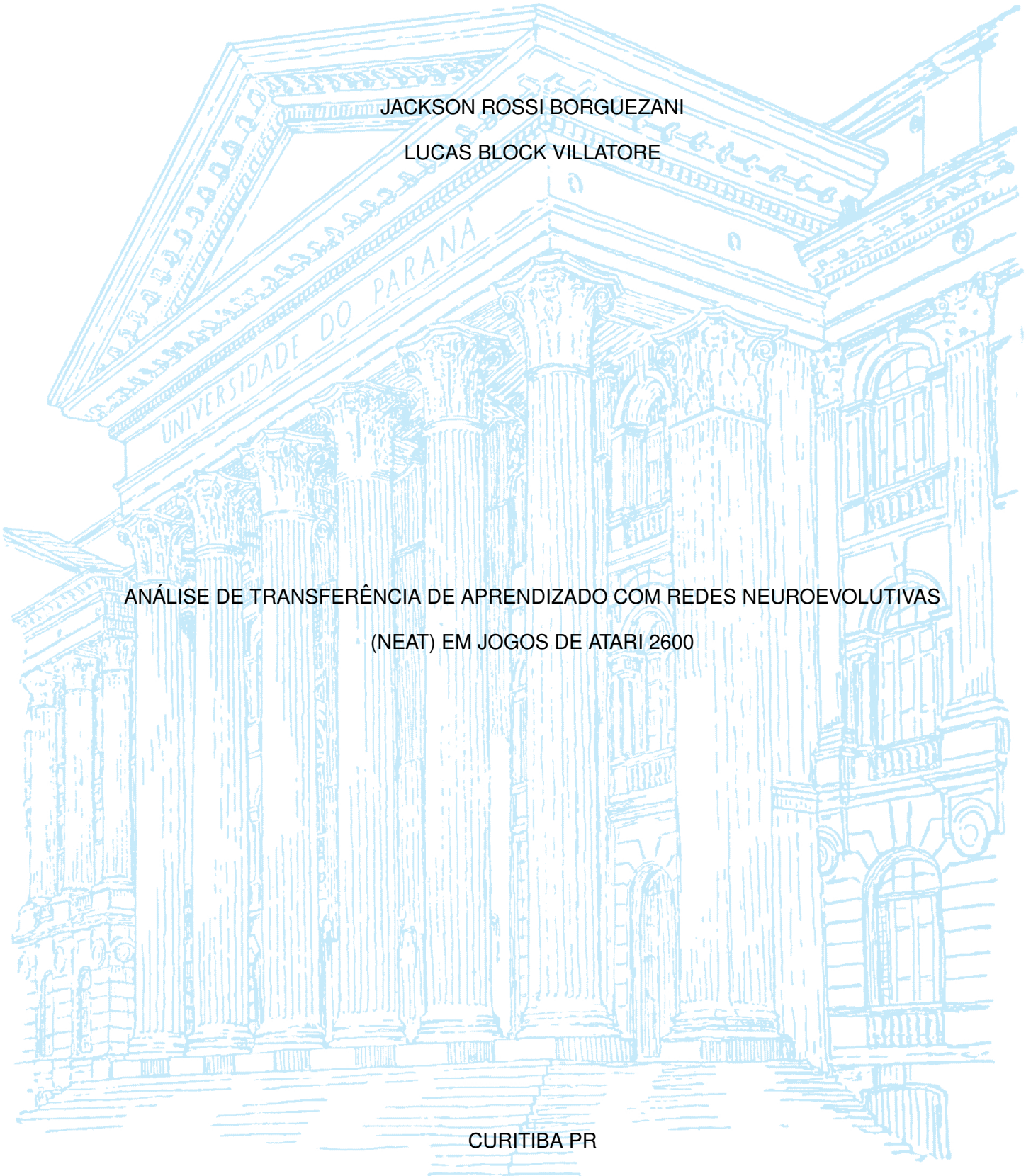
UNIVERSIDADE FEDERAL DO PARANÁ

JACKSON ROSSI BORGUEZANI

LUCAS BLOCK VILLATORE



ANÁLISE DE TRANSFERÊNCIA DE APRENDIZADO COM REDES NEUROEVOLUTIVAS (NEAT) EM JOGOS DE ATARI 2600

CURITIBA PR

2023

JACKSON ROSSI BORGUEZANI
LUCAS BLOCK VILLATORE

ANÁLISE DE TRANSFERÊNCIA DE APRENDIZADO COM REDES NEUROEVOLUTIVAS (NEAT) EM JOGOS DE ATARI 2600

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Eduardo Jaques Spinosa.

CURITIBA PR

2023

AGRADECIMENTOS

Gostaríamos de aproveitar este momento para expressar nossa gratidão e reconhecimento a todas as pessoas e organizações que nos ajudaram durante a realização deste trabalho de conclusão de curso, e também durante todos esses anos de graduação.

Eu, Jackson, inicialmente gostaria de agradecer ao meu namorado e grande incentivador Wender Diego, que se desdobrou em esforços para me ajudar na elaboração do trabalho. Obrigado pelos cafés, por limpar a casa e ouvir minhas lamentações.

Também gostaria de agradecer aos meus amigos e colegas de turma, especialmente Lucas, por topar realizar esse trabalho em dupla, pelo companheirismo e sinergia. E à Djenifer R. Pereira e Fernando M. Kioteka, meu muito obrigado. Vocês foram fundamentais para minha formação, por isso merecem meu eterno agradecimento.

Eu, Lucas, gostaria de agradecer a todos que me auxiliaram nessa jornada, primeiramente meus pais, que se não fosse por eles eu não estaria aqui. Também os amigos que fiz pelo caminho, que ajudaram esse caminho a ser mais descontraído. E por fim, agradecer também o Jackson pelo trabalho e dedicação nesse trabalho.

Em conjunto, gostaríamos de agradecer ao nosso orientador, Eduardo Spinosa, por sua orientação e feedbacks valiosos ao longo do processo. Seu conhecimento e expertise na área foram fundamentais para a conclusão deste trabalho.

Não poderíamos deixar de agradecer às nossas família e amigos, que nos apoiaram e incentivaram durante todo o processo. Seus conselhos, incentivos e palavras de encorajamento nos mantiveram motivados e confiantes em nossa capacidade de realizar este trabalho.

Por fim, gostaríamos de agradecer à Universidade Federal do Paraná pelo ensino de qualidade e a Pró-Reitoria de Assuntos Estudantis pela bolsa permanência. Sem esse apoio, esta graduação não seria possível.

Mais uma vez, agradecemos a todos os envolvidos, direta e indiretamente, por seu apoio, ajuda e contribuição para a conclusão deste trabalho.

RESUMO

Este trabalho tem como objetivo explorar a aplicação da transferência de aprendizado em jogos do Atari 2600 que possuam características semelhantes. Para isso, utilizamos o algoritmo *Neuro Evolution of Augmented Topologies* (NEAT) e avaliamos sua capacidade de aprendizagem em dois jogos diferentes, Breakout e Tennis. Apresentamos duas modelagens distintas, um modelo com parâmetros normalizados e o outro não, para representar os jogos e examinamos suas vantagens e desvantagens. A avaliação levou em conta a taxa de aptidão obtida nos testes, realizada em dois cenários: com a técnica de transferência de aprendizado, tendo como base o jogo Pong, e sem a aplicação dessa técnica. Os resultados obtidos mostraram que no modelo com parâmetros normalizados é possível extrair o aprendizado de um jogo e aplicá-lo como base para solucionar outro, na maioria dos casos testados. Além disso, na aprendizagem por transferência houve ganho na pontuação inicial de aptidão com um menor número de gerações. Isso sugere que a transferência de aprendizado pode ser uma estratégia útil para acelerar o processo de aprendizagem em jogos similares, o que pode ser especialmente útil em contextos onde o tempo é um fator crítico, como na indústria de jogos. Ademais, esses resultados podem contribuir para o desenvolvimento de algoritmos de aprendizado de máquina capazes de generalizar melhor em ambientes com características semelhantes aos jogos estudados.

Palavras-chave: Inteligência Artificial. Transferência de Aprendizado. Algoritmos Genéticos. NEAT. Jogos Eletrônicos.

ABSTRACT

This study aims to explore the application of transfer learning in Atari 2600 games with similar characteristics. For this, we used the Neuro Evolution of Augmented Topologies (NEAT) algorithm and evaluated its learning ability in two different games, Breakout and Tennis. We present two distinct models, one model with normalized parameters and the other not, to represent the games and examine their advantages and disadvantages. The evaluation took into account the fitness rate obtained in the tests, carried out in two scenarios: with the transfer learning technique, based on the Pong game, and without the application of this technique. The results obtained showed that in the model with normalized parameters it is possible to extract the learning from a game and apply it as a basis to solve another one, in most of the cases tested. In addition, in transfer learning, there was a gain in the initial fitness score with a smaller number of generations. This suggests that learning transfer can be a useful strategy to accelerate the learning process in similar games, which can be especially useful in contexts where time is a critical factor, as in the games industry. Furthermore, these results can contribute to the development of machine learning algorithms capable of better generalizing in environments with characteristics similar to the studied games.

Keywords: Artificial Intelligence. Learning Transfer. Genetic Algorithms. NEAT. Electronic Games.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

GPU	*Graphics Processing Units*
IA	Inteligência Artificial
NEAT	*NeuroEvolution of Augmenting Topologies*
RNA	Redes Neurais Artificiais
AM	Aprendizado de Máquina
AR	Aprendizado por Reforço
AG	Algoritimo Genético
TA	Transferência de Aprendizado
ALE	*Arcade Learning Environment*
CE	Computação Evolutiva
API	*Application Programming Interface*

LISTA DE SÍMBOLOS

α	Quantidade total de frames da bola em jogo
β	Quantidade de bolas rebatidas
γ	Quantidade de blocos destruídos
ω	Player score
τ	Tempo alinhado

SUMÁRIO

1 INTRODUÇÃO

A evolução da velocidade dos computadores, a disponibilidade e escala reduzida das infraestruturas e a redução do custo das GPUs (*Graphics Processing Units*) permitiram grandes avanços na área de IA, com novas técnicas e tecnologias, se tornando um dos assuntos mais populares da atualidade. Diversas técnicas que geralmente eram feitas em laboratórios especializados, agora podem ser realizadas no computador pessoal. Esta nova realidade democratizou o acesso a um mundo antes restrito a poucos (Hodjat, 2018). Tais avanços estão presentes em vários setores, incluindo tecnologia, saúde, finanças, vendas, marketing, segurança, entre outros. Algumas das aplicações mais comuns incluem reconhecimento de voz e imagem, processamento de linguagem natural, *chatbots* e assistentes virtuais, automação de tarefas, diagnóstico médico assistido, detecção de fraudes e risco financeiro.

Junto com os avanços, surgiram também desafios no campo de Aprendizado de Máquina. Dentre eles está a resolução de alguns problemas complexos que levam muito tempo para serem resolvidos por técnicas clássicas, que tentam resolver um problema de cada vez. Além disso, a falta de informações suficientes sobre o problema pode dificultar a obtenção da solução exata. Uma solução encontrada é o Aprendizado por Transferência (ou Transferência de Aprendizado), que consiste em transferir o conhecimento de tarefas anteriores para uma nova tarefa de destino com alguns dados de treinamento, enquanto as técnicas tradicionais de aprendizado de máquina tentam aprender cada tarefa do zero (Sinno Jialin Pan, 2010).

Além disso, surgiram as heurísticas que têm como objetivo encontrar soluções aproximadas para determinado problema, em situações em que não é possível ou viável encontrar a solução ótima. Trata-se de simplificar um problema complexo dividindo-o por pequenas partes, de resolução mais fácil, procurando através das respostas de cada uma a possibilidade de chegar à resposta do problema principal. No entanto, em algumas situações, as heurísticas podem levar a soluções sub-ótimas ou inaceitáveis, e é necessário avaliar cuidadosamente o seu uso. Uma derivação das heurísticas são as meta-heurísticas, que se inspiram na natureza, operando através de repetidas tentativas, utilizando um ou mais agentes (como cromossomos em Algoritmos Genéticos). Geralmente usam mecanismo de competição-cooperação e buscam resolver o problema de forma genérica, como problemas de otimização, classificação, agrupamentos, etc., onde não se conhece algoritmo eficiente.

Por exemplo, o algoritmo NEAT (*NeuroEvolution of Augmenting Topologies*) é uma técnica de otimização para evoluir redes neurais artificiais. Ele permite que a estrutura da rede evolua ao longo do tempo, adicionando e removendo conexões e neurônios para melhor se ajustar ao problema. O NEAT usa uma abordagem genética para evoluir as redes neurais e uma técnica chamada especiação para incentivar a diversidade genética. Com isso, o algoritmo é capaz de encontrar soluções mais eficientes e adaptáveis para problemas complexos. Algoritmos Genéticos (AG, desenvolvida por Holland (1975)) compõem a Computação Evolucionária, uma área da Inteligência Artificial que engloba um conjunto de métodos computacionais. São métodos inspirados na Teoria da Evolução das Espécies de Charles Darwin para a solução de problemas.

A utilização de jogos eletrônicos é comum para desenvolver e avaliar algoritmos de aprendizagem por reforço, pois os ambientes simulados são mais acessíveis financeiramente do que os ambientes reais para o treinamento e avaliação dos algoritmos. Ademais, os jogos oferecem um ambiente sem interferências externas e com resultados de testes consistentes, ideal para o desenvolvimento de sistemas de inteligência artificial.

Em 1997, o supercomputador DeepBlue da IBM saiu vitorioso em uma partida de xadrez contra o renomado jogador russo Garry Kasparov. Em 2016, o AI Alpha Go derrotou o jogador profissional Lee Sedol em uma partida de Go, uma façanha que antes se pensava ser impossível devido à alta velocidade do jogo, complexidade e vasta gama de probabilidades. Mais recentemente, em 2019, OpenAI Five venceu uma partida de melhor de três contra uma equipe profissional de Dota 2, um jogo altamente competitivo com múltiplos objetivos que exigem cooperação e colaboração entre todos os cinco jogadores. Esses eventos demonstram que a comunidade científica continuará a buscar objetivos cada vez mais complexos no campo da IA em jogos.

Alguns estudos aliaram o uso de algoritmos evolucionários com ambiente de jogos Atari para validação dos experimentos. A. Tupper (2000) avaliou a capacidade de usar neuroevolução para evoluir agentes capazes de jogar jogos de Atari usando representações de estado compactas e revela que redes neurais simples e pequenas podem jogar esses jogos com eficiência . Em seu trabalho, Hausknecht et al. (2014) desenvolveu uma abordagem de neuroevolução para jogar Atari em geral, comparando quatro estratégias evolucionárias diferentes que superaram a pontuação humana em três jogos Atari diferentes - Bowling, Kung Fu Master e Video Pinball.

Ainda, há trabalhos que aplicam a transferência de aprendizado em algoritmos genéticos, como B. Koçer (2010) evidenciou que o algoritmo de transferência genética é uma boa escolha quando você tem dois problemas de otimização semelhantes. Por fim, um estudo com o uso de algoritmos genéticos para aumentar a aprendizagem de transferência para redes de dependência relacional apresentou resultados melhores ou competitivos quando comparado com outro *framework* padrão, no entanto os resultados dessa abordagem geralmente são seguidos por um tempo de execução mais longo do que os linhas de base (compostas por log-verossimilhança condicional (CLL), área sob a curva ROC (AUC ROC) e área sob a curva PR (AUC PR) (Figueiredo, 2021).

2 PROPOSTA E OBJETIVOS

O objetivo geral do trabalho é analisar o desempenho do uso de transferência de aprendizado em redes neuroevolutivas de topologias aumentantes e explorar o impacto da alteração de alguns parâmetros do NEAT como o tamanho da população, a função de ativação e as conexões iniciais da rede, através de um compilado de jogos de Atari 2600 de domínios similares.

Para isso, as seguintes questões serão abordadas:

1. Quais as vantagens e desvantagens da transferência de aprendizado (TA) com o algoritmo NEAT ?
2. A rede treinada a partir da transferência de aprendizado consegue alcançar os objetivos base do jogo ?
3. Comparar a taxa de aptidão obtida pelo agente usando a transferência de aprendizado com um agente que não utiliza a transferência de de aprendizado.

2.1 OBJETIVOS ESPECÍFICOS

Como metas específicas a serem alcançadas para atingir o objetivo geral do projeto, temos:

- Construir um *framework* para realizar a experimentação e disponibilizá-lo publicamente.
- Elencar o(s) domínio(s) dos jogos/cenários do Atari 2600 mais relevantes a serem utilizados.
- Mensurar o desempenho da transferência de aprendizado, levando em consideração a função de aptidão e o tempo de execução do algoritmo, bem como diagnosticar os fatores impactantes no resultado e suas limitações.

O trabalho está organizado da seguinte maneira; o capítulo 3 apresenta o referencial teórico e alguns conceitos base. O capítulo 4 detalha o modelo de estudo comparativo através de experimentos e os desafios encontrados. O capítulo 5 apresenta os resultados, enquanto o capítulo 6 conclui o trabalho.

3 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo apresentará as definições dos conceitos base de aprendizado de máquina, algoritmos genéticos, NEAT e transferência de aprendizado. Em seguida, os trabalhos relacionados à transferência de aprendizado com algoritmos genéticos e o uso do ambiente Atari para validação existentes na literatura.

3.1 APRENDIZADO DE MÁQUINA

Existem três tipos principais de Aprendizagem de Máquina (AM), em inglês *Machine Learning (ML)*: aprendizagem supervisionada, aprendizagem não supervisionada e aprendizagem por reforço.

A aprendizagem supervisionada é o tipo de aprendizagem de máquina mais comum, o qual o modelo é treinado com dados rotulados, isto é, dados que incluem tanto as entradas quanto as saídas desejadas. O objetivo da aprendizagem supervisionada é encontrar a relação entre as entradas e as saídas e produzir uma função que possa prever as saídas de novas entradas. Exemplos de problemas de aprendizagem supervisionada incluem classificação de imagens, previsão de séries temporais e análise de sentimento.

Na aprendizagem não supervisionada o modelo é treinado com dados não rotulados, isto é, apenas com entradas, sem informações sobre as saídas desejadas. O objetivo da aprendizagem não supervisionada é descobrir padrões ou estruturas nos dados. Exemplos de problemas de aprendizagem não supervisionada incluem agrupamento de dados, redução de dimensionalidade e detecção de anomalias.

Já a aprendizagem por reforço é o tipo de aprendizagem de máquina em que o modelo é treinado através da experimentação e da recepção de recompensas ou punições. O modelo aprende a maximizar a recompensa ao longo do tempo através da tentativa e erro. Exemplos de problemas de aprendizagem por reforço incluem jogos, robótica e controle de sistemas.

