Image Processing for Discovering the Rules of Video Games

Erik Culberson, Neil Ferman, and Viljo Wagner

1 Introduction

There are many examples of machine learning algorithms being able to play video games when the algorithms are integrated directly within the game. This is a costly process and needs hours of development work as well as needing more access to resources such as the source code of the video game. This approach also gives algorithms more in-game knowledge than a normal player would have. By removing this advantage from the algorithms we will be able to see how well the algorithms can compare to human players given an equal playing field. General Video Game Playing AI is a little newer, but has been approached in [2]. Given the same inputs as a human player, a screen image as an input and a set of key controls as output, can the algorithm discover the set of rules of the game as well as the rules of the game world?

2 Problem

Current research into artificial learning in games focus heavily on algorithms' ability to discover the optimal or fastest solution to the current level or problem. The algorithms are given the rules and knowledge of the game and the environment before hand. This forces the algorithm to only be able to be used for one specific game. To create a general algorithm that can play multiple games, the first step is to allow the algorithm to learn the rules of the game on its own. This paper focuses on the question of whether an AI can learn the rules of the game, including inputs, as well as rewards and penalties of certain actions, and discover goals using only screenshots taken as the game is played.

3 Game and Emulator

We chose to use Ms. Pac-Man as our test game for this project. We chose this game due to its simplistic nature in understanding the game by image, but complication in AI. Ms. Pac-Man is a game where you play as Ms. Pac-Man, a character inside of a maze. Your objective is to eat little dots called pellets inside of the maze. There are four ghosts within the maze who chase Ms. Pac-Man that you want to avoid. The four ghosts are different colors and behave in slightly different ways. What makes the AI of Ms. Pac-Man interesting is how the ghosts have an expected behavior, but also have a random chance of taking a random route instead of following their expected behavior [3]. This makes the AI less predictable for a player (and indeed for an AI) to learn.

To run the game, and be able to connect it with our algorithms, we used an emulator called FCEUX. FCEUX is an emulator capable of running games designed to run on the Nintendo Entertainment System (NES). FCEUX has plugins that allow a user to run lua scripts through it. It also offers debugging tools like memory address lookup and emulation speed. This makes it an ideal candidate for teaching an AI to play Ms. Pac-Man.

4 Approach

Several different algorithms were needed to achieve our goal. We will break each overall algorithm into steps for the project and go into detail of how each algorithm works. The steps are as follows:

Image Capture: Capture images of the game as it is played, frame by frame.

Image Recognition: Convert the image into an array of colors and coordinates, and separate images by parts.

Learning: Pass the encoded images as well as a set of valid inputs into a learning algorithm, to be used to learn the rules of the game and update accordingly. We tried two different learning algorithms: QLearning, and NEAT.

5 Bridge

Our bridge script was used as a way to connect the chosen emulator (FCEUX) with our image capture script. The bridge script was written in lua, and uses lua sockets to communicate with our python image capturing script. The bridge script contains the following functions:

Set input key
Get Inputs
Get Screen
Get Location
Get Points
Reset
Done (ONLY DEATH)
Skip Frame
Get Palet Count

6 Image Processing

The image processing algorithm gets the image from the lua bridge and applies the following operations to identify object in an image and pass them on to the Q-Learning/NEAT algorithms.

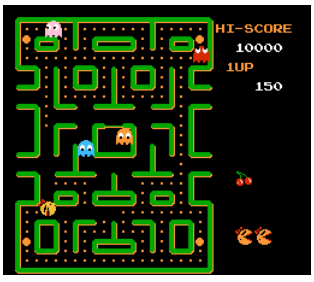
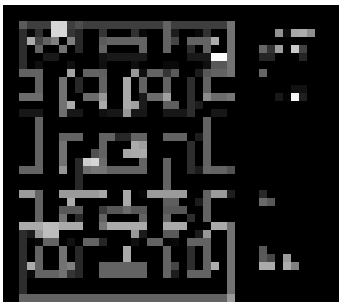


Image to Image Delta: If the current image is not the first image processed, then the algorithm will compare the previous image to the current one and depending on the delta-score it will decide whether the current image is different enough to warrant a complete reprocessing of the current image or if it is sufficient to just process the regions of the image that have changed.

Change Resolution: The resolution is being reduced to reduce the information density with the thought that only the most prominent features will be available afterwards. This works well for finding characters and objects in the image but most of the time any text based information like the score is lost.

Convert to Grayscale: This steps helps improve the speed of the algorithm since the algorithm has to only deal with 8-bits of color instead of 24-bits. The grayscale conversion function can be provided with weight parameters for red, green and blue (RGB). This allows the highlighting of certain objects of interest.



Neighbor-Matrix: The neighbor matrix is just there to determine which fields are of interest. For example when processing the first image there is only information on the part of the image that has been processed while further iterations have information on previous run throughs. The matrix is always relative to the

pixel in the image that is being processed.

The current Image Processing algorithm does no image classification but the algorithm has been set up in a way that it can be fed into a machine learning algorithm to optimize the weights for the different games that could be played, and then classify the different objects that are outputted.

