AI for Chrome Offline Dinosaur Game

ABSTRACT

In this project, we implement both feature-extraction based algorithms and an end-to-end deep reinforcement learning method to learn to control Chrome offline dinosaur game directly from high-dimensional game screen input. Results show that compared with the pixel feature based algorithms, deep reinforcement learning is more powerful and effective. It leverages the high-dimensional sensory input directly and avoids potential errors in feature extraction. Finally, we propose special training methods to tackle class imbalance problems caused by the increase in game velocity. After training, our Deep-Q AI is able to outperform human experts.

Keywords

Deep Q-Learning; MLP; Feature Extraction with OpenCV

1. INTRODUCTION

Learning human-level control policies directly from high-dimensional sensory data (such as vision) is a long-standing challenge for controlling system design. Many classical control algorithms are based on accurate modeling or domain knowledge of the underlying dynamics in the system. However, for systems with unknown dynamics and high dimensional input, these methods are unrealistic and hard to generalize.

In this project, we propose Deep Q-learning, online learning Multi Layered Perceptron (MLP) and rule-based decision-making (Human Optimization) for learning to control the game agent in T-Rex, the classic game embedded in Chrome offline mode, directly from high-dimensional image input. All methods are based on two ways of state-space simplification. The first way is to extract pixel-based features and detect objects using computer vision approaches for imitating human player, such as MLP. The other is to automatically learn patterns from resized raw game image without manual feature extraction, such as Deep Q-learning. To be specific, the input to Deep Q-learning algorithm is a stack of 4 images. We then use reinforcement learning to update the training samples in neural network and leverage a Convolutional Neural Network(CNN) to predict which action to take under given circumstances. Similarly, we take the extracted pixel features as input and use MLP or rule-based decision-making (Human Optimization) for prediction. The results reveal that our approaches significantly outperform even the experienced human player in the game.

This is a joint project for CS221 and CS229. For CS221, we implemented the extraction of pixel-based features from the game screenshot and the MLP algorithm. We also focus on the implementation of Q-learning framework, and the comparison between hand coded online learning methods and deep reinforcement learning. For CS229, we focus our efforts on the deep Q-learning network model, the choices of hyperparameter of training, the special training method and analysis of our training process. We also analyse the effect of acceleration in T-Rex and compare our game with other games.

2. RELATED WORKS

The well-known story of TD-gammon is one of the milestones in reinforcement learning. It used a model-free TD-learning algorithm similar to Q-learning and achieved human-expert level performance [4]. Since then, the use of reinforcement learning has popularized and various attempts have been made to apply reinforcement learning on games. In 2013, Google Deepmind proposed the use of deep reinforcement learning on training agents to play the 2600 Atari games [3, 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 dynamics 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. To process the image data, they use a deep Q-network (DQN) to directly evaluate the Q function for Q-learning. An experience replay is also applied to de-correlate experiences. This framework is model-free and can generalize to a lot of similar problems. After their research, many papers tried to make improvements. Further improvements involve prioritizing experience replay, more efficient training, and better stability when training [2].

3. DATASET AND FEATURES

Our game is implemented in Pygame, a handy python game package which allows us to extract game screen shots at each frame. Without touching the underlying dynamics of the T-Rex game, the state-space would be a set of screen shots in the form of 1200x300 grid of RGB pixels. However, modeling this large state-space directly is difficult and computationally expensive. Thus, we apply preprocessing to raw pixels for data filtering and feature extraction. Different preprocessing procedure is adopted in different algorithms.

3.1 Preprocessing in Q-Learning

In Q-Learning model, we applly the standard preprocessing in atari games according to [3]. Firstly we convert the image to grayscale, and then resize the image to 80×80 grid of pixels. Finally we stack the last 4 frames to produce an 80x80x4 input array for the Deep Q-Learning network.

3.2 Preprocessing in Pixel Feature Based Algorithms

In other pixel feature based algorithms (Human Optimized and MLP), we extract pixel-based features from the raw game image pixels for state-space reduction. The features involve the bounding boxes of T-Rex and obstacles, relative moving speed of obstacles as well as the status of the T-Rex (whether the T-Rex is jumping, dropping or ducking etc). We used OpenCV [1], an open source computer vision tool for pixel-based feature extraction. Those features come from an intuitive understanding of how a human agent would play the game: identify the dinosaur, obstacles, their relative positions and velocities, and then decide the next action for T-Rex. The details are as follows.

