A LITERATURE REVIEW OF AI FOR CLASSIC VIDEO GAMES USING REINFORCEMENT LEARNING

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for

CS 297

By

Shivika Sodhi

December 8, 2016

Abstract

Deep reinforcement learning is the road to build artificially intelligent machines that can perform tasks similar to that of human beings, without any prior training. It is essential for training an agent to make smart decisions under uncertain conditions and to take small actions in order to achieve a higher overarching goal. In this paper, some preliminary research is done in order to understand how reinforcement learning and deep learning techniques can be combined, along with an approximation function called Q-learning, to train an agent to play a classic game. This deep neural network model will successfully learn to control policies directly from high-dimensional sensory input using reinforcement learning. The challenge is that the agent only sees the pixels and the rewards, similar to a human player. The main deep neural net used was Convolutional Neural Networks (CNNs).

1 INTRODUCTION

Our primary motivation behind choosing this topic was to understand the impact of deep learning on Artificial Intelligence. Through this project, we aim to demonstrate the basic ability of a neural network to learn to play Archon. Our approach is inspired by DeepMind's strategy of successfully learning control policies directly from high-dimensional sensory input using a minimal amount of prior information about the game. We chose Archon as there hasn't been a successful attempt yet to use reinforcement learning the play the game.

Apart from game playing, there are various other applications of this research, like, self-driving cars, simulated quadrupeds learning to run and leap, robots learning how to perform complex manipulation tasks that defy explicit programming and even in the medical industry.

Archon: The Light and the Dark, is a classic game, similar in concept to chess, but with an additional feature where the landing on another player's piece results in an arcade-style fight to determine the victor. In this mode, the two pieces are placed into a full-screen 'combat arena' and must battle to determine the winner.

Deep learning (also known as deep structured learning, hierarchical training or deep machine learning) is a branch of machine learning based on a set of algorithms that attempt to model high level abstractions in data by using a deep graph with multiple processing layers, composed of multiple linear and non-linear transformations. Based on these techniques, we plan on designing a model that will train the machine to make critical decisions in order to play the game well. This project is aimed at understanding how effective deep learning algorithms are in training a machine to think like (emulate) a human being.

2 DELIVERABLE 1

In this project we want to teach a neural network to play Archon, and make decisions like a human being. It takes the human brain one-tenth of a second to process the information of an object seen by the human eye. This deliverable is a commencement of our objective of designing a model to simulate the decision making process of the brain. In this deliverable we design a python program intended to take screenshots of a particular part of the screen, every one-tenth of a second. These images will then be fed into the model in the form of raw pixels, and based on those pixels the neural network will be trained. The screen capturing is done with the help of a python library called pyscreenshot. This library is a cross-platform wrapper which either copies the contents of the entire desktop or parts of it with the help of a function called grab(). To grab parts of the screen, coordinates of the target are defined in an argument of grab() called bbox.

The captured contents are then stored into a Python Imaging Library (PIL) image memory. The back-end library PIL, contains a module called Image, which consists of a number of factory functions. One of those functions is called resize(), which returns a resized copy of the image, and has two parameters namely size and resample. Size contains a tuple of the pixels of the width and height of the image, and resample is an optional sampling filter that consists of a high quality downsampling filter called PIL.Image.ANTIALIAS. After the downsampling, the image is converted into a black and white mode from an RGB mode, with the help of a function, convert() from the same library. This function converts the image to black and white by setting the mode as “L”.

The downsampled image is then saved into a directory with the help of another function of Image called save(). This entire operation of grabbing an image, downsampling it and then storing it into a folder repeats for about one tenth of a second, due to a while condition.

3 DELIVERABLE 2

In order to be able to play the game, the machine needs to control it. This deliverable

Deliverable 2 comprised of another program in python that controls the keyboard and mouse functions of the laptop. This would further help in controlling the game and then take screenshots of it.

This program was designed with the help of a cross-platform module for GUI automation for human beings, called PyAutoGUI. This library helps control keyboard and mouse movements from a python script. Linux needed another library called python3-Xlib to be installed before installing PyAutoGUI.

For the mouse to move to a certain coordinate on the screen, the python library PyAutoGUI, contains a function called locateOnScreen(). The function returns left, top, width and height of the matching region with the image that's passed as it's parameter. Once it fetches the width and height of the image, the program calls another function of the library called moveTo(), which contains the coordinates of the target as its parameters. After reaching the target position, the function doubleClick() starts the emulator by doing a double click on its icon.

Once the game starts, another function of the library called press(), calls the keyboard function and then a string is passed to it from pyautogui.KEYBOARD_KEYS, in the form of an argument. This string is 'enter' and it emulates the enter key of the keyboard. Thus the game begins, but now the program will need to wait for some time before giving any other control instruction. Hence, another parameter called interval, is passed as the argument of the press() function. The program then waits for the time specified in the interval parameter. Once the game loads, it asks to select a few options, the navigation on the screen of the game is done through the arrow keys of the keyboard. This is done by passing 'left', 'right', 'up' or 'down' in the argument of the press() function. Once the program reaches the target coordinate of the screen, then with the help of 'ctrl' argument it selects the required action needed to be performed.

