## FUNDAÇÃO ESCOLA TÉCNICA LIBERATO SALZANO VIEIRA DA CUNHA CURSO TÉCNICO DE ELETRÔNICA

**GABRIEL DORNELLES MONTEIRO**

**Smart Kart – Inteligência artificial autodidata para condução de veículos**

## Orientador: Raul Faviero de Mesquita

**Novo Hamburgo, 2019.**

**GABRIEL DORNELLES MONTEIRO**

**SMART KART – INTELIGÊNCIA ARTIFICIAL AUTODIDATA PARA CONDUÇÃO DE VEÍCULOS**

Relatório apresentado à comissão avaliadora da Mostratec.

Orientador: Raul Faviero de Mesquita

## Novo Hamburgo, Outubro de 2019.

## RESUMO

O Smart Kart é um algoritmo para aprendizado de máquina que conta com a criação de cérebros artificiais para condução de veículos criados do zero pela própria máquina. A ideia central é que o mesmo cérebro consiga dirigir em diferentes pistas, sendo isso inviável através do algoritmo original que é aplicado, o NEAT, já que este é um algoritmo para solução de problemas específicos, onde para uma nova pista é necessário um novo treino e um novo cérebro artificial. O algoritmo Neuro Evolution of Augmenting Topologies (NEAT) é aplicado inicialmente, pois sua arquitetura de algoritmo genético permite a criação de cérebros artificiais partindo da menor estrutura possível de um cérebro, ao mesmo tempo que conta com a especiação, que protege a inovação apresentada pelo algoritmo ao longo das gerações de soluções propostas. Utilizando este algoritmo com a linguagem Lua e a API Bizhawk, que permite a escrita e leitura de dados no Mario Kart 64, foi montado um novo algoritmo para condução de veículos utilizando o Mario Kart 64 como simulador, onde existem diversos modelos de pistas diferentes. O algoritmo conta com a visão de uma imagem 12x12 que representa a ele o que é o asfalto onde ele deve dirigir e o que não é, e baseado nisso, o algoritmo cria topologias de cérebros artificiais que mapeiam essa imagem até as 8 saídas possíveis(acelerar, frear, direita, esquerda, cima, baixo, L, R), transformando essa informação da imagem junto a conexões e neurônios em algo que dão sentido ao seu comportamento, assim como fazem os humanos e animais. Para o controle e transformação do algoritmo foi implementada funções que cruzam os genomas (o conjunto de todos os genes, ou seja, o cérebro) de diferentes espécies e também um algoritmo de sobre população, que da aos genomas do algoritmo uma oportunidade de adaptar os cérebros que desenvolvem em uma pista para outra sem a necessidade de complexifica-los a ponto de se tornarem específicos da pista onde treinaram, além de claro toda a codificação do algoritmo original de evolução do NEAT e as devidas adaptações para que o mesmo apresente boas curvas de aprendizado. Dessa maneira temos a estrutura de uma rede neural, um dos modelos de machine learning, onde a imagem são as entradas, os botões possíveis são as saídas, e a hidden layer é produzida pelo próprio algoritmo, simulando um cérebro real que cria conexões e neurônios para resolver os problemas. Os melhores cérebros no sentido de desempenho, são cruzados e levados a frente, dessa maneira, quando um bom resultado é alcançado, diversos cérebros novos são construídos a partir deste, podendo então melhorar o desempenho através de pequenas mudanças chamadas mutações, ou não, a chave para isto é que eventualmente alguma das espécies mostra uma boa solução, e a mesma domina as outras dentro da população global de soluções, evoluindo mais e mais, até alcançar maestria.

Palavras chave: Algoritmo Genético, Machine learning, Redes neurais artificiais, Deep learning, Neuroevolução.

## ABSTRACT

The Smart Kart is a machine learning algorithm that create artificial brains for driving vehicles from scratch by the machine itself. The central idea is that the same brain can drive on different tracks, which is not viable through the original algorithm that is applied, called NEAT, as this is a specific problem-solving algorithm, where a new training is needed, and a new artificial brain, for each track. The Neuro Evolution of Augmenting Topologies (NEAT) algorithm is initially applied because its genetic algorithm architecture allows the creation of artificial brains from the smallest possible structure of a brain, while relying on speciation, which protects the innovation presented by the algorithm over the generations of proposed solutions. Using this algorithm with the Lua language and Bizhawk API, which allows writing and reading data on Mario Kart 64, a new vehicle driving algorithm was built using Mario Kart 64 as a simulator, where there are several different track models. The algorithm relies on the vision of a 12x12 image representing what is the asphalt where it should drive and what is not, and based on that, the algorithm creates artificial brain topologies that map this image to the 8 possible outputs (accelerate, brake, right, left, up, down, L, R), transforming this image information using connections and neurons into something that gives meaning to their behavior, just as humans and animals do. For the control and transformation of the algorithm functions have been implemented, like the one that cross the genomes (the set of all genes, ie the brain) of different species and also an overpopulation algorithm, which gives the algorithm genomes an opportunity to adapt the brains that have been developed from one track to another without having to complexify them to the point where they become specific to the track where they trained, and of course all the coding of the original NEAT algorithm and the appropriate adaptations to make it show us good curves of learning. Thus we have the structure of a neural network, one of the machine learning models, where the image pixels are the inputs, the possible buttons are the outputs, and the hidden layer is produced by the algorithm itself, simulating a real brain that creates connections and neurons to solve the problems. The best performance brains make a crossover, and their genes are passed on to the next generations. So when a good result is achieved, a lot of new brains are built from it, then the performance can be improved through small changes, called mutations, or not. The key to this is that eventually one of the species shows a good solution, and it dominates the others within the global population, evolving more and more until mastery is achieved.

Keywords: Genetic Algorithm, Machine learning, Artificial neural network, Deep learning, Neuro-evolution.

**LISTA DE FIGURAS**

# Figura 1: Topologia básica de uma rede neural .................................................. 10

# Figura 2: Modelo de rede neural Feed Forward ................................................... 11

# Figura 3 : Dados do algoritmo NEAT para o problema DPV

# (Double Pole Balancing with velocities) ………………........................................... 13

**Figura 4: Double Pole Balancing with velocities……………….……………………. 13**

**Figura 5: Benchmark dos algoritmos para o DPV ................................................. 14**

**Figura 6: Estrutura básica de um genoma no NEAT ............................................. 19**

**Figura 7: Função de ativação Binary Step ............................................................. 20**

**Figura 8: Função de ativação Sigmoide, original e variante ................................ 21**

**Figura 9: Funcionamento dos pesos em um neurônio ......................................... 22**

**Figura 10: Mutação por conexões topologicamente ............................................. 25**

**Figura 11: Mutação por adição de neurônio topologicamente ............................. 25**

**Figura 12: O crossover de duas topologias ............................................................ 27**

**Figura 13: Gráfico do funcionamento do Overpopulation ..................................... 33**

**Figura 14: Treinamento inicial do Smart Kart, Frappe Snowland ......................... 35**

**Figura 15: Pistas do Mario Kart 64 ........................................................................... 36**

**Figura 16: Bizhawk RAM Search .............................................................................. 37**

**Figura 17: Mapa dos registradores de Mario Kart 64 ............................................. 38**

**Figura 18: Mapa de colisões nos registradores ..................................................... 39**

**Figura 19: RenaKunisaki Github .............................................................................. 40**

**Figura 20: Form para controle do Smart Kart ......................................................... 41**

**Figura 21: Interface gráfica do Smart Kart .............................................................. 42**

**Figura 22: Geração x Desempenho MMF TWEANN’s ............................................. 43**

**Figura 23: Geração x Desempenho LR TWEANN’s ................................................ 44**

**Figura 24: Geração x Desempenho KPT TWEANN’s .............................................. 45**

**Figura 25: Geração x Desempenho FS TWEANN’s ................................................. 46**

**Figura 26: Geração x Desempenho MR TWEANN’s ................................................ 47**

**Figura 27: Smart Kart, benchmark final, EIC / NEAT ………………………………… 48**

**SUMÁRIO**

**......................................................................................................................... 7**

**1. INTRODUÇÃO .......................................................................................... 8**

**1.1. Tema ......................................................................................................... 8**

**1.2. Justificativa .............................................................................................. 8**

**1.3. Problema ................................................................................................... 8**

**1.4. Objetivo ..................................................................................................... 8**

**2. REFERÊNCIAL TEÓRICO ........................................................................ 9**

**2.1. Algoritmo Genético .................................................................................. 9**

**2.2. Redes neurais Artificiais .......................................................................... 9**

**2.3. Neuro Evolution of Augmenting Topologies ......................................... 12**

**2.4. API Bizhawk .............................................................................................. 15**

**2.5. Linguagem Lua ......................................................................................... 15**

**2.6. Compiladores e interpretadores ............................................................. 16**

**3. METODOLOGIA ..................................................................................... 17**

**3.0.1. Introdução ao funcionamento do algoritmo ....................................... 17**

**3.1. A codificação da estrutura de um genoma ......................................... 17**

**3.1.1. Neurônios e função de ativação .......................................................... 19**

**3.2. Iniciando uma população ..................................................................... 23**

**3.3. Mutações genéticas .............................................................................. 23**

**3.3.1. Mutação por pesos ............................................................................... 23**

**3.3.2. Mutação por conexões ......................................................................... 24**

**3.3.3. Mutação por adição de neurônio ......................................................... 25**

**3.4. Crossover e Especiação ....................................................................... 28**

**3.4.1. Como é feita a Especiação? ................................................................. 30**

**3.4.2. Cálculo do compartilhamento de desempenho................................... 30**

**3.5. Parâmetros para a primeira população ................................................ 31**

**3.5.1. Early Interspecies Crossover................................................................. 31**

**3.5.2. Overpopulation........................................................................................ 32**

**3.6. Mapeamento e pistas benchmark.......................................................... 35**

**3.7. Leitura e escrita de dados na memória (registradores)....................... 37**

**3.8. Armazenamento dos dados da rede neural ...................................................... 40**

**3.9. Interface gráfica ................................................................................................... 41**

**4. ANÁLISE DE DADOS ........................................................................................... 43**

**4.1. Curvas de aprendizado com inicialização TWEANN’s ..................................... 43**

**4.2. Curva de aprendizado com inicialização EIC / NEAT ....................................... 48**

**5.CONCLUSÃO ............................................................................................................ 49**

**REFERÊNCIAS ............................................................................................................. 50**

**1. INTRODUÇÃO**

**1.1 Tema**

Algoritmo genético de evolução, ciências da computação.

**1.2 Justificativa**

Nos dias de hoje mais do que nunca estão presentes em nossas vidas algoritmos inteligentes que analisam o tipo de conteúdo que consumimos e nos recomendam conteúdos parecidos, como o Youtube, Spotify, Netflix, e mesmo sites de vendas de produtos hoje trabalham com algoritmos em conjunto com redes sociais como o Facebook, onde acabam aparecendo produtos semelhantes ou até mesmo os que você procura em um anúncio na rede social. Estes algoritmos por sua vez são programados para estudar, analisar e reconhecer padrões, o que podemos ver que é muito mais eficiente comparado a um algoritmo pré-programado para realizar uma tarefa especifica, que eram os que estavam no mercado antes da ascensão destas empresas. A aplicação das redes neurais artificiais (RNA) tem sido usada para resolver uma grande quantidade de tarefas que são difíceis de resolver utilizando programação baseada em regras comuns. Um algoritmo que consegue resolver diferentes problemas apresentados a ele sem a intervenção humana é de grande ajuda em diferentes áreas do mercado.

**1.3 Problema**

Como desenvolver um algoritmo capaz de aprender a dirigir as pistas de corrida em Mario Kart 64?

**1.4 Objetivo**

Desenvolver um algoritmo que sem auxilio externo humano consiga aprender a conduzir veículos, e prove isto conduzindo-o em diferentes percursos.

# 2. REFERÊNCIAL TEÓRICO

# 2.1 Algoritmo Genético

# O algoritmo genético é um método para resolver problemas de otimização restritos e irrestritos, que se baseia na seleção natural, o processo que impulsiona a evolução biológica. O algoritmo genético modifica repetidamente uma população de soluções individuais. A cada passo, o algoritmo genético seleciona indivíduos aleatoriamente da população atual para serem pais e os usa para produzir os filhos para a próxima geração. Por gerações sucessivas, a população "evolui" em direção a uma solução otimizada. Você pode aplicar o algoritmo genético para resolver uma variedade de problemas de otimização que não são adequados para algoritmos de otimização padrão, incluindo problemas nos quais a função objetivo é descontínua, não diferenciável, estocástica ou altamente não-linear.O algoritmo genético usa três tipos principais de regras em cada etapa para criar a próxima geração a partir da população atual:

# As regras de seleção selecionam os indivíduos, chamados pais, que contribuem para a população na próxima geração.

# As regras de cruzamento combinam dois pais para formar filhos para a próxima geração.

# As regras de mutação aplicam mudanças aleatórias em pais individuais para formar filhos.

# Baseado nisto, os algoritmos genéticos possuem hoje uma vasta bibliografia de diferentes autores que criam algoritmos baseado nesta ideia de evolução biológica.

