Mari/o

Tom Rochette <tom.rochette@coreteks.org>

December 21, 2025 — 77e1b28a

0.1 Context

During the month of June 2015, a AI based on NEAT (written in LUA and executed on the BizHawk emulator) was demonstrated on youtube playing Super Mario World. At around the same time, some other developers used the same algorithm to attempt to learn to play Super Mario Kart.

0.2 Learned in this study

0.3 Things to explore

- Will it reproduce an already tried genome?
- Is transitivity (a node going to a node then to another node, making the middle node useless) removed?
- Genetic evolution is biased toward remembering good neural network while forgetting neural network connections to avoid

1 Overview

What this algorithm does is attempt to construct a state machine based on the environment state (as input). In other word, provided an X by Y grid with Z possible values for each cell, tell me what output should be emitted.

In this game, the use of a neural network with weights does not make sense. Given an environment state, many valid actions can be taken, but at the same time many invalid actions should be avoided at all cost. Obviously, this leads us to say that one can generate a dictionary of all the possible environment state and its appropriate output. However, doing so will likely mean generate a huge amount of data for something that might be generalizable.

1.1 Things to improve

- Reduce the number of attempts that are “stupid”
- Prevent saving a state which is too close to the end

1.2 Things to try

- Use a quadtree approach to learning (1 -> 4 -> 16 squares and so on)
- Add generators (sin/cos/square/triangle)
- Implement save state system
- Make it learn
  - to use the least amount of keys
  - to have the simplest neural network
- Attempt to learn patterns from the neural network (generate/reuse neural network components)
- Synthesize the improvements from all the species of the current generation into the next one

- Like learning an instrument, you need to focus on the parts you have difficulties with. Using the save/load state system, it would save/load states in 5s parts and attempt to improve its best for that part.
- Give an immense penalty to neural network that ends up with a death
- Run the same ~5s segment over 5 generations, then move onto the next 5s segment
  - Divide and conquer approach: 5s, 5s -> 10s (test the first 2 segments together)
- Learn from examples: provide him with a couple of "dumb" but okay examples to learn from
- Compare every generation best neural network against each other from the start/randomly
- Fitness aware online agent (knows it's losing fitness by not moving)
- Try to map the problem of learning mario to a function optimization problem
- Attempt to detect difficult parts within a map so that the AI may learn from it
  - For instance, no network that runs into a hole should be allowed to reproduce, thus given a starting point where the AI is in front of a hole, it will have to learn very soon how to jump over it. Due to how gene reproduction works, it should make certain behaviors such as "jump when you are close to a whole" have very strong and clear responses
    - Basically, implement some form of natural selection where agents are tested against a near death scenario and keep only the survivors
      - This assumes enough complexity within agents for the pool of agents to provide a few survivors to the near death scenario

1.3 Things to add (for review purposes)

- Load a specific test (generation/species/genome)
- Record training time (compare how much time is spent using "from start" vs "from checkpoints")
  - I have some doubt that the "from start" method takes a lot more time but doing so learns more quickly (in the sense of less generations/species), but X minutes of training for both may end up giving one a clear advantage over the other

1.4 Difficult sections in level 1

- First enemy
- Bullet + enemy
- High wall
- Slope pipe + enemy

1.5 Metrics to measure

- Rightmost
- Distance traveled
- Score

- Compare the advantages of learning from the parts you have difficulties with vs always from the start
  - Always starting from the beginning makes it appear like it is always "improving" since it's basically going through stuff it knows while starting "from a checkpoint" is constantly under learning pressure (will reject any neural network that cannot successfully complete the task)
    - That is a good thing if we want to find the perfect (or good enough) algorithms, but if there's a point at which all algorithms will fail, we might discard more efficient algorithms because they end with a lower score since they've crossed less distance
- Explore more of the map rapidly
- Even if you are the most retarded, if you have a position advantage, you'll get a +1000 fitness bonus. This means that the real most fit individual will have to be extremely good to compete against your unfair advantage.

13x13 inputs
8 outputs

Am I giving fitness to unfit species simply because they are lucky enough to load a state with an initial higher fitness? Yes and no. Yes, compare to other species under the same generation they are provided with an advantage, but if they end up not providing any benefits over the long run they will be removed from the pool.

2 See also

- My fork of the original source code

3 References

- Demo video
- Related paper: Evolving Neural Networks through Augmenting Topologies
- Original source code