Principles of Programming Languages: Baseline Milestone - 1

Topic: Domain Specific Language for Reinforcement Learning based Agents in Games

Group 5

Soham Chitnis 2020A7PS1723G	Rijul Radhu 2020A7PS1430G	Aakash Tiwari 2020A7PS0981G
Dwij Dixit 2020A7PS2129G	Naishadh Sheth 2020A7PS0148G	Atharva Limaye 2020A7PS1721G

Domain Description -

The project domain focuses on making the building of RL based multiagent games easier. Agents will be trained so that those agents can be used to play the games against users. DSL's built for this domain come under the category of **Domain-specific entertainment languages.** RL algorithms make an agent learn to optimize its behavior by receiving rewards from a surrounding interactive environment. Unfortunately, current state of the art languages are quite technically complicated for game developers who work in the RL based game development domain. This project aims to solve this issue by giving a crisp programming language which can ease the development process by giving the developer features of object oriented and functional programming concepts.

Setup -

System Requirements:

- Linux
- Python 3.10.4
- Pip 22.02

To install the dependencies
$ ***pip install -r requirements.txt***

NOTE: requirements.txt is attached

Functionality of Components -

The environment of SuperMarioBros requires wrappers like SkipFrame, GrayScaleObservation, ResizeObservation, and FrameStack for rendering the environment. The class Mario defines utility for training, saving checkpoints, methods to select actions, and learning the model. MetricLogger is used for logging the loss, Q-value, Rewards, and plotting. The class MarioNet defines the Convolutional Neural Network for training the model. In the main, we train the model in an online fashion with a defined number of episodes.

The environment of SuperMarioBros requires wrappers like SkipFrame, GrayScaleObservation, ResizeObservation, and FrameStack for rendering the environment. The class Mario defines utility for training, saving checkpoints, methods to select actions, and learning the model. MetricLogger is used for logging the loss, Q-value, Rewards, and plotting.

The class MarioNet defines the Convolutional Neural Network for training the model. In the main, we train the model in an online fashion with a defined number of episodes. The model learns and remembers the action taken at a given state this helps the model take the same action whenever it encounters it again this is done using mario.cache(). Env.step() takes the selected action to give the next state.

- CNN architecture has 3 convolution layers with 2 Fully-connected layers.
- Activation function uses ReLU. The optimizer used Adam.
- The loss function used Smooth L1 Loss for training.
- The size of the frame is 84 x 84.The learning rate is 0.00025.
- Currently, the number of episodes is set to 30.

Compilation and Running -

To run the baseline,
$ *python3 baseline_group_5.py*

The game GUI opens up and runs for the given number of episodes.

Domain Understanding -

In general, games and simulations may have a significant impact on our experiences. We can test innovative methods for conducting research, delivering instructions, and instructing and educating people in virtual environments. By providing a computer language designed exclusively for game production, the time and money required for game development can be saved. Game developers can then concentrate on investigating and developing cutting-edge designs for making novel games, instructing co-developers, and other uses by avoiding to specify details irrelevant to the game logic but important to the correct function of the chosen tool. Since its introduction, deep reinforcement learning (DRL) has made significant progress. In general, DRL agents do actions based on deep neural network-based policies after receiving high-dimensional inputs at each step. With an end-to-end approach, this learning mechanism adjusts the policy to maximize return.

Baseline issues -

1. **Long code length -** The code for the RL tends to get very long due to several classes and functions for image preprocessing, utilities for training, making model, rendering, wrappers required for gym and other standard inclusions, environment and the main Mario class specifying the environment of Mario.
    a. The large number of methods and variables in the python file introduces several mutable variables, whose unintended editing consequences can cause the code to run into bugs.
    b. Debugging with this large code length can cause several delays due to size and technical complexity of the involved code.

2. **Lack of cognitive ease -** The length of the code, and involved technicalities of frameworks utilized causes the code to be fairly incomprehensible to write, and even more so to read in an efficient manner. This not only causes difficulty for debugging in case of errors, but also causes difficulty in while extending or editing to code for trial of new added or substituted policies.

3. **Environment sensitivity -** The prerequisite tools and libraries are sensitive with respect to their implementations and environment, which causes added burden over the game developer.

4. **Understanding of pytorch and tensorflow frameworks** - Implementation of RL based agents requires extensive understanding of pytorch and tensorflow frameworks - the standard today for implementation of RL algorithms. Also, the developer must be acquainted with tools for pre-processing of images to be fed to the algorithm.

Conclusion -

As is evident from the above-listed issues, implementation of RL-based policy agents for the game is a complicated process under the current state of implementation tools, and libraries, and there is a requirement of abstraction of these algorithms and implementations to the game developers - who are not sensitive to the freedom given by the current frameworks for modifications of several aspects of the RL components, namely the networks, preprocessor, and testing, and reward logical module. These features, if abstracted out front the current implementations can cause a large saving in terms of effort, time and economic resources involved in development of these games. This also allows the game developers the ability to test and implement several agents.

References -

1. *Hado van Hasselt, Arthur Guez, & David Silver. (2016). Deep reinforcement learning with double Q-Learning. National Conference on Artificial Intelligence, 30(1), 2094–2100. http://lib-arxiv-008.serverfarm.cornell.edu/abs/1509.06461*
2. *Train a Mario-playing RL Agent — PyTorch Tutorials 1.12.1+cu102 documentation*. (n.d.). Retrieved October 27, 2022, from https://pytorch.org/tutorials/intermediate/mario_rl_tutorial.html
3. Aditya Grover, Maruan Al-Shedivat, Jayesh K. Gupta, Yura Burda, & Harrison Edwards. (2018). Learning Policy Representations in Multiagent Systems. *International Conference on Machine Learning*, 1802–1811. http://proceedings.mlr.press/v80/grover18a/grover18a.pdf