# 2.2 Redes Neurais Artificiais

# Em ciência da computação e campos relacionados, redes neurais artificiais (RNAs) são modelos computacionais inspirados pelo sistema nervoso central de um animal (em particular o cérebro) que são capazes de realizar o aprendizado de máquina bem como o reconhecimento de padrões, simulando o comportamento de redes neurais biológicas. As redes neurais artificiais hoje vem conseguindo resolver uma série de problemas que não eram resolvidos pela programação estrutural.

# Figura 1: Topologia básica de uma rede neural.

# 

# Fonte: Research Gate, Neural Network Topology.

# Este é o modelo mais simples de rede neural artificial, e ele funciona de maneira análoga a nosso cérebro humano, e também o animal. Nós possuímos sentidos: visão, audição, olfato, paladar e tato, e baseado nisto, nosso cérebro cria conexões entre nossos neurônios, isso nos faz ter certo comportamento baseado em que tipo de informação estamos recebendo destes sentidos. Para a rede neural artificial é semelhante, a camada de entrada (Input Layer) recebe informações do que estamos trabalhando, os como pixels de uma imagem ou os sinais de áudio, passa essas informações para a Hidden Layer, que são os neurônios, e nesses neurônios a informação de entrada é processada de alguma maneira entregando algo na camada de saída (Output Layer). Um exemplo simples pra entender isto é o reconhecimento de imagem: Damos a rede neural artificial uma imagem de um gato, ela processa os pixels dessa imagem (que são suas entradas) na hidden layer e nos entrega algum resultado na saida, definida por nós humanos, exemplo: 1 = gato, 2 = cachorro. Para que isso ocorra existem diferentes algoritmos para o aprendizado dessa rede neural artificial.

# Dentro da área de RNAs existem diversos algoritmos que são aplicados, sendo que muitos deles estão presentes diariamente na nossa vida. Alguns dos mais famosos são: Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), sendo estes, modelos de redes neurais considerados “Feed Forward”.

# Figura 2: Modelo de rede neural Feed Forward.

# 

# Fonte: Sean Cantrell, 2018.

# O modelo Feed Forward consiste em ligar todas as entradas a todos neurônios na hidden layer, e quando há mais de uma hidden layer, acontece o que chamamos de Deep Learning, com um maior processamento de dados, mas também obviamente, uma maior precisão por parte do algoritmo em realizar sua tarefa.

# Além dos algoritmos de Feed Forward, existem também os algoritmos genéticos sendo aplicados dentro da área de Redes neurais artificiais, dentre eles, o que é a base desta pesquisa é o próximo tópico a ser abordado.

# 2.3 Neuro Evolution of Augmenting Topologies

# Neuro Evolution of Augmenting Topologies (NEAT) é um algoritmo genético desenvolvido por Kenneth O. Stanley e Riisto Miikkulainen do departamento das ciências da computação da universidade do Texas, em Austin. Trata-se de um algoritmo que simula a rede neural do cérebro capaz de reconhecer padrões. A ideia central do algoritmo e o que ele apresenta comparado ao melhor método de topologia fixa é explicada por Kenneth O. Stanley e Risto Miikkulainen no MIT Press uma editora universitária afiliada ao Massachusetts Institute of Technology, em Cambridge, Massachusetts:

# An important question in neuroevolution is how to gain an advantage from evolving neural network topologies along with weights. We present a method, NeuroEvolution of Augmenting Topologies (NEAT) that outperforms the best fixed-topology method on a challenging benchmark reinforcement learning task. We claim that the increased efficiency is due to (1) employing a principled method of crossover of different topologies, (2) protecting structural innovation using speciation, and (3) incrementally growing from minimal structure. We test this claim through a series of ablation studies that demonstrate that each component is necessary to the system as a whole and to each other. What results is significantly faster learning. NEAT is also an important contribution to GAs because it shows how it is possible for evolution to both optimize and complexify solutions simultaneously, offering the possibility of evolving increasingly complex solutions over generations, and strengthening the analogy with biological evolution.

# Tradução:

# Uma questão importante na neuroevolução é como obter uma vantagem da evolução das topologias de redes neurais junto com os pesos. Apresentamos um método, NeuroEvolution of Augmenting Topologies (NEAT), que supera o melhor método de topologia fixa em uma tarefa desafiadora de aprendizado de reforço de benchmark. Afirmamos que a maior eficiência se deve a (1) empregar um método baseado em princípios de cruzamento de topologias diferentes, (2) proteger a inovação estrutural usando a especiação e (3) crescer progressivamente a partir da estrutura mínima. Testamos essa afirmação através de uma série de estudos de ablação que demonstram que cada componente é necessário para o sistema como um todo e para o outro. O que resulta é uma aprendizagem significantemente mais rápida. O NEAT também é uma contribuição importante para os AGs (algoritmos genéticos) porque mostra como é possível a evolução tanto para soluções otimizadas como complexas simultaneamente, oferecendo a possibilidade de evoluir soluções cada vez mais complexas ao longo de gerações e fortalecendo a analogia com a evolução biológica. (STANLEY, 2002, tradução nossa).

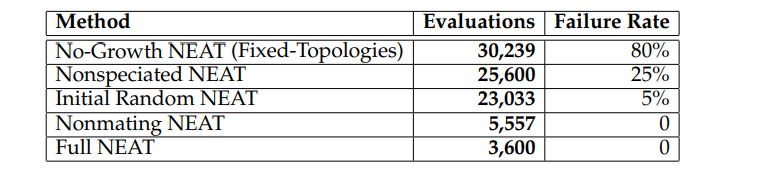
Os motivos do NEAT superar a melhor topologia fixa apresentados por eles são:

1° - É empregado o cruzamento de diferentes topologias, o chamado Crossover.

2° - As inovações estruturais são protegidas através da especiação.

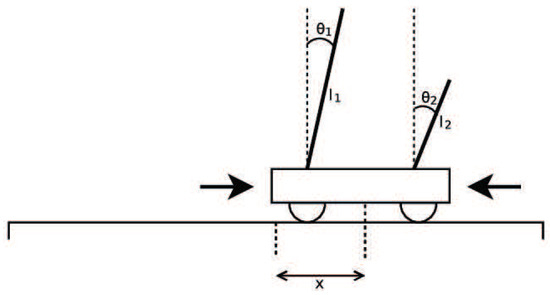
3° - O crescimento incremental é feito a partir de uma estrutura mínima. Os componentes do sistema são necessários para como um todo e para si mesmos, o que resulta em uma aprendizagem mais rápida. NEAT apresenta uma grande contribuição aos algoritmos genéticos pois ele mostra como é possível para a evolução otimizar e complexar soluções simultaneamente ao decorrer das gerações, o que é justamente a evolução biológica que é a analogia proposta inicialmente pela Neuroevolução. Kenneth O. Stanley é membro fundador da Uber Labs e também desenvolvedor do HyperNEAT, tendo contribuído bastante para os algoritmos genéticos durante sua vida com desenvolvimentos e aplicações. No paper existem os dados e explicações de diversos benchmarks provando a superioriade do NEAT mesmo em relação a outros algoritmos genéticos que não contam com a especiação.

**Figura 3 : Dados do algoritmo NEAT para o problema DPV (Double Pole Balancing with velocities).­­­**



Fonte: Evolutionary Computation Volume 10, Number 2, página 117.

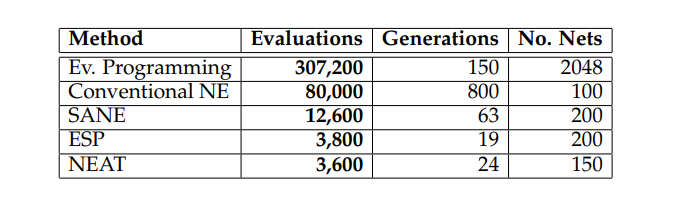
**Figura 4: Double Pole Balancing with velocities.**



Fonte: MDPI.

O DPV é um benchmark comum entre algoritmos, trata-se de balancear dois polos diferentes utilizando direita e esquerda baseado nos ângulos dos polos e na velocidade da base onde estão postos, que desloca-se reagindo a sua movimentação e também ao ângulo atual dos polos. É notável que o algoritmo completo do NEAT consegue tratar o problema com mais precisão e com um número bem menor de avaliações, principalmente comparado ao Nonspeciated NEAT, que é em essência o esqueleto de um algoritmo genético comum hoje. O algoritmo também é comparado a outros algoritmos que eram utilizados para redes neurais recorrentes (RNN), como o ESP (Enforced Subpopulations) e o SANE (Symbiotic, Adaptive Neuro-Evolution), algo interessante é que Riisto Mikkulainen é um dos autores tanto dos dois algoritmos acima quanto do NEAT, o que mostra seu grande envolvimento com a neuroevolução.

**Figura 5: Benchmark dos algoritmos para o DPV.**



Fonte: Evolutionary Computation Volume 10, Number 2, página 115.

Outros benchmarks são realizados, como o DPNV (Double Pole Balancing No Velocities). Todos os testes estão no NEAT paper, ou no livro Evolutionary Computation Volume 10, Number 2. Estes benchmark’s mostram que é um ótimo algoritmo de neuroevolução, e recentemente, ele voltou a ser estudado e aplicado, visto sua performance e o fato de não ser um algoritmo de topologia fixa.

**2.4 API Bizhawk**

O Bizhawk é um emulador de multi sistemas, como Super Nintendo, Nintendo 64 e Playstation 1. O diferencial do Bizhawk em relação a emuladores existentes que visam a emulação como seu foco, é que o Bizhawk possui ferramentas para a leitura de dados das ROM’s (read only memory) dos jogos que emula, além de possuir um sistema para a leitura de scripts em Lua. O emulador já disponibiliza algumas bibliotecas, sendo elas: Userdata, bit, emu, event, gameinfo, genesis, joypad, mainmemory, memory, memorysavestate, movie, nes, snes, SQL, bizstring, client, comm, console, forms, gui, input, savestate, tastudio e LuaCanvas. A biblioteca mainmemory permite a leitura e escrita de dados direto nos registradores da ROM de Mario Kart 64, com uma vasta possibilidade de escolha: leitura e escrita de float, 8, 16, 24 e 32 bits podendo ser Big Endian ou Little Endian (leitura do bit mais signiticativo), sendo possível escolher se a leitura/escrita é Signed ou Unsigned também. A biblioteca GUI( Graphical User Interface) é a responsável por fazer a visualização de o que esta acontecendo por tras do algoritmo através de recursos gráficos simples, como linhas, retângulos e textos, e a forms é responsável por criar janelas de controle disponibilizando caixas de texto, checkbox etc...

**2.5 Linguagem Lua**

A linguagem de programação Lua é a primeira linguagem desenvolvida fora do primeiro mundo, tendo sido inteiramente projetada na PUC-Rio, no Rio de Janeiro. Lua permite programação procedural, programação orientada a objetos, programação funcional, programação orientada a dados e descrição de dados, sendo uma linguagem projetada para estender aplicações. Vários benchmarks mostram Lua como a linguagem mais rápida dentre as linguagens de script interpretadas. É uma linguagem muito utilizada em diversas aplicações industriais como o Photoshop e Ginga (TV digital), e também em jogos hoje de grande renome como o World of Warcraft e Angry Birds. A linguagem é descrita por seus autores: Roberto Ierusalimschy, Luiz Henrique de Figueiredo, Waldemar Celes como:

Lua é poderosa (e simples). Um conceito fundamental no projeto de Lua é fornecer meta-mecanismos para a implementação de construções, em vez de fornecer uma multidão de construções diretamente na linguagem. Por exemplo, embora Lua não seja uma linguagem puramente orientada a objetos, ela fornece meta-mecanismos para a implementação de classes e herança. Os metamecanismos de Lua trazem uma economia de conceitos e mantêm a linguagem pequena, ao mesmo tempo que permitem que a semântica seja estendida de maneiras não convencionais.

Lua é distribuída sob a licença MIT, licença criada pelo Instituto de tecnologia de Massachusetts e pode ser usada até mesmo para fins comerciais sem custo ou burocracia. Lua é usada para estender programas escritos não só em C e C++, mas também em Java, C#, Smalltalk, Fortran, Ada, Erlang, e mesmo outras linguagens de script, como Perl e também Ruby.

