Intelligent Systems

Sonic The Hedgehog 2 Agent with Proximal Policy Optimization

Profesor:

Dr. Benjamin Valdes Aguirre

Carlos Alberto Becerra Ortiz A01153661

November 17, 2022

1. Introduction

This report describes the training of a deep reinforcement learning agent to play a retro game. The retro game in particular is “Sonic the Hedgehog 2”. Training and testing of the agent will be done on the first level of the game, “Emerald Hill Zone Act 1”. The reinforcement learning algorithm used is Proximal Policy Optimization (PPO), which strikes a balance between ease of implementation, sample complexity trying to compute an update each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small. The game environment is made using Open Ai Gym Retro.

2. Sonic the Hedgehog 2 Video Game

Sonic the Hedgehog 2(SH2) is a fast, momentum-based platformer game, meaning Sonic’s movement is fast and many of the level design tropes make use of Sonic’s speed to build up momentum. Sonic can jump, spin attack, spin dash and collect rings in order to get to the goal of each level.

3. Task Definition

In reinforcement learning, the agent interacts with the environment by taking an action based on the current state of the agent, and then transitions to a new state and gets a reward from the environment. The state in this case is a frame of the game in each timestep. Although, each state is made out of N stack of consecutive game’s frames, rather than just a single frame. This stack of frames will be the input in the PPO algorithm.

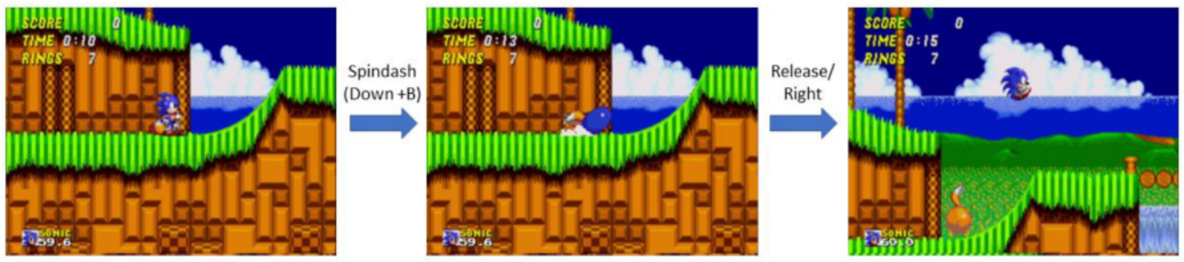


Figure 1. An example of state, action sequence when playing SH2

As for the agent, it will choose an action from the list of possible actions in each state the agent is currently in at each timestep. By default, because of SH2 will be used in a SEGA Genesis emulator, Open AI Gym Retro provides 12 possible actions to choose from, each representing the buttons on SEGA Genesis controller, which are:

A, B, C, Y, X, Z, MODE, START, UP, DOWN, LEFT, RIGHT. The number of possible actions is modified to fit our game better, because some buttons result in the same action for our game. The actions are redefined to 8 possible actions for our case, which are:

- **NO OPERATION:** Sonic stands still
- **LEFT:** Sonic moves to the left
- **RIGHT:** Sonic moves to the right
- **LEFT + DOWN:** Sonic moves to the left and rolls into a ball
- **RIGHT + DOWN:** Sonic moves to the right and rolls into a ball
- **DOWN:** Sonic ducks or rolls into a ball
- **DOWN + B:** Sonic charges his spin-dash, when these buttons are released, Sonic will launch to the direction he is facing in ball form
- **B:** Sonic jumps

The reward, also referred to as "score", is obtained by increasing the x position of the agent. This translates to as the further the agent goes to the right, the higher the reward score is. Reaching the goal will give the agent bonus points up to 1000 depending on how fast the agent finishes the stage.

4. Implementation

In this section, explanations about the implementation settings, input data and training will be provided.

Input Processing

The input data, state, which are represented by a stack of frames are preprocessed first before getting fed into the network. Preprocessing is done to reduce complexity of data as some information in the data can be removed and simplified to lessen the computation burden and fasten the training process. The preprocessing includes:

- Converting frames to grayscale, as the color channels don't really matter for our case.
- Resizing the frames to 84x84.
- Normalizing the pixel values to a range of 0 to 1
- Stacking consecutive frames together, to give more sense and context to the network.

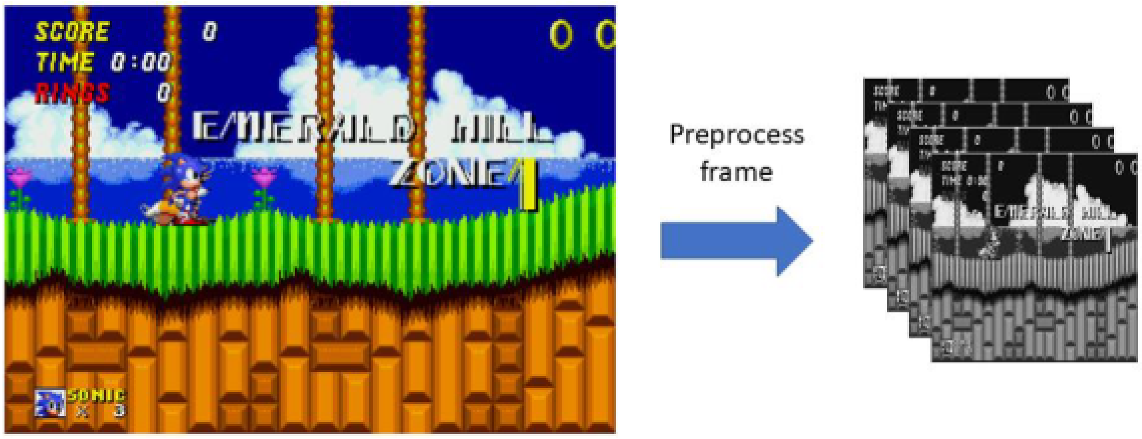
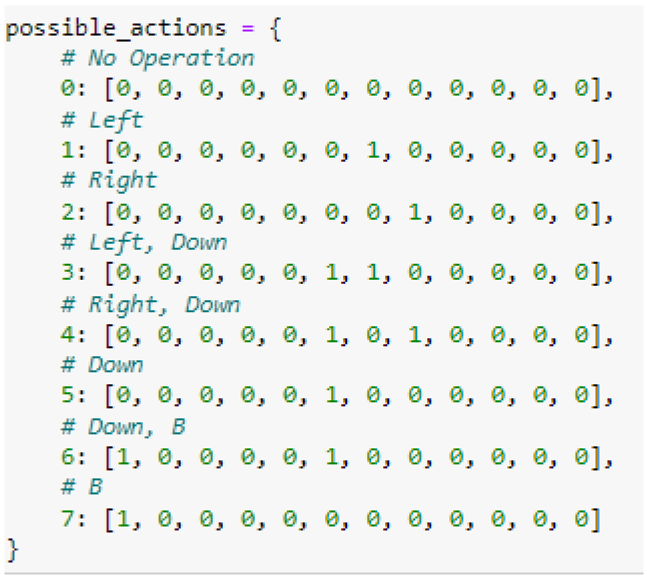


Figure 2. The frame is preprocessed before being fed into the network

Limit Agent Actions

The multi binary action space was modified to only use the actions that best suits the game



```
possible_actions = {
    # No Operation
    0: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    # Left
    1: [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    # Right
    2: [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
    # Left, Down
    3: [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
    # Right, Down
    4: [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
    # Down
    5: [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    # Down, B
    6: [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    # B
    7: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
}
```

Define Hyperparameters

We define the hyper parameters that will be used for the data network

Hyperparameter	Value
n_steps	4096 - 8192

gamma	0.8 - 0.9999
learning_rate	0.00001 - 0.0001
clip_range	0.1-0.4
gae_lambda	0.8-0.99

Experiment

In order to train our agent we will perform an experiment. It will start by creating and evaluating a policy. We will start by doing 10 trials in which every trial will create its own neural network using Convolutional Neural Network (CNN), learn for a total of 100000 timesteps and will be evaluated by 10 episodes. We can't parallelize this task since retro gym struggles with parallelization.

After the trials are finished we will use the resulting model with the best score. This experiment will be performed 2 times.

Experiment A Model Parameters	**Experiment B Model Data Parameters**
'n_steps': 7997, 'gamma': 0.8663323510397305, 'learning_rate': 3.2838836969752185e-05, 'clip_range': 0.35180745648710154, 'gae_lambda': 0.9313547419561927 **score: 6719**	'n_steps': 2922, 'gamma': 0.9399919339167541, 'learning_rate': 5e-7, 'clip_range': 0.1998614906944592, 'gae_lambda': 0.8541019293324146 **score: 7303**

The produced model will be used for the training and every 10,000 time steps the model will be updated and start from there again to create an updated model.
Once this process is finished the last resulting model in each experiment will be evaluated in order to compare the results.

Experiment A	**Experiment B**
Mean Reward: 1709.0	Mean Reward: 4810.0

As we can see the experiment performs better than experiment B. But once the resulting model of experiment B the actions of the agent experiment A. Model B makes Sonic move further but not that much mainly because the model needs more training in order to make significant improvements. But due to time and technical issues the experiments were much smaller than an ideal training.

5. Conclusions

In order to get better and more representative results in future experiments, amore exhaustive training is needed since the ones that were performed used smaller paramates in the timesteps, episodes and trials from an ideal size.