ECE4179 Neural Networks & Deep Learning

Final Project

Project Title:

Real Gaming Boy

(Deep Q Learning on Chrome Dino Game)

Student Name & ID:

Tong Siew Wen (29095093)

Introduction

This project is aimed to train a model to play the Chrome Dino game (also known as T rex game) via deep reinforcement learning. Ever since the first paper that successfully trained a model to learn playing Atari games and can even surpass human experts [1], many work has been done to replicate this outcome on other games.

This is not the first project attempting to apply deep reinforcement learning in the Chrome Dino game, and previous work has shown promising results. Authors in [2, 3] compared the performance of Deep Expected SARSA, Q-Network (DQN), Double DQN and Duelling DQN for this game, and showed that Duel DQN and Double DQN gives better performance than DQN, while Expected SARSA performs worst. Different from [2, 3] which uses CNN as their model and feeds in stacked images, [4] uses MLP via numerical data such as distance and height of nearest obstacles, and it is shown that MLP models gives worse performance than CNN model in [2, 3]. [5] also did a comparison between performance of MLP DQN model and CNN DQN model, and has proved that CNN models give a better performance for the Chrome Dino game, as some information is lost when using numerical data as inputs.

Methodology

Although previous work shows that Duel DQN and Double DQN performs better than DQN, due to the time limit, it is decided to use DQN instead as DQN is less complicated and easier to code and train. The Deep Q Learning framework can be summarised in the block diagram below:

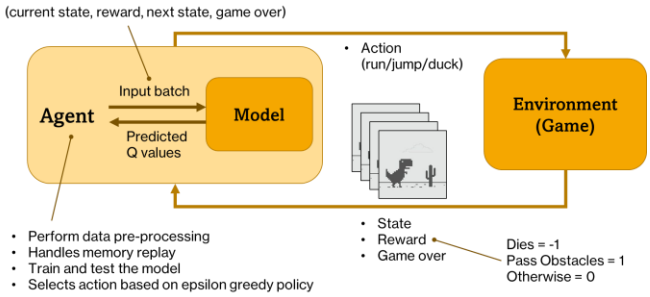


Figure 1: Deep Q Learning framework

As shown in the diagram above, the Environment, Agent and Model interact with each other in Deep Q Learning. The environment is coded using pygame module, with reference made to [6]. The game (environment) code is edited to have a closer representation of the actual Chrome Dino game, and returns the state of the game, reward of the action taken, as well as whether the game is over. The reward system is defined to be -1 if the dinosaur hits an obstacles and dies, 1 if the dinosaur jumps or duck pass an obstacle, and 0 otherwise. The state is a stack of four images which is resized to 90x90 and converted to grayscale. Stacking four images aims to give some temporal information to the model.

The CNN model structure is as shown below:

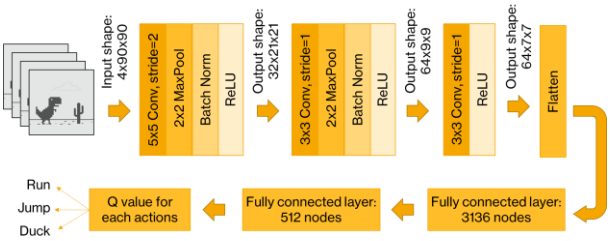


Figure 2: Deep Q Model architecture (CNN network)

Replay memory is also implemented i.e., storing and replaying observation tuples (current state, reward, next state, game over). As the consecutive states are highly correlated, if the model is trained consecutively, the model will skew to the dataset distribution of different states. Hence, saving the observation tuples, and randomly selecting a minibatch per training step will avoid this problem. Similar to other deep learning networks, a greater batch size will give better performance as the data distribution will be less bias but having a big batch size will slow down the training and occupies great amount of GPU memory. Taking this consideration into account, the batch size is chosen to be 64.

The model is trained using greedy epsilon policy where the model picks the action with the highest Q value for maximum reward. The Q value of the action taken is computed using Bellman's equation:

$$Q(s,a) = r + \gamma max_{a'} Q(s',a')$$

Equation 1: Bellman's equation

where r is the reward of the action taken, s is the current state, s' is the next state, a is the action taken, a' is the next action taken and ɣ is the discount factor, a parameter with value between 0 to 1 that determines how much a model take consideration into future rewards when choosing an action. Since this is a regularisation problem, mean square error is calculated between the predicted Q value and the actual Q value of the action taken. ɣ is selected to be 0.99 so that the model is far-sighted and will take actions with consideration of future rewards.

Implementing the greedy epsilon exploration strategy also allows the model to explore different

actions during training. However, as discussed in [3, 5], the value of the initial epsilon should not be high as the agent can't take any action while the dinosaur is off the ground, hence taking random actions will easily cause the dinosaur to die. However, the exploration should not be eliminated completely as having some amount of exploration of the model is proved to help to improve the performance in Deep Q Learning. Hence, the initial epsilon value is chosen to be 0.1 and will decrement linearly to 0.001 over 2 million steps.

Adam optimiser is used with learning rate of 1e-4, and the model is trained for 5000 games. The performance of the model is measured by the score of each game, while the model loss is the mean loss per game, i.e. the total loss of the minibatch randomly chosen from the replay memory, divided by the number of frames for the game.

Results and Discussion

Initially, the greedy epsilon exploration strategy is designed to start from 0.9 and decrease linearly to 1e-3 over 1000 steps. The result of the model is not satisfactory, as it is observed that the model is not learning, shown in the plots below:

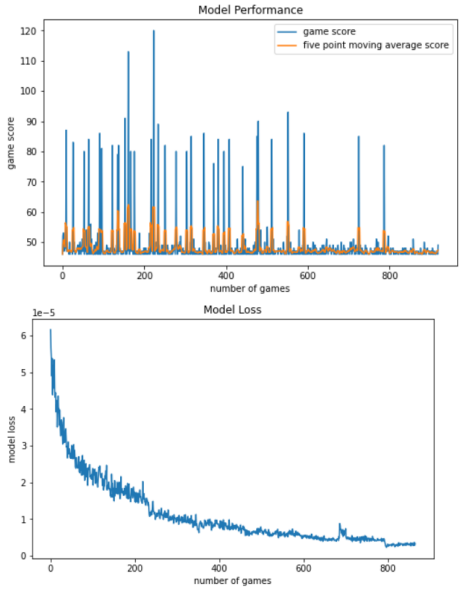


Figure 3: Model Performance (top) and Loss (bottom) when having a big initial epsilon value and a small discount factor

As shown in the plot, the model loss has fully converged, but the model performance shows otherwise as the score of each game deteriorates as training proceeds, and when visualising the performance, the model is observed to be biased to one action. It is then realised that Chrome Dino game is not suitable for common exploration strategy used in other games. Instead, the initial epsilon should start with a smaller value [3], as discussed earlier. Another reason will be having a discount factor that is too low. Chrome Dino game will require higher value of discount factor, as the model will need to take consideration of not just the nearest obstacles, as different landing position might cause insufficient reaction time for the Dinosaur to pass the following obstacles. After these are corrected, the model starts to learn.

As suggested in [7], He normal initialisation [8] will be beneficial as the model uses ReLU activation layers. However, as shown in Figure 4 below, it seems that He initialisation might not be suitable for this problem as the initial loss is huge. Instead, He uniform initialisation, which is also the default initialisation for convolutional and linear layers [9], is a better initialiser for this problem since it gives a low initial loss, as shown in Figure 5. It is also observed that the right initialisation is important, as when model performance in Figure 4 and 5 are compared, model performance in Figure 5 is better and has a higher average score.

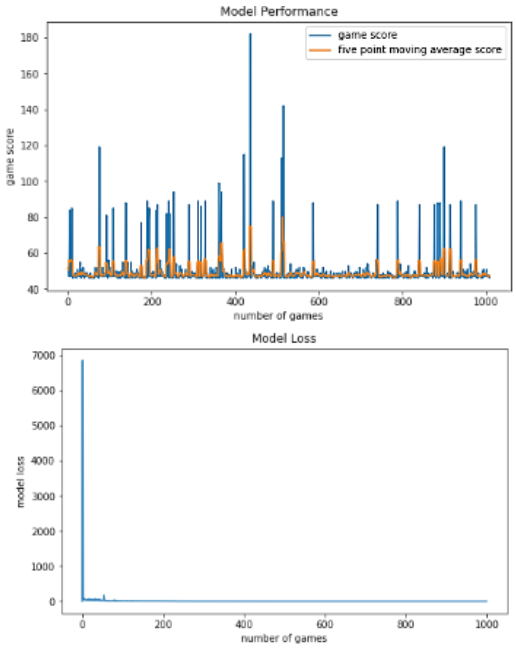
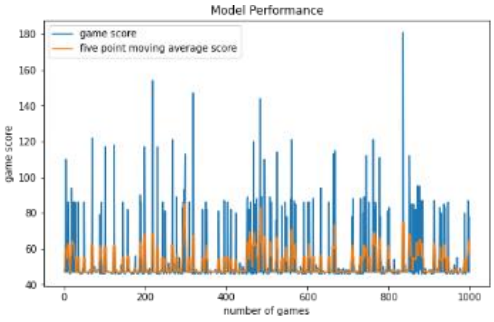


Figure 4: Model Performance (top) and Model Loss (bottom) when using He normal distribution initialisation.



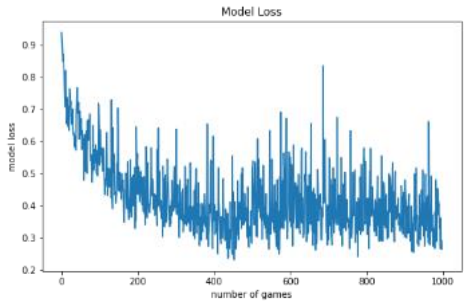


Figure 5: Model Performance (top) and Model Loss (bottom) when using He uniform distribution initialisation.

