UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA - ITEC
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA - PPGEE

UMA METODOLOGIA BIOLOGICAMENTE INSPIRADA PARA PROJETO AUTOMÁTICO DE REDES NEURAIS ARTIFICIAIS USANDO SISTEMAS-L PARAMÉTRICOS COM MEMÓRIA

LÍDIO MAURO LIMA DE CAMPOS

TD: 16 / 2016

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2016

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA - ITEC
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA - PPGEE

UMA METODOLOGIA BIOLOGICAMENTE INSPIRADA PARA PROJETO AUTOMÁTICO DE REDES NEURAIS ARTIFICIAIS USANDO SISTEMAS-L PARAMÉTRICOS COM MEMÓRIA

LÍDIO MAURO LIMA DE CAMPOS

TD: 16 / 2016

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2016

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA - ITEC
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA - PPGEE

LÍDIO MAURO LIMA DE CAMPOS

UMA METODOLOGIA BIOLOGICAMENTE INSPIRADA PARA PROJETO AUTOMÁTICO DE REDES NEURAIS ARTIFICIAIS USANDO SISTEMAS-L PARAMÉTRICOS COM MEMÓRIA

Tese submetida à Banca Examinadora do Programa de Pós-Graduação em Engenharia Elétrica da UFPA para a obtenção do Grau de Doutor em Engenharia Elétrica na área de Computação Aplicada.

Orientador: Prof. Dr. Roberto Célio Limão de Oliveira - UFPA
Co-Orientador: Prof. Dr. Mauro Roisenberg - UFSC

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2016

Dados Internacionais de Catalogação-na-Publicação (CIP)
Sistema de Bibliotecas da UFPA

Campos, Lídio Mauro Lima de, 1970 –
Uma metodologia biologicamente inspirada para
projeto automático de redes neurais artificiais usando sistemas-l paramétricos
com memória / Lídio Mauro Lima de Campos – 2016.

Orientador: Prof. Dr. Roberto Célio Limão de Oliveira;
Coorientador: Prof. Dr. Mauro Roisenberg.
Tese (Doutorado) – Universidade Federal do Pará, Instituto de
Tecnologia, Pós-Graduação em Engenharia Elétrica, Belém,
2016.

1.Redes neurais (computação). 2. Computação evolucionária.
3-Algoritmos genéticos. I. Título.

CDD 23 . ed. 006. 32

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA - ITEC
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA - PPGEE

UMA METODOLOGIA BIOLOGICAMENTE INSPIRADA PARA PROJETO AUTOMÁTICO DE REDES NEURAIS ARTIFICIAIS USANDO SISTEMAS-L PARAMÉTRICOS COM MEMÓRIA

AUTOR: LÍDIO MAURO LIMA DE CAMPOS

TESE DE DOUTORADO SUBMETIDA À AVALIAÇÃO DA BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA DA UNIVERSIDADE FEDERAL DO PARÁ E JULGADA ADEQUADA PARA OBTENÇÃO DO GRAU DE DOUTOR EM ENGENHARIA ELÉTRICA NA ÁREA DE COMPUTAÇÃO APLICADA.

APROVADA EM ____/_____/_____

BANCA EXAMINADORA:

Prof. Dr. Roberto Célio Limão de Oliveira - Orientador
PPGEE – UFPA

Prof. Dr. Mauro Roisenberg – Co-Orientador
(INE – UFSC)

Prof. Dra. Adriana Rosa Garcez Castro - Examinadora Interna
(PPGEE – UFPA)

Prof. Dr. Ádamo Lima de Santana - Examinador Interno
(PPGEE – UFPA)

Prof. Dr. Orlando Shigueo Ohashi Junior - Examinador Externo
(UFRA)

Prof. Dr. Fábio Meneghetti Ugulino de Araújo- Examinador Externo
(UFRN)

VISTO:

Prof. Dr. Evaldo Gonçalves Pelaes
(COORDENADOR DO PPGEE/ITEC/UFPA)

AGRADECIMENTOS

Ao concluir esta Tese de Doutorado, agradeço primeiramente a Deus.

Aos professores do Programa de Pós-Graduação em Engenharia Elétrica da UFPA, Prof. Dr. Evaldo Gonçalves Pelaes, Prof. Dra. Carolina de Mattos Affonso, Prof. Dr.Ádamo Lima de Santana, Prof. Dr. Carlos Tavares da Costa Junior e Prof.Dr.Roberto Célio Limão de Oliveira por partilharem seus conhecimentos.

Ao meu orientador Prof. Dr. Roberto Célio Limão de Oliveira e Co-Orientador Prof. Dr. Mauro Roisenberg pelo profissionalismo e ajuda constante.

Aos Servidores do Programa de Pós-Graduação em Engenharia Elétrica da UFPA Socorro Palheta e Davi, pela atenção que sempre me dispensaram.

Aos Membros da Banca Examinadora.

Ao Coordenador do Campus de Castanhal Dr. João Batista Santiago Ramos e Vice Coordenador Prof. Dr. Milton Begeres. Ao prof. Dr. Adriano Sales dos Santos Silva.

A todos meus colegas professores da Faculdade de Sistemas de Informação do Campus de Castanhal-Pa, da qual faço parte.

Ao Meu Pai Edilson Teixeira e Mãe Terezinha Campos, aos meus Irmãos Edilson Teixeira e Gustavo Augusto Campos, minhas Cunhadas Viviane e Nylma, meus sobrinhos Bruno, Bernardo, Gustavo e Maran Atha.

E, em especial, a minha Esposa Paula Josiane e filha Maria Lycia Sá Campos, pela paciência e incentivo que me transmitem.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACA	Average Classification Accuracy
ADN	Ácido Desoxirribonucleico
AE(s)	Algoritmo(s) Evolucionários(s)
AG(s)	Algoritmo(s) Genético(s)
ANE(s)	Algoritmo(s) Neuroevolutivo(s)
AQ	Algoritmo *QuickPropagation*
AR	Algoritmo de Retropropagação
ARN	Ácido Ribonucleico
CP	Comprimento da pétala
CPPN	Compositional Pattern Producing Network
CP(s)	Célula(s) de Purkimje
CS	Comprimento Sepal
DARPA	Defense Advanced Research Projects Agency
ECD	Esquema de Codificação Direto
DLP	Depressão de Longo-Prazo
DR	Detection Rate
EC	Esforço Computacional
EG	Evolução Gramatical
FP	False Positives
FP(s)	Fibra(s) Paralela(s)
FTP	File Transfer Protocol
GPGPU	General Purpose Graphics Processing Unit
GRG	Graph Rewriting Grammar
GRM	Gramática Geradora de Grafos
ECI	Esquema de Codificação Indireto
HTTP	Hypertext Transfer Protocol
IPOP	Indivíduos da População
KDDCUP'99	KDD Cup 1999 Data
LP	Largura da Pétala
LS	Largura das Sépalas

MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MDA	Modelo de Desenvolvimento Artificial
MSE	Mean Square Error
NEAT	NeuroEvolution of Augmenting Topologies
Npar	Número de Parâmetros.
PARNAs	Projeto Automático de Redes Neurais Artificiais
PLP	Potenciação de Longo-Prazo
PRE	Precision
PRM	Método de Poda
PST	Predição de Séries Temporais
RAE	Mean Relative Error
RNA(s)	Rede(s) Neural(is) Artificial(is)
RNR(s)	Rede(s) Neural(is) Recorrente(s)
RPMC	Rede Perceptron Multicamada
Sistema-L	Sistema de Lindenmayer
SMAPE	Symetric Mean Absolute Percentage
SPA	Seleção Proporcional à Aptidão
TCP/IP	Pilha de Protocolos TCP/IP
Telnet	Protocolo de Rede Utilizado na Internet
TP	True Positive
UCM	Unobserved Component model
UCP	Unidade Central de Processamento
UDP	User Datagram Protocol

LISTA DE SÍMBOLOS

F_i	Aptidão do indivíduo i
p_i	Probabilidade de Seleção do indivíduo i
X_i	Representa as p entradas das RNAs
W_i	Representa os p pesos associados às entradas das RNAs
η	Taxa de aprendizado
α	taxa de *momentum*
$\prod$	Conjunto finito de regras de reescrita (regras de produção)
$\sum$	Conjunto finito de símbolos ou alfabeto da linguagem
$\sum^*$	Conjunto de todas as palavras possíveis sobre $\sum$
$\sum^+$	Conjunto de todas as palavras sobre $\sum$, excetuando-se a palavra vazia
xj *hj* *oj*	Níveis de ativação das unidades das camada de entrada, oculta e saída da RNA
f(a)	Função de ativação
N	Comprimento dos cromossomos
V	Matriz de pesos da camada de entrada para a intermediária para Redes Recorrentes
W	Matriz de pesos da camada oculta para a de saída para Redes Neurais Recorrentes

RESUMO

Essa tese propõe um algoritmo neuro-evolutivo (ANE) que utiliza um esquema de codificação indireto compacto para representar seus genótipos (um conjunto de dez regras de produção de um sistema de lindenmayer com memória), além disso, possui a habilidade de reuso dos genótipos e automaticamente construir redes neurais modulares, hierárquicas e recorrentes. Um algoritmo genético evolui um sistema de *lindenmayer* (sistema-l) que é usado para projetar a arquitetura de redes neurais. Essa codificação neural proporciona redução de escalabilidade e do espaço de busca em relação a outros métodos, possibilitando uma busca mais eficiente no espaço infinito de arquiteturas de redes neurais. Em adição, o sistema usa um mecanismo de checagem paralelo do genoma que aumenta o paralelismo implícito e a convergência do AG. A função fitness do ANE recompensa redes neurais que são facilmente implementadas. Essa é a primeira tentativa de gerar redes recorrentes a partir dessa combinação de metáforas. O ANE foi testado utilizando cinco bancos de dados do mundo real para classificação e três bens conhecidos para predição de séries temporais (PST). Os resultados são estatisticamente comparados com algoritmos proeminentes citados no estado da arte e com vários métodos de predição (*ADANN, ARIMA, UCM e Forecast Pro*®). Na maioria dos casos, o ANE superou os outros métodos produzindo classificação e predição de séries temporais mais precisas com um menor esforço computacional. Esses resultados são atribuídos a melhoria da eficácia e eficiência no processo de tomada de decisão. O resultado é uma arquitetura de rede neural otimizada para resolver problemas de classificação e simular problemas dinâmicos.

PALAVRAS-CHAVES: Algoritmos neuroevolutivos. Projeto automático de redes neurais artificiais. Computação bioinspirada.

ABSTRACT

This thesis proposes a hybrid neuro-evolutive algorithm (NEA) that uses a compact indirect encoding scheme (IES) for representing its genotypes (a set of ten production rules of a Lindenmayer System with memory), moreover has the ability to reuse the genotypes and automatically build modular, hierarchical and recurrent neural networks. A genetic algorithm (GA) evolves a Lindenmayer System (L-System) that is used to design the neural network's architecture. This basic neural codification confers scalability and search space reduction in relation to other methods. Furthermore, the system uses a parallel genome scan engine that increases both the implicit parallelism and convergence of the GA. The fitness function of the NEA rewards economical artificial neural networks (ANNs) that are easily implemented. The NEA was tested on five real-world classification datasets and three well-known datasets for time series forecasting (TSF). The results are statistically compared against established state-of-the-art algorithms and various forecasting methods (ADANN, ARIMA, UCM, and Forecast Pro®). In most cases, our NEA outperformed the other methods, delivering the most accurate classification and time series forecasting with the least computational effort. These superior results are attributed to the improved effectiveness and efficiency of NEA in the decision-making process. The result is an optimized neural network architecture for solving classification problems and simulating dynamical systems.

KEYWORDS: Neuro-evolutive algorithm. Automatic design of artificial neural networks. Bio-inspired computing.

CAPÍTULO 1 – INTRODUÇÃO

"A igualdade foi criada por que os homens não são idênticos".

François Jacob

1.1 MOTIVAÇÃO E DESCRIÇÃO GERAL DO PROBLEMA

Atualmente, as aplicações de Redes Neurais Artificiais (RNAs) incluem classificação (RIVERO et al., 2010), sistemas de controle (LI et al., 2014), predição (DONATE; SANCHEZ; DE MIGUEL, 2012) e muitas outras (CZAJKOWSKI; PATAN; SZYMASKI, 2014). A atratividade das RNAs advém das características notáveis de processamento de informações do sistema biológico, como não-linearidade, alto paralelismo, robustez, tolerância a falhas, a aprendizagem, a capacidade de lidar com informações imprecisas e sua capacidade de generalizar.

O desenvolvimento de uma arquitetura otimizada e o seu treinamento são os problemas mais importantes na utilização das redes neurais. Uma vez que a arquitetura neural depende da classe de problema a ser resolvida, o processo de projetar tal arquitetura envolve um método heurístico de tentativa e erro para selecionar que tipo de função de transferência e algoritmo de treinamento devem ser usados para ajustar os pesos sinápticos. Todos esses requisitos afetam a capacidade de aprendizado e generalização da RNA e como consequência, esse procedimento necessita ser executado por especialistas, que devem experimentar diferentes topologias e treinar cada uma, com o objetivo de determinar qual delas é a melhor para simular um determinado problema. Para mitigar essas deficiências, alguns métodos têm sido propostos para automatizar esses processos.

Algumas aplicações, descritas no capítulo 2, demonstram que o projeto de RNAs é melhorado considerando a sua junção com Algoritmos Evolucionários (AEs), desde que os paradigmas neurais e evolucionários podem se combinar de forma sinergética. A obtenção de uma arquitetura otimizada de RNA não é tarefa simples, o tempo e o esforço requeridos para o projeto são dependentes da natureza e complexidade da tarefa que se pretende simular. Esta dependência conduz a uma quantidade de tempo e esforço sendo gasto para encontrar a arquitetura de RNA ideal para simular determinada classe de problema.

Além disso, o processo evolucionário é uma forma mais integrada e racional de projeto de RNAs, pois permite que aspectos simples de projeto sejam levados em conta de forma integrada e não requer nenhum conhecimento especializado do problema. Os AEs são especialmente úteis para problemas complexos de otimização onde o número de parâmetros a serem otimizados são grandes e soluções analíticas são difíceis de serem obtidas. Ou seja, AEs podem ajudar a encontrar a solução ótima global sobre um domínio. AEs tornam-se nesse sentido, úteis para representar uma solução adequada para resolver o problema de projeto automático de RNA.

Algoritmos neuro-evolutivos (ANEs) projetam e/ou treinam RNAs através de AEs. Algoritmos Bioinspirados estão ganhando popularidade como solucionadores eficientes de problemas de otimização não-lineares (KROMER; PLATOS; SNÁSEL, 2014). Ademais, os métodos com maior inspiração biológica possibilitam que boa parte da compreensão cientifica sobre sistemas naturais (suas contrapartes reais), ainda pouco conhecidas, sejam melhor entendidas por análogos artificiais, como bem especula (Dalkins, 2004, pag. 47). Nessa tese, verifica-se se um novo ANE bioinspirado pode se constituir como uma ferramenta eficiente para projeto automático de RNAs.

Os ANEs se inserem nos métodos evolutivos aplicados a tarefas de aprendizado por reforço ou reconhecimento de padrões. A Figura 1 ilustra os conceitos básicos de um ANE. Em uma determinada geração, cada RNA da população é avaliada na simulação de uma tarefa. Após isso, as que tem melhor desempenho são selecionadas, dessas são criadas novas populações por cruzamento e mutação e o processo se repete.

O cenário atual dos ANEs aponta alguns desafios: obter representações indiretas que codifiquem RNAs e que diminuam o espaço de busca dos AEs, a obtenção de RNAs robustas com maior organização e regularidade e RNAs recorrentes. Além disso, a diminuição do esforço computacional dos ANEs.

Figura 1 – Os passos básicos da Neuroevolução

Fonte: Adaptado de Whiteson (2012).

Como discutido por Stanley e Miikkulainen (2003), o aumento da complexidade da computação evolucionária demanda métodos de codificação mais sofisticados do que os de mapeamento direto entre genótipo e fenótipo (DONATE; SANCHEZ; DE MIGUEL, 2012; MILLER; TODD; HEDGE, 1989). Um esquema de codificação indireto (AHMADIZAR et al., 2015; HORNBY; POLLACK, 2002; LEE; SEO; SIM, 2008; SOLTANIAN et al., 2013; STANLEY; DAVID; JASON, 2009) permite uma representação mais compacta e escalável do que um esquema de codificação direto (ECD).

Construir um esquema de codificação eficiente capaz de representar estruturas repetitivas e recorrentes é uma tarefa desafiadora para ANEs baseados em ECIs. Os estudos de Miller et al. (1989), Dasgupta e McGregor (1992), Niska et al. (2004); Lee, Seo e Sim (2008), Tsoulos, Gavrilis e Glavas (2008), Donate, Sanchez e de Miguel (2012), Soltanian et al. (2013), Sanchez e Melin (2014), Ahmadizar et al. (2015) foram limitados a Redes Neurais diretas multicamadas (RNDM) que utilizam o algoritmo de retropropagação (AR) para ajustar os pesos das RNAs. ANEs que evoluem RNRs são raros na literatura (BEER; GALLAGHER, 1992; HORNBY; POLLACK, 2002) e, portanto, constituem-se uma lacuna a ser explorada no ramo dos ANEs. Nas secções seguintes são apresentados uma síntese metodologia empregada, os objetivos e contribuições deste trabalho, as propostas atuais e limitações e os trabalhos publicados durante o doutorado.

1.2 SUMÁRIO DA METODOLOGIA

Nessa pesquisa, propõe-se um sistema artificial hibrido biologicamente inspirado denominado *Artificial Development and Evolution of ANNs (ADEANN)*. O *ADEANN* integra dois componentes: O primeiro é uma representação generativa para os genótipos (um conjunto de dez regras de produção de um sistema-L) por meio de um ECI compacto. O ECI também conduz e controla o processo de mapeamento dos genótipos para os fenótipos (morfologias neurais complexas). Para imitar o esquema de codificação do ADN e diminuir a escalabilidade, o ECI, utilizado pelo *ADEANN*, possibilita a representação de fenótipos complexos por meio de um genótipo compacto. Assim, o processo de busca é realizado num espaço de busca de menor dimensão. Além disso, o ECI implementa os princípios organizacionais de hierarquia, modularidade e reutilização de genes. O Segundo componente é um algoritmo genético (AG), uma representação simplificada da evolução natural. Em problemas de busca local baseados em

AG, uma *string* de bits é denominada cromossomo (o genótipo). Cada *bit* no cromossomo é um gene e um conjunto de genes representam os parâmetros de uma função a ser otimizada. A cada *string* é atribuído um valor de aptidão que indica a qualidade da sua solução codificada (o fenótipo). Com objetivo de melhorar o realismo biológico do AG, utilizado pelo *ADEANN*, o mesmo evolui a representação generativa. O processo evolutivo pode ser considerado como alterações genéticas temporais em ADNs hipotéticos de uma população de indivíduos, regulados por um mecanismo de seleção artificial. A metodologia é apresentada em detalhes no capítulo 6.

1.3 PROPOSTAS ATUAIS E LIMITAÇÕES

No capítulo 2, discutem-se as propostas atuais, nessa secção abordam-se de forma sintética as principais abordagens pesquisadas e os prós e contras de cada método e como essas limitações podem ser superadas.

Os métodos que utilizam codificações diretas especificam cada nó e conexão que irão aparecer no fenótipo (arquitetura de rede neural) no genótipo (cromossomo) (DONATE; SANCHEZ; DE MIGUEL, 2012; HARP; SAMAD; GUHA, 1990; MILLER; TODD; HEDGE, 1989; MONTANA; DAVIS, 1989; SANCHEZ; MELIN, 2014; STANLEY; MIIKKULAINEN, 2002; WHITLEY, 1989; WHITLEY; SCHAFFER, 1992), por isso apresentam problemas de escalabilidade. Ou seja, a medida que os tamanhos das RNAs geradas aumentam de tamanho, os genótipos que as representam crescem bastante, o que não é desejável, pois o espaço de busca também aumenta. Nos métodos de Miller, Todd e Hedge (1989) e Dasgupta e McGregor (1992) o tamanho da matriz de conectividade que representa uma determinada RNA cresce de forma quadrática (N^2) com o número de nós da rede. No método de Stanley e Miikkulainen (2002) a estrutura de dados que representa o genótipo, cresce linearmente (N) com o número de conexões entre dois nós. Assim, estas representações podem crescer indefinidamente a medida que número de nós aumenta. Outra questão importante é que todos os métodos citados anteriormente restringiram seus projetos a RNAs diretamente alimentadas e foram testadas apenas na simulação de problemas estáticos.

Um levantamento realizado no estado da arte, a ser apresentado no capítulo 2 mostra que algumas abordagens, que utilizam codificações indiretas, possibilitam gerar arquiteturas de RNAs mais complexas, por meio de representações genotípicas mais compactas do que as codificações diretas (AHMADIZAR et al., 2015; CLUNE; CHEN; LIPSON, 2013; LEE et al.,

2013; RISI; STANLEY, 2011, 2012; SOLTANIAN et al., 2013; STANLEY; DAVID; JASON, 2009; TOWNSEND; KEEDWELL; GALTON, 2013; TSOULOS; GAVRILIS; GLAVAS, 2008). No capítulo 2, detalham-se essas abordagens e faz-se uma análise crítica detalhada das mesmas considerando-se esses aspectos. A seguir, apresentam-se vários métodos baseados em ECDs.

Os métodos de codificações indiretos tendem a melhorar o problema da escalabilidade, com exceção do método de KITANO (1990) em que o aumento de escala persiste, esse método é descrito na secção 2.3.1.1. O método *GADON* proposto por Boozarjomehry e Svrcek (2001) centra-se nas questões de escalabilidade. Este método é bastante robusto tendo sido testado em simulação de processos químicos industriais. A metodologia possibilita gerar, apenas, topologias diretas e reduz o problema da escalabilidade, pois utiliza um cromossomo de 65 bits para codificação de um Sistema-L, maiores detalhes sobre o método são apresentados na secção 2.3.3.1.

O *HyperNEAT*, discutido na secção 2.3.4.1, é uma abordagem neuroevolutiva estendida do *NeuroEvolution of Augmenting Topologies (NEAT)*, o método considera que uma boa representação para uma RNA deve ser capaz de representar o seu padrão de conectividade de forma compacta, em razão disso usa um método de codificação denominado *Compositional Pattern Producing Network (CPPN*), que é uma metáfora de desenvolvimento biológico, desempenhando o papel do ácido desoxirribonucleico (ADN) na natureza, em um alto nível de abstração. O *CPPN*, secção 2.3.4.1.1, é uma rede de funções matemáticas de geram os pesos das RNAs baseados na geometria do domínio do problema e que melhora a aprendizagem em tarefas de aprendizagem por reforço (STANLEY; DAVID; JASON, 2009). Esta disposição geométrica é chamada substrato. Em suma, o *HyperNEAT* evolui um padrão de conectividade de uma RNA de acordo com geometria particular do substrato. O *HyperNEAT* não é necessariamente a melhor escolha para todos os tipos de problemas, a criação e configuração do substrato, que possibilita explorar a geometria da tarefa, não é simples e para cada tipo de problema simulado, um substrato diferente deve ser escolhido. O que pode não valer a pena se o esforço de criação do mesmo, não proporciona uma vantagem significativa. Risi e Stanley (2011) discutem que o uso do *HyperNEAT* é vantajoso quando o problema a ser simulado possui: um grande número de entradas e saídas, resolução variável, relações geométricas entre entradas e saídas.

Conforme mencionam Risi e Stanley (2011, 2012) uma limitação significativa do *HyperNEAT* é que o usuário tem de definir literalmente a localização dos nós da camada oculta

dentro de um substrato. O *ES-HyperNEAT* (RISI; STANLEY, 2011, 2012), descrito em detalhes na secção 2.3.4.2, é uma extensão do *HyperNEAT* que superou essa limitação, o mesmo pode determinar a densidade adequada e posição de neurônios por conta própria e ainda preservar os avanços introduzidos pelo *HyperNEAT* original, essa é uma vantagem do método. O *ES-HyperNEAT* tem se mostrado promissor, permitindo a evolução de RNAs plásticas regulares, o que é um objetivo importante da neuroevolução. Entretanto, o problema de escolha e configuração do substrato, para uma tarefa especifica, persiste igualmente ao *HyperNEAT*.

Os métodos descritos a seguir pertencem a classe dos que utilizam evolução gramatical (EG). Lee, Seo e Sim (2008) buscaram maior inspiração biológica no aspecto de codificação indireta das regras de produção. Essas abordagens são inspiradas na codificação cromossômica visando gerar metáforas do desenvolvimento biológico. Na pesquisa de Lee, Seo e Sim (2008), similar ao ADN biológico, a informação é codificada usando os símbolos A,G,T e C. Uma sequência desses três símbolos é denominado códon e cada gene é codificado pela sequência de códons, que são traduzidos em regras de produção de um modelo de desenvolvimento artificial. Esse método é restrito a redes neurais diretas com uma única camada oculta e não pode gerar redes neurais recorrentes (RNRs).

Hornby e Pollack (2002) definiram uma classe de representação denominada representação generativa, que caracteristicamente reusa elementos do genótipo no mapeamento para o fenótipo. Estes autores examinaram a evolução biológica do cérebro utilizando Sistemas-L como modelo artificial de desenvolvimento. Eles exploraram simultaneamente a evolução morfológica e controladores neurais de robôs simulados.

Tsoulos, Gavrilis e Glavas (2008) usaram EG para treinar e projetar RNAs diretas com uma camada escondida. Soltanian et al. (2013) projetaram topologias neurais usando EG e otimizaram os pesos com o algoritmo de retropropagação, as avaliações de rede em relação aos dados de treinamento também foram realizadas pelo AR. Entretanto, os seus projetos foram restritos a redes neurais diretas.

Ahmadizar et al. (2015) desenvolveram um algoritmo para evoluir simultaneamente a topologia e os pesos das conexões de RNAs por meio da combinação da EG e um AG. O método proposto é testado em alguns *benchmarkings* de classificação do mundo real e os resultados são comparados estatisticamente em relação a outros existentes na literatura. Entretanto, a aplicabilidade do *GEGA* a outras aplicações tais como predição de séries

temporais (PST) não foi testada e o modelo não pode gerar RNRs. Ou seja, o projeto dos autores e restrito a RNAs diretas multicamadas.

1.4 OBJETIVOS DA TESE

1.4.1 Objetivo geral

Definir uma nova abordagem para um ANE, utilizando evolução gramatical (EG) e algoritmo genético (AG), com objetivo de lidar com o projeto otimizado de sistemas classificadores, baseados em redes neurais, visando simular problemas de reconhecimento de padrões e sistemas dinâmicos com boa capacidade preditiva e de generalização.

1.4.2 Objetivos específicos

a) apresentar e analisar os principais aspectos da natureza sobre os quais se buscou inspiração para o desenvolvimento do ANE;
b) projetar e implementar em ambiente computacional um modelo de desenvolvimento artificial indiretamente codificado (IES), baseado em Sistema-L com memória, que incorpore aspectos de inspiração biológica baseado na codificação cromossômica do ADN e que diminua o espaço de busca do AG e reduza o problema de escalabilidade de outros métodos que utilizam ECD e ECI;
c) implementar em ambiente computacional um Algoritmo Genético (AG) inspirado em mecanismos da evolução biológica, que possibilite evoluir regras de produção codificadas de um Sistema-L de memória;
d) integrar o modelo artificial de desenvolvimento, baseado em Sistema-L e o AG, visando obter um ANE que viabilize gerar automaticamente redes neurais (diretas e recorrentes) e selecionar arquiteturas econômicas capazes de simular de tarefas de reconhecimento de padrões e sistemas dinâmicos;
e) testar o ANE na simulação de problemas de classificação e sistemas dinâmicos e comparar o desempenho do mesmo, por meio de testes estatísticos apropriados, com outros proeminentes ANEs e métodos existentes na literatura;
f) demonstrar que diferentes arquiteturas de RNAs podem ser obtidas por meio do ANE proposto.

1.5 CONTRIBUIÇÃO DA TESE

a) A abordagem evolucionária proposta nessa tese, consiste em si, de uma forma sinérgica e complementar para projetar RNAs, sem requerer conhecimento especializado do problema. Nesse sentido, o *ADEANN* é útil por representar uma solução adequada para resolver o problema do projeto automático de RNAs. A proposta de ANE apresentada no capítulo 6, utiliza uma combinação de EG e AG para evoluir topologias de RNAs, com um esforço mínimo do especialista para customização do sistema. Possibilitando uma busca mais eficiente no espaço infinito de arquiteturas de redes neurais;
b) O Modelo de Desenvolvimento Artificial (MDA) baseado em Sistema-L, apresentado no capítulo 6, na secção 6.3.1, possibilita gerar arquiteturas RNAs diretas e recorrentes, sem a necessidade de configurações adicionais de substratos, para cada tipo especifico de problema, como ocorre com os métodos *HyperNEAT* (STANLEY; DAVID; JASON, 2009), e *ES-HyperNEAT* (RISI; STANLEY, 2011, 2012);
c) Normalmente em problemas de busca local onde os AGs tradicionais são empregados, uma cadeia de *bits* é denominada de cromossomo, cada *bit* é um gene e conjuntos de genes representam os parâmetros de uma função que será otimizada. Entretanto, nessa pesquisa com o objetivo de aproximar o AG dos processos biológicos, utilizou-se a inspiração biológica como técnica computacional de projeto. Inicialmente, considerou-se que o desenvolvimento dos neurônios é conduzido pela informação genética que está codificada, de forma compacta, no ADN e que, quando seguida, resultará na forma final do esquema de interconexão dos neurônios. Essa compactação, minimiza a informação requerida para descrever objetos complexos. Um outro ponto é que a evolução descreve mudanças temporais no código genético ADN. Esses dois processos naturais são híbridos tal que o ADN contido nas células, também pode gerar células. Por outro lado, as mudanças no ADN são passadas as gerações futuras. Buscando inspiração biológica nesses dois processos naturais, propõe-se um sistema artificial hibrido, descrito suscintamente na secção 1.2 e apresentado em detalhes no capítulo 6, que abstrai esses mecanismos naturais em um nível aceitável de complexidade. A

inspiração biológica subjaz a originalidade da nossa abordagem. Essa é a primeira tentativa de gerar RNRs por meio dessas metáforas combinadas;

d) Hornby e Pollack (2002) e Lee, Seo e Sim (2008) são os únicos trabalhos que utilizam Sistema-L como metáforas do desenvolvimento biológico. Entretanto, o método de Lee, Seo e Sim (2008) é restrito a arquiteturas redes neurais diretas com uma única camada escondida. Apenas o trabalho de Hornby e Pollack (2002) possibilita gerar redes recorrente. Entretanto, no ECI usado por Hornby e Pollack (2002), os genótipos codificam um conjunto de 20 regras de produção de um Sistema-L. No ECI utilizado pelo *ADEANN*, inspirado na codificação do ADN biológico, os genótipos codificam um Sistema-L paramétrico com memória que é composto de apenas 10 regras de produção. Dessa forma, o ECI utilizado pelo *ADEANN* é mais compacto do que o de Hornby e Pollack (2002), o que reduz o espaço de busca de todas as soluções factíveis. Além disso, o mecanismo de memória utilizado no nosso enfoque, possibilita o reuso de estruturas fenotípicas (reescrita de nós e conexões) em diferentes estágios do desenvolvimento. Esse reuso é uma importante capacidade dos ANEs. ANEs que evoluem RNRs são raros na literatura (BEER; GALLAGHER, 1992; HORNBY; POLLACK, 2002);

e) Outra contribuição do *ADEANN*, diz respeito a escalabilidade. O método de codificação indireto utilizado pelo mesmo permite armazenar regras de produção compactas, que possibilitam gerar arquiteturas complexas de RNAs diretas ou recorrentes. Essa possibilidade de conduzir a uma codificação compacta, oferece ganhos significativos em termos de redução de escalabilidade e do espaço de busca, em relação a outros métodos que utilizam codificação direta ou indireta, dentre eles citam-se os métodos de Cantu-Paz e Kammath (2005), Stanley e Miikkulainen (2002), Whitley (1989), Miller, Todd e Hedge (1989), Harp, Samad e Guha (1990), Montana e Davis (1989), Whitley e Schaffer (1992), Kitano (1990), Hornby e Pollack (2002) e Ahmadizar et al. (2015). Na secção 8.1, faz-se uma discussão mais detalhada sobre esse aspecto;

f) Adicionalmente, o *ADEANN* implementa um mecanismo de verificação paralela do genoma que aumenta o nível de paralelismo implícito do AG tradicional. Nessa tese, investiga-se se esse mecanismo torna a convergência do AG mais rápida do que os outros métodos que utilizam mecanismos de verificação fixos, e se como consequência irá reduzir o esforço computacional durante o processo de busca. Os

resultados de simulação, apresentados no capítulo 7 mostram que o *ADEANN* apresenta convergência mais rápida que os demais métodos;

g) A função de aptidão, equação 29, utilizada pelo *ADEANN* se constitui um importante instrumento para direcionar a busca do sistema para arquiteturas econômicas de RNAs que podem resolver problemas de classificação e PST ao mesmo tempo.

1.6 PUBLICAÇÕES REALIZADAS DURANTE O DOUTORADO

1.6.1 Artigos publicados em periódico

DE CAMPOS, L. M. L.; DE OLIVEIRA, R. C. L.; ROISENBERG, M. Optimization of neural networks through grammatical evolution and a genetic algorithm. **Expert Systems with Applications**, Oxford, v. 56, p. 368-384, 2016.

1.6.2 Capítulo de Livro

DE CAMPOS, L. M. L.; DE OLIVEIRA, R. C. L.; ROISENBERG, M. Network intrusion detection system using datamining. In: 13th ENGINEERING APPLICATIONS OF NEURAL NETWORK CONFERENCE, 13., 2012, London, UK. **Proceedings…** London: EANN, 2012. v. 1, p. 104-113.

1.6.3 Artigos publicados em conferências

a) DE CAMPOS, L. M. L.; DE OLIVEIRA, R. C. L.; ROISENBERG, M. A hybrid neuro-evolutive algorithm for neural network optimization. In: IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE (IEEE WCCI), INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2016, Vancouver, Canada. **Proceedings…** Piscataway: IEEE Press, 2016. v. 1, p. 1-8;
b) DE CAMPOS, L. M. L.; DE OLIVEIRA, R. C. L.; ROISENBERG, M. Evolving artificial neural networks through l-system and evolutionary computation. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), 2015, Killarney, Ireland. **Proceedings…** Piscataway: IEEE Press, 2015.

c) DE CAMPOS, L. M. L.; DE OLIVEIRA, R. C. L. Artificial development and evolution of arttificial neural networks using parametric l-systems with memory. In: INTERNATIONAL FLINS CONFERENCE ON DECISION MAKING AND SOFT COMPUTING (FLINS2014), 11., Aug. 2014, João Pessoa, Brazil. **Proceedings…**, 2014;

d) DE CAMPOS, L. M. L.; DE OLIVEIRA, R. C. L. A Comparative Analysis of Methodologies for Automatic Design of Artificial Neural Networks - From the Beginnings until Today. In: BRICS COUNTRIES CONGRESS (BRICS-CCI), 1.; BRAZILIAN CONGRESS (CBIC) ON COMPUTATIONAL INTELLIGENCE, 11., 2013. **Proceedings…**, 2013;

e) DE CAMPOS, L. M. L.; DE OLIVEIRA, R. C. L.; ROISENBERG, M. Automatic design of neural networks with L-Systems and genetic algorithms: a biologically inspired methodology. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), 2011, San Jose, EUA. **Proceedings…** Piscataway: IEEE Press, 2011.

1.7 ORGANIZAÇÃO DA TESE

O capítulo 2 apresenta uma revisão bibliográfica histórica e elabora-se o estado da arte sobre o tema de pesquisa projeto automático de RNAs.

No capítulo 3 apresentam-se modelos de desenvolvimento artificiais baseados em Sistemas de Lindemayer e aplicações desses para modelagem de sistemas de desenvolvimento biologicos.

O capítulo 4 inicialmente apresenta a estrutura típica de neurônio biológico e seu funcionamento. Adicionalmente, apresentam-se diversas funções de ativação e as arquiteturas de RNAs diretas e Recorrentes.

O capítulo 5 apresenta conceitos de evolução biológica, seleção natutal e alguns conceitos de biologia molecular. Após isso, discutem-se alguns conceitos de computação evolucionária focando nos AGs, que foram utilizados nessa pesquisa.

O capítulo 6 apresenta um novo algoritmo neuroevolutivo, que incorpora aspectos de inspiração biológica e possibilita gerar arquiteturas variadas de RNAs diretas e recorrentes.

O capítulo 7 apresenta os resultados de simulação para problemas de classificação e sistemas dinâmicos usando redes diretas e recorrentes.

No capítulo 8 conclui-se a tese e indicam-se trabalhos futuros a serem desenvolvidos.

CAPÍTULO 2 – PROJETO AUTOMÁTICO DE REDES NEURAIS – ESTADO DA ARTE

"Se os fatos não se encaixam na teoria, modifique os fatos."

Albert Einstein

2.1 INTRODUÇÃO

Uma metodologia que tem se mostrado promissora para autonomaticamente gerar RNAs é a Neuroevolução, ou seja, RNAs são evoluídas através de Algoritmos Evolucionários. No presente capítulo faz uma revisão bibliográfica histórica e elabora-se o estado da arte sobre esse tema de pesquisa, desde as primeiras metodologias surgidas na década de 80 até os dias atuais. Inicialmente, apresentam-se metodologias que utilizam codificação direta, em seguida, discute-se que o interesse atual é a obtenção de RNAs geradas por codificações indiretas, em que a descrição da solução é comprimida de forma que essa informação pode ser reutilizada, essa compressão permite gerar topologias mais complexas de RNAs, evitam problemas de escalabilidade e reduzem o espaço de busca do AG. Além disso, apresenta-se uma metodologia de otimização de morfologias neuronais pertencente ao ramo da neurociência computacional.

2.2 MÉTODOS DE CODIFICAÇÃO DIRETA

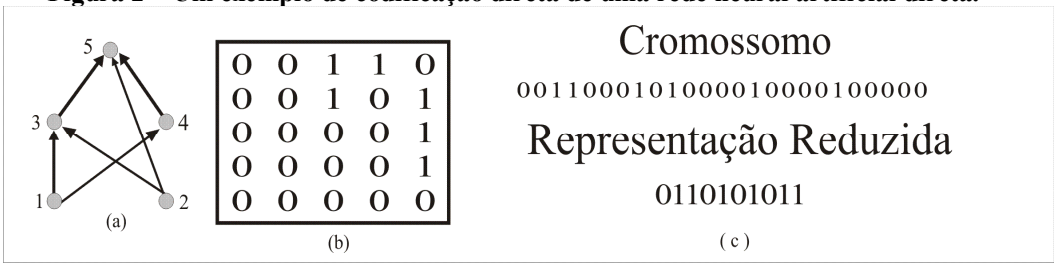
2.2.1 Primeiros métodos de codificação direta– década de 80

Os primeiros trabalhos sobre Projeto Automático de Redes Neurais Artificiais (PARNAs) utilizavam codificação direta, dentre eles citam-se: Whitley (1989), Miller, Todd e Hedge (1989), Harp; Samad e Guha (1990). Essas metodologias evoluíam arquiteturas de RNAs diretas. Montana e Davis (1989) propuseram o primeiro enfoque para evoluir pesos de RNAs. Whitley e Schaffer (1992) descreveram uma coleção de vários artigos que utilizam a combinação de AGs e RNAs para evolução de pesos ou topologias de RNAs, o período coberto foi até o ano de 1992. O esquema de codificação direto (ECD) também é ilustrado na pesquisa realizada por Miller, Todd e Hedge (1989), que restringiram seus projetos a redes diretamente alimentadas com um número fixo de unidades para o qual o AG evolui as conexões da topologia.

Na pesquisa de Miller, Todd e Hedge (1989), uma Matriz C=(cij) NxN representa uma rede neural com N nós, onde cij=1 indica a presença ou cij=0 ausência de conexão do nó i para o nó j. A Figura 2 mostra um exemplo de esquema de codificação direta de uma RNA. A matriz de conexão é ilustrada na Figura 2(b). A conversão da matriz de conectividade para o cromossomo é mostrada na Figura 2(c). Miller, Todd e Hedge (1989) usaram um operador de *crossover* que randomicamente seleciona o índice de uma linha e troca as linhas correspondentes entre dois genitores para criar dois descendentes. O valor de aptidão utilizado foi a soma dos quadrados dos erros durante o treinamento fixados na última época. Os autores testaram o Algoritmo Genético em três tarefas *XOR*, *Four Quadrant* e um Codificador/Decodificador simples.

Uma vantagem do método é a simplicidade de implementação. Gruau, Whitley e Pyeatt (1996) considera que uma das grandes desvantagens métodos diretos é que o comprimento do cromossomo, que representa a topologia da rede neural, aumenta exponencialmente à medida que o número de neurônios cresce.

Figura 2 – Um exemplo de codificação direta de uma rede neural artificial direta.



(a), (b) e (c), mostram a sua arquitetura, sua matriz de conectividade e a representação de sua *string* binária, respectivamente.
Fonte: adaptado de Miller, Todd e Hedge (1989).

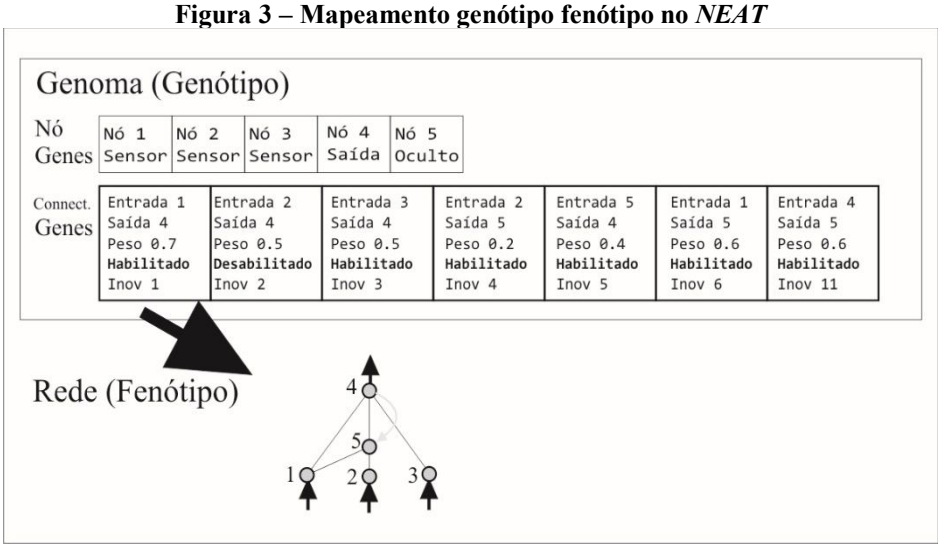
2.2.2 NeuroEvolution of Augmenting Topologies (NEAT)

Em 2002, Stanley e Miikkulainen (2002) desenvolveram uma nova metodologia, codificada diretamente, denominada NEAT, a seguir descrevem-se os princípios do NEAT.

2.2.2.1 Codificação Cromossômica do NEAT

O esquema de codificação genético do *NEAT* é mantido por meio de uma lista onde cada gene especifica o nó de entrada o de saída, o peso da conexão, um gene de conexão que é expresso por um bit habilitado ou não e um número de inovação, conforme ilustrado na Figura 3. As mutações no *NEAT* podem alterar ambos os pesos e conexões da estrutura da rede. Na mutação adição de conexão (Figura 4), uma única conexão de um gene novo com um peso aleatório é adicionada ligando dois nós previamente desconectados. Na mutação de adição de nó, uma conexão existente é dividida e o novo nó colocado onde a conexão antiga costumava ser colocada. A conexão antiga é desabilitada e duas novas conexões são adicionadas ao genoma.

Figura 3 – Mapeamento genótipo fenótipo no *NEAT*



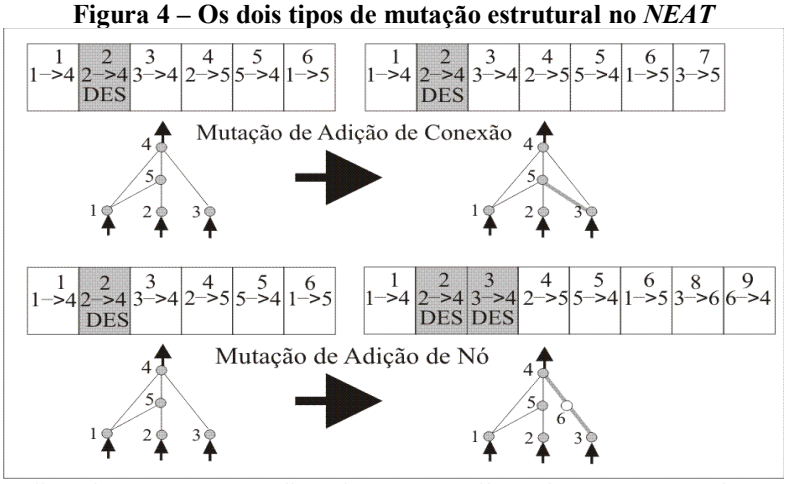
Um GENÓTIPO é descrito para produzir o fenótipo. Há três nós de entrada, um escondido e um nó de saída, e sete definições de conexão, uma das quais é recorrente. O segundo gene é desativado, de modo que a ligação especifica (entre os nodos 2 e 4) não seja expressa no fenótipo, de acordo com o que é mostrado na segunda coluna da segunda lista mostrada acima.
Fonte: adaptado de Stanley e Miikkulainen (2002).

2.2.2.2 Princípios do NEAT

No *NEAT* sempre que um novo gene aparece (por meio de mutação estrutural), estabelece-se uma cronologia de aparecimento do mesmo no sistema. Desse modo, a origem do histórico de cada gene no sistema é conhecida em toda a evolução (STANLEY; MIIKKULAINEN, 2002). O *NEAT* protege a inovação através da especiação, a ideia é dividir a população em espécies tal que topologias semelhantes pertençam à mesma espécie, o que se torna um problema de agrupamento de topologias correspondentes. Outra estratégia adotada

pelo *NEAT* foi minimizar a dimensionalidade através do crescimento incremental de estruturas minimas.

Figura 4 – Os dois tipos de mutação estrutural no *NEAT*



Ambos os tipos, a adição de uma conexão e adição de um nó, são ilustradas com os genes de conexão de uma rede representada acima por seus fenótipos. O número em cima de cada genoma é o número de inovação desse gene. Os números de inovação são marcos históricos que identificam o histórico original dos antepassados de cada gene. Os novos genes são atribuídos novos números cada vez maiores. Em adição a uma conexão, um gene de conexão único novo é adicionado ao fim do genoma dado o número inovação seguinte disponível.
Fonte: adaptado de Stanley e Miikkulainen (2002).

O *NEAT* foi testado na simulação de dois problemas o *XOR* e o balanceamento de pólos. Em parte, os bons resultados alcançados pelo *NEAT* são devidos às partes do mesmo, que trabalham juntas. Uma das vantagens do método é encorajar a evolução de soluções mínimas. O método usa codificação direta que não permite representações genotípicas compactas o que é uma desvantagem. A estrutura de dados do *NEAT*, que armazenam os genótipos (arquiteturas de RNAs), cresce linearmente com o número de sinapses entre dois neurônios, ou seja, o método apresenta problemas de escalabilidade.

2.2.3 *Optimization of modular granular neural networks using hierarchical genetic algorithms (MOHGA)*

Sánchez, Melin e Castillo (2015) propuseram um novo modelo de otimização multi-objetivo para um AG hierárquico denominado *MOHGA* com base na abordagem de micro-GA para otimização de redes neurais modulares. A abordagem é utilizada no reconhecimento da íris. O *MOHGA* divide os dados automaticamente em grânulos ou submódulos e escolhe quais dados são utilizados para o treinamento e quais são para testes. O método proposto é responsável por determinar o número de grânulos ou submódulos e a porcentagem de dados

para o treinamento das redes e que podem permitir melhores resultados. Entretanto, o método não foi testado em outras aplicações tais como predicação de séries temporais, além disso, o mesmo não possibilita gerar redes neurais recorrentes.

2.2.4 Evolving Artificial Neural Networks

Donate, Sanchez e De Miguel (2012) propuseram um novo método para projeto automático de RNAs, aplicado a PST usando AG. O método altera ambos: os pesos das conexões sinápticas e a topologia das RNAs. O objetivo foi melhorar a precisão da PST. Entretanto, esse projeto foi restrito as redes neurais diretas multicamadas, que não são tão apropriadas para simular esse tipo de tarefa de PST.

2.3 MÉTODOS DE CODIFICAÇÃO INDIRETA

Nessa secção apresentam-se diversos métodos que utilizam ECI. Inicialmente, apresentam-se métodos que utilizam gramáticas Kitano (1990), Ahmadizar et al. (2015) e Sistemas-L Boozarjomehry e Svrcek (2001), Lee, Seo e Sim (2008) e finalmente os que utilizam substratos **(**Stanley et al. 2009), RISI; STANLEY, 2012).

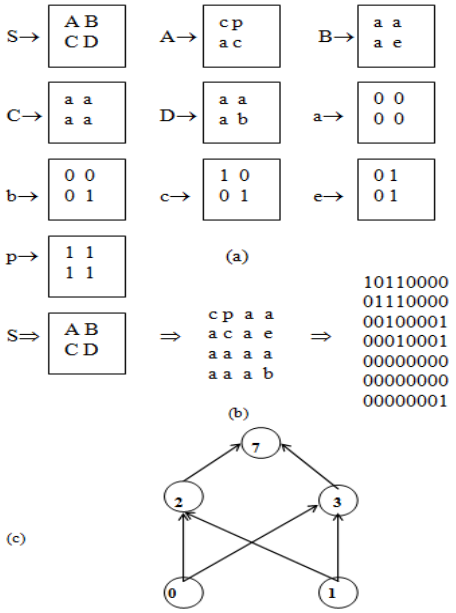
2.3.1 Método de codificação gramatical

2.3.1.1 Codificação cromossômica do Método de Kitano (1990)

Uma das primeiras pesquisas em modelos de desenvolvimentos artificiais foi a de Kitano (1990). O mesmo usou um tipo de gramática livre de contexto, um exemplo simples é mostrado na Figura 5. A matriz 8x8 da Figura 5(b), representa o esquema de conexão de uma rede neural mostrada na Figura 5(c). A gramática utilizada por Kitano (1990) foi: G={S,A,B,C,D.....Z,a,b,c,e,....p},{0,1},P,S}, as regras de produção são mostradas abaixo, na Figura 5(a). O cromossomo é ilustrado na Figura 6 sendo o mesmo dividido em regras separadas, cada qual com 5 posições. A primeira posição (célula) é o lado esquerdo da regra, as segundas até a quinta são ocupadas pelos 4 símbolos da matriz do lado direito da regra. Os possíveis genes em cada célula são os símbolos A-Z e a-p. A primeira posição do cromossomo é destinada a ser ocupada pelo símbolo inicial (axioma) S, pelo menos uma regra conduzindo S

a uma matriz 2x2 é necessária para iniciar o processo. Todos os outros símbolos são escolhidos ao acaso. Uma RNA é gerada aplicando-se as regras gramaticais codificadas pelo cromossomo para um número pré-determinado de iterações. As regras que levam a-p representadas pelas 16 matrizes 2x2 nos símbolos terminais 0 e 1, são fixas e não são representados no cromossomo. A aptidão foi calculada pela construção de redes neurais a partir da gramática, treinando-as com o AR e medindo a soma dos quadrados dos erros fixados na última época. Os problemas simulados foram o *XOR* e um codificador/decodificador simples.

Figura 5 – Ilustração do método de Kitano "gramática para geração de grafos"



(a) Regras da gramática, (b) Matriz de conexão produzida da gramática, (c) A rede resultante.
Fonte: adaptado de Kitano (1990).

Figura 6 – Ilustração do cromossomo codificando as regras de produção

S	A	B	C	D	A	C	p	a	c	B	a	a	a	e	.....

Fonte: adaptado de Kitano (1990).

2.3.1.2 Limitações do método de codificação gramatical

O método de Kitano (1990) utiliza um conjunto de regras para a construção de matrizes de conectividade com base na execução das regras, cada regra de desenvolvimento consiste em um lado esquerdo que é um elemento não terminal e um lado direito, que é uma matriz 2 × 2 com um elemento terminal ou um elemento não terminal. Como o tamanho final da matriz de conectividade desse método é $(2x2)^2$, o mesmo ainda sofre do problema de

escalabilidade. Ou seja, à medida que redes maiores vão sendo geradas esse método apresentará problemas de escalabilidade.

2.3.2 Métodos de codificação gramatical – tendências após o Método de Kitano (1990)

Por volta de do ano de 1992 surgiram outros métodos baseados em evolução gramatical. Os trabalhos de Boers e Kuiper (1992), Boers, Kuiper e Happel (1993) e Vaario (1993, 1994) buscaram inspiração na natureza e utilizaram uma outra espécie de gramática que não pertence a hierarquia das gramáticas propostas por Chomsky (1956), mas que se assemelha as linguagens sensíveis ao contexto, utilizando apenas um contexto a esquerda ou à direita. Essas pesquisas utilizaram um sistema de reescrita denominado Sistemas-L, que foram inicialmente usados para modelar o desenvolvimento de plantas (LINDENMAYER, 1968). Os Sistemas-L são mais adequados para modelar processos de desenvolvimento biológico, onde ocorrem múltiplas divisões celulares em paralelo e de forma interdependentes.

Como outro marco importante cita-se Yao (1999) que publicou outro estado da arte. Em sua pesquisa ele faz uma análise de vários trabalhos que estudam o problema de projeto automático de RNAs. Embora, nessa pesquisa sejam apresentados um número grande de metodologias de projeto automático de RNAs desenvolvidas até final da década de 90, o autor cita apenas um trabalho Yao e Shi (1995), como metodologia biologicamente inspirada e que usa coevolução e outro de Merrill e Port (1991), que utiliza geometria fractal para gerar arquiteturas de RNAs. Outras pesquisas importantes nesse período, que não usam Sistemas-L, foram os trabalhos de Voigt, Born e Santibanez-Koref (1993), Vonk, Jain e Johnson (1995), Mjolsness, Sharp e Alpert (1989), Gruau (1992) e Harp, Samad e Guha (1990).

2.3.3 Métodos de codificação indireta que utilizam evolução gramatical

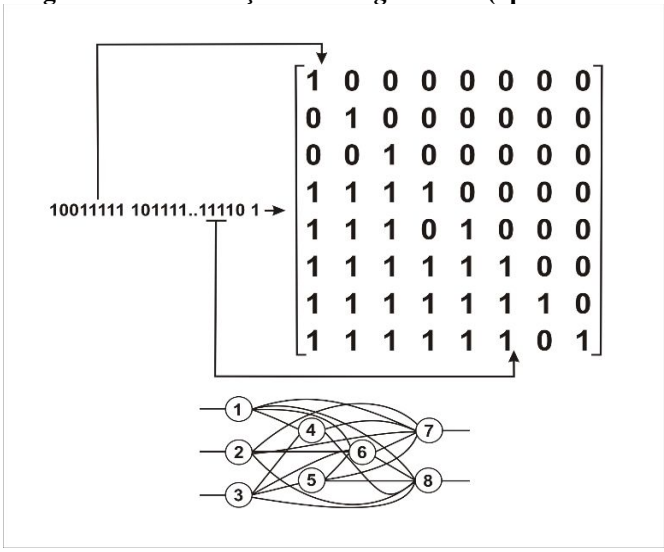
2.3.3.1 Método GADON

2.3.3.1.1 Introdução

Boozarjomehry e Svrcek (2001) descrevem um método para projetar e otimizar estruturas de redes neurais, que é um método paralelo que usa um Sistema-L (LINDENMAYER, 1968) livre de contexto para codificar as regras de desenvolvimento no

genótipo. Para evitar o problema da escalabilidade, em vez de fazer o axioma crescer em duas dimensões (i.e substituindo cada elemento da matriz por uma matriz 2x2), como ocorre no método de Kitano (1990), o crescimento se dá em uma única dimensão. O espaço de busca se limita a redes diretas, restringindo esse domínio e possibilitando que as conexões entre neurônios não sejam limitadas apenas entre camadas adjacentes, irá possibilitar a uma matriz triangular menor como mostra a Figura 7.

Figura 7 – Diferentes estágios de decodificação da *string* binária (após o crescimento), método *GADON*



Fonte: adaptado de Boozarjomehry e Svrcek (2001).

2.3.3.1.2 Método *GADON* – codificação cromossômica

Uma vantagem do método *GADON* de Boozarjomehry e Svrcek (2001) é que o mesmo melhora o problema de escalabilidade das metodologias apresentadas nas secções anteriores, uma vez que uma rede neural com 'N' neurônios irá resultar em uma *string* binária de tamanho (N(N+1) /2) oposto de N^2 do método de Kitano (1990), o que reduz o espaço de busca resultando em convergência do método em poucas gerações. Boozarjomehry e Svrcek (2001) não especificam formalmente o Sistema-L utilizado, os autores descrevem de forma geral que o mesmo é livre de contexto (LINDENMAYER, 1968).

2.3.3.1.3 Método *GADON* – Função de Aptidão

A função aptidão f dada pela equação 1, usada no *GADON,* é uma combinação linear do erro de treinamento, erro de validação e do número de conexões no neurônio.

$$\mathbf{f = -k1.et-k2.ev-k3.nc,} \quad (1)$$

Onde: **k1**=fator de peso do erro de treinamento; **k2**=fator de peso do erro de validação; **k3**=fator do número de conexões da rede, **et**=erro de treinamento, **ev**=erro de validação, **nc**=número de conexões da rede.

Os melhores valores dos fatores de peso da função aptidão foram encontrados por meio de um procedimento interativo (BOOZARJOMEHRY; SVRCEK, 2001), os valores são k1=0.5, k2=0.5 e k3=0.1.

