PROJECT DESCRIPTION

Overview

We have implemented A3C (Asynchronous Advantage Actor Critic) to create an agent which learns how to play a 2D game by a method of reinforcement learning using rewards and punishments. The agent starts not knowing anything about a game or it's environment. It is provided with a set of basic controls and given information about what actions it can take in the given environment.

These actions might include basic movement controls or a point and shoot fire control. The agent can choose whatever action it wants to perform and try it on the environment to see how the environment performs.

Along with the control information, knowledge about the rewards, which bring the agent closer to goal state, and punishments, which move the agent away from the goal state is provided in the environment. These rewards might include additional score, time bonus or level-up in a game. The punishments might include a negative score, time penalty or the death of the character in the game. Every agent is given a particular goal, which could be anything from completing a particular level within a set time frame or achieving a high score for the level.

Initially, the agent tries out different actions and keeps a record of its score (reward/punishment) for every action it takes. These actions might be successful in completing a goal or may move the agent away from the goad, in which case, the agent, will record that particular move as a non-useful move. For an action which results in a positive score, it will be recorded as a useful action and utilized in the future run of the game.

Over the period of the training time, these actions and their respective rewards and punishments will formulate the neural links between the initial state and goal state and help the agent achieve a final goal.

Thus, with a reinforced system of rewards and punishments, the agent learns how to play a 2D environment game.

Implementation

We have Implemented our Model using Pytorch. Our Model is based on A3C(Asynchronous Actor Critic Method), which is a deep reinforcement learning method. We Start by creating

the brain of our network , then we create a shared network that our workers can use. We use convulational neural networks to process the environment provide to us by OpenAI gym.

1. The A3C algorithm begins by constructing the global network. This network will consist of convolutional layers to process spatial dependencies, followed by an LSTM layer to process temporal dependencies, and finally, value and policy output layers.

2. Next, a set of worker agents, each with their own network and environment are created. Each of these workers are run on a separate processor thread, so there should be no more workers than there are threads on your CPU.

3. Each worker begins by setting its network parameters to those of the global network. Each worker then interacts with its own copy of the environment and collects experience. Each keeps a list of experience tuples *(observation, action, reward, done, value)* that is constantly added to from interactions with the environment.

4. Once the worker's experience history is large enough, we use it to determine discounted return and advantage, and use those to calculate value and policy losses. We also calculate an entropy (H) of the policy. This corresponds to the spread of action probabilities. If the policy outputs actions with relatively similar probabilities, then entropy will be high, but if the policy suggests a single action with a large probability then entropy will be low. We use the entropy as a means of improving exploration, by encouraging the model to be conservative regarding its sureness of the correct action.

5. A worker then uses these losses to obtain gradients with respect to its network parameters. Each of these gradients are typically clipped in order to prevent overly-large parameter updates which can destabilize the policy.
6. A worker then uses the gradients to update the global network parameters. In this way, the global network is constantly being updated by each of the agents, as they interact with their environment.

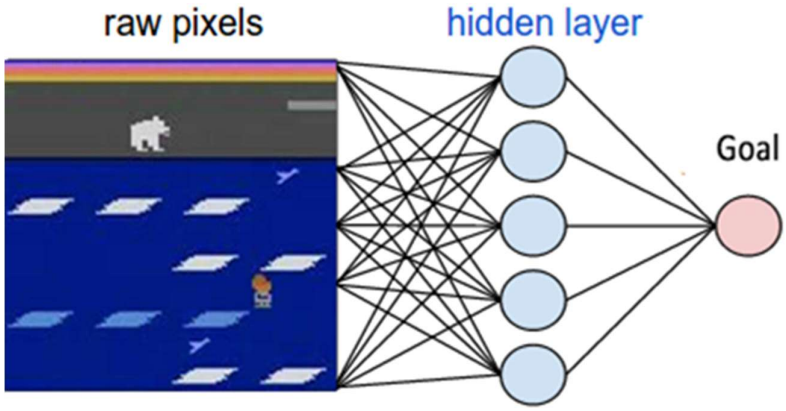

Fig. 1 Neural Network representation for frostbite

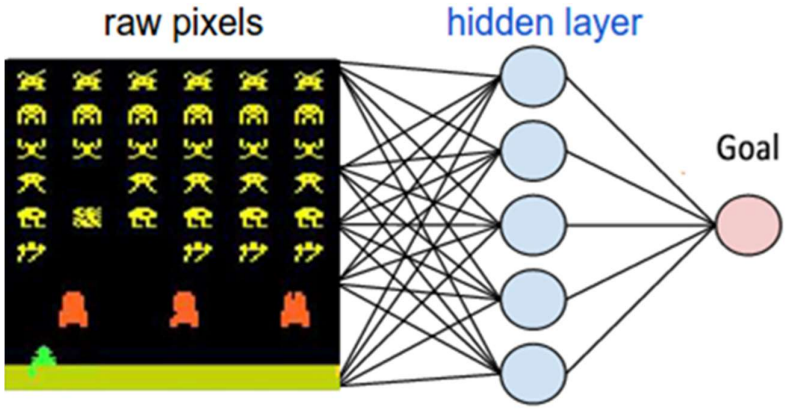

Fig. 2 Neural Network representation for space invader

The general outline of the code architecture is:

