Revista brasileira de iniciação científica
ISSN: 2359-232X

# ENXAME DE PARTÍCULAS APLICADO A UMA REDE NEURAL COMO BACKPROPAGATION PARA OTIMIZAÇÃO DE TEMPO DE JOGO

# PARTICLE SWARM APPLIED TO A NEURAL NETWORK AS BACKPROPAGATION FOR OPTIMIZING GAME TIME

# CHISPA DE PARTÍCULAS APLICADA A UNA RED NEURAL COMO RETROPROPAGACIÓN PARA LA OPTIMIZACIÓN DEL TIEMPO DE JUEGO

Ana Paula Capeletti Ramos Almeida[1]
Evandro Alves Nakajima[2]

**Resumo:** Este trabalho explora o algoritmo enxame de partículas como *backpropagation* de uma rede neural, baseando-se no jogo *offline* do Google Chrome, que é um *endless runner* (jogo onde o personagem corre automaticamente e continuamente para a direita), com o objetivo de maximizar o tempo de jogo. Para otimizar a rede foi utilizado o Particle Swarm Opmization (PSO) que é uma técnica de *backpropagation* criada para simular comportamento de um bando de pássaro. Pode-se concluir que o algoritmo PSO foi eficiente para o processo de otimização uma vez que os jogadores conseguiram atingir o tempo máximo estipulado em apenas 4 rodadas.
**Palavras-chave:** Aprendizado de máquina. Particle swarm optimization. Jogo do Chrome.

**Abstract:** This work explores the particle swarm algorithm as a backpropagation of a neural network, based on the Google Chrome offline game, which is an endless runner (game where the character runs automatically and continuously to the right), with the objective of maximizing time of play. To optimize the network, Particle Swarm Opmization (PSO) was used, which is a backpropagation technique created to simulate the behavior of a flock of birds. It can be concluded that the PSO algorithm was efficient for the optimization process since the players managed to reach the maximum time stipulated in just 4 rounds.
**Keywords:** Machine learning. Particle swarm optimization. Chrome game.

**Resumen:** Este trabajo explora el algoritmo de enjambre de partículas como retropropagación de una red neuronal, basado en el juego offline de Google Chrome, que es un corredor sin fin, con el objetivo de maximizar el tiempo de juego. Para optimizar la red, se utilizó Particle Swarm Opmization (PSO), que es una técnica de retropropagación creada para simular el comportamiento de una bandada de pájaros. Se puede concluir que el algoritmo de PSO fue eficiente para el proceso de optimización ya que los jugadores lograron alcanzar el tiempo máximo estipulado en solo 4 rondas.
**Palabras-clave:** Aprendizaje automático. Optimización de Enjambre de partículas. Juego de Chrome.

**Submetido 01/04/2021** **Aceito 27/12/2022** **Publicado 29/12/2021**

[1] Graduanda. Universidade Tecnológica Federal do Paraná. E-mail: anaalm@alunos.utfpr.edu.br. Orcid: 0000-0002-1992-4575;
[2] Mestre. Universidade Tecnológica Federal do Paraná. E-mail: enakajima@utfpr.edu.br. Orcid: 0000-0003-4758-9505.

Revista brasileira de iniciação científica
ISSN: 2359-232X

**Introdução**

Uma das áreas de pesquisa com uma ampla visibilidade na atualidade é a simulação de capacidades cognitivas de um ser humano, a qual tem por objetivo projetar máquinas capazes de exibir um comportamento inteligente (Haykin, 2007).

No cérebro o neurônio é responsável pela resposta a estímulos do meio externo ao corpo, como a luz e o calor. Um interesse que surge desses fatos é a tentativa de reproduzir o funcionamento da inteligência do cérebro em um meio técnico. Utilizar uma estrutura artificial para mapear as sinapses realizadas pelos neurônios, assim transformando as redes neurais biológicas em artificiais (Haykin, 2007).

O primeiro grande trabalho (Norvig; Russell, 2013) reconhecido como IA foi realizado por Warrem Macculloch e Walter Pitts em 1943, que aborda a lei do tudo ou nada: "*the all or none law of nervous activity is sufficient to ensure that the activity of any neuron may be represented as a proposition*" (Mcculloch; Pitts, 1943). Já o primeiro artigo a respeito de IA presente na base de dados do *Web of Science* é datado de 1960 e intitulado "*Bibliography on Simulation, Gaming, Artificial Intelligence and Allied Topics*", do autor Martin Shubik, contendo referências sobre inteligência artificial, jogos e sobre o método de Monte Carlo (Shubik, 1960).

Ainda pela base de dados do *Web of Science* pode-se verificar o crescente interesse da pesquisa sobre IA, uma vez que o número de artigos publicados em 2020 (3253 artigos) é aproximadamente 10 vezes maior do que há 5 anos (329 artigos em 2015) e 19 vezes maior de que há 10 anos (173 artigos em 2010) (Wos, 2021).

Com base no que foi apresentado, o objetivo deste trabalho é desenvolver e avaliar uma rede neural com *backpropagation* baseado no enxame de partículas (método matemático não determinístico amplamente utilizado (Khare; Rangnekar, 2013) para otimização da solução de um problema.

O problema proposto neste trabalho foi um jogo que se baseia no jogo *offline* do Google Chrome, onde o personagem (dinossauro) corre automaticamente e continuamente para a direita, com o propósito de maximizar o tempo de jogo que consecutivamente aumenta a pontuação do jogador.