A linguagem Lua não possui uma função main, seu código roda dentro de um loop while caso queira ser repetido, ou na ausência de um o código roda uma única vez, lendo tudo desde a primeira linha até a última. Ela também não possui variáveis, mas sim valores, e esses valores possuem tipos, que podem ser: nil, boolean, number, string, function, userdata, thread e table. Nenhum dos valores precisa declarar seu tipo, tudo que precisa fazer é declarar um valor e dar um conteúdo a ele que o Lua interpretará que tipo ele é, mas é possível declara-lo como “Local”, dizendo que ele só pertence a função onde foi declarado. O table é a única estrutura de dados em Lua, e ele funciona semelhante a arrays e estruturas padrão das outras linguagens, a diferença é que ao declarar um table ele pode ser tanto um array quanto uma estrutura (struct), e seus índices podem ser outros tables.

**2.6 Compiladores e interpretadores**

Inicialmente, para o aprendizado da linguagem utilizei uma IDE Eclipse, o Lua Development Tools, um compilador simples assim como o Dev c++ para C e C++, que possibilita a aplicação de algoritmos simples e uma janela console. Lua Development Tools (LDT) é uma IDE (Integrated Development Environment), ou ambiente de desenvolvimento integrado em português, distribuída pela Fundação Eclipse que é também desenvolvedora de outras diversas IDE’s para outras linguagens. LDT é uma ferramenta open source licenciada sob EPL (licença pública Eclipse). O LDT utiliza o Metalua (um compilador de linguagem) para analisar o código fonte lua e fornecer assistência

ao usuário.

O compilador principal é o próprio Bizhawk mencionado anteriormente, desta maneira, a IDE que utilizo para visualização e edição do código é o Sublime Text 3, que reconhece script Lua (reconhece a síntaxe) e acho visualmente agradável de escrever o código.

**3. METODOLOGIA**

**3.0.1 Introdução ao funcionamento do algoritmo**

O NEAT é um algoritmo genético baseado em evolução, mais especificamente na evolução das espécies de Charles Darwin (1872). A primeira grande diferença do NEAT para os algoritmos mais famosos de redes neurais de hoje como CNN’s e os diversos modelos utilizados com as bibliotecas da linguagem Phyton é que o NEAT não é um algoritmo de topologia fixa, e isso significa que os cérebros artificiais utilizados no algoritmo são produzidos pelo próprio algoritmo. A maioria dos modelos de redes neurais parte de um cérebro artificial pré definido por humanos, com entradas definidas (Input Layer), quantidade de neurônios na hidden layer, quantas camadas de Hidden layer (no caso do Deep learning) e quais são suas saídas (Output Layer), o que estes algoritmos fazem é simplesmente atualizar os pesos através de diferentes algoritmos, sendo o mais comum e utilizado o Backpropagation, calculado através da função derivada da função utilizada na ativação dos neurônios, realimentando a rede neural com este erro calculado pela função derivada até que o erro seja mínimo. Esta é a maneira mais convencional de treinamento de redes neurais, o NEAT supera diversos destes modelos. Como? O algoritmo parte da menor estrutura possível, e isto significa justamente que ele parte de uma topologia onde sequer há uma Hidden layer, onde teoricamente então sequer temos um cérebro, o que não deixa de ser verdade, então como exatamente o NEAT supera os modelos de topologia fixa? A resposta é: Evolução das espécies.

**3.1 A codificação da estrutura de um genoma**

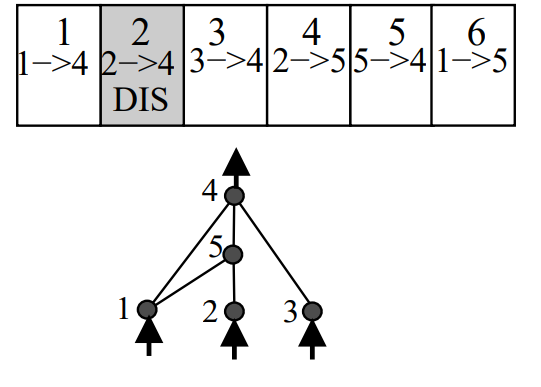
Em biologia, o genoma é toda a informação hereditária de um organismo que está codificada em seu DNA, ou seja, o conjunto de todos os genes de um ser vivo**.**  Ao codifica-lo, ele não é diferente, mas o que é um gene especificamente em um algoritmo?

Na biologia, os genes controlam não só a estrutura e as funções metabólicas das células, mas também todo o organismo. Quando localizados em células reprodutivas, eles passam sua informação para a próxima geração. E é isso que são os genes dentro do algoritmo, a menor unidade possível programada, responsável por todo o controle do sistema. O gene é na verdade dentro do algoritmo uma estrutura que contém 5 valores, os chamados: In, Out, Innovation Number, Enabled/Disabled e Weight. Esses valores definem o que cada gene é dentro do algoritmo, os genes são basicamente as conexões criadas entre os neurônios, ou simplesmente entre a entrada e a saída:

* In: O “In” é o valor que define de onde este gene está vindo, se é um dos pixels de entrada da imagem 12x12, ou se ele já vem de um outro neurônio gerado pelo algoritmo na Hidden Layer.
* Out: O “Out” é o valor que define para onde este gene está indo, se é uma das 8 saídas possíveis, que são Acelerar, Frear, Item, Cima, Baixo, Esquerda, Direita, L e R ou se é um outro neurônio produzido pelo algoritmo na Hidden Layer.
* Innovation Number: O “Innovation Number” é resumidamente o nome dado ao gene dentro da história onde ele pertence, dessa maneira é possível marcar os genes e ter um controle sobre as estruturas para aplicar o algoritmo de especiação, isso será aprofundado nos próximos tópicos.
* Enabled/Disabled: O “Enabled/Disabled” é a informação que nos diz se este gene está ativado ou desativado dentro da estrutura do genoma a que pertence, que é algo que ocorre durante as mutações, que será também aprofundado nos próximos tópicos.
* Weight: O “Weight”, é literalmente o peso carregado dentro deste gene, o valor carregado de uma entrada a uma saída, são eles quem definem o momento de ativar ou não um neurônio.

Pode parecer algo superficial neste momento, mas é extremamente importante entender o que é um gene para que seja possível o entendimento do algoritmo como um todo, mesmo com as lacunas e dúvidas que aparecem agora. Um gene pode ser visto da seguinte maneira:

**Figura 6: Estrutura básica de um genoma no NEAT.**



Fonte: Evolutionary Computation Volume 10, Number 2 página 107.

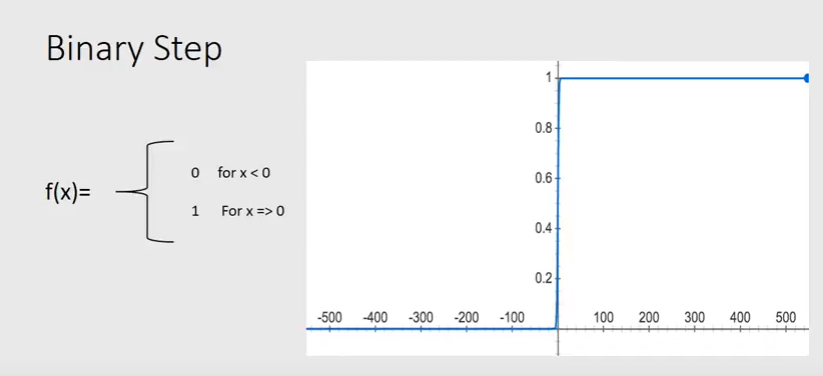
Os círculos escuros são chamados de neurônios, e essa é a estrutura básica de um genoma, que possui 6 genes neste exemplo. No primeiro bloco temos um “1” no topo, este é o Innovation Number, ou seja, o nome do gene que liga a entrada 1 para a saída 4 (1->4). Já no segundo bloco vemos a presença do termo “DIS” dentro do bloco, que basicamente nos diz que este gene não está ativo (Disabled) na estrutura do genoma atual, pois como podemos ver, da entrada 2 até a saída 4 existe um neurônio que impede esta ligação, mas vemos que no bloco 5 este neurônio está ligado a saída 4 normalmente, o que acontece é que ao longo da evolução desta topologia um neurônio foi adicionado, mas é um erro remover o gene Innovation Number 2, que ligava a entrada 2 a saída 4, pois ao manter este gene é possível verificar os genomas que possuem este ancestral em comum nas gerações futuras, que é algo essencial para o cálculo que separa os genomas em espécies.

**3.1.1 Neurônios e função de ativação**

No tópico anterior os neurônios apareceram, mas o que são estes neurônios exatamente? Os neurônios são o local onde os genes se encontram, carregando seus pesos, afinal, não tem sentido simplesmente carregar os pesos se nada for feito com isto. Um neurônio é o lugar onde e decidido de maneira binaria se algo deve ser feito ou não, neste caso, o neurônio ativa ou não, dizendo se a informação passada para o próximo é ativação ou não, se deve acelerar ou não, se deve ir para esquerda ou não etc. Dentro de um neurônio ocorre o que se chama função de ativação, que simula a ativação de um neurônio real, através de funções matemáticas. Para a função matemática diversas funções podem ser utilizadas, no caso das redes Feed Forward é preferível uma função de ativação que possua uma boa área acima do eixo X em sua derivada, pois calcular isto é o chamado Backpropagation, que é justamente analisar a taxa de mudança fora dos extremos, onde é de fato binário (Ativa ou não ativa).

Para os algoritmos genéticos normalmente é utilizado a função sigmoide, que também é chamada de logística, pois ela representa a ativação de um neurônio real, ao passo de que funciona como uma função Binary Step.

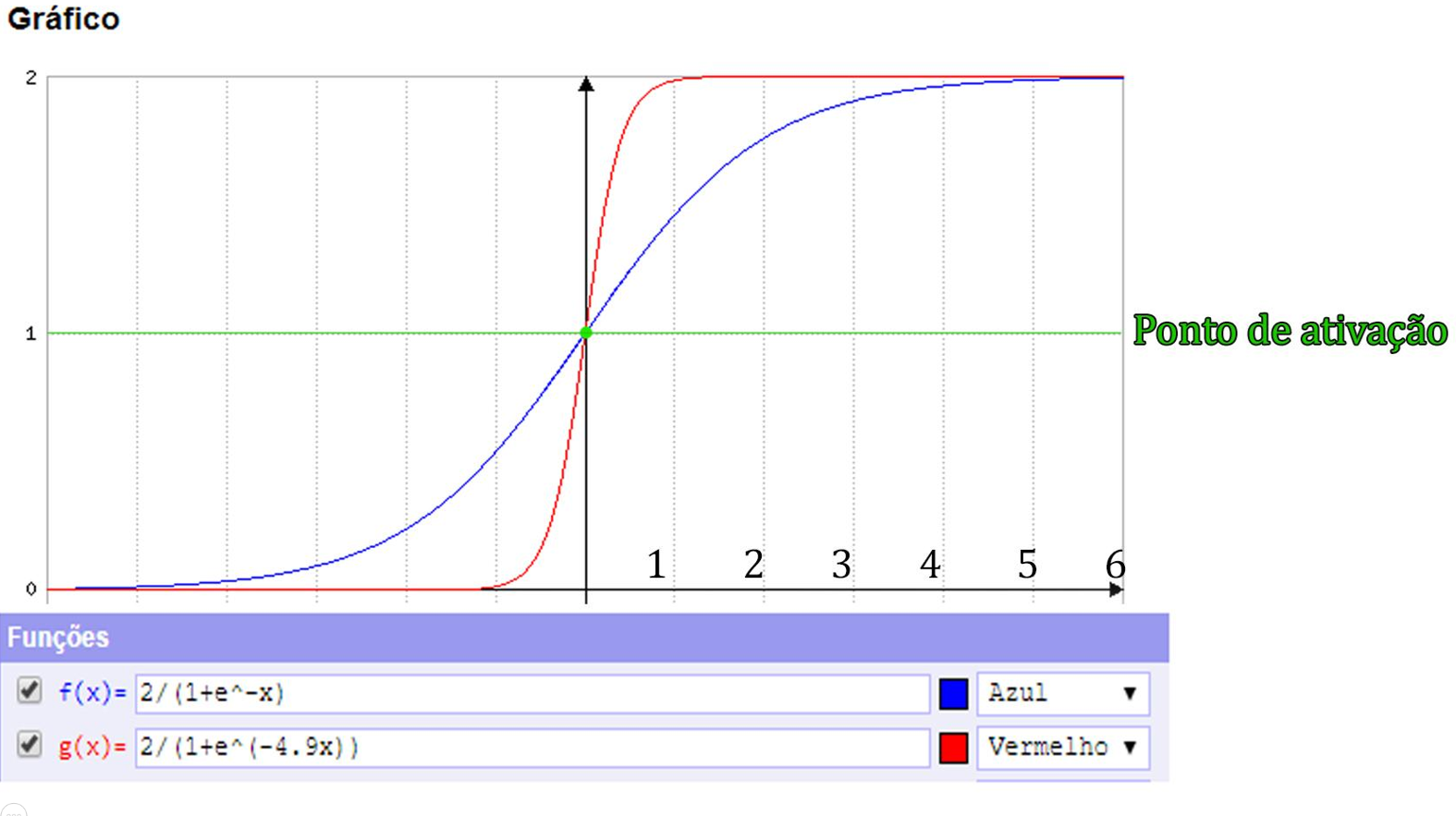
**Figura 7: Função de ativação Binary Step.**



Fonte: Deval Shah, Towards Data Science, 2017.

A função de ativação Binary Step é simples, ao ser a função que decide a ativação de um neurônio, para pesos acima de 0 (eixo X) ela é 1 (eixo Y), e para pesos abaixo de zero ela é zero, porém este modelo não é muito utilizado pelo motivo de ser uma transformada linear, muito brusca, no valor de ativação (zero), e por isso ela é considerada um modelo de regressão linear (a relação linear entre a variável dependente e uma variável independente). Para resolver isto é utilizada a função sigmóide:

**Figura 8: Função de ativação Sigmoide, original e variante.**

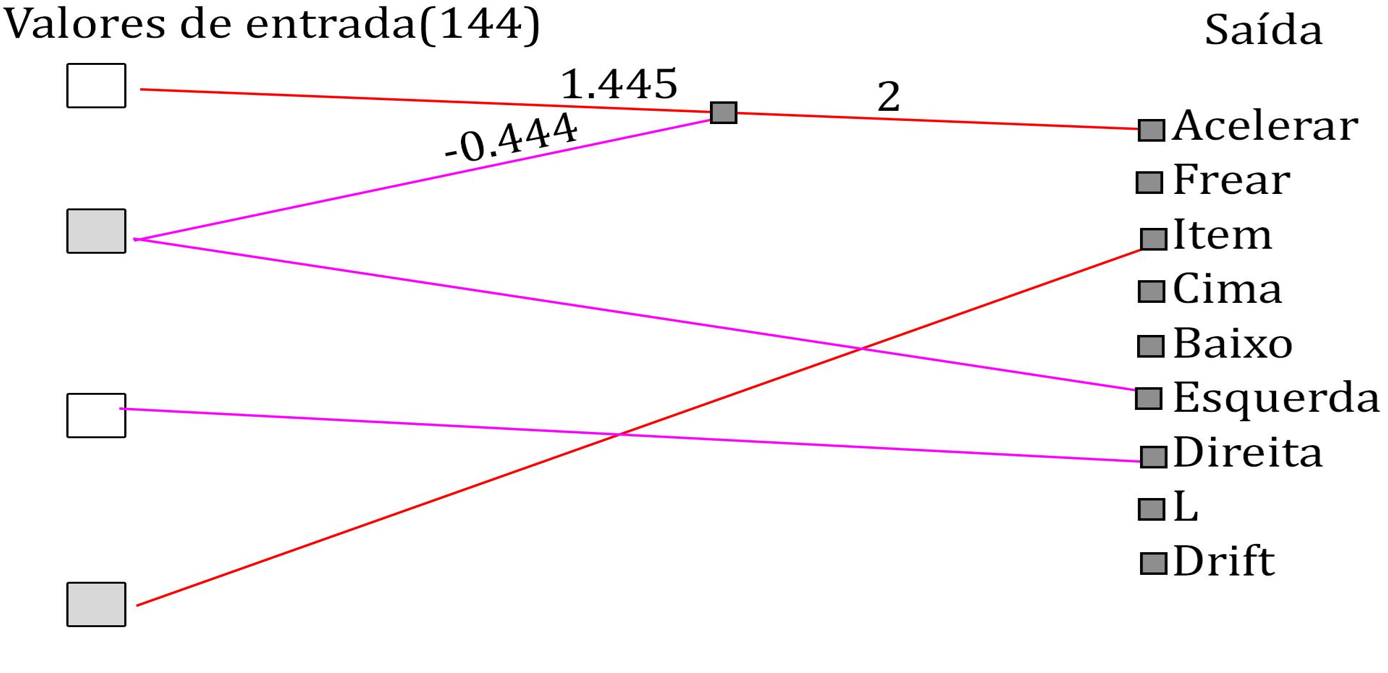


Fonte: O autor (2019).

* A função em azul é uma sigmoide comum: para todo X real.
* A função em vermelho é uma sigmoide ajustada para uma melhor ativação nos extremos, que tende a jogar os valores para a ativação máxima 2 para valores de X maiores que 0, e para a ativação mínima 0 para X menores que 0:

Ambas estão ajustadas para terem ativação máxima em 2 e mínima em 0, pois desta maneira é trabalhado somente com números positivos na ativação. A sigmoide resolve o problema de regressão linear, pois é uma curva exponencial em sua ativação, dando o elemento de não-linearidade. A sigmoide vermelha é proposta por Kenneth Stanley, no seu artigo sobre o NEAT, onde ele mostra seus benchmarks utilizando este modelo, e não uma sigmoide comum. No Smart Kart é utilizada a sigmoide vermelha (g(x)), e como funciona essa ativação é o que veremos agora:

**Figura 9: Funcionamento dos pesos em um neurônio**



Fonte: O autor (2019).

Este é um modelo para explicar que montei baseado no algoritmo real do Smart Kart, as linhas em vermelho são genes que estão carregando pesos maiores que 0 enquanto as roxas carregam valores menores que 0. Os pequenos quadrados em cinza escuro são os neurônios, e toda saída possui um neurônio antes de passar o valor para o registrador do botão especifico que está sendo ativado. Para neurônios que recebem um único gene é simples, o peso deste gene é o valor que entra como X na sigmoide, enquanto para neurônios que recebem mais de um gene, é feito um somatório dos pesos, e este somatório é o X da sigmoide. O X da sigmoide é então dado por:

No caso do exemplo temos 1.445 – 0.444 = 1.001, se olharmos na figura 8, vemos que 1 em x já é a ativação máxima, que é 2 em y, o que passa ao algoritmo uma ativação no botão de acelerar, o que o fará começar o movimento. Lembrando que esses pesos são lidos pelos pixels da pista em tempo real.

Essa é a ideia central dos conceitos do algoritmo, os próximos tópicos se referem a evolução do algoritmo, e como é possível alcançar os resultados.

**3.2 Iniciando uma população**

Ao iniciar uma população de genomas, cria-se uma estrutura de dados de um genoma, dentro dessa estrutura existe a estrutura de dados dos genes, e para que o algoritmo aprenda a conduzir o veículo completamente do zero, a primeira população é uma série de 200 genomas que possuem neurônios e ligações completamente aleatórias, de qualquer uma das 144 entradas (imagem 12x12) para qualquer uma das saídas, com um peso também aleatório. Todos estes genomas começam com suas estruturas vazias, e a maneira como recebem estes genes e neurônios aleatoriamente é trabalho das mutações genéticas, que são responsáveis tanto pelo início da geração de uma população de genomas, quanto para a criação de cérebros artificiais que evoluem em direção a uma solução otimizada.

**3.3 Mutações genéticas**

As mutações são o primeiro fator a montar o algoritmo de neuroevolução de fato, pois este é um dos motores do NEAT utilizados ao invés de Backpropagation. Dentro dos algoritmos genéticos existem diversos tipos de mutação, e nada nos impede de criar novas mutações também. Para o Smart Kart utilizo 3 modelos de mutação, e os próximos tópicos são a respeito destes modelos.

**3.3.1 Mutação por pesos**

A mutação nos pesos é o primeiro modelo, e por tratar-se da mutação nos pesos de uma rede neural estocástica (pesos indeterminados com origem aleatória) existe um termo presente no cálculo da mutação que é necessário entender, que se chama Perturbação do peso. Para isso utilizarei um trecho que explica, dos autores Yeming Wen, Paul Vicol, Jimmy Ba, da universidade de Toronto, que escreveram o paper FLIPOUT: EFFICIENT PSEUDO-INDEPENDENT WEIGHT PERTURBATIONS ON MINI-BATCHES:

We use the term “weight perturbation” to refer to a class of methods which sample the weights of a neural network stochastically at training time. More precisely, let f(x, W) denote the output of a network with weights W on input x. The weights are sampled from a distribution qθ parameterizedby θ. We aim to minimize the expected loss E(x,y)∼D,W∼qθ [L(f(x, W), y)], where L is a loss function, and D denotes the data distribution. The distribution qθ can often be described in terms of perturbations: W = W + ∆W, where W are the mean weights (typically represented explicitly as part of θ) and ∆W is a stochastic perturbation.

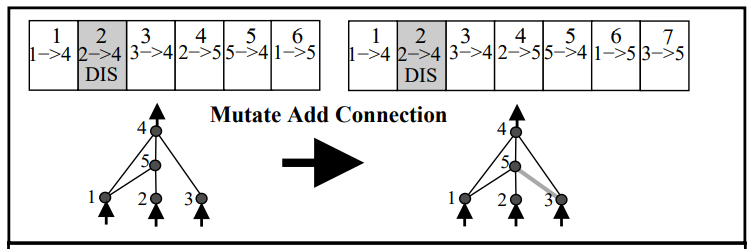
Por se tratar de pesos aleatórios e imprevisíveis, é incluído no cálculo a perturbação do peso, que nada mais é do que um multiplicador para minimizar a perda esperada. O cálculo feito é o seguinte:

Isto é, o peso mutado é o peso atual somado ao cálculo dentro dos colchetes, que é um valor aleatório entre -1 e 1, que define o sinal do valor a ser mutado, e um valor aleatório entre 0 e 1 que é multiplicado pelo aleatório anterior do sinal, no fim isto tudo é multiplicado por 0.9, que é a perturbação do peso, valor retirado do artigo original do NEAT, na sessão Parameter Settings. A chance desta mutação ocorrer é de 25%, o que nos diz que teoricamente ¼ da população irá receber mutação nos pesos de sua topologia, ou que em 4 gerações uma topologia receberá uma alteração nos pesos.

**3.3.2 Mutação por conexões**

A mutação por conexões é a mais simples e é a que permite que as topologias cresçam aos poucos. O que ela faz é basicamente criar um novo gene, ou seja, uma nova conexão aleatória entre dois pontos que não estão ligados na topologia atual, pegando uma entrada aleatória e fazendo a conexão entre ela e um outro neurônio, podendo ser este neurônio um dos de saída ou algum criado na hidden layer pelo próprio algoritmo, que é a mutação do próximo tópico. A mutação dentro do algoritmo incrementa a variável de Innovation number, dando então a esta nova conexão um novo nome, e adicionando o novo gene dentro da topologia que recebeu a mutação, a chance desta mutação ocorrer é de 100%, garantindo que todo genoma receberá ao menos uma mutação, veja a figura 10:

**Figura 10: Mutação por conexões topologicamente**

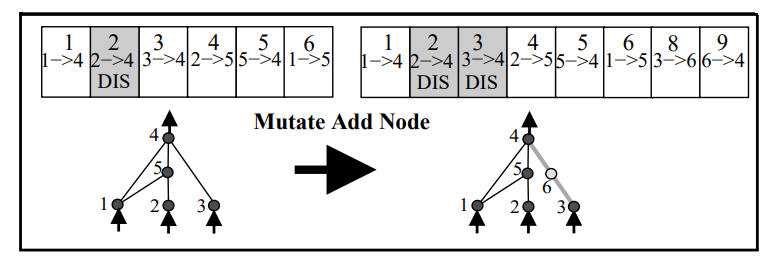


Fonte: Evolutionary Computation Volume 10, Number 2, página 107.

**3.3.3 Mutação por adição de neurônio**

Esta mutação adiciona um novo neurônio na topologia do genoma, então como deve estar imaginando, sim, essa é a função responsável pela criação e da hidden layer, e também de certa forma a responsável por adicionar o Deep learning a topologia, não é exatamente o Deep learning, pois o termo Deep Learning é especifico para redes onde há uma série de neurônios e todos eles estão interligados a algo tanto na sua entrada quanto na saída, mas é um Deep learning no sentido de processamento de informação dentro do cérebro artificial. A chance de ocorrer é 40% e seu funcionamento é o seguinte:

**Figura 11: Mutação por adição de neurônio topologicamente**

****

Fonte: Evolutionary Computation Volume 10, Number 2, página 107.

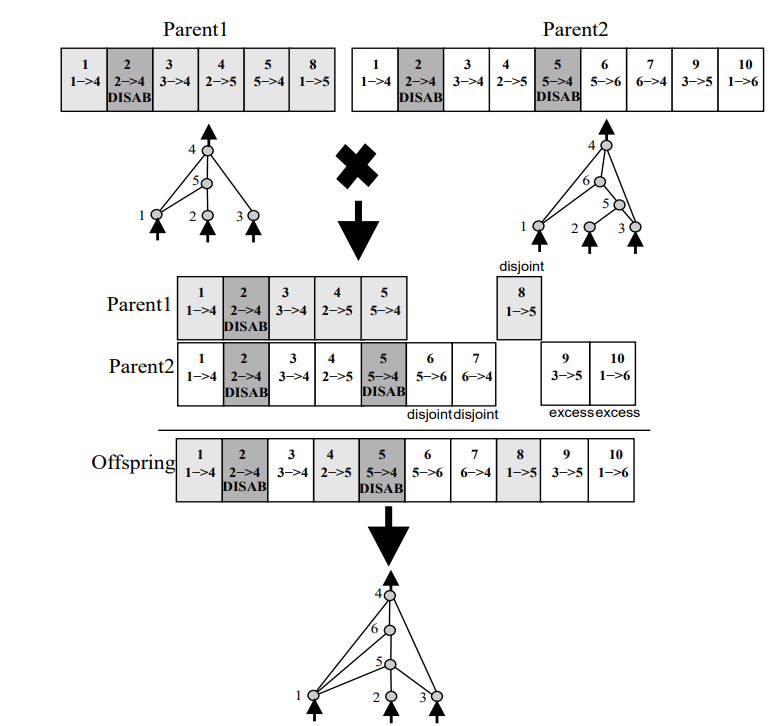
O que ocorre é que quando esta mutação acontece, uma conexão é escolhida aleatoriamente para ser separada, na imagem vemos que o Innovation Number 3, o gene que liga a entrada 3 para a saída 4 está marcado como disabled após a mutação, pois é nessa conexão que um novo neurônio foi adicionado. Como resultado disso, dois novos genes são criados, o 8 e o 9, ligando a antiga entrada 3 para o nosso neurônio, e esse neurônio para a saída 4. Parece algo sem utilidade, pois tínhamos uma conexão indo de 3 para 4 e agora temos uma conexão indo de 3 para 4 com um neurônio no caminho, mas o ponto é que a partir da criação desse neurônio, mais de um gene pode ser o valor de entrada deste neurônio, e este mesmo pode entregar sua ativação para um outro neurônio, podendo ser este de saída ou um outro criado pela mutação.

**3.4 Crossover e Especiação**

O Crossover é o motor principal de um algoritmo genético, sua função é utilizar topologias que nos mostram bom desempenho e cruza-las para criar uma nova para a próxima geração de topologias, isto é, cérebros artificiais, desta maneira temos a evolução. O crossover de um algoritmo genético é normalmente aplicado única e exclusivamente aos melhores genomas no cálculo de desempenho, que varia de aplicação a aplicação, no caso do Smart Kart, é um cálculo simples de distância alcançada em relação ao tempo. É neste momento que o NEAT começa a propor boas soluções para aumentar a eficiência da evolução, pois no NEAT o crossover é aplicado em uma área muito maior de genomas, e o motivo de funcionar se chama Especiação. É extremamente complicado de entender um sem o outro, por este motivo veremos os dois tópicos simultaneamente a seguir.

**­**

**Figura 12: O crossover de duas topologias**

****

Fonte: Evolutionary Computation Volume 10, Number 2, página 109.

Na figura 12 é feito o cruzamento de duas topologias, um Parent1 e um Parent2, que seriam algo como o pai e a mãe do genoma que será fruto do cruzamento. Na imagem vemos a presença de dois termos: Disjoint e Excess, e veremos agora o que eles são:

* Disjoint: O Disjoint, que é traduzido como disjunção, é justamente um gene que não está presente na topologia de ambos os genomas que irão realizar o crossover, marcado como um extra, mas que historicamente surgiu antes do gene mais novo da topologia que não o possui, ou seja, seu Innovation Number é menor do que o maior Innovation Number na topologia onde não há esse gene.
* Excess: O Excess é justamente o gene em excesso ao comparar duas topologias, um gene excess é um gene que possui Innovation Number ainda maior que o maior gene presente na menor topologia, na figura 12, o maior Innovation Number do Parent1 (que é a menor topologia em questão) é o Innovation Number 8, logo todos os genes acima deste serão marcados como excesso.

Assim é feito o cruzamento de topologias, os genes que carregam o disabled permanecem com disabled no gene em especifico do herdeiro, quanto aos genes disjoint e excess, ambos são mantidos na topologia do herdeiro.

Mas para que servem os genes disjoint e excess? Isso é o que veremos no próximo tópico.

**3.4.1 Como é feita a Especiação?**

O que é a especiação afinal? A especiação é um algoritmo que segrega os genomas dentro de espécies, desta maneira eles competem dentro da sua própria espécie, e não comparado a uma população global. Isto nos dá um enorme aumento de performance, pois ao invés de utilizar um único modelo que está dominando dentro da população global de genomas, temos diversas espécies diferentes evoluindo e mostrando maneiras diferentes de se alcançar a solução do problema. Como o cálculo de desempenho é feito por distância por tempo, um genoma que somente acelera é considerado muito melhor do que um genoma que sabe diferenciar perfeitamente direita de esquerda, e este genoma é muito mais fácil de evoluir para a solução do que o genoma que somente acelera, mas em um algoritmo genético genérico nós desconsideramos esta topologia e damos toda estrutura ao genoma que somente acelera de evoluir. É um exemplo mais extremo, mas o ponto é que ao complexificar as topologias, é extremamente difícil de dizer qual delas é a mais próxima da solução, de conduzir um veículo com maestria, e é um grande desperdício jogar diversas formas diferentes apresentadas por um algoritmo genético de solucionar o problema.

A especiação já havia sido estudada dentro dos algoritmos genéticos antes do NEAT, mas nunca utilizada em neuroevolução, mas era comumente utilizada em *otimização de função multimodal* (Mahfoud, 1995), onde funções possuem mais de uma solução, e os algoritmos genéticos utilizando especiação encontravam estas soluções. Também foi utilizada em *cooperative coevolution of modular systems of multiple solutions* (Darwen e Yao, 1996; Potter e De Jong, 1995).

A ideia de especiar é ótima, mas como aplica-la que se torna um grande problema, e é por isso que ela não foi implementada diretamente no algoritmo que é a base do NEAT, o TWEANN’s (Topology and Weight Evolving Artificial Neural Network), que se tornou um modelo de algoritmo, no qual o NEAT está inserido. O NEAT apresenta o cálculo da distância de compatibilidade (δ), resolvendo o problema, que era justamente criar uma equação capaz que classificar o quão semelhante são duas estruturas dentro da população. O cálculo é o seguinte:

.

Onde:

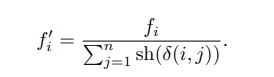
* E: Excess;
* D: Disjoint;
* W: Diferença de pesos;
* c1,c2,c3: Coeficientes determinantes do peso de cada fator dentro da equação;
* N: Número de genes da topologia.

Essa equação é criada a partir de fatores que são chamados de marcadores históricos, que são justamente o Excess e o Disjoint. Através destes marcadores é possível comparar dois genomas e ver o quão próximos eles são através dos ancestrais que compartilham, pois quanto mais excessos e disjunções houverem entre dois genomas, menor é o passado evolucionário que compartilham, logo menos compatíveis ele são a entrarem dentro de uma mesma espécie. A diferença dos pesos é um cálculo que pega o módulo da diferença do pesos do gene que está presente em ambas topologias, para vermos o quão diferente é o peso no gene em questão. Se os genomas tiverem menos de 20 genes, o N possui valor 1, dessa maneira os fatores possuem seu valor bruto e a compatibilidade de topologias pequenas se torna mais precisa.

Com isto é possível especiar os genomas e cruza-los com o crossover. Com este algoritmo descrito até aqui já é possível aplicar uma boa neuroevolução com o algoritmo genético, mas ainda existe um problema: As espécies com maior desempenho irão dominar rapidamente dentro da população global, mesmo que um único genoma dentro da espécie esteja performando bem.

**3.4.2 Cálculo do compartilhamento de desempenho**

Para que uma espécie não evolua demais dentro da população global o NEAT utiliza *explicit fitness sharing* (Goldberg e Richardson, 1987), e a ideia é que o tamanho das espécies sejam controlados mesmo quemuitos genomas dentro de uma espécie estejam performando bem. Para isso é necessário ajustar o desempenho original para um compartilhado, o cálculo é o seguinte:



Onde é calculado o desempenho ajustado para um genoma de acordo com a sua distância para qualquer outro organismo dentro da espécie. O é 0 quando a distância de um genoma para é maior que o threshold (, calculado justamente na função da distância de compatibilidade, ou 1 para quando o threshold não foi alcançado.

Dessa maneira, a cada espécie é atribuido um número diferente de herdeiros possíveis, calculados pelo , proporcionalmente ao desempenho ajustado. O é calculado através da soma do desempenho dos membros da espécie, dividido pelo número de membros da espécie.

A reprodução da espécie é feita removendo os genomas com menor desempenho da população, e inserindo os novos genomas através do crossover no lugar destes que foram removidos, novos genomas estes criados pelos genomas restantes dentro da espécie (que não foram removidos pela seleção natural).

**3.5 Parâmetros para a população**

Para a primeira população o sistema do NEAT não é exatamente implementado. O que é implementado é na verdade um sistema mais simples e genérico do TWEANN’s, onde as primeiras estruturas são aleatórias, semelhante ao Initial Random NEAT, e por isso existe um certo problema que é descrito por K. Stanley e R.Miikkulainen em sua publicação, onde alguns genomas não possuem nenhum caminho de entrada para uma saída, e esses genomas seguem as regras globais, logo demoram um pouco até saírem da população, tomando tempo do algoritmo. Mas o algoritmo segue o NEAT no sentido de que todas essas estruturas partem do mínimo, sem hidden nodes (a hidden layer), e ligando diretamente algumas entradas em algumas saídas. Visto que a mutação é “lenta”, no sentido de que adiciona uma única conexão ao genoma por mutação, é possível tentar otimizar mesmo as topologias que possuem o problema acima.

Dessa maneira, as conexões e os neurônios presentes nas topologias dos genomas são todos justificados pelo cálculo de desempenho, apesar de que inicialmente não são justificados, essas topologias que não demonstram desempenho eventualmente são eliminadas da população, mesmo que tome um pouco de tempo, em favor de partir de estruturas mínimas e aleatórias que podem ser otimizadas. Para isso funcionar bem dois algoritmos foram incluídos por mim, que são os próximos tópicos.

**3.5.1 Early Interspecies Crossover**

O foco do Smart Kart é que o algoritmo seja capaz de aprender conduzir um veículo em si, e não que se torne um algoritmo de aprendizado de pistas especificas, que é o que ele faz neste momento.

O primeiro problema que aparece na prática da aplicação deste algoritmo é que inicialmente todas topologias aleatórias são vistas como únicas pelo cálculo da distância de compatibilidade, ou seja, em uma população de 100 genomas teremos 100 espécies diferentes, e como no NEAT os genomas competem dentro da espécie, o que temos são 100 genomas competindo com eles mesmos. Dessa maneira, a neuroevolução é sim possível, mas muito mais lenta do que pode ser, pois um genoma que aprende a acelerar e deslocar para a direita é classificado como diferente do genoma que sabe acelerar e deslocar para a esquerda. O que ocorre nesse caso é que ambos terão que aprender a deslocar-se na outra direção, tomando as vezes 6 ou 7 gerações para que isso aconteça, enquanto seria muito mais rápido cruza-los para a criação de um genoma que consegue deslocar-se para ambas direções. Para que isso seja melhorado, inicialmente existe o Interspecies Crossover, que nada mais é do que cruzar genomas de diferentes espécies, algo que está presente no NEAT, mas com uma chance de 0.001, o que beira o impossível. No Smart Kart as topologias são cruzadas inicialmente baseado única e exclusivamente pelo seu desempenho, pois os genomas que aceleram e deslocam em alguma das direções tem experimentalmente um desempenho maior do que os que apenas aceleram. Com isso é possível a criação de uma topologia simples que possui a base de uma rede neural artificial para a condução de veículos, não se trata de um excelente condutor, mas de uma boa topologia que pode finalmente ser a base para as próximas gerações. A partir do momento em que essa topologia é alcançada, o Early Interspecies Crossover é desligado e o algoritmo segue normalmente com a especiação.

**3.5.2 Overpopulation**

O algoritmo de neuroevolução de topologias aumentadas tende a complexificar as redes neurais para a pista em que está sendo feito o treinamento, isso significa que com o tempo as topologias que começaram pequenas serão grandes e complexas para otimizar mais e mais a solução do problema, concluindo a pista em questão cada vez mais rápido, mas utilizando topologias extremamente especificas para essa otimização. Para entender isso sempre utilizo o Princípio da Incompatibilidade (Zadeh, 1973):

Principle of incompatibilty

Stated informally, the essence of this principle is that as the complexibility of a system increases, our ability to make precise and yet significant statements about its behavior diminishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics.

Princípio da incompatibilidade

Conforme a complexidade de um sistema aumenta, a nossa habilidade em fazer afirmações precisas e ainda significativas sobre o seu comportamento diminui, até que um limiar é atingido além do qual, precisão e significado (ou relevância) se tornam quase características mutuamente exclusivas. (Lotfali Askar Zadeh, 1973, tradução nossa).

Esse princípio é exatamente o motivo de existir o algoritmo de Overpopulation. Ao longo da pesquisa poucas topologias eram capazes de dirigir em pistas diferentes com uma boa resposta (dirigir dentro da pista, e não fora dela), e isso se deve ao motivo de que se tornam especificas demais para a pista de treinamento, ao ponto de que se tornam complexas demais para serem evoluídas, e principalmente, complexas demais para fazer mudanças de otimização.

A solução vem com os algoritmos de Overpopulation e Early Interspecies Crossover. Primeiramente o algoritmo monta topologias simples para a condução de veículos através do crossover entre espécies, e em seguida sua missão é otimizar está topologia simples para que seja possível conduzir o veículo com certa habilidade em pistas diferentes, e para fazer essa otimização o Smart Kart utiliza o Overpopulation:

**Figura 13: Gráfico do funcionamento do Overpopulation**

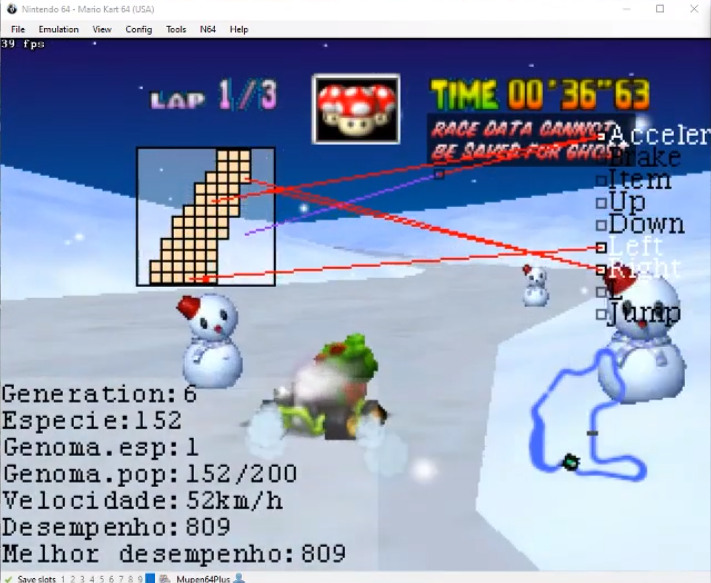
Fonte: O autor (2019).

O objetivo do Overpopulation é fazer com que as topologias que foram criadas do zero e adquiriram certa experiência em uma outra pista se tornem capazes de otimizar esta topologia para a nova pista onde serão treinadas.

Ao simplesmente inserir os dados e topologias adquiridos no treinamento anterior o que acontece é que a maioria das topologias não tem a oportunidade de consertar suas topologias a tempo, pois em 5 gerações uma espécie que não demonstra avanços no desempenho compartilhado é simplesmente eliminada. O problema que surge é que as topologias passam a complexificar a solução dentro de 5 gerações, as topologias que eram simples, agora são complexas e se torna cada vez mais difícil de otimiza-las para a nova pista. O que existe na verdade é uma inércia a ser quebrada, de maneira que o algoritmo precisa de uma força para simplesmente corrigir as topologias simples com mais uma conexão, ou as vezes simplesmente uma correção no peso. Isto é facilmente resolvido com o Backpropagation com as redes neurais pré-definidas, mas o Smart Kart está criando a sua topologia, então o Overpopulation da essa força a ele. No exemplo da Figura 13, trata-se do gráfico do melhor resultado apresentado pelo Smart Kart, onde todos os algoritmos descritos até aqui foram aplicados. Uma boa topologia foi criada, muito simples e ótima para ser otimizada em outras pistas. O treinamento foi feito em 1 hora e 20 minutos, e é possível ver o treino completo no Youtube, os links estarão disponíveis no final do relatório.

O Overpopulation da Figura 13 decresce em 20% por geração para estabilizar nas 200 novamente, mas caso 400 genomas na população (100% de crescimento) não sejam suficientes para a otimização da topologia para a nova pista, é possível colocar até 1000 genomas (400% de crescimento), mas o cálculo de decrescimento precisa ser reajustado. A curva entre o ponto que é pico do Overpopulation e o ponto em que volta a ser 200 é exponencial, mas pode ser linear, contanto que o pico se mantenha por duas gerações, dessa maneira temos uma boa precisão do crossover da primeira geração no pico do Overpopulation, pois se a curva for linear a seleção natural acaba sendo rígida demais com os genomas na sua descida (próximo a voltar ao tamanho original da população), o que pode resultar em uma geração de genomas com otimizações aleatórias que mostraram melhor resultado, mas não exatamente a melhor população possível, por isto sempre utilizo uma curva exponencial.

**Figura 14: Treinamento inicial do Smart Kart, Frappe Snowland.**



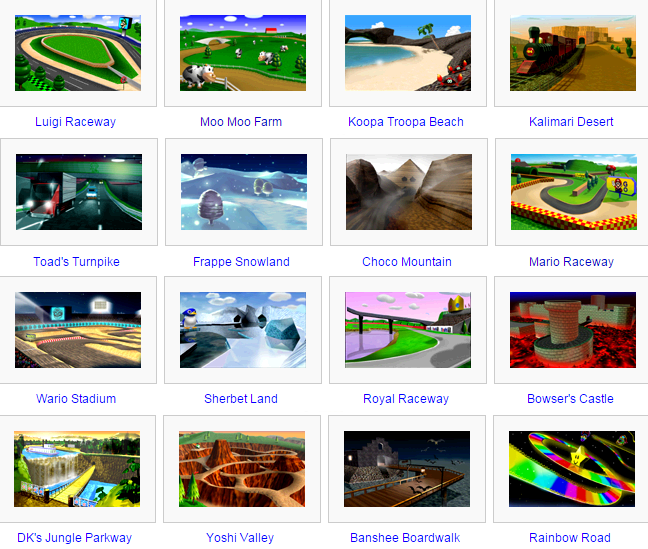
Fonte: O autor (2019).

A figura 14 é uma ilustração da topologia em questão que é desenvolvida do zero, possui somente um neurônio na hidden layer e suas outras conexões são diretamente de uma entrada para uma saída, e o genoma da imagem já á capaz de dar uma volta na pista em questão (Frappe Snowland).

**3.6 Mapeamento e pistas benchmark**

O mapeamento das pistas do jogo para as entradas da rede neural são feitos em tempo real. Para que isso aconteça é feita uma varredura da tela no eixo X e no eixo Z, onde os pixels da tela são lidos e dependendo do seu valor ele é marcado ou não como um bloco de pista. O que é possível de mapear em questão é aquilo que é a pista principal, o tipo principal de asfalto assim digamos. Trata-se de um mapeamento bastante simples e por este motivo os benchmarks são feitos em pistas que não possuem vários tipos de solo, já que para dar a entrada para a rede neural é feito uma binarização, onde “1” é pista e “0” qualquer outro tipo de solo ou objeto. O Mario Kart 64 conta com 16 pistas, 7 delas são possíveis de utilizar como um bom Benchmark, e outras 5 são mapeadas sem uma boa precisão de o que é o solo que o kart deve dirigir e o que não é, pelo motivo anterior.

**Figura 15: Pistas do Mario Kart 64**



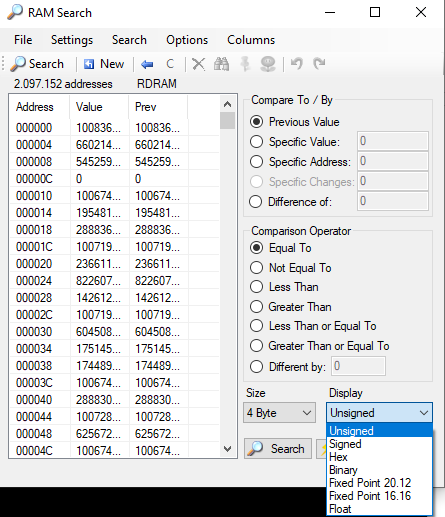
Fonte: Fantendo, 2015.

Os benchmarks principais são feitos em: Luigi Raceway, Moo Farm, Kalamari Desert, Frappe Snowland, Choco Mountain, Mario Raceway e Rainbow Road, que são pistas que são de 95 a 100% constituídas de um único tipo de solo na textura principal. Já Koopa Troopa Beach, Royal Raceway, Wario Stadium, DK’s Jungle Parkway e Toad’s Turnpike possuem diferentes problemas que impedem uma boa execução do mapeamento, como texturas de boosters (solos que aceleram o kart) onde seria o asfalto e também outros tipos de textura que não são mapeados.

**3.7 Leitura e escrita de dados na memória (registradores)**

O Mario Kart 64 possui 8,388,608 endereços de memória, o Bizhawk nos permite olhar para isto com 1byte (8,388,608 endereços), 2bytes (4,194,304 endereços) e com 4bytes (2,097,152 endereços), e também disponibiliza funções para que seja possível filtra-los.

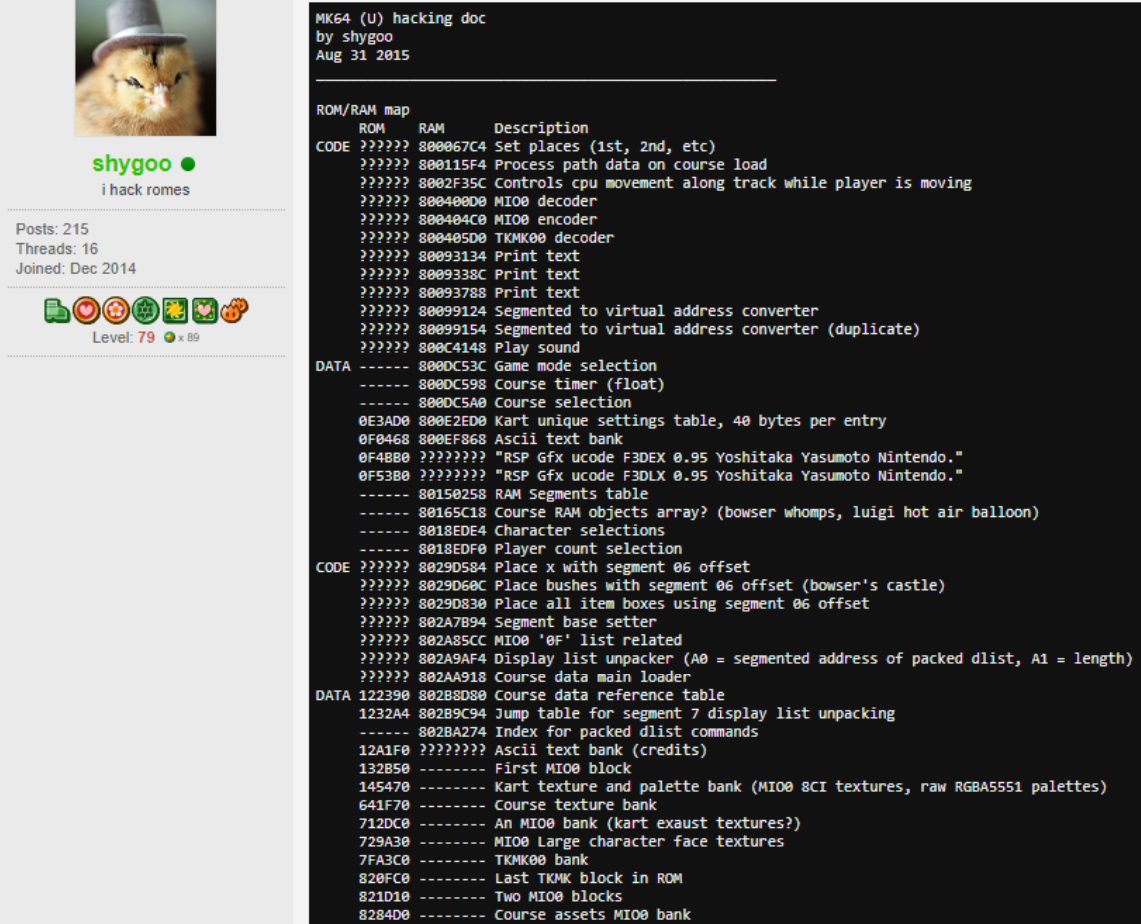
**Figura 16: Bizhawk RAM Search**



Fonte: O autor (2019).

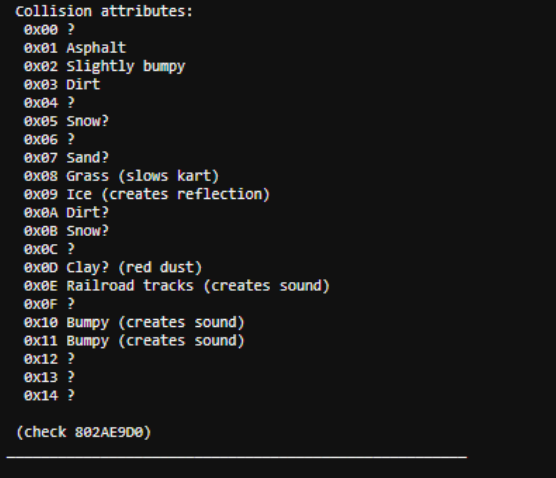
Nestes endereços (registradores) são armazenados todos os dados de Mario Kart 64, desde atributos como velocidade (em vetores), item, personagem, pista selecionada, até endereços que guardam simplesmente os tipos de textura a serem carregados no jogo. Encontrar os endereços que guardam valores dinâmicos não é uma tarefa difícil, pois em alguns minutos de teste é possível reduzir estes 2 milhões para 40 endereços com os filtros e testar um a um, mas é uma tarefa extremamente difícil para registradores com valores estáticos. A maioria dos endereços utilizados no Smart Kart foram encontrados previamente por um usuário na internet chamado Shygoo, que faz a engenharia reversa de Mario Kart 64 desde 2015, seu objetivo é criar uma extensão onde seja possível criar as próprias pistas dentro do jogo. Um registrador que é utilizado e não havia sido mapeado é o que guarda o valor do início da colisão de cada pista. O endereço não havia sido mapeado, porém Shygoo tinha uma ideia de onde ele estaria, eu então o encontrei testando os endereços que estavam dentro de um espaço mais curto determinado por ele. Com isso é possível mapear as pistas para a entrada da rede neural.

**Figura 17: Mapa dos registradores de Mario Kart 64**



Fonte: Shygoo, 2015.

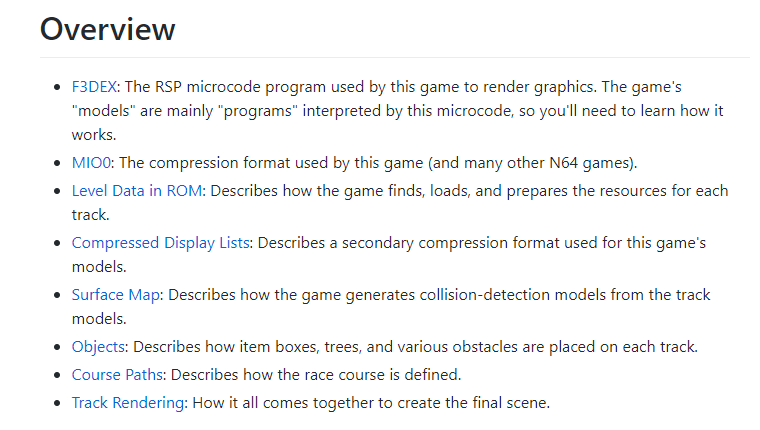
**Figura 18: Mapa de colisões nos registradores**



Fonte: Shygoo, 2015.

Muitos outros endereços foram mapeados por ele, mas as figuras 17 e 18 ilustram alguns deles para que possamos entender do que se trata. Um usuário do Github chamado RenaKunisaki também contribuiu bastante para minha pesquisa, também com endereços da memória de Mario Kart 64. Suas contribuições são muitas dentro da engenharia reversa de Mario Kart 64, e ele foi o primeiro a criar uma pista customizada dentro de Mario Kart 64, utilizando somente um Hex Editor em 2008, a Figura 19 mostra os tópicos onde ele trabalhou:

**Figura 19: RenaKunisaki Github**



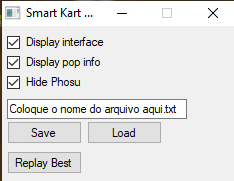
Fonte: RenaKunisaki, 2017.

RenaKunisaki trabalhou durante 3 anos com a engenharia reversa de Mario Kart 64. Graças a estes dois usuários que disponibilizaram todo seu trabalho no Github e YouTube, tive facilidade de acessar e escrever nos endereços necessários dentro do script em Lua, o que compensou a escolha da API no lugar de uma linguagem mais utilizada como o Phyton e de bibliotecas como o Keras e TensorFlow.

**3.8 Armazenamento dos dados da rede neural**

Os dados das redes neurais são salvos para que o treino possa continuar em outro momento, ou mesmo para que possamos ver os resultados novamente. Todos os dados de uma população são guardados dentro de arquivos txt ao fim da geração, desta maneira é criado um backup de todas gerações, e não é necessário carregar todos os dados desde a primeira geração, somente a da geração atual, que ela continuará a evoluir do ponto onde foi salva. O que é salvo em específico são as informações essenciais, então todos os genes de todos os genomas estão salvos dentro deste arquivos. Também algumas informações adicionais sobre o melhor genoma da população, como o número de genes em sua topologia e seu desempenho marcado como melhor desempenho global a ser superado pelas espécies. O algoritmo em si utilizado aqui é semelhante ao de Mari/o (Sethbling, 2015) e Mega IA X (Igor Haag e Gabriel Hoffmann, 2018), apenas adaptei as variáveis a serem salvas e carregadas dentro dos arquivos para as que estão no código do Smart Kart. Utilizando a biblioteca forms do Bizhawk para criar uma interação com o usuário, é possível escrever o nome do arquivo a ser salvo ou carregado, e também há um botão para salvar e outro para carregar dados.

**Figura 20: Form para controle do Smart Kart**

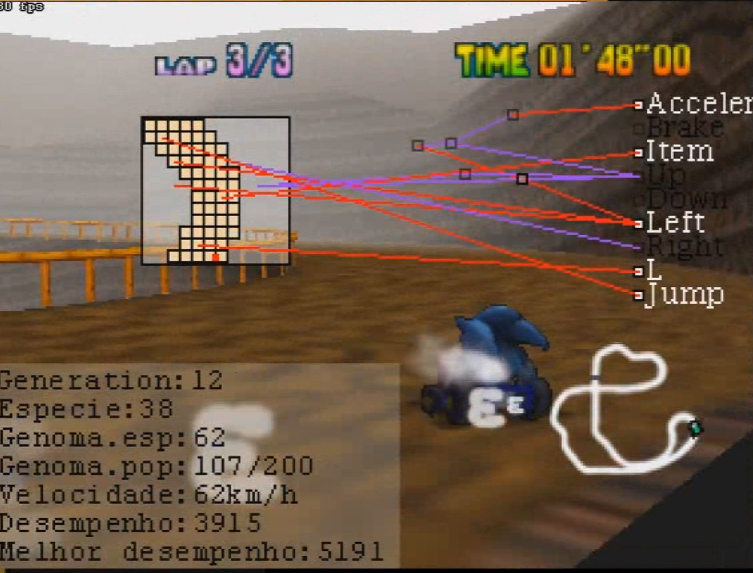


Fonte: O autor (2019).

**3.9 Interface gráfica**

A interface gráfica é feita utilizando a biblioteca GUI do Bizhawk. Com ela é possível criar pequenos polígonos, elipses, linhas e texto. É difícil de dizer se as coisas estão indo bem ou não sem ver o que de fato está acontecendo com a rede neural, então semelhante ao form, utilizei um modelo criado por Sethbling para mostrar o que está acontecendo por trás da rede neural, que é basicamente um algoritmo que tenta graficamente desenhar os neurônios e ligar as entradas até as saídas, criando linhas entre os valores. O algoritmo acaba criando uma interface um tanto difícil de ser observada e compreendida para grandes topologias, mas visto que é o melhor que o emulador disponibiliza, já é de grande ajuda, e funciona bem com topologias pequenas.

**Figura 21: Interface gráfica do Smart Kart**



Fonte: O autor (2019).

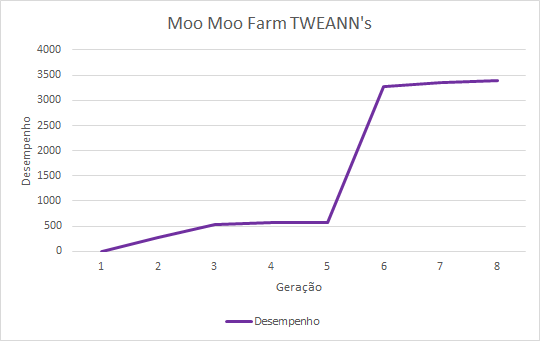
As linhas em vermelho representam pesos maiores que 0, e as roxas pesos menores que 0. Os quadrados em pastel são a representação da pista dentro da caixa com fundo transparente 12x12. Os quadrados com fundo branco nas saídas são neurônios ativados, os que estão escuros são neurônios não ativados. Os quadrados entre a caixa 12x12 e as saídas são os neurônios da Hidden Layer produzida pelo algoritmo, sendo os que estão com borda preta estão ativados, e os que estão com baixa opacidade desativados. No canto inferior esquerdo estão as informações da população: Geração, espécie, o número do genoma dentro da espécie, o número do genoma dentro da população global, sua velocidade (vetores x e z analisados), seu desempenho atual e o melhor desempenho já registrado.

**4. ANÁLISE DE DADOS**

**4.1 Curvas de aprendizado com inicialização TWEANN’s**

Muitos testes foram realizados dentro destas pistas, é possível encontrar com mais precisão e detalhamento no caderno de campo da pesquisa o que acontece de fato nestes testes. Seguem as curvas com as populações sendo inicializadas de maneira aleatória com a especiação desde a primeira geração:

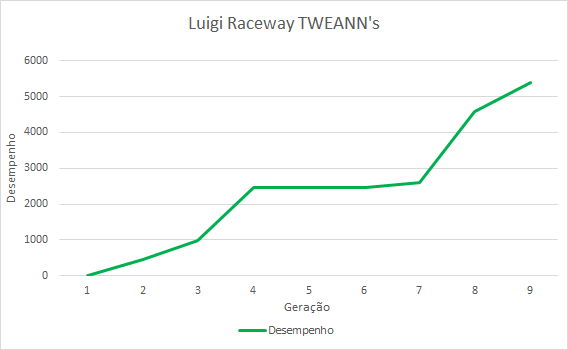
**Figura 22: Geração x Desempenho MMF TWEANN’s**



Fonte: O autor (2019).

Moo Moo Farm é uma pista oval e não existe outro tipo de solo a ser mapeado que não o principal, logo o kart sequer pode sair da pista, então é consideravelmente rápido criar topologias que concluem a pista mesmo que com um algoritmo genético simples. Essa é a única pista que foi concluída por um algoritmo genético padrão, cruzando os melhores da geração, sem especiação ou qualquer outro algoritmo.

**Figura 23: Geração x Desempenho LR TWEANN’s**



Fonte: O autor (2019).