3.2 REDES NEURAIS ARTIFICIAIS

Uma Rede Neural Artificial (RNA), do inglês *Artificial Neural Networks (ANN)*, (Figura 3.1) é um sistema paralelo e distribuído, composto de unidades de processamento triviais (neurônios artificiais) alocados em uma ou diversas camadas interconectadas por um grande número de conexões com o objetivo de calcular funções matemáticas. Estas ligações normalmente estão integradas a pesos, que controlam a influência de cada entrada na saída final da rede. Durante o treinamento, esses pesos são ajustados para minimizar a diferença entre as saídas desejadas da rede e as saídas reais, produzidas a partir dos dados de treinamento.

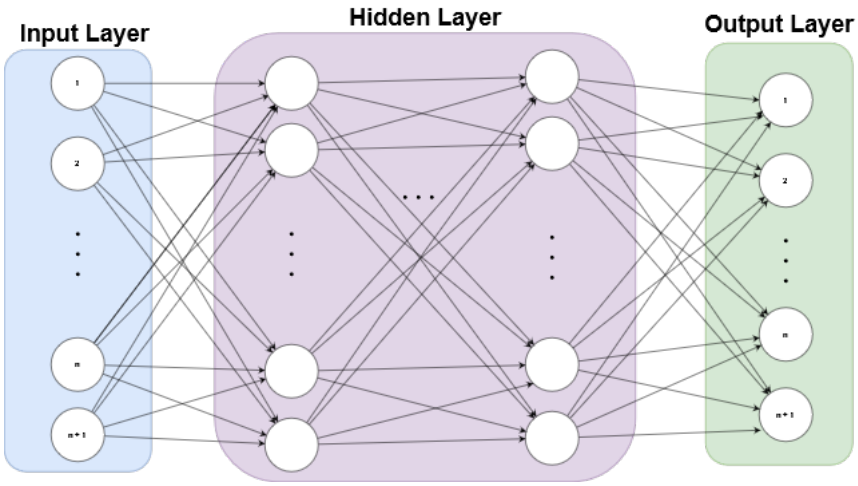


Figura 3.1: Exemplo de Rede Neural
Fonte: Os Autores

Um exemplo de como uma RNA simples pode funcionar para resolver um problema de classificação:

1. Os dados de entrada são fornecidos à rede, geralmente através de uma camada de entrada (*Input Layer*) composta por vários neurônios.

2. Esses neurônios de entrada processam os dados de entrada, aplicando operações matemáticas simples e produzindo saídas.

3. As saídas dos neurônios de entrada são usadas como entrada para os neurônios da camada escondida (*Hidden Layer*), que realizam outra série de operações matemáticas.

4. Esse processo continua através de várias camadas escondidas, até chegar à camada de saída (*Output Layer*), que produz as saídas finais da rede.

5. A saída final da rede é comparada com as saídas desejadas, produzidas a partir dos dados de treinamento.

6. Se a diferença entre a saída desejada e a saída real da rede for suficientemente grande, os pesos das conexões entre os neurônios são ajustados para minimizar essa diferença.

7. Este processo é repetido várias vezes com os mesmos dados de treinamento, ajustando os pesos a cada iteração, até que a diferença entre a saída desejada e a saída real da rede seja satisfatória.

8. Uma vez que a rede é treinada, ela pode ser usada para classificar novos dados, processando-os da mesma forma que os dados de treinamento e produzindo saídas.

As RNAs são atrativas devido ao seu poder de aprendizagem através de exemplos e da generalização da informação obtida. A generalização está ligada à aptidão da rede em aprender por intermédio de um conjunto de amostras e, em seguida, apresentar respostas para informações não apresentadas anteriormente. Além disso, outra propriedade que faz das RNAs importantes é a eficácia de se auto-organizar e a possibilidade de processamento temporal (de Pádua Braga et al., 2000).

3.3 ALGORITMOS GENÉTICOS

Antes de definir o que são algoritmos genéticos e como funcionam, é importante contextualizar sua origem e relação com a computação evolutiva.

Computação Evolucionária (CE) é uma área da Inteligência Artificial que propõe uma nova abordagem para resolução de problemas baseada na Seleção Natural de Darwin (1859). A CE consiste em técnicas de busca e otimização inspiradas na evolução natural das espécies, criando uma população de indivíduos que competem pela sobrevivência. As técnicas incluem Programação Evolucionária, Estratégias Evolucionárias, Algoritmos Genéticos e Programação Genética, sendo cada vez mais utilizadas pela comunidade de IA para modelar inteligência computacional (Barreto, 1997). Algoritmos Genéticos (AG) e Programação Genética (PG) são as duas principais frentes de pesquisa em CE, concebidos por Holland (1975) com o objetivo de estudar a adaptação e seleção natural na natureza e incorporar esses conceitos aos computadores (Mitchell, 1997). Os AGs têm aplicações em diversas áreas científicas, incluindo otimização de soluções, aprendizado de máquina, estratégias matemáticas, análise econômica, engenharia e biologia (Mitchell, 1997).

Um **Algoritmo Genético (AG)** tem como objetivo a otimização e a busca da melhor solução para uma vasta série de problemas. Seu funcionamento simula o processo de evolução natural, que permitiu o surgimento de uma grande variedade de seres vivos. O ponto fundamental é avaliar a capacidade de uma certa espécie em se adequar ao seu ambiente (Kramer, 2017).

Esses algoritmos imitam o processo natural de sobrevivência e reprodução das populações, que é fundamental para a sua evolução. Na natureza, os indivíduos de uma população competem uns com os outros para sobreviverem, buscando recursos como alimentos ou se reproduzindo. Aqueles que são mais aptos terão mais descendentes, enquanto aqueles menos aptos terão menos. Para implementar um AG, é preciso:

- Ter representações das possíveis soluções do problema no formato de um código genético;
- Ter uma população inicial com diversidade o suficiente para permitir a combinação de características e produzir novas soluções;
- Ter um método para avaliar a qualidade de uma solução potencial;
- Ter um processo de combinação de soluções para gerar novos indivíduos na população;
- Ter um critério para escolher as soluções que permanecerão na população ou serão retiradas;
- E ter um procedimento para introduzir alterações periodicamente na população para manter sua diversidade e permitir soluções inovadoras.

Assim, os Algoritmos Genéticos funcionam por tratar soluções de problemas como indivíduos de uma população, e essa população evolui a cada geração. É necessário construir um modelo de evolução para que os indivíduos representem soluções. O processo de execução dos algoritmos segue os seguintes passos: escolher uma população inicial, avaliar todos os indivíduos com base em uma função de aptidão, selecionar os indivíduos de melhor qualidade como base para a criação de uma nova geração, obter a nova geração aplicando operações de mistura de características nos indivíduos selecionados e, por fim, repetir esses passos até encontrar uma solução aceitável ou até o número de passos ser alcançado.

A seguir, serão descritos em mais detalhes os principais componentes apresentados na figura 3.2.

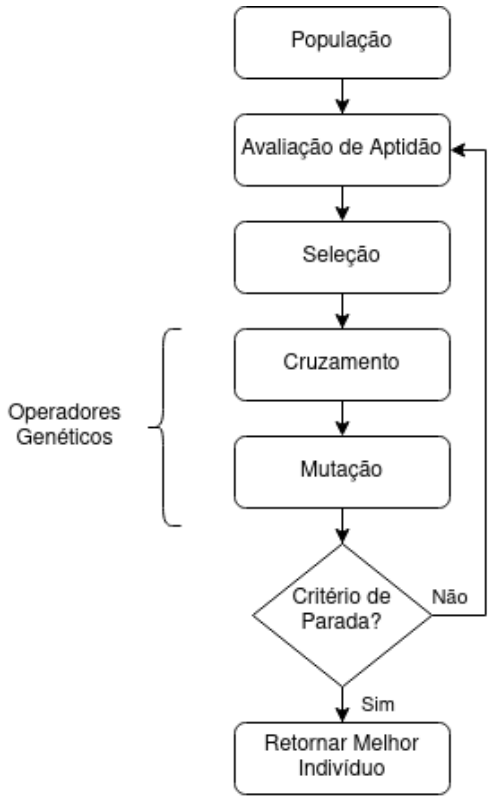


Figura 3.2: Estrutura Básica de Algoritmo Genético
Fonte: Os Autores

3.3.1 População

A população é o conjunto de indivíduos considerados como solução e é usada para criar um novo conjunto de indivíduos para análise. O tamanho da população pode afetar o desempenho global e a eficiência dos algoritmos genéticos. Populações muito pequenas podem perder a diversidade necessária para convergir a uma solução satisfatória, enquanto populações muito grandes podem afetar negativamente a eficiência do algoritmo devido à demora na avaliação da função de aptidão e ao uso de mais recursos computacionais.

A representação adequada de problemas para a utilização de algoritmos genéticos como ferramentas de solução é de grande importância. Uma das formas mais comuns de representação é através de sequências de bits para cada atributo do problema, que são concatenadas para formar o indivíduo. No entanto, outras formas de codificação usando o próprio alfabeto do atributo podem ser utilizadas. A escolha da forma mais apropriada está diretamente relacionada ao tipo de problema que está sendo solucionado.

3.3.2 Avaliação de Aptidão (*Fitness*)

A *Fitness* tem como objetivo atribuir o valor de aptidão de cada indivíduo da população. Essa função é fundamental para avaliar o quão próximo o indivíduo está da solução ideal e garantir que as soluções adequadas sejam identificadas corretamente. É essencial que a função de avaliação seja precisa e capaz de distinguir corretamente as soluções boas das ruins, evitando

que soluções promissoras sejam descartadas e que recursos computacionais sejam gastos desnecessariamente.

3.3.3 Seleção

Existem várias formas de selecionar os indivíduos para aplicar os operadores genéticos, incluindo as seleções por Roleta e por Torneio. No método de seleção por Roleta, cada indivíduo é representado em uma roleta de acordo com seu valor de aptidão, com aqueles de maior aptidão recebendo uma porção maior da roleta. Já na seleção por Torneio, um número de indivíduos são selecionados aleatoriamente e aquele com a melhor aptidão é escolhido para reprodução. Ambos os métodos são utilizados para garantir a diversidade da população e aumentar a chance de encontrar soluções melhores.

3.3.4 Operadores Genéticos

Os operadores genéticos são responsáveis por transformar a população de uma geração em outra, permitindo a busca por resultados satisfatórios em algoritmo genético. A diversidade e a adaptação das características da população são mantidas pelos operadores genéticos, sendo que o cruzamento e a mutação são os mais importantes. O cruzamento consiste em combinar características de dois indivíduos, enquanto a mutação modifica aleatoriamente algumas características do indivíduo, mantendo a diversidade genética. Os operadores genéticos são fundamentais para garantir que a população evolua ao longo das gerações e alcance soluções melhores.

3.3.4.1 Cruzamento (Crossover)

O operador genético de cruzamento é considerado o mais importante. Ele é utilizado para criar novos indivíduos combinando características de dois indivíduos "pais". Este operador é importante para diversificar a população e manter características de adaptação adquiridas pelas gerações anteriores. Existem vários tipos de cruzamento, incluindo o cruzamento em um ponto e o cruzamento em dois pontos, que são ilustrados nas figuras 3.3 e 3.4.

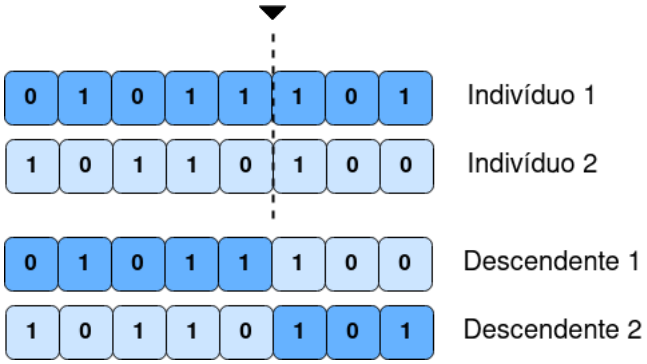


Figura 3.3: Cruzamento de 1 Ponto
Fonte: Os Autores

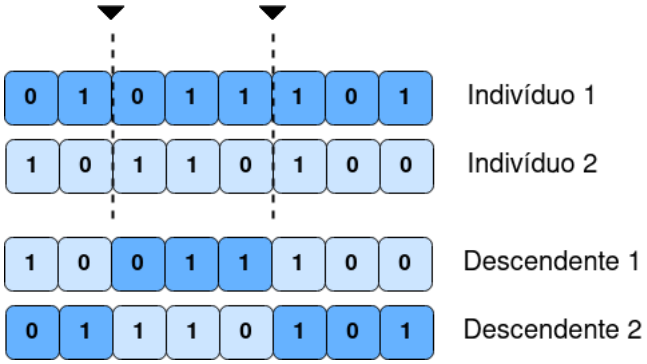


Figura 3.4: Cruzamento de 2 Pontos
Fonte: Os Autores

O cruzamento com um ponto é caracterizado pela seleção de um ponto de corte aleatório no cromossomo, e cada um dos dois descendentes recebe informações genéticas de ambos os pais. Já com o cruzamento em dois pontos, um dos descendentes recebe a parte central de um dos pais e as partes extremas do outro pai, e o outro descendente recebe o contrário.

3.3.4.2 Mutação

A mutação é uma operação que modifica aleatoriamente uma característica do indivíduo. A mutação é importante para criar novos valores de características que não existiam ou apareciam em pequena quantidade na população, mantendo assim a diversidade genética. O operador de mutação é aplicado com uma taxa de mutação geralmente pequena para evitar grandes alterações na população.

3.3.5 Geração

A cada etapa, uma nova geração de indivíduos é produzida a partir da população anterior. Essa nova coleção de indivíduos é conhecida como "Geração". É por meio da geração de um grande número de gerações que se torna possível obter resultados com os Algoritmos Genéticos.

3.4 NEAT

O algoritmo NEAT (*NeuroEvolution of Augmenting Topologies*) é um algoritmo de aprendizado de máquina desenvolvido por K. Stanley (2002) que combina conceitos de algoritmos genéticos com a ideia de que as redes neurais podem ser incrementadas ao longo do tempo, adicionando novos neurônios e conexões. O objetivo do NEAT é evoluir redes neurais complexas capazes de resolver problemas difíceis.

O NEAT é baseado em duas ideias principais: a primeira é que as redes neurais evoluem melhor quando a topologia (ou seja, a estrutura) da rede pode mudar ao longo do tempo; e a segunda é que é importante manter a diversidade genética na população de redes neurais, de modo que novas soluções possam ser exploradas.

A primeira ideia é implementada através da utilização de um algoritmo de “*crossover*” genético, e a segunda através do uso de um mecanismo de “espécies”. Além destas, o NEAT inclui vários outros conceitos importantes, como a codificação genética, mutação, e estrutura mínima.

3.4.1 Codificação Genética

A codificação genética no NEAT é feita através da representação de uma rede neural em um genoma. O genoma é composto de duas lista de genes, uma de nós e outra de conexões. Cada nó possui sua classificação, entre entrada, saída e oculto. Cada conexão tem um número de inovação, que indica quando ele foi introduzido na rede, um peso associado, e informações sobre a origem e o destino da conexão. O número de inovação é um identificador único para cada ligação e permite a comparação as redes, verificando a sua semelhança. Se as conexões em duas redes distintas compartilham o mesmo número de inovação, podemos garantir que essas conexões são estruturalmente idênticas.

A Figura 3.5 mostra o exemplo de um genótipo que produz um fenótipo. Existem 3 nós de entrada, um oculto e um nó de saída, e sete definições de conexão, uma das quais é recorrente. O segundo gene está desabilitado, então a conexão que ele especifica (entre os nós 2 e 4) não é expressa no fenótipo.

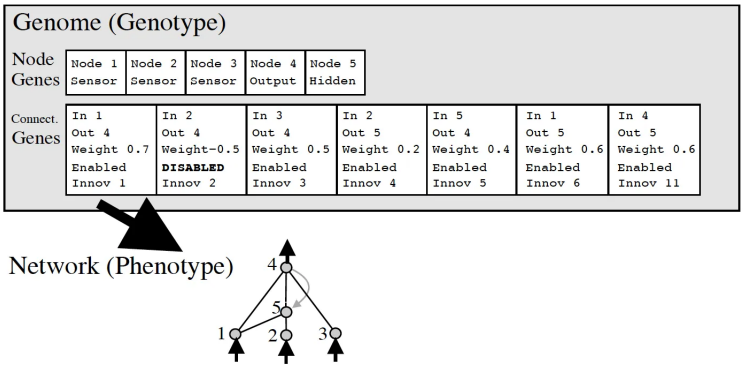


Figura 3.5: Exemplo de mapeamento de genótipo para fenótipo no NEAT
Fonte: (K. Stanley, 2002)

3.4.2 Mutação

A mutação é um processo importante no NEAT que permite a introdução de novas características na rede neural. Existem três tipos de mutação no NEAT: adicionar uma nova conexão, adicionar um novo neurônio, e alterar o peso de uma conexão existente. A mutação é controlada por um conjunto de parâmetros, que afetam a probabilidade de cada tipo de mutação ocorrer.

Além disso, o NEAT usa uma técnica chamada “inovação” para rastrear as mudanças evolutivas na estrutura da rede. Cada nova conexão ou neurônio adicionado a uma rede é considerado uma inovação única e recebe um número de inovação. Isso permite que o NEAT determine quais topologias são mais antigas e quais são mais recentes, o que pode ser útil para avaliar a complexidade da rede.

A seguir, a Figura 3.6 ilustra dois tipos de mutação estrutural no NEAT. Ambos os tipos, adicionando uma conexão e adicionando um nó, são representados com os genes de conexão de uma rede mostrados acima de seus fenótipos. O número superior em cada genoma é o número de inovação desse gene. Os números de inovação são marcadores históricos que identificam o ancestral histórico original de cada gene. Novos genes recebem novos números cada vez mais altos. Ao adicionar uma conexão, um único novo gene de conexão é adicionado ao final do genoma e recebe o próximo número de inovação disponível. Ao adicionar um novo nó, o gene de conexão que está sendo dividido é desativado e dois novos genes de conexão são adicionados.

O novo nó está entre as duas novas conexões. Um novo gene de nodo (não representado) que representa esse novo nodo também é adicionado ao genoma.

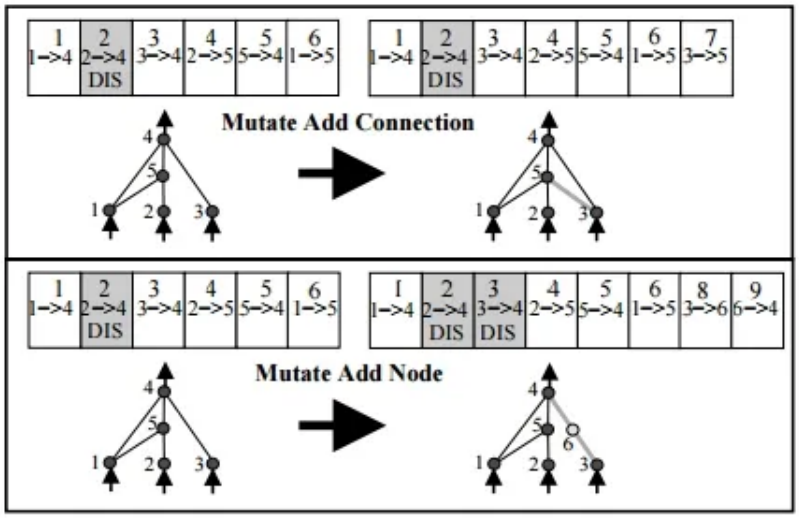


Figura 3.6: Dois tipos de mutação estrutural do NEAT
Fonte: (K. Stanley, 2002)

3.4.3 Crossover

Uma das ideias principais do NEAT é que as redes neurais evoluem melhor quando a topologia (ou seja, a estrutura) da rede pode mudar ao longo do tempo. Isso é implementado através da utilização de um algoritmo de “crossover” genético, que permite a criação de novas topologias de rede a partir da combinação de duas redes existentes. Isso significa que o NEAT pode evoluir redes neurais com diferentes topologias, incluindo aquelas com camadas ocultas, conexões recorrentes e estruturas mais complexas.

As convenções concorrentes no algoritmo NEAT são um mecanismo que promove a competição saudável entre espécies de redes neurais durante a evolução. Essas convenções incluem a introdução de um sistema de pontuação baseado no desempenho relativo de uma espécie em relação às outras espécies, e uma medida de compatibilidade genética, que agrupa as redes neurais com genomas mais semelhantes na mesma espécie. A competição entre espécies é importante para incentivar a evolução de soluções mais inovadoras e eficientes para problemas de aprendizado de máquina e inteligência artificial.

O problema das convenções concorrentes é ilustrado na Figura 3.7. As duas redes calculam exatamente a mesma função, embora suas unidades ocultas apareçam em uma ordem diferente e sejam representadas por cromossomos diferentes, tornando-as incompatíveis para cruzamento.

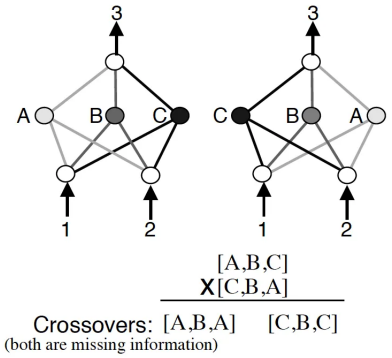


Figura 3.7: Problema das convenções concorrentes do NEAT
Fonte: (K. Stanley, 2002)

Para contornar esse problema, os números de inovação são fundamentais. Ele permite alinhar as redes umas sobre as outras e descobrir quais conexões desempenham o mesmo papel. Em resumo, o funcionamento consiste em:

1. Selecionar um pai dominante. Este é o pai que especificará a estrutura da rede filha.
2. Encontrar todas as conexões compartilhadas por ambos os pais. Com o número de inovação é possível pegar cada conexão cujo número de inovação apareça em ambas as redes principais.
3. Para qualquer conexão compartilhada por ambos os pais, atribui-se aleatoriamente ao filho uma das conexões. Isso garante que os pesos de qualquer conexão compartilhada na rede filha venham aleatoriamente de qualquer um dos pais.
4. Finalmente, para quaisquer conexões não compartilhadas por ambas as redes, o filho as herda do pai dominante.

A Figura 3.8 a seguir ilustra o processo.

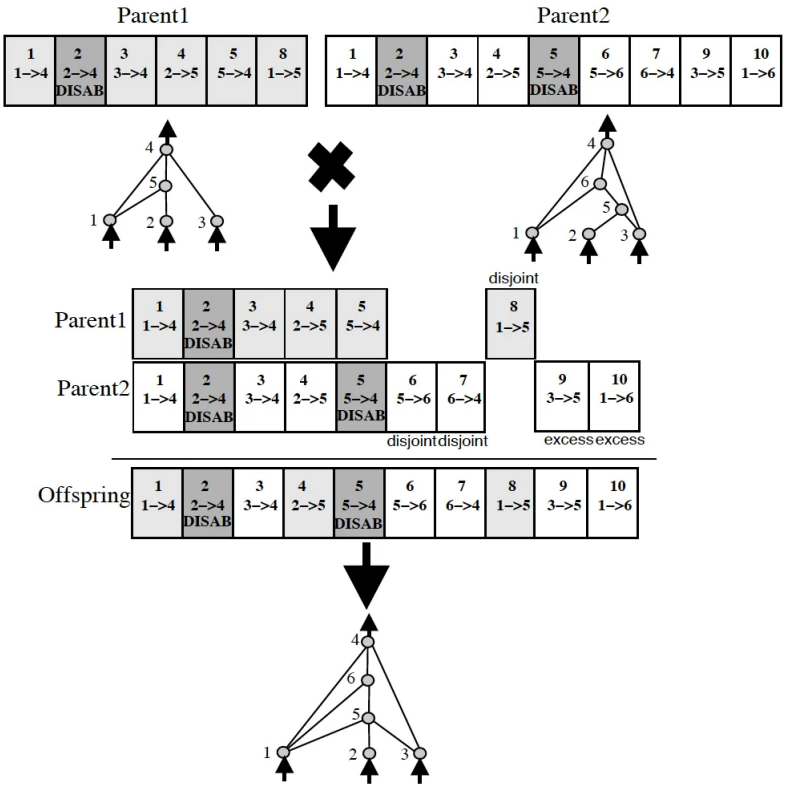


Figura 3.8: Combinação de genomas para diferentes topologias de rede usando números de inovação
Fonte: (K. Stanley, 2002)

3.4.4 Especiação

A especiação é um mecanismo importante no NEAT para manter a diversidade genética na população de redes neurais. O algoritmo agrupa as redes neurais em diferentes espécies com base em suas características genéticas, como o número de inovações e a compatibilidade genética. A especiação permite que o NEAT evolua diferentes tipos de redes neurais, mantendo a diversidade genética.

3.4.5 Estrutura Mínima

Por fim, o NEAT tem o conceito de mínima estrutura, que ajuda a manter a complexidade da rede sob controle. A ideia é que uma rede neural deve ser a menor possível para realizar a tarefa desejada. Isso é conseguido através da atribuição de uma penalidade para redes neurais maiores, o que encoraja o algoritmo a encontrar soluções mais simples e elegantes.

Em resumo, o algoritmo genético NEAT é uma abordagem poderosa para o treinamento de redes neurais artificiais que inclui várias ideias inovadoras, como codificação genética, mutação, *crossover*, especiação e mínima estrutura. Essas ideias permitem a criação de redes neurais mais complexas e eficientes para resolver problemas de aprendizado de máquina e inteligência artificial.

3.5 TRANSFERÊNCIA DE APRENDIZADO

Transferência de Aprendizado (TA) (do inglês *Transfer Learning*) é uma técnica de aprendizado de máquina que tem como objetivo reaproveitar o conhecimento de uma rede

previamente treinada na resolução de um problema similar. Essa é uma técnica que se assemelha na forma que seres humanos aprendem, que consiste no uso de conhecimentos adquiridos anteriormente na resolução de novos problemas.

Dinh T. T. H (2015), em seu estudo, estabeleceu que para realizar a TA precisamos de pelo menos duas tarefas, como mostra a Figura 3.9.

1. **A tarefa de origem**: que vai ser responsável para treinar a rede com o conhecimento base.

2. **A tarefa de destino**: que é o problema que queremos resolver. Por definição, a tarefa de origem é uma rede treinada para resolver um subproblema utilizando técnicas de ML. A tarefa de destino utiliza a rede da tarefa de origem já treinada para a resolução de um problema específico.

Por exemplo, como tarefa de origem, treinamos uma rede a partir do zero para identificar animais quadrúpedes em geral. Como tarefa de destino, utilizamos essa rede pré treinada e especificamos o treinamento dela para identificar cachorros. Dessa forma, conseguimos aplicar a Transferência de Aprendizado na detecção de imagens de cachorros a partir das de animais quadrúpedes.

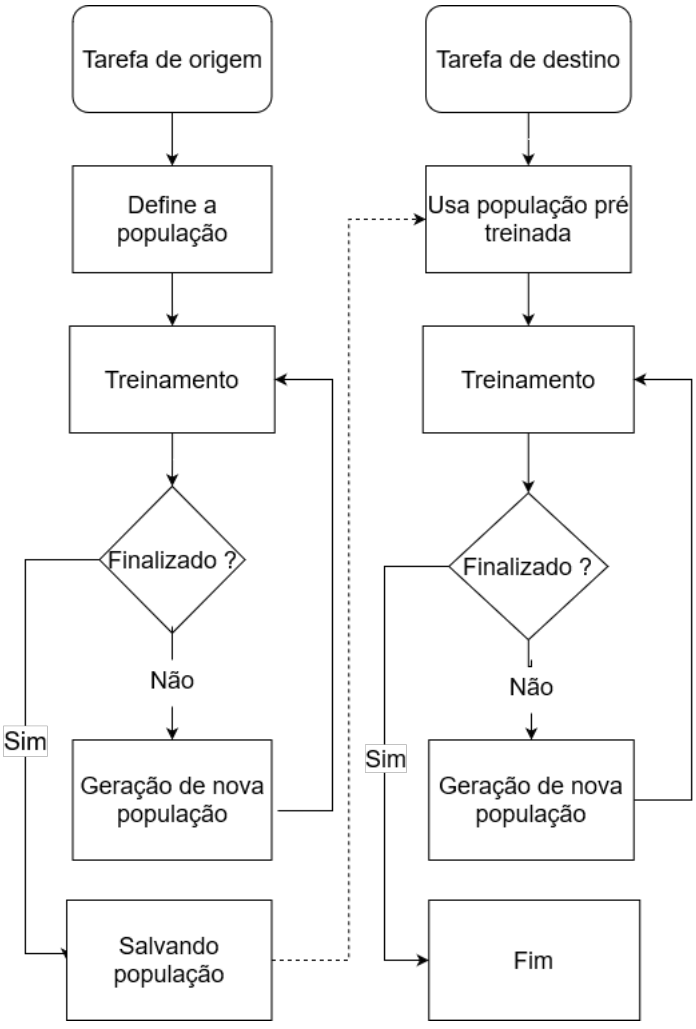


Figura 3.9: Exemplo de fluxo de Transferência de Aprendizado
Fonte: Os Autores

Com os parâmetros definidos corretamente, a utilização da TA permite que modelos já treinados auxiliem na resolução de tarefas mais complexas ou com dados escassos. Além disso, otimiza o tempo de treinamento, dispensa o treinamento de uma nova rede a cada novo problema a resolver. Essa é uma técnica bastante utilizada em problemas de classificação de imagens na qual a amostragem de dados do problema é baixa (Ravishankar, 2016).

Por fim, para garantir a qualidade final do modelo, o domínio do problema na tarefa de origem precisa estar relacionado com o domínio do problema da tarefa de destino. Se essa condição não for respeitada pode ocorrer o Treinamento Negativo (TN), ou *overfitting*, onde o treinamento da tarefa de destino é prejudicado, tornando o aprendizado mais lento e difícil (Hosna, 2022).

4 EXPERIMENTOS

Este capítulo apresenta as ferramentas utilizadas, explica o funcionamento dos jogos no Atari 2600 e detalha como foram realizados os experimentos de treinamento e Transferência de Aprendizado.

4.1 FERRAMENTAS UTILIZADAS

A biblioteca NEAT-Python é uma implementação da NEAT em Python, que permite ao usuário treinar redes neurais utilizando o algoritmo NEAT. A biblioteca fornece uma API para criar e treinar redes neurais usando evolução genética.

OpenAI Gym (Brockman et al., 2016) é uma biblioteca de ambientes de simulação para treinar algoritmos de aprendizado por reforço. Ele fornece uma interface comum para vários ambientes de simulação, incluindo jogos, robótica e outros domínios, para que os algoritmos de aprendizado por reforço possam ser comparados e testados de forma consistente. Pois construir um novo ambiente é complexo e pequenas variações nos parâmetros ou no sistema de recompensas podem ter um impacto significativo no resultado, tornando difícil a reprodução dos resultados.

Stella é um emulador escrito em C++ para jogos do Atari 2600. Emula com precisão o *hardware* do console, permitindo jogar ROMs exatamente como no *hardware* original. Possui recursos adicionais, incluindo um *debugger*, suporte a *joysticks*, configurações de áudio e salvar/carregar jogos.

Implementado a partir do emulador Stella, o *framework Arcade Learning Environment* (ALE) (Bellemare et al., 2013) fornece uma interface para diversos jogos do Atari 2600. ALE implementa algumas ferramentas que auxiliam no desenvolvimento de agentes inteligentes. Essa ferramenta fornece ao usuário a representação atual do jogo que é utilizada no aprendizado do agente e também as recompensas que são obtidas quando o agente toma alguma ação durante o andamento do jogo.

4.2 SISTEMA DE APRENDIZADO

A Figura 4.1 representa como ocorre as interações entre as ferramentas utilizadas. No sistema proposto, o algoritmo NEAT é responsável pela geração dos agentes inteligentes. Esses indivíduos interagem com o ambiente simulado, disponibilizado pela biblioteca *OpenAI Gym* e, a partir das informações recebidas, tomam alguma ação.

Para os testes, os indivíduos interagem com o simulador até a finalização da partida, que é sinalizado pelo simulador, ou até alcançar um limite máximo de 3000 ações realizadas. Após o treinamento, é avaliado a aptidão desse indivíduo de acordo com parâmetros estabelecidos para o jogo.

Esse processo é repetido para todos os indivíduos da população, e após o treinamento da população, o NEAT é responsável pela evolução, selecionando os melhores indivíduos. Para a realização da transferência de aprendizado, o processo de aprendizado é realizado com no máximo 100 gerações.

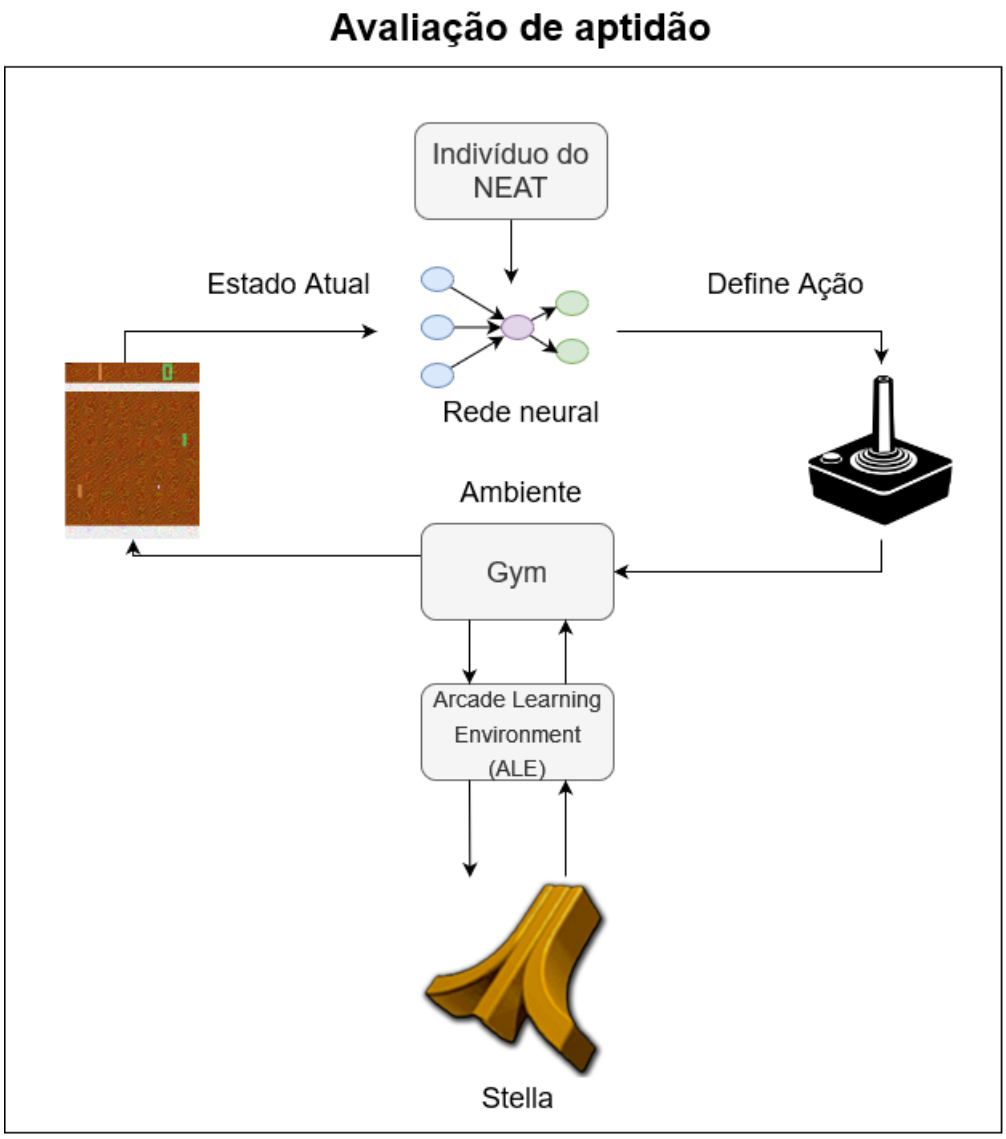


Figura 4.1: Sistema de aprendizado
Fonte: Os Autores

4.3 ATARI 2600

Essa seção apresenta quais são os jogos selecionados para o experimento, o ambiente, as ações possíveis e as recompensas.

4.3.1 Ambientação

Dentre os 59 jogos fornecidos pela biblioteca *Open AI Gym*, foram escolhidos 3 jogos para os experimentos. Para representar esses jogos do Atari 2600, a ferramenta disponibiliza dois tipos de representação do jogo: formato matricial e formato RAM.

No formato matricial, o jogo é representado por uma matriz de pixels coloridos separados em 3 canais de cores, cada canal de cor com 160 pixels de largura por 210 pixels de altura. O jogo é representado por uma combinação de valores entre 0 e 255.

No formato RAM, o jogo é representado pelos 128 bytes de RAM do Jogo, e o Gym representa essa informação como um vetor de 128 posições variando os valores entre 0 a 255. Todos os objetos vistos pela interface do jogo são mapeados para alguma posição nesse vetor.

Nesse experimento os dados são extraídos usando o formato RAM. Através da ferramenta disponibilizada por Anand et al. (2019), os dados são extraídos e convertidos para informações compreensíveis pelo ser humano. O código base utilizado é fornecido por Mila-Iqia (2019), além do simulador Stella para auxiliar na extração dos dados. A partir do simulador, utilizamos o *debugger* (Figura 4.2) para classificar algumas informações da RAM e incrementar a biblioteca de extração de dados.

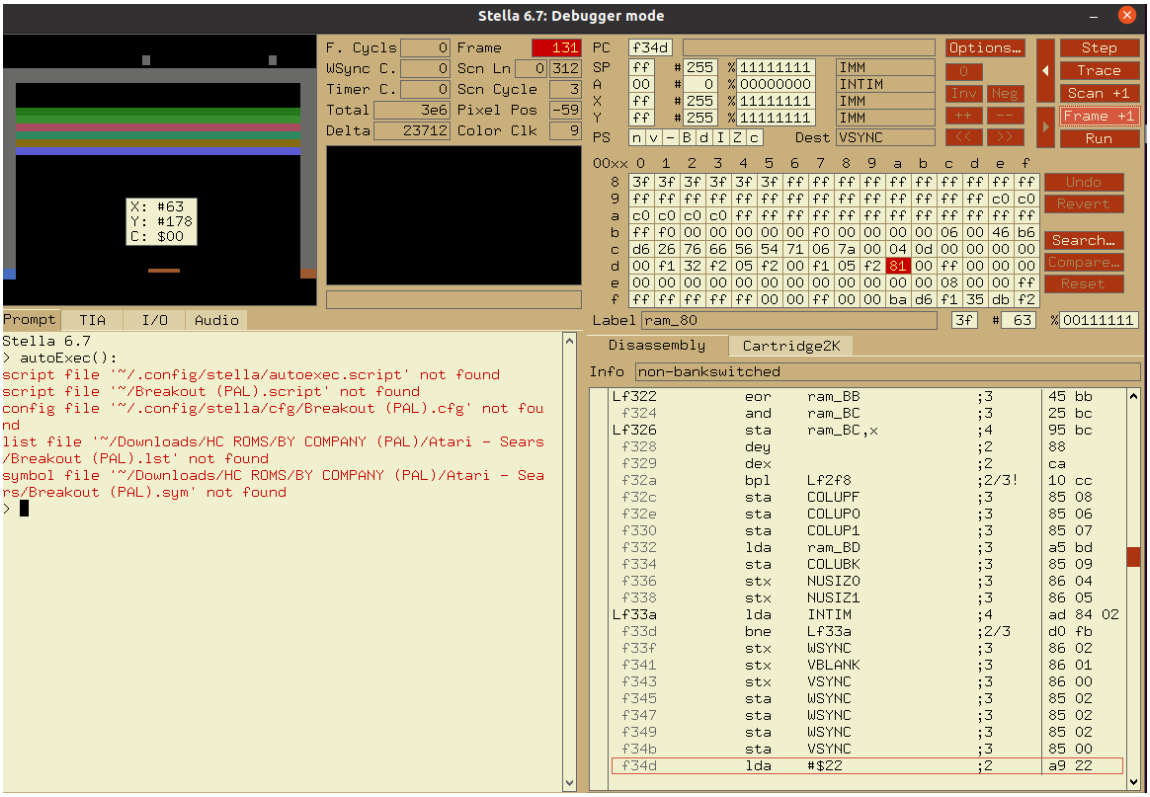


Figura 4.2: Debugger Stella
Fonte: Os Autores

4.3.2 Ações

O Atari 2600 possui um periférico para capturar movimentos nos seus jogos. O *joystick* se move em 8 direções diferentes: cima, cima + esquerda, esquerda, esquerda + baixo, baixo, baixo + direita, direita, cima + direita, com um botão adicional de ação. Em cada jogo, as ações possíveis são um subconjunto das ações disponíveis no *joystick* do Atari 2600. Na biblioteca *Open AI Gym*, as ações dos jogos são associadas a um índice que as representa. As ações disponíveis para cada jogo podem variar de acordo com a escolha do jogo, mas, se necessário, a biblioteca permite expandir o espaço amostral de ações para o máximo possível (representado na Tabela 4.1).

Conjunto de ações possíveis	
Índice	Descrição
0	NOOP
1	FIRE
2	UP
3	RIGHT
4	LEFT
5	DOWN
6	UP + RIGHT
7	UP + LEFT
8	DOWN + RIGHT
9	DOWN + LEFT
10	UP + FIRE
12	RIGHT + FIRE
12	LEFT + FIRE
13	DOWN + FIRE
14	UP + RIGHT + FIRE
15	UP + LEFT + FIRE
16	DOWN + RIGHT +FIRE
17	DOWN + LEFT + FIRE

Tabela 4.1: Espaço amostral das ações no Atari 2600

4.3.3 Jogos Selecionados

Pong, Breakout e Tennis foram escolhidos dentre a diversidade de jogos oferecidos pelo Atari 2600 para serem utilizados como modelo nos experimentos de Transferência de Aprendizado. Esses jogos foram escolhidos devido às suas características similares, tais como os controles de entrada e estratégia de jogabilidade, permitindo a extração e abstração das informações essenciais dos jogos.

4.3.3.1 Pong

No jogo Pong, o jogador é representado por uma raquete (Figura 4.3), podendo se movimentar na vertical e tem como objetivo rebater a bola para o campo adversário. Para pontuar, o jogador é recompensado cada vez que o seu adversário não consegue rebater a bola. O jogo se encerra quando o primeiro jogador alcança vinte e um pontos.

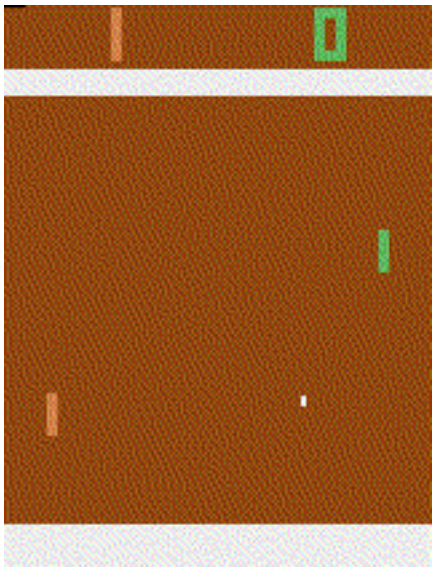


Figura 4.3: Captura da tela do jogo Pong
Fonte: Os Autores

A tabela 4.2 lista todas ações básicas disponibilizadas pela biblioteca ALE do jogo Pong.

Conjunto de ações possíveis	
Índice	Descrição
0	NOOP
1	FIRE
2	RIGHT
3	LEFT
4	RIGHT + FIRE
5	LEFT + FIRE

Tabela 4.2: Conjunto de ações possíveis - Pong

4.3.3.2 Breakout

O objetivo do jogo é destruir todos os tijolos na tela usando uma bola rebatida por uma raquete controlada pelo jogador (Figura 4.4).

No início do jogo, uma fileira de tijolos é exibida na parte superior da tela. A bola é então liberada pela raquete do jogador, que deve controlá-la rebatendo-a de volta para cima para destruir os tijolos. Se a bola passar por baixo da raquete, o jogador perde uma vida. O jogo termina quando todos os tijolos são destruídos ou quando o jogador perde todas as suas vidas.

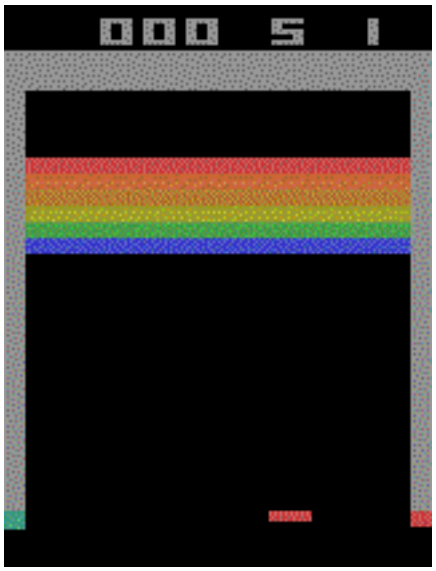


Figura 4.4: Captura da tela do jogo Breakout
Fonte : Os Autores

A tabela 4.3 lista todas ações básicas disponibilizadas pela biblioteca ALE do jogo Breakout.

Conjunto de ações possíveis	
Índice	Descrição
0	NOOP
1	FIRE
2	RIGHT
3	LEFT

Tabela 4.3: Conjunto de ações possíveis - Breakout

4.3.3.3 Tennis

Cada jogador controla uma raquete (Figura 4.5 para rebater uma bola virtual. O objetivo é manter a bola no ar, trocando-a de lado e evitando que ela caia no seu próprio campo. Conforme o andamento do jogo, a velocidade da bola é aumentada.

A pontuação é concedida a cada vez que um jogador não consegue rebater a bola para o campo adversário. A partida se encerra quando um dos jogadores atinge 4 pontos.

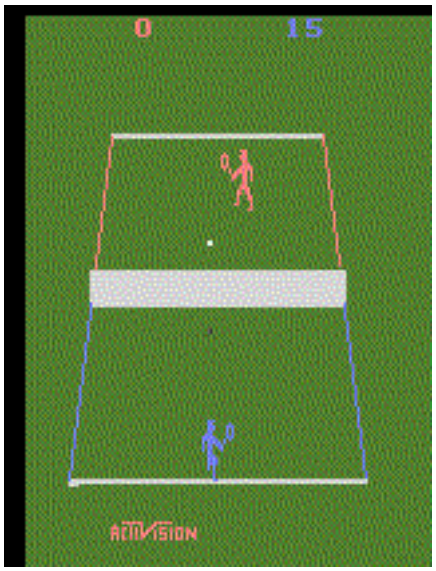


Figura 4.5: Captura da tela do jogo Tennis
Fonte: Os Autores

A tabela 4.4 lista todas ações básicas disponibilizadas pela biblioteca ALE do jogo Tennis.

Conjunto de ações possíveis	
Índice	Descrição
0	NOOP
1	FIRE
2	UP
3	RIGHT
4	LEFT
5	DOWN
6	UP + RIGHT
7	UP + LEFT
8	DOWN + RIGHT
9	DOWN + LEFT
10	UP + FIRE
12	RIGHT + FIRE
12	LEFT + FIRE
13	DOWN + FIRE
14	UP + RIGHT + FIRE
15	UP + LEFT + FIRE
16	DOWN + RIGHT +FIRE
17	DOWN + LEFT + FIRE

Tabela 4.4: Conjunto de ações possíveis - Tennis

4.4 MODELAGEM DA REDE

Essa seção apresenta as configurações utilizadas no NEAT e as arquiteturas das Redes Neurais para a Transferência de Aprendizado em cada jogo.

4.4.1 Transferência de Aprendizado

O objetivo da transferência de aprendizado é reaproveitar o conhecimento adquirido em um problema e aplicá-lo em outros problemas semelhantes, cortando etapas no processo de solução. Neste caso, estamos considerando três jogos, Pong, Breakout e Tennis, para avaliar qual é o conhecimento base que pode ser reaproveitado em todos eles.

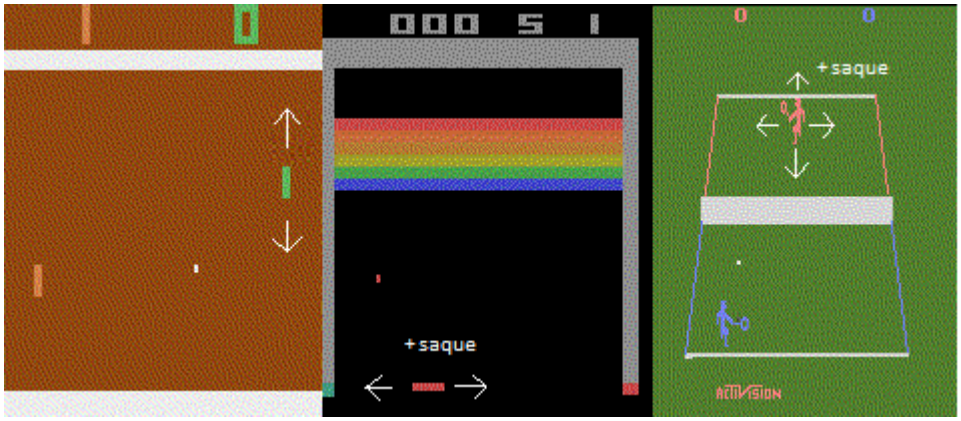


Figura 4.6: Movimentação dos jogos
Fonte: Os Autores

No jogo de Pong, a rede de inteligência artificial deve ser capaz de aprender a controlar a raquete verticalmente, rebatendo a bola e a mantendo o jogo. Além disso, a rede precisa ser capaz de prever a trajetória da bola e se ajustar de acordo, a fim de evitar que ela passe pela raquete.

Já no Breakout, a rede precisa aprender a se mover da esquerda para a direita, garantindo que a bola não ultrapasse os limites da raquete. Quando o jogador perder uma vida, a rede deve ser capaz de atirar a bola de volta ao jogo. É importante que a rede seja capaz de identificar padrões e aprender com suas ações, a fim de melhorar a performance ao longo do tempo.

Em Tennis, a rede de inteligência artificial precisa ser capaz de se mover no espaço bidimensional do campo, movendo-se para a esquerda, direita, cima e baixo. Além disso, o agente deve ser capaz de sacar a bola corretamente nos momentos certos, a fim de manter o jogo e garantir que a bola fique sempre em jogo. É importante que a rede seja capaz de identificar as jogadas do adversário e reagir rapidamente para mantê-lo no jogo.

Analisando o contexto de cada jogo, o aprendizado básico que cada rede neural deve adquirir inicialmente nos treinamentos é a movimentação no campo. O jogo Pong, como mostra a figura 4.6 possui o campo de movimentação mais simples, devido a isso será usado como base para o aprendizado dos outros dois, como representado na figura 3.9.

Na transferência de aprendizado, é importante que as entradas e saídas da rede neural tenham significado similares tanto no jogo de origem quanto no de destino. Para garantir isso, modelamos a arquitetura da rede de maneira a torná-la coerente em todos os jogos, isto é, garantir que todos os jogos tenham a mesma entrada. Além disso, para assegurar que a rede aprenda as particularidades do jogo de destino, permitimos que ela capture estas características específicas, como por exemplo, a necessidade de sacar no Breakout ou no Tennis. Desta forma, a rede não é limitada apenas pelas ações do seu jogo base.

Para avaliar a eficiência da transferência de aprendizado, testamos duas arquiteturas diferentes de redes neurais em cada jogo. Uma arquitetura baseada em posições absolutas e outra em posições relativas ao objetivo. Em ambas as arquiteturas, realizamos testes com o espaço amostral de ações padrão e também com um espaço amostral expandido.

4.4.2 Configuração do NEAT

A implementação NEAT-Python é altamente configurável e oferece vários parâmetros que podem ser ajustados para melhorar o desempenho do algoritmo. Nesta seção, vamos discutir alguns dos principais parâmetros de configuração [1] que modificamos e são **compartilhados entre todos os treinamentos dos jogos**.

Abaixo, estão descritos a definição dos parâmetros chave para o treinamento e quais são os critérios de escolha dos valores.

1. **Tamanho da população**: esse parâmetro representa o número de indivíduos em cada geração. Para determinar o valor, a escolha deve garantir que exista uma variação de espécies para o treinamento.

2. **Função de ativação**: a função de ativação é responsável por introduzir não-linearidade nas saídas dos neurônios. Algumas funções de ativação, como a função sigmoide, são mais adequadas para problemas de classificação, enquanto outras, como a função ReLU, são mais adequadas para problemas de regressão. A escolha da função de ativação também pode afetar a sensibilidade da rede neural aos valores iniciais dos pesos e bias, bem como a sensibilidade aos gradientes durante o processo de treinamento.

3. **Número de entradas**: para determinar o número de entradas da Rede Neural é preciso considerar a quantidade de informações que serão fornecidas à rede, a fim de representar adequadamente o ambiente em que o agente está inserido.

4. **Número de saídas**: para determinar o valor desse parâmetro, o valor deve ser igual a quantidade de ações disponíveis que o agente pode tomar.

5. **Taxa de mutação**: a taxa de mutação é responsável controlar a frequência em que ocorre as mutações durante o treinamento da Rede Neural NEAT, esse parâmetro deve possuir um valor para que seja permitido, após o treinamento inicial no jogo base, a rede consiga evoluir para realizar a especialização do jogo de destino.

6. **Número de gerações**: define um número máximo de gerações que serão executadas para o treinamento da Rede Neural, para a escolha desse valor, o parâmetro deve ser grande o suficiente para que tenha efeito a transferência de aprendizado e permita a visualização da estagnação máxima que essa rede pode chegar.

4.4.2.1 Exploração dos Parâmetros

A exploração dos parâmetros do algoritmo NEAT é uma parte crucial para obter um desempenho ideal do algoritmo. Isso porque o desempenho pode ser altamente dependente da configuração dos parâmetros. Se os parâmetros forem definidos de forma inadequada, pode haver problemas como convergência lenta, falta de diversidade ou superespecialização.

Dessa forma, com intuito de analisar o impacto dos parâmetros e definir as melhores configurações, realizamos **testes exploratórios** com os parâmetros de tamanho da população, a conexão inicial da rede, o número de nós ocultos da rede e a função de ativação, como listado a seguir na Tabela 4.5.

[1] A relação completa dos parâmetros utilizados se encontra no Apêndice A

Parâmetro	Configurações Testadas
Tamanho da População	50, 150, 300, 600
Número de Nós Ocultos	0, 3, 6, 9
Função de Ativação	SIGMOID, RELU, TANH, CUBE, EXP
Conexão Inicial	FULL_DIRECT, UNCONNECTED

Tabela 4.5: Configurações Exploradas do NEAT

Os resultados obtidos estão disponíveis na seção 5.

4.4.3 Modelo A

Construímos dois modelos para testar a transferência de aprendizado, o modelo A com posição absoluta e o modelo B, com posição relativa (normalizado). A posição absoluta é a localização exata de um objeto em relação a um ponto de referência fixo, usando coordenadas cartesianas, como x e y. A posição absoluta é sempre a mesma, independentemente do ponto de vista ou referência adotado.

Para o treinamento do Modelo A, a arquitetura da rede recebe como entrada as posições absolutas dos pontos de interesse:

Entrada	Valor mínimo	Valor Máximo
Jogador - Posição X	0	160
Jogador - Posição Y	0	210
Bola - Posição X	0	160
Bola - Posição Y	0	210

Tabela 4.6: Conjunto de entradas do Modelo A

De acordo com as entradas definidas na Tabela 4.6, espera-se que a rede possa:

- Entender como funciona a movimentação da bola e do jogador;
- Entender quando a bola está ou não fora do jogo, de acordo com as coordenadas.

4.4.4 Modelo B

Para o treinamento do Modelo B, com parâmetros normalizados, definimos uma arquitetura onde as entradas estão baseadas em uma posição relativa ao agente. A posição relativa é a localização do agente em relação a outros objetos ou a um ponto de referência variável. Por exemplo, podemos dizer que a bola está esquerda ou direita do agente, ou também, a quantidade de *frames* em que a bola está fora jogo.

Entrada	Valor mínimo	Valor Máximo
Bola à esquerda do agente	-1	1
Bola à direita do agente	-1	1
Quantidade de frames em que a bola está fora de jogo	0	3000

Tabela 4.7: Conjunto de entradas do Modelo B

De acordo com as entradas definidas na Tabela 4.7, espera-se que a rede possa:

- Entender para onde deve se movimentar;
- Entender quando a bola está fora de jogo.

4.4.5 Pong

Para treinar a rede base, extraímos informações relevantes do jogo para que sejam utilizadas pela rede neural. Dentre elas, temos as posições absolutas das entidades dos jogos - modelo A (Tabela 4.6), e as posições relativas do agente em relação ao seu objetivo - modelo B (Tabela 4.7), ilustradas na Figura 4.7. Dessa forma, conseguimos reaproveitar o conhecimento adquirido do Pong para os outros jogos.

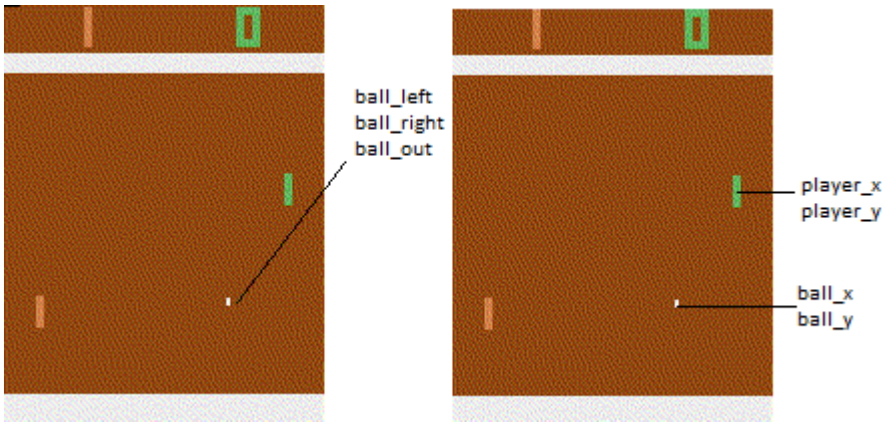


Figura 4.7: Entrada da rede: Modelo B à direita e Modelo A à esquerda - Pong
Fonte: Os Autores

No modelo da solução mais próxima do ideal, o agente deve conseguir movimentar-se pelo ambiente, mantendo a bola em jogo o maior tempo possível, para isso, o jogador deve:

- Sempre estar alinhado com a bola no eixo principal;
- Possuir a pontuação máxima ao final do jogo;
- O jogo deve durar o máximo possível.

Com base nesses itens e em resultados de experimentos iniciais, foi projetada a função *Fitness* 4.1 para o jogo Pong. Os valores reais multiplicados foram definidos arbitrariamente no início e refinados ao longo dos testes. A Tabela 4.8 apresenta os limites máximos e mínimos de cada parâmetro da função *fitness*

$$Fitness = \tau * 0,05 + \omega * 2 + \alpha * 0,01 \tag{4.1}$$

Termo	Valor mínimo	Valor máximo
Tempo alinhado (τ) * 0.05	0	240
Player Score (ω) * 2	0	42
Quantidade total de frames (α) * 0.01	0	30
Fitness Máxima	0	312

Tabela 4.8: Valores máximos e mínimos de cada termo - Pong

O treinamento da rede foi realizado com 30 gerações. Este número foi estabelecido a fim de evitar que o agente se torne especialista em somente movimentar pelo campo, impactando negativamente no reaproveitamento de aprendizado em outro jogo.

4.4.6 Breakout

No treinamento da rede do Breakout, é utilizada a população previamente treinada no ambiente do Pong. Essa população possui o aprendizado base, que consiste em movimentar a raquete em direção a bola, e espera-se que o agente, através de premiação moderada, consiga evoluir aprendendo novas ações, como recolocar a bola em jogo manualmente sempre quando necessário. Com base em resultados de experimentos iniciais, atribuímos pesos para ações distintas para evitar a especialização em uma tarefa específica. Esses pesos são balanceados para que o indivíduo não perca o conhecimento prévio, mas também consiga assimile ações específicas do jogo Breakout. Os modelos de entrada da rede são ilustrados na Figura 4.8.

Para a solução ideal mais próxima, o agente deve possuir as seguintes características:

- Sempre estar alinhado com a bola no eixo principal;
- Atirar a bola para o jogo quando perder uma vida;
- Destruir a maior quantidade de blocos.

Com base nesses itens e em resultados de experimentos iniciais, foi projetada a função *Fitness* 4.2 para o jogo Breakout. Os valores reais multiplicados foram definidos arbitrariamente no início e refinados ao longo dos testes. A Tabela 4.9 apresenta os limites máximos e mínimos de cada parâmetro da função *fitness*

$$Fitness = \tau * 0,03 + \gamma * 0,05 + \alpha * 0,05 \tag{4.2}$$

Termo	Valor minimo	Valor máximo
Tempo alinhado (τ) * 0.03	0	90
Frames de bola em jogo (α) * 0.05	0	150
Quantidade blocos destruidos (γ) * 0.5	0	15
Fitness Máxima	0	255

Tabela 4.9: Valores máximos e mínimos de cada termo - Breakout

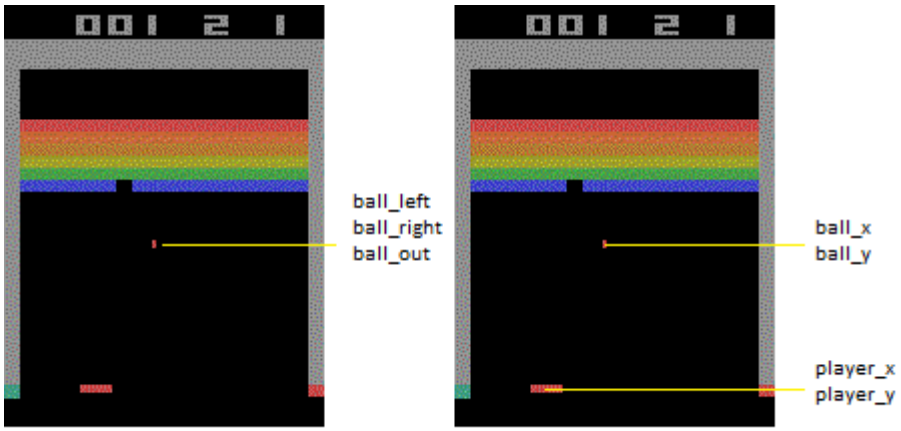


Figura 4.8: Entrada da rede: Modelo B à direita e Modelo A à esquerda - Breakout
Fonte: Os Autores

4.4.7 Tennis

Para a rede do Tennis, utilizando a população do Pong como base, o agente deve aprender a movimentar em todos os eixos do mapa, não só no eixo vertical, e também deve sacar a bola quando for necessário. Os modelos de entrada da rede são ilustrados na Figura 4.8.

Dessa forma, o indivíduo ideal deve possuir as seguintes características:

- Sempre estar alinhado com a bola no eixo principal;
- Aprender a movimentar em todos os eixos do mapa;
- Rebater a bola a maior quantidade de vezes possíveis;
- Manter a bola em jogo, isto é, sacar quando precisar.

Com base nesses itens e em resultados de experimentos iniciais, foi projetada a função *Fitness* 4.3 para o jogo Tennis. Os valores reais multiplicados foram definidos arbitrariamente no início e refinados ao longo dos testes. A Tabela 4.10 apresenta os limites máximos e mínimos de cada parâmetro da função *fitness*

$$Fitness = \tau * 0,05 + \beta * 0,06 + \alpha * 0,01 \tag{4.3}$$

Termo	Valor mínimo	Valor máximo
Tempo alinhado (τ) * 0.05	0	150
Quantidade bolas rebatidas (β) * 0.06	0	180
Quantidade frames de bola em jogo (α) * 0.01	0	30
Fitness Máxima	0	360

Tabela 4.10: Valores máximos e mínimos de cada termo - Tennis

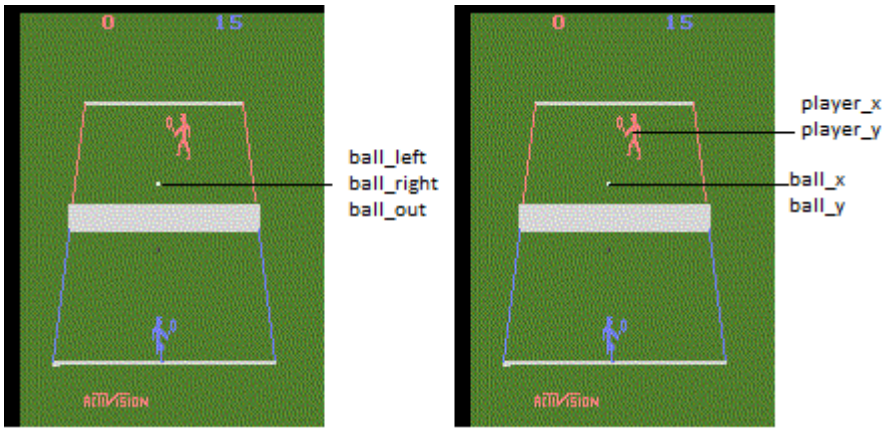


Figura 4.9: Entrada da rede: Modelo B à direita e Modelo A à esquerda - Tennis
Fonte: Os Autores

4.5 DESAFIOS DE IMPLEMENTAÇÃO

A implementação de redes neuroevolutivas para transferência de aprendizado em jogos Atari 2600 apresentou diversos desafios que precisaram ser superados para que o modelo fosse efetivo e produzisse resultados satisfatórios.

Um dos principais desafios que encontramos foi a falta de trabalhos relacionados ao tema. Poucos estudos têm abordado a transferência de aprendizado de redes neuroevolutivas em jogos Atari 2600, abordando os três temas juntos, há apenas estudos isolados ou em pares, o que dificulta a orientação para a construção de um modelo eficiente e confiável.

Além disso, a modelagem e construção das redes neuroevolutivas também é um desafio, já que há poucas referências práticas disponíveis para orientar a escolha dos parâmetros de entrada e a arquitetura da rede.

A representação correta do ambiente do jogo também foi um problema. A utilização das imagens do jogo como entrada tornou o processamento lento e ineficiente. Para solucionar esse problema, optamos por utilizar uma representação dos elementos da cena direto da memória RAM emulada, o que demandou um estudo aprofundado sobre o funcionamento exato da RAM do jogo e do auxílio de ferramentas de *debugger*.

Definir os parâmetros de entrada da rede foi outra dificuldade enfrentada. A escolha dos parâmetros pode impactar significativamente o desempenho da rede, por isso é importante explorar as configurações realizar testes para encontrar a melhor configuração.

Além disso, é complexo modelar uma *fitness* fiel que capture as características importantes do jogo e possa ser utilizada para avaliar o desempenho da rede. Inicialmente a *fitness* era calculada utilizando a pontuação do jogo, mas era ineficaz para a evolução dos agentes. Então o ideal foi definir mais alguns parâmetros importantes que sejam recompensados, como o tempo de jogo, tarefas objetivas do jogo, entre outros.

Por fim, é importante ressaltar que a implementação de redes neuroevolutivas para transferência de aprendizado em jogos Atari 2600 demanda um longo tempo de estudo e exploração do funcionamento e das variáveis do NEAT, bem como o entendimento do problema, o que exige dedicação e paciência por parte do desenvolvedor.

5 RESULTADOS

Inicialmente apresentamos os resultados da exploração dos parâmetros do algoritmo NEAT. Em seguida, os resultados de transferência de aprendizado divididos em três partes: jogos treinados com o modelo A, jogos treinados com modelo B e a comparação entre os dois modelos respectivos. Nos gráficos expostos, os números são resultados de uma execução, da melhor pontuação, entretanto retrata um comportamento típico, baseado em várias execuções anteriores. O tempo de execução dos testes está presente no Apêndice B.

5.1 EXPLORAÇÃO DOS PARÂMETROS

Os testes de exploração dos parâmetros do algoritmo NEAT foram aplicados no jogo *Pong* (escolhido como base). Quando a variação de um dos parâmetros está sendo analisada, os valores padrão dos demais são descritos na Tabela 5.1 a seguir.

Parâmetro	Configuração Padrão
Tamanho da População	150
Número de Nós Ocultos	0
Função de Ativação	RELU
Conexão Inicial	UNCONNECTED

Tabela 5.1: Configuração padrão utilizada nos testes exploratórios do NEAT

5.1.1 Função de Ativação

O primeiro teste foi com foco na análise da função de ativação[1]. O NEAT oferece várias funções de ativação padrão, como *cube, exp, relu, sigmoid e tanh* . Durante a avaliação, observou-se que as funções *cube, exp e relu* apresentaram resultados similares e satisfatórios, com base na Figura 5.1. Por outro lado, a função *sigmoid* foi um pouco melhor que as outras. No entanto, a função *tanh* levou mais tempo para convergir e foi a menos eficaz das cinco funções testadas.

Ao selecionar a função de ativação, foi considerado essencial que a rede sempre escolhesse a melhor ação possível para garantir o melhor desempenho. Embora a função *sigmoid* tenha obtido os melhores resultados, essa função tem a possibilidade de gerar valores repetidos, devido a isso tornou-se inadequada para o objetivo em questão. Por essa razão, o *relu* foi escolhido como a função de ativação.

5.1.2 Conexão Inicial

Os testes dos parâmetros do NEAT também incluíram a avaliação do parâmetro que determina a conexão inicial dos nós da rede no segundo experimento. Os valores desse parâmetro são: totalmente conectado e desconectado. Durante a avaliação, com base na Figura 5.2, o totalmente conectado apresentou um desempenho significativamente melhor do que a sua contraparte, a rede desconectada. Entretanto, ao aplicar a técnica de transferência de aprendizado com a rede inicial do totalmente conectado, os resultados obtidos não foram satisfatórios.

[1] O gráfico de crescimento de cada função de ativação testada pode ser consultado no Apêndice A

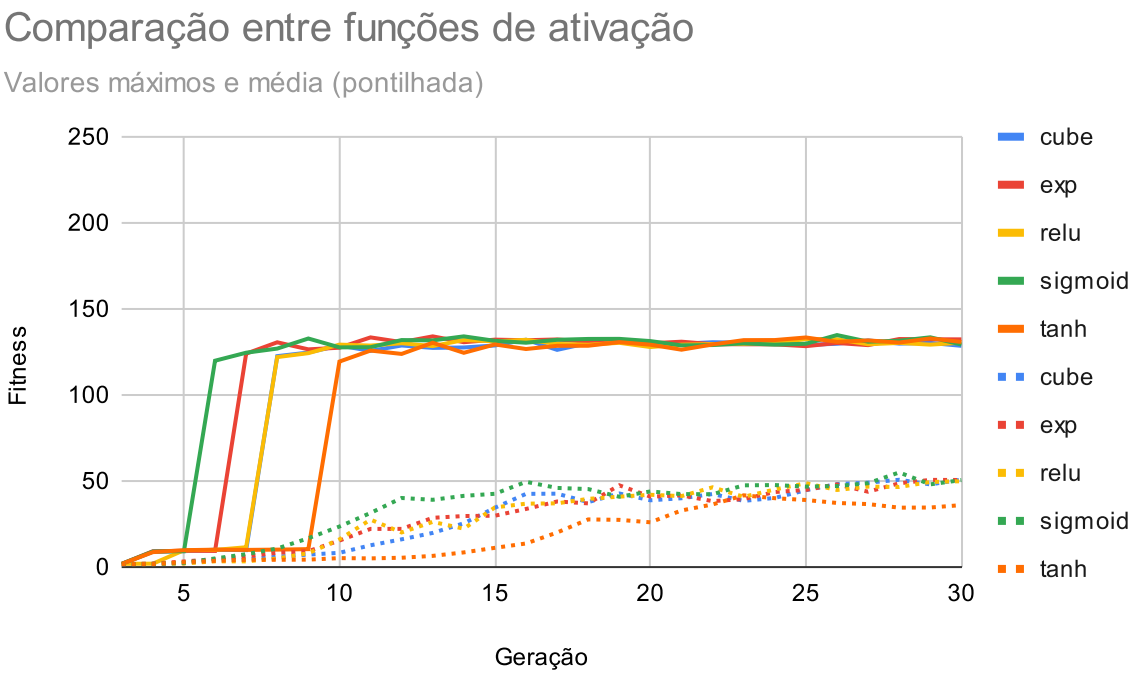


Figura 5.1: Exploração de parâmetros: Função de ativação - Jogo Pong
Fonte: Os Autores

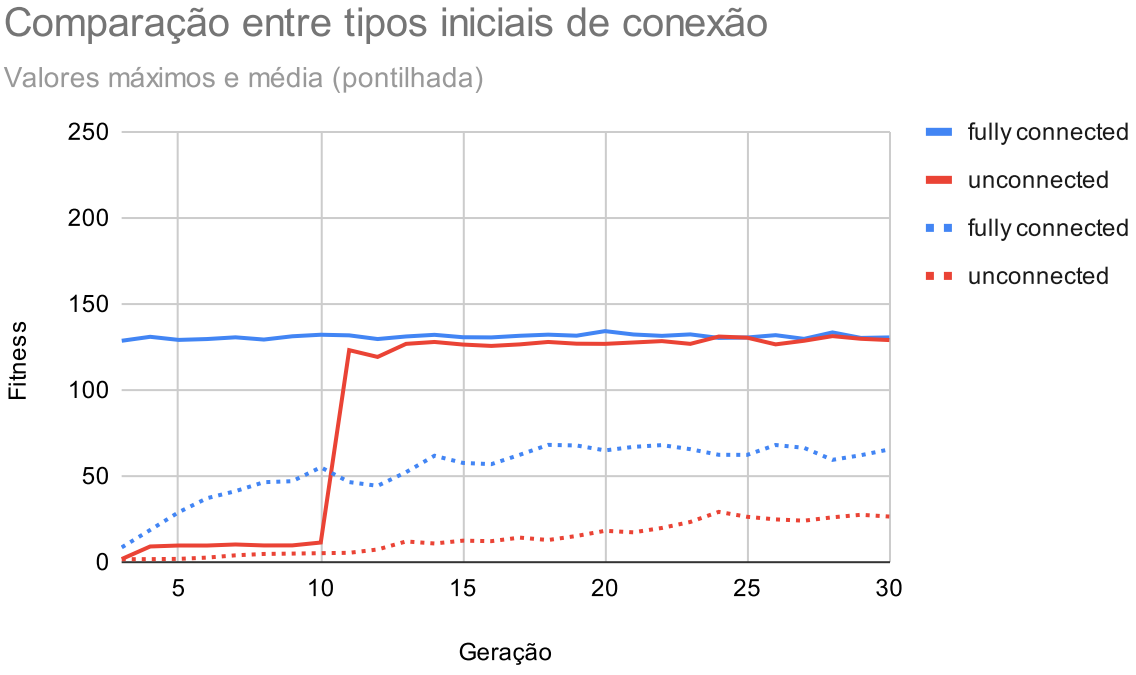


Figura 5.2: Exploração de parâmetros: Conexão inicial da rede - Jogo Pong
Fonte: Os Autores

Apesar do fato de que o desconectado leva mais tempo para convergir, ao aplicar a transferência de aprendizado, a rede desconectada se saiu melhor em relação ao totalmente conectado. Por esse motivo, optou-se por utilizar a conexão inicial desconectada na rede.

5.1.3 Tamanho da População

O tamanho da população foi avaliado como próximo parâmetro. Testes foram realizados com populações de 50, 150, 300 e 600 indivíduos, e os resultados mostraram que o tamanho da população afetou diretamente a velocidade de convergência da rede, como mostra a Figura 5.3.

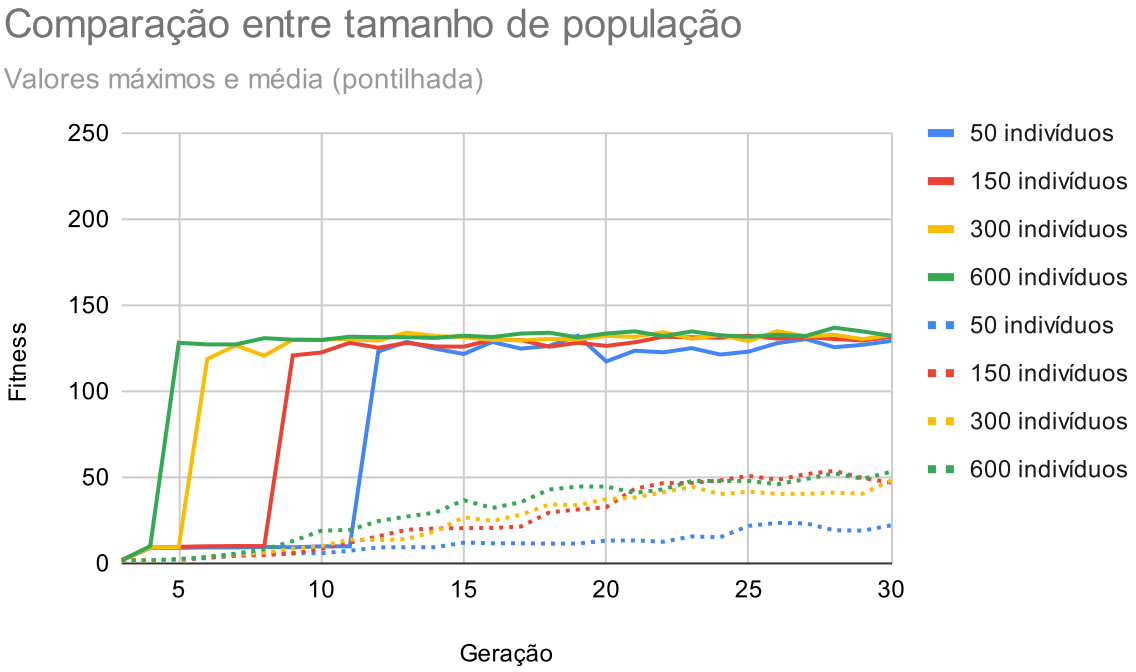


Figura 5.3: Exploração de parâmetros: Tamanho da população - Jogo Pong
Fonte: Os Autores

As redes com 150 e 300 indivíduos apresentaram resultados satisfatórios, enquanto a rede com 600 indivíduos obteve desempenho excelente, porém com um tempo de treinamento muito longo. Por isso, optou-se por utilizar uma população de 150 indivíduos, que obteve resultados semelhantes à população de 300 indivíduos, mas com um tempo de treinamento significativamente menor.

5.1.4 Número de Nós Ocultos

O próximo parâmetro avaliado foi o número de camadas ocultas da rede. Foram realizados testes com números de camadas iguais a 0, 3, 6 e 9. Observou-se que, a partir da Figura 5.4, à medida que o número de camadas aumentava, a rede apresentava uma convergência pior. Isso pode ser explicado pelo fato de que a rede precisa aprender a lidar com a complexidade adicional de cada camada oculta. Por essa razão, a rede com 0 camadas extras teve um desempenho melhor, permitindo que o algoritmo evoluísse para uma solução mais próxima do ideal.

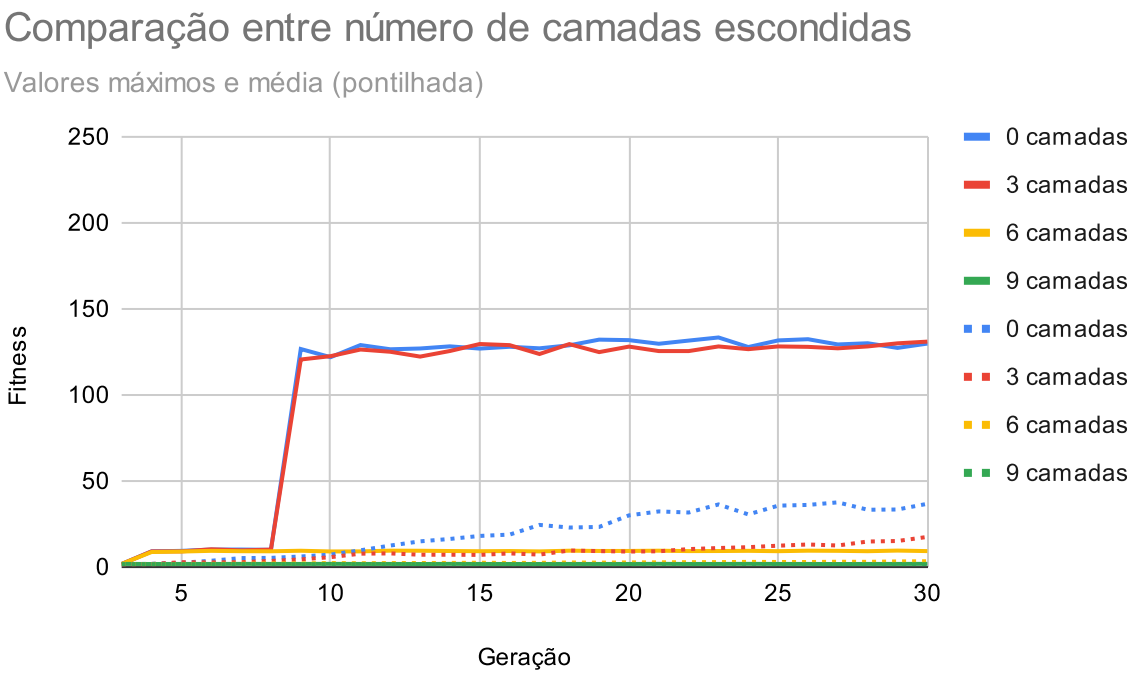


Figura 5.4: Exploração de parâmetros: Camada oculta - Jogo Pong
Fonte: Os Autores

5.2 MODELO A

A seguir, serão apresentados os resultados obtidos utilizando a arquitetura que considera as posições X e Y do jogador e da bola como entrada. Serão apresentados os resultados do treinamento do jogo base Pong, bem como os resultados obtidos sem e com transferência de aprendizado nos jogos Breakout e Tennis.

5.2.1 Pong

Analisando o gráfico da avaliação de aptidão (Figura 5.5), o modelo desenvolvido com entrada de todas as ações para jogar Pong não conseguiu convergir para a solução ideal estabelecida. Um dos fatores que contribuíram para esse resultado foi a baixa ou nenhuma relevância de algumas entradas e saídas da rede, como é o caso da posição X do jogador. Como a jogador se move somente na vertical, sem variação de valores nessa coordenada, essa entrada acaba não sendo utilizada pela rede para decidir o melhor movimento.

Além disso, outros fatores que podem ter influenciado o desempenho do modelo foram a quantidade de valores possíveis para as entradas X e Y da bola e Y do jogador. Como o modelo utiliza posições absolutas, a rede precisa abstrair essas informações para calcular a distância da bola para o jogador. No entanto, devido ao treinamento ter apenas 30 gerações, essa abstração não foi feita adequadamente durante esse curto período. Isso pode ter dificultado a capacidade da rede de tomar decisões mais precisas e efetivas durante o jogo.

Todos esses fatores contribuíram para a dificuldade do modelo, com o espaço amostral completo, em convergir para a solução ideal. Devido a isso, a transferência de aprendizado dessa arquitetura terá impactos nos próximos jogos, visto que não foi possível aprender o conhecimento base.

Entretanto, analisando o modelo B com espaço amostral reduzido, observa-se que houve convergência.

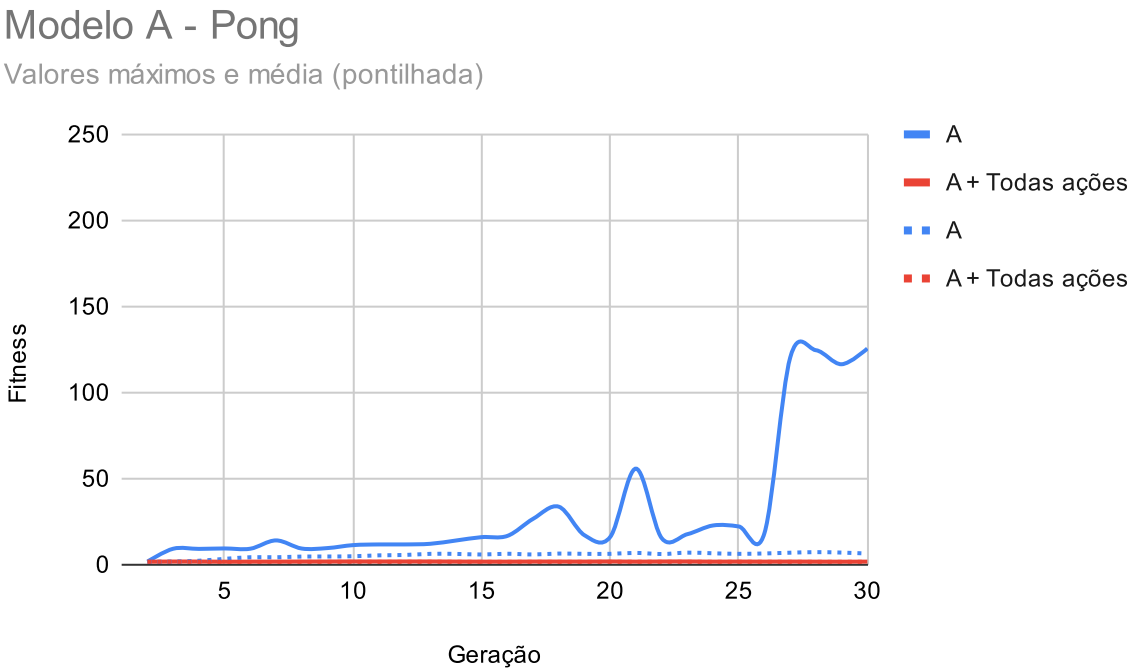


Figura 5.5: Resultados treinamento clássico do jogo Pong - Modelo A
Fonte: Os Autores

5.2.2 Breakout

Nesse estudo, foram realizados treinamentos clássicos no jogo Breakout com duas opções de espaço amostral de ações, sendo uma expandida e outra limitada. Foi constatado que o espaço amostral expandido teve um desempenho melhor em relação ao espaço limitado, em termos de taxa de *fitness*, principalmente a partir da octogésima (80ª) geração (Figura 5.6).

No entanto, ao realizar a transferência de aprendizado a partir da população do Pong, os resultados obtidos foram piores do que o treinamento clássico. Isso se deve ao fato de que a população inicial do Pong não conseguiu abstrair o conhecimento adquirido previamente, o que acabou impactando negativamente o treinamento do Breakout.

Esse problema se dá porque a população inicial do Pong tinha um conhecimento prévio de movimentação baseado em coordenadas absolutas, que não pôde ser abstraído para uma nova ambientação. Como resultado, os indivíduos dessa população acabaram prejudicando o treinamento da rede no novo jogo, ocasionando um treinamento negativo.

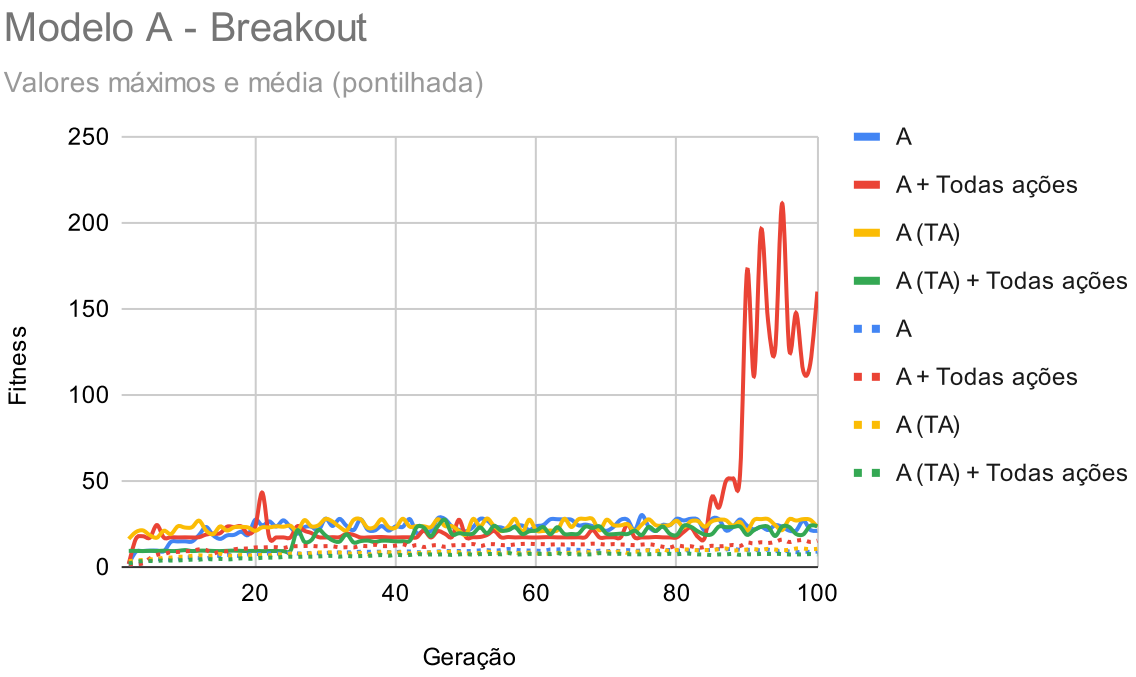


Figura 5.6: Resultados treinamento clássico e por transferência de aprendizado do jogo Breakout - Modelo A
Fonte: Os Autores

5.2.3 Tennis

Para o treinamento no jogo Tennis, foi constatado que o desempenho do treinamento clássico com as ações limitadas se manteve similar ao do treinamento clássico com todas as ações (Figura 5.7).

Já na transferência de aprendizado do jogo Pong, houve um ganho inicial no treinamento da rede no treinamento clássico. Entretanto, na metade das gerações ambos os treinamentos de TA, com espaço amostral reduzido e completo, ficaram semelhantes, em faixas próximas de pontuação. A população inicial utilizada, com conhecimento prévio de movimentação a partir de coordenadas absolutas do Pong, prejudicou o treinamento da rede inicialmente, resultando em um desempenho inferior. No entanto, mesmo com essa dificuldade inicial, a rede conseguiu evoluir e convergir para um resultado igual ao treinamento sem transferência de aprendizado a partir da sexagésima (60ª) geração.

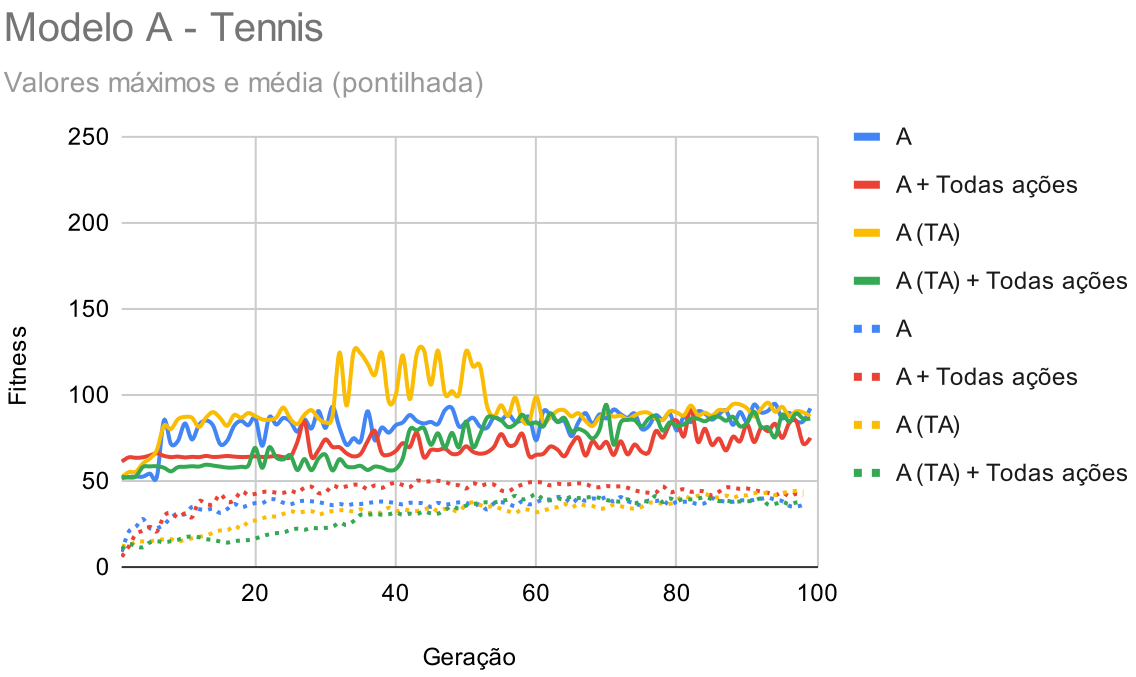


Figura 5.7: Resultados treinamento clássico e por transferência de aprendizado do jogo Tennis - Modelo A
Fonte: Os Autores

5.3 MODELO B

A seguir, serão apresentados os resultados obtidos com a arquitetura que utiliza as posições relativas do jogador em relação à bola como entrada. Serão mostrados os resultados do treinamento do jogo base Pong, bem como os resultados do treinamento sem e com transferência de aprendizado nos jogos Breakout e Tennis.

5.3.1 Pong

Para esse segundo modelo, a rede conseguiu convergir para uma solução a partir da décima quinta (15ª) geração, no espaço amostral reduzido, e vigésima quarta (24ª) geração utilizando o espaço amostral expandido (Figura 5.8). O retardo da convergência do modelo B utilizando o espaço amostral expandido, se deve ao tempo gasto utilizado para descobrir quais são as ações relevantes ao contexto do Pong durante o treinamento.

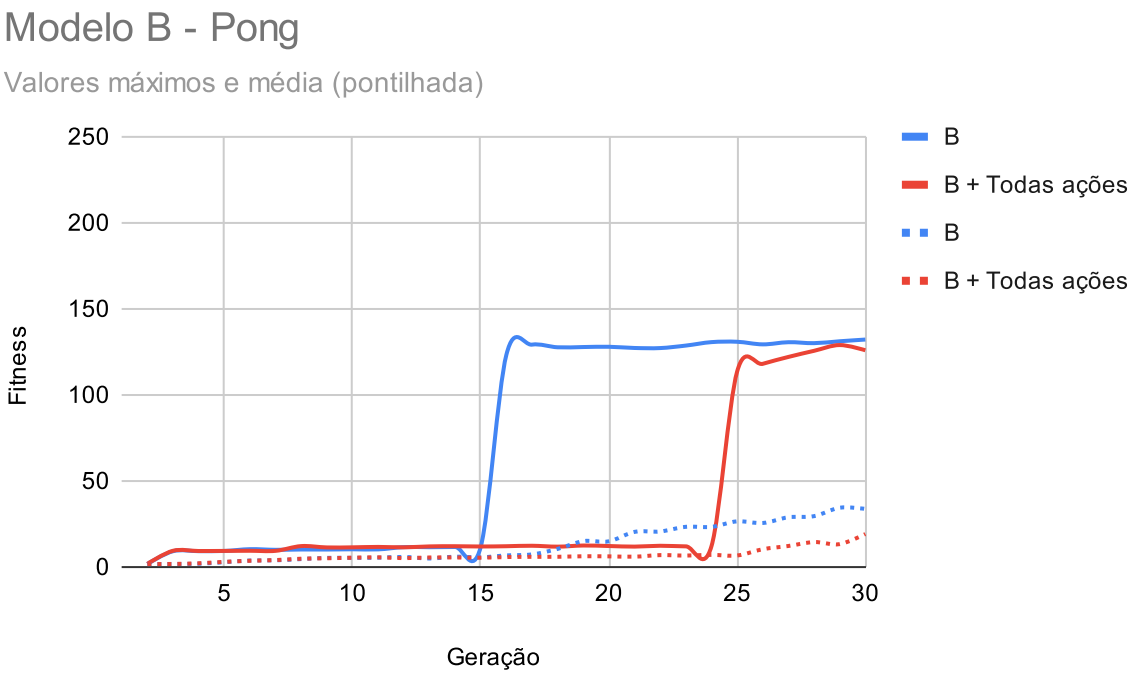


Figura 5.8: Resultados treinamento clássico do jogo Pong - Modelo B
Fonte: Os Autores

5.3.2 Breakout

Para o treinamento clássico no modelo B, existe uma diferença inicial no treinamento entre o Breakout, com o modelo de todas as ações disponíveis saindo com vantagem, comparado ao espaço amostral reduzido em 4 ações. NO entanto, ao decorrer das geração esse desempenho se assemelha. Para o jogo do Breakout, existem duas ações principais que o agente deve aprender, movimentar-se no campo e retornar a bola para o jogo quando o jogador perde uma vida. Analisando o espaço amostral de ações expandido, temos que, existem 13 de 18 ações distintas que retornam a bola em jogo, devido a isso, existe uma probabilidade maior que o agente retome a bola para o jogo mais cedo. Essa maior probabilidade da rede retomar a bola para o jogo, aliado a recompensa ao jogador pela quantidade de frames que a bola está em jogo (descrita na função *fitness*), consegue receber uma pontuação maior que as outras implementações.

Para o treinamento aplicando a transferência de aprendizado, não existem diferenças significativas entre o espaço amostral expandido ou reduzido.

Analisando a Figura 5.9, quando comparamos o treinamento clássico com o da transferência, podemos fazer algumas observações. Utilizando o espaço amostral reduzido, a TA se sai superior em grande parte do treinamento, devido ao seu conhecimento base de movimentação, com ambas as redes se estagnando com a *fitness* igual a partir da vigésima geração (20ª). Ao longo das gerações ambos os modelos se assemelham em desempenho.

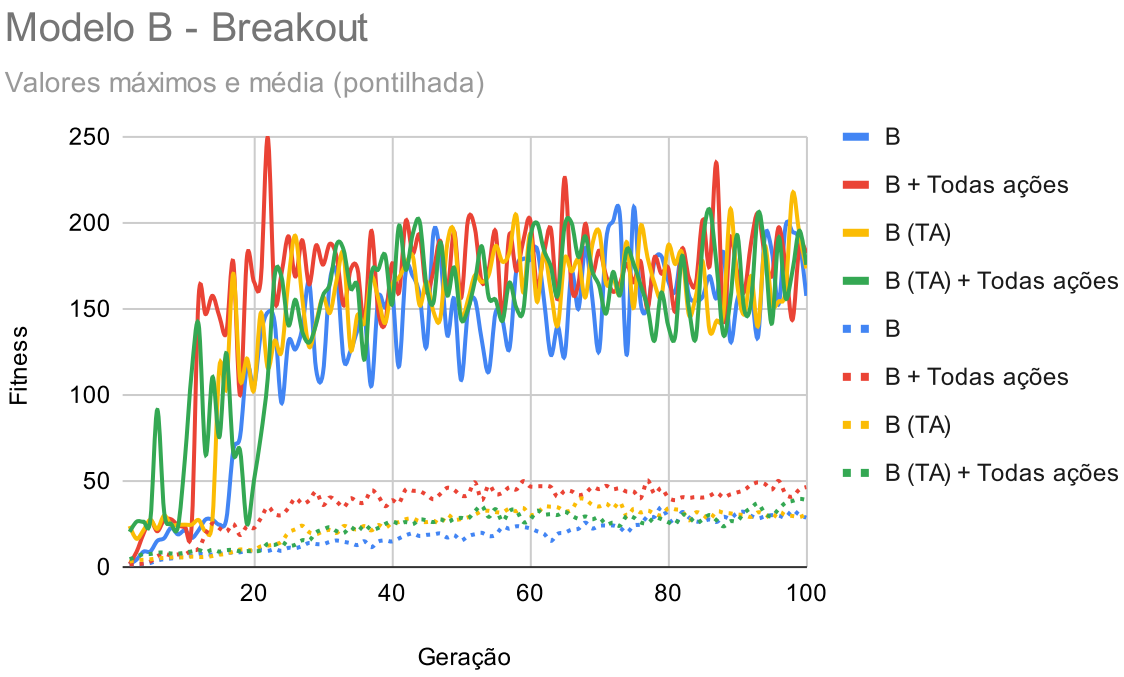


Figura 5.9: Resultados treinamento clássico e por transferência de aprendizado do jogo Breakout - Modelo B
Fonte: Os Autores

5.3.3 Tennis

Neste modelo, o treinamento clássico teve uma diferença notável ao comparar o Tennis com o espaço amostral expandido e espaço amostral reduzido em 4 ações. A partir da décima (10ª) geração houve uma disparidade entre o desempenho dos tipos de treinamento clássico. Essa discrepância se deve a limitação que a rede vai possuir entre os treinamentos. Para o treinamento com ações reduzidas, só vai ser possível o jogador movimentar-se em um eixo e rebater a bola quando necessário. Essa limitação de movimentação mostra-se bastante prejudicial, visto que a rede neural não consegue uma pontuação acima de 125 pontos na fitness em nenhum momento.

Para o treinamento aplicando a transferência de aprendizado, mostra-se também uma grande superioridade em relação aos treinamentos com espaço amostral reduzido e expandido, devido ao mesmo motivo citado anteriormente.

Analisando o gráfico da Figura 5.10, nota-se que a rede com a TA com todas as ações disponíveis inicia com uma taxa fitness alta e mantém vantagem inicial até a quadragésima geração (40ª). O treinamento a partir da população do Pong, otimiza o treinamento do agente, garantindo um conhecimento prévio de movimentação em um dos dois eixos disponíveis, permitindo que a rede neural possa focar em expandir o aprendizado na movimentação do eixo restante, e também na necessidade de sacar a bola quando necessário.

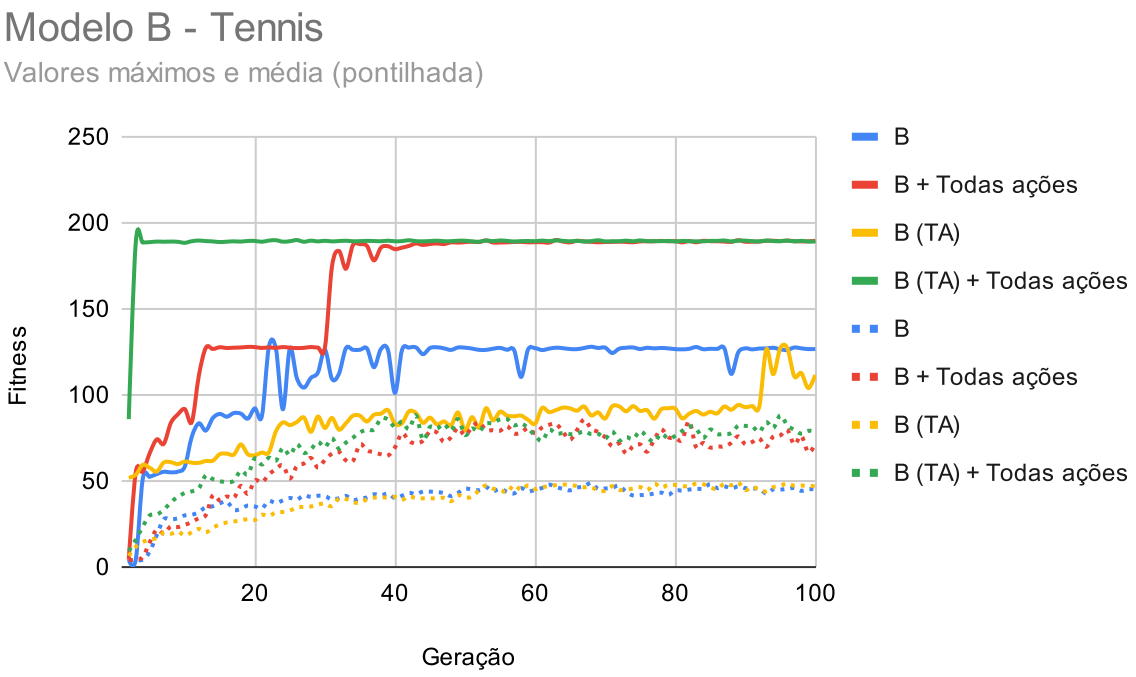


Figura 5.10: Resultados treinamento clássico e por transferência de aprendizado do jogo Tennis - Modelo B
Fonte: Os Autores

5.4 COMPARATIVO ENTRE MODELOS A E B

Nesta seção comparamos o resultado dos treinamentos dos jogos entre os modelos A e B. Esses treinamentos são compostos pelo método de treinamento clássico (do zero) e pelo treinamento de transferência de aprendizado. Apenas o jogo Pong não possui transferência de aprendizado.

5.4.1 Pong

No treinamento clássico entre modelos A e B, ilustrado na Figura 5.11, nota-se que o modelo B, que utiliza coordenadas relativas, obteve evolução a partir da décima quinta (15ª) geração, com destaque para o modelo de quatro ações. No modelo A, o jogo progrediu lentamente e alcançou o outro modelo na trigésima (30ª) geração, que é o fim do intervalo observado.

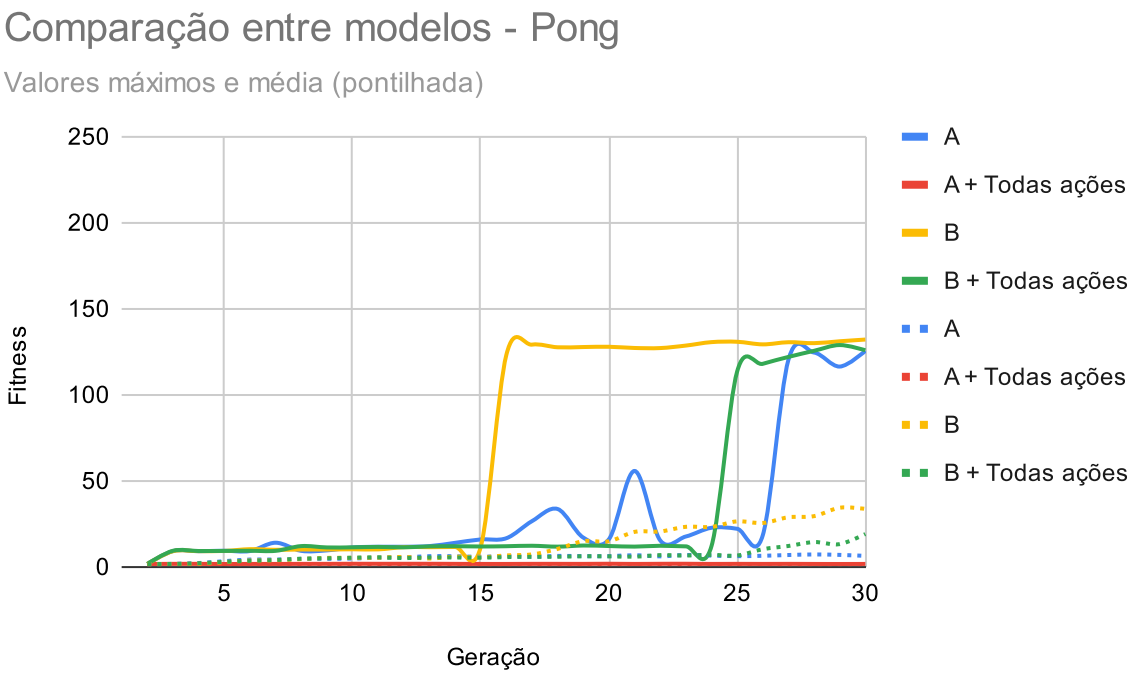


Figura 5.11: Resultados do treinamento do jogo Pong comparando os modelos A e B
Fonte: Os Autores

5.4.2 Breakout

Analisando o teste de treinamento clássico com entrada de quatro ações na rede entre modelos A e B, ilustrado na Figura 5.12, observa-se que o modelo B progrediu notavelmente, atingindo maior pontuação na taxa *fitness* que o modelo A, que se manteve estagnado. Em relação a transferência de aprendizado, o modelo B manteve a vantagem sobre o modelo A, por volta da décima quinta (15ª) geração do treinamento.

O exemplo com entrada de todas as ações do jogo, no modelo clássico, conquistou vantagem relevante no modelo B, em relação ao modelo A que manteve uma faixa constante, mas ao final do treinamento se igualaram, como mostra a Figura 5.13. Em relação a transferência de aprendizado, o modelo B obteve maior taxa de aptidão comparado ao modelo A, que se manteve estagnado.

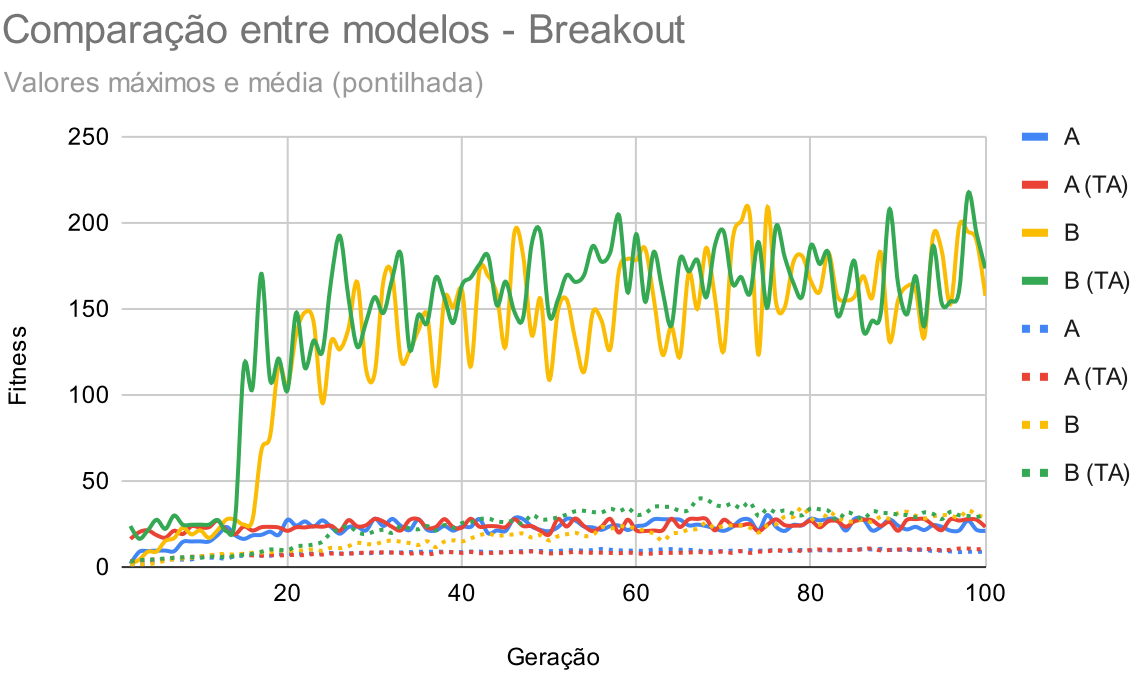


Figura 5.12: Comparação dos modelos A e B no jogo Breakout
Fonte: Os Autores

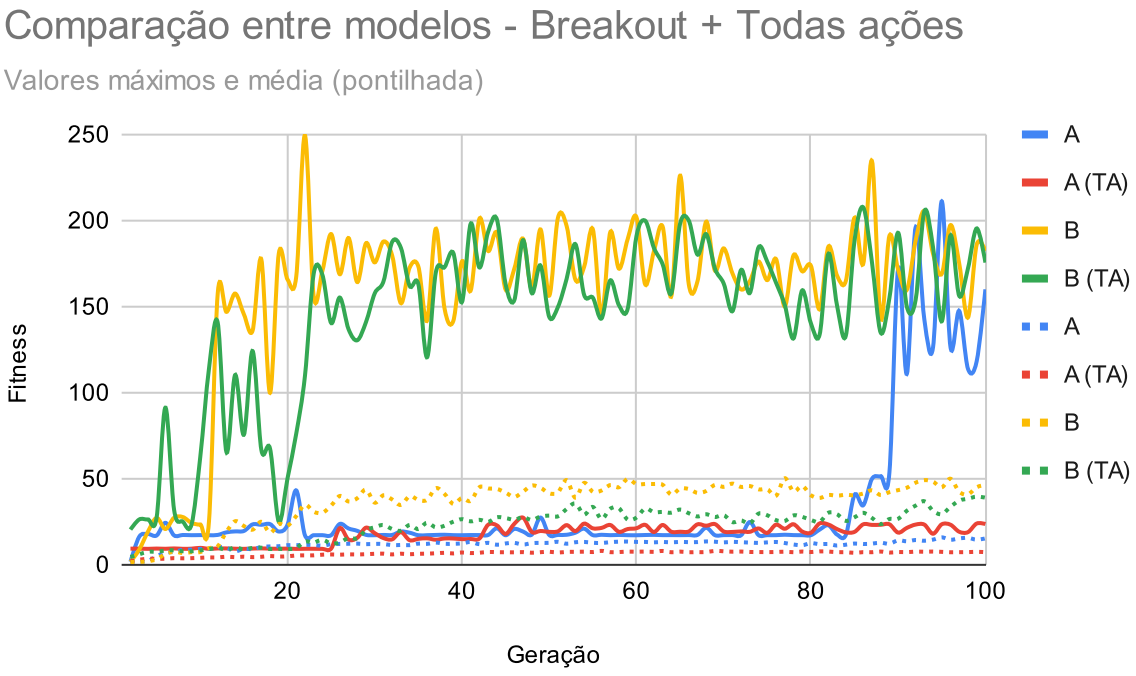


Figura 5.13: Comparação dos modelos A e B no jogo Breakout com entrada de todas as ações
Fonte: Os Autores

5.4.3 Tennis

Em análise, o treinamento do zero com quatro ações de entrada manteve desempenho similar nos modelos A e B, com o modelo B abrindo vantagem, como ilustra a Figura 5.14. Com a transferência de aprendizado, o modelo A também ficou com desempenho próximo ao do modelo B, com exceção do intervalo entre as gerações 35 a 55. Após esse intervalo divergente, o modelo B abriu vantagem novamente próximo as gerações finais.

No treinamento com entrada da rede com espaço amostral completo, como mostra a Figura 5.15, o modelo B teve melhor desempenho em relação ao modelo A, no treinamento do zero. Em relação ao treinamentos com transferência de aprendizado, o modelo B também abriu vantagem sobre o modelo A, atingindo maior taxa fitness.

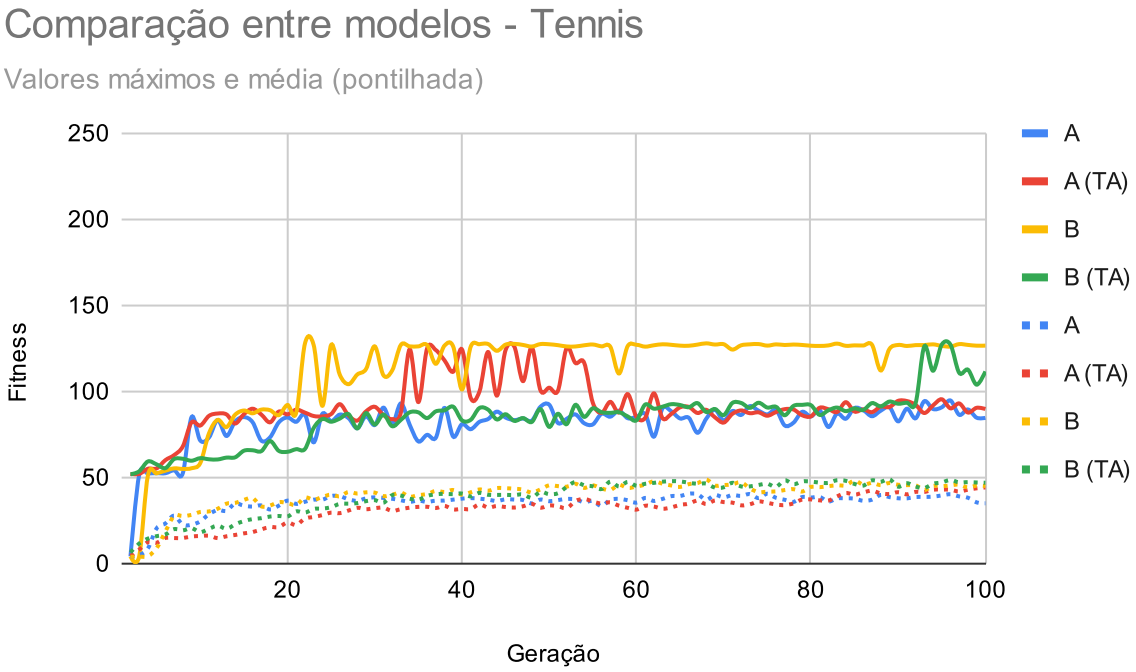


Figura 5.14: Comparação dos modelos A e B no jogo Tennis
Fonte: Os Autores

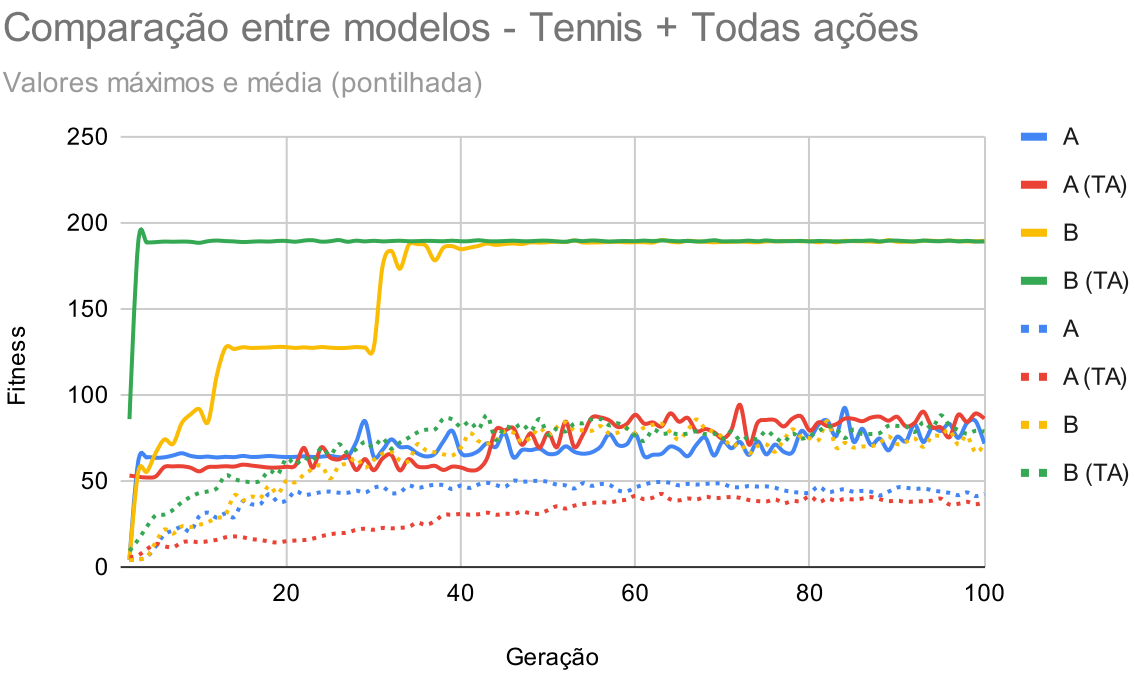


Figura 5.15: Comparação dos modelos A e B no jogo Tennis com entrada de todas as ações
Fonte: Os Autores

6 CONCLUSÃO

Em conclusão, este trabalho apresentou o desenvolvimento de agentes de IA baseados em algoritmo neuroevolutivo e transferência de aprendizado em ambientes de jogos eletrônicos do console Atari 2600. Os agentes utilizam endereços de memória RAM emulada para escolher as ações com base no *frame* do jogo e são treinados a partir de recompensas obtidas pela *fitness*.

Foram realizados experimentos para explorar alguns parâmetros no NEAT e entender o impacto na taxa de aprendizagem. Também foi realizado um experimento para analisar o desempenho de redes neurais neuroevolutivas para jogar três jogos de domínios similares e realizar a transferência de aprendizado entre eles. Cada jogo foi testado com dois tamanhos de espaço amostral de entrada diferentes, um com espaço amostral reduzido e outro com o espaço amostral completo, e dois modelos de representação do jogo, o modelo A com posições absolutas e o modelo B com posições relativas.

Em resumo, o modelo B, que utiliza posições relativas (normalizadas), teve melhor pontuação de aptidão em comparação ao modelo A, com posições absolutas, devido à facilidade de compreensão das entradas pela rede e menor necessidade de readaptação na transferência de aprendizado.

Em relação ao espaço de ações, a limitação do espaço amostral leva a um treinamento mais rápido, mas com menor desempenho, enquanto um espaço de ações maior permite maior exploração e, portanto, tende a ter um melhor desempenho, embora com maior tempo de treinamento. Os resultados mostraram um melhor desempenho nas redes com entradas de maior espaço amostral.

Quanto à transferência de aprendizado, há vantagens na economia de tempo no treinamento para atingir uma taxa *fitness* maior. Nos testes realizados, em determinados casos a transferência de aprendizado com espaço amostral completo atingiu uma taxa *fitness* maior em menos tempo, otimizando o tempo de treinamento. Vale ressaltar que isso ocorre desde que a rede do problema de origem seja compatível com a rede do problema de destino, o que pode ser um desafio realizar essa compatibilização e não garante a convergência.

Com relação ao tempo de treinamento, em média, cada treinamento do zero com 100 gerações leva em média 25 minutos para ser concluído, dependendo do jogo, e a transferência de aprendizado não possui diferença de tempo. Esse tempo não é considerado elevado, comparado a métodos de aprendizado por reforço profundo (de Carvalho, 2021), e abre a possibilidade de criar novos cenários de testes. É importante destacar que cada jogo possui peculiaridades específicas, e que é necessário analisar cada caso individualmente para encontrar a melhor estratégia de aprendizado para cada jogo. Por isso, são necessários mais testes para aprimorar as técnicas de aprendizado de máquina e aprimorar a capacidade dos agentes em jogos mais complexos.

Ao longo do desenvolvimento desse estudo encontramos alguns desafios. Um dos principais foi a demanda de tempo necessária para construir o *framework*, modelar a *fitness* e representar o jogo como entrada da rede, devido a complexidade na modelagem e representação dos problemas. Outro desafio é a escassez de trabalhos relacionados que abordam os três temas principais do TCC juntos: transferência de aprendizado, NEAT e jogos de Atari 2600. Todos esses desafios podem tornar o processo de desenvolvimento do TCC mais demorado e complexo, mas também podem ser oportunidades para o pesquisador explorar novas ideias e soluções criativas.

Algumas limitações deste trabalho são os testes realizados num mesmo escopo e restringido a 3 jogos selecionados, ao número relativamente baixo de execuções nos resultados e nas gerações de treinamento, além da representação do jogo para a entrada da rede restrita ao mapeamento dos itens do cenário na memória RAM emulada do console.

Como trabalho futuro, é sugerido testar a transferência de aprendizado com jogos de outros domínios, ampliar a análise e testes dos parâmetros do NEAT e testar outras formas de representar o ambiente do jogo para a rede neural, ampliando a visão da rede. Em resumo, este trabalho demonstrou a viabilidade do uso de algoritmos neuroevolutivos e transferência de aprendizado em ambientes de jogos eletrônicos, abrindo caminho para pesquisas futuras em outras aplicações.

REFERÊNCIAS

A. Tupper, K. N. (2000). An algorithm for transfer learning in a heterogeneous environment. Em *GECCO '20: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, páginas 99–100, Cancún Mexico. Association for Computing Machinery (ACM).

Anand, A., Racah, E., Ozair, S., Bengio, Y., Côté, M.-A. e Hjelm, R. D. (2019). Unsupervised state representation learning in atari. *arXiv preprint arXiv:1906.08226*.

B. Koçer, A. A. (2010). Genetic transfer learning. *Elsevier*, 37:6997–7002.

Barreto, J. (1997). *Inteligência Artificial. No Limiar do Século XXI*. ISBN 859003822X. Edições, 1997.

Bellemare, M. G., Naddaf, Y., Veness, J. e Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Brockman, G., Cheung, V., Ludwig Pettersson, J. S., Schulman, J., Tang, J. e Zaremba, W. (2016). Openai gym. `https://github.com/openai/gym`. Acessado em 27/12/2022.

de Carvalho, P. W. K. (2021). Análise de algoritmos de aprendizagem por reforço profundo em jogos de atari 2600.

de Pádua Braga, A., Ludermir, T. B. e de Leon Ferreira Carvalho, A. C. P. (2000). *Redes neurais artificiais: teoria e aplicações*, volume 1. LTC.

Dinh T. T. H, Thi Huong Chu, Q. U. N. (2015). Transfer learning in genetic programming. *IEEE Congress on Evolutionary Computation*.

Figueiredo, L. (2021). Transferência de aprendizado para redes de dependência relacional com boosting através de algoritmos genéticos. Dissertação de Mestrado, UFRJ, Rio de Janeiro - RJ.

Hausknecht, M., Lehman, J., Miikkulainen, R. e Stone, P. (2014). A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*.

Hodjat, B. (2018). The ai resurgence: why now? `https://www.wired.com/insights/2015/03/ai-resurgence-now/`. Acessado em 26/06/2022.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*, volume 1. The University of Michigan Press, Ann Arbor, MI.

Hosna, A. (2022). Transfer learning: a friendly introduction. *Journal of Big Data*.

K. Stanley, R. M. (2002). Evolving neural networks through augmenting topologies. *The MIT Press Journals*, 10(2):99 – 127.

Kramer, O. (2017). *Genetic Algorithm Essentials*, volume 679. Springer International Publishing.

Mila-Iqia (2019). Unsupervised state representation learning in atari. `https://github.com/mila-iqia/atari-representation-learning`. Acessado em 04/01/2023.

Mitchell, M. (1997). *An introduction to genetic algorithms*. The MIT Press Journals.

Ravishankar, H. (2016). Understanding the mechanisms of deep transfer learning for medical images. *Springer International Publishing*, 10008:188–196.

Sinno Jialin Pan, Q. Y. (2010). A survey on transfer learning. *IEEE*, 22(10):1345–1359.

APÊNDICE A – PARÂMETROS

A.1 CONFIGURAÇÃO DO ARQUIVO CONFIG-NEAT

Tipo	Parâmetro	Valor
NEAT	fitness_criterion	mean
	fitness_threshold	10000
	pop_size	150
	reset_on_extinction	False

Tabela A.1: Configuração do arquivo NEAT - Parâmetros gerais

Tipo	Parâmetro	Valor
Especies	compatibility_threshold	2.0

Tabela A.3: Configuração do arquivo NEAT - Parâmetros de espécies

Tipo	Parâmetro	Valor
Estagnação	species_fitness_func	mean
	max_stagnation	20
	species_elitism	1

Tabela A.4: Configuração do arquivo NEAT - Parâmetros de estagnação

Tipo	Parâmetro	Valor
Reprodução	elitism	3
	survival_threshold	0.2

Tabela A.5: Configuração do arquivo NEAT - Parâmetros de reprodução

Tipo	Parâmetro	Valor
Genoma	activation_default	relu
	activation_mutate_rate	0.0
	activation_options	relu
	aggregation_default	sum
	aggregation_mutate_rate	0.0
	aggregation_options	sum
	bias_init_mean	0.0
	bias_init_stdev	1.0
	bias_max_value	30.0
	bias_min_value	-30.0
	bias_mutate_power	0.5
	bias_mutate_rate	0.7
	bias_replace_rate	0.1
	compatibility_disjoint_coefficient	1.0
	compatibility_weight_coefficient	0.5
	conn_add_prob	0.5
	conn_delete_prob	0.5
	enabled_default	True
	enabled_mutated_rate	0.35
	feed_forward	True
	initial_connection	unconnected
	node_add_prob	0.4
	node_delete_prob	0.4
	num_hidden	0
	num_inputs	4
	num_outputs	18
	response_init_mean	1.0
	response_init_stdev	0.0
	response_max_value	30.0
	response_min_value	-30.0
	response_mutate_power	0.0
	response_mutate_rate	0.0
	response_replace_rate	0.0
	weight_init_mean	0.0
	weight_init_stdev	1.0
	weight_max_value	30
	weight_min_value	-30
	weight_mutate_power	0.5
	weight_mutate_rate	0.8
	weight_replace_rate	0.1

Tabela A.2: Configuração do arquivo NEAT - Parâmetros de genoma

A.2 FUNÇÕES DE ATIVAÇÃO

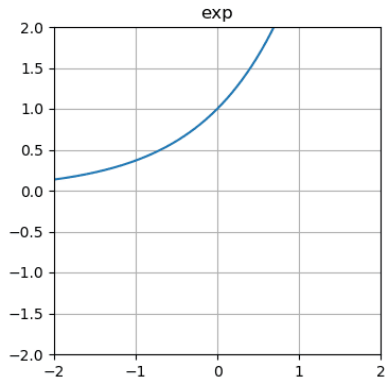


Figura A.1: Função de ativação - Exp

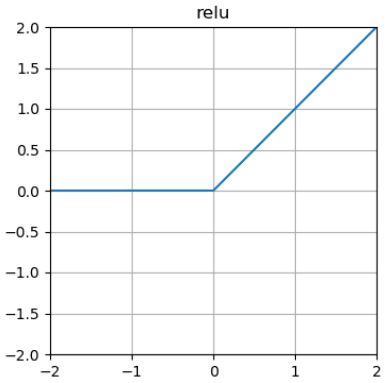


Figura A.2: Função de ativação - Relu

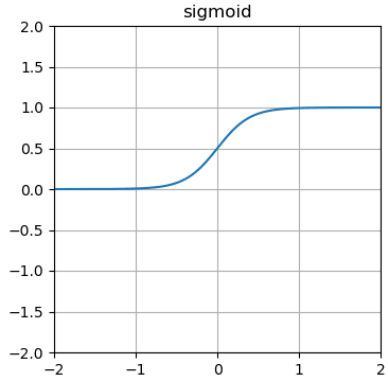


Figura A.3: Função de ativação - Sigmoid

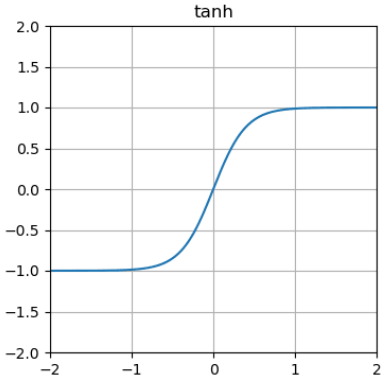


Figura A.4: Função de ativação - Tanh

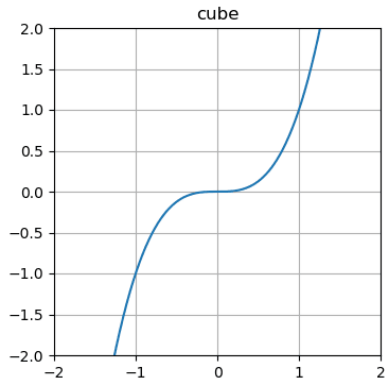


Figura A.5: Função de ativação - Cube

APÊNDICE B – TREINAMENTO

B.1 TEMPO MÉDIO DE TREINAMENTO

Modelo	Tempo Médio
Pong (A)	8 minutos
Pong (A) + Todas ações	8 minutos 20 s
Pong (B)	8 minutos
Pong (B) + Todas ações	8 minutos 33s
Breakout (A)	23 minutos
Breakout (A) + Todas ações	24 minutos
Breakout (B)	20 minutos
Breakout (B) + Todas ações	22 minutos
Tennis (A)	33 minutos
Tennis (A) + Todas ações	32 minutos
Tennis (B)	30 minutos
Tennis (B) + Todas ações	30 minutos

Tabela B.1: Tempo médio de treinamento para cada modelo