UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Síntese automática de redes neurais artificiais com conexões à frente arbitrárias

Wilfredo Jaime Puma Villanueva

Orientador: Prof. Dr. Fernando José Von Zuben

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: **Engenharia de Computação**.

Campinas – São Paulo – Brasil
Dezembro, 2011

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

P968s

Puma Villanueva, Wilfredo Jaime
Síntese automática de redes neurais artificiais com conexões à frente arbitrárias / Wilfredo Jaime Puma Villanueva. --Campinas, SP: [s.n.], 2011.

Orientador: Fernando José Von Zuben.
Tese de Doutorado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes neurais . 2. Aprendizado do computador. 3. Computação evolutiva. 4. Previsão de series temporais. 5. Classificação. I. Von Zuben, Fernando José. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Automatic synthesis of artificial neural networks with arbitrary feedforward connections
Palavras-chave em Inglês: Neural networks, Machine learning, Evolutionary computation, Time series prediction, Classification
Área de concentração: Engenharia de Computação
Titulação: Doutor em Engenharia Elétrica
Banca examinadora: Guilherme de Alencar Barreto, Ricardo José Gabrielli Barreto Campello, Christiano Lyra Filho, Romis Ribeiro de Faissol Attux
Data da defesa: 07-12-2011
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidato: Wilfredo Jaime Puma Villanueva

Data da Defesa: 7 de dezembro de 2011

Título da Tese: "Síntese automática de redes neurais artificiais com conexões à frente arbitrárias"

Prof. Dr. Fernando José Von Zuben (Presidente): Fernando José Von Zuben

Prof. Dr. Guilherme de Alencar Barreto: Guilherme de Alencar Barreto

Prof. Dr. Ricardo José Gabrielli Barreto Campello: Ricardo Campello

Prof. Dr. Christiano Lyra Filho: _____

Prof. Dr. Romis Ribeiro de Faissol Attux: _____

Resumo

Esta tese apresenta duas metodologias de síntese automática de redes neurais artificiais com conexões à frente arbitrárias, com a proposição da arquitetura via computação evolutiva ou via um método construtivo, enquanto que os pesos sinápticos são definidos por técnicas de otimização não-linear. O processo de treinamento supervisionado visa parcimônia do modelo e máxima capacidade de generalização. Quando comparada a iniciativas similares encontradas na literatura, a versão construtiva da metodologia, denominada CoACFNNA, inova também ao permitir a síntese de arquiteturas mais flexíveis, com capacidade de mapeamento linear e não-linear, e ao promover baixo custo computacional. Este algoritmo construtivo parte de uma rede neural mínima, toma decisões de inserção/poda baseadas em análise de sensibilidade e em índices de informação mútua, relaxa o erro de treinamento para evitar convergência prematura e ajusta os pesos sinápticos via um método quasi-Newton com escalonamento automático. Estudos comparativos envolvendo abordagens alternativas baseadas em redes neurais, tais como MLPs, mistura heterogênea de especialistas, Cascade Correlation e a EPNet, baseada em programação evolutiva, indicam que a metodologia é promissora, tendo sido aplicada junto a problemas artificiais e reais, de classificação e de regressão.

Abstract

This thesis presents two methodologies for the automatic synthesis of artificial neural networks with arbitrary feed-forward connections, with the proposition of the architecture based on evolutionary computation and on a constructive method, whereas the synaptic weights are defined by nonlinear optimization techniques. The supervised learning process aims at parsimony of the model and maximum generalization capability. When compared to similar approaches in the literature, the constructive version of the methodology, denoted CoACFNNA, innovates also by allowing the synthesis of more flexible architectures, with linear and nonlinear mapping capability, and by promoting low computational cost. This constructive algorithm starts with a minimum neural network, takes decisions of insertion/pruning based on sensitivity analysis and also mutual information indices, relaxes the training error to avoid premature convergence, and adjusts the synaptic weights by means of a quasi-Newton method with automatic scaling. Comparative studies involving alternative approaches based on neural networks, such as MLPs, mixture of heterogeneous experts, cascade correlation and the EPNet, based on evolutionary programming, indicate that the proposal is promising, being applied to artificial and real problems, for classification and regression.

Dedicatória

A meus pais:
Justina Brígida e Demetrio Flavio
e irmãos:
Ronald Flavio e Luis Alfredo

Agradecimentos

A meus pais, por ser fruto do amor e do cuidado deles, e por me brindarem as condições necessárias de crescimento. A meus irmãos e familiares por estar presente sempre nos seus pensamentos.

Ao meu orientador e amigo, o Prof. Dr. Fernando José Von Zuben, por ter confiado e apostado em meu trabalho, por ser o guia neste caminho percorrido e pelo exemplo de pessoa comprometida com o trabalho.

Ao meu colega e amigo, Eurípedes P. dos Santos, pelas frutíferas conversas e trabalhos em conjunto, desde a época do meu mestrado, que contribuíram tanto em aspectos de forma como de conteúdo para este trabalho. Obrigado “Maestro”.

A todos os meus colegas do LBiC/DCA/FEEC/UNICAMP, pela amizade e respeito.

À Jeanne Dobgenski, por ter contribuído no caminho de minha chegada ao DCA/FEEC.

Aos professores da pós-graduação, pelo excelente trabalho que realizam.

Ao pessoal do setor administrativo, por estarem sempre dispostos a atender nossas solicitações da melhor forma.

À CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), pelo apoio financeiro.

Em geral à FEEC/Unicamp, às pessoas que a compõem, desde a diretoria e coordenação até o pessoal de serviços, por cuidarem das melhores condições para que os alunos desenvolvam seus projetos de pesquisa.

Conteúdo

Lista de Figuras

Capítulo 1

Capítulo 2

Capítulo 3

Capítulo 4

Capítulo 5

Capítulo 6

Lista de Tabelas

Capítulo 3

Capítulo 4

Capítulo 5

Lista de Abreviaturas

ACF	:	Função de Auto-correlação (*Auto-correlation Function*)
ACFNNs	:	Redes neurais com conexões à frente arbitrárias (*Arbitrarily Connected Feed-forward Neural Networks*)
BCP	:	*Barycentric Correction Procedure*
CasCorr	:	*Cascade Correlation*
CCA	:	*Curvilinear Component Analysis*
CoACFNNA	:	Algoritmo construtivo de ACFNN (*Constructive ACFNN algorithm*)
EP	:	Programação Evolutiva (*Evolutionary Programming*)
EPNet	:	*Evolutionary Programming Network*
ES	:	Estratégias Evolutivas (*Evolutionary Strategies*)
EQM	:	Erro Quadrático Médio (**MSE -** *Mean Square Error*)
EM	:	Maximização da Esperança (*Expectation Maximization*)
GA	:	Algoritmo Genético (*Genetic Algorithm*)
GAME	:	*Group of Adaptive Models Evolution*
ICA	:	*Independent Component Analisys*
MBP	:	Backpropagation Modificado (*Modified Backpropagation*)
ME	:	Mistura de Especialistas (*Mixture of Experts*)

MHE : Mistura de especialistas heterogêneos (*Mixture of Heterogeneous Experts*)

MI : Informação Mútua (*Mutual Information*)

MLP : Perceptron de multiplas camadas (*Multi-Layer Perceptron*)

MSE : *Mean Square Error*

PCA : Analise de Componentes Principais (*Principal Component Analysis*)

PSO : *Particle Swarm Optimization*

GP : Programação Genética (*Genetic Programming*)

PRM : *Pocket Ratchet Modification*

RBF : Funções de Base Radial (*Radial Basis Function*)

RNAs : Redes Neurais Artificiais

SA : Recozimento Simulado (*Simulated Annealing*)

STD : Desvio Padrão (*Standard Deviation*)

SVD : *Singular Value Decomposition*

TS : Busca Tabu (*Tabu Search*)

TSP : Problema do Caixeiro Viajante (*Traveling Salesperson Problem*)

Lista de Símbolos

X	:	Matriz das entradas dos padrões de treinamento, cada linha um padrão, cada coluna uma variável ou atributo.
Y	:	Vetor das saídas dos padrões de treinamento, cada linha esta associada ao valor da linha correspondente em **X.**
χ	:	Conjunto dos padrões de treinamento, incluído a saída. Exclui a **X** e **Y**
w^a	:	Representam aos pesos das conexões antes da camada oculta numa RNA MLP com uma camada oculta.
w^d	:	Representam aos pesos das conexões depois da camada oculta numa RNA MLP com uma camada oculta.
$\hat{\mathbf{y}}$	:	Resposta da RNA MLP ante o ingresso de um padrão **x**.
E	:	Valor da somatória dos quadrados dos erros de um subconjunto de amostras (no EPNet sofre uma modificação, ver equação 3.2).
W	:	Representação matricial dos pesos sinápticos *w*.
f_o , f_s	:	Funções de ativação (transferência) dos neurônios ocultos e de saída de uma rede MLP, respectivamente.
Φ	:	Representação de um conjunto de funções de ativação (por camadas).
λ	:	Taxa de aprendizado (tamanho do passo) associado ao método da descida do gradiente.
g , **G**	:	Representações vetorial e matricial das derivadas de todos os pesos sinápticos de uma RNA.

μ	:	Termo “momentum” como parte da função de atualização dos pesos sinápticos do método da descida do gradiente.
H , **J**	:	Representação matricial da hessiana e jacobiana associadas aos pesos sinápticos da uma RNA, respectivamente.
θ_i	:	Vetor de parâmetros do especialista *i.*
v	:	Vetor de parâmetros da rede *gating*.
Θ	:	Conjunto de parâmetros totais, especialistas e rede *gating*.
$P(\mathbf{y} \mid \mathbf{x}, \theta_i)$	:	Probabilidade do especialista *i* gerar a saída **y,** dado a entrada **x** e o vetor de seus parâmetros θ_i .
$P(i \mid \mathbf{x}, \mathbf{v}^0)$	:	Probabilidade de se escolher ao especialista *i*, dado a entrada **x** e o vetor dos parâmetros da rede *gating* **v**.
$g_i(\mathbf{x})$	:	Valor da saída *i* da rede *gating* ante um padrão de entrada **x**.
$l(\chi, \Theta)$	:	Função de verossimilhança que recebe ao conjunto de padrões de treinamento χ e ao conjunto de todos os parâmetros a ajustar Θ .
S	:	Representa uma série temporal.
S	:	Representa a função de correlação a ser maximizada no *CasCorr*.
MI	:	Representa o resultado do cálculo da informação mútua.
SEr	:	Taxa de relaxamento do erro (*slack error rate*).
SEr_min	:	*SEr* mínimo.
SEr_max	:	*SEr* máximo.

Capítulo 1

Introdução

As redes neurais artificiais apresentaram grande destaque a partir de meados da década 1981-1990 e continuam a exercer um papel relevante em tarefas de classificação e regressão a partir de treinamento supervisionado (CHERKASSKY & MULIER, 2007), assim como em aplicações envolvendo agrupamento de dados a partir de treinamento não-supervisionado (KOHONEN, 1989) e até na solução de problemas de otimização (JOYA *et al.*, 1997). No contexto de treinamento supervisionado, pode-se atribuir o sucesso das redes neurais à sua flexibilidade de síntese de mapeamentos multidimensionais não-lineares de entrada-saída, refletida em sua capacidade de aproximação universal (HORNIK *et al.*, 1989; HAYKIN, 2008).

No entanto, a propriedade de aproximação universal é apenas de natureza existencial, não fornecendo um mecanismo sistemático de se chegar à rede neural mais indicada em cada aplicação. Além de desafios associados ao processo de otimização de parâmetros e estrutura da rede neural durante a etapa de treinamento supervisionado, um outro desafio está associado à capacidade de generalização da rede neural. Em virtude do processo de treinamento supervisionado se dar com base em um conjunto finito e possivelmente ruidoso de dados observados, é necessário controlar a flexibilidade do mapeamento resultante, de modo que este não seja mais complexo do que o necessário para maximizar o desempenho no treinamento (GEMAN *et al.*, 1992; RAVIV & INTRATOR, 1999).

Esta tese procura abordar simultaneamente esses dois desafios no projeto de redes neurais artificiais a partir de treinamento supervisionado, ao propor mecanismos que buscam definir automaticamente parâmetros e estrutura da rede neural, assim como buscam

maximizar a capacidade de generalização. O estudo se restringe ao projeto de redes neurais que apresentam apenas conexões à frente (*feedforward*).

Inicialmente, é proposta uma abordagem evolutiva para a otimização da estrutura da rede neural, similar a outras abordagens da literatura (YAO & LIU, 1997), visando explorar o espaço de busca contendo todas as arquiteturas possíveis de redes neurais com conexões à frente, havendo apenas um limite máximo de neurônios por camada e de camadas. Embora se caracterize como uma abordagem capaz de produzir bons resultados, o excessivo custo computacional associado à exploração do referido espaço de busca por uma população de soluções candidatas, e também a necessidade de se lidar com o problema da permutação, caracterizado pelo fato de que certos indivíduos, embora possuam representações diferentes dentro da população, possuem graus de aptidão iguais quando avaliados, cria dificuldades para se sustentar a abordagem evolutiva.

Isso levou à proposição de um método construtivo inédito para treinamento supervisionado, voltado para a síntese de redes neurais artificiais com conexões à frente arbitrárias (ACFNNs, do inglês *Arbitrarily Connected Feedforward Neural Networks*), tanto no contexto de classificação como de regressão, em treinamento supervisionado a partir de dados amostrados. O algoritmo foi denominado CoACFNNA (do inglês *Constructive ACFNN Algorithm*).

Comparado à abordagem evolutiva, o custo computacional desta nova proposta construtiva é significativamente mais baixo, além de conduzir a desempenhos melhores das redes neurais obtidas. A explicação para o custo mais baixo está no fato de o CoACFNNA sempre trabalhar com uma única rede neural, e não com uma população de redes neurais candidatas, como é feito na abordagem evolutiva. E comparada a outras abordagens construtivas da literatura, como *Cascade Correlation* (FAHLMAN & LEBIERE, 1990), esta nova proposta não só implementa mecanismos mais elaborados de tomada de decisão (inserção ou remoção de neurônios e/ou conexões), baseada em índices derivados de teoria de informação e em análise de sensibilidade a variações locais, como também permite a obtenção de arquiteturas menos restritas e, portanto, com uma maior capacidade de responder às demandas intrínsecas da aplicação. O CoACFNNA conta com um mecanismo

que permite controlar a convergência propondo uma estratégia de relaxação do erro de treinamento, visando inicialmente explorar o espaço de possibilidades e, posteriormente, permitindo refinar a solução final.

Além disso, as arquiteturas resultantes são bem menos restritas que as tradicionais redes neurais do tipo perceptron de múltiplas camadas (MLP, do inglês *Multi-Layer Perceptron*) (RUMELHART & MCCLELLAND, 1986), e partem sempre de uma configuração inicial sem camadas ocultas, seguida da inserção sequencial de neurônios e conexões, incluindo também etapas de poda. Ainda comparado ao treinamento de uma MLP, será mostrado que esta maior flexibilidade da arquitetura resultante, embora esteja associada a um processo de treinamento mais elaborado, conduz a um melhor desempenho em termos de generalização e a arquiteturas comparativamente mais parcimoniosas.

A conectividade entre os neurônios em arquiteturas ACFNNs, sendo arbitrária e dedicada a cada aplicação, tende a se aproximar da forma pouco padronizada com que os neurônios são conectados nas redes neurais biológicas. Nas arquiteturas ACFNNs, conserva-se a idéia da formação de camadas de neurônios ocultos e não-conectividade entre neurônios da mesma camada, mas libera-se a conectividade à frente entre os neurônios de todas as camadas da rede. Inclui-se também a possibilidade de conexões diretas entre neurônios da camada de entrada e neurônios da camada de saída. Com isso, dependendo das demandas de cada aplicação, o número final de camadas pode variar bastante, assim como o número final de conexões entre neurônios, não havendo necessariamente uma correlação deste com aquele. Além disso, dada a possibilidade de conexão direta dos neurônios da camada de entrada com os neurônios da camada de saída, cria-se a possibilidade de realização conjunta de mapeamentos lineares e não-lineares de entrada-saída, o que não é admitido, por exemplo, em arquiteturas MLP e em algumas propostas construtivas. Mas talvez o aspecto mais relevante, em termos práticos, esteja no fato de que, com o emprego do CoACFNNA, o número de camadas, de neurônios por camada e de conexões entre neurônios é definido automaticamente e de forma construtiva pelo processo de treinamento supervisionado. Logo, o projetista fica liberado desta tarefa, o que não ocorre quando se consideram arquiteturas definidas a priori, como no caso de uma MLP.

Na Figura 1.1, parte A, mostra-se um problema bem conhecido de classificação envolvendo duas classes, denominado de problema das duas espirais. O projetista deveria saber que não conseguiria resolver este problema se usasse menos de 15 neurônios com suas 77 conexões numa rede MLP de uma camada oculta (Figura 1.1, parte B) ou que teria sucesso se usasse 14 neurônios numa rede MLP de duas camadas ocultas, com 7 neurônios em cada camada e totalizando 93 conexões (Figura 1.1, parte C). Por outro lado, a Figura 1.1, parte D, mostra uma instância de arquitetura ACFNN que obtém desempenho equivalente utilizando apenas 7 neurônios em 4 camadas ocultas e 54 conexões ao todo, sendo que estes valores foram determinados automaticamente pelo próprio algoritmo de treinamento e partindo do mesmo conjunto de dados amostrados. Nas três arquiteturas consideradas (Figura 1.1, partes B, C e D), os neurônios ocultos possuem a mesma função de ativação, no caso, a função tangente hiperbólica, e a definição da classe se dá pela saída de maior valor.

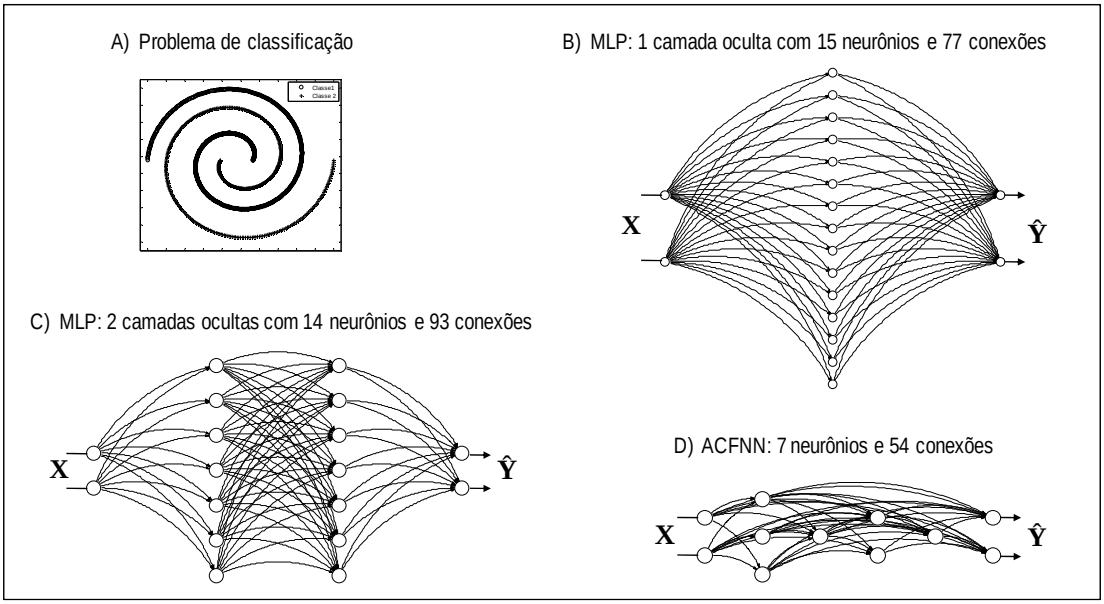


Figura 1.1– Exemplo de arquiteturas distintas de redes neurais que resolvem corretamente um mesmo problema, sendo que a arquitetura da parte D foi determinada automaticamente durante o processo de treinamento.

A determinação automática do número de camadas, de neurônios por camada e de conexões isenta o projetista da tarefa de especificação da arquitetura da rede neural e aumenta a flexibilidade do projeto, permitindo responder apropriadamente a um conjunto

mais amplo de demandas de aplicação. Como a inserção ou poda de neurônios e/ou conexões é feita de forma gradual e sempre visando maximizar o desempenho (a partir de índices derivados de teoria de informação), a arquitetura de rede neural resultante tenderá a maximizar a capacidade de generalização. Aplicações que requeiram mapeamentos de entrada-saída mais complexos tendem, como esperado, a promover a obtenção de arquiteturas com mais neurônios e conexões, e possivelmente com mais camadas ocultas.

Dois aspectos desta nova abordagem construtiva também podem estar presentes em misturas de especialistas (PUMA-VILLANUEVA *et al*., 2005): a existência simultânea de mapeamentos lineares e não-lineares e a divisão de tarefa entre módulos constituintes. É por essa razão que os experimentos a serem considerados também incluem misturas de especialistas heterogêneos (contendo especialistas lineares e não-lineares) como competidores. São também considerados como competidores nas análises experimentais uma abordagem construtiva mais restrita denominada *Cascade Correlation* e uma proposta da literatura que também produz redes do tipo ACFNN, baseada em computação evolutiva e *simulated annealing* e denominada EPNet (YAO & LIU, 1997).

A tese está organizada da seguinte forma: o Capítulo 2 é dedicado a formalizar os conceitos fundamentais envolvidos nesta tese e que servem de apoio para os desenvolvimentos realizados nos capítulos seguintes, incluindo a proposta inédita de método construtivo para ACFNNs. O Capítulo 3 descreve duas formas de síntese de ACFNNs via algoritmos de computação evolutiva, a primeira empregando algoritmos genéticos e concebida durante o desenvolvimento da pesquisa e a segunda baseada em programação evolutiva e já proposta na literatura. O Capítulo 4, por sua vez, descreve em detalhes a proposta desta tese, mais especificamente o CoACFNNA. O Capítulo 5 apresenta a aplicação do CoACFNNA e sua comparação com propostas alternativas da literatura. Para isto, foi escolhida uma série de problemas sintéticos e reais (*benchmarks*) no contexto de classificação de padrões. Finalmente, o Capítulo 6 traz as conclusões e perspectivas de trabalhos futuros associados a esta pesquisa.

Capítulo 2

Fundamentação e conceitos básicos relativos à proposta de tese

Resumo: Este capítulo visa apresentar os principais conceitos que sustentam o desenvolvimento das contribuições originais deste trabalho. O objetivo é transmitir ao leitor as principais definições, incluindo algumas interpretações alternativas e também dicas de boas práticas de uso das metodologias que compõem a pesquisa. Os seguintes tópicos serão apresentados: (*i*) introdução às redes neurais artificiais (RNAs), empregando uma taxonomia; (*ii*) descrição da rede neural perceptron de múltiplas camadas (MLP, do inglês *Multi-Layered Perceptron*), seu processo de treinamento e principais aplicações; (*iii*) formalização da mistura de especialistas heterogêneos; (*iv*) revisão da literatura em algoritmos construtivos para a síntese de RNAs; (*v*) conceitos básicos de algoritmos de computação evolutiva; e (*vi*) apresentação de um dos principais conceitos derivados da teoria de informação: a informação mútua.

2.1 Introdução às redes neurais artificiais

Redes Neurais Artificiais (RNAs) podem ser entendidas como dispositivos de processamento de informação caracterizados pela interconexão de unidades elementares de processamento (neurônios artificiais), simples e similares entre si (HAYKIN, 2008). É uma iniciativa de modelagem matemática de algumas tarefas do sistema cognitivo, sendo atribuído ao *padrão de conexões* e aos *valores dos pesos sinápticos* o papel de moldar o

comportamento de entrada-saída da rede, ou seja, a forma como a rede neural irá responder a certos estímulos de entrada.

As principais motivações para o uso de redes neurais artificiais são as seguintes:

- Habilidade de aprender a partir de dados observados de um processo que se pretende modelar/emular (CHERKASSKY & MULIER, 2007);
- Capacidade de aproximação universal de mapeamentos estáticos não-lineares e multidimensionais (HORNIK *et al.*, 1989);
- Capacidade de aproximação de mapeamentos dinâmicos não-lineares e multidimensionais (NARENDRA & PARTHASARATHY, 1990; NARENDRA & PARTHASARATHY, 1991);
- Habilidade para implementação de memória associativa e para a solução de problemas combinatórios pela busca de pontos de equilíbrio em dinâmicas de relaxação (HOPFIELD, 1982);
- Capacidade de auto-organização e de treinamento não-supervisionado (KOHONEN, 1989).

Dessas motivações, as duas primeiras sustentam o desenvolvimento desta pesquisa.

A seguir, será apresentada uma breve taxonomia das RNAs mais conhecidas na literatura.

2.1.1 Breve taxonomia das RNAs

As redes neurais artificiais podem ser classificadas segundo os critérios explicitados ao longo das próximas sub-seções.

2.1.1.1 Tipo de associação entre as informações de entrada e saída

***Auto-associativas*:**

A rede armazena certos padrões de entrada, recebidos no processo de treinamento, por ajuste de sinapses. Quando se lhe apresenta uma informação incompleta ou com ruído, ela realizará uma associação e responderá com o padrão mais parecido dentre os já armazenados. Exemplos deste tipo de redes: Rede de Hopfield (HOPFIELD, 1982), a família de arquiteturas ART (CARPENTER & GROSSBERG, 1988), Mapas Auto-Organizáveis de Kohonen (KOHONEN, 1989).

***Hetero-associativas*:**

A rede armazena certas associações de entrada-saída recebidas no processo de treinamento, por ajuste de sinapses. Assim, quando se lhe apresenta um certo estímulo de entrada, ela deverá responder gerando a correspondente saída. Exemplos: rede MLP (do inglês *Multi-Layer Perceptron*) (RUMELHART & MCCLELLAND, 1986), rede RBF (do inglês *Radial Basis Function*) (BROOMHEAD & LOWE, 1988).

2.1.1.2 Tipo de arquitetura

***Pelo número de camadas*:**

Redes de uma camada, onde cada um dos neurônios recebe a entrada e produz a saída final. Este tipo de rede geralmente está associado a tarefas auto-associativas, por exemplo, reconstruindo padrões incompletos ou com ruído.