Luigi Raceway é uma pista mais interessante, possui três tipos de solo e o Smart Kart facilmente pode sair do principal, como consequência, os genomas são penalizados em seu desempenho por dirigir fora da pista quando comparados aos que dirigem da maneira correta. Esta pista em especifico mostrou o grande potencial da neuroevolução com algoritmos genéticos, pois foi concluída em 1 minuto 55 segundos e 64 centésimos, um ótimo tempo, na verdade 2 segundos mais rápido do que meu melhor resultado, e bem melhor do que a maioria dos jogadores. Ao mesmo tempo que um algoritmo genético de evolução não faz aquilo que nos é óbvio, ele também mostra soluções que não são óbvias a nós humanos. O Smart Kart aprendeu a dirigir de uma maneira um tanto quanto inusitada nesta pista, pois após alcançar ótimos resultados, o tempo ainda continuou a ser melhorado. O que acontece é que há cerca de 4 anos atrás foi descoberto o que é chamado hoje de método de execução de manobras perfeitas dentro do Mario Kart 64, e o Smart Kart dirige de uma maneira semelhante. O Mario Kart 64 possui um recurso onde o kart é desacelerado baseado no seu ângulo nas curvas (registradores de seno e cosseno na memória), ou seja, consideremos o kart de frente 90°, quando ele atinge cerca de 150° o kart é desacelerado, pois a velocidade é calculada na memória em três vetores, e esse ângulo desacelera o vetor da velocidade do eixo X, porém existe uma maneira de não ser desacelerado nas curvas, e consiste em virar o kart até um pouco antes desse limite rapidamente e ir para a direção oposta, dessa maneira os vetores se anulam e o kart volta a estar em 90°. É possível fazer diversas curvas dessa maneira, sendo muito mais rápido do que a maneira convencional de segurar a direção que está fazendo a curva. Descobri o método ao conversar com Beck Abney, o melhor jogador de Mario Kart 64 do mundo, e ele comentou o que estava acontecendo e se disse surpreso pela velocidade do algoritmo, visto que ele se movimenta para direita e esquerda mesmo nas retas, semelhante ao problema do pêndulo invertido, corrigindo ao pressionar a direção oposta no momento em que está indo na direção errada, para compensar. Beck Abney concluiu a pista 2 segundos mais rápido que o Smart Kart ao fazer o melhor possível dentro da pista.

**Figura 24: Geração x Desempenho KPT TWEANN’s**

Fonte: O autor (2019).

Este é um benchmark interessante, pois este é o único resultado onde a pista foi concluída dentre 10 tentativas. A pista possui dois solos principais e somente um deles é mapeado para a entrada da rede neural, como consequência existem diversos momentos na pista onde a visão do Smart Kart é nula, como se não houvesse pista no lugar onde está. Ainda sim dentre os 10 testes esta topologia se desenvolveu contornando os problemas, mesmo batendo nas paredes diversas vezes, o cérebro artificial acelera sempre, esperando que a pista reapareça. Vemos a presença de uma reta enorme na sua curva de aprendizado, e isto se deve ao fato de que realmente não é esperado que o algoritmo consiga aprender, mas ele complexifica bastante as topologias até que seja possível alcançar um melhor resultado, afinal, esta é a natureza de um algoritmo genético.

**Figura 25: Geração x Desempenho FS TWEANN’s**

Fonte: O autor (2019).

Frappe Snowland é uma pista 95% mapeada, pois em sua última reta há uma ponte logo antes da linha de chegada, a ponte em questão não é mapeada, logo na visão do algoritmo é como se não houvesse nada ali. O algoritmo aprende rapidamente a chegar até ponte, quase concluindo a pista, mas leva mais 20 gerações complexificando a topologia para resolver o problema novamente, como em Koopa Troopa Beach. A diferença entre esta pista e a anterior é que somente este pequeno trecho no final não está mapeado, então ela também é utilizada como benchmark principal. A topologia da figura 25 possui 69 genes, e é a maior que já foi produzida em todos os testes. Como esperado, ela é extremamente especifica para a pista em questão, e não conduz bem o veículo em nenhuma outra pista.

**Figura 26: Geração x Desempenho MR TWEANN’s**

Fonte: O autor (2019).

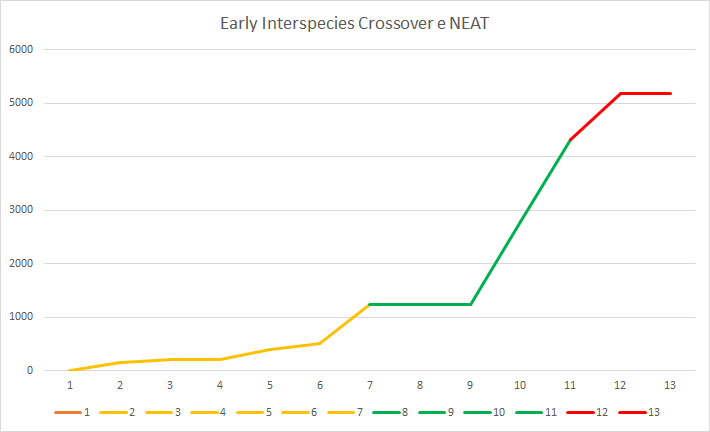
Mario Raceway é uma pista bem simples e completamente mapeada. Ela possui uma curva de 180° bastante brusca, então na geração 4 já existem genomas que conseguem conduzir o veículo, porém apenas na geração 7 a curva é superada, e na geração 10 todos os genomas já são capazes de concluir a pista. A topologia desta pista é bem pequena (8 genes) e foi a que apresentou os melhores resultados para o propósito do Smart Kart antes do EIC e Overpopulation, ela é capaz de dirigir em Luigi Raceway e Moo Moo farm.

As pistas Choco Mountain, Kalamari Desert e Rainbow Road são utilizadas como benchmark somente do EIC / NEAT, pois nunca foram concluídas pelo algoritmo inicializado por TWEANN’s, o que já nos mostra que o EIC e Overpopulation funcionam aumentando a eficiência.

**4.2 Curva de aprendizado com inicialização EIC / NEAT**

As curvas com o EIC (Early Interspecies Crossover) e especiação desde a primeira geração, lembrando que o EIC só é ativo durante a obtenção de uma topologia base.

**Figura 27: Smart Kart, benchmark final, EIC / NEAT**



Fonte: O autor (2019).

Temos em amarelo o treino inicial em Frappe Snowland, que está disponível no YouTube, em verde o aprendizado em Luigi Raceway, utilizando o Overpopulation, que como podemos ver o aprendizado fica estagnado por duas gerações, porém volta a ser otimizado na geração 9, onde os genomas voltam a evoluir na descida do Overpopulation. Em vermelho o algoritmo em contato com Choco Mountain, onde ele conclui durante a geração 12, não exatamente com a topologia modelo que é a do gráfico, que consegue dar duas voltas e não conclui a última curva, mas com uma de outra espécie, o que nos mostra que mais de uma espécie está evoluindo simultaneamente graças a especiação.

O mesmo genoma, ou seja, o mesmo cérebro artificial é capaz de dirigir em 5 pistas diferentes, dando 3 voltas: Moo Moo Farm, Luigi Raceway, Kalamari Desert, Choco Mountain e Rainbow Road, além de é claro chegar até a ponte em Frappe Snowland. É possível assistir a este genoma dirigindo a pistas diferentes simultaneamente em meu canal no Youtube:

< https://www.youtube.com/watch?v=qGenSqozPvU > , Genoma multi pistas, O autor (2019). < https://www.youtube.com/watch?v=AwHlrEmoWLg > , SK Frappe Snowland – Treinamento, O autor (2019).

**5. CONCLUSÃO**

O algoritmo se prova capaz de conduzir o kart em pistas completamente diferentes a partir do mesmo cérebro artificial, utilizando o Overpopulation e o Early Interspecies Crossover para quebrar a inércia produzida pelo algoritmo genético em si.

A especiação nos garante um leque grande de soluções, e em Choco Mountain podemos ver que ela funciona muito bem, pois as topologias em questão não são tão pequenas assim (possuem em torno de 15~30 genes), e ainda é possível otimizar e aperfeiçoar a condução.

O Algoritmo foi todo desenvolvido visando situações reais, e por isso o mapeamento é a simulação de um sensor lendo o espaço ao redor do kart em tempo real. Acredito que ele é apto a ser desenvolvido para testes em campo, construindo os cérebros em simuladores claro, para evitar perdas materiais, visto que o algoritmo está aprendendo a conduzir um veículo completamente do zero.

Por curiosidade, o tempo para que isso aconteça do zero até a maestria é em torno de 10 a 16 horas, mas pode ser mais rápido, dependendo do processador do computador onde o algoritmo estiver sendo executado.

**Referências:**

MUNIZ, André. **Redes Neurais Evolucionárias**. 19 de maio de 2015.

Disponível em: <https://www.researchgate.net/publication/267297071\_Redes\_Neurais\_Evolucionarias> Acesso em: 5 de abril de 2019.

STANLEY, Kenneth e MIIKKULAINEN, Risto. **Evolving Neural Networks through Augmenting Topologies**. 2002. Disponível em: <http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf> Acesso em: 5 de abril de 2019.

STANLEY, Kenneth. **The NeuroEvolution of Augmenting Topologies (NEAT) Users Page**. 31 de março de 2003. Disponível em: <https://www.cs.ucf.edu/~kstanley/neat.html#intro>. Acesso em: 5 de abril de 2019.

GOMEZ, Faustino e MIIKKULAINEN, Risto. **Robust non-linear control through neuroevolution**. Disponível em: <http://people.idsia.ch/~tino/papers/gomez.tr02-292.pdf>

WEN, Yeming, VICOL, Paul e BA, Jimmy, **FLIPOUT: EFFICIENT PSEUDO-INDEPENDENT WEIGHT PERTURBATIONS ON MINI-BATCHES**. Disponível em: <https://openreview.net/pdf?id=rJNpifWAb>

STANLEY, Kenneth. **Neuroevolution and Developmental Encoding NeuroEvolution of Augmenting Topologies (NEAT)**. 25 de Setembro, 2006. Disponível em: < https://slideplayer.com/slide/13570940/release/woothee>.

LERUSALIMSCHY, Roberto, FIGUEIREDO, Luiz e CELES, Waldemar. **Lua 5.1 Reference Manual**, tradução por MEDEIROS, Sérgio e revisão por LERUSALIMSCHY, Roberto.

Disponível em: < https://www.lua.org/manual/5.1/pt/> Acesso em: 4 de março de 2019.

Biblioteca Math em Lua:

Disponível em: <http://lua-users.org/wiki/MathLibraryTutorial>.

FACURE, Matheus. **Funções de ativação: Entendendo a importância da ativação correta nas redes neurais**:

Disponível em: <https://matheusfacure.github.io/2017/07/12/activ-func/>.

GROOM, James. **EmuLuaLibrary.Gui.cs**.

Disponível em: <https://github.com/TASVideos/BizHawk/blob/master/BizHawk.Client.EmuHawk/tools/Lua/Libraries/EmuLuaLibrary.Gui.cs>.

BORGES, Augusto. **Aprendizado de Máquinas - USP - neurônio numa rede neural funções de transferência**. Disponível em: <https://youtu.be/Dc6rxNy-OPE >

WHEATERTON, Drew. **Partially automate the process of executing perfect maneuvers in Mario Kart 64**. Disponível em: <https://github.com/weatherton/BizHawkMarioKart64/blob/master/MarioKart64\_AutoTransmission.lua >

**Bizhawk / Lua Functions**:

Disponível em: <http://tasvideos.org/Bizhawk/LuaFunctions.html>.

GOODFELLOW, Ian e BENGIO, Yoshua. **The Deep Learning Book.**

Disponível em :< http://www.deeplearningbook.org/>.

Data Science Academy. **Deep Learning Book Brasil.**

Disponível em:<http://deeplearningbook.com.br>.

ZEGKLJAN. Stackoverflow userpage:

Disponível em:< https://stackoverflow.com/users/461202/zegkljan >.

DONOVAN, Steve. **Lua Unofficial FAQ (uFAQ)**. Disponível em: < https://www.luafaq.org/#T1.10>.

LE, Hannah. **NeuroEvolution, NEAT Algorithm and My NEAT**. Disponível em: <https://medium.com/datadriveninvestor/neuroevolution-neat-algorithm-and-my-neat-b83c5174d8b0>.

LE, Hannah. **Evolving A Soft Robot To Walk On Land Using CPPN-NEAT**, The Knowledge Society. Janeiro de 2019. Apresentação Disponível em:

<https://www.youtube.com/watch?v=WiPZSieT6qs>.

Artigo disponível em: <https://medium.com/@hannah.lgbhan/growing-and-evolving-soft-robots-to-walk-in-new-environments-1e6f15c8bb6e>.

SHYGOO. **Mk64project**.

Disponível em: <https://github.com/shygoo/mk64project>.

SHYGOO. **Mario Kart 64 hacking general discussion**. Disponível em: <http://shygoo.net/o64-backup/mk64\_hacking/Mario%20Kart%2064%20Hacking%20General%20Discussion%20-%20Origami64.html>.

RENAKUNISAKI, 2017. **Wiki documents the inner workings of Mario Kart 64 and how to extract, render, and modify its courses**. Disponível em: <https://github.com/RenaKunisaki/mariokart64/wiki>. Acesso em