

CREATING AN ARTIFICIAL GAME PLAYER FOR MODIFIED STICK RUNNING USING NEUROEVOLUTION OF AUGMENTING TOPOLOGIES (NEAT)

Presentor: Aldrich S. Goco
Adviser: Rozano S. Maniaol



Stick Running

Stick Running is an endless running game platform game. It is a single player game wherein the human player controls the game character to jump in order to avoid all of the of the varying obstacles as the game character attempt to run as far as possible. Once the game character run into any obstacle then the game will be over.



Artificial Intelligence in Video Games

- Rule-Based (Ex. IF-THEN)
- Machine Learning



Machine Learning in Video Games

- Reinforcement Learning
- Genetic Algorithm



Genetic Algorithm

- Inspired by biological evolution
- Iterative process that discovers the fitness landscape via mutations to discover optima

Key Components:

- Generate Fitness

Fitness function

- Select top organism “Survival of the fittest”

- Replicate

With chance of mutation



NEUROEVOLUTION OF AUGMENTING TOPOLOGIES (NEAT)

Genetic Algorithm + Neural Network

Fixed Topologies

- Search space is too large
- Unnecessary Complexity

Augmenting Topologies

- Necessary complexity
- Purposeful motivation behind topology changes



Genome / Neural Network

Key Components:

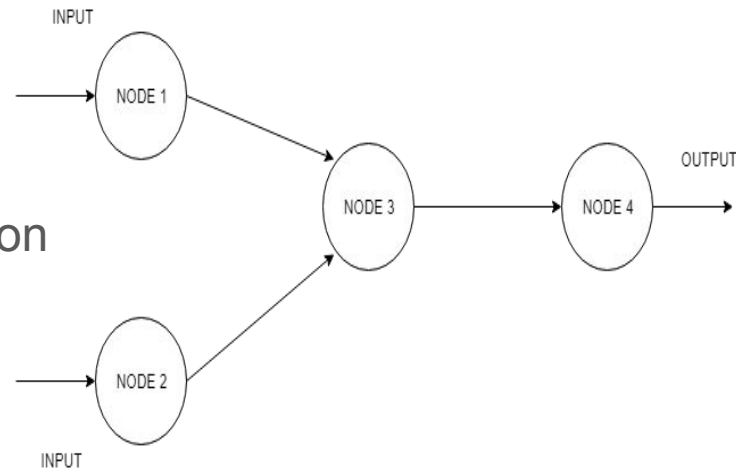
Node Gene / Neuron

Receives input

Calculates output via Activation Function

Connection Gene / Edge

Internode , weighted connection



NEURAL NETWORK

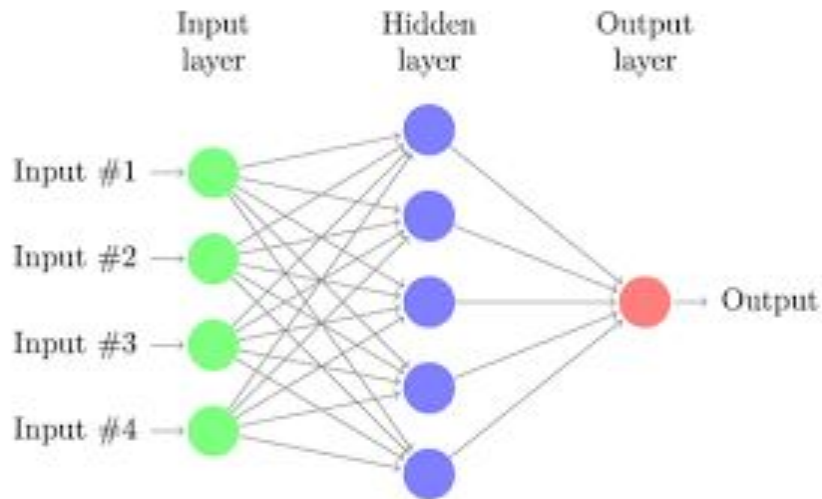
Features

-Layers

Input

Hidden

Output



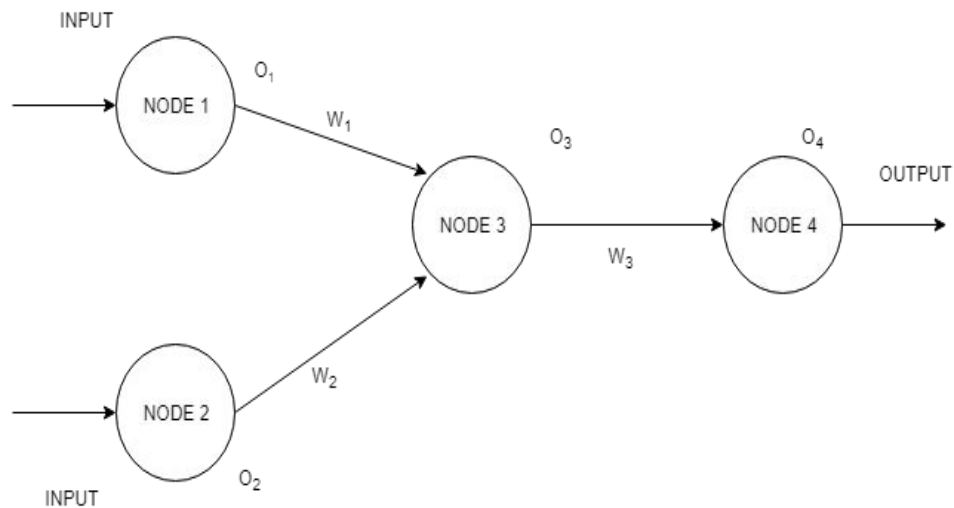
NEURAL NETWORK

$O_1 = \text{Norm}(\text{Input1})$

$O_2 = \text{Norm}(\text{Input2})$

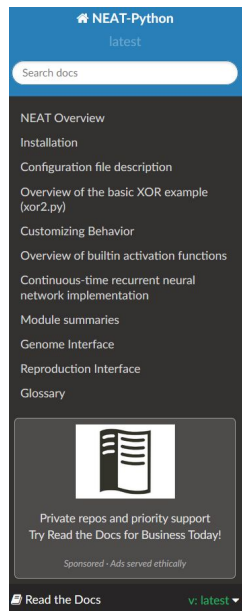
$O_3 = \text{Sigm}(O_1 * w_1 + O_2 * w_2)$

$O_4 = \text{Sigm}(O_3 * w_3)$



NEAT-Python Library

- Neural Network
- Evaluation through NEAT algorithm
- Statistics



Docs » Welcome to NEAT-Python's documentation!

[Edit on GitHub](#)

Welcome to NEAT-Python's documentation! 🐍

NEAT is a method developed by Kenneth O. Stanley for evolving arbitrary neural networks. NEAT-Python is a pure Python implementation of NEAT, with no dependencies other than the Python standard library.

Note

Some of the example code has other dependencies; please see each example's README.md file for additional details and installation/setup instructions for the main code for each. In addition to dependencies varying with different examples, visualization of the results (via [visualize.py](#) modules) frequently requires [graphviz](#) and/or [matplotlib](#). TODO: Improve README.md file information for the examples.

Support for HyperNEAT and other extensions to NEAT is planned once the fundamental NEAT implementation is more complete and stable.

For further information regarding general concepts and theory, please see [Selected Publications](#) on Stanley's website, or his recent [AMA on Reddit](#).

If you encounter any confusing or incorrect information in this documentation, please open an issue in the [GitHub project](#).

Contents:

- [NEAT Overview](#)

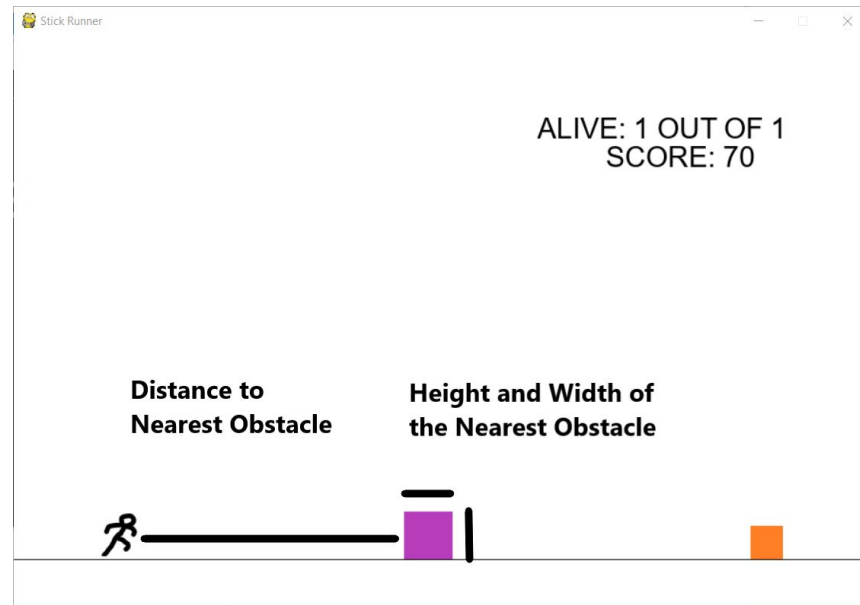
Stick Running - Game Player

Inputs

- player's distance to the nearest obstacle
- nearest obstacle's width
- nearest obstacle's height

Output

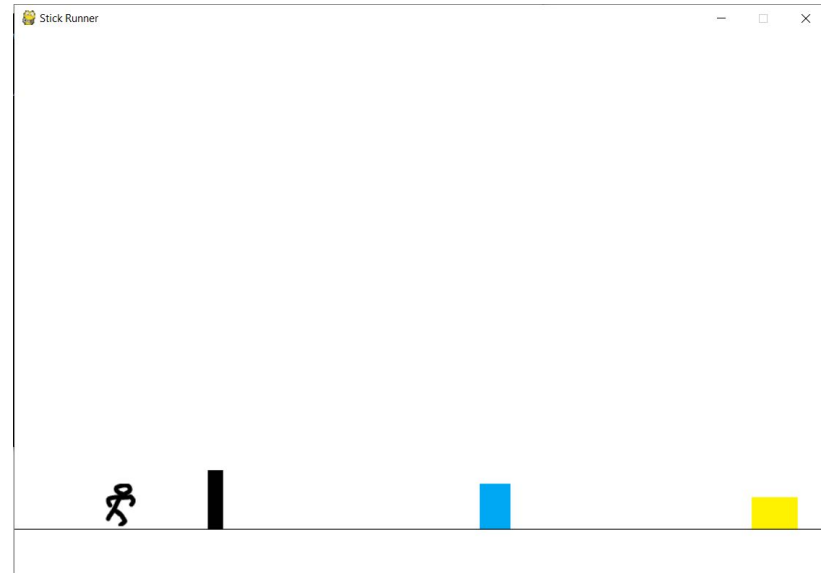
- To jump? Or not to jump



Stick Running - Fitness Function

- **Traveled Distance (TD)**: a counter that increases in every interaction of the agent to the environment
- **Score**: the number of obstacles already transposed

$$\text{FitnessFunction} = \text{Score} + (\text{TD} * 0.001)$$



Stick Running - Game Environment

These seven obstacles then are randomly generated with a minimum gap of 300 pixels to one another.



(a)



(b)



(c)



(d)



(e)



(f)



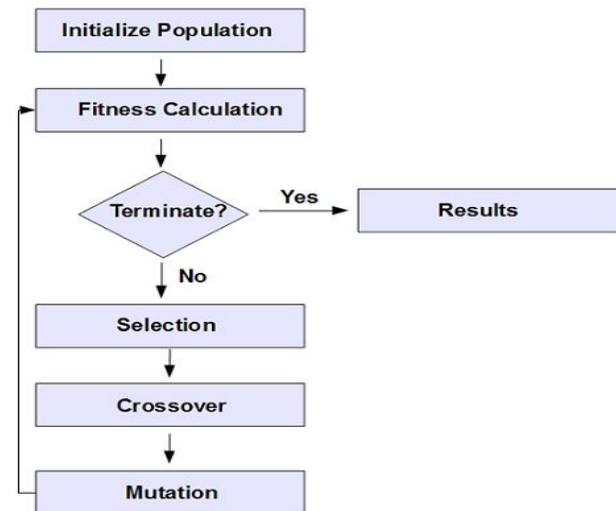
(g)

Fig. 5: Seven Different Obstacles in the Game






NEAT - Overview

- Stops when an individual reaches the Maximum fitness value or the allowed number of generation is reached.



NEAT - Fitness Calculation

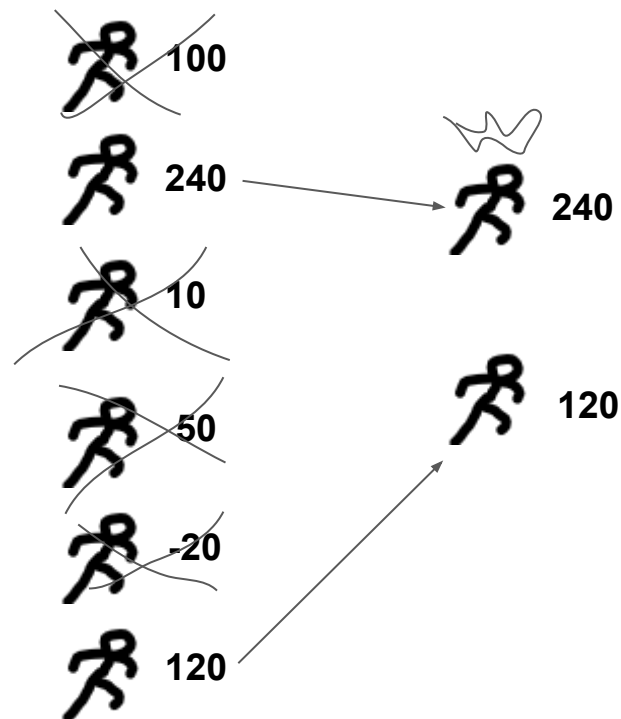
Calculates the fitness value of each individual on the population using the fitness function

 100 240 10 50 -20 120

NEAT - Selection

Elitism = 2 genomes.

Survival_threshold = 20%

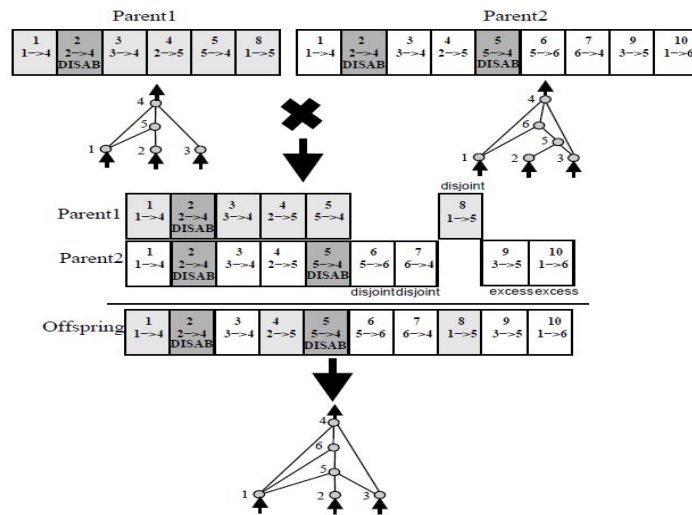


NEAT - Crossover

Innovation Number

Disjoint

Excess



NEAT - Mutation

Weight and Bias Mutate

Weight Adjust (80%)

Weight Replace (10%)

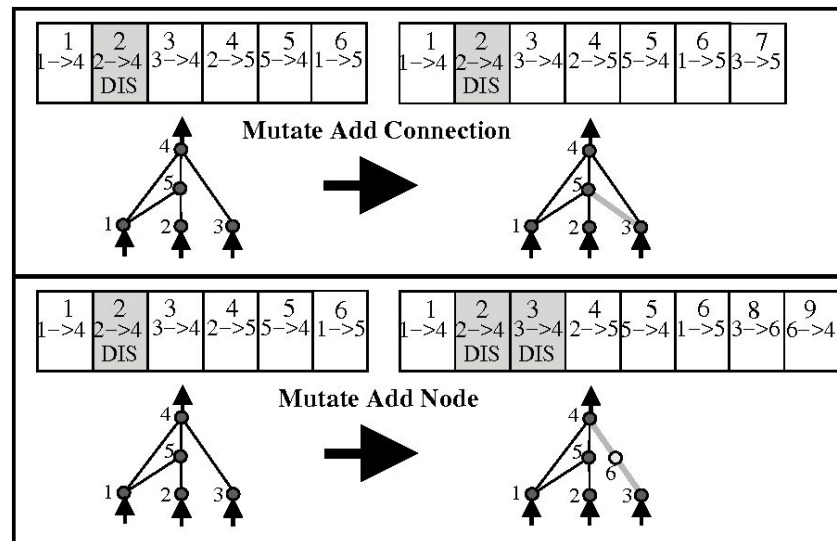
Bias Adjust (70%)

Bias Replace (10%)

Structural Mutate

Add/Delete Node (20%)

Add/Delete Connection (20%)



NEAT - Speciation

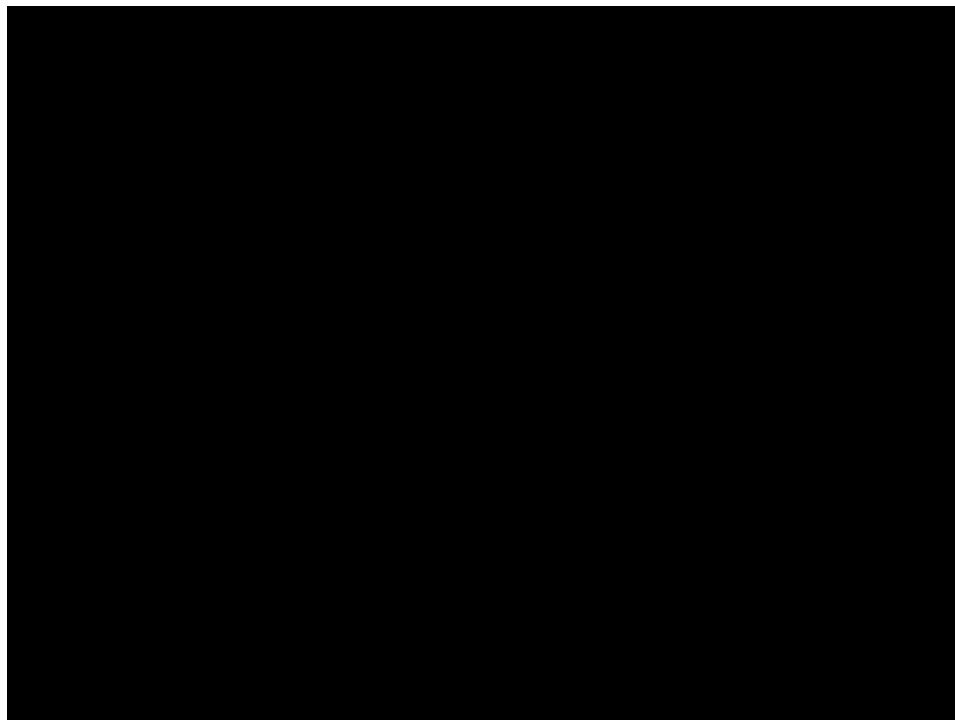
- Preserving Innovation
- Compatibility distance with the basis of Disjoint, excess and average weight of the matching

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \overline{W}$$



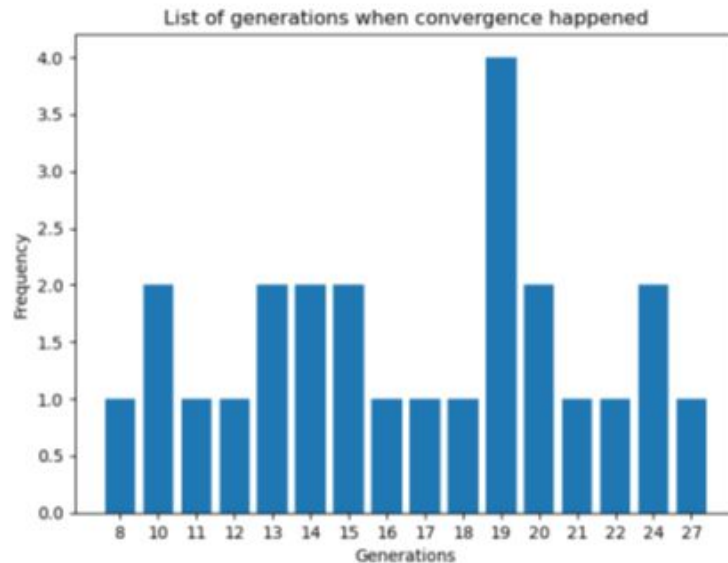
NEAT - Experiment

The NEAT algorithm using sigmoid as an activation function was run for 30 times in which the maximum generation allowed is set to 30 and the maximum fitness cap of about 50 scores or 50 obstacles to be transposed.



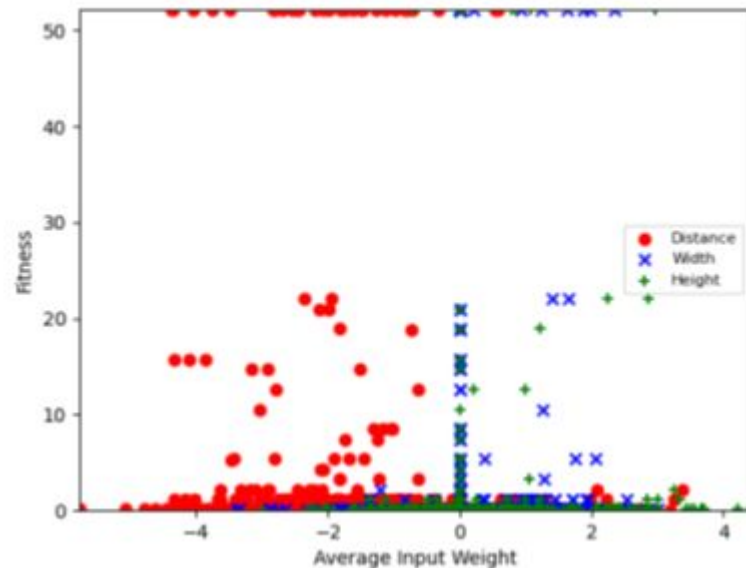
NEAT - Result (Generation x Frequency Success Training)

- In 25 runs out of 30 runs, the NEAT algorithm has successfully produced an game agent that has breached 50 obstacles.
- The experiment has shown a success rate of about 83.33 percentage.
- The generations in which the breakthrough has occurred and its frequency can be seen on the graph.
- The average generation wherein a successful game agent can be produced is at 16.8 generation.



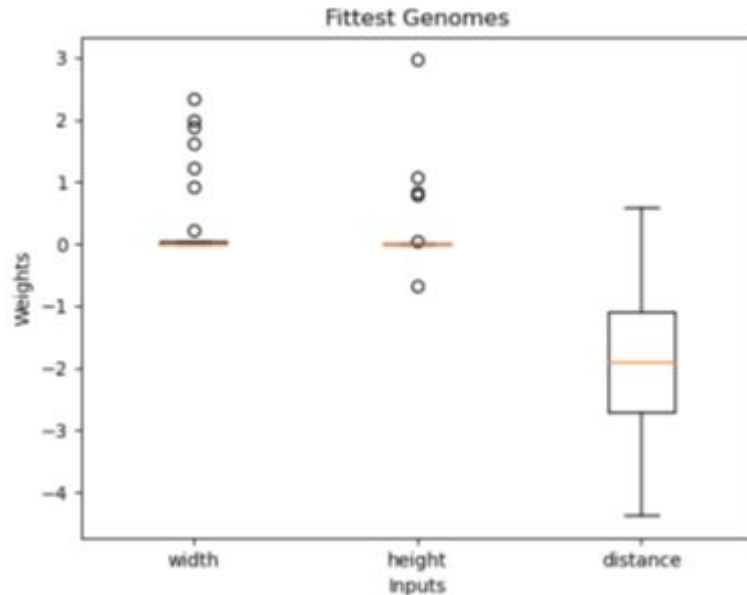
NEAT - Result (Average Input Weights x Fitness)

- After every experiment runs the final 30 individuals are recorded, which would total to 900 individuals documented after 30 trial runs.
- Each individual has 3 inputs which means there are a total of 900 average weights for each of the 3 types of input.
- zero average input weights indicates that the respective input type it belongs to is currently inactive, or in other words they are unimportant.
- 764 activated average weight input for the distanceInput
- 322 activated average weight input for the widthInput.
- 237 activated avg weight input for the heightInput.
- NEAT algorithm has decided that the distanceInput is the most important input for all the genomes while the widthInput follows behind with the heightInput taking the last.



NEAT - Result (Distribution of Average Weight inputs of fittest genomes)

- Only 28 out of 900 individuals recorded after has managed to transpose 50 obstacles
- There are 28 activated distanceInput with 26 are on negative scale while 2 are on positive scale.
- There are 7 activated widthInput with all on positive scale.
- There are 6 activated heightInput with 1 on negative scale while 5 are on positive scale.
- The distanceInput is still more influential compared to the other two input type.



NEAT - Result

- The dominant negative polarity of the average weight input of distanceInput indicates that it is inversely proportionate to the output of the topology.
- This implies that the agent will tend to not jump when the obstacle is still far.
- Inversely, it will lead the agent to jump up when the obstacles are near.



NEAT - Conclusion

By using only 30 individuals, the NEAT algorithm was run for 30 trials with a maximum of 30 generations per trial. The NEAT algorithm was able to produce an individual that has breached the maximum specified score of 50 with a success rate of 83.33 percentage and an average of breakthrough at 16.8 generations. After the experiment, the NEAT algorithm has designated that the distanceInput as the most important input amongst the other which included the widthInput and heightInput. Additionally, the NEAT algorithm has put heavy importance on designating negative weight inputs for the distanceInput to ensure inverse proportionality to the output. It ensures that the game agent will jump much proficiently whenever an obstacle is near it. Overall, this shows that the NEAT algorithm can be used to create an artificial game agent capable of playing the Modified Stick Running.