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

Figure 1: Image after filtering background

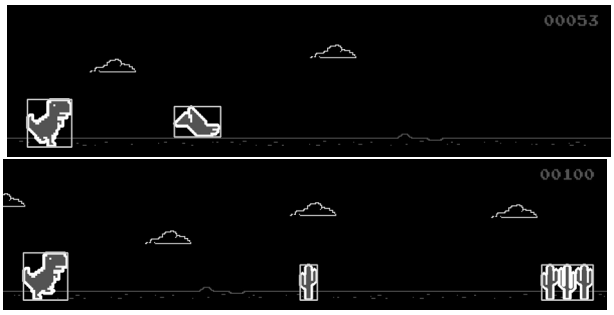

Figure 2: Results for bounding boxes and objects classification. a)Classified as T-Rex and bird; b) Classified as T-Rex, Cactus and Cactus.

grayscale to reduce the state-space. Figure 1 shows an example image after filtering the background.

Object Detection

After filtering the background, the objects are easily separated. We leverage OpenCV to detect the contours of the objects and find the corresponding bounding boxes. Each bounding box is represented by the x, y positions of the upper-left corner as well as the width and height.

Object Classification

Given bounding boxes, it is important to determine the type of each object. Since there are only three kinds of sprites (T-Rex, cactus, bird), we can use the width and height of the bounding boxes to classify the objects. We've compared the object detection results with the ground truth fetched from the game engine. Results show that our algorithm can accurately detect the objects using raw pixel input (almost 100%).

Object Tracking

The final step of pixel-based feature extraction is to calculate the speed of the obstacles and whether T-Rex is jumping. We track objects from frame to frame in order to measure the derivatives of each object on the screen. The tracking algorithm is a simple greedy matching of the nearest bounding box with the same type. By comparing the position of the detected object in two adjacent frames, we calculate the moving speed for each obstacle. Furthermore, by calculating the vertical speed of the T-Rex, we can infer whether the T-Rex is jumping up or dropping down.

4. MODELS

4.1 Rule-based Decision-making (Human Optimization)

In this method, we imitate the experienced human players and learn their greedy policy. Intuitively, human player would consider the ratio of the height of the obstacles and the width to get the optimized jumping position.

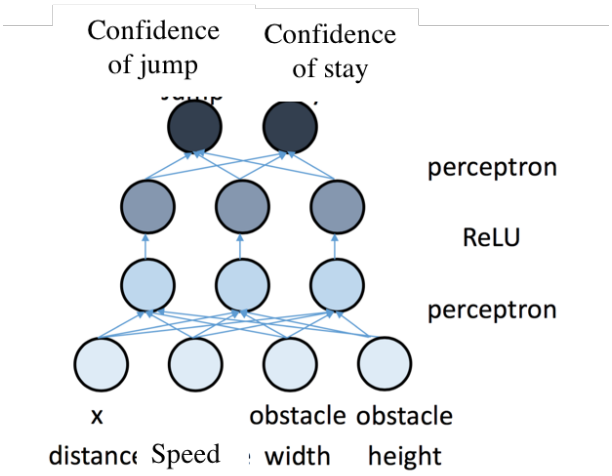

Figure 3: Structure of Multi Layered Perceptron

In addition, they would consider relative velocity and the distance from the obstacles in order to infer the security distance given their reaction capacity. Therefore, we manually tune these parameters to acquire the most reasonable jumping position ahead of the obstacles, given their distances, relative velocity as well as the height and width.

4.2 Multi Layered Perceptron

We propose an online learning Multi Layered Perceptron (MLP), which takes the pixel-based features as input, to predict the optimized jumping position ahead of the obstacles.

In this work, we leverage the pixel-based features extracted by our image processor using OpenCV, including the information of a list of obstacles in front of the agent. For a single obstacle, such as a cactus, we extract at least four features, including 1) its distance from the agent, 2) its height, 3) its width, and 4) the relative velocity to our agent. The features then go through a multi-layered perception network which conceives decisions about whether to jump or stay on the ground under current circumstances.