Revista brasileira de iniciação científica
ISSN: 2359-232X

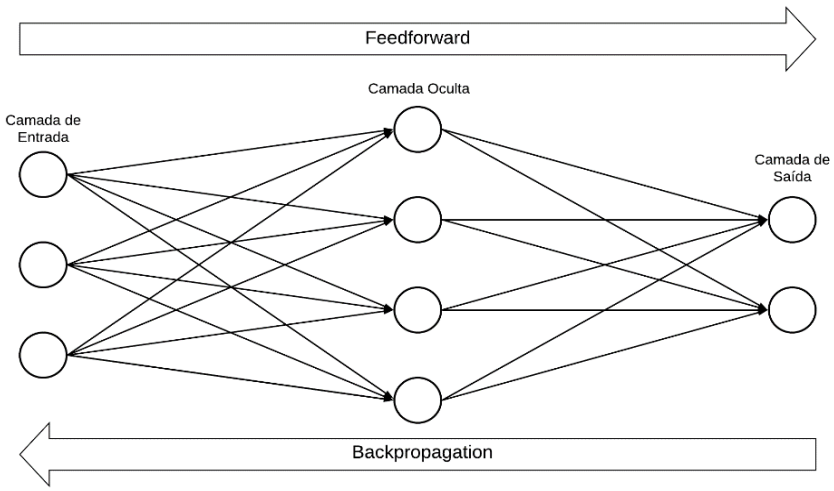
**Redes Neurais**

O cérebro é um computador altamente complexo, não-linear e paralelo. Ele possui a capacidade de organizar seus constituintes estruturais, conhecidos como neurônios, de forma a realizar processamentos mais rápido que os computadores digitais existentes (Haykin, 2007). Na literatura é possível verificar diversas áreas que aplicam as técnicas de redes neurais, dentre elas: processos de manufatura, processos químicos e processamento de matérias, reconhecimento automático de alvos, reconhecimento de caracteres, reconhecimento de padrões, otimizações, robótica, diagnóstico médico, sensoriamento remoto, processamento de voz, biometria e análise de dados (Oludare *et al.*, 2018).

As redes neurais são compostas minimamente por 1 camada de entrada e 1 camada de saída, sendo que entre essas camadas podem conter inúmeras camadas intermediárias, também denominadas de camadas ocultas. A saída dos neurônios de cada camada são as entradas dos neurônios da camada subsequente, desde a camada de entrada, até a camada de saída (Aguiar, 2010).

A rede neural se mostra eficiente na tomada de decisões e em achar padrões, uma vez que aproxima entradas e saídas, convergindo para resposta correta (saída) baseando-se nos dados de entrada fornecidos (Martiniano *et al.*, 2016). A rede neural genérica (Figura 1), que é a mais comum, tem dois processos: *Feedforward* e *Backpropagation*.

Figura 1 - Rede neural genérica




Fonte: Imagem adaptada de Mohammad (2013).

Revista brasileira de iniciação científica
ISSN: 2359-232X

Em uma rede neural *feedforward* (isto é, que possui apenas esse processo), a informação se move em apenas uma direção: dos nós de entrada, através dos nós ocultos para os nós de saída. Não há ciclos ou loops na rede, ou seja, o *feedforward* tem como objetivo classificar o conjunto de estímulos aplicados externamente (coluna de entrada) e enviar essas informações para as próximas camadas (Haykin, 2007).

Os pesos, em geral representados pela letra $w$, são valores que representam o grau de importância que determinada entrada possui em relação a um determinado neurônio (Haykin, 2007), ou seja, esse valor (o peso) para ser atribuído ao seu grau de relevância precisa ser multiplicado pelos valores de cada entrada, os quais passam para camada oculta. Posteriormente, os valores das camadas ocultas são multiplicados por novos pesos e somados, gerando resultados para camada de saída. Se o valor de saída do *feedforward* for um valor específico, executa-se uma ação.

Um dos grandes avanços das Redes Neurais Artificias (Aguiar, 2010) foi a implementação do *backpropagation*, formalizado por Rumelhart, Hinton e Williams (Rumelhart, *et al.*, 1986) pois, ela proporciona o treinamento de redes *Perceptron* Multi-camadas resultando em uma rede com grande poder de generalização e possibilitando a implementação de diversas aplicações, idealizadas atualmente (Aguiar, 2010).

No *backpropagation* o processo de treinamento/aprendizagem ocorre e, para ajustar os pesos, são utilizados algoritmos de otimização (Slowik; Bialko, 2008). Os algoritmos utilizados para a resolução de problemas de otimização podem ser de natureza determinística ou probabilística (Polak, 1971).

**Programação Orientada a Objetos (POO) e Java**

Segundo Manzano (2014), a linguagem de programação Java foi desenvolvida por um grupo de profissionais da empresa *Sun Microsystens, Inc*. a partir de um projeto intitulado *Green*, sob supervisão de James Gosling. A primeira versão do *Java Developer's Kit* (JDK 1.0) foi lançada em janeiro de 1996, por essa mesma empresa, sendo inicialmente disponibilizada para plataformas *Sun Solaris* e *Microsoft Windows*. Dentre as vantagens da linguagem Java pode-se citar: suporte ao paradigma de programação orientada a objetos; independência do sistema operacional em uso, baseando-se no conceito de portabilidade; não opera com uso de

Revista brasileira de iniciação científica
ISSN: 2359-232X

ponteiros, reduzindo a complexidade de gerenciar variáveis e objetos definidos em memória; permite a criação de programas baseados no critério de paralelismo (Sanderson, 2016).

Anterior a linguagem Java, o conceito de programação utilizando orientação a objetos surgiu em meados de 1960, sendo popularizada pela empresa Xerox em 1984, com o lançamento da linguagem *SmallTalk* (Manzano, 2014).

De acordo com Börstle, Bruce e Michiels (2003, p.1) "*object-Oriented Technology (OOT) has become a major topic in most computer science curricula and most universities now teach OOT from the very beginning*", visto que é o paradigma mais amplamente utilizado nas linguagens e ambientes de programação na atualidade (Java, .NET, C++, Ruby, Python, PHP 5). Inúmeros aspectos devem ser considerados para se pensar orientado a objeto, como classes, encapsulamento, objetos, associação, herança e polimorfismo, os quais são os conceitos essenciais (Vahldick, 2007).

