Implementing Supervised Deep Learning With Feedforward Neural Network Using Genetic Algorithms

Background

Neural networks and genetic algorithms are two techniques for optimization and learning, each with its own strengths and weaknesses. Multilayer feedforward neural networks possess a number of properties which make them particularly suited to complex pattern classification problems. However, their application to some real world problems has been hampered by the lack of a training algorithm which reliably finds a nearly globally optimal set of weights in a relatively short time, especially when there is no pre-existing training data.

Genetic algorithms are a class of optimization procedures which are good at exploring a large and complex space in an intelligent way to find values close to the global optimum. Hence, they are well suited to the problem of training feedforward networks.

Purpose

- To learn the processes and the principles of neural networks (FNN, CNN, gradient descent, activation functions, backpropagation, etc.) and genetic algorithms (selection, SPBX, mutation, fitness function, etc.).
- To demonstrate this learning by implementing genetic algorithms and neural networks in simulated problems from scratch.
- To combine genetic algorithms and neural networks to solve simulated problems without pre-existing training data.
- To speculate future work and the application of these processes.

Engineering Goal

- Design a smart dots simulation in which the dots find the most efficient way to the goal using GAs (Genetic Algorithms).
- Solve a classic game of Snake by designing and implementing a FNN (Feedforward Neural Network) trained by GAs.

Smart Dots Simulation

Fitness function

1. Reward dots that reach the goal generously.
2. If the dots don't reach the goal, Reward dots for getting closer.
3. Penalize dots for taking a lot of steps.

$$f(distance\ to\ goal, steps) = \frac{1}{distance\ to\ goal^2} + \frac{10000}{steps^2}$$

Crossover

SPBX (Single Point Binary Crossover)

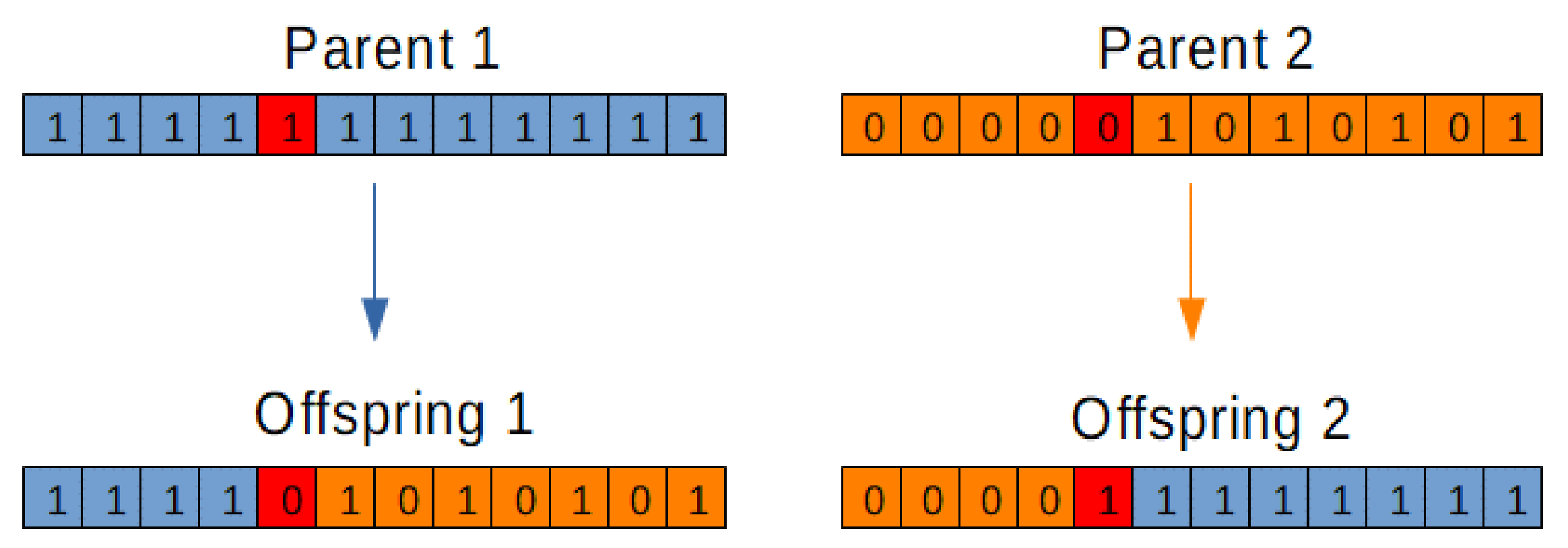

Figure 1. Single Point Binary Crossover

Selection

Roulette Wheel Selection

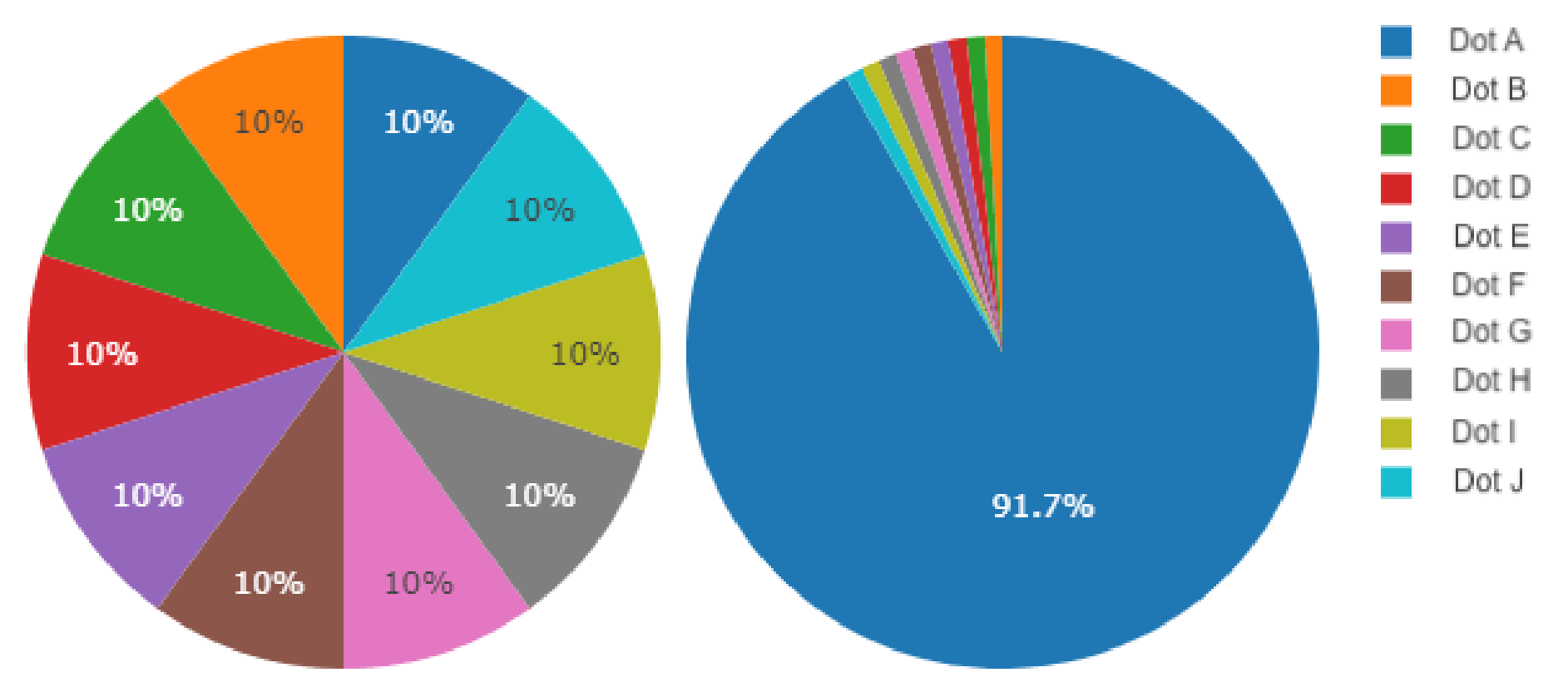

Figure 2. Equal Vs. Unequal Fitness in Roulette Wheel

Mutation

Generate a random number from 0 – 1
If this number is less than the mutation rate (0.05)
Set this gene (direction vector) as a random direction
(In Snake FNN's case, add a random number from -0.33 to 0.33 to the weight, if this makes the weight greater than 1 or less than -1, set it to 1 and -1 accordingly)

Snake FNN

Snake Vision

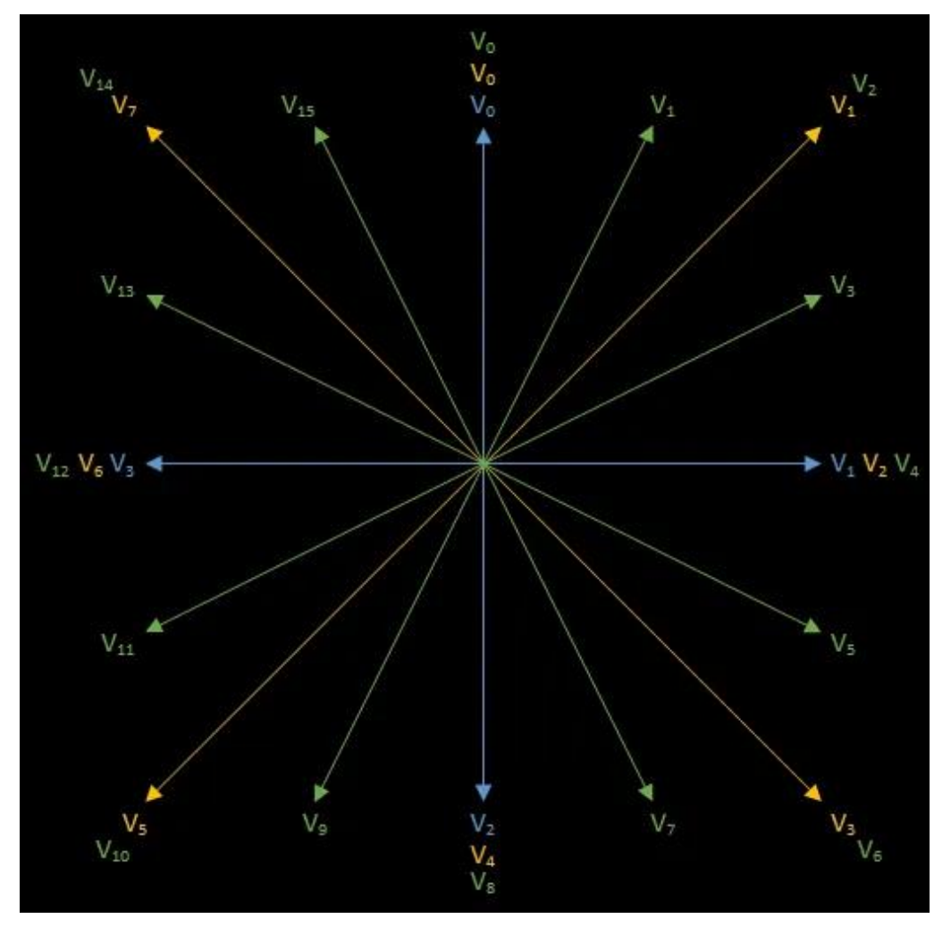

Figure 4. Depiction of 4, 8 and 16 Direction Vision

Look for:
1. Distance to wall
2. Distance to apple (if any)
3. Distance to nearest body part (if any)

Neural Network

32 input nodes (vision, head direction, tail direction), 20 + 12 hidden nodes, 4 output nodes (directions to move)

$$a^{(1)} = \sigma(Wa^{(0)} + b) = \sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix}\begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right)$$

Figure 5. Function for the Activation of Hidden Layer 1

Activation Functions

Sigmoid $f(x) = \dfrac{1}{1 + e^{-x}}$

ReLU $f(x) = max(0, x)$

Figures 5, 6. Sigmoid (used in output nodes) and ReLU (used in input and hidden nodes) Activation Functions

Fitness function

1. Reward snakes early on for exploration + finding a couple apples.
2. Have an increasing reward for snakes as they find more apples.
3. Penalize snakes for taking a lot of steps.

$$f(steps, apples) = steps + (2^{apples} + 500 \times apples^{2.1}) - (0.25 \times steps^{1.3} \times apples^{1.2})$$

Training Results (FNN)

Fitness Sum v.s Generation

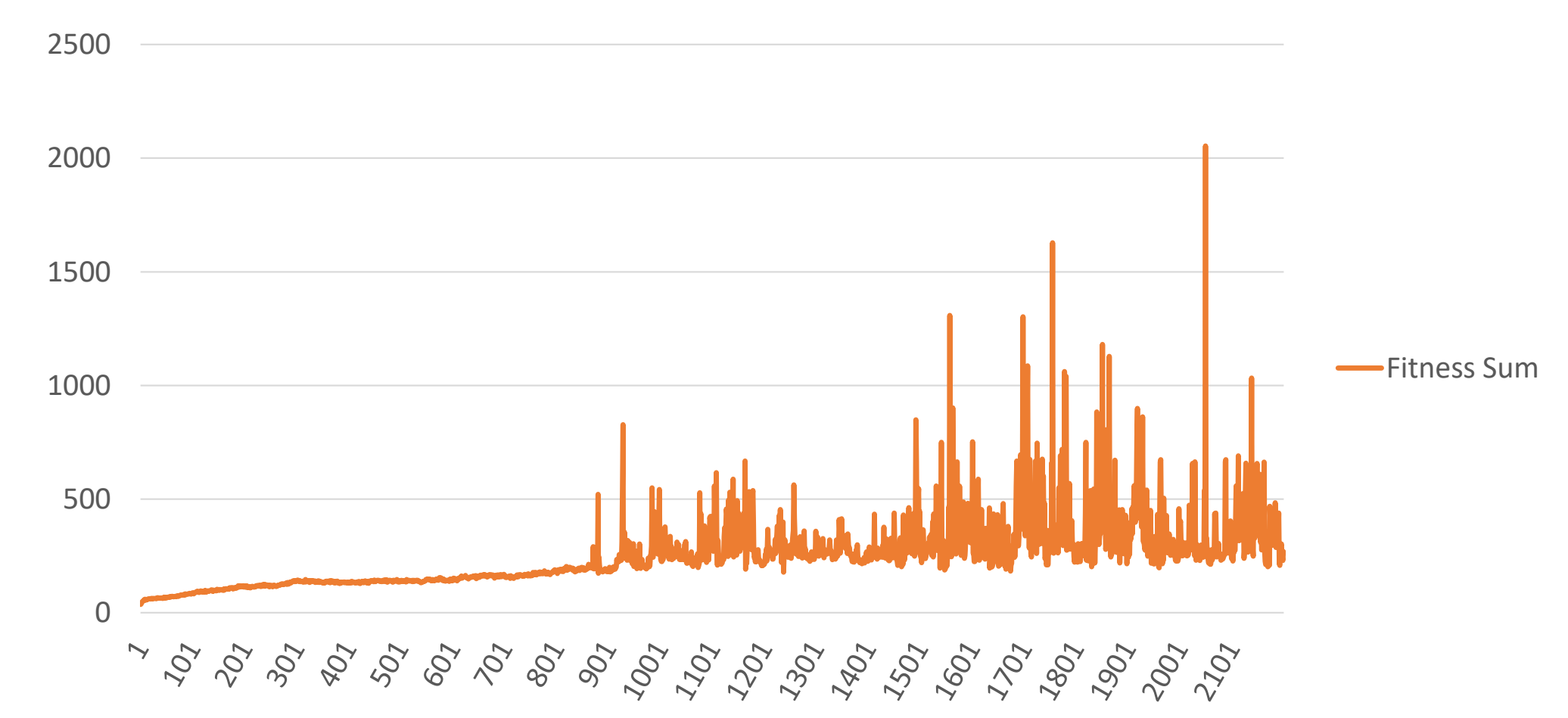

Figure 6. Fitness Sum Over Generation (Note: Cube root of Fitness Sum is used here to accommodate graph)

Best Score v.s Generation

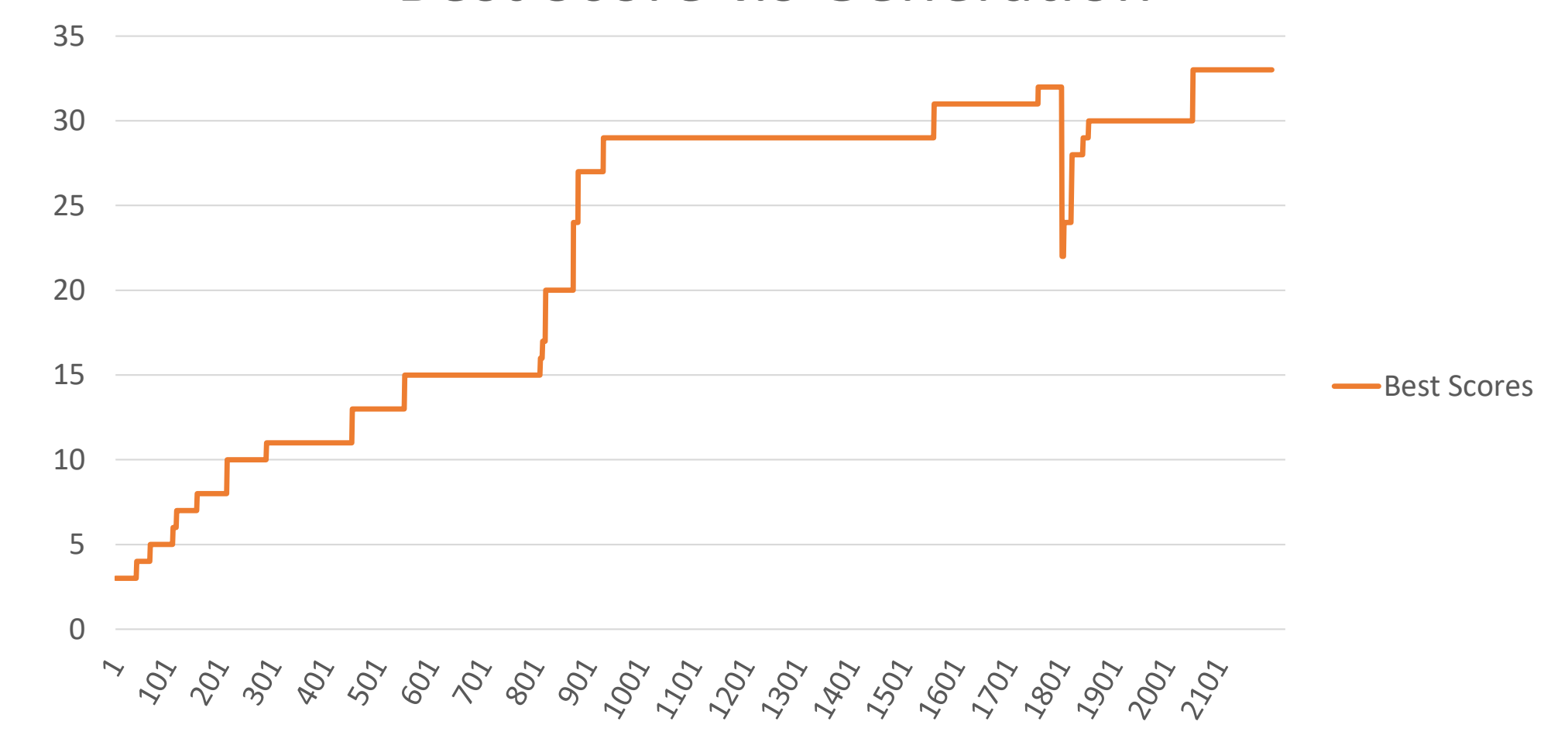

Figure 7. Best Score Over Generation (Note: the decline at generation 1801 is caused by loading generation 1800 after a pause, so the previous best score isn't carried over)

Analysis and Conclusion

As shown in Figures 6 and 7, the genetic algorithm has trained the Snake FNN to reach local maximum (local minimum for errors) over 2200 generations. As expected, the graphs show a rapid increase in the fitness sum/best score in the beginning, and this increase slows down as generation count increases, and as the fitness sum/ best score approaches the local maximum. The neural network has become more and more optimized for the problem over time. Towards the end, a popular game strategy in the population begins to emerge and becomes more and more obvious. The Smart Dots Simulation also shows success from how rapidly it finds a solution and optimizes it. Overall, this project has been very successful.

I have accomplished a number of things with my work on using genetic algorithms to train feedforward networks. In the held of GAs, I have demonstrated an application of a genetic algorithm to a simple searching problem by implementing my own adaption of GAs in a program from scratch and finding an optimal solution in a relatively short time. In the held of neural networks, I have utilized an original adaption of GAs as the training algorithm in a game of Snake where the traditional backpropagation algorithm would be inapplicable. I have also programmed a functional FNN from scratch and applied it to Snake. Through the two parts of this project, I demonstrated my learning of the principles of GAs and FNN and my ability to adapt them and apply them to solve different problems.

Application and Future Work

The work described here only touches the surface of the potential for using genetic algorithms to train neural networks. In the realm of feedforward networks, there are a number of other variables with which one might experiment. Different crossover methods, mutation methods, and fitness functions are some examples. Further studies need to be conducted on the effectiveness of other genetic algorithms (for example, NEAT) and other neural networks (for example, Long Short Term Memory). Finally, as a general-purpose optimization tool, genetic algorithms should be applicable to any type of neural network (which is also something I will work on). The existence of genetic algorithms for training could aid in the development of other types of neural networks. And all these neural networks, combined with the appropriate genetic algorithm, could be applied in various fields, such as security, medicine research, object recognition, to increase efficiency.