Final project report

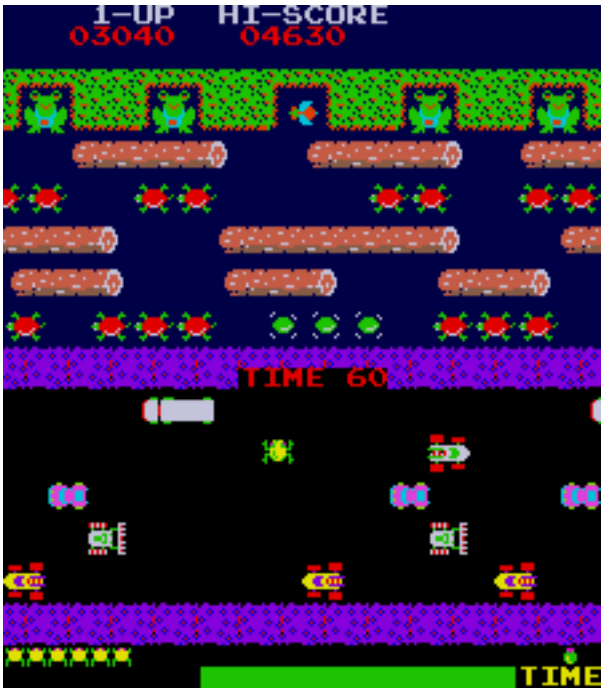
Antoni Dąbrowski, Dominik Trautman

February 20, 2022

1 Introduction

The main idea of our project was to implement an algorithm that would mimic living organisms and their adjusting to an environment. To do this we decided to explore a research field of neuro-evolution. It is a form of artificial intelligence that uses evolutionary algorithms to generate artificial neural networks - its parameters and topology. As it is most commonly applied in artificial life, general game playing and evolutionary robotics, we decided to solve game Frogger by implementing an algorithm called NEAT. **NeuroEvolution of Augmenting Topologies**[1] is a one of the greatest algorithms in its domain. Its authors - Kenneth O. Stanley and Risto Miikkulainen - won the 2017 International Society for Artificial Life (ISAL) Award for Outstanding Paper of the Decade 2002 - 2012. It is due to three key techniques: tracking genes with historical markers to allow crossover among topologies, applying speciation (the evolution of species) to preserve innovations, and developing topologies incrementally from simple initial structures ("complexifying").

As a testing environment for our algorithm, we chose an old Japanese arcade game **Frogger**[2] developed in 1981 by Konami and manufactured by Sega.



The goal of the game is to guide a frog to each of the empty "frog homes" at the top of the screen. The game starts with three, five, or seven frogs, depending on the settings. Losing them

[1] http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf
[2] https://en.wikipedia.org/wiki/Frogger

all ends the game. The only way player can control the game is the 4-direction joystick used to navigate the frog; each push causes the frog to hop once in a chosen direction.

2 Description of the taken approach

2.1 Algorithm

Algorithm 1 Neuro-evolution of augmenting topologies

```
1: procedure NEAT
2:     population ← initializePopulation()
3:     species ← speciation(population)
4:     fitnessValues ← measureFitness(species)
5:     while not terminationCondition(fitnessValues) do
6:         species ← crossover(species)
7:         species ← weightMutation(species)
8:         species ← topologyMutation(species)
9:         species ← speciation(population)
10:        fitnessValues ← measureFitness(species)
11:    end while
12:    return species.bestIndividual
13: end procedure
```