Praticamente todo processamento que ocorre em sistemas baseados em linguagens de programação orientada a objetos ocorrem através dos objetos. Esses objetos são instâncias das classes, criadas pelo usuário para conter as especificações e as funcionalidades que os objetos devem ter. Essas funcionalidades são descritas através de métodos, que são equivalentes a procedimentos ou funções, com a restrição de manipular apenas suas variáveis locais e os atributos que foram definidos para a classe (Harbour, 2010).

Para além da linguagem e do paradigma de programação em si, este trabalho propõe a construção de um jogo, pois segundo Xu (2018) a produção de jogos tem potencial de prover um grande e efetivo conhecimento nas diferentes áreas que são integradas, tais como humanidades, matemática, física, arte, inteligência artificial, dentre outros.

**Métodos de Otimização e Particle Swarm Optimization**

Métodos determinísticos, tais como o método Simplex (para problemas lineares) e o método de Newton-Raphson (para problemas não lineares), conseguem se aproximar ao ponto ótimo por meio de uma sequência determinística, que gera possíveis soluções, empregando um ponto de referência como ponto de partida e um vetor de direção para avançar no espaço de busca. Em geral, a obtenção do vetor direção de busca requer do uso de pelo menos a primeira derivada da função objetivo em relação às variáveis do problema (Saramago, 2003).

Revista brasileira de iniciação científica
ISSN: 2359-232X

Neste tipo de método, a função objetivo e as restrições precisam ser representadas por funções matemáticas ou relações funcionais, sendo que a função objetivo precisa ser contínua e diferenciável num espaço de busca convexo (Polak, 1971; Tavares; Correia, 1999).

Métodos probabilísticos, por outro lado, são baseados em algoritmos de origem probabilística, os quais procuram imitar fenômenos ou processos encontrados na natureza. Neste grupo destacam-se, por exemplo, as técnicas de inteligência computacional e suas subáreas (Saramago, 2003).

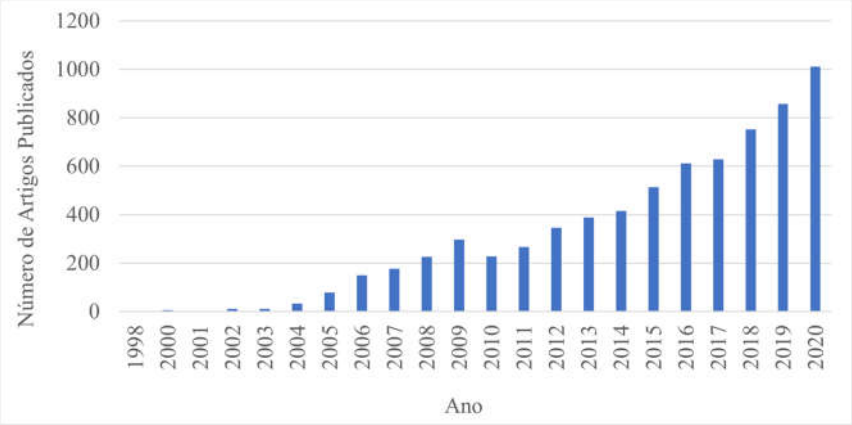
Dentre as vantagens da utilização dos métodos de otimização baseados nos algoritmos probabilísticos determinísticos destacam-se: não requerem que a função objetivo seja contínua ou diferenciável; trabalham adequadamente tanto com parâmetros contínuos quanto com discretos; não necessitam de formulações complexas ou reformulações para o problema; realizam buscas simultâneas no espaço de possíveis soluções através de uma população de indivíduos; otimizam um grande número de variáveis, desde que a avaliação da função objetivo não tenha um custo computacional demasiadamente alto (Bastos, 2004).

Para otimizar a rede neural deste trabalho foi utilizado o *Particle Swarm Opmization* (*PSO*) que é uma técnica estocástica criada em 1995 por James Kennedy e Russell Eberhart para simular comportamento de um bando de pássaros (Kennedy; Eberthart, 1995). Um enxame pode ser caracterizado como um conjunto estruturado de organismos interativos. A ideia do algoritmo utiliza como base o bando de pássaros e, deste modo, costuma-se utilizar termos técnicos, tais como: espaço de busca, solução ótima, partículas, iteração, melhor local e melhor global, que são, respectivamente, a área sobrevoada pelos pássaros, a localização do objetivo (alimento, ninho), os pássaros, o número de vezes que o grupo irá se movimentar, a melhor posição conhecida pelo pássaro (experiência individual) e a melhor posição conhecida pelo bando de pássaros (experiência coletiva).

Pesquisas envolvendo o método *PSO* relacionado às redes neurais vem crescendo a cada ano, como indica a pesquisa realizada na base de dados *Web of Science* (Wos, 2021) com os termos "*particle swarm optimization*" e "*neural network*", a qual indica a primeira publicação (Zhenya, 1998) com o tema no ano de 1998 e mais de mil publicações em 2020, com um crescimento anual médio de 16,34% nos últimos dez anos, como indicado na Figura 2.

Figura 2 – Artigos publicados por ano com os termos "*Particle swarm optimization*" e "*neural network*"

Revista brasileira de iniciação científica
ISSN: 2359-232X




Fonte: Elaborada pelo autor em consulta à base da dados Wos (2021)

O esquema do PSO**,** descrito para N partículas, pode ser apresentado da seguinte forma (Marini; Walczak, 2015):

1. Inicialização:

a. Inicialize a posição $\boldsymbol{X_i^0}$ para todo $\boldsymbol{i}$ de $\mathbf{1}$ a $\boldsymbol{N}$;

b. Inicialize a melhor posição como a posição inicial;

c. Calcule a função fitness (objetivo) para cada partícula e faça a melhor partícula aquela que recebeu o melhor resultado da função.

2. Até atingir o critério de parada, faça:

a. Atualize a velocidade

$$\boldsymbol{v_i^{k+1} = v_i^k + c_1 \cdot \left(p_i - k_i^k\right) + c_2 \cdot \left(g - x_i^k\right)}$$

b. Atualize a posição da partícula;

c. Calcule a função fitness de cada partícula;

d. Atualize a melhor partícula local;

e. Atualize a melhor partícula global.

