**JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY**

**ICS 2405 KNOWLEDGE BASED SYSTEMS**

**COURSEWORK: SPACE INVADERS GAME PLAYED BY A.I AGENT**

**COURSEWORK REPORT**

**Name:** COLLINS WANJOHI NJERU **Reg No:** SCT-211-9782/2015

**Problem/Question**- create an artificial intelligence agent that will learn how to play the classic Atari Space Invaders game.

Space Invaders Overview

Space Invaders is a two-dimensional fixed shooter game in which the player controls a ship with lasers by moving it horizontally across the bottom of the screen and firing at descending aliens. The aim is to defeat five rows of ten aliens that move horizontally back and forth across the screen as they advance towards 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. The aliens attempt to destroy the ship 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 ship is partially protected by several stationary defence bunkers that are gradually destroyed by projectiles from the aliens and player.

For this particular project, we selected a convolutional neural network to learn from frames captured from the game screen. The AI agent learns only from the environment’s state; it only considers the game score and pixel data fed to it to make a decision. The method getPresentFrame() grabs a screenshot of the current game screen. The method getNextFrame() updates the screen from the present frame and then grabs a screenshot of that game frame that comes afterwards. This helps the AI agent view and learn about ‘near-future’ variables such as bullet trajectories and enemy positions. This is same as how we, humans, make a chess move decision by anticipating what the opponent might consider for his/her next move. The AI agent receives the frames as pixel data to feed the training graph. The AI agent saves learning progress in a checkpoints file after every 5000 steps. The replay memory is saved in a deque data structure since it provides fast data appends and pops compared to other data structures. The createGraph() method builds a five-layer convolutional graph session for the network. The output layer has 4 ‘neurons’ that represent the possible actions for the AI agent. We use a rectified linear unit activation function to convolve the 4D input layer and tensors into a 2D convolution. The CNN analyses pixel data from getPresentFrame and getNextFrame methods to make a decision. A decision can either be to stay put, shoot, move left or move right. Stay/Do nothing is the most probable decision.

|  |  |  |
| --- | --- | --- |
| **Agent Action** | **ACTION** | **PROBABILITY** |
| action [0] | Stay/Do nothing | 0.6 |
| action [1] | Shoot | 0.2 |
| action [2] | Move left | 0.1 |
| action [3] | Move right | 0.1 |

*Table 1: Table of Action-Probability mapping*

These actions’ probabilities are set on the file RL.py i.e.

maxIndex = choice((0,1,2,3), 1, p=(0.60, 0.20, 0.10,0.10))

Possible AI agent modes;

* Observing- the agent watches the game and movements by the enemy group.
* Training- the agent randomly selects a batch of 100 frames from the replay memory to train on.
* Model-only- the agent makes moves based on saved memory without training.

Once the game is over, the system delays for 2 seconds and restarts the game.

Files in the project include:

spaceinvaders.py- game. Contains main function to run the program.

RL.py- Reinforcement learning using 5-layer Convolution Neural Network.

checkpoints- directory to save progress of reinforcement learning process.

Run the game using the terminal command 'python spaceinvaders.py'. If you're using Python 3, replace the command "python" with "python3".

CNN analyses present and next frame of the game to make a decision. A decision can either be to stay put, shoot, move left or move right. Stay/Do nothing is the most probable decision. The AI agent creates a Q-table with column=all possible actions, row=all possible states. Each cell of the Q-table is filled with the max reward for doing that action in the given state. Max reward is calculated using the Bellman Equation. Reward=1 if alien is shot, 0 if ship is killed or no alien is killed; implemented using a step function. The AI agent fills in the Q-Table accordingly. At the start of the game, the AI agent uses an epsilon-greedy technique- random actions. As the AI unintentionally stumbles upon rewards or penalties, it starts to associate performing certain actions in certain states as either good or bad. As training goes on, the weight values are readjusted such that at the end of the learning, the calculated outputs given by an AI agent are similar to the expected outputs. The value of epsilon (learning rate) slowly decays from one to zero as training progresses. At zero epsilon, the AI agent has fully learnt the environment and makes the best moves possible. The AI agent saves learning progress in a checkpoints file after every 5000 steps.

**What I Have Learnt**

Basics of creating a PyGame 2D game environment- I learnt how to create a screen, work with sprites and how to display the sprites within the screen window.

Basics of Deep learning- I learnt that agents begin with an epsilon-greedy technique to make random decisions. The agents then, over time, recalculate their weights and bias vectors through back-propagation to match the calculated values with expected values. I learnt about the concept of images represented as tensors which are matrices of number arrays that have more than 3 dimensions. I learnt how an agent converts images, of environments it wants to learn, into arrays (Chris Nicholson, 2019), a filter is passed over the image to identify features using convolution and activation algorithms and max pooling algorithms to feed to a training graph.

**REFERENCES**

Original Space Invaders source code: <https://github.com/leerob/Space_Invaders>  
Original reinforcement learning agent code source: Siraj Raval <https://github.com/llSourcell/pong_neural_network_live>

Chris Nicholson (2019). A Beginner's Guide to Multilayer Perceptrons (MLP). <https://pathmind.com/wiki/multilayer-perceptron>

Chris Nicholson (2019). A Beginner's Guide to Convolutional Neural Networks (CNNs). <https://pathmind.com/wiki/convolutional-network>

James Loy (May 14, 2018). How to build your own Neural Network from scratch in Python. <https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6>