2.3.3.1.4 Método *GADON* – Treinamento das RNAs e parâmetros do algoritmo genético

O algoritmo de retropropagação (AR) (RUMELHART; MCLELLAND, 1986; NARENDRA; PARTHASARATHY, 1991) foi usado para treinar a rede e avaliar o treinamento da função aptidão. Foi necessário generalizar o algoritmo de AR fazendo-o aplicável a todos os tipos de redes diretas válidas. Os parâmetros padrão para o algoritmo *GADON* foram os seguintes: tamanho do cromossomo 65, tamanho da população 20, limiar de convergência 0.95 , taxa de mutação 0.01 e **k1**=0.5, **k2**=0.5 e **k3**=0.1.

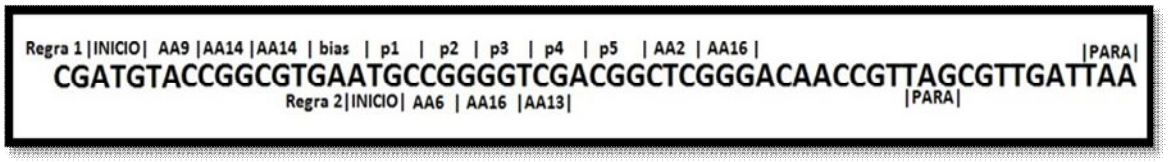
2.3.3.1.5 Método *GADON* – Potencialidades

A proposta de Boozarjomehry e Svrcek (2001) é menos intensiva computacionalmente do que os métodos de codificação direta e o método de Kitano (1990), e pode ser aplicado ao projeto automático de redes neurais que simulem problemas complexos tais como: no projeto de redes neurais utilizadas na simulação de processos industriais. O Algoritmo *GADON* foi testado em alguns *benchmarks* (*XOR*, *Pattern Copying*). Em adição a esses problemas, o *GADON* foi usado para obter a estruturas ótimas de redes neurais imitando o comportamento dinâmico de dois processos não lineares comumente encontrados em processos químicos industriais, tais como: a modelagem da dinâmica do processo de neutralização do *PH* e reações químicas não lineares utilizando reatores perfeitamente ajustados (RPA). A metodologia só é capaz de gerar topologias de redes neurais diretas.

2.3.3.2 Método baseado na codificação do ADN com Sistema-L livre de contexto

Um método biologicamente inspirado, que usa Sistema-L, foi desenvolvido por Lee, Seo e Sim (2008). Similar ao ADN biológico, a informação é codificada usando os símbolos A,G,T e C. Uma sequência de três desses símbolos é conhecido como códon e cada gene é codificado pela sequência de códons começando com o de início e terminando com o de parada. A sequência de códons entre esses delimitadores é traduzida em uma regra de produção para o desenvolvimento de um controlador neural. Como mostrado na Figura 8 a representação do cromossomo pode ter múltiplas interpretações, pois os códons de **INÍCIO (START)** e **PARADA (STOP)** permitem a sobreposição de codificações de genes no ADN, isto tem a vantagem de compactação, uma vez que podem ser codificadas mais regras do que nos genomas em que as regras não podem se sobrepor.

Figura 8 – Regras de Produção Codificadas no ADN



Fonte: adaptado de Lee, Seo e Sim (2008).

2.3.3.2.1 Codificação e extração das regras de produção no ADN

Na pesquisa de Lee, Seo e Sim (2008) um cromossomo do ADN é traduzido em um aminoácido, ver Tabela 1 e depois em uma regra de produção do Sistema-L. O mesmo usa o alfabeto **V={A,B,C,D}**. A Tabela 1 mostra a translação do código do ADN onde cada códon é convertido em um aminoácido hipotético. A Tabela 2 converte um aminoácido para um nó ou para uma faixa de conexão (x,y) onde os parâmetros x e y denotam o índice do primeiro e último nós a serem conectados, sendo esses parâmetros a máxima faixa de conexão. A Tabela 3 mostra uma regra de produção na forma **A→B** que é composta de nove códons correspondentes a um nó predecessor **(A)**, um nó sucessor **(B)**, uma faixa de conexão **(x,y)**, um bias **(Wo)**, 5 pesos de conexão **(W1,W2,W3,W4,W5),** os mesmos são necessários devido a máxima faixa de conexão ser cinco. Um nó simples predecessor pode ter múltiplos nós sucessores tal como na regra **A→BC**. Bias e pesos são valores reais calculados pela equação 2. O valor do bias e pesos tem valores na faixa de -3.2 a 3.1 em intervalos de 0.1.

A Tabela 4 mostra um exemplo de translação do código do ADN, Figura 8, em regras de produção. Duas podem ser criadas desde que dois códons de **INICIO ATG** existem no cromossomo. O Códon **TAC** seguido de **ATG** é traduzido em **AA9** (consultar Tabela 1), que corresponde ao nó **C** de acordo com a Tabela 2. O próximo códon **CGG** é traduzido para **AA14** que equivale ao nó **D**. O próximo códon **CGT** é também traduzido para **AA14**, que corresponde a faixa de conexão **(4,4)**. O próximo GAA denota um bias cujo valor é calculado por **$((3x4^2+2x4^1+2x4^0)-32/10)=2.6$**, de acordo com a equação 2. Os próximos cinco códons determinam os valores dos pesos. Este procedimento é repetido para a próxima regra de produção até que o códon de **PARADA (STOP)** é encontrado. A primeira regra de produção é representada por **p(C)=D(4,4).A(4,5)** e a segunda obtida é **p(B)=D(1,5),** ver Tabela 4, essas regras são as mesmas mostradas na Figura 8.

Tabela 1 – Translação do código do ADN

Códon	Aminoácido	Códon	Aminoácido	Códon	Aminoácido	Códon	Aminoácido
TTT	AA1	**TCT**	AA5	**TAT**	AA9	**TGT**	AA13
TTC		**TCC**		**TAC**		**TCG**	
TTA		**TCA**		**TAA**	STOP	**TGA**	STOP
TTG		**TCG**		**TAG**		**TGG**	
CTT	AA2	**CCT**	AA6	**CAT**	AA10	**CGT**	AA14
CTC		**CCC**		**CAC**		**CGC**	
CTA		**CCA**		**CAA**		**CGA**	
CTG		**CCG**		**CAG**		**CGG**	
ATT	AA3	**ACT**	AA7	**AAT**	AA11	**AGT**	AA15
ATC		**ACC**		**AAC**		**AGC**	
ATA	AA3 /START	**ACA**		**AAA**		**AGA**	
ATG		**ACG**		**AAG**		**AGG**	
GTT	AA4	**GCT**	AA8	**GAT**	AA12	**GGT**	AA16
GTC		**GCC**		**GAC**		**GGC**	
GTA		**GCA**		**GAA**		**GGA**	
GTG		**GCG**		**GAG**		**GGG**	

Fonte: adaptado de Lee, Seo e Sim (2008).

Tabela 2 – Tabela de transcrição de aminoácidos

Aminoácido	Nome do Nó	Faixa de Conexão	Aminoácido	Nome do Nó	Faixa de Conexão
AA1	**A**	**(1,4)**	**AA9**	**C**	**(1,5)**
AA2	**A**	**(1,1)**	**AA10**	**C**	**(3,3)**
AA3	**A**	**(1,3)**	**AA11**	**C**	**(3,5)**
AA4	**A**	**(1,2)**	**AA12**	**C**	**(3,4)**
AA5	**B**	**(2,5)**	**AA13**	**D**	**(1,5)**
AA6	**B**	**(2,2)**	**AA14**	**D**	**(4,4)**
AA7	**B**	**(2,4)**	**AA15**	**D**	**(5,5)**
AA8	**B**	**(2,3)**	**AA16**	**D**	**(4,5)**

Fonte: adaptado de Lee, Seo e Sim (2008).

Tabela 3 – Codificação no ADN de uma regra de produção

Nó (P)	Nó (S)	Faixa de Conexão	Bias	Pesos
A	**B**	**(x,y)**	**Wo**	**W1,W2,W3,W4,W5**

Fonte: adaptado de Lee, Seo e Sim (2008).

$$w = \frac{(b_2 4^2 + b_1 4^1 + b_0 4^0) - 32}{10}$$

(2)

Onde, bo, b1 e b2 são os três símbolos do códon do ADN (por exemplo: ACG). Os valores de cada símbolo ADN são T=0, C=1, A=2 e G=3.

Tabela 4 –Tradução das regras de produção do ADN

	Predecessor	Sucessor
Regra 1	(AA9)	(AA14)(AA14)GAA TGC CGG GGT CCA (AA2)(AA16)ACA ACC ACC GTT AGC GTT CGG
	C	D (4,4) 2.6 -1,9 -0.1 2.8 -1.0 - 0.1 A (4,5) 0.6 0.5 0.5 1.6 1.3 1.6
Regra 2	(AA6)	(AA16)(AA13)ACG GCT CGG GAC AAC CAC
	B	D (1,5) 0.7 2.0 -0.1 2.5 0.9 -0.7

Fonte: adaptado de Lee, Seo e Sim (2008).

Para construir uma rede neural um Sistema-L **G={V,P,w}** é usado onde **V={A,B,C,D}**, **P={p1,p2,p3,p4}** e **w=4,** as regras de produção são como:

p1=p(A)=A(5,5)C(1,2)D(2,3)B(2,4)

p2=p(B)=D(1,5)B(2,4)C(1,4)

p3=p(C)=A(3,3)D(1,4)C(2,4)

p4=p(D)=B(4,5)

A Rede neural é criada após três passos de reescrita usando regras evoluídas, por meio das quais obtemos as seguintes *strings*. S1, S2 e S3.

S1:A(5,5)C(1,2)D(2,3)B(2,4)

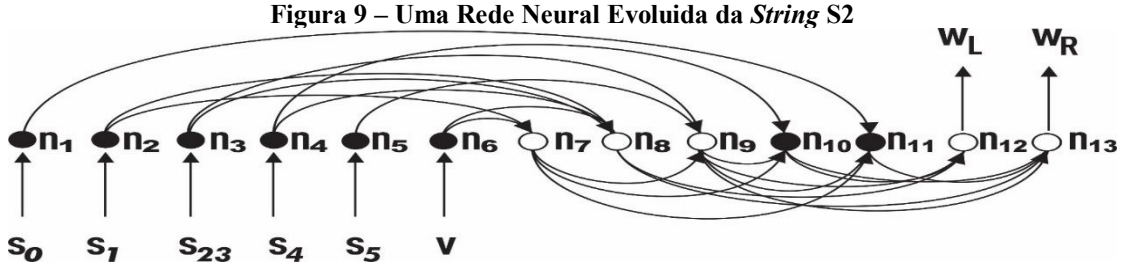
S2:A(5,5)C(1,2)D(2,3)B(2,4)A(3,3)D(1,2)C(2,4)B(4,5) D(1,5)B(2,4)C(1,4)

S3:ACDBADCBDBCACDBBADCDBCBDBCADC

A Figura 9 mostra uma arquitetura de rede evoluída obtida a partir da *string* S2. A *string* obtida a partir de S3 foi descartada, pois a rede apresentou baixa performance. Utilizando a *string* S2 ilustrada acima A(x,y)=A(5,5)=n1, o nó n1 estará conectado entre o nó 11 (pois existem 6 neurônios na entrada e soma-se a esse valor a faixa de conexão x do nó A que é 5) e o nó 11 (pois existem 6 neurônios na entrada e soma-se a esse valor a faixa de conexão y do nó A que é 5), como o neurônio 1 (n1) faz parte da camada de entrada, o mesmo

não pode ter conexões com neurônios dessa mesma camada, por isso soma-se 6 a sua faixa de conexão (x,y), no presente caso tem-se duas conexões entre n1 e n11, sendo uma redundante. Da mesma forma para C(1,2)=n2, o nó 2 deve ser conectado entre os nós 7 e 8. Para os nós que estão situados após os neurônios da camada de entrada, por exemplo o nó 7, C(2,4), o mesmo deverá estar conectado entre os neurônios 9 e 11, pois não existem conexões recorrentes, ou seja, todas devem partir do nó sete para frente. O nó 8, D(4,5) é conectado aos nós 12 e 13 e o nó 9, representado por D(1,5), é conectado aos nós 10 e 11.

Figura 9 – Uma Rede Neural Evoluida da *String* S2



Fonte: Adaptado de Dong-Wook, Sang-Wook e Kwee-Bo (2008).

2.3.3.2.2 Avaliação de Aptidão

Um robô móvel *Khepera* é usado para testar como desenvolver controlador usando rede neural evolucionária. O robô tem oito sensores de proximidade (seis na frente e dois na parte traseira). O alcance de detecção é aproximadamente 50 mm. Para evoluir a rede neural, a função de aptidão é dada por:

$$\mathbf{Fitness} = \frac{1}{2}\left(\frac{d_{max}-d_R}{d_{max}} + \frac{c_{max}-c_R}{c_{max}}\right)$$

(3)

Onde: dmax denota a máxima distância do ponto de partida ao destino, **dR** é distância do robô ao destino, **cmax** é um valor máximo predefinido do número máximo de colisões e **cR** o número de colisões. Se **cR** é maior do que cmax, então **cR**=cmax.

O robô móvel encontrou o objetivo sem colisões em 95 gerações e após isso atingiu o valor máximo de aptidão. A mesma metodologia foi utilizada por Dong-Wook, Sang-Wook e Kwee-Bo (2008) para predição de séries temporais com um passo à frente, os mesmos utilizaram as séries de *Mackey-Glass* (MACKEY; GLASS, 1977) e *Sunspot*.

2.3.3.3 Desenvolvimento de RNAs por meio da combinação de evolução gramatical e algoritmo genético (GEGA)

Os autores Ahmadizar et al. (2015) desenvolveram um algoritmo para evoluir simultaneamente a topologia e pesos das conexões de RNAs, por meio da combinação da EG e um AG. EG é utilizado ao projeto da topologia da RNA, enquanto o AG é incorporado para a adaptação dos pesos da RNA. O Algoritmo proposto necessita de um esforço mínimo do especialista para sua customização e é capaz de gerar redes diretamente alimentadas com uma única camada. Além disso, devido ao fato de que a capacidade de generalização de uma RNA pode diminuir por causa de problemas de o*verfitting*, o algoritmo, denominado *GEGA*, utiliza uma nova abordagem de penalidade adaptativa para simplificar as RNAs geradas durante o processo de evolução. Como resultado, o método produz RNAs muito mais simples com boa capacidade de generalização e que são fáceis de implementar. O método proposto é testado em alguns *benchmarkings* de classificação do mundo real e os resultados são comparados estatisticamente em relação a outros existentes na literatura. Os mesmos indicam que o algoritmo supera os outros e fornece o melhor desempenho global em termos de precisão da classificação e do número de neurônios ocultos. Os resultados também mostram a contribuição da abordagem de penalidade proposta na simplicidade e na habilidade de generalização das redes geradas.

2.3.3.3.1 Representação do método de codificação

O *GEGA* utiliza um esquema de codificação hibrido (direto e indireto) em que cada cromossomo inclui duas partes: um para a topologia da RNA representado por um vetor de números inteiros na faixa de [0,255]. O número de genes na parte da topologia em cada cromossomo na população inicial é determinado para ser igual a 100. Para gerar a topologia, uma gramática BNF é empregada. A mesma é representada por uma tupla **{N,T,P,S}**, onde **N** denota um conjunto de símbolos não terminais, **T** o conjunto de símbolos terminais, **P** as regras de produção e **S** o axioma ou símbolo inicial que inicia o processo de geração. A gramática é capaz de gerar qualquer rede direta com uma camada escondida sendo a mesma mostrada na Figura 10, onde **x1...xn** são as entradas e **w** os pesos das conexões, **sig** denota a função de ativação sigmóide. A Figura 11 ilustra um exemplo de representação para o cromossomo para um problema com duas entradas **x1** e **x2** e uma saída **O**. As etapas do

processo de geração são ilustradas na Figura 12, o axioma **<S>**, inicia o processo, desde que o mesmo não representa um símbolo terminal o mesmo deve disparar uma regra de produção, da gramática proposta. Como **<S>** tem duas regras de produção **<Node>** e **<Node>+<S>,** que podem ser disparadas, para determinar delas deve ser executa utiliza-se como critério o resto da divisão da primeira entrada da parte da topologia dividido pelo número de regras de produção. Por exemplo, se o resto é 1 a regra de produção com o índice 1, que é, **<Node>+<S>** é utilizada. Continuando dessa forma, até que não existam mais símbolos não terminais na sequencia gerada, o processo de geração terminará com sucesso.

Figura 10 – Gramática Proposta para geração da Topologia da RNA

Expressão	Valor Gene	Número de Produções	Índice da Produção Selecionada
<S>	111	2	111 mod 2 = 1
<Node>+<S>	---	1	0
w*sig(<Sum>+w)+<S>	47	2	47 mod 2 = 1
w*sig(w*<xxList>+<Sum>+w)+<S>	58	2	58 mod 2 = 0
w*sig(w*x1+<Sum>+w)+<S>	12	2	12 mod 2 = 0
w*sig(w*x1+w*<xxList>+w)+<S>	39	2	39 mod 2 = 1
w*sig(w*x1+w*x2+w)+<S>	28	2	28 mod 2 = 0
w*sig(w*x1+w*x2+w)+<Node>	---	1	0
w*sig(w*x1+w*x2+w)+w*sig(<Sum>+w)	26	2	26 mod 2 = 0
w*sig(w*x1+w*x2+w)+w*sig(w*<xxList>+w)	125	2	125 mod 2 = 1
w*sig(w*x1+w*x2+w)+w*sig(w*x2+w)			**Genótipo Gerado**

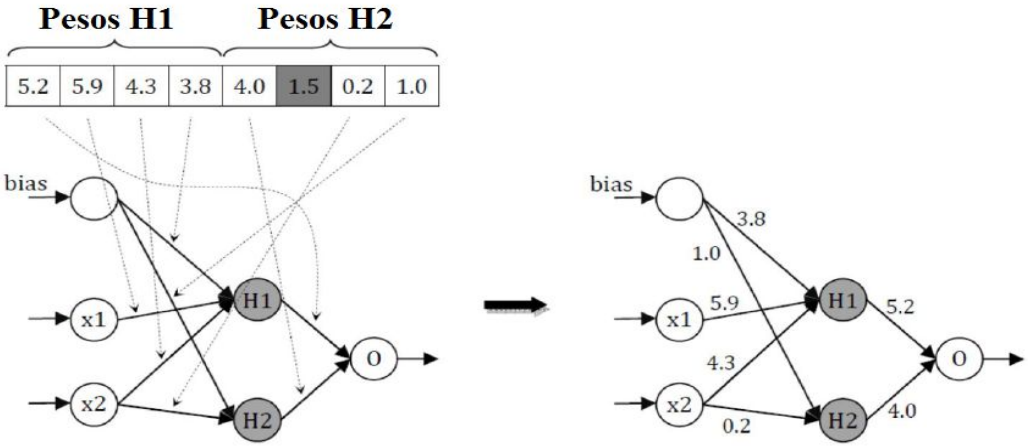
Fonte: adaptado de Ahmadizar et al. (2015).

Figura 11 – Um exemplo de codificação cromossômica.

Parte Pesos	5.2	5.9	4.3	3.8	4.0	1.5	0.2	1.0
Parte Topologia	111	47	58	12	39	28	26	125

Fonte: adaptado de Ahmadizar et al. (2015).

Figura 12 – A RNA correspondente ao cromossomo mostrado na Figura 11



Fonte: adaptado de Ahmadizar et al. (2015).

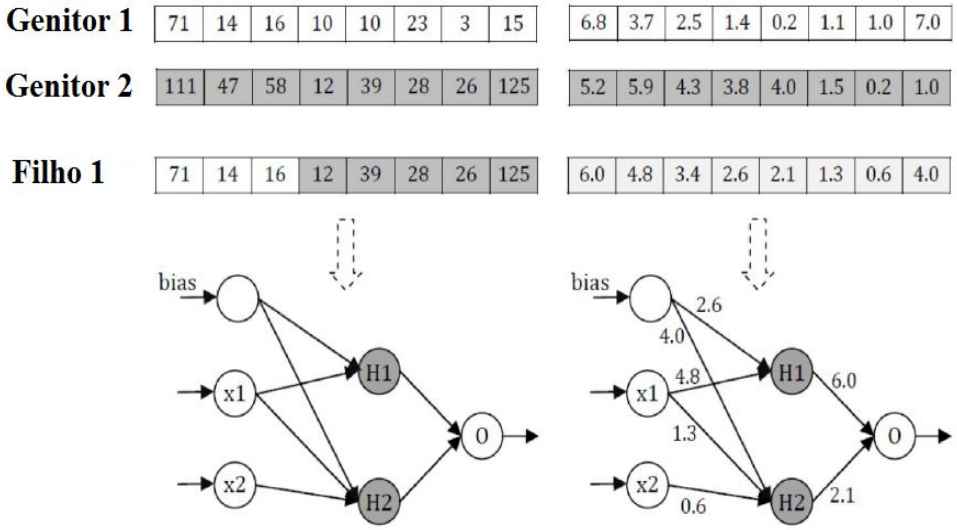
A partir da topologia gerada mostrada na Figura 12, a rede resultante tem dois neurônios escondidos; o primeiro neurônio **H1**, tem duas conexões com **x1** e **x2**. Enquanto que, o segundo, **H2** tem uma única conexão com **x2**. Se os pesos mostrados na Figura 11 são inseridos na sequência, a seguinte *string*, que é então a entrada para o neurônio de saída da RNA mostrada na Figura 2.12, torna-se então:

$$5.2 \text{ sig } (5.9 \text{ x } 1 + 4.3 \text{ x } 2 + 3.8) + 4.0 \text{ sig } (0.2 \text{ x } 2 + 1.0) \quad (4)$$

2.3.3.3.2 Operador de cruzamento

Por recombinação do código genético de dois ancestrais, o operador genético de cruzamento produz duas soluções em uma região não visitada do espaço de busca. Para cada par de genitores, o operador de cruzamento, que é uma combinação de dois pontos de cruzamento diferentes, é aplicado de acordo com uma probabilidade de cruzamento **Pc**. No caso do *GEGA*, foi utilizado um único ponto de corte, sendo o mesmo executado na parte topologia dos dois cromossomos, enquanto que uma recombinação intermediária é aplicada a parte dos pesos. A Figura 13 ilustra um exemplo da operação de cruzamento.

Figura 13 – Um exemplo da operação de cruzamento

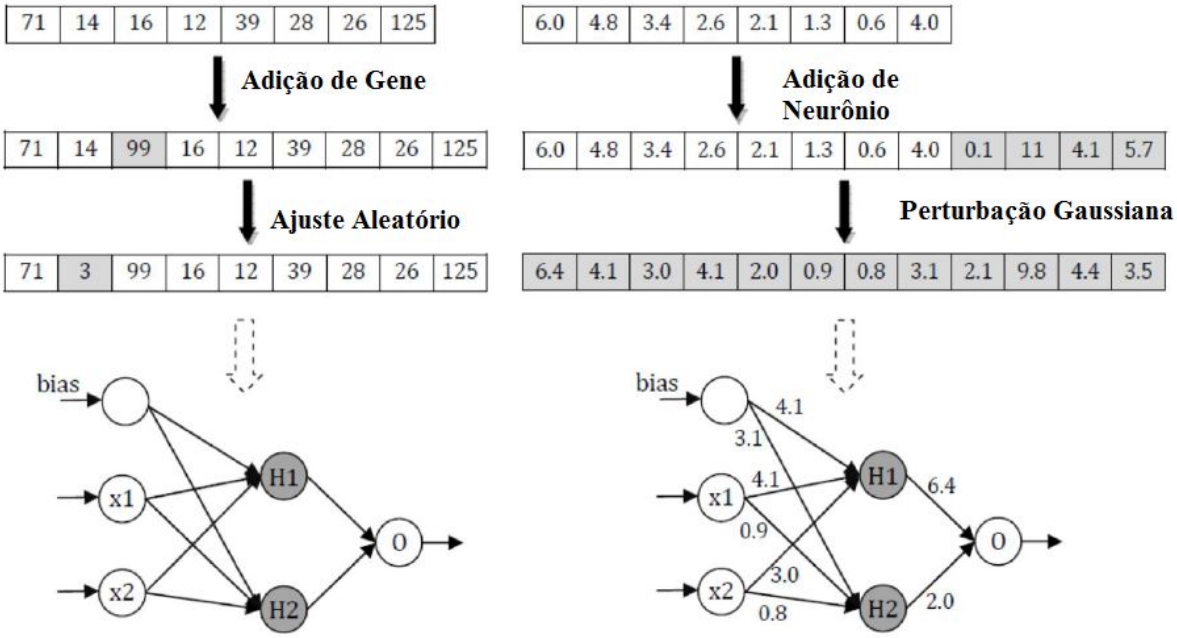


Fonte: adaptado de Ahmadizar et al. (2015).

2.3.3.3.3 Operador de mutação

Após a operação de cruzamento, a de mutação é realizada varrendo os valores dos genes dos cromossomos, sendo aplicada a cada descendente gerado com o objetivo de prevenir que o algoritmo seja direcionado para mínimos locais. No *GEGA* utilizam-se quatro tipos de operadores de mutação. Primeiro um novo gene é adicionado a parte topologia do cromossomo com uma probabilidade **Pm**, a posição do gene e o seu valor são escolhidos randomicamente. Para cada gene na parte topologia do cromossomo, um novo valor é escolhido randomicamente no intervalo [0,255] com probabilidade **Pm**. Além disso, com probabilidade **Pm/2** um neurônio é adicionado ou deletado a partir da parte peso do cromossomo. Então, para cada gene na parte peso uma quantidade é randomicamente escolhida para ser aplicada uma perturbação gaussiana com média zero e desvio padrão randomicamente selecionado de {0.2,0.5,1,2,5} é adicionado ao valor do gene atual, a Figura 14 ilustra esse processo.

Figura 14 – Um exemplo da operação de mutação



Fonte: adaptado de Ahmadizar et al. (2015).

2.3.3.3.4 Avaliação da Aptidão

GEGA procurou gerar RNAs que tenham não somente um erro médio quadrático pequeno no treinamento, mas uma capacidade de generalização alta na fase de testes. Um novo mecanismo de penalidade é incorporado ao *GEGA* para torná-lo capaz de gerar RNAs simples com boa capacidade de generalização. Dessa forma, o erro (isto é, a função de aptidão de treino a ser minimizada no processo evolucionário) para um indivíduo *i* é definida por:

$$E(i) = EMQ(i) + p^{co}h(i) \quad (5)$$

Onde: **h(i)** : é o número de neurônios da camada escondida do indivíduo **i** p^{co} é o coeficiente de penalidade.

Considerando, o $\overline{EMQ}$ e $\bar{h}$, respectivamente, como o erro médio quadrático e o número médio de neurônios na camada intermediária da população atual. O Coeficiente de penalidade é calculado como:

$$p^{co} = q(\bar{h})/\overline{EMQ} \quad (6)$$

Onde: $q(\bar{h})$ é uma função crescente em $(\bar{h})$.

Uma vez que o algoritmo gera RNAs com um pequeno número de neurônios na camada intermediária, porém com um alto erro médio quadrático, uma pequena penalidade tende a gerar RNAs mais complexas nas gerações seguintes, com o objetivo de melhorar o erro médio quadrático e vice e versa.

2.3.3.4 Outros métodos que utilizam evolução gramatical

Tsoulos, Gavrilis e Glavas (2008) usaram evolução gramatical para o projeto e treinamento de RNA com uma única camada escondida. Rivero et al. (2010) utilizou programação genética para projeto e treinamento de RNA diretas. Tsoulos, Gavrilis e Glavas (2008) discutem que o uso da evolução gramatical tem o benefício de permitir formatação fácil do processo de busca, permitindo um esquema de codificação compacto oferecendo vantagens em termos de escalabilidade. Esses métodos geram genótipos que são muito menores do que os fenótipos que eles produzem. Consequentemente, o espaço de busca torna-se mais viável. Soltanian et al. (2013) aplicou evolução gramatical apenas para o projeto da topologia enquanto os pesos são otimizados pelo AR, que é utilizado para o treinamento e testes das RNAs.

2.3.4 Métodos de Codificação utilizam Evolução de Substratos

2.3.4.1 "Hybercube based NeuroEvolution of Augmenting Topologies (HyperNEAT)"

O método (HyperNEAT), (Gauci e Stanley, 2008, 2010) e (Lee et al., 2013), é um método de neuroevolução, que significa que o mesmo evolui RNAs por meio de um algoritmo evolucionário. O método *HyperNEAT* (Stanley, 2007), (Stanley et al. 2009) é uma evolução do método *NEAT*, apresentado na secção 2.2.2. O *HyperNEAT* se baseia em uma teoria que hipotetiza que uma boa representação para uma RNA deve ser capaz de codificar o seu padrão de conectividade de forma compacta. Esse enfoque de codificação usado no *HyperNEAT* é denominado *Compositional Pattern Producing Networks* (*CPPNs*), que são abstrações de desenvolvimento que podem representar padrões com regularidades tais como: simetria, repetição e repetição com variação. Dessa forma, os padrões de pesos através da conectividade de uma RNA podem ser gerados através de uma função da sua geometria (D'AMBROSIO;

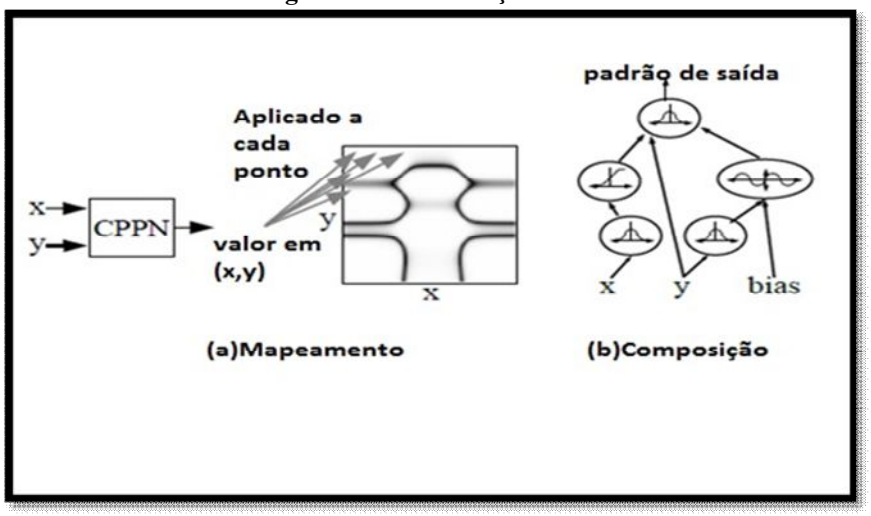
STANLEY, 2007). Na maioria das codificações diretas, tal geometria não pode ser explorada porque a mesma não pode ser aproveitada no esquema de codificação.

2.3.4.1.1 Codificação cromossômica utilizada no *HyperNEAT*

Para melhor entender o esquema de codificação utilizado pelo *HyperNEAT*, deve-se considerar que a ideia subjacente é a de que padrões *CPPNs* , tais como aqueles observados na natureza podem ser descritos em um alto nível como uma composição de funções que são escolhidas para representar vários padrões em comum. Por exemplo, Stanley (2007), Stanley, Bryant e Miikkulainen (2005), Clune et al. (2011) e Clune, Chen e Lipson (2013) mostraram que devido a função gaussiana ser simétrica, quando é composta com qualquer outra função, o resultado é um padrão simétrico. O objetivo dessa codificação é permitir que os padrões com regularidades com simetria (função gaussiana), repetição (funções periódicas como o seno) e a repetição com variação (soma de funções periódicas e aperiódicas) possam ser representados como redes de funções simples. A Figura 15 mostra como um padrão espacial bidimensional pode ser gerado por um *CPPN* que recebe duas entradas.

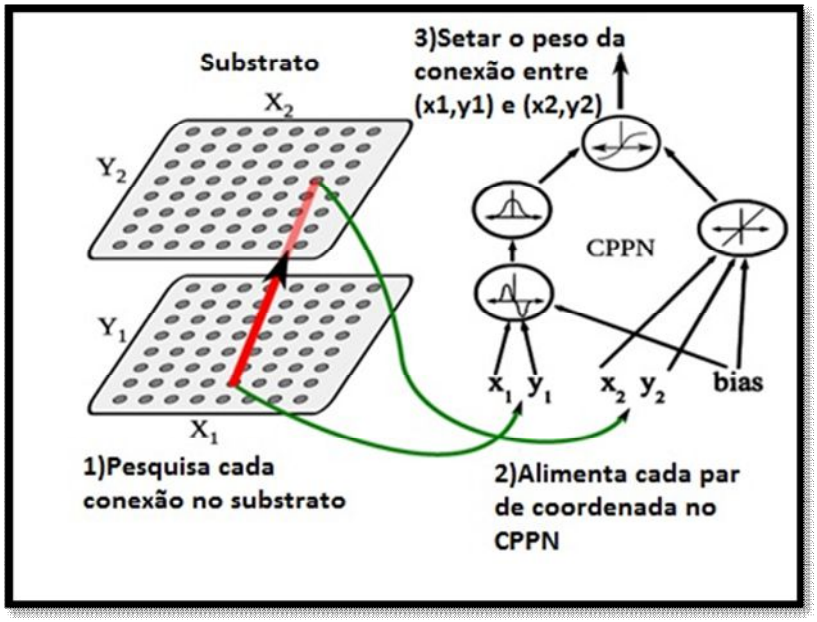
Na Figura 15 apresentou-se a ideia geral de como o *CPPN* pode representar um padrão espacial, onde o mesmo ilustra o mapeamento entre o genótipo *CPPN* e o fenótipo (intensidade de cor especificada pelas coordenadas (x,y) do objeto mostrado na Figura 2.15 a). A Figura 16 ilustra como essa ideia pode ser estendida para representações fenotípicas de RNAs. Na Figura 16 o *CPPN* toma como entrada duas posições em vez de uma e a sua saída especifica o peso da conexão que liga os dois nodos correspondentes. O *CPPN* pode ter como entradas quatro pontos x1; y1; x2 e y2; em um espaço de quatro dimensões e indicar os pesos de conexões entre os mesmos. Entretanto, podem também indicar a conexão entre dois pontos bidimensionais (x1;y1) e (x2;y2), sendo que a saída do *CPPN* representa o peso da referida conexão.

Figura 15 – Codificação CPPN



O CPPN pega dois argumentos x e y, que são as coordenadas em um espaço bidimensional. Quando todas as coordenadas são desenhadas com uma intensidade correspondente a saída do *CPPN*, o resultado é um padrão espacial, o que pode ser visto como um fenótipo cujo genótipo é o *CPPN*. (b) Internamente, o CPPN é um grafo que determina quais funções estão conectadas. Como em uma RNA, as conexões são ponderadas de tal modo que a saída de uma função é multiplicada pelo peso da sua conexão de saída. O *CPPN* em (b), na verdade, produz o padrão em (a).
Fonte: adaptado de Gauci e Stanley (2008, 2010).

Figura 16 – Hipercubo baseado na Interpretação do Padrão Geométrico de Conectividade.



Uma grade de nós é chamada substrato, à qual são atribuídas coordenadas. (1) A cada conexão potencial no substrato é consultada para determinar a sua presença e peso, a linha dirigida mostrado no substrato representa um exemplo de conexão que é consultada. (2) Para cada consulta, o *CPPN* toma como entrada as posições dos dois pontos e (3) emite o peso da ligação entre eles. Desta forma, *CPPNs* conjuntivos produzem padrões regulares de conexões no espaço.
Fonte: adaptado de Gauci e Stanley (2010).

Ao consultar todas as possíveis conexões entre um conjunto de pontos desta maneira, um *CPPN* pode produzir uma RNA, em que cada ponto consultado é a posição de um

neurônio. O substrato ilustrado na Figura 16 é uma representação no formato sanduíche, na pesquisa de Stanley, David e Jason (2009) são referenciados outros tipos de configurações tais como: em formato de grade, tridimensional e circular.

O *CPPN* desempenha o papel do ADN na natureza, porém em um nível de abstração que não considera que o desenvolvimento do sistema nervoso é produzido pela informação genética codificada no ADN (contida em cada célula do organismo) e que quando seguida resultará na forma final do sistema nervoso. Ou seja, o *CPPN*, é uma rede de funções matemáticas que codifica um padrão de pesos que é desenhado através da geometria da rede. Não existe uma associação entre os genes influenciando o desenvolvimento e sua transmissão às gerações posteriores.

2.3.4.1.2 Evolução do *CPPN* e Medição da Aptidão

A abordagem dos trabalhos de Stanley (2007), Risi et al. (2009), Risi, Lehman e Stanley (2010), Gauci e Stanley (2010) e Clune, Chen e Lipson (2013) evoluem *CPPNs* utilizando *NEAT*, apresentado na secção 2.2.2. Esta abordagem é chamada *HyperNEAT* porque *NEAT* evolui *CPPNs* que representam padrões espaciais no hiperespaço. O esquema básico do algoritmo *HyperNEAT* procede como segue no Algoritmo descrito a seguir:

Quadro 1 – Algoritmo Básico *HyperNEAT*

1-Input: Escolha a Configuração do substrato (i.e nós de entrada, saídas e atribuições)
Output: Solução do *CPPN*
2-Inicialize a população de *CPPNs* mínimos com pesos aleatórios;
3-Enquanto o critério de parada não é atingido **faça**
(a)Para cada CPPN na população **faça**
Para cada possível conexão no substrato faça
(i) Pesquise o *CPPN* para o peso de conexão w;
Se Abs(w)> "threshold" **então**
Criar uma ligação com um peso proporcionalmente dimensionada para w (Figura 2.12);
f**im**
fim
(ii) Execute o substrato como uma RNA no domínio da tarefa para verificar a Aptidão;
fim
(b)Reproduzir *CPPNs* usando o *NEAT* para produzir a próxima geração;
fim
Saída da CPPN vencedora;

Fonte: Gauci e Stanley (2010).

2.3.4.1.3 Resultados do *HyperNEAT*

Na tarefa de discriminação visual, Stanley, David e Janson (2009) compararam o *HyperNEAT* com o *P-NEAT*, que é um *NEAT* que evolui *perceptrons*. O *HyperNEAT* aprendeu a generalizar a partir de seu treinamento, a diferença entre o desempenho do *HyperNEAT* em generalização e avaliação não é significativa após a primeira geração. Por outro lado, *P-NEAT* obteve desempenho significativamente pior no ensaio de generalização após a geração 51 ($p < 0.01$). Esta disparidade de generalização reflete a habilidade fundamental *HyperNEAT* em aprender o conceito geométrico subjacente da tarefa, que pode ser generalizada em todo o substrato. *P-NEAT* só pode descobrir cada peso conexão adequado independentemente do conceito geométrico subjacente. A segunda tarefa simulada no trabalho de Stanley, David e Janson (2009) foi a do robô de coleta de alimentos, os resultados confirmam a superioridade dos métodos de codificação indireta em relação aos de codificação direta.

Na pesquisa de Gauci e Stanley (2010) o jogo de damas é escolhido para os experimentos porque ele é intuitivamente geométrico. A ideia do *HyperNEAT* é aprender a partir da geometria, gerando a rede como uma função direta da geometria da tarefa. A seguir detalham-se as quatro estratégias utilizadas na pesquisa de Gauci e Stanley (2010) utilizou-se um *NEAT*-Regular, secção 2.2.2, a segunda estratégia foi o *NEAT-EI* com a tentativa de aumentar a capacidade do *NEAT* para levar em conta regularidades geométricas do tabuleiro, a terceira utilizou o *HyperNEAT* e a última o *FT-NEAT* onde somente os pesos da RNA são evoluídos, mas não a topologia.

O papel da geometria mostrou-se potencialmente útil para a aprendizagem no domínio de damas. Os métodos *NEAT* regular e *FT-NEAT* não foram capazes de derrotar a heurística determinística em nenhuma execução do treinamento, enquanto *NEAT-EI* e *HyperNEAT* foram capazes de derrotá-las em todas as 20 execuções. *HyperNEAT* foi capaz de encontrar soluções de forma relativamente rápidas, enquanto *NEAT-EI* levou muito mais tempo, buscando soluções através do espaço multidimensional de RNAs. Além disso, as soluções produzidas pelo *HyperNEAT* generalizam significativamente melhor do que as soluções produzidas pelo *NEAT-EI*.

Apesar de suas capacidades novas, uma limitação significativa do *HyperNEAT* é que as posições dos nós conectados através desta abordagem devem ser decididas a priori pelo

projetista. Em outras palavras no *HyperNEAT* original, o projetista deve definir literalmente as localizações dos nós dentro do substrato.

2.3.4.2 ES-HyperNEAT

Nas pesquisas de Risi e Stanley (2011, 2012) apresenta-se uma metodologia de projeto automático de RNAs que utiliza um conjunto de técnicas avançadas em neuroevolução em um novo método denominado Substrato-Adaptativo *HyperNEAT* Evoluível. A abordagem combinada é plenamente capaz de determinar a geometria, densidade e a plasticidade de uma RNA evoluída neuromodulada. O Algoritmo recém-introduzido substrato evoluível *HyperNEAT* (*ES-HyperNEAT*) Risi e Stanley (2011, 2012) demonstrou que a colocação e densidade dos nós escondidos numa RNA podem ser determinados com base na informação implícita contida em um padrão de resolução de pesos infinito, evitando assim a necessidade de evoluir a colocação explícita.

2.3.4.2.1 *ES-HyperNEAT* iterado idéia fundamental

A idéia introduzida por Risi et al. (2009) e Risi, Hughes e Stanley (2010) é de que uma representação que codifica o padrão de conectividade da rede automaticamente contém informação implícita sobre onde os nós devem ser colocados. No *HyperNEAT* o padrão de conectividade é descrito pelo *CPPN* (seção 2.3.4.1.1), onde cada ponto no espaço de quatro dimensões denota uma conexão potencial entre dois pontos bidimensionais, pois o *CPPN* leva x1,y1,x2,y2 como entradas, que é uma função de coordenadas infinitas contínuas possíveis para estes pontos. Em outras palavras, o *CPPN* codifica potencialmente um número infinito de pesos de conexão dentro do hipercubo de pesos a partir do qual um subconjunto deve ser escolhido para ser incorporado no substrato da RNA. Se uma conexão é escolhida para ser incluída, em seguida, por necessidade, os nós que se conectam também devem ser incluídos no substrato.

Outra informação importante é que, para qualquer padrão determinado, há alguma densidade acima do qual o aumento adicional da mesma não oferece nenhuma vantagem. Por exemplo, se o hipercubo é um gradiente uniforme de pesos de conexões máximos (isto é, todos os pesos são a mesma constante), em efeito ele codifica um substrato que computa a mesma

função em cada nó. Assim, adicionando estes nós, não adicionará nenhuma informação nova (RISI; STANLEY, 2012).

Assim, a resposta para a questão sobre quais conexões devem ser incluídas no *ES-HyperNEAT* de modo que resolução suficiente possa capturar informações no hipercubo mais do que qualquer que seria redundante. Portanto, é necessário um algoritmo que possa escolher muitos pontos para expressar regiões de alta variância e menos pontos para expressar regiões de relativa homogeneidade (RISI; STANLEY, 2012).

2.3.4.2.2 Algoritmo de escolha de pontos *quadtree*

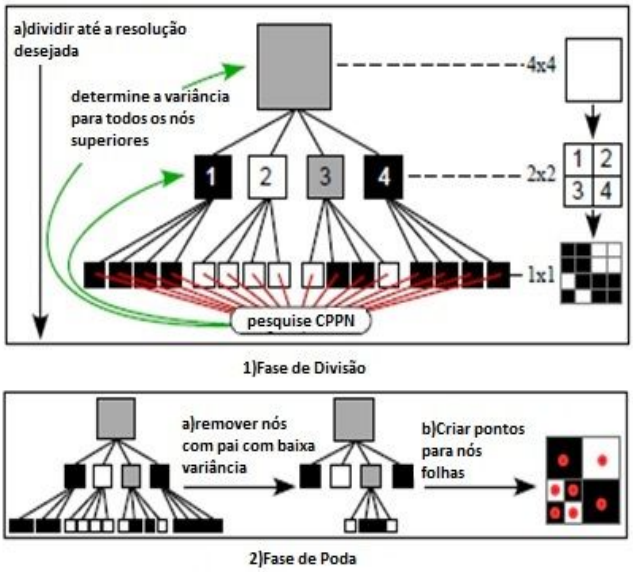
Para executar a tarefa de escolher pontos (os pesos) a expressar, uma estrutura de dados é necessária que aloca espaço a ser representado em níveis variáveis de granularidade. Tal técnica de resolução multidimensional é chamada *quadtree* (FINKEL; BENTLEY, 1974) e tem sido aplicada com sucesso em vários campos que variam de reconhecimento de padrões a codificação de imagem (ROSENFELD, 1980; STROBACH, 1991) e baseia-se em dividir recursivamente uma região bidimensional em quatro sub-regiões. Dessa forma, a decomposição de uma região em quatro novas regiões pode ser representada como uma subárvore cujo pai é a região original com um descendente em cada região decomposta. A divisão recursiva de regiões pode ser repetida até que a resolução desejada seja alcançada ou não seja necessária mais subdivisão.

O algoritmo de escolha de ponto *quadtree* funciona em duas fases (Figura 17): Na fase de divisão **o** *quadtree* é criado recursivamente subdividindo o substrato inicial até uma resolução inicial desejada **r** é atingida (por exemplo, 1x1). Uma vez que esta resolução é alcançada, para cada folha *quadtree*, correspondendo ao quadrado (x1; y1; x2; y2), o CPPN é consultado na posição $\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right)$ e o valor **w** resultante é armazenado. Dados esses valores **(w1 w2,..wk)** para uma subárvore do nó *quadtree* **p** e média $\bar{w}$, a variância do nó quadtree **p** pode ser $\sigma_p^2 = \frac{1}{k}\sum_1^k (\bar{w} - w_i)^2$. Esta variância é um indicador heurístico de heterogeneidade (ou seja, presença de informação) de uma região. Se a variância do pai de uma folha *quadtree* é ainda maior do que uma dada divisão de limiar **dt**, então a fase de divisão pode ser reaplicada para o correspondente quadrado, permitindo densidades cada vez mais

elevadas. Um nível de resolução máximo **rm** pode ser configurado para colocar um limite superior ao número de nós possíveis.

A representação *quadtree* criada na fase de divisão serve como um indicador de variação heurística para decidir sobre a colocação e densidade de pontos a expressar. Porque se mais pontos devem ser expressos mais elevada é a variância de uma região, uma **fase de poda** é executada em seguida (Figura 17), na qual os nós *quadtree* são removidos cujas variâncias dos pais são menores que um limiar de variância σ_p^2**.** Subsequentemente, os pontos são criados para todos os nós folha resultantes. O resultado é uma maior resolução em zonas de maior variação.

Figura 17 – *Quadtree* exemplo de extração de informações para um *CPPN* bidimensional



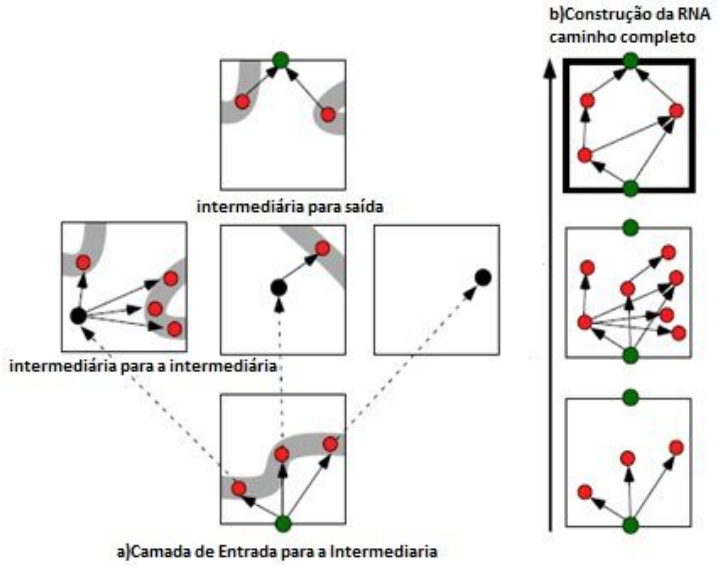
O algoritmo funciona em duas fases principais. (1) Na fase de divisão o *quadtree* é criado recursivamente dividindo cada quadrado em quatro novos quadrados até a resolução desejada seja atingida. Subsequentemente o valor *CPPN* para cada folha e a variância de cada nó superior é determinado. Os nós em cinza na Figura têm uma variância superior a zero. Em seguida, na fase de poda (2), todos os nós *quadtree* são removidos cujo os pais têm uma variância menor do que um determinado limiar (2a). Os pontos são criados para todos *quadtree* resultantes que ficam (2b). Dessa forma, a densidade dos pontos em diferentes regiões irá corresponder à quantidade de informação na referida região.
Fonte: Risi e Stanley (2012).

2.3.4.2.3 Conclusão da rede iterada

Esta seção apresenta como o *ES-HyperNEAT* define a colocação dos neurônios escondidos a partir das entradas e saídas da RNA, que permite restringir a busca a duas dimensões. Esta abordagem concentra a busca no hipercubo para descobrir redes funcionais em que cada nó escondido contribui para a saída da RNA e recebe informação (pelo menos

indiretamente) de, pelo menos, um neurônio de entrada, ignorando as partes do hipercubo que são desconectados. A ideia por trás da realização deste algoritmo é apresentada na Figura 18. Em vez de pesquisar diretamente no espaço do hipercubo quadrimensional (são necessárias quatro dimensões para representar um padrão de conectividade bidimensional), o algoritmo analisa uma sequência de duas dimensões transversais do hipercubo, uma de cada vez, para descobrir quais conexões incluir na RNA. Por exemplo, dado um nó de entrada o enfoque da escolha do ponto por aproximação *quadtree* é aplicado apenas as duas dimensões dos padrões de conectividade de saída a partir do nó único (Figura 18a). Esse processo pode ser aplicado de forma iterativa aos nós escondidos descobertos até que um nível de iteração máximo definido pelo usuário é atingido ou não são descobertas mais informações no hipercubo (Figura 18b). Para amarrar a rede para as saídas, a abordagem escolhe conexões com base em padrões de conectividade de entrada (Figura 18c). Uma vez que todos os neurônios ocultos são descobertos, apenas aqueles que são mantidos tem um caminho para a entrada e um neurônio de saída (Figura 18).

Figura 18 – Conclusão de Rede Iterada



O algoritmo é iniciado iterativamente descobrindo a colocação dos neurônios escondidos a partir das entradas (a) e então amarra a rede para as saídas (c). O padrão bidimensional em (a) representa padrões de conectividade de saída de um nó de entrada único enquanto o padrão em (c) representa o padrão de conectividade de entrada para um nó de saída único. Desta forma, as regiões de alta variância são solicitadas apenas na secção transversal bidimensional do hipercubo contendo a origem ou o nó de destino. O algoritmo pode ser aplicado de forma iterativa à descoberta de nós ocultos (b). Apenas esses nós são mantidos que tem um caminho para um neurônio de entrada e de saída (d). Dessa forma, a busca através do hipercubo está restrita topologias funcionais de RNAs.

Fonte: Risi e Stanley (2012).

2.3.4.2.4 Evoluindo redes plásticas com *ES-HyperNEAT*

Ao contrário das tradicionais RNAs estáticas aplicadas em neuroevolução cujos pesos não mudam durante a sua vida útil, RNAs plásticas podem aprender durante a sua vida, mudando suas forças das conexões sinápticas internas seguindo uma regra de aprendizagem *Hebbiana* que modifica pesos sinápticos baseados em atividades pré e pós-sinápticas dos neurônios. A regra de plasticidade generalizada *Hebbiana* assume a forma: **wji=η.[*Aoj oi+Boj+Coj+D*],** onde **η** é a taxa de aprendizagem, ***oj*** e ***oi*** são os níveis de ativação dos neurônios pré e pós-sinápticos e **A-D** são os termos correlação, termo pré-sináptico **B**, termo pós-sináptico **C** e constante **D** respectivamente. Os estudos de Risi e Stanley (2012) sugerem formas mais elaboradas de aprendizagem necessitam de outros mecanismos além da plasticidade Hebbiana, em especial neuro-modulação.

Numa rede com neuromodulação, outros neurônios podem alterar o grau de plasticidade potencial entre os neurônios pré e pós-sinápticos com base nos seus níveis de ativação. O benefício da adição de neuromodulação é que ela permite que a RNA possa altera o nível de plasticidade em neurônios específicos em momentos específicos.

2.3.4.2.5 Enfoque unificado: *ES-HyperNEAT* Adaptativo

Esta seção introduz a abordagem *ES-HyperNEAT* adaptativo completa que pode evoluir geometrias neurais, densidade e plasticidade juntas. *ES-HyperNEAT* adaptativo aumenta as quatro dimensões do *CPPN* passa a codificar padrões de conectividade com seis saídas adicionais para além da saída de peso usual: taxa de aprendizagem, termo de correlação **A**, termo pré-sináptico **B**, termo pós-sináptico **C**, constante **D** e parâmetro de modulação **M**. O padrão produzido pelo parâmetro **M** codifica conexões modulatórias, como explicado no parágrafo seguinte. Quando o *CPPN* é inicialmente consultado, estes parâmetros são permanentemente armazenados para cada conexão, o que permite que os pesos possam ser modificados durante o tempo de vida da RNA. Além disso, para o seu padrão de ativação, cada neurônio i também calcula a sua ativação modulatória ***mi*** com base nos tipos de suas conexões de entrada:

$$m_i = \sum_{wji \in Mod} wji.oj \qquad (7)$$

O peso de uma conexão entre os neurônios i e j, muda seguindo alterações da regra plasticidade modulada:

$$\Delta w_{ji} = \tanh\left(\frac{m_i}{2}\right).\eta.[Ao_j o_i + Bo_j + Co_j + D] \quad (8)$$

2.3.4.2.6 Experimentos realizados com o *ES-HyperNEAT*

Os experimentos realizados por Risi e Stanley (2011) tiveram por foco duas tarefas: primeira foi o domínio de dupla tarefa e a segunda foi o domínio de navegação em labirinto introduzido na pesquisa de Lehman e Stanley (2008). O *ES-HyperNEAT* resolveu o domínio dupla tarefa em todas as execuções e levou em média 33 gerações (σ = 31). Estes resultados sugerem que essa tarefa se beneficia da capacidade do *ES-HyperNEAT* de gerar redes com pouca conectividade. A complexidade média das soluções *CPPN* descobertas para a primeira tarefa foi de 9.7 nós ocultos (σ=6.7). O *ES-HyperNEAT* significativamente supera todas as variantes de substrato fixo e encontra solução em 95% dos 20 ensaios para o caso da tarefa de navegação em Labirinto. Na pesquisa de Risi e Stanley (2011) conclui-se que quando se dita a localização dos nós escondidos a priori fica mais difícil para o *CPPN* representar o padrão correto.

2.4 OTIMIZAÇÃO DE MORFOLOGIAS NEURAIS PARA RECONHECIMENTO DE PADRÕES

A pesquisa de De Sousa et al. (2012) aborda o problema de como a estrutura dendrítica e outras propriedades morfológicas do neurônio afetam o desempenho do reconhecimento de padrões. O objetivo dessa pesquisa de De Sousa et al. (2014) foi caracterizar as implicações da morfologia neuronal em memórias associativas e no reconhecimento de padrões. É assumido que a memória associativa é baseada na alteração dos pesos sinápticos (plasticidade sináptica). A mudança de plasticidade em uma sinapse pode permanecer por milissegundos até alguns segundos (plasticidade sináptica de curto prazo, mais conhecida como Depressão de Longo-Prazo – DLP), ou por minutos até horas ou mesmo dias (plasticidade de longo prazo, denominada Potenciação de Longo-Prazo – PLP).

Inicialmente a pesquisa foca na DLP das sinapses que ocorre entre as fibras paralelas e células de Purkimje (CPs), a mesma é frequentemente denominada DLP cerebelar atuando no aprendizado motor e por último a PLP que é um tipo de plasticidade sináptica que ocorre quando as células pré-sinápticas e pós-sinápticas são ativadas ao mesmo tempo. Com base nessas suposições a pesquisa investiga a importância da estrutura dendrítica e outras propriedades morfológicas no desempenho do reconhecimento de padrões. Com base nessas, De Sousa et al. (2012) formulou duas regras de aprendizado: a DLP e PLP, as mesmas são variações da regra *hebbiana* formulada por Hebb (1949). Tendo sido implementadas em dois tipos de modelos computacionais, RNAs e modelos comportamentais (onde cada segmento do neurônio é representado por um circuito resistor-capacitor RC, o qual é composto por um resistor e capacitor em paralelo. Esses dois modelos são usados para analisar o efeito da plasticidade sináptica no reconhecimento de padrões.

As técnicas utilizadas para geração de árvores dendrítica com diferentes morfologias foram as seguintes: inicialmente as árvores dendrítica foram produzidas gerando exaustivamente cada morfologia possível, por último as árvores dendrítica foram evoluídas por algoritmos evolucionários. A partir dessas morfologias, construíram-se modelos comportamentais dos neurônios baseados em condutância, após isso avaliaram-se o desempenho dos modelos neuronais resultantes quantificando-se as suas habilidades para discriminar padrões aprendidos e novos, essas morfologias foram testadas na presença ou não de condutâncias. Os resultados mostram que a morfologia tem um efeito considerável no reconhecimento de padrões, além disso, que neurônios com uma pequena profundidade média são os melhores reconhecedores de padrões e que a simetria não se correlaciona com o desempenho. A pesquisa mostra também que o algoritmo evolucionário pode encontrar morfologias efetivas e que a combinação dos parâmetros morfológicos tem um papel fundamental na determinação dos neurônios em tarefas de reconhecimento de padrões.

2.5 CONSIDERAÇÕES FINAIS

No presente capítulo apresentou-se uma revisão histórica de pesquisas voltadas a evolução de RNAs cobrindo o período de 1989 aos dias atuais, bem como um estado da arte sobre o referido tema. Esse capítulo serviu de referencial norteador para definição do escopo dessa tese doutorado, considerando o que já foi publicado em relação ao tema de pesquisa proposto, dando-se ênfase aos últimos quatro anos. Essa etapa foi de suma importância, pois

pode-se identificar quais vertentes ainda não foram pesquisadas e assim pode-se explorá-las, visando preencher determinadas lacunas nessa área de pesquisa e superar entraves teóricos e/ou metodológicos existentes. Nó próximo capítulo apresentam-se modelos artificiais de desenvolvimento biológico focando nos Sistemas-L que nessa pesquisa são usados para formalizar o modelo de desenvolvimento artificial de neurônios proposto no capítulo 6.

CAPÍTULO 3 – MODELOS ARTIFICIAIS DE DESENVOLVIMENTO

"*Até que ponto a matemática será suficiente para descrever e a física para explicar, a fábrica do corpo, ninguém pode prever*"

Prusinkiewicz e Runions (2012)

3.1 INTRODUÇÃO AOS SISTEMAS-L

Nesse capítulo apresentam-se conceitos de Sistemas-L, que consistem de um novo tipo de mecanismo de reescrita de *strings*. A diferença essencial entre a gramática de Chomsky (1956) e os Sistemas-L. consiste no fato de que nas gramáticas de Chomsky as regras de produção são aplicadas sequencialmente, enquanto que nos Sistemas-L são aplicadas em paralelo e simultaneamente substitui todas as letras em uma dada palavra.

Definição 3.1: Um Sistema-L Livre de contexto (Sistema 0L) pode ser definido como uma gramática **G** de uma linguagem **L**, **G={Σ, ∏, α}**, onde Σ é o conjunto finito de símbolos ou alfabeto da linguagem, então Σ^* denota o conjunto de todas as palavras possíveis sobre **Σ**, analogamente Σ^+ representa o conjunto de todas as palavras sobre **Σ,** excetuando-se a palavra vazia, ou seja, $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. O ∏ é o conjunto finito de regras de reescrita (regras de produção) e $\alpha \in \Sigma^*$ é a *string* de ínicio (axioma) de **L**. $\prod = \{\pi | \pi : \Sigma \rightarrow \Sigma^*\}$ é o conjunto de regras de reescrita ou regras de produção. O Sistema-L gera a linguagem a **L(G)**={α} U **σ(α)** U **σ(σ(α))...**= $U_{i \geq 0} \sigma^i(\alpha)$. Onde σ^i representa a derivação ***i***.

3.1.1 Sistemas-L livres de contexto

Esses sistemas são os tipos mais básicos de Sistemas-L. As regras de produção são da forma **Predecessor→Sucessor**, a interpretação é da seguinte forma: o predecessor é substituído pelo sucessor.

Como exemplo ilustrativo da utilização de Sistemas-L livres do contexto, considere a gramática **G={Σ, ∏, α}** com **Σ={A,B,C}, ∏={A→BA, B→CB, C→AC}** e **α=ABC,** considerando-se as regras de produção aplicadas ao axioma, assim, obtém-se a linguagem **L={ABC, BACBAC, CBBAACCBBAAC}**. Aplicando-se a primeira regra do conjunto ∏ ao axioma **ABC**, obtém-se a *string* **BABC** e a segunda regra do conjunto ∏ a essa *string*,

obtém-se **BACBC** e a terceira regra a *string* anterior, obtém-se **BACBAC**, que corresponde ao segundo elemento do conjunto L, o terceiro elemento é obtido por processo análogo, considerando a *string* **BACBAC** como estado inicial.

O exemplo apresentado anteriormente mostra que cada regra de produção é aplicada em paralelo em passos consecutivos de reescrita. Após cada passo, todos os símbolos da cadeia de caracteres original, que sejam idênticos aos do lado esquerdo das regras de produção, são substituídos pelos sucessores das regras, como definido pelo lado direito das mesmas. A **linguagem L** é o conjunto de todas as cadeias que são geradas aplicando-se as regras de produção.

3.1.2 Sistemas-L sensíveis ao contexto

Uma extensão para o Sistema-L apresentado anteriormente é o sensível ao contexto, que é usado para gerar regras de reescrita condicionais. A regra de reescrita $\mathbf{A<B\rightarrow C}$, expressa que **B** deve ser reescrito somente se é precedido por **A**, em geral as regras de reescrita sensíveis ao contexto terão a forma: $\mathbf{L<P>R\rightarrow S}$, com $\mathbf{P \in \Sigma}$ **e** $\mathbf{L,R,S \in \Sigma^{+}}$. **P** é o predecessor, e **S** o sucessor, são o que anteriormente foi chamado de contexto esquerdo e direito respectivamente da regra de reescrita, o que determina se a regra de reescrita será aplicada ao símbolo **P** na *string*. O símbolo **P** será reescrito somente se tiver **L** a sua esquerda e **R** a sua direita. Suponha a seguinte *string* **ABBACAADBAABBAC**, com as seguintes regras de reescrita: $\mathbf{CA<A\rightarrow EG}$, $\mathbf{A<A>B\rightarrow BE}$, $\mathbf{B\rightarrow RT}$. A *string* que resulta após uma iteração é: **ARTRTACAEGDRTABERTRTAC**. Se duas regras de reescrita são aplicadas a certo símbolo, um com e outro sem contexto, o com contexto é usado. A regra de reescrita que é mais específica prevalecerá.

3.1.3 Sistemas-L estocásticos

Definição 3.2: Um Sistema-L estocástico é uma quadrupla ordenada $\mathbf{G_{\pi} = (\Sigma, \Pi, \alpha, \pi)}$. O alfabeto Σ , o axioma α, o conjunto de regras de produção Π, que são definidos da mesma forma que o Sistema-L convencional e função $\boldsymbol{\pi : P\rightarrow(0,1]}$, chamada distribuição de probabilidade, que mapeia o conjunto de regras de produção em um conjunto de probabilidades. É assumido que para toda letra $\mathbf{a} \in \Sigma$, a soma de todas as probabilidades das

regras de produção com o predecessor é igual a 1. Um exemplo desse Sistema-L estocástico é o que dá origem a uma sequência de retângulos com diferentes áreas, designando um retângulo de área **i** por **Xi**, esse sistema pode ser definido por:

Alfabeto: {X1,X2,X3,X4,X5}
Axioma: X3

Regra 1: X1→X1X1
Regra 2 : (com probabilidade 1/3) X2→X1X3
(com probabilidade 1/3) X2→ X3X1
(com probabilidade 1/3) X2 → X2X2
Regra 3 : (com probabilidade 1/5) X3 →X3X3
(com probabilidade 1/5) X3 →X2X4
(com probabilidade 1/5) X3 → X4X2
(com probabilidade 1/5) X3 → X1X5
(com probabilidade 1/5) X3→X5X1
Regra 4: (com probabilidade 1/3) X4 →X3X5
(com probabilidade 1/3) X4 →X5X3
(com probabilidade 1/3) X4 →X4X4
Regra 5: X5 →X5X5

As regras de produção foram definidas de modo que a média da área dos retângulos produzidos seja igual a área do retângulo que lhes deu origem.

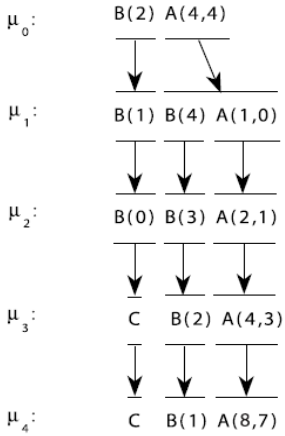
3.1.4 Sistemas-L Paramétrico

Definição 3.3: Um **Sistema-L** paramétrico é definido como uma quadrupla ordenada **G={∑, ∏, α, λ},** onde os elementos **∑, ∏, α,** já foram descritos na Definição 3.1 e **λ** é o conjunto de parâmetros formais. Os símbolos **:** e → são usados para separar os três componentes de uma regra de produção: o predecessor, o sucessor e a condição. Por exemplo, uma regra de produção com predecessor ***A(t),*** condição ***t > 5*** e sucessor ***B(t+1)CD(t-0.5,t-2)*** é escrita como:

$$A(t): t > 5 \rightarrow B(t+1)CD((t-0.5),(t-2)) \quad (9)$$

Um exemplo de um Sistema-L paramétrico é mostrado abaixo, as palavras obtidas após depois das primeiras derivações são mostradas na (Figura 19).

Figura 19 – Sequência inicial de palavras geradas pelo Sistema-L paramétrico especificado na Equação 9



Fonte: Prusinkiewicz e Lindenmayer (2004, p. 43).

α : *B(2)A(4,4)*(3,4)

$p_1 : A(x,y) : y \leq 3 \rightarrow A(x*2, x+y)$
$p_2 : A(x,y) : y > 3 \rightarrow B(x)A(x/y, 0)$
$p_3 : B(x) : x < 1 \rightarrow C$
$p_4 : B(x) : x \geq 1 \rightarrow B(x-1)$

3.2 APLICAÇÕES DE SISTEMAS-L

Nas seções seguintes apresentam-se algumas aplicações de Sistemas-L para desenvolvimento de plantas e neurônios.

3.2.1 Interpretação de cadeias de caracteres pela tartaruga

Uma aplicação interessante de Sistema-L é uma interpretação gráfica de cadeias de caracteres com base na noção de uma tartaruga em movimento. Esta interpretação foi originalmente proposta por Szilard e Quinton (1979). Um estado da tartaruga é definido como uma tripla **(x, y, α)**, onde as coordenadas cartesianas **(x, y)** representam a posição da tartaruga e o ângulo **α** é chamado de posição da tartaruga e é interpretado como a direção em que a tartaruga está se movimentado. Dado o tamanho do passo **d** e **δ** ângulo de incremento, a tartaruga pode responder a comandos pelos seguintes símbolos da Tabela 5.

Tabela 5 – Comandos para o Movimento da Tartaruga

Comando	Ação
F	Avançar um passo de comprimento **d**. O estado da tartaruga muda para **(x',y',α)**, onde **x'=x+d.cos(α)** e **y'=y+d.sen(α)**. Uma linha entre os pontos **(x,y)** e **(x',y')** é desenhada.
F	Avançar um passo de comprimento **d** sem desenhar uma linha.
+	Virar à direita por um ângulo **δ**. O próximo estado da tartaruga é **(x,y, α+δ)**, assume-se aqui que a orientação positiva dos ângulos é no sentido horário.
-	Virar à esquerda por um ângulo **δ** . O próximo estado da tartaruga é **(x,y,α-δ)**.

Fonte: Prusinkiewicz e Lindenmayer (2004, p. 7).

3.2.2 Fractais

A curva de Koch é um fractal que pode ser construído através de um Sistema-L descrito do seguinte modo:

Alfabeto : {F,+,-}
Axioma : F
Regra : $F \rightarrow F+F-\,-F+F$

Onde: 'F' pode ser interpretado como a indicação para desenhar um segmento de reta na direção corrente, **'+'** representa uma rotação no sentido anti-horário e **'-'** uma rotação no sentido horário. Este Sistema-L resulta em:

1ª iteração F + F − −F + F
2ª iteração F + F − −F + F + F + F − −F + F − −F + F − −F + F + F + F − −F + F
3ª iteração F + F − −F + F + F + F − −F + F − −F + F − −F + F + F + F − −F + F
+F + F − −F + F + F + F − −F + F − −F + F − −F + F + F + F − −F+
F − −F + F − −F + F + F + F − −F + F − −F + F − −F + F + F + F−
−F + F + F + F − −F + F + F + F − −F + F − −F + F − −F + F + F + F− − F + F... ....

O resultado das primeiras quatro iterações do processo de construção da curva de Koch, para rotações de 60°, está ilustrado na Figura 21. Na mesma é possível observar que partes da curva de Koch numa determinada iteração tem similaridades. Note-se que a ordem dos símbolos é importante num Sistema-L.

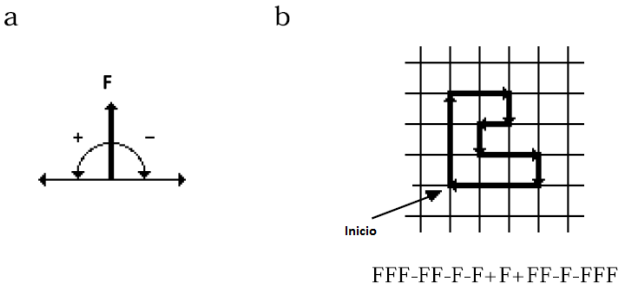
3.2.3 Sistemas-L com memória para desenvolvimento de plantas

Lindenmayer (1968) introduziu uma notação para armazenar estados, com o objetivo de modelar ramificações encontradas em algumas plantas, desde de algas até árvores. Para isso, dois novos símbolos foram adicionados, que são interpretados durante o movimento da tartaruga, secção 3.2.1, da seguinte forma:

- **'['** Armazenar o estado atual em uma pilha, assim como outros atributos como a cor e a largura das linhas desenhadas;
- **']'** Recuperar o estado da pilha e fazer dele o estado corrente. Não é desenhada linha, entretanto em geral a posição muda.

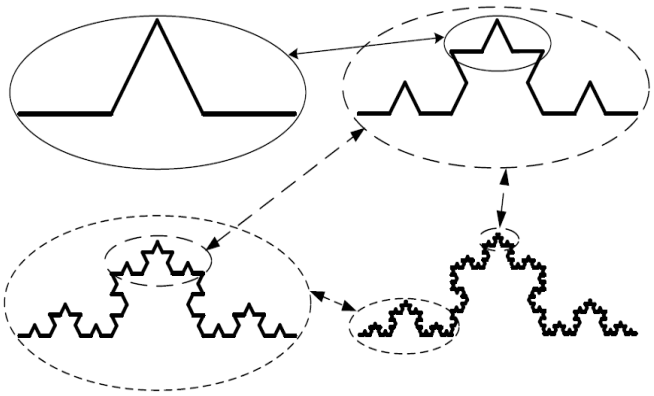
Um exemplo de utilização desse Sistema-L é mostrado na Figura 20.

Figura 20 – Exemplo de utilização do Sistema-L descrito na secção 3.2.1



(a) interpretação dos símbolos F, -, +. (b) A interpretação da cadeia de caracteres FFF+FF+F+F-F-FF+F+FFF, para um ângulo de incremento deδ =90°.
Fonte: Prusinkiewicz; Lindenmayer (2004, p. 7).

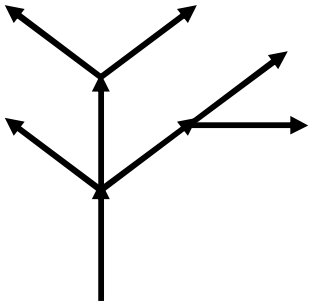
Figura 21 – Exemplo de um sistema-L e das similaridades entre escalas, curva de Koch



Fonte: Ferreira (2005, p. 65).

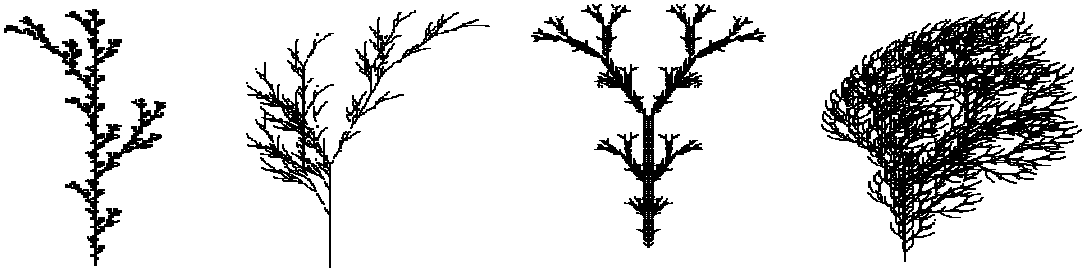
Como exemplo de utilização considere a sequência F[-F][+F[+F]F]F[-F][+F], a interpretação para a sequência é mostrada na Figura 22. Outros exemplos de utilização do Sistema-L com memória para modelamento do crescimento de plantas são mostrados na Figura 23.

Figura 22 – Exemplo de Sistema-L com memória



Fonte: Prusinkiewicz e Lindenmayer (2004, p. 24).

Figura 23 – Sistema-L com memória para modelagem do desenvolvimento de plantas



a)n=5, δ=25.7°,	b)n=5, δ=22.5°	c)n=6, δ=25.7°,	d) n=7, δ=22.5°,
Axioma=F	Axioma=X	Axioma=Y	Axioma=F
F→F[+F]F[-F]F	X→F-[[X]+X]+F[+FX]-X , F→FF	Y→YFX[+Y][-Y] X→X[-FFF][+FFF]FX	F→FF+[+F-F-F]-[-F+F+F]

Fonte: Prusinkiewicz e Lindenmayer (2004, p. 25).

Uma extensão do formalismo de Sistemas-L com memória que considera que o contexto a esquerda pode ser usado para simular sinais de controle que se propagam de maneira acropetal (da raiz em direção ao topo da planta modelada), enquanto que o contexto a direita representa o sinal se propagando de forma basípetal (do topo em direção à raiz da planta). Por exemplo, o Sistema-L seguinte simula a propagação de um sinal na direção acrópeta em uma estrutura com memória que não cresce (Figura 24a). Onde F_b representa o segmento que o sinal já alcançou e F_a o segmento que o sinal ainda não alcançou.

$$\#ignore: \pm$$

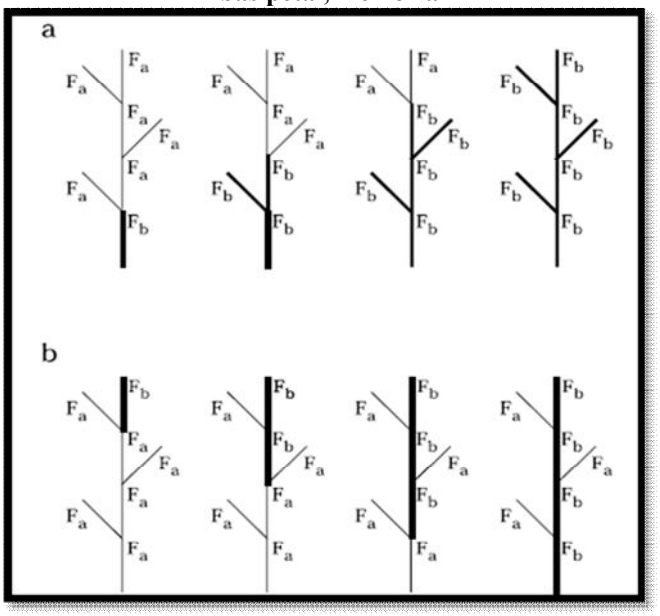
$$\omega: F_b[+F_a]F_a[-F_a]F_a[+F_a]F_a$$
$$p_1: F_b < F_a \rightarrow F_b$$

De maneira análoga um sinal na direção basipetal (Figura 24b) pode ser simulado de forma análoga pelo seguinte Sistema-L.

$$\#ignore: \pm$$

$$\omega: F_b[+F_a]F_a[-F_a]F_a[+F_a]F_a$$
$$p_1: F_a > F_b \rightarrow F_b$$

Figura 24 – Propagação do sinal em Sistemas-L com memória e sensíveis ao contexto: (a) acrópeta, (b) basípetal, memória



Fonte: Prusinkiewic e Lindenmayer (2004, p. 33).

3.3 MODELOS GEOMÉTRICOS DE MORFOGÊNESE

Nas secções seguintes apresentam-se alguns modelos de desenvolvimento artificial de neurônios, que utilizam Sistemas-L.

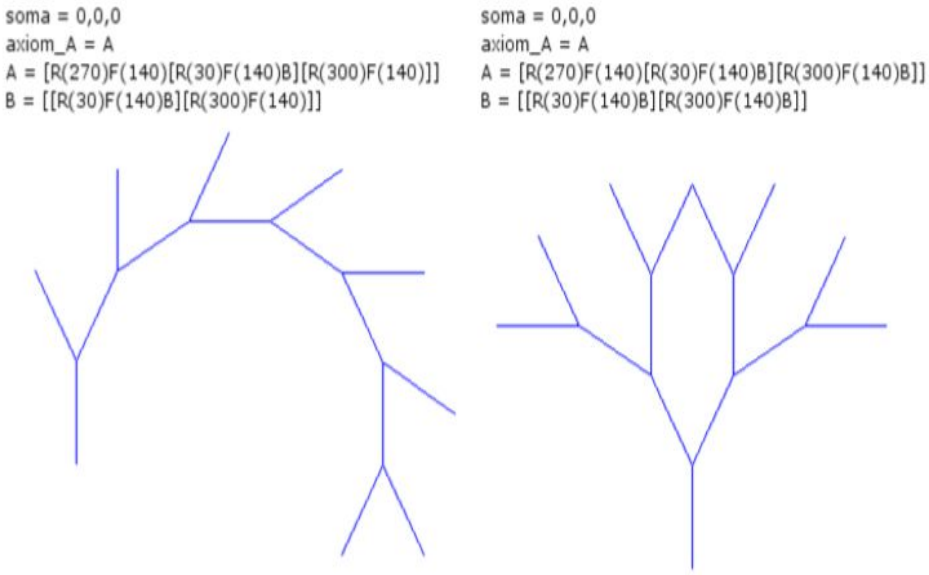
3.3.1 EvOL-Neuron

EvOL-Neuron é uma ferramenta para gerar neurônios virtuais desenvolvido por Torben-Nielsen, Tuyls e Postma (2008). Essa ferramenta usa um Sistema-L para gerar morfologias neurais. Um exemplo é ilustrado na Figura 25. O *EvOL-Neuron* usa um conjunto de regras baseadas em um alfabeto simples. Três símbolos são usados: **F(x)** para mover a frente **x** vezes, **R(x)** para rotacionar **x** graus e **E(x)** para aumentar o ângulo **x** vezes. Em

adição, os símbolos [ ] são usados para determinar quando uma nova ramificação começa e termina, respectivamente. Além disso, o número de ramificações é determinado pelo número de ciclos que a regra é chamada.

De Sousa et al. (2014) menciona que o algoritmo usa um componente (axioma) estocástico, que não permite controle total sobre o processo do processo de geração de morfologias. Além disso, não permite ter controle de árvores válidas, por exemplo não restringe o processo de geração que resulte em árvores dendriticas com um determinado número de ramificações, não sendo possível comparar árvores dendriticas com tamanhos idênticos.

Figura 25 – Exemplos de Morfologias Geradas com o EvOL-Neuron



Esses neurônios foram gerados com as regras do Sistema-L dadas acima. Note que as regras usadas para gerar ambas as morfologias são análogas, apenas com diferenças na última parte : para a árvore simétrica (a direita), a regra B é chamada no final de cada regra para fazer a árvore crescer de forma simétrica. Para gerar a árvore assimétrica (na esquerda), 7 ciclos foram necessários, para a simétrica apenas três.
Fonte: de Sousa et al. (2014).

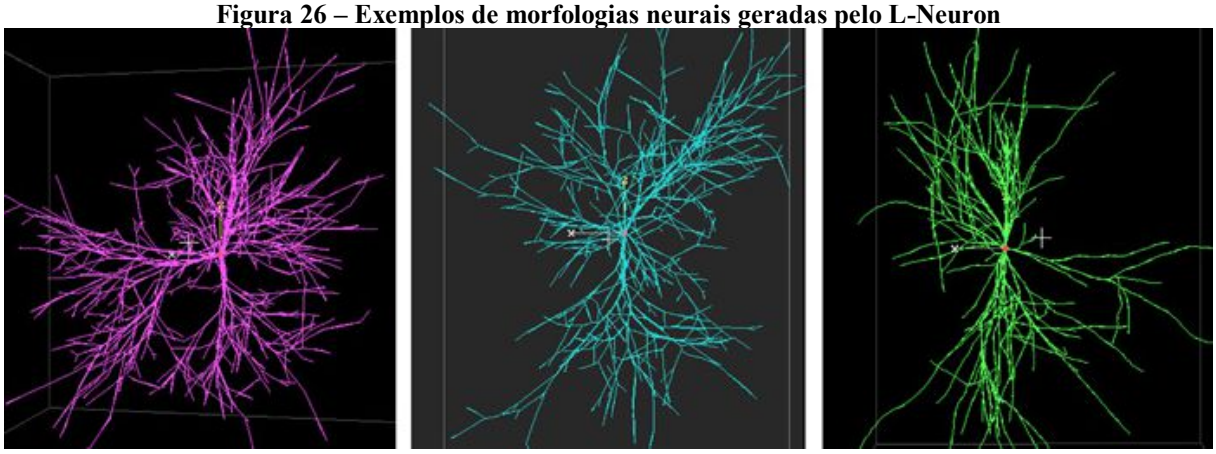
3.3.2 L-Neuron

Esse modelo foi proposto por Ascoli e Krichmar (2000), usou a mesma sintaxe do Sistema-L anterior. Entretanto, as regras de produção (da secção 3.3.1) são substituídas por regras baseadas na pesquisa de Hillman (1979). Essas regras permitem ao modelo descrever o neurônio em termos de parâmetros fundamentais, que são expressos por características dendriticas tais como diâmetro, comprimento, ramificações e ângulo. Exemplos de morfologias são mostradas na Figura 26. O L-Neuron usa um conjunto de parâmetros para gerar classes

morfológicas em vez de por exemplo, um neurônio simples, para gerar variabilidade morfológica entre os vários neurônios em uma rede. Para fazer isso, o modelo lê uma lista de parâmetros estocasticamente, aplicando distribuições uniformes e gaussianas. O resultado é uma lista de múltiplos neurônios diferentes.

Como menciona de Sousa et al. (2014) o uso de parâmetros randômicos para gerar classes morfológicas pode resultar em perda de controle sobre as morfologias geradas. Outra considerável limitação é o número de regras elevadas necessárias para descrever uma classe morfológica.

Figura 26 – Exemplos de morfologias neurais geradas pelo L-Neuron



Fonte de Sousa et al. (2014).

3.4 CONSIDERAÇÕES FINAIS

No presente capítulo foram apresentados conceitos de Sistemas-L e aplicações dos mesmos, dentre elas o desenvolvimento de neurônios artificiais. No próximo capitulo apresentam-se conceitos sobre RNAs e no capítulo 6, secção 6.3.1, apresenta-se um Sistema-L utilizado para o desenvolvimento de arquiteturas de redes neurais diretas e recorrentes.

CAPÍTULO 4 – REDES NEURAIS

"O concreto é a parcela do abstrato que o uso tornou familiar."

Paul Langevin

4.1 INTRODUÇÃO

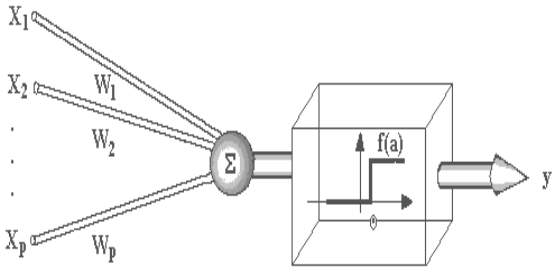
Neste capítulo apresentam-se conceitos sobre RNAs. Adicionalmente, são discutidos alguns mecanismos de aprendizado de RNAs, bem como arquiteturas padrões usadas em tarefas de reconhecimento de padrões e finalmente destacam-se alguns aspectos relativos a questão do projeto de RNAs.

4.2 CONCEITOS SOBRE REDES NEURAIS ARTIFICIAIS

As RNAs foram inspiradas na estrutura e comportamento dos neurônios biológicos existentes no cérebro humano. Cada neurônio está conectado com alguns outros, formando uma rede que realiza um processamento contínuo e paralelo. Em 1943, McCulloch e Pitts propuseram o primeiro modelo de neurônio artificial. No mesmo, os dendritos são representados por entradas, cujas ligações com o corpo celular artificial são realizadas através dos pesos das conexões entre os neurônios (simulando as sinapses). Os estímulos são processados pela função soma e o limiar (*threshold*) de disparo do neurônio biológico é representado por uma função de ativação. A Figura 27 ilustra o esquema de neurônio artificial de McCullock e Pitts, assim como a equação 10 mostra como a sáida do neurônio é calculada, onde:

- X_i representa as p entradas;
- W_i representa os p pesos associados às entradas. Basicamente as entradas são multiplicadas por seus respectivos pesos, podendo gerar sinais positivos (excitatórios) ou negativos (inibitórios);
- Σ representa o combinador linear, que executa o somatório dos sinais calculados pelos produtos das entradas por seus pesos;
- $f(a)$ representa a função de ativação, neste caso, a função linear;
- y representa a saída calculada.

Figura 27 – Esquema de neurônio artificial de McCullock e Pitts.



Fonte: Braga, de Carvalho e Ludemir (2007, p. 5).

$$y = f\left(\sum_{i=0}^{p} x_i * w_i\right) \tag{10}$$

4.2.1 Tipos de redes neurais

A arquitetura de uma RNA é determinada pela sua estrutura topologica, i.e, pela conectividade global e funções de transferência de cada nó da RNA. Existem diferentes tipos de arquiteturas de redes neurais, alguns exemplos comuns são:

a) **redes *perceptron* multicamada (RPMC)**: que consiste em uma rede totalmente conectada diretamente alimentada com uma camada de entrada de neurônios, uma ou mais camadas escondidas e uma camada de saída. O valor de saída é obtido através da sequência de funções de ativação definido na camada escondida. Normalmente, neste tipo de rede o processo de aprendizagem supervisionado é o AR, que usa o gradiente descendente como método de busca no espaço de pesos para minimizar o erro entre a saída obtida e a atual;
b) **redes perceptron com camada única**: esta é um tipo particular de rede RPMC com uma única camada;
c) **redes de Hopfield**: são redes autoassociativas que agem como uma memória e podem recuperar um padrão armazenado, mesmo quando a entrada é uma versão ruidosa do mesmo. Essa rede tem uma topologia com camada única totalmente conectada. Depois que um padrão de entrada é apresentado, a rede irá convergir por meio de uma regra de atualização de estados para um padrão estável. Uma rede de Hopfield não tem unidades de entrada, uma vez que um vetor de entrada

simplesmente define a ativação inicial de cada unidade. Este valor de ativação será enviado para outras unidades e em qualquer instante do tempo, o estado da rede é o vetor de todos os estados da unidade;

d) **mapas auto-organizáveis**: os mapas auto-organizáveis de Kohonen fazem parte de um grupo de redes neurais chamado redes baseadas em modelos de competição, ou simplesmente redes competitivas. Estas redes combinam competição com uma forma de aprendizagem para fazer os ajustes de seus pesos. Outra característica importante deste tipo de rede é que elas utilizam treinamento não supervisionado, onde a rede busca encontrar similaridades baseando-se apenas nos padrões de entrada. O principal objetivo dos mapas auto-organizáveis de Kohonen é agrupar os dados de entrada que são semelhantes entre si formando classes ou agrupamentos denominados *clusters*. Durante o processo de auto-organização do mapa, a unidade do *cluster* cujo vetor de pesos mais se aproxima do vetor dos padrões de entrada é escolhida como sendo a "vencedora". Essa unidade e suas unidades vizinhas têm seus pesos atualizados segundo uma regra específica.

e) **redes recorrentes**: uma rede recorrente pode ter conexões que voltem de nós de saída aos nós de entrada e, na verdade, podem ter conexões arbitrárias entre quaisquer nós. Deste modo, o estado interno de uma rede recorrente pode ser alterado conforme conjuntos de entradas sejam apresentado a ela e, assim, pode-se dizer que ela tem uma memória. Isto é particularmente útil na solução de problemas que não dependam somente das entradas atuais, mas de todas as anteriores. Ao aprender, a rede recorrente alimenta suas entradas por meio da rede, incluindo alimentação de dados de volta, das saídas às entradas.

4.2.2 Aprendizado em RNAs

A natureza adptativa do aprendizado por exemplos de RNAs é uma característica muito importante desses paradigmas computacionais. O processo de treinamento modifica os pesos das RNAs com a finalidade de melhorar um determinado critério de desempenho, que corresponde a uma função aptidão, ao longo do tempo. Esse proceso é normalmente denominado regra de aprendizado que pode ser classificado em:

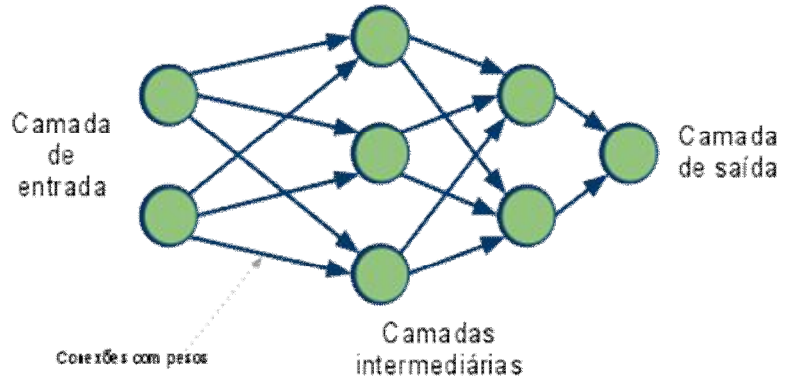
a) **Aprendizado supervisionado**: este tipo de treinamento, utiliza em sua estrutura uma espécie de instrutor que confere o quanto a rede está próxima de uma solução aceitável. No treinamento supervisionado, um conjunto de dados é apresentado como entrada para a rede e, associado a este conjunto de entrada, é apresentado um conjunto de saída. O algoritmo de aprendizado busca ajustar os pesos das conexões entre os neurônios de tal forma que, para o conjunto de entrada informado, a rede seja capaz de calcular uma saída o mais próximo possível da saída informada. Os algoritmos apresentados nas secções 4.2.3 e 4.2.4 são supervisionados.

b) **Aprendizado não supervisionado**: conhecido também como aprendizado auto supervisionado por não possuir instrutores. Este tipo de treinamento utiliza apenas as informações das entradas e realiza o aprendizado através de agrupamentos e conceitos de vizinhança, sendo muito utilizado para encontrar padrões entre os dados.

4.2.3 Rede perceptron multicamada (RPMC)

A rede RPMC representa uma generalização do *perceptron* proposto por Rosenblatt (1958). Sua arquitetura, ilustrada na Figura 28, consiste em várias camadas compostas por nós (neurônios) onde cada neurônio de uma camada possui ligação com todos os neurônios da camada seguinte.

Figura 28 – Arquitetura de uma rede RPMC

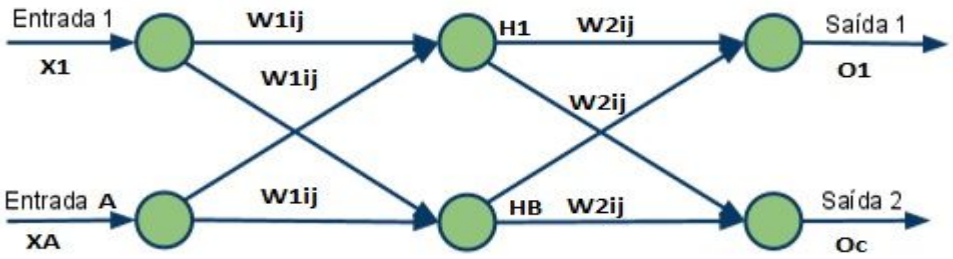


Fonte: adaptado de Rosenblatt (1958).

Conforme visto na Figura 28, a rede RPMC possui uma camada de entrada, que atua como os sensores da rede captando os estímulos do ambiente e pode ter uma ou mais camadas

intermediárias que é onde a maior parte do processamento é realizada através das conexões e seus pesos respectivos, podem ser considerados como extratoras de características e uma camada de saída onde o resultado final é concluído e apresentado. Ainda na Figura 28 não existem conexões entre a saída de um neurônio e algum outro neurônio localizado em uma camada anterior ao primeiro, ou seja, não possui ciclos, fato que caracteriza uma rede *feedforward*. O proceso de treinamento de redes RPMC utilizando o AR Rumelhart e McLelland (1986), conhecido como regra delta generalizada, é comumente realizado mediante as aplicações sucessivas de duas fases bem especificas. A formulação matemática do AR e o detalhamento do aprendizado para esse tipo de arquitetura são apresentados no ANEXO A. A arquitetura da RPMC, ilustrada na Figura 29, foi utilizada nas deduções apresentadas no mesmo. Um artificio utilizado para que as RPMCs realizem processamento temporal, envolve o uso de janelas de tempo, em que a RNA passa a utilizar trechos de dados temporais, como se eles formassem um padrão estático, entretanto essa não é a melhor solução indicada para simulação desse tipo de processamento. Para que a RNA seja considerada dinâmica é necessário que ela possua menória.

Figura 29 – Arquitetura da rede RPMC utilizada no ANEXO A



Fonte: Braga, de Carvalho e Ludemir (2007, p. 35).

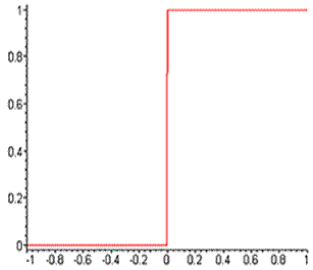
4.2.3.1 Função de ativação

As principais funções de ativação mais conhecidas são:

a) **função limiar (Degrau):** nesta função, representada pela equação 11, a saída é binária. A mesma é utilizada no modelo de McCulloch e Pits. A Figura 30 ilustra o comportamento desta função:

$$f(a) = \begin{cases} 1, se\ a \geq 0 \\ 0, se\ a \leq 0 \end{cases} \quad (11)$$

Figura 30 – Gráfico da função limiar

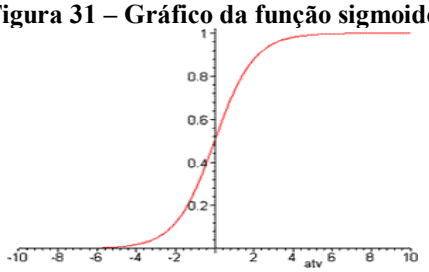


Fonte: o autor (2016).

b) **função sigmóide**: esta função, formalizada pela equação 12 com comportaento ilustrado na Figura 31, pode assumir valores entre 0 e 1, onde $v \in \mathbb{R}$ e β é uma constante real associada ao nível de inclinação da função sigmoide frente ao seu ponto de inflexão. A função sigmoide, além de ser totalmente diferenciável em todo o seu dominio, possui uma região semilinear, que é importante para aproxiamção de funções contínuas. Dependendo do tipo de problema a ser abordado, neurônios da camada de saída com funções de ativação lineares, como a apresentada na secção 4.2.3.1(d) , podem também ser utilizados Braga, de Carvalho e Ludemir (2007, p. 9). Por exemplo quando se utiliza uma RPMC como aproxiamdor de funções, pelo teorema Kolmogorov, a função sigmoide, pode ser utilizada nas camadas escondidas em conjunto com uma linear na saída afim de fazer combinações lineares e aproximar qualquer função. Entretanto, as funções sigmoides e tangente hiperbólica são também comumente utilizadas em problemas de classificação Ahmadizar et al. (2015) e predição de séries temporais Donate, Sanchez e De Miguel (2012).

$$f(a) = sgm(a) = \frac{1}{(1 + e^{-\beta a})} \quad (12)$$

Figura 31 – Gráfico da função sigmoide

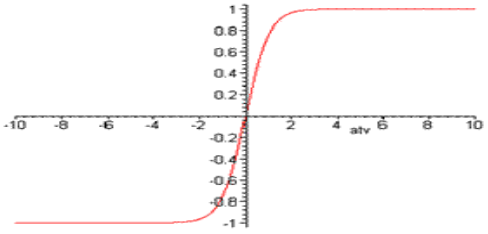


Fonte: o autor (2016).

c) **função tangente hiperbólica:** esta função, formalizada pela equação 13 com comportamento ilustrado na Figura 32, pode assumir valores v entre -1 e 1, onde $v \in \mathbb{R}$:

$$f(a) = \tanh(a) = \frac{1-e^{-a}}{1+e^{-a}} \tag{13}$$

Figura 32 – Gráfico da função tangente hiperbólica



Fonte: o autor (2016).

d) **função Linear –** A função linear ou função identidade produz resultados de saída idênticos aos valores do potencial de ativação **{u},** tendo sua expressão matemática definida por **g(u)=u** . Uma das aplicabilidades dessa função é na utilização das RNAs como aproximadores universais de funções, visando mapear variáveis de entrada e saída de processos.

4.2.3.2 Versões aperfeiçoadas do AR

Algumas variações do AR tem sido propostas com o objetivo de tornar o processo de convergencia mais eficiente, a seguir apresentam-se algumas.

4.2.3.2.1 Método de inserção do termo de momentum

A inserção do termo de momentum se configura como uma das variações mais simples de ser efetuada no AR, pois, basta inserir um único parâmetro visando ponderar o quão as matrizes sinápticas foram alteradas entre duas iterações anteriores e sucessivas. Formalmente, considerando os neurônios pertencentes à L-ésima camada, tem-se:

$$w_{ji}^{(L)}(t+1) = w_{ji}^{(L)}(t) + \alpha\left(w_{ji}^{(L)}(t) - w_{ji}^{(L)}(t-1)\right) + \eta\delta_j^{(L)}.Y_i^{(L-1)} \quad (14)$$

Onde: α é definida como taxa de *momentum*, valor compreendido entre 0 e 1 e η é a taxa de aprendizado. Assim, por intermédio do termo *momentum*, o processo de convergência da rede se torna bem mais eficiente, pois se leva em consideração o critério de quão afastada está a solução atual da final. O uso do temo momentum implica acelerar a convergência da rede à razão de $\eta/(1-\alpha)$. Normalmente valores compreendidos entre ($0{,}05 \leq \eta \leq 0{,}75$) e ($0 \leq \alpha \leq 0{,}9$) são recomendados para treinamento de RPMC (RUMELHART; MCLELLAND, 1986). Nas simulações de problemas de classificação, descritas no capítulo 7, utilizou-se o AR com a inserção do termo *momentum*.

4.2.3.2.2 Algoritmo *QuickPropagation* (AQ)

O AQ foi proposto por Fahlman (1988). As Hipóteses são as seguintes : A curva do erro em relação aos pesos da rede, para cada um dos pesos, pode ser aproximada por uma parábola, com concavidade voltada para cima. A alteração na inclinação dessa curva de erro depende somente do peso em questão, não sendo influenciada pela alteração dos demais pesos da rede. A equação de modificação dos pesos é dada por:

$$\Delta_w^{(t)} = \frac{S^{(t)}}{S^{(t-1)} - S^{(t)}}\Delta_w^{(t-1)} \quad (15)$$

Onde: **S(t)** e **S(t − 1)** são os valores atual e anterior de $\frac{\partial E}{\partial w}$.

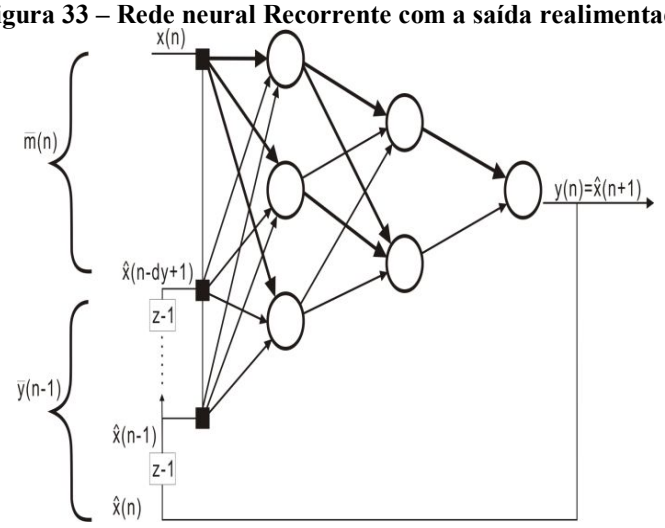
4.2.4 Redes neural recorrente com saída realimentada a entrada

A recorrência e/ou a consideração de valores passados de entrada/saída aumentam a capacidade de aprendizagem das RNAs, assim como, possibilitam capturar a dinâmica do sistema. Assim, as RNRs, são ferramentas eficientes para a simulação de sistemas dinâmicos, cujo comportamento é o considerado variante no tempo ou dependentes dele. Como exemplos de aplicação, tem-se a previsão de valores fututos para ações do mercado financeiro frente a um horizonte semanal, ou então, a predição do consumo de energia elétrica para os próximos meses. As saídas dos sistemas dinamicos, assumindo um instante de tempo qualquer ,

dependem dos seus valores anteriores de saída e entrada. As realimentações envolvem a utilização de ramos particulares compostos por atrasos unitários denotados por z^{-1}, o que resulta em um comportamento dinâmico não-linear em virtude da natureza não-linear dos próprios neurônios. Esta característica de dinâmica não-linear da estrutura permite que este tipo de rede neural seja utilizado em problemas e aplicações que necessitem representar estados. A Figura 33 ilustra uma topologia recorrente que foi utilizada em nossas simulações para problemas dinâmicos.

O algoritmo de treinamento utilizado utilizado para a atualização dos pesos da RNR mostrada na Figura 33, considerando os atrazos no tempo, foi *Backpropagation Through Time-(BPTT)* (DONG; KAISHENG; YU, 2015). A formulação matemática do mesmo é descrita a seguir, utilizando a seguinte notação:

Figura 33 – Rede neural Recorrente com a saída realimentada



Fonte: adaptado de Guo (2013).

Denota-se os níveis de ativação das unidades da camada de entrada por *xj*, da camada oculta por *hj* e da camada de saída por *oj*. A matriz de pesos da camada de entrada para a intermediária é denotada por ***V***, enquanto que a da camada oculta para a de saída é denotada por ***W***. A Camada de entrada consiste em dois componentes, **x(t)** e a ativação anterior da camada oculta **o(t -1)**. A matriz de pesos correspondentes é dada por ***U***. Onde o erro na camada de saída é denotado por:

$$e_o^{(t)} = o_j^{(t)} - y_j^{(t)} \tag{16}$$

A matriz de atualização de pesos da camada de saída é atualizada de acordo com a equação 17:

$$W_{(t+1)} = W_{(t)} + \eta h_{(t)} e_{0(t)}^{T} \tag{17}$$

Em seguida, os gradientes de erros são propagadas da camada de saída para a camada oculta:

$$e_{h}^{(t)} = d_{h}\left(e_{0(t)}^{T} V, t\right) \tag{18}$$

Em que o vetor de erro é obtido utilizando a função de $d_h(\cdot)$

$$d_{h,t}(x,t) = xf'(net_j) \tag{19}$$

A matriz de pesos **V** entre a camada de entrada e camada oculta são o atualizado conforme:

$$V_{(t+1)} = V_{(t)} + \eta x_{(t)} e_{n(t)}^{T} \tag{20}$$

A matriz de pesos recorrentes são atualizados por:

$$U_{(t+1)} = U_{(t)} + \eta o_{(t-1)} e_{n(t)}^{T} \tag{21}$$

Para o AR com atraso no tempo, a propagação do erro é feita de forma recursiva. As equações que descrevem as atualizações de pesos recursivamente são apresentadas no Anexo B.

Como os ANEs que evoluem RNRs são raros na literatura (BEER; GALLAGHER, 1992; HORNBY; POLLACK, 2002), os mesmos constituem-se uma lacuna a ser explorada nesse ramo de pesquisa. Em razão disso, utilizou-se a arquitetura de RNR descrita nessa secção para a simulação de problemas de PST descritos no capítulo 7.

4.2.5 Considerações sobre projetos de RNAs

Para muitas aplicações, as RNAs tem se mostrado convincentes como ferramentas eficientes na solução de problemas. No entanto, o tarefa de sintonizar as suas arquiteturas para atingir um desempenho próximo de ótimo continua a ser um desafio. Existem, um número de parâmetros a serem definidos e ajustados e que podem afetar quão fácil a solução é encotrada. Alguns destes parâmetros são o tipo de RNA que deve ser escolhida, o número de camadas e

neurõnios e os pesos das conexões que definem a sua arquitetura. Os dados de treinamento são também importantantes, ou seja uma boa dose de atenção deve ser dada aos dados de teste para ter certeza que a RNA irá generalizar correntamente com os dados que não foram utilizados durante o treinamento.

4.2.5.1 Arquitetura de RNA

O projeto da arquitetura de uma RNA é crucial para o sucesso de uma aplicação, pois tem impacto significante na capacidade processamento de informação de uma RNA. O problema principal é que não existe uma maneira sistemática automática para projetar uma arquitetura próxima da ótima.

O projeto de uma arquitetura ótima de RNA deve ser tratado como um problema de busca no espaço de arquiteturas, onde cada ponto representa uma possivel arquitetura. Dados alguns critérios (ótimos) de desempenho, isto é, menor erro de treinamento, aquitetura econômica etc. O nível de desempenho de todas as arquiteturas formam uma superficie discreta no espaço de soluções. O projeto de arquiteturas de RNAs é equivalente a encontrar o ponto mais alto na superficie. Algumas características dessa superfície de busca são sumarizadas a seguir:

a) a superfície de busca é infinitamente grande uma vez que o número de neurônios e conexões possíveis são ilimitados;
b) a mesma é complexa e ruidosa desde que mapeamento de uma arquitetura para o seu desempenho é indireto;
c) a superfície é enganadora visto que arquiteturas semelhantes podem ter um desempenho bastante diferente;
d) a superfície é multimodal já que diferentes arquiteturas podem ter desempenho similar;
e) a superfície é não diferenciável desde que alterações no número de neurônios e conexões são discretos e podem ter efeitos sobre a desempenho da estratatégia evolucionária da RNA.

4.2.5.2 Seleção de dados

Um dos mais importantes fatores do treinamento de RNAs é a disponibilidade e integridade dos dados. Eles devem representar todos os estados possiveis do problema considerado e eles devem possuir padrões suficientes para construir os conjuntos de treino e testes. Os dados selecionados para o treinamento devem ser representativos do espaço completo que uma classe pode ocupar. A consistência de todos os dados tem de ser garantida, uma vez que é comum encontrar erros em um banco de dados grande. Além disso, o aumento indiscriminado de neurônios assim como o incremento de camadas intermediárias não asseguram a generalização apropriada em relação às amostras pertencentes aos subconjuntos de teste. Invarialmente, tais ações prematuras tendem a levar a saída da RNA a circustancia de memorização excessiva (*overfitting*), em que esta acaba decorando as suas respostas frente aos estimulos introduzidos em suas entradas. Nessas condições, o erro quadrático durante a fase de aprendizado tende a ser bem baixo, contudo, durante a fase de generalização frente aos subconjuntos de teste, o erro quadrático tende a assumir valores elevados. Esse é um importante parâmetro para medir a generalização da RNA. A Generalização representa a habilidade de um sistema de aprendizado em mapear corretamente novas entradas, não usadas durante o treinamento em saídas corretas, boa generalização dependem do conjunto de treinamento e da Arquitetura da RNA.

4.3 CONSIDERAÇÕES FINAIS

Neste capítulo apresentou-se a fundamentação teórica sobre RNA destacando-se as questões relativas ao projeto das mesmas, no capítulo seguinte apresentam-se os conceitos de evolução biologica, biologia molecular e computação evolucionária.

CAPITULO 5 – EVOLUÇÃO BIOLÓGICA E COMPUTAÇÃO EVOLUCIONÁRIA

"Não é o mais forte que sobrevive, nem o mais inteligente, mas o que melhor se adapta às mudanças."

Charles Darwin

5.1 INTRODUÇÃO

Nesse capítulo, apresentam-se os conceitos sobre Evolução Biológica, Biologia Molecular e Computação Evolucionária que nortearam o desenvolvimento da metodologia biologicamente inspirada para projeto automático de RNAs introduzida no capítulo 6.

5.2 EVOLUÇÃO E SELEÇÃO NATURAL

Em um sentido mais amplo, evolução é meramente mudança, a evolução biológica é a mudança nas propriedades de organismos que transcendem o período de vida de um único indivíduo (MEYER; EL-HANI, 2005). Aquelas que são consideradas evolutivas são aquelas herdadas via material genético. Sendo assim, a natureza deve possuir também mecanismos pelo qual as criaturas mais aptas sejam capazes de transmitir a seus descendentes as características genéticas exibidas por eles por meio do código genético.

DAWKINS (2004, p. 42) menciona que:

> A seleção natural é o relojoeiro cego, porque prevê, não planeja consequências, não tem propósito em vista. Mas os resultados vivos da seleção natural nos deixam pasmos porque parecem ter sidos estruturados por um relojoeiro magistral, dando uma ideia de desígnio e planejamento.

A seleção natural mudou profundamente a maneira de compreender a natureza. Por meio de seus mecanismos ela é capaz de produzir mudanças nas características de populações de espécies. Entretanto, a ideia de que o mais apto sobrevive e direciona o processo evolutivo nem sempre é válido, o que se pode observar na natureza são evidencias diretas da seleção natural em ação, entretanto a mesma não explica tudo (MEYER; EL-HANI, 2005).

5.3 FUNDAMENTOS DE BIOLOGIA MOLECULAR

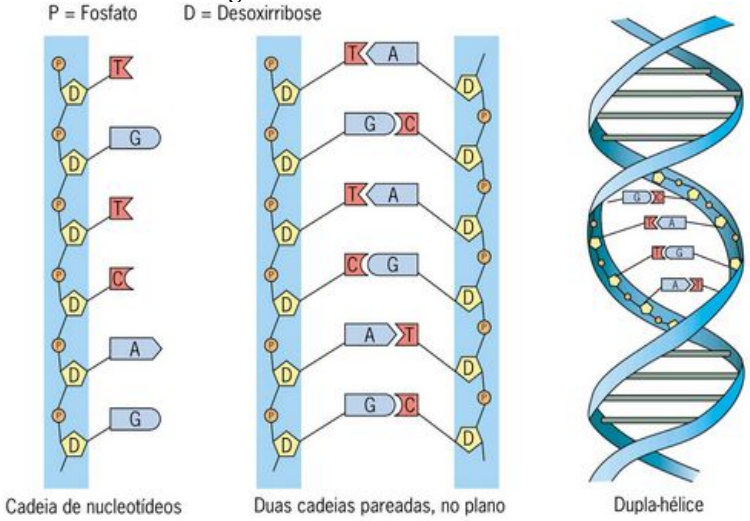
Apesar da complexidade da célula, parece haver alguns princípios de organização, que são conservados em todos os organismos. Toda vida no planeta depende de três tipos de moléculas ADN, ácido ribonucleico (ARN) e proteínas. Pode-se dizer que o ADN carrega uma vasta biblioteca que descreve como a célula funciona, o ARN age para transferir certas peças curtas dessa biblioteca para diferentes lugares na célula, no qual esses pequenos volumes de informação são usados como modelos para sintetizar proteínas (JONES; PEVZNER, 2004).

O ADN foi descoberto em 1869 por Johann Friedrich Miescher. Em 1900 era sabido que o ADN era uma molécula longa consistindo de quatro tipos de bases: adenina (A), timina (T), guanina (G) e citosina (C). Originalmente, os biólogos descobriram cinco tipos de bases, sendo o quinto uracila (U), que é quimicamente semelhante à timina. Por volta de 1920, os ácidos nucléicos foram agrupados em duas classes chamadas ADN e ARN, que diferem ligeiramente na sua nas suas composições de base: ADN usa T enquanto ARN usa U. A parte que guarda a informação importante sobre os programas para a síntese de proteínas são as bases nitrogenadas.

5.3.1 Qual a estrutura do ADN?

A era moderna do ADN começou em 1953, quando James Watson e Francis Crick, determinaram a estrutura helicoidal dupla de uma molécula de ADN. Em 1951, Maurice Wilkins e Rosalind Franklin obtiveram imagens nítidas de raios X que sugeriram que o ADN é uma molécula helicoidal. Milhares de nucleotídeos empilham-se em forma linear para constituir uma cadeia de molécula de ADN. O grupo fosfato liga o carbono 3´do açúcar de um nucleotídeo ao carbono 5´ do açúcar do nucleotídeo seguinte, por meio de uma ligação química. Sendo este constituído de duas cadeias de nucleotídeos que se enrolam originando uma estrutura de *dupla-hélice,* como mostrado na Figura 34. É interessante notar que essas duas cadeias se dispõem de modo antiparalelo, ou seja, enquanto uma vai da direção 5´para 3´ a outra ocorre na direção oposta. Ambas as cadeias são unidas por *pontes de hidrogênio.*

Figura 34 – Estrutura do ADN



Fonte: adaptado de Albert et al. (2002).

5.3.2 Quem carrega a informação entre o ADN e Proteína?

Em 1950 Paul Zamecnik descobriu que a síntese de proteínas no citoplasma acontece com a ajuda de certas moléculas grandes chamadas ribossomos que contêm ARN. Isto levou a suspeita que o ARN pode ser o agente intermediário entre o ADN e a proteína. Assim, o ADN serve como um molde usado para copiar um determinado gene em um ARN mensageiro (ARNm) que transporta informação genética do gene para ribossomo para fazer uma determinada proteína. Existem então duas etapas a primeira é a transcrição do ADN no ARN e em seguida a translação do ARN em proteínas o Quadro 2 mostra essas etapas. Assim o ADN serve como um molde usado para copiar um gene particular em um ARN mensageiro (ARNm) que carrega a informação genética do gene para o ribossomo fazer uma proteína particular.

Quadro 2 – A transcrição do ADN em ARN e a tradução do ARN em proteína

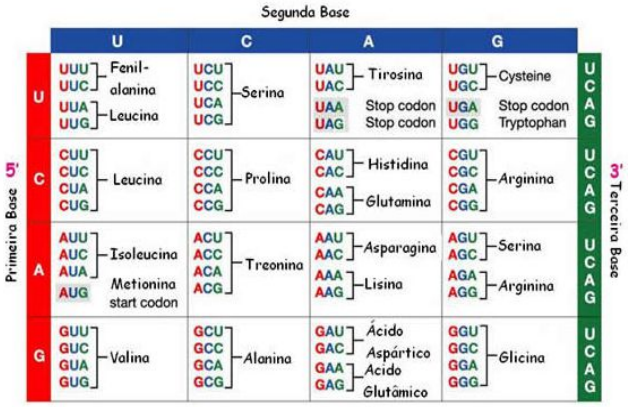
DNA	TAC	CGC	GGC	TAT	TAC	TGC	CAG	GAA	GGA	ACT
RNA	AUG	GCG	CCG	AUA	AUG	ACG	GUC	CUU	CCU	UGA
Proteína	Met	Ala	Pro	Ile	Met	Thr	Val	Leu	Pro	Stop

Cada aminoácido é representado com três letras, por exemplo, Met significa o aminoácido metionina.
Fonte: Jones e Pevzner (2004, p. 65).

Para revelar o código responsável pela transformação do ADN em proteínas ele conjecturou que triplas de letras consecutivas no ADN (chamadas códons) são responsáveis pela sequência de aminoácidos numa proteína. Existem $4^3 = 64$ códons diferentes, que é mais do que três vezes tão grande como o número de aminoácidos. Para explicar esta redundância

biólogos conjecturaram que o código genético responsável pela transformação do ADN em proteína é degenerado: diferentes triplas de nucleotídeos podem codificar para o mesmo aminoácido. Biólogos buscaram descobrir quais triplas codificam que aminoácidos e, por fim, dos anos 1960 descobriram o código genético (Figura 35). A regra das triplas foi confirmada e, portanto, é hoje aceita como de fato (JONES; PEVZNER, 2004).

Figura 35 – Código Genético a partir da perspectiva do ARNm.



O códon para a metionina, ou AUG, também age como um "códon" de início da transcrição. Esse códon é traduzido como ilustrado no Quadro 2.
Fonte: Jones e Pevzner (2004, p. 66).

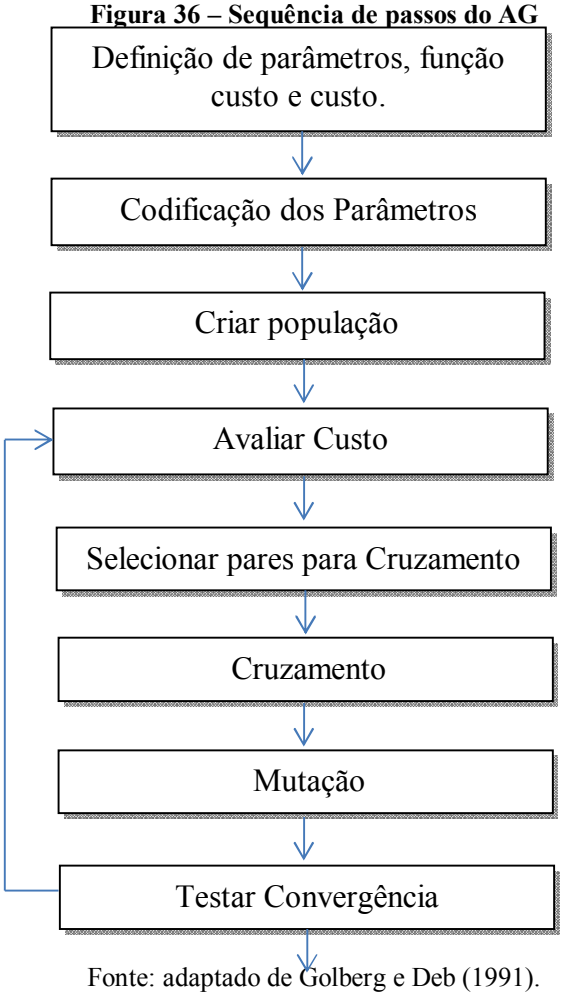
5.4 ALGORITMOS GENÉTICOS

O Algoritmo Genético (AG) pertence à classe dos métodos de otimização natural. O mesmo é um subconjunto dos AE, que modela processos biológicos relacionados à evolução e que utiliza conceitos provenientes do princípio de seleção natural para otimizar funções de custo complexas. Um AG permite que uma população composta de muitos indivíduos evolua sob regras de seleção especificadas para um estado que maximiza a aptidão (i.e a função custo). O método foi desenvolvido por John Holland (1975) ao longo dos anos 1960 e 1970 e foi popularizado por um dos seus alunos, David Golberg (1989) que foi capaz de resolver um problema difícil envolvendo o controle de transmissão em gasodutos. Atualmente o AG é muito empregado na resolução de diversos problemas dentre eles: bioinformática, descoberta de regiões regulatórias (FOGEL et al., 2004), sistema de Reconhecimento de Face (PARSI; SALEHI; DOOSTMOHAMMADI, 2012).

O AG inicia, igualmente a outros algoritmos de otimização, definindo os parâmetros de otimização, a função custo e o custo e termina da mesma forma testando a convergência, entretanto o AG é bem diferente desses algoritmos. Os AG's têm encontrado grande aplicação

em problemas de busca e otimização, mesmo em espaços de busca de pouco entendimento, de dimensão e complexidade elevados, onde métodos de busca clássicos são inapropriados. No caso específico do projeto automático de RNAs, discutido na secção 4.2.5.1, que é tratado com um problema de busca local em um espaço de dimensão infinitamente grande, os AG's se constituem como uma ferramenta importante para solucioná-lo. Uma sequência de passos de um AG é mostrada na Figura 36.

Figura 36 – Sequência de passos do AG



Fonte: adaptado de Golberg e Deb (1991).

5.4.1 Inspiração biológica dos algoritmos genéticos

Um AG genético tradicional aplica alguns princípios da biologia evolutiva na busca por soluções satisfatórias para um dado problema. Sendo os Algoritmos Genéticos – AGs inspirados na teoria evolutiva, nele encontramos modeladas as quatro forças evolutivas:

a) **Derivação gênica**: trata-se de uma força evolutiva de origem estocástica e que explica variações aleatórias nas frequências dos genótipos. É modelada no AG nos momentos em que se faz a escolha dos indivíduos de forma estocástica e no momento da seleção não determinística da sobrevivência dos novos indivíduos gerados;
b) **Mutação**: é a origem de variação nos genótipos possibilitando introduzir novidades dentro da população, nos AGs é modelada por operadores genéticos.
c) **Seleção natural**: pode ser vista como a sobrevivência diferenciada de sequências genotípicas baseada em sua aptidão ao longo das gerações (ou eventos reprodutivos). Nos Algoritmos Genéticos é modelada nos processos de seleção dos indivíduos para reprodução e na sobrevivência dos novos indivíduos;
d) **Migração:** corresponde ao fluxo gênico, que é a troca de informações genética entre populações. É modelada nos AGs no momento da entrada de novos indivíduos dentro da população (CUSTÓDIO; BARBOSA; DARDENE, 2004).

5.4.2 Componentes do algoritmo genético

5.4.2.1 Indivíduos

É de uso comum na área de AGs utilizar os termos genoma e mesmo cromossoma como um sinônimo para indivíduo. Um indivíduo pode ser visto como um conjunto de genes (genótipo), apesar de toda representação por parte do algoritmo ser baseada única e exclusivamente em termos do genótipo, toda avaliação é baseada em seu fenótipo (conjunto de características observáveis no objeto resultante do processo de decodificação dos genes).

5.4.2.2 Populações

Um AG manipula a frequência com que determinados genes aparecem nas populações sobre os quais atua. Uma geração diz respeito ao número de vezes pela qual uma população passou pelo processo de seleção, reprodução, mutação e atualização. O grau de convergência representa o quão próximo a média de adaptação da atual geração está de suas anteriores. É objetivo dos AGs fazer a população convergir em um valor ótimo de adaptação, sendo a diversidade fundamental para a amplitude de busca, evitando-se assim convergência prematura.

5.4.2.3 Função custo

Uma **função custo** gera uma saída a partir de um conjunto de parâmetros de entrada (um cromossomo). A mesma pode ser uma função matemática, um experimento ou até um jogo. O objetivo é modificar a saída para um valor desejado, encontrando os valores apropriados para os parâmetros de entrada. Nós fazemos isso, sem pensar quando enchemos uma banheira com água, o **custo** é a diferença entre a temperatura atual e desejável de água. Os **parâmetros de entrada** são quanto às torneiras quente e fria são utilizadas. O AG começa definindo um cromossomo ou um vetor de parâmetros que serão otimizados. Se um cromossomo tem $\boldsymbol{N_{par}}$ parâmetros dados por **p1, p2,**....,**p**$\boldsymbol{N_{par}}$, então o cromossomo é escrito como um vetor de $\boldsymbol{N_{par}}$ elementos.

$$\boldsymbol{cromossomo} = [\mathbf{p1}, \mathbf{p2}, \ldots, \mathbf{p}N_{par}] \tag{22}$$

Cada cromossomo tem um custo encontrado avaliando a função custo, **f**, em **p1**, **p2**,....,p$\boldsymbol{N_{par}}$:

$$\text{custo} = \mathbf{f}(\text{cromossomo}) = f(p1, p2, \ldots, p\boldsymbol{N_{par}}) \tag{23}$$

A representação de possíveis soluções no espaço de busca de um problema define a estrutura do cromossomo a ser manipulado pelo algoritmo. Essa depende do tipo do problema e do que se deseja manipular geneticamente. Uma representação bastante utilizada é a binária, devido a sua facilidade de manipulação pelos operadores genéticos. Um binário pode representar um número real $X_R \epsilon [X_{min}, X_{max}]$ com precisão de *p* casas decimais. Para isso são necessários K bits, sendo K calculado pela inequação (24):

$$2^K \geq (X_{max} - X_{min}) \, x \, 10^p \tag{24}$$

A decodificação de cromossomo consiste basicamente na construção da solução real do problema a partir do cromossomo, esse processo constrói a solução para que esta seja avaliada

pelo problema. Na transformação para o número real considera-se o intervalo de valores ou comprimento contínuo do domínio (C) dos reais de tal forma que:

$$X_R = X_b \cdot \frac{C}{2^n - 1} + X_{min} \quad (25)$$

Onde: X_b é o inteiro correspondente ao binário**;** **n** é o número de bits do cromossomo**,** **C** é comprimento do domínio da variável **X,** dado por **C**=$|(X_{max} - X_{min})|$.

5.4.2.4 População Inicial

O Algoritmo genético começa com uma população de cromossomos denominada população inicial. A mesma tem N_{ipop} cromossomos sendo uma matriz preenchida com zeros e uns gerados randomicamente por:

$$IPOP - round\{random(N_{ipop}, N_{bits})\} \quad (26)$$

Onde a função $random(N_{ipop}, N_{bits})$ gera uma matriz N_{ipop}X N_{bits} de números randômicos uniformes entre zero e um. A função **round{}** arredonda o número para o inteiro mais próximo. Cada linha na matriz é um cromossomo que corresponde a valores discretos, após isso os parâmetros são passados para a função custo para avaliação. Uma população inicial grande prove ao AG uma amostra satisfatória do espaço de busca.

5.4.2.5 Seleção

Nos processos evolutivos os indivíduos se reproduzem diferenciadamente de acordo com a sua aptidão, isto é, aqueles melhor adaptados tendem a se reproduzir com mais frequência. Em AGs, o método de Seleção Proporcional à Aptidão (SPA) (GOLBERG, 1989) seleciona os indivíduos com uma probabilidade proporcional à sua aptidão. Então a probabilidade do indivíduo i ser selecionado é dada por:

$$p_i = \frac{F_i}{\sum_{j=1}^{N} F_j} \quad (27)$$

Onde $F_i \geq 0$ é a aptidão do indivíduo **i** e **n** tamanho da população. Esse método é também conhecido como seleção por roleta, onde a probabilidade de cada indivíduo ser selecionado para reprodução corresponde a uma divisão em uma roleta de tamanho proporcional à sua aptidão relativa ao restante da população. Dessa forma, nos casos em que existam na população indivíduos com aptidão muito melhor do que outros, os mesmos acabam sendo selecionados com uma frequência muito alta. Este fato pode ocasionar a convergência prematura, isto é, a perda de diversidade. Outro problema que ocorre é que quando as aptidões dos indivíduos apresentam valores muito próximos, com isso a distribuição de probabilidade de seleção dos indivíduos se torna quase uniforme tornando o processo uma seleção aleatória.

Como o objetivo de superar os efeitos negativos da roleta *roulette whell selection* ao invés de se utilizar diretamente o valor da aptidão, pode-se utilizar o posto do indivíduo na população *rank selection*. Dessa forma, a população é ordenada pelos valores de aptidão e cada indivíduo recebe um posto. A probabilidade de seleção é dada de acordo com o posto na denominada seleção por posto (BAKER, 1985). Outra técnica é a Seleção por Torneio *tournament selection* (BRINDLE, 1981; GOLBERG; DEB, 1991), que considera um determinado número de indivíduos (o tamanho do torneio) que são escolhidos aleatoriamente e o melhor dentre eles é selecionado para reprodução.

5.4.2.6 Substituição parental

Após a reprodução, os indivíduos novos devem substituir os anteriores (pais). O Elitismo é uma das técnicas mais utilizadas, a mesma consiste em copiar os indivíduos mais adaptados para a próxima geração com a finalidade na população sem correr o risco de perdê-los durante o evento reprodutivo. Em um AG os indivíduos de uma população se reproduzem (pela aplicação dos operadores genéticos) até que uma nova população, de tamanho igual a original, seja formada. Com isso, os indivíduos da nova população (filhos) substituem os da antiga, chamada de parental, seguindo um critério de seleção, isto é, de substituição parental.

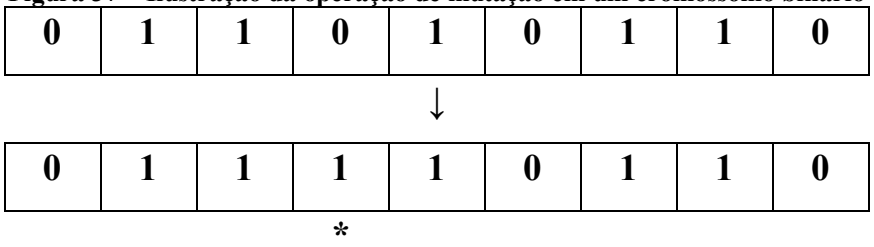
5.4.2.7 Operadores genéticos

Nessa seção descrevem-se dois operadores básicos de um AG: mutação e recombinação. Os mesmos fazem com que os novos cromossomos gerados durante a reprodução sejam diferentes dos pais, isto é, introduzem mudanças genotípicas na população.

5.4.2.7.1 Mutação

O operador de mutação funciona da seguinte maneira, após um pai ser selecionado para reprodução o seu cromossomo é copiado para um cromossomo filho. Uma posição no cromossomo é selecionada de acordo com determinado critério, podendo ser aleatório ou dependente da representação, assim o valor do cromossomo nessa posição é alterado (mutado) para outro valor. Por exemplo, em uma representação binária determinada posição pode conter um bit 0 e ter seu valor alterado 1 e vice-versa (Figura 37).

Figura 37 – Ilustração da operação de mutação em um cromossomo binário

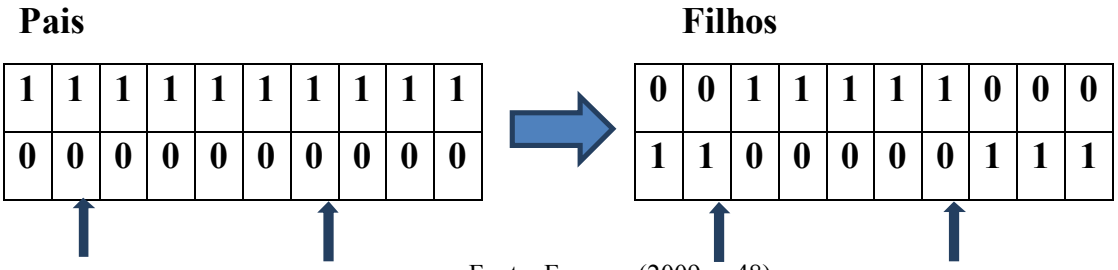


Fonte: Fonseca (2009, p. 47).

5.4.2.7.2 Recombinação

Essa operação troca informações entre dois cromossomos pais para gerar dois filhos de forma análoga ao fenômeno natural de recombinação que ocorre durante a meiose, isto é, divisão celular para geração de gametas (células reprodutivas). A recombinação, também denominada *crossover*, opera da seguinte forma: uma posição dentro do cromossomo é escolhida aleatoriamente e é chamada de ponto de corte. Então, os dois indivíduos pais trocam o conteúdo dos seus cromossomos a partir do ponto de corte. Este é o operador de *crossover* de 1 ponto podendo ser generalizado para N pontos. A Figura 38 mostra a aplicação de crossover de dois pontos em um cromossomo de representação binária.

Figura 38 – *Crossover* de dois pontos. Onde setas verticais indicam os pontos de corte



Fonte: Fonseca (2009, p 48).

5.4.2.8 Parâmetros

Encontrar os parâmetros ótimos para os operadores pode se tornar muito difícil quando são aplicados vários tipos de mutações e recombinações. Pode-se dizer que escolher os parâmetros "ótimos" pode ser uma tarefa tão difícil quanto resolver o problema inicial. A simples definição da taxa de aplicação dos operadores com base em experimentação é um método bastante eficiente, pois o conjunto de parâmetros a ser testado é muito grande. Além disso, os parâmetros não são independentes e um conjunto que resulte em bons resultados pode não funcionar em outras instâncias do mesmo problema.

5.4.2.9 Inicialização

Na maioria dos AGs a geração da população inicial é um processo aleatório. Nele, os indivíduos são criados e os valores contidos nos seus cromossomos são determinados por um gerador de número aleatórios. De fato, o processo de geração da população inicial é um processo estocástico que depende da "semente" utilizada pelo gerador. Mais precisamente, em computadores, os geradores de números aleatórios são geradores de números pseudoaleatórios, uma vez que sempre a mesma sequência de números é gerada para cada semente.

A geração da população inicial pode ser um processo altamente ineficiente no que diz respeito à qualidade das soluções e principalmente em relação à viabilidade dessas. Isso quer dizer que um processo de geração aleatória possui grandes chances, se a representação permitir, de resultar em soluções inválidas dentro de um espaço de busca. Para contornar esses problemas, a geração dos indivíduos iniciais pode ser feita segundo a heurística do problema (EIBEN; SMITH, 2003).

5.4.2.10 Critério de parada

A definição do critério de parada é fundamental para a execução eficiente de um AG. Por exemplo, se o critério escolhido tornar a execução muito curta o AG pode encontrar apenas soluções subótimas, enquanto execuções muito longas irão desperdiçar recursos computacionais. Se um valor ótimo já for encontrado para a função objetivo, um critério óbvio é terminar a execução quando esse valor for encontrado. Porém, essa situação raramente ocorre em aplicações práticas, devendo-se recorrer a outros critérios de parada.

É comum a utilização de um número máximo de chamadas à função de aptidão (avaliações de função) sendo esse critério eficaz para fixar o custo computacional de uma execução. Pode-se também utilizar informações sobre o estado da função aptidão, isto é, seguir algum critério quanto às melhorias nos valores de aptidão. Segundo essa estratégia, a execução de um AG pode ser concluída quando as melhorias, ao final de certo número de avaliações de função, estiverem abaixo de um valor predeterminado (estagnação).

5.4.3 Considerações finais

Nesse capítulo, apresentaram-se conceitos sobre evolução, seleção natural, e biologia molecular. Além disso, introduziram-se alguns conceitos de computação evolucionária e algoritmos genéticos, que são modelos computacionais que simplificam o que ocorre nos processos evolutivos naturais. No próximo capitulo, apresenta-se um Algoritmo Neuroevolutivo, que incorpora aspectos de inspiração biológica.

CAPÍTULO 6 – METODOLOGIA PROPOSTA

"*Quanto mais conhecemos, mais amamos*."
Leonardo da Vinci

6.1 INTRODUÇÃO

No capítulo 2, foram revisadas várias pesquisas sobre ANEs. Além disso, no capítulo 1, apontaram-se algumas lacunas não cobertas pelos mesmos. A seguir, apresenta-se um novo ANE que incorpora alguns aspectos de inspiração biológica (discutidos na secção 6.2). E que, adicionalmente, leva em consideração alguns conceitos levantados no capítulo 3, onde discutiram-se alguns modelos de desenvolvimento artificias e que serviram de fundamento para a construção do modelo descrito na secção 6.3.1 e que possibilita gerar as arquiteturas diretas e recorrentes, apresentadas no capitulo 4. E finalmente, o capítulo 5 introduz conceitos de evolução biológica e computação evolucionária, que serviram de base para construção de um novo AG inspirado no processo evolutivo natural. Na certeza de que nesta pesquisa será dado um passo a mais no desenvolvimento de ANEs.

6.2 MODELO DE DESENVOLVIMENTO ARTIFICIAL E EVOLUÇÃO, SEGUNDO DAWKINS (2004)

Nessa seção descrevem-se algumas características distintivas que são essenciais para a formalização de um modelo de desenvolvimento artificial associado a um algoritmo evolutivo. Muitas dessas características foram abordadas por Dawkins (2004), que apresenta um modelo de desenvolvimento artificial simulado em computador, denominado **DESENVOLVIMENTO**, integrado a outro que tenta simular o processo evolutivo de formas de criaturas artificiais, mostrando como ocorrem mudanças sucessivas durante a evolução, sendo esse denominado **EVOLUÇÃO**.

Pressuposto 1: Os livros de ciência da computação costumam ilustrar o poder do que chamam programação recursiva como um procedimento simples de crescimento em árvore. O computador começa desenhando uma linha vertical. Em seguida, a linha ramifica-se em duas. Então cada ramo se divide em dois sub-ramos, que por sua vez partem-se em sub-ramos e

assim por diante. Esse processo é recursivo porque a mesma regra (neste caso a ***regra de ramificação***) aplica-se em âmbito local por toda a árvore (DAWKINS, 2004).

Pressuposto 2: Como bem nos ensina Dawkins (2004) a ramificação recursiva também é uma boa metáfora para o desenvolvimento de plantas e animais em geral. O autor não menciona que os embriões de animais se parecem com árvores que se ramificam, e não se parecem, mas ressalta que todos os embriões crescem por divisão celular. As células sempre sem dividem em duas células filhas. E os genes sempre exercem seus efeitos finais sobre os organismos por meio de influências locais sobre as células e sobre os padrões de ramificação em duas direções da divisão celular.

Pressuposto 3: Dawkins (2004) defende que a regra simples de ramificação no traçado de árvores parece ser um processo promissor análogo ao desenvolvimento embrionário. Além disso, introduz a ideia de que esse processo pode ser traduzido para um procedimento de computador, rotulado como **DESENVOLVIMENTO** e incorporado a um programa maior denominado **EVOLUÇÃO**. Para a criação desse programa maior, inicialmente, considerou-se que os genes, em nível biológico, fazem duas coisas: influenciam o desenvolvimento e são transmitidos às gerações posteriores.

Pressuposto 4: Dawkins (2004, p. 89) considera que os genes deveriam exercer ligeira influência sobre as regras de desenho no processo de desenvolvimento artificial. Por exemplo, um gene poderia influenciar o ângulo de ramificação; outro o comprimento de algum ramo específico. Outra coisa que o gene poderia fazer é influenciar a profundidade da recursividade, ou seja, o número de ramificações sucessivas. E então esses mesmos genes serão ou não transmitidos à geração seguinte.

Dawkins (2004) nos ensina que no processo de desenvolvimento de uma árvore ou planta, cada uma delas tem uma "fórmula genética própria". O que é válido também para os genes biológicos, que só começam a ter algum significado quando são traduzidos, mediante a síntese de proteínas, descrita na seção 5.3.2 (DAWKINS, 2004).

Pressuposto 5: Dawkins (2004) sugere que no desenvolvimento do programa de computador os dois procedimentos intitulados **DESENVOLVIMENTO** e **REPRODUÇÃO** deveriam ser interdependentes. Assim sendo, a **REPRODUÇÃO** transmitiria valores de genes para o **DESENVOLVIMENTO**, dessa forma, os mesmos influenciariam as regras de crescimento. Ao montar, assim, os dois módulos de programas, **DESENVOLVIMENTO** E **REPRODUÇÃO,** deve-se levar em consideração a possibilidade de ocorrer mutação nos

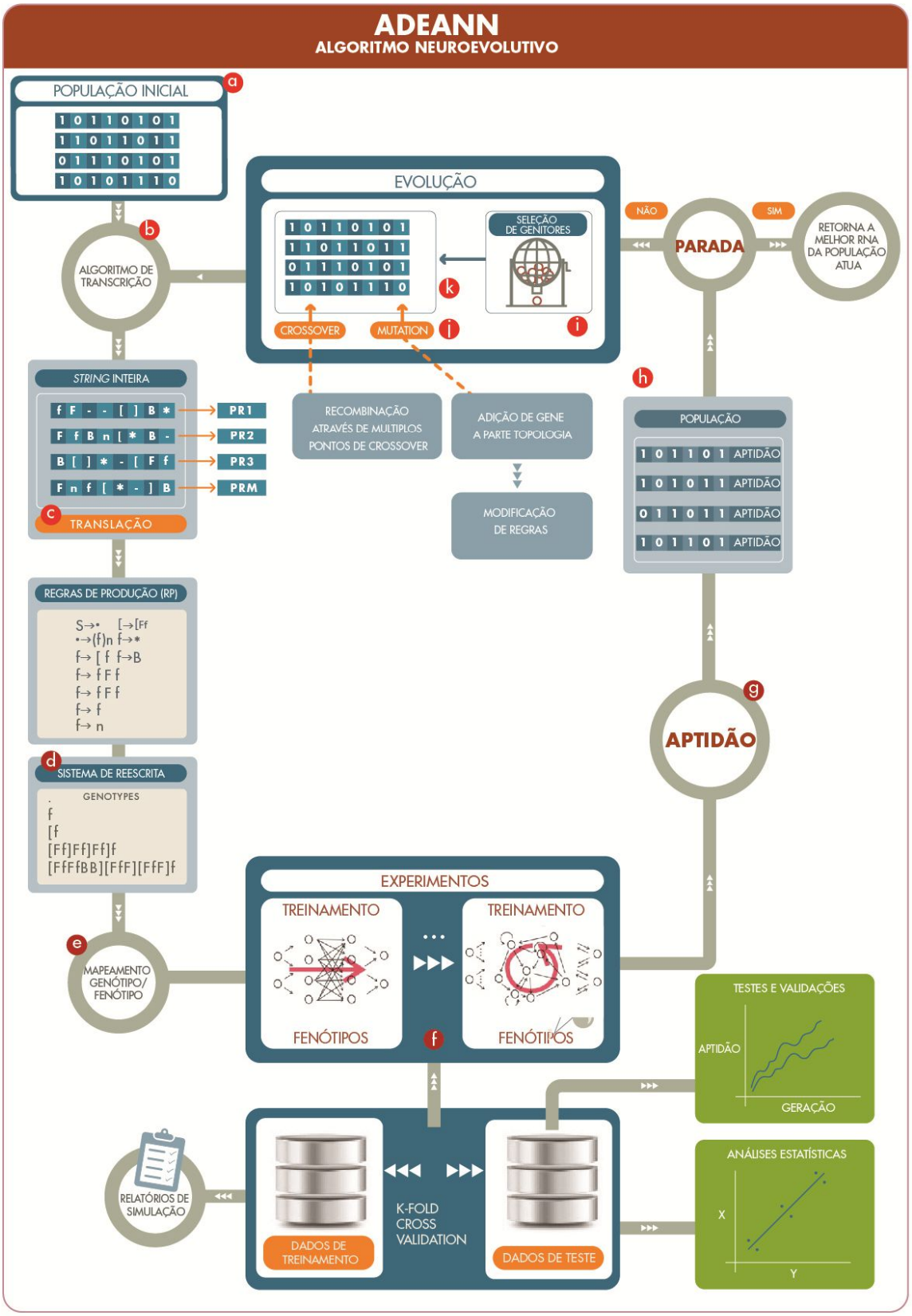
genes transmitidos, para as gerações seguintes, pela **REPRODUÇÃO**. O **DESENVOLVIMENTO,** então, traduz esses genes em ações de desenho.

Pressuposto 6: Os dois módulos devem ser juntados em um grande programa intitulado **EVOLUÇÃO**. Onde a evolução consiste basicamente na repetição incessante da **REPRODUÇÃO**. Em cada geração, a **REPRODUÇÃO** passa os genes que lhe são fornecidos pela geração anterior à geração seguinte, mas com pequenos erros aleatórios-mutações. Dawkins (2004) menciona que os modelos simulados em computador podem nos ajudar a aprender algo sobre a evolução real. Um modelo de desenvolvimento artificial inspirado biologicamente deve possuir capacidade de organização. A forma das plantas não é codificada no ADN, o que existe no ADN é uma fórmula genética que quando seguida gera a forma final de um organismo ou órgão.

6.3 ANE PROPOSTO

A estrutura geral do *ADEANN* é mostrada na Figura 39. O processo de otimização realizado pelo sistema é realizado em várias etapas. O AG inicializa uma população de indivíduos aleatoriamente com valores zeros e uns (Figura 39a). Após isso, para cada indivíduo da população, são realizados os processos de transcrição (Figura 39b) e de tradução (Figura 39c) desses *bits* em regras de produção válidas. Ambos os processos são executados pela Função Extração-de-Regras-com-o-AG, a ser apresentada na seção 6.3.2. Depois de encontrar as regras de produção válidas (Tabela 6), o sistema de reescrita gera os genótipos (Figura 39d). Todos os genótipos são mapeados para fenótipos (arquiteturas de rede neurais) (Figura 39e). As RNAs são então treinadas (Figura 39f) e após a validação, os testes são realizados. Para medir a precisão da classificação de cada RNA, um valor de aptidão é calculado (Figura 39g). Os genótipos são classificados de acordo com o desempenho de cada rede neural (Figura 39h). O AG seleciona os melhores indivíduos (Figura 39i), para a aplicação dos operadores genéticos (mutação e de cruzamento), (Figura 39j), após isso uma nova população é obtida (Figura 39k). Os passos anteriores são repetidos durante um número “n” de gerações. Finalmente, os relatórios da simulação são gerados e as análises estatísticas são realizadas.

Figura 39 – Algoritmo Neuro Evolutivo Proposto



Fonte: o autor (2016).

6.3.1 Codificação neural pelo Sistema-L

Com objetivo de imitar o mecanismo de desenvolvimento de estruturas, incluindo o cérebro, utilizou-se um sistema-L paramétrico com memória. Com base no **pressuposto 1,** de Dawkins (2004), citado na seção anterior, adotamos um direcionamento em relação a fórmula genética, no sentido de que a mesma fosse recursiva e com possibilidades de gerar ramificações no processo de desenvolvimento artificial. Do **pressuposto 2**, conclui-se que a ramificação recursiva poderia ser uma boa metáfora para o desenvolvimento artificial de neurônios. O **pressuposto 3** de Dawkins (2004) defende que a regra de ramificação parece ser promissora no traçado de árvores como um processo análogo ao desenvolvimento embrionário. No **pressuposto 4**, Dawkins (2004) considera os genes deveriam exercer alguma ligeira influência sobre as regras de desenho. Para modelar a metáfora da codificação cromossômica e o processo de desenvolvimento de órgãos, dentre eles do sistema nervoso, foi considerado que o desenvolvimento de plantas e animais é produzido pela informação genética contida em cada célula do organismo. Onde cada uma contém a mesma informação genética (o genótipo), a qual determina a forma como cada célula se comporta e, por consequência, a forma final e função do organismo (o fenótipo). Como mencionado na seção anterior quando seguida a fórmula genética (receita) codificada no ADN resultará na forma final do organismo, ou seja, a mesma possibilitara à organização do processo de desenvolvimento de arquiteturas neurais desejadas.

A seguir apresenta-se um Sistema-L que implementa aspectos de organização, modularidade, repetição (uso múltiplo de subestruturas) e hierarquia (composição recursiva de subestruturas) na obtenção de topologias de RNAs. Na construção do modelo, tomou-se por base alguns Sistemas-L existentes, dentre eles destacam-se o citado na secção 3.2.1 que representa a interpretação gráfica da cadeia de caracteres gerada com base no movimento de uma tartaruga, o da seção 3.2.3, onde é apresentado um Sistemas-L com memória para o desenvolvimento de plantas e finalmente os métodos de reescrita de aresta (PRUSINKIEWICZ; LINDENMAYER, 2004) e reescrita de nós (PRUSINKIEWICZ; LINDENMAYER, 2004).

O Sistema-L proposto nessa pesquisa consiste em um modelo que permite gerar conexões diretas e recorrentes, com possibilidade de gerar múltiplas ramificações e contempla os princípios: organização, modularidade, repetição e hierarquia. O mesmo pode ser descrito

como uma gramática **G={∑, ∏, α}**, onde o alfabeto é **∑={. , *f* , F, n, [,], *, B}** e **α = .** é o axioma, as regras de produção ∏ são descritas na Tabela 6.

Tabela 6 – Regras de Produção do Sistema-L proposto

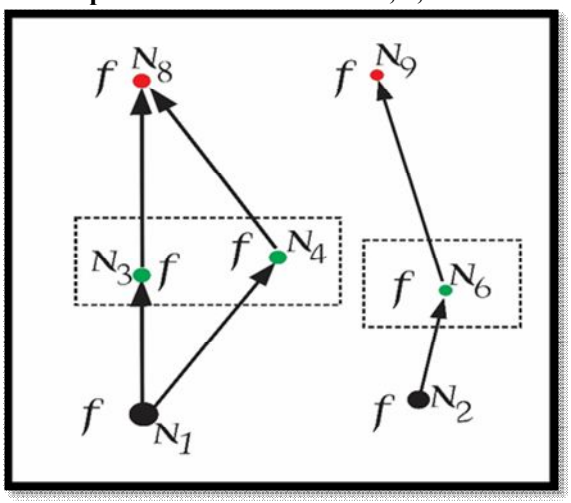
Identificador da Regra	Regras
1,2	S → . (axioma) (2) . → (f..f)n
3	(3.1)f→[f (3.2) f→fFf (3.3) f→fF (3.4) f→n
3,4,5	(3.5) f→f (3.6) f→fB (4) [→[Ff] (5) f→f*

Fonte: o autor (2016).

Onde: "**f**" denota um neurônio; "**F"** é uma conexão entre neurônios com peso w; "**[**" e "**]"** indicam, respectivamente, armazenamento e recuperação, do estado corrente do desenvolvimento, "*" significa que a *string* anteriormente armazenada é recuperada, "**B**" é uma conexão de um neurônio com um bloco de neurônios.

Na Figura 40 a seguir, apresenta-se um exemplo de utilização das mesmas na condução do processo de formação da arquitetura de uma rede neural. O exemplo mostra de forma simplificada como o desenvolvimento da rede de neurônios artificiais se processa, vale ressaltar que o *ADEANN* possibilita gerar topologias variadas e maiores do que a apresentada. Essa etapa é antecedida pela extração das regras de produção descrita na secção 6.3.2.

Figura 40 – Arquitetura da Rede Iterada gerada após a execução das iterações 1 a 12 (IT.1 a IT.12), apresentadas nas Tabelas 7, 8, 9 e 10

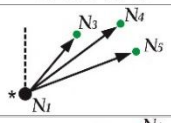
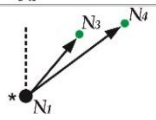
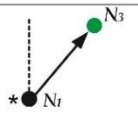


Fonte: o autor (2016).

Começando com o axioma **α = .** , o mapeamento entre o genótipo e fenótipo, durante a primeira etapa do desenvolvimento é ilustrado na última linha da Tabela 7, sendo representado

pela iteração1 **(IT.1).** O genótipo **(.)** representa o início do processo de desenvolvimento que tem como (fenótipo) a produção de um ponto **(.)** também, esse mapeamento é obtido pela regra de produção 1 **(R1),** ilustrada na Tabela 6.

Tabela 7 – Processo de Construção da Rede Iterada, iniciando da camada de entrada para a camada intermediária, representação da primeira ramificação

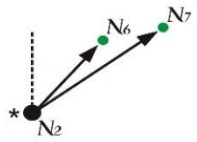
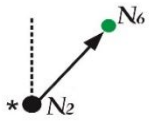
REGRAS	GENÓTIPO	FENÓTIPO	IT
R_4	$n_3\ n_4\ n_5\ n_1$ $[\,F\,f^{\uparrow}]\,F\,f^{\uparrow}]\,F\,f^{\uparrow}]\,f^{\uparrow}$ $E_1\quad E_1\quad E_1\quad E_1$		IT. 6 ARMAZENA RAMIFICAÇÃO DE N_1
R_4	$n_3\ n_4\ n_1$ $[\,F\,f^{\uparrow}]\,F\,f^{\uparrow}]\,f^{\uparrow}$ $E_1\quad E_1\quad E_1$		IT.5
R_4	$n_3\ n_1$ $[\,F\,f^{\uparrow}]\,f^{\uparrow}$ $E_1\quad E_1$		IT.4
R_3	n_1 $[\,f^{\uparrow}$ E_1	* N_1	IT.3
R_2	n_1 $f^{\uparrow}$ E_1	N_1	IT.2
R_1	.	.	IT.1

Fonte: o autor (2016).

O significado da regra de produção 2 **(R2)** é substituir o ponto inicial gerado na etapa anterior por uma sequência de possíveis neurônios da camada de entrada, onde cada f que aparece no sucessor da regra **(R2**), consiste no número possível de neurônios da camada de entrada, especificados por **n**, por isso a regra tem a forma $. \rightarrow (f..f)n$. Percebe-se na quinta linha da Tabela 7 e quarta da Tabela 8, na terceira coluna, que na segunda iteração **IT.2**, são gerados dois possíveis neurônios na camada de entrada, os mesmos possuem fenótipos representados por N_1 e N_2. Observa-se também, a relação entre genótipo e fenótipo na quinta linha da Tabela 7, no qual o estado inicial do genótipo f é **E1,** onde f representa um possível neurônio n_1 e que representa o fenótipo **N1,** ver a Iteração 2 **(IT.2**) mostrada na Tabela 7. Processo análogo é observado na terceira linha e segunda coluna da Tabela 8, percebe-se que o

estado inicial, apontado pelo genótipo, é **E2** e que o genótipo f gera um possível neurônio, que é mapeado para o fenótipo **N2,** ver a Iteração 2 **(IT.2**)

Tabela 8 – Processo de Construção da Rede Iterada, iniciando da camada de entrada para a camada intermediária, representação da segunda ramificação

REGRAS	GENÓTIPO	FENÓTIPO	IT
R_4	$[\,F\,f^{\uparrow}]\,F\,f^{\uparrow}]\,f^{\uparrow}$ (E2 E2 E2; n_6 n_7 n_2)		IT. 9 ARMAZENA RAMIFICAÇÃO DE N2
R_4	$[\,F\,f^{\uparrow}]\,f^{\uparrow}$ (E2 E2; n_6 n_2)		IT.8
R_3	$[\,f^{\uparrow}$ (E2; n_2)	* N2	IT.7
R_2	$f^{\uparrow}$ (n_2)	N2	IT.2
R_1	•	•	IT.1

Fonte: o autor (2016).

Na terceira iteração **(IT.3)**, quarta linha e segunda coluna da Tabela 7, aplica-se a regra 3.1, $f \rightarrow [\,f$, onde o colchete **[** significa armazenar o estado atual, onde o mesmo é **(E1)** e f representa o possível neurônio desenhado na iteração 2 **(IT.2)**, que nesse caso é n_1**.** O fenótipo produzido é ilustrado na quarta linha e terceira coluna da Tabela 7, onde vê-se desenhado um neurônio **N1,** o asterisco **(*)** representa a posição corrente, a partir da qual o desenvolvimento da primeira ramificação começará.

Na quarta iteração **(IT.4),** mostrada na terceira linha e segunda coluna da Tabela 7, inicia-se o processo de construção da primeira ramificação com a utilização da quarta regra de produção **(4)** $[\rightarrow [Ff]$**,** onde o símbolo **F** significa desenhar uma conexão com peso w . Ou seja, a aplicação de **(R4)** ao genótipo $[f$**,** obtido na iteração 3 **(IT.3)**, produz o genótipo $[Ff]f$**,** cujo interpretação é a seguinte: a partir do estado **E1**, desenha-se uma conexão com

peso ***w***, o ***f*** representa um possível neurônio n_3. O colchete **]** significa recuperar o estado corrente **E1** e o ***f*** que finaliza a *string* que representa o neurônio n_1 gerado anteriormente na iteração 3 **(IT.3)**. O mapeamento entre genótipo e fenótipo obtido após a aplicação de **(R4)** é ilustrado, na terceira linha e terceira coluna da Tabela 7, com isso o processo de construção da primeira ramificação é iniciado na iteração 4 **(IT.4)**.

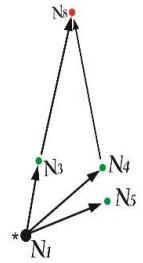
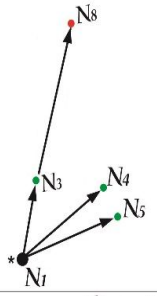
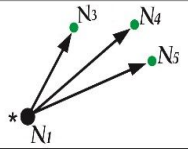
Nas iterações cinco **(IT.5)** e seis **(IT.6)**, mostradas na Tabela 7, a quarta regra de produção **(4)** [→ [***Ff***] é aplicada a *string* [***Ff***]***f***, sendo que na iteração 5 **(IT.5)** a *string* resultante será [***Ff***]***Ff***]***f***. O mapeamento entre o genótipo e o fenótipo é apresentado na segunda linha e terceira coluna, da Tabela 7. Percebe-se que uma conexão entre **N1 e N4** é adicionada nessa ramificação. A conclusão da mesma é obtida na sexta iteração **(IT.6),** aplicando-se novamente a regra **(4)** a *string* [***Ff***]***Ff***]***f*** , que terá como *string* resultante [***Ff***]***Ff***]***Ff***]***f*****.** O mapeamento entre genótipo e o fenótipo é apresentado na primeira linha e terceira coluna da Tabela 7, onde uma nova conexão é inserida entre **N1 e N5**. Dessa forma, tem-se a primeira ramificação construída a partir de **N1,** de acordo com o que é mostrado na primeira linha e terceira coluna da Tabela 7. Na **(IT.6)** armazena-se o conteúdo do genótipo que representa essa ramificação [***Ff***]***Ff***]***Ff***], para uso posterior, esse procedimento é similar aos utilizado por Prusinkiewicz e Lindenmayer (2004).

Na Tabela 8 inicia-se o processo de construção da segunda ramificação, na iteração 1 **(IT.1)**, aplicando-se a regra 3.1, ***f*** → [***f*** a *string* ***f***, gerada na segunda iteração **(IT.2)**. O colchete **[** significa armazenar o estado corrente, que no caso é (**E2)**, onde ***f*** significa desenhar um possível neurônio, que nesse caso é n_2**.** O genótipo produzido **[f** é ilustrado na terceira linha e segunda coluna da Tabela 8, onde vê-se desenhado um neurônio **N2,** o asterisco **(*)**, que indica o estado atual armazenado **(E2)**. As outras etapas **(IT.8 e IT.9)** que seguem são idênticas as apresentadas anteriormente na construção da primeira ramificação **(IT.4, IT.5 e IT.6)**, mostradas na Tabela 7, vale destacar que na iteração 9 **(IT.9)** armazena-se o genótipo que representa a segunda ramificação (representada pela *string* [***Ff***]***Ff***]) para uso posterior.

Com isso, as construções das duas ramificações a partir de **N1 e N2** são finalizadas, a próxima etapa será construir as conexões da camada oculta para a camada de saída, o que é mostrado a seguir.

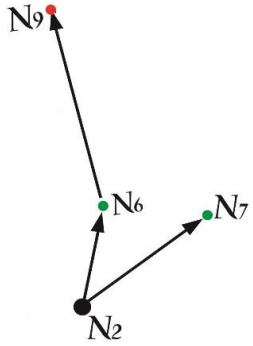
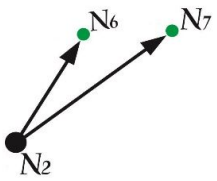
O processo final de construção da rede iterada, com conexões partindo da camada intermediária com direção a camada de saída, a partir das iterações 6 **(IT.6)** e 9 **(IT.9)** é ilustrado nas Tabelas 9 e 10 respectivamente. Na última linha da Tabela 9, percebe-se que as estruturas genotípicas e fenotípicas apresentadas são as mesmas da primeira linha da Tabela 7, na qual foi finalizada a construção da primeira ramificação a partir do neurônio **N1.** Considerando que o genótipo $[Ff]Ff]Ff]f$ produziu essa primeira ramificação na iteração 6 **(IT.6)**, o processo de desenvolvimento da rede continua a partir dessa *string*. A próxima etapa consiste na aplicação da regra 3.2 $f \rightarrow fFf$ a *string* $[Ff]Ff]Ff]f$ obtida na iteração 6 **(IT.6),** tomado por base o primeiro f da *string* da esquerda para a direita e a regra 3.5 $f \rightarrow f$ considerando os três f**'s** restantes**,** a *string* resultante passa a ser: $[fFf]Ff]Ff]f$. Na segunda linha da Tabela 9, detalha-se melhor essa representação genotípica, na mesma associa-se cada f a um possível neurônio. Percebe-se o surgimento de um novo neurônio referenciado por n_8. A representação fenotípica apresentada na terceira coluna dessa mesma linha mostra o neurônio em vermelho, sendo esse processo executado na iteração 10 **(IT.10)**. Dessa forma, uma conexão entre **N3** e **N8** é estabelecida.

Tabela 9 – Processo de Construção da Rede Iterada, direcionada da camada oculta para a camada de saída, a partir da primeira ramificação obtida na Tabela 7

REGRAS	GENÓTIPO	FENÓTIPO	IT
$R_{3.4}$	$n_3\ n_8\ n_4\ n_5\ n_1$ $[F\,n\,F\,n\,]F\,n\,F\,]F\,n\,]n$ $E_1\quad E_1\quad E_1\quad E_1$		IT.11
$R_{3.3}$	$n_3\ n_8\ n_4\ n_5\ n_1$ $[F\,f\,F\,f\,]F\,f\,\ F\,]F\,f\,]f$ $E_1\quad E_1\quad E_1\quad E_1$		
$R_{3.2}$	$n_1\ n_8\ n_4\ n_5\ n_1$ $[F\,f\,F\,f\,]F\,f\,]F\,f\,]f$ $E_1\quad E_1\quad E_1\quad E_1$		IT.10
NA	$n_3\ n_4\ n_5\ n_1$ $[F\,f\,]F\,f\,]F\,f\,]f$ $E_1\quad E_1\quad E_1\quad E_1$		IT.6

Fonte: o autor (2016).

Tabela 10 – Processo de Construção da Rede Iterada, direcionada da camada oculta para a camada de saída, a partir da segunda ramificação obtida na Tabela 8 (IT.9)

REGRAS	GENÓTIPO	FENÓTIPO	IT
$R_{3.4}$	$[\,F\,n^{n_6}\,F\,n^{n_9}\,]\,F\,n^{n_7}\,]\,n^{n_1}$ $E_2 \qquad E_2 \qquad E_2$		IT.12
$R_{3.2}$	$[\,F\,f^{n_6}\,F\,f^{n_9}\,]\,F\,f^{n_7}\,]\,f^{n_1}$ $E_2 \qquad E_2 \qquad E_2$		
NA	$[\,F\,f^{n_6}\,]\,F\,f^{n_7}\,]\,f^{n_1}$ $E_2 \qquad E_2 \qquad E_2$		IT.9

Fonte: o autor (2016).

Na iteração 11 **(IT.11)** aplica-se a regra (3.3) $f \rightarrow Ff$ a *string* $[FfFf]Ff]Ff]f$ gerada na iteração 10 **(IT.10),** tomando por base o terceiro f da esquerda para a direita e a regra 3.5 $f \rightarrow f$ aplicada aos quatro **f's** restantes. Com isso, a *string* resultante passa a ser: $[FfFf]FfF]Ff]f$, essa representação genotípica é representada detalhadamente na primeira linha e segunda coluna da Tabela 9. Observa-se a associação de cada f a um possível neurônio, na representação fenotípica apresentada na terceira coluna dessa mesma linha uma nova conexão entre **N4 e N8** surge. E finalmente aplica-se regra 3.4 $f \rightarrow n$ a *string* $[FfFf]FfF]Ff]f$, onde o significado dessa regra é tornar os possíveis neurônios definitivos na arquitetura gerada, com isso a *string* resultante torna-se: $[FnFn]FnF]Fn]n$. Dessa forma, conclui-se na iteração 11 **(IT.11)** o processo de construção da rede iterada para a primeira ramificação.

O processo de construção da segunda ramificação é análogo ao anterior, sendo o mesmo apresentado na Tabela 10. Considerando que o genótipo $[Ff]Ff]f$ produziu a segunda ramificação na iteração 9 **(IT.9)** mostrada na Tabela 8. O processo de desenvolvimento da rede

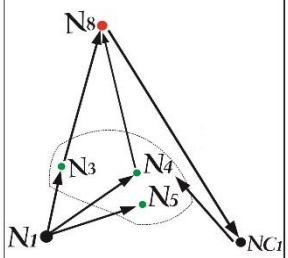
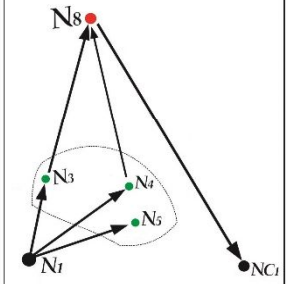
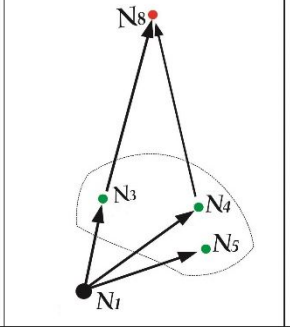
continua a partir da *string* $[Ff]Ff]f$. Considerando a aplicação da regra 3.2 $f \rightarrow fFf$ a *string* anterior, tomando por base o primeiro f, da esquerda para a direita e a regra 3.5 $f \rightarrow f$ aplicada aos dois f's restantes, a *string* resultante passa a ser: $[fFfF]Ff]f$. Na segunda linha e segunda coluna, da Tabela 10, detalha-se melhor essa representação genotípica, onde se associa cada f a um possível neurônio, percebe-se o surgimento de um novo referenciado por n_9. A representação fenotípica apresentada na terceira coluna dessa mesma linha mostra o neurônio em vermelho, sendo esse processo executado na iteração 12 **(IT.12)**. Dessa forma, uma conexão entre **N6** e **N9** é estabelecida. Finalizando o processo, aplica-se a regra 3.4 $f \rightarrow n$ a *string* $[fFfF]Ff]f$, o significado da regra é tornar os possíveis neurônios, definitivos na arquitetura gerada. Com isso, a *string* resultante torna-se: $[FnFn]Fn]n$, conclui-se assim na iteração 12 **(IT.12)** o processo de construção da rede iterada que desenha as conexões da camada intermediária ao segundo neurônio da camada de saída.

Após as iterações 11 e 12 **(IT.11)** e **(IT.12)** percebe-se que nem todos os neurônios gerados na camada intermediária, como é o caso dos destacados em verde nas Tabelas 9 e 10, possuem uma conexão para um neurônio na camada de saída, como é o caso de **N5** na Tabela 9 e **N7** na Tabela 10. Esses neurônios serão eliminados no processo de construção da rede iterada final. A Figura 40 ilustra a arquitetura modular da rede final após a conclusão de todas as iterações apresentadas anteriormente. Cada módulo é representado pelos neurônios remanescentes das ramificações 1 e 2 geradas nas iterações 6 e 9 **(IT.6)** e **(IT.9)**.

O Sistema-L apresentado na Tabela 6 possibilita a obtenção de conexões recorrentes, um exemplo simplificado é ilustrado na Tabela 11, onde se apresenta o processo de desenho de conexões recorrentes direcionadas da camada de saída para a camada de contexto. Adotando-se como ponto de partida a primeira ramificação obtida na iteração 11 **(IT.11)** na Tabela 9. Considerando a representação genotípica $[FfFf]FfF]Ff]f$, apresentada na primeira linha e segunda coluna da Tabela 6.9 e aplicando-se a regra 3.4, $f \rightarrow n$, a todos os $f's$ dessa *string* com exceção do segundo, adotando-se como referência a *string* anterior, obtém-se como resultado $[FnFf]FnF]Fn]n$. Após isso, aplica-se a regra 3.3 a *string* anterior, obtendo-se a seguinte representação genotípica $[FnFfFf]FnF]Fn]n$. Percebe-se na segunda linha e

terceira coluna da Tabela 11 que surge uma conexão do neurônio n_8 para um neurônio na camada de contexto, denominado N_{c1}. Para a obtenção da conexão de N_{c1} aos neurônios da camada intermediária, é necessário aplicar a regra 3.6 a *string* obtida na iteração 13 **(IT.13)**, obtendo-se como resultado $[FnFfFfB]FnF]Fn]n$. O que conduz a representação fenotípica apresentada na primeira linha da Tabela 11**,** onde aparece uma conexão partindo de Nn_{c1} na direção da região delimitadora do módulo formado pelos neurônios **(N3,N4,N5),** essa conexão é representada pelo **B** na *string* anterior, o que significa que **NC1** é conectado a todos os neurônios do bloco **(N3,N4,N5)**. Essa etapa, é finalizada na iteração 14 **(IT.14**), com a aplicação da regra 3.4, que transforma os neurônios provisórios em permanentes.

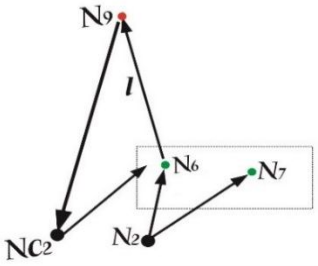
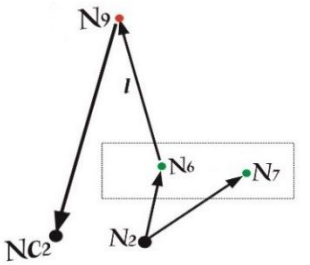
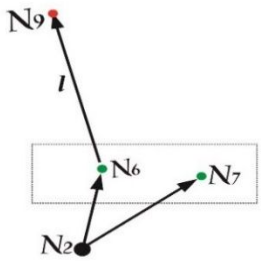
Tabela 11 – Processo de Construção da Rede Iterada, direcionada da camada saída para a camada de contexto, considerando recorrências, a partir da primeira ramificação obtida na Tabela 9 (IT.11)

REGRAS	GENÓTIPO	FENÓTIPO	IT
R3.4	[F n F n F n B] F n F] F n] n		IT.14
R3.2	[F n F f F f B] F n F] F n] n		
R3.3	[F n F f F f] F n F] F n] n		IT.13
R3.4	[F n F f] F n F] F n] n		
inicio	[F f F f] F f F] F f] f		IT.11

Fonte: o autor (2016).

Na Tabela 12 é apresentado um processo análogo de construção de recorrências direcionada da camada saída para a camada de contexto, adotando-se como ponto de partida a segunda ramificação obtida na Tabela 10 **(IT.12)**. Entretanto, agora a realimentação é gerada partindo de neurônio **N9** para o neurônio de contexto n_{c2} e desse na direção da região delimitadora do módulo formado pelos neurônios **(N6,N7)**. Essa conexão é representada pelo **B** na *string* ilustrada na primeira linha e segunda coluna da Tabela 12, o que significa que **NC2** é conectado a todos os neurônios do bloco **(N6,N7)**. Após as iterações **14 (IT.14) e 16 (IT.16)** obtém-se uma rede recorrente, com a mesma arquitetura da apresentada na Figura 6, com duas recorrências partindo de **N8** e **N9** em direção aos neurônios N_{c1} e N_{c2} e desses em relação à camada intermediária. Para cada conexão específica existe um peso associado e para cada **B** que aparece na *string* associa-se um vetor de pesos, que significa que um único neurônio está conectado a todos os neurônios de um bloco, conexões redundantes são eliminadas.

Tabela 12 – Processo de Construção da Rede Iterada, direcionada da camada saída para a camada de contexto, considerando recorrências, a partir da segunda ramificação obtida na Tabela 6.5 (IT.12)

REGRAS	GENÓTIPO	FENÓTIPO	IT
$R_{3.4}$	$[\,F\,n\,F\,n\,F\,n\,B\,]\,F\,n\,]\,n$ (E1)		IT.16
$R_{3.2}$	$[\,F\,n\,F\,f\,F\,f\,B\,]\,F\,n\,]\,n$ (E2)		
$R_{3.3}$	$[\,F\,n\,F\,f\,F\,f\,]\,F\,n\,]\,n$ (E2)		IT.15
$R_{3.4}$	$[\,F\,n\,F\,f\,]\,F\,n\,]\,n$ (E2)		
inicio	$[\,F\,f\,F\,f\,]\,F\,f\,]\,f$ (E2)		IT.12

Fonte: o autor (2016).

O Sistema-L apresentado na Tabela 6 é um modelo generalizado que permite gerar arquiteturas complexas diretas e recorrentes, por exemplo, com um número "n" de camadas. Para ilustrar a construção de uma rede com duas camadas ocultas, a seguir detalham-se as etapas de construção da rede obtida na Tabela 13.

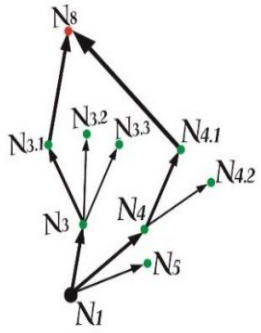
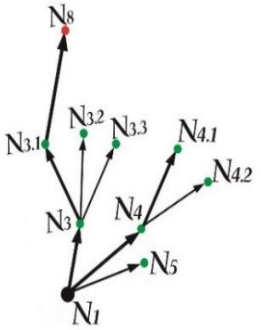
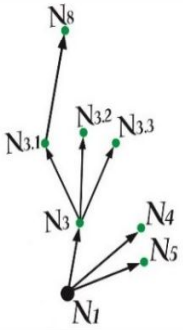
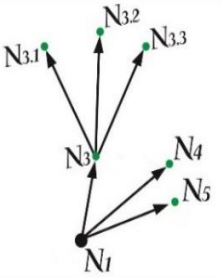
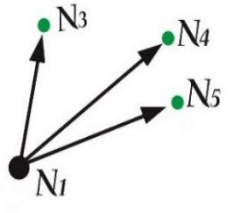
Na Tabela 13, o processo de desenvolvimento da rede tem início com a estrutura que foi obtida e armazenada na iteração 6 **(IT.6)**, entretanto uma nova regra de produção, denominada **R5**, foi incorporada a gramática, $f \rightarrow f*$, onde o significado do asterisco(*****) é recuperar e reescrever a estrutura armazenada em **(IT.6)**, na Tabela 7, e que nessa iteração é representada pela *string* $[Ff]Ff]Ff]$. Esse procedimento é similar aos métodos de reescrita de arestas e nós apresentados por Prusinkiewicz e Lindenmayer (2004).

O processo de desenvolvimento agora continua partindo-se da *string* **[Ff]Ff]Ff]f** . A regra 5**, f→f*** significa **f→f**$[Ff]Ff]Ff]$**,** onde o asterisco denota que a *string* $[Ff]Ff]Ff]$ armazenada anteriormente é recuperada para uso. Aplicando-se a regra 5 considerando apenas o primeiro **f** da *string* **[Ff]Ff]Ff]f** a string resultante será **[Ff[Ff]Ff]Ff]]Ff]Ff]f.** O que representa o fenótipo mostrado em (IT.7). Uma nova ramificação é formada (Reuso de estrutura fenotípica) a partir de N3. Aplicando-se a regra (3.2) ao segundo **f** da *string* anterior **[Ff[Ff]Ff]Ff]]Ff]Ff]f** resultará em: **[Ff[FfFf]Ff]Ff]]Ff]Ff]f**, O que representa o fenótipo mostrado em (IT.8), dessa forma uma nova conexão surge partindo de N3.1 em direção a N8.

Aplicando-se as regras (5) e (3.2) a *string* anterior **[Ff[FfFf]Ff]Ff]]Ff]Ff]f** , resultará em **[Ff[FfFf]Ff]Ff]]Ff]FfF]Ff]]Ff]f.** Percebe-se que surge uma nova ramificação a partir de N4 e uma conexão de N4.1 para N8 em **(IT.10).**

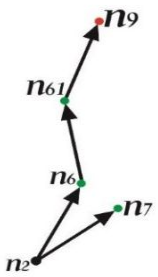
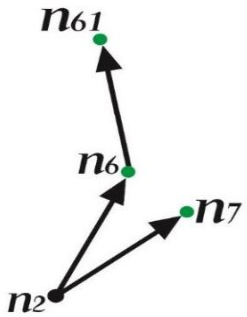
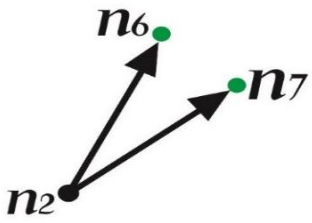
Na Tabela 14 ilustra-se um procedimento análogo ao anterior para expansão da ramificação a partir da estrutura obtida na iteração 9 **(IT.9)** e que é apresentada na terceira linha da Tabela 14. Na Figura 41 apresenta-se a rede final obtida após as etapas ilustradas nas Tabelas 13 e 14. Percebe-se na Figura 41 que só permanecem os neurônios, que fazem parte de um caminho que une um neurônio da camada de entrada para outro da camada de saída. Para prever neurônios com diferentes funções de ativação, será necessário, modificar as regras de produção descritas na Tabela 6.

Tabela 13 – Processo de Construção da Rede Iterada, com duas camadas ocultas, a partir da primeira ramificação obtida na Tabela 7 (IT.6)

REGRAS	GENÓTIPO	FENÓTIPO	IT
$R_{3.2}$	$n_3\ n_{31}\ n_8\ n_{32}\ n_{33}\ n_4\ n_{41}\ n_5\ n_{42}\ n_6\ n_1$ $[F\,f\ [F\,f\ F\,f\ ]F\,f\ ]F\,f\ ]]F\,f\ [F\,f\ F\ ]F\,f\ ]]F\,f\ ]f$ $E_1\ E_3\ E_3\ E_3\ E_3E_1\ E_4\ E_4\ E_4E_1\ E_1$		IT.10
R_5	$n_3\ n_{31}\ n_8\ n_{32}\ n_{33}\ n_4\ n_{41}\ n_{42}\ n_5\ n_1$ $[F\,f\ [F\,f\ F\,f\ ]F\,f\ ]F\,f\ ]]F\,f\ [F\,f\ ]F\,f\ ]]F\,f\ ]f$ $E_1\ E_3\ E_3\ E_3\ E_3E_1\ E_4\ E_4\ E_4E_1\ E_1$		IT.9
$R_{3.2}$	$n_3\ n_{31}\ n_8\ n_{32}\ n_{33}\ n_4\ n_5\ n_1$ $[F\,f\ [F\,f\ F\,f\ ]F\,f\ ]F\,f\ ]]F\,f\ ]f\ ]f$ $E_1\ E_3\ E_3\ E_3\ E_3E_1\ E_1\ E_1$		IT.8
R_5	$n_3\ n_{31}\ n_{32}\ n_{33}\ n_4\ n_5\ n_1$ $[F\,f\ [F\,f\ ]F\,f\ ]F\,f\ ]]F\,f\ ]F\,f\ ]f$ $E_1\ E_3\ E_3\ E_3\ E_3E_1\ E_1\ E_1$		IT.7
inicio	$n_3\ n_4\ n_5\ n_1$ $[F\,f\ ]F\,f\ ]F\,f\ ]f$ $E_1\ E_1\ E_1\ E_1$		IT.6

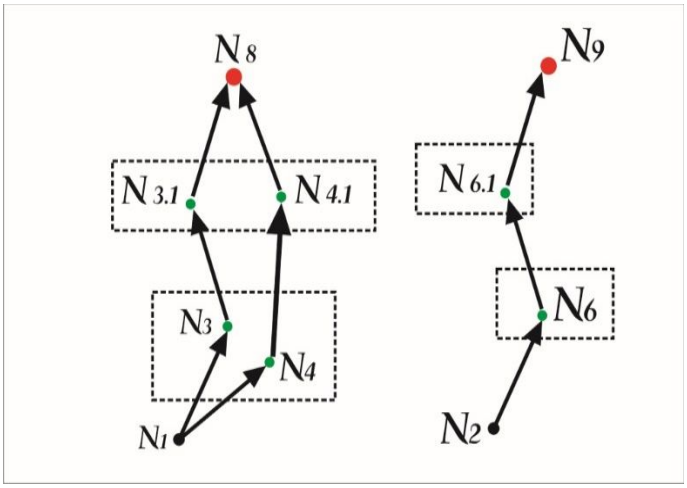
Fonte: o autor (2016).

Tabela 14 – Processo de Construção da Rede Iterada, com duas camadas ocultas, a partir da segunda ramificação obtida na Tabela 8 (IT.9)

REGRA	GENÓTIPO	FENÓTIPO	IT
R3.1	$n_6\ n_{61}\ n_9\ n_7\ n_2$ $[\,F\ f\ [\,F\ f\ F\ f\ ]]\ F\ f\ ]\ f$		IT. 11
R5	$n_6\ n_{61}\ n_7\ n_2$ $[\,F\ f\ [\,F\ f\ ]]\ F\ f\ ]\ f$ $E_2\ E_6\ E_6\ E_2\ E_2$		IT. 10
INICIO	$n_6\ n_7\ n_2$ $[\,F\ f\ ]\ F\ f\ ]\ f$ $E_2\ E_2\ E_2$		IT. 09

Fonte: o autor (2016).

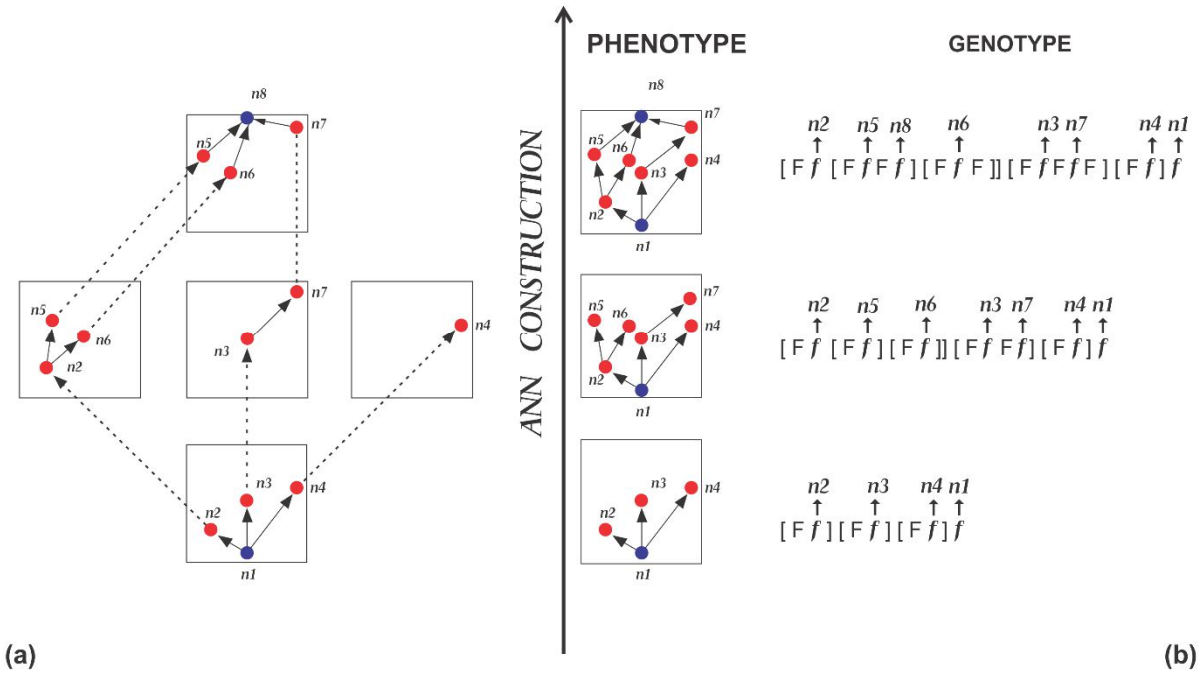
Figura 41 – Arquitetura da Rede Iterada gerada após a execução dos procedimentos ilustrados nas Tabelas 13 e 14



Fonte: o autor (2016).

O ECI utilizado pelo *ADEANN*, apresentado na secção 6.3.2 gera RNAs com um número variável de neurônios no intervalo [X,Y], onde X e Y são computados pelas equações 41 e 42. De uma maneira desmembrada, a Figura 42a ilustra a geração incremental de uma RNA parcialmente conectada, mostrando cada fase do processo de desenvolvimento. Inicialmente, o algoritmo iterativamente define a localização dos neurônios da camada oculta e suas conexões a partir dos neurônios da camada de entrada. Em seguida, determina-se a localização dos outros neurônios ocultos e suas conexões. Finalmente, ele determina a localização dos neurônios de saída e suas conexões a partir dos neurônios ocultos. O processo de construção da RNA é ilustrado na Figura 42b. Todo o processo segue uma propriedade biológica importante denominada princípio de desenvolvimento hierárquico (utilizando-se composição recursiva de subestruturas). A pesquisa através do espaço de busca é restrita a topologias RNA funcionais. No Anexo C mostram-se alguns exemplos de mapeamentos entre genótipos e fenótipos.

Figura 42 – Processo de Construção de uma rede iterada



Fonte: o autor (2016).

6.3.2 Extração de regras com algoritmo genético

Essa secção é dedicada ao processo de extração de regras realizado pelo AG. Essa etapa é realizada antes da descrita na secção 6.3.1. Como o *ADEANN* é um sistema altamente biologicamente inspirado, nessa secção, apresenta-se um ECI inspirado na codificação genética

do ADN. Além disso, partindo-se do pressuposto de que o cérebro humano manipula um conjunto de redes recorrentes com funções especializadas, onde as recorrências são importantes para o entendimento de elementos cognitivos como a memória a curto prazo. Ampliou-se a capacidade do Sistema-L, proposto na secção anterior, para que o mesmo pudesse gerar RNRs. Essas, são mais próximas das redes neurais biológicas do que as redes neurais diretas, que não são as mais adequadas para se explicar a aprendizagem e os processos adaptativos. RNRs, que são capazes também de realizarem processamentos complexos, tais como a simulação de sistemas dinâmicos.

Em adição, com o objetivo de aproximar o AG dos processos evolucionários biológicos. Considerou-se que os genes dos cromossomos (sequencias de ADNs hipotéticos) codificam uma receita (regras de produção de um Sistema-L descrito na secção 6.3.1 e ilustradas na Tabela 6. As regras recursivas mostradas nessa tabela conduzem todas as etapas do desenvolvimento dos neurônios, conforme destacado na secção 6.1.

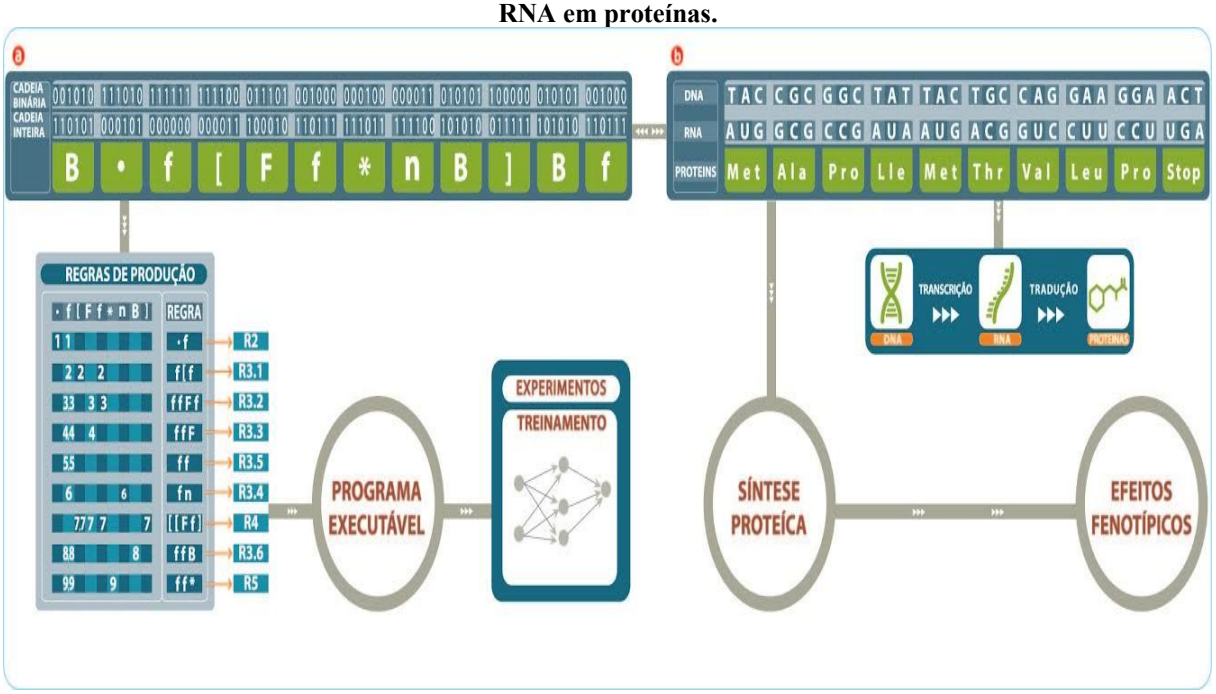
Dawkins (2004) enfatiza que o gene biológico só tem algum significado quando são traduzidos por meio da síntese de proteínas em "regras de crescimento" para o desenvolvimento dos órgãos. Entretanto, os geneticistas não conhecem a fórmula genética completa que conduz esse processo de desenvolvimento (DAWKINS, 2004).

Dawkins (2004) explica que, a nível biológico, o ADN não é um modelo pronto, mas pode ser visto como uma receita, ou seja, em vez de definir a estrutura de um organismo, o ADN fornece um conjunto de instruções que coordenam como o organismo irá se desenvolver ao longo do tempo.

No processamento genético biológico Figura 43b, o ADN é transcrito em ácido ribonucleico (ARN) e o ARN é traduzido em proteínas. As proteínas são derivadas a partir de uma sequência de aminoácidos codificadas por "códons" (grupos de três nucleotídeos do ADN selecionados entre U,G, A e C). A Tabela 15 mostra os produtos da tradução de diferentes códons, por exemplo o códon UUU é traduzido em fenilalanina (Phe) e UUA em Leucina (Leu). Na Figura 43(b), a proteína é formada por uma sequência de aminoácidos começando por metionina (Met) e terminando com prolina (Pro). A síntese de proteínas dispara todos os estágios do desenvolvimento neural (efeitos fenotípicos), tal como mostrado na Figura 43b. O fluxo de informação é ADN→Transcrição→ARN→Translação→Proteína, que é conhecido como o dogma central da biologia.

Os elementos do alfabeto Σ={. , *f* , **F, n, [,], *, B}** do Sistema-L, descrito na secção 6.3.1 são ilustrados em negrito na Tabela 15, os mesmos se constituem uma metáfora do código genético (a mensagem contida no ADN). Cada sequência de dois bits representa um nucleotídeo, por exemplo, o conjunto (00,01,10,11) simboliza (U,C,A,G) no código genético original. Da mesma forma, seis *bits* representam três nucleotídeos, ou seja, (000000,011111) simboliza (UUU,CGG). A seguir apresenta-se uma função Extração de Regras com o AG, a qual imita a transcrição do ADN e a translação do ARN, de acordo com o que foi descrito no parágrafo anterior. O fluxo de informação metafórico no modelo artificial é *String* binária→transcrição→*String* Inteira→Regras de produção, ver Figura 43a esquerda.

Figura 43 – (a) No processo análogo artificial, uma *string* binária é transcrita em uma *string* inteira e essa é transladada em regras de produção de um Sistema-L, (b) Transcrição do ADN em ARN e translação do RNA em proteínas.



Fonte: o autor (2016).

As etapas 1 e 2 da função Extração de Regras de Produção com o AG imitam o processo de transcrição do ADN em ARN, como mostrado na Figura 43b, essas etapas são repetidas para todos os indivíduos na população do AG. Todas as *strings* inteiras obtidas após a etapa 4 desse algoritmo, ver Figura 43a, são armazenadas na matriz 'D' e avaliadas na etapa 5. A etapa 5 faz a translação das *strings* inteiras em regras de produção válidas do Sistema-L proposto na secção 6.3.1. As etapas 5, 5.1 e 5.2 imitam o processo de translação do RNA em proteínas. Sendo essas etapas repetidas para todos os indivíduos na população.

A Figura 43a, a esquerda, ilustra o processo de extração de regras de produção (translação de uma *string* inteira em regras de produção válidas) para um indivíduo da população. Nessa figura, o processo de transcrição produz a *string* inteira **B.f[Ff*nB]Bf.** Após isso, o algoritmo procura, para cada indivíduo da população a menor *string* contendo todas as regras de produção válidas, sendo nesse caso a *string* **.f[Ff*nB]**. Após encontrar a menor *string* na Figura 43a nós identificamos, em que posições da mesma as regras são encontradas. Por exemplo a regra 2 (**.→f**) é encontrada nas posições 1 e 2 da string **.f[Ff*nB]**. A regra 3.1 (**f→[f) é** encontrada nas posições 2, e 3 e 5. A sequência ordenada de caracteres (regras de produção válidas extraídas da *string* mínima **.f[Ff*nB]**) são análogas a sequência de aminoácidos que se juntam para formar as proteínas, Figura 43b. Onde a síntese de proteínas inicia, conduz e controla todos os estágios do desenvolvimento biológico. Como o algoritmo 28 (Quadro 3) começa a leitura dos cromossomos em diferentes posições e em ambas as direções, o nível de paralelismo implícito do AG aumenta, aliviando o problema da escalabilidade.

Tabela 15 – Código Genético a partir da perspectiva do *ARNm*, na mesma tabela a metáfora do *ADN*

	00 (U)	01 (C)	10 (G)	11 (A)	
00 (U)	**f** (UUU) Phe	**F** (UCU) Ser	**n** (UAU) Tyr	**.** (UGU) Cys	00 (U)
00 (U)	**n** (UUC) Phe	**.** (UCC) Ser	**f** (UAC) Tyr	**F** (UGC) Cys	01 (C)
00 (U)	**F** (UUA) Leu	**f** (UCA) Ser	**B** (UAA) Stop	**f** (UGA) Stop	10 (A)
00 (U)	**[** (UUG) Leu	**n** (UCG) Ser	**[** (UAG) Stop	* (UGG) Trp	11 (G)
01 (C)	**f** (CUU) Leu	**]** (CCU) Pro	**n** (CAU) His	* (CGU) Arg	00 (U)
01 (C)	* (CUC) Leu	**F** (CCC) Pro	**f** (CAC) His	**F** (CGC) Arg	01 (C)
01 (C)	**]** (CUA) Leu	**f** (CCA) Pro	* (CAA) Gln	**[** (CGA) Arg	10 (A)
01 (C)	**f** (CUG) Leu	* (CCG) Pro	**B** (CAG) Gln	**]** (CGG) Arg	11 (G)
10 (A)	* (AUU) lle	**]** (ACU) Thr	**n** (AAU) Asn	**f** (AGU) Ser	00 (U)
10 (A)	**f** (AUC) lle	**B** (ACC) Thr	**f** (AAC) Asn	**B** (AGC) Ser	01 (C)
10 (A)	**F** (AUA) lle	**[** (ACA) Thr	**B** (AAA) Lys	**n** (AGA) Arg	10 (A)
10 (A)	* (AUG) Met	**f** (ACG) Thr	* (AAG) Lys	**]** (AGG) Arg	11 (G)
11 (G)	**]** (GUU) Val	**[** (GCU) Ala	**F** (GAU) Asp	**n** (GGU) Gly	00 (U)
11 (G)	**n** (GUC) Val	**B** (GCC) Ala	**[** (GAC) Asp	**.** (GGC) Gly	01 (C)
11 (G)	**f** (GUA) Val	**]** (CGA) Ala	**B** (GAA) Glu	**F** (GGA) Gly	10 (A)
11 (G)	**B** (GUG) Val	**f** (GCG) Ala	* (GAG) Glu	**[** (GGG) Gly	11 (G)

Fonte: o autor (2016).

Quadro 3 – Algoritmo 28: Extração de Regras de Produção com o AG

1.Alocar dinamicamente uma matriz de inteiros ***'B'*** *com dimensão* ***[I x G]****, onde* ***I*** *é o número de indivíduos da população e* ***G*** *o número de genes desejados, atribuir randomicamente a cada posição da matriz* ***(i,g)*** *valores* ***0*** *ou* ***1****.*

2.Obter uma matriz complementar da matriz 'B'(trocando-se 0 por 1 e 1 por 0) e com a mesma dimensão.

3. Alocar dinamicamente uma matriz de caracteres ***'D'*** *com dimensão* ***[I x (G/6)]****, onde* ***I*** *é o número de indivíduos da população e* ***G*** *o número de genes desejados.*

4.Para cada cromossomo (individuo) da população (cada linha da matriz binária 'B', leia uma sequência de 6 bits, começando do primeiro. Cada grupo de seis bits deve ser convertido para um símbolo do alfabeto definido pela gramática (Sistema-L descrito na secção 6.3.1 e de acordo com a Tabela 15). A ordem de leitura da Tabela 15 é a seguinte: para determinar a sequência de bits que codifica o caractere ***'n'****, seguem-se os seguintes passos: determina-se qual das quatro linhas da esquerda corresponde aos dois primeiros bits, por exemplo* ***(00)****, então se escolhe a coluna de acordo com os dois bits do meio, por exemplo* ***(10)*** *e finalmente escolhe-se uma linha da direita usando os dois últimos bits, por exemplo* ***(00)****, ou seja, o caractere* ***'n'*** *é codificado por* ***001000 e 111111 codifica ' ['****. A Figura 43 ilustra um exemplo de conversão de binários em caractere. Para cada posição (i',g') da matriz 'D', atribua esses valores convertidos de binário para caracteres. Cada linha na matriz 'D' é representada por uma "string" inteira, tal como ilustrado na Figura 43(a).*

5.Para cada indivíduo (linha) da Tabela ***'D'*** *obtida no passo 4, encontrar todas as "substrings" que codificam uma regra de produção válida, lendo a partir do primeiro caractere e continuando a leitura em posições contiguas ou não.*

5.1.Rejeitar todos os caracteres até que a substring $.f[Ff \circ nB]$*, seja encontrada.*

5.2.Repetir o passo 5 para todos os indivíduos da população, começando a ler cada linha da matriz 'D' a partir de outras posições do começo para o fim e vice e versa. Para os indivíduos da população que não possuírem regras de produção válidas será atribuído zero ao fitness.

Fonte: o autor (2016).

Coletivamente essas regras formam uma espécie de programa executável que conduz e controla as etapas do desenvolvimento dos neurônios. Uma vez que um conjunto de regras de produção válidas tenha sido determinado para cada linha da matriz 'D', o desenvolvimento neuronal apresentado na secção 6.3.1 é iniciado. A população resultante de redes neurais é treinada e a aptidão é calculada pela equação 29. Os indivíduos são classificados de acordo com a aptidão e selecionados para cruzamentos e mutações sendo, os processos (extração de regras válidas, reescrita das regras, mapeamento genótipo → fenótipo, treinamento da rede

neural, atribuição de fitness, seleção dos indivíduos para aplicação dos operados genéticos) repetidos para todos os indivíduos da população e por um número "n" de gerações.

6.3.3 Avaliação do *Fitness*

A aptidão avaliada pelo *MSE* e pelo número de neurônios da camada intermediária da rede neural treinada, equação 29. O *MSE* mede o valor esperado do quadrado da distância entre o valor predito e o real, ou seja, o mesmo mede a qualidade do preditor. O *ADEANN* ajusta os pesos e a topologia da rede neural para minimizar o MSE no treinamento e teste da rede neural. O *ADEANN* visa não só minimizar o MSE de uma rede neural na fase de treinamento, mas também conduza a rede neural a uma boa capacidade de generalização e predição.

$$\boldsymbol{Fitness\ 1 = A1/(MSE) + A2/(NCIH)} \tag{28}$$

$$\boldsymbol{Fitness\ 2 = [\{exp(-MSE)\ X\ exp(-NCIH)\} + {}^{1}/_{(MSE\ X\ NCIH)}]} \tag{29}$$

Onde: MSE= Erro médio quadrático no teste da RNA, **NCIH**=Número de neurônios na camada intermediária.

No *ADEANN*, propõem-se um mecanismo de penalização que privilegia redes neurais econômicas. Com esse objetivo, nós comparamos duas funções de aptidão dadas pelas equações 28 e 29. No capítulo 7 discutem-se os resultados obtidos para essas duas equações. A função de aptidão dada pela equação 29, foi a mais adequada para implementar esse mecanismo de penalização. Um resultado satisfatório para essa função é um valor que representará uma arquitetura econômica com boa capacidade de generalização, pequeno erro, facilidade de implementação e com um número variável de neurônios no intervalo [X,Y], onde X e Y são computados pelas equações 41 e 42.

6.3.4 Mecanismo de seleção

Neste trabalho, adota-se a seleção por torneio (ST) para selecionar os indivíduos para a reprodução. Essa estratégia é uma das mais utilizadas em AGs (MILLER; GOLDBERG,

1996). Na ST uma associação de acasalamento consiste dos vencedores do torneio. Usualmente a aptidão média é maior na associação de acasalamento do que na população. A diferença de aptidão entre a associação de acasalamento e a população reflete a intensidade de seleção (I), dado pela equação 30. É esperado que a seleção por torneio melhore a aptidão de cada geração subsequente (OLADELE; SADIKU, 2013). Para calcular a intensidade de seleção (I), deve-se resolver a equação 30. A solução aproximada é dada pela equação 31.

Aqui *x* e *y* são os valores de aptidão da população e *t* é o tamanho do torneio. A Tabela 16 ilustra como a intensidade de seleção varia com o tamanho do torneio *t*.

$$I = \int_{-\infty}^{\infty} tx \frac{1}{2\pi} e^{\frac{-x^2}{2}} \left(\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}} d_y\right)^{t-1} dx \tag{30}$$

$$I \approx \sqrt{2(\ln(t) - \ln(\sqrt{4.44 \ln(t)}))} \tag{31}$$

$$LD(t) = t^{-1/t-1} - t^{-t/t-1} \tag{32}$$

Tabela 16 – Relação entre tamanho do torneio e intensidade de seleção

Tamanho do Torneio (t)	1	2	3	5	10	30
Intensidade de Seleção (I)	0	0.56	0.85	1.15	1.53	2.04

Fonte: o autor (2016).

Na seleção por torneio, um valor alto do tamanho do torneio *t,* conduz a uma perda de diversidade (LD) (OLADELE; SADIKU, 2013). Como fica evidente a partir da equação 32, aproximadamente 50% da população é perdida para tamanho do torneio (t=5), o que não é desejável, pois cruzamentos em populações homogêneas não produzem soluções novas. Por conseguinte, o tamanho do torneio definido como t=3, significando que três cromossomos competem entre si, e o melhor cromossomo dentre esses três é selecionado para reprodução.

6.3.5 Operações de cruzamento e mutação

Por meio da recombinação dos códigos genéticos de dois genitores, o operador de crossover produz duas soluções em uma região ainda não visitados do espaço de busca. Para cada par de genitores, o operador de crossover é aplicado de acordo com a probabilidade de

cruzamento ***Pc***. Existem, uma série de estudos, principalmente de natureza empírica, que têm mostrado os benefícios de operadores de cruzamento que envolvem pontos de cruzamento múltiplos (DE JONG; SPEARS, 1992; OLADELE; SADIKU, 2013). O *ADEANN* usa operadores de cruzamento multiponto, escolhidos aleatoriamente, proporcional ao comprimento cromossomo dividido por seis e multiplicado pela taxa de Crossover ***Cr***. Após a aplicação do operador de cruzamento, o operador de mutação, que varia os valores dos genes de um cromossomo, é aplicado a cada indivíduo da prole resultante. Ao explorar novos espaços de solução, visa prevenir que o AG fique preso em um ótimo local. O *ADEANN* usa vários pontos de mutação escolhidos aleatoriamente com uma probabilidade ***Pm*** proporcional ao comprimento cromossomo dividido por seis e multiplicado pela taxa de mutação ***Mr***.

6.4 CONSIDERAÇÕES FINAIS

Nesse capítulo apresentou-se um novo algoritmo Neuroevolutivo que incorpora alguns aspectos de inspiração biológica e que considera que os princípios de organização e desenvolvimento do sistema nervoso é um processo construtivo conduzido pela informação genética codificada no ADN. Dessa forma, o *ADEANN* possibilita evoluir redes neurais usando essas ideias como técnicas computacionais de projeto. Na presente metodologia foram unidas três metáforas biológicas, dentre elas o AG que evolui um conjunto de regras de produção de um **Sistema-L** capazes de gerarem arquiteturas de **RNAs** hierárquicas, modulares, diretas e recorrentes capazes de simularem problemas estáticos e dinâmicos. No capítulo seguinte apresentam-se os resultados de simulação que confirmam as potencialidades do *ADEANN*.

CAPÍTULO 7 – RESULTADOS DE SIMULAÇÕES PARA PROBLEMAS ESTÁTICOS E DINÂMICOS

"Não existem métodos fáceis para resolver problemas difíceis"

René Descartes

7.1 INTRODUÇÃO

Nesse capítulo, é avaliado o desempenho do *ADEANN* em evoluir arquiteturas de redes neurais visando simular tarefas de classificação e predição de séries temporais. Antes de conduzir os experimentos formais, foi necessário definir os parâmetros que foram utilizados pelo *ADEANN*. Os mesmos foram definidos empiricamente, baseados em vários experimentos utilizando-se o problema do *XOR* como base de testes. Diferentes configurações desses parâmetros foram testadas de acordo com a Tabela 17. Na secção 7.4, discutem-se como os mesmos foram obtidos. Para validar o *ADEANN* e fazer comparações com outros ANEs em problemas de classificação, testou-se o sistema utilizando-se cinco bancos de dados escolhidos dentre vários disponíveis no repositório da Universidade da Califórnia (NEWMAN et al., 1998). A Tabela 22 apresenta uma breve descrição dos mesmos, para cada um, são apresentados, o número de entradas, classes e padrões. Finalmente, comparou-se o desempenho do *ADEANN* com dois outros métodos, estatísticos de predição (*ARIMA* e *UCM*), com outro ANE denominado *ADANN* proposto por (DONATE; SANCHEZ; DE MIGUEL, 2012) e um software de predição denominado (*Forecast Pro*®), descritos na secção 7.10.1.

7.2 PROBLEMAS DE CLASSIFICAÇÃO

Segundo Faceli et al. (2011), classificação consiste em identificar objetos, através da extração de suas características, a partir de dados sobre o objeto. Não necessariamente o objeto necessita ser concreto. Sendo possível classificar padrões comportamentais, sonoros ou numéricos. Formalmente, um problema de classificação pode ser definido da seguinte maneira:

Considere um conjunto de exemplos de treinamento composto por pares $(\mathbf{x_i}, \mathbf{c_j})$, no qual x_i representa um vetor de atributos de entrada que descrevem um exemplo e $\mathbf{c_j}$ sua classe associada, encontrar uma função que mapeie cada $\mathbf{x_i}$ para sua classe associada $\mathbf{c_j}$, tal que $\boldsymbol{i = 1,}$

2, ..., n, em que ***n*** é o número de exemplos de treinamento, e ***j = 1, 2, ..., m***, em que ***m*** é o número classes do problema.

O algoritmo de classificação tem por finalidade encontrar alguma correlação entre atributos e uma classe, de modo que o processo de classificação possa usá-la para predizer a classe de um exemplo novo e desconhecido.

7.3 MÉTRICAS PARA AVALIAR A CLASSIFICAÇÃO

Com o objetivo de avaliar o desempenho (acurácia) absoluto e relativo de diversos modelos de classificação, a seguir apresentam-se algumas métricas utilizadas para avaliar os resultados obtidos pelas redes neurais:

a) ***MSE*****:** *Mean Square Error*, ou Erro Quadrático Médio, na qual *n* é o número de amostras, x_i é o valor fornecido pelo classificador para a i-ésima amostra e $\bar{x}$ é a média dos valores de todas as amostras:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2 \tag{33}$$

b) ***RAE***: *Mean Relative Error*. Erro relativo médio. Mais uma medida frequentemente utilizada para estimar a qualidade de um classificador, que deve ser minimizada. Pode ser calculada utilizando a Equação 34, na qual *n* é o número de amostras, x_i é o valor fornecido pelo classificador para a i-ésima amostra, $\bar{x}$ é a média dos valores de todas as amostras, e x_i^* é o valor correto para que deve ser fornecido pelo classificador, para a amostra em questão:

$$RAE = \frac{\sum_{i=1}^{n}\left|x_i - x_i^*\right|}{\sum_{i=1}^{n}\left|x_i - \bar{x}\right|} \tag{34}$$

c) **EC**: Esforço Computacional. Rivero et al. (2010 apud AHMADIZAR et al., 2015) discutem que o tempo de UCP necessário para resolver um problema não pode ser usado para comparar adequadamente os resultados de diversos métodos, pois esse tempo depende da plataforma computacional utilizada, métodos e habilidades de programação e outros fatores. Portanto o tempo de UCP, não é uma métrica prática para ser utilizada quando se comparam resultados com outros previamente obtidos. Dessa forma, nós usamos o esforço computacional para medir o tempo computacional. Calculou-se o esforço computacional de cada algoritmo considerando-se o esforço para treinamento de cada indivíduo (número de épocas de treinamento), o tamanho da população e o número de gerações:

$$\boldsymbol{ECmédio} = \sum_{j=1}^{G}(EC[j])/\mathrm{G} \tag{35}$$

Onde: G=número de gerações, P=número de indivíduos por população, ep[i]=número de épocas para treinar um indivíduo [i] e $EC[j] = \sum_{i=1}^{P} ep[i]/P$; j=1 até G.

d) ***ACA***: *Average Classification Accuracy*, ou Precisão de Média de Classificação:

$$\boldsymbol{ACA}(\%) = \frac{\textit{Predições Corretas}}{\textit{número de amostras } (n)} * 100\% \tag{36}$$

Em adição as métricas apresentadas anteriormente as seguintes, foram utilizadas para avaliar o desempenho dos classificadores utilizados no problema 5 (secção 7.7.5);

e) ***DR*: ***Detection Rate***, Taxa de detecção:

$$\boldsymbol{DR}(\%) = \frac{\textit{Total de ataques detectados}}{\textit{Total de Ataques}} x100 \tag{37}$$

f) ***FP*: ***False Positives***, Falsos positivos:

$$\boldsymbol{FP}(\%) = \frac{\textit{Total de processos normais classificados como intrusão}}{\textit{Total de processos Normais}} x100 \tag{38}$$

g) ***TP*:** *True Positive*, Verdadeiros Positivos:

$$TP(\%) = \frac{Total\ de\ processos\ normais\ classificados\ como\ normal}{Total\ \ de\ processos\ Normais} x100 \quad (39)$$

h) **PRE**: *Precision*: Precisão:

$$PRE = \frac{TP}{TP+FP} \quad (40)$$

7.4 DEFINIÇÃO DE PARÂMETROS

O problema do *XOR* foi utilizado, apenas, como base de testes para definição dos parâmetros padrões utilizados pelo *ADEANN*. Os experimentos foram conduzidos em um computador V14T-5470-A60 core i7 com 8GB de memória. Os resultados obtidos após vinte experimentos, utilizando os parâmetros da Tabela 17, são ilustrados na Tabela 18. Na mesma observa-se que nos últimos quinze experimentos (quinze últimas linhas da Tabela 18), a função aptidão dada pela Equação 29 automaticamente direciona o processo de busca realizados pelo AG para redes neurais com um pequeno número de neurônios na camada oculta e que possuem boa capacidade de generalização. Todas as redes neurais retornadas pelo processo de busca, nos quinze últimos experimentos, possuem dois neurônios na camada de entrada, dois na camada escondida e um na camada de saída, o que pode ser descrito como uma RNA=(2,2,1). O que nem sempre é possível com o uso da Equação 28, pois o processo de determinação dos coeficientes A1 e A2 utilizando tentativa e erro consome muito tempo. Além disso, não existe a garantia de que o processo de busca retornará uma rede neural pequena. Os resultados dos primeiros cinco experimentos, sumarizados na Tabela 18 (cinco primeiras linhas), mostram isso, todas as redes neurais retornadas pela busca possuem 8,7,6 ou 5 neurônios na camada escondida. Considerando-se número de neurônios obtidos na camada intermedirária e o *MSE* obtido na fase de generalização, e a aptidão a cada RNA, calcula pelas equações 28 e 29. Os resultados do melhor (nonagésimo) e pior (primeiro) experimentos são destacados em negrito na Tabela 18.

Na Tabela 21, são apresentados a faixa de valores dos parâmetros utilizados nos experimentos de classificação e predição de séries temporais. As simulações realizadas para o *XOR* serviram apenas para definição dessas faixas de valores.

Tabela 17 – Parâmetros utilizados pelo Algoritmo Genético nos 20 experimentos realizados para o problema do *XOR*

Exp / Fit	Gerações / Indivíduos	(Tamanho Cromossomo)	Pontos de (Cruzamento, Mutação)	Taxa de Cruzamento	Taxa de Mutação	Regras Válidas (%)
1/Eq1	50 x 30	180	(9,3)	0.6	0.01	6.67%
2/Eq1	50 x 30	216	(10,3)	0.6	0.01	16.67%
3/Eq1	50 x 30	252	(12,4)	0.6	0.01	25%
4/Eq1	60 x 30	252	(12,4)	0.9	0.01	23.33%
5/Eq1	50 x 20	276	(13,5)	0.6	0.01	20.28%
6/Eq2	50 x 30	180	(18,3)	0.3	0.01	9%
7/Eq2	50 x 30	180	(18,3)	0.6	0.01	13.40%
8/Eq2	50 x 30	180	(18,3)	0.9	0.01	11.33%
9/Eq2	50 x 20	216	(32,28)	0.6	0.01	22.33%
10/Eq2	50 x 20	216	(32,28)	0.9	0.01	21.53%
11/Eq2	50 x 30	252	(37,33)	0.6	0.01	28.67%
12/Eq2	60 x 30	252	(37,33)	0.9	0.01	26.73%
13/Eq2	50 x 30	300	(45,40)	0.6	0.01	51.20%
14/Eq2	60 x 30	300	(45,40)	0.9	0.01	41.93%
15/Eq2	50 x 30	360	(54,48)	0.6	0.01	50.73%
16/Eq2	50 x 30	360	(54,48)	0.9	0.01	54.33%
17/Eq2	50 x 30	390	(58,52)	0.6	0.01	63.33%
18/Eq2	50 x 30	390	(58,52)	0.9	0.01	63.40%
19/Eq2	100 x 30	516	(77,68)	0.9	0.01	96.67%
20/Eq2	100 x 30	516	(77,68)	0.6	0.01	90%

Fonte: o autor (2016).

A partir da Tabela 20, nós observamos que em 100 execuções, o *NEAT* (Stanley, 2002) encontra uma arquitetura de RNA para o *XOR* em 32 gerações (4755 RNAs avaliadas, σ=2.553), o número médio de neurônios na camada intermediária foi μ=2.35. O décimo terceiro experimento, ilustrado na Tabela 18, produziu a solução mais próxima com σ=3.31 e μ=1.65 em 50 gerações. O *ADEANN* teve uma precisão média de classificação similar ao DENN (ILONEN; KAMARAINEN; LAMPINEN, 2003), porém tem uma melhor capacidade de generalização (*MSE*$\leq$0.000024). *ADEANN* resolveu facilmente o problema do *XOR* direcionando a busca para RNAs pequenas. O número de neurônios e conexões foram próximos da solução ótima, considerando-se que para o problema do *XOR*, existe uma solução com um neurônio na camada intermediária e com duas conexões partindo direto dos dois neurônios da camada de entrada para o neurônio da camada de saída. O número total de RNAs treinadas nos 20 experimentos destacados na Tabela 17 foram 30100. O que comprova que o espaço de busca é grande, mesmo para problemas simples, como é o caso do *XOR*.

A faixa de valores permitidos para neurônios na camada intermediária, para esse problema, foi variável entre (2 e 8), esses valores são gerados pelo *L-System* por meio da variável **NR**, gerada aleatoriamente, discutida anteriormente no capítulo 6 e que define o número de neurônios por ramificação que surgem a partir dos neurônios da camada de entrada. O nosso método de codificação indireto, apresentado na secção 6.3.2, possibilita gerar RNAs com número de neurônios na faixa [X,Y], cujos valores são dados pelas equações 41 e 42. Essa faixa de valores é limitada, pois o modelo de desenvolvimento artificial apresentado na secção 6.3.1 gera ramificações a partir dos neurônios da camada de entrada em direção a camada oculta com no mínimo um neurônio, esse valor é definido pela variável aleatória **NR,** que é inicializada aleatoriamente.

$$X=[NENT + NENT*£(NR)min + NSAI] \quad (41)$$

$$Y=[NENT + NENT*£(NR)max + NSAI] \quad (42)$$

Onde: NENT=número de entradas da RNA, £(NR)min=valor mínimo do valor aleatório (NR), £(NR)max=valor máximo do valor aleatório (NR), NSAI= Número de Saídas da RNA e NR=número de neurônios por ramificação.

Taxas de cruzamento com valores de 0.6 e 0.9 foram adequadas para todos os experimentos. Para cromossomos menores que 300 bits a taxa de cruzamento com valor de 0.6 possibilitou obter maior percentual de regras de produção válidas e para cromossomos maiores que 300 bits uma taxa de cruzamento com valor 0.9 foi mais adequada, na maioria dos casos. **A taxa de mutação** de 0.01 mostrou-se adequada em todos os experimentos. Os valores para as taxas de cruzamento e mutação são consistentes com os da literatura. Lin, Lee e Hong (2003) discutem que valores típicos para a taxa de mutação devem estar no intervalo de [0.01, 0.08] enquanto para a taxa de cruzamento devem estar na faixa de [0.001, 0.05]. Quanto maior o tamanho do cromossomo, maior o número de pontos de corte (cruzamento, mutação) utilizados. Os mesmos foram essenciais para a obtenção de uma maior percentagem de regras de produção válidas após cada da geração, como ilustra a Tabela 17 (%Regras válidas). Ao realizar os cruzamentos e mutações, o material genético foi melhorando ao longo do processo evolutivo, possibilitando obter maior diversidade de RNAs.

Tabela 18 – Resultados obtidos por simulação para o problema do *XOR* utilizando-se a aptidão dada pela equação 28 (cinco primeiros experimentos, Eq1) e equação 29 (para os experimentos restantes, Eq2)

Exp/Fit	Melhor Rede	MSE na Generalização	Número Médio de Neurônios (μ)	Variância Número Neurônios (σ^2)	Desvio Padrão do Número de Neurônios (σ)	Coeficiente de Variação Número de Neurônios (CV= $\frac{\sigma}{\mu}$)
1/Eq1	**(2,8,1)**	**0.000046**	**6.50**	**2.25**	**1.50**	**0.23**
2/Eq1	(2,7,1)	0.000049	4.80	5.36	2.32	0.48
3/Eq1	(2,5,1)	0.000048	4.80	2.96	1.72	0.36
4/Eq1	(2,6,1)	0.000032	3.86	3.27	1.81	0.47
5/Eq1	(2,6,1)	0.000029	6.00	3.75	1.94	0.32
6/Eq2	(2,2,1)	0.000076	5.67	6.89	2.62	0.46
7/Eq2	(2,2,1)	0.000075	3.20	0.56	0.75	0.23
8/Eq2	(2,2,1)	0.000051	4.50	4.75	2.18	0.48
9/Eq2	(2,2,1)	0.000056	6.00	4.29	2.07	0.35
10/Eq2	(2,2,1)	0.000056	3.71	5.92	2.43	0.65
11/Eq2	(2,2,1)	0.000056	3.89	0.56	3.65	0.49
12/Eq2	(2,2,1)	0.000055	4.36	2.96	1.72	0.39
13/Eq2	(2,2,1)	0.000053	3.31	2.71	1.65	0.50
14/Eq2	(2,2,1)	0.000053	3.77	4.02	2.01	0.53
15/Eq2	(2,2,1)	0.000056	3.69	4.59	2.14	0.58
16/Eq2	(2,2,1)	0.000056	3.71	3.50	1.87	0.50
17/Eq2	(2,2,1)	0.000053	3.79	2.59	1.61	0.42
18/Eq2	(2,2,1)	0.000053	5.10	4.19	2.05	0.40
19/Eq2	**(2,2,1)**	**0.000024**	**5.28**	**4.13**	**2.03**	**0.39**
20/Eq2	(2,2,1)	0.000025	4.15	4.72	2.17	0.52

Fonte: o autor (2016).

Usou-se a função sigmoide $\varphi(x) = \frac{1}{1+e^{-kx}}$, como função de ativação em todos os neurônios das RNAs. Pelos resultados de simulação apresentados na Tabela 18, a menor RNA que apresentou o menor erro médio quadrático **(MSE)** na generalização foi obtida no experimento 19. No qual em média uma rede solução teve μ=*5.28* neurônios escondidos, desvio padrão $\sigma = 2.03$ e coeficiente de variação ***CV***=$\frac{\sigma}{\mu} = 0.39$. Essa RNA teve a seguinte especificação (neurônios na camada de entrada, neurônios na camada intermediária, neurônios na camada de saída RNA= (2,2,1) e obteve erro médio quadrático ***MSE***=*0.000024*. A Figura 46a ilustra a mesma destacando-se os pesos. O total de RNAs treinadas nos 20 experimentos foi de 30100. A maior RNA, que melhor conseguiu generalizar para o mesmo ***MSE*** anterior, teve a seguinte especificação (2,8,1), a mesma é ilustrada na Figura 46b. A Figura 45 ilustra a evolução dos melhores aptidões e aptidões médias para o problema do *XOR*, no experimento

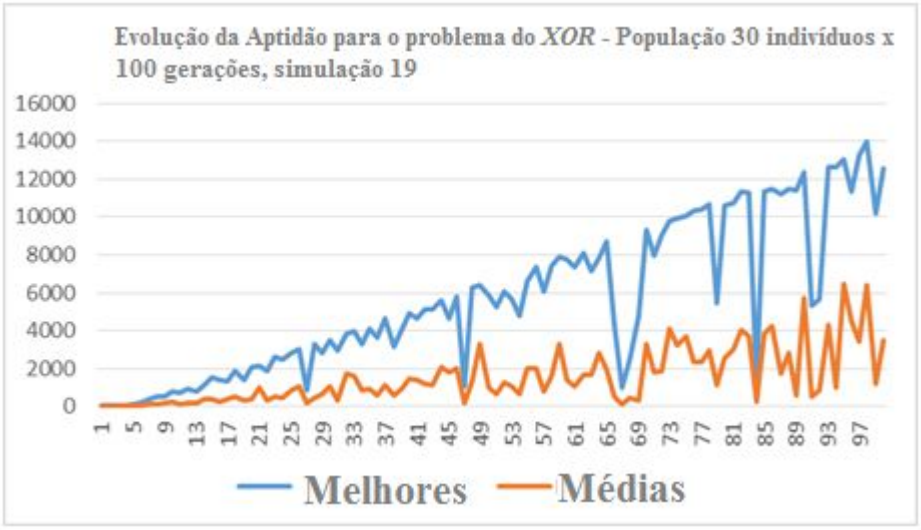
19 com a aptidão avaliada pela equação 29 e a Figura 44, no experimento 5, com a aptidão avaliada pela equação 28.

A Tabela 19 ilustra um relatório que classifica as RNAs obtidas na melhor geração do melhor experimento (19/Eq2) para o problema do *XOR*. Conclui-se que a aptidão dada pela Equação 29 avalia como melhores as redes neurais que possuem não apenas uma topologia pequena, mas também leva em consideração uma boa capacidade de generalização (mínimo *MSE*). O *ADEANN* resolve o problema do *XOR* sem problema em fazer as arquiteturas pequenas. O número de conexões está próximo do ideal, considerando que a menor RNA para o problema do *XOR,* tem um único neurônio na camada oculta. *ADEANN* é muito consistente em encontrar uma solução. Os parâmetros mais adequados para configuração do *ADEANN* são ilustrados na Tabela 21.

Figura 44 – Melhores Aptidões e Aptidões Médias para o problema do *XOR* para 1800 indivíduos treinados e avaliados pela equação 28

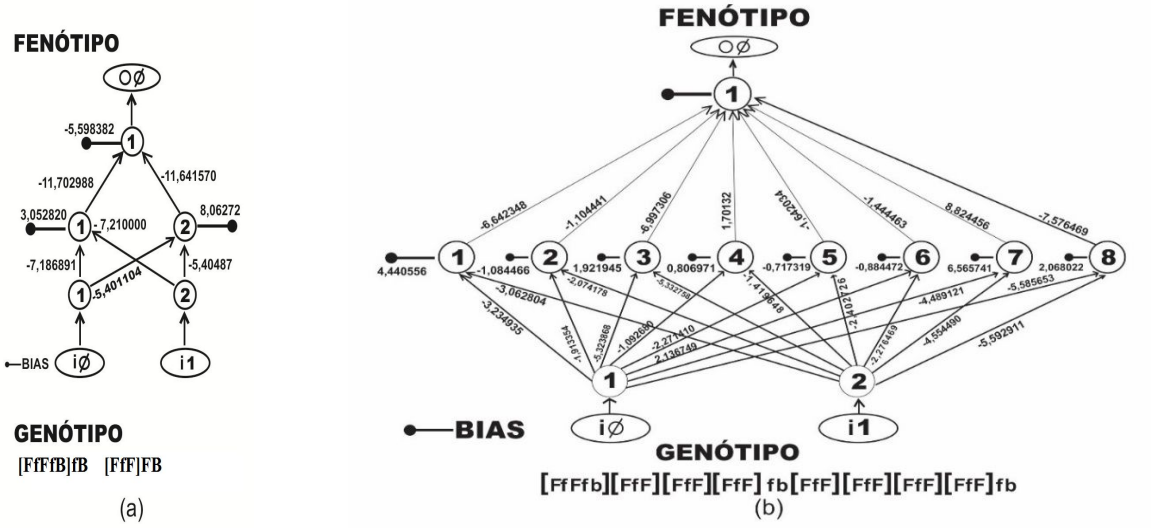
Fonte: o autor (2016).

Figura 45 – Melhores Aptidões e Aptidões Médias para o problema do *XOR* para 3000 indivíduos treinados e avaliados pela equação 29



Fonte: o autor (2016).

Figura 46 – Menor e maior RNA encontradas, no experimento 5, para o problema do *XOR*



Fonte: o autor (2016).

Tabela 19 – Relatório de RNAs obtidas na melhor geração do experimento 19 para o problema do *XOR*

Indivíduo	Fitness	MSE	Posto	Neurônios na Camada Oculta
17	13985.5439	0.000024	1	2
26	13256.1569	0.000025	2	2
30	13151.1494	0.000025	3	2
18	10886.7193	0.000023	4	3
9	9124.0701	0.000022	5	4
19	8849.6821	0.000023	6	4
29	8646.4842	0.000023	7	4
13	8630.4373	0.000019	8	5
15	8542.7480	0.000020	9	5
28	7582.9939	0.000019	10	6
16	7098.3295	0.000018	11	7
21	7065.2360	0.000018	12	7
11	7025.3559	0.000020	13	6
24	7008.1944	0.000024	14	5
23	6718.8742	0.000019	15	7
25	6618.9136	0.000022	16	6
20	5943.2253	0.000019	17	8
14	5846.7073	0.000019	18	8
22	5735.4986	0.000019	19	8
12	5661.5306	0.000020	20	8
8	5206.7516	0.000024	21	7
10	5145.3511	0.000022	22	8
7	4727.2277	0.000042	23	4
6	2434.5209	0.000103	24	3
5	482.7356	0.000347	25	5
4	157.8053	0.001324	26	4
1	49.7981	0.083420	27	2
3	34.3141	0.005233	28	5
2	5.4452	0.020869	29	8
27	0.0	0.0	30	-

Fonte: o autor (2016).

Tabela 20 – Comparação entre o *ADEANN* e outras metodologias para o problema do *XOR*, onde**: a -** precisão de média de classificação**, b-** número médio de neurônios**, c**-desvio padrão do número de neurônios**, d-** MSE

Problema	NEAT		ADEANN		DENN	
XOR	-	**2.35**	98.94% [a]	3.31[b]	**98.97%**	-
	2.553	-	**1.65**[c]	**0.000024**[d]	-	0.004865

Fonte: o autor (2016).

Tabela 21 – Parâmetros usados nos experimentos

Tamanho da população	Gerações	Taxa de Cruzamento	Taxa de Mutação	Probabilidade de Elitismo
[30-100]	[50-500]	[0.3,0.9]	0.1	2%

Fonte: o autor (2016).

7.5 ESTATÍSTICAS PARA PROBLEMAS DE CLASSIFICAÇÃO

Devido à natureza estocástica do Algoritmo Neuroevolutivo (*ANE*) proposto, deve-se executar o mesmo várias vezes para cada banco de dados seguido de comparações com outros ANEs por meio de testes estatísticos apropriados. Nessa pesquisa nós utilizamos a *técnica k-fold cross-validatio*n para medir a performance do algoritmo. Nessa técnica, os dados são randomicamente divididos em k conjuntos não sobrepostos $\boldsymbol{D_1}$, $\boldsymbol{D_2}$,..., $\boldsymbol{D_k}$. O algoritmo executa k vezes, em cada execução i (1≤i≤k), o mesmo é treinado com $\boldsymbol{D \backslash D_i}$ conjuntos e testado em $\boldsymbol{D_i}$. Cada experimento foi repetido t vezes . Em cada execução do algoritmo, o conjunto de testes é usado para calcular a estimativa de erro para o classificador (Equação 36).

Para avaliar a significância dos resultados obtidos pelo *ADEANN* e outros ANEs, foram realizados testes estatísticos (testes-t com nível de significância α=0.05, descrito no anexo E) considerando três critérios: **C1**-a precisão de classificação média (dado pela Equação 36), **C2**-o esforço computacional (dado pela equação 7.3) e **C3**-o número médio de neurônios na camada intermediária, respectivamente. De maneira similar a Ahmadizar et al. (2015) se não existe diferença estatística significante entre o desempenho de dois ANEs em um dado bando de dados de acordo com o primeiro, segundo, ou terceiro critérios, ambos são recompensados com 1 ponto. Mas, se existe diferença entre ambos de acordo com o primeiro, ou segundo ou terceiro critério, o algoritmo que melhor executa é recompensado com 2 pontos e o outro com

zero. O desempenho global de cada ANE é então calculado somando todos os pontos obtidos em uma comparação pareada em todos os bancos de dados. Para a validação das hipóteses, considerou-se o valor-p, assim: Se o valor-p>α, aceita-se $(H_O: \mu_1 = \mu_2)$, caso contrário se o valor-p≤α rejeita-se H_O e aceita-se $(H_1: \mu_1 \neq \mu_2)$.

Quadro 4 – Algoritmo *k-fold cross-validation*

1.Arranjar os exemplos de treinamento em ordem randômica

2.Dividir os exemplos de treinamento em "k" conjuntos não sobrepostos D_1, D_2,..., D_k. (K pedaços de aproximadamente D/k exemplos cada).

3.Para i=1….k;

Treine o classificador usando todos os exemplos que não pertencem ao conjunto i $(D \backslash D_i)$.

Teste classificador em todos os exemplos no conjunto i (D_i).

Computar n_i, o número de exemplos no conjunto **i** que foram classificados erradamente.

4. Retorne a seguinte estimativa de erro para o classificador:

$$E = \frac{\sum_{i=1}^{k} n_i}{D} * 100\%$$

OBS: Para **t** execuções do algoritmo: $E = \frac{\sum_{i=1}^{t} E_i}{t}$

Fonte: o autor (2016).

Para a realização de cada teste-t verificou-se, por meio do teste de *Shapiro – Wilk* (descrito no anexo F), se as amostras são provenientes de distribuições normais. Um baixo valor-p indica uma distribuição não-normal. Neste estudo, nós utilizamos o nível de significância α = 0.05, assim um valor-p> α indica que a condição de normalidade é atingida, ou seja, a Hipótese nula é aceita $(H_O:$ A amostra provém de uma distribuição Normal). Caso contrário, se o valor-*p*≤α rejeita-se H_O e aceita-se $(H_1:$ A amostra não provém de uma distribuição Normal**)**. Neste caso, a independência dos eventos é óbvia, dado que eles são execuções independentes dos algoritmos com condições iniciais geradas aleatoriamente. Se as

amostras seguem uma distribuição normal, consequentemente seguem uma distribuição (t de *student*).

7.6 DESCRIÇÃO DOS BANCOS DE DADOS

Para validar o *ADEANN* e fazer comparações com outro ANEs, nós selecionamos cinco bancos dos dados do mundo real, usados em problemas de classificação, do repositório da Universidade da Califórnia em Irvine (NEWMAN et al, 1998). A Tabela 22 apresenta uma breve descrição dos mesmos. Na secção 7.7 descrevem-se detalhadamente cada um.

Tabela 22 – Descrição dos bancos de dados utilizados nessa pesquisa

Bancos de Dados	Número de Entradas	Número de Classes	Padrões
1-Breast Câncer I	**15**	**3**	**390**
2-Breast Câncer II	**9**	**2**	**699**
3-Iris Flower	**4**	**3**	**150**
4-Heart Disease	**13**	**2**	**351**
5-KDDCUP'99	**42**	**5**	**125973**

Fonte: o autor (2016).

7.7 COMPARAÇÃO COM OUTROS ALGORITMOS NEUROEVOLUTIVOS

Nesta secção, o desempenho do *ADEANN* é avaliado experimentalmente e comparado com outros proeminentes ANEs referenciados na literatura. A precisão média de classificação das RNAs geradas no conjunto de testes é testada com a técnica *k-fold cross-validation*, a média e o desvio padrão do número de neurônios ocultos e o esforço computacional foram medidos. Os métodos são comparados com outros ANEs pertencentes a três categorias. A primeira categoria **(I)** estudada em nossas comparações incluem algoritmos de treinamento, que evoluem somente os pesos das conexões das RNAs. *G3PCX* (DEB; ANAND; JOSHI, 2002) e *GA* (CANTU-PAZ; KAMMATH, 2005) são dois métodos proeminentes que se enquadram nesta categoria. A segunda categoria **(II)** é composta de algoritmos que evoluem somente a topologia de RNAs e os pesos são otimizados por meio de métodos baseados no gradiente descendente para minimização do erro na saída da rede. Os métodos apresentados por Cantu-Paz e Kammath (2005): método matricial (*MM*), método de poda (*PRM*), gramática

geradora de grafos (*GRM*) e o *GE-BP* de Soltanian et al. (2013) são métodos proeminentes pertencentes a mesma. A terceira **(III)** categoria evolui simultaneamente a topologia e pesos de conexão. Os métodos *GEGA* (AHMADIZAR et al., 2015), *GP* (RIVERO et al., 2010) e *GE* (TSOULOS; GAVRILIS; GLAVAS, 2008) são métodos proeminentes dessa categoria.

7.7.1 Problema 1 – Predição do efeito de uma nova droga no Câncer de Mama *(Breast Câncer I)*

O problema consiste em encontrar um mapeamento análogo entre um conjunto de 15 tumores induzidos experimentalmente e um tumor clínico, baseado em suas reações conhecidas à mesma droga, de forma que sejamos capazes de predizer o efeito de uma nova droga em um tumor clínico após ter estudado os tumores experimentais. As soluções são baseadas na similaridade presumida entre esse tumor e um conjunto de tumores experimentais (ISLAM; YAO; MURASE, 2003). A da Tabela 25 ilustra uma amostra dos dados que foram utilizados nos experimentos, as colunas representam os tumores induzidos experimentalmente (entradas da RNA: T1 a T15). Cada linha da tabela representa o efeito de uma droga (D1 a D15) nos quinze tumores clínicos (T1 a T15), a saída da RNA representa o mapeamento análogo entre o efeito dessa mesma droga em um tumor clínico (TC: 1-com efeito, 0-sem efeito).

A Tabela 23 mostra os parâmetros utilizados no AG para simular o problema de predição de uma nova droga em tumores malignos de câncer de mama. O melhor experimento mostra que o *ADEANN* encontra uma RNA solução para o problema de predição de uma nova droga no câncer de mama em 50 gerações. No melhor experimento, o número médio de neurônios da camada escondida das RNAs geradas foi de $\mu=16.36$, com desvio padrão de $\sigma = 4.73$ e $MSE=0.000004$. Esse experimento apresentou o menor desvio padrão do número de neurônios em relação à média, ou seja, a amostra de RNAs geradas na melhor geração foi mais homogênea **(CV<30%)**, a mesma mostrou boa capacidade de generalização para um MSE **<=0.0001**. A menor RNA obtida nas simulações tem a seguinte especificação (15,14,1), ou seja, com 14 neurônios na camada intermediária apresentou um $MSE = 0.000004$, a mesma mostrou boa capacidade de generalização para um $MSE <=0.0001$. A maior RNA obtida teve a seguinte especificação (15,28,1) e apresentou um $MSE = 0.000004$. Entretanto, como a aptidão, dada pela Equação 29, privilegia as menores RNAs com boa capacidade generalização, a RNA (15,14,1) foi selecionada como a melhor. A Figura 47 ilustra

a topologia da mesma. Os testes realizados para essa RNA (15,14,1), gerada no experimento 28, que apresentou um ***MSE = 0.000004*** são apresentados na Tabela 26. As melhores aptidões e as aptidões médias para o problema de predição de uma nova droga em tumores malignos de câncer de mama obtidos no experimento 28 para 1500 indivíduos treinados são apresentados na Figura 48.

Tabela 23 – Parâmetros utilizados pelo Algoritmo Genético em 4 experimentos do problema do Câncer de Mama

Exp/ Fit	Gerações X Indivíduos	(Tamanho Cromossomo)	Pontos de Corte (Cruzamento, Mutação)	Taxa de Cruzamento	Taxa de Mutação	% Regras Válidas
26/Eq2	50 x 30	300	(45,40)	0.9	0.1	49.80%
27/Eq2	50 x 30	360	(54,48)	0.9	0.1	50%
28/Eq2	**50 x 30**	**390**	**(58,52)**	**0.9**	**0.1**	**70.60%**
29/Eq2	50 x 30	516	(77,68)	0.9	0.1	96.67%

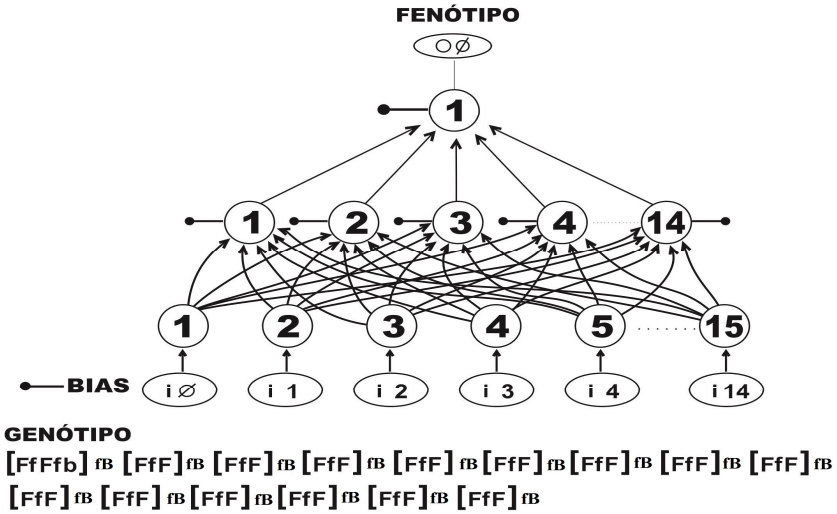
Fonte: o autor (2016).

Tabela 24 – Resultados obtidos nas Simulações para o problema do Câncer de Mama utilizando a aptidão dada pela equação 29

Exp/Fit	Melhor Rede	MSE na Generalização	Número Médio de Neurônios (μ)	Variância Número Neurônios (σ^2)	Desvio Padrão do Número de Neurônios (σ)	Coeficiente de Variação Número de Neurônios (CV= $\frac{\sigma}{\mu}$)
26/Eq2	(15,14,1)	0.000004	21.38	118.23	10.87	0.51
27/Eq2	(15,14,1)	0.000004	21.80	122.83	11.08	0.51
28/Eq2	(15,14,1)	0.000004	16.36	22.41	4.73	0.29
29/Eq2	(15,14,1)	0.000004	18.90	65.47	8.09	0.43

Fonte: o autor (2016).

Figura 47 – Menor RNA encontrada, no experimento 28, para o problema de predição do efeito de drogas no câncer de mama



Fonte: o autor (2016).

Tabela 25 – Conjunto de Treinamento para o problema de predição de uma nova droga em tumores malignos de câncer de mama

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	TC
D1	1.0	1.0	0.5	0.5	1.0	0.5	1.0	1.0	0.5	0.5	0.5	1.0	1.0	1.0	0.5	1.0
D2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0
D3	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0
D4	1.0	1.0	0.5	0.0	0.0	0.5	1.0	0.5	0.5	0.5	0.5	1.0	1.0	0.5	0.5	1.0
D5	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0
D6	1.0	1.0	0.0	1.0	0.0	0.5	1.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.5	1.0
D7	1.0	1.0	0.0	1.0	0.0	0.5	1.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.5	1.0
D8	0.0	1.0	0.0	1.0	0.0	0.5	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.5	1.0	1.0
D9	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.5	1.0	0.0	0.0	1.0	1.0
D10	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0
D11	1.0	1.0	1.0	0.0	0.0	0.5	0.0	0.0	0.5	0.0	0.5	1.0	1.0	0.0	0.5	1.0
D12	1.0	0.5	1.0	0.5	0.0	0.5	0.0	0.0	0.5	0.0	1.0	0.5	1.0	0.5	0.5	1.0
D13	0.0	0.0	1.0	0.0	1.0	0.0	0.5	0.5	0.5	0.0	0.5	1.0	1.0	0.5	0.5	0.0
D14	1.0	1.0	1.0	0.0	0.5	0.5	0.0	0.5	0.0	0.0	0.0	0.5	0.5	0.5	0.0	0.0
D15	1.0	0.5	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0
D16	1.0	0.5	0.5	1.0	1.0	1.0	0.0	0.5	1.0	1.0	0.5	0.5	1.0	0.5	1.0	1.0
D17	1.0	0.0	0.5	1.0	1.0	0.5	1.0	0.5	0.5	1.0	0.5	0.5	0.0	0.5	0.5	1.0
D18	1.0	0.5	1.0	0.5	0.0	0.5	1.0	1.0	0.5	0.0	0.5	1.0	1.0	0.5	0.5	1.0
D19	0.0	0.0	1.0	1.0	0.5	1.0	1.0	1.0	0.5	0.0	0.5	0.5	1.0	0.5	0.5	1.0
D20	1.0	1.0	1.0	0.0	0.0	0.5	0.0	1.0	0.5	0.5	0.5	0.5	1.0	1.0	1.0	0.0
D21	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0
D22	1.0	0.0	0.5	1.0	0.5	1.0	0.0	0.5	1.0	1.0	0.5	0.5	1.0	0.0	1.0	1.0
D23	1.0	0.0	0.0	1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.5	1.0	0.0	0.5	0.5	1.0
D24	1.0	0.5	0.5	0.5	0.0	0.5	1.0	0.5	0.5	0.0	0.5	1.0	0.5	0.5	0.5	1.0
D25	0.0	0.0	0.5	0.5	0.0	0.0	1.0	1.0	1.0	0.5	0.5	0.0	0.0	1.0	1.0	1.0
D26	0.0	0.5	0.5	0.0	0.0	0.5	1.0	0.5	1.0	0.5	0.5	0.0	0.5	0.5	1.0	0.5

Fonte: o autor (2016).

7.7.2 Problema 2 – Detecção do Câncer de Mama (BC)

Para resolução desse problema, utilizaram-se os dados disponíveis no repositório da Universidade da California - Irvine, *UCI Machine Learning Repository*, com informações

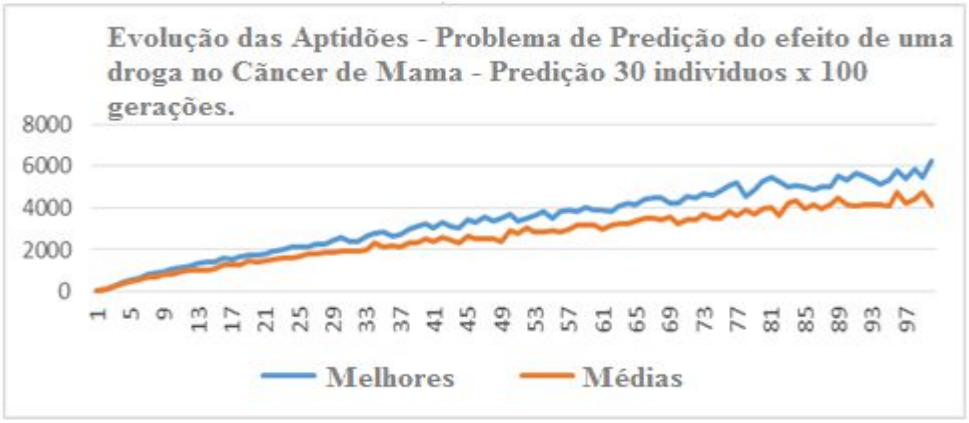
sobre Câncer de Mama. O banco de dados foi criado pelo *Dr. W.H. Wolberg* do Hospital Universitário de *Wisconsin, Madison*. Os dados foram sendo coletadas pelo *Dr. Wolberg*, conforme os casos clínicos tratados foram sendo acumulados cronologicamente. A descrição do banco de dados sobre câncer de mama é apresentada na Tabela 27.

Tabela 26 – Testes Realizados para a RNA (15,14,1) para o problema do Câncer de Mama utilizando o Fitness dado pela equação 2 no experimento 28

Saída Obtida	Saída Desejada	Erro na Saída da Rede
0.999700	1.0	0.000000045
0.998717	1.0	0.000000823
0.999973	1.0	0.000000000
0.999963	1.0	0.000000001
0.999546	1.0	0.000000103
0.998482	1.0	0.000001153
0.996649	1.0	0.000005616
0.997020	1.0	0.000004441
0.007759	0.0	0.000030104
0.999862	1.0	0.000000000
0.999046	1.0	0.000000455

Fonte: o autor (2016).

Figura 48 – Melhores Fitness e Fitness Médio para o problema de predição de uma nova droga em tumores malignos de câncer de mama, experimento 29, para 3000 indivíduos treinados e aptidão dada pela equação 29.



Fonte: o autor (2016).

Tabela 27 – Descrição da Base de Dados sobre câncer de Mama

Total de Instâncias	699	Área de Pesquisa:	Biológicas
Número de Atributos	9	Tipo dos Atributos	Real
Registros Incompletos	16	Data dos Dados	Nov /1995

Fonte: o autor (2016).

Cada instância de dado contém 9 atributos e duas classes. Os valores dos atributos são calculados (valores de 1 a 10) a partir da imagem digitalizada de uma amostra aspirada por meio de uma cânula ou agulha fina da massa de uma mama. Eles descrevem as características

dos núcleos celulares presentes na imagem. Cada instância pertence a uma das 2 classes possíveis: benignos ou malignos, os atributos são descritos na Tabela 28. Embora no total existam 699 medidas individuais (do conjunto de todos os parâmetros), 16 instâncias possuem registros incompletos, sendo que esses foram eliminados, ficando assim o banco de dados com 683 registros. Sendo utilizada a técnica *k-fold cross validation* (k=5) para divisão randômica dos dados. O banco de dados sobre câncer de mama foi utilizado devido à dificuldade inerente da diagnose de câncer desse tipo de doença.

Tabela 28 – Descrição dos Atributos para a base de dados sobre Câncer de mama

1-Espessura dos grupos (Clump Thickness)	Células benignas tendem a ser agrupadas em monocamadas, enquanto que as células cancerosas são muitas vezes agrupadas em multicamadas.
2-Uniformidade de tamanho e 3-forma da célula	As células cancerosas tendem a variar em tamanho e forma. Devido a isso, esses parâmetros são úteis para determinar se as células são cancerosas ou não.
4-Adesão Marginal (Marginal Adhesion)	As células normais tendem a ficar juntas. As células cancerosas tendem a perder essa capacidade. Então perda de adesão é um sinal de malignidade.
5-Tamanho único das células epiteliais (Single Epithelial Cell Size)	Está relacionada com a uniformidade mencionado acima. As células epiteliais que estão significativamente aumentadas pode ser uma célula maligna.
6-Núcleos nus (Bare Nuclei)	Este é um termo usado para núcleos que não são rodeados pelo citoplasma (o resto da célula). São tipicamente vistos em tumores benignos.
7-Suavidade da cromatina (Bland Chromatin)	Descreve uma textura uniforme do núcleo visto em células benignas. Em células cancerosas, a cromatina tende a ser mais grosseira.
8-Nucléolo normal (Normal Nucleoli)	Nucléolos são pequenas estruturas existentes no núcleo. Em células normais, o nucléolo é geralmente muito pequeno, quando visível. Em células cancerosas os nucléolos se tornam mais proeminentes.
9-Mitose	Patologistas podem determinar o grau de um tumor contando o número de mitoses.

Fonte: o autor (2016).

A Tabela 29 compara os resultados do *ADEANN* com o *G3PCX* (DEB; ANAND; JOSHI, 2002), *GA* (CANTU-PAZ; KAMMATH, 2005), o *GEGA* (AHMADIZAR et al., 2015), com os métodos apresentados por (CANTU-PAZ; KAMMATH, 2005): *MM*, *PRM*, *GRM* e o *GE-BP* de Soltanian et al. (2013), *GP* (Rivero et al., 2010) e *GE* (TSOULOS; GAVRILIS; GLAVAS, 2008). Os dados da Tabela 29, usados para as comparações foram obtidos de (AHMADIZAR et al., 2015), na replicação dos experimentos utilizaram-se os mesmos bancos de dados, com os exemplos de treinamento e testes divididos em *k* conjuntos não sobrepostos, sendo o valor de *k* igual para todos algoritmos comparados. Esse mesmo procedimento foi utilizado para os bancos de dados do *Iris Flower* e detecção de doenças cardíacas.

Tabela 29 – Comparação entre o ADEANN e outros métodos para o problema de Classificação do Câncer de Mama

BANCO DE DADOS	GEGA			ADEANN			G3PCX			GA		
Cancer de Mama (I)	96.61		**3.8**	**99.01**[a]		8[b]	98.94		5	98.88		5
	1	0.05	9200	**0**[c]	**0.0003**[d]	25550[e]	-	-	11500	-	-	**8400**
	MM			**ADEANN**			**GRM**			**GE-BP**		
Cancer de Mama (II)	96.77		**-**	**98.52**[a]		8[b]	96.71		5	95.88		**4.6**
	-	-	115000	**0**[c]	**0.0004**[d]	**33247**[e]	-	-	300000	0.8	-	92000
	GEGA			**ADEANN**			**GE**			**GP**		
Cancer de Mama (III)	96.19		**2**	**98.03**[a]		8[b]	96.02		3	96.27		3.3
	0	-	92000	**0**[c]	**0.0003**[d]	**75046**[e]	0.5	-	92000	0.5	-	92000

[a] Precisão média de classificação, [b] número médio de neurônios na camada intermediária, [c] desvio padrão do número de neurônios da camada intermediária, [d] erro médio quadrático na generalização, [e] esforço computacional.

Fonte: adaptado de Ahmadizar et al. (2015) e Rivero et al. (2010).

A Tabela 29 compara os resultados obtidos pelo *ADEANN*, em três repetições utilizando a técnica *k-fold cross validation* (k=5) com outros ANEs pertencentes as três categorias de algoritmos (I), (II) e (III) discutidos na secção 7.7. Nas três repetições, o *ADEANN* encontra uma RNA para o problema do câncer de mama em 50 gerações. Os valores do erro médio quadrático ilustrados na Tabela 29 foram obtidos na fase de generalização. Os demais valores foram obtidos de Ahmadizar et al. (2015), os melhores resultados são

destacados em negrito. Considerando a primeira categoria de ANE (I), o menor e maior esforço computacional são despendidos pelo *GA* (CANTU-PAZ; KAMMATH, 2005) e o *ADEANN* respectivamente. Entretanto, em todas as categorias de ANEs o *ADEANN* obteve maior precisão média de classificação do que os outros métodos (I, II e III) e requer menor esforço computacional do que os métodos pertencentes as categorias (II) e (III). Em relação a primeira categoria de ANE (I), o *ADEANN* obteve menor erro médio quadrático do que o *GEGA* (AHMADIZAR et al., 2015) na fase de generalização.

7.7.3 Problema 3 – Classificação usando o banco de dados *Iris Flower (IF)*

Outro banco de dados utilizado para classificação foi o IF, onde quatro variáveis contínuas são usadas para resolvê-lo. As mesmas são medidas em milímetros, levando em consideração quatro características das flores: largura da pétala (LP), comprimento da pétala (CP), a largura das sépalas (LS) e comprimento sepal (CS). As medidas correspondem a 150 flores pertencentes a três espécies distintas de íris: Tipo 0 - setosa, Tipo 1- virginica e Tipo 2 - versicolor. A Tabela 30 compara os resultados obtidos pelo *ADEANN*, em de três repetições utilizando a técnica *k-fold cross validation* (k=2), com outros ANEs pertencentes as três categorias de algoritmos (I), (II) e (III) discutidos na secção 7.7.

Os resultados ilustrados na Tabela 30 para os algoritmos tipo (I),(II) e (III) foram obtidos de Ahmadizar et al. (2015) e Rivero et al. (2010). Na melhor e na segunda melhor repetição o *ADEANN* encontra uma rede neural solução para o problema do IF em 50 gerações e no terceiro em 100 gerações.

Tabela 30 – Comparação entre o *ADEANN* e outros métodos para o problema de Classificação do *Iris Flower*

BANCO DE DADOS	GEGA	ADEANN	G3PCX	GA
Iris Flower (I)	**96.13** **2.7** 1.1 - 102500	94.11[a] 4.65[b] 47[c] .0006[d] **6255**[e]	89.73 5 - - 10250	88.67 5 - - 8200
	MM	**ADEANN**	**GRM**	**GE-BP**
Iris Flower (II)	92.40 - - - 400000	94.11[a] **4.06**[b] 1.2[c] .0004[d] **37124**[e]	92.93 - - - 1200000	**95.72** 4.5 0.7 - 250000
	GEGA	**ADEANN**	**GE**	**GP**
Iris Flower (III)	**95.70** **2.1** 0.3 - 250000	95.10[a] 4.61[b] 1,5[c] .0003[d] **225113**[e]	94.93 3.5 0.6 - 250000	95.22 8.9 0.5 - 320000

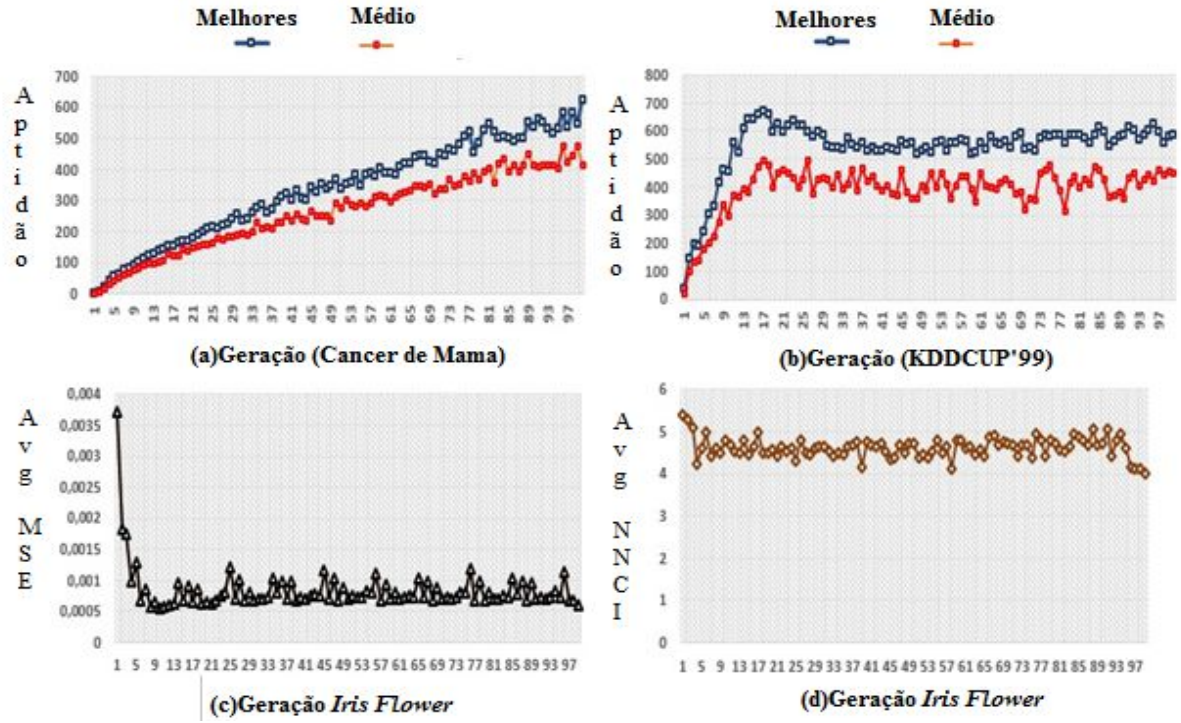
[a]Precisão média de classificação (%), [b]número médio de neurônios na camada intermediária, [c]desvio padrão do número de neurônios da camada intermediária, [d]erro médio quadrático na generalização, [e]esforço computacional.

Fonte: adaptado de Ahmadizar et al. (2015) e Rivero et al. (2010).

Em todas as categorias de ANEs (I), (II) e (III), o *ADEANN* requer o menor esforço computacional que os outros métodos. Em relação a categoria (I) e (III) o *GEGA* atingiu a melhor precisão média de classificação, enquanto que na categoria (II) *GE-BP*. Soltanian et al. (2013) obteve melhor precisão média de classificação. Considerando as categorias (I) e (III) o *GEGA* (AHMADIZAR et al., 2015) tipicamente gera as menores arquiteturas de RNAs, entretanto quando comparado com os ANEs da categoria (II) o *ADEANN* retorna as menores arquiteturas de redes neurais.

As Figuras 49c e 49d plotam o erro médio quadrático e o número de neurônios na camada intermediária durante o processo de busca de redes neurais soluções para o problema de classificação IF. Inicialmente, o *ADEANN* obtém RNAs com valores elevados do erro médio quadrático. Melhores arquiteturas de RNAs são exploradas em gerações subsequentes pela função aptidão dada pela Equação 29. Dessa forma, o erro médio quadrático e o número médio de neurônios da camada intermediária diminuem após algumas gerações e então e permanece dentro dos limites aceitáveis.

Figura 49 – Resultados de simulação e desempenho obtido pelo *ADEANN*



Fonte: o autor (2016).

7.7.4 Problema 4 – Detecção de doenças cardíacas em pacientes

No presente problema, o objetivo é detectar se a doença cardíaca está presente ou não em pacientes estudados. Estes dados correspondem a 13 medições realizadas a partir de 303 pacientes no Hospital de *Cleveland*. A Tabela 31 descreve os campos do banco de dados de doenças cardíacas. A Tabela 32 compara os resultados obtidos pelo *ADEANN*, em de três repetições utilizando a técnica *k-fold cross validation* (k=5), com outros ANEs pertencentes as três categorias de algoritmos (I), (II) e (III) discutidos na secção 7.7.

Os resultados ilustrados na Tabela 32 para os algoritmos tipo (I), (II) e (III) foram obtidos de Ahmadizar et al. (2015) e Rivero et al. (2010). Na melhor repetição o *ADEANN* encontra uma rede neural solução para o problema do *IF* em 50 gerações e na segunda e terceira melhores repetições em 100 gerações. Em todas as categorias de ANEs (I, II, e III) o *ADEANN* requer menor esforço computacional do que todos os métodos e tem a melhor precisão de classificação do que a maioria dos métodos pertencentes a categoria (II) e (IIII). *G3PCX* (DEB; ANAND; JOSHI, 2002) tem a melhor precisão de classificação do que a maioria dos métodos pertencentes a categoria (I). O *GEGA* (AHMADIZAR et al., 2015) gera redes neurais menores do que todos os métodos pertencentes as categorias I e III.

Tabela 31 – Descrição dos atributos para o banco de dados de doenças cardíacas

Atributo	Faixa de Valores
Idade em anos	Contínuo
Sexo do Paciente	1=masculino 0=feminino
Tipo da Angina	1=típica, 2=atípica, 3=dor não anginosa, 4=assintótica.
Pressão arterial de repouso (em mm Hg)	Contínuo
O colesterol em mg / dl	Contínuo
Açúcar no sangue em jejum	0=false, 1=true.
Resultados do eletrocardiograma	0=normal, 1=anormal, 2=hipertrofia ventricular.
Frequência cardíaca máxima atingida	Contínua
Angina Induzida pelo exercício	0=não, 1=sim.
Depressão ST induzido pelo exercício em relação ao descanso	Contínuo
A inclinação do pico do exercício	1=inclinada para cima, 2=sem inclinação, 3=inclinada para baixo.
Número de grandes vasos, coloridos por fluoro cópia	0-3 Contínuo
Normal, defeito fixo, defeito reversível	3,6,7

Fonte: o autor (2016).

Tabela 32 – Comparação entre o ADEANN e outros métodos para o problema de Classificação de Doenças Cardíacas

BANCO DE DADOS	GEGA	ADEANN	G3PCX	GA
Doença Cardiaca (I)	79.55 **2.9** 1.3 - 15200	8733 [a] 12 [b] 0 [c] 0.001 [d] **10831** [e]	**90.42** 5 0 - 19000	89.72 5 - - 15200
	MM	**ADEANN**	**GRM**	**GE-BP**
Doença Cardiaca (II)	76.78 - - - 380000	**87.33** [a] **12** [b] 0 [c] 0.001 [d] **83810** [e]	72.8 - - - 600000	80.80 **2.5** 0.7 - 200000
	GEGA	**ADEANN**	**GE**	**GP**
Doença Cardiaca (III)	82.24 **2.4** 1.0 - 200000	**89.33** [a] **12** [b] 0 [c] 0.001 [d] **145717** [e]	79.60 3.3 1.4 - 200000	80.71 **2.9** 0.3 - 200000

[a] Precisão média de classificação (%), [b] número médio de neurônios na camada intermediária, [c] desvio padrão do número de neurônios da camada intermediária, [d] erro médio quadrático na generalização, [e] esforço computacional.

Fonte: adaptado de Ahmadizar et al. (2015) e Rivero et al. (2010).

As Tabelas 33, 34 e 35 comparam estatisticamente o desempenho do *ADEANN* com outros métodos por meio da abordagem mencionada na secção 7.5. Como pode ser visto, *ADEANN* proporciona o melhor desempenho geral em relação aos métodos das categorias II e III (Tabelas 34 e 35). Considerando apenas a categoria I, *GEGA* (AHMADIZAR et al., 2015) obteve o melhor desempenho global (Tabela 33) e o *ADEANN* obteve melhor desempenho em relação aos algoritmos: *G3PCX* (DEB; ANAND; JOSHI, 2002) e o *GA* (CANTU-PAZ; KAMMATH, 2005).

Tabela 33 – Comparação estatística entre o *ADEANN* e algoritmos pertencentes a Categoria I

BANCO DE DADOS	GEGA	ADEANN	G3PCX	GA
Cancer de Mama (I)	10	6	8	10
Iris Flower (I)	12	14	2	4
Doença Cardiaca (I)	8	8	8	8
Desempenho Global	**30**	28	18	22

Fonte: o autor (2016).

Tabela 34 – Comparação estatística entre o *ADEANN* e algoritmos pertencentes a Categoria II

BANCO DE DADOS	MM	ADEANN	GRM	GE-BP
Cancer de Mama (II)	6	12	6	8
Iris Flower (II)	2	14	2	8
Doença Cardiaca (II)	4	10	0	8
Desempenho Global	12	**36**	8	24

Fonte: o autor (2016).

Tabela 35 – Comparação estatística entre o *ADEANN* e algoritmos pertencentes a Categoria III

BANCO DE DADOS	GEGA	ADEANN	GE	GP
Cancer de Mama (III)	8	12	4	6
Iris Flower (III)	14	10	6	4
Doença Cardiaca (III)	10	12	6	6
Desempenho Global	32	**34**	14	16

Fonte: o autor (2016).

No anexo D são apresentados os resultados detalhados dos testes estatísticos (testes-t) que deram origem as pontuações ilustradas nas Tabelas 33, 34 e 35 e testes de *Shapiro-Wilk*. Os valores numéricos correspondentes, ilustrados nas Tabelas D.1 , D.2 e D.3 do anexo D, são os valores-p associados a cada teste estatístico (teste-t ou *Shapiro-Wilk*). Para a atribuição da pontuação a cada algoritmo de classificação especificado nas Tabelas 29, 30 e 32, consideram-se os critérios apresentados na secção 7.5. Os procedimentos detalhados para realização dos testes estatísticos (teste-t e *Shapiro-Wilk*) são apresentados nos anexos E e F.

7.7.5 Problema 5 – Detecção de intrusão em redes TCP-IP

O quinto banco de dados utilizado em problemas de classificação foi o *KDDCUP'99*, o mesmo é baseado na iniciativa da Agência de Projetos de Pesquisa Avançada de Defesa que forneceu a base de dados para projetista de Sistemas de Detecção de Intrusão. A Tabela 36 apresenta os campos que representam as características clássicas de uma conexão em redes *TCP/IP*, sendo por isso denominadas características intrínsecas de conexões *TCP/IP*. A Tabela 37 ilustra as características baseadas em conhecimento do especialista, as mesmas foram obtidas através da análise de informações contidas, primordialmente na área de dados dos pacotes *TCP*, *UDP* e *IP*. Nesta área, encontram-se normalmente, cabeçalhos de aplicações de nível superior tais como *Telnet*, *FTP*, *http* etc. A Tabela 38 ilustra as características temporais de cada conexão. Os ataques simulados caem em uma das quatros categorias, ilustradas nas Tabela 39.

Tabela 36 – Características intrínsecas de conexões TCP/IP

Nome	Descrição	Tipo
"Duration"	Duração em segundos da conexão	Contínua
"Protocol_type"	Tipo de protocolo usado na conexão, i.e. tcp, udp, etc.	Discreta

“Src_bytes”	Número de bytes enviados da fonte para o destino	Contínua
“Dst_bytes”	Número de bytes enviados do destino para a fonte	Contínua
“Flag”	Status da conexão (normal ou erro)	Discreta
“Land”	1 se conexão é de/para o mesmo host; 0 caso contrário	Discreta
“Wrong_fragment”	Número de fragmentos com erro	Contínua
“Urgent”	Número de pacotes com flag urgente habilitado	Contínua

Fonte: o autor (2016).

Tabela 37 – Características de conexão por conhecimento especialista

Nome	Descrição	Tipo
“Hot”	Número de indicadores chaves (“hot”)	Contínua
“Num_failed_logins”	Tentativas de login sem sucesso	Contínua
“Logged_in”	1 se login efetuado com sucesso; 0 caso contrário	Discreta
“Num_compromised”	Número de condições de “comprometimento”	Contínua
“Root_shell”	1 se shell root foi obtido; 0 caso contrário	Discreta
“Su_attempted”	1 se comando “su root” foi tentado; 0 caso contrário	Discreta
“Num_root”	Número de acessos como root	Contínua
“Num_file_creations”	Número de operações de criação de arquivos	Contínua
“Num_shells”	Número de “shells prompts” obtidos	Contínua
“Num_access_files”	Número de operações em arquivos de controle de acesso	Contínua
“Num_outbound_cmds”	Número de comandos externos em uma sessão ftp	Contínua
“Is_hot_login”	1 se o login pertence a lista “hot”; 0 caso contrário	Discreta
“Is_guest_login”	1 se o login usou a conta guest; 0 caso contrário	Discreta

Fonte: o autor (2016).

Tabela 38 – Características temporais: janela de 2 segundos

Nome	Descrição	Tipo
“Count”	Número de conexões iguais a esta para este mesmo “host” nos últimos 2 segundos	Contínua
“Srv_count”	Número de conexões para o mesmo serviço que o usado nesta conexão nos últimos 2 segundos	Contínua
“Serror_rate”	% de conexões que possuem erro “SYN”	Contínua
“Srv_serror_rate”	% de conexões que possuem erro “SYN” para este serviço	Contínua
“Rerror_rate”	% de conexões que possuem erro “REJ”	Contínua
“Srv_rerror_rate”	% de conexões que possuem erro “REJ” para este serviço	Contínua
“Same_srv_rate”	% de conexões para um mesmo serviço	Contínua
“Diff_srv_rate”	% de conexões para serviços diferentes	Contínua
“Srv_diff_host_rate”	% de conexões deste mesmo serviço para hosts diferentes	Contínua
“Dst_host_count”	Número de conexões com mesmo host de destino que esta	Contínua
“Dst_host_srv_count”	Número de conexões com mesmo host de destino e mesmo serviço que esta	Contínua

(continua)

Tabela 38 – Características temporais: janela de 2 segundos (continuação)

Nome	Descrição	Tipo
"Dst_host_same_srv_count"	% de conexões com o mesmo host de destino e mesmo serviço que esta	Contínua
"Dst_host_diff_srv_rate"	% de conexões com o mesmo host de destino e services diferentes que esta	Contínua
"Dst_host_same_src_port_rate"	%conexões c/mesmo host de destino e mesma porta de origem que a conexão atual	Contínua
"Dst_host_srv_diff_host_rate"	% de conexões para o mesmo serviço vindo de diferentes hosts.	Contínua
"Dst_host_serror_rate"	% de conexões para o mesmo host que o da conexão atual e que possui um erro S0.	Contínua
"Dst_host_srv_serror_rate"	% de conexões para o mesmo host e serviço que o da conexão atual e que possui um erro S0.	Contínua
"Dst_host_rerror_rate"	% de conexões para o mesmo host que apresentem flag RST	Contínua
"Dst_host_srv_rerror_rate"	% de conexões para o mesmo host e serviço da conexão atual que apresentem flag RST.	Contínua

Fonte: o autor (2016).

Tabela 39 – Categorias de ataque

Categoria do Ataque	Descrição
NDS (Negação de Serviço)	Atacante envia um grande número de mensagens que esgote algum dos recursos da vítima, como CPU, memória, banda,etc. Ex: "syn flood"
U2R (User to Root attack)	Atacante acessa o sistema como usuário normal (ganho por : sniffing password, um dicionário local ou engenharia social) e passa a explorar vulnerabilidades para ganhar acesso como root ao sistema. Ex: "buffer overflow"
R2L (Remote to local attack)	Ocorre quando um atacante tem a habilidade de enviar pacotes para uma máquina através da rede, mas não tem uma conta nessa máquina e explora alguma vulnerabilidade para ganhar acesso local como usuário da máquina. Ex: "guessing password"
Probing	É uma tentativa de reunir informações sobre uma rede de computador com o propósito de burlar os controles de segurança. Ex: "port scanning"

Fonte: o autor (2016).

Nós reduzimos o número de atributos do banco de dados *KDDCUP'99* de 42 para 19. Alguns atributos que tinham valor único ou zero foram eliminados dentre eles: *num-outbound-*

cmds, Land, wrong-fragment, Urgent, Hot, num-compromissed, num-root, num-file-creation, num-access-files, diff-srv-rate, dst-host-same-srv-rate, dst-host-srv-diff-host-rate, dst-host-rerror-rate, dst-host-srv-serror-rate. Além disso, foram eliminados atributos com alto valor de correlação, convencionou-se chamar atributos fortemente correlacionados aqueles que possuíam coeficientes de correlação maiores ou iguais a 0.8. Atributos altamente correlacionados influenciam um ao outro e trazem pouca informação, então não é interessante ter atributos correlacionados nesse problema. Por essa razão os seguintes atributos foram eliminados: *serror-rate, same-srv-rate, srv-serror-rate, dst-host-srv-serror-rate, rerror-rate, srv-rerror-rate, srv-count.*

Outros atributos, foram normalizados para valores no intervalo [0,1] : *Count, num-failed-logins, dst-host-srv-rerror-rate, num-shells e dst-host-count.* A normalização é necessária para que se tenha atributos com a mesma ordem de magnitude. A Taxa de Detecção e Falsos Positivos (FP) foram usados para estimar o desempenho dos classificadores neurais, Equações 37 e 38.

A Tabela 40 mostra os parâmetros utilizados no AG para simular o problema de Detecção de Intrusão em redes *TCP-IP* usando o banco de dados *KDDCUP'99*. O melhor experimento **(1/Eq2)** mostra que o *ADEANN* encontra uma rede neural solução para o problema de detecção de intrusão em 100 gerações. No melhor experimento, o número médio de neurônios da camada escondida das RNAs geradas foi de μ***=25.66***, com desvio padrão de $\sigma = 15.36$ e ***MSE = 0.000166***. A menor rede neural que mostrou boa capacidade de generalização para um ***MSE* <=0.001,** possui a seguinte especificação (20,19,1), ou seja, com 19 neurônios na camada intermediária apresentou um ***MSE = 0.000166.*** A Figura 50 ilustra a arquitetura da mesma. A maior RNA obtida teve a seguinte especificação (20,76,1) e apresentou um ***MSE = 0.000230.*** Entretanto como a aptidão, dada pela equação 29, privilegia as menores RNAs com boa capacidade generalização, a RNA (20,19,1) foi classificada como a melhor. O pior experimento foi o quinto (**5/Eq2**), conforme ilustrado na Tabela 41, a melhor rede nesse experimento não obteve boa capacidade de generalização (***MSE = 0.000809)*** em relação as demais redes neurais obtidas nos demais experimentos.

Tabela 40 – Parâmetros utilizados pelo Algoritmo Genético em 5 experimentos do problema de detecção de intrusão usando o *KDDCUP'99*

Exp/ Fit	Gerações X Indivíduos	(Tamanho Cromossomo)	Pontos de Corte (Cruzamento, Mutação)	Taxa de Cruzamento	Taxa de Mutação	% Regras Válidas
1/Eq2	100 x 30	516	(77,68)	0.9	0.1	96.10%
2/Eq2	100 x 30	390	(58,52)	0.9	0.1	83.33%
3/Eq2	100 x 30	360	(54,48)	0.9	0.1	60.50%
4/Eq2	100 x 30	300	(45,40)	0.9	0.1	45.63%
5/Eq2	100 x 30	516	(77,68)	0.9	0.1	90%

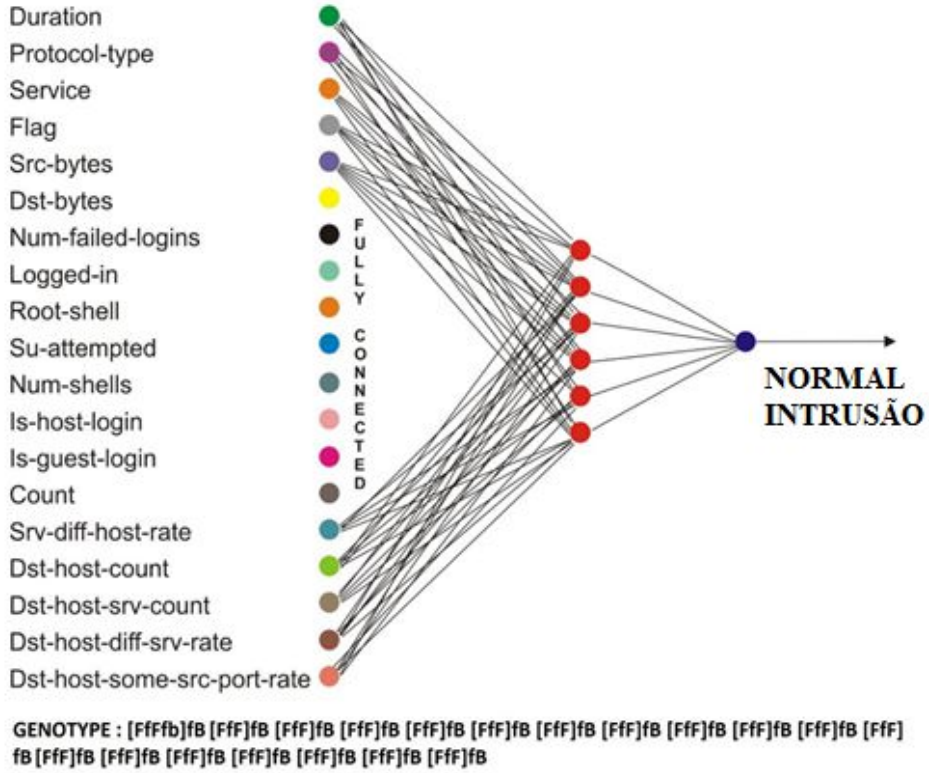
Fonte: o autor (2016).

Tabela 41 – Resultados obtidos nas Simulações para o problema de detecção de intrusão usando o *KDDCUP'99* utilizando o Fitness dado pela equação 29

Exp/Fit	Melhor Rede	EMQ na Generalização	Número Médio de Neurônios (μ)	Variância Número Neurônios (σ^2)	Desvio Padrão do Número de Neurônios (σ)	Coeficiente de Variação Número de Neurônios (CV= $\frac{\sigma}{\mu}$)
1/Eq2	**(20,19,1)**	**0.000166**	**25.66**	**235.95**	**15.36**	**0.6**
2/Eq2	(20,19,1)	0.000175	27.60	191.04	13.82	0.5
3/Eq2	(20,19,1)	0.000170	36.00	475.79	21.81	0.61
4/Eq2	(20,19,1)	0.000172	25.93	218.21	14.77	0.57
5/Eq2	**(20,19,1)**	**0.000809**	**29.07**	**344.44**	**18.56**	**0.64**

Fonte: o autor (2016).

Figura 50 – A melhor arquitetura retornada pelo processo de busca e atributos de entrada utilizados nas simulações para o problema de classificação usando o *KDDCUP'99*



Fonte: o autor (2016).

A Tabela 42 apresenta um relatório das redes neurais soluções retornadas pelo processo de busca e classificadas pelo fitness dado pela Equação 29, na melhor geração do experimento 1.

As Figuras 49a 49b, mostram a melhor aptidão e a aptidão média para os problemas do câncer de mama e para o *KDDCUP'99*. No início do processo evolutivo, o *ADEANN* gera redes neurais com baixa aptidão. Permitindo que a função aptidão, dada pela equação 29, explore melhores topologias em gerações subsequentes, o que ocasiona um aumento das aptidões dos melhores indivíduos e da média da população. Os resultados estatísticos para outros métodos são também sumarizados na Tabela 43. O *ADEANN* fornece uma taxa detecção mais precisa que os demais métodos para ambas as classes de conexões normal e intrusão (97.67%, 97.79%) respectivamente. Além disso, os valores para falsos positivos mantiveram-se dentro de limites aceitáveis de 5%, os valores obtidos para FP(%) foram 3.4% e 1.82% para as classes normal e intrusão respectivamente.

Tabela 42 – Relatório das RNAs obtidas na melhor geração do experimento 1 para o problema de detecção de intrusão usando o *KDDCUP'99*

Indivíduo	Fitness	EMQ	Posto	Neurônios na Camada Oculta
24	301.0893	0.000166	1	19
27	274.0356	0.000182	2	19
9	259.4156	0.000193	3	19
29	259.4156	0.000193	4	19
4	258.7411	0.000193	5	19
19	254.7179	0.000187	6	20
15	247.8550	0.000202	7	19
28	246.3843	0.000203	8	19
18	243.7600	0.000205	9	19
23	239.6513	0.000209	10	19
11	237.5290	0.000211	11	19
10	232.0944	0.000205	12	20
7	231.7228	0.000216	13	19
24	228.2874	0.000190	14	22
26	227.3073	0.000220	15	19
30	222.9335	0.000195	16	22
21	220.3378	0.000216	17	20
2	218.4657	0.000218	18	20
16	218.2475	0.000229	19	19
6	217.0995	0.000219	20	20
5	200.4611	0.000217	21	22
13	189.7516	0.000203	22	25
22	164.1013	0.000234	23	25
25	110.0893	0.000233	24	38
14	106.6540	0.000240	25	38
20	106.3431	0.000241	26	38
1	57.9954	0.000224	27	76

(continua)

Tabela 42 – Relatório das RNAs obtidas na melhor geração do experimento 1 para o problema de detecção de intrusão usando o *KDDCUP'99* (continuação)

Indivíduo	Fitness	EMQ	Posto	Neurônios na Camada Oculta
17	57.3447	0.000226	28	76
12	56.4520	0.000230	29	76
3	0.0	0.0	30	-

Fonte: o autor (2016).

Tabela 43 – Comparações do *KDDCUP´99* com outros métodos para o problema do *KDDCUP´99*

Dataset	**(Devikrishna e Ramakrishna, 2007)**		**ADEANN**		**(Tavallaee et al. 2009)**		**(Pagano, 2011)**	
KDDCUP´99	96.33	-	**97.67** [a]	**3.4** [b]	94.44	-	92.26	-
	92.0	-	**97.79**[c]	**1.82**[d]	94.25	-	-	-

Onde: a) Taxa de detecção (DR%) para a classe normal b) Falsos Positivos para a classe Normal (FP% c)Taxa de detecção (DR%) para a classe de Intrusão e d)Falsos Positivos para a Classe Intrusão (FP%).
Fonte: o autor (2016).

7.8 RESULTADOS DE SIMULAÇÃO DE SISTEMAS DINÂMICOS

Nas secções anteriores, foram apresentados resultados de simulação para problemas estáticos usando redes diretas multicamadas. É sabido que, uma Rede direta multicamada munida de retardos, é capaz de aproximar, com precisão, um sistema dinâmico (BARRETO, 2001). Com o objetivo contemplar a simulação dessa classe de problema, nas secções seguintes apresentam-se resultados de simulação para PST. ANEs que são capazes de evoluir Redes Neurais Recorrentes são raros na literatura (BEER; GALLAGHER, 1992). Dessa forma, por meio dos resultados apresentados a seguir, ampliam-se as possibilidades do *ADEANN* em gerar topologias complexas de RNAs e simular problemas com maior grau de dificuldade.

7.9 SÉRIES TEMPORAIS

Uma série temporal, ou série histórica, consiste num conjunto de observações (discretas ou contínuas) no tempo. Uma notação utilizada para denotar uma série temporal é ***Z(t)* (t=1,2,...,N),** onde os valores ***Z1, Z2, Z3... , Zn*** representam uma série temporal de tamanho N. Grande quantidade de fenômenos físicos, biológicos, econômicas, etc, podem ser enquadrados nessa categoria.

A predição de séries temporais tem por finalidade determinar uma estimativa para os seus valores futuros da mesma, *ZN+1, ZN+2, …, ZN+p*, tomando por base a série temporal escalar ***Z(t)***. Dessa forma, os termos atuais da série temporal são utilizados para estimar valores futuros por meio de alguma técnica de predição. Assim, estima-se uma série relacionada ao futuro ***PN+1, PN+2, …, PN+p,*** o que pode ser definido como extrapolação.

7.9.1 Predição de séries temporais com RNAs

O problema de predição de séries temporais com RNAs consiste em obter o relacionamento do valor predito "t" e os valores dos elementos anteriores da série temporal ***(t-1,t-2,.....,t-k)*** para se obter uma função ($\mathbf{f}: \Re^n \rightarrow \Re$) tal como descrita pela Equação 43.

$$a_1 = f(a_{t-1}, a_{t-2}, \ldots, a_{t-k}) \tag{43}$$

Com a finalidade se obter uma RNA única para predizer os valores da série temporal, inicialmente os valores da mesma devem ser normalizados entre [0,1] e a RNA retornará os valores resultantes, o processo inverso é realizado, retornando-os para a escala original. As RNAs geradas pelo *ADEANN* para PST possuem um único neurônio na camada de saída (*1 to N ahead forecast*), pois conforme discute (DONATE; SANCHEZ; DE MIGUEL, 2012) se fossem permitidos vários na camada de saída a tarefa de predição seria realizada para diversos valores futuros de saída em uma sequência (*1 ahead forecast*). Esse método conduziria a predição do valor t a partir de ***t-k,....,t-2,t-1***, mas também a predizer valores ***t+1, t+2, ..., t+n***, a partir de ***t-k,.....,t-2,t-1*** , então a cada passo da predição, dados anteriores importantes estariam ausentes para ***t+1,t+2,....,t+n*** .

Assim, a série temporal será transformado em um conjunto de padrões dependendo dos ***k*** nós de entrada de uma determinada RNA, cada padrão consistirá de ***k*** valores de entrada e um de saída, que corresponde a um valor normalizado da série temporal que será predito.

Esses padrões serão usados para treinar e validar cada RNA gerada pelo *ADEANN*. Então os padrões serão divididos em dois conjuntos treinamento e validação, conforme descrito na Tabela 45.

7.9 AVALIAÇÃO DAS TÉCNICAS DE PREDIÇÃO

Dado que a acurácia é o critério mais relevante na avaliação das técnicas de predição, é necessário defini-la matematicamente. Com o objetivo de avaliar o desempenho (acurácia) absoluto e relativo de diversos modelos de predição, diversos modelos matemáticos foram propostos com o tempo, alguns deles são (ACZEL, 1993):

a) **ME:** *Mean Error,* ou Erro Médio, pode ser definido como o somatório dos erros (erro total) dividido pelo número de observações realizadas:

$$ME = \frac{\sum_{i=1}^{n} e_i}{n} \tag{44}$$

b) **MAE:** *Mean Absolute Error*, *ou* Erro Absoluto Médio, é o erro médio tomado em termos absolutos, para que um erro positivo não seja anulado por outro negativo:

$$MAE = \frac{\sum_{i=1}^{n} |e_i|}{n} \tag{45}$$

c) **MSE:** *Mean Square Error*, ou Erro Quadrático Médio, foi definido com a mesma finalidade de não anular os erros durante sua somatória:

$$MSE = \frac{\sum_{i=1}^{n} e_i^2}{n} \tag{46}$$

d) **MAPE**: *Mean Absolute Percentage Error*, ou Erro Percentual Absoluto médio, é uma medida do erro absoluto médio em termos percentuais, para que se tenha uma ideia do erro comparado com o valor previsto, e também para permitir comparações com modelos que utilizam dados diferentes:

$$MAPE = 100.\frac{\sum_{i=1}^{n} \left[\frac{X_i - F_i}{X_i}\right]}{n} \tag{47}$$

e) **SMAPE:** *Symetric Mean Absolute Percentage Error*, ou Erro Percentual Absoluto Médio Simétrico, é uma adaptação do MAPE, evitando que grandes erros pontuais tenham peso exagerado na medida de acurácias. É recomendado quando existirem falhas ou picos repentinos na demanda:

$$SMAPE = \frac{1}{H}\sum_{t=T+1}^{t+H} \frac{|y_t - F_t|}{(|y_t| + |F_t|)/2} . 100\% \quad (48)$$

Onde: t=T+1,...., t+H é o número de valores passados, h denota o horizonte de predição.

7.10 CONTEXTO DAS APLICAÇÕES

7.10.1 Descrição dos Bancos de Dados

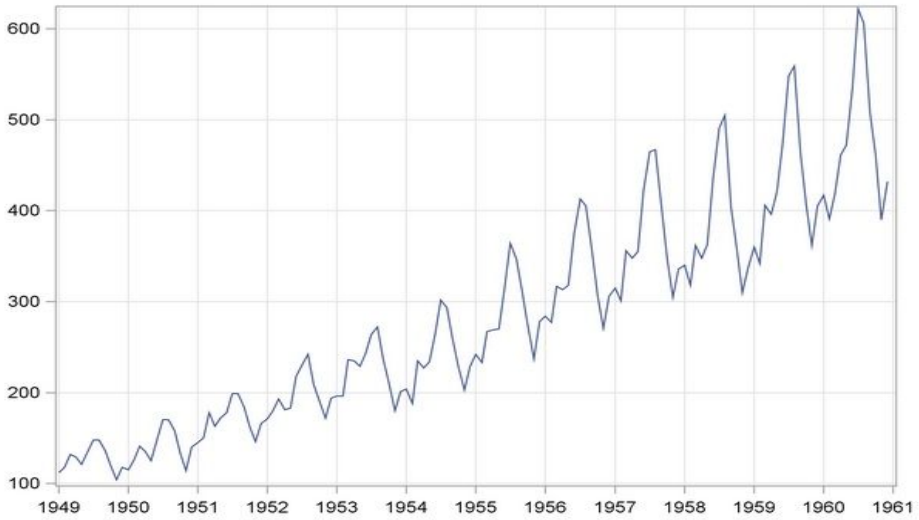
Para a PST, três bancos de dados do mundo real bem conhecidos foram escolhidos (Passageiros, Temperatura e Dow-*Jones*) (Tabela 45), esses conjuntos de dados foram usados no NN3 (Neural Forecasting *Competition* 2007) competição de PST, que teve por objetivo de desenvolver um método único de inteligência computacional a ser utilizado para simulação de tarefas de PST. A série temporal de Passageiros representa o número de viajantes de uma companhia aérea internacional, medido mensalmente a partir de janeiro 1949 até dezembro de 1960 (Figura 51). A série temporal de temperatura mostra a temperatura média mensal do ar medida pelo Castelo de *Nottingham* a partir de 1920 até 1939 (Figura 52). *Dow-Jones* apresenta os fechamentos mensais do *Dow - Jones Industrial Average* de agosto 1968 até agosto de 1981 (Figura 53).

Tabela 44 – Descrição dos bancos de dados usados nos experimentos de predição de séries temporais

Bancos de Dados	Entradas	Padrões
Passageiros	2	144
Temperatura	2	239
Dow-Jones	2	290

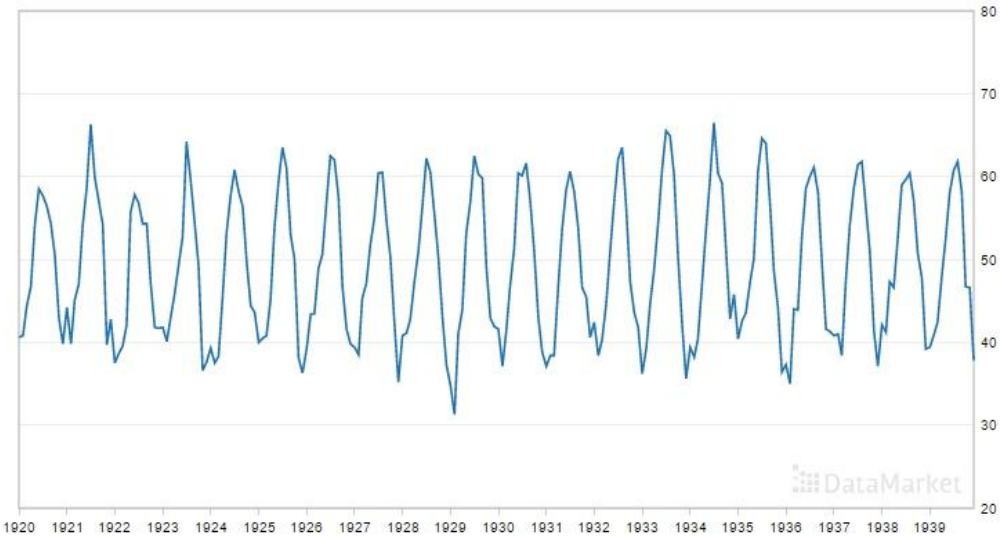
Fonte: o autor (2016).

Figura 51 – Série Temporal de Passageiros



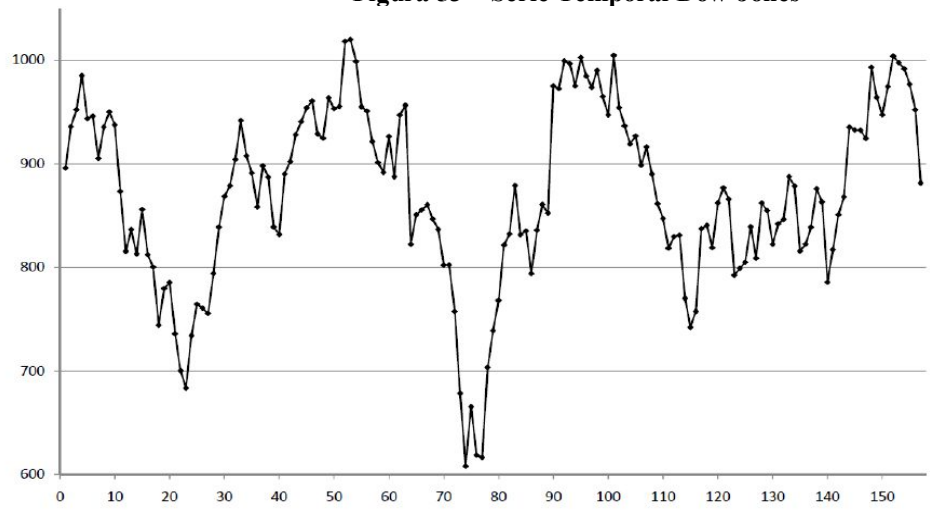
Fonte: adaptado de Box e Jenkins (1976).

Figura 52 – Série Temporal Temperatura



Fonte: adaptado de Anderson (1976).

Figura 53 – Série Temporal Dow-Jones



Fonte: adaptado de Hipel e McLeod (1994).

7.10.2 Estatísticas para predição de séries temporais

Conforme mencionado por Donate, Sanchez e de Miguel (2012) as séries temporais (Passageiros, Temperatura e *Dow-Jones*) apresentam diferentes características sazonais e de tendência. Como algumas séries temporais foram medidas mensalmente, eles têm um período de K = 12. Por outro lado, algumas delas também têm uma tendência. A Tabela 45 mostra todas essas características, bem como o número de amostras de entrada (dados usados para projetar a RNA) e amostras de saída (conjunto de teste ou predição) para cada série temporal.

Tabela 45 – Descrição das séries temporais utilizadas para treinamento e testes em tarefas de TSF

Bancos de Dados	Amostras de Entrada	Amostras De Saída	Período (K=12)	Normalização Utilizada
Passageiros	122	22	12	Z-Score
Temperatura	214	25	12	Escala Decimal
Dow-Jones	240	50	12	Escala Decimal

Fonte: o autor (2016).

A eficiência do método de busca, proposto pelo *ADEANN*, em retornar topologias de RNAs mais adequadas, como baixo custo computacional e com boa capacidade de generalização é comparada com as seguintes técnicas: *ARIMA* (Modelo Auto Regressivo Integrado de Média Móvel), Box e Jenkins (1976), *ADANN* (DONATE; SANCHEZ; DE MIGUEL, 2012) que é um método de evolução das redes neurais artificiais através de algoritmos genéticos, discutido no estado da arte, *Unobserved Component model* (UCM) (DONATE; SANCHEZ; DE MIGUEL, 2012) e um software para predição denominado Forecast Pro®.

Conforme discutido por Donate, Sanchez e de Miguel (2012), um modelo *ARIMA* é constituído de uma combinação de três componentes: número de termos auto regressivos (ou seja, os valores da série temporal como uma regressão linear dos valores anteriores, parâmetro **p**); tendência ou número de diferenças para tornar a série estacionária (parâmetro $\mathbf{d}=\mathbf{Z}_{(t)} - \mathbf{Z}_{(t-1)}=\Delta\mathbf{Z}_{(t)}$); número de termos da média móvel (ou seja, os valores de tempo de série como a adição da média e uma regressão linear do ruído, parâmetro **q**). Portanto, um modelo *ARIMA* (p, d, q) para uma série temporal univariada é uma combinação linear de seus valores e erros passados. Um modelo de componentes não observados (UCM) (HARVEY,

1989) é baseado na captura das características explícitas da série temporal que está sendo estudada, correspondendo a seus movimentos de longo prazo e seus padrões cíclicos e repetitivos, que são representados pela tendência e padrão sazonal. Nestes modelos, tais componentes podem variar estocasticamente ao longo do tempo, para que eles possam se adaptar às mudanças que as séries temporais experimentam.

As métricas usadas para medir o desempenho do *ADEANN* nos problemas de predição de séries temporais foram: erro médio quadrático médio dado pela Equação (33) e erro médio percentual absoluto simétrico dado pela Equação (35). *MSE* é uma métrica popular para predição de séries temporais. *SMAPE* tem a vantagem de ser independente de escala, assim, ele pode ser melhor usado para comparar métodos entre diferentes séries. Para a comparação da predição, foram utilizados *SMAPE* (variando de 0% a 100%). Em todas as medidas, valores mais baixos indicam melhores predições.

7.10.3 Testes estatísticos

Para comparar estatisticamente a performance do *ADEANN* com os quatro métodos, descritos na secção 7.10.2, foram realizados testes-t (cujo procedimento é descrito no ANEXO E), com nível de significância α=0.05, considerando dois critérios: o *SMAPE* (em percentagem), dado pela Equação (35), como o critério principal, e *MSE*, dado pela Equação (33), como o secundário. De maneira similar a Ahmadizar et al. (2015), se não houver nenhuma diferença estatística significativa entre os desempenhos de dois algoritmos comparados usando um determinado banco de dados em termos do primeiro ou segundo critérios, os dois algoritmos são ditos como tendo desempenho igual e ambos são recompensados com 1 ponto. Entretanto, no caso em que existe uma diferença estatística significativa entre as suas performances de acordo com os critérios primário ou secundário, o algoritmo que executa melhor é premiado com 2 pontos e o outro com zero. O desempenho global de cada algoritmo é então calculado pela soma dos pontos alcançados nestas comparações de pares em todos bancos de dados. Para a validação das hipóteses, considerou-se o valor-p, assim: Se o valor-p>α, aceita-se $(H_0: \mu_1 = \mu_2)$, caso contrário se o valor-p≤α rejeita-se H_0 e aceita-se $(H_1: \mu_1 \neq \mu_2)$.

Para a realização de cada *teste-t*, para os dois critérios *SMAPE* e *MSE,* verificou-se, por meio do teste de *Shapiro – Wilk* (descrito no anexo F), se as amostras são provenientes de

distribuições normais. Todos os testes obtiveram os valores-p associados. Assim, um baixo valor-p indica uma distribuição não-normal. Neste estudo, nós utilizamos o nível de significância $\alpha = 0.05$, assim um valor-p>α indica que a condição de normalidade é atingida, ou seja, a Hipótese nula é aceita (H_0: A amostra provém de uma distribuição Normal). Caso contrário, se o valor-p≤α rejeita-se H_0 e aceita-se (H_1: A amostra não provém de uma distribuição Normal). Neste caso, a independência dos eventos é óbvia, dado que eles são execuções independentes dos algoritmos com condições iniciais geradas aleatoriamente. Se as amostras seguem uma distribuição normal, consequentemente seguem uma distribuição (*t de student*).

7.10.4 Resultados de simulações

Um *benchmarking* composto por três séries temporais (passageiros, temperatura e *Dow - Jones*), previamente descritos na Tabela 45, foi utilizado para comparar o desempenho do *ADEANN* com dois outros métodos estatísticos de predição (*ARIMA* e *UCM*), com outro ANE denominado *ADANN* proposto por Donate, Sanchez e de Miguel (2012), e um software de predição denominado *Forecast Pro*®, os dados para comparação foram obtidos de Donate, Sanchez e de Miguel (2012).

A Tabela 46 resume os resultados obtidos para cada série temporal utilizando-se estes cinco métodos diferentes para prever os valores futuros. Duas métricas de erro *SMAPE* (%), dado pela Equação 35, e *MSE*, dado pela Equação 33, foram utilizadas para comparar os métodos. Na Tabela 46 apresentam-se os melhores resultados obtidos pelo *ADEANN* para a predição das séries temporais: Passageiros, Temperatura e *Dow-Jones*. Além disso, os melhores resultados, dentre todos os métodos, são representados em negrito.

Tabela 46 – Resultados com diferentes métodos de predição, CE significa esforço computacional e NNHL número de neurônios na camada intermediária

Série Temporal		*ADANN*	*ARIMA*	*UCM*	*Forecast Pro*	*ADEANN*
Passageiros	**SMAPE(%)**	3.05	3.14	2.37	4.5	**2.30**
	MSE	$0.59x10^{-3}$	$0.41x10^{-3}$	$\mathbf{0.28x10^{-3}}$	$0.75x10^{-2}$	$0.39x10^{-3}$
	CE NNHL	- -	- -	- -	- -	359348 2
Temperatura	**SMAPE(%)**	3.79	3.98	3.38	3.42	**2.02**
	MSE	$0.31x10^{-2}$	$0.29x10^{-2}$	$0.24x10^{-2}$	$0.25x10^{-2}$	$\mathbf{0.17x10^{-2}}$
	CE NNHL	- -	- -	- -	- -	445622 2
Dow-Jones	**SMAPE(%)**	4.76	4.78	4.78	4.78	**0.71**
	MSE	$0.10x10^{-1}$	$0.12x10^{-2}$	$0.12x10^{-1}$	$0.12x10^{-1}$	$\mathbf{0.26x10^{-4}}$
	CE NNHL	- -	- -	- -	- -	1499919 3

Fonte: o autor (2016).

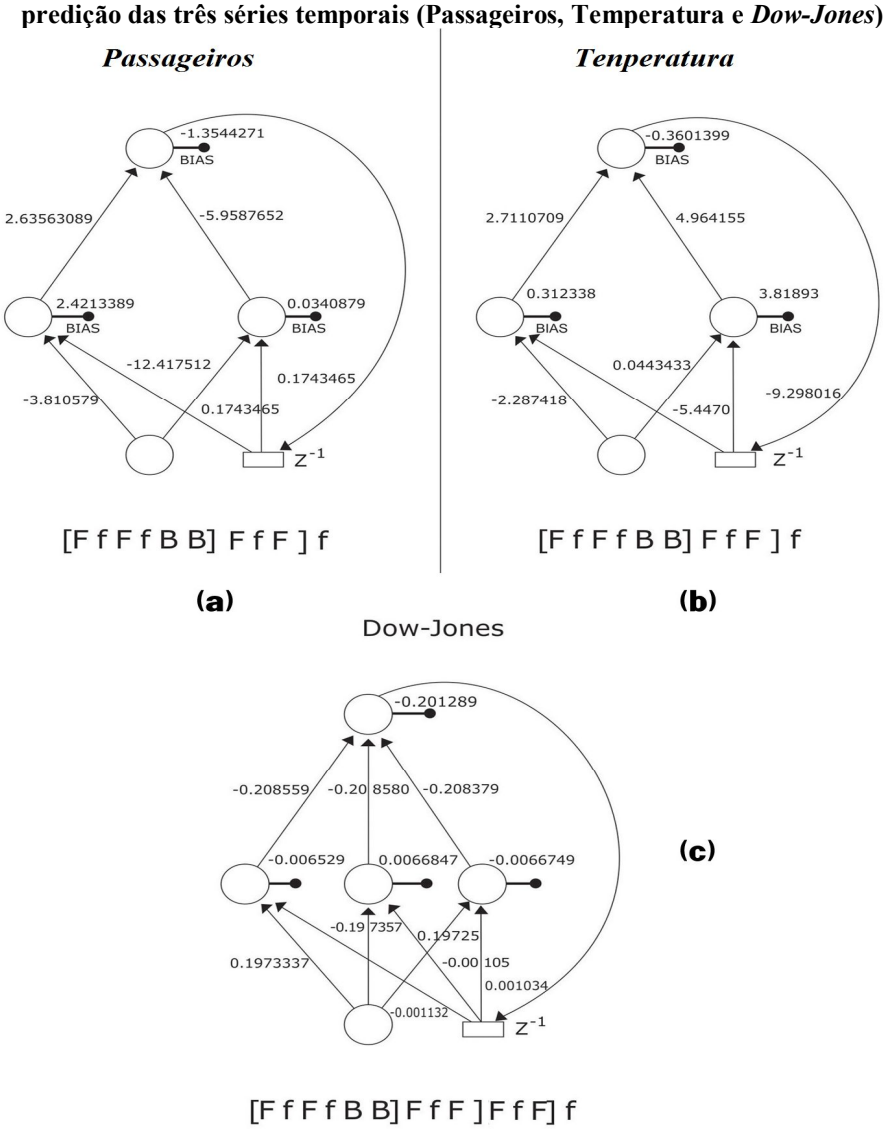
A primeira informação que pode ser obtida a partir da Tabela 46 é que o *ADEANN* supera ambos os métodos estatísticos (*ARIMA* e *UCM*), a ferramenta de software *Forecast Pro®,* e o *ADANN* (DONATE; SANCHEZ; DE MIGUEL, 2012) na predição das séries Temperatura e *Dow-Jones*. Para estas duas séries, o *ADEANN* tem o melhor desempenho, com os seguintes valores: *SMAPE* (%) = (2,02% e 0,71%), *MSE* = ($0.17x\ 10^{-2}$ e $0.26x10^{-4}$), respectivamente. Em relação a série de Passageiros, o *ADEANN* supera o *ARIMA*, o *ADANN* (DONATE; SANCHEZ; DE MIGUEL, 2012) e o software *Forecast Pro*, considerando ambos os critérios *SMAPE* (%) e *MSE*. Além disso, supera o *UCM* mas apenas quando considerado o *SMAPE* (%), com uma diferença de 3 %. *UCM* supera *ADEANN* considerando apenas o critério *MSE,* com uma diferença de 0.11. No entanto, *SMAPE* (%) representa uma métrica melhor para o erro quando comparada com o *MSE*. Considerando a série de passageiros e temperatura, *UCM* supera *ARIMA*, *ADANN* e *Forecast Pro®.*

Donate, Sanchez e de Miguel (2012) não forneceram informações sobre o esforço computacional e o número médio de neurônios da camada intermediária, obtidos nas simulações de predição para as três séries temporais previamente analisadas. Apenas o *ADEANN* apresenta estes valores, que são resumidos na Tabela 46. Pode-se observar que em termos de neurônios ocultos, *ADEANN* seleciona arquiteturas de rede neural econômicas com boa capacidade preditiva. Para a predição das três séries temporais (Passageiros, Temperatura e *Dow - Jones*) o número de neurônios ocultos de cada rede neural foi de 2, 2, e 3,

respectivamente (Figura 54). *ADEANN* resolveu o problema de predição das três séries temporais sem problemas em buscar arquiteturas econômicas de RNAs, com boa capacidade de predição.

Tabela 47 compara estatisticamente o desempenho dos *ADEANN* com quatro outros métodos para previsão de séries temporais, discutidos na secção 7.10.2. Como visto, *ADEANN* alcança os melhores resultados para quase todos os conjuntos de dados descritos na Tabela 46 e fornece o melhor desempenho global também de acordo com os critérios e procedimentos descritos na secção 7.10.3. No anexo G são apresentados os resultados detalhados dos testes estatísticos (*teste-t*) que deram origem as pontuações destacadas na Tabela 46 e *Shapiro-Wilk*. Os valores numéricos correspondentes, ilustrados na Tabela G.1 do Anexo G, são os valores-p associados a cada teste estatístico (*teste-t* ou *Shapiro-Wilk*). Para atribuição da pontuação a cada método de predição da Tabela 47, consideram-se os critérios apresentados na secção 7.10.3.

Figura 54 – Arquiteturas de Redes Neurais Recorrentes retornadas pelo processo de busca nas tarefas de predição das três séries temporais (Passageiros, Temperatura e *Dow-Jones*)



Fonte: o autor (2016).

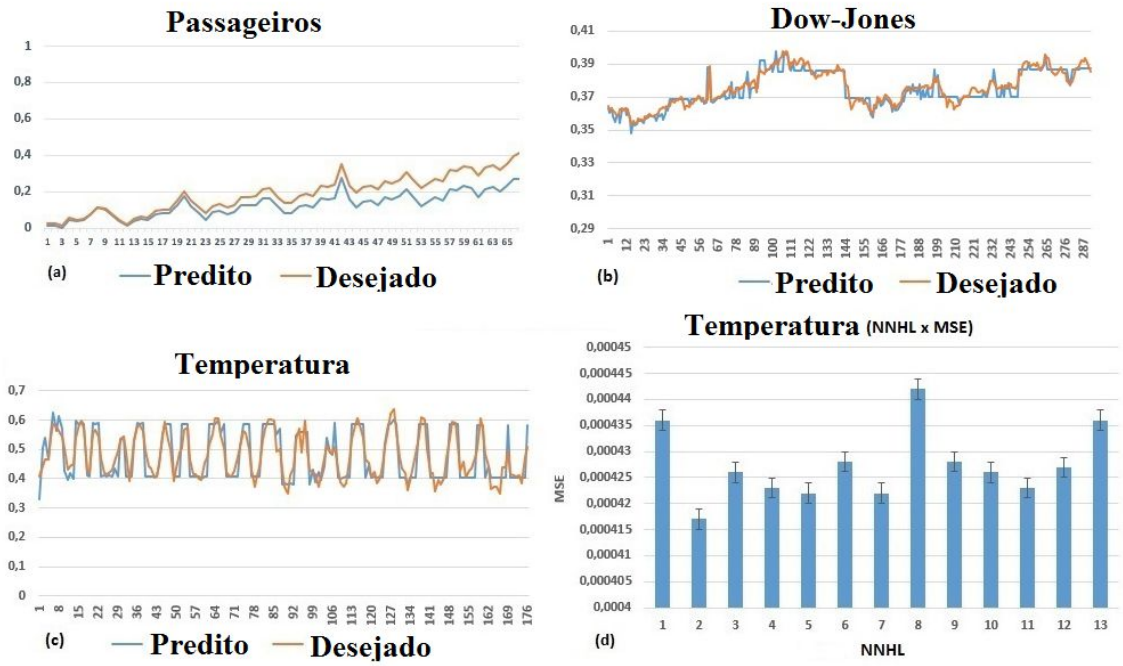
Tabela 47 – Comparação Estatística entre o *ADEANN* e outros algoritmos por meio de *testes-t* (p<0.05)

Série Temporal		*ADANN*	*ARIMA*	*UCM*	*Forecast Pro*	*ADEANN*
Passageiros	**SMAPE(%)**	4	2	6	0	8
	MSE	2	2	8	2	6
Temperatura	**SMAPE(%)**	2	0	6	4	8
	MSE	0	2	4	4	8
Dow-Jones	**SMAPE(%)**	6	0	0	0	8
	MSE	6	0	0	0	8
Desempenho Global		20	6	24	10	**46**

Fonte: o autor (2016).

Para se ter uma melhor ideia sobre quão próximos dos valores reais, foram as predições de cada método, um gráfico para cada série temporal, mostrando todas as predições obtidas é apresentado. Um *zoom* para cada predição da série de Passageiros, temperatura e *Dow-Jones* é ilustrado na Figura 55a, 55b, e 55c. Pode-se observar que os valores preditos são próximos aos reais. A Figura 55d mostra a relação entre o número de neurônios na camada oculta e o MSE para a predição da série temporal de temperatura. A Tabela 48 sumariza os valores obtidos e desejados na predição da série temporal *Dow-Jones* e a Tabela 49 mostra os valores de aptidão obtidos no melhor experimento para a predição da série temporal *Dow-Jones*.

Figura 55 – Gráficos obtidos nas simulações de predição para as três séries temporais (Passageiros, Temperatura e Dow-Jones)



Fonte: o autor (2016).

Tabela 48 – Valores Obtidos e desejados para o problema de predição da série temporal *Dow-Jones*

Obtido	Desejado	Erro
0,376923	0,3767	2,49E-08
0,370349	0,3751	1,13E-05
0,37682	0,3769	3,20E-09
0,375123	0,376	3,85E-07
0,3783	0,3785	2,00E-08
0,370381	0,3776	2,61E-05
0,370388	0,3755	1,31E-05
0,370396	0,3755	1,30E-05
0,370404	0,3751	1,10E-05
0,370412	0,3776	2,58E-05
0,38675	0,3847	2,10E-06
0,386757	0,383	7,06E-06
0,386765	0,3881	8,91E-07
0,386773	0,3899	4,89E-06
0,386781	0,3917	1,21E-05
0,386789	0,3913	1,02E-05
0,3908	0,3901	2,45E-07
0,386804	0,3886	1,61E-06
0,386812	0,3892	2,85E-06
0,38682	0,3899	4,74E-06
0,386828	0,3886	1,57E-06
0,386836	0,3908	7,86E-06
0,38762	0,3875	7,20E-09
0,386051	0,386	1,30E-09
0,386859	0,388	6,51E-07
0,3897	0,3895	2,00E-08
0,395523	0,3954	7,56E-09
0,386882	0,3933	2,06E-05
0,38689	0,3937	2,32E-05
0,386898	0,3869	2,00E-12
0,386906	0,3852	1,46E-06
0,386914	0,3837	5,16E-06
0,386921	0,3832	6,92E-06
0,386929	0,3849	2,06E-06
0,386937	0,3863	2,03E-07
0,386945	0,3878	3,66E-07
0,386953	0,3855	1,06E-06
0,38696	0,3843	3,54E-06
0,386968	0,3847	2,57E-06
0,386976	0,3801	2,36E-05
0,378654	0,3787	1,06E-09
0,376662	0,3776	4,40E-07
0,386999	0,3797	2,66E-05
0,387007	0,3821	1,20E-05
0,387015	0,3877	2,35E-07
0,387023	0,3875	1,14E-07
0,387031	0,389	1,94E-06
0,387038	0,391	7,85E-06
0,387046	0,3924	1,43E-05
0,387054	0,3918	1,13E-05
0,387062	0,3936	2,14E-05
0,38707	0,3911	8,12E-06
0,387077	0,3891	2,05E-06
0,387085	0,3855	1,26E-06

Fonte: o autor (2016).

Tabela 49 – Valores Obtidos de Fitness para a série temporal *Dow-Jones*

Geração	Melhor Fitness	Fitness Médio
1	1471,233	899,1115
2	4365,08	1934,296
3	8143,029	3348,883
4	3658,445	1421,061
5	8928,261	3420,189
6	8469,147	3311,193
7	8216,753	2644,228
8	3819,48	1274,09
9	5028,3	2154,783
10	9181,605	3240,612
11	4365,538	1455,811
12	7955,012	2106,864
13	9189,377	2951,157
14	9038,028	2817,029
15	3287,674	1014,61
16	6289,221	1964,161
17	6523,433	2606,165
18	4511,817	1609,023
19	6738,28	2283,209
20	7672,572	2601,935
21	7183,317	2075,393
22	4123,058	1244,428
23	7896,524	3075,052
24	4031,055	1346,403
25	8268,729	3300,481
26	8244,961	2843,476
27	8120,356	3483,932
28	4353,106	1524,551
29	4743,832	1970,013
30	9140,462	3359,922
31	4079,969	1300,173
32	8400,976	2786,199
33	9254,284	3426,721
34	9091,657	3739,367
35	3365,313	836,5545
36	6222,022	2282,871
37	7911,801	2018,258
38	4648,002	1697,758
39	7346,174	1942,956
40	8415,941	2911,454
41	8448,596	3236,542
42	3894,553	1629,365
43	7695,349	3066,622
44	4435,186	1027,856
45	7068,866	3061,907
46	7962,029	3160,233
47	7919,028	3140,049
48	5377,186	1627,374
49	4648,787	1682,53
50	9191,028	3904,689
51	3753,043	1155,608
52	9286,461	3105,489
53	9169,713	3707,181
54	8890,479	4129,824
55	8150,766	3194,762
56	16227,91	7315,722

Tabela 49 – Valores Obtidos de Fitness para a série temporal Dow-Jones (continuação)

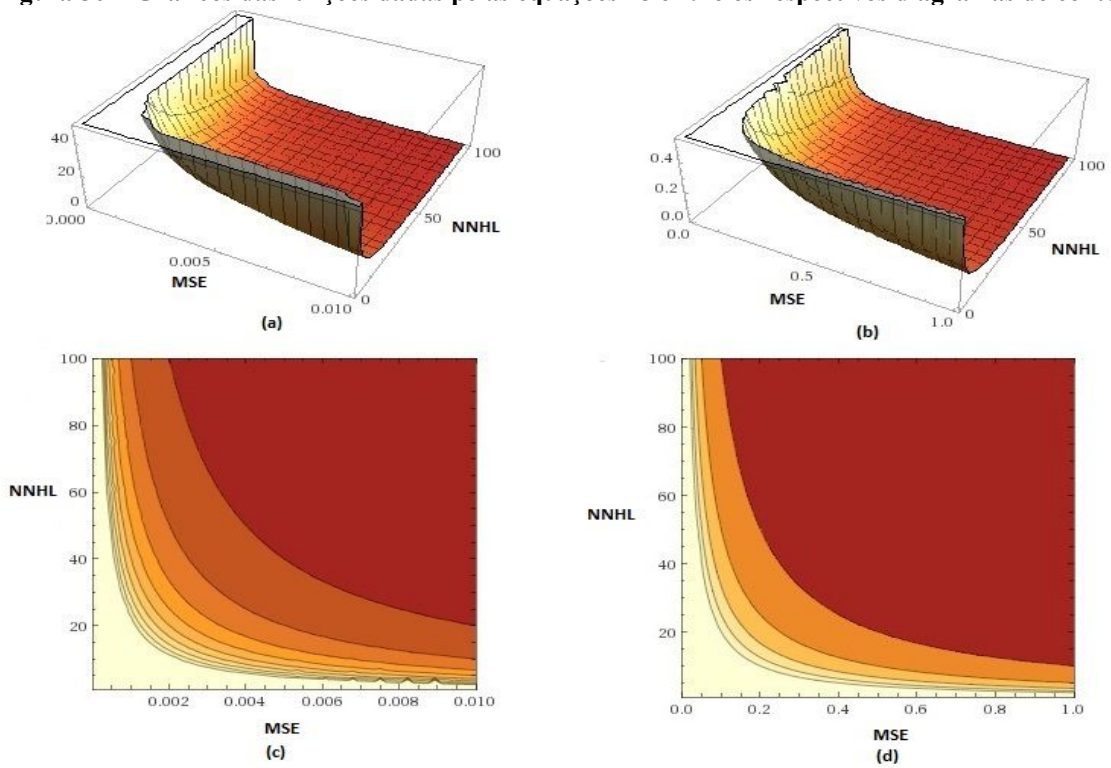
57	32002,19	8589,833

58	11791,49	4804,973
59	22898,86	8094,232
60	28432,11	9919,873
61	29271,48	11852,01
62	8494,366	3181,697
63	20147,89	8909,453
64	12992,16	3957,401
65	14070,48	5193,435
66	20919,24	8476,804
67	21111,31	7516,909
68	16486,66	5561,069
69	11292,32	4487,267
70	28254,34	10693,25
71	9056,06	3004,778
72	31701,49	11133,8
73	27866,01	11783,03
74	25137,38	7499,518
75	8985,877	2110,258
76	14121,57	6148,42
77	31129,79	11789,33
78	11696,49	4893,038
79	23549,82	6101,701
80	30301,62	11538,43
81	29618,73	9960,82
82	8499,6	2625,1
83	19025,19	7957,886
84	20887,87	6134,44
85	11834,21	4262,754
86	18641,65	5414,966
87	21100,48	8375,225
88	18868,57	6475,773
89	11828,46	4525,477
90	26486,19	12323,92
91	10065,18	2858,552
92	33107,5	9655,741
93	26180,51	8313,46
94	24288,19	9611,477
95	10458,61	4881,188
96	14031,07	5334,676
97	29723,68	9825,807
98	11470,4	4357,262
99	25049,89	6569,752
100	30888,32	11272,91

A seleção de uma função fitness adequada é essencial para definir o espaço de busca do AG. As Figuras 56a e 56b apresentam as frentes ótimas e subótimas de Pareto dadas pelas equações 28 e 29, respectivamente e as Figuras 56c e 56d apresentam as curvas de nível destas funções. Devido à proximidade das linhas da Figura 56c, a conclusão é que os desníveis da superfície do gráfico mostrado na Figura 56a são muito acentuados. No entanto, na Figura 56d as linhas estão mais afastadas, ou seja, na superfície mostrada os desníveis são menos acentuados. Assim, a utilização da função de aptidão ilustrada na Figura 56b e definida pela equação 29, torna mais eficiente o processo de busca evitando a estabilização em mínimos

locais. A determinação de coeficientes apropriados (A1 e A2) para a equação 28 é um processo exaustivo, que consome muito tempo, sendo, portanto, um processo de tentativa e erro para selecionar os coeficientes (A1 e A2) adequados para a equação 28. Dependendo dos valores dessas constantes nem sempre a busca do AG é direcionada para arquiteturas econômicas de RNAs.

Figura 56 – Gráficos das funções dadas pelas equações 28 e 29 e os respectivos diagramas de contorno



Fonte: o autor (2016).

7.11 CONSIDERAÇÕES FINAIS

No presente capítulo, apresentaram-se os resultados de simulação para problemas estáticos e dinâmicos usando redes diretas e recorrentes. Adicionalmente, apresentaram-se os aspectos teóricos relativos a classificação de dados e PST, métricas de avaliação para classificação e predição e alguns procedimentos para realização de testes estatísticos nesse contexto. Além disso, detalharam-se os planejamentos dos experimentos e os bancos de dados utilizados em tarefas de classificação e PST. Finalmente, comparou-se o desempenho do *ADEANN* com outros nove ANEs utilizados para classificação, dois métodos estatísticos de predição (*ARIMA* e *UCM*), outro ANE denominado *ADANN* proposto por Donate, Sanchez e de Miguel (2012) e um software de predição denominado *Forecast Pro*®. No próximo

capítulo apresentam-se as discussões e as conclusões finais da nossa pesquisa e as indicações de trabalhos futuros.

CAPÍTULO 8 – CONCLUSÃO

O trabalho descrito nessa tese insere-se no campo dos ANEs, que consistem em projetar e/ou treinar RNAs através de algoritmos evolucionários AEs. As RNAs oferecem a possibilidade de inferir mapeamentos desconhecidos entre entrada-saída. Além disso, são excelentes no reconhecimento de padrões e aproximação de funções. ANEs ajudam na solução de problemas que são difíceis de serem solucionados por métodos determinísticos padrões.

A metodologia proposta nessa tese, demonstra que os AEs representam uma técnica adequada para resolver o problema de projeto automático de RNAs ao proporcionar resultados satisfatórios na simulação de um conjunto de problemas, estáticos e dinâmicos, do mundo real. Uma vantagem do método proposto é a redução drástica do esforço requerido do especialista humano para projetar e configurar uma rede neural para um dado problema. Muito esforço foi dedicado para melhorar o desempenho do AE, tendo sido propostos diferentes esquemas de seleção e operadores genéticos existentes na literatura.

Para fazer a busca de forma eficiente no espaço de arquiteturas possíveis o *ADEANN* representa topologias neurais em seus cromossomos usando um sistema de codificação gramatical. Esta representação, permite um processo de busca mais eficiente, em um espaço de busca menor, pois os tamanhos dos cromossomos utilizados pelo *ADEANN s*ão pequenos quando comparados com outros métodos, ver secção 8.1 (b). Além disso, o sistema proposto implementa automaticamente uma abordagem de penalização que recompensa redes neurais econômicas, que possuem boa capacidade de generalização e capacidade preditiva e que são facilmente implementadas.

O desempenho do *ADEANN* é confirmado em vários experimentos de classificação e previsão de séries temporais. Os resultados são estatisticamente comparados com vários ANEs e métodos de predição bem conhecidos reportados na literatura e discutidos no estado-da-arte. O ANE aqui proposto supera outros métodos e produz o melhor desempenho global.

8.1 COMENTÁRIOS SOBRE O ADEANN

A seguir apresentam-se vários comentários sobre o *ADEANN* focando nos conceitos mais importantes sobre o quais essa tese foi direcionada.

a) **Dependência mínima do especialista**: esta tese propõe um sistema automatizado de reconhecimento de padrões e previsão de séries temporais, exigindo apenas um conhecimento mínimo do usuário. O sistema GP (RIVERO et al., 2010) tem vários parâmetros, três dos quais (o coeficiente de penalidade, a altura das árvores, e o número máximo de entradas para cada neurônio) que devem ser definidos por um especialista. No entanto, o *ADEANN* tem apenas cinco parâmetros (listados na Tabela 21), que podem ser definidos pelo usuário;

b) **Esquema de codificação dos genótipos compacto**: no *ADEANN* o ECI inspirado na codificação genética do ADN, um único gene é utilizado várias vezes em diferentes fases do desenvolvimento neuronal. Os genes reutilizados (regras de produção codificadas) permitem representações compactas de fenótipos complexos e redução dimensional do espaço de busca. Para elucidar a compressão fornecida por ECI do *ADEANN,* consideramos a redução de escalabilidade em relação a outros métodos. Nos métodos de Kitano (1990) e Boozarjomehry e Svrcek (2001), os tamanhos das redes geradas são funções quadráticas do comprimento dos cromossomos, dados por N^2 e (N (N + 1) / 2), respectivamente. Por exemplo, para os comprimentos de cromossomos listados na Tabela 17, que variam entre 180-516, as redes neurais que seriam geradas pelos métodos de Kitano (1990) e Boozarjomehry e Svrcek (2001) conteriam (no máximo) 13 a 22 neurónios e 18 a 31 neurônios, respectivamente. Em contraste, dependendo do valor de NR gerado aleatoriamente, nas Equações 41 e 42, o *ADEANN* pode gerar RNAs muito maiores, pois os tamanhos das redes neurais geradas (fenótipos) são independentes do comprimento do cromossomo, o que é uma vantagem. O método proposto no *ADEANN* pode manipular cromossomas, que são 30% menores (redução de 256 bits para 180) do que aqueles usados no *Graph Rewriting Grammar* (GRG) (CANTU-PAZ; KAMATH, 2005). No ECI proposto por Hornby e Pollack (2002), os genótipos codificam vinte regras de reescrita de um Sistema-L. O ECI do *ADEANN* codifica um Sistema L paramétrico com apenas dez regras de produção, demonstrando que o mesmo é mais compacto do que a versão proposta por Hornby e Pollack (2002). O método de codificação baseado no ADN, utilizado pelo *ADEANN* pode representar arbitrariamente regras de desenvolvimento de qualquer número e comprimento (LEE; SEO; SIM, 2007). Além disso, o mecanismo de memória utilizado em nossa abordagem permite gerar vias de

desenvolvimento separadas. Rivero et al. (2010) utiliza uma lista para armazenar neurônios existentes, e referencia-los por operadores especiais. Em contraste, o método utilizado pelo *ADEANN* gera grafos codificados por uma gramática no AG. Portanto, múltiplos neurónios estão implicitamente referenciados dentro dos cromossomos do AG, ao invés de armazenar topologias neurais prontas. Por conseguinte, as arquiteturas de redes neurais geradas pelo *ADEANN* podem crescer indefinidamente, independentemente da estrutura de dados;

c) **Implementa o paralelismo implícito**: para tornar o AG, utilizado pelo *ADEANN,* mais próximo do processo evolutivo biológico, o mesmo evolui uma representação generativa (um A*DN* hipotético). Os genótipos do mesmo, codificam um conjunto de dez regras de reescrita de um Sistema L. Com objetivo de modelar a abordagem construtiva dos processos evolutivos reais, no método utilizado pelo *ADEANN,* as regras do Sistema L são automaticamente extraídas pelo AG. Desta forma, pode-se controlar o número e o tempo de disparos das mesmas, isto é, o número de vezes de que cada regra é aplicada e em que contexto. O processo de extração de regras pelo AG no *ADEANN* é inspirado na síntese de proteínas, que é um processo paralelo que consiste na obtenção de nucleotídeos que constituem as proteínas. Como uma metáfora desse processo, a função extração de regras com o AG (secção 6.3.2), implementa esse mecanismo similar de paralelismo, lendo cada cromossomo a partir de diversas posições simultaneamente, proporcionando um maior paralelismo implícito no processo de extração de regras pelo AG. O que torna o *ADEANN* mais rápido do que os demais.

d) **Otimização de redes neurais**: o *ADEANN* também permite a otimização de redes neurais. Como discutido na subseção 6.3.3, a função de aptidão dada pela Equação 29 automaticamente implementa uma abordagem de penalização que recompensa redes neurais econômicas e com melhores capacidades de generalização e facilidade de implementação (ver os últimos quinze linhas e segunda coluna da Tabela 18). Esta função seleciona redes neurais baseadas em dois critérios: o número de neurônios da camada escondida e o *MSE*. Portanto, redes neurais menores obterão valores de aptidão maiores do que redes neurais maiores que possuam o mesmo desempenho na generalização. ANEs que evoluem redes recorrentes são raramente reportados na literatura (HORNBY; POLLACK, 2002). O *ADEANN* preenche parcialmente essa lacuna, possibilitando a evolução desse

tipo de arquitetura neural para simulação de problemas de reconhecimento de padrões e sistemas dinâmicos;

e) **Performance**: o desempenho e a robustez do *ADEANN*, em problemas de classificação, foram avaliados pelo método de validação cruzada. Conforme mostrado na secção 7.7.4, o *ADEANN* obteve um melhor desempenho global dentre todos os métodos pertencentes a categoria (II) (Tabela 34) e (III) de algoritmos (Tabela 35) e teve um desempenho comparável à de outros métodos pertencentes a categoria (I) (Tabela 33). O *ADEANN* requer menor esforço computacional do que todos os outros métodos pertencentes a todas as demais categorias, (Tabelas 29, 30 e 32). O *ADEANN* alcança a melhor precisão média de classificação em 55,5% dos casos. O desempenho do *ADEANN* em tarefas de predição de séries temporais também foi comparado com os métodos *ARIMA* (BOX; JENKINKS, 1976), *UCM, ADANN* (DONATE; SANCHEZ; DE MIGUEL, 2012) e o software de previsão *Forecat Pro®*. A Tabela 46 resume os resultados obtidos na simulação da predição de três séries temporais. O *ADEANN* superou os outros métodos na predição de todas as três séries temporais (Passageiros, temperatura e *Dow-Jones*), mas produziu maior *MSE* do que método *UCM* na predição da série temporal Passageiros. Conforme ilustrado na Tabela 47, o *ADEANN* obteve os melhores resultados em quase todos os conjuntos de dados e o melhor desempenho global dentre todos os métodos. Além disso, dentre os métodos de evolução gramatical, ou seja, *GEGA* (AHMADIZAR et al., 2015), *GE* (TSOULOS; GAVRILIS; GLAVAS, 2008) e *GE-GP* (SOLTANIAN et al., 2013), o *ADEANN* é o único bioinspirado, o que contribui para a sua originalidade.

f) No entanto, a abordagem proposta nessa tese também possui pontos fracos. A mesma é muito mais sensível às condições iniciais e constantes do que outros métodos existentes que não utilizam Sistemas-L como modelos de desenvolvimento. À medida que a população inicial é inicializada aleatoriamente com zeros e uns, os genótipos gerados são difíceis de serem controlados. Portanto, serão necessários testar outros métodos para configuração da população inicial, tais como métodos estocásticos, baseados, por exemplo em uma distribuição Normal (Gaussiana). Além disso, como a NR variável nas Equações 41 e 42 é gerada de forma aleatória, o algoritmo pode não gerar todas as topologias possíveis neurônios no intervalo [X, Y] numa geração específica.

8.2 PESQUISA FUTURA

Em trabalhos futuros, sugere-se focar em outros desafios, como a possibilidade gerar redes parcialmente conectadas e o uso de RNAs com aprendizagem Hebbiana, explorando assim os aspectos da plasticidade neural e otimização local de nós. Uma outra tarefa futura é a implementação do *ADEANN* de forma paralelizada segundo o modelo de computação paralela *GPGPU (General Purpose Graphics Processing Unit),* utilizando-se UCPs com arquitetura multicores e aplicação intensiva de memória CPU/I-O. Para adaptar o modelo proposto nessa tese a outros modelos, tais como *deep learning* será necessário adaptar e implementar o Sistema-L prevendo os quatro casos, descritos no anexo C. Além disso, será necessário implementar o algoritmo de aprendizado para essa arquitetura de RNA, considerando múltiplas camadas, bem como encontrar uma abordagem de aprendizado rápida e com um baixo *overfitting*.

REFERÊNCIAS

ACZEL, A. D. **Complete business statistics**. 2. ed. Irwin: Homewood, 1993.

AHMADIZAR, F. et al. Artificial neural networks development by means of a novel combination of gramatical evolution and genetic algorithm. **Engineering Applications of Artificial Intelligence**, Luxembourg, v. 39, p. 1-13, 2015.

ALBERTS, B. et al. **Molecular biology of the cell**. 5. ed. New York; London: Garland Science, 2002.

ANDERSON, O. D. **Time series analysis and forecasting**: the Box-Jenkins approach. London: Butterworths, 1976.

ASCOLI, G. A.; KRICHMAR, J. L. L-Neuron : a modeling tool for the eficiente generation and parsimonious description of dendritic morphology. **Neurocomputing**. Amsterdam, p. 32-33, 1003-1011, 2000.

BAKER, J. E. Adaptive selection methods for genetic algorithms. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 1., 1985, Mahwah, USA. **Proceedings…**, [S.l.], 1985. p. 101-111.

BARRETO, J. M. **Inteligência artificial no limiar do século XXI**: abordagem híbrida, simbólica, conexionista e evolucionária. Florianópolis: UFSC, 2001.

BEER, R. D.; GALLAGHER, J. C. Evolving dynamical neural networks for adaptive behavior. **Adaptive Behavior**, v . 1, n. 1, p. 91-122, 1992.

BOERS, E.; KUIPER, H. **Biological metaphors and the design of artificial neural networks**. 1992. Thesis (Master) – Leiden University, Leiden, 1992.

BOERS, E.; KUIPER, H.; HAPPEL, B. **Designing modular artificial neural networks**. Leiden: Departament of Computer Science and Theoretical Psychology, Leiden University, 1993.

BOOZARJOMEHRY, R. B.; SVRCEK, W. Y. Automatic design of neural network structures. **Computers and Chemical Engineering**, [s.l.], v. 25, p. 1075-1088, 2001.

BOX, G. E. P.; JENKINS, G. M. **Time series analysis**: forecasting and control. 2. ed. San Francisco: Holden-Day, 1976.

BRAGA, A. P.; DE CARVALHO, A. C. P. L.; LUDEMIR, T. B. **Redes neurais artificiais**: teoria e aplicações. 2. Ed. Rio de Janeiro: LTC, 2007.

BRINDLE A. **Genetic algorithms for function optimization**. 1981. Thesis (Doctoral) – University of Alberta, Alberta, 1981.

CANTU-PAZ, E.; KAMATH, C. An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. **IEEE Transactions on Systems, Man, and Cybernetics, Part B**, Cambridge, v. 35, n. 5, p. 915-927, 2005.

CHOMSKY, N. Three models for the description of language. **IRE Transactions on Information Theory**, Cambridge, v. 2, n. 3, p. 113-124, 1956.

CLUNE, J. et al. On the performance of indirect encoding across the continuum of regularity. **IEEE Transactions on Evolutionary Computation**, Cambridge, v. 15, n. 3, p. 346-367, 2011.

CLUNE, J.; CHEN, A.; LIPSON, H. Upload any object and evolve it: injecting complex geometric patterns into CPPNs for further evolution. In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, 2013. **Proceedings…**, 2013. p. 3395-3402.

CUSTÓDIO, F. L.; BARBOSA, H. C. J. C.; DARDENE, L. E. Investigation of three-dimensional lattice HP protein folding model using a genetic algorithm. **Genetics and Molecular Biology**, [s.l.], v. 27, p. 611-615, 2004.

CZAJKOWSKI, A.; PATAN, K.; SZYMASKI, M. Application of the state space neural network to the fault tolerant control system of the plc-controlled laboratory stand. **Engineering Applications of Artificial Intelligence**, Luxembourg, v. 30, p. 168-178, 2014.

D'AMBROSIO, D.; STANLEY, K. O. A novel generative encoding for exploiting neural network sensor and output geometry. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE (GECCO), 2007, New York. 2007. **Proceedings…** New York: ACM Press, 2007.

DASGUPTA, D.; MCGREGOR, D. Designing application-specific neural networks using the structured genetic algorithm. INTERNATIONAL CONFERENCE ON COMBINATIONS OF GENETIC ALGORITHMS AND NEURAL NETWORKS, 1992. **Proceedings…** 1992. p. 87-96.

DAWKINS, R. **The blind watchmaker**: why the evidence of evolution reveals a universe without design. [S.l.]: Norton and Co., 2004.

DE CAMPOS, L. M. L.; DE OLIVEIRA, R. C. L.; ROISENBERG, M. Evolving artificial neural networks through L-system and evolutionary computation. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), July 2015. **Proceedings…**, 2015. p. 1-9.

DE CAMPOS, L. M. L.; OLIVEIRA, R. C. L. A Comparative analysis of methodologies for automatic design of artificial neural networks from the beginnings until today. In: BRICS COUNTRIES CONGRESS (BRICS-CCI), 1., and BRAZILIAN CONGRESS ON COMPUTATIONAL INTELLIGENCE (CBIC), 11., 2013, Porto de Galinhas. **Proceedings…** 2013. v. 1, p. 1-8.

DE JONG, K. A.; SPEARS, W. M. A formal analysis of the role of multi-point crossover in genetic algorithms. **Annals of Mathematics and Artificial Intelligence**, [s.l.], v. 5, n. 1, p. 1-26, 1992.

DE SOUSA, G. et al. Evolving dendritic morphology and parameters in biologically realistic model neurons for pattern recognition. In: VILLA, A. E. P. et al. (Ed.). **Artificial neural networks and machine learning**: ICANN. Berlin: Springer, 2012. p. 355-362.

_____. Synaptic plasticity and pattern recognition in cerebellar purkinje cells. In: CUNTZ, H.; REMME, M. W.; TORBEN-NIELSEN, B. (Eds.). **The computing dendrite**. Philadelphia: Spring, 2014. (Springer Series in Computational Neuroscience, v. 11, p. 433-448).

DEB, K.; ANAND, A.; JOSHI, D. A computationally efficient evolutionary algorithm for real-parameter optimization. **Evolutionary Computation**, Cambridge, v. 10, n. 4, p. 371-395, 2002.

DEVIKRISHNA, K. S.; RAMAKRISHNA, B. B. An artificial neural network based intrusion detection system and classifiation of attacks. **International Journal of Engineering Research and Applications**, [s.l.], v. 3, n. 4, p. 1959-1964, 2007.

DONATE, J. P.; SANCHEZ, G. G.; DE MIGUEL, A. S. Time series forecasting. a comparative study between an evolving artificial neural networks system and statistical methods. **International Journal on Artificial Intelligence Tools**, Singapore, v. 21, n. 1, 2012.

DONG, Y.; KAISHENG, Y.; YU, Z. The Computational network toolkit. **IEEE Signal Processing Magazine**, Cambridge, n. 123, Nov. 2015.

DONG-WOOK, L.; SANG-WOOK, S.; KWEE-BO, S. Evolvable neural networks based on developmental models for mobile robot navigation. **International Journal of Fuzzy Logic and Intelligent Systems**, [s.l.], v. 7, n. 3, p. 176-181, 2007.

EIBEN, A. E.; SMITH, J. E. **Introduction to evolutionary computation**. New York: Springer, 2003.

FACELI, K. et al. **Inteligência** artificial: uma abordagem de aprendizado de máquina. Rio de Janeiro: Grupo Gen, 2011.

FAHLMAN, S. E. Faster-learning variations on back-propagation: an empirical study. In: CONNECTIONIST MODELS SUMMER SCHOOL, 1988. **Proceedings...**, Los Altos: Morgan-Kaufmann, 1988.

FERREIRA, P. J. S. S. **Modelação de tráfego em redes de telecomunicações**: modelos Markovianos e baseados em sistemas de Lindenmayer. Tese (Doutorado) – Departamento de Eletrônica e Telecomunicações, Universidade de Aveiro, Aveiro, 2005.

FINKEL, R.; BENTLEY, J. Quad trees: a data structure for retrieval on composite keys. **Actainformatica**, [s.l.], v. 4, n. 1, p. 1-9, 1974.

FOGEL, G. B. et al. Discovery of sequence motifs related to coexpression of genes using evolutionary computation. **Nucleic Acids Research**, London, v. 32, n. 13, p. 3826-3835, 2004.

FONSECA, L. G. **Otimização evolucionista via algoritmos genéticos assistidos por meta-modelos baseados em similaridade**. 2009. Tese (Doutorado) – Laboratório Nacional de Computação Cientifica, Petrópolis, 2009.

GAUCI, J.; STANLEY, K. O. A case study on the critical role of geometric regularity in machine learning. In: CONFERENCE ON ARTIFICIAL INTELLIGENCE (AAAI), 23., 2008, Menlo Park. **Proceedings**… Washington: AAAI Press, 2008.

GAUCI, J.; STANLEY, K. O. Autonomous evolution of topographic regularities in artificial neural networks. **Neural Computation**, Cambridge, v. 22, n. 7, p. 1860-1898, 2010.

GOLBERG, D. E. **Genetic algorithms in search, optimization, and machine learning**. New York: Addison-Wesley, 1989.

GOLBERG, D. E.; DEB, K. A. Comparative analysis of selection schemes used in genetic algorithms. **Foundations of Genetic Algorithms**, [s.l.], v. 1, p. 69-93, 1991.

GRUAU, F. Genetic synthesis of boolean neural networks with a cell rewriting developmental process. In: INTERNATIONAL WORKSHOP ON COMBINATIONS OF GENETIC ALGORITHMS AND NEURAL NETWORKS: (COGANN-92), 1992. **Proceedings**… Los Alamitos: IEEE Computer Society Press, 1992. p. 55-74.

GRUAU, F.; WHITLEY, D.; PYEATT, L. A comparison between cellular encoding and direct encoding for genetic neural networks. In: KOZA, J. R. et al. (Ed.). **Genetic Programming 1996**: Proceedings of the First Annual Conference. Cambridge: MIT Press, 1996. p. 81-89.

GUO, J. **Backpropagation through time**. Heilongjiang: Harbin Institute of Technology, 2013.

HARP, S. A.; SAMAD, T.; GUHA, A. Designing application-specific neural networks using the genetic algorithm. In: TOURETZKY, D. S. (Ed.). **Advances in neural information processing systems 2**. San Mateo: Morgan Kaufman, 1990. p. 447-454.

HARVEY, A. **Forecasting structural time series models and kalman filter**. Cambridge: Cambridge Univsersity Press, 1989.

HEBB, D. O. **Organization of Behaviour**: a neuropsychological theory. New York: Wiley, 1949.

HILLMAN, D. Neuronal shape parameters and substructures as a basis of neuronal form. In: SCHMITT, F. (Ed.). **The neurosciences**: fourth study program. Cambridge: MIT Press, 1979. p. 477-498.

HIPEL, K. W.; MCLEOD, A. I. **Time series modelling of water resources and environmental systems**. Amsterdam: Elsevier, 1994.

HOLLAND, J. H. **Adaptation in natural and artificial systems**. Cambridge: MIT Press, 1975.

HORNBY, G. S.; POLLACK, J. B. Creating high-level components with a generative representation for body-brain evolution. **Artificial Life**, Cambridge, v. 8, n. 3, p. 223-246, 2002.

ILONEN, J.; KAMARAINEN, J. K.; LAMPINEN, J. Differential evolution training algorithm for feed-forward neural network. **Neural Processing Letters**, Dordrecht, v. 17, p. 93-105, 2003.

ISLAM, M.; YAO, X.; MURASE, K. A constructive algorithm for training cooperative neural networks ensembles. **IEEE Transactions on Neural Networks**, Cambridge, v. 14, n. 4, p. 820-834, 2003.

JONES, N. C.; PEVZNER, P. A. **An introduction to bioinformatics algorithms**. Cambridge: MIT Press, 2004.

KITANO, H. Designing neural networks using genetic algorithms with graph generation system. **Complex Systems**, Champaign, v. 4, n. 4, p. 461-476, 1990.

KROMER, P.; PLATOS, J.; SNÁSEL, V. Nature-inspired meta-heuristics on modern GPUS: state of the art and brief survey of selected algorithms. **International Journal of Parallel Programming**, Norwell, v. 42, n. 5, p. 681-709. 2014.

LEE, D. W.; SEO, S. W.; SIM, K. B. Evolvable neural networks for time series prediction with adaptive learning interval. **International Journal of Fuzzy Logic and Intelligent Systems**, Seul, v. 8, n. 1, p. 31-36, 2008.

LEE, S. et al. Evolving gaits for physical robots with the HyperNEAT generative encoding: the benefits of simulation. **Applications of Evolutionary Computing**, Berlin, p. 540-549, 2013.

LEHMAN, J.; STANLEY, K. O. Exploiting open-endedness to solve problems through the search for novelty. In: BULLOCK, S. et al. (Ed.). **Proceedings of the Eleventh International Conference on Artificial Life (Alife XI)**. Cambridge: MIT Press, 2008.

LI, X. L., et al. Nonlinear adaptive control using multiple models and dynamic neural networks. **Neurocomputing**. Amsterdam, v. 136, p. 190-200, 2014.

LIN, W. Y.; LEE, W. Y.; HONG, T. P. Adapting crossover and mutation rates in genetic algorithms. **Journal of Information Science and Engineering**, Taiwan, v. 19, n. 5, p. 889-903, 2003.

LINDENMAYER, A. Mathematical models for celular interaction in development, parts I and II. **Journal of Theorical Biology**, Amsterdam, v. 18, p. 280-315, 1968.

MACKEY, M. C.; GLASS, L. Oscillation and chaos in physiological control systems. **Science**, New York, v. 197, n. 4300, p. 287-289, 1977.

MCCLELLAND, J. L.; RUMELHART, D. E.; PDP GROUP. **Parallel distributed processing**: psycological and biological models. Cambridge: The MIT Press, 1986. v. 2.

MCCULLOCH, W.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, [s.l.], v. 5, p. 115-133, 1943.
MERRILL, J. W. L.; PORT, R. F. Fractally configured neural networks. **Neural Networks**, Tokyo, v. 4, n. 1, p. 53-60, 1991.

MEYER, D.; EL-HANI, C. N. **Evolução**: o sentido da biologia. São Paulo: UNESP, 2005.
MILLER, B. L.; GOLDBERG, D. E. Genetic algorithms, tournament se- lection, and the effects of noise. **Complex Systems**, Champaign, v. 9, p. 193-212, 1996.

MILLER, G. F.; TODD, P. M.; HEDGE, S. U. Designing neural networks using genetic algorithms. In: SCHAFFER, J. D. (Ed.). In: CONFERENCE ON GENETIC ALGORITHMS AND THEIR APPLICATIONS, 3., 1989. **Proceedings…** San Mateo: Morgan Kaufman, 1989. p. 379-384.

MINSKY, M. L.; PAPERT, S. A. **Perceptrons**: an introduction to computational geometry., Cambridge: MIT Press, 1969.

MJOLSNESS, E.; SHARP, D. H.; ALPERT, B. K. Scaling, machine learning, and genetic neural nets. **Advances in Applied Mathematics**, [s.l.], v. 10, p. 137-163, 1989.

MONTANA, D. J.; DAVIS, L. D. Training feedforward networks using genetic algorithms. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL, 1989. **Proceedings…** San Mateo: Morgan Kaufman, 1989.

NARENDRA, K. S.; PARTHASARATHY, K. Gradient methods for the optimization of dynamical systems containing neural networks. **IEEE Transaction on Neural Networks**, Cambridge, v. 2, p. 252-262, 1991.

NEWMAN, D. J. et al. **UCI Repository of machine learning databases**. Irvine: Department of Information and Computer Science, University of California, 1998. Available from: <http://www.ics.uci.edu/~mlearn/MLRepository.html>. Access: 17 July 2016.

NISKA, H. et al. Evolving the neural network model for forecasting air pollution time series. **Engineering Applications of Artificial Intelligence**, Luxembourg, v. 17, p. 159-167, 2004.

OLADELE, R. O.; SADIKU, J. S. Genetic algorithm performance with different selection methods in solving multi-objective network design problem. **International Journal of Computer Applications**, [s.l.], n. 70, p. 5-9, 2013.
p. 563-570.

PAGANO, M. W. **Evaluation of diagonal confidence-weighted learning on the KDD Cup 1999 dataset for network intrusion detection systems**. Baltimore: Johns Hopkins University, Department of Computer Science, 2011.

PARSI, A.; SALEHI, M.; DOOSTMOHAMMADI, A. Swap training: a genetic algorithm based feature selection method applied on face recognition system. **World of Computer Science and Information Technology Journal**, [s.l.], v. 2, n. 4, p. 125-130, 2012.

PRUSINKIEWICZ, P.; LINDENMAYER, A. **The algorithmic beauty of plants**. NewYork: SpringerVerlag, 2004.
PRUSINKIEWICZ, P.; RUNIONS, A. Computational models of plant development and form. **New Phytologist**, [s.l.], n. 193, p. 549-569, 2012.

RISI, S. et al. How novelty search escapes the deceptive trap of learning to learn. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE (GECCO), 2009, New York. **Proceedings…** New York: ACM Press, 2009.

RISI, S.; HUGHES, C. E.; STANLEY, K. O. Evolving plastic neural networks with novelty search. **Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems**, Thousand Oaks, v. 18, n. 6, p. 470-491, 2010.

RISI, S.; LEHMAN, J.; STANLEY, K. O. Evolving the placement and density of neurons in the hyperneat substrate. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE (GECCO), 2010, New York. **Proceedings…** New York: ACM Press, 2010.

RISI, S.; STANLEY, K. O. A unified approach to evolving plasticity and neural geometry. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), 2012. **Proceedings…** Cambridge: IEEE, 2012. p. 1-8, 10-15.

_____. Enhancing ES-HyperNEAT to evolve more complex regular neural networks. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE (GECCO), 2011. **Proceedings…** New York: ACM, 2011. p. 1539-1546.

RIVERO, D. et al. Generation and simplification of artificial neural networks by means of genetic programming. **Neurocomputing**, Amsterdam, v. 73, n. 1618, p. 3200-3223, 2010.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological Review**, [s.l.], v. 65, n. 6, p. 386-408, 1958.

ROSENFELD, A. Quadtrees and pyramids for pattern recognition and image processing. In: CONFERENCE ON PATTERN RECOGNITION, 5., 1980. **Proceedings…** Cambridge: IEEE Press, 1980. P. 802-809.

RUMELHART, D. E.; MCLELLAND, J. L. **Parallel distributed processing**: exploration in the microstructure of cognition. Cambridge: MIT Press, 1986.

SANCHEZ, D.; MELIN, P. Optimization of modular granular neural net- works using hierarchical genetic algorithms for human recognition using the ear biometric measure. **Engineering Applications of Artificial Intelligence**, Luxembourg, v. 27, p. 41-56, 2014.

SÁNCHEZ, D.; MELIN, P.; CASTILLO, O. Optimization of modular granular neural networks using a hierarchical genetic algorithm based on the database complexity applied to human recognition. **Information Sciences**, [s.l.], v. 309, p. 73-101, 2015.

SOLTANIAN, K. et al. Artificial neural networks generation using grammatical evolution. In: IRANIAN CONFERENCE ON ELECTRICAL ENGINEERING (ICEE), 21. **Annals…** 2013. p. 1-5.

STANLEY, K. O. Compositional pattern producing networks: a novel abstraction of development. **Genetic Programming and Evolvable Machines**, [s.l.], v. 8, n. 2 esp., p. 131-162, 2007.

STANLEY, K. O.; BRYANT, B. D.; MIIKKULAINEN, R. Real-time neuroevolution in the NERO video game. **IEEE Transactions on Evolutionary Computation**, Cambridge, v. 9, n. 6, p. 653-668, 2005.

STANLEY, K. O.; DAVID, B. D. A.; JASON, G. A hypercube-based encoding for evolving large-scale neural networks. **Artificial Life**, Cambridge, v. 15, n. 2, p. 185-212, 2009.

STANLEY, K. O.; MIIKKULAINEN, R. A Taxonomy for artificial embryogeny. **Artificial Life**, Cambridge, v. 9, n. 2, p. 93-130, 2003.

_____. Evolving neural networks through augmenting topologies. **Evolutionary Computation**, Cambridge, v. 10, n. 2, p. 99-127, 2002.

STROBACH, P. Quadtree-structured recursive plane decomposition coding of images. **IEEE Transactions on Signal Processing**, Cambridge, v. 39, p. 1380-1397, 1991.

SZILARD, A. L.; QUINTON, R. E. An interpretation for DOL systems by computer graphics. **The Science Terrapin**, [s.l.], p. 8-13, 1979.

TAVALLAEE, M. E.; BAGHERI, W. L. U.; GHORBANI, A. A detailed analysis of the kdd cup 99 data set. In: IEEE SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE FOR SECURITY AND DEFENSE APPLICATIONS, 2009. **Proceedings…** Cambridge: MIT Press, 2009. p.1-6.

TORBEN-NIELSEN, B.; TUYLS, K.; POSTMA, E. Evol-neuron: neuronal morphology generation. **Neurocomputing**. Amsterdam, v. 71, p. 963-972, 2009.

TOWNSEND, J.; KEEDWELL, E.; GALTON, A. Artificial development of biologically plausible neural-symbolic networks. **Cognitive Computation**, Berlin, v. 6, n. 1, p. 1-17, 2013.

TSOULOS, I.; GAVRILIS, D.; GLAVAS, E. Neural network construction and training using grammatical evolution. **Neurocomputing**. Amsterdam, v. 72, n. 13, p. 269-277, 2008.

VAARIO, J. **An emergent modeling method for artificial neural networks**. 1993. Thesis (Doctoral) – University of Tokyo, Tokyo, 1993.

VAARIO, J. From evolutionary computation to computational evolution. **Informatica**, [s.l.], v. 18, p. 417-434, 1994.

VOIGT, H. M.; BORN, J.; SANTIBANEZ-KOREF, I. **Evolutionary structuring of artificial neural networks**. Berlin: Bionics and Evolution Techniques Laboratory, University of Berlin, 1993.

VONK, E.; JAIN, L. C.; JOHNSON, R. Using genetic algorithms with grammar encoding to generate neural networks. In: IEEE INTERNATIONAL CONFERENCE OF NEURAL NETWORKS, 2011. **Proceedings**… Cambridge, 2011. part. 4, p. 1928-1931.

WHITESON, S. Evolutionary computation for reinforcement learning. In: WIERING, M. A.; VAN OTTERLO, M. (Eds.). **Reinforcement learning**: state of the art. Berlin: Springer, 2012. p. 325-352.

WHITLEY, D. The genitor algorithm and selection pressure: why rank based allocation of reproduction is best. In: INTERNATIONAL ON GENETIC ALGORITHMS AND THEIR APPLICATIONS, 3., 1989. **Proceedings**… 1989. p. 116-121.

WHITLEY, L. D.; SCHAFFER, J. D. In: INTERNATIONAL WORKSHOP ON COMBINATIONS OF GENETIC ALGORITHMS AND NEURAL NETWORKS: (COGANN-92), 1992. **Proceedings**… Los Alamitos: IEEE Computer Society Press, 1992.

YAO, X. Evolving Artificial Neural Networks. **Proceedings of the IEEE**, Cambridge, v. 87, n. 9, p. 1423-1447, 1999.

YAO, X.; SHI, Y. A preliminary study on designing artificial neural networks using co-evolution. In: IEEE SINGAPORE INTERNATIONAL CONFERENCE OF INTELLIGENT CONTROL AND INSTRUMENTATION, Singapore, June 1995. **Proceedings…** Cambridge: IEEE Press, 1995. p. 149-154.

ANEXO A – Algoritmo Retropropagação

Os passos do algoritmo de retropropagação são descritos abaixo:

Seja A o número de unidades da camada de entrada, conforme determinado pelo comprimento dos vetores de entrada de treinamento. Seja C o número de unidades da camada de saída. Seja B o número de unidades da camada oculta. As camadas de entrada e oculta, têm, cada uma, uma unidade extra usada como limite; portanto as unidades dessas camadas, às vezes, serão indexadas pelos intervalos (0,....,A) e (0,.....,B). Denota-se os níveis de ativação das unidades da camada de entrada por xj, da camada oculta por hj e da camada de saída por oj. Os pesos que conectam a camada de entrada a camada oculta são denotados por $w1_{ij}$, onde i indexa as unidades de entrada e o j, as unidades ocultas. Da mesma forma, os pesos que conectam a camada oculta à camada de saída são denotados por $w2_{ij}$, com i indexando as unidades ocultas e j as unidades de saída.

Inicializar os pesos da rede. Cada peso deve ser ajustado aleatoriamente para um número entre –0.1 e 0.1.

$w1_{ij}$=aleatório(-0.1;0.1), $w2_{ij}$=aleatório(-0.1;0.1) para todo i=0,.....,A; j=1,.....,B;

para todo i=0,.....,B; j=0,.....,C.

Inicializar as ativações das unidades limite. Seus valores nunca mudam.

x_o=1.0 e h_o=1.0

Escolher um par de padrão de entrada-saída. Supondo que o vetor de entrada seja x_i e o vetor de saída seja y_j. Atribuem-se níveis de ativação às unidades de entrada.

Propagar a ativação das unidades da camada de entrada para as unidades da camada oculta, usando a função de ativação.

$$h_j = \frac{1}{1+e^{-\sum_{i=0}^{A} w1ij.xi}} \text{ , para todo j=1,...,B}$$

(A1)

Propaga-se as ativações das unidades da camada oculta para as unidades da camada de saída, usando a função de ativação.

$$o_j = \frac{1}{1+e^{-\sum_{i=0}^{B} w2ij.xi}} \text{ para todo j=1,.....,C}$$

(A2)

Computar os erros das unidades da camada de saída, denotados por $\partial 2_j$. Os erros baseiam-se na saída real da rede (o_j) e na saída (y_j).

$$\delta 2_j = o_j(1-o_j)(y_j - o_j) \text{ para todo j=1,.....,C} \quad \text{(A3)}$$

Computar os erros das unidades da camada oculta, denotados por δ1j.

$$\delta 1_j = h_j(1-h_j)\sum_{1=0}^{C} \delta 2_i.w2_{ij} \text{ para todo j=1,.....,B} \quad \text{(A4)}$$

Ajuste dos pesos entre a camada oculta e a camada de saída. O coeficiente de aprendizagem é denotado por η, sua função é a mesma de na aprendizagem por perceptrons.

$$\Delta w2_{ij}(t+1) = \Delta w2_{ij}(t) + \eta\delta 2_j.h_i \text{ para todo i=0,.....,B; j=1,.....,C}$$

(A5)

Ajuste os pesos entre a camada de entrada e a camada oculta

$$\Delta w1_{ij}(t+1) = \Delta w1_{ij}(t) + \eta\delta 1_j.x_i \text{ para todo i=0,.....,A; j=1,.....,B} \quad \text{(A6)}$$

Vá para a etapa 4 e repita. Quando todos os pares entrada-saída tiverem sido apresentados à rede, uma época terá sido completada. Repita as etapas de 4 a 10 para tantas épocas quantas forem desejadas.

ATUALIZAÇÃO DOS PESOS DA CAMADA INTERMEDIÁRIA E CAMADA DE SAÍDA PARA REDES DIRETAS

Esta seção contém a dedução do algoritmo "backpropagation" (regra delta generalizada RDG). A explicação aqui apresentada é mínima, o leitor interessado pode encontrar um estudo mais aprofundado em ou Freeman e Skapura (1991). A dedução será apresentada para uma rede de três camadas e pode ser facilmente generalizada para rede de mais de três camadas.

O erro para um nó de saída durante o treinamento é δm=(ynm-om), com oj a saída obtida e yj a saída desejada para o nó j de saída. O erro que é minimizado pela regra delta generalizada é dado pela equação A7.

$$\varepsilon_n(t) = 1/2\sum_{m=0}^{C} \delta_m{}^2 \quad \text{(A7)}$$

Com C o número de nós de saída. Deseja-se minimizar ε, logo o melhor caminho é ir no sentido contrário ao gradiente, já que este aponta para o sentido crescente da função.

$$\varepsilon_n(t) = 1/2\sum_{m=0}^{C}(y_{nm} - o_m)^2 = 1/2\sum_{m=0}^{C}\varepsilon_{nm} \tag{A8}$$

Onde: n=número de padrões que serão apresentados durante o treinamento.As atualizações dos pesos são dadas pelas equações (A9) e (A10).

$$\Delta w1_{jk}(t+1) = \Delta w1_{jk}(t) - \eta\frac{\partial\varepsilon_n}{\partial w1(t)_{jk}} \tag{A9}$$

$$\Delta w2_{km}(t+1) = \Delta w2_{km}(t) - \eta\frac{\partial\varepsilon_n}{\partial w2(t)_{km}} \tag{A10}$$

$$\text{onde } S_{jk}(t) = \frac{-\partial\varepsilon_n}{\partial w1_{jk}(t)},\ r_{km}(t) = \frac{-\partial\varepsilon_n}{\partial w2_{km}(t)} \tag{A11}$$

Deseja-se determinar as variações dos pesos relativos aos nós de saída, calcula-se o gradiente negativo $-\nabla E$ com relação aos pesos $w2_{km}$. Considerando cada componente de ∇E, separadamente obtém-se a equação (A12).

$$\frac{\partial\varepsilon_n(t)}{\partial w2_{km}(t)} = \frac{1}{2}\frac{\partial\varepsilon_{nm}}{\partial o_m}\frac{\partial o_m}{\partial w2_{km}} \tag{A12}$$

$$\frac{\partial\varepsilon_{nm}}{\partial o_m} = -2(y_{nm} - o_m) \tag{A13}$$

$$\frac{\partial o_m}{\partial w2_{km}} = \frac{\partial}{\partial w2_{km}}\left[\frac{1}{1+e^{-\sum_{K=0}^{B} w2_{km}.hk}}\right] \tag{A14}$$

, u=$1+e^{-\sum_{K=0}^{B} w2_{km}.hk}$ (A.15), $o_m = \frac{1}{u}$, u-1=$e^{-\sum_{K=0}^{B} w2_{km}.hk}$ (A16)

$$\frac{\partial o_m}{\partial w2_{km}} = \frac{\partial o_m}{\partial_u}.\frac{\partial_u}{\partial w2_{km}} \tag{A17}$$

$$\frac{\partial o_m}{\partial u} = -\frac{1}{u^2},\ \frac{\partial u}{\partial w2_{km}} = (u-1)(-h_k) \tag{A18),(A19}$$

$$\frac{\partial o_m}{\partial w2_{km}} = h_k.\frac{(u-1)}{u^2},\ \frac{\partial o_m}{\partial w2_{km}} = \frac{(1/o_m - 1)}{1/o_m{}^2} = (1-o_m).o_m.h_k \tag{A20}$$

$$\frac{\partial \varepsilon_n}{\partial w2_{km}(t)} = \frac{1}{2}.-2(y_{nm}-o_m)(1-o_m)o_m.h_k \tag{A21}$$

$$\frac{\partial \varepsilon_n}{\partial w2_{km}(t)} = -(y_{nm}-o_m)(1-o_m).o_m.h_k \tag{A22}$$

O cálculo para os pesos relativos a camada intermediária é mostrado a seguir:

$$\frac{\partial \varepsilon_n}{\partial w1_{jk}(t)} = \frac{1}{2}\sum_{m=0}^{C}\frac{\partial \varepsilon_{nm}}{\partial o_m}\frac{\partial o_m}{\partial h_k}\frac{\partial h_k}{\partial w1_{jk}} \tag{A23}$$

$$\frac{\partial \varepsilon_{nm}}{\partial o_m} = -2(y_{nm}-o_m) \tag{A24}$$

$$\frac{\partial o_m}{\partial h_k} = \frac{\partial}{\partial h_k}\left[\frac{1}{1+e^{-\sum_{K=0}^{B} w2_{km}.hk}}\right] \text{(A.25)} \quad \text{u}=1+e^{-\sum_{K=0}^{B} w2_{km}.hk} \tag{A25}$$

$$o_m = \frac{1}{u}\ ,\ \text{u-1}=e^{-\sum_{K=0}^{B} w2_{km}.hk}\ ,\ \frac{\partial o_m}{\partial h_k} = \frac{\partial o_m}{\partial u}.\frac{\partial u}{\partial h_k} \tag{A26}$$

$$\frac{\partial o_m}{\partial u} = -\frac{1}{u^2}\ ,\ \frac{\partial u}{\partial h_k} = (u-1)(-w2_{km}) \tag{A27}$$

$$\frac{\partial o_m}{\partial h_k} = \frac{-1}{u^2}.\frac{(u-1)(-w2_{km})}{} = \frac{(u-1)(w2_{km})}{u^2} \tag{A28}$$

$$\frac{\partial o_m}{\partial h_k} = \frac{(1/o_m-1)(w2_{km})}{1/{o_m}^2} = (1-o_m).o_m.w2_{km} \tag{A29}$$

$$\frac{\partial h_k}{\partial w1_{jk}} = \frac{\partial}{\partial w1_{jk}}\left[\frac{1}{1+e^{-\sum_{j=0}^{A} w1jk.xnj}}\right] \tag{A30}$$

$$\text{u}=1+e^{-\sum_{j=0}^{A} w1jk.xnj}\ ,\ \text{u-1}=e^{-\sum_{j=0}^{A} w1jk.xnj} \tag{A31}$$

$$h_k = \frac{1}{u}\ ,\ \frac{\partial h_k}{\partial w1_{jk}} = \frac{\partial h_k}{\partial u}.\frac{\partial u}{\partial w1_{jk}} \tag{A32}$$

$$\frac{\partial h_k}{\partial u} = -\frac{1}{u^2}\ ,\ \frac{\partial u}{\partial w1_{jk}} = -(u-1)(x_n) \tag{A33}$$

$$\frac{\partial h_k}{\partial w1_{jk}} = \frac{(u-1)(x_{nj})}{u^2} = \frac{(\frac{1}{h_k}-1)}{\frac{1}{h_k^2}}.x_n \quad \text{(A34)}$$

$$\frac{\partial h_k}{\partial w1_{jk}} = (1-h_k).h_k.x_{nj}\,i \quad \text{(A35)}$$

$$\frac{\partial \varepsilon_m}{\partial w1_{jk}(t)} = \frac{1}{2}\sum_{m=0}^{C}\frac{\partial \varepsilon_{nm}}{\partial \theta_m}\frac{\partial \theta_m}{\partial h_k}\frac{\partial h_k}{\partial w1_{jk}} \quad \text{(A36)}$$

$$\frac{\partial \varepsilon_n}{\partial w1_{jk}(t)} = \frac{1}{2}\sum_{m=0}^{C} -2(y_{nm}-o_m)(1-o_m).o_m.w2_{km}.(1-h_k)h_k.x_{nj} \quad \text{(A37)}$$

Cálculo de $S_{JK}(t)$ e $r_{KM}(t)$

$$S_{jk}(t) = \frac{-\partial \varepsilon_n}{\partial w1_{jk}(t)} = (1-h_k)h_k.x_{nj}\sum_{m=0}^{C}(y_{nm}-o_m)(1-o_m).o_m.w2_{km} \quad \text{(A38)}$$

$$r_{km}(t) = \frac{-\partial \varepsilon_n}{\partial w2_{km}(t)} = (y_{nm}-o_m)(1-o_m).o_m.h_k \quad \text{(A39)}$$

As equações (A40) e (A41) mostram as fórmulas para atualização dos pesos, da camada de saída e da camada intermediária, respectivamente.

$$\Delta w2_{km}(t+1) = \Delta w2_{km}(t) + \eta\,[(y_{nm}-o_m)(1-o_m).o_m].h_k \quad \text{(A40)}$$

$$\Delta w1_{jk}(t+1) = \Delta w1_{jk}(t) + \eta\,[(1-h_k)h_k.\sum_{m=0}^{C}(y_{nm}-o_m)(1-o_m).o_m.w2_{km}]x_{nj} \quad \text{(A41)}$$

Método de inserção do termo de momentum

A inserção do termo de momentum se configura como uma das variações mais simples de ser efetuada no algoritmo backpropagation, pois, basta inserir um único parâmetro visando ponderar o quão as matrizes sinápticas foram alteradas entre duas iterações anteriores e sucessivas. Formalmente, considerando os neurônios pertencentes à L-ésima camada, tem-se:

$$w_{ji}^{(L)}(t+1) = w_{ji}^{(L)}(t) + \alpha\left(w_{ji}^{(L)}(t) - w_{ji}^{(L)}(t-1)\right) + \mu\delta_j^{(L)}.Y_i^{(L-1)} \quad \text{(A42)}$$

Onde: α é definida como taxa de momentum, valor compreendido entre 0 e 1 e μ é a taxa de aprendizado

ANEXO B – Backpropagation com atraso no tempo

As saídas na camada intermediária são agora dadas pela equação B1.

$$hk(t)= \frac{1}{1 + e^{-\sum_{m=0}^{C} om\ (t-1).\ w3(t)_{mk} + \sum_{j=0}^{A} Xnj\ (t).\ w1(t)_{jk}}}, \tag{B1}$$

O termo om(t-1) refere-se a cada saída que é realimentada, h0=1 e C é o número de neurônios da camada de saída. A atualização dos pesos relativos as recorrências é dada por:

$$\Delta w3_{mk}(t+1) = \Delta w3_{mk}(t) - \eta \frac{\partial \varepsilon_n}{\partial w3(t)_{mk}} \tag{B2}$$

Desenvolvendo-se a equação B2:

$$\frac{\partial \varepsilon_n}{\partial w3_{mk}(t)} = \frac{1}{2} \sum_{m=0}^{C} \frac{\partial \varepsilon_{nm}}{\partial o_m} \frac{\partial o_m}{\partial h_k} \frac{\partial h_k}{\partial w3_{mk}} \tag{B3}$$

$$\frac{\partial \varepsilon_{nm}}{\partial o_m} = \frac{\partial \varepsilon_{nm}}{\partial o_m} = -2(y_{nm}(t) - o_m(t)) \tag{B4}$$

$$\frac{\partial o_m}{\partial h_k} = (1 - o_m).o_m.w2_{km} \tag{B5}$$

$$\frac{\partial h_k}{\partial w3_{mk}} = \frac{\partial}{\partial w3_{mk}} \left[\frac{1}{1 + e^{-\sum_{m=0}^{C} om\ (t-1).w3(t)_{mk} + \sum_{j=0}^{A} x_{nj}(t).w1(t)_{jk}}} \right] \tag{B6}$$

$$u_k(t) = 1 + e^{-\sum_{m=0}^{C} om\ (t-1).\ w3(t)_{mk} + \sum_{j=0}^{A} X_{nj}\ (t).\ w1(t)_{jk}} \tag{B7}$$

$$u_k(t) - 1 = e^{-\sum_{m=0}^{C} om(t-1).w3(t)_{mk} + \sum_{j=0}^{A} Xnj(t).w1(t)_{jk}} \tag{B8}$$

$$h_k(t) = \frac{1}{uk(t)}, \frac{\partial h_k(t)}{\partial u(t)} = -\frac{1}{uk\ (t)^2}, \tag{B9}$$

$$\frac{\partial h_k(t)}{\partial w3(t)_{mk}} = \frac{\partial h_k(t)}{\partial u_k(t)} \frac{\partial u_k(t)}{\partial w3(t)_{mk}} \tag{B10}$$

$$\frac{\partial h_k(t)}{\partial w3(t)_{mk}} = (u_k(t) - 1)(-om(t-1)) \tag{B11}$$

$$\frac{\partial h_k(t)}{\partial w3_{mk}(t)} = -\frac{1}{u_k(t)^2}(u_k(t) - 1).(om(t-1)) \tag{B12}$$

$$\frac{\partial h_k(t)}{\partial w3mk(t)} = \frac{(\frac{1}{h_k(t)} - 1)}{(\frac{1}{h_k(t)})^2}.om\ (t-1) \tag{B13}$$

$$\frac{\partial h_k(t)}{\partial w3_{mk}(t)} = (1 - h_k(t))\, h_k(t).om\ (t-1) \tag{B14}$$

$$\frac{\partial \varepsilon_n}{\partial w3_{mk}(t)} = (1 - h_k(t))\, h_k(t) \sum_{m=0}^{C} (o_m(t) - y_{nm}(t))(1 - o_m(t)).o_m(t).om(t-1).w2_{km}(t) \tag{B15}$$

Logo

$$\Delta w3_{mk}(t+1) = \Delta w3_{mk}(t) + \eta(1 - h_k(t))h_k(t) \sum_{m=0}^{C} (y_{nm}(t) - o_m(t))(1 - o_m(t))o_m(t).om(t-1)w2_{km}(t) \tag{B16}$$

As equações B17 e B18 representam as atualizações dos pesos, da camada de saída e da camada intermediária, respectivamente.

$$\Delta w2_{km}(t+1) = \Delta w2_{km}(t) + \eta\ [(y_{nm} - o_m)(1 - o_m).o_m].h_k \tag{B17}$$

$$\Delta w1_{jk}(t+1) = \Delta w1_{jk}(t) + \eta\ [(1 - h_k)h_k . \sum_{m=0}^{C} (y_{nm} - o_m)(1 - o_m).o_m.w2_{km}]x_{nj} \tag{B18}$$

ANEXO C - Mapeamento genótipo/fenótipo

Tabela C.1-Rede Direta (ENTRADAS,SAÍDAS)=(1,1) – Exemplos 1 e 2

String Anterior	Regra	String Posterior
●	●→ f (R1)	f
f	f → [f (R3.1)	[f
[f	[→ [F f] (R4) 1x	[F f] f NR=1
[F f] f	f → f F f (R 3.2) 1x	[F f F f] f NSAI=1
Onde: NSAI=Número de saídas 1x regra repetida uma vez NR= Neurônios por Ramificação		 [F f Ff] f
String Anterior	**Regra**	**String Posterior**
●	●→ f (R1)	f
f	f → [f (R3.1)	[f
[f	[→ [F f] (R4) 1x	[F f] f NR=1
[F f] f	[→ [F f] (R4) 1x	[F f] F f] f NR=2
[F f] F f] f	f → f Ff (R3.2) 1x	[F f F f] F f] f NSAI=1
[F f F f] F f] f	f→ f F (R3.3) 1x	[F f F f] F f F] f NSAI=1
Onde: NSAI=Número de saídas 1x regra repetida uma vez NR= Neurônios por Ramificação		
MODELO GERAL:	$[FfFf]^{1}....(FfF])^{NR-1}f$	

Tabela C.2-Rede Direta (ENTRADAS,SAÍDAS)=(1,N) - Exemplo 1

String	Regra	String Posterior
●	●→ f (R1)	f
f	f → [f (R3.1)	[f
[f	[→ [F f] (R4) 1x	[F f] f NR=2
[F f] f	[→ [F f] (R4) 1x	[F f] F f] f NR=2
[F f] F f] f	f → f F f (R3.2) 1x	[F f F f] F f] f NSAI=2
[F f F f] F f] f	f → f F f (R3.2) 1x	[F f F f] F f F f] f NSAI=2
[F f F f] F f F f] f	f → f B (R3.6) 1x	[F f F f B] F f F f] f NSAI=2
[F f F f B] F f F f] f	f → f B (R3.6) 1x	[F f F f B] F f F f B] f NSAI=2
[F f F f B] F f F f B] f **Onde:** NSAI=Número de saídas 1x regra repetida uma vez NR= Neurônios por Ramificação		

Tabela C.3 - Rede Direta (ENTRADAS,SAÍDAS)=(1,N) – Exemplo 2

String Anterior	Regra	String Posterior	
●	● → f (R1)	f	
f	f → [f (R3.1)	[f	
[f	[→ [F f] (R4)	[F f] f	NR=3
[F f] f	[→ [F f] (R4)	[F f] F f] f	NR=3
[F f] F f] f	[→ [F f] (R4)	[F f] F f] F f] f	NR=3
[F f] F f] F f] f	f → f F f (R3.2)	[F f F f] F f] F f] f	NSAI=3
[F f F f] F f] F f] f	f → f F f (R3.2)	[F f F f] F f F f] F f] f	NSAI=3
[F f F f] F f F f] F f] f	f → f F f (R3.2)	[F f F f] F f F f] F f] f	NSAI=3
[F f F f] F f F f] F f F f] f	f → f B (R3.6)	[F f F f B] F f F f] F f F f] f	NSAI=3
[F f F f B] F f F f] F f F f] f	f → f B (R3.6)	[F f F f B] F f F f B] F f F f] f	NSAI=3
[F f F f B] F f F f B] F f F f] f	f → f B (R3.6)	[F f F f B] F f F f B] F f F f B] f	NSAI=3
Onde: NSAI=Número de saídas 1x regra repetida uma vez NR= Neurônios por Ramificação NENT=Número de entradas			
MODELO GERAL:	[(F f F f B])^{NR} f		

Tabela C.4-Rede Direta (ENTRADAS,SAIDAS)=(N,1) – Exemplo 1

String Anterior		Regra		String Posterior	
●	●	●→ f	● → f R1	f	f
f	f	f → [f	f → [f R3.1	[f	[f
[f	[f	[→ [F f]	[→ [F f] R4	[F f] f	[F f] f NR=1 NR=1
[F f] f	[F f] f	f → f F f		[F f F f] f	[F f] f NSAI=1
[F f F f] f	[F f] f	f → f B	f → f B R3.6	[F f F f B] f	[F f B] f NENT=2
[F f F f B] f	[F f B] f	f → f B	f → f B R3.6	[F f F f B] f B	[F f B] fB NENT=2
MODELO GERAL:		[FfFfB](FfB)$^{NR1-1}$]fB ..... ([(FfB])^{NRn} fB)$^{NENT-1}$			

Tabela C.5-Modelo Geral Rede Direta (ENTRADAS,SAIDAS)=(N,M)

MODELO GERAL:	[FfFfB][(FfB])$^{NR1-1}$ fB ..... [FfFfB][(FfB])$^{NRn-1}$fB

Tabela C.6-Rede Recorrente (ENTRADAS,SAÍDAS)=(1,1)

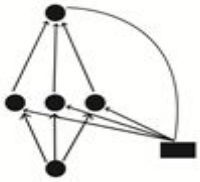
String Anterior	Regra	String Posterior
●	●→ f (R1)	●
f	f → [f (R3.1)	[f
[f	[→ [F f] (R4) NR Vezes	[F f] F f] … F f] f O termo Ff] repete-se NR vezes
[Ff]Ff] ... Ff] f	f → f F f (R3.2) 1x 1º f	[F fFf]F f] ..F f]f
[FfFf]Ff] ... Ff]f	f → f B (R.3.6) 1x 1º f	[FfFfB]Ff] ... Ff]f
[FfFfB]Ff] .. Ff]f	f → f B (R.3.6) 1x Todos os f's	[FfFfBB]FfB]...FfB]fB
MODELO GERAL: [FfFfBB](FfB])$^{NR-1}$ fB **Onde:** NR= Neurônios por Ramificação 1x = uma vez		

Tabela C.7-Modelos Gerais para Redes Recorrentes (1,N),(N,1)(N.M)

(ENTRADAS,SAÍDAS)=(1,N)	**MODELO GERAL:**	$[(FfFfBB])^{NSAI}(FfB])^{NR-NSAI}f$
(ENTRADAS,SAÍDAS)=(N,1)	**MODELO GERAL:**	$[FfFfBB] ....(FfB])^{NR-1}fB \quad ([(FfB])^{NRn}fB)^{NENT-1}$
(ENTRADAS,SAÍDAS)=(N,M)	**MODELO GERAL:**	$[(FfFfBB])^{NSAI}(FfB])^{NR-NSAI}fB \quad ([(FfB])^{NRn}fB)^{NENT-1}$

ANEXO D – Resultados dos Testes Estatísticos para problemas de Classificação

Tabela D.1 – Resultados dos testes estatísticos, teste-t e Shapiro-Wilk (SW) obtidos para algoritmos TIPO I.

		GEGA			ADEANN			G3PCX			GA		
		ADEANN	**G3PCX**	**GA**	**GEGA**	**G3PCX**	**GA**	**GEGA**	**ADEANN**	**GA**	**GEGA**	**ADEANN**	**G3PCX**
BC (I) C1	Teste-t	$4.37E^{-13}$	$4.32E^{-9}$	$1.084E^{-11}$	$4.37E^{-13}$	$2.97E^{-6}$	$4.14E^{-7}$	$4.32E^{-9}$	$2.57E^{-16}$	$2.113E^{-9}$	$1.084E^{-11}$	$4.14E^{-7}$	$2.113E^{-9}$
	S-Wilk	0.09315	0.8976	0.7595	0.09315	0.1517	0.1517	0.89760	0.1517	0.5425	0.7595	0.1517	0.5425
	Pontos	0	0	0	2	2	2	2	0	2	2	0	0
IF (I) C1	Teste-t	$6.474E^{-15}$	0.00136	$2.2\ E^{-16}$	$6.474E^{-15}$	$3.106E^{-7}$	$3.47E^{-9}$	0.00136	$3.106E^{-7}$	$2.58E^{-10}$	$2.2\ E^{-16}$	$3.47E^{-9}$	$2.58E^{-10}$
	S-Wilk	0.6336	0.07963	0.1071	0.6336	0.3029	0.1555	0.07963	0.3029	0.899	0.1071	0.1555	0.893
	Pontos	2	2	2	0	2	2	0	0	2	0	0	0
HD (I) C1	Teste-t	0.04019	$2.2E^{-16}$	$5.98E^{-16}$	0.04019	$5.108E^{-14}$	$1.57E^{-10}$	$2.2E^{-16}$	$5.108E^{-14}$	$1.2115E^{-6}$	$5.98E^{-16}$	$1.57E^{-10}$	$1.2115E^{-6}$
	S-Wilk	0.07832	0.167	0.167	0.07832	0.167	0.167	0.167	0.167	0.2556	0.167	0.167	0.2556
	Pontos	0	0	0	2	0	0	2	2	2	2	2	0
BC (I) C2	Teste-t	$2.2E^{-16}$	$1.269E^{-9}$	$4.036E^{-7}$	$2.2E^{-16}$	$1.178E^{-9}$	$7.83E^{-10}$	$1.269E^{-9}$	$1.178E^{-9}$	$2.55E^{-8}$	$4.036E^{-7}$	$7.83E^{-10}$	$2.55E^{-8}$
	S-Wilk	0.167	0.3271	0.3271	0.167	0.3271	0.3271	0.3271	0.3271	0.3271	0.3271	0.3271	0.3271
	Pontos	2	2	0	0	0	0	0	2	0	2	2	2
IF (I) C2	Teste-t	$1.52E^{-9}$	0.2587	$1.51E^{-13}$	$1.52E^{-9}$	$1.52E^{-8}$	$6.59E^{-8}$	0.2587	$1.52E^{-8}$	$7.95E^{-8}$	$1.51E^{-13}$	$6.59E^{-8}$	$7.95E^{-8}$
	S-Wilk	0.3271	0.3271	0.1671	0.3271	0.3271	0.3271	0.3271	0.3271	0.1026	0.1671	0.3271	0.1026
	Pontos	0	1	0	2	2	2	1	0	0	2	0	2
HD (I) C2	Teste-t	$3.65E^{-9}$	$4.855E^{-9-}$	0.1341	$3.65E^{-9}$	$2.21E^{-9}$	$7.791E^{-9}$	$4.855E^{-9-}$	$2.21E^{-9}$	$2.038E^{-10}$	0.1341	$7.791E^{-9}$	$2.038E^{-10}$
	S-Wilk	0.3585	0.3589	0.7904	0.3585	0.1412	0.1412	0.3589	0.1412	0.1025	0.7904	0.1412	0.1025
	Pontos	0	2	1	2	2	2	0	0	0	1	0	2
BC (I) C3	Teste-t	$4.629E^{-9}$	$2.308E^{-9}$	$1.286E^{-8}$	$4.629E^{-9}$	$3.72E^{-6}$	$2.2E^{-16}$	$2.308E^{-7}$	$3.72E^{-6}$	0.805	$1.286E^{-8}$	$2.2E^{-16}$	0.805
	S-Wilk	0.893	0.893	0.167	0.893	0.1161	0.167	0.8931	0.1161	0.167	0.167	0.167	0.167
	Pontos	2	2	2	0	0	0	0	2	1	0	2	1
IF (I) C3	Teste-t	$1.371E^{-9}$	$4.28E^{-10}$	$6.278E^{-10}$	$1.371E^{-9}$	$3.347E^{-7}$	$3.338E^{-7}$	$4.28E^{-10}$	$3.347E^{-7}$	0.805	$6.278E^{-10}$	$3.338E^{-7}$	0.805
	S-Wilk	0.3271	0.3271	0.1161	0.3271	0.1025	0.1161	0.3271	0.1025	0.167	0.1161	0.1161	0.167
	Pontos	2	2	2	0	2	2	0	0	1	0	0	1
HD (I) H3	Teste-t	$2.43E^{-10}$	$3.357E^{-8}$	$4.95E^{-8}$	$2.43E^{-10}$	$1.058E^{-94}$	$1.382E^{-11}$	$3.357E^{-8}$	$1.058E^{-94}$	0.805	$4.95\ E^{-8}$	$1.382E^{-11}$	0.805
	S-Wilk	0.7595	0.7595	0.7818	0.7595	0.7818	0.2319	0.7595	0.7818	0.167	0.7818	0.2319	0.167
	Pontos	2	2	2	0	0	0	0	2	1	0	2	1

Tabela D.2 – Resultados dos testes estatísticos, teste-t e Shapiro-Wilk (SW) obtidos para algoritmos TIPO II.

		MM			ADEANN			GRM			GE-BP		
		ADEANN	**GRM**	**GE-BP**	**MM**	**GRM**	**GE-BP**	**MM**	**ADEANN**	**GE-BP**	**MM**	**ADEANN**	**GRM**
BC (II) C1	Teste-t	$1.87E^{-12}$	$3.066E^{-5}$	$4.25\ E^{-12}$	$1.87E^{-12}$	$8.47E^{-6}$	$5.49E^{-14}$	$3.066E^{-5}$	$8.47E^{-6}$	$4.8275E^{-7}$	$4.25\ E^{-12}$	$5.49E^{-14}$	$4.8275E^{-7}$
	S-Wilk	0.1025	0.09259	0.167	0.1025	0.3721	0.167	0.09259	0.3721	0.1025	0.167	0.167	0.1025
	Pontos	0	2	2	2	2	2	0	0	2	0	0	0
IF (II) C1	Teste-t	$7.144E^{-12}$	$1.17E^{-10}$	$1.765\ E^{-12}$	$7.144E^{-12}$	$2.37E^{-9}$	$3.693E^{-13}$	$1.17E^{-10}$	$2.37E^{-9}$	$4.425E^{-14}$	$1.765\ E^{-12}$	$3.693E^{-13}$	$4.425E^{-14}$
	S-Wilk	0.3271	0.1258	0.3343	0.3271	0.1242	0.167	0.1258	0.1242	0.167	0.3343	0.167	0.167
	Pontos	0	0	0	2	2	2	2	0	0	2	0	2
HD (II) C1	Teste-t	$2.98E^{-9}$	$2.120E^{-6}$	$2.05\ E^{-12}$	$2.98E^{-9}$	$2.2E^{-16}$	$7.77E^{-13}$	$2.120E^{-6}$	$2.2E^{-16}$	$1.2115E^{-6}$	$2.05\ E^{-12}$	$7.77E^{-13}$	$1.2115E^{-6}$
	S-Wilk	0.1161	0.1025	0.1161	0.1161	0.1967	0.1161	0.1025	0.1967	0.2556	0.1161	0.1161	0.2556
	Pontos	0	2	0	2	2	2	0	0	0	2	0	2
BC (II) C2	Teste-t	-	-	-	-	$8.02E^{-10}$	$7.052E^{-16}$	-	$8.02E^{-10}$	$5.99E^{-8}$	-	$7.052E^{-16}$	$5.99E^{-9}$
	S-Wilk	-	-	-	-	0.2617	0.1557	-	0.2617	0.1013	-	0.1557	0.1013
	Pontos	0	0	0	0	0	0	0	2	0	0	2	2
IF (II) C2	Teste-t	-	-	-	-	-	$1.71E^{-10}$	-	-	-	-	$1.71E^{-10}$	-
	S-Wilk	-	-	-	-	-	0.1161	-	-	-	-	0.1161	-
	Pontos	0	0	0	0	0	2	0	0	0	0	0	0
HD (II) C2	Teste-t	-	-	-	-	-	$3.621E^{-13}$	-	-	-	-	$3.621E^{-13}$	-
	S-Wilk	-	-	-	-	-	0.1161	-	-	-	-	0.1161	-
	Pontos	0	0	0	0	0	0	0	0	0	0	2	0
BC (II) C3	Teste-t	2.10^{-16}	$9.67E^{-16}$	$6.25E^{-14}$	$2.2E^{-16}$	$4.68E^{-14}$	$2.24E^{-16}$	$9.67E^{-16}$	4.68^{-14}	$2.851E^{-16}$	$6.25E^{-14}$	$2.24E^{-16}$	$2.851E^{-16}$
	S-Wilk	0.1074	0.1161	0.1161	0.1074	0.1025	0.3271	0.1161	0.1025	0.1476	0.1161	0.3271	0.1476
	Pontos	0	2	0	2	2	2	0	0	0	2	0	2
IF (II) C3	Teste-t	$7.592E^{-15}$	$1.56E^{-15}$	$4.17\ E^{-13}$	$7.59E^{-15}$	$1.71E^{-14}$	$5.73E^{-16}$	$1.56E^{-15}$	$1.71E^{-14}$	$7.82E^{-16}$	$4.17\ E^{-13}$	$5.73E^{-16}$	$7.82E^{-16}$
	S-Wilk	0.1465	0.1645	0.2075	0.1465	0.1732	0.1736	0.1645	0.1732	0.2164	0.2075	0.1736	0.2164
	Pontos	0	2	0	2	2	2	0	0	0	2	0	2
HD (II) H3	Teste-t	$3.77E^{-16}$	$2.27E^{-16}$	$1.029\ E^{-13}$	$3.77E^{-18}$	$1.25E^{-14}$	$2.47E^{-13}$	$2.27E^{-16}$	$1.25E^{-14}$	$2.2E^{-16}$	$1.029\ E^{-13}$	$2.47E^{-13}$	$2.2E^{-16}$
	S-Wilk	0.1161	0.1027	0.107	0.1161	0.101	0.101	0.1027	0.101	0.1161	0.107	0.101	0.1161
	Pontos	0	2	2	0	2	2	2	0	0	0	0	2

Tabela D.3 – Resultados dos testes estatísticos, teste-t e Shapiro-Wilk (SW) obtidos para algoritmos TIPO III.

		GEGA			ADEANN			GE			GP		
		ADEANN	**GE**	**GP**	**GEGA**	**GE**	**GP**	**GEGA**	**ADEANN**	**GP**	**GEGA**	**ADEANN**	**GE**
BC (III) C1	Teste-t	$2.32E^{-13}$	$9.78E^{-8}$	$4.801\ E^{-7}$	$2.32E^{-13}$	$2.749E^{-8}$	$1.056E^{-7}$	$9.78E^{-6}$	$2.749E^{-8}$	$5.175E^{-6}$	$4.801E^{-13}$	$1.056E^{-6}$	$5.175E^{-6}$
	S-Wilk	0.167	0.3271	0.1025	0.167	0.07924	0.106	0.3721	0.07924	0.118	0.1025	0.106	0.118
	Pontos	0	2	0	2	2	2	0	0	0	2	0	2
IF (III) C1	Teste-t	$6.236E^{-6}$	$7.76E^{-9}$	$2.011\ E^{-9}$	$6.236E^{-6}$	$6.156E^{-13}$	$2.53E^{-7}$	$7.76E^{-6}$	$6.156E^{-13}$	$5.25E^{-9}$	$2.011E^{-9}$	$2.53E^{-6}$	$5.25E^{-5}$
	S-Wilk	0.0963	0.0988	0.1209	0.0963	0.3202	0.01147	0.0988	0.3202	0.08368	0.1209	0.01147	0.08368
	Pontos	2	2	2	0	2	0	0	0	0	0	2	2
HD (III) C1	Teste-t	$8.93E^{-8}$	$7.40E^{-9}$	$4.54\ E^{-7}$	$8.93E^{-8}$	$4.329E^{-9}$	$4.66E^{-7}$	$7.40E^{-9}$	$4.329E^{-9}$	$2.525E^{-7}$	$4.54E^{-9}$	$4.66E^{-7}$	$2.52E^{-7}$
	S-Wilk	0.079	0.08973	0.07918	0.079	0.1866	0.1804	0.08973	0.1866	0.1813	0.07918	0.1804	0.1813
	Pontos	0	2	2	2	2	2	0	0	0	0	0	2
BC (III) C2	Teste-t	4.66^{-14}	0.002054	$7.74\ E^{-6}$	$4.66E^{-14}$	1.37^{-7}	$2.2E^{-166}$	0.002054	$1.379E^{-7}$	$5.35E^{-9}$	$7.74E^{-6}$	$2.2E^{-16}$	$5.35E^{-9}$
	S-Wilk	0.9606	0.08994	0.09918	0.9606	0.09032	0.1079	0.08994	0.00932	0.2132	0.09918	0.1079	0.2132
	Pontos	2	2	2	0	0	0	0	2	2	0	2	0
IF (III) C2	Teste-t	$1.35E^{-7}$	$5.014E^{-7}$	$2.153\ E^{-6}$	$1.356E^{-7}$	$4.68E^{-6}$	$1.22E^{-8}$	$5.014E^{-7}$	4.68^{-6}	$2.39E^{-8}$	$2.15E^{-6}$	$1.22E^{-8}$	$2.39E^{-8}$
	S-Wilk	0.0832	0.08016	0.08224	0.0832	0.2077	0.1275	0.08016	0.2077	0.129	0.08224	0.1275	0.129
	Pontos	2	2	2	0	0	2	0	2	2	0	0	0
HD (III) C2	Teste-t	2.2^{-16}	$7.45E^{-8}$	$2.42\ E^{-7}$	2.2^{-16}	$7.30E^{-10}$	$2.66E^{-16}$	$7.45E^{-8}$	$7.30E^{-10}$	$1.74E^{-10}$	$2.424E^{-7}$	$2.2E^{-167}$	$1.742E^{-10}$
	S-Wilk	0.001352	0.2057	0.1886	0.001352	0.1866	0.2057	0.2057	0.1866	0.2263	0.1886	0.2057	0.2263
	Pontos	2	2	2	0	0	0	0	2	0	0	2	2
BC (III) C3	Teste-t	2.10^{-10}	0.1734	0.6278	2.10^{-10}	$1.16E^{-11}$	$1.14E^{-11}$	0.1734	$1.16E^{-11}$	0.07594	0.6278	$1.14E^{-11}$	0.07594
	S-Wilk	0.1891	0.1671	0.2373	0.1891	0.1033	0.1252	0.1671	0.1033	0.1252	0.2372	0.1252	0.1252
	Pontos	0	1	1	2	2	2	1	0	1	1	0	1
IF (III) C3	Teste-t	$5.322E^{-12}$	0.0751	$6.80\ E^{-13}$	$5.32E^{-12}$	$5.38E^{-12}$	$2.2E^{-16}$	0.0751	$5.38E^{-12}$	$2.2E^{-16}$	$6.8\ E^{-13}$	$2.2E^{-16}$	$2.2E^{-16}$
	S-Wilk	0.1252	0.1363	0.1011	0.1252	0.1011	0.1714	0.1363	0.1011	0.4589	0.1011	0.1714	0.4589
	Pontos	0	1	2	2	2	2	1	0	2	0	0	0
HD (III) H3	Teste-t	$2.2E^{-16}$	0.1381	$2.31\ E^{-13}$	$2.2E^{-16}$	$1.097E^{-13}$	$2.2E^{-16}$	0.1381	$1.097E^{-13}$	$2.2E^{-16}$	$2.31\ E^{-13}$	$2.2E^{-16}$	$2.2E^{-16}$
	S-Wilk	0.1161	0.07788	0.1012	0.1161	0.1012	0.4038	0.0077	0.1012	0.2096	0.1012	0.1161	0.2096
	Pontos	0	1	0	2	2	2	1	0	2	0	0	2

ANEXO E – (Teste-t) para duas amostras independentes

Esse teste estatístico é aplicado sempre que se deseja comparar as médias de uma variável quantitativa em dois grupos diferentes de sujeitos e se desconhecem as suas respectivas variâncias. Considera-se que as variâncias das populações são iguais, porém desconhecidas, ou seja $\sigma_1^2=\sigma_2^2=\sigma^2$.

A) Procedimento para realização do Teste-t

1) **Estabelecem-se as Hipóteses**

$$H_0: \mu_1 = \mu_2$$
$$H_1: \mu_1 \neq \mu_2$$

2) **Calcula-se o Desvio Padrão agrupado (*Sp*)**

$$Sp = \sqrt{\frac{(n_1 - 1)\ x\ S_1^2 + (n_2 - 1)\ x\ S_2^2}{n_1 - n_2 - 2}}$$

Onde : n_1 e n_2 são os tamanhos das amostras 1 e 2 e S_1 é do desvio padrão da amostra 1 e S_2 é desvio padrão da amostra 2.

3) **Estabelecer o nível de significância do teste (α=0.05, 95%)**

4) **Encontrar na tabela de distribuição (t-Student) os valores críticos (Tcrítico)** . Considerando b=$n_1 - n_2 - 2$ graus de liberdades. A Tabela E.1 ilsutra um exemplo.

Tabela E.1 – Ilustração da tabela de distribuição t-Student, destacando o valor critico (Tcrítico=5.841) considerando nível de confiança de 95% α=0.05 e três graus de liberdade b=3.

	A	0.05	0.01
B			
1		63.656	
2		9.925	
3		**5.841**	
...		...	
....		.....	
N		.....	

5) **Calcula-se o valor da estatística (Tobs)**

$$Tobs = \frac{(\bar{x} - \bar{y})}{Sp\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

6) **Se *Tobs* < *Tcrítico (valor tabelado obtido no item 4)* .** Rejeita-se a hipótese nula $(H_0: \mu_1 = \mu_2)$, ou seja, estatíticamente as médias não são iguais (aceita-se $H_1: \mu_1 \neq \mu_2$).

OBS: Em termos práticos o passo 6 é resolvido pelo software estatístico "R" consderando o cáculo da estatística "p-value". Assim: Se "p-value>α" aceita-se $(H_0: \mu_1 = \mu_2)$, caso contrário "p-value≤α" rejeita-se H_0 e aceita-se $(H_1: \mu_1 \neq \mu_2)$.

ANEXO F – (Teste de Normalidade de Shapiro - Wilk)

O teste de Shapiro-wilk, calcula a variável estatística (w) que investiga se uma amostra aleatória provém de uma distribuição normal.

A) Procedimento para realização do Teste de Shapiro - Wilk

1) **Estabelecem-se as Hipóteses**

   H_0: A amostra provém de uma distribuição Normal.

   H_1: A amostra não provém de uma distribuição Normal.

2) **Estabelecer o nível de significância do teste (α=0.05, 95%)**

3) **Calcula-se a estatística de Teste:**

3.1) Ordenar as n observações das duas amostras : x[1], x[2], x[3],.....,x[n]

3.2) Calcular $\sum_{i=1}^{n}(x_i - \overline{x})^2$

3.3)Calcular b:

$$b = \begin{cases} \sum_{i=1}^{\frac{n}{2}} a_{n-i+1} \ (x_{n-i+1} - x_i), \text{se número de elementos da amostra (n) é par} \\ \sum_{i=1}^{(n+1)/2} a_{n-i+1} \ (x_{n-i+1} - x_i), \text{se número de elementos da amostra (n)é impar} \end{cases}$$

Em que a_{n-i+1} são valores gerados pelas médias, variâncias e covariâncias das estatísticas de ordem de uma amostra de tamanho n de uma distribuição normal. Seus valores são tabelados de acordo com a Tabela F.1

Tabela F.1 – Ilustração da tabela para cálculo do elemento a_{n-i+1} destacando-se o valor 0,5379 para i=1 e n-i+1=10 para n=10.

i\n-i+1	2	3	4	5	6	7	8	9	10	11	12
1	0,7071	0,7071	0,6872	0,6646	0,6431	0,6233	0,6062	0,5888	**0,5739**	0,5601	0,5475
2			0,1677	0,2413	0,2806	0,3031	0,3164	0,3244	0,3291	0,3315	0,3325
3					0,0875	0,1401	0,1743	0,1976	0,2141	0,2260	0,2347
4							0,0561	0,0947	0,1224	0,1429	0,1586
5									0,0399	0,0695	0,0922
6											0,0303

3.4) Calcular W;

$$W = \frac{b^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

4) Se W > $W(\alpha, n)$. Assim, podemos afirmar com nível de significância de 5% que a amostra provém de uma população normal. O valor de $W(\alpha, n)$ é tabelado para α (nível de confiança) e para n (tamanho da amostra).

OBS: O Passo 3.1 pode-se ser generalizado para "n" amostras de acordo com a soma de n variáveis aleatórias com distribuição normal. O que é provado abaixo:

$$X_1 \sim N(\mu_1, \sigma_1^2)$$

$$X_2 \sim N(\mu_2, \sigma_2^2)$$

.....

$$X_n \sim N(\mu_n, \sigma_n^2)$$

$$Y \sim N(\mu_1 + \mu_2 + \cdots + \mu_n, \ \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_n^2)$$

$$E(Y) = E(X_1 + X_2 + \ldots + X_n) = E(X_1) + E(X_2) + \ldots + E(X_n)$$

$$Var(Y) = \sigma_1^2 + \sigma_2^2 + \ldots + \sigma_n^2$$

OBS: Em termos práticos o passo 4 é resolvido pelo software estatístico "R" considerando o cáculo da estatística "p-value". Assim: Se "p-value>α" aceita-se (H_0: A amostra provém de uma distribuição Normal), caso contrário "p-value≤α" rejeita-se H_0 e aceita-se (H_1: A amostra não provém de uma distribuição Normal).

ANEXO G – Resultados dos testes estatísticos realizados para Predição de Séries Temporais

Tabela G.1 – Resultados dos testes estatísticos, teste-t e Shapiro-Wilk (SW) obtidos para a predição das series temporais Passageiros (TFI), Temperatura (TFII) e Down-Jones (TFIII).

TFI		ADANN				ARIMA				UCM				FP				ADEANN			
		ARIMA	**UCM**	**FP**	**ADEANN**	**ADANN**	**UCM**	**FP**	**ADEANN**	**ADANN**	**ARIMA**	**FP**	**ADEANN**	**ADANN**	**ARIMA**	**UCM**	**ADEANN**	**ADANN**	**ARIMA**	**UCM**	**FP**
SMAPE	t-test	4.062E^{-8}	1.20E^{-11}	4.122E^{-13}	1.019E^{-11}	4.062E^{-8}	5.078E^{-7}	1.47E^{-11}	4.321E^{-11}	1.20E^{-11}	5.078E^{-7}	9.79E^{-12}	3.67E^{-10}	4.12E^{-13}	1.47E^{-11}	9.79E^{-12}	7.856E^{-10}	1.02E^{-11}	4.32E^{-11}	3.67E^{-10}	7.856E^{-12}
	SW	0.167	0.2279	0.1713	0.1654	0.167	0.1389	0.129	0.1278	0.2279	0.1389	0.3623	0.4736	0.1713	0.129	0.3623	0.1256	0.1654	0.1278	0.4736	0.1256
	Points	2	0	2	0	0	0	2	0	2	2	2	0	0	0	0	0	2	2	2	2
MSE	t-test	2.488E^{-9}	6.603E^{-9}	1.191E^{-5}	7.44E-6	2.488E^{-9}	4.91E^{-5}	0.0001856	0.00066	6.603E^{-9}	4.91E^{-5}	1.677E^{-10}	2.2E^{-16}	1.191E^{-5}	0.000185	1.677E^{-10}	2.92E^{-14}	7.44E^{-6}	0.0006606	2.2E^{-16}	2.92E^{-14}
	SW	0.2927	0.9518	0.1274	0.1399	0.2927	0.4615	0.2202	0.108	0.9518	0.4615	0.108	0.2605	0.1274	0.2202	0.108	0.108	0.1399	0.108	0.2605	0.108
	Points	0	0	2	0	2	0	0	0	2	2	2	2	0	2	0	0	2	2	0	2
TFII																					
		ARIMA	**UCM**	**FP**	**ADEANN**	**ADANN**	**UCM**	**FP**	**ADEANN**	**ADANN**	**ARIMA**	**FP**	**ADEANN**	**ADANN**	**ARIMA**	**UCM**	**ADEANN**	**ADANN**	**ARIMA**	**UCM**	**FP**
SMAPE	t-test	2.046E^{-9}	4.468E^{-10}	2.47E^{-6}	1.131E^{-7}	2.046E^{-9}	1.905E^{-5}	2.208E^{-5}	1.906E^{-6}	4.468E^{-10}	1.905E^{-5}	7.295E-7	4.021E^{-10}	2.47E^{-6}	2.208E^{-5}	7.295E^{-7}	2.83E^{-14}	1.131E^{-7}	1.906E^{-6}	4.021E^{-10}	2.83E^{-14}
	SW	0.167	0.549	0.1111	0.1011	0.167	0.07864	0.7864	0.7864	0.549	0.07864	0.07864	0.07888	0.1111	0.07864	0.07864	0.07887	0.1011	0.07864	0.07887	0.07887
	Points	2	0	0	0	0	0	0	0	2	2	2	0	2	2	0	0	2	2	2	2
MSE	t-test	0.004079	0.0004408	0.0006343	0.0002236	0.004079	5.165E^{-8}	9.763E^{-7}	9.239E^{-8}	0.0004408	5.16E^{-3}	0.00024	7.093E^{-5}	0.0001343	9.763E^{-7}	0.00024	7.093E^{-5}	0.0002236	9.239E^{-8}	7.093E^{-5}	9.78E^{-12}
	SW	0.0806	0.08045	0.07787	0.07786	0.0806	0.07786	0.137	0.08332	0.08045	0.07786	0.38	0.1188	0.07787	0.137	0.38	0.1188	0.07786	0.08332	0.1188	0.2279
	Points	0	0	0	0	2	0	0	0	2	2	0	0	2	2	2	0	2	2	2	2
TFIII																					
		ARIMA	**UCM**	**FP**	**ADEANN**	**ADANN**	**UCM**	**FP**	**ADEANN**	**ADANN**	**ARIMA**	**FP**	**ADEANN**	**ADANN**	**ARIMA**	**UCM**	**ADEANN**	**ADANN**	**ARIMA**	**UCM**	**FP**
SMAPE	t-test	1.192E^{-6}	1.298E^{-6}	1.299E^{-6}	4.212E^{-10}	1.192E^{-6}	0.3868	0.3288	2.437E^{-9}	1.298E^{-6}	0.3868	0.3857	6.729E^{-9}	1.299E^{-6}	0.3288	0.3857	5.241E^{-9}	4.212E^{-10}	2.437E^{-9}	6.729E^{-9}	5.241
	SW	0.1643	0.19	0.1903	0.217	0.1643	0.7828	0.7828	0.2309	0.19	0.7828	0.7627	0.07914	0.1903	0.7828	0.7627	0.0875	0.217	0.2309	0.07914	0.0875
	Points	2	2	2	0	0	1	1	0	0	1	1	0	0	1	1	0	2	2	2	2
MSE	t-test	1.648E^{-5}	1.462E^{-5}	1.334E^{-5}	2.826E^{-5}	1.648E^{-5}	0.7109	0.9902	2.369E^{-5}	1.462E^{-5}	0.7109	0.5986	2.369E^{-5}	1.334E^{-5}	0.9902	0.05986	0.00011	2.826E^{-5}	2.369E^{-5}	2.369E^{-3}	0.00011
	SW	0.167	0.2456	0.1717	0.1356	0.167	0.2319	0.2381	0.1411	0.2456	0.2319	0.141	0.141	0.1717	0.2381	0.141	0.2512	0.1356	0.1411	0.1410	0.2512
	Points	2	2	2	0	0	1	1	0	0	1	0	0	0	1	1	0	2	2	2	2