Em que $X_i^k$ posição da partícula $i$ na rodada $k$, $p_i$ é a melhor posição da partícula $i$, $g$ é a melhor posição entre todas as partículas, $c_1$ e $c_2$ são constantes aleatórias definidas previamente.

**Construção do jogo**

O jogo criado é um *endless runner*, baseado no T Red do Chrome (GOOGLE, 2014), desenvolvido em Java pelo aplicativo *eclipse-workspace*, executado em um desktop com

Revista brasileira de iniciação científica
ISSN: 2359-232X

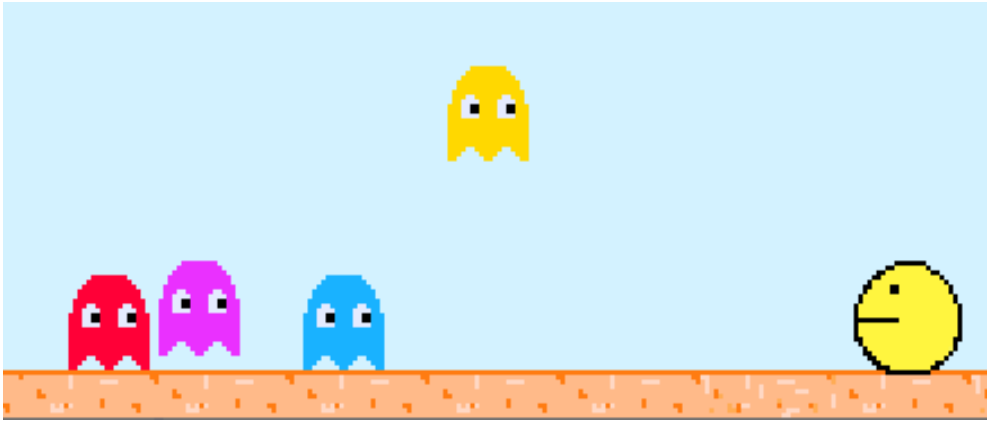
processador intel core i7 4790 K, 32 Gb RAM (DDR 4, 1600 MHz), duas placas de vídeo gtx970, fonte 1200 W e 3 SSDsem raid0. O jogo foi desenvolvido para utilizar o *PSO* como método de *backpropagation*, operando testes para maximizar o tempo de jogo, assim chegando ao aprendizado total da rede, deixando-a infindável.

Há quatro classes principais na física do jogo: jogador, inimigo, nuvem e chão, responsáveis pelo que é mostrado na tela. Para cada classe de renderização há uma classe *entity* que armazena os dados de suas respectivas classes (como posição e velocidade).

As classes jogador, inimigo e nuvem possuem a *extends* da classe pai, de suas respectivas classes *entities*. Cada uma delas é responsável por gerenciar os objetos armazenados em suas respectivas *entities*, seja criando animações ou mesmo estabelecendo condições para gerar novos objetos. Dessa forma, os jogadores irão receber a função pulo, que ao ser ativada desloca o jogador para cima a uma velocidade de 18 pixels por segundo (Figura 3), até atingir a altura máxima predefinida (85 pixels), que ao atingir ele inicia o processo de queda, a uma velocidade 17 pixels por segundo. Outras classes importantes para o propósito deste trabalho são: pesos, rede neural e a classe *game*.

Classe Pesos: gera pesos aleatórios até que algum jogador (fantasma) pule o primeiro inimigo (*pacman*), que ocorre após seis segundos (Figura 3). A classe peso então não é mais executada e o jogo passa a usar a classe rede neural até o final (quando todos os fantasmas colidem com os inimigos). Se nenhum fantasma passa o primeiro inimigo o jogo reinicia e executa a classe pesos, iniciando aleatoriamente todos os pesos novamente.

Figura 3 – Tela do jogo em execução

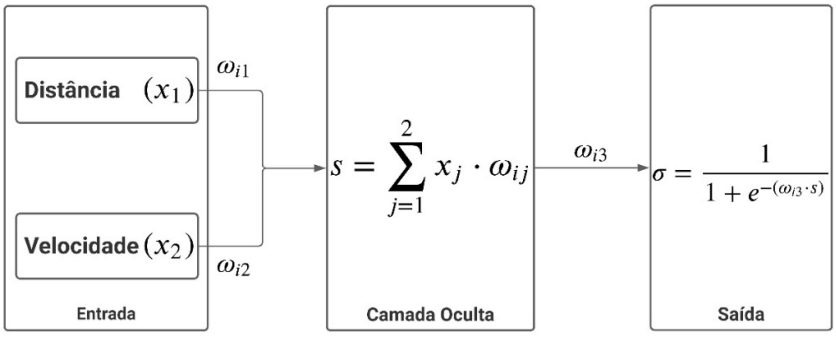


Revista brasileira de iniciação científica
ISSN: 2359-232X

Fonte: Elaborado pelo autor.

Classe rede neural: é responsável por executar o *feedfoward*, tomando a decisão de quando o jogador deve pular, e por aplicar o algoritmo *PSO* após o final do jogo para otimização dos pesos (*backpropagation*). Após os pesos $\omega_{ij}$ serem gerados pela classe Pesos ou pela classe *PSO*, cada um é multiplicado pelo seu respectivo parâmetro $(j)$ de seu respectivo jogador $(i)$. A saída da rede neural, para cada jogador, é um número real no intervalo $(0,1)$, estabelecido pela função sigmoide aplicada no resultado da camada oculta multiplicada pelo seu respectivo peso (Figura 4). Quando o valor na saída de cada jogador é superior à 0,5, ativa-se a função pulo da classe jogador, fazendo com que aquele jogador se desloque verticalmente, podendo assim pular o obstáculo à frente.

Figura 4 - Rede neural implementada para cada jogador i




Fonte: Elaborado pelo autor.