We use online training to improve the performance of MLP. When the agent hits obstacles while jumping, we assume it should have jumped ahead of the position in which it takes off. Comparatively, when the agent hits obstacles while dropping, we infer there should have been a delay in the current jumping and the agent should have stayed on the ground on the current jumping points. In addition, if the agent crashes into the obstacles while staying on the ground, we assume it should have jumped on the latest movement.

4.3 Deep Reinforcement Learning

In this section, we will discuss MDP formulation, the basic idea and parameters of our algorithm.

MDP Formulation

We choose model-free Q-learning algorithm as our reinforcement method so that we don't need to build a complex MDP model and learn the transition probability. The action space of our MDP model is that the agent either do nothing($a = 0$) or jump($a = 1$). Note that when the T-Rex performs jumping, it loses the ability to control itself

for a while and keeps flying until it reach the ground again. This property is quite different from a lot of other Atari games, where the agent can control itself any time in the whole game. In order to capture the speed of objects, we input four frames' images as the state of MDP. The discount factor γ is chosen to be 0.95 to make the Q converge faster. The reward function is defined as giving a negative value whenever the agent dies. The value is set to -100.

Q-learning Method

When we train our agent using Q-learning method, the agent takes action under certain policy and gets series observations from the game, which is a path shown as:

$$\{s_0; a_1, r_1, s_1; a_2, r_2, s_2; ...; a_n, r_n, s_n; \}$$

If the state space is discrete and the transition is known, for each (s, a, r, s) tuple, the Q value of each action can be updated using the Bellman equation:

$$Q_\pi(s,a) = \sum_{s'} T(s,a,s')[Reward(s,a,s') + \gamma \max_{a'} Q_\pi(s',a')]$$

When transitions are unknown, we can directly update Q using every single observation tuple (s, a, r, s'):

$$\begin{aligned} Q_{opt}(s,a) &\leftarrow Q_\pi(s,a) - \eta[Q_{opt}(s,a) - u] \\ &= Q_{opt}(s,a) - \eta[Q_{opt}(s,a) - (r + \max_{a'} Q_{opt}(s',a'))] \end{aligned}$$

Due to the high-dimensional image input, the state will be really large and impossible to learn. So we use a Convolutional Neural Network to functionally approximate the Q value of each state. Assuming the weights of CNN is w, the learning process is then equivalent to optimize the cost for all observations:

$$\min_w \sum_{(s,a,r,s')} (Q_{opt}(s,a) - (r + \max_{a'} Q_{opt}(s',a')))^2$$

The optimization can be done using the backward propagation training method of CNN.

Batch Training

If we train our CNN at every single frame, the training may suffer from several problems. Since the state at time t is highly correlated with state at time $t+1$, gradient descent after consecutive steps will cause erratic updates making the training very slow. In order to fix this problem and make the training faster as well as more stable, we use the batch training method. During training, we maintain a memory with capacity 50000 and save the observations at each frame. Then after the observation phase we would choose a batch of data from the memory randomly to train our CNN. This experience replay strategy makes the training much more efficient and the experience observed are no longer highly correlated.

Training Parameters

If we choose the optimal action every time, the Q-learning training may converge to a local optimal policy. In order to make the agent explore more states and policies, we use the ϵ-greedy method to do random action under some probability. The value of ϵ decreases linearly overtime. The choice of initial ϵ and the total steps of exploring are essential for our training because our game is quite different from other games: First, the agent can not control itself and take any action when the T-Rex has already jumped into sky. As a result, even a small ϵ will make the agent die very fast. Second, the difficulty of T-Rex game increases gradually (by changing speed of T-Rex). After we have decreased the ϵ and kept the T-Rex alive for a long time, the agent can not explore any more during the high speed mode of game. This contradiction makes the regular training method with linear decreasing ϵ not so effective and we need to find some new training method for our model, which will be explained in the following section.