Redes de várias camadas, onde os neurônios estão dispostos em vários níveis ou camadas, que obedecem a certos padrões de conexão. Podem ser destacadas a *camada de entrada*, que recebe os padrões de entrada; uma ou várias camadas ocultas, que geralmente realizam o mapeamento de classes ou regressão de dados, e a *camada de saída*, que nos casos auto-associativos realiza um processamento de associação, mas nos casos hetero-associativos compõe a saída combinando a informação proveniente da última camada oculta.

***Pelo tipo de conexões*:**

Redes *Feedforward*, onde os sentidos das conexões são à frente. Por exemplo: MLP, RBF.

Feedforward/feedback, onde existem conexões de realimentação, além das diretas. Por exemplo: redes recorrentes (CONNOR & MARTIN, 1994). Essas redes podem ser totalmente ou parcialmente recorrentes (DOS SANTOS & VON ZUBEN, 2000).

2.1.1.3 Tipo de mecanismo de aprendizagem

A aprendizagem é um processo no qual a RNA modifica seus pesos (conexões sinápticas) e/ou outros parâmetros estruturais em função de:

- Informação de entrada;
- Informação de entrada associada a uma saída desejada.

Análogo ao caso biológico, onde os estímulos recebidos promovem modificações nas intensidades das sinapses, a maior parte dos mecanismos de aprendizado em redes neurais artificiais buscam ajustar as conexões ou pesos sinápticos em resposta aos estímulos recebidos.

O fundamental no processo de aprendizagem é definir como tais pesos serão alterados quando se requer que a rede aprenda uma nova informação. Tal processo pode ser classificado como *supervisionado* ou *não-supervisionado*, e vai depender do problema a resolver. Existem também processos semi-supervisionados (CHAPELLE *et al*., 2006), mas estes não serão considerados neste trabalho.

Supervisionado:

Nesta forma de aprendizado, existe a ideia de um supervisor que determina a resposta que a rede deverá dar para uma entrada determinada. Este mecanismo está fortemente vinculado a redes do tipo hetero-associativas. O supervisor verifica a saída da rede e, caso ela não coincida com a saída desejada, faz um ajuste nos pesos das conexões visando minimizar esta diferença. Por exemplo:

- *Por correção do erro*: onde o ajuste deve se dar visando minimizar o erro cometido, ou seja, a diferença entre a saída da rede e a saída desejada. Um dos algoritmos mais conhecidos relacionado a este tipo de aprendizado é o *backpropagation* (WERBOS, 1974; RUMELHART & MCCLELLAND, 1986) para redes *feedforward*, e extensões deste algoritmo para redes *feedforward/feedback* ou redes recorrentes (PINEDA, 1987; ALMEIDA, 1987).
- *Por reforço*: variante com um grau menor de supervisão que a anterior, no qual não se dispõe de um exemplo completo do comportamento desejado ou saída desejada. Esta forma de treinamento é análoga ao agir de um crítico, que, em vez de minimizar o erro, fornece apenas um indicativo de nível de sucesso ou fracasso vinculado a uma sequência de ações da rede neural. Alguns exemplos deste tipo de algoritmo: *Linear Reward-Penalty* (NARENDRA & THATHACHER, 1974), *Associative Reward-Penalty* (BARTO & ANANDAN, 1985).

Não-supervisionado:

Para este tipo de aprendizado, não existe supervisor nem crítico. É conhecido também como aprendizado auto-supervisionado, não requerendo indicativos de comportamento desejado para a rede neural. Com isso, interpreta-se o processo de ajuste de conexões como resultado de um processo de auto-organização. Aplica-se em redes auto-associativas e pode ser de dois tipos:

- *Hebbiano* (HEBB, 1949): conexões associadas a neurônios que se encontram ativos simultaneamente tendem a ser fortalecidas, enquanto que conexões associadas a neurônios que sofrem ativações em instantes descorrelacionados no tempo, tendem a ser enfraquecidas.
- *Competitivo e cooperativo*: nesta classe de algoritmos, os neurônios concorrem por representar as amostras de entrada e aquele que vence a competição tem os valores de suas conexões ajustadas, de forma incremental, na direção da amostra de entrada, assim como os neurônios vizinhos ao neurônio vencedor, embora estes últimos sofram ajustes com menor intensidade.

2.1.1.4 Tipo de procedimento de ajuste das conexões sinápticas

Off-line:

O ajuste dos pesos se dá anteriormente à colocação em operação da rede neural. Quando em operação, os pesos sinápticos da rede neural são fixos.

On-line:

Neste modo, não há distinção de fases de treinamento e operação, pois os pesos variam de forma incremental sempre que se apresenta um novo estímulo de entrada e o comportamento a cada instante da rede neural depende dos valores atuais dos pesos sinápticos.

2.1.2 Rede neural do tipo perceptron de múltiplas camadas: MLP

A partir da taxonomia das RNAs apresentada anteriormente, a rede neural MLP (RUMELHART & MCCLELLAND, 1986) é uma rede hetero-associativa, com conexões à frente, que pode apresentar uma ou mais camadas ocultas com neurônios dotados de funções de ativação sigmoidais: logística ou tangente hiperbólica. O aprendizado obedece a um processo supervisionado por correção do erro (*off-line* ou *on-line*). Esta rede apresenta capacidade de aproximação universal (HORNIK *et al.*, 1989).

2.1.2.1 Arquitetura de uma MLP com uma camada oculta

Na Figura 2.1, apresenta-se a arquitetura de uma rede neural MLP com *ne* neurônios na camada de entrada, uma camada oculta com *no* neurônios e *ns* neurônios de saída, sendo que, no caso geral, podem existir múltiplas camadas ocultas com um número arbitrário de neurônios em cada uma delas. Percebe-se que a rede neural MLP é especializada em implementar mapeamentos multidimensionais de entrada-saída.

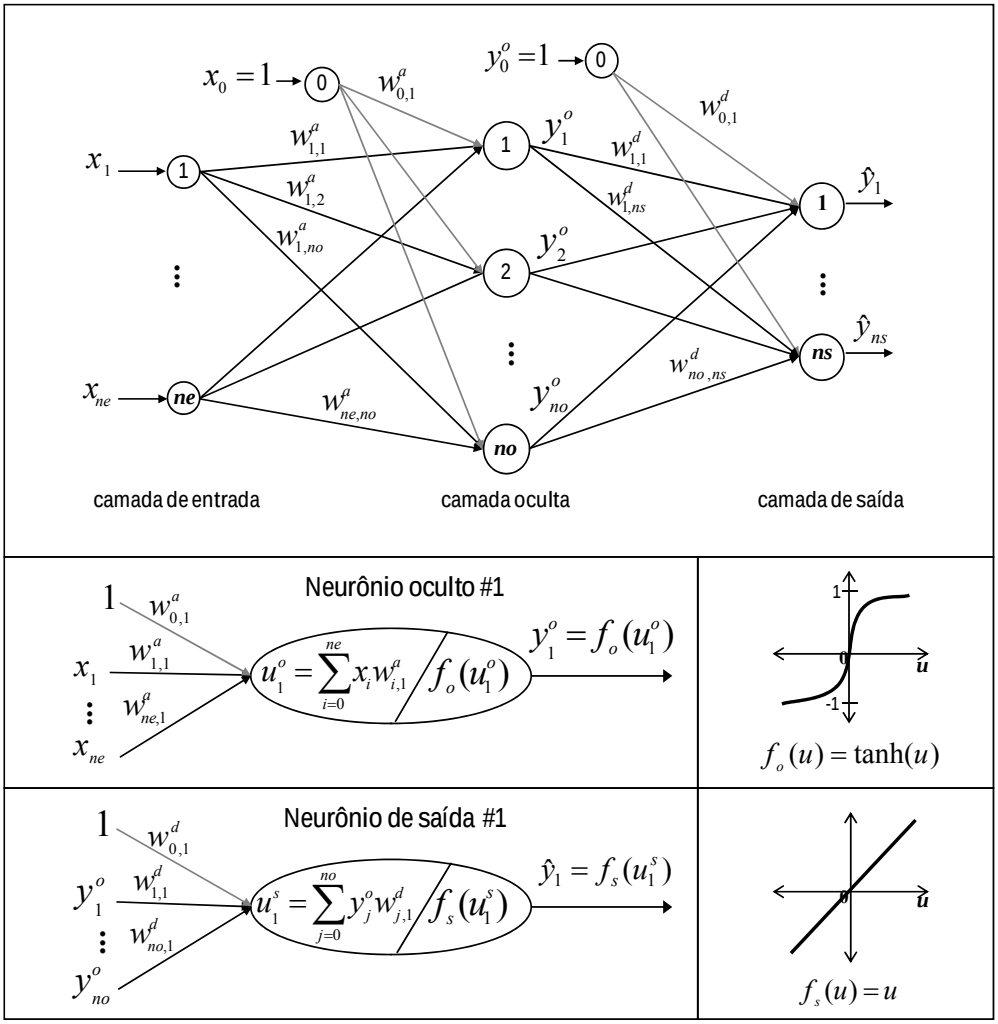


Figura 2.1- Arquitetura da rede MLP: acima, arquitetura completa com uma camada oculta; abaixo, representação do primeiro neurônio oculto e do primeiro neurônio de saída, com suas correspondentes funções de ativação tangente hiperbólica (*tanh*) e *linear*.

É dado destaque também, nesta Figura 2.1, para a representação de dois neurônios: ao neurônio #1 da camada oculta e ao neurônio #1 da camada de saída da rede. Repare que cada neurônio das camadas oculta e de saída realiza uma combinação linear de suas respectivas entradas com os respectivos pesos sinápticos $w_{i,j}^a$ e $w_{j,k}^d$ ($i = 0,1,2,...,ne$; $j = 0,1,2,...,no$ e $k = 1,2,...,ns$), produzindo as ativações internas u_i^o (camada oculta) e u_k^s (camada de saída). As entradas x_0 e y_0^o são constantes e iguais a 1, e foram consideradas para representar a polarização dos neurônios, respectivamente, das camadas oculta e de

saída. Essas entradas constantes atribuem um grau de flexibilidade adicional ao mapeamento não-linear multidimensional que a rede neural pode realizar.

Após a combinação linear, as ativações internas são aplicadas nas funções de ativação f_o e f_s dos neurônios das camadas oculta e de saída, respectivamente, para formar a saída final de cada neurônio. As funções f_o são do tipo sigmoidal e f_s do tipo linear. Esta configuração é comumente empregada em problemas de regressão de dados, incluindo o problema de predição de séries temporais. Vale indicar que não existem ativações internas nos neurônios da camada de entrada (aqueles numerados de 0 a *ne*) dado que a sua função é apenas distribuir todas as *ne* entradas à camada seguinte da rede neural.

As saídas dos neurônios da camada oculta, y_j^o ($j = 1,2,...,no$), podem ser representadas pela equação (2.1), onde *ne* é o número de entradas, e $w_{i,j}^a$ ($i = 0,1,2,...,ne$ e $j = 1,2,...,no$) são os pesos das conexões sinápticas antes da camada oculta:

$$y_j^o = f_o\left(\sum_{i=0}^{ne} x_i w_{i,j}^a\right). \tag{2.1}$$

Algo que será útil posteriormente é levar em conta que o valor das funções de ativação f_o e f_s, de cada neurônio (camada oculta e de saída), depende dos valores dos pesos das conexões que conduzem sinais até aquele neurônio, como indicado nas equações (2.1) e (2.2).

Na equação (2.2), $w_{j,k}^d$ ($j = 0,1,2,...,no$ e $k = 1,2,...,ns$) são os pesos das conexões sinápticas depois da camada oculta e $\hat{y}_k$ corresponde à *k*-ésima saída da rede neural:

$$\hat{y}_k = f_s\left(\sum_{j=0}^{no} y_j^o w_{j,k}^d\right). \tag{2.2}$$

2.1.2.2 Arquitetura de uma MLP com duas camadas ocultas

A Figura 2.2 apresenta uma proposta de arquitetura de uma rede neural MLP com duas camadas ocultas, *ne* entradas e *ns* saídas. Os pesos das conexões intercamadas são representados pelas matrizes $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$ e $\mathbf{W}^{(3)}$, sendo que o número de neurônios em cada camada oculta pode ser arbitrário e igual ou distinto daquele definido para a outra camada oculta.

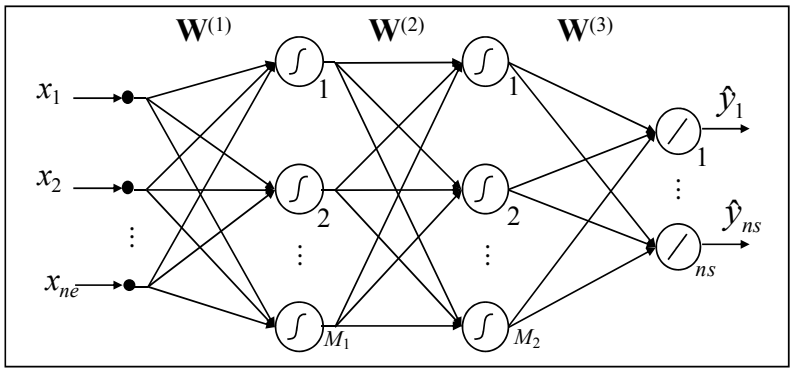


Figura 2.2 – Arquitetura da rede MLP com duas camadas ocultas.

Cada saída da rede é calculada seguindo a equação (2.3), na qual $\Phi_j^1(\cdot)$, $\Phi_k^2(\cdot)$ e $\Phi_l^3(\cdot)$ representam as funções de ativação da primeira e segunda camadas ocultas e da camada de saída, respectivamente, sendo do tipo tangente hiperbólica para as duas primeiras e identidade para os neurônios da camada de saída.

$$\hat{y}_l = \Phi_l^3\left\{\sum_{k=1}^{M_2} w_{kl}^{(3)}\Phi_k^2\left[\sum_{j=1}^{M_1} w_{jk}^{(2)}\Phi_j^1\left(\sum_{i=1}^{ne} w_{ij}^{(1)}x_i\right)\right]\right\}. \tag{2.3}$$

Nesta formulação, não foram contempladas as entradas de polarização, as quais podem ser prontamente adicionadas. Quando comparada à Figura 2.1, os pesos $\mathbf{W}^{(1)}$ correspondem aos pesos $w_{i,j}^a$ e os pesos $\mathbf{W}^{(3)}$ correspondem aos pesos $w_{j,k}^d$.

2.1.2.3 Algoritmo de treinamento supervisionado

O treinamento de uma MLP consiste em encontrar valores adequados para todos os pesos das conexões (como, por exemplo, $w_{i,j}^{a}$ e $w_{j,k}^{d}$ da rede neural da Figura 2.1, para todos os valores admissíveis dos índices i, j e k), de forma que, ao ingressar com todos os padrões de entrada disponíveis durante a fase de treinamento, a saída da rede seja o valor mais próximo ao valor desejado para cada entrada. Busca-se, portanto, minimizar o erro acumulado associado a todas as saídas produzidas pela rede. Para isto, precisa-se de:

Dados de treinamento: Chamados de amostras de treinamento, onde cada amostra consta de valores de entrada (**x**) associados com valores de saída desejada (**y**).

Uma função-objetivo: Por exemplo, a *função somatória dos quadrados dos erros* (*E*), apresentada na equação (2.4), onde o erro é entendido ser a diferença entre o valor de saída obtido pela rede $\hat{y}_k^p$ e o valor de saída desejado y_k^p, na forma:

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{ns}\left(\hat{y}_k^p - y_k^p\right)^2 , \tag{2.4}$$

com P sendo o número total de padrões de treinamento, $p = 1,2,...,P$ e $k = 1,2,...,ns$.

Vetor gradiente dos pesos sinápticos (**g**): Considere a rede MLP de uma camada oculta mostrada na Figura 2.1, e as equações (2.1) e (2.2). O vetor gradiente é composto pela derivada da função-objetivo *E* em relação a cada um dos pesos sinápticos $w_{i,j}^{a}$ e $w_{j,k}^{d}$, para todos os valores possíveis dos índices i, j e k.

As derivadas dos pesos depois da camada oculta $w_{j,k}^{d}$ ($j = 0,1,2,...,no$ e $k = 1,2,...,ns$):

$$\frac{\partial E}{w_{j,k}^{d}} = \sum_{p=1}^{P}\frac{\partial E^p}{\partial w_{j,k}^{d}} = \sum_{p=1}^{P}\frac{\partial E^p}{\partial \hat{y}_k^p}\frac{\partial \hat{y}_k^p}{\partial w_{j,k}^{d}} = \sum_{p=1}^{P}\frac{\partial E^p}{\partial \hat{y}_k^p}\frac{\partial \hat{y}_k^p}{\partial u_k^{s,p}}\frac{\partial u_k^{s,p}}{\partial w_{j,k}^{d}} . \tag{2.5}$$

Para as derivadas dos pesos antes da camada oculta $w_{i,j}^{a}$ ($i = 0,1,2,...,ne$ e $j = 0,1,2,...,no$), previamente define-se:

$$\frac{\partial E^p}{\partial y_j^{o,p}} = \sum_{k=1}^{ns} \frac{\partial E^p}{\partial u_k^{s,p}} \frac{\partial u_k^{s,p}}{\partial y_j^{o,p}} = \sum_{k=1}^{ns} \frac{\partial E^p}{\partial \hat{y}_k^p} \frac{\partial \hat{y}_k^p}{\partial u_k^{s,p}} \frac{\partial u_k^{s,p}}{\partial y_j^{o,p}}, \tag{2.6}$$

obtendo-se:

$$\frac{\partial E}{w_{i,j}^a} = \sum_{p=1}^{P} \frac{\partial E^p}{\partial w_{i,j}^a} = \sum_{p=1}^{P} \frac{\partial E^p}{\partial u_j^{o,p}} \frac{\partial u_j^{o,p}}{\partial w_{i,j}^a} = \sum_{p=1}^{P} \frac{\partial E^p}{\partial y_j^{o,p}} \frac{\partial y_j^{o,p}}{\partial u_j^{o,p}} \frac{\partial u_j^{o,p}}{\partial w_{i,j}^a} \tag{2.7}$$

Finalmente, o vetor gradiente **g** dos pesos sinápticos estaria composto por todas as derivadas parciais $\frac{\partial E}{w_{i,j}^a}$ e $\frac{\partial E}{w_{j,k}^d}$ ($i = 0,1,2,...,ne$; $j = 0,1,2,...,no$ e $k = 1,2,...,ns$) dispostas em ordem arbitrária e em forma de um vetor.

Ajuste dos pesos: Requer um algoritmo para otimização não-linear irrestrita. São várias as possibilidades para a MLP. O mais básico dos algoritmos de otimização emprega informação de primeira ordem (vetor gradiente da função-objetivo) e corresponde ao método do gradiente. Métodos que empregam informação de 2a. ordem (matriz hessiana ou aproximações da sua inversa) também são utilizados (BATTITI, 1992). Qualquer que seja o método de otimização, será necessário empregar o algoritmo de retro-propagação, também denominado de *backpropagation* (WERBOS, 1974) para obter o vetor gradiente.

Procedimento para evitar sobre-ajuste (overfitting): Também conhecido como sobre-treinamento, acontece quando a rede parece estar representando o problema cada vez melhor, ou seja, o erro junto ao conjunto de treinamento vai diminuindo. A questão é que, em algum ponto deste processo, a capacidade de responder adequadamente a um novo conjunto de dados, também denominada de capacidade de generalização, pode começar a piorar (PRECHELT, 1997). Dentre as formas empíricas de se evitar o sobre-ajuste (BISHOP, 1995), destacam-se: (*i*) validação cruzada; (*ii*) incorporação de um termo de penalidade na função-objetivo buscando minimizar a norma do vetor formado pelos pesos das conexões sinápticas; e (*iii*) inserção de ruído nos dados de treinamento.

Em relação aos métodos de ajuste dos pesos sinápticos mais conhecidos na literatura, pode-se mencionar os seguintes:

Gradiente simples: O uso do negativo do vetor gradiente como uma direção para a minimização foi utilizada pela primeira vez por Cauchy (1847). Neste método, parte-se de uma posição inicial e iterativamente move-se ao longo das direções de descida até que o ponto ótimo seja encontrado. O método do gradiente simples pode parecer ser uma boa técnica de minimização irrestrita. No entanto, devido ao fato de que a direção do gradiente provê informação local, o método pode não ser eficaz na maioria dos problemas.

Gradiente Conjugado: Este método melhora de forma significativa a característica de convergência do método do gradiente simples, ao incorporar a ideia das direções conjugadas (SHEWCHUK, 1994; WADE, 2006) envolvendo o uso do gradiente da função-objetivo. Qualquer método de minimização que faz uso das direções conjugadas é quadraticamente convergente. Esta propriedade de convergência quadrática é muito útil porque garante que o método irá minimizar uma função quadrática (convexa) em n passos (com n sendo o número de variáveis a ajustar, ou seja, a dimensão do espaço de otimização) ou menos.

Métodos de Newton e quasi-Newton: Os algoritmos baseados no método de Newton utilizam de forma direta a informação de segunda ordem ou, em outras palavras, calculam a matriz hessiana e a utilizam na definição da direção da minimização, a qual envolve o cálculo da inversa da matriz hessiana. Isso permite, por exemplo, encontrar o ponto mínimo de uma função quadrática em apenas uma iteração, mas requer a inversão da matriz hessiana, a qual pode ser de elevada dimensão. Por outra parte, os métodos quasi-Newton (DENNIS & SCHNABEL, 1996) utilizam a mesma abordagem dos baseados em Newton, com a diferença de que não trabalham diretamente com a matriz hessiana (não precisam calcular as segundas derivadas da função-objetivo) e sim com uma aproximação da inversa de dita matriz a partir do vetor gradiente. Com isso, os métodos quasi-Newton tendem a promover ganhos em termos de tempo de processamento e uso de memória.

Visando exemplificar de forma gráfica o comportamento em operação de alguns dos métodos de ajuste dos pesos sinápticos, as Figuras 2.3 e 2.4 contêm um exemplo didático[1] de ajuste dos pesos de uma rede MLP com uma entrada e uma saída e com 7 pesos sinápticos a ajustar (como ilustrado na parte superior da Figura 2.3).

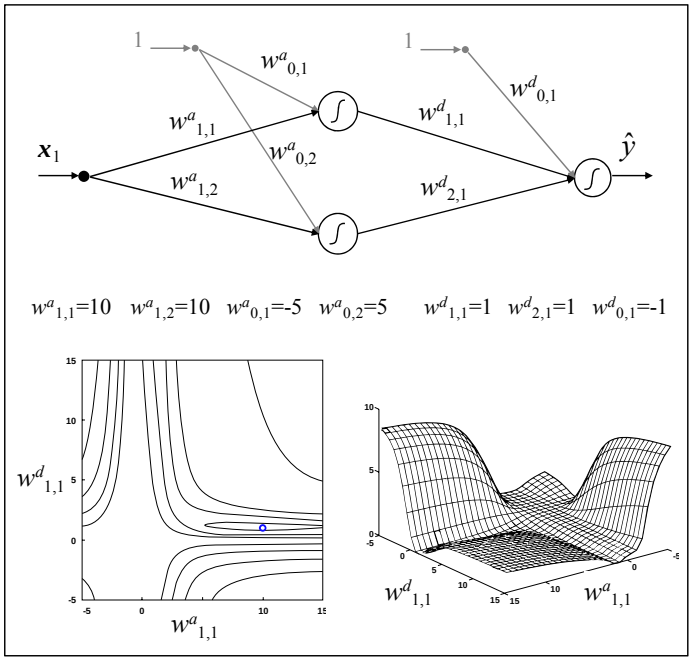


Figura 2.3 – Arquitetura e mapeamento da função-objetivo *E* para 2 dos 7 pesos sinápticos do exemplo.

Na parte inferior desta Figura 2.3, mostra-se à esquerda as curvas de nível e à direita o valor da própria função-objetivo *E* em relação a apenas 2 dos 7 pesos sinápticos ($w^a_{1,1}$, um dos pesos antes da camada oculta, e $w^d_{1,1}$, um dos pesos depois da camada oculta). O objetivo é minimizar a função-objetivo (erro de treinamento) e, por conseguinte, encontrar o menor valor deste mapeamento. O ponto de partida (valores iniciais dos pesos sinápticos) foi fixado e se encontra indicado na região intermediária da Figura 2.3.

[1] Este exemplo foi baseado nas notas de aula do livro "*Neural Network Design*" (HAGAN *et al.*, 1996), associados ao capítulo 12. Disponível em: http://hagan.okstate.edu/nnd.html.

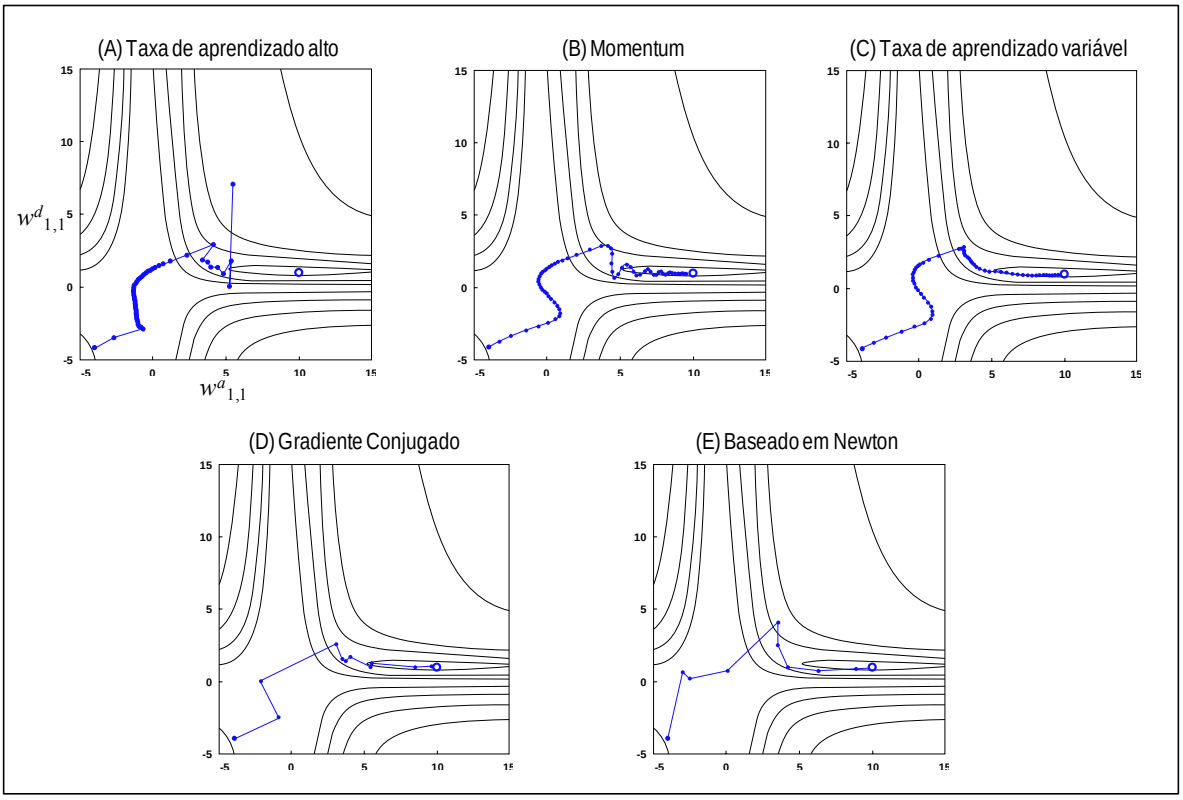


Figura 2.4 – Exemplos de trajetórias associadas ao ajuste de dois pesos sinápticos para diferentes algoritmos de otimização.

Seguindo a sequência do exemplo didático, a Figura 2.4 mostra as trajetórias descritas pelos métodos de otimização considerados:

(**A**) Método do gradiente simples com taxa de aprendizado fixado num valor alto, o que provoca oscilações e impede a convergência do processo de aprendizado. Entende-se por taxa de aprendizado o tamanho do passo dado na direção oposta daquela apontada pelo vetor gradiente (λ, na equação (2.8)). Taxas de aprendizado muito baixas também causam dificuldades, pois tornam o treinamento demorado e requerem maior quantidade de iterações. Logo, deve existir um intervalo de valores de passo mais adequados para cada problema de otimização.

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda \mathbf{g}_i \,, \tag{2.8}$$

onde $\mathbf{g}_i$ representa o vetor gradiente na iteração i.

Na Figura 2.4(a), se observa a dificuldade para se atingir o ponto de mínimo e a grande quantidade de iterações utilizadas.

(**B**) A inclusão do termo de momentum (μ) na equação de atualização dos pesos sinápticos tem por objetivo reduzir a tendência de oscilação e, assim, agir no sentido de aumentar a velocidade de convergência do treinamento. Seu valor pode variar de 0,0 (anula seu efeito) a 1,0. A equação (2.9) apresenta a inclusão do termo de momentum:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda \mathbf{g}_i + \mu(\mathbf{w}_i - \mathbf{w}_{i-1}). \tag{2.9}$$

Na Figura 2.4(b), mostra-se a trajetória com o uso do termo de momentum e o mínimo é atingido. No entanto, o número de iterações empregado pelo processo iterativo de ajuste de pesos segue sendo alto.

(**C**) Outra variante do método do gradiente simples é a utilização de taxa de aprendizado variável. Uma maneira de implementar esse auto-ajuste é a seguinte: Se a função-objetivo *E* aumenta acima de certa porcentagem ζ após a atualização dos pesos, então a atualização dos pesos é descartada e a taxa de aprendizado (λ) é diminuída por certo fator, por exemplo, $\rho\lambda$ (sendo $0 < \rho < 1$). Zerar o termo de momentum ($\mu = 0$) também pode ser adotado aqui. Se a função-objetivo *E* diminui após a atualização dos pesos, então esta atualização é aceita e a taxa de aprendizado é incrementada por um fator $\eta > 1$. O momentum μ recupera seu valor original (caso este tenha sido previamente zerado). Se a função-objetivo *E* aumenta abaixo de certa porcentagem ζ , então a atualização dos pesos é aceita e a taxa de aprendizado e o termo de momentum são mantidos sem alteração até a próxima iteração.

Percebe-se, na Figura 2.4(c), que o algoritmo produz uma trajetória suave até o ótimo, mas ainda contando com um elevado número de iterações.

(**D**) Este quadro da Figura 2.4 mostra a trajetória do método do Gradiente Conjugado, cuja equação padrão de atualização dos pesos sinápticos é dada pela seguinte equação:

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \lambda_i \mathbf{d}_i , \tag{2.10}$$

onde o vetor $\mathbf{d}_i$ indica uma nova forma de direção de busca que, na primeira iteração do algoritmo, usa a informação do vetor gradiente da seguinte forma: ($\mathbf{d}_0 = -\mathbf{g}_0$). Para iterações subsequentes, $\mathbf{d}_i$ é atualizada como segue:

$$\mathbf{d}_i = -\mathbf{g}_i + \beta_i \mathbf{d}_{i-1}. \tag{2.11}$$

O cálculo de β_i pode ser efetuado de mais de uma forma, como, por exemplo:

$$\beta_i = \frac{\Delta \mathbf{g}_{i-1}^T \mathbf{g}_i}{\Delta \mathbf{g}_{i-1}^T \mathbf{d}_{i-1}}, \text{ ou } \beta_i = \frac{\mathbf{g}_i^T \mathbf{g}_i}{\mathbf{g}_{i-1}^T \mathbf{g}_{i-1}}, \text{ ou } \beta_i = \frac{\Delta \mathbf{g}_{i-1}^T \mathbf{g}_i}{\mathbf{g}_{i-1}^T \mathbf{g}_{i-1}}. \tag{2.12}$$

O uso das direções conjugadas melhorou consideravelmente o processo de aprendizado da rede, como ilustrado na Figura 2.4(d). O número de iterações (apenas 10) foi drasticamente inferior quando comparado aos outros casos (gradiente simples). Cabe destacar que existem versões de algoritmos de gradiente conjugado cuja complexidade cresce linearmente com o número de pesos sinápticos passíveis de ajuste, e não quadraticamente (PEARLMUTTER, 1994). Essas implementações motivam o seu emprego em casos em que o número de pesos sinápticos é bastante elevado.

(**E**) Este último quadro da Figura 2.4 ilustra a trajetória do ajuste promovido pelo algoritmo de Levenberg-Marquardt (HAGAN, 1994), o qual é baseado no método de Newton. A equação padrão para a atualização dos pesos sinápticos é deduzida a partir da expansão em série de Taylor até segunda ordem, resultando:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \left[\mathbf{H}_i\right]^{-1} \mathbf{g}_i, \tag{2.13}$$

onde **H** representa a matriz hessiana. Este método encontra o ponto mínimo de uma função quadrática em apenas um passo. Caso a função seja não-quadrática, a convergência não é mais garantida, requerendo a adoção de um fator de redução do passo no intervalo (0−1), produzindo:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda_i \left[\mathbf{H}_i\right]^{-1} \mathbf{g}_i. \tag{2.14}$$

O algoritmo de Levenberg-Marquardt aproxima a matriz hessiana a partir da matriz jacobiana **J**, da seguinte forma:

$$\mathbf{H} = \mathbf{J}^T \mathbf{J} \,. \tag{2.15}$$

A matriz **J** é calculada a partir das derivadas de primeira ordem da função objetivo (E) em relação a cada peso sináptico da rede e para cada amostra de treinamento p, como mostrado a seguir:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial E^1}{\partial w_1} & \frac{\partial E^1}{\partial w_2} & \cdots & \frac{\partial E^1}{\partial w_n} \\ \frac{\partial E^2}{\partial w_1} & \frac{\partial E^2}{\partial w_2} & \cdots & \frac{\partial E^2}{\partial w_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial E^P}{\partial w_1} & \frac{\partial E^P}{\partial w_2} & \cdots & \frac{\partial E^P}{\partial w_n} \end{bmatrix}, \tag{2.16}$$

onde n representa o número total de pesos sinápticos em toda a rede e lembrando que P representa o número de padrões de treinamento.

Este método pode também modificar a diagonal da matriz $\mathbf{H}_i$:

$$\tilde{\mathbf{H}}_i = [\mathbf{H}_i] + \alpha_i [\mathbf{I}], \tag{2.17}$$

sendo **I** a matriz identidade e α_i uma constante positiva empregada visando tornar $\mathbf{H}_i$ definida positiva, toda vez que esta condição não é atendida por $\mathbf{H}_i$. O desempenho deste algoritmo é notavelmente superior àquele obtido via técnicas baseadas em gradiente simples. Foi preciso um menor número de iterações (apenas 8) e com um desempenho equivalente àquele produzido pelo gradiente conjugado. Neste exemplo, verifica-se a vantagem em usar informação de segunda ordem. No entanto, o custo computacional para o cálculo da matriz jacobiana **J** restringe a utilização deste algoritmo para problemas de pequeno e médio porte, medido em termos do número de pesos sinápticos a ajustar. Os métodos quasi-Newton surgiram visando aliviar esta desvantagem.

2.1.2.4 Uma forma de evitar o sobre-ajuste no treinamento

Uma técnica bem conhecida para se evitar o sobre-ajuste consiste em separar os padrões em três conjuntos: treinamento, validação e teste (PRECHELT, 1997; PRECHELT, 1998). O conjunto de treinamento é usado para ajustar os pesos sinápticos. Após cada época de ajuste de pesos, calcula-se o erro junto ao conjunto de validação. Assim, quando este valor apresentar uma tendência definida de aumento, será um indicativo de sobre-ajuste e o treinamento deverá ser interrompido. O conjunto de teste, por sua vez, é utilizado para indicar como ficaria o desempenho da rede neural em operação. É suposto que os três conjuntos contêm amostras independentes e são todos capazes de representar bem o problema que está sendo abordado. Por exemplo, espera-se que um bom desempenho junto ao conjunto de validação implique em um bom desempenho junto ao conjunto de teste. Na Figura 2.5, ilustra-se este método. Na prática, particularmente quando há recursos computacionais disponíveis, o mais indicado é sobre-treinar a rede neural e ir armazenando o conjunto de pesos associado ao valor mínimo do erro junto ao conjunto de validação. Como o comportamento do erro de validação pode ser errático, detectar quando se atingiu o mínimo pode ser pouco confiável, sendo mais indicado forçar o sobre-treinamento e tomar o conjunto de pesos associado ao instante em que o treinamento deveria ter parado, embora tenha seguido adiante.

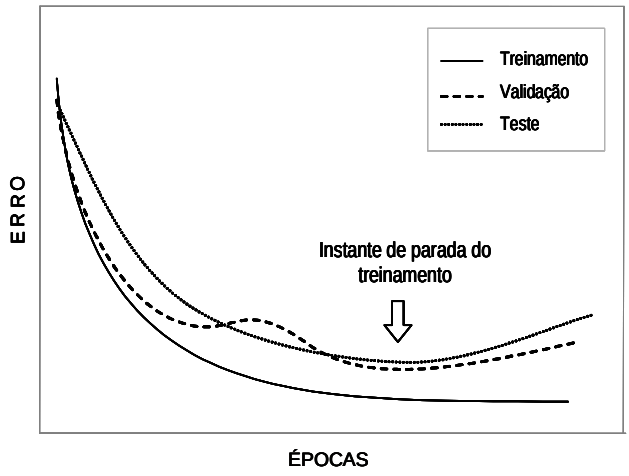


Figura 2.5 – Um exemplo de aplicação do critério de parada visando evitar sobreajuste.

2.1.3 Principais aplicações das RNAs

2.1.3.1 Problemas de reconhecimento de padrões

Agrupamento ou clusterização de dados:

- As amostras não estão associadas a classes, ou seja, não se encontram previamente rotuladas.
- O objetivo é descobrir agrupamentos em que os elementos pertencentes a cada grupo apresentem características em comum.
- Aplica-se aprendizado não-supervisionado.

  Exemplos:

  - ✓ Mineração de dados e de textos;
  - ✓ Síntese e quantização de informação.

Classificação de padrões:

- Cada amostra está associada a uma classe, ou seja, as amostras estão previamente rotuladas.
- O objetivo é sintetizar um classificador capaz de generalizar corretamente, ou seja, classificar adequadamente novas amostras ainda não-rotuladas.
- Aplica-se aprendizado supervisionado.

  Exemplos:

  - ✓ Reconhecimento de pessoas pelo rosto, voz ou digitais.
  - ✓ Reconhecimento e classificação de textos, reconhecimento óptico de caracteres (OCR, do inglês *Optical Character Recognition*).

2.1.3.2 Problemas de regressão de dados

Dentre os problemas que envolvem regressão de dados, tem-se:

- Aproximação de funções (VON ZUBEN, 1996);
- Identificação de sistemas (JAIN & MAO, 1996);
- Controle de processos (NØRGAARD *et al*., 2000);
- Filtragem adaptativa (WIDROW & STEARNS, 1985);
- Predição de séries temporais (ZHANG *et al*., 1998).

2.1.3.3 Problemas de otimização combinatória

Problemas de otimização combinatória podem ser resolvidos via RNAs, por exemplo, problemas de caixeiro viajante (TSP, do inglês *Travelling Salesperson Problem*) e problemas de roteamento de veículos. Para tanto, podem ser empregadas redes neurais de Hopfield (HOPFIELD, 1982) e redes neurais auto-organizáveis (GOMES & VON ZUBEN, 2002).

2.1.3.4 Outras aplicações

- Encriptação de dados (LI *et al*., 2004);
- Redução da dimensão de dados (HINTON & SALAKHUTDINOV, 2006).

Evidentemente, a lista de possíveis aplicações não é exaustiva, apenas ilustrativa.

2.2 Mistura de especialistas heterogêneos

Mistura de especialistas heterogêneos – MHE (PUMA-VILLANUEVA *et al*., 2005) é uma proposta de comitê de máquinas na qual o espaço de entrada é automaticamente dividido em regiões durante o treinamento e, para cada região, existe um único ou um subconjunto de especialistas heterogêneos mais indicados para atuar, os quais são também concebidos durante o processo de treinamento. A denominação de especialistas heterogêneos se deve ao fato de que os especialistas podem ser de diferentes naturezas, lineares ou não-lineares,

por exemplo. Também a forma de atribuição de papeis aos especialistas pode ser linear ou não-linear, além de poder ser gradual e contemplar sobreposições de regiões. Isto se consegue a partir da implementação de uma "rede *gating*" (ver Figura 2.6) que define os coeficientes ($\mathbf{g}_1$, ..., $\mathbf{g}_m$) da combinação convexa envolvendo a saída de cada especialista ($\mathbf{y}_1$, ..., $\mathbf{y}_m$). Esses coeficientes devem ser sempre não-negativos e somar na unidade.

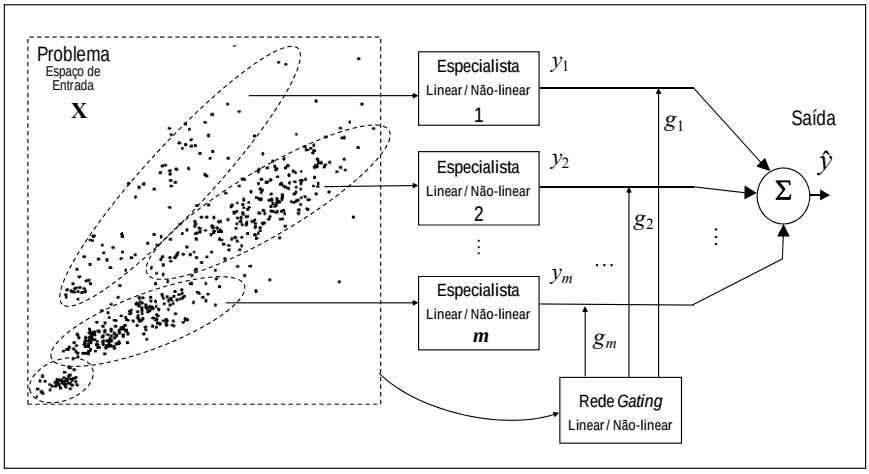


Figura 2.6 – Estrutura típica de uma arquitetura de mistura de especialistas heterogêneos.

Na Seção 2.1.2.3, foi visto que a função-objetivo que guia o processo de treinamento de uma MLP comumente é a função somatória dos erros quadráticos, equação (2.4), tendo como objetivo a minimização desta função. No caso de MEs, a função-objetivo é baseada na interpretação de MEs como modelos de mistura (MCLACHLAN & BASFORD, 1988; LIMA, 2004), o que implica numa função de verossimilhança, a qual deverá ser maximizada. Cada especialista terá uma função densidade de probabilidade condicional associada, com as saídas da rede *gating* desempenhando o papel de coeficientes da mistura. A Figura 2.6 ilustra o papel da rede *gating* na definição de um modelo de mistura empregado na interpretação de MEs. Observe que o espaço de entrada foi decomposto em regiões, as quais foram alocadas a cada especialista. Pode haver sobreposição entre as regiões de atuação dos especialistas.

2.2.1 Arquitetura de Mistura de Especialistas

A arquitetura a ser considerada foi apresentada na Figura 2.6, sendo composta por *m* módulos referidos como redes especialistas, cada um implementando uma função parametrizada $\mathbf{y}_i = f_i(\boldsymbol{\theta}_i, \mathbf{x})$ da entrada $\mathbf{x}$ para a saída $\mathbf{y}_i$, onde $\boldsymbol{\theta}_i$ é o vetor de parâmetros do especialista *i*. A saída de cada uma das redes especialistas recebe uma interpretação probabilística, considerando que o especialista *i* gera a saída $\mathbf{y}_i$ com probabilidade $P(\mathbf{y}_i \mid \mathbf{x}, \boldsymbol{\theta}_i)$, onde $\mathbf{y}_i$ pertence ao espaço amostral da variável aleatória $\mathbf{y}$.

Considerando que diferentes redes especialistas são apropriadas para diferentes regiões do espaço de entrada, a arquitetura requer um mecanismo capaz de identificar, para cada entrada $\mathbf{x}$, que especialista (ou combinação deles) é mais capaz de produzir a saída correta, em termos probabilísticos. Isto é realizado pela rede *gating*.

A interpretação probabilística da rede *gating* é de um sistema que calcula, para cada especialista, a probabilidade dele gerar a saída desejada, com base apenas no conhecimento da entrada $\mathbf{x}$. Essas probabilidades são expressas pelos coeficientes $\mathbf{g}_i$ (*i*=1, ..., *m*), de modo que estes devem ser não-negativos e devem produzir sempre o valor unitário quando somados, para cada $\mathbf{x}$. Estes coeficientes não são constantes fixas, mas variam em função da entrada $\mathbf{x}$. Caso os coeficientes $\mathbf{g}_i$ (*i*=1, ..., *m*) fossem constantes e as redes especialistas atuassem junto a todos os aspectos do problema, resultaria uma abordagem do tipo *ensemble*, que corresponde a um comitê de máquinas estático (PERRONE & COOPER, 1993; DIETTERICH, 2000; KUNCHEVA, 2004).

Há muitas formas de garantir que os coeficientes $\mathbf{g}_i$ (*i*=1, ..., *m*) atendam as restrições acima. Uma abordagem é utilizar a função *softmax* (JACOBS *et al.*, 1991). A função *softmax* define um conjunto de variáveis intermediárias ξ_i (*i*=1, ..., *m*) como funções da entrada $\mathbf{x}$ e de um vetor de parâmetros $\mathbf{v}_i$ (*i*=1, ..., *m*) na forma:

$$\xi_i = \xi_i(\mathbf{x}, \mathbf{v}_i)\,. \tag{2.18}$$

Com isso, os coeficientes $\mathbf{g}_i$ (*i*=1, ..., *m*) podem ser definidos em termos de ξ_i (*i*=1, ..., *m*) como segue:

$$g_i = \frac{\exp(\xi_i)}{\sum_{j=1}^{m} \exp(\xi_j)}. \tag{2.19}$$

A partir desta definição, os coeficientes g_i (i=1, ..., m) passam a respeitar as restrições impostas, isto é, são não-negativos e, somados, produzem sempre o valor unitário, para cada **x**. Uma interpretação probabilística para as variáveis intermediárias ξ_i (i=1, ..., m) é que elas pertencem a uma família de distribuições exponenciais de probabilidade (JORDAN & JACOBS, 1994).

A seguir, será especificado o modelo de probabilidade adotado para a arquitetura de mistura de especialistas. Considere que o conjunto de treinamento $\chi = \{(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})\}_{t=1}^{N}$ é gerado da seguinte forma: dada uma entrada **x**, um especialista i é escolhido com probabilidade $P(i \mid \mathbf{x}, \mathbf{v}^0)$ (onde o sobrescrito "0" será usado para distinguir os valores reais dos parâmetros do modelo de probabilidade adotado daqueles estimados pela rede *gating* ou pela rede especialista). Dada a escolha do especialista e dada a entrada, a saída desejada y é suposta ser gerada de acordo com a probabilidade $P(\mathbf{y} \mid \mathbf{x}, \theta_i^0)$. Supõe-se que cada um dos pares de entrada-saída é gerado independentemente.

Observe que uma dada saída pode ser gerada de m formas diferentes, correspondendo aos m especialistas. Assim, a probabilidade total de geração de **y** a partir de **x** é dada pela soma sobre i, na forma:

$$P(\mathbf{y} \mid \mathbf{x}, \Theta^0) = \sum_{i=1}^{m} P(i \mid \mathbf{x}, \mathbf{v}^0) P(\mathbf{y} \mid \mathbf{x}, \theta_i^0), \tag{2.20}$$

onde Θ^0 denota o vetor contendo todos os parâmetros, na forma $\Theta = [\theta_1^0, \theta_2^0, \ldots, \theta_m^0, \mathbf{v}^0]^T$. A função densidade de probabilidade na equação (2.20) é conhecida como *mistura de densidade* ou *função de verossimilhança* da mistura. É uma mistura de densidade no espaço de saída, condicionada à escolha da entrada, onde $P(i \mid \mathbf{x}, \mathbf{v}^0)$ é a probabilidade de se escolher o especialista i, dada a entrada **x** e o parâmetro $\mathbf{v}^0$ da rede *gating*, e $P(\mathbf{y} \mid \mathbf{x}, \theta_i^0)$ é a probabilidade do especialista i gerar a saída **y**, dada a entrada **x** e seu vetor de parâmetros θ_i^0.

É tarefa da rede *gating* modelar as probabilidades $P(i \mid \mathbf{x}, \mathbf{v}^0)$, i=1, ..., m. É possível parametrizar estas probabilidades via equações (2.18) e (2.19).

A saída da mistura de densidade pode ser calculada através da média condicional. A média condicional $\mathbf{y} = \mathrm{E}\left[P\left(\mathbf{y} \mid \mathbf{x}, \Theta^0\right)\right]$ é obtida tomando o valor esperado da equação (2.20):

$$\mathbf{y} = \sum_{i=1}^{m} \mathbf{g}_i \mathbf{y}_i \,, \tag{2.21}$$

onde $\mathbf{y}_i$ é a média condicional associada à distribuição de probabilidade $P(\mathbf{y} \mid \mathbf{x}, \theta_i^0)$.

2.2.2 Formas de aprendizado em mistura de especialistas

O mecanismo de aprendizado de MEs pode ser realizado de várias formas. No entanto, a função-objetivo é sempre a *função de verossimilhança* da mistura, a qual foi apresentada na equação (2.20) e representa a função-base que guiará o processo de aprendizado. Em termos de otimização de parâmetros, seria a função-objetivo a ser maximizada.

Uma vez definida a função-objetivo, resta definir como será realizado o ajuste dos parâmetros, tanto dos especialistas quanto da rede *gating*. Na literatura, há duas estratégias propostas: a primeira pode ser vista como uma tentativa inicial e foi proposta por JACOB *et al.* (1991). Ela diz respeito à forma de ajuste de parâmetros e pode ser vista como um ajuste *acoplado* e/ou *simultâneo*, como ilustrado no quadro esquerdo da Figura 2.7, onde todos os parâmetros, tanto dos especialistas como da rede *gating*, são ajustados ao mesmo tempo. Esta forma de ajuste aumenta o risco de convergência para um mínimo local indesejado.

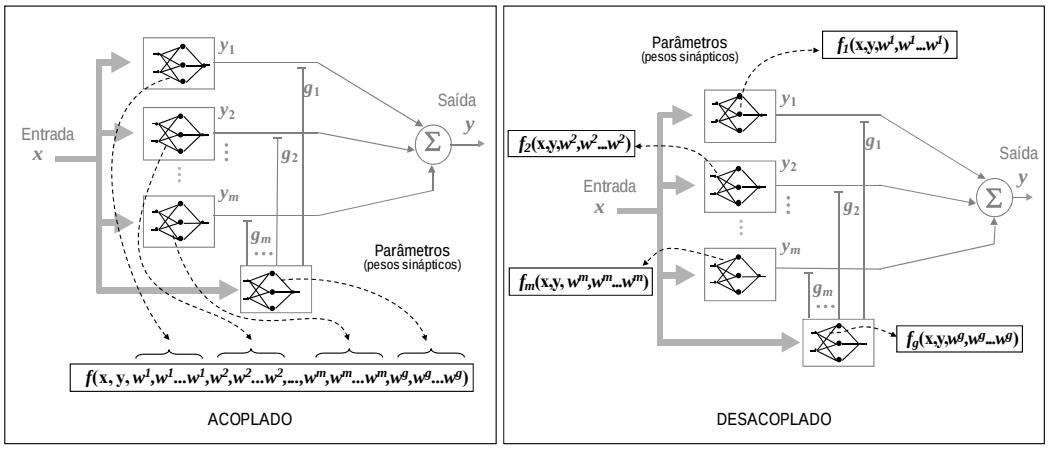


Figura 2.7 - Aprendizados acoplado e desacoplado para mistura de especialistas.

A segunda forma para o ajuste dos parâmetros de MEs é mais sofisticada e foi proposta por JORDAN & JACOBS (1994), sendo conhecida como método *desacoplado*. Para realizar o desacoplamento entre os especialistas e a rede *gating*, é utilizado o algoritmo de maximização da esperança (EM, do inglês *Expectation Maximization*) para MEs, e que foi inicialmente proposto por DEMPSTER *et al.* (1977) em outro contexto, voltado para aprendizado não-supervisionado. O algoritmo EM permite desacoplar o ajuste dos parâmetros do modelo de MEs. Dessa forma, o processo de treinamento, tanto dos especialistas quanto da rede *gating*, pode ser realizado independentemente, a cada passo incremental. Uma consequência direta é a possibilidade de explorar de forma mais efetiva o espaço de busca das soluções, consequentemente diminuindo o risco de cair num mínimo local indesejado, quando comparado ao método acoplado. O quadro direito da Figura 2.7 ilustra esta forma de ajuste de parâmetros. Observe que, neste caso, a *função* de *verossimilhança* dada pela equação (2.20) pode ser decomposta em soma de funções, as quais dependem individualmente dos parâmetros de cada especialista e da rede *gating*. Logo, a maximização da *função de verossimilhança* pode ser realizada maximizando-se essas funções individuais, de forma incremental e iterativa.

A seguir, será apresentado em detalhe o treinamento desacoplado.

2.2.2.1 Treinamento desacoplado via o método EM (*Expectation-Maximization*)

Para desenvolver este método de ajuste de parâmetros, parte-se do princípio de maximização da função de verossimilhança, definida na equação (2.20). Como é comumente adotado em estatística, é mais conveniente trabalhar com o logaritmo da verossimilhança que com a própria verossimilhança. Tomando o logaritmo de *m* densidades na forma da equação (2.20), chega-se à seguinte medida de verossimilhança:

$$l(\chi, \Theta) = \sum_{t=1}^{N} \log \sum_{i=1}^{m} P(i \mid \mathbf{x}^{(t)}, \mathbf{v}) P(\mathbf{y}^{(t)} \mid \mathbf{x}^{(t)}, \theta_i) \,, \tag{2.22}$$

sendo *N* o número de padrões de treinamento.

Partindo da equação (2.22), é possível utilizar o algoritmo de maximização da esperança (EM). Nesta abordagem, o ajuste dos parâmetros da rede *gating* e dos especialistas compreende dois passos bem definidos e engloba todo o conjunto de treinamento. A ideia é que a maximização da função de verossimilhança pode ser simplificada se cada padrão puder ser associado a exatamente um único especialista (indicado por variáveis chamadas de variáveis ausentes, que serão iguais ao valor 1 para um especialista e 0 para os demais). As variáveis ausentes serão importantes para a simplificação da otimização do logaritmo da verossimilhança. Os passos são os seguintes:

Passo E: É calculado o valor esperado para as variáveis ausentes (considerando que os parâmetros de todos os especialistas são conhecidos):

$$h_i^{(k)}(\mathbf{y}^{(t)} \mid \mathbf{x}^{(t)}) = \frac{g_i^{(k)}(\mathbf{x}^{(t)}, \mathbf{v}) \phi_i(\mathbf{y}^{(t)} \mid \mathbf{x}^{(t)})}{\sum_{j=1}^{m} g_j^{(k)}(\mathbf{x}^{(t)}, \mathbf{v}) \phi_j(\mathbf{y}^{(t)} \mid \mathbf{x}^{(t)})} \,, \tag{2.23}$$

onde $h_i^{(k)}(\mathbf{y}^{(t)} \mid \mathbf{x}^{(t)})$ são os valores das variáveis ausentes para o *t*-ésimo padrão de treinamento e o índice *k* indica o *k*-ésimo Passo E.

Passo M: O termo a ser maximizado pela rede *gating* será:

$$E_{gate} = \sum_{t=1}^{N}\sum_{i=1}^{m} h_i^{(k)}(\mathbf{y}^{(t)}, \mathbf{x}^{(t)}) \log(g_i(\mathbf{x}^{(t)})), \tag{2.24}$$

e para o conjunto de especialistas será:

$$E_{especialista} = \sum_{t=1}^{N}\sum_{i=1}^{m} h_i(\mathbf{y}^{(t)}, \mathbf{x}^{(t)}) \log(\phi_i(\mathbf{y}^{(t)} \mid \mathbf{x}^{(t)})), \tag{2.25}$$

onde ϕ é a densidade de probabilidade condicional:

$$\phi_i(\mathbf{y}^{(t)} \mid \mathbf{x}^{(t)}) = \frac{1}{(2\pi)^{d/2}\sigma_i} \exp\left\{-\frac{(\mathbf{y}^{(t)} - \mathbf{y}_i^{(t)})^T(\mathbf{y}^{(t)} - \mathbf{y}_i^{(t)})}{2\sigma_i^2}\right\}, \tag{2.26}$$

Com *d* sendo a dimensão de $\mathbf{y}^{(t)}$ e $\mathbf{y}_i^{(t)}$ sendo a saída do especialista *i* para o *t*-ésimo padrão de treinamento.

As *m* saídas da rede *gating* são dadas pela função *softmax*, da equação (2.19), com $\xi_i = g_i$.

Ao invés de usar uma rede *gating* com função de ativação *softmax* (conforme definido acima), uma outra abordagem é utilizar funções gaussianas normalizadas, cada uma centrada na região de atuação de um especialista. São conhecidas como MEs locais (LME, do inglês *local mixture of experts*), pois o espaço de entrada é dividido por hiper-elipsoides, facilitando a contribuição de vários especialistas para uma sub-região. As saídas (α_i e α_j) da rede *gating* assumirão uma nova forma ao passar pela nova função *softmax* dada por:

$$g_i(\mathbf{x}) = \frac{\alpha_i P(\mathbf{x} \mid \mathbf{v}_i)}{\sum_{j=1}^{m} \alpha_j P(\mathbf{x} \mid \mathbf{v}_j)}, \tag{2.27}$$

$$P(\mathbf{x} \mid \mathbf{v}_i) = (2\pi)^{-n/2}\left|\Sigma_i\right|^{-1/2} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{r}_i)^T \Sigma_i^{-1}(\mathbf{x}-\mathbf{r}_i)}. \tag{2.28}$$

Assim, a influência do *i*-ésimo especialista está localizada numa região ao redor de $\mathbf{r}_i$ (XU *et al.*, 1995; RAMAMURTI & GHOSH, 1996).

2.3 Algoritmos construtivos para o treinamento de redes neurais

Em abordagens tradicionais (por exemplo, a MLP) o processo de treinamento da rede exige a definição prévia da arquitetura (número de camadas ocultas e de neurônios por camada oculta) antes do treinamento dos pesos sinápticos da rede. Com isso, invariavelmente a definição da arquitetura acaba sendo feita de forma arbitrária e/ou seguindo a intuição do usuário. Assim, fica aumentado o risco da arquitetura não ser a mais adequada para o problema, resultando num aprendizado deficiente. Já no caso dos algoritmos construtivos, a definição da arquitetura da rede é feita automaticamente durante o treinamento, o qual também realiza o ajuste dos pesos sinápticos. Há também uma tendência de que os algoritmos construtivos forneçam arquiteturas mais parcimoniosas em termos de número de neurônios e conexões sinápticas (GONÇALVES *et al*., 1998), o que pode contribuir para uma boa capacidade de generalização.

Um comentário em favor dos algoritmos construtivos: "*a possibilidade de adaptação da rede neural para um determinado problema é uma das vantagens das técnicas construtivas... (elas) têm também importantes efeitos na velocidade da convergência do processo de treinamento. Em muitos métodos construtivos, a adição de um novo neurônio oculto implica na atualização de um pequeno grupo de pesos sinápticos, geralmente só aqueles associados ao neurônio a ser adicionado*" (MUSELLI, 1998).

Dentre os principais aspectos dos algoritmos construtivos, pode-se mencionar os seguintes:

Número de neurônios adicionados: Refere-se à quantidade de neurônios que se adiciona na rede a cada iteração do processo construtivo. Na maioria dos casos, é feita a adição de um neurônio a cada iteração.

Padrão de conexões do novo neurônio: Refere-se a um tipo de política de conexão do novo neurônio a ser adicionado à rede. Por exemplo, o novo neurônio será conectado com todos os neurônios de todas as camadas ocultas prévias, ou apenas com os neurônios da camada de entrada e da camada imediatamente anterior.

Inserção e/ou poda: Os algoritmos construtivos englobam não só algoritmos que partem de uma arquitetura mínima e buscam incrementar componentes (neurônios e conexões) na rede, mas também etapas de poda, que podem levar à redução do número de componentes da arquitetura, caso esta redução implique uma melhora de desempenho da rede.

Direção do crescimento da rede: Destacam-se três casos. O primeiro mostra crescimento a partir da camada de entrada até a camada de saída. E o segundo a partir da camada de saída até a camada de entrada. Todas aquelas estratégias que não se encaixam nas duas primeiras compõem o terceiro caso.

Funcionalidade dos neurônios inseridos: Embora a maioria dos algoritmos construtivos não faça diferença entre os neurônios ocultos, existem alguns que consideram a criação de camadas nas quais um neurônio assume um papel de maestro e os demais de auxiliares.

Ajuste dos pesos sinápticos do novo neurônio: A maioria dos algoritmos construtivos adota técnicas de ajuste de pesos locais e que só ajustam as novas conexões que o novo neurônio inserido na rede traz consigo. Dentre as técnicas mais utilizadas, pode-se citar: Pocket (GALLANT, 1986), Pocket Ratchet Modification PRM (GALLANT, 1990), Quickprop (FAHLMAN, 1988), Barycentric Correction Procedure – BCP (POULARD & LABRECHE, 1995).

Critério de parada: O critério de parada mais utilizado é o desempenho da rede, seja em termos de taxa de acertos/erro para problemas de classificação ou a minimização quantitativa do erro, como o erro quadrático médio para problemas de regressão de dados. O processo construtivo é concluído quando estes valores permanecem constantes ou com variação não-significativa entre duas ou mais iterações. Em casos especiais de restrições de uso de memória, o critério de parada pode ser complementado com a definição de um número máximo de neurônios a ser atingido pelo algoritmo construtivo.

Forma da rede: Refere-se à configuração topológica que a rede vai adquirindo à medida que vai sendo construída. Por exemplo, existem formas de pirâmide e torre com apenas um neurônio de saída associado a problemas de bi-classificação, ou outra de uso mais geral em forma de cascata com múltiplos neurônios de saída associados a problemas tanto de classificação quanto regressão de dados.

Tipo de variáveis de entrada: Alguns algoritmos construtivos aceitam apenas entradas binárias, enquanto que outros admitem valores categóricos ordinais e não-ordinais, e também valores numéricos.

Tipo de problemas que resolvem: As primeiras propostas de algoritmos construtivos estavam direcionadas a problemas de classificação de padrões (de duas e de mais de duas classes), onde as entradas são associadas a uma ou mais classes. Em seguida, foram estendidas para problemas de regressão de dados (FAHLMAN & LEBIERE, 1990), que se caracterizam por um mapeamento contínuo de entradas e saídas. Existem também algoritmos construtivos para problemas de agrupamento (*clustering*) (FRITZKE, 1995).

Dentre as contribuições mais importantes de algoritmos construtivos que usam unicamente estratégias de incremento de componentes (conexões e neurônios) na rede, é possível citar os algoritmos *Tower* e *Pyramid* (GALLANT, 1994), concebidos para problemas de classificação de duas classes e com um único neurônio de saída. Estes algoritmos partem de um único neurônio na camada oculta (que é também o neurônio de saída). Cada neurônio inserido na rede é o único neurônio de saída. No *Tower*, cada novo neurônio é conectado a todas as entradas e ao último neurônio inserido na rede. Já o *Pyramid* difere do *Tower* apenas na conectividade do novo neurônio, o qual é conectado com todos os neurônios previamente inserido na rede. Por outra parte, ambos utilizam funções de transferência com limiar (*threshold*) e as conexões do novo neurônio são treinadas empregando o algoritmo *Pocket Ratchet Modification* – PRM (GALLANT, 1990).

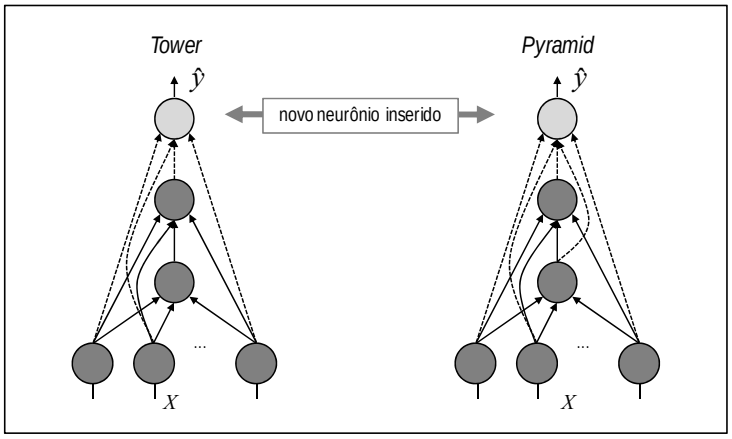


Figura 2.8 – Exemplos de redes neurais produzidas pelos algoritmos construtivos *Tower* e *Pyramid*.

Outro algoritmo com forma distinta de crescimento é o *Upstart* (FREAN, 1990). Este cresce em forma de árvore binária e parte da camada de saída em direção à camada de entrada. Durante o processo construtivo, um neurônio se divide em outros dois neurônios visando corrigir o erro cometido pelo neurônio ancestral. Nesta estratégia de correção do erro, os neurônios descendentes têm a tarefa direta de corrigir os erros do neurônio ancestral. Indiretamente, na maioria dos casos acaba-se por reduzir o erro de classificação da rede neural como um todo.

Também foram propostos algoritmos construtivos com maior abrangência de aplicação, que além de tratar problemas de classificação resolvem também problemas de regressão de dados. Pode-se citar o algoritmo *Cascade Correlation* (*CasCorr*) (FAHLMAN & LEBIERE, 1990), que começa sem neurônios ocultos e ajusta apenas os pesos sinápticos dos neurônios de saída. A cada iteração, busca-se ganho de desempenho ao se adicionar um neurônio oculto por vez, de forma a produzir uma cascata, fixando-se os pesos dos neurônios já inseridos anteriormente e reajustando os pesos dos neurônios de saída. Com isso, utiliza-se uma estratégia de congelamento dos pesos sinápticos para apenas ajustar os pesos associados ao novo neurônio inserido e os pesos dos neurônios de saída. Este algoritmo será descrito em forma detalhada na Seção 2.3.1.

Considerando as propostas de algoritmos evolutivos que utilizam somente estratégias de poda, tem-se o trabalho de REED (1993), o qual descreve algoritmos que partem com uma arquitetura de rede maior do que a necessária e procedem com a remoção de suas conexões e neurônios ocultos. Lahnajärvi e colaboradores (LAHNAJÄRVI *et al.*, 2002) propuseram uma taxonomia para as estratégias de poda: (*i*) Algoritmos que utilizam estimativas de sensibilidade da função de erro para remover conexões e neurônios ocultos. Aqueles que, ao serem removidos temporariamente produzem menor efeito, são finalmente selecionados para remoção permanente da rede. (*ii*) Algoritmos munidos com um termo de penalidade adicionado na função-objetivo, de forma a penalizar redes de grande tamanho e que vão contra o princípio da parcimônia.

Também podem ser encontradas na literatura propostas de algoritmos que combinam ambas as estratégias (crescimento e poda). Estes tentam tomar vantagem de ambas as estratégias para determinar a arquitetura da rede (FIESLER, 1994; GHOSH & TUMER, 1994). Cabe destacar aqui o algoritmo de aprendizado por busca de projeção (PPL, do inglês *projection pursuit learning*), o qual produz redes neurais com uma única camada oculta, mas define automaticamente o número de neurônios e o formato de suas funções de ativação, visando maximizar a capacidade de generalização (HOLSCHUH, 2008; HWANG *et al.*, 1994; VON ZUBEN, 1996).

No recente trabalho de Franco e colaboradores (FRANCO *et al.*, 2009), é possível encontrar descrições detalhadas destes e outros algoritmos construtivos associados a problemas de classificação de dados.

2.3.1 O algoritmo construtivo *Cascade-Correlation* (*CasCorr*)

O termo *CasCorr* pode ser atribuído tanto à arquitetura quanto ao algoritmo de aprendizado construtivo. Além de ajustar os pesos sinápticos, também define a topologia da rede neural, numa forma especial em cascata. O algoritmo começa a partir de uma rede de arquitetura mínima (modelo linear, sem neurônios ocultos) e ajusta otimamente os pesos dos neurônios de saída. Em seguida, adiciona novos neurônios ocultos, um de cada vez,

criando uma arquitetura de rede de múltiplas camadas ocultas, com apenas um neurônio por camada, caracterizando a configuração em cascata.

O *CasCorr* combina duas ideias: (*i*) a arquitetura em forma de cascada (como ilustrado na Figura 2.9, considerando apenas dois neurônios ocultos), onde os neurônios ocultos estão dispostos de forma que cada um recebe conexões de todos os neurônios precedentes (inseridos anteriormente e considerando também os da camada de entrada); (*ii*) o algoritmo de treinamento da rede, que é o responsável pela dinâmica de inserção de novos neurônios (demanda do problema) e do ajuste dos pesos sinápticos.

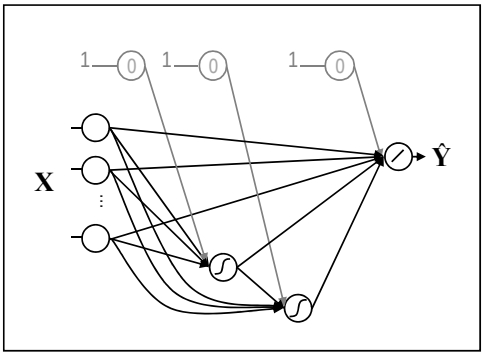
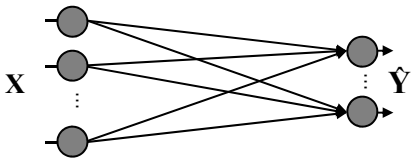
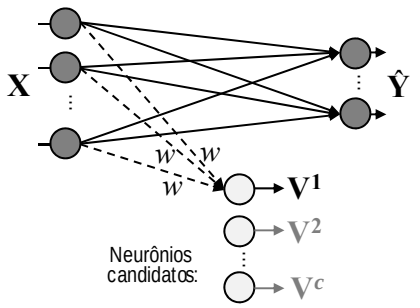
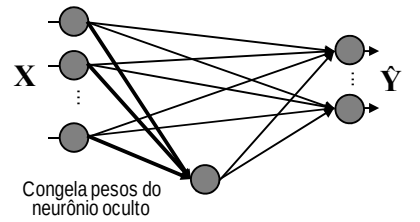


Figura 2.9 – Arquitetura da rede neural do tipo *Cascade Correlation* (*CasCorr*).

Em relação aos componentes da arquitetura, a *CasCorr* é bem parecida à tradicional MLP. Por exemplo, as funções de ativação dos neurônios ocultos são do tipo sigmoidal, seja a função logística ou a tangente hiperbólica. Da mesma forma, os neurônios de saída possuem funções lineares. A diferença frente à MLP está: (*i*) no número de neurônios nas camadas ocultas das redes, sendo que na *CasCorr* admite-se apenas um neurônio em cada camada oculta, enquanto na MLP este número é arbitrário e pode ser distinto para cada camada; (*ii*) na disposição das conexões, pois na MLP as conexões são restritas a ligarem neurônios de camadas contíguas, enquanto que na *CasCorr* cada neurônio recebe todas as conexões possíveis de neurônios em camadas anteriores. Em ambas as redes não se admitem conexões entre neurônios da mesma camada, nem conexões recorrentes. O procedimento de ajuste dos pesos sinápticos é também bem distinto entre ambas as propostas de arquitetura, como já descrito.

A seguir, a Figura 2.10 contém uma sequência de passos extraídos de FAHLMAN & LEBIERE (1990) e que descrevem o processo de treinamento para o *CasCorr*. À esquerda, a descrição dos passos e, à direita, suas figuras associadas.

P1: Parte-se com uma arquitetura de rede mínima, sem neurônios ocultos e com todas as conexões entre as camadas de entrada e saída. Ajustam-se os pesos da rede minimizando o EQM via o algoritmo *Quickprop* (*backpropagation* modificado).	
P2: Ajustam-se, de forma independente, os pesos das conexões entrantes dos c neurônios candidatos (c definido pelo usuário) maximizando a função de correlação S entre o comportamento de saída do neurônio e o erro entre a saída desejada e aquela produzida pela configuração atual de rede neural. Nas equações, o índice o indica cada saída da rede, p cada padrão de treinamento, $\mathbf{V}$ a saída de cada neurônio candidato, $\overline{\mathbf{V}}$ o valor médio de $\mathbf{V}$, $\mathbf{E}$ o erro da rede e $\overline{\mathbf{E}}$ o valor médio de $\mathbf{E}$. O vetor gradiente desta função S é calculado a partir da equação que descreve as derivadas parciais $\partial S / \partial w_i$. Nesta última equação, σ_o é o sinal da correlação entre a saída do neurônio candidato e a saída da rede o, f'_p é a derivada da função de ativação do neurônio candidato para o padrão de treinamento p, e $I_{i,p}$ é o valor da entrada i do neurônio candidato gerada pelo padrão de treinamento p.	 $S = \sum_o \left| \sum_p (\mathbf{V}_p - \overline{\mathbf{V}})(\mathbf{E}_{p,o} - \overline{\mathbf{E}}_o) \right|$ $\frac{\partial S}{\partial w_i} = \sum_{p,o} \sigma_o (\mathbf{E}_{p,o} - \overline{\mathbf{E}}_o) f'_p I_{i,p}$
P3: Após ter ajustado os pesos das conexões dos c neurônios candidatos, escolhe-se o neurônio cuja correlação S seja a mais alta. Em seguida "congelam-se" os valores destes pesos (linhas mais grossas na figura à direita), estes pesos permanecerão inalterados até o fim do processo de treinamento. Em seguida, criam-se as conexões do novo neurônio com os neurônios de saída da rede e treinam-se os pesos destas conexões junto com as conexões restantes (não-congeladas) minimizando a função do EQM da rede.	

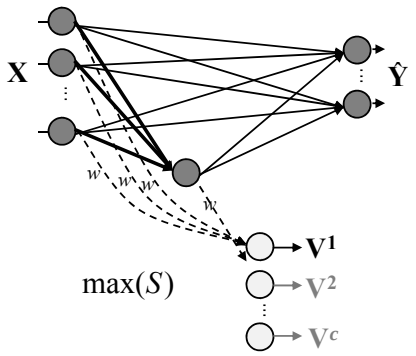
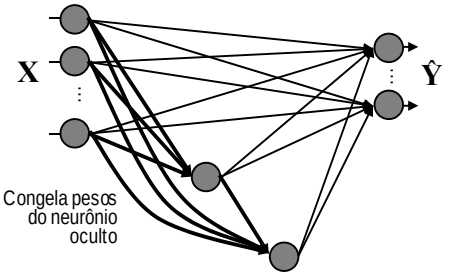
P4: A partir deste ponto, repete-se o processo de inserção de um novo neurônio na rede, criando-se os candidatos, treinando seus pesos visando maximizar a correlação *S*.	
P5: Em seguida, congelam-se os pesos das conexões do novo neurônio (estes pesos, assim como os dos outros neurônios já inseridos, se manterão inalterados). Criam-se as conexões de saída do novo neurônio com os neurônios da camada de saída e, finalmente, treinam-se os pesos visando a minimização do EQM. Repetem-se estes passos até não obter mais uma diminuição no erro, ou caso um valor de erro desejado seja atingido (critérios de parada alternativos).	

Figura 2.10 – Passos do processo de treinamento no algoritmo *CasCorr*.

2.4 Algoritmos de computação evolutiva

Os princípios da evolução natural, seleção natural e genética serviram de inspiração para a concepção de algoritmos que capturam os seus principais mecanismos. Vislumbrou-se que estes princípios evolutivos, que supostamente levaram ao aparecimento e aperfeiçoamento de espécies de seres vivos no nosso planeta, podiam também ser usados para o aparecimento e aprimoramento de soluções de alta qualidade junto a problemas reais da engenharia, como os de *busca estocástica* e problemas de *otimização*.

Este paradigma da computação evolutiva sugere um conjunto de mecanismos agindo numa população de indivíduos (candidatos à solução do problema) com o objetivo de melhorar, em média, a sua adequação em relação ao ambiente, ou seja, o seu desempenho geral em relação a um problema proposto (GOLDBERG, 1989; SRINIVAS & PATNAIK, 1994).

Os algoritmos que formam parte do paradigma de computação evolutiva são os seguintes: algoritmos genéticos (GA), programação evolutiva (EP), estratégias evolutivas (ES) e programação genética (GP). Os algoritmos GA e EP serão alvo de investigação nesta tese e serão tratados mais detalhadamente. Os principais conceitos que estes algoritmos têm em comum são os seguintes:

Codificação do problema: É importante que o problema a resolver (busca/otimização) seja codificado num formato estruturado que permita a aplicação dos mecanismos de evolução em computador. Esta codificação deverá ser capaz de representar uma solução do problema, independente da qualidade da solução. Pode-se fazer uma analogia com o cromossomo (onde se encontra a informação de DNA) dos seres vivos. O cromossomo seria a codificação (*genótipo*) que é o ponto de partida para se chegar a um ser vivo (*fenótipo*), com todas as suas características visíveis (o ser humano, por exemplo). Cada elemento do cromossomo é conhecido como gene. Os possíveis valores que um determinado gene pode assumir são denominados alelos.

A Figura 2.11 tem como objetivo estender os conceitos do cromossomo biológico para ilustrar casos de codificação de soluções de dois problemas computacionais reais. A parte da esquerda da figura ilustra o problema de otimização da arquitetura de redes neurais arbitrariamente conectadas (ACFNN, do inglês *arbitrarily connected feedforward neural network*). A codificação deverá permitir representar qualquer arquitetura que siga as especificações deste tipo de rede neural, ou seja, todas as soluções-candidatas devem admitir uma representação na codificação proposta. Para isto, foram utilizados valores binários dentro do vetor que representa o cromossomo (de acordo com o GA tradicional). Nesta parte da figura, o vetor que representa o cromossomo codifica a arquitetura de rede mostrada logo abaixo. Resulta uma rede neural com um total de 7 neurônios, 3 dos quais pertencentes às 2 camadas ocultas. A matriz de conexões abaixo da rede serve de auxílio na geração do vetor que representa o cromossomo, pois nela são empregados os valores 0 e 1 para representar, respectivamente, a não-existência e a existência de conexões entre os neurônios linha/coluna. O vetor que representa o cromossomo foi preenchido percorrendo sequencialmente os elementos das linhas da triangular superior da matriz.

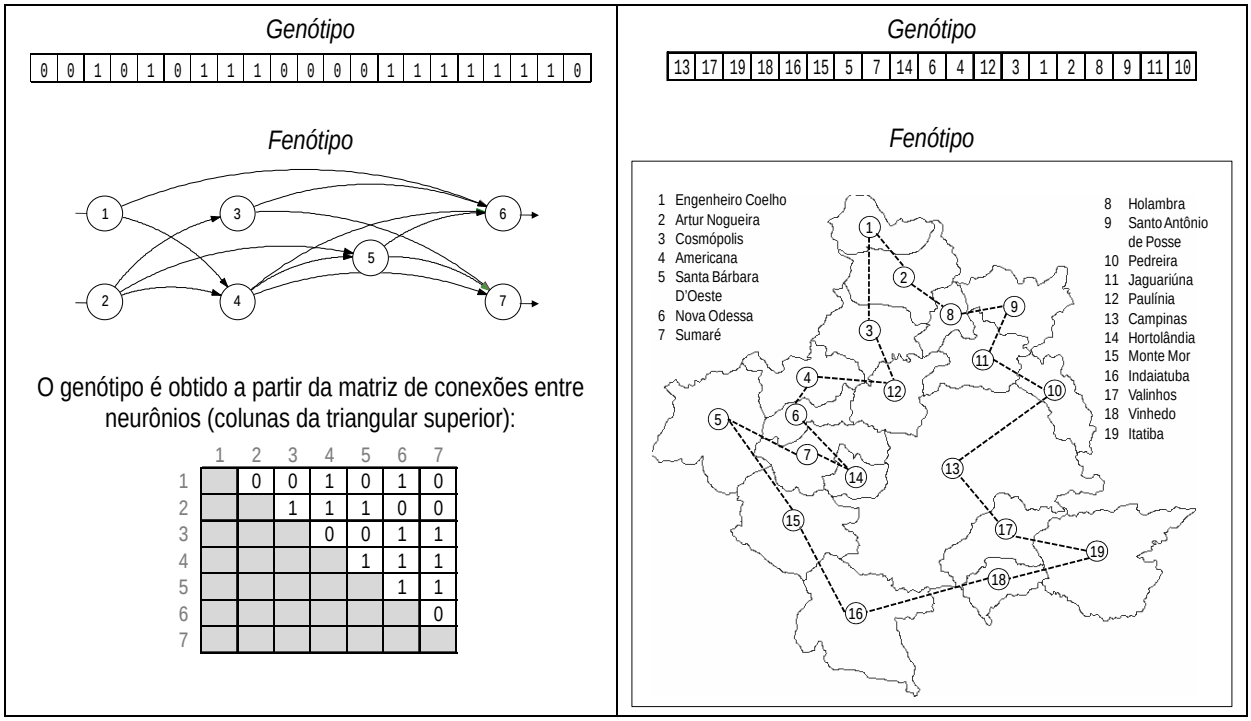


Figura 2.11 – Exemplos ilustrativos de codificação e mapeamento *genótipo* ⇔ *fenótipo*.

O GA tradicional foi proposto para uso de codificação com valores binários, porque com esta codificação é possível representar tanto variáveis categóricas não-ordinais, como categóricas ordinais e até variáveis numéricas, seguindo, por exemplo, a representação quantizada adotada em computação digital, a chamada codificação em ponto flutuante.

As principais desvantagens desta codificação estão na criação de cromossomos binários de comprimento elevado, em certas aplicações, e a dificuldade de manter a relação entre número de bits diferentes no cromossomo e proximidade de grau de adaptação entre as soluções-candidatas (MICHALEWICZ, 1996). Com isso, tem-se um aumento de custo computacional e, possivelmente, uma maior complexidade da superfície de adaptação, que é aquela que permite mapear cada ponto do espaço de soluções-candidatas em um único valor de adaptação. Por esta razão, já foram propostas na literatura diversas alternativas que buscam empregar outros tipos de codificação, como codificação inteira e real (que na verdade recorre à representação em ponto flutuante do ambiente de computação digital) (BÄCK *et al*, 1997; FOGEL, 1995).

Um exemplo de codificação usando valores discretos inteiros é ilustrada na parte direita da Figura 2.11. Trata-se de uma solução-candidata para o problema do *caixeiro viajante* (*traveling salesperson problem*), que corresponde a um bem conhecido problema de otimização combinatória (MICHALEWICZ , 1996). Este problema consiste em encontrar a sequência de cidades que o caixeiro deverá visitar, passando uma única vez por cada cidade e retornando à cidade de origem de forma a minimizar a distância total percorrida. A ordem de complexidade do problema cresce de forma fatorial segundo o número de cidades, pois a única forma conhecida para se garantir a obtenção da solução ótima é testar todas as permutações de cidades e ficar com aquela de menor percurso total. O número de possíveis permutações, ou seja, de soluções-candidatas, é de $\frac{(Nc-1)!}{2}$, sendo *Nc* o número de cidades, isso quando não importa qual será a cidade de origem e nem o sentido de execução do percurso, se horário ou anti-horário. Uma solução para este problema pode ser codificada com um vetor de números inteiros (*genótipo*) contendo os índices que foram arbitrária e unicamente atribuídos a cada cidade. Fica evidente que a ordem destes índices no cromossomo definirá a ordem em que o caixeiro deverá visitar as cidades (*fenótipo*).

População de indivíduos e sua inicialização: Os algoritmos evolutivos são de tipo populacional, o que significa que cada indivíduo desta população interage, seguindo princípios evolutivos, com os demais indivíduos. Cada indivíduo representa uma possível solução do problema, ao contrário de outros algoritmos (como *Simulated Annealing* e Busca Tabu) em que apenas uma única solução é considerada a cada passo. Após ter definida a forma como o problema será codificado, o que acaba por especificar o espaço de busca (que contém todas as soluções-candidatas), resta a criação de uma *população inicial* de indivíduos, sendo que a forma comumente utilizada é a geração aleatória de indivíduos. Na ausência de qualquer conhecimento acerca da localização da solução ótima ou ao menos de regiões promissoras do espaço de busca, a inicialização aleatória promove uma amostragem uniforme do espaço de busca. Quando se dispõe de algum conhecimento inicial, este pode ser utilizado para polarizar a inicialização. Além disso, em situações em que o espaço de busca contém tanto soluções factíveis quanto infactíveis, a inicialização deve se ocupar em evitar amostras infactíveis.

Operador de seleção: Este operador tem a função de favorecer indivíduos mais aptos quando eles são selecionados para gerar descendentes, tomados a partir da população em curso. Embora existam propostas de seleção determinísticas, normalmente é inserido algum grau de aleatoriedade a este processo, visando preservar níveis mínimos de diversidade na população. Com isso, é possível que nem todos os melhores indivíduos sejam selecionados e é possível que alguns dentre os piores indivíduos sejam selecionados.

Operador de recombinação: Conhecido também como operador de *crossover*, está encarregado da geração de novos indivíduos da população por meio da troca de material genético entre indivíduos previamente selecionados. Não há garantias de que as soluções resultantes desta recombinação venham a produzir soluções-candidatas de melhor nível de adaptação, mas existe a possibilidade de que bons trechos de código, presentes em indivíduos diferentes, possam ser simultaneamente herdados por um único indivíduo, numa das recombinações implementadas.

Operador de mutação: O operador de mutação implica na modificação dos valores de um subconjunto de elementos do cromossomo, muitas vezes chamados de genes. A mutação pode ser interpretada como um processo de perturbação aleatória de uma dada solução-candidata, e acaba representando uma fonte de inovação para o processo de busca promovido pela evolução e também um recurso a mais para preservar um nível mínimo de diversidade na população. A existência de mutação garante inclusive que, dados recursos computacionais suficientes, é possível atingir qualquer ponto do espaço de busca a partir do emprego iterativo deste operador.

A Figura 2.12 ilustra o fluxograma de um algoritmo evolutivo padrão.

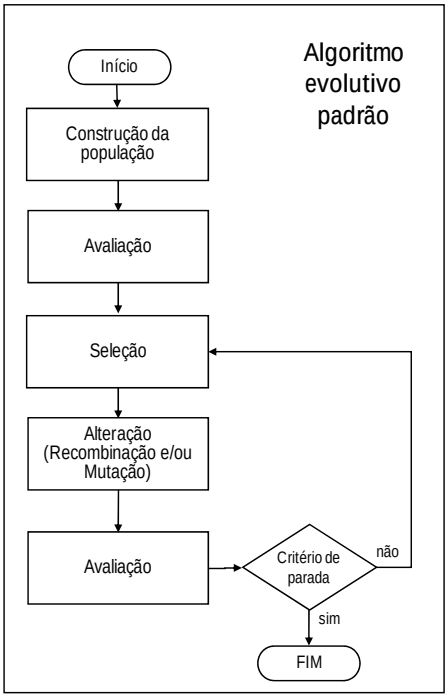


Figura 2.12 – Fluxograma de um algoritmo evolutivo padrão.

Procedimento de Busca Local: Consiste na aplicação de certas ações com o intuito de melhorar a aptidão de um indivíduo da população. O objetivo é acelerar a convergência do indivíduo para um mínimo local mais próximo. O tempo computacional que esta tarefa pode demandar geralmente é proporcional à qualidade do mínimo local alcançado, razão pela qual não se acostuma aplicar ao total de indivíduos da população e sim a uma pequena parcela deles. As ações que compõem o procedimento de busca local variam de acordo com o problema que está sendo resolvido. Por exemplo, no problema do caixeiro viajante, uma ação pode ser testar (avaliar) todas as possibilidades de troca de rotas entre duas cidades e consolidar a troca se esta conduzir a um incremento da aptidão do indivíduo. Outro exemplo de busca local para o problema de evoluir os pesos sinápticos de uma rede neural: uma ação pode ser utilizar um método analítico de otimização baseado na *descida do gradiente*. Esta ação, por usar informação a partir das derivadas da função do erro de treinamento em relação aos pesos sinápticos, leva à redução do erro e, consequentemente à melhora da aptidão do indivíduo. Todos os algoritmos baseados em computação evolutiva admitem a aplicação de procedimentos de busca local.

2.4.1 Algoritmo genético (GA)

O algoritmo genético (GA, do inglês *genetic algorithm*) foi desenvolvido por J. H. Holland (HOLLAND, 1975) e emprega uma notação proveniente da teoria da evolução natural e da genética. Assim, um indivíduo da população é representado por um único cromossomo (comumente representado no computador por um vetor de atributos), o qual contém a codificação (genótipo) de uma possível solução do problema (fenótipo). Na versão de Holland para um GA, os alelos são representados por valores binários 0 e 1.

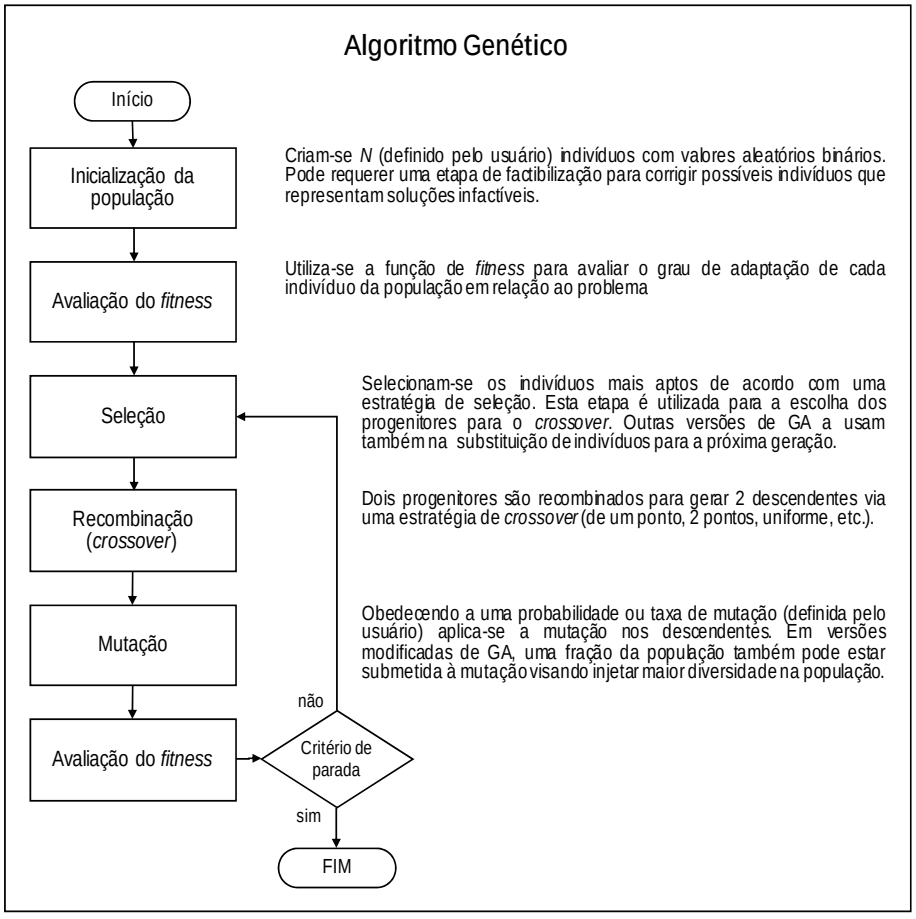


Figura 2.13 – Fluxograma de um algoritmo genético padrão.

2.4.2 Programação evolutiva (EP)

A EP (do inglês *evolutionary programming*) foi desenvolvida por FOGEL (1962, 1964), tendo como propósito a evolução de máquinas de estado finito. A transformação de sequências de símbolos de entrada em sequências de símbolos de saída pelas máquinas de estado finito visava tratar a predição de séries temporais levando em conta informações históricas da série. A EP, posteriormente, foi estendida para problemas de otimização de parâmetros, onde a codificação é do tipo real.

Uma diferença importante da EP para o GA é que a EP tende a enfatizar mais o vínculo comportamental, no espaço fenotípico, entre um ancestral e seu correspondente descendente, mais do que o vínculo genotípico. Esta manutenção de comportamento é devida à ausência do operador de *crossover*, o qual é utilizado para a geração de descendentes no GA. O operador de mutação é que assume essa responsabilidade. Assim, o descendente é o resultado da submissão do progenitor à operação de mutação.

Em EP, considerando problemas de codificação real, à ideia de indivíduo (vetor cromossomo) são acrescentados os parâmetros de *auto-adaptação*. Por tal motivo, dada a população de tamanho P, cada indivíduo $i = \{1, 2, .., P\}$ está constituído por uma dupla de vetores de tamanhos iguais $(\mathbf{x}_i, \boldsymbol{\sigma}_i)$, o vetor cromossomo e seu vetor de desvio-padrão associado, respectivamente. Estes vetores têm seus valores iniciais aleatórios extraídos de uma distribuição uniforme em intervalos especificados pelo usuário.

Cada indivíduo ancestral $(\mathbf{x}_i, \boldsymbol{\sigma}_i)$ gera um único descendente $(\mathbf{x}_i', \boldsymbol{\sigma}_i')$ via o operador de mutação, de acordo com as seguintes equações:

$$\boldsymbol{\sigma}_i' = \boldsymbol{\sigma}_i + N\left(0, \frac{\boldsymbol{\sigma}_i}{c}\right) \tag{2.29}$$

$$\mathbf{x}_i' = \mathbf{x}_i + N(0, \boldsymbol{\sigma}_i') \tag{2.30}$$

onde $N(0, \boldsymbol{\sigma}_i)$ representa um vetor de variáveis aleatórias gaussianas com média zero e desvio-padrão para cada elemento indicado no vetor $\boldsymbol{\sigma}_i$, e c uma constante de escala.

O algoritmo de EP padrão é apresentado na Figura 2.14 (FOGEL, 1994, 1995):

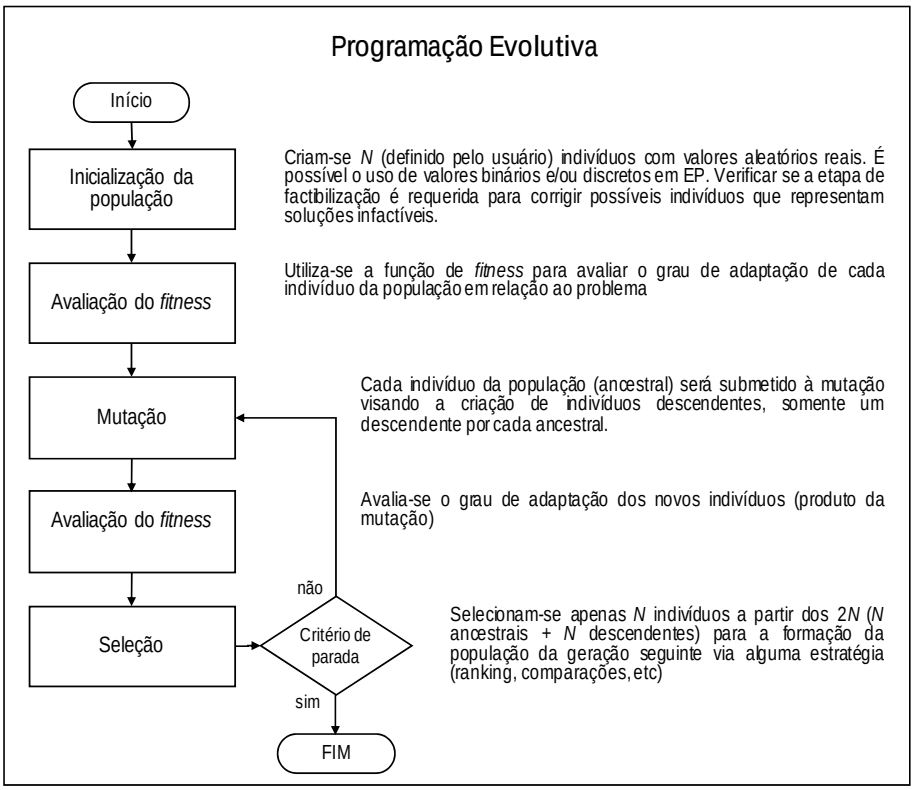


Figura 2.14 – Fluxograma de um algoritmo padrão de programação evolutiva.

Embora o EP seja mais utilizado em problemas de otimização em espaços contínuos, é possível aplicá-lo em problemas definidos em espaços discretos, como é o caso de codificação binária usado por Yao & Liu (1997) para otimizar a arquitetura de redes neurais, sendo que os pesos sinápticos foram ajustados via outras metodologias.

2.5 Índices derivados de teoria de informação

A teoria de informação, ao formalizar o conceito de entropia em processamento de sinais, pode fornecer um elenco amplo de índices capazes de mensurar associações entre variáveis e divergência entre modelos, dentre outras funcionalidades.

Na definição de critérios de desempenho e no ajuste de parâmetros no treinamento de redes neurais artificiais, particularmente aqueles dotados de mecanismos de aprendizado supervisionado, geralmente se adotam estatísticas de segunda ordem, devidamente formalizadas nos trabalhos pioneiros de Wiener (WIENER, 1949) e Widrow (WIDROW & STEARNS, 1985). Essas estatísticas de segunda ordem foram originalmente desenvolvidas para lidarem com modelos lineares nos parâmetros, e foram imediatamente estendidas para tratar o caso de modelos não-lineares nos parâmetros. Tratam-se de decisões de projeto que vêm sendo tomadas pela grande maioria dos projetistas de sistemas adaptativos, por mais de quatro décadas. No entanto, existem critérios mais abrangentes que, por exemplo, a tradicional estimativa de erro quadrático médio, capazes de considerar estatísticas de ordem superior e pertencentes à teoria de informação. Esses critérios mais abrangentes são essenciais em aplicações como filtragem adaptativa (HAYKIN, 2000), processamento de sinais (MENDEL, 1991), e análise de componentes independentes (BOSCOLO *et al.*, 2001).

Em aprendizado de máquina, as aplicações mais difundidas de critérios fundamentados em teoria de informação estão em seleção de variáveis e seleção de características (PRINCIPE, 2010), muito utilizadas em aplicações que vão desde predição de séries temporais até classificação de padrões (GUYON & ELISSEEFF, 2003). O conceito mais empregado nesses contextos é o de informação mútua.

Dentre as várias definições que se pode encontrar na literatura para a informação mútua (MI, do inglês *mutual information*), é possível destacar:

- Informação mútua mede a quantidade de informação que uma variável contém sobre uma outra variável.
- É uma medida de dependência entre duas variáveis (SHANNON, 1949).
- É a informação armazenada de uma variável em relação a uma outra.
- É o grau de previsibilidade de uma segunda variável, conhecendo-se a primeira.

Em termos práticos, dados dois vetores contendo amostras de duas variáveis aleatórias, se elas são independentes, então a informação mútua entre os dois vetores tende a zero. Se

as variáveis forem fortemente dependentes, então a informação mútua entre elas será máxima. Correlação e dependência estão diretamente associadas com informação mútua.

Os índices de informação mútua foram inicialmente utilizados na área de comunicações e estão embasados na Teoria de Informação (COVER & THOMAS, 1991; SHANNON & WEAVER, 1949). Posteriormente, essas ideias foram utilizadas para medir correlações não-lineares entre duas variáveis (CELLUCCI *et al.*, 2005; FRASER & SWINNEY, 1986). Na atualidade, informação mútua está sendo empregada em várias áreas do conhecimento, dentre elas destaca-se Aprendizado de Máquina (PRINCIPE, 2010).

A formulação para o cálculo da informação mútua de variáveis discretas (**x** e **y)**, denotada na forma $\mathrm{MI}(\mathbf{x},\mathbf{y})$, é fornecida na seguinte equação:

$$\mathrm{MI}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N}\sum_{j=1}^{N} P_{xy}(x_i, y_j)\log_2\left\{\frac{P_{xy}(x_i, y_j)}{P_x(x_i)P_y(y_j)}\right\} \tag{2.31}$$

onde **x** e **y** são variáveis que podem assumir valores em qualquer escala. $P_{xy}(x_i, y_j)$ é a probabilidade conjunta de $\mathbf{x} = x_i$ e $\mathbf{y} = y_j$, $P_x(x_i)$ e $P_y(y_j)$ são as probabilidades marginais de **x** e **y**, respectivamente, e *N* é o número de amostras consideradas no cálculo.

Com N finito, o cálculo da informação mútua leva a um valor aproximado. Repare que, para o cálculo de $\mathrm{MI}(\mathbf{x},\mathbf{y})$, é preciso aproximar uma função densidade de probabilidade envolvendo duas variáveis.

Nesta tese, a implementação para o cálculo da $\mathrm{MI}(\mathbf{x},\mathbf{y})$ está baseada no trabalho de CELLUCCI *et al.* (2005), o qual descreve o uso de dois tipos de histogramas, um uniforme e outro adaptativo, para o cálculo das probabilidades conjunta e marginais de **x** e **y**. Foi considerada aqui a primeira proposta, histograma uniforme de $\sqrt{N/5}$ elementos.

As Figuras 2.15 e 2.16 têm como objetivo mostrar exemplos de uso desta medida de informação mútua. Na Figura 2.15, mostram-se as variáveis **x** e **y** em três configurações: (1) à esquerda, sem nenhuma correlação entre elas, ambas as variáveis obedecem a distribuições uniformes aleatórias e independentes; (2) na parte central, uma situação na

que a correlação é máxima, no caso linear com $\mathbf{x} = \mathbf{y}$; e (3) à direita, um caso de correlação não-linear visando mostrar a capacidade do índice de informação mútua de capturar correlações deste tipo. Foi incluída a medida do coeficiente de correlação de Pearson (*Corr*) para efeito de comparação. Analisando o primeiro caso, as duas correlações apresentaram valores baixos, próximos de zero para indicar a não existência de associação entre **x** e **y**. Para o segundo caso, situação de máxima correlação, *Corr* indicou 1.0 (seu valor máximo, independente do número de pontos, e positivo porque à medida que **x** cresce, **y** também cresce). Já no caso do índice de informação mútua, seu valor máximo depende do número de amostras das variáveis e, neste caso, o valor indicado (3,7004) é o seu valor máximo por se tratar da mesma variável ($\mathbf{x} = \mathbf{y}$). Finalmente para o terceiro caso, *Corr* indicou um valor baixo, –0,1280 (em termos de valor absoluto, seria aproximadamente a oitava parte da medida de máxima correlação), próximo à situação de variáveis aleatórias independentes. No entanto, há forte correlação entre as variáveis, só que esta correlação é não-linear. O índice de informação mútua, por sua vez, indicou um valor de 0,8757 que seria aproximadamente a quarta parte da medida de máxima correlação. Este exemplo consegue mostrar a maior robustez da informação mútua na detecção de correlações não-lineares.

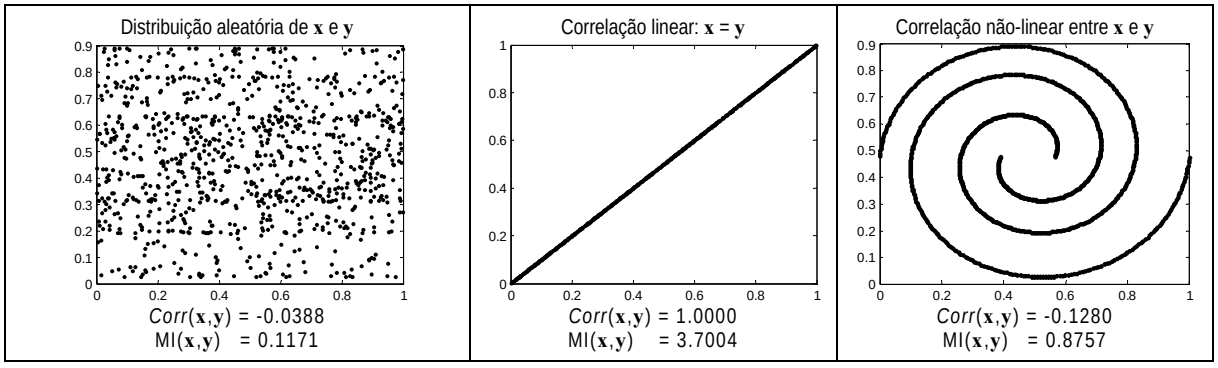


Figura 2.15 – Exemplos didáticos do uso de informação mútua comparado ao coeficiente de Pearson.

Um outro exemplo que ilustra o uso de informação mútua é apresentado na Figura 2.16, desta vez aplicado a dados reais. É o caso do problema da definição da janela de atrasos no contexto de predição de séries temporais. A janela de atrasos é um valor que indica o número mínimo de valores passados da série temporal necessário para sintetizar um modelo

preditor (que pode ser uma RNA, por exemplo). Para realizar esta tarefa, é preciso montar duas sequências de valores da série: S_t e S_{t-1}, ambas do mesmo tamanho. S_{t-1} deve conter valores com um atraso de S_t, e medir a correlação MI (S_t, S_{t-1}). Na Figura 2.16, mediu-se a correlação até 50 unidades de atraso para fins de identificar a janela. Na parte esquerda da Figura 2.16, duas séries temporais de natureza financeira e mensais, pelo que se espera uma janela de valor 12. Na série de cima, é fácil perceber a sazonalidade mensal com certos picos a cada 12 observações. Observa-se que tanto *Corr* como *MI* identificaram que a cada 12 atrasos o valor da correlação é alto. Logo, a janela mínima seria de 12. Já para a segunda série, o *Corr* não conseguiu identificar algum tipo de pico de alta correlação, e o *MI* conseguiu identificar picos a cada 12 observações. Novamente, este exemplo sugere a robustez de índices vinculados à Teoria de Informação, como é o caso aqui da informação mútua.