Classe *game*: é a classe mais importante, pois é ela que faz o jogo funcionar. Possui várias variáveis que as outras classes utilizam, desde o valor da altura, até a quantidade de jogadores. Essa classe é responsável por chamar as outras funções quando necessário e nela há a *extends* do componente canvas, que é uma classe própria do Java que representa uma área retangular em branco na tela da qual o aplicativo pode desenhar ou interceptar eventos de entrada do usuário (Oracle, 2018).

A fim de que os jogos não fiquem iguais a parte gráfica deste jogo foi alterada (Figura 3), desde o chão aos obstáculos. Para verificar a derrota dos jogadores (fantasmas) foi necessário criar um retângulo em cada jogador e em cada obstáculo, com alturas iguais à maior quantidade de pixels verticais do objeto contido (jogador ou obstáculo) e largura igual à maior

Revista brasileira de iniciação científica
ISSN: 2359-232X

quantidade de pixels horizontais do objeto contido. Quando o programa identifica uma colisão, ou seja, quando ocorre a intersecção dos dois retângulos (entre jogador e obstáculo), o jogador colidido é removido do jogo, até que se inicie novamente. Quando todos os jogadores são removidos, o jogo executa a classe rede neural e inicia novamente.

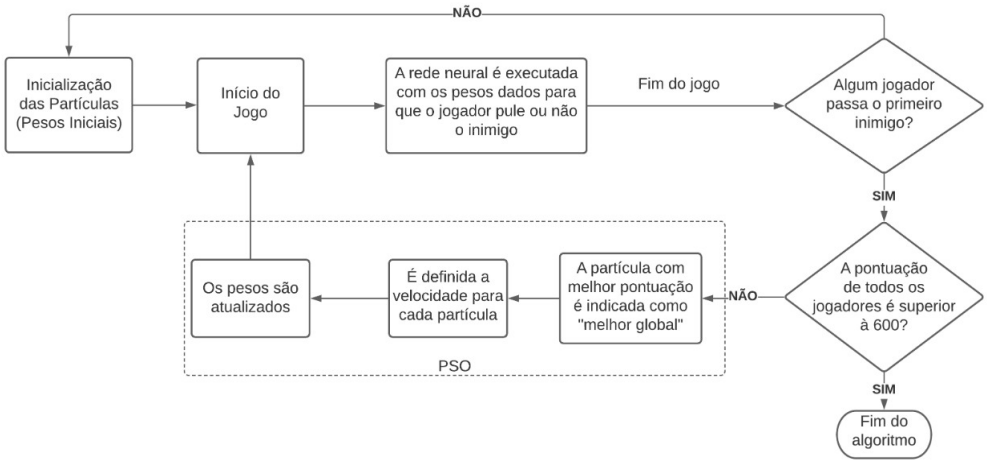
A velocidade do mapa foi ajustada para iniciar em 0,5 pixels por segundo sendo incrementada até atingir um máximo de 5 pixels por segundo, o que acontece após 300 segundos de execução do jogo, não gerando novas dificuldades após esse ponto. Sendo assim, o jogo foi executado por 600 segundos, garantindo que o jogador sobreviva pelo tempo que se desejar.

**Funcionamento do PSO Dentro do Jogo**

No *PSO*, cada solução candidata (possível solução) é chamada de partícula e ela representa uma ponte em um espaço $N$-dimensional, se $N$ for o número de parâmetros a serem otimizados. Para este trabalho o número de parâmetros é igual ao número de pesos da rede neural, isto é, $N = 15$ (três pesos para cada um dos cinco jogadores).

Os pesos iniciais são os pesos atuais (gerados inicialmente pela classe pesos), o jogo é executado mais uma vez e ao final é computada uma pontuação individual (quantidade de segundos que cada jogador levou até colidir com um obstáculo), como exposto na Figura 5. Esse valor define a função objetivo a ser maximizada e é denominado no código como $f_{atual}$.

Figura 5 – Fluxograma do funcionamento do algoritmo




Fonte: Elaborado pelo autor.

Revista brasileira de iniciação científica
ISSN: 2359-232X

A velocidade das partículas foi definida como a uma velocidade inicial ($v_0 = 0{,}3$) dividida pela raiz quadrada da rodada (iniciada a partir de quando o primeiro jogador passa o primeiro inimigo), diminuindo assim o valor da velocidade das partículas, permitindo que as partículas se aproximem do ponto de máximo para função/pontuação, pois o número de rodadas não foi pré-estabelecido (Marini; Walczak, 2015).

**Resultados e Discussão**

Os pesos foram gerados no intervalo de -1 a 2. Na quarta rodada o jogador 1 consegue atingir à pontuação máxima (600), posteriormente os jogadores 3 e 5 (na sexta rodada) e o jogador 4 (na oitava rodada) atinge a pontuação máxima (Tabela 1 e Figura 6). Por fim, o jogador 2 atinge a pontuação máxima na décima rodada, encerrando assim a execução do programa. É possível observar, através da Figura 6 e da Tabela 1 que mesmo o jogador 5 tendo uma pontuação melhor nas rodadas 2 e 3 não é necessariamente o primeiro a atingir o objetivo, exemplificando a capacidade do algoritmo de fazer as considerações locais e globais.

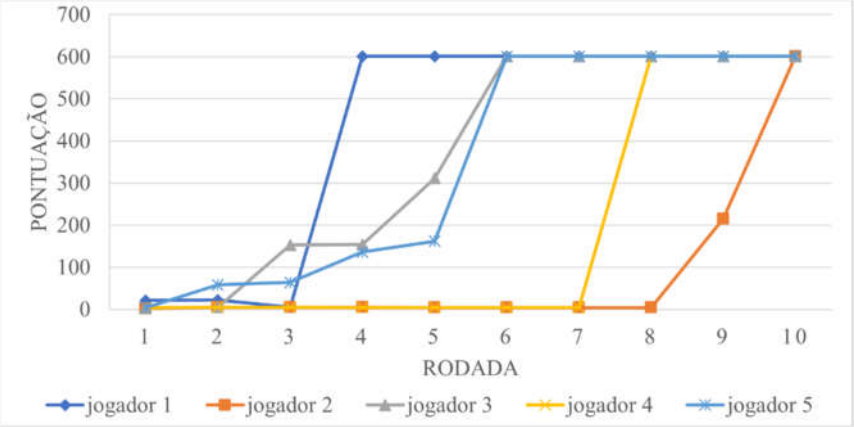
Tabela 1 – Pontuação dos jogadores por rodada

