**Final Submission**

1. Introduction

We started with the idea to train an AI that plays the ***T-Rex Run*** (Google Chrome Dino game) using Q-learning.

The final state of the project consists of a two-fold implementation. First, we use an Object-detection model which does real-time screen capture and gives out the coordinates of the objects on the screen. These coordinates are used to create game state which is then used to train the Q-learning model. We have used Q-learning with Widrow-Hoff rule.

1. Topics

* Use an existing trained object recognition for our game
* Perform basic pre-processing operations on images
* Getting your game to interact with a real game via an emulator
* Q-learning
* Experiments to evaluate AI approach

1. Team

The team members were: Himanshu Budhia (<https://github.com/budhiahimanshu96/>) and Amitesh Deshpande (<https://github.com/amiteshdeshpande/>).

1. Instructions

* Set-up TensorFlow and object detection environment in the system (GPU preferred).

You may follow this link: <https://bit.ly/2UWmIOb>

* Install pyautogui and selenium libraries if not already present.
* Once this is installed and tested, clone or download the project repository from GitHub using this link: <https://github.com/amiteshdeshpande/GoogleChrome-DinoAI>
* There is already an object-detection model trained for our game in the ‘dino\_chrome’ folder with the file ‘[frozen\_inference\_graph.pb](https://github.com/amiteshdeshpande/GoogleChrome-DinoAI/blob/master/dino_chrome/frozen_inference_graph.pb)’. This file will be now used while playing the game.
* Chrome driver also needs to be installed for the next steps. It can be found here: <http://chromedriver.chromium.org/downloads>
* After installing, you need to set the chrome driver executable path in the file: ‘GameModule.py’.
* Now, there is also a trained weights file for the Q-Learning model from our implementation available in the ‘Checkpoints’ directory with name ‘weight\_matrix.npy’.
* To run the implementation, open terminal and run the ‘Test.py’ file (make sure to use the TensorFlow environment installed in the step 1).

1. Systems

In the project, the main implementation is of the Q-Learning model using the Widrow-Hoff rule. To implement this model, we used the technique and steps taught in the class.

Q learning update rule:

Q(a,s) = (1-α)Q(a,s) + α(r + γ maxa’ Q(a’,s’))

where:  
 α is the learning rate (0.01)

γ is the discount factor (0.1)

r is the reward (-10 for death)

maxa’ Q(a’s’) is the best action’s Q-value in the state reached from the previous state

Using Widrow-Hoff rule our q-functions are of form:

Q(a,s) = Σ waijvij

where vij is +1 if there is a obstacle in square (i,j) in our 1x6 environment, and -1 if the space is clear; and waij is a learned weight specific to an action *a* and a square *ij*.

The most important design decisions to take were the creation of game state for the Q-Learning model, and the reward system which will guide the AI towards learning how to play the game. The state was created based on the screen capture and the location of the obstacle detected on the screen using object detection. The acceleration of dino was set to be constant so that the game speed does not hinder with object detection process and to maintain a balance between them.

1. Failures and setbacks

One of the most difficult part of the project was set-up of object detection environment. There were a lot of dependencies to take care of for the installation of object detection library. We had to consult many different tutorials and videos for this to work. Even after the set-up, the training of the model took nearly 2 complete days. We were able to achieve good results for the object detection as shown in the document: object\_detection\_result.pdf.

When this model was trained, our next goal was to detect object in real-time. Initially, when we implemented this real-time, the average time to detect objects in each image was nearly 4s which was not feasible for the further work. We worked on this by making some changes in the conversion of image to NumPy arrays. This helped us do the detection in 3s which was still not good enough. After scratching our heads, we realised that we were making a new TensorFlow session for each image, which was itself taking a lot of time. Finally, after fixing the issue, we were able to get the real-time object detection time for our game to as low as 10ms.

Next, we faced some major setbacks when we started to implement Q-learning model. The implementation was straight forward but the training results were not as expected. Initially, we had a 4x4 environment and a reward system based on both positive and negative rewards. This did not give good results. The weight matrix was very random for every step trained. We also tried making the reward only negative (when the dino died) which did improve the results sometimes but overall, this was a failure.

On further brainstorming, we thought the reason for poor results could be the environment size. The 4x4 environment had many states which were never going to affect the action of dino (like the region exactly above the dino). So, we tried a new environment of dimension 1x16. But still the results were poor (trying different permutations of rewards as well). We also found that variables ‘maxQ’ and ‘bestmaxQ’ should be set to a negative value. The reason being, q values of environment were being calculated as negative as there is no positive reward, so when we initially set ‘maxQ’ and ‘bestmaxQ’ to 0, the best move was always being selected as ‘do nothing’.

We now decided to work with just the 1x1 environment which would be right in-front of the dino to make sure if it able to understand the environment and our understanding is correct or not. This approach gave very good results. We were able to achieve a high score of approx. 300, which gave us some hope to work towards refining the result further.

In the end, we realised that the ‘ducking’ feature of the dino was unnecessary (the dino can jump instead of ducking). We changed the number of possible actions to only 2 now (jump or do nothing). We also finalised on a 1x6 environment with every 1x1 block maintaining the state of 50 pixels of the screen.

1. Successes

After the final implementation, we trained the model for 500000 steps. We obtained a very good weight matrix. On testing the game using these weights, we observed a high score of 706 for the game.

The game changes from day time to night time and the object detection was trained for both the scenarios. But we did not factor in the transition time between them. So, the object detection seemingly fails here because every time there is a transition, we can see the dino dying.

As some future work, we can see this being taken forward to be trained over more sets of states in terms of object detection. There is also a possibility to make the model completely based on CNN. We could pass the entire image as the input and the model will learn based on the truth values of the labelled images.

1. Evaluation

Our experimentation with AI was based on trying different rewards and environments. We tried a combination of 4x4 environment, 1x16 environment, 1x1 environment, and 1x6 environment with negative reward (only on dino crashing), both positive-negative reward (whole numbers), and both positive-negative reward based on current game score.

|  |  |
| --- | --- |
| **Environment** | **Max Score** |
| 4 x 4 | 175 |
| 1 x 16 | 230 |
| 1 x 1 | 320 |
| 1 x 6 | 706 |

Q-learning tested for at least 5 games

1. Lessons learned

We were able to understand the object detection using TensorFlow and dependencies management in the installation of the same. We understood the importance of the environment in the Q-learning model and how it impacts the correctness of training. Also, we realised that the Q-learning is very much dependant on the game or application (we initially implemented model similar to Frogger’s model in assignment 5, which essentially failed). We were able to appreciate the role of reward in the learning of weights for q-learning model (the rewards need to be adjusted such that the agent gets prize only when necessary and not for all small/wrong actions).