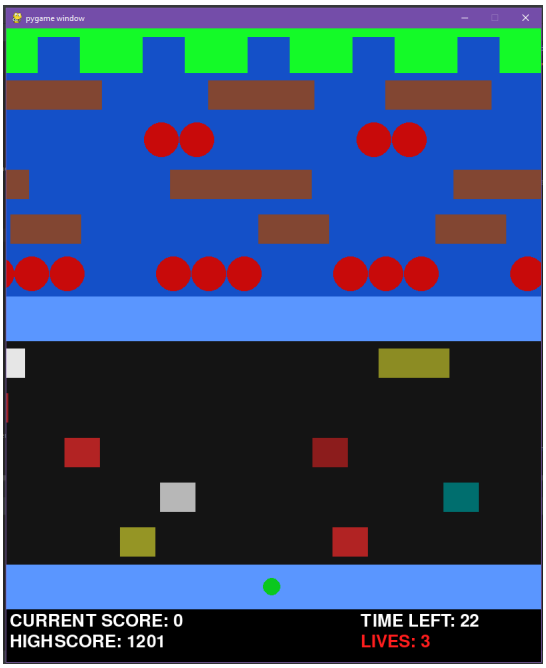
At first glance NEAT looks like a standard genetic algorithm - in a loop performs some evolutionary operators, then measures the performance of each individual and selects a new population. However, even on this level, there is a novelty - speciation. In nature, different individuals tend to be in different species that compete in different niches. Thus, innovation is implicitly protected within a niche. This partially resolves the main problem of evolutionary algorithms - premature convergence.

As it was mentioned each individual is representing some particular structure of a neural network with its weights and biases. What's nontrivial, is how to crossover those graph structures. To do that NEAT authors proposed a new, unique and most natural way of development. To start with as simple networks as possible and then grow its structure and by tracking each added connection and node, which enable fast measuring similarities between graphs.

2.2 Benchmarks

XOR - To check whether our algorithm is working correctly, we gave it a simple task to find a network that solves the xor problem, as it is the smallest problem that requires at least one neuron in the hidden layer.
Frogger - Main problem to solve.
Frogger (non-deterministic map) - Extension of previous benchmark with map generated differently each time.

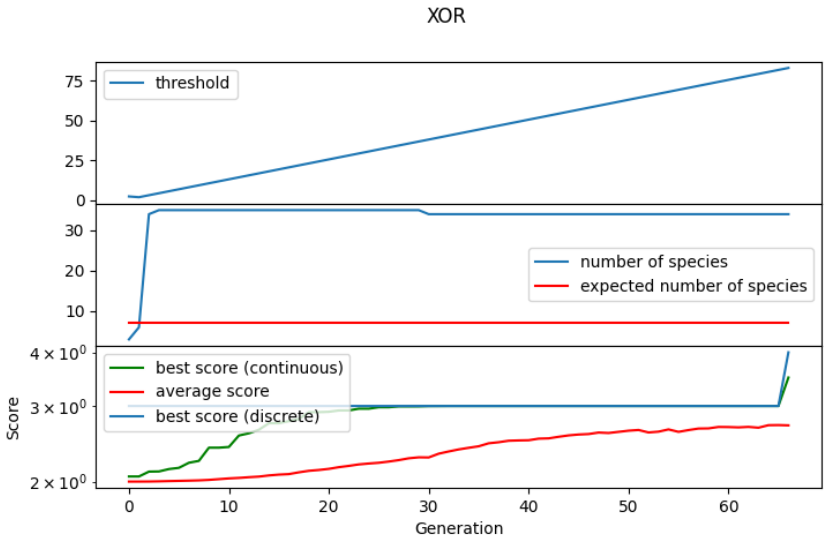
We used our implementation of the game. The inputs we pass consist of a 15×13 board of short integers. Each tile might be either safe, deadly, goal or frog. Visualization of the game can be turned off. The speed at which the game runs can be adjusted. It's possible to simulate a 30fps game with about 366-time speedup, which proves very useful for training.