| **Jogador** | **Rodada** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **1** | 21 | 22 | 5 | 600 | 600 | 600 | 600 | 600 | 600 | 600 |
| **2** | 2 | 5 | 5 | 5 | 4 | 4 | 4 | 4 | 215 | 600 |
| **3** | 3 | 5 | 153 | 154 | 311 | 600 | 600 | 600 | 600 | 600 |
| **4** | 4 | 5 | 5 | 4 | 4 | 4 | 4 | 600 | 600 | 600 |
| **5** | 4 | 58 | 64 | 136 | 162 | 600 | 600 | 600 | 600 | 600 |

Fonte: Elaborado pelo autor

Revista brasileira de iniciação científica
ISSN: 2359-232X

Figura 6 - Pontuação dos jogadores por rodada




Fonte: Elaborado pelo autor

O peso 1 converge para aproximadamente 0,668, para todos os jogadores, após 5 rodadas (Tabela 2 e Figura 7).

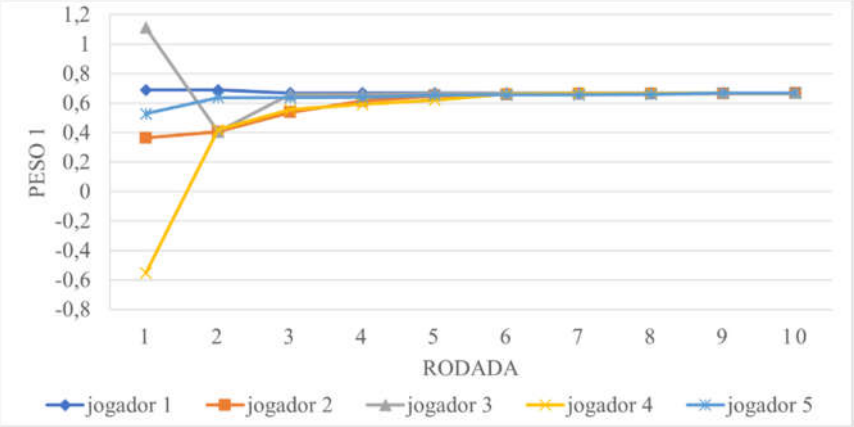
Tabela 2 - Peso 1 dos jogadores por rodada

| **Jogador** | **Rodada** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **1** | 0,688 | 0,689 | 0,668 | 0,668 | 0,668 | 0,668 | 0,668 | 0,668 | 0,668 | 0,668 |
| **2** | 0,363 | 0,404 | 0,537 | 0,613 | 0,648 | 0,658 | 0,661 | 0,662 | 0,663 | 0,665 |
| **3** | 1,110 | 0,404 | 0,657 | 0,657 | 0,665 | 0,667 | 0,668 | 0,668 | 0,668 | 0,668 |
| **4** | -0,554 | 0,418 | 0,555 | 0,592 | 0,620 | 0,662 | 0,665 | 0,665 | 0,667 | 0,667 |
| **5** | 0,527 | 0,636 | 0,636 | 0,638 | 0,653 | 0,655 | 0,655 | 0,658 | 0,668 | 0,668 |

Fonte: Elaborado pelo autor

Revista brasileira de iniciação científica
ISSN: 2359-232X

Figura 7 – Peso 1 dos jogadores por rodada




Fonte: Elaborado pelo autor

O peso 2 converge para aproximadamente 0,198 para todos os jogadores na décima rodada, sendo que para o jogador 1 esse valor é atingido na quarta rodada (Tabela 3, Figura 8).

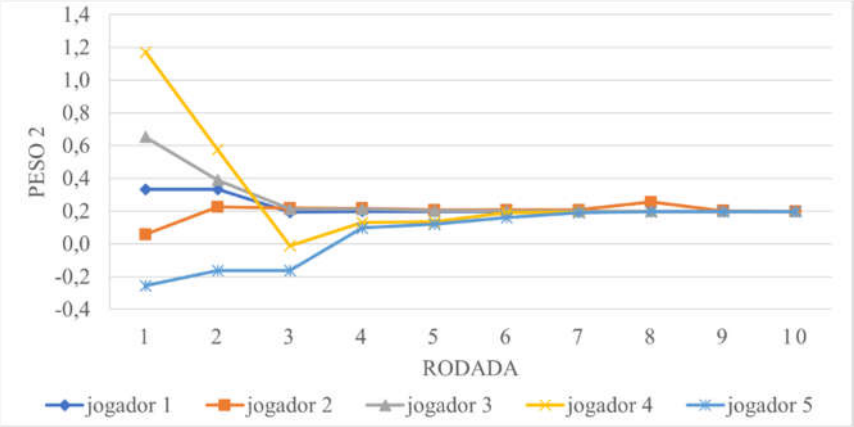
Tabela 3 - Peso 2 dos jogadores por rodada

| **Jogador** | **Rodada** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **1** | 0,333 | 0,333 | 0,193 | 0,199 | 0,199 | 0,199 | 0,199 | 0,199 | 0,199 | 0,199 |
| **2** | 0,058 | 0,226 | 0,218 | 0,217 | 0,209 | 0,209 | 0,208 | 0,255 | 0,202 | 0,198 |
| **3** | 0,653 | 0,390 | 0,213 | 0,213 | 0,204 | 0,201 | 0,200 | 0,199 | 0,199 | 0,199 |
| **4** | 1,172 | 0,576 | -0,012 | 0,131 | 0,135 | 0,191 | 0,194 | 0,195 | 0,196 | 0,197 |
| **5** | -0,255 | -0,163 | -0,163 | 0,098 | 0,120 | 0,161 | 0,191 | 0,197 | 0,197 | 0,197 |

Fonte: Elaborado pelo autor

Revista brasileira de iniciação científica
ISSN: 2359-232X

Figura 8 - Peso 2 dos jogadores a cada rodada




Fonte: Elaborado pelo autor

O peso 3 converge para 0,743 (Tabela 4 e Figura 9). É possível notar que o peso 3 do jogador 4 se aproxima de 0,74 apenas na rodada 8, onde atinge a pontuação máxima.

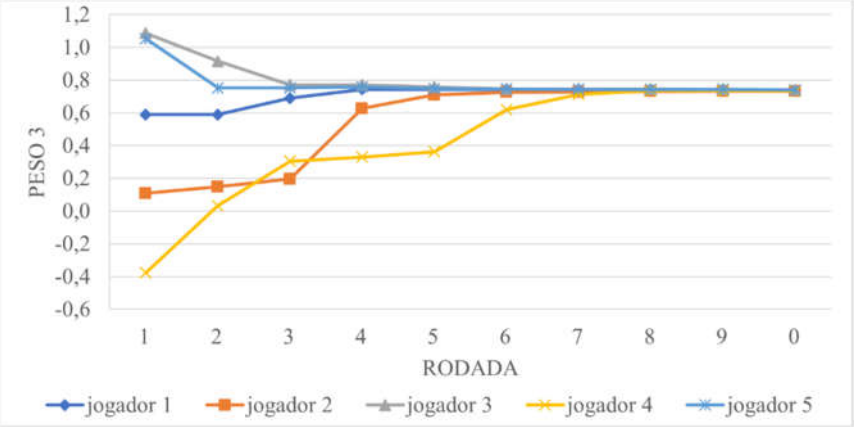
Tabela 4 - Peso 3 dos jogadores por rodada

| **Jogador** | **Rodada** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **1** | 0,589 | 0,590 | 0,690 | 0,743 | 0,743 | 0,743 | 0,743 | 0,743 | 0,743 | 0,734 |
| **2** | 0,109 | 0,148 | 0,197 | 0,627 | 0,710 | 0,726 | 0,728 | 0,731 | 0,732 | 0,732 |
| **3** | 1,087 | 0,915 | 0,771 | 0,771 | 0,759 | 0,746 | 0,746 | 0,745 | 0,743 | 0,741 |
| **4** | -0,376 | 0,033 | 0,305 | 0,330 | 0,362 | 0,620 | 0,713 | 0,735 | 0,735 | 0,735 |
| **5** | 1,054 | 0,752 | 0,752 | 0,758 | 0,747 | 0,746 | 0,744 | 0,744 | 0,744 | 0,740 |

Fonte: Elaborado pelo autor

Revista brasileira de iniciação científica
ISSN: 2359-232X

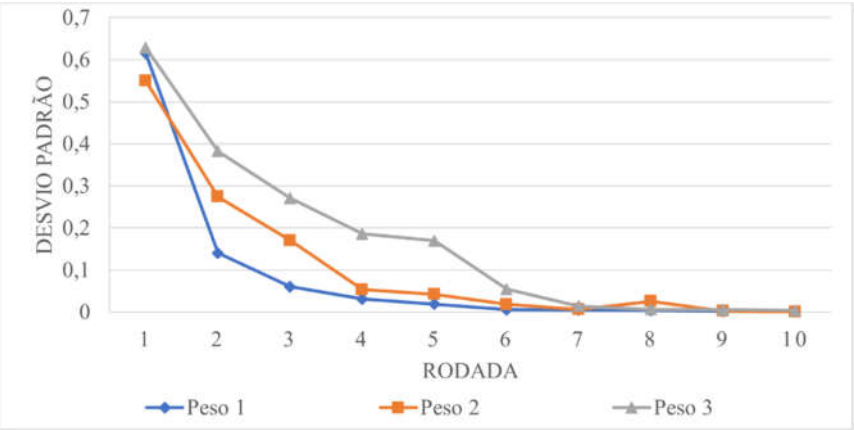
Figura 9 - Peso 3 dos jogadores por rodada




Fonte: Elaborado pelo autor

Em cada rodada, para cada peso $w_i$ foi calculado o desvio padrão (Figura 10), sendo que ao final da décima rodada, os valores dos desvios obtidos foram de aproximadamente 0,001 para o peso 1, 0,001 para o peso 2 e 0,004 para o peso 3.

Figura 10 – Desvio padrão dos pesos por rodada




Fonte: Elaborado pelo autor

Revista brasileira de iniciação científica
ISSN: 2359-232X

As Figuras 7, 8, 9 exemplificam o comportamento das partículas no espaço de busca, pois é possível observar, em cada uma delas, a convergência na direção do melhor global a cada rodada. A Figura 10 apresenta, através do desvio padrão, quão próximas as partículas estão a cada rodada, sendo possível observar um valor próximo de zero para todos os pesos a partir da rodada 9. Esse comportamento no desvio padrão reflete as pontuações obtidas por todos os jogadores como um todo, sendo que na rodada 9 todos obtém pontuação superior à 200 e na rodada 10 todos obtém pontuação máxima.

**Conclusão**

O objetivo tratado neste trabalho foi desenvolver o algoritmo enxame de partículas como *backpropagation* de uma rede neural aplicada a um jogo que se baseia no jogo *offline* do Google Chrome, com o propósito de maximizar a pontuação (tempo) do jogador.

Após quatro rodadas um dos jogadores atingiu o tempo máximo estabelecido (Tabela 1 e Figura 7), sendo os pesos $w_1$, $w_2$ e $w_3$ desse jogador na quarta rodada iguais à 0,668, 0,199 e 0,743 respectivamente. Neste momento pode-se afirmar que uma solução foi encontrada pois, ao replicar os mesmos pesos aos demais jogadores, esses devem atingir o tempo máximo estabelecido. O algoritmo continua sendo executado para observação do comportamento das partículas em cada rodada, o qual demonstra a convergência dos pesos dessas partículas para o melhor valor obtido.

Pode-se concluir, através dos dados exibidos nas Tabelas 1, 2, 3 e 4, que o algoritmo *PSO* foi eficiente para o processo de otimização uma vez que o objetivo foi alcançado por um dos jogadores em 4 rodadas, necessitando apenas de 15 partículas (3 pesos × 5 jogadores). A convergência dos demais jogadores ocorre até a decima rodada. Os resultados obtidos neste trabalho sugerem a aplicabilidade e eficiência do método *PSO* como algoritmo de aprendizagem de redes neurais, quem podem ser aplicadas nas mais diversas áreas.

Revista brasileira de iniciação científica
ISSN: 2359-232X

## Referências

AGUIAR, F. G. **Utilização de Redes Neurais Artificiais para detecção de padrões de vazamento em dutos**. 2010. p.95. Dissertação (Mestrado em Engenharia Mecânica)-Universidade de São Paulo, São Carlos, 2010.

BASTOS, E. A. **Otimização de Seções Retangulares de Concreto Armado Submetidas à Flexo-Compressão Oblíqua Utilizando Algoritmos Genéticos.** 2004. p.168. Dissertação (Mestrado em Engenharia Civil)–Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2004.

BÖRSTLER, J.; BRUCE, K.; MICHIELS, I. Sixth workshop on pedagogies and tools for learning object oriented concepts. **ECOOP**, p. 84-87, 2003. Acesso em: 04 out. 2020.

HARBOUR, J. S. **Programação de Games com Java**. Tradução da 2.ed. Boston: Cengage Learning, 2010.

HAYKIN, S. **Redes Neurais:** princípios e prática. Porto Alegre: Bookman Editora, 2007.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. **IEEE**, p. 1942-1948, 1995. Acesso em: 04 out. 2020.

KHARE, A.; RANGNEKAR, S. A review of particle swarm optimization and its applications in solar photovoltaic system. **Applied Soft Computing**, v. 13, n. 5, p. 2997-3006, 2013. Acesso em: 04 jan. 2021.

MARINI, F.; WALCZAK, B. Particle swarm optimization (PSO). A tutorial. **Chemometrics and Intelligent Laboratory Systems**, v. 149, p. 153-165, 2015. Acesso em: 07 out. 2020.

MARTINIANO, A.; FERREIRA, R.P.; FERREIRA, A.; FERREIRA, A.; SASSI, R.J. Utilizando uma rede neural artificial para aproximação da função de evolução do sistema de Lorentz. **Revista Produção e Desenvolvimento**, v. 2, n. 1, p. 26-38, abr. 2016. Acesso em: 01 out. 2020.

MANZANO, J. A. N. G.; JÚNIOR, R. A. C. **Programação de computadores com java**. Saraiva Educação SA, 2014.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, v. 5, n. 4, p. 115-133, 1943. Acesso em: 02 jan. 2021.

MOHAMMAD, A. A.; SOHRAB, Z.; ALI, L.; ALI, E.; IOANNIS, C. Reservoir permeability prediction by neural networks combined with hybrid genetic algorithm and particle swarm optimization. **Geophysical Prospecting**, v. 61, n. 3, p. 582-598, 2013. Acesso em: 18 jan. 2021.

NORVIG, P.; RUSSELL, S. **Inteligência Artificial:** Tradução da 3.ed. Rio de Janeiro: Elsevier Brasil, 2013.

OLUDARE, I. A.; AMAN, J.; ABIODUN, E. O.; KEMI, V. D.; NACHAAT, A. M.; HUMAIRA, A. State-of-the-art in artificial neural network applications: A survey. **Heliyon**, v. 4, n. 11, p. e00938, 2018. Acesso em: 01 jun. 2021.

ORACLE. Class Canvas. **Java Platform**. 7.ed, 2018. Disponível em: <https://docs.oracle.com/javase/7/docs/api/java/awt/Canvas.html>. Acesso em: 06 out. 2020.

POLAK, E., **Computational Methods in Optimization:** A Unified Approach. Cambridge: Academic Press, 1971.

RUMELHART D. E.; HINTON G. E.; WILLIANS R. J. Learinig representations by back propagation error. **Nature**, v. 323, nº. 9, p. 533-536, 1986. Acesso em: 22 jan. 2021.

SANDERSON, S. **Java For Beginners: a simple start to java-written by a software engineer**. Scotts Valley: CreateSpace Independent Publishing Platform, 2016. Acesso em: 01 jun. 2021.

SARAMAGO, S. F. P.; Métodos de otimização randômica: Algoritmos genéricos e Simulated annealling. **Sociedade Brasileira de Matemática Aplicada e Computacional**, v. 6, p. 25-26, 2003. Acesso em: 01 abr. 2021.

SHUBIK, Martin. Bibliography on simulation, gaming, artificial intelligence and allied topics. Journal of the American Statistical Association, v. 55, n. 292, p. 736-751, 1960. Acesso em: 01 jun. 2021.

SLOWIK, A.; BIALKO, M. Training of Artificial Neural Networks Using Differential Evolution Algorithm, **IEEE**, p. 60-65. 2008. Acesso em: 02 out. 2020.

TAVARES, L.V.; Correia, F. N. **Optimização linear e não linear: conceitos, métodos e algortimos**. 2.ed. Lisboa: Livraria Portugal, 1986.

VAHLDICK, A. Uma experiência lúdica no ensino de programação orientada a objetos. **XVIII Simpósio Brasileiro de Informática na Educação**, Blumenau, n.1, 2007.

WOS. **Trust the Difference.** Web of Science Fact Book, 2021. Disponível em: <https://www.webofknowledge.com>. Acesso em: 09 mar. 2021.

XU, C. W. **Learning Java with Games**. 2.ed. New York: Springer, 2018.

ZHENYA, H.; Chengjian, W.; Luxi Y.; Xiqi G.; Susu Y. Extracting rules from fuzzy neural network by particle swarm optimisation, **IEEE**, p. 74-77. 1998. Acesso em: 09 mar. 2021.