FLAP-AI BIRD

Ramya BN1, Chandana C Gowda2, Poojitha KB3, Shubhashree Bhat4, Yashwanth L5

¹ Assistant Professor, Artificial Intelligence and Machine Learning, Jyothy Institute of Technology, Bengaluru, Karnataka, India

^{2,3,4,5} Student, Artificial Intelligence and Machine Learning, Jyothy Institute of Technology, Bengaluru, Karnataka, India

ABSTRACT

This study presents the application of Deep Q-Learning, a variant of Deep Reinforcement Learning (DRL), for training an AI agent to play the game Flappy Bird autonomously. Flappy Bird, due to its dynamic and continuous environment, poses a challenge that tests real-time decision-making capabilities. The proposed approach utilizes a Deep Q-Network (DQN) which leverages convolutional neural networks (CNNs) to estimate Q-values directly from pixel input. Core components such as experience replay, target networks, and epsilon-greedy policies are integrated to ensure stable and efficient learning. Extensive experimentation highlights how varying hyperparameters such as learning rate, replay buffer size, and exploration rates impact convergence and gameplay efficiency. The final agent demonstrates robust decision-making, achieving high scores consistently, and provides insights into DRL's applicability in constrained, real-time environments.

Keyword : Training an Agent to Play Flappy Bird Using Reinforcement Learning Techniques

1. Introduction

In recent years, Reinforcement Learning (RL) has evolved rapidly, especially with the introduction of deep learning techniques. The integration of neural networks with RL has resulted in Deep Reinforcement Learning (DRL), enabling agents to learn directly from high-dimensional sensory inputs. Flappy Bird, a 2D side-scrolling mobile game, requires the player to navigate a bird through an endless series of pipes by tapping to flap. The game is deceptively simple but demands precise control and split-second decision-making, making it a suitable benchmark for reinforcement learning research. Traditional rule-based systems fail to generalize well to new game scenarios. Therefore, applying DRL enables the agent to learn optimal control strategies through trial-and-error interaction. The goal is to maximize cumulative future rewards by selecting actions that avoid collisions and progress further in the game. In this paper, we implement and evaluate a Deep Q-Network (DQN) tailored for Flappy Bird. The paper aims to highlight key components of DQN training and offer empirical results that support its effectiveness.

The Neuro evolution technique is presented as a powerful strategy to evolve Artificial Neural Networks (ANN) in unsupervised learning problems (problems where there is no input and output table). It offers an alternative way to find the best configuration for an ANN without depending on a correct output value, which is commonly used to generate an error to optimize the networks settings through Descending Gradient algorithms like Backpropagation. On the other hand, the game Flappy Bird 1 shows itself as a promising virtual testing environment to optimize agents whose goal is to learn the behavior of nondeterministic phenomena. It is a popular game that was initially developed for mobile devices. Its goal is simply to keep the bird, the player, alive as long as possible by passing through a gap between pair of pipes without colliding with them When the screen is touched, the bird will perform a jump, on all other moments, it will fall gradually as a result of gravity.

1.1 Related Work

The seminal work by Mnih et al. (2015) introduced Deep Q-Learning for playing Atari games, where DQNs learned directly from raw pixels and achieved performance comparable to or surpassing human players. Since then, numerous extensions have been proposed, including Double DQN (Van Hasselt et al., 2016), Dueling DQN (Wang et al., 2016), and Prioritized Experience Replay (Schaul et al., 2016). Flappy Bird has also received attention in AI research as a toy problem. Researchers have applied Q-learning with discrete state encoding, supervised learning using imitation data, and evolutionary strategies. However, many of these approaches lacked scalability or required

hand-crafted features. By leveraging deep learning, our approach avoids feature engineering and generalizes better across game states. Other DRL applications include AlphaGo (Silver et al., 2016), which demonstrated the potential of DRL in highly complex board games, and autonomous vehicle control systems using PPO and DDPG, which address continuous control environments. The success of DRL across diverse domains validates its potential for mastering simple games like Flappy Bird and serves as a step towards broader general intelligence in agents.

1.2 Application of Neuroevolution

The application of Neuroevolution in games has been used for some time with satisfactory results in the creation of intelligent agents capable of human and even super-human level of playing [10]. Within the scope of Neuroevolution, NEAT is one of the most popular and promising techniques in the context of games. The uses of NEAT ranges from finding a Go player agent independent on the board size [15], to use in real time environments such as Neuro-Evolving Robotic Operative (NERO). In the latter case, it is called real-time Neuroevolution of Augmenting Topologies (rt-NEAT) [13]. In a NERO context, NEAT is employed in the training of groups of virtual robots capable of playing against other teams. A very distant NEAT variation is content-generating Neuro-Evolution of Augmenting Topologies (cg-NEAT) [4], in which NEAT is used to generate the contents of a game called Galatic Arms Race (GAR). It allows the game to change during execution, increasing players' immersion and retaining their attention. We also call attention to NEAT's use in the development of playing agents for Fighting Games, where building a consistent form of measuring fitness is quite relevant [6].

2. Methodology

Our agent is based on the Deep Q-Network (DQN) framework. The environment provides frames of size 84x84 pixels. We preprocess these frames by converting them to grayscale, resizing, and stacking four consecutive frames to capture motion. The agent selects one of two actions: flap or do nothing. Three components are essential to this work: fitness func tion calculation, presented in Section III-A; how to expose the scenario to the agent, presented in Section III-B; and phenotype settings, presented in Section III-C. Fitness To compute the fitness of the agent three components are used, called Scenario Fitness Components (SFC)



Figure 1: A graphic illustration of the first state representation



Traveled Distance (TD): a counter that increases in each interaction of the agent with the environment; Score: the number of pairs of pipes already transposed; Y Factor (Δ Y): the value obtained when the agent fails on any part of the scenario, which is calculated by the difference between the y coordinate of the agent and the y coordinate of the midpoint of the passage between the following pipes, defined as: Δ Y =y agent -y passage (1) Y Factor plays a crucial role since it enables to penalize the performance of the agent in the fitness function with a value that expresses how far the agent was from the objective, the passage between pipes, before failing. three Y Factors are highlighted, Δ Y1, Δ Y2 and Δ Y3, each of them corresponding to a different agent of the scenario. When this occurs the interaction with the environment ceases and the agent's performance is measured. The Y Factor is used to transmit a quality value to the fitness calculation, since it differentiates agents that failed in the same pair of pipes but in different heights. The goal is to ensure that agents closer to the passage are considered better than others further away, something that would not be possible if the performance of the agent was based only in TD and score.



Fig -1: The architecture diagram of flappy bird project







Fig 3: The three types of scenarios used in learning.

3. Experiment and Results

When the agent receives the information given by the environment (Ay, By and Cy), it is processed by the net work, generating a probability of executing a jump. This probability will determine the action to execute according to the rule: Action = Jump if (Out \geq b) else None, where b was 0.4. In the configuration of the NEAT parameters, the fol lowing values were established, which can then be used to reproduce this work:

• Population: a population of 30 individuals is used. It is not a big population, which enables a faster execution, and it is large enough to allow a genomic diversity.

• Elitism: an elitism of 18 individuals was chosen, a value that compared to the previous parameter is equivalent. Inputs. to 60% of the population. This value allows the preservation of innovations and it is small enough to not incur in stagnation or the absence of innovations.

• Compatibility threshold: the value given to this term is 3.0, which is large enough to not create many species initially and is small enough to not completely prevent the formation of species within the population. It is interesting to note that a large number of species in a population can generate intersections between species which can lead to problems in optimization especially when the population is small.

• Mutation rate: this parameter has a value of 0.05, which causes the connections to not activate immediately as the topology increases, they are activated gradually.

• Weight and Bias: the average initial generation of weights and bias were 0 and 0.01, respectively, with a standard deviation of 1.3 in both. These values allow a slightly more varied distribution when the weights are generated, thus getting closer to the topologies of better performance.

• Probabilities to add or remove connections: the like lihood of adding connections and the likelihood of removing connections were set to 0.7 and 0.2, re spectively. These values symbolize a greater affection for a robust topology through connection creation, a proposal aimed at solving complex problems. If the topology does not increase, but the optimization finds good individuals it is because a simpler topology was sufficient for the problem, since NEAT starts from less complex to more complex topologies.

• Probabilities to add or remove nodes: the probability of adding nodes was set to 0.7 and the probability of removing nodes was set to 0.2, as in the above parameters, and they have a similar explanation.