1. **model.py:** This contains our main model, that is the brain. Thus it contains our basic convulational neural network and LSTM layer.

2. **shared_optim.py:** This contains the shared optimizer for our workers

3. **train.py:** In this module we train our model, thus we calculate and update various policies

4. **test.py:**In this module, we test the environments and print various information such as episode length, rewards and episode duration

5. **envs.py:** In this module, we manage the ATARI environments provided by OpenAI Gym, and also record the videos

6. **main.py:** This is the driver module,i.e the main module

Frostbite

About

Frostbite is a 1983 action game designed by Steve Cartwright for the Atari 2600, and published by Activision in 1983. The object of Frostbite is to help Frostbite Bailey build igloos by jumping on floating blocks of ice, while trying to avoid deadly hazards like clams, snow geese, Alaskan king crabs, polar bears, and the rapidly dropping temperature.

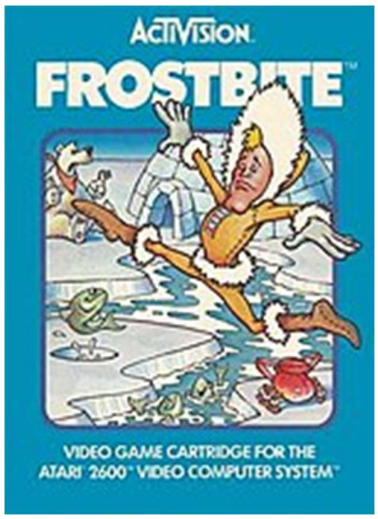

Fig. 3 Frostbite

In this environment, the observation is an RGB image of the screen, which is an array of shape (210, 160, 3) Each action is repeatedly performed for a duration of kkk frames, where kkk is uniformly sampled from {2,3,4}\{2, 3, 4\}{2,3,4}.

Game Description

The bottom two thirds of the screen are covered by a mass of water with four rows of ice blocks floating horizontally.

The player moves by jumping from one row to another while trying to avoid various kinds of foes including crabs and birds. There are also fish which grant extra points.

On the top of the screen is the shore where the player must build the Igloo. From the fourth level onwards there is also a polar bear walking around on the shore which must be avoided.

The game levels alternate between large ice blocks and little ice pieces. The levels with the little pieces are actually easier, since one can walk left or right over them without falling in the water.

Each time the player jumps on a piece of ice in a row its color changes from white to blue and the player gets an ice block in the Igloo on the shore. The player has the ability to change

the direction in which the ice is flowing by pressing the fire button, but that costs a piece of the Igloo.

Fig. 4 Forstbite Gameplay

After the player has jumped on all the pieces on the screen, they all turn back to white and one can jump on them again. When all the 15 ice blocks required for building the Igloo are gathered, the player has to get back to the shore and get inside it, thus proceeding to the next level. On every level the enemies and the ice blocks move slightly faster than in the previous level making the game more difficult.

Each level must be completed in 45 seconds, (represented as the declining temperature,) else the Eskimo dies frozen. The faster the level is completed the more bonus points are awarded to the player.

If player makes it past level 20, a "magic" fish will appear between the temperature gage and the number of lives remaining, this serves no real purpose other than as an Easter egg to the game.

