T-Rex Game using Artificial Intelligence

Gaurav Aidasani, Tanmay Maheshwari, Neha Shah
Team 3

November 25, 2018

1 Introduction and Overview

Games and artificial intelligence have a long history together. Much research on Artificial Intelligence in games is concerned with constructing agents for playing games, with or without a learning component. In this project, we propose to use Deep Q-learning and online learning Multi Layer Perceptron (MLP) for learning to control the game agent in T-Rex, the game embedded in Chrome offline mode. [1]

About The Game: The T-Rex in the game is an infinite runner, which can jump over cacti, and dodge underneath obstacles. Controls are basic. Space to jump and the down arrow to duck. The game speeds up with passage of time. The goal of the game is to survive for as long as possible.

Related work

In 2013, Google Deepmind proposed the use of deep reinforcement learning on training agents to play the 2600 Atari games [2]. Taking just the pixels and reward received from the game as inputs, they were able to reach human-expert performance in multiple Atari games. The main advantage of Deep-Q learning is that no specification of the game is needed in spite of the high dimensional image input. The agent is able to learn to play the game without knowing the underlying game logic. This framework is model-free and can generalize to a lot of similar problems.

2 Methods

2.1 Preprocessing for obtaining image features

We aim at extracting pixel-based features from the image. The features involve the bounding boxes of T-Rex and obstacles,and the status of the T-Rex (whether the T-Rex is jumping). We used OpenCV, an open source computer vision tool for pixel-based feature extraction. These features come from an intuitive understanding of how a human agent would play the game: identify the dinosaur, obstacles, their relative positions, and then decide the next action for T-Rex.

Background Filtering: The first step in extracting pixel-based features from the screen shot is to filter out useless pixels in prediction, such as the horizon and clouds. We also convert the image into grayscale to reduce the state-space.

Object Detection: After filtering the background, the objects can be easily identified. We use OpenCV to detect the contours of the objects and find the corresponding bounding boxes. Each bounding box is represented by the (x,y) co-ordinate of the upper-left corner as well as the width and height.

Object Classification: Given the bounding boxes, it is important to determine the type of each object. In our model, there are four types of cacti and one dinosaur. The classification of the object is done by template matching with the help of OpenCV.

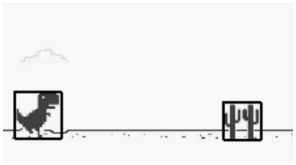

Figure 1: **Detection of T-Rex and Cactus**

Object Tracking: We track objects from frame to frame in order to measure the derivatives of each object on the screen. By comparing the position of the detected object in two adjacent frames, we calculate the moving speed for each obstacle.

2.2 Preprocessing for Q-Learning

In Q-Learning model, we apply the standard preprocessing in Atari games according to [2]. Firstly, we convert the image to grayscale, and then resize the image to 80x80 grid of pixels. Finally, we stack the last 4 frames to produce an 80x80x4 input array for the Deep Q-Learning network.

3 Models

3.1 Baseline

Given the features extracted from the screen, we implement a baseline imitating the human players, where the T-Rex jumps whenever the nearest obstacle is close enough (less than 120 pixels). This baseline follows the intuitive greedy player strategy.

3.2 Multi-Layer Perceptron

We built an online learning Multi Layer Perceptron (MLP), which takes the pixel-based features as input and predicts the optimized jumping position ahead of the obstacles. The features of the Multi-Layer Perceptron are as follows: distance between cactus and T-Rex, height of the obstacles and width of the obstacles.

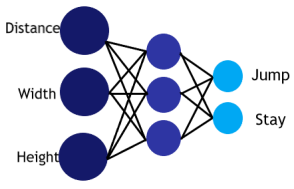

Figure 2: **MLP Architecture**

Online learning uses information one record at a time. The neural network continuously gets a record and updates the weights until one of the stopping rules is met. If all the records are used once and none of the stopping rules are met, then the process continues by recycling the data records.