3.1 Fitness and scores

The results of fitness and scores are presented in Section IV-A. The speciation chart is shown in Section IV-B. Finally, the final network topology is presented in Section IV-C. A. Fitness and Scores presents the fitness results. The x-axis of the chart corresponds to generations, from the beginning going up to 100, while the y-axis corresponds to the average fitness (blue line) and the best fitness (red line) on every generation. It can be seen that in about generation 20 the fitness stabilizes until the end of the tests. The algorithm is able to achieve an optimal score since the first generations, showing the robustness of the applied strategy. The score chart is very similar. The blue line is the average score and the red line is the best score achieved on every generation. The stabilization of the scores occurs again around the 20th generation, agreeing with the fitness. In both figures the mean values stabilize in values that represent around 2/3 of the maximum values in each chart. The maximum score in the fitness chart is equal to 2.0, the value when all SFFs are equal to 2.0, which occurs when TD =195, scores = 3 and $\Delta Y = 0$. This means that the agent was able to pass through all three pairs of pipes and did not shock into anything. This implies that the SFF = $\alpha+\beta$ =1.0+1.0 = 2.0, leading to a AFF = [(1.0×2.0+2.0+6.0×2.0)/(1.0+2.0+6.0)] = (18.0/9.0) = 2.0, since k1 = 1.0, k2 = 2.0 and k3 = 6.0.



Fig -2: Fitness and score

4. CONCLUSIONS

In this study, we successfully applied a Deep Q-Network to train an agent to play Flappy Bird autonomously. Through a carefully designed neural architecture and proper training strategies, the agent learned to survive and achieve high scores in a dynamic environment. The experiment reinforced the importance of experience replay, target network stabilization, and exploration strategies in DRL. Our findings are consistent with prior research and further demonstrate the generalizability of DQN across environments. Future work includes applying Double DQN and Dueling DQN to further reduce overestimation bias and enhance policy learning. Adding Long Short-Term Memory (LSTM) components could enable better temporal reasoning. Moreover, transferring the trained model to similar but unseen game environments may test generalization capabilities.

Because the obstacles used have gaps near the environment borders and an intermediate one, when the agent manages to maximize its result in these three cases it then masters all possible variations within these limits. Considering that NEAT always searches for the simplest solution to a problem and that the fitness presented together with the division into three scenarios help the NEAT find a perceptron network architecture using a single species, this solution is the simplest. The techniques discussed in this work helped the algorithm to find this solution in a short time, thus proving its effectiveness. Using a population of only 30 individuals, the evolution ary algorithm was able to converge to an optimal behavior after about twenty generations. At this point, an agent is able to play the game indefinitely. This shows that this strategy can find optimal solutions in a short number of generations. As future works, this minimal training strategy can be tested in simple platform games that show some kind of action repetition through their stages, and also with different types of learning algorithms.

6. REFERENCES

[1]. Mnih, V., et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

[2]. Sutton, R.S. and Barto, A.G. Reinforcement Learning: An Introduction. MIT press (2018).

[3]. Bellemare, M.G., et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents." *JAIR*, 47 (2013).

[4]. Silver, D., et al. "Mastering the game of Go with deep neural networks and tree search." *Nature* 529.7587 (2016): 484-489.

[5]. Wang, Z., et al. "Dueling network architectures for deep reinforcement learning." ICML (2016).

[6]. M. Henaff, J. J. Zhao, and Y. LeCun, "Prediction under uncertainty with error-encoding networks," CoRR, vol. abs/1711.04994, 2017. [Online]. Available: http://arxiv.org/ abs/1711.04994

[7]. S. Samothrakis, D. Perez-Liebana, S. M. Lucas, and M. Fasli, "Neuroevolution for General Video Game Playing," 2015 IEEE Conference on Computational Intelligence and Games, CIG 2015- Proceedings, pp. 200–207, 2015.

[8]. K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," IEEE Transactions on Evolutionary Computation, vol. 9, no. 6, pp. 653–668, 2005.

[9]. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis

[10]. "Human-level control through deep reinforcement learning," in Nature 518, p. 529-533, 2015.

[11]. R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, Mas sachusetts: The MIT Press, 2nd ed., 1992.

[12]. 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.

[13]. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," arXiv preprint arXiv:1606.01540, 2016.

[14]. N. Tasfi, "Ple: A reinforcement learning environment," 2016.
[15]. G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," 1994.

[16]. C. J. C. H. Watkins and P. Dayan, "Q-learning," in Machine Learning volume 8, p. 279–292, 1992.

[17]F. S. Melo and M. I. Ribeiro, "Q-learning with linear function approximation," in International Conference on Computational Learning Theory, pp. 308–322, 2007.

[18]. B. Lau, "Using keras and deep q-network to play flappy bird," 2016.