Graph

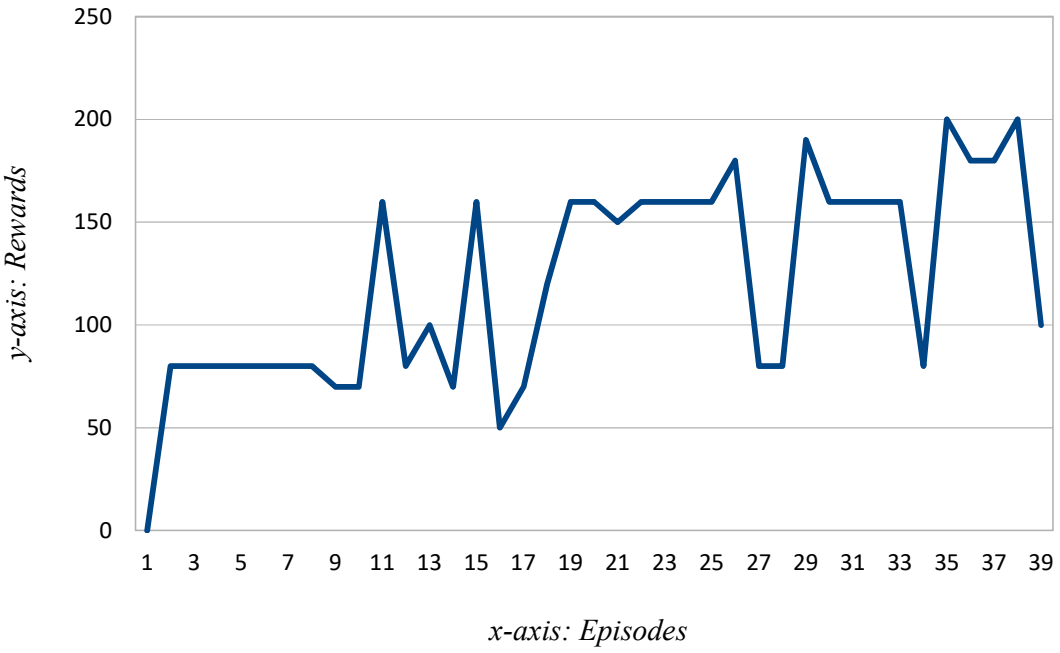

Fig. 5 Episode-Reward graph

The x-axis shows the episode value and the y-axis shows the reward which the agent has achieved for each episode. The graph represents the data of 39 episodes with varying degree of reward value.

Initially the reward value starts as zero, and with gradual learning, the reward value increases with a varying degree of success. A major increase spice can be noticed at the 11th episode with a major downfall at the 17th episode.

The reward value stabilizes between the 18th and 25th episode and then further spikes and downfalls are observed which show that the agent finds a somewhat successful method to achieve a reward but once it tries to move forward with this approach, a sudden downfall indicates that the method learned by the agent is not appropriate enough to achieve the final goal.

The maximum reward achieved by the agent is 200, which is first achieved at episode 35, and then again at episode 37, representing a correct strategy achieved by the agent which leads to a successful goal state.

Space Invader

About

Space Invaders is an arcade game created by Tomohiro Nishikado and released in 1978. Space Invaders is one of the earliest shooting games; the aim is to defeat waves of aliens with a laser to earn as many points as possible. In this environment, the observation is an RGB image of the screen, which is an array of shape (210, 160, 3) Each action is repeatedly performed for a duration of kkk frames, where kkk is uniformly sampled from {2,3,4}\{2, 3, 4\} {2,3,4}.

Fig. 6 Space Invaders

Game description

Space Invaders is a two-dimensional shooter game in which the player controls a laser cannon by moving it horizontally across the bottom of the screen and firing at descending aliens. The aim is to defeat five rows of eleven aliens—some versions feature different numbers—that move horizontally back and forth across the screen as they advance toward the bottom of the screen. The player defeats an alien, and earns points, by shooting it with the laser cannon. As more aliens are defeated, the aliens' movement and the game's music both speed up.

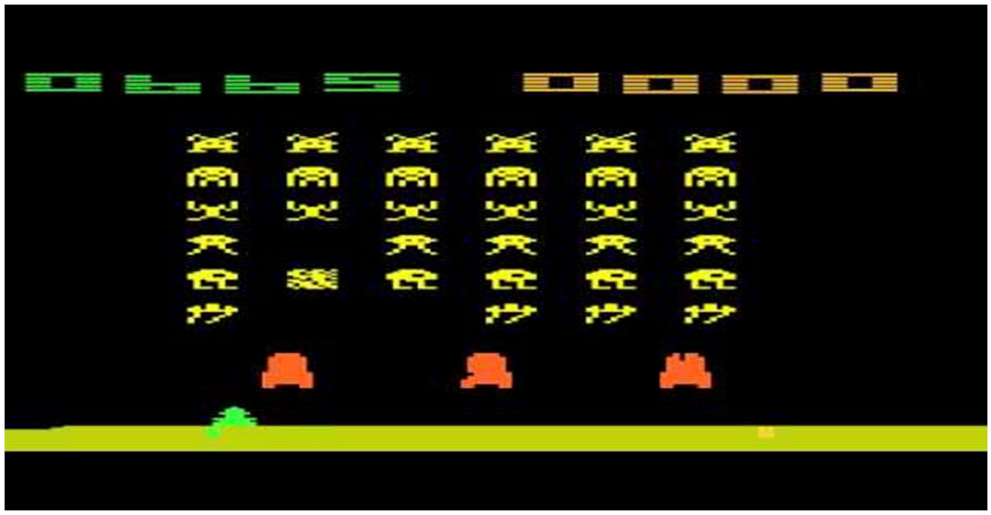

Fig. 7 Space Invader Gameplay

The aliens attempt to destroy the cannon by firing at it while they approach the bottom of the screen. If they reach the bottom, the alien invasion is successful and the game ends. A special "mystery ship" will occasionally move across the top of the screen and award bonus points if destroyed.

