RUAS IC 2019 Phase 2 Evaluation

Title :AI Game Bot using R.L

Theme: AI Games

Application No: RUASIC/19/78

Mentor : Pallavi R Kumar

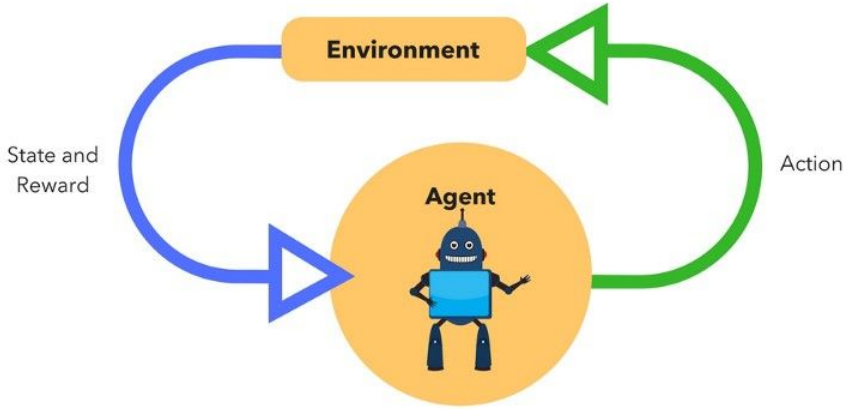
Team Leader: Jyothiprakash S

Institute: RUAS

Introduction

- Traditionally, game programmers used to use heuristic if-then-else type decisions to make educated guesses.
- But game developers can only predict so many scenarios and edge cases
- Ideal case- Bot doesn't run in circles!
- Game developers then tried to mimic how humans would play a game, and modeled human intelligence in a game bot i.e logic that uses the reinforcement learning observes the game's previous state and reward (such as the pixels seen on the screen or the game score). It then comes up with an action to perform on the environment.
- **R.L :**
  - There are three main branches in machine learning: Supervised Learning (*learning from labeled data*), Unsupervised Learning (*learning patterns from unlabeled data*), and Reinforcement Learning (**discovering data/labels through exploration and a reward signal**). RL not only do you start **without labels**, but **without data** too.
  - In RL, an **agent** interacts with it's **environment** through a sequence of events:
    - Agent observes the environment's initial state `s`
    - Agent chooses an action `a` and receives a reward `r` and a new state `s_` from the environment
    - The animal's brain reward function has been tuned for millions of years to optimize for survival, that is, for every action we take in the world we get a positive reward if its expected value increases our chance of survival or a negative reward if it lowers our chance of survival (think pleasure when you eat or pain when you fall down).

Introduction



- **Open AI Gym**: RL doesn't require training data as it gathers the information required from it's environment. In this case the game. OpenAI's gym has ready made gaming environments saving us the time in developing graphics of the game and concentrate on developing the RL model.

  It gives us a great way to train and test out RL models through games, which are great for RL, as we have clear **actions** (**left, right, etc.**), **states** (**could be position of players or pixels of screen, etc.**), and **rewards** (**points**).
- **Colab:** Colab is an engine that let's us run python programs on GPU for free.
- Tensorflow and keras frameworks

Method

The bot can be made to learn several games, thus offers generalization. Here, it's made to learn by playing two games, Lunar lander and Cartpole.

- Setting up the environment- The environment is loaded from gym using pip.
- **Lunar lander game:** The goal, as you can imagine, is to land on the moon! There are four discrete **actions** available: **do nothing**, **fire left orientation engine**, **fire main engine**, **fire right orientation engine.** The **state** is the coordinates and position of the lander. The **reward** is a combination of how close the lander is to the landing pad and how close it is to zero speed, basically **the closer it is to landing the higher the reward**. There are other things that affect the reward such as, firing the main engine deducts points on every frame, moving away from the landing pad deducts points, crashing deducts points, etc. This reward function is determined by the Lunar Lander environment. The game or episode ends when the lander lands, crashes, or flies off away from the screen.
- **Cart pole game**: A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

We start with an initial environment. It doesn't have any associated reward yet, but it has a state (S_t).Then for each iteration, an agent takes current state (S_t), picks best (based on model prediction) action (A_t) and executes it on an environment. Subsequently, environment returns a reward (R_t+1) for a given action, a new state (S_t+1) and an information if the new state is terminal. The process repeats until termination.

Multiple episodes of the game are run and data is gathered from it using which the training is performed. A convolution Neural network can be built which takes in pixel data. Here, the data can be gathered to gym. A DNN is built and trained from the data acquired by gym. This method is particularly called as DQN - deep queue network. Lunar lander is trained using policy gradient and cart pole with DQN however both can be applied to various games, this is just as an example.

Method

Deep queue network–Q-learning is a model-free reinforcement learning malgorithm. The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. It does not require a model (hence the connotation "model-free") of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations.The DeepMind system used a deep convolutional neural network, with layers of tiled convolutional filters to mimic the effects of receptive fields. Reinforcement learning is unstable or divergent when a nonlinear function approximator such as a neural network is used to represent Q. This instability comes from the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and the data distribution, and the correlations between Q and the target values.The technique used experience replay, a biologically inspired mechanism that uses a random sample of prior actions instead of the most recent action to proceed. [Wikipedia Q learning]

Policy Gradient is another widely used algorithms in reinforcement learning (RL). Here, instead of learning a value function that tells us what is the expected sum of rewards given a state and an action, we learn directly the policy function that maps state to action (select actions without using a value function). They are better than DQN because- 1. better convergence, 2.it is more effective in high dimensional action spaces and 3. they can learn stochastic policies

Once trained using the above methods, performance is measured and tuned by tuning the various hyperparameters like learning rate, discount, number of episodes, number of epochs and changing/adding layers etc. Loss is measured using loss functions lke cross entropy loss or MSE etc.

Finally render and generate video using gym functions to see the gameplay, constrast and compare bot's gameplay at several iterations of training to see the bot learning to play.

Method

Steps followed to arrive at the required result

- Get the `state` from the environment.
- Feed forward our policy network to predict the probability of each `action`we should take. We'll sample from this distribution to choose which `action` to take (i.e. toss a biased coin).
- Receive the `reward` and the next state `state_` from the environment for the `action` we took.
- Store this transition sequence of `state`, `action`, `reward`, for later training.
- Repeat steps 1–4. If we receive the `done` flag from the game it means the episode is over.
- Once the episode is over, we train our neural network to learn from our stored transitions using our **reward guided loss function**.
- Play next episode and repeat steps above! Eventually our agent will get good at the game and start getting some high scores.

Code along with notebook www.github.com/fistus/ruasic

Application

Recent advancements in RL game bots.

AlphaGO - Deepmind built a bot which beat the world's best GO player in 2017. AlphaGo Zero's neural network was trained using TensorFlow, with 64 GPU workers and 19 CPU parameter servers. Only four TPUs were used for inference. The neural network initially knew nothing about Go beyond the rules. Unlike earlier versions of AlphaGo, Zero only perceived the board's stones, rather than having some rare human-programmed edge cases to help recognize unusual Go board positions. The AI engaged in reinforcement learning, playing against itself until it could anticipate its own moves and how those moves would affect the game's outcome. In the first three days AlphaGo Zero played 4.9 million games against itself in quick succession.] It appeared to develop the skills required to beat top humans within just a few days, whereas the earlier AlphaGo took months of training to achieve the same level.**The hardware cost for a single AlphaGo Zero system, including custom components, has been quoted as around $25 million [Wikipedia - Alpha Go zero]**

Open AI 5

Is a system that performs as a team of bots playing against human players in the competitive five-on-five video game, Dota 2. The bots learn to play against humans at a high skill level entirely through AI. This is an whole another level as it requires highly intelligent functionalities like coordination , long term planning and choosing right actions in very large action space.

The system beat best dota team in on june 2018

	OpenAI 1v1 bot (2017)	OpenAI Five (2018)
CPUs	60,000 CPU cores on Azure	128,000 preemptible CPU cores on GCP
GPUs	256 K80 GPUs on Azure	256 P100 GPUs on GCP
Experience collected	~300 years per day	~180 years per day (~900 years per day counting each hero separately)
Size of observation	~3.3 kB	~36.8 kB
Observations per second of gameplay	10	7.5
Batch size	8,388,608 observations	1,048,576 observations
Batches per minute	~20	~60

DeepStar: Similar for starcraft

Applications

- The reason these companies are investing this extensively is because RL offers generalisation .
-
- This is considered as a pathway to artificial general intelligence.

- To gain an insight and derive real life applications of RL. For example, self driving cars can utilize RL for learning from the environment instead of behaviorial cloning(currently used DL tech for self driving cars. Or lunar lander game type application in real field to land rockets safely.

Applications

- In particular to gaming bots, they needn’t have to be trained for super human level like the examples above
- They can be utilised as realistic opponents.
- Current day bots are hard programmed which makes the experience of gaming either boring or too difficult either way predectible.
- By using ai bots not only it is more realistic, but also keeps the gamer more engaged and on their edge as they can’t anticipate the intelligent bot’s next move.

Conclusion

A simple game bot was developed for 2d games using Reinforcement learning methodologies, using Open AI gym, tensorflow etc

The bot was able to excel at the given games.

Further aim would be to generalize the algorithm to be able to learn and play different games.