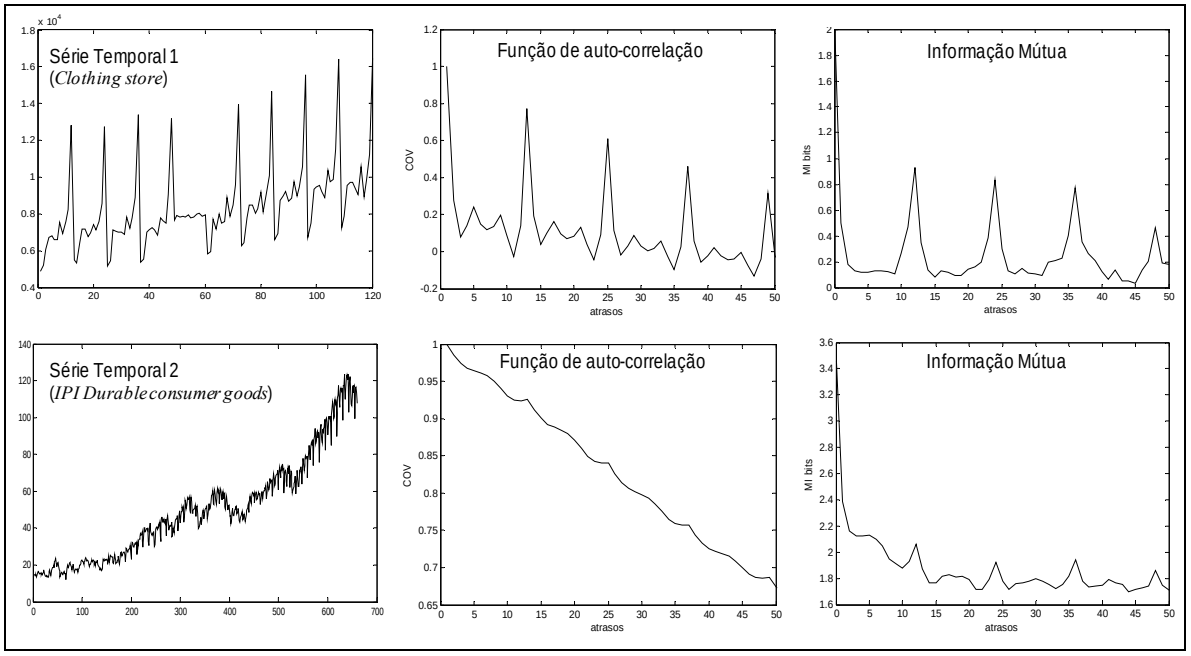


Figura 2.16 – Exemplos do uso de correlação e informação mútua na definição da janela de atrasos em séries temporais.

2.6 Considerações finais

Cada tema abordado neste capítulo será novamente tratado nos próximos capítulos, mas sobre um ponto de vista mais aplicativo. A seguir, uma breve descrição da participação, no decorrer desta tese, de cada um destes temas:

- As redes MLP, com uma e duas camadas ocultas, fazem parte dos estudos comparativos apresentados nos Capítulos 4 e 5 envolvendo problemas de regressão e classificação de dados. O ajuste dos pesos sinápticos das redes MLP é realizado utilizando um método quasi-Newton (o qual é descrito no Capítulo 4).
- Modelos de mistura de especialistas heterogêneos, com treinamento desacoplado via o algoritmo EM, fazem parte também dos estudos comparativos. Aqui, tanto as redes especialistas como a rede *gating* são representadas por redes MLP, com uma camada oculta (modelo não-linear) e redes sem camadas ocultas (modelos lineares). O ajuste dos pesos sinápticos dos especialistas e da rede *gating* é realizado com o mesmo algoritmo quasi-Newton.
- O algoritmo construtivo *Cascade-Correlation* foi implementado segundo FAHLMAN & LEBIERE (1990) e compõe também o elenco de modelos dos estudos comparativos.
- Em relação aos algoritmos de computação evolutiva (algoritmos genéticos GA e programação evolutiva EP): A primeira proposta desta tese para sintetizar redes do tipo ACFNN é baseada em GA (ACFNN-GA) com busca local (ajuste dos pesos sinápticos das redes ACFNNs via o algoritmo quasi-Newton). Na literatura, existe um algoritmo baseado em EP chamado EPNet (YAO & LIU, 1997), e que também sintetiza redes do tipo ACFNN. No Capítulo 3, são descritos ambos os algoritmos evolutivos e comparados seus desempenhos em problemas de classificação de dados.
- Os conceitos de informação mútua (derivados da Teoria de Informação) são aplicados como parte da segunda proposta desta tese: o algoritmo construtivo CoACFNNA para a síntese de redes do tipo ACFNN. A descrição mais detalhada desta proposta, onde os pesos das redes são ajustados também pelo algoritmo quasi-Newton, é apresentada no Capítulo 4.

Capítulo 3

Algoritmos evolutivos para a síntese de redes neurais com conexões à frente arbitrárias

Resumo: O objetivo deste capítulo é a descrição de duas metodologias baseadas em computação evolutiva para a síntese de redes neurais com conexões diretas e arbitrárias (ACFNNs, do inglês *Arbitrarily Connected Feedforward Neural Networks*). A primeira delas é baseada em um algoritmo genético (GA, do inglês *Genetic Algorithm*), sendo proposta pelo autor desta tese, e a segunda está baseada em programação evolutiva (EP, do inglês *Evolutionary Programming*), sendo conhecida na literatura como EPNet (*EP Network*) e proposta por YAO & LIU (1997). Visto que os conceitos básicos dos algoritmos GA e EP foram apresentados no Capítulo 2, os esforços neste capítulo estão concentrados na aplicação e na análise desses algoritmos para o problema de síntese de ACFNNs.

3.1 Considerações acerca da evolução de redes neurais artificiais

A ideia de evoluir redes neurais artificiais surgiu de forma natural ao constatar a capacidade de busca dos algoritmos de computação evolutiva na resolução de problemas de otimização, tanto em espaços discretos como contínuos. Em consequência, é possível conceber a evolução de: (*i*) unicamente a arquitetura da rede utilizando codificação binária, por exemplo; (*ii*) evoluir somente os pesos sinápticos, dada uma arquitetura, utilizando

propostas de algoritmos evolutivos com codificação real, por exemplo; e (*iii*) evoluir simultaneamente arquitetura e pesos sinápticos. Outros parâmetros vinculados ao projeto de uma rede neural, como aqueles associados às funções de ativação, também podem ser alvo de evolução.

As principais motivações para a evolução de redes neurais artificiais são as seguintes:

- Em lugar de ficar a cargo do usuário, promove-se a definição automática da arquitetura de rede neural que tende a ser a mais adequada ao problema, e que inclui o número de camadas ocultas e de neurônios em cada uma delas, os tipos de funções de ativação dos neurônios e a conectividade entre os neurônios.
- O ajuste dos pesos sinápticos pode ganhar maior robustez ao utilizar técnicas que exploram melhor espaços de busca multimodais, evitando a convergência prematura do processo de otimização para mínimos locais de baixa qualidade.

Na literatura, é possível identificar duas formas de codificação da arquitetura e/ou pesos sinápticos de uma RNA: *codificação direta* e *codificação indireta*. Alguns autores definem de forma particular cada uma delas. Por exemplo, para YAO & LIU (1997) a *codificação direta* é aquela em que a maior quantidade de informação (conexão, neurônio, peso sináptico, etc.) é codificada e representada no vetor de atributos (ou cromossomo). Na *codificação indireta*, somente os parâmetros mais importantes da arquitetura, como número de camadas ocultas e o número de neurônios em cada camada são codificados, deixando os demais parâmetros pré-definidos ou a serem especificados durante o processo de treinamento. Por outra parte, para KORDÍK *et al.* (2010) a *codificação direta* envolve a codificação dos pesos sinápticos, os quais, por consequência, já definem a topologia da rede neural, e a *codificação indireta* (de desenvolvimento ou generativa) requer que um processamento adicional se responsabilize por decisões de projeto que não ficam especificadas no cromossomo. O objeto da evolução, neste caso, está nesta etapa de processamento adicional.

Com o propósito de otimizar simultaneamente a arquitetura e os pesos sinápticos a partir de processos evolutivos, é possível citar os seguintes trabalhos pioneiros: ANGELINE *et al.*

(1994), FOGEL (1995), MANIEZZO (1994), WHITLEY *et al.* (1990) e YAO & LIU (1997). Como trabalhos recentes que evoluem apenas a arquitetura e empregam outras técnicas de otimização para ajustar os pesos sinápticos, tem-se o trabalho de PUMA-VILLANUEVA & VON ZUBEN (2010), no qual os pesos sinápticos são ajustados usando um algoritmo de otimização analítico baseado no cálculo do gradiente específico para arquiteturas de ACFNNs, conforme formalizado em WILAMOWSKI *et al.* (2008). Esta abordagem é geralmente interpretada como um algoritmo evolutivo com operador de busca local para o projeto de redes neurais artificiais (WHITLEY *et al.*, 1990; YAO & SHI, 1995).

Existem vantagens e desvantagens associadas às duas propostas, sendo que uma delas promove a evolução de arquitetura e pesos e a outra a evolução somente da arquitetura, com os pesos ajustados via algoritmos de otimização não-linear. YAO & LIU (1997) utilizaram programação evolutiva para evoluir a arquitetura e um algoritmo de *backpropagation* modificado junto com *simulated annealing* (KIRKPATRICK *et al.*, 1983) para o ajuste dos pesos. YAO & LIU (1997) argumentam que o uso de otimização não-linear para o ajuste dos pesos introduz duas fontes de ruído no processo evolutivo: a primeira está associada à inicialização aleatória dos pesos sinápticos, permitindo que uma mesma arquitetura de rede possa apresentar diferentes graus de aptidão (*fitness*). A segunda fonte de ruído refere-se ao algoritmo de ajuste dos pesos sinápticos, pois diferentes algoritmos podem produzir diferentes resultados. Por exemplo, métodos populacionais como *programação evolutiva* (EP) ou *estratégias evolutivas* (ES) podem produzir erros de treinamento menores do que o *gradiente simples*, devido à capacidade de busca global.

Em adição a estas desvantagens relacionadas à evolução de redes neurais, existe outro fator limitador associado, o qual é conhecido como o *problema da permutação* (BELEW *et al.*, 1991; HANCOCK, 1992), ou problema de convenção da competição (SCHAFFER *et al.*, 1992). Este problema pode ser descrito de diferentes formas:

- Em termos evolutivos, diferentes genótipos podem apresentar um mesmo grau de aptidão (ou graus de aptidão muito parecidos) associados a seus correspondentes fenótipos;

- Em termos práticos, significa que diferentes arquiteturas de redes neurais podem ter um mesmo erro de treinamento (aptidão ou *fitness*). Essas arquiteturas podem ter topologias idênticas ou até topologias distintas. No caso de topologias idênticas, neurônios que exercem papel equivalente dentro da arquitetura da rede podem trocar de posição sem alterar o comportamento de entrada-saída da rede, razão pela qual se fala em problema da permutação. Essas questões podem se tornar ainda mais acentuadas quando se consideram arquiteturas mais flexíveis, como aquelas que serão tratadas neste estudo, envolvendo conexões à frente arbitrárias.

O problema da permutação tende a degradar o processo evolutivo por apresentar múltiplas soluções-candidatas com graus de adaptação similares, o que dificulta a busca e pode comprometer o desempenho do operador de *crossover*. Para atenuar este efeito, algumas propostas adotam procedimentos de busca local visando tornar a busca mais efetiva (PUMA-VILLANUEVA & VON ZUBEN, 2010), e outras recorrem a um elenco de vários operadores de mutação, que permite aos descendentes herdar aspectos comportamentais distintos do seu único progenitor (YAO & LIU, 1997).

Recentemente, KORDÍK *et al.* (2010) propuseram o GAME (*Group of Adaptive Models Evolution*), uma abordagem de meta-aprendizado para otimizar a arquitetura e os pesos sinápticos de uma RNA. A otimização da arquitetura das redes é guiada por um algoritmo genético com nichos ecológicos (*niching* GA) e o ajuste dos pesos sinápticos de cada neurônio é realizado de forma independente utilizando vários métodos (18 ao total, entre analíticos e aproximados): 2 analíticos, quasi-Newton e gradiente conjugado; e, dentre os aproximados, 4 baseados em computação evolutiva, 2 em enxame de partículas, 5 em colônia de formigas, 1 híbrido (PSO+GA) e 4 baseados em busca estocástica ortogonal. Cada nicho tende a gerar descendentes com aptidão superior à de seus progenitores, sendo que esta abordagem pode ser interpretada como sendo multipopulacional.

3.2 Evoluindo ACFNNs via Algoritmos Genéticos

Esta seção concentra-se na descrição da implementação de uma versão de GA para a evolução de redes neurais com conexões diretas e arbitrárias (ACFNNs, do inglês *arbitrarily connected feedforward neural networks*), tanto para problemas de regressão como de classificação de padrões. A Seção 2.4.1, Capítulo 2 desta tese, traz os conceitos básicos de um algoritmo genético.

Esta proposta será denominada ACFNN-GA, e o conteúdo desta seção se baseia na primeira proposta de formalização feita em PUMA-VILLANUEVA & VON ZUBEN (2010), a qual foi aplicada junto a problemas de regressão de dados, mais especificamente predição de séries temporais. As diferenças que serão apresentadas aqui se concentram nos seguintes pontos:

- Treinamento das redes candidatas utilizando um algoritmo de ajuste dos pesos sinápticos baseado num método quasi-Newton com escalonamento automático, que substitui com vantagens o algoritmo de gradiente simples empregado em PUMA-VILLANUEVA & VON ZUBEN (2010);
- Aplicação junto a problemas de classificação de padrões.

Para a descrição da nova versão do ACFNN-GA, foi adotada a seguinte divisão de tópicos: codificação do problema, descrição dos operadores genéticos (seleção, *crossover*, mutação e busca local) e, finalmente, a aplicação junto a dois problemas de classificação de diferentes graus de complexidade.

3.2.1 Codificação do problema

Para a codificação do problema, o usuário precisa definir o *número máximo de neurônios ocultos* das redes a evoluir. O número de neurônios das camadas de entrada e de saída são previamente estabelecidos em função das dimensões dos próprios dados de treinamento (número de variáveis de entrada e de saída do problema). Em consequência,

define-se o *número máximo de neurônios* de todas as redes neurais candidatas (*NoMaxNeuronios*) como:

$$NoMaxNeuronios = \begin{array}{c}\text{número de}\\ \text{neurônios da}\\ \text{camada de entrada}\end{array} + \begin{array}{c}\textit{número máximo}\\ \textit{de neurônios}\\ \textit{ocultos}\end{array} + \begin{array}{c}\text{número de}\\ \text{neurônios da}\\ \text{camada de saída}\end{array}$$

A Figura 3.1 ajuda a mostrar a forma como uma arquitetura ACFNN é representada num cromossomo com valores binários. O vetor binário que representa o cromossomo está composto pelos valores do triângulo superior da matriz, que tem *NoMaxNeuronios* como número de linhas e colunas. Os valores 0 e 1, nesta parte da matriz, significam a ausência e a presença de conexão entre os neurônios da linha e da coluna, respectivamente. Ainda na Figura 3.1, mostra-se um exemplo didático para a codificação da rede neural na parte da esquerda, composta por 7 neurônios no total, sendo 3 na camada de entrada, 1 na camada de saída e 3 neurônios ocultos distribuídos em duas camadas. À direita da Figura 3.1, a matriz com os elementos 0 e 1 da triangular superior e, abaixo desta, o vetor cromossomo como resultado do empilhamento dos valores da triangular superior da matriz numa ordem determinada pela sequência de linhas. Desta forma, uma vez definido o valor de *NoMaxNeuronios*, o tamanho do vetor binário que representa o cromossomo é definido automaticamente e fixado com o seguinte valor:

$NoMaxNeuronios \times (NoMaxNeuronios - 1) \div 2$. No exemplo, o tamanho é de 21, pois *NoMaxNeuronios* foi tomado como sendo 7.

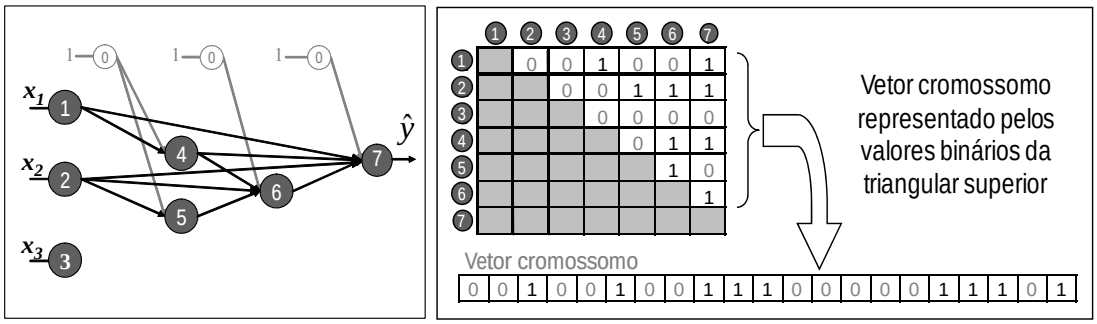


Figura 3.1 – Codificação do vetor binário que representa o cromossomo para uma arquitetura de rede ACFNN.

3.2.2 Inicialização da população e mecanismo de factibilização

A população é inicializada de forma aleatória com valores binários 0 e 1 preenchendo arbitrariamente os elementos da parte triangular superior de cada matriz associada a uma solução-candidata. O tamanho da população é definido pelo usuário.

A factibilização dos indivíduos é realizada seguindo a sequência de procedimentos indicados na Tabela 3.1. No lado esquerdo da Tabela 3.1, indicam-se os casos de infactibilidade e, na direita, o tratamento a ser adotado para se obter uma rede neural factível.

Tabela 3.1 – Casos e procedimentos de recuperação de factibilidade.

Neurônios ocultos sem conexões entrantes, mas com conexões de saída	Conectam-se de forma aleatória com os neurônios da camada de entrada da rede.
Neurônios ocultos com conexões de entrada, mas sem conexões de saída	Conectam-se aleatoriamente com os neurônios da camada de saída da rede.
Conexões entre neurônios da mesma camada	Eliminam-se essas conexões.
Neurônios de saída sem conexões entrantes	Conectam-se de forma aleatória com os neurônios da camada de entrada da rede.
Todos os neurônios de entrada sem conexões	Escolhe-se um grupo aleatório deles e criam-se conexões aleatórias entre estes e os neurônios da camada de saída da rede.

Note que, pela forma da codificação do problema, é possível casos em que os neurônios ocultos estejam sem conexões de entrada e sem conexões de saída, o qual não significa um caso de infactibilidade e sim um caso em que o modelo é puramente linear, ao conectar diretamente os neurônios de entrada com os de saída, sem utilizar neurônios ocultos.

3.2.3 A função de aptidão (*fitness*)

A função de aptidão requer previamente o processo de ajuste dos pesos sinápticos das redes codificadas nos cromossomos que compõem a população de indivíduos. Nesta versão da metodologia, é empregado o método quasi-Newton com escalonamento automático para o ajuste dos pesos das redes do tipo ACFNN. O cálculo do vetor gradiente para este tipo de rede neural e o método quasi-Newton propriamente dito são formalmente descritos no início do Capítulo 4.

O método de retenção (forma mais simples de validação cruzada *holdout*) foi empregado de forma a utilizar o subconjunto de *treinamento* para o cálculo do gradiente e ajuste dos pesos sinápticos e o subconjunto de *validação* para evitar o sobre-ajuste do treinamento e para o cálculo da função de *fitness* propriamente dita. A Equação 3.1 define a expressão para o cálculo da função de *fitness*:

$$fitness = \frac{1}{1 + MSE_{validação}}, \tag{3.1}$$

onde o $MSE_{validação}$ refere-se ao erro quadrático médio usando o subconjunto de validação.

3.2.4 Operadores do GA

Operador de seleção:

O usuário define o número de P progenitores que serão selecionados da população, seguindo a estratégia da roleta (BÄCK *et al*., 1997).

Operador de cruzamento (*crossover*):

Um par de novos indivíduos é produzido por cada par de progenitores (considerando todas as possibilidades de pares dos P progenitores) via cruzamento de k pontos, onde k é escolhido aleatoriamente de uma distribuição uniforme na faixa de [1, 2, 3]. Em consequência, os P progenitores produzem $P\times(P-1)$ novos indivíduos (descendentes). A

seguir, calcula-se a aptidão de cada um dos indivíduos descendentes, e os piores *P* indivíduos da população são substituídos pelos melhores *P* descendentes.

Operador de mutação:

A decisão de aplicar mutação é realizada obedecendo a uma *taxa de mutação* definida pelo usuário. Neste trabalho, apenas o melhor indivíduo da população é preservado da mutação, mas uma cópia dele é mutada e substitui o pior indivíduo da população. A taxa de mutação não se mantém fixa durante todas as gerações. Ela vai decaindo gradativamente a cada geração, começando de 0,85, e programada para decair linearmente até um valor limitante inferior de 0,17, ao se atingir um número máximo de gerações. A razão desta forma de programação da taxa de mutação é a de equilibrar o compromisso de exploração e explotação na busca pela melhor arquitetura de rede.

Foram considerados 4 tipos de mutação. Somente um deles é utilizado a cada ocorrência de mutação, com probabilidade uniforme (0,25 para cada um deles). A seguir, a descrição dos 4 tipos de mutação adotados:

Tipo 1: A forma tradicional de mutação para codificação binária. Escolhe-se de forma aleatória um número de genes e trocam-se seus valores, 0 por 1 e 1 por 0.

Tipo 2: Eliminação de um neurônio oculto. Se existirem neurônios ocultos na rede, escolhe-se de forma aleatória um deles e suas conexões entrantes são ligadas de forma direta com as conexões que este mesmo neurônio estabelece com os neurônios de saída.

Tipo 3: Adição de um neurônio oculto. Se existirem neurônios ocultos (pelo menos 1) dentro da codificação do indivíduo e que não estejam sendo utilizados na rede, então escolhe-se de forma aleatória um destes neurônios para formar parte da rede em uma das seguintes formas: (1) fazendo parte de alguma das camadas ocultas já existentes, ou (2) criando uma nova camada oculta. A decisão por uma destas formas é aleatória e com uma mesma probabilidade de escolha (0,5). Para estabelecer as conexões do novo neurônio na rede, copiam-se as conexões de entrada e saída do neurônio escolhido. Caso seja o primeiro neurônio oculto, conecta-se com todas as entradas e saídas da rede.

Tipo 4: Eliminação de uma camada oculta. Elimina-se a camada oculta com o menor número de neurônios.

Operador de busca local:

Neste trabalho, a busca local foi aplicada apenas em uma parcela da população (na quarta parte e escolhidos aleatoriamente) e a cada 10 gerações. São executadas três formas de busca local de maneira consecutiva e considerando a estratégia de melhor melhora (do inglês *best improvement*).

1. **Eliminação de um neurônio oculto**. Procedimento que busca a eliminação do neurônio oculto que, na sua ausência, incrementa mais o valor da função de *fitness*. As conexões de entrada e saída do neurônio são ligadas diretamente e a rede é re-treinada a partir de novos pesos sinápticos.
2. **Eliminação de conexão**. De forma análoga ao procedimento anterior, este operador busca eliminar a conexão que, na sua ausência, incrementa mais o valor da função de *fitness*. O re-treinamento da rede é feito tendo como ponto de partida os pesos ajustados previamente. A eliminação de uma conexão pode levar a situações de infactibilidade, as quais devem ser evitadas.
3. **Adição de nova conexão.** Neste caso, a busca é pela adição da conexão na rede que incremente mais a função de *fitness*. O re-treinamento da rede é feito partindo dos pesos com seus valores previamente ajustados.

3.2.5 Aplicação do ACFNN-GA

Foram considerados dois problemas de classificação de padrões comumente tratados na literatura (*benchmarks*), os quais fazem parte de um número maior de problemas utilizados no Capítulo 5, que trata exclusivamente de simulações de vários modelos de classificadores baseados em redes neurais, incluindo a proposta principal desta tese, a ser apresentada no Capítulo 4. Vale indicar que, nesta seção, pretende-se analisar o funcionamento do ACFNN-GA para, ao final deste capítulo, compará-lo com a outra metodologia baseada,

também, em computação evolutiva (EPNet). Cabe destacar também que o número de saídas coincide com o número de classes, sendo que a classe será indicada pela saída de maior valor, dentre todas as saídas da rede neural.

Os problemas são: *Wine* e *Two Spirals*.

- *Wine*: Este problema foi escolhido por requerer poucos neurônios ocultos, revelando um baixo grau de não-linearidade para a classificação correta de suas amostras. Por tal motivo, implica um desafio para o ACFNN-GA convergir para redes parcimoniosas e, ao mesmo tempo, com bom desempenho. O problema é real e pertence à indústria de vinhos: ele conta com 178 amostras, 13 variáveis de entrada e 3 classes. Consequentemente, a camada de entrada estará constituída por 13 neurônios e a camada de saída por 3 neurônios.
- *Two Spirals*: Problema que, ao contrário do *Wine*, requer alta não-linearidade para a classificação correta de suas amostras. Isto significa que as redes devem possuir um maior número de neurônios na sua camada oculta. Ele é de natureza artificial e conta com 944 amostras, 2 variáveis de entrada e 2 classes.
- As simulações computacionais foram realizadas num notebook com CPU Intel CORE i7 Q740 de 1,73Ghz e 4Gb de memória RAM, e o algoritmo foi implementado em Matlab.

A Tabela 3.2 informa os principais parâmetros utilizados para a resolução de ambos os problemas via o ACFNN-GA:

Tabela 3.2 – Definição dos principais parâmetros para ambos os problemas via o ACFNN-GA.

