# Neuroevolução: uma aplicação em jogos

1st Victor de Oliveira
*Centro de Matemática, Computação e Cognição*
*Universidade Federal do ABC*
Santo André, Brasil
victor.oliveira@aluno.ufabc.edu.br
RA: 11095612

2nd Diego Leivas dos Santos
*Centro de Matemática, Computação e Cognição*
*Universidade Federal do ABC*
Santo André, Brasil
diego.leivas@aluno.ufabc.edu.br
RA: 11082711

3rd Richard da Cruz Lopes
*Centro de Matemática, Computação e Cognição*
*Universidade Federal do ABC*
Santo André, Brasil
richard.lopes1995@gmail.com
RA: 11122015

***Abstract*—This short paper presents an intelligent agent trained using neuroevolution techniques to play the evoman game.**

## I. Introdução

Plataformas de jogos são excelentes *frameworks* para treino, implementação e teste de algoritmos para criação de agentes inteligentes. Neste relatório, iremos tratar da implementação, análise e resultados de um agente baseado em neuroevolução que irá jogar contra oito inimigos diferentes no jogo evoman. O objetivo é simples: derrotar o maior número de inimigos. Porém, dentre os oito disponíveis, podemos escolher quatro para treino somente, e tentar generalizar o resultado, isto é, além de otimizar o agente, precisamos tomar cuidado para que ele não se especialize. A fim de realizar tais tarefas, vamos criar diversos experimentos variando parâmetros do modelo, bem como analisar curvas de *overfitting* pensando em generalização e como definir métricas apropriadas de *fitness*. Este projeto foi implementado utilizando a linguagem Python.

Este relatório está organizado da seguinte maneira: Na seção II iremos definir o ambientes e o PEAS em que nosso agente está situado. Na seção III iremos tratar de uma breve fundamentação teórica do algoritmo proposto (Não é um introdução ao tópico, e sim a solução utilizada por nós), bem como alguns detalhes de sua implementação. Na seção IV iremos discutir os experimentos realizados, como os dados foram coletados e como as métricas estão definidas. Na seção V faremos uma análise dos resultados obtidos, e por fim, na seção VI fecharemos com uma conclusão sobre o que foi discutido.

## II. Ambiente do Agente (PEAS)

PEAS é a sigla para Performance, Environment, Actuators e Sensors. Este descreve as propriedades de cada tipo para que um tipo de agente adequado seja escolhido. Aplicado ao EvoMan, temos o seguinte PEAS:

- Agente: EvoMan
- P: Vencer o maior número de adversários
- E: Fase, adversário
- A: Correr para frente, Correr para trás, Abaixar, Pular e Atirar
- S: distância entre o EvoMan e o adversário

## III. Fundamentação Teórica

Redes Neurais Artificais tem sido utilizadas em diversas aplicações nos últimos anos, variando entre classificação de padrões até em controle de robôs. Porém, a maioria dessas aplicações estão situadas no campo de Aprendizado de Máquina denominados de aprendizado supervisionado, isto é, dado um conjunto de valores de entrada, temos as respectivas saídas, assium conseguimos treinar a rede conforme estes valores. Porém, diversas aplicações não contam com dados categorizados, ou são muito caros para adquirir. Então, para o treino da rede, outras abordagens são necessárias, como a utilização de algoritmos genéticos como forma de otimização dos pesos desta rede.

A arquitetura do algoritmo proposto é simples. Primeiramente, definimos uma rede neural com pesos aleatórios e uma camada intermediária, a figura 1 exemplifica a rede.

Esta rede então será utilizada para jogar o evoman. A questão agora é como otimizar esta rede para que ela treine, mesmo sem exemplos, a solução é a utilização de algoritmos genéticos. Algoritmos Genéticos é uma classe de algoritmos que se baseia em heurísticas de computação evolutiva. Ela utiliza de uma forma simplificada a teoria neodarwinista para criar soluções de otimização. Basicamente, o algoritmo é composto de uma população, gerações, função de cruzamento, também conhecido como *crossover*, mutação e fitness. Os passos do algoritmos são brevemente descritos:

- Primeiro, iniciamos uma população de forma aleatória. No nosso exemplo, são geradas redes neurais com pesos aleatórios.
- Calcula-se o fitness baseado em algum critério para cada indivíduo.

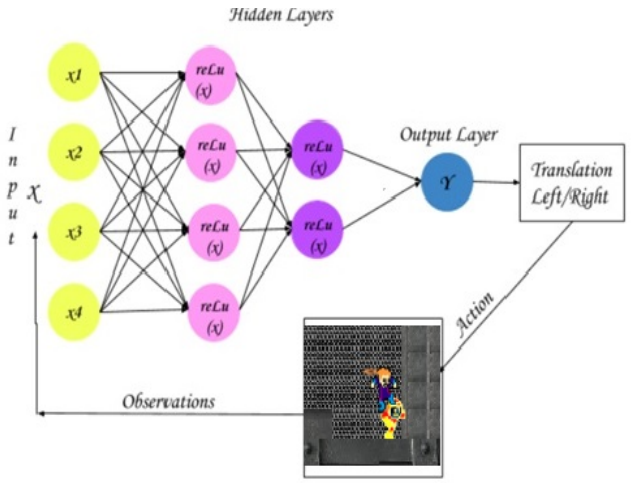


Fig. 1. Exemplo de um rede neural para jogar evoman.

- Após a primeira população ser gerada, passamos para uma geração seguinte. Nela, são gerados filhos baseado em cruzamento dos pais da geração anterior.
- Cruzamento ocorre entre dois pais escolhidos de forma aleatória. Definimos dois tipos de cruzamento: discreto e contínuo. Cruzamento discreto escolhe pontos da rede de forma aleatória e troca entre pai e filho, enquanto que o contínuo é baseado em Estratégias Evolutivas para cromossomos com valores reais, definidos conformae a equação abaixo.
- Nesta etapa, o filho tem uma probabilidade escolhida como parâmetro de sofrer mutação, isto é, altera-se o valor de um peso da rede de forma aleatória.
- Por fim, gera-se uma competição entre pais e filhos para a escolha dos indivíduos da geração seguinte, isto é, baseado nos seus valores de *fitness* queremos que aqueles com melhores soluções permaneçam.

$$w = w_{pai} * \alpha + (1 - \alpha) * w_{mae}$$

Onde w é o peso novo da rede, e alpha é definido entre (0, 1).

Iteramos de geração em geração, até que conseguimos chegar em mínimos locais, ou como esperado, em mínimos globais. Portanto, este tipo de solução trata de um problema baseado em otimização para treino da rede utilizando heurísticas evolutivas.

### A. Métricas

Por fim, vamos discutir como é definido o valor de *fitness* e o que ele representa.

O fitness para cada jogo, isto é, para cada vez que enfrentamos um inimigos é calculado da seguinte forma:

$$fitness = player\_life - enemy\_life - \log(time)$$

Para os valores finais em multiplayer, calcula-se a média do fitness obtida subtraindo-se o desvio padrão.

Iremos abordar estas questões na análise dos resultados, mas podemos ver que é o grande influenciado dos nossos objetivos, pois ele considera a vida do *player* e não dá pontos extras caso o inimigo seja morto, ou seja, dependendo de como queremos abordar o problema, poderíamos modificá-lo. Neste relatório, mantemos o valor calculado padrão e analisamos os resultados em cima dele.

## IV. Experimentos

Para testarmos a validação do algoritmo escolhido, foram realizados diversos experimentos variando-se parâmetros como tamanho da população, número de gerações, tamanho da rede neural, estratégia evolutiva, inimigos treinados. A tabela I resume todos os experimentos realizados.

TABLE I
Experimentos realizados.

| NE | TP | NG | TM | NN | NI | EE |
|---|---|---|---|---|---|---|
| A | 50 | 20 | 0.20 | 10 | [1,2,4,8] | Não |
| B | 50 | 20 | 0.20 | 10 | [1,3,4,8] | Não |
| C | 50 | 50 | 0.30 | 10 | [1,2,4,8] | Não |
| D | 50 | 50 | 0.20 | 10 | [1,3,4,8] | Não |
| E | 100 | 100 | 0.30 | 20 | [1,2,4,8] | Sim |
| F | 100 | 100 | 0.30 | 20 | [1,2,4,8] | Não |
| G | 100 | 120 | 0.30 | 20 | [1,2,4,8] | Sim |
| H | 100 | 100 | 0.30 | 20 | [1,5,7,8] | Não |
| I | 100 | 100 | 0.30 | 20 | [2,3,4,6] | Não |
| J | 100 | 100 | 0.30 | 20 | [1,3,4,5,8] | Não |

Onde temos que:

- NE: Nome do Experimento;
- TP: Tamanho da População;
- NG: Número de Gerações;
- NN: Número de neurônios na camada intermediária da rede;
- NI: Número dos Inimigos;
- EE: Estratégia Evolutiva.

Conforme podemos ver, os experimentos tratam de variar o máximo possível a configuração do algoritmo para que o máximo possível de solução sejam testadas e analisadas.

## V. Análise dos Resultados

### A. Valores de Fitness

Na primeira etapa, após a coleta dos dados realizados, analisamos os valores de fitness médio, máximo, mínimo e o desvio padrão para cada experimento. Desta forma, conseguimos capturar entre as diferentes composições dos nosso experimentos como as populações evoluem, em quais taxas, a ocorrência de mínimos locais, etc. A 2 mostra os resultados.

Podemos ver alguns padrões interessantes pela figura acima, tanto em relação ao grupo de parâmetros utilizados em um experimento bem como típico comportamente de abordagens que utilizam algoritmos genéticos, como, por exemplo:

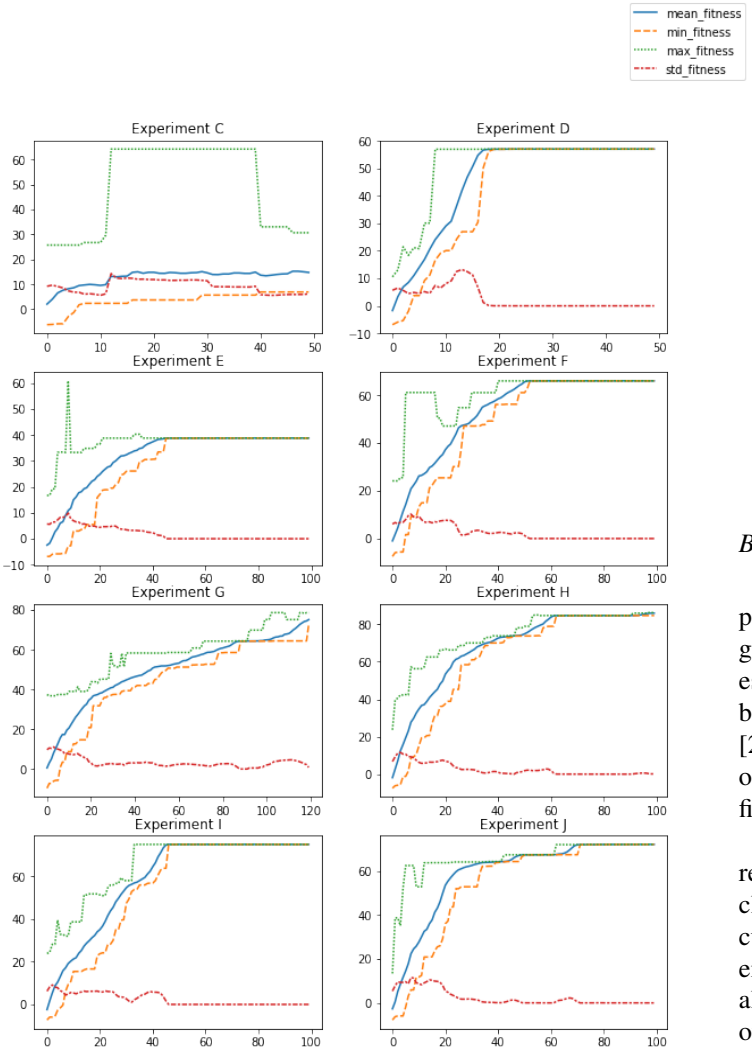


Fig. 2. Curvas de mínimo, máximo, médio e desvio padrão para todos os experimentos..

- Para as soluções que não utilizam estratégia evolutiva, temos um gráfico ascendente em formato de escada para as melhores soluções de cada geração (linha verde pontilhada), típico de algoritmos genéticos. Eles podem indicar como a melhor solução até aquela geração fica presa em mínimos locais. Já para os experimentos E e G, que utilizam estratégias evolutivas, temos curvas mais suaves, provavelmente por utilizar premissas de números reais e não ter cruzamento discreto em pontos, como ocorre nos outros casos.
- Como não implementamos a estratégia elitista, vemos que nem sempre a melhor solução permanece para a geração seguinte.
- O desvio padrão da população diminui sempre conforme as gerações avançam. Isto é totalmente esperado já que conforme as soluções se tornam mais especializadas, elas deixam de fazer *exploitation*, para realizar *exploration* da melhor solução até então e por fim convergir.
- O fitness médio (indicado pela linha azul) aumenta a cada geração. Isto é interessante, pois também não há uma garantia dada pelo algoritmo. O que pode acontecer aqui é que pelo número de entrada e pela complexidade do problema não ser tão grande, ele consegue convergir para um mínimo local.
- O experimento G é o que contém o maior número de gerações e curiosamente ele não atinge um mínimo local, talvez continuar iterando possa melhorar ainda mais sua solução final.
- H e J tem exatamente o mesmo conjunto de parâmetros, exceto os inimigos. Apesar de termos somente uma iteração, aqui podemos ver como a escolha dos inimigos influência no fitness final, mais para frente vamos analisar como eles se comparam com relação aos inimigos de teste.

### *B. Curva de Overfitting*

Conforme descrito na seção A, realizamos um experimento para verificarmos a curva de overfitting que o modelo por gerar por treinar com quatro inimigos diferentes daqueles que estamos testando. Neste caso, utilizamos o experimento como baseline, isto é, após ele ter sido treinado com os inimigos [2,3,5,6], testamos todas as suas populações e gerações para os inimigos [1,5,7,8]. A figura 3 ilustra as curvas de melhor fitness por geração e o fitness médio.

Temos então uma curva que mostra o que seria o treino da rede neural em aprendizado supervisionado, e a curva azul clara representando o erro de teste. Conforme esperado, a curva de treino ó aumenta, enquanto que a curva de teste checa em um ponto máximo e após isso só decai, ilustrando que o algoritmo está realizando *overfitting*. Aqui, podemos notar que overfitting significa que os algoritmo de neuroevolução está especializando-se nas fases em que ele é otimizado, o que é algo totalmente esperado. Porém, nossa intenção é sempre generalizar o agente o máximo possível, por isto essa análise é crucial. Interessante notar também que a curva alta de teste acontece de forma rápida, e não trata-se de um fitness bom. Provavelmente isto acontece pois a configuração dos inimigos é tal que dificulta o processo de generalização do modelo.

### *C. Análise das Melhores Soluções*

Nesta seção vamos abordar e analisar os resultados obtidos para todos os inimigos dos experimentos definidos anteriormente. Devido ao alto custo computacional, não conseguimos finalizar todas as curvas para todos os experimentos, por isso trataremos aqui somente dos três melhores indivíduos da última geração. A figura 4 ilustra o fitness médio obtido para todos os inimigos por experimento.

Vemos que os experimentos que mais se destacam no geral são os G e I. Interessante notar que estes foram treinados com inimigos opostos e estratégias evolutivas diferentes. Enquanto G utiliza crossover de números reais, I utiliza na forma discreta definida por pontos. Sabemos que o fitness no modo *multi-player* é definido como valor médio para todos os inimigos. Porém, é interessante analisar também quantos inimigos foram

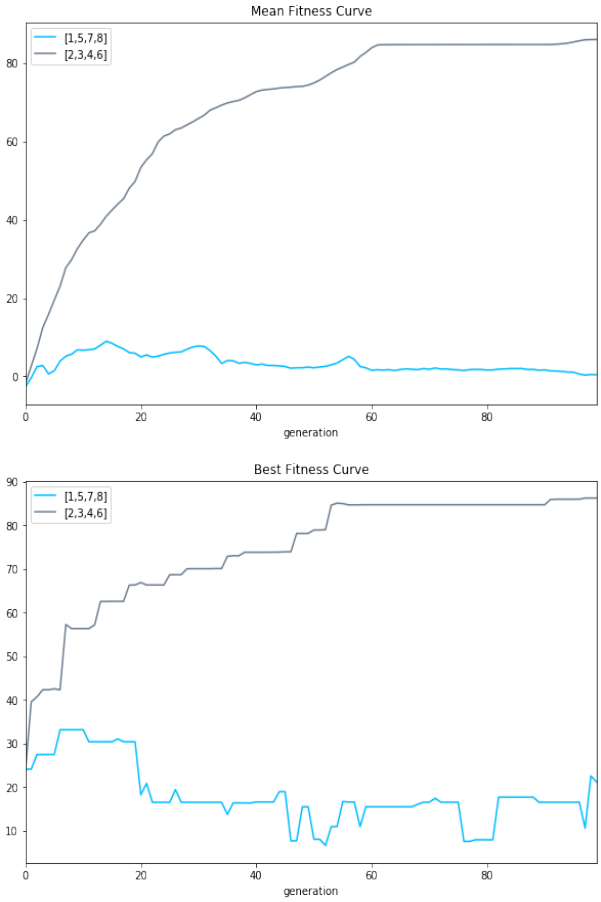


Fig. 3. Curva de overfitting para melhor solução e valor médio da população por geração.

mortos, vida do player, do inimigo e o tempo de duração médio do jogo. A tabela II traz a referência desses valores para cada jogo.