7 QLearning

QLearning is an algorithm that uses Model-Based Learning. It was the first algorithm we tried to implement. It is an evolved form of Markov Chains. To summarize QLearning, it takes a state, and then has transitions from its current state into future states. Each transition has a probability of being taken to move into the next state. Each possible state that can be transitioned into (known as a future state) has a reward or penalty associated with it. QLearning will at first pick a future state at random, and then, based on whether a reward or penalty is reached, update the transition probability. What makes QLearning so appealing for learning games is that it can on occasion, take a route that seems sub-optimal to explore whether that future state (or any subsequent future state) might lead to greater rewards. This is good for games that have random chance outcomes that are sometimes bad, but sometimes good. Our QLearning algorithm would take as input the boundaries of the screen, locations of enemy as well as Ms. Pac-Man, certain actions that gained rewards (increase score, eat pellets), and the state of Ms. Pac-Man (not dead, and dead). However, the QLearning did not seem to be very successful at learning the game. It mostly moved at random, and just oscillated back and forth between a couple of different hallways, and didnt seek out pellets or avoid ghosts.

8 NEAT (Neuro-Evolution Augmenting Topologies)

Our NEAT algorithm was the second algorithm we tried but it did not fare better than Q-Learning. NEAT, better explained in the original paper [8], is a population of full neural networks that use a genetic algorithm to form the neural network structure and weights. This allows the algorithm to find a minimalist neural network that can complete the task at hand.

We were not able to get this algorithm to work either and the reasons why we think the NEAT algorithm failed was, for two reasons, there are too many inputs or we did not give it enough time to train. Even though we reduced the inputs from are original 172,032 inputs to 2,688 inputs we did not take into account that because the NEAT algorithm starts off with 0 hidden nodes that it would take so long to generate a full connected graph that could take in all those inputs and create meaningful connection between all of the nodes. One solution to this is to do more image processing beforehand to find all the variables such as player position, enemy position, and so on to reduce the number of inputs which give the algorithm a chance to learn what the meaning behind the inputs are.

Another solution would be to allow the algorithm to learn for a longer period of time. The NEAT algorithm ran for 121 generations with a population of 150. Over the 121 generations the best performing networks max fitness hovered around 39 out of 177 never showing improvement from generation to generation. As you can see from FIGURE REF HERE over generation 111 to 121 there is no improvement in the fitness. Given more time for the algorithm to build a bigger graph or reducing the input we believe that the algorithm would be able to improve its fitness. Another problem with our algorithm is that it took a very long time to learn. It took roughly 6 days to learn 121 generations. There are a couple of ways we think we can improve this time. The easiest being to introduce a timeout function that will fail the network after a set number of frames that have not added to the fitness of the network.

9 Results

We set out to create an algorithm that could learn and play virtually any game based on screenshots and learning inputs/rules of the game. The algorithms we came up with wound

up having very limited capabilities. Currently the screen size of the game needs to be the same, and the games score needs to be provided to the algorithm so it can track rewards. We did not have enough time to really see how either QLearning or NEAT could really learn the rules of the game. Learning algorithms can take several days or weeks before real progress can be seen, and we have only been able to run it for a few days. The algorithm is able to learn to navigate the field after some time. However, learning past that (like, say, that eating blue ghosts can cause a reward) is unknown. A major limiting factor was that the Image Processing was literally just image processing which found objects based on given parameters but did no classification so the Q-Learning/NEAT algorithms had to figure out what the objects are by themselves. This is also why we decided to pull the position for the game character and the score from the Emulator RAM so we could at least start training the learning algorithms.

10 What we learned and Future Ideas

We had a lot of trouble connecting so many different, separate ideas, together. Especially connecting the image processing to our Learning algorithms was a big slow down for us. It is also not possible to get lua sockets on a Windows machine, a problem that prevented one of the researchers in improving QLearning, since he had Windows. There are better image processing libraries out there we would like to experiment with, such as Tensorflow. This could more readily solve the issue of classifying objects. NEAT and QLearning could both be greatly improved, as well as the inputs received by them could be improved so they can show learning and improvement more robustly. As is always the case with life, as the project went on, new ideas were brought to our attention continuously that we didnt have time to implement or look into in great detail. We would like more time to experiment with more learning algorithms in the future.

11 Conclusion

References

[1] S. Sabour, N. Frosst, and G. E. Hinton, Dynamic Routing Between Capsules, Neural Information Processing Systems, 2017.

[2] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, "General Video Game AI: Competition, Challenges and Opportunities," Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2016.

[3] T. Thompson, "What's the Deal With Pac-Man?", Exploring artificial intelligence in games: in academia, indie and AAA gaming, 2014. Available: https://aiandgames.com/ai-and-pacman/

[4] S. Raval, "Q Learning Explained", December 1, 2017. Available: https://www.youtube.com/watch?v=aCEvtRtNO-M

[5] J. Bradberry, Introduction to Monte Carlo Tree Search, 2015. Available: https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/

[6] Harrison, Using a neural network to solve OpenAI's CartPole balancing environment, 2017. Available: https://pythonprogramming.net/openai-cartpole-neural-network-example-machine-learning-tutorial

[7] J. Levine, Monte Carlo Tree Search, 2017. Available: https://www.youtube.com/watch?v=UXW2yZndl7U

[8] K. Stanley, R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies", The MIT Press Journals, 2002. Available: http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf