Programação Genética Cartesiana Paralela na Geração de Redes Neuronais Artificiais para o Reconhecimento de Atividade Humana

Bruno Marcos Pinheiro da Silva

JUIZ DE FORA

MARÇO, 2021

Programação Genética Cartesiana Paralela na Geração de Redes Neuronais Artificiais para o Reconhecimento de Atividade Humana

Bruno Marcos Pinheiro da Silva

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Heder Soares Bernardino

JUIZ DE FORA

MARÇO, 2021

Programação Genética Cartesiana Paralela na Geração de Redes Neuronais Artificiais para o Reconhecimento de Atividade Humana

Bruno Marcos Pinheiro da Silva

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Heder Soares Bernardino
Doutor em Modelagem Computacional

Alex Borges Vieira
Doutor em Ciências da Computação

Luciana Conceição Dias Campos
Doutora em Engenharia Elétrica

JUIZ DE FORA

15 DE MARÇO, 2021

Aos meus pais, irmãos e amigos.

Resumo

O Reconhecimento de Atividade Humana (RAH) é um problema promissor e aplicável em diversas situações reais, como na assistência médica remota ou em casas inteligentes. Sua solução, entretanto, apresenta algumas dificuldades, como o grande volume de dados e características disponíveis, associadas à coleta de dados, em geral, por sensores. Uma maneira para tratar o problema é através de modelos de aprendizado de máquina, como Redes Neurais Artificiais Evoluídas pela Programação Genética Cartesiana (CGPANN), mas esta abordagem é computacionalmente cara. Sendo assim, este trabalho propõe o desenvolvimento de técnicas de computação de alto desempenho sobre a CGPANN, assim como sua aplicação no treinamento de modelos para o problema de RAH. Para isso, a arquitetura de Unidades de Processamento Gráfico (GPU) é utilizada e diferentes estruturas de dados para a CGPANN são analisadas quanto ao tempo de execução do algoritmo na GPU e, posteriormente, quanto a qualidade dos modelos em relação ao erro de classificação para o problema de reconhecimento de atividades. Os resultados obtidos apontam uma redução no tempo de execução pela utilização de abordagens paralelas em relação ao algoritmo sequencial, além de indicarem que os modelos da CGPANN para o RAH são promissores em relação àqueles da literatura.

Keywords: Reconhecimento de Atividade Humana, Aprendizado de Máquina, Programação Genética Cartesiana, Redes Neurais Artificiais

Abstract

Human Activity Recognition (HAR) is a promising problem and applicable to a wide range of real-life situations, such as remote medical assistance and smart home appliances. Its solution, however, presents some difficulties, like the large volume of available data and features to be analyzed, associated with the data collection systems, generally made by sensors. A possible approach to deal with this problem is through machine learning models, such as Cartesian Genetic Programming of Artificial Neural Networks (CGPANN), but this technique is computationally expensive. Therefore, this work proposes the development of high-performance computing techniques for CGPANN, as well as its application on HAR. To do so, the architecture of Graphics Processing Units (GPUs) is used, and different data structures for the parallel CPGANN algorithm on the GPU are analyzed regarding its execution time. Next, the quality of the models related to their classification error when applied to a HAR dataset is also analyzed. The results show a decrease in computational time when the proposed parallel CGPANN algorithm is compared to the sequential approach. Moreover, the results point out that CGPANN's models for HAR are promising when compared to those from the literature.

Keywords: Human Activity Recognition, Machine Learning, Cartesian Genetic Programming, Artificial Neural Networks

Agradecimentos

Aos meus pais e irmãos pelo apoio, incentivo e amor durante este percurso.

Ao professor Heder pela orientação, amizade e paciência durante estes anos de trabalho e aprendizado.

Aos meus amigos, pelos momentos de descontração, pelo apoio e ajuda nos momentos difíceis.

Ao GET Computação, que foi fundamental para minha formação. Em especial aos getianos e getianas dos quais pude me tornar amigo e compartilhar tantos aprendizados.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

RAH	Reconhecimento de Atividade Humana
AE	Algoritmo Evolutivo
RNA	Rede Neural Artificial
PG	Programação Genética
CGP	Programação Genética Cartesiana
CGPANN	Programação Genética Cartesiana de Redes Neurais Artificiais
GPU	Unidade de Processamento Gráfico
GPGPU	Unidade de Processamento Gráfico de Propósito Geral
NEAT	Neuroevolução de topologias aumentantes
NeVA	Algoritmo Neuroevolucionário
ESP	*Enforced Subpopulations*
CoSyNE	*Cooperative Synapse Neuroevolution*
ECG	Eletrocardiograma
TEB	Impedância Elétrica Torácica
EDA	Atividade Eletrodérmica
EEG	Eletroencefalograma
CU	Unidades de Computação
PE	Elementos de Processamento
CNN	Rede Neural Convolucional

1 Introdução

1.1 Apresentação do Tema

A análise de sinais biológicos extraídos de sensores com o intuito de reconhecer diferentes tipos de atividade humana, como atividade física, emocional e mental é uma problemática atual e relevante, por exemplo, para as áreas de saúde e de cuidados preventivos (Mohino-Herranz et al., 2019). Neste contexto, uma abordagem para resolver o problema é a criação de modelos que possam predizer, baseado nos dados obtidos dos sensores, as atividades sendo performadas.

O aprendizado de máquina consiste em métodos que são capazes de identificar padrões sobre conjuntos de dados e, a partir disso, fazer predições ou extrair conhecimento sobre novos dados (Murphy, 2012). Como parte da área de Inteligência Artificial, este campo vem ganhando crescente destaque nos dias atuais, dada a abrangência de problemas aos quais suas técnicas podem ser aplicadas. A classificação de texto, o processamento de linguagem natural, o reconhecimento de imagens e o auxílio a diagnósticos médicos são exemplos desses problemas (Mohri et al., 2018).

Uma das formas do aprendizado de máquina é o aprendizado supervisionado, que utiliza dados de entrada associados a saídas previamente conhecidas para treinar modelos que, quando aplicados sobre dados desconhecidos, sejam capazes de predizer qual será a saída correta. Tais modelos são ditos preditivos e problemas de classificação, como o reconhecimento de atividade humana citado anteriormente, se enquadram nesta categoria.

Redes Neurais Artificiais são modelos bioinspirados nas redes neuronais que formam o cérebro de animais e são úteis, por exemplo, para a resolução de problemas de classificação, uma vez que modelam relações complexas do mundo real de forma flexível e se ajustam automaticamente com base em dados (Zhang, 2000). Entretanto, existem dificuldades no que tange a construção destes modelos, como o estabelecimento de topologias adequadas e o ajuste de pesos e parâmetros que compõem essas topologias.

Uma sub-área do aprendizado de máquina de interesse para este trabalho é a

Neuroevolução, que consiste na aplicação de Algoritmos Evolutivos (AEs) para o treinamento de Redes Neurais Artificiais (RNAs) (Turner, 2015). AEs são técnicas heurísticas inspiradas na evolução biológica descrita por Darwin (1859). A esta categoria pertencem algoritmos amplamente conhecidos, tais como os Algoritmos Genéticos (Goldberg, 1989), a Programação Genética (Koza, 1992) e a Programação Genética Cartesiana (Miller, 2011). Estas técnicas permitem, por exemplo, a busca por soluções de um problema sem uma definição prévia da estrutura exata desta solução (Poli et al., 2008).

Dessa forma, a aplicação de AEs para a descoberta de topologias de RNAs é promissora, pois tais métodos permitem a busca de topologias, pesos e/ou funções de ativação da rede sem a presença de especialistas. Uma técnica neuroevolutiva conhecida foi proposta por Khan et al. (2010) e denominada de programação genética cartesiana de redes neurais artificiais (CGPANN), que faz uso da modelagem de indivíduos como grafos para codificar e evoluir redes neurais. Como apontado por seus autores, a técnica se mostrou competitiva em relação a outros algoritmos como a Neuroevolução de Topologias Aumentantes (NEAT), o Algoritmo NeuroEvolucionário (NeVA), *Enforced Subpopulations* (ESP) e *Cooperative Synapse Neuroevolution* (CoSyNE) por gerar soluções com um menor número de gerações para os problemas estudados.

1.2 Descrição do Problema

Um ponto negativo da Neuroevolução é o alto custo computacional de treinamento das RNAs. Esta característica é associada à generalidade dos AEs que, mesmo com poucos indivíduos, requerem um grande número de avaliações de função objetivo por geração, especialmente em grandes conjuntos de dados (Turner, 2015; Galván and Mooney, 2020). Neste sentido, uma abordagem plausível para amenizar tal problema é a aplicação de técnicas de computação de alto desempenho sobre o processo neuroevolutivo.

Uma proposta é a extensão de um método já amplamente estudado dentro da Neuroevolução, como a CGPANN, para utilizar técnicas de computação de alto desempenho. Na literatura, diversas características de paralelismo em algoritmos evolutivos são descritas, seja na Programação Genética (Augusto e Barbosa, 2013) ou na Programação Genética Cartesiana (Harding e Banzhaf, 2007).

Nestes estudos, entretanto, conceitos da Programação Genética Cartesiana (utilizados pela CGPANN) não são levados em consideração. Dentre estes, por exemplo, a representação por grafos é de grande relevância, pois esta pode penalizar a implementação em certos ambientes paralelos, especificamente em Unidades de Processamento Gráfico de Propósito Geral (GPGPUs). Um outro fator que pode ser considerado é o tratamento de informações extras associadas à estrutura da CGPANN, como os pesos de conexões, dentro desses ambientes paralelos. Com isso, levanta-se a questão quanto a possibilidade da aplicação de técnicas de computação de alto desempenho eficientes sobre a CGPANN e os possíveis ganhos computacionais oriundos de propostas que considerem as especificidades do algoritmo utilizado e do contexto de sua aplicação.

1.3 Objetivos

O principal objetivo proposto é projetar métodos de computação de alto desempenho na geração de Redes Neurais Artificiais através da Programação Genética Cartesiana e aplicá-los ao problema de reconhecimento de atividade humana. Para alcançar tal objetivo, propõe-se especificamente:

- Avaliar a implementação de diferentes estruturas para a CGPANN paralela;
- Identificar características relevantes para a implementação das técnicas propostas em GPGPUs;
- Aplicar o uso da técnica proposta no treinamento de um classificador para identificar tipos de atividade humana com base em dados de sensores;
- Avaliar os resultados através da comparação do desempenho computacional e de métricas de desempenho para classificadores dos modelos encontrados com aqueles descritos na literatura.

2 Reconhecimento de Atividade Humana

O Reconhecimento de Atividade Humana (RAH) é um problema que vem ganhando destaque dado a crescente presença de sensores e sistemas de Internet das Coisas (IoT). O RAH geralmente faz uso de dados obtidos destes sistemas para reconhecer uma atividade sendo desenvolvida por uma pessoa, como caminhar, subir escadas ou correr, mas também existem abordagens de reconhecimento através de vídeos e imagens extraídos de câmeras que, entretanto, são menos comuns (Chen et al., 2020).

O problema do RAH é majoritariamente tratado pela literatura em relação a atividades que envolvem movimento. Porém, o reconhecimento de emoções (Horlings et al., 2008) e atividades mentais (Millán et al., 2002) também se enquadram nesta mesma categoria de problemas, compartilhando as técnicas aplicadas e desafios enfrentados.

O RAH pode ser tratado como um problema de aprendizado supervisionado. Este tipo de problema utiliza dados de entrada associados a saídas previamente conhecidas para treinar modelos preditivos que possam classificar corretamente novos dados. O fluxo do processo de reconhecimento de atividades é ilustrado na Figura 2.1 que se aplica a problemas gerais de classificação pela coleta de dados, extração de atributos, treinamento de modelos e a classificação em si.

O RAH é definido formalmente a seguir, como proposto por Lara e Labrador (2013). Dado um conjunto $S = \{S_0, ..., S_{k-1}\}$ de k séries temporais de atributos, todas definidas dentro de um intervalo de tempo $I = [t_{inicial}, t_{final}]$ sobre o qual a obtenção dos atributos aconteceu. O objetivo do problema é encontrar, com base no conjunto S, uma partição P de I e um conjunto de rótulos representando a atividade desenvolvida em cada intervalo da partição $P =< I_0, ..., I_{r-1} >$, em que r é o número de intervalos encontrados, de forma que:

$$\bigcup_{i=0}^{r-1} P_i = I$$

Através desta definição, objetiva-se definir tanto a atividade desenvolvida quanto

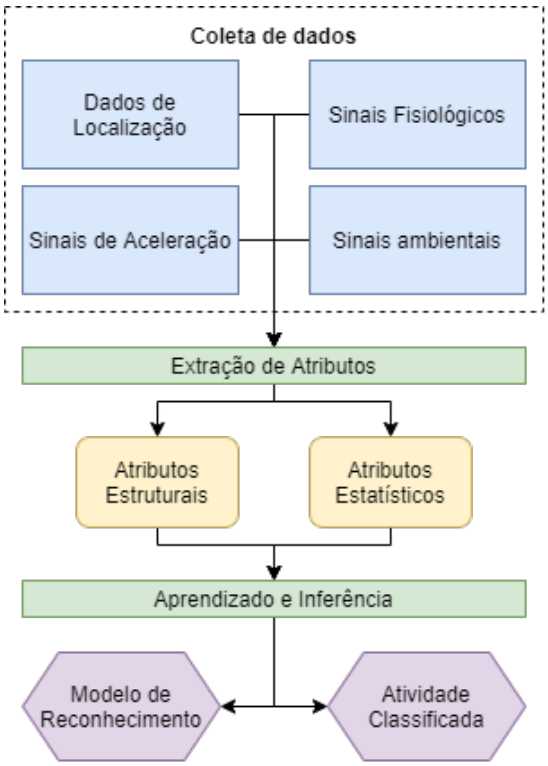


Figura 2.1: Etapas do problema de reconhecimento de atividades. Adaptado de Lara e Labrador (2013)

o intervalo de tempo específico durante o qual ela foi realizada. Pela complexidade deste sistema, existe a proposta de uma variação relaxada do RAH. Neste, define-se um conjunto $W = \{W_0, ..., W_{n-1}\}$ de n janelas de tempo de tamanho igual, cada uma associada a um conjunto S de atributos (como previamente definidos), e um conjunto A com m rótulos de atividades $A = \{a_0, ..., a_{m-1}\}$. O problema relaxado consiste em determinar um mapeamento dos rótulos de A para os intervalos de W com base nos atributos de S.

A diferença entre as duas definições é exemplificada na Figura 2.2. Nota-se que, no problema relaxado, existem erros, simbolizados na cor vermelha, devido a transições entre diferentes atividades que ocorrem dentro das janelas de tempo pré-definidas que, entretanto, não são de grande relevância considerando-se que o número de janelas de tempo é muito superior ao número de mudanças entre atividades (Lara e Labrador, 2013).

As aplicações do RAH são extensas, o que justifica o emergente número de estudos na área. Dentre suas possíveis aplicações, é possível elencar o monitoramento de idosos (Paraschiakos et al., 2020; Schrader et al., 2020), a análise de desempenho esportivo (Mitchell et al., 2013), o monitoramento de estresse (Mohino-Herranz et al., 2015), a

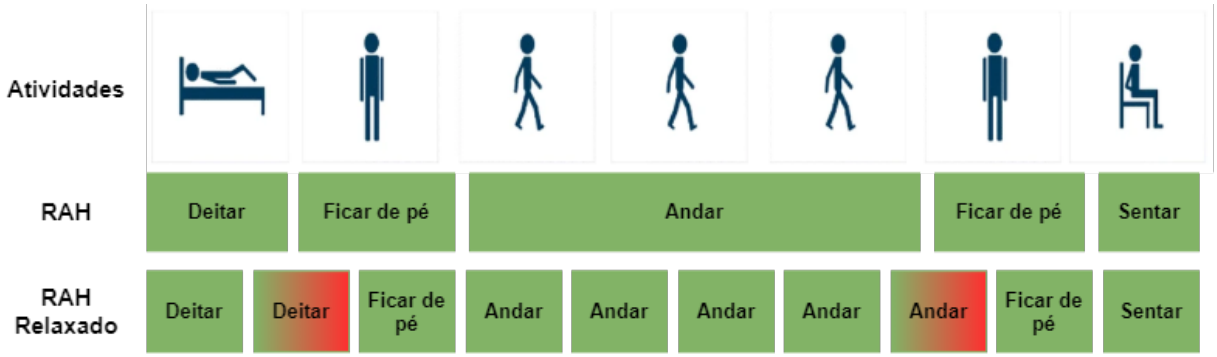


Figura 2.2: O problema RAH e sua variação relaxada. Adaptado de Schrader et al. (2020)

classificação de atividades suspeitas (Jie et al., 2008) e a classificação de atividades do dia a dia, como dormir, andar e tomar banho (Kasteren et. al., 2008).

Para a extração de atributos a serem utilizados no treinamento de modelos, o sensoriamento de dados biológicos é uma prática comum, realizada pela medida de informações de eletrocardiograma (ECG), impedância elétrica torácica (TEB), atividade eletrodérmica (EDA) e eletroencefalograma (EEG) mas, com a grande disponibilidade de dispositivos celulares, muitos trabalhos propõem o uso de dados de sensores de giroscópio, acelerômetro e GPS destes dispositivos para a classificação (Kwapisz et al., 2011).

No presente trabalho, o conjunto de dados "*Activity recognition using wearable physiological measurements*" apresentado em Mohino-Herranz et al. (2019) é utilizado. Este é composto de leituras de sensores vestíveis de ECG, TEB, e EDA, extraídos de 40 (quarenta) pessoas performando atividades específicas, que foram classificadas pelo seu tipo em atividades física, emocional, mental e neutra (descanso), resultando em um conjunto com 4480 instâncias. Estes dados brutos foram previamente processados a fim de extrair características relevantes, resultando em 151 características de TEB, 104 de EDA e 174 de ECG, em um total de 533 características. Estas são compostas por parâmetros médicos, assim como os seguintes parâmetros estatísticos: mediana, desvio padrão, média aparada de 25%, assimetria, curtose, máximo, mínimo, percentil 25%, percentil 75%, média geométrica, média harmônica e desvio padrão médio. O processo completo de extração de características é detalhado por Mohino-Herranz et al. (2019).

Quanto aos desafios na área, Chen et al. (2020) indica a falta de padronização de metodologias para comparação justa entre diferentes métodos e, como também indicado

por Lara e Labrador (2013), a dificuldade de definição dos atributos a serem utilizados no treinamento de modelos, além da heterogeneidade dos dados disponíveis, também são dificuldades enfrentadas. Kim et al. (2010) aponta a dificuldade de reconhecimento de atividades concorrentes e correlacionadas, devido a complexidade dos dados disponíveis para análise. Por exemplo, uma pessoa poderia ler um livro e tomar café ao mesmo tempo, dificultando a classificação correta entre as atividades.

3 Métodos

Neste capítulo, os principais métodos relacionados ao tema do trabalho são apresentados. Descreve-se a Neuroevolução pela definição de RNAs e AEs, suas aplicações e desafios, tópicos relacionados à computação de alto desempenho e algumas métricas de avaliação utilizadas.

3.1 Neuroevolução

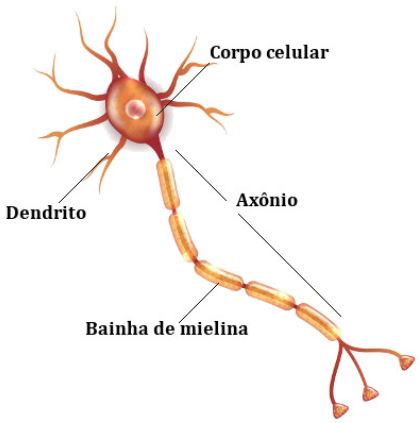
Neuroevolução é uma subárea do aprendizado de máquina que propõe a geração de topologias de Redes Neurais Artificiais pela aplicação de Algoritmos Evolutivos. Para compreendê-la, portanto, esta seção introduz os conceitos pertinentes a RNAs e AES, assim como as características inerentes à Neuroevolução.

3.1.1 Redes Neurais Artificiais

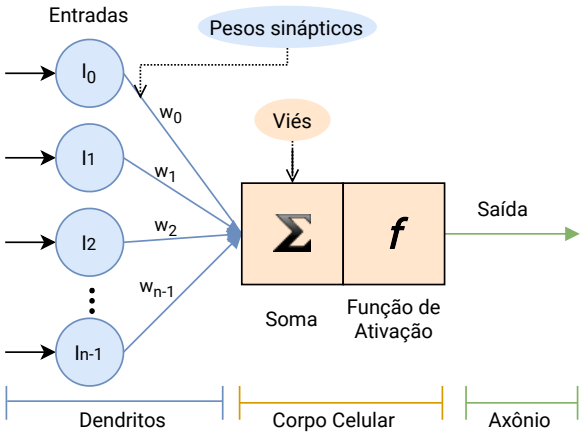
Redes Neurais Artificiais são modelos bioinspirados nas redes neuronais do cérebro de animais. Um neurônio biológico consiste em um corpo celular que recebe estímulos de outros neurônios através de dendritos e, caso suficientemente estimulados, propagam este impulso para outros neurônios através de seu axônio (Turner, 2015). Baseado nisso, RNAs são compostas por neurônios artificiais que podem ser comparados com um neurônio biológico, como indica a Figura 3.1.

De forma geral, cada neurônio artificial recebe entradas de outros neurônios, realiza uma soma ponderada pelos pesos sinápticos $\mathbf{w}$ de todas estas entradas com um valor de viés e aplica uma função de ativação f ao resultado. Esta função varia para cada modelo, mas dentre as mais comuns, destacam-se a função logística sigmoide (Eq. 3.1), a tangente hiperbólica (Eq. 3.2) e a *Rectified Linear Unit* (ReLU) (Eq. 3.3) (Ramachandran et al., 2017).

$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}} \tag{3.1}$$



(a) Representação de um neurônio real[1].



(b) Representação de um neurônio artificial

Figura 3.1: Representação de um neurônio biológico (a) e um artificial(b).

$$tanh(\mathbf{x}) = \frac{e^{\mathbf{x}} - e^{-\mathbf{x}}}{e^{\mathbf{x}} + e^{-\mathbf{x}}} \tag{3.2}$$

$$relu(\mathbf{x}) = max(0, \mathbf{x}). \tag{3.3}$$

Com isso, uma RNA consiste em uma série de neurônios artificiais interconectados sob determinada topologia. Esta topologia usualmente é constituída por um grafo direcionado ponderado e acíclico no caso de redes *feed-forward*, ou com ciclos em redes recorrentes (Jain et al., 1996).

A estrutura de RNAs pode ser totalmente conectada, com camadas ocultas, ou possuir uma organização esparsa ou arbitrária, em que os nós do grafo não são totalmente conectados com seus vizinhos nem organizados em camadas bem definidas (Gajurel et al., 2020). Estas organizações são exemplificadas na Figura 3.2.

Um desafio geral enfrentado é a definição específica de cada parâmetro da estrutura destas redes, como suas topologias - o número de nós, a quantidade de camadas ocultas e a conexão entre os nós - os pesos das conexões, ou até mesmo a função de ativação ideal para determinado tipo de problema. Abordagens para otimização dos pesos como a *backpropagation* são amplamente utilizadas mas, nestes casos, a topologia é mantida constante e muitas vezes definida por tentativa e erro. Em geral, este processo pode ser árduo e requerer a análise de especialistas. Portanto, o uso de métodos capazes

[1]Disponível em: <https://brasilescola.uol.com.br/o-que-e/biologia/o-que-e-neuronio.htm>

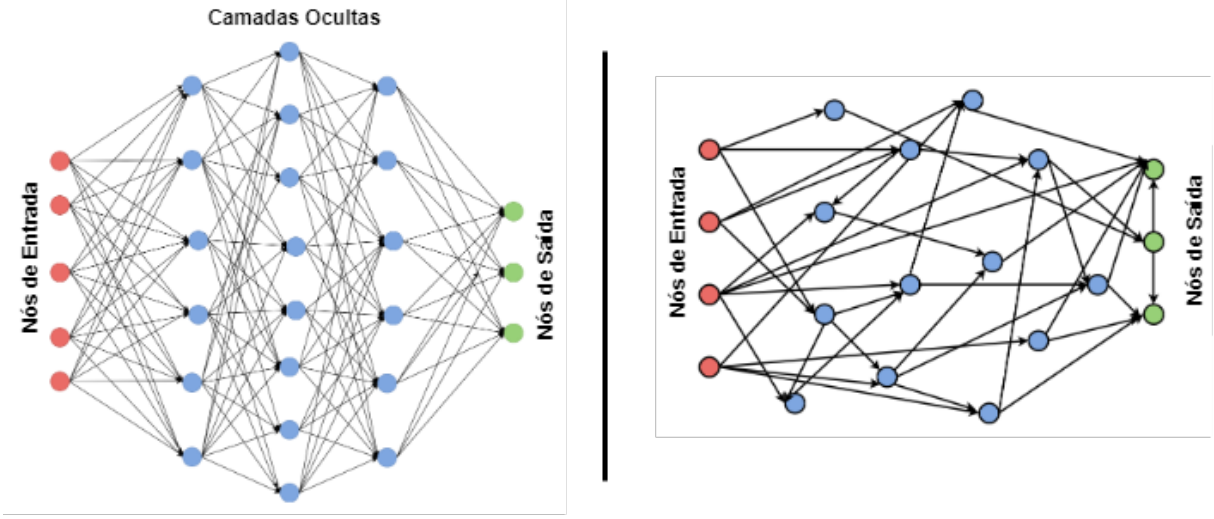


Figura 3.2: Exemplo de uma RNA totalmente conectada (esq.) e uma topologia arbitrária (dir.). Adaptado de Gajurel et al. (2020)

de modelar estas estruturas automaticamente mostra-se promissor.

3.1.2 Algoritmos Evolutivos

Algoritmos evolutivos são técnicas heurísticas estocásticas inspiradas na evolução biológica descrita por Darwin (1859), especialmente no conceito de sobrevivência dos melhores. Estes são algoritmos populacionais em que um conjunto de soluções candidatas (população) são evoluídas através da aplicação de operadores genéticos, como a mutação e recombinação, e avaliadas sob uma determinada função objetivo, que determina a qualidade de cada indivíduo. Esta qualidade é utilizada para guiar a evolução da população, em que indivíduos de melhor qualidade são preferencialmente escolhidos para continuarem no processo evolutivo, garantindo a permanência de características mais promissoras durante a busca.

Em geral, um algoritmo evolutivo é definido por uma série de componentes, que são sua representação (indivíduos), a função de aptidão, os operadores evolutivos (mutação e recombinação), e critérios de seleção dos indivíduos pais para evolução e permanência na população, como descrito por Eiben e Smith (2015). Estes componentes integram diretamente o algoritmo, que pode ser visto no Algoritmo 1.

A esta categoria de técnicas pertencem propostas amplamente conhecidas, tais como os Algoritmos Genéticos (Goldberg, 1989), a Programação Genética (PG) (Koza,

Algoritmo 1: Esquema geral de um Algoritmo Evolutivo

```
1  início
2      Inicializa a população;
3      Calcula a aptidão dos indivíduos;
4      enquanto Critério de parada não for satisfeito faça
5          Seleciona indivíduos para reprodução;
6          Aplica a combinação entre os indivíduos selecionados;
7          Aplica a mutação nos indivíduos resultantes;
8          Calcula a aptidão dos indivíduos gerados;
9          Seleciona indivíduos para a próxima população;
10     fim enqto
11 fim
```

1992) e a Programação Genética Cartesiana (Miller, 2011). Em geral, estas abordagens possuem muitas semelhanças por compartilharem das características definidas para os AEs, diferindo em suas propostas de estruturação de indivíduos ou operadores genéticos, por exemplo.

Dentre os benefícios do uso de AEs, destacam-se a possibilidade de busca por soluções de um problema sem uma definição prévia de sua estrutura exata (Poli et al., 2008) e sua capacidade de explorar espaços de busca complexos e com possivelmente diversos mínimos locais sem convergir prematuramente para um deles (Galván and Mooney, 2020).

3.1.3 Programação Genética Cartesiana

A Programação Genética Cartesiana (CGP) é uma técnica de Programação Genética originalmente proposta por Miller e Thomson (2000), em que a representação dos indivíduos é tipicamente dada por grafos direcionados acíclicos. Cada nó codifica um gene de função através de uma tabela de referência (por exemplo: 0 para soma, 1 para subtração e 2 para multiplicação). As conexões entre os nós são genes de conexão, e determinam a origem das entradas de cada nó, enquanto o resultado obtido em alguns destes são os genes de saída, representando o resultado final da CGP. Um exemplo desta representação é mostrado na Figura 3.3, em que existem três nós, três dados de entrada e duas saídas. Na representação abaixo do grafo, os nós cinzas contém códigos para funções, como já indicado anteriormente, os brancos são as conexões entre os nós e os quadrados pretos

representam as saídas, armazenadas como os índices dos nós de onde os resultados são obtidos (neste caso, os nós de índice 3 e 4). Nota-se que, caracteristicamente, nem todos os nós da CGP precisam contribuir para a saída do programa, sendo considerados "inativos". No exemplo, o nó 5 está inativo.

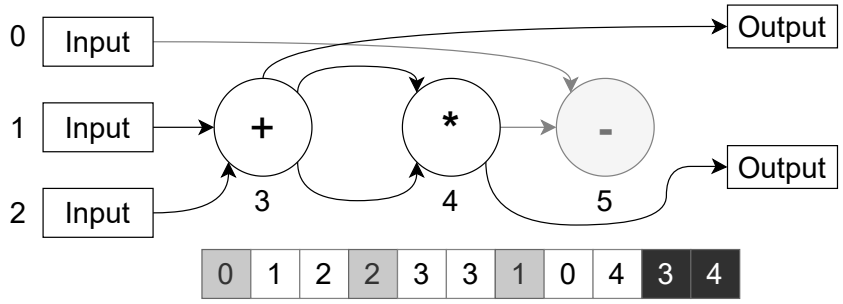


Figura 3.3: Exemplo de um indivíduo da CGP. Adaptado de Melo Neto et al. (2018).

A mutação na CGP padrão pode ser executada de duas formas: pontual ou probabilisticamente (Miller, 2019). Na mutação pontual, uma certa porcentagem pré definida de genes do pai é alterada para gerar os filhos. Já na mutação probabilística, cada gene pode ser alterado para um outro valor válido, dada uma probabilidade pré-determinada. Esta segunda abordagem tende a ser computacionalmente mais custosa, dado que todos os genes devem ser considerados, enquanto a primeira pode ser limitada no número de genes alterados.

Uma característica importante da mutação na CGP é a possibilidade de mutações ocorrerem em nós inativos, não causando nenhuma alteração efetiva no resultado do indivíduo. Algumas técnicas foram propostas por Goldman e Punch (2013) para lidar com este problema. A primeira, *skip*, consiste em comparar o filho gerado com o pai e, caso seus genes ativos sejam iguais, a avaliação não é feita no filho. Já a abordagem *accumulate*, ao identificar que um filho é idêntico ao pai, entra em um processo iterativo para alterar o filho até que este seja efetivamente diferente. Finalmente, a abordagem *single* altera o processo de mutação de forma que os genes a serem alterados são sorteados aleatoriamente até que um gene ativo seja alterado, dando fim ao processo e garantindo que o filho seja diferente do pai. Resultados experimentais mostram que estas abordagens apresentam um desempenho semelhante entre si, mas superior à mutação padrão.

Considerando a CGP padrão, não há a aplicação de recombinação entre in-

divíduos, uma vez que estudos preliminares indicaram uma baixa eficiência desta etapa em melhorar os indivíduos da CGP (Miller, 2011). Além disso, uma estratégia evolutiva comumente usada para evoluir a população da CGP é a $(1 + \lambda)$. Nesta, a partir da população de tamanho $(1 + \lambda)$, o melhor indivíduo da geração corrente é selecionado para gerar λ filhos através de mutações. Na próxima geração, o melhor indivíduo é escolhido dentre o pai e os filhos gerados, dando prosseguimento à geração dos novos indivíduos.

Assim sendo, os parâmetros base da CGP são λ, a probabilidade de mutação, o número máximo de nós, a aridade dos nós, e o conjunto de funções. Um pseudo código do processo de busca da CGP é mostrado no Algoritmo 2.

Algoritmo 2: Algoritmo de busca da CGP

```
1  início
2      Inicializa os parâmetros (λ, PROB_MUT, MAX_GEN, etc) ;
3      Cria a população inicial com (1 + λ) indivíduos aleatoriamente;
4      Avalia a aptidão dos indivíduos;
5      α ← indivíduo com melhor aptidão;
6      for gen = 1 até MAX_GEN do
7          for i = 1 até λ do
8              Avalia a aptidão do indivíduo i;
9              se Acurácia de i ≥ Acurácia de α então
10                 α ← i;
11             fim se
12         end for
13         Cria a próxima geração com α e com λ cópias de α aplicando
           mutações nas topologias e pesos;
14     end for
15 fim
```

3.1.4 Características da Neuroevolução

Considerando a dificuldade de configurar manualmente as topologias de RNAs, a definição automática destas topologias através de AEs vem ganhando popularidade, dados os benefícios destes algoritmos. Esta combinação de técnicas compõe a área da Neuroevolução.

Como pode ser observado na Figura 3.4, as etapas presentes em um algoritmo evolutivo compõem, de forma geral, o processo neuroevolutivo. Por isso, as considerações sobre AEs quanto a definição da estrutura dos indivíduos, operadores genéticos e métricas de avaliação continuam relevantes neste contexto. Estas características, entretanto, são

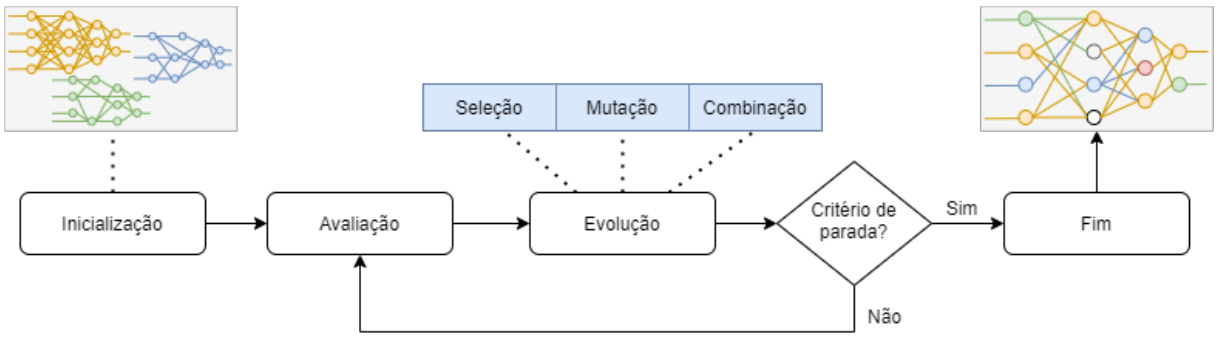


Figura 3.4: Esquema Geral de um algoritmo neuroevolutivo.

definidas com base nas propriedades das RNAs sendo evoluídas. Um exemplo simples é a aplicação de um AG para otimização dos pesos de conexões de uma rede neural (Yao, 1999). Nesse caso, o indivíduo precisa representar somente uma sequência de valores numéricos que representem estes pesos, e não necessariamente a rede completa, que é avaliada para cada combinação destes pesos, proposta pelo AG.

É importante considerar que existem desvantagens associadas à neuroevolução. O principal ponto negativo é o alto custo computacional associado aos AEs. Devido a sua generalidade, estas técnicas requerem um alto número de avaliações de função objetivo por geração, um problema que é agravado com o uso de grandes bases de dados (Turner, 2015; Galván and Mooney, 2020).

3.1.5 Programação Genética Cartesiana de Redes Neurais Artificiais

Através de uma adaptação da CGP padrão para o treinamento de RNAs, Khan et al. (2010) propuseram a Programação Genética Cartesiana de Redes Neurais Artificiais. Esta técnica estende a estrutura base da CGP pela adição de pesos associados às conexões entre os nós, como exemplificado na Figura 3.5. Na imagem, representam-se três nós, três dados de entrada e duas saídas. Na representação abaixo do grafo, os nós cinzas contém códigos para funções (F0, F1, F2), os brancos são as conexões entre os nós com seus respectivos pesos, os quadrados pretos são os índices para os nós de saída e o nó 5 está inativo. Nota-se que as principais diferenças para a CGP podem ser identificadas na presença dos pesos e nos genes de função que, neste caso, podem ser funções de ativação das Redes Neurais,

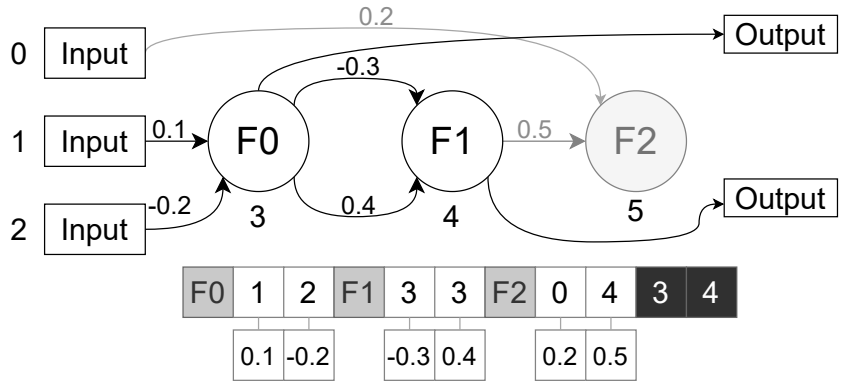


Figura 3.5: Exemplo de um indivíduo da CGPANN. Adaptado de Melo Neto et al. (2018).

de forma que a RNA é codificada diretamente, tendo toda sua topologia representada pelo indivíduo.

Os parâmetros e o processo evolutivo desta técnica são similares àqueles da CGP, exceto pela mutação, em que, adicionalmente às mudanças na topologia e na função de ativação, os pesos também podem ser alterados para novos valores dentro de um intervalo pré definido. Como apontado por seus autores, a técnica se mostrou competitiva em relação a outros algoritmos neuroevolutivos como NEAT, NeVA, ESP e CoSyNE por gerar soluções com um menor número de avaliações para os problemas estudados. Entretanto, Turner (2015) aponta que a CGPANN tende a obter resultados pouco satisfatórios em problemas de classificação, devido principalmente à evolução conjunto de topologia e pesos da rede, necessitando de um processo para o ajuste fino destes parâmetros a fim de melhorar seus resultados.

Um pseudo código da CGPANN é mostrado no Algoritmo 3, baseado naquele apresentado por Melo Neto et al. (2018). Neste, o processo evolutivo conta com avaliações sobre dados de treinamento, validação e teste, como aplicado no presente trabalho. Nesta configuração, o melhor indivíduo para os dados de treinamento é utilizado para gerar novos indivíduos (linha 18), enquanto o melhor sob os dados de validação é utilizado para verificar a acurácia final sob os dados de teste (linha 20). Neste caso, os dados de validação não são utilizados como decisão de parada para o algoritmo de busca, e sim para identificar indivíduos com uma boa acurácia e que não estejam sobre-ajustados aos dados de treinamento. A divisão dos dados em grupos de treinamento, validação e teste é detalhada no Capítulo 6.

Algoritmo 3: Pseudo Código da CGPANN

```
1  início
2      Inicializa os parâmetros (λ, PROB_MUT, MAX_GEN, etc) ;
3      Cria (1 + λ) indivíduos;
4      Avalia a acurácia dos indivíduos (TREINAMENTO);
5      Cria a rede α para armazenar a melhor (TREINAMENTO);
6      Cria a rede β para armazenar a melhor (VALIDAÇÃO);
7      for gen = 1 até MAX_GEN do
8          for i = 1 até λ do
9              Avalia a acurácia do indivíduo i (TREINAMENTO);
10             Avalia a acurácia do indivíduo i (VALIDAÇÃO);
11             se Acurácia de i ≥ Acurácia de α (TREINAMENTO) então
12                 α ← i;
13             fim se
14             se Acurácia de i ≥ Acurácia de β (VALIDAÇÃO) então
15                 β ← i;
16             fim se
17         end for
18         Cria λ cópias de α e aplica mutações nas topologias e pesos;
19     end for
20     Avalia a acurácia de β (TESTE);
21 fim
```

3.2 Computação de Alto Desempenho

Uma alternativa para lidar com o alto custo computacional é pela aplicação de técnicas de computação de alto desempenho. Este tema, entretanto, tange uma ampla gama de técnicas, conceitos, ferramentas e algoritmos que fogem do escopo deste trabalho. Por isso, nas subseções seguintes, os principais tópicos relacionados ao trabalho aqui desenvolvido são brevemente introduzidos. Assim, discute-se o OpenCL, ferramenta utilizada para a geração do código paralelo, as unidades de processamento gráfico e algumas técnicas de paralelismo sobre programas genéticos.

3.2.1 OpenCL

O OpenCL (*Open Computing Language*) é um padrão aberto para programação paralela em plataformas heterogêneas (Khronos, 2011). Este *framework* permite o desenvolvimento de aplicações para dispositivos físicos de diferentes tipos, tais como CPUs e GPUs de diferentes fabricantes (Gaster et. al., 2013). Aqui, esta ferramenta é brevemente introduzida com o objetivo de auxiliar no entendimento das técnicas paralelas desenvolvidas e

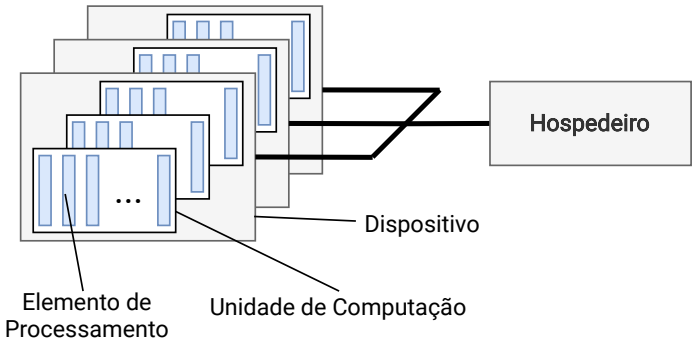


Figura 3.6: Organização da plataforma OpenCL, adaptado de Khronos (2011).

do funcionamento dos dispositivos utilizados.

O modelo de plataforma do OpenCL é composto por um dispositivo hospedeiro (*host*) conectado a diversos dispositivos. Cada um destes dispositivos, independente do seu tipo (CPU ou GPU) são compostos por uma ou mais unidades de computação - do inglês, *compute units* (CU) - que correspondem aos núcleos do processador. Cada uma destas unidades é formada por um ou mais elementos de processamento - *processing elements* (PE) - onde as instruções são efetivamente executadas. A Figura 3.6 representa esta organização.

A execução do código OpenCL é dividida entre o código que executa no sistema hospedeiro e aquele executado nos dispositivos (Khronos, 2011). O programa do hospedeiro executa normalmente na CPU, e é responsável por gerenciar a execução dos programas paralelos, controlando os dispositivos, os objetos de memória e enfileirando instruções a serem executadas em paralelo. Estas operações são disponibilizadas através da interface de programação de aplicações (API) do OpenCL. As instruções que executam nos dispositivos são chamadas de *kernels*. Estas são funções escritas em OpenCL C, um subconjunto de instruções da especificação C99, que executam o código em paralelo nos elementos de processamento.

A execução dos *kernels* ocorre em um espaço N-dimensional, que pode ter uma, duas ou três dimensões. Cada elemento independente corresponde a um item de trabalho (*work-item*), que executa o código paralelo sob diferentes dados. Estes itens são agrupados em grupos de trabalho (*work-groups*), que podem ser sincronizados e dividir um mesmo espaço de memória. Quanto à nomenclatura, o número de itens dentro de um grupo de

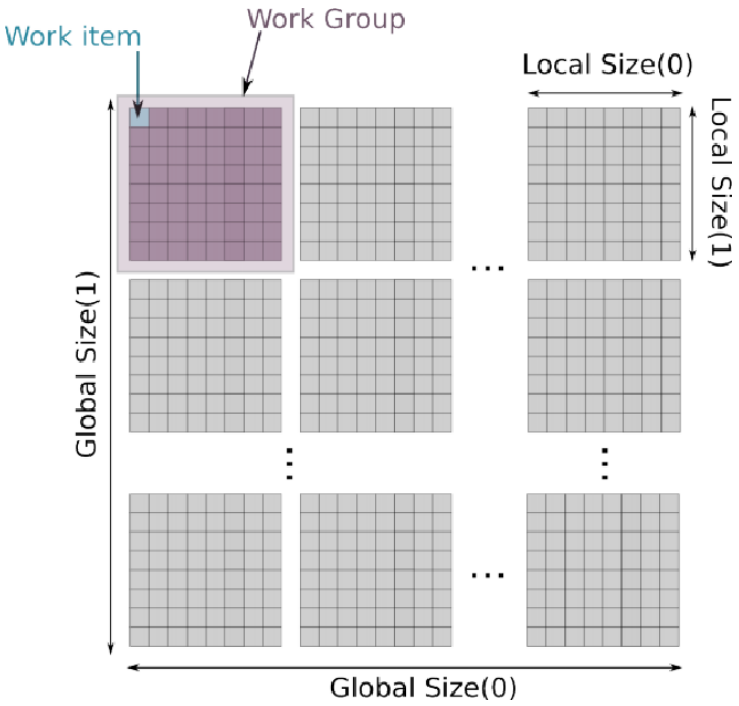


Figura 3.7: Abstração da organização dos dispositivos em itens e grupos de trabalho, obtido de Tompson e Schlachter (2012).

trabalho é chamado de tamanho local (*local-size*), enquanto o número total de itens de trabalho utilizados é o tamanho global (*global-size*). Este conceito é exemplificado na figura 3.7.

Uma outra abstração importante é o modelo de memória. Esta é dividida em quatro espaços: a memória global, a constante, a local e a privada (Khronos, 2011). A memória global é acessível por todos os itens e grupos de trabalho, e é alocada pelo hospedeiro. A memória constante faz parte da memória global, mas serve somente para leitura e, em alguns casos, pode ser implementada como *cache*, sendo mais eficiente. A região da memória local é compartilhada dentro de um mesmo grupo de trabalho, enquanto a memória privada remete a um único item de trabalho. Estes espaços de memória estão presentes nos dispositivos paralelos, enquanto a memória do hospedeiro, correspondente à memória disponível em CPU durante a execução do código sequencial, existe separadamente e instruções para leitura e escrita entre estes dois espaços são necessárias. O modelo de memória descrito é esquematizado na Figura 3.8.

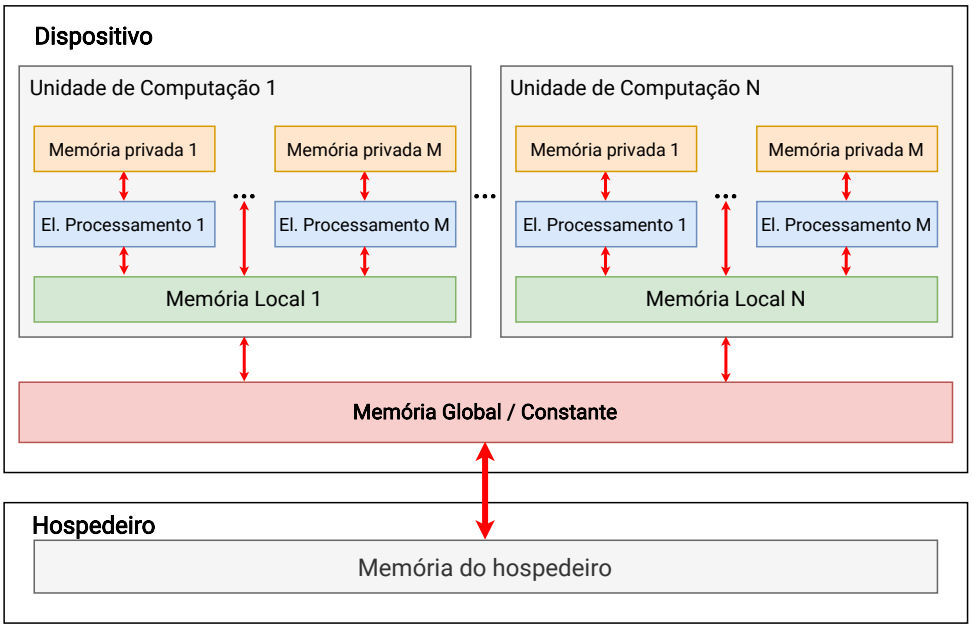


Figura 3.8: Organização da memória no OpenCL. Adaptado de Khronos (2011).

3.2.2 Unidades de Processamento Gráfico

Associado ao paralelismo, o uso de Unidades de Processamento Gráfico (GPUs) para a redução do tempo de processamento de aplicações gerais, não necessariamente relacionadas a gráficos, tem mostrado resultados promissores. O modelo SIMD (*Single Instruction, Multiple Data*) destas placas permitem um paralelismo massivo sobre dados, característica relevante para aplicações de aprendizado de máquina (Steinkraus et al., 2005).

A arquitetura de GPUs é composta por uma grande quantidade de núcleos simples, correspondentes a elementos de processamento. Todos estes elementos dentro de um grupo de trabalho executam a mesma operação por ciclo, mas sobre dados diferentes, caracterizando o modelo SIMD. Em relação às Unidades de Processamento Central (CPUs), as GPUs possuem uma quantidade reduzida de memória e cache, de forma que esta é uma característica importante a ser considerada durante a elaboração de algoritmos na GPU. A comparação da organização destes processadores é mostrada na Figura 3.9. Nota-se a presença de uma estrutura de controle mais complexa e uma quantidade grande de cache em CPUs, necessárias para o paralelismo de instruções presentes neste tipo de processador.

Uma característica importante das GPUs é que seus itens de trabalho (ou *threads*) estão sempre agrupados em um tamanho pré definido. Por exemplo, em GPUs da NVI-

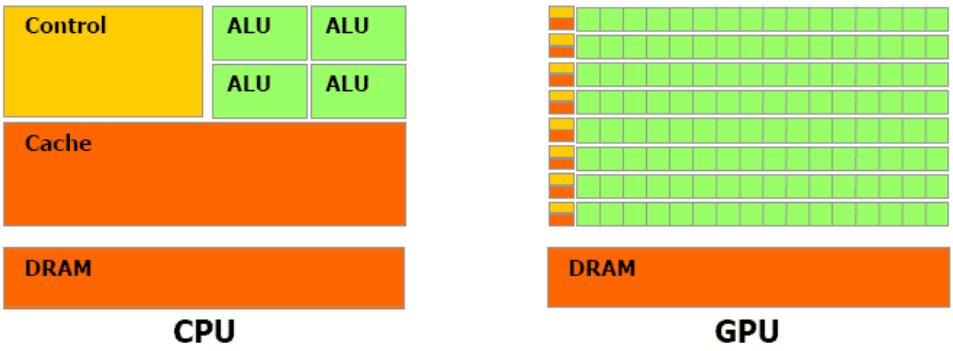


Figura 3.9: Comparação da estrutura de CPUs e GPUs.[2]

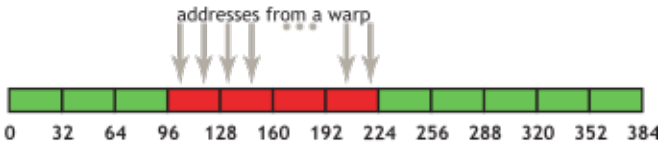
DIA, 32 itens são agrupados em um grupo chamado de *warp*, enquanto em arquiteturas da AMD, 32 ou 64 itens são agrupados em grupos denominados *wavefronts* (aqui vamos utilizar a nomenclatura *wavefront* por ser a mais comum ao OpenCL). Nesta organização, todos os itens de um *wavefront* executam exatamente a mesma instrução a cada instante de forma paralela. Caso alguma instrução entre alguns destes itens seja divergente, as instruções de cada *thread* são executadas sequencialmente (NVIDIA, 2017). Este tipo de execução configura a arquitetura "single instruction, multiple threads" (SIMT) que, em linhas gerais, é uma extensão da arquitetura SIMD pela utilização de threads.

Existe um total de seis tipos de espaço de memória em GPUs: de registrador, constante, compartilhada, local, global e de texturas (Mei e Chu, 2016). Em geral, estes espaços são diretamente mapeáveis à abstração fornecida pelo OpenCL.

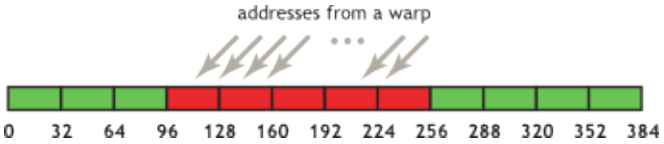
A memória global é a que possui maior tamanho e é acessada pelos elementos de processamento por transações de tamanho fixo, como 32, 64 ou 128 bytes. Esta é uma memória de leitura e escrita com alta latência e, em dispositivos mais recentes, dotadas de sistema de cache. A principal característica desta memória é seu acesso coalescido (agregado, aglutinado). Isso quer dizer que, quando os elementos de processamento dentro de um *wavefront* precisam acessar a memória global, estas requisições são aglomeradas no mínimo de transações necessárias para suprir as requisições (NVIDIA, 2017). Assim, caso estes acessos sejam dispersos, uma grande quantidade de dados não utilizadas são carregados, dado que as leituras sempre ocorrem em blocos de tamanho fixo. A Figura 3.10 ilustra uma situação simples em que o desalinhamento dos endereços acessados,

[2]Disponível em: <https://sites.google.com/site/daveshshingari/explorations/computer-architecture/gpu-architecture>

representados pelas setas, gera um acesso extra à memória, representado pelos blocos vermelhos. Em casos de acessos dispersos, este número pode aumentar significativamente.



(a) Acesso coalescido.



(b) Acesso sequencial, mas desalinhado.

Figura 3.10: Número de segmentos necessários para atender a uma requisição em um acesso coalescido (a) e um desalinhado (b). Obtido de NVIDIA (2013).

A memória de texturas existe na mesma região que a memória global, apresentando também uma alta latência. Além disso, apresenta um sistema de cache e funciona somente para leitura. Sua principal diferença é que seu cache é otimizado para acessos com localidade 2D, ou seja, elementos de um grupo de trabalho que acessam endereços de textura próximos tendem a ter um desempenho melhor. Portanto, ela é indicada para casos em que o acesso à memória não seja coalescido (caso contrário, o uso de memória global pode ser melhor) (NVIDIA, 2017).

A memória compartilhada corresponde à memória local no OpenCL, acessível somente aos elementos de processamento dentro de uma unidade de computação. Esta memória é mais rápida que a memória global, porém disponível em menor quantidade. Para explorá-la, algumas técnicas transferem uma única vez dados da memória global para a compartilhada, utilizando-a para os acessos subsequentes.

3.2.3 Programação Genética Paralela

Koza (1992) apresenta de forma teórica as principais maneiras pelas quais um método de programação genética é naturalmente paralelizável. Na primeira delas, o paralelismo sob o conjunto de dados pode ser explorado, dado que as execuções de um indivíduo sob as diferentes entradas são independentes, como exemplificado na Figura 3.11. Nota-se, entretanto, que uma etapa para acumular os resultados parciais é necessária, dado

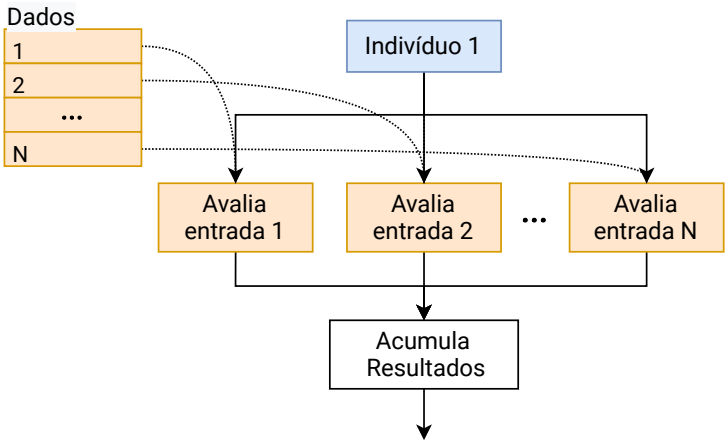


Figura 3.11: Esquema do paralelismo sob dados para PGs. Adaptado de Koza (1992).

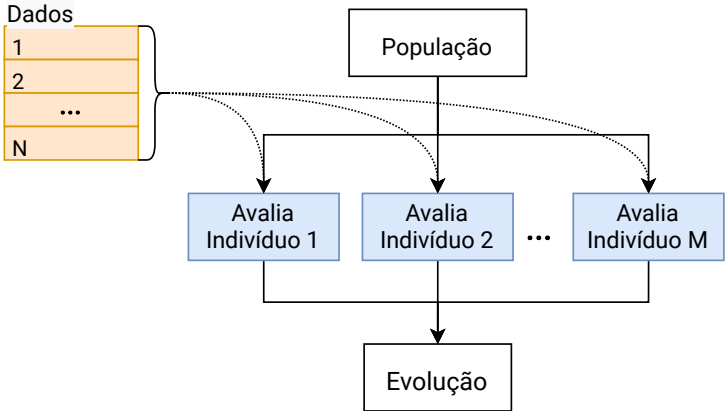


Figura 3.12: Esquema do paralelismo sob indivíduos para PGs. Adaptado de Koza (1992).

que a qualidade final do indivíduo depende de todos os seus resultados sob as diferentes entradas do conjunto de dados. Já a segunda abordagem trata do paralelismo sob a população, uma vez que cada indivíduo pode ser avaliado independentemente dos outros, como esquematizado na Figura 3.12.

No contexto da neuroevolução e do presente trabalho, a GPU é utilizada de forma a gerar um paralelismo massivo sobre a etapa de avaliação do algoritmo evolutivo, permitindo a paralelização tanto em nível de indivíduos, quanto em nível de dados, como exemplificado na Figura 3.13. Os indivíduos (modelos) podem ser mapeados para as unidades de computação da GPU, que podem avaliar os indivíduos simultaneamente sobre as entradas do conjunto de dados em seus elementos de processamento, associando as duas abordagens descritas anteriormente.

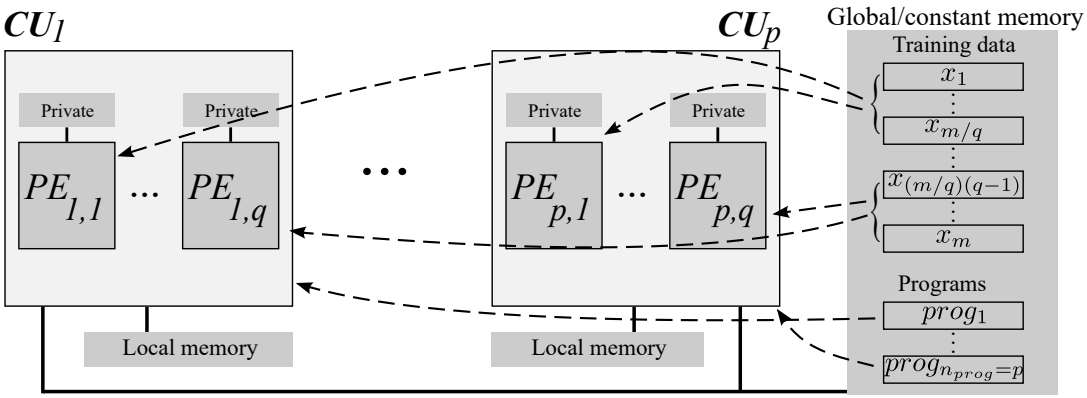


Figura 3.13: Esquema do uso da GPU para a avaliação paralela dos indivíduos de um AE. Adaptado de Russo et al. (2017).

3.3 Métricas

Neste trabalho, algumas métricas de desempenho são utilizadas para a análise dos resultados. Portando, uma breve introdução se mostra necessária a fim de garantir a compreensão dos métodos aplicados. Estas métricas são utilizadas em problemas de classificação a fim de verificar efetivamente a capacidade preditiva do modelo.

Considerando um problema de classificação binária, temos duas opções de classificação, uma positiva e uma negativa (Shmueli, 2019). Por exemplo, se estamos interessados em classificar, a partir de dados médicos, se uma pessoa está ou não doente, classificamos positivamente as amostras que indicam a doença e negativamente as amostras de pessoas saudáveis.

Neste caso, existem dois erros que podem ser cometidos durante a classificação: um falso positivo (FP), quando classifica-se uma amostra como positiva quando esta é, na verdade, negativa; ou um falso negativo (FN), em que uma amostra positiva é classificada como negativa (Murphy, 2012). De forma similar, ao classificarmos uma amostra corretamente, a designamos como verdadeiramente positiva (TP) ou verdadeiramente negativa (TN). Estes dados podem ser usados para construir a chamada "matriz de confusão", que resume a capacidade do modelo em classificar corretamente as diferentes classes, como exemplificado na Figura 3.14.

A precisão de um modelo (Eq. 3.4) determina sua capacidade em, dada uma amostra positiva, classificá-la corretamente, correspondendo à proporção, dentre amostras

		Classe Esperada	
		Positivo	Negativo
Classe Predita	Positivo	TP	FP
	Negativo	FN	TN

Figura 3.14: Exemplo da estrutura de uma matriz de confusão.

positivamente classificadas, que realmente são positivas. Já a revocação (Eq. 3.5) - do inglês, *recall* - corresponde à proporção entre as amostras classificadas corretamente como positivas e o número de amostras verdadeiramente positivas, ou seja, a proporção de itens positivos que são corretamente classificados. A acurácia (Eq. 3.6) indica a proporção de itens que foram corretamente classificados, seja positiva ou negativamente.

Uma outra métrica comumente utilizada é a f_β-*score* (Eq. 3.7). Esta é a média harmônica ponderada entre a precisão e revocação, de forma que o parâmetro β determina a importância maior ou menor para cada um destes valores. Nota-se para β entre 0 e 1, maior a importância da precisão, valores maiores que 1 beneficiam a revocação e para $\beta = 1$, ambas as métricas são igualmente consideradas.

$$Precisão(P) = \frac{TP}{TP + FP} \tag{3.4}$$

$$Revocação(R) = \frac{TP}{TP + FN} \tag{3.5}$$

$$Acurácia(A) = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.6}$$

$$F_\beta = (1 + \beta^2)\frac{Precisão \times Revocação}{\beta^2 \times Precisão + Revocação} \tag{3.7}$$

É importante considerar que este tipo de análise pode ser naturalmente aplicado à problemas multiclasse. Nesta situação, entretanto, as métricas anteriormente descritas são calculadas parada cada classe específica. A Figura 3.15 mostra um exemplo de uma matriz de confusão para um problema multiclasse em que poderíamos estar classificando

se uma imagem é de um carro, uma moto ou de um ônibus. Neste sentido, tomando a classe "carro" como exemplo, podemos calcular a precisão, a revocação e os *f-scores* para a classe, assim como a acurácia para o modelo como um todo, como segue:

		Classe Esperada		
		Carro	Moto	Ônibus
Classe Predita	Carro	10	5	4
	Moto	2	7	1
	Ônibus	1	2	8

Figura 3.15: Exemplo da estrutura de uma matriz de confusão para um problema multi-classe.

$$Precisão\_Carro = \frac{10}{10+5+4} = 0.53$$

$$Revocação\_Carro = \frac{10}{10+2+1} = 0.76$$

$$Acurácia = \frac{10+7+8}{40} = 0.62$$

$$F_1 = (1+1^2)\frac{0.53 \times 0.76}{1^2 \times 0.53 + 0.76} = 0.62$$

$$F_2 = (1+2^2)\frac{0.53 \times 0.76}{2^2 \times 0.53 + 0.76} = 0.69$$

4 Revisão bibliográfica

Neste capítulo, apresenta-se trabalhos relacionados aos problemas propostos na monografia. Assim, abordagens sobre a solução do Problema de Reconhecimento de Atividade são discutidas, assim como pesquisas sobre a CGPANN e em computação de alto desempenho aplicada sobre programas genéticos, em especial à CGP.

4.1 Problema de Reconhecimento de Atividade

Os primeiros trabalhos com alguma relação ao RAH data da década de 90, por um estudo de postura e reconhecimento de movimentos por Foerster et. al. (1999). Desde então, outros trabalhos vêm sendo propostos na área para em aplicações e através do uso de diferentes técnicas e algoritmos. Assim sendo, os trabalhos aqui apresentados sobre o RAH seguem uma ordem cronológica, focados nas técnicas empregadas para a solução do problema.

Os autores Kasteren et. al. (2008) aplicaram Modelos Ocultos de Markov (HMM) e Campos Aleatórios Condicionais (CRF) para o reconhecimento de atividades do dia-a-dia dentro de uma casa, no qual os HMM obtiveram os melhores resultados, com 79.4% de acurácia de classificação. Já Jatoba et. al. (2008) usaram outros classificadores, como k-vizinhos mais próximos (kNN), Naive Bayes (NB), Árvores de Classificação e Regressão (CART) e sistema de inferência neuro-fuzzy adaptável (ANFIS) na classificação de atividades semelhantes. Neste caso, as técnicas CART e ANFIS obtiveram os melhores resultados, com 86.6% e 85.9% de acurácia de classificação, respectivamente.

Em um trabalho posterior, Zhu e Sheng (2009) utiliza HMMs associadas à Redes Neurais Artificiais treinadas pela ferramenta de Redes Neurais do *software* MATLAB (*Neural Network Toolbox*), também para a classificação de atividades do dia-a-dia. Neste caso, entretanto, as RNAs são usadas em uma etapa anterior à classificação, alimentando o HMM com seus resultados, e não diretamente na classificação dos dados. Os autores relataram acurácias de classificação de até 92.50%.

No contexto do uso de RNAs, Sharma et al. (2008) e Oniga e Suto (2014) aplicaram estes classificadores sobre o problema de RAH. No primeiro trabalho, atividades gerais como descansar, caminhar e andar são classificadas, enquanto no segundo, o foco é identificar a postura corporal e dos braços. Em ambos os trabalhos, as RNAs foram treinadas pela ferramenta de Redes Neurais do MATLAB. Acurácias de classificação de 91.82% e 99% foram relatadas, respectivamente.

Uma tendência atual é a aplicação de métodos de Aprendizado Profundo em problemas de aprendizado de máquina e, neste sentido, algumas aplicações sobre o RAH vêm surgindo. Por exemplo, Duffner et. al. (2014) utilizou Redes Neurais Convolucionais para o reconhecimento de gestos a partir de dados de acelerômetros e giroscópios, obtendo resultados melhores que outros métodos do estado da arte para o conjunto de dados explorado, como HMM. Semelhantemente, Ronao and Cho (2016) também aplica CNNs no problema, em um estudo da capacidade desta técnica em extrair automaticamente características complexas do conjunto de dados, também obtendo melhores resultados que outros modelos da literatura, como NB, MLP e SVMs, sem a necessidade de definir características manualmente.

Focado no processo de seleção de características, Mohino-Herranz et al. (2019) aplica um algoritmo genético (AG) para selecionar diferentes quantidades destas características. O foco é a classificação do tipo de atividade sendo performada, como atividade mental, emocional, física e neutra, e as seguintes técnicas são usadas: Classificador de Mínimos Quadrados Linear (LSLC) e Quadrático (LSQC), Máquinas de Vetor de Suporte (SVM), Perceptrons Multicamada (MLP), Florestas Aleatórias (RFs), k-vizinhos mais próximos (kNN) e sua variação de centroides (CDNN). Os autores indicam que AGs são eficientes na seleção de atributos, diminuindo de 533 para 40 as características necessárias para obtenção do melhor erro de classificação, sendo este de 22.2% para um classificador linear de mínimos quadrados.

Existe uma grande variedade de problemas aos quais o RAH é aplicável. Nota-se que todos estes trabalhos tiveram sucesso em utilizar diferentes modelos para diferentes aplicações. Naturalmente, o processo de aquisição de dados, o ambiente de aplicação, os sensores utilizados e o processo de extração de características diferem substancialmente

entre os trabalhos, dificultando uma comparação justa entre os resultados apresentados. Por isso, na análise de trabalhos existentes na literatura, as técnicas, os conjuntos de dados utilizados e os resultados finais apresentam muitas diferenças, dificultando a comparação justa entre eles. Sendo assim, a análise destes trabalhos foca nos métodos computacionais empregados a fim de compará-los com a proposta do presente trabalho.

Entretanto, ressalta-se que uma característica em comum é a descrição do processo de aquisição de dados e da seleção de características que, entretanto, não são diretamente estudados no presente trabalho, dado que o conjunto de dados utilizado já foi previamente pré processado por Mohino-Herranz et al. (2019). Aqui, dada a popularidade dos métodos de aprendizado profundo e a aplicação de diferentes técnicas para o RAH, propõe-se a aplicação da CGPANN para solucioná-lo e analisar seu desempenho neste contexto.

4.2 Computação de Alto Desempenho

A CGPANN foi inicialmente proposta por Khan et al. (2010) e, desde então, tem sido aplicada com sucesso em diferentes problemas, como na detecção de câncer de mama (Ahmad e Khan, 2012), na reconstrução de sinais de áudio (Khan e Khan, 2017) e na previsão de sobrecargas elétricas Khan e Arshad (2016). O foco destes trabalhos, entretanto, é na resolução dos problemas e na acurácia dos modelos, enquanto detalhes sobre o custo computacional e o tempo de execução não são discutidos. Entretanto, este ainda é um problema corrente no que tange Algoritmos Evolucionários como a CGPANN, ainda que programas genéticos sejam naturalmente paralelos.

Em um dos primeiros trabalhos sobre PG, Koza (1992) descreve as principais abordagem de paralelismo sobre esta técnica. A primeira é o paralelismo sobre o conjunto de dados, em que as avaliações de função objetivo são executadas em paralelo. A segunda trata do paralelismo sobre a população, em que os indivíduos da PG são executados em paralelo.

Em trabalhos posteriores, ambas as abordagem foram aplicadas simultaneamente com sucesso. Em um deles, (Robilliard et. al., 2009; Robilliard et al., 2009) utilizaram GPUs e a linguagem CUDA para explorar o paralelismo de PGs baseados em árvores,

resultando em uma aceleração de até 80x em relação à execução sequencial.

Entretanto, um dos primeiros trabalhos a propor o uso de GPUs foi apresentado por Yu et. al. (2005). Neste, tanto a avaliação quanto a evolução dos indivíduos foram implementadas para a GPU, e os indivíduos eram representados como texturas 2D na placa gráfica. Em relação ao código em CPU, os autores relataram uma aceleração de até 17x. Uma técnica similar de paralelismo sobre dados e indivíduos foi aplicada por (Augusto e Barbosa, 2013), utilizando GPUs e o *framework* OpenCL. Neste, PGs baseados em árvores também foram utilizados e uma aceleração de até 126x foi atingida em comparação ao código sequencial.

A grande parte dos trabalhos que tratam de técnicas paralelas de programação genética utilizam a estrutura de árvores (Chitty, 2017), de forma que são poucos os trabalhos que aplicam a CGP neste contexto. Utilizando esta estrutura, Harding e Banzhaf (2007) implementa uma abordagem de paralelismo sob dados em GPUs, atingindo uma aceleração de 22.38x sobre o código em CPU. Em um trabalho seguinte, Harding e Banzhaf (2009) propõe uma implementação da CGP distribuída, também utilizando GPUs. Neste caso, o código CUDA é gerado, distribuído por um *cluster* e compilado em tempo de execução. Em ambos estes trabalhos, entretanto, a CGP é utilizado de forma arbitrária, de forma que os autores indicam que os resultados poderiam ter sido obtidos utilizando outros tipos de programação genética, como a linear ou a baseada em árvores.

Em um trabalho um pouco mais recente, (Vašíček e Slanỳ, 2012) traduz a fase de avaliação dos indivíduos da CGP para um código de máquina binário para CPUs. Neste caso, a CGP é utilizada por poder ter seus nós avaliados de forma linear, ao invés de uma avaliação recursiva de sua estrutura, como geralmente é aplicado para técnicas que utilizam uma estrutura de árvore, em contraste com a representação em grafos da CGP. A técnica atingiu uma aceleração de até 177x em comparação à abordagens em que as estruras são interpretadas, sem a compilação de código de máquina.

5 Métodos Propostos

Neste capítulo, os métodos propostos para o trabalho são apresentados. O uso da CGPANN para o RAH é brevemente introduzido e dá-se ênfase às estruturas de dados desenvolvidas para a CGPANN paralela.

A aplicação da CGPANN para o RAH é feita de forma direta, utilizando o conjunto de dados "*Activity recognition using wearable physiological measurements*", previamente discutido no Capítulo 2. Neste caso, todo o conjunto de 4480 instâncias e 533 características é utilizado como entrada para o algoritmo. Os indivíduos da CGPANN representam os modelos para o RAH, que são inicialmente gerados de forma aleatória e automaticamente evoluídos, através do fluxo do algoritmo da CGPANN.

As técnicas paralelas implementadas para a CGPANN adotam o paralelismo simultâneo sobre dados e sobre a população, como previamente descrito no Capítulo 3. Sendo assim, propõe-se um total de oito estruturas de dado para representar os indivíduos da CGPANN, objetivando principalmente explorar características específicas das GPUs, como a latência de acesso à memória global e sua memória de texturas.

5.1 Padrão

A primeira estrutura de dados proposta é referida como "Padrão". Esta consiste em nós contendo índices inteiros para suas conexões de entrada, seja do conjunto de dados ou de outros nós, assim como uma lista dos pesos em ponto flutuante associados a cada entrada. Estes nós são indexados por uma lista de índices na estrutura principal, que também armazena os índices dos nós de saída e estruturas auxiliares, como uma lista de nós ativos. Esta estrutura é representada na Figura 5.1.

Esta primeira proposta não possui melhorias quanto ao espaço de memória ocupada ou ao número de acessos à memória global da GPU. Entretanto, esta é a estrutura utilizada em códigos sequenciais, além de ser usada como base para as representações descritas à seguir, servindo de referência para comparações de performance com as demais.

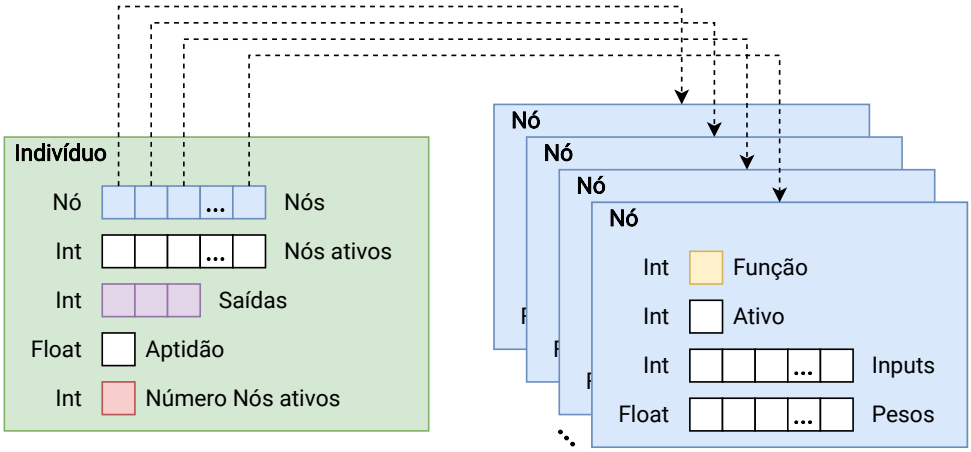


Figura 5.1: Exemplo da estrutura da CGPANN padrão.

5.2 Compacta

A abordagem proposta para reduzir os acessos à memória global da GPU consiste no uso da estrutura "Compacta". Esta representação possui a mesma organização da estrutura "Padrão", mas os campos que armazenam os índices de entradas e de nós ativos são compactados em memória. Nota-se que estes índices, armazenados como inteiros de 32-bits sem sinal, são limitados ao valor máximo do número de nós somado ao número de entradas possíveis do conjunto de dados. Desta forma, para representá-los completamente em memória, nem todos os 32-bits ocupados são efetivamente necessários. Para fazer uso dessa característica, dois valores distintos de 16-bits são armazenados em um único endereço de 32-bits, reduzindo pela metade o espaço em memória necessário para estes campos.

Esta técnica faz uso de operações bit-a-bit para deslocar e aplicar máscaras aos bits relevantes na localização adequada, permitindo compactar e recuperar estes valores. Por exemplo, dados dois inteiros sem sinal A e B. Primeiro desloca-se A por 16 bits para a esquerda (A << 16), preenchendo os bits mais significantes com zeros. Feito isso, aplica-se a operação bit-a-bit *AND* entre (A << 16) e B ((A << 16) AND B). Este conceito é exemplificado na Figura 5.2.

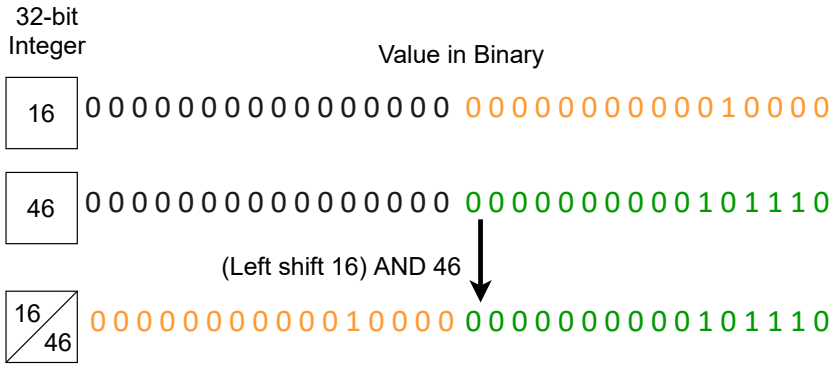


Figura 5.2: Exemplo da manipulação de memória para armazenarmos dois inteiros em uma mesma posição de memória.

5.3 Imagem

Com o objetivo de utilizar a memória de texturas da GPU, os indivíduos da CGPANN são mapeados em matrizes 2D que, na placa gráfica, são interpretadas como imagens. Esta abordagem permite que a seleção do número de canais RGBA utilizados para a imagem, de forma que determinam-se as estruturas de "Imagem R", "Imagem RG" e "Imagem RGBA".

No código em GPU, uma imagem é acessada através de seus pixels e, dessa forma, uma imagem que possui somente o canal R armazena um valor por pixel, enquanto uma RG possui dois valores e a RGBA, quatro. Estes valores podem ser configurados como inteiros de 16 ou 32 bits. Neste caso, utilizam-se valores inteiros de 32-bits cada.

A representação 2D dos indivíduos é exemplificada na Figura 5.3. Nesta, é possível observar que, devido ao formato 2D da imagem, alguns espaços são desperdiçados a fim de manter a estrutura com certa organização lógica. Além disso, como traduzimos os indivíduos da estrutura padrão para a imagem, somente os nós ativos que são efetivamente copiados. Já diferença entre as variantes R, RG e RGBA é mostrada na Figura 5.4, onde é possível notar (i) como os pixels são distribuídos em cada estrutura e (ii) a necessidade de inserirmos preenchimentos em alguns casos a fim de garantir que todos os pixels estejam com todos os canais completos.

As outras três estruturas estudadas são compostas pela combinação da abordagem "Compacta" com as de "Imagem", resultando nas propostas de Imagem Compacta R, RG e RGBA. Nestas situações, a estrutura "Padrão" é compactada com operações bit a bit e convertida para o formato de imagem antes de serem carregada para a GPU.

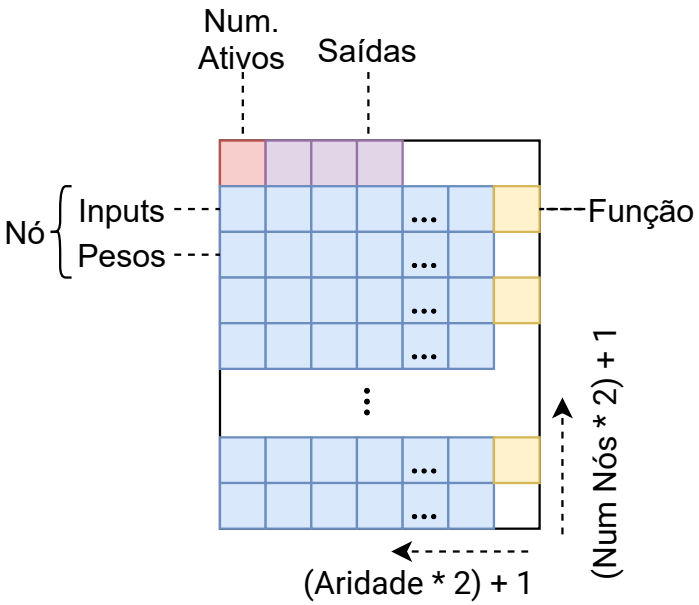


Figura 5.3: Exemplo da representação de um indivíduo como imagem na GPU.

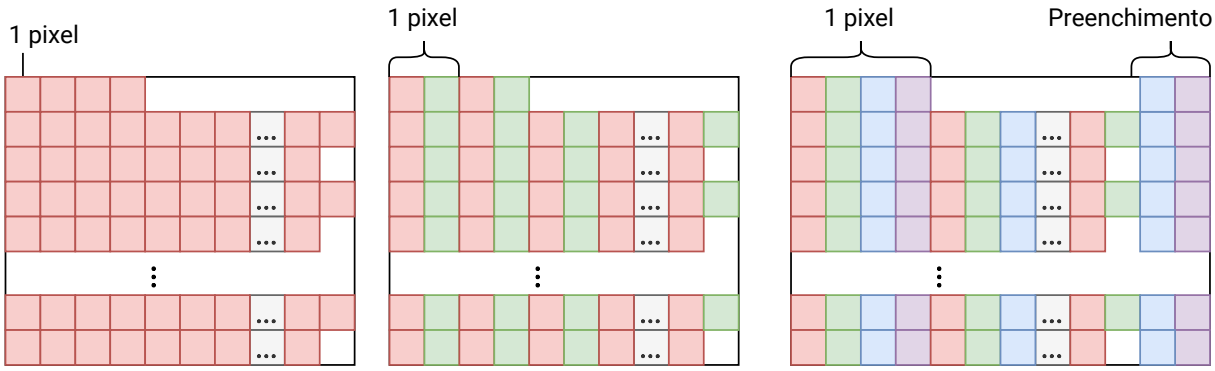


Figura 5.4: Exemplo da distribuição de pixels nas estruturas R, RG e RGBA, respectivamente.

6 Experimentos Computacionais

Este capítulo trata dos procedimentos metodológicos adotados no desenvolvimento do trabalho. Dois experimentos distintos são propostos neste contexto. Inicialmente descreve-se os procedimentos comuns a ambos e, nas seções seguintes, os procedimentos específicos a cada um deles.

Dada a natureza estocástica dos métodos desenvolvidos, valores como o tempo decorrido para executar o programa e a qualidade dos modelos encontrados quanto a sua acurácia de classificação variam entre diferentes execuções e, por isso, diversas execuções independentes para cada experimento são executadas a fim de se obter um maior número de dados a serem analisados e verificar a significância estatística dos resultados.

Quanto ao tratamento dos conjuntos de dados, a técnica de validação cruzada estratificada com "*k-folds*" é aplicada. Nesta, o conjunto de dados é dividido em k grupos de tamanho igual e com a mesma proporção de instâncias de cada classe nos grupos. Com isso, o algoritmo de treinamento é executado k vezes de forma que, em cada execução, um dos k grupos é utilizado como conjunto de teste, enquanto os outros $k-1$ são aleatoriamente divididos para os conjuntos de treinamento e validação. Dois conjuntos de dados diferentes são utilizados para os experimentos. Seus detalhes são descritos nas seções seguintes. Entretanto, em ambos os casos, o dados são divididos em subconjuntos de treinamento, validação e teste. Nesta configuração, o conjunto de treinamento é utilizado durante o processo evolutivo da CGPANN para o treinamento dos modelos e prosseguimento das gerações do algoritmo. Ao mesmo tempo, o conjunto de validação é utilizado para analisar esses modelos quanto a sua capacidade de generalização. Ao fim do processo evolutivo, o indivíduo com melhor acurácia quanto aos dados de validação é avaliado sobre o conjunto de teste, aferindo sua qualidade final. Este procedimento visa evitar o superajuste dos modelos aos dados em que foram treinados, assim como validar sua generalidade, através da avaliação sobre instâncias não utilizadas durante o treinamento.

As abordagens propostas são implementadas na linguagem de programação C++[3] e com a biblioteca OpenCL para o desenvolvimento e execução do código paralelo executado na GPU. Os experimentos são executados em uma máquina com o sistema operacional Windows 10 64-bit, usando o compilador g++ MinGW 5.3 e a biblioteca OpenCL 1.2 da NVIDIA, em uma CPU AMD Ryzen 5 3600X 6-Core 3.79GHz e uma GPU NVIDIA Geforce GTX 750Ti.

Em ambos os experimentos, os parâmetros da CGPANN indicados na Tabela 6.1 são utilizados. Estes foram definidos de acordo com o proposto por Melo Neto et al. (2018), a fim de permitir a comparação de resultados com aqueles apresentados na literatura. Em relação especificamente ao número de indivíduos filhos criados na estratégia evolucionária, seu valor é igual ao numero de núcleos disponíveis na GPU, de forma a utilizá-la de forma mais eficiente. Nota-se também que o número de nós corresponde aos nós internos da CGPANN, não considerando os nós de entrada e saída.

Tabela 6.1: Parâmetros utilizados para a CGPANN.

Parâmetro	**Valor**
Gerações	50000
Tipo de Mutação	Probabilística
Taxa de Mutação	0.05
Indivíduos	6
Estratégia evolucionária	(1+5)-ES
Número de Nós	500
Aridade dos Nós	20
Função de ativação	Função Sigmóide

6.1 Experimento 1: Análise de Desempenho Computacional

Para o primeiro experimento, propõe-se a análise de tempos de execução do algoritmo paralelo da CGPANN utilizando as diferentes estruturas de dados descritas no Capítulo 5. Neste momento, o tempo total de processamento é aferido, assim como o tempo decorrido exclusivamente durante a etapa paralela do algoritmo (tempo de *kernel*), referente à avaliação dos indivíduos.

[3]Código fonte disponível em https://github.com/bmarcosps/P-CGPANN

Utiliza-se a base de dados *Pima Indians Diabetes* disponível no *UCI Machine Learning Repository*[4]. As características desta base são resumidas na Tabela 6.2. Assim como os parâmetros da CGPANN, este conjunto foi utilizado de acordo com a metodologia proposta por Melo Neto et al. (2018), com a normalização linear dos dados para o intervalo [0, 1] e a aplicação da validação cruzada estratificada com $k = 10$, com a divisão de 7 folds para o conjunto de treinamento, 2 para validação e 1 para teste.

Tabela 6.2: Característica do Conjunto de Dados do Experimento 1.

Dataset	**Instâncias**	**Atributos**	**Classes**
Pima Indians Diabetes	768	8	2

Este conjunto é comumente utilizado para *benchmarks* em trabalhos de aprendizado de máquina e possui um número de instâncias representativo para a análise do desempenho computacional das diferentes técnicas propostas. É importante notar que em Melo Neto et al. (2018) a CGPANN é aplicada à esta base de dados e os modelos obtidos alcançam uma acurácia média de 0.7361 (ou 26.39% de erro médio). Assim, estes resultados não são detalhados aqui, uma vez que o foco deste experimento é a análise do tempo computacional e os resultados do algoritmo quanto à qualidade dos modelos não são alterados pelas técnicas aqui discutidas.

Os resultados obtidos são mostrados nas Tabelas 6.3 e 6.4. Cada linha corresponde a uma das estruturas de dados propostas de forma que: I_R, I_RG, e I_RGBA correspondem às estruturas em imagem, que utilizam a memória de texturas da GPU e I_R_C, I_RG_C, e I_RGBA_C são suas variantes compactas. Os tempos para a execução sequencial também são exibidos, de forma que seu tempo de *kernel* corresponde à etapa de avaliação dos indivíduos.

As medidas de tempo são feitas para cada configuração dos 10 *folds*, para três execuções independentes, resultando em 30 medidas para tempos totais e 30 para tempos de *kernel*. Assim, a média, mediana e desvio padrão levam em conta o tempo de execução para cada fold dentre os 30 executados, enquanto os tempos totais de cada tabela remetem à soma do tempo de execução destas 30 medidas realizadas para o tempo total e de kernel.

Nota-se, primeiramente, que os tempos de execução sequencial são muito superi-

[4]https://archive.ics.uci.edu/ml/index.php

ores às abordagens paralelas. Além disso, todas as estruturas propostas apresentam um tempo inferior à estrutura Padrão.

Observando os tempos totais mostrados na Tabela 6.3, o melhor desempenho é obtido ao utilizar a memória de texturas da GPU através da estrutura de Imagem RGBA. Este resultado indica o potencial da utilização da arquitetura de GPUs e de suas características específicas com o intuito de acelerar aplicações.

Em relação aos tempos de *kernel* exibidos na Tabela 6.4, o melhor resultado foi obtido pela representação em Imagem RG Compacta, ou seja, associando ambas as abordagens propostas. A divergência quanto ao resultado anterior pode ser explicada pelo tempo extra gasto para a alocação e conversão da estrutura Padrão para as outras representações antes da fase de avaliação. De qualquer maneira, indica-se que otimizar o número de consultas à memória, além do uso de diferentes tipos de memória, pode também resultar em ganhos de desempenho.

Em específico quanto à estrutura da CGPANN, é possível analisar que, dado que os acessos aos nós do grafo ocorrem de forma dispersa, a localidade 2D da memória de texturas ajuda a evitar que o padrão de acesso não coalescido prejudique o desempenho. Da mesma maneira, ao compactá-la, o número de requisições à memória global é reduzido, também melhorando o tempo de execução.

O tempo total exibido corresponde à execução do algoritmo evolutivo, com as etapas de inicialização da população, avaliação e evolução dos indivíduos. Já o tempo de *kernel*, como previamente mencionado, corresponde somente à etapa de avaliação, onde o paralelismo é aplicado. Observando o tempo gasto pelo algoritmo sequencial, a etapa de

Tabela 6.3: Resultados para os tempos totais (*host + kernel*).

Método	**Total(s)**	**Média(s)**	**Mediana(s)**	**Desv. Pad.(s)**
Sequencial	466.199, 69	15.539, 99	18239, 59	7095, 78
Padrão	9.688, 64	322, 954	336, 99	127, 93
Compacta	8.788, 47	292, 95	304, 12	106, 02
I_R	9.264, 74	308, 82	321, 43	117, 36
I_RG	8.751, 11	291, 70	303, 20	108, 56
I_RGBA	**8.651,99**	**288,40**	**299,55**	107, 53
I_R_C	9.515, 02	317, 17	329, 42	113, 18
I_RG_C	8.905, 37	296, 85	307, 63	**103,02**
I_RGBA_C	9.158, 86	305, 30	317, 01	105, 75

Tabela 6.4: Resultados para o tempo de *kernel*.

Método	**Total(s)**	**Média(s)**	**Mediana(s)**	**Desv. Pad.(s)**
Sequencial	464.141, 90	15.471, 40	18.168, 85	7.089, 27
Padrão	7.277, 13	242, 57	255, 44	121, 11
Compacta	6.086, 86	202, 90	213, 12	99, 34
I_R	6.647, 01	221, 57	232, 73	108, 70
I_RG	6.144, 29	204, 81	215, 04	99, 89
I_RGBA	6.064, 65	202, 15	212, 00	98, 70
I_R_C	6.543, 78	218, 13	229, 38	106, 50
I_RG_C	**5.926,13**	**197,54**	**207,17**	**96,35**
I_RGBA_C	6.135, 17	204, 51	214, 74	98, 87

avaliação representa 99% do tempo total. Entretanto, para o algoritmo paralelo Padrão, o tempo de avaliação engloba 75% do tempo total, de forma similar às outras estruturas paralelas analisadas. Esta redução da parcela paralelizável é explicada principalmente pelo tempo extra gasto nas abordagens paralelas para a preparação dos dispositivos, como as trocas de memória e conversão de estruturas. Entretanto, esta ainda representa a parcela de tempo mais significativa do algoritmo.

A Tabela 6.5 dispõe os *speedups* calculados para os resultados obtidos em tempos totais e de *kernel* das estruturas paralelas desenvolvidas em relação ao algoritmo sequencial. Esta métrica consiste na relação entre um tempo de execução de referência e o tempo da técnica sendo analisada, indicando o quanto mais rápido um algoritmo é em relação à referência. Neste caso, as considerações anteriores quanto ao ganho computacional do algoritmo sequencial para o paralelo são visíveis pelo valor de *speedup* para a estrutura padrão. Nota-se também que a estrutura I_RGBA obteve o melhor desempenho para o tempo total, com um *speedup* de 53.88, enquanto a estrutura I_RG_C atingiu o melhor desempenho para o tempo de *kernel*, com um *speedup* de 78.32.

6.2 Experimento 2: CGPANN para o Problema de RAH

No segundo experimento, propõe-se a aplicação dos modelos treinados pela CGPANN na resolução do problema de Reconhecimento de Atividade Humana. Assim, a acurácia dos modelos é utilizada como métrica de desempenho e a média dentre todas as execuções

Tabela 6.5: *Speedups* para os tempos totais e de *kernel* em relação ao algoritmo Sequencial.

Estruturas	***Speedup*** **Tempo Total**	***Speedup*** **Tempo de Kernel**
Padrão	48.12	63.78
Compacta	53.05	76.25
I_R	50.32	69.83
I_RG	53.27	75.54
I_RGBA	53.88	76.53
I_R_C	49.00	70.93
I_RG_C	52.35	78.32
I_RGBA_C	50.90	75.65

independentes é utilizada para comparação com os resultados apresentados por Mohino-Herranz et al. (2019).

O mesmo conjunto de dados proposto pelo autor, "*Activity recognition using wearable physiological measurements*", também disponível no *UCI Machine Learning Repository*, é utilizado no trabalho, assim como os procedimento metodológicos relacionados à configuração e execução de experimentos computacionais, a fim de possibilitar uma comparação justa entre os resultados. Suas características são resumidas na Tabela 6.6.

Tabela 6.6: Características do Conjunto de Dados do Experimento 2.

Dataset	**Instâncias**	**Atributos**	**Classes**
Activity Recognition	4480	533	4

Esta base de dados é previamente normalizada utilizando a média μ e o desvio padrão σ pelo método *z-score*, definido pela relação $z = \frac{x-\mu}{\sigma}$. Além disso, a validação cruzada estratificada com $k = 40$ é aplicada, de forma que cada *fold* corresponde a dados relacionados a uma pessoa da base original. Neste experimento, 30 folds são utilizados para o conjunto de treinamento, 9 para validação e 1 para teste e 10 execuções independentes são executadas. Estes procedimentos foram utilizados por Mohino-Herranz et al. (2019) e, por isso, são aplicados neste trabalho para permitir uma comparação justa entre os resultados.

Como previamente mencionado, a acurácia média dos modelos dentre todas as dez execuções independentes é utilizada como comparação aos resultados em Mohino-Herranz et al. (2019), uma vez que esta é a métrica adotada pelos autores. Algumas medidas estatísticas como a melhor acurácia encontrada, o desvio padrão, a mediana, matriz de

Tabela 6.7: Resultados para o erro médio dos modelos comparados e o mínimo, mediana e desvio padrão do erro da CGPANN.

Método	**Erro Médio (%)**	**Max. Atributos**	**Melhor Erro (%)**	**Mediana (%)**	**Desvio Padrão**
CGPANN	**33.35**	533	**7.14**	32.14	12.44
LSLC	**22.2**	40			
LSQC	26.2	40			
LINSVM	22.5	40			
RBFSVM	28.6	40			
MLP8	24.9	20			
MLP12	25.6	20			
MLP16	26.1	10			
kNN	28.7	10			
CDNN	27.0	5			
RF	25.5	20			

Tabela 6.8: Precisão, Revocação e F1-score obtidos pela CGPANN.

Classe	**Precisão**	**Revocação**	**F1-score**
Emocional	0.51	0.59	0.55
Física	0.87	0.92	0.89
Mental	0.51	0.43	0.47
Neutra	0.78	0.73	0.75

confusão e as métricas descritas no Capítulo 3 entre todas as execuções também são calculadas.

O erro de classificação médio obtido pela CGPANN, em comparação com os outros classificadores da literatura, é mostrado na Tabela 6.7. No trabalho de referência, o conjunto de dados passa por um processo prévio de seleção de características, limitando seu número para o treinamento dos modelos e, por isso, essa informação também é mostrada. O melhor erro, a mediana e desvio padrão dentre todas as execuções independentes da CGPANN também são mostrados.

Na Figura 6.1, as matrizes de confusão para o melhor classificador da literatura e para a CGPANN são mostradas. Os resultados foram calculados como a porcentagem de erros e acertos dentre os modelos finais de todos os folds, em todas as execuções independentes. Com estes dados, também foi possível calcular as métricas de precisão, acurácia, revocação e *F1-measure* previamente apresentadas. Estes resultados foram resumidos na Tabela 6.8.

		Classe Predita			
	Atividade	Neutra	Emocional	Mental	Física
Classe Verdadeira	Neutra	80,90%	5,50%	12,20%	1,30%
	Emocional	5,00%	72,60%	18,70%	3,80%
	Mental	7,70%	30,50%	58,90%	2,90%
	Física	0,10%	1,80%	2,30%	95,80%

(a) Matriz de confusão para LSQC usando 40 características (Mohino-Herranz et al., 2019)

		Classe Predita			
	Atividade	Neutra	Emocional	Mental	Física
Classe Verdadeira	Neutra	73,10%	13,50%	11,28%	2,13%
	Emocional	8,83%	59,14%	26,24%	5,79%
	Mental	10,66%	40,45%	42,75%	6,14%
	Física	1,11%	3,83%	3,46%	91,60%

(b) Matriz de confusão para a CGPANN usando 533 características.

Figura 6.1: Matriz de confusão apresentada pela literatura(a) e obtida no trabalho(b).

Tabela 6.9: Tempo de execução para as estruturas Padrão e Compacta para o *dataset* de RAH.

Método	Total(s)	Média(s)	Mediana(s)	Desv. Pad.(s)
Padrão	117.714,39	980,95	1.007,81	273,08
Compacta	**107.600,40**	**896,67**	**918,38**	**243,58**

Como pode ser observado, a CGPANN não obteve um erro médio melhor que as outras propostas da literatura neste experimento (Tabela 6.7). Entretanto, por não utilizar uma etapa de seleção de características antes do treinamento, o processo geral é simplificado, permitindo que o algoritmo encontre os atributos mais relevantes automaticamente durante o processo evolutivo.

Ainda assim, o melhor modelo encontrado pela CGPANN apresenta uma taxa de erro inferior àqueles da literatura, indicando que a CGPANN é promissora para encontrar bons modelos e que a qualidade geral dentre diferentes execuções ainda pode ser melhorada. Além disso, todos as 533 características foram utilizadas.

Quanto às métricas extraídas da matriz de confusão (Tabela 6.8), os valores de precisão e revocação para cada classe ficaram, em geral, próximos, não havendo uma tendência dos modelos a gerarem mais falsos positivos ou falsos negativos. Nota-se, como também indicado em Mohino-Herranz et al. (2019), que a atividade mais "fácil" de ser identificada é a física, seguida da neutra, emocional e mental.

Para o problema de RAH em geral, uma preferência por uma melhor precisão ou revocação depende da aplicação. Por exemplo, para um sistema de monitoramento médico remoto, uma revocação alta poderia ser mais importante, dado que situações importantes (como identificar a queda de um idoso) não podem apresentar falsos negativos. Já em sistemas de casas inteligentes e automação, por exemplo, uma precisão maior pode ser mais vantajosa, dado que falsos positivos podem gerar ações indesejadas no sistema.

Como um teste preliminar, três execuções independentes da CGPANN paralela com as estruturas Padrão e Compacta foram executadas. Os tempos obtidos são apresentados na Tabela 6.9. Neste caso, a proposta Compacta foi capaz de obter valores do tempo total, médio e de mediana inferiores à estrutura Padrão.

Com estes dados, também calcula-se o *speedup* alcançado pela estrutura Com-

Tabela 6.10: Speedups da estrutura Compacta em relação à Padrão.

Instâncias	**Atributos**	**Classes**	**Speedup**
768	8	2	1.10
4480	533	4	1.09

pacta em relação à Padrão. Uma comparação deste valor em relação ao número de instâncias, atributos e classes dos conjuntos de dados utilizados em ambos os experimentos é mostrada na Tabela 6.10. Nota-se que o *speedup* é um pouco inferior para o conjunto com mais instâncias. Entretanto, dada a análise somente sob estes dois conjuntos, não é possível estabelecer com precisão uma regra para o comportamento do ganho computacional em problemas de tamanhos diferentes.

7 Conclusão

Neste trabalho, questões pertinentes ao custo computacional de algoritmos evolutivos, especificamente à Programação Genética Cartesiana de Redes Neurais Artificiais, foram discutidas. Dentro desta temática, um algoritmo da GPANN paralela em Unidades de Processamento Gráfico foi analisado quanto ao seu tempo de processamento sob a perspectiva de oito estruturas de dados distintas, que foram propostas e implementadas no trabalho. Tais estruturas foram modeladas a fim de explorar características específicas das GPUs, visando diminuir o número de acessos à memória global e fazer uso da memória de texturas durante a avaliação dos indivíduos da programação genética.

Além disso, dada a crescente importância do problema de reconhecimento de atividade humana e sua ampla aplicabilidade, a técnica desenvolvida foi utilizada em sua resolução. A capacidade da CGPANN em resolvê-lo foi analisada em relação aos resultados presentes na literatura.

Os resultados obtidos quanto ao custo computacional da CGPANN paralela indicam que o uso de estruturas modeladas para GPUs gera ganhos quanto ao tempo de execução. Em específico, nota-se que o uso da memória de texturas para o armazenamento dos indivíduos é promissor, dado que este tipo de memória pode ajudar a minimizar o custo de acessos dispersos, como geralmente ocorre durante o caminhamento em grafos. Além disso, uma estrutura compacta, mesmo utilizando operações extras para codificar e decodificar informações, foi capaz de gerar melhorias no tempo de execução do algoritmo. Em relação ao tempo total, a estrutura Padrão obteve um redução de 97.92% em relação ao tempo sequencial, enquanto as estruturas Compacta e Imagem RGBA atingiram tempos 9.29% e 10.7% menores que a Padrão, respectivamente. De forma semelhante, sob o conjunto de dados do RAH, a abordagem Compacta obteve uma redução de 8.59% do tempo em relação à estrutura Padrão.

Nos experimentos desenvolvidos para o RAH, a CGPANN obteve um erro médio de 33.35%, que é superior aos classificadores comparados, sendo este 50.22% maior que o erro médio do melhor classificador da literatura. Assim, é reforçada a necessidade de

otimizações direcionadas à topologia das redes, a fim de melhorar o desempenho geral do algoritmo em diferentes execuções. Entretanto, o melhor modelo encontrado pela CGPANN apresenta um erro de 7.14%, que é 67.84% menor que o melhor caso médio da literatura, indicando que a técnica é promissora e capaz de encontrar modelos competitivos para problemas do mundo real. Nota-se também que a técnica dispensa procedimentos de seleção de características antes do treinamento dos modelos, em contraste com os métodos apresentados pela literatura.

Através dos resultados alcançados, é possível elencar como trabalhos futuros a aplicação de técnicas híbridas para um ajuste fino dos parâmetros da CGPANN, como seus pesos, a fim de melhorar a qualidade dos modelos gerados para o problema de RAH. Além disso, aplicar a técnica sob diferentes conjuntos de dados, validando-a em outros contextos também se mostra necessário. Quanto à CGPANN paralela, o uso de GPUs pode ser ainda mais explorado pela análise das estruturas propostas em dispositivos mais recentes, de diferentes fabricantes e com um maior número de unidades de computação e elementos de processamento, uma vez que estas características podem afetar a performance.

Bibliografia

Ahmad, A. M.; Khan, G. M. **Bio-signal processing using cartesian genetic programming evolved artificial neural network (cgpann)**. In: 2012 10th International Conference on Frontiers of Information Technology, p. 261–268. IEEE, Dec 2012.

Augusto, D. A.; Barbosa, H. J. Accelerated parallel genetic programming tree evaluation with opencl. **Journal of Parallel and Distributed Computing**, v.73, n.1, p. 86–100, Jan 2013.

Chen, K.; Zhang, D.; Yao, L.; Guo, B.; Yu, Z. ; Liu, Y. Deep learning for sensor-based human activity recognition: Overview, challenges and opportunities. **arXiv:2001.07416 [cs]**, Jan 2020. arXiv: 2001.07416.

Chitty, D. M. Faster gpu-based genetic programming using a two-dimensional stack. **Soft Computing**, v.21, n.14, p. 3859–3878, 2017.

Darwin, C. The evolution of species. **New York: The Modern Library**, 1859.

Duffner, S.; Berlemont, S.; Lefebvre, G. ; Garcia, C. **3d gesture classification with convolutional neural networks**. In: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), p. 5432–5436. IEEE, 2014.

Eiben, A.; Smith, J. **Introduction to Evolutionary Computing**. Natural Computing Series. Springer Berlin Heidelberg, 2015.

Foerster, F.; Smeja, M. ; Fahrenberg, J. Detection of posture and motion by accelerometry: a validation study in ambulatory monitoring. **Computers in Human Behavior**, v.15, n.5, p. 571–583, Sep 1999.

Gajurel, A.; Louis, S. J. ; Harris, F. C. Gpu acceleration of sparse neural networks. **arXiv:2005.04347 [cs]**, May 2020. arXiv: 2005.04347.

Galván, E.; Mooney, P. Neuroevolution in deep neural networks: Current trends and future challenges. **arXiv:2006.05415 [cs]**, Jun 2020. arXiv: 2006.05415.

Gaster, B. R.; Howes, L.; Kaeli, D. R.; Mistry, P. ; Schaa, D. **Chapter 2 - Introduction to OpenCL**, p. 15–38. Morgan Kaufmann, second edition. ed., 2013.

Goldberg, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Addison-Wesley Publishing Company, 1989. Google-Books-ID: 2IIJAAAACAAJ.

Goldman, B. W.; Punch, W. F. **Reducing wasted evaluations in cartesian genetic programming**. In: European Conference on Genetic Programming, p. 61–72. Springer, 2013.

Harding, S.; Banzhaf, W. **Fast genetic programming on gpus**. In: Proceedings of the 10th European Conference on Genetic Programming, volume 4445 of LNCS, p. 90–101. Springer, 2007.

Harding, S. L.; Banzhaf, W. **Distributed genetic programming on gpus using cuda**. In: Workshop on Parallel Architectures and Bioinspired Algorithms, p. 1–10. Citeseer, 2009.

Horlings, R.; Datcu, D. ; Rothkrantz, L. J. M. **Emotion recognition using brain activity**. In: Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing - CompSysTech '08, p. II.1. ACM Press, 2008.

Jain, A.; Mao, J. ; Mohiuddin, K. Artificial neural networks: a tutorial. **Computer**, v.29, n.3, p. 31–44, Mar 1996.

Jatoba, L. C.; Grossmann, U.; Kunze, C.; Ottenbacher, J. ; Stork, W. **Context-aware mobile health monitoring: Evaluation of different pattern recognition methods for classification of physical activity**. In: 2008 30th annual international conference of the ieee engineering in medicine and biology society, p. 5250–5253. IEEE, 2008.

Yin, J.; Yang, Q. ; Pan, J. Sensor-based abnormal human-activity detection. **IEEE Transactions on Knowledge and Data Engineering**, v.20, n.8, p. 1082–1090, Aug 2008.

Van Kasteren, T.; Noulas, A.; Englebienne, G. ; Kröse, B. **Accurate activity recognition in a home setting**. In: Proceedings of the 10th international conference on Ubiquitous computing, p. 1–9, 2008.

Khan, M. M.; Khan, G. M. ; Miller, J. F. **Evolution of neural networks using cartesian genetic programming**. In: IEEE Congress on Evolutionary Computation, p. 1–8. IEEE, Jul 2010.

Khan, G. M.; Arshad, R. Electricity peak load forecasting using cgp based neuro evolutionary techniques. **International Journal of Computational Intelligence Systems**, v.9, n.2, p. 376–395, 2016.

Khan, N. M.; Khan, G. M. **Audio signal reconstruction using cartesian genetic programming evolved artificial neural network (cgpann)**. In: 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), p. 568–573. IEEE, 2017.

Group, K. O. W.; others. The opencl specification version 1.1. **http://www. khronos. org/registry/cl/specs/opencl-1.1. pdf**, 2011.

Kim, E.; Helal, S. ; Cook, D. Human activity recognition and pattern discovery. **IEEE Pervasive Computing**, v.9, n.1, p. 48–53, Jan 2010.

Koza, J. R. **Genetic programming: on the programming of computers by means of natural selection**. Complex adaptive systems. MIT Press, 1992.

Kwapisz, J. R.; Weiss, G. M. ; Moore, S. A. Activity recognition using cell phone accelerometers. **ACM SIGKDD Explorations Newsletter**, v.12, n.2, p. 74–82, Mar 2011.

Lara, O. D.; Labrador, M. A. A survey on human activity recognition using wearable sensors. **IEEE Communications Surveys & Tutorials**, v.15, n.3, p. 1192–1209, 2013.

Mei, X.; Chu, X. Dissecting gpu memory hierarchy through microbenchmarking. **IEEE Transactions on Parallel and Distributed Systems**, v.28, n.1, p. 72–86, 2016.

Melo Neto, J. M.; Bernardino, H. S. ; Barbosa, H. J. **Hybridization of cartesian genetic programming and differential evolution for generating classifiers based on neural networks**. In: 2018 IEEE Congress on Evolutionary Computation (CEC), p. 1–8. IEEE, Jul 2018.

Miller, J. F.; Thomson, P. **Cartesian Genetic Programming**, volume 1802 de **Lecture Notes in Computer Science**, p. 121–132. Springer Berlin Heidelberg, 2000.

Millan, J. d. R.; Mourino, J.; Franze, M.; Cincotti, F.; Varsta, M.; Heikkonen, J. ; Babiloni, F. A local neural classifier for the recognition of eeg patterns associated to mental tasks. **IEEE Transactions on Neural Networks**, v.13, n.3, p. 678–686, May 2002.

Miller, J. F. **Cartesian genetic programming**. In: Cartesian Genetic Programming, p. 17–34. Springer, 2011.

Miller, J. F. Cartesian genetic programming: its status and future. **Genetic Programming and Evolvable Machines**, Aug 2019.

Mitchell, E.; Monaghan, D. ; O'Connor, N. E. Classification of sporting activities using smartphone accelerometers. **Sensors (Basel, Switzerland)**, v.13, n.4, p. 5317–5337, Apr 2013.

Mohino-Herranz, I.; Gil-Pita, R.; Ferreira, J.; Rosa-Zurera, M. ; Seoane, F. Assessment of mental, emotional and physical stress through analysis of physiological signals using smartphones. **Sensors**, v.15, n.10, p. 25607–25627, Oct 2015.

Mohri, M.; Rostamizadeh, A. ; Talwalkar, A. **Foundations of Machine Learning, second edition**. MIT Press, Dec 2018.

Mohino-Herranz, I.; Gil-Pita, R.; Rosa-Zurera, M. ; Seoane, F. Activity recognition using wearable physiological measurements: Selection of features from a comprehensive literature study. **Sensors (Basel, Switzerland)**, v.19, n.24, Dec 2019.

Murphy, K. P. **Machine learning: a probabilistic perspective**. MIT press, 2012.

NVIDIA. Cuda c best practices guide. **NVIDIA, July**, 2013.

NVIDIA. Cuda c programming guide. **NVIDIA, July**, 2017.

Oniga, S.; Suto, J. **Human activity recognition using neural networks**. In: Proceedings of the 2014 15th International Carpathian Control Conference (ICCC), p. 403–406. IEEE, May 2014.

Paraschiakos, S.; Cachucho, R.; Moed, M.; van Heemst, D.; Mooijaart, S.; Slagboom, E. P.; Knobbe, A. ; Beekman, M. Activity recognition using wearable sensors for tracking the elderly. **User Modeling and User-Adapted Interaction**, v.30, n.3, p. 567–605, Jul 2020.

Poli, R.; Langdon, W. B.; McPhee, N. F. ; Koza, J. R. **A field guide to genetic programming**. Lulu Press], 2008.

Ramachandran, P.; Zoph, B. ; Le, Q. V. Searching for activation functions. **arXiv:1710.05941 [cs]**, Oct 2017. arXiv: 1710.05941.

Robilliard, D.; Marion-Poty, V. ; Fonlupt, C. Genetic programming on graphics processing units. **Genetic Programming and Evolvable Machines**, v.10, n.4, p. 447–471, Dec 2009.

Robilliard, D.; Marion, V. ; Fonlupt, C. **High performance genetic programming on gpu**. In: Proceedings of the 2009 workshop on Bio-inspired algorithms for distributed systems - BADS '09, p. 85. ACM Press, 2009.

Ronao, C. A.; Cho, S.-B. Human activity recognition with smartphone sensors using deep learning neural networks. **Expert Systems with Applications**, v.59, p. 235–244, Oct 2016.

Russo, I. L.; Bernardino, H. S. ; Barbosa, H. J. A massively parallel grammatical evolution technique with opencl. **Journal of Parallel and Distributed Computing**, v.109, p. 333–349, Nov 2017.

Schrader, L.; Vargas Toro, A.; Konietzny, S.; Rüping, S.; Schäpers, B.; Steinböck, M.; Krewer, C.; Müller, F.; Güttler, J. ; Bock, T. Advanced sensing and human activity recognition in early intervention and rehabilitation of elderly people. **Journal of Population Ageing**, v.13, n.2, p. 139–165, Jun 2020.

Sharma, A.; Lee, Y.-D. ; Chung, W.-Y. **High accuracy human activity monitoring using neural network**. In: 2008 Third International Conference on Convergence and Hybrid Information Technology, p. 430–435. IEEE, Nov 2008.

Shmueli, B. **Multi-class metrics made simple, part i: Precision and recall**, Jun 2019.

Steinkraus, D.; Buck, I. ; Simard, P. **Using gpus for machine learning algorithms**. In: Eighth International Conference on Document Analysis and Recognition (ICDAR'05), p. 1115–1120 Vol. 2. IEEE, 2005.

Tompson, J.; Schlachter, K. An introduction to the opencl programming model. **Person Education**, v.49, p. 31, 2012.

Turner, A. **Evolving artificial neural networks using Cartesian genetic programming**. 2015. Tese de Doutorado - University of York.

Vašíček, Z.; Slanỳ, K. **Efficient phenotype evaluation in cartesian genetic programming**. In: European Conference on Genetic Programming, p. 266–278. Springer, 2012.

Yao, X. Evolving artificial neural networks. **Proceedings of the IEEE**, v.87, n.9, p. 1423–1447, Sep 1999.

Yu, Q.; Chen, C. ; Pan, Z. **Parallel genetic algorithms on programmable graphics hardware**. In: International Conference on Natural Computation, p. 1051–1059. Springer, 2005.

Zhang, G. Neural networks for classification: a survey. **IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)**, v.30, n.4, p. 451–462, Nov 2000.

Zhu, C.; Sheng, W. **Human daily activity recognition in robot-assisted living using multi-sensor fusion**. In: 2009 ieee international conference on robotics and automation, p. 2154–2159. IEEE, 2009.