Número máximo de neurônios ocultos	10 neurônios para o problema *Wine* e 15 neurônios para o problema *Two Spirals*. Consequentemente, o tamanho do vetor binário que representa o cromossomo para cada problema é de 325 e 171, respectivamente.
Tamanho da população	20 indivíduos.
Número de progenitores P	4
Número máximo de gerações	200
Número de épocas de treinamento dos pesos da rede via método quasi-Newton	500

As Figuras 3.2 e 3.3 mostram os resultados da evolução para ambos os problemas. Cada uma das figuras mostra a seguinte informação:

- Na parte superior, a arquitetura de rede tipo ACFNN resultante após completar o processo evolutivo e, ao lado, suas correspondentes taxas de classificação correta para os subconjuntos de treinamento, validação e teste.
- Na parte intermediária, o gráfico do lado esquerdo mostra a evolução do *fitness* do melhor indivíduo assim como o do *fitness* médio da população a cada geração. E o gráfico do lado direito mostra o resultado da aplicação do operador de *crossover* num gráfico de duas barras. A primeira barra contabiliza as vezes em que o descendente teve menor aptidão que um dos seus dois progenitores e a segunda barra contabiliza as vezes em que o descendente apresentou aptidão igual ou superior a um dos seus dois progenitores.
- Na parte inferior, outros dois gráficos: o da esquerda que busca mostrar as estatísticas do número de neurônios que foram utilizados pelos indivíduos (redes) da população ao decorrer das gerações. Nele, de cor cinza mais claro, apresenta-se o número de neurônios mínimo e máximo da população, enfatizando sua concentração (usou-se o comando *boxplot* do Matlab). Em seguida, uma curva de

cor cinza mais escuro mostra o número de neurônios médio da população e a curva preta indica o número de neurônios ocultos do melhor indivíduo da população a cada geração. O gráfico do lado direito busca informar o momento em que o melhor indivíduo da população alcança o desempenho desejado. Desempenho, em termos de taxas de classificação correta (subconjuntos de treinamento, validação e teste) ao decorrer de cada uma das gerações do processo.

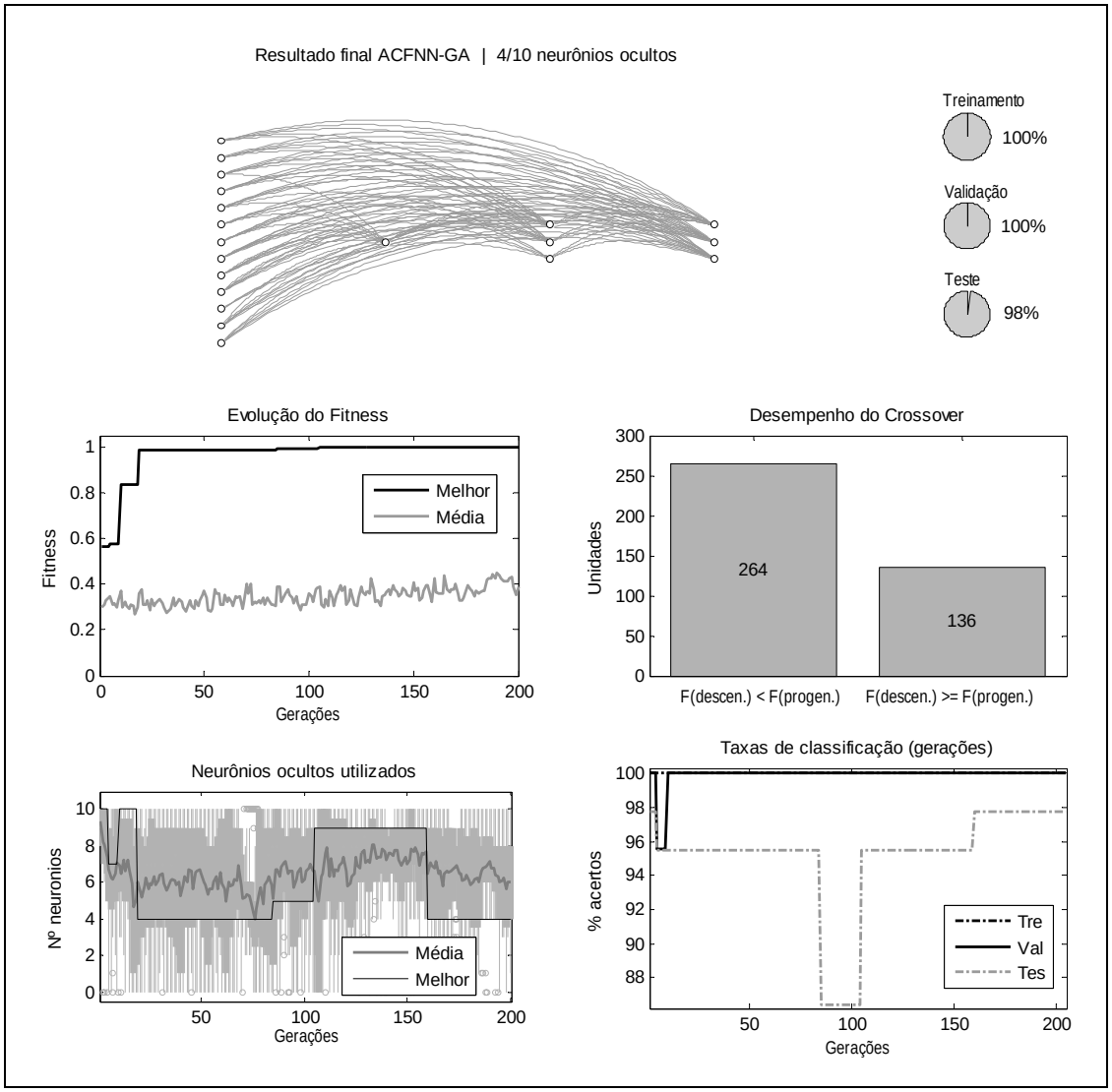


Figura 3.2 – Resultados produzidos pelo ACFNN-GA para o problema *Wine*.

Analisando os resultados da Figura 3.2 para o problema *Wine*, é possível constatar que a melhor rede resultante conseguiu o desempenho desejado junto ao subconjunto de validação, alcançando 100% de classificação correta. Já para o subconjunto de teste, a taxa de classificação correta foi de 98%. E esse desempenho foi alcançado utilizando apenas 4 dos 10 neurônios ocultos disponíveis. No entanto, o gráfico do desempenho do *crossover* mostra a deficiência para a geração de descendentes com aptidão superior a dos progenitores, o número de vezes em que os descendentes foram piores que os progenitores é de aproximadamente o dobro do que o número de vezes em que eles foram melhores. A consequência disto é constatada no gráfico que informa o *fitness* médio da população, o qual está bem abaixo do *fitness* do melhor indivíduo. Finalmente, na parte inferior da figura, o gráfico da esquerda mostra que o número médio de neurônios ocultos das redes das populações, nas últimas gerações, convergiu para o valor em torno de 6 neurônios ocultos. É possível verificar também que o número de neurônios ocultos para o melhor indivíduo a cada geração, nas primeiras gerações, alcançou o valor máximo de 10, em seguida caiu para 4, subiu para 9 e, nas últimas gerações, estabeleceu-se em 4 neurônios. Já no gráfico da direita, que mostra as taxas de desempenho de classificação do melhor indivíduo a cada geração, percebe-se já na primeira geração o desempenho de 100% e com o máximo de neurônios ocultos. Mas, no decorrer da evolução, o desempenho manteve-se no máximo e o número de neurônios ocultos caiu até finalmente 4 neurônios, o que indica que os operadores de busca local e, principalmente, de mutação conseguiram aumentar a parcimônia das redes em evolução. O tempo gasto na execução do ACFNN-GA para este problema foi de 53,78 minutos.

A Figura 3.3 mostra os resultados para análise correspondente ao segundo problema, o *Two Spirals*. Na parte superior da figura, mostra-se a rede ACFNN resultante, a qual possui 12 dos 15 neurônios ocultos disponíveis e o desempenho em termos de porcentagem de classificação correta foi a máxima junto aos três subconjuntos de dados. Na parte do meio da figura, o gráfico da direita mostra o desempenho do *crossover*, indicando de modo similar ao que havia sido verificado no estudo de caso anterior que a geração de descendentes menos aptos que os progenitores foi maior. Já o gráfico da esquerda, com a evolução do *fitness*, mostra que o *fitness* médio da população teve um incremento no

decorrer das gerações e que o melhor indivíduo já conseguia o maior valor de *fitness* desde a primeira geração, embora utilizando o máximo de neurônios ocultos disponível (15 neurônios), como aponta o gráfico esquerdo da parte inferior da figura (neurônios ocultos utilizados pela população a cada geração), onde é possível constatar que o melhor indivíduo utilizou 15 neurônios nas primeiras gerações e, em seguida, foi diminuindo até convergir em 12 neurônios, o que coincide com o valor médio da população. Já o gráfico da direita, que mostra as taxas de classificação correta pelo melhor indivíduo a cada geração, informa o já revelado no gráfico da evolução do *fitness*, que em todas as gerações o melhor indivíduo conseguia resolver o problema sem nenhum erro de classificação. O processo evolutivo permitiu, ao longo das gerações, encontrar a ACFNN mais parcimoniosa, a qual teve finalmente 12 neurônios ocultos. O tempo computacional utilizado para a resolução deste problema foi de 73,85 minutos.

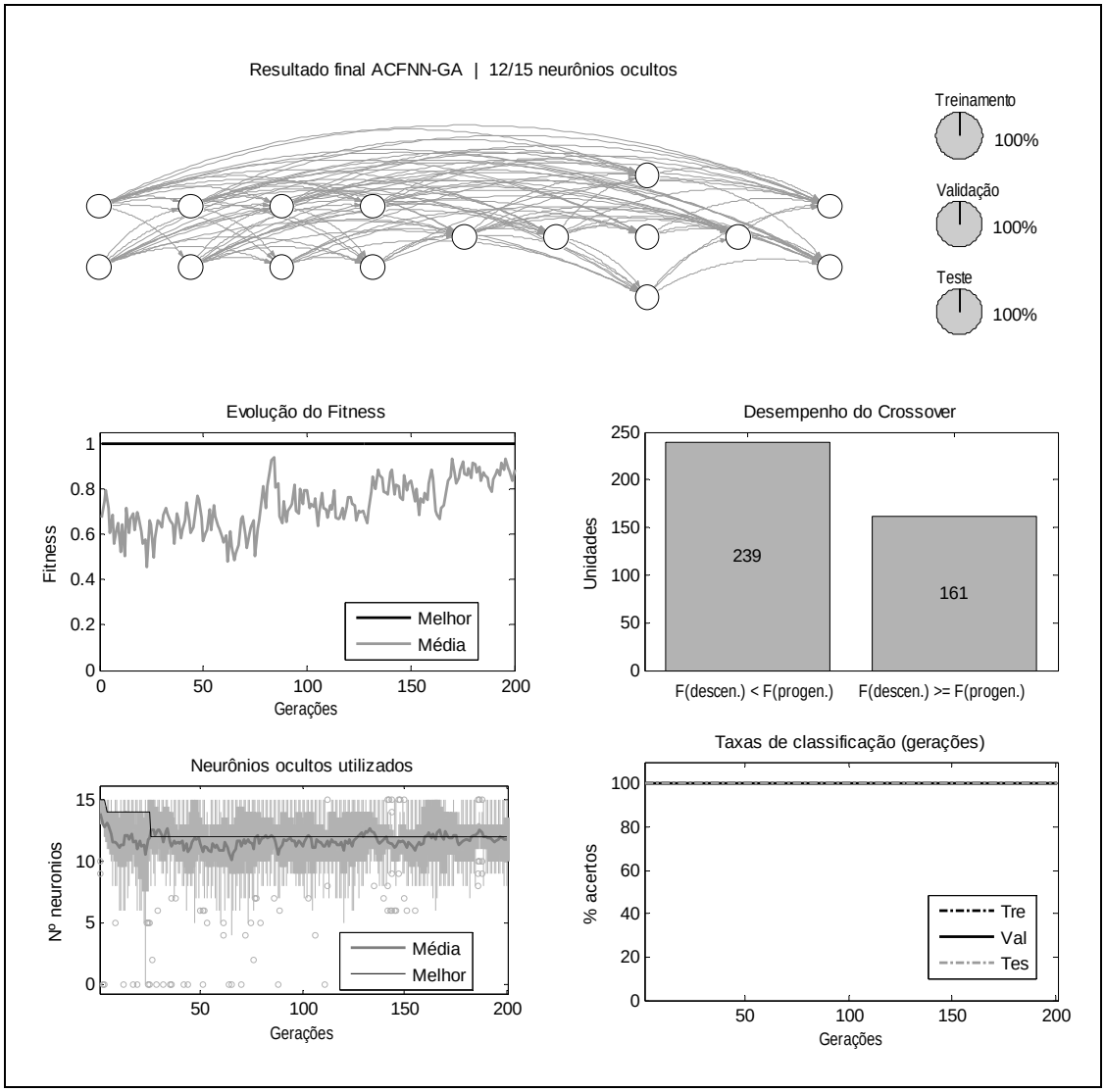


Figura 3.3 – Resultados produzidos pelo ACFNN-GA para o problema *Two Spirals.*

Na próxima seção, descreve-se a proposta de YAO & LIU (1997), a qual está baseada em programação evolutiva e será utilizada para efeitos de comparação com o ACFNN-GA.

3.3 Evoluindo ACFNNs via Programação Evolutiva

O EPNet foi proposto por YAO & LIU (1997) e pode ser interpretado, de acordo com uma sugestão de seus próprios criadores, como um sistema híbrido que combina três estratégias bem conhecidas de aprendizado de máquina:

1. Programação Evolutiva (EP) (FOGEL, 1995): Implementado de forma a otimizar a arquitetura ACFNN. Cada indivíduo da população representa uma rede candidata. Visto que a EP não trabalha com o operador de recombinação (*crossover*), então é possível ter uma população codificada com cromossomos (indivíduos) de tamanhos distintos, o que permite uma maior exploração do espaço de busca. Os autores colocaram maior atenção no operador de mutação, o qual contém até 6 formas distintas de mutação no intuito de melhorar o desempenho e parcimônia das redes resultantes.
2. *Backpropagation* Modificado (MBP): Utilizado para o ajuste dos pesos sinápticos das ACFNNs que estão sendo evoluídas. Este método pode ser interpretado como uma forma local e incremental de ajuste dos pesos sinápticos.
3. Recozimento Simulado (SA - *Simulated Annealing*): Empregado também para o ajuste dos pesos sinápticos, caso o MBP não consiga minimizar o erro de treinamento. A atuação do SA pode ser vista como uma forma de escapar de mínimos locais alcançados pelo MBP e permitir uma exploração mais abrangente do espaço de busca dos pesos da rede.

Nesta metodologia, o MBP e o SA são considerados como duas formas de mutação guiada que atuam diretamente nos pesos sinápticos da rede. As outras 4 formas de mutação atuam na arquitetura da rede propriamente dita: eliminação de neurônios, eliminação de conexões, adição de conexões e adição de neurônios. Estas 6 mutações são aplicadas nessa mesma ordem. A Figura 3.4 ilustra a sequência de passos que compõem o EPNet.

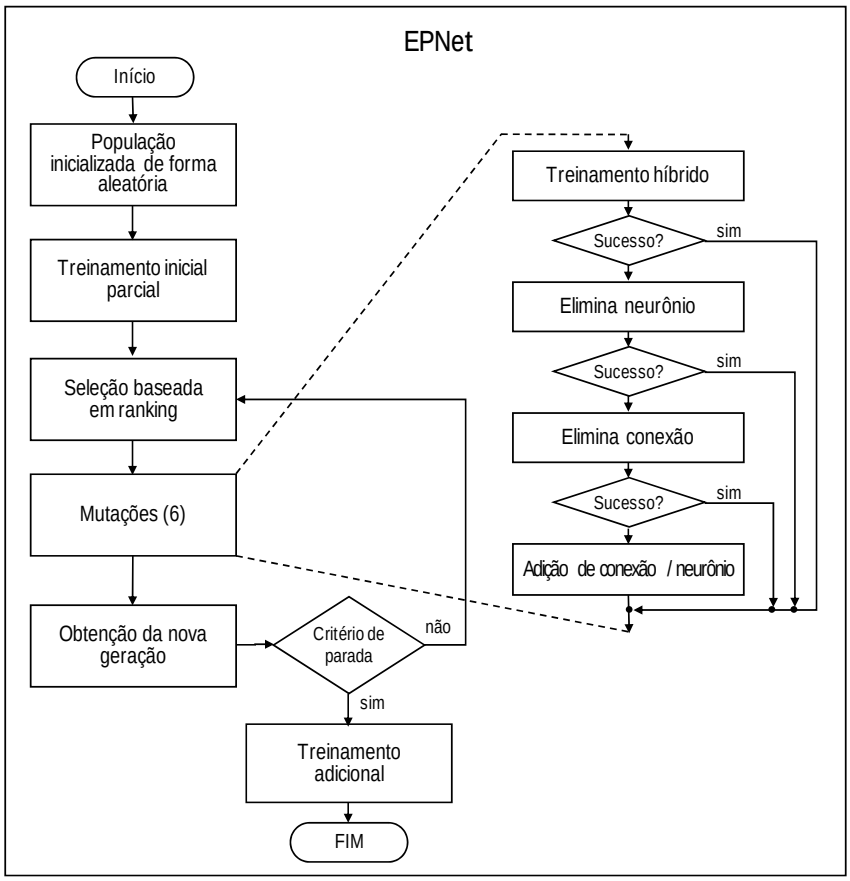


Figura 3.4 – Fluxograma dos passos da metodologia EPNet (YAO & LIU, 1997).

Passo 1 – Codificação e inicialização da população: A codificação é binária, do tipo indireta e similar à codificação utilizada pelo ACFNN-GA. A população é de *M* indivíduos e inicializada de forma aleatória (sendo este valor *M* definido pelo usuário). Nesta inicialização, deve-se levar em conta que o número de neurônios é gerado uniformemente dentro de uma faixa de valores (de *minN* até *maxN*) definidos, também, pelo usuário. As conexões também são inicializadas aleatoriamente. Da mesma forma, os pesos iniciais das conexões das redes são gerados aleatoriamente, dentro de uma faixa pequena de valores.

Passo 2 – Treinamento parcial: Treinam-se parcialmente as redes da população usando o MBP com taxa de aprendizado adaptativo. O número de épocas K_0 é especificado pelo usuário. O valor do erro *E* é calculado usando o subconjunto de validação para cada rede. Se *E* não foi reduzido de forma significativa, então se supõe que o treinamento tenha

convergido para um mínimo local e a rede é marcada como "mal-sucedida". Caso contrário, é considerada como "bem-sucedida". O erro E foi originalmente proposto por PRECHELT (1994) e foi utilizado como uma forma de função de *fitness*, sendo que quanto menor o seu valor melhor aptidão possui o indivíduo. A equação (3.2) indica a forma como é calculado E:

$$E = 100 \times \frac{o_{\max} - o_{\min}}{N \times ns} \sum_{p=1}^{N} \sum_{i=1}^{ns} \left(\hat{Y}_i^{\,p} - Y_i^{\,p} \right)^2 \tag{3.2}$$

Este calculo para E faz com que o erro seja menos dependente do número de amostras N (do subconjunto de validação) e do número de saídas da rede (ns). $o_{\max}$ e $o_{\min}$ representam os valores máximo e mínimo de excursão das saídas da rede, respectivamente.

Passo 3 – Ranking da população: Ordene a população de acordo com o erro E, do melhor para o pior indivíduo.

Passo 4 – Critério de parada: Se a melhor rede evoluída tem desempenho aceitável ou se o número máximo de gerações for atingido (critério de parada), então termine o processo evolutivo e vá para o Passo 11. Caso contrário, continue.

Passo 5 – Seleção do progenitor: Use a *seleção baseada em ranking* para escolher um indivíduo (rede) da população para fazer o papel de progenitor. Se este estiver marcado como "bem-sucedido", vá para o Passo 6. Caso contrário, vá para o Passo 7.

Passo 6 – MBP (Mutação #1): Treine parcialmente o indivíduo progenitor por K_1 épocas usando o MBP para obter o indivíduo descendente e marque este da mesma forma como no Passo 2. K_1 é um parâmetro definido pelo usuário. Substitua o indivíduo progenitor pelo descendente na população atual e vá para o Passo 3.

Passo 7 – AS (Mutação #2): Treine o indivíduo progenitor com o algoritmo SA para obter o indivíduo descendente. Se o SA (após K_3 iterações) conseguir reduzir o erro E de forma significativa, então marque o descendente como "bem-sucedido" e substitua o progenitor

pelo descendente na população atual e vá para o Passo 3. De outra forma, descarte o indivíduo descendente e vá para o Passo 8.

Passo 8 - Eliminação de neurônios (Mutação #3): Primeiro, decida o número de neurônios ocultos (N_{ocultos}) para eliminar, de forma aleatória, dentro de uma faixa de valores definidos pelo usuário. N_{ocultos} é geralmente um valor pequeno, não mais de três na maioria dos casos. Em seguida, escolha aleatoriamente N_{ocultos} neurônios ocultos da rede progenitora para serem eliminados. Treine parcialmente a rede podada via o algoritmo MBP para obter o indivíduo descendente. Se o descendente for melhor do que o pior indivíduo da população, então o substitua e vá para o Passo 3. Caso contrário, descarte o descendente e vá para o Passo 9.

Passo 9 - Eliminação de conexões (Mutação #4): Calcule a importância de cada conexão da rede representada pelo indivíduo progenitor usando o método descrito a seguir. Decida o número de conexões a serem eliminadas da mesma forma como definido no Passo 8. Elimine aleatoriamente as conexões da rede progenitora de acordo com a importância calculada. Em seguida, treine parcialmente via o algoritmo MBP para obter um indivíduo descendente. Se o descendente for melhor do que o pior indivíduo da população atual, então substitua este pelo descendente e vá para o Passo 3. Caso contrário, vá para o Passo 10.

A forma de calcular a importância de cada peso foi originalmente utilizada em FINNOFF *et al.* (1993), sendo definida via um teste de significância para o desvio de zero dos pesos sinápticos. Considere a atualização dos pesos $\Delta w_{i,j}(w) = -\lambda[\partial L_p / \partial w_{i,j}]$ pelo gradiente local da função linear de erro L:

$$L = \sum_{p=1}^{P} \sum_{i=1}^{n} \left| \hat{Y}_i^{\,p} - Y_i^{\,p} \right| . \tag{3.3}$$

Em relação à amostra p e o peso sináptico $w_{i,j}$, a significância do desvio de zero de $w_{i,j}$ é definida pela variável *teste*:

$$teste(w_{i,j}) = \frac{\sum_{p=1}^{P} \xi_{i,j}^{p}}{\sqrt{\sum_{p=1}^{P} (\xi_{i,j}^{p} - \overline{\xi}_{i,j})^2}} \tag{3.4}$$

onde $\xi_{i,j}^{p} = w_{i,j} + \Delta w_{i,j}^{p}(w)$, e $\overline{\xi}_{i,j}^{p}$ representa o valor médio do conjunto $\xi_{i,j}^{p}$, p=1,...,P. Um valor alto para $teste(w_{i,j})$ indica alta importância da conexão com o peso sináptico $w_{i,j}$.

A principal vantagem deste método em relação a outros é que ele não requer a realização do ajuste de pesos até a convergência para testar a conexão, nem requer parâmetros extras. Este método pode ser usado também para conexões com peso zero e, assim, pode-se estender seu uso para testar conexões que poderiam ser adicionadas à rede.

Passo 10 - Adição de conexões e neurônios (Mutações #5 e #6): Decide-se o número de conexões e neurônios a serem adicionados da mesma forma que no Passo 8. Calcule a importância aproximada de cada conexão virtual com peso zero. Aleatoriamente, adicione a conexão na rede progenitora para obter o descendente #1 de acordo com sua importância. A adição de cada neurônio é efetuada via uma cópia de um neurônio escolhido aleatoriamente na rede progenitora. A nova rede incrementada com os novos neurônios constitui o descendente #2. Em seguida, treinam-se parcialmente os dois descendentes via MBP para obter o melhor dos dois descendentes, o qual substituirá o pior indivíduo da população.

Passo 11 – Treinamento adicional: Após a conclusão do processo evolutivo, realiza-se um treinamento adicional, combinando os subconjuntos de *treinamento* e *validação*, até convergir.

3.3.1 Aplicação do EPNet

A aplicação do EPNet na presente seção desta tese será realizada a partir de uma implementação própria e seguindo as informações encontradas no artigo associado a esta metodologia, cujo título é: "*A New Evolutionary System for Evolving Artificial Neural Networks*" (YAO & LIU, 1997). Os problemas de classificação escolhidos foram os mesmos

utilizados para o ACFNN-GA: *Wine* e *Two Spirals*. As simulações computacionais foram realizadas no mesmo computador usado para executar o ACFNN-GA (notebook com CPU Intel CORE i7 Q740 de 1,73Ghz e 4Gb de memória RAM), e o algoritmo foi implementado em Matlab.

Na Tabela 3.3, apresenta-se a definição dos principais parâmetros desta metodologia.

Tabela 3.3 – Definição dos principais parâmetros para ambos os problemas via o EPNet.

Número de neurônios ocultos [$minN$ e $maxN$] iniciais	Para o problema *Wine*: [2 até 5]; Para o problema *Two Spirals*: [5 até 10].
Tamanho da população	20 indivíduos
Número máximo de gerações	500
Número de épocas de treinamento dos pesos da rede via MBP e SA: K_0 (MBP-inicialização), K_1 (MBP-Mutação #1) e K_2 (SA-Mutação #2)	500, 750 e 250 (respectivamente)

Para o caso do EPNet, este requer mais gerações visto que o elenco de mutações é aplicado sobre um único indivíduo a cada geração. As Figuras 3.5 e 3.6 mostram os resultados para a análise do EPNet junto aos dois problemas considerados. A informação destas figuras é similar à das Figuras 3.2 e 3.3 para o ACFNN-GA, tendo uma única diferença na parte intermediária da figura. O gráfico à direita, que mostrava o *desempenho do crossover* (ACFNN-GA), foi substituído por um gráfico de barras que mostra o número de neurônios ocultos (barras de cor preta) e o número de conexões (barras de cor cinza) de cada indivíduo da população na última geração. Vale salientar que as barras estão ordenadas segundo o *fitness* de seus respectivos indivíduos. Assim, a primeira barra corresponde à do melhor indivíduo da população.