The laser cannon is partially protected by several stationary defence bunkers—the number varies by version—that are gradually destroyed by numerous blasts from the aliens or player. A game will also end if the player's last laser base is destroyed.

Graph

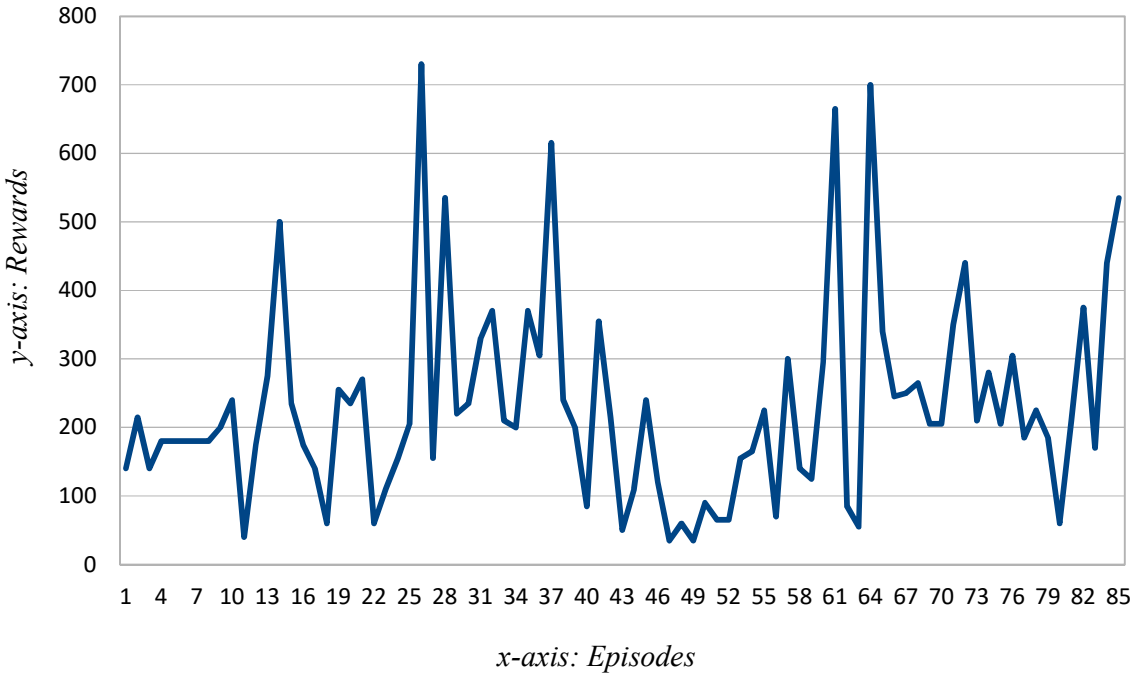

Fig. 8 Episode-Reward graph

The x-axis shows the episode value and the y-axis shows the reward which the agent has achieved for each episode. The graph represents the data of 39 episodes with varying degree of reward value.

Initially the reward value starts as zero, and with gradual learning, the reward value increases with a varying degree of success. A major increase spice can be noticed at the 11th episode with a major downfall at the 17th episode.

The reward value stabilizes between the 18th and 25th episode and then further spikes and downfalls are observed which show that the agent finds a somewhat successful method to achieve a reward but once it tries to move forward with this approach, a sudden downfall indicates that the method learned by the agent is not appropriate enough to achieve the final goal.

The maximum reward achieved by the agent is 200, which is first achieved at episode 35, and then again at episode 37, representing a correct strategy achieved by the agent which leads to a successful goal state.

Conclusion

We were able to see that the agent is successfully able to learn how to play the two games with sufficient efficiency and hence the method of reinforcement learning can be stated to be an optimum method to implement neural networks for use in technologies other than games.

Actor critic methods and A3C in particular are algorithms which can be applied to numerous instances of machine learning procedures and make them faster and more efficient to deal with. The lack of GPU usage also helps for systems which do not have enough computation power to run large scale simulations and training model. Hence, these algorithms provide an acceptable alternative.