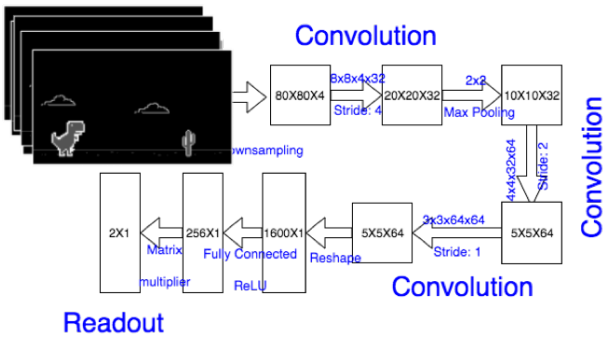

Figure 4: CNN architecture

CNN Architecture

The CNN used in our reinforcement learning contains three convolution layers(with ReLU layers follow them), one max pooling layer and two fully connected layers. The four images first go through a convolution layer with 32 filters of size 8×8 with stride 4, followed by a ReLU layer. Then a 2×2 max pooling is applied to the output of convolution. The tensor then go through two convolution layers with 64 filters of size 4×4, stride 2 and 64 filters of size 3×3, stride 1. Finally the flattened tensor go through two fully connected layers and outputs the Q values for two actions.

5. EXPERIMENT AND DISCUSSION

5.1 Settings

We leverage the online learning method to collect the training data and fit our model iteratively. For those ruled-based methods without training, we manually tune the important parameters and run for certain rounds to test the robustness of the methods as well as the performance.

Initially, there is acceleration in T-Rex's relative velocity. We test all the baseline and proposed models under three scenarios, including the ones 1) with normal acceleration and velocity ranging from 6 and 13, 2) fixed velocity of 6 and 3) fixed velocity of 9. We find that game velocity has a significant impact on the agent's performance. For instance, in those rule-based baseline methods, the length of reaction window is strictly related to the velocity. Once the agent reaches a high speed, it is more likely to run into the obstacles so the jumping should be scheduled earlier. In addition, for online learning model, the acceleration results in an imbalanced class of different speed during the training process which we would discuss later.

For evaluation, we report the averaged scores and the standard deviation.

5.2 Baselines

We compare our Deep Q-learning methods and MLP supervised learning methods with following three baseline.

Keep-Jump

Keep-Jump is a naive model in which the agent keeps jumping regardless of its distances from the obstacles nor the attributes such as the height and width. This baseline represents the performance of those agents without artificial intelligence over the game.

Human Optimized Approaches

Given the features extracted from the game image, we implement one baseline imitating the human players, where the T-Rex jumps whenever the first obstacle is close enough (less than 200 pixels). This baseline follows the intuitive greedy player strategy. It only considers the closest obstacle on the screen.

Human Player Records

We invite some experienced players who hold records of 1000 points in T-Rex games and ask them to compete with our agents. In this way, we could see different decision making process between human and AI as well as their performance.

5.3 Comparison with the Game Flappy Birds

We notice, there are some prior CS229 projects on using deep Q learning to train Game AI, including some interesting ones for the game Flappy Birds. However, our game is much more difficult than theirs and we propose better solution comparatively as follows.

Firstly, the state-space are more constrained in game Flappy Birds, which makes it easier to coverage in Deep Q-learning. For instance, the distance between adjacent pipes are fixed and the bird travels at fixed speed rates. In this way, states could be discretized manually given the relationship between velocity. However, in T-Rex, the positions of the obstacles are random and the velocity is increasing with acceleration. In this way, class imbalance would occur and needs to be handled carefully. Moreover, since it is guaranteed that the bird could trespass the space from top to the bottom between two adjacent pipes, only the nearest obstacles need to be considered. In comparison, in both MLP and Deep Q-learning, we need to consider a list of obstacles because the difference in jumping position would result in different landing position affecting the reaction window for the following obstacles.

One of the significant difference is that, in Flappy Birds, the agent is allowed to adjust gesture and take additional actions to avoid the obstacles while in T-Rex, once the jumping position is determined, so would the results. There is no additional action allowed while jumping to avoid the upcoming obstacles so the difficulty increases.

Last but not the least, the only obstacles in Flappy Birds is pipes. In T-Rex, there are different types of obstacles (cactus and birds) in different height and width. In addition, even a same obstacle with different velocity would have totally different impacts on the decision-making process. Thus, we believe there are more challenges to tackle within our case.

5.4 Online Learning MLP

In this section, we analyze the results of MLP model. Given a list of obstacles to learn the optimized jumping position, we notice the average scores cease to increase over 900 rounds iteration. In addition to the deviation of image processor, we believe the overwhelming computation over parameters in a fully-connected network undermines the performance as well. To support it, we conduct a similar experiment with less obstacles information. Illustrated in Fig 3, the average scores increases significantly and standard deviation decreases in line.

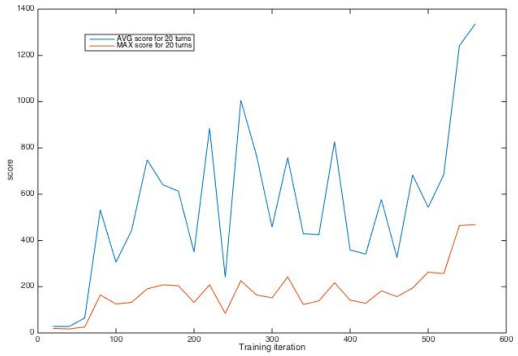

Figure 5: Training curve of MLP

5.5 Deep Q-learning

As is stated in previous sections, the velocity of the game has a significant impact on the performance of Deep Q-learning algorithm. We trained our Deep Q-Learning model in the game with acceleration and the game with constant speed. The comparison of these two scenarios is illustrated in Figure 6 a). When learning to play the game with constant speed, Deep Q-learning is able to achieve much higher performance within fewer iterations than learning to play under varying speed. Therefore, we need to find a new training method to effectively learn to play the game under acceleration.

In traditional exploit-exploration strategies for Deep Q-learning, ϵ value decreases overtime to make sure the agent explore more states at the beginning stage under a large ϵ and then gradually decrease the ϵ to ensure the final online policy converge to the optimal policy. However, since our T-Rex game accelerates gradually and the agent need to deal with different speed modes in a single round of game, the regular training method for Deep Q-learning has some problems. This can be explained by our learning curve shown in Figure 6 b). Firstly when we use the regular training method (shown in TRAINING PHASE I, with blue background in Figure 6 b), the learning curve flattens after 1.3 million turns of training, where the average of 20 test turns is about 750 and max score of 20 rounds is 1000. The error analysis shows that under most cases, the T-Rex dies due to the random actions (Even though our ϵ value begins from a small value of 0.1). As is explained in previous section, this is because the agent can not control itself while jumping. So even a small ϵ will result in T-Rex's random jump. Once it jumps, the agent loses the ability to control, and is likely to run into the adjacent obstacles under circumstances with high velocity. To solve this problem, we introduce a TRAINING PHASE II. We make the ϵ equals zero, meaning no exploration. Then the performance of our algorithm continues to increase and reaches an average score of 1000 and max score of 2000 for 20 test turns.

However, difficulty level changes with the increasing velocity. In TRAINING PHASE II, only when we finish the exploration stage can we reach the high speed mode. In this

Algorithm	Average 20	Max	Std
Keep-jump	41	111	23
Human-expert	910	1500	420
Human-optimized	196	4670	134
MLP	469	1335	150
Deep Q-learning	1216	2501	678

Table 1: Comparison between different models

way, class imbalance occurs with far fewer training items in high speed cases comparatively. The imbalance would make the agent even less likely to respond to high speed mode after 1000 scores. In addition, ϵ-greedy prohibited the agent from getting higher scores. It is because random jump would result in unexpected crash into the obstacles. However, we should not cancel the ϵ-greedy under high-speed mode entirely. Otherwise, we can play to the high speed mode to cover the imbalance cases but no exploration occurs under this situation.

To make the agent not only see the high speed mode but also keep the exploration, we use another training method in the TRAINING PHASE III. We train our agent with large acceleration for 0.5 million iterations and keep the ϵ-greedy at the same time. During these iterations, the agent will observe a lot of high speed samples. Subsequently, the model is retrained for an extra 10000 iterations under normal acceleration mode to get used to the original game. This training method largely increases the performance of our algorithm and make the average score a 25% increment. The maximum score also reaches the 2500 points with an increment of 500 points. As a result, the performance of our algorithm after three training phases is better than the human experts.

5.6 Comparison Between Different Models

In this section, we compare the performance of baselines and our proposed models. Table 1 displays the difference over the average scores in 20 rounds, maximum scores and standard deviation of these models. We find that the baseline Keep-jump get the lowest average and maximum scores. Intuitively, the agent misses the most optimized jumping position in continuous jump. We notice that both MLP and Deep Q-learning methods outperform the human optimized approach with 2.39 times and 6.20 times the average scores. In addition, Deep Q-learning beats the human experts in both average scores and maximum scores. The overwhelming maximum scores in human-optimized method could be explained with the randomness of the position and size of the obstacles. In cases where obstacles are sparse, the human-optimized approaches perfectly handles the distance between adjacent obstacles and would survive even in extremely high speed mode. In summary, our MLP and Deep Q-learning methods are more intelligent than the rule-based decision making. Specifically, Deep Q-learning succeeds to upbeat the human intelligence in T-Rex game.

6. FUTURE WORKS

In our project, neither MLP nor Deep Q-learning handles velocity well enough. In Deep Q-learning, we notice velocity makes great impact over the jumping position selection. During the process of training agent with different

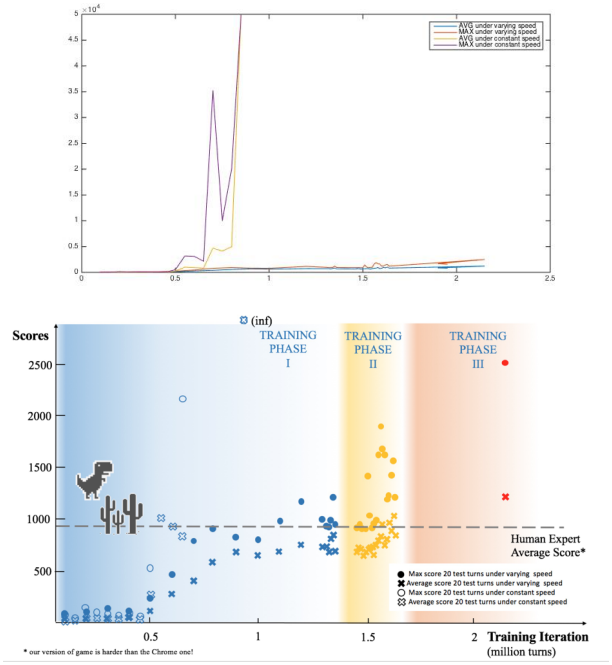

Figure 6: Training curve of Deep Q-Learning with different acceleration in different phases

acceleration and velocity, we observe that the agent tends to use policies incorrectly fitting the current velocity when the relative speed changes. Two possible explanations are as follows. Firstly, we simplify the computation in neural networks by resizing the raw game image into 80×80 pixels. In this way, the edge of the obstacles would be obscure which negatively influenced the prediction. Another problem is that we use four images in stacking to infer the relative velocity based on their differences. If more channels are added in the game image, more deviation would be detected in both MLP and Deep Q-learning in order to capture the change of velocity.

7. CONCLUSION

Several approaches were used to achieve the AI that can play Chrome Offline Dinosaur Game. For the feature-extraction based algorithm, computer version methods can recognize the T-Rex and obstacles from the images. Carefully designed feature extraction algorithms can successfully abstract the state and AI built upon them can improve its performance significantly compared with naive baseline. MLP learned from online training can strengthen the AI further for it refines the parameters automatically by experience. However, feature-extraction based algorithm have their limits and can not outperform the human experts. For end-to-end Deep-Q learning method, our result shows that it can successfully play the game by learning straightly from the pixels without feature extraction, and is much stronger than the feature-based method. Finally, specially designed training method can help us overcome the training difficulties caused by the properties of our game, which further improves our AI's performance and helps achieve super-human results.

8. REFERENCES

[1] G. Bradski et al. The opencv library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.