TABLE II
EXPERIMENTOS REALIZADOS.

| NE | PL | EL | TM | NI mortos |
|---|---|---|---|---|
| C | -5.76 | 26.92 | 191.56 | 1 |
| D | -15.51 | 6.35 | 131.63 | 1 |
| E | -12.28 | 8.29 | 132.28 | 2 |
| F | -11.28 | -2.8 | 152.17 | 3 |
| G | -3.62 | -3.72 | 245.54 | 2 |
| H | -4.89 | -3.10 | 195.82 | 4 |
| I | -4.36 | -5.87 | 285.23 | 3 |
| J | -7.24 | -3.73 | 164.39 | 3 |

Para os resultados acima, notamos algumas que o fitness definido varia conforme o nosso objetivo. Neste caso, apesar dele capturar o fitness médio para todos os inimigos, ele leva em conta fatores como vida do jogador, isto é, nem sempre o melhor fitness matou o maior número de inimigos, conforme vemos pelo experimento H, onde o fitness médio é baixo, mas o número de inimigos derrotados é o maior de todos. Também podemos perceber que dado uma tarefa de

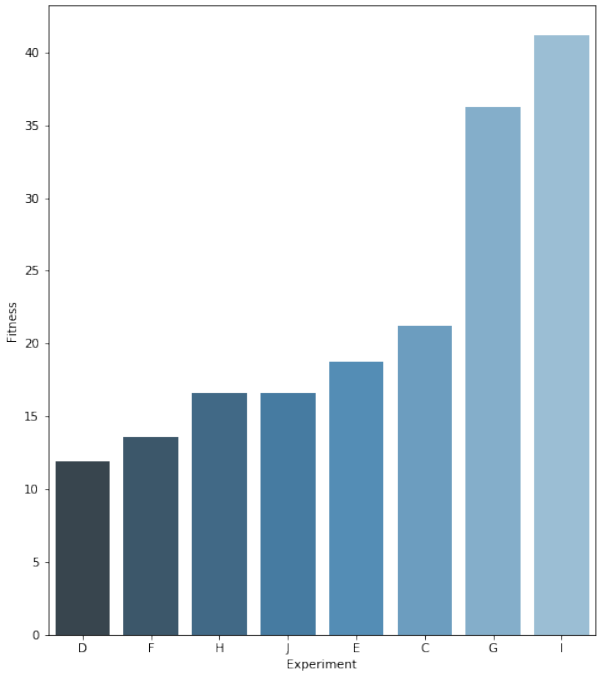


Fig. 4. Melhor fitness para cada experimento para todos os inimigos.

generalização, escolher aquele que tira mais vida do inimigo pode ser mais apropriado, pois ele não especializa em algumas fases, e apesar de não matar muitos inimigos, consegue quase esgotá-los. Talvez com um pouco mais de iterações ele consiga matar mais alguns, o caso do exemplo G.

Portanto, como nosso objetivo definido é derrotar o maior número de inimigos, e também uma generalização melhor, escolhemos o experimento I como nosso agente final, pois ele consegue ser mais generalista e não se especializa tanto nos inimigos em que foi treinado.

### *D. Conclusão*

Neste relatório, tratamos de como agentes inteligentes podem ser criados utilizando redes neurais e heurísticas evolutivas como algoritmos genéticos. Os resultados são interessantes e podem ser estendidos em diferentes vertentes. Conforme discutido, conseguimos verificar como o fitness influência nas soluções obtidas, bem como os parâmetros mexem com a estrutura da função que iremos otimizar, por exemplo, ao utilizar ou não Estratégias Evolutivas. Também conseguimos notar como *overfitting* e falta de generalização pode ser um problema se não controlado de maneira adequado, mais testes são necessários para conclusões maiores. Por fim, tratou-se de uma experiência satisfatória e o entedimento da disciplina foi complementado e aumento de forma significativa ao implementar do zero soluções deste nível de complexidade.

## REFERENCES

[1] Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org
[2] Floreano, D., Dürr, P., Mattiussi, C. (2008). Neuroevolution: from architectures to learning. Evolutionary intelligence, 1(1), 47-62.

[3] Russell, S. J., Norvig, P. (2016). Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited,.