3 Presentation of the results

3.1 XOR

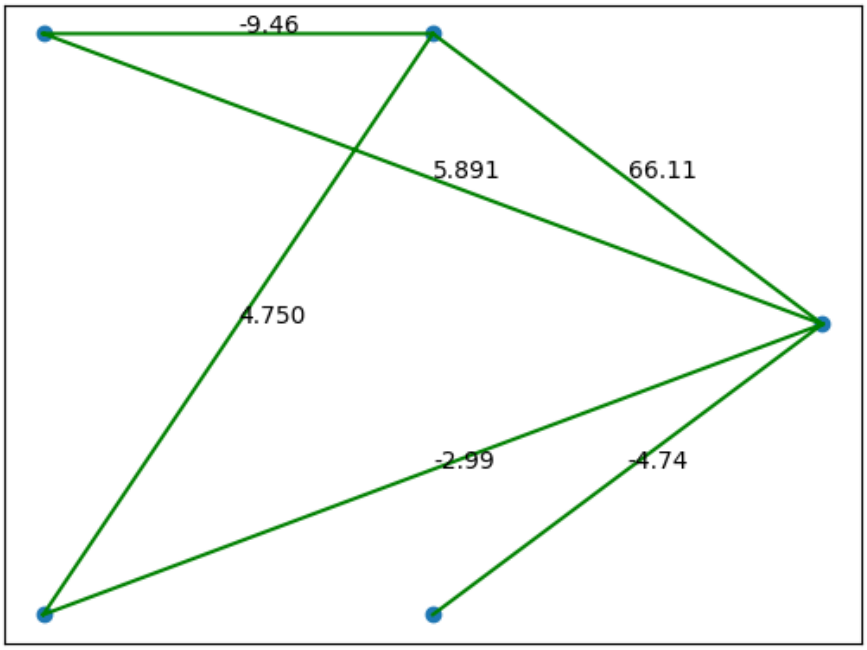
After whole lot of debugging we achieved a perfect score:



Threshold - We have a method to calculate similarities between two individuals. Based on this measure we check whether it is smaller than the threshold. If it's true then those two should be in one specie. However, it is quite an unstable measurement, so the threshold has to be adjusted dynamically to stabilize the number of species at the desired level. (red - desired level)
Number of species - Provided in the research paper method of speciation does not guarantee a constant number of species, therefore we are constantly measuring deviation from the expected value.
Performance - NEAT uses a very common among genetic algorithms method called elitism, which

preserves unchanged few best individuals in each specie. Generally, it causes constant growth of the best score of the next populations. However, to prevent premature convergence, specie that is not improving in a fixed number of generations is penalized - in our case deleted. That explains potential fluctuations.

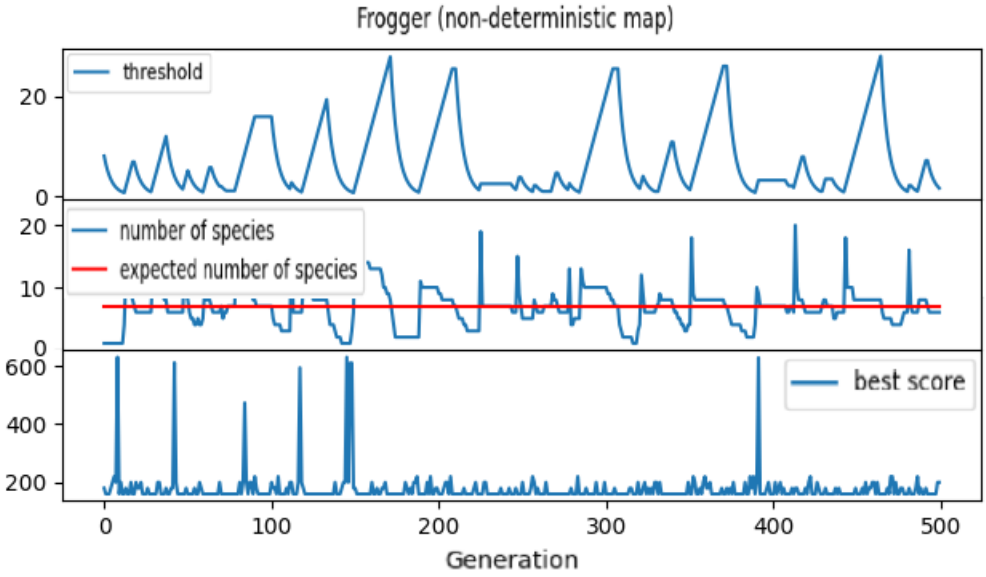
In this benchmark we gave each network four types of inputs (0,0), (1,1), (1,0), (0,1) and expected values 0, 0, 1, 1. For discrete xor, each value higher than 0.5 was classified as 1 and lower as 0. It was a fitness function considered in Kenneth's paper. However, it was not suitable for training. Therefore we introduced the continuous xor function. It was measuring distance from correct solution e.g. output 0.1, 0.1, 0.9, 0.9 have error 0.4 and score 3.6.



The plot shows a representation of a winning individual (except biases). What might be not clear at first is a node in a second layer that does not have any incoming connections. It actually does not affect network performance. However, it is an intermediate stage in evolution. In the next generation, there is a possibility of adding a connection between this node and some nodes from the first layer.

3.2 Frogger (non-deterministic map)

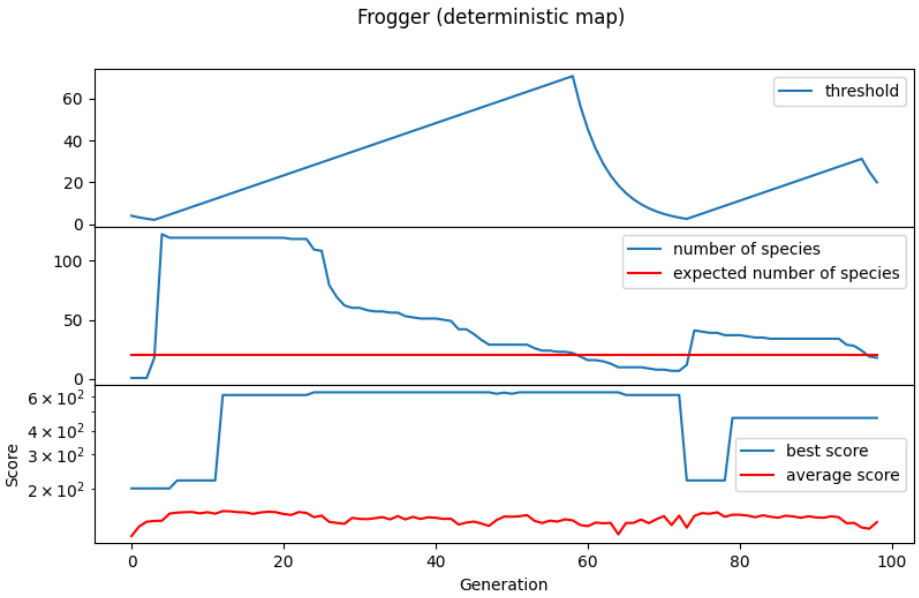
And here a fun part begins. At first, we were so proud of our algorithm that we gave it our hardest benchmark. However, it barely could lead one frog to its home (score 400+).



After spending a lot of time trying to tune hyper-parameters without any significant improvements, we chose a different strategy. Our idea was simple - it is easier to train algorithm in a deterministic world and maybe pre-trained models would learn faster how to behave in a non-deterministic environment. So from that point, we focused on the next benchmark.

3.3 Frogger (deterministic map)

At this point we started getting better and better results. First run goes as follows:



And shown us that although our algorithm has different dedicated enhancements to prevent premature convergence it still stucks in local optima.

3.3.1 Additional improvements

At this point, we saw two ways of overcoming our problem. First refers to further hyperparameter tuning. However doing it manually would not lead us to any greater knowledge and understanding

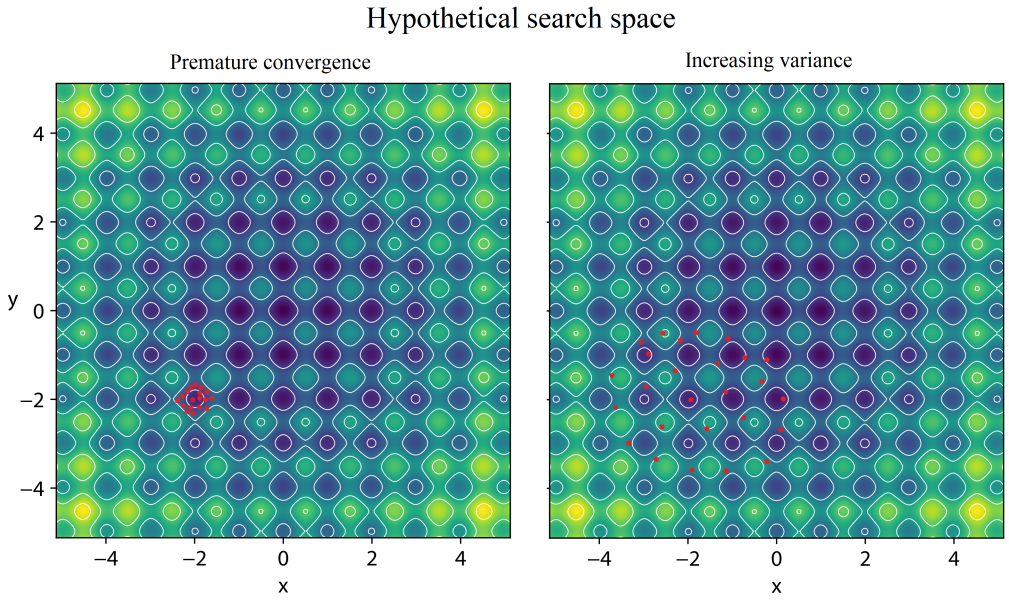
of a problem - it would be just a boring process of guessing. Therefore we could write some smart meta-algorithm to do that for us - test our algorithm with different settings of initial parameters. But still, it does not seem like a good solution as it would be too time consumable. Such an algorithm would have to train the population from scratch for any proposed configuration of meta parameters, to measure its performance. It would take about half an hour each. Considering that we have about 12 different numerical hyperparameters estimated time would be not acceptable.

The main thought of the other approach was to tune hyperparameters dynamically during the process of evolution rather than looking for a perfect initial setting. This tactic is commonly used in many more advanced genetic algorithms (e.g. CMA-ES), mostly to set the range of mutation. It is not only motivated by the problem that I mentioned earlier, but by fact that such an algorithm is able to reach a goal even faster than one with the best, constant setting of hyperparameters. Following that idea, we focused on two parameters that looked most crucial.

Mutation rate - there are actually four sub-parameters in it: the probability of local weight mutation, the probability of global weight mutation, the probability of adding a connection, and the probability of adding a node.

Extinction time - it is the number of generations after which specie will be killed if it is not improving. In place of the extinct part of the population, we were inserting 'primitive' individuals, created like in the initial stage.

We are already penalizing species that are not improving. However, it is quite radical. Maybe a more gradual way would be better. Therefore we were increasing the probability of weight and topological mutation as specie stops improving. It is motivated by the thought that although it is not the solution that we look for, it might be closer to it than the original/basic model that we were going to use for replacement.