Another point worth mentioning is that with Batch Normalisation, even though the wrong initialisation is used, the loss converges in one to two games, and the training remains stable. This is also mentioned in [10] that with Batch Normalisation we can be less careful with the initialisation. Since the default initialisation is more suitable, it is used thereafter. The model is trained for another 4000 games, and the model performance is shown in Figure 6 and 7.

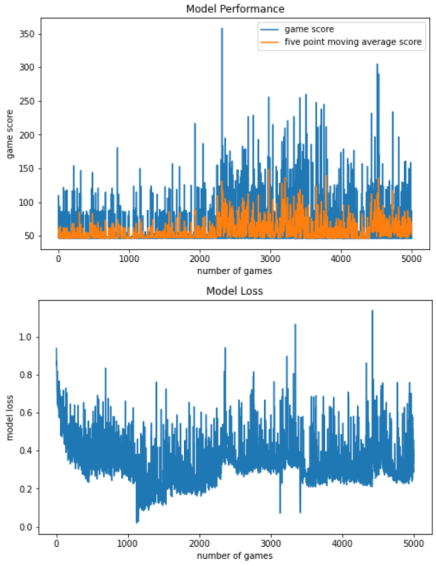


Figure 6: Model Performance (top) and Model Loss (bottom) after training the model for 5000 games

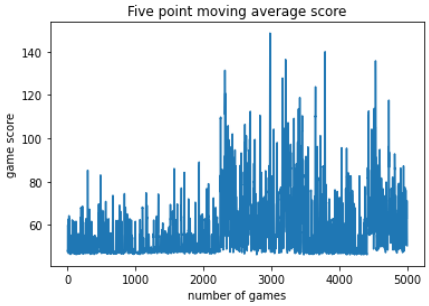


Figure 7: Five point moving average score after 5000 games

From the plot of the model performance, it can be observed that there is an obvious improvement after 2000 games, where the average score increases from approximately 60 per game to 100 per game. The model also achieved a maximum score of 358 at the 2310th game. The plot of the model loss also shows that the loss increases after the 2000th game. This could be due to the game speed increment after every 100 points; hence the model needs to learn to adapt to different speeds as the Dinosaur is survives longer in the game. After training completes, the model is tested, and results shows that the model can pass at least one obstacle on average (around 80 points) and achieved a maximum score of 221 (equivalent to passing five obstacles). However, it is observed visually that the Dinosaur is ducking randomly even when there are no obstacles. This can be solved by having a small reward when the Dinosaur chooses to run when there is no obstacles. However, since this does not cause the Dinosaur to die, this solution is be implemented as it would not improve the performance of the model other than helping the model to better mimic a human player.

However, if compared to the previous work done, this performance is not as good as theirs. After comparing their methods, a few inferences can be made. First, max pooling layers might be the cause of low performance. [3] mentioned that max pooling and average pooling might cause the model to ignore small movement of the state, causing it to be less sensitive to obstacle distance. Second, number of actions considered will increase the level of difficulty for the model to learn. Authors in [4] has attempted to train a model with all three actions but failed to achieve good results. After they changed their model to only consider two actions, the performance of the model is much higher. Lastly, more pre-processing on the input might have improve the model. In the Chrome Dino game, there are lots of other objects, such as the clouds and small stones, which might have confused the model to take them as obstacles. For instance, authors in [2, 5] did a further step to perform background filtering on the game screenshots. Cropping out blank spaces in the screenshot could also improve the performance. Thus, it is believed these future improvements will help improve the performance.

Conclusion

This project uses DQN to approach the problem of training a model to play Chrome Dino game. A CNN model is designed and will output the predicted value of three actions: run, jump, and duck. The model is trained for 5000 games, the highest score achieved is 358 during training, and 221 during testing.

References

[1] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] D. Marwah, S. Srivastava, A. Gupta, and S. Verma, *Chrome Dino Run using Reinforcement Learning*. 2020.

[3] Y. Zheng, "Reinforcement Learning and Video Games," *arXiv preprint arXiv:1909.04751*, 2019.

[4] J. L.-P. Vikas Munukutla, Elijah Freeman, Emanuel Pinilla, "My Dinosaur Project," 2019.

[5] "AI for Chrome Offline Dinosaur Game." [Online]. Available: http://cs229.stanford.edu/proj2016/report/KeZhaoWei-AIForChromeOfflineDinosaurGame-report.pdf.

[6] codewmax. "ChromeDinosaur." https://github.com/codewmax/ChromeDinosaur (accessed April, 2021).

[7] F. M. Graetz. "How to match DeepMind's Deep Q-Learning score in Breakout." https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756 (accessed 20 May, 2021).

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026-1034.

[9] "Pytorch nn Modules (source code)." https://github.com/pytorch/pytorch/tree/master/torch/nn/modules (accessed 20 May, 2021).

[10] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, 2015: PMLR, pp. 448-456.