Process automation through the stochastic neuroevolution of fixed network topologies

Spanidis, M.

ABSTRACT

Neural networks have been widely used for regression analysis, classification, control, and predictive modelling tasks. This project investigates the performance of a single hidden layer, feed forward artificial neural network with various topologies in an effort to automate a monotonous process of shooting aliens. The agent and its environment have been developed in the Python programming language. The optimal topology for a single hidden layer network with dense perceptron's is presented within this work.

CONTENTS

NOMENCLATURE

$g(x)$	Target function	Y	Output matrix
$f(x;w)$	Generic function for a neural network	W	Weight matrix
x	Input	B	Bias matrix
w	Weight	j	Unit index
b	Bias	i	Input index
$\sigma(x)$	The activation function	l	Layer index

Introduction

Neural networks are the universal function approximators. These functions continue to show applicability in a variety of domains including medical diagnosis, financial forecasting, and machine diagnostics [1].

The aim of this project is to investigate the influence of solution convergence on a process automation task by varying the topology of a dense, single hidden layer feed forward neural network. The agent and task environment were developed in the Python programming language. The process to be automated is the series of tasks involved to complete the "Alien Invaders" game. The target outcome is for the agent to shoot aliens either before they collide with the agent or reach the bottom of the display screen.

Modelling

Environment

The task environment consists of six objects: the ship (agent), four aliens, and a bullet. At round initialization, four aliens are evenly spaced at the top of the display as illustrated in Figure 1.

Figure 1
Agent and task environment at round initialization.

Ships have access to a single bullet. If the bullet collides with an alien or reaches past the top of the display, the ship's holster is reloaded, and the ship will have access to a new bullet to fire. Should the ship's bullet collide with alien, the alien that was hit will disappear, and a new alien will spawn at a random position at the

top of the screen. Figure 2 shows a screenshot of random spawning of aliens through the duration of the round.

Figure 2
Aliens will spawn at random positions along the top of the display.

Alien's and bullets movement is constrained along the y-axis with a fixed speed. The ship is only capable of moving along the x-axis with a fixed speed. Should an alien hit the ship or reach the bottom of the display, the round is over. Additionally, should the ship move outside the boundaries of the display, the round is over.

Agent

The generic mathematical relationship of a neural network can be described as $g(x) \approx f(x;w)$. The function $f(x;w)$ is composed of subsequent functions represented as shown in Equation 2.

$$y_j = \sigma\left(\sum_{i=1}^{n} x_i w_i + b\right) \tag{2}$$

The activation function, $\sigma(x)$, exists in a variety of forms including the sigmoid, softmax, tanh, ReLU, and a step function; these functions have the purpose of introducing non-linearity [1].

Some combination of these functions produces the output $f(x;w)$, which is used to approximate $g(x)$. This is illustrated in Figure 3, whereby the output of the network is defined by Equation 2.

$$\begin{aligned} y = f(x;w) = f^{(5)}\Big(f^{(3)}\Big(f^{(1)}(x_1) \\ + f^{(4)}\left(f^{(1)}(x_1) + f^{(2)}(x_2)\right)\Big) \\ + f^{4}\left(f^{1}(x_1) + f^{2}(x_2)\right)\Big) \end{aligned} \tag{2}$$

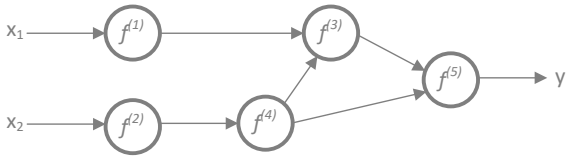


Figure 3
Example of a feed-forward neural network. Adapted from *[2]*.

For this project, a dense single hidden layer feed forward neural network is used to model the agent's response to environmental stimuli. Figure 4 illustrates the general topology of a dense feed forward network; the output of the function can be represented as Equation 3.

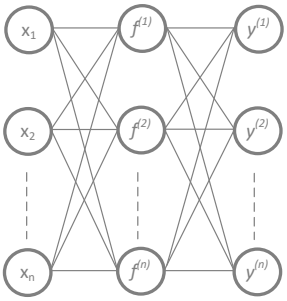


Figure 4
Example of a dense feed-forward neural network.

$$Y^{(l)} = f^{(l)}\left(Y^{(l-1)} \cdot W^{(l)} + B^{(l)}\right). \tag{3}$$

The agent's neural network has a fixed inner and outer topology with neurons corresponding to select environmental stimulus and actions respectively as illustrated in Figure 5. Hidden neurons have a sigmoid activation. The agent's actions are dependent on the maximum value of the three output neurons, these neurons have an activation function of $f(x) = \begin{cases} 1 \; if \; x > 0.5 \\ 0 \; if \; x \leq 0.5 \end{cases}$; if the maximum value of the output neuron response is less than 0.5 no action is taken. The number of hidden neurons is specified by the user before game initialization.

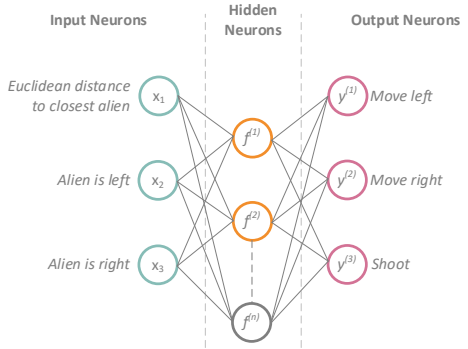


Figure 5
Agent network topology.

Learning

Agent's learn desirable outcomes through reinforcement learning. Should an agent use a set of actions such that the outcome of those actions results in an alien-bullet collision, the agent is awarded a fitness score of +1. Negative behavior such as agent's flying across the screen and out of the screen boundaries, colliding with aliens, or losing the game are punished by awarding a fitness score of -1 to the agent.

Learning is predicated on biological evolution. The premise is such that there exists some genotype that contains the genetic information necessary for maximum species success within a particular environment. A subset of the set of genotypes compete within an environment and are inherited depending on their success. In the case of neuroevolution, genotypes are represented as the combination of $W^{(l)}$ and $B^{(l)}$.

A population of agents are generated during the start of the game. Each agent contains randomly initialized $W^{(l)}$ and $B^{(l)}$. Each agent competes within the set of agents. At the end of a round, agents are ordered from how successful they were in the previous round based on their fitness score. The most successful agents are selected to be parents and are used to "breed" new agents that will compete in the next round. The w and b of these "children" consists of a random combination of the w and b of their parents along with a random chance to modify w and b to some random value. Figure 6 illustrates the process of neuroevolution.

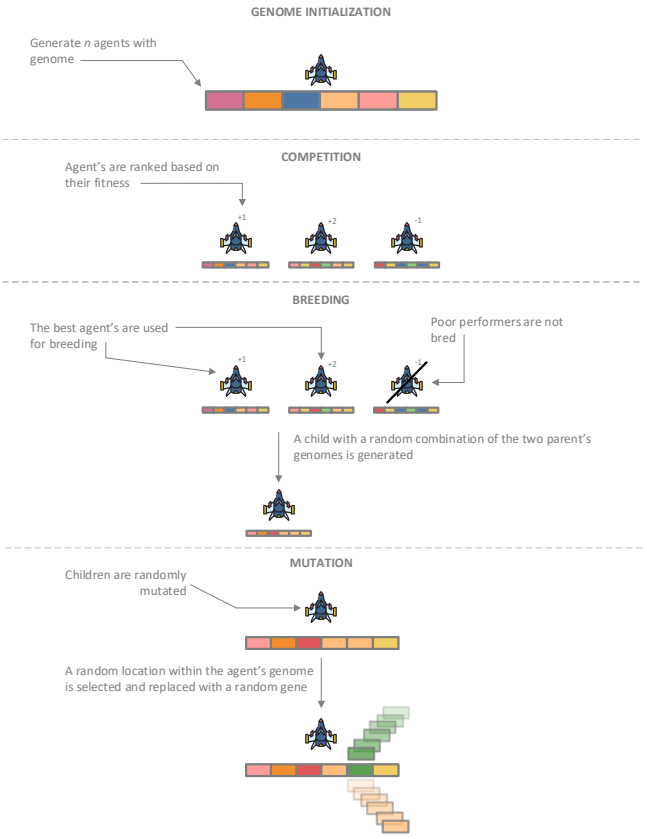


Figure 6
Illustration of neuroevolution. Colors represent the numbers contained in the weight and bias matrix and vector.

This process is repeated until a set of weights and biases exist that allow the agent to complete the task perpetually.

Methodology

A population of 60 agents were initialized with a fixed topology. Weights were randomly initialized between -0.5 and 0.5. Biases were initialized as null values. Agent's were allowed to compete until the agent was capable of perpetually completing the task or until the agent was determined to converge on an inadequate solution. Four topologies have been investigated to determine to topology that converges on the correct solution in the shortest period of time. All python scripts used area available via the link provided within the Appendix.

Results

Table 1 summarises the results for various topologies. Figure 7 illustrates the fitness of the topologies throughout each iteration. Note that for the case of a single neuron in the hidden layer, the trail was terminated after 900 iterations as it was expected that the topology would not be capable of modelling a solution to the problem.

Table 1
Summary of results.

Topology	Time elapsed to converge on solution
Single hidden neuron	Undefined
Two hidden neurons	5002s
Three hidden neurons	8004s
Four hidden neurons	12126s

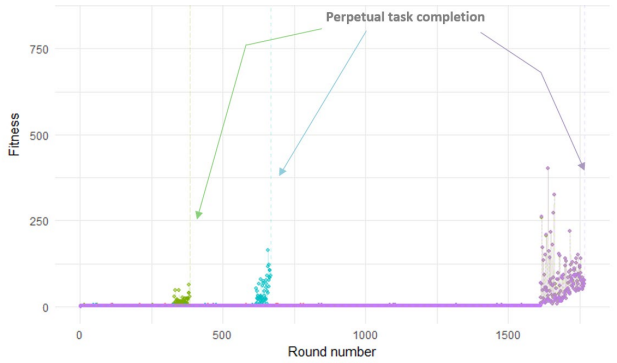


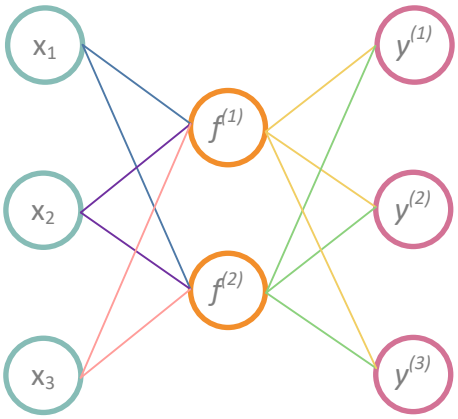
Figure 7
Fitness scores for each round based on topology. After over 900 rounds, the single neuron topology was unable to converge on a solution.

Discussion

The single neuron topology was unable to develop a satisfactory solution. This is because the target function that generalized the agent's response to the environmental stimulus likely cannot be approximated using a single neuron. Rather, at least two neurons must be used to obtain an approximation of the target function that models a satisfactory response of the agent.

Adding hidden neurons decreases the speed of solution convergence. This is a result of the increased number of parameters that must be tuned during training. The optimum topology contains the minimum neurons required to approximate the target function. Figure 8 illustrates the topology along with the parameters that resulted in the best agent.

It should be noted that the times required for training listed in Table 1 are strictly to act as a refence for comparison between topologies. The training for a specific topology can vary throughout trails as mutation occurs randomly with. The 2-neuron topology was observed to converge with as few as approximately 100 iterations and as many as approximately 500 iterations. Nevertheless, a reduction in topology will generally provide a more optimum solution as there will be fewer parameters to adjust through mutation.



Parameter	Value
$w_{1,1}^1$	0.21
$w_{1,2}^1$	0.41
$w_{2,1}^1$	-0.80
$w_{2,2}^1$	0.71
$w_{3,1}^1$	0.29
$w_{3,2}^1$	0.12
b_1^2	0.13
b_1^2	-0.52
$w_{1,1}^2$	0.29
$w_{1,2}^2$	-0.97
$w_{1,3}^2$	-0.069
$w_{2,1}^2$	-0.39
$w_{2,2}^2$	-0.55
$w_{2,3}^2$	0.73
b_1^3	0.15
b_2^3	-0.75
b_3^3	-0.72

Figure 8
Topology of the best performing agent.

Conclusion

Neuroevolution can be used to automate processes. In this project, a monotonous process of shooting aliens was automated using stochastic neuroevolution of fixed network topologies. The optimum topology was determined to have a single hidden layer of two neurons. There are several algorithms that have been developed to obtain optimum topologies using neuroevolution. Most notably is the NEAT algorithm which obtains a solution by algorithmically altering agent topologies in increasing complexity [3].

Future work includes investigating the performance of topologies using various input parameters. For example, using the screen's greyscale pixel values as input parameters with a deeper network to obtain a topology that can recognize aliens rather than explicitly being told where the alien is relative to the agent using Euclidean distances.

References

[1] L. De Marchi and L. Mitchell, Hands on Neural Networks - Learn how to build and train your first neural network model using Python, 2019.

[2] I. Vasilev, Advanced Deep Learning with Python., 2019.

[3] K. O. Stanley and R. Miikkulainen, "Efficient Evolution of Neural Network Topologies," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, 2002.

[4] B. Software, "Gentle Intro To Neural Nets #1 Learning To Play A Game," [Online]. Available: https://www.youtube.com/watch?v=pX7vlnG6eC8&list=PLZ1QII7yudbebDQ1Kiqdh1LNz6PavcptO&index=1&ab_channel=BluefeverSoftware.

[5] C. G. Ggos and Z. Tadmor, Principles of Polymer Processing, John Wiley & Sons, 1979.

[6] "Icon from flaticons," [Online]. Available: https://www.flaticon.com/free-icon/spaceship_1702089?term=spaceship&page=1&position=6. [Accessed 2020].

Appendix

The scripts used to complete this project are publicly available at: *https://github.com/michaelspanidis/michaelspanidis.github.io/tree/master/projectdocs/alien%20invasion*