05. This reward would be positive if the ant moves forward (in positive x direction).

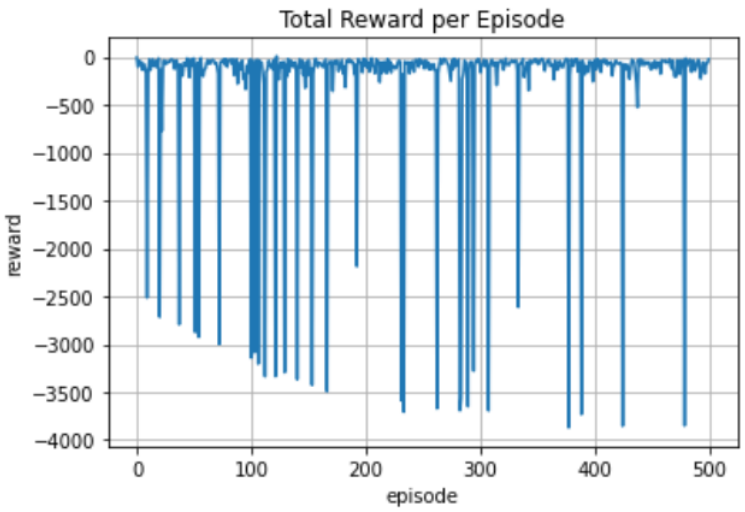
- ctrl_cost: A negative reward for penalising the ant if it takes actions that are too large. It is measured as ctrl_cost_weight * sum(action2) where ctr_cost_weight is a parameter set for the control and has a default value of 0.5.
- contact_cost: A negative reward for penalising the ant if the external contact force is too large. It is calculated contact_cost_weight * sum(clip(external contact force to contact_force_range)2).
- The total reward returned is reward = healthy_reward + forward_reward - ctrl_cost.

Termination: The episode terminates when the ant is unhealthy. The ant is said to be unhealthy if any of the following happens:

- Any of the state space values is no longer finite.
- The z-coordinate of the torso is not in the closed interval given by healthy_z_range (defaults to [0.2, 1.0])

Truncation: The episode duration reaches a 1000 timesteps

Results:



Model performance during training.

The model is trained for 500 episodes with a max timesteps of 1600 and a discount factor 0.99. The model fails to learn the optimal path and does not improve as training progresses.

Model performance during evaluation.

References:

1. https://www.tensorflow.org/api_docs/python/tf
2. https://numpy.org/doc/stable/user/quickstart.html
3. https://keras.io/api/
4. Lecture Slides on Piazza
5. https://datascience.stackexchange.com/questions/49625/ppo-a2c-for-continuous-action-spaces-math-and-code
6. https://www.youtube.com/watch?v=kWHSH2HgbNQ