Maxwell Johnson
Final Project Report CSCI 166
December 2, 2024

Teaching an AI to Play Flappy Bird using NEAT

Domain Selection and Setup

Environment Description
For this project, I implemented a custom Flappy Bird environment using Pygame. I call it Tensor Bird. Flappy Bird presents an interesting reinforcement learning challenge with:

(The Bird)
- Bird's vertical position (y-coordinate)
- Bird's vertical velocity

(The Obstacles)
- Distance to current pipe gap
- Distance to next pipe gap
- Vertical distance to current pipe's bottom
- Vertical distance to next pipe's bottom
- Horizontal distance to current pipe
- Horizontal distance to next pipe

Action Space
- Binary choice each frame: jump or don't jump

Reward Structure:
- +0.1 for each frame survived
- +5.0 for successfully passing through a pipe gap
- -1.0 for collision with pipes, ceiling, or floor

Goal:
Navigate the bird through as many pipe gaps as possible while avoiding collisions.

Rationale for Domain Selection
Flappy Bird was chosen for several reasons:
1. Clear objective with immediate feedback
2. Continuous state space with discrete actions
3. Physics-based environment requiring temporal decision making
4. Balance between complexity and tractability
5. Visual feedback makes it easy to assess agent performance

Model Design and Implementation

Architecture Overview

Before I settled on genetic algorithms I tried traditional deep learning. It wasn't very effective. The birds struggled to make it past 4 pipes even with extensive training.

After that I found an article about someone who used NEAT for this very problem and had success, so I copied his strategy.

The way this works in broad terms is the bird that performs the best gets its brain copied 20 times to make the next generation. Small, random alterations are performed on those copies. Then they are tested against the game to see how well they do. Presumably at least one of the 20 copies will do better than its parent, and the whole process starts again. It mimics biologic evolution, hence the name genetic algorithms.

Rather than traditional Q-learning approaches, this project uses NEAT (NeuroEvolution of Augmenting Topologies) for several reasons:

1. NEAT can evolve both network topology and weights
2. Well-suited for real-time control problems
3. Can handle continuous state spaces effectively
4. No need for experience replay or target networks

The neural network starts with:
- 8 input neurons (state variables)
- 1 output neuron (jump probability)
- No hidden layers initially (evolved as needed)

Key Implementation Components

1. Bird Class:
- Handles physics simulation
- Manages rotational animation
- Implements collision detection

2. Pipe Class:
- Generates randomly positioned gaps
- Handles movement and collision boxes
- Tracks scoring

3. NEAT Implementation:
- Population size: 20 genomes
- Fitness function combines survival time and pipe clearances
- Species preservation through explicit fitness sharing
- Progressive complexity growth through structural mutation

Hyperparameter Tuning and Experimentation

NEAT Configuration Parameters

Key parameters from config-feedforward.txt:

pop_size = 20 - the number of birds

node_add_prob = 0.2 - the chance to add hidden layer nodes

node_delete_prob = 0.2 - the chance to remove hidden layer nodes. Inputs and output never get deleted

conn_add_prob = 0.5 - the chance to add a connection between 2 nodes. All nodes start fully connected.

conn_delete_prob = 0.5 - the chance to add a connection between 2 nodes. All nodes start fully connected.

weight_mutate_rate = 0.8 - the rate that the weights for a given node will change

Parameter Tuning Process

Through experimentation, several key findings emerged:

1. Population Size:

Left it at 20 and never changed it. The main limitation is computational power. More is of course always better as it gives a higher chance for a beneficial mutation to occur.

2. Mutation Rates:
- Higher connection mutation (0.5) vs node mutation (0.2)
- Encourages weight optimization before structural changes
- Helps maintain simpler networks while still allowing complexity when needed

3. Game Physics:
- Adjusted gravity and jump velocity for smoother control. The goal was to make it feel like Flappy Bird. I created a human playable version of the game just to test how it feels when I play it to get it closer to the original game.
- Tuned pipe spacing and gap size for appropriate difficulty

4. Inputs:
- Originally the birds could see the next pipe and its own position. This worked well but it wasn't enough to make the birds immortal. I added the next pipe as well along with bird velocity. This helped but it's still not enough for immortality (the bird never loses).

Results Analysis and Evaluation

Training Progress
The NEAT algorithm showed consistent improvement across generations:

1. Early Generations (1-5):
- Random jumping behavior
- Most birds failing to pass first pipe
- Maximum fitness < 100

2. Middle Generations (6-15):
- Development of basic gap targeting
- Inconsistent but improving performance
- They often fly into the ceiling or floor when encountering a large pipe
- Maximum fitness 200-500

3. Late Generations (16+):
- Reliable pipe navigation
- Emergence of smooth flight patterns
- Maximum fitness > 1000

Novel Contributions and Future Work

Novel Aspects

1. Efficient State Representation:
- Reduced input dimensionality while maintaining performance
- Included both current and next pipe information
- Normalized inputs for better generalization
- Very fast learning, often with game mastery in less than 3 minutes

2. Visual Feedback System:
- Implemented death markers for failure analysis
- Real-time performance visualization

Future Improvements

1. Training Enhancements:
- Respawn birds on death mid training run, to maximize computer resources
- Level state periodic saving, to replay particularly difficult sections to navigate to force faster evolution
- Experiment with different hyperparameters

2. Gameplay Extensions:
- Add variable pipe speeds
- Implement moving gaps

3. Architecture Exploration:
- Test different activation functions. This currently uses tanh only.
- Experiment with fixed topology evolution. I would like to see the effects of more hidden neurons.

Conclusion

The genetic algorithm approach effectively solved the problem. It learned to play flappy bird very quickly with minimal resources. It's also pretty lightweight to set up in terms of lines of code. Most of this project was getting Flappy Bird to work, not the actual AI.

This project was a very good example of choosing the right tool for the job. My initial fumblings with tensorflow proved to be a huge waste of time. The NEAT library ended up being much easier and much more effective. This is a great example of how if all you have is a hammer, everything looks like a nail. Because I had only used tensorflow KERAS in the past, that's what I immediately went to even though it wasn't a good choice for this application.