Mastering the game of Pong using ML Agents and Self-play

https://github.com/TeamDebug/Pong

Abstract:

Manufactured by **Atari** in 1972, Pong is one of the simple classic video games that fit well enough to become one of the earliest problems in the fields of **Artificial Intelligence.** While **Reinforcement Learning** was booming, various machine-learning approaches have been introduced for the AI agents to take the optimal actions in future by attempting different allowed actions in different states and use the collection of experiences to observe the outcome of those actions. For a game like Pong that can be considered Markov Decision Process (**MDP**) model, it is explicit for an AI agent to master this game via Policy Gradient (**PG**) methods.

Objective:

Our objective is to build a Pong AI that is trained completely using self-play. The AI will play with itself until it masters the game. The AI would master Pong to an extent that it would be near impossible for humans to beat the AI. Unity Machine Learning Agents Toolkit (ML Agents) is used in the project. ML Agents utilizes two deep reinforcement learning algorithms from OpenAI, which are Proximal Policy Optimisation (PPO) and Soft Actor-Critic (SAC). We chose to work with PPO.

How PPO works:

Proximal Policy Optimization was a breakthrough made by **OpenAI** in 2018, that proved to be pretty effective in navigating any game environment. If tuned well, this powerful algorithm maximizes the expected reward by learning the entire game through self-play. As OpenAI states,

"Proximal Policy Optimization (PPO), which perform comparably or better than state-of-the-art approaches while being much simpler to implement and tune."

- The neural network that represents the AI agent, learns the policy where it takes the current state from the environment to decide what action it should take based on the game state.
- Unlike Supervised Learning, the common problem of RL, the credit assignment problem where it is hard to decide which action of AI contributed to the reward received, can be resolved as policy gradients run policies for a while observing what actions led to high rewards, then increasing their probability through backpropagating gradients. Not to mention, PPO penalizes the agent if the new policy update deviates much from the old policy and also generates a ratio that differentiates between the new and old policy and clips this ratio from 0.8 to 1.2. Thus, PPO limits the policy update.

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

Here, see $r_t(\theta)$ is the ratio of the probability between the new and old policy where a_t is current policy and s_t is the last policy that was used to collect sample actions and θ is the policy parameter:

→ The action is more probable in the current policy than the old policy if $r_t(\theta) > 1$.

→ The action is less probable for current policy than for the old one if $r_t(\theta)$ is between 0 and 1.

- At each step, the collection of experiences is discarded after computing the policy update and we get a gradient at the end of each game from the sample action.
- The gradient encourages actions that led to high rewards and game-wins and actions that led to lose the games are discouraged.

Training Method:

The agent is given 7 observations as the input which are:

- x and y coordinate of the pong to know the current pong position
- x and y component of the velocity of the pong to know its heading
- Self y coordinate
- Enemy agent y coordinate
- Whether or not the agent playing on the right side or left side

Training is done by using the self-play. The two different agents with different Team IDs compete with each other. The pong is spawned at a random y-coordinate of the centre line with a random heading. The agents then decide on an action using the given observations. The agent only decides when the pong is heading towards its side. The action space of the agent is continuous. It is a single floating-point number, clamped between –1 and 1. This is then feed into the movement of the pad, a positive meaning to move up and vice versa. The magnitude of the output is proportional to the speed of the movement. Reward is then provided based on the performance of the agent. The decision rate was set at 50 per second.

Reward Structure:

For hitting back the pong = +0.25

For winning = +1

For losing = –1 – margin by which it missed

Movement penalty = –0.005 times the agent output

The “hit back” reward was added to encourage the agent to defend

The movement penalty was given to discourage the agent from moving too much or too fast

For losing, it is giving a –1 penalty as well as an additional penalty proportional to the margin by which the pad missed the pong. This encourages the agent to guide the pads closer to the pong. We noticed that by adding this extra penalty, training was much faster and more stable. This extra penalty helped as a guidance for the agent.

Training data:

Training configuration

trainer_type: ppo

hyperparameters:

batch_size: 1024

buffer_size: 10240

learning_rate: 0.001

beta: 0.005

learning_rate_schedule: linear

reward_signals:

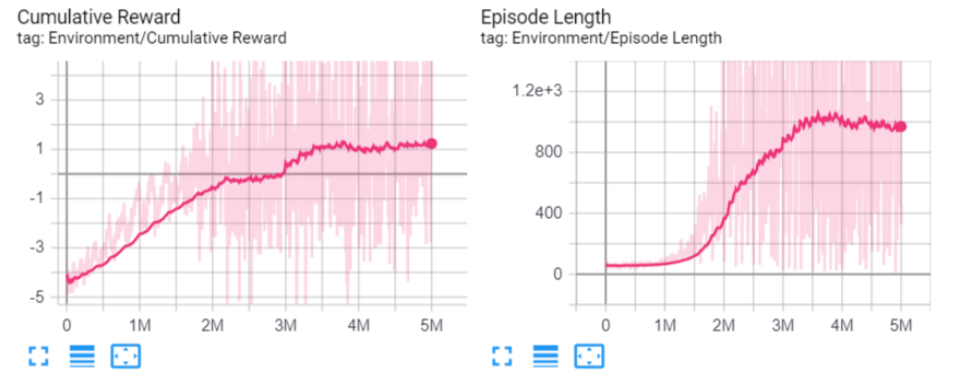
extrinsic:

gamma: 0.99

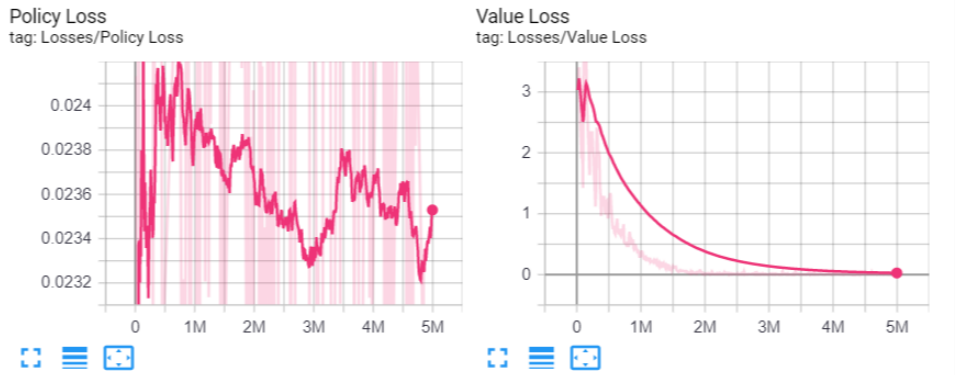
strength: 1.0

max_steps: 5000000

Tensorboard Data

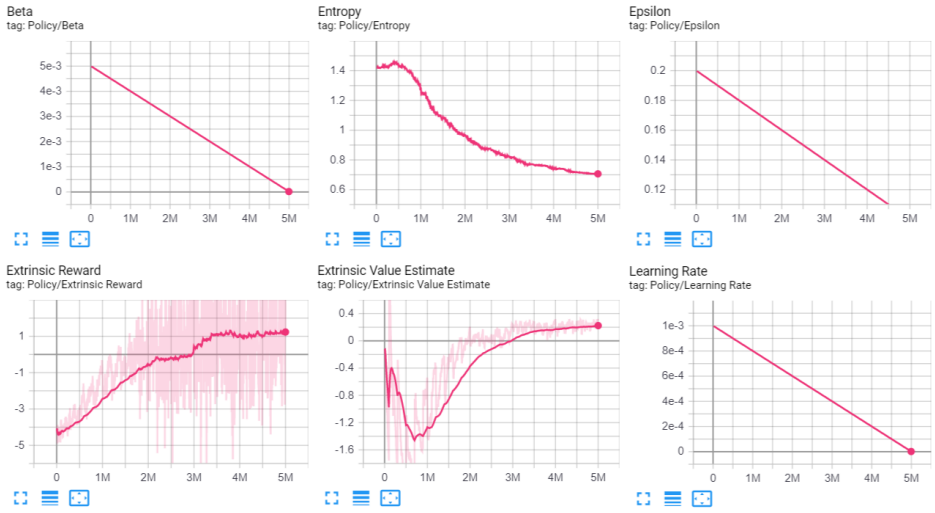


Cumulative Reward and Episode Length (Game Length) increases with time



Value Loss (difference between predicted reward and actual reward) decreases

Meaning the agent is getting better at predicting the reward with time



Findings:

Initially, the agent has no idea what to do. It moves around randomly and misses the pong most of the time. However, once in a while, the pong lands on the pad. This gives the agent a reward of 0.25. After 30 min, the agent slowly tries to track the motion of the ball and moves towards the position of the ball. It realises the smaller the margin of missing, the lesser the penalty. After 1 hour 30 min, the agent manages to defend reliably. Average game length starts increasing. After 2 hours, the agent manages to defend back almost every time. The agent follows the pong almost perfectly. After 3 hours of training, the agent reached near perfection. The agent has fully learnt the policy and further training does not improve the performance. Another behaviour we noticed is that the agent tries not to move too much to avoid penalty. It moves fast only when it needs to.

Human player statistics:

To understand how good the agent was performing, the agent performance was tested with multiple human players. We have each player play two matches as a warm up, followed by the actual run. The data we recorded includes the overall match score, how many times the human player managed to hit the pong back (defence score) in the entire match. The data we gathered are as follows:

AI Score	Human Score	Defence Score
10	1	1
10	0	3
10	0	4
10	1	4
10	0	11
10	2	11
10	1	15
10	1	18
10	1	30

The average defence score of the humans was 11. However, if the AI is left to play against each other, it manages an impressive defence score of around 400. Also, no human was able to score above 2 points. Therefore, the AI has mastered Pong well enough to defeat the average player by a huge margin.

Further development:

Although the AI performs extremely well, there is some scope for further development in the future. To make the game fair for humans, reaction time for the AI could be added. Currently the AI receives the most updated data, meaning the data is perfect and the AI is aware of the exact position of the pong at all times. Introducing a delay equivalent to human reaction time (15 to 25 ms) would show a ghost image of the pong rather than the current image. Therefore, the AI has to actively predict the position of the pong in order to defend it.

Another possible change could be limiting the actions the AI takes per second. Professional human gamers could take around 10 actions per second. Therefore, reducing the actions per second to 5 or 10 would make it more competitive. The AI needs to make better decisions with less errors in order to compensate for slow decisions.

References:

1. Minh,V., Kavuhcuoglu,K., Silver, D. (2013) Playing Atari with Deep Reinforcement Learning. Retrieved from https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf?fbclid=IwAR3Q8yCGG-g9DwuHNBwJQdNN2QsJMh2Hq-oOTfgZLATIZo9JD2Wzz_k2SMg

2. https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html?fbclid=IwAR3Q8yCGG-g9DwuHNBwJQdNN2QsJMh2Hq-oOTfgZLATIZo9JD2Wzz_k2SMg#:~:text=Two%20main%20components%20in%20policy,model%20and%20the%20value%20function.&text=Critic%20updates%20the%20value%20function,direction%20suggested%20by%20the%20critic

3. https://openai.com/blog/openai-baselines-ppo/?fbclid=IwAR3Q8yCGG-g9DwuHNBwJQdNN2QsJMh2Hq-oOTfgZLATIZo9JD2Wzz_k2SMg

4. https://nervanasystems.github.io/coach/components/agents/policy_optimization/ppo.html?fbclid=IwAR3Q8yCGG-g9DwuHNBwJQdNN2QsJMh2Hq-oOTfgZLATIZo9JD2Wzz_k2SMg