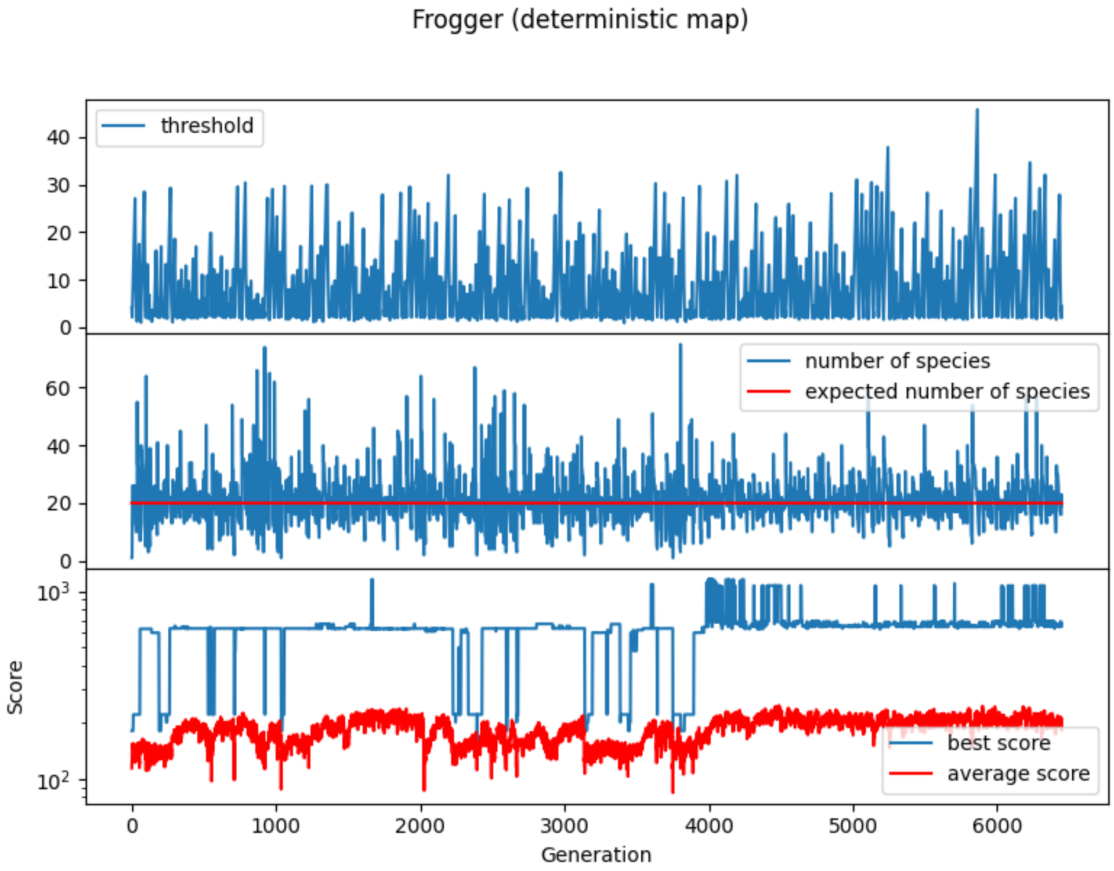


The graphic above illustrates specie stuck in a local optima (left). Process of increasing probability and range of mutation is leading to wider spread of individuals in next generations and most likely better results (right).

It is worth to mention, that theoretically our algorithm should always find solution, like an algorithm generating random neural networks will eventually find the right one. Therefore the battle is not for being able to solve a problem, but for efficiency and expected time of searching. This lead us to considerations about 'extinction time' parameter. So far our algorithm were getting better and better results to a point when the score function were getting completely flat. Then from time to time it drops as each specie achieves it extinction time. It seems possible that we gave species to much time for exploration one single local optimum. Therefor we decrease 'extinction

time' if there is no improvement in specified number of generations. However we set it back to initial level, each time algorithm achieve better result.

The results of those improvements were significant. The frog were able to pass the board three times:



However, we didn't have time for further analysis of the application of those models to the previous benchmark.

4 Discussion

The main idea of neuro-evolution algorithms is that they should be able to work in any given environment. However, it turns out that using NEAT for specific benchmarks requires lots of work with tuning hyper-parameters. We did our best to make it work, although there are still many possible improvements. One we regret that we did not use, was applying recurrent connections. In some articles, researchers were mentioning, that this enhancement provides more sophisticated behaviors, which lead to achieving better results.