4 DELIVERABLE 3

A recent research by the team at DeepMind pioneered the idea that neural networks can have a remarkable performance at game playing, using a minimal amount of prior information about the game. When it comes to artificial neural network and deep learning in image processing, CNN is the most successful algorithm around. Moreover, as there hasn't been a successful attempt yet, to use a convolutional network to learn to play Archon we decided on building out own network to fill this gap. The purpose of this deliverable was to get started with deep neural networks. It aims at building a CNN comprising of two convolutional layers and a multilayer neural network, with the help of a python library called Keras. The dataset used to train this neural network was the MNIST dataset, the 'hello world' data for image processing.

The data is loaded using a predefined function in Keras, where it is structured as a 3-dimensional array of instance, image width and image height. But for a multilayer perceptron model, the image needs to be reduced down to a vector of pixels, hence the numpy array is reduced to a two dimensional array. After the transformation, the gray scale pixel values are then normalized. As the target variable in this training data is an integer from 0 to 9, this problem is a multi-class classification problem. Thus we transform the vector of class integers into a binary matrix by performing one hot encoding on the target data.

After the preprocessing, we design a convolutional neural network model. CNNs are a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. They are made up of neurons that have learnable weights and biases. Each neuron receives certain inputs, performs a dot product operation between its weights and a small region it is connected to in the input volume. The entire network expresses a single differentiable score function: from raw image pixels on one end to class scores at the other.

The first hidden layer or the input layer of this model is a convolutional layer called a Convolution2D, having 30 feature maps, each of size 5 by 5 and a rectifier activation function. This layer expects images with having a structure, [pixels][width][height]. The next layer is a pooling layer, called MaxPooling2D and configured with a pool size of 2 by 2. The layer after that is another convolutional layer having 15 feature maps, with each map being of size 3 by 3, whose output would be sent to another pooling layer. The next layer is a regularization layer using a dropout, which is configured to randomly exclude twenty percent of neurons in the layer to reduce overfitting. The output of this layer is sent to another layer that flattens the matrix to a vector and then sends it to a fully connected layer with 128 neurons and a rectilinear activation function. This fully connected layer outputs to another fully connected layer containing 50 neurons and then finally to an output layer containing 10 classes and a softmax activation function to output probability-like predictions for each class.

5 DELIVERABLE 4

Now that the neural network has been built, it is important to observe how well it performs with actual pictures instead of the MNIST dataset. Hence this deliverable involved applying the neural network model we built in the previous deliverable to observe if it's able to correctly predict what digit the user scribbled on a simple software called paint, after a screenshot of the same is taken by a python program. Once the screenshot is taken, the image is downscaled and then stored in a folder, the predicted model grabs the image from that folder and supplies it to the CNN model so that its label can be predicted.

6 WORKING WITH KERAS, THEANO AND TENSORFLOW TO PLAY ARCHON:

In deep learning, a brain-inspired architecture is used, in which connections between layers of simulated neurons are strengthened on the basis of experience. When Archon is being played by program initially in an emulator, where it makes random decisions, screenshots are taken every one tenth of a second [1]. These screenshots are then converted into matrices and then supplied to the training model which is built using the functionalities of the library called Theano. This model is a simple multi-layer neural network. Then comes the application of reinforcement learning, which is a decision making system inspired by the neurotransmitter dopamine reward system in the animal brain. Using only the screen's pixels and game score as input, the algorithm learns on its own by trial and error, which actions (whether to go left, right or up) to take at any given time to achieve the maximum reward [4]. We observed that out of the three libraries mentioned above, Keras was the most efficient library in predicting the MNIST images.

7 CHALLENGES FACED DURING THE TRAINING

Learning to control agents directly from high-dimensional sensory inputs like vision and speech is one of the long-standing challenges of reinforcement learning. Extracting raw pixels from millions of images to form vectors and thus training using CNN takes a lot of computational power and GPUs are needed at some point. We performed the training using a CPU and thus faced problems as the training was initially slow. Then we changed the computation of feature maps and thus saved time there.

Another challenge is that the agent only sees the pixels and the rewards, similar to a human player [3]. Using just this information, it is able to successfully play the game at a human [4] or sometimes super-human level.

8 CONCLUSION

In this paper, we have talked about performing experiments to train the machine to play Archon on its own using deep neural networks. The ability of the deep learning model, to master difficult control policies using only raw pixels as input, will be demonstrated. We plan to use convolutional layers and implement them using a library called Keras. The images used to train the machine will be too many as we will take screenshots every one tenth of a second, thus we have thought of down sampling them, apply CNN and then discard the images after computing the relevant functions.

References

[1] M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47:253–279, 2013.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. DeepMind Technologies, 47:253–279, 2013.

[3] S. Russell and P. Norvig, "Decision making" in Artificial Intelligence: A Modern Approach, 3rd ed. USA: Pearson Education, 2003.

[4] S. Russell and P. Norvig, "Making Complex Decisions" in Artificial Intelligence: A Modern Approach, 3rd ed. USA: Pearson Education, 2003.

[5] M. Stevens and S.Pradhan, "Playing Tetris with Deep Reinforcement Learning", Stanford

[6] R. Sutton, "Reinforcement Learning" in Reinforcement Learning: An Introduction, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[7] C. Watkins, “Technical Note: Q-learning,” Kluwer Academic Publishers, Netherlands, 1992, pp 279-292.

[8] Deep Learning Tutorial, by LISA Labs. (2015, September 01). [Online]. Available: http://deeplearning.net/tutorial/deeplearning.pdf.