PING PONG USING REINFORCEMENT LEARNING

Annant Maheshwari - 201IT109
Dept. of Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
Email : annant.201it109@nitk.edu.in

Harish Gumnur - 201IT223
Dept. of Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
Email : gumnurharish.201it223@nitk.edu.in

Malvika Koushik - 201IT134
Dept. of Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
Email : malvikakoushik.201it134@nitk.edu.in

Poorna Hegde - 201IT240
Dept. of Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
Email : poornahegde.201it240@nitk.edu.in

***Abstract*—Being one of the earliest computer games, the Pong game is well-known for its simplicity, which makes it suitable for becoming one of the very first problems in Artificial Intelligence and Machine Learning. The goal is to create a self-playing agent that can compete against humans. In the past there have been introduced various Reinforcement Learning approaches to solve this problem, notably one that uses Deep Q-networks as a policy network. In this paper, we propose a reinforcement learning approach using Convolutional Neural Networks(CNNs) that analyse each frame to retrieve state information. Our AI agent plays against a regularly programmed bot.**

I. Introduction

One of the three fundamental machine learning domains is reinforcement learning (RL), along with supervised and unsupervised learning. In RL, a robot or artificial agent learns by trial and error what the best course of action is in a given situation. Reinforcement learning has recently found remarkable success in a variety of video and board games.

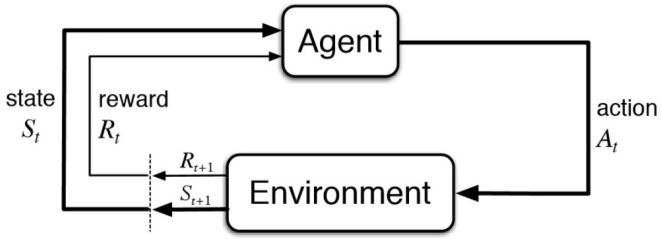

Fig. 1. Reinforcement Learning Paradigm

Source: TowardsDataScience

The agent, the environment, and the agent's state within the environment are the three main components that make up an RL problem. The agent solves the RL challenge by making intelligent decisions. It engages in environmental interaction by taking actions and then reaping the benefits of those actions. The environment serves as a visual representation of the issue that the agent must deal with and resolve. It could be a virtual environment or one that is real. The agent's and its environment's present state at a certain time-step is known as the state.

The time-step increases as the agent changes states in the environment each time it does an action. The RL agent will have finished an episode once it has performed enough actions in its environment, where an episode comprises of 3 intermediate states (i.e., the states between the initial and terminal states). The agent's main objective is to maximise the benefits it receives in every given episode. To accomplish this, it goes through a process known as training where it completes numerous episodes in its environment to learn the optimum strategy (or optimal policy) to use.

II. Problem Statement

The objective of this project is to train an AI agent to play the famous game - Pong against a regular programmed bot in a reinforcement learning environment with a graphical user interface.

III. Literature Survey

[1]The paper introduces a sample-efficient RL technique with states such as position, velocity, spin, orientations and velocity. It also uses an actor-critic approach for its policy-gradient algorithm. The paper goes through the process by setting an epsilon value which is the trade-off between "observe" and "take-action". Often human feedback is taken to shape the reward function according of the task.

A Q-learning version is used to train the network, and stochastic gradient descent is used to update the weights. Correlated data and non-stationary distributions might be problematic, thus they employ system for replaying experiences that samples previous transitions at random

distribution trainings across many previous behaviours in a smooth manner. Naturally during the start of the game the agent observes or explores more than taking action. As the game progresses, this behaviour changes.

It's observed that the patterns for the agent to learn takes quite some time. The agent would thoroughly need to know the different states before taking action. This thereby increasing training time for the same.

This research demonstrates how these difficulties can be solved by a convolutional neural network to learn effective control policies from raw video input in challenging Reinforcement Learning situations. The model is a convolutional neural network trained using a variation of Q-learning, using raw pixels as its input and a value function that forecasts rewards as its output. With no modifications to the architecture or learning process, we apply our method to seven Atari 2600 games from the Arcade Learning Environment. On six of the games, we discover that it performs better than any prior methods.

IV. Methodology

Pong is a great illustration of a straightforward real-world problem. Our goal is to play as one of the paddles in the ATARI 2600 version while the other is controlled by a regular program, and the goal is to bounce the ball past the other player. The game's basic mechanics are as follows: after receiving a grayscale image frame (a 48x48x1 array with numbers 0 to 255 representing pixel values), we choose whether to move the paddle upward or downward (i.e. a binary choice). Every time we make a decision, the game simulator puts our option into action and awards us a score:

- +1 if we successfully evaded the opponent
- -1 if we missed the ball
- 0 otherwise.

Naturally, moving the paddle is what we want to do in order to maximise our rewards.

The frames are parameterised by using various characteristics of the paddles and the ball. Our agent paddle is denoted by Paddle-1, and the regular programmed bot paddle is denoted by Paddle-2. These parameters are:

- Paddle1YPos
- Paddle2YPos
- ballXpos
- ballYpos
- ballXDirection
- ballYDirection

The bot that plays accurately is basically programmed to follow the ball to its full extent at all times, keeping in mind that the positions are a bit varied as they have different sizes. The paddle moves up or down depending on whether the ball is in the lower or upper half of the window screen. We also make sure that the paddle doesn't move out of the screen, by checking its Y-position with the window height.

Next, we have our agent paddle, that's also characterised by its Y-position, and a speed at which it moves up or down. This paddle's movements are based on the action that is got as a result of the neural network that we have built, and explained further. We also need to make sure that paddle stays on the screen, and its direction is changed when it hits the top or the bottom of the screen.

Finally, we have the ping pong ball that the paddles are playing with. This is characterized by its X and Y positions, directions and speeds. The ball is updated whenever there's a collision with the learning agent paddle, and the score is increased. If the agent misses the ball, the ball goes past it and hits the screen. Again, the direction is changed but with a negative/null score. On the opposite side, if the bot somehow misses the ball, the agent wins that episode and the score is updated accordingly. Otherwise, the score stays the same and the ball is sent back to the agent paddle.

We will first establish the policy network that will be used to implement our player (or "agent"). This network will evaluate the situation and make a decision regarding what to do (move up or down). Our preferred straightforward computing unit will be a two-layer neural network that accepts the original image pixels—2304 total numbers (48*48*1)—and outputs a single number indicating the likelihood of going up. It should be noted that we only produce a probability of rising up because this is normal practise for stochastic policies. We will sample from this distribution each cycle to determine the actual relocation.

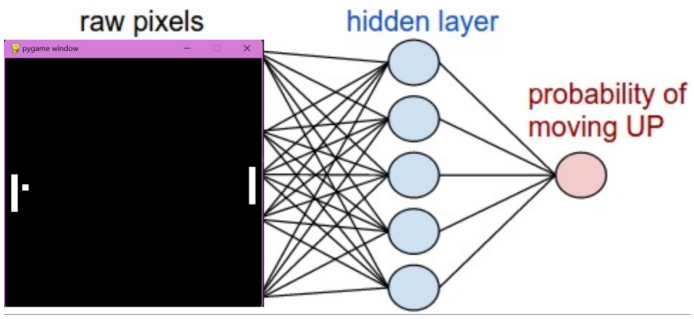

Fig. 2. Model Layout by taking in the pixels of Our Frame

We employ the final sigmoid nonlinearity, which reduces the output probability to the interval [0,1]. If the ball is at the top and our paddle is in the middle, for example, the neurons in the hidden layer with weights placed along the rows may detect these situations, and the weights in the next array can then determine whether we should be going up or down in each circumstance. The player will naturally begin to spasm as soon as the two-dimensional arrays' initial random values

are encountered.

Preprocessing is carried out in a little amount. The policy network should ideally get at least 2 frames so that it can detect movements. We will send the difference between the frames to the network to simplify things a little (i.e. subtraction of current and last frame).

We have a stochastic policy that samples actions and then encourages future actions that ultimately result in positive results while discouraging future actions that ultimately result in negative outcomes. Also, if we eventually win the game, the award does not even have to be +1 or -1. It might be a subjective gauge for some potential quality.

We'll play 100 games of Pong and then setup the policy network. Assuming that each game consists of 200 frames, this means that we have made 20,000 decisions about whether to move up or down. For each of these decisions, we know the parameter gradient, which indicates how we should adjust the parameters to promote that decision in that state in the future. Now all that's left to do is assign a grade of good or terrible to each decision we've made.

As an example, think about how a person might learn to play Pong. You demonstrate the game to them, explain that you are in charge of a paddle that you can move up and down and that your goal is to bounce the ball past the other player, who is an AI, and you're good to go. Some of the differences are:

In real-world situations, the goal is typically communicated in some way (such as the English used above), but in a typical RL scenario, you assume an arbitrary reward function that you must learn through interactions with the environment. It can be argued that if a human entered the game of Pong without having any prior knowledge of the reward function—indeed, especially if the reward function was some static but random function—the human would have a difficult time figuring out what to do, whereas Policy Gradients would be unconcerned and probably perform much better. In a similar vein, whereas humans would probably fail if we randomly permuted the pixels in the frames, our Policy Gradient method could not even detect the change.

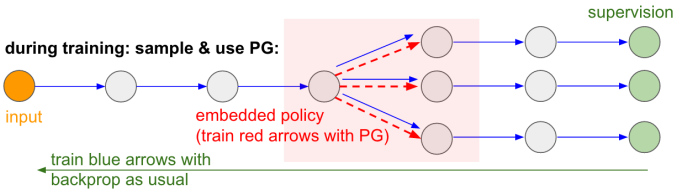

Fig. 3. Model Layout with the Neural Network

For Policy Gradients to finally and gradually change the policy parameters in favour of repeated moves that provide high rewards, they must really experience a positive reward and experience it frequently. Without actually going through the gratifying or unrewarding transition, humans can use our abstract model to predict what is likely to be rewarding.

In our project, we use PyGame which is a free Python toolkit that enables the development of games that can be based on reinforcement learning algorithms as wells by offering a standardised API for interacting with various learning environments and bulding the graphics for it to display on a desktop window. It helps visualise the workings of the RL model, which otherwise would be visible only on text.

We use multiple draw and shape functions that replicate the two paddles and the ball on a screen, and keep drawing at every frame by taking the various positions into considerations. This is done for every frame, and the frames are updated very frequently, to make it look as smooth as possible. The position graphics are updated every time we make a change or an action, which reflects real-time movement.

The ball and the paddles are considered to be rectangles on the screen, and are projected onto the screen accordingly. PyGame provides a way to update these figures on the screen by taking in every dimension of the geometric figures. The ball's size is much smaller than the paddle, and hence can be assumed to be a point-object relatively for the paddle to hit.

V. Conclusion

On an HP Laptop 14S Inter Core i3 10th Gen, the agent who played this game was trained for 10,000 episodes over the course of about 1 hour. We observed a gradual increase in the responsiveness of our paddle, although it still stutters at times. The agent is effectively able to learn from past frames, and takes appropriate action.

The trained AI Agent would win each episode by a margin of 21 points to 18.5 when compared to the untrained AI Agent, whose running mean score per episode over the previous 100 episodes was 2.5 at the time we stopped training.

The running mean score per episode reached 5 after training for an additional 5000 episodes after suspending the training script for about 1 hours; hence, the trained AI Agent would triumph in each episode by a margin of 21 to 16. With more episodes, there are unmistakably smaller and noisier returns. By slowing down learning, the noise could be decreased at the expense of training time.

Fig-4 shows a frame of size 400x400 pixels in the game. In this, we have our agent paddle on the left, the ball in the

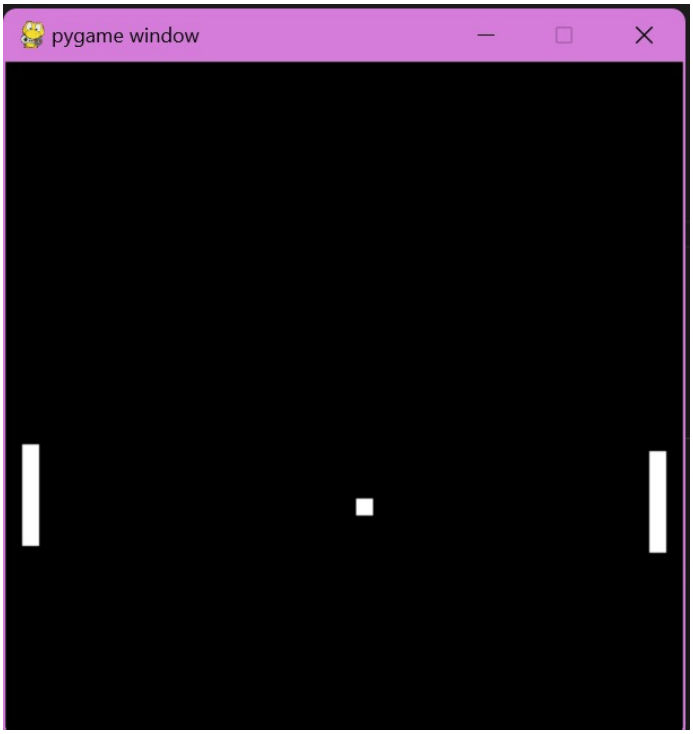

Fig. 4. Snapshot of the Pong game in motion

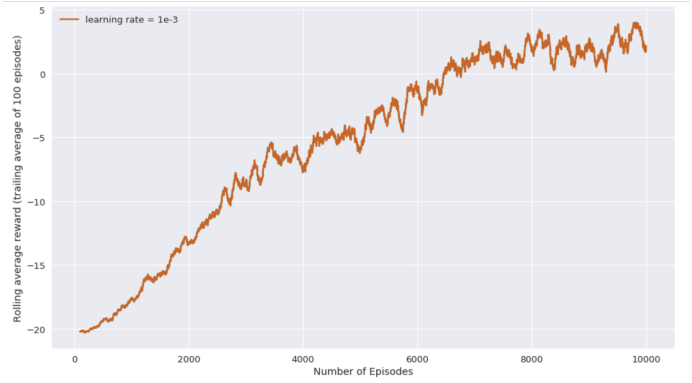

Fig. 5. Average reward per episode for the agent

centre and the regular programmed bot paddle on the right. The two paddles are constricted to move only in the y-axis, i.e. either up or down, whereas the ball can move in both X and Y directions, which are denoted by their respective quadrants. If the ball hits either of the two ends, its direction is reversed. However, the ball is allowed to hit the top and the bottom edge, and can bounce off at an angle.

Fig-5 shows a graph for the average reward got per episode of the game for the agent. The learning rate is plotted. As we can see, the agent takes thousands of episodes to start scoring for himself. The learning rate starts off smooth, and then becomes little erratic.

REFERENCES

[1] J. Tebbe, L. Krauch, Y. Gao and A. Zell, "Sample-efficient Reinforcement Learning in Robotic Table Tennis," 2021 IEEE International Conference on Robotics and Automation (ICRA), 2021, pp. 4171-4178, doi: 10.1109/ICRA48506.2021.9560764.

[2] Playing Pong using Q-Learning West Chester University

[3] Learning to Play Pong using Policy Gradient Learning MIWAI 2017 https://doi.org/10.48550/arXiv.1807.08452

[4] S. Nagendra, N. Podila, R. Ugarakhod and K. George, "Comparison of reinforcement learning algorithms applied to the cart-pole problem," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017, pp. 26-32, doi: 10.1109/ICACCI.2017.8125811.