Title: Investigating Machine Learning Strategies for creating agents that play simple games.

Authors: Antonio Caceres, Matthew Kazan, Benjamin Wyant

Abstract:

We will design agents to play Flappy Bird using genetic algorithms and reinforcement learning. We will be designing and comparing the two approaches to see what trade-offs each has and which is best for this particular application. The NEAT algorithm is a genetic algorithm that evolves neural networks through a process of selection and mutation, while reinforcement learning is a technique in which agents learn from trial and error through interactions with their environment. We experimented with both algorithms on the game Flappy Bird and evaluated their success based on the complexity of the implementation, the complexity of the resulting agent, and the time it took to train.

Introduction/Background:

This has been a project we have wanted to do for a long time. Genetic algorithms are a distinctly complex topic that we have had very little experience with and gaining skills in implementing NEAT could help with more complex projects in the future. Additionally, reinforcement learning is an incredibly important technique that will undoubtedly provide useful knowledge and skills when implemented. We initially wanted to do a more complex project but getting several machine learning techniques built from scratch when none of us had ever attempted a similar project was going to be challenging enough.

Similar projects are fairly common which allowed us the necessary resources to complete the project in a tight timeframe. The white paper left quite a few implementation details vague, especially in regards to the innovation history so we used a GitHub repo to figure out the remaining parts (Code-Bullet). Additionally, our comparison is relatively unique as we haven't seen a direct comparison between the two algorithms for a game like Flappy Bird.

Our Goal for this project was to learn how these algorithms worked, how to implement them, and gain the skills necessary to work on more complex algorithms in the future. While also providing a useful comparison for us to use when deciding on which algorithm to use for a given task.

Data Analysis:

While this project did not have a traditional data set, we trained on a custom implementation of the game based on FlapPyBird (Sourabh Verma). The game was randomly generated but was seeded to allow us to rerun the same game repeatedly for training purposes. Aside from the game itself, there is no training data. One of the reasons these algorithms are so valued is that they do not require training data, and learn by doing.

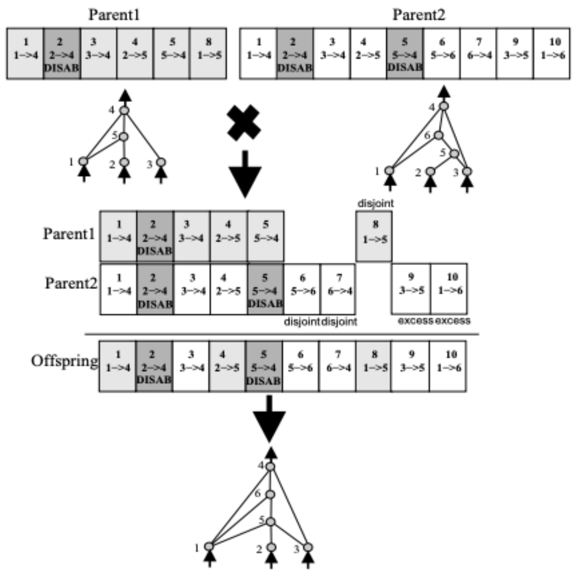
Methods:

Feature Selection:

For our AI agent to play Flappy Bird (successfully), we needed to provide our model inputs, also known as features. Our model did not rely on a large dataset with features and labels. Instead, the feature that we used to train our model was our bird's "vision." The "vision" was a set of 4 features. The first and second features are the player's y-velocity and the horizontal distance to the closest pipe. The last two features were the vertical distances to the bottom of the top pipe and the top of the bottom pipe. We chose these inputs because they made intuitive sense. That is, when we played the game ourselves, these were the four features that we looked at to determine when to have the bird "flap."

ML Models Employed and Rational:

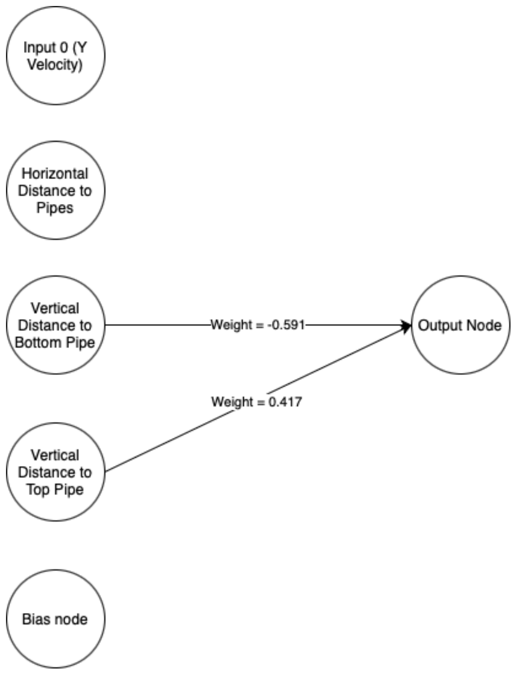
The first algorithm we used was a genetic algorithm called NeuroEvolution of Augmenting Topologies, also called NEAT. (Stanley, Kenneth O., and Risto Miikkulainen) NEAT is a genetic algorithm used to augment a neural network, referred to as the genome, through mutations and breeding. The concept is simple, starting with the most basic genome possible, mutate or crossover the genes of each genome based on how well it performed within the population. Each genome is composed of genes, made up of nodes and connections. Each agent that plays the game receives a fitness score based on how long it survived and its score. A critical component of the NEAT algorithm is tracking genes through historical markings. These markings are tracked by using innovation numbers. Innovation numbers are used to keep track of when connections between nodes in a genome occur. As stated in the NEAT paper "Whenever a new gene appears (through structural mutation), a global innovation number is incremented and assigned to that gene." (Stanley, Kenneth O., and Risto Miikkulainen) This tracking of genes through an innovation history allows easy crossover between different genomes because only the innovation numbers need to be compared. These innovation numbers are also key for speciating. Our population was divided into its own niches of genomes, each of which we called a species. Creating separate species allows for the protection of new topological innovations. This is important because a new species needs time to optimize its network structure through competition within its own niche. Successfully implementing this allowed us to create a genome that was able to play Flappy Bird.



Hyperparameter Tuning Methods:

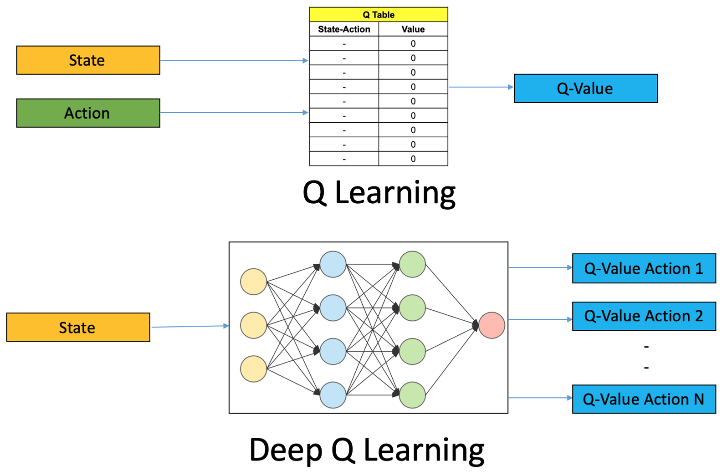
For our genetic algorithm, there are many hyperparameters that can be tuned. Tuning hyperparameters helps with faster training of the birds, resulting in fewer generations needed to optimize the network topology. The hyperparameter which has the largest impact on how many generations are needed for a genome to be optimized is the size of the population. When using a population of 2000 players, by the third generation our player's genome was already working so well that it got a score of over 150 (Matt's all time personal best is 112). Another group of hyperparameters to tune were those that dealt with mutation. We found that our NEAT implementation works best when we have an 80% chance of mutating a weight, a 90% chance of perturbing the weight if the connection weight is being mutated, a 10% chance of a new connection being added, and a 3% chance of a node being added. Another hyperparameter we tuned was the chance of a child's genome being created from cloning vs. being created from breeding two parents. We determined that if the population size is large, the chance of mutating and adding a new connection or new node can be decreased. We ended up using a population size between 20-100 because it allows for a better visualization of evolutions over time, so we kept the new node and connection chances higher.