3.3 Deep Q-learning

Q-Learning: We have used Q-learning, a technique of Reinforcement Learning, where we try to approximate a special function which drives the action-selection policy for any sequence of environment states. Q-learning is a model-less implementation of Reinforcement Learning where a table of Q values is maintained against each state, action taken and the resulting reward.

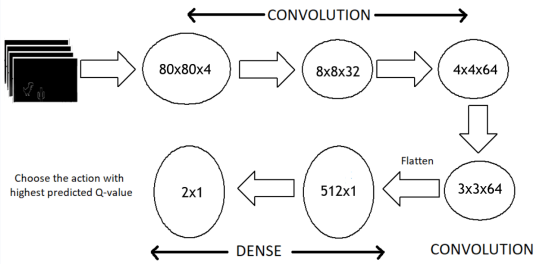

Figure 3: **CNN Architecture**

Training : The agent is trained so that it achieves a score as high as possible. The agent learns which objects in the image are harmful (cactus and birds) and which are not (clouds). The reward function is defined by giving a negative value whenever the agent dies. The agent is trained with the help of a convolutional neural network.

Initialization:

1. Start with no action and get initial state

2. Observe game-play for specific number of steps

On every iteration:

1. Predict and perform an action

2. Store experience in Replay Memory

3. Choose a batch randomly from Replay Memory and train model on it

4. Restart if game over

4 Experimental Analyses

Comparison between different models:

In this project, we have noticed the game velocity has a huge impact on the performance of the models.

When the game is running with constant velocity, all three models perform better than any human being. However, acceleration of the game affects the Baseline and the MLP models. Due to increase in speed, the Baseline model is unable to jump at the appropriate time and the T-Rex runs into obstacles.

In the online learning MLP model, the acceleration results in different data records which affects its entire neural network.

Results

From Fig. 4 and Fig. 5, it is clear that the Q-Learning model requires a huge amount of training time. After 200 iterations

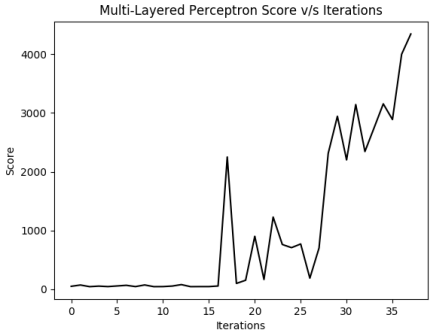

Figure 4: **Learning Curve of MLP**

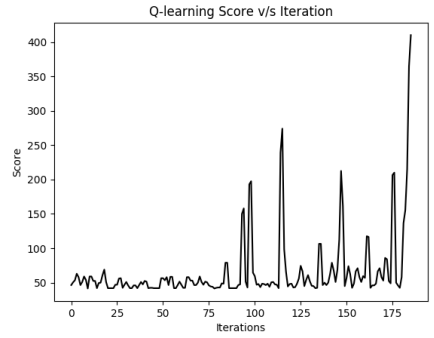

Figure 5: **Learning Curve of Q-Learning**

the maximum score of the Q-Learning model is 400 whereas the MLP model shows a high score of approximately 4000. However, if it was to be continuously trained for one month or more, it would be unbeatable.

5 Discussion and Future Directions

The models successfully learn to play the T-Rex game from the pixels and reward, achieving high performance in constant speed scenarios. However, in game with acceleration, the models have a hard time capturing velocity change.

For the MLP model, speed could be added as an additional feature for the neural network. For Q-Learning, specially designed training method can help us overcome the training difficulties caused by the properties of our game, which further improves our model's performance and helps achieve superhuman results.

Generic models can be implemented for similar type games such as Flappy Birds.

References

[1] Google Chrome *Dino game* chrome://dino/

[2] Volodymyr M., Koray K., David S., Alex G., Ioannis A., Daan W. and Martin R.2013 *Playing Atari with Deep Reinforcement Learning*[online] DeepMind Technologies [cited 19 December 2013]. Available from World Wide Web: (https://arxiv.org/abs/1312.5602v1).

[3] Reinforcement Learning Explained *Q-Learning* https://ai.intel.com/demystifying-deep-reinforcement-learning/