Analisando a Figura 3.5 (*Wine*), na parte superior, a rede resultante convergiu para uma configuração que utilizar apenas 3 neurônios e seu desempenho em termos de taxa de classificação correta foi de 100% nos três subconjuntos de dados (treinamento, validação e teste). Na parte intermediária da figura, mostra-se à esquerda o gráfico com a evolução do

erro (que pode ser interpretado como o inverso do valor do *fitness*: quanto menor o valor, melhor a aptidão) para o melhor indivíduo como para o valor médio da população, a cada geração. No gráfico, se observa um decaimento de ambas as curvas de forma gradual, como era de se esperar. Na parte direita, o gráfico com o número de neurônios e conexões da população atual (na última geração) mostra a diversidade da população com diferentes números de neurônios e conexões. Cada indivíduo da população segue um caminho independente guiado pelas múltiplas mutações. Finalmente, na parte inferior da figura, o gráfico da esquerda revela o número de neurônios ocultos da população a cada geração, observa-se que a população se manteve com um número de neurônios dentro da faixa de inicialização (de 2 até 5), embora em algumas gerações apresentou-se casos de indivíduos com até 6 neurônios ocultos, mas o número médio de neurônios convergiu em torno de 3 e 4 neurônios (curva cinza) e o melhor indivíduo (curva preta) começou com 3 neurônios, passou várias gerações com apenas 2, subiu até 4 e, finalmente, terminou com 3 neurônios ocultos. Já no gráfico à direita, mostrando o desempenho em termos de taxas de classificação do melhor indivíduo ao decorrer das gerações, observam-se curvas coerentes, em especial para o conjunto de validação, a qual começou em torno de 95% e foi melhorando gradativamente até 100%. Nesse caminho, as outras curvas experimentaram certas variações. O tempo utilizado por esta versão própria do EPNet para este problema foi de 37,53 minutos.

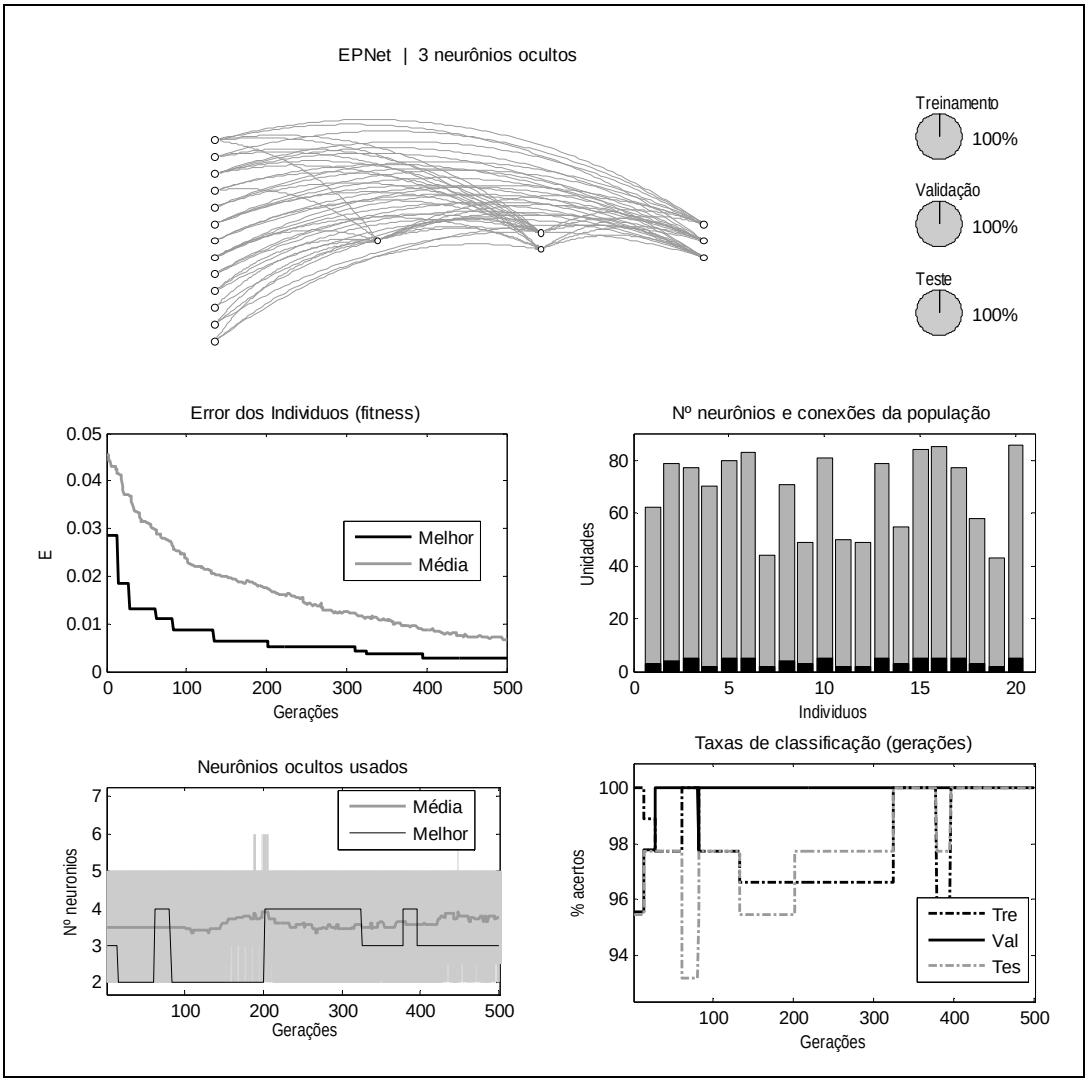


Figura 3.5 – Resultados produzidos pelo EPNet para o problema *Wine*.

A Figura 3.6 mostra os resultados para a análise do outro problema, o *Two Spirals*. Os resultados foram coerentes: trata-se de um problema não-linear e o EPNet conseguiu uma rede com 9 neurônios ocultos e com desempenho de 100% nos três subconjuntos de dados, como mostrado na parte superior da figura. Na parte média da figura, as curvas do erro do melhor indivíduo como do erro médio da população apresentaram o comportamento desejado, o que mostra que a população toda conseguiu evoluir de forma coerente. O gráfico da direita mostra os neurônios e conexões da população da última geração. Os dois gráficos da parte inferior da figura (neurônios ocultos utilizados e taxas de classificação)

revelam o caminho que seguiu o melhor indivíduo. É interessante ver que, ao início, mesmo com 10 neurônios ocultos, o melhor indivíduo não conseguiu bom desempenho em termos de taxas de classificação correta. No entanto, ao longo das gerações, o valor foi na maior parte do processo evolutivo de 6 neurônios e com desempenho em torno de 96%. Ao atingir os 9 neurônios próximo à geração número 400, seu desempenho chegou a 100%. A média do número de neurônios da população a cada geração também aumentou e, inclusive, superou o valor limite superior de inicialização, que foi 10, para 13, nas últimas gerações. O tempo computacional demandado para este problema foi de 91,20 minutos.

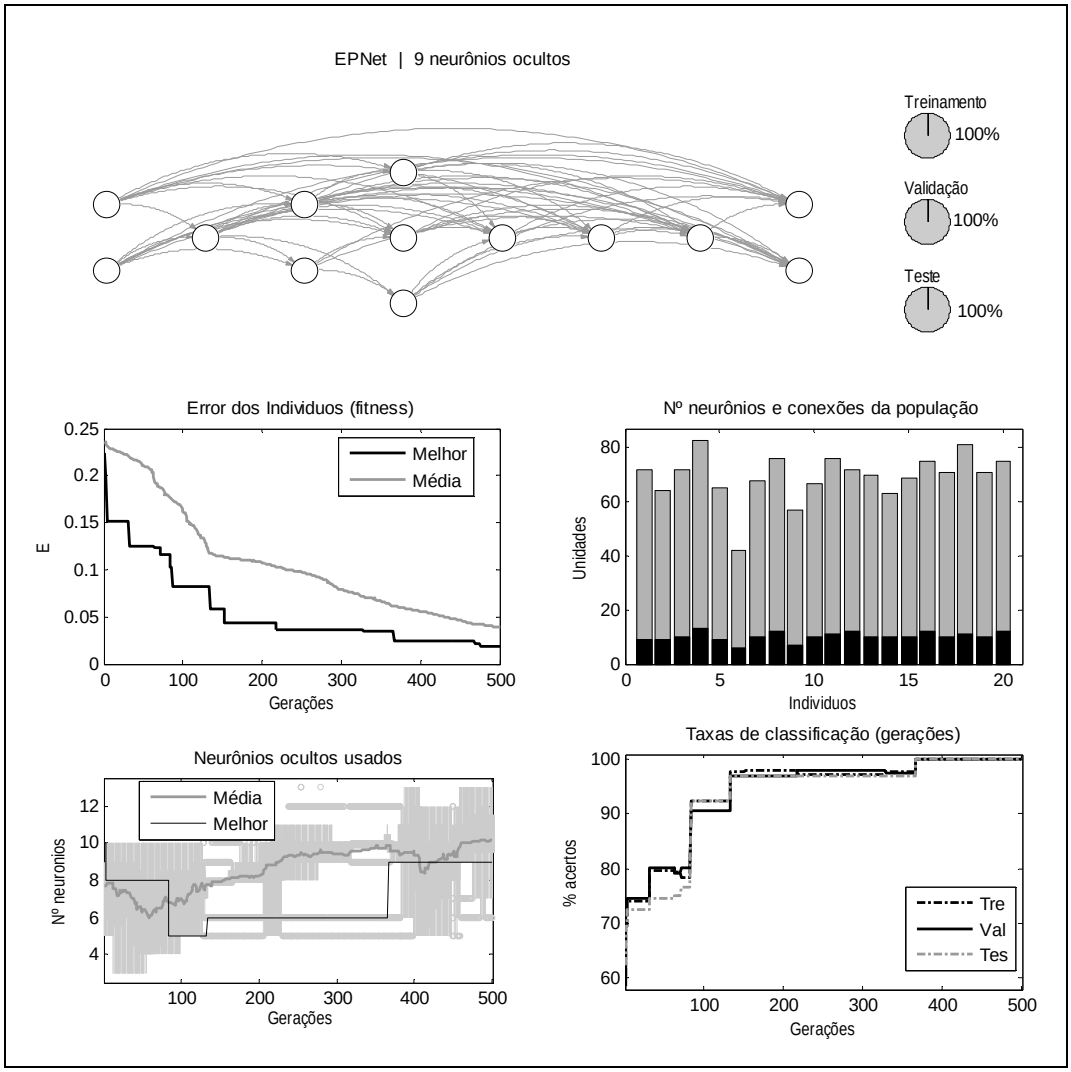


Figura 3.6 – Resultados produzidos pelo EPNet para o problema *Two Spirals*.

Os resultados do EPNet mostraram-se melhores do que aqueles alcançados para o ACFNN-GA, incluindo a parcimônia das redes obtidas. No entanto, um ponto negativo desta metodologia é o grau de dependência do usuário na definição adequada de parâmetros. Principalmente na definição da faixa de inicialização do número de neurônios ocultos da população. Esta inicialização exige certo grau de experiência do usuário para estimar a demanda do número de neurônios ocultos do problema. Visando analisar melhor este ponto, foram realizados dois testes envolvendo o problema *Two Spirals*, com faixas de inicialização sub-estimadas (de 3 até 5) e sobre-estimada (de 13 até 18) para verificar o grau de coerência do resultado em termos de neurônios ocultos. Os resultados destes testes são apresentados na Figura 3.7.

A informação mostrada na Figura 3.7 está distribuída da seguinte forma: na parte superior os resultado para o caso em que a faixa de inicialização do número de neurônios foi de 3 até 5. Na parte inferior, tem-se o caso com faixa de 13 até 18. Em ambos os casos, mostram-se: a rede resultante e suas taxas de classificação, o gráfico de número de neurônios utilizados pela população a cada geração (incluindo informação do melhor indivíduo e da média da população) e o gráfico com o desempenho (taxas de classificação) do melhor indivíduo a cada geração. O número de gerações foi incrementado para 750 para dar maior tempo ao processo, visto que a inicialização foi propositadamente sub- e sobre-estimada.

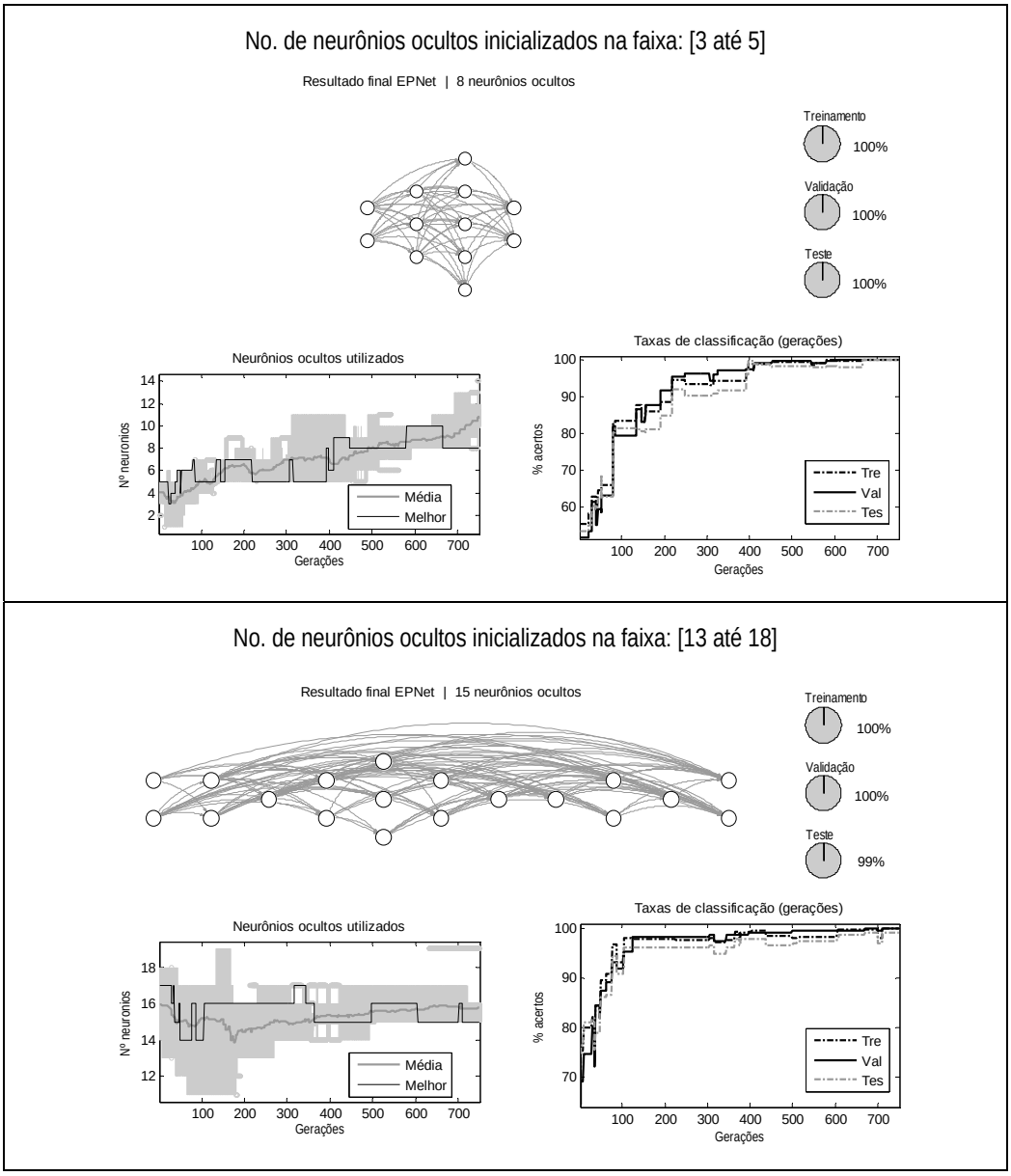


Figura 3.7 – Resultados produzidos pelo EPNet variando a faixa de inicialização de neurônios ocultos para o problema *Two Spirals*.

- Analisando os resultados, para o caso de inicialização com sub-estimação (de 3 até 5). O resultado foi positivo e até convergiu para uma rede mais parcimoniosa com apenas 8 neurônios ocultos distribuídos em duas camadas. Os pontos desfavoráveis que podem ser citados referem-se a: (1) Necessidade de aumentar o número de gerações. Na figura, observa-se que o desempenho ótimo foi alcançado perto da

geração 700. (2) Ao se inicializar com valores muito baixos, há o risco de não crescimento da rede em termos de profundidade (número de camadas ocultas) o que pode ser desfavorável para problemas de alta complexidade. Isto é uma deficiência própria da metodologia, em que a mutação que incrementa o número de neurônios da rede (Mutação #6) apenas duplica um neurônio existente, fazendo crescer a camada, e não criando uma nova camada. O tempo demandado para este teste foi de 108,42 minutos.

- Para o caso de sobre-estimação (de 13 até 18), este demandou 189,65 minutos. O EPNet mostrou dificuldade para produzir uma rede parcimoniosa. A rede obtida possui 15 neurônios ocultos distribuídos em 9 camadas. A interpretação para este caso é que a característica multimodal do mapeamento entrada/saída do problema é mais acentuada quando se excede o número de neurônios ocultos (e camadas ocultas) que o problema requer. Isto pode levar à convergência para mínimos locais e o EPNet não consegue fugir destas situações porque, ao submeter um indivíduo às Mutações #3 e #4, que buscam podar a rede (eliminar neurônios e conexões, respectivamente), estas fazem um papel de operador de busca local (em relação ao pior indivíduo da população) mais do que um papel de mutação propriamente dita, pois somente concretizam a ação e substituição do pior indivíduo se o erro após a mutação for melhor do que o erro do pior indivíduo. Caso contrário, não haverá poda e direciona-se o algoritmo para as Mutações #5 e #6 (Passo 10), que são de adição de conexões e neurônios.

3.4 Considerações finais

As considerações finais estão associadas à implementação e utilização das duas metodologias de computação evolutiva (ACFNN-GA e EPNet). Embora o número de problemas testados seja reduzido (*Wine* e *Two Spirals*) eles foram escolhidos por apresentarem diferentes graus de complexidade, sendo o *Two Spirals* o problema de maior exigência devido a sua natureza não-linear.

Conclusões em relação ao ACFNN-GA:

- Para obter resultados mais robustos, recomenda-se o uso de algoritmos genéticos modificados de forma a superar o problema de permutação. Por exemplo, o algoritmo GAME (KORDÍK *et al.*, 2010), que foi citado na seção 3.1. Este algoritmo faz uso de uma versão de algoritmo genético utilizando o método de *Niching Deterministic Crowding* (MAHFOUD, 1995) para manter uma divisão da população em sub-populações (nichos ecológicos), justamente para garantir que o *crossover* gere descendentes com melhor aptidão que seus progenitores.
- Outro ponto que pode ser melhorado é associado ao tamanho dos indivíduos da população. Recomenda-se o uso de indivíduos de tamanho variável e não com um tamanho fixo e sobre-estimado. A busca por indivíduos que representem redes mais parcimoniosas pode não ser bem sucedida, dependendo da complexidade do problema, além da questão do excessivo custo computacional envolvido durante a evolução.

Conclusões em relação ao EPNet:

- O EPNet mostrou-se mais robusto nestes experimentos, comparado ao ACFNN-GA, pois conseguiu redes com maior grau de parcimônia e apresenta uma evolução coerente e progressiva, graças ao uso de *programação evolutiva* (EP) e um elenco mais amplo de operadores de mutação operando como busca local. Por estes pontos e adicionando o fato de não utilizar *crossover*, o EPNet consegue, como seus criadores o indicaram, evoluir comportamentos, pois o indivíduo descendente herda uma parcela de comportamento do progenitor.
- Um ponto desfavorável do EPNet que poderia ser comentado reside na etapa da inicialização dos parâmetros por parte do usuário, especificamente na definição da faixa de valores para o número de neurônios ocultos iniciais da população. Esta tarefa exige por parte do usuário certo grau de familiaridade com a complexidade do problema para definir uma faixa de valores que contenha o número de neurônios ocultos mais adequado (parcimonioso) que o problema requer. Recomenda-se

inicializar com valores baixos e elevar o número de gerações, caso se desconheça uma boa aproximação da demanda de neurônios ocultos do problema a tratar. Esta recomendação está baseada no experimento associado à Figura 3.7.

Capítulo 4

Algoritmo construtivo para a síntese de redes neurais com conexões à frente arbitrárias

Resumo: Este capítulo apresenta o algoritmo CoACFNNA (do inglês *Constructive Arbitrarily Connected Feedforward Neural Network Algorithm*) como a principal contribuição desta tese para a síntese de redes neurais artificiais com conexões diretas e arbitrárias (ACFNNs, do inglês *Arbitrarily Connected Feedforward Neural Networks*). Na parte inicial do capítulo, apresentam-se duas considerações prévias: (*i*) o cálculo do vetor gradiente para redes do tipo ACFNN, baseado em resultados da literatura e lembrando que as arquiteturas resultantes podem apresentar topologias bastante diversas e arbitrárias, e (*ii*) o ajuste de pesos sinápticos usando o método quasi-Newton com escalonamento automático, sendo que esta forma de ajuste também representa uma contribuição original da tese. Posteriormente, apresenta-se a descrição detalhada do CoACFNNA e, para finalizar, discutem-se testes preliminares no contexto de regressão de dados (predição de quatro séries temporais). Cabe destacar que o Capítulo 5 está destinado à realização de uma maior quantidade de testes de caráter comparativo, com aplicações junto a problemas de classificação de dados e redução de dimensão.

4.1. Considerações iniciais

O significado de *algoritmo construtivo* usado nesta tese engloba tanto procedimentos de *adição* como de *poda* de componentes (neurônios e conexões). Na literatura, é possível

encontrar algoritmos em que o termo construtivo está associado unicamente a procedimentos de adição de componentes, a partir de arquiteturas iniciais com composição mínima. Por outra parte, os algoritmos que somente realizam procedimentos de poda/remoção de componentes são considerados como *algoritmos de poda* e não construtivos (ver Seção 2.3 do Capítulo 2), e tendem a partir de arquiteturas sobredimensionadas, com excesso de neurônios e/ou conexões. O CoACFNNA (PUMA-VILLANUEVA *et al*., 2012) usa ambos os procedimentos, de adição e de poda, sendo que existem outras propostas da literatura que também adotam este paradigma, mas não no contexto de ACFNNs.

O conhecimento adquirido com a pesquisa, implementação e análise de uma ampla gama de modelos, arquiteturas e algoritmos de treinamento tais como, redes MLP de uma e várias camadas ocultas, o algoritmo construtivo *Cascade Correlation* (Capítulo 2), a síntese de ACFNNs via algoritmos de computação evolutiva (Capítulo 3), técnicas de ajuste de pesos sinápticos de RNAs utilizando algoritmos de caráter analítico, baseado em conceitos de otimização não-linear, permitiram a idealização e implementação do CoACFNNA. Os aspectos fundamentais levados em conta na viabilização desta proposta foram os seguintes:

- O custo computacional versus a qualidade do resultado. O elevado custo computacional por parte dos algoritmos baseados em busca populacional (no caso desta tese, os algoritmos ACFNN-GA e EPNet) para este tipo particular de problema. Embora existam ganhos em termos de desempenho e parcimônia nas redes geradas, estes ganhos geralmente não justificam o custo computacional demandado. Este cenário motiva a elaboração de um novo algoritmo que encontre boas soluções e com um menor custo computacional, que pode inclusive conduzir a reduções de ordem de grandeza nos custos.
- Estratégia de construção utilizando procedimentos de adição e poda. Dentre os aspectos destacáveis dos Capítulos 2 e 3, pode-se argumentar que é conveniente começar com propostas de redes com uma arquitetura inicial de composição mínima ou próxima a isso (por exemplo, o algoritmo construtivo *Cascade Correlation* e o EPNet inicializando a população de indivíduos com um número bem reduzido de neurônios

ocultos) e ir adicionando neurônios e conexões de acordo com a demanda do problema, em vez de começar com arquiteturas que podem apresentar, de início, um número elevado de componentes, possivelmente não vinculados ainda às demandas específicas da aplicação. Este cenário leva a pensar num novo algoritmo construtivo que combine procedimentos de adição e poda arbitrária de neurônios e conexões, de forma cíclica e sequencial, que leve a redes possivelmente mais parcimoniosas, sem detrimento do desempenho e sem elevar muito o custo computacional da fase de concepção da rede neural artificial.

- Decisões de projeto mais dedicadas às demandas específicas da aplicação, usando informação a priori, quando disponível. Neste ponto, destaca-se que a maioria das decisões envolvidas nos procedimentos de adição e poda de neurônios e conexões nos algoritmos ACFNN-GA e EPNet (descritos no Capítulo 3) carecem de alguma forma de busca guiada, com exceção da *eliminação de conexões* do EPNet, que utiliza uma forma de mensurar a importância das conexões. Os procedimentos restantes são basicamente aleatórios. Este cenário pode ser melhorado ao inserir decisões guiadas, embora de abrangência local. Nesse sentido, neste trabalho consideram-se duas formas de melhorar a tomada de decisões: (*i*) uma forma de análise de sensibilidade para eliminar conexões; (*ii*) o uso de um índice de informação mútua na eliminação de neurônios e na adição de conexões na rede. Isto favorece o incremento na qualidade das decisões e tende a promover uma redução significativa do tempo de processamento.

- Ajuste dos pesos sinápticos usando uma técnica analítica de otimização não-linear, a qual emprega informações de segunda ordem. A viabilidade do cálculo do vetor gradiente dos pesos sinápticos de arquiteturas de redes do tipo ACFNNs permite a utilização de técnicas analíticas de ajuste dos pesos sinápticos como, por exemplo, gradiente simples, gradiente conjugado, métodos quasi-Newton, dentre outros. Neste trabalho, é proposto o uso de um método quasi-Newton com escalonamento automático, em razão de utilizar informação de segunda ordem e pela rapidez na sua convergência.

As duas subseções seguintes apresentam o cálculo do vetor gradiente das arquiteturas de rede ACFNN e o algoritmo de ajuste de pesos sinápticos baseado num método quasi-

Newton, que serão utilizados pelo algoritmo construtivo proposto nesta tese, o qual será descrito após a formalização de dois de seus módulos principais.

4.1.1. Cálculo do vetor gradiente em ACFNNs