Analysis:


Input 0 (Y Velocity)
Horizontal Distance to Pipes
Vertical Distance to Bottom Pipe
Weight = -0.591
Output Node
Weight = 0.417
Vertical Distance to Top Pipe
Bias node

NEAT was able to solve Flappy Bird quite easily. The game only has a few possible inputs and it turns out that only one is required for a perfect agent. Once the "brain" connects the input node which gives the difference in height between the player and the top pipe, the agent can quickly learn the weights needed to successfully play Flappy Bird indefinitely. With a population larger than 200, there is a good chance of getting a perfect agent within the first generation, and with a smaller population size learning still gets done fairly quickly. NEAT was able to create a perfect agent with a minimal neural network structure. This success in Flappy Bird demonstrates the potential of NEAT in solving a wide range of problems that require intelligent decision-making.

Deep Q Learning was our second choice of algorithm. The main idea behind Deep Q Learning is to learn the optimal policy to maximize the reward (game score) for each state, and action tuple the bird encounters. We used a deep neural network in TensorFlow Keras to represent the Q-function, which in our case is the Bellman equation. Our bird was trained on a state space of size 4, (x-velocity, y-velocity distance to top pipe, distance to ground) and an action space of 2 (flap or no flap). Our specific algorithm implemented experience replay, “which has been shown to improve sample efficiency and stability by storing a fixed number of the most recently collected transitions for training.” (Fedus) Experience replay also prevents catastrophic forgetting from occurring, which often plagues many RL models, especially those that use Q-Learning. However, long training times and TensorFlow overhead hindered our Agent to get promising results.



One of the most important hyperparameters is the size of the replay buffer, which stores the agent's experiences for training. We found that a replay buffer size of 1 million experiences works well for Flappy Bird. Another important hyperparameter is the discount factor, which controls the trade-off between immediate and future rewards. We used a discount factor of 0.99, which gave good results.

Conclusion:

Ultimately we feel that NEAT was the better algorithm for this particular problem. It solved the game more efficiently and was able to play skillfully more quickly. While knowing which algorithm was better for this particular application is helpful. The most important part of this comparison was what we learned while building both algorithms.

NEAT had a few tricky aspects to the implementation but once those were working the rest of the algorithm is fairly straightforward and made intuitive sense. It also has a plethora of resources that can be used to assist any implementation which we needed to figure out the connection history implementation. That is by far the most challenging aspect of the implementation, it required several iterations and even then we relied on existing implementations to fully explain how it worked. Deciding to implement our project in Python made this more difficult as not having types made aliasing issues more difficult to debug. One of the benefits of using NEAT for this game was that it provided understandable readouts on what information the agent is using to play and what is unnecessary. It also was fairly computationally cheap and we were able to run populations of 2000 without serious problems.

Deep Q Learning was easier to implement because of the use of Keras. Apart from using different batch sizes and different replay buffers, we found it hard to optimize our algorithm to get quicker, better results. Our challenges were rooted in the fact that we had limited knowledge of what was happening in the hidden layers of the neural net. On top of this, DQN was computationally intensive and required lots of training time to get to a playable algorithm.

Overall, we feel that both algorithms were well implemented, could easily be used to play more complicated games, and gave us the knowledge and experience we were looking for to take to future projects.

Author Contributions:

M.K. and B.W. conceived the idea and drafted the initial game to be used for the study. They also were the key developers behind the NEAT implementation. A.C. focused on the RL implementation for Flappy Bird as well as the presentation aspect. Volodymyr Mnih et al. were the first to propose combining a neural net with the Bellman equations to produce state-action pairs. All authors worked together on the final report to ensure all contributions were accurately documented.

Works Cited

Code-Bullet. Flappy-Bird-AI, GitHub, 2018, github.com/Code-Bullet/Flappy-Bird-AI.

Sourabh Verma. "FlapPyBird." GitHub, https://github.com/sourabhv/FlapPyBird.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., & Dabney, W. (2020). Revisiting Fundamentals of Experience Replay. *ArXiv*. /abs/2007.06700

Stanley, Kenneth O., and Risto Miikkulainen. “The MIT Press Journals - the University of Texas at Austin.” *The MIT Press Journals*, MIT, 2002, https://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf.