Jacob Centner

COSC 5550

Final Project Report

12/17/2018

**Introduction**

For my AI final project, I attempted to write a program that would play the game “Chrome Dino.” As a reminder, Chrome Dino can be accessed in the Google Chrome browser by navigating to “chrome://dino” or by attempting to use the browser when there is no internet connection. The game randomly sends cactus and vultures at the player which must be jumped over. The pace of the game increases as time goes on, making it necessary to carefully time each jump. The longer a player lasts, the higher score they receive. Also, the game is played, the environment changes from “day” to “night”, causing the black-on-white graphics to invert to white-on-black.

My goal for the project were, first and foremost, to learn. I had never before attempted a project of this kind and I wanted to make full use of the opportunity to learn more about AI. Besides that, I intended to write a program that would learn to play better than myself.

**Methodology**

The approach I took was to use image processing to capture the current game state and train a convolutional neural network to perform binary classification on the game state. The categories were: 0 (don’t jump) or 1 (jump), where the model output a single value based on the determined probability of each category. After the model was trained, a separate python script captured the game state, fed it to the model, and used the output to determine whether or not to jump based on a configurable threshold (set to 0.20 for the best results).

Training data for the model was created by recording my input as I played the game, sans any mistakes I made. I used a library called ‘keyboard’ to record key input in real time, which was used to sort the images into two folders: don’t jump (0) or jump (1). Approximately 17,000 training images were used in total, of which around 2,000 were used as a validation set. The process was not completed automated, requiring that I manually delete some images after I lost each game and reset a counter used to uniquely name images. This part of the project is in the file, “DinoRunner Training Data Generation.ipynb”.

I used the TensorFlow GPU library with the Keras API to create and train the CNN. The architecture of the CNN consisted of alternating convolutional and pooling layers (4 of each), the output of which was flattened, subjected to dropout (factor 0.5), and then classified with two densely connected layers and output through a single neuron. The convolutional layers and densely connected layers used the relu activation function, while the output layer used sigmoid. Full details of the CNN are available by examining the attached code (see “DinoRunner Train Model.ipynb”).

Additional libraries were used to accomplish secondary functions. Image processing was performed with OpenCV. I significantly modified the original capture of the game state in order to reduce image size, eliminate nonessential features, and otherwise preprocess the image to be used in the training data. Additionally, I performed limited data augmentation with OpenCV. To interact with the web page, I used Selenium to launch Chrome and navigate to the proper URL. I used the library pyautogui to send key input to the page.

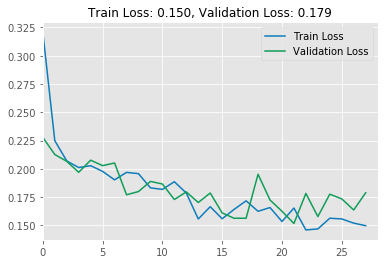
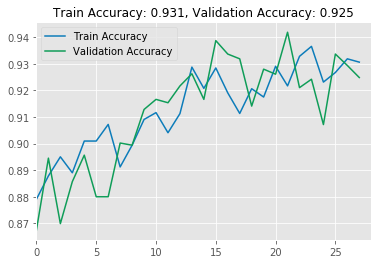
**Experiment**

Overall, a number of issues in my approach prevented me from reaching optimal performance. While my method for collecting training data was far from the optimal approach, it did the job for my purposes here. It is one of the first things I would change. The data was unbalanced, as there were many more instances of “don’t jump” than “jump.” Also, the AI certainly learned to have my own human reaction delay, which I believe ended up being the limiting factor in its performance. Analyzing the A while it plays reveals that eventually, it begins to react too slowly to objects that approach. This may also be due to my lack of training data past scores of around 2000, as that is as far as I reached.

Besides training data, I also think my CNN architecture could use refinement. I experienced severe overfitting problems throughout the project, and while I eventually reduced them, I think my model is still too complex for the task. Simplification could be performed by using fewer convolutional/padding layers, or by decreasing the density of some or all the hidden layers. My training accuracy and lost graphs for one of the most recent models follow below. Overfitting may have occurred around epoch 17 in this example; I never fully resolved it.

In the end, I did achieve what I set out to. The best score my AI reached was **2645**, which was better than any score I achieved on my own (exceeding my personal best by ~25%). This performance is relatively consistent; it achieves 2000 or better score almost every attempt with the model included in the attached code. With additional fine tuning and better data collection methods, I think the AI could achieve much higher scores.

I also learned much more than I intended to. I feel comfortable using TensorFlow GPU, Keras, and OpenCV, three resources I had never before used. My python skills improved substantially as a result of this project. Finally, I understand far better the ins and outs of using CNNs and applying them to real-world situations. Should I ever revisit this project, I would like to try again using reinforcement learning rather than a CNN. Reinforcement learning seems to be the more successful approach to playing video games using AI. Since the current score of the game is an easily identifiable reward, and the game only requires a single input, a reinforcement learning-based AI should be relatively simple to implement.



Included with this report: My code (three Jupyter notebooks), a copy of my PowerPoint presentation, a recording of the AI running (in .gif format), and a copy of my initial proposal.

Note: my programming environment was uniquely configured for this project. To run the code in DinoRunner AI Program (which uses the model to play the game), a python 3.6 environment with certain versions of TensorFlow GPU, Keras, Selenium, OpenCV, Pillow, pyautogui, and numpy installed must be used. I did not design this project to be easily portable; if you would like a demonstration of the code running, I would be happy to schedule a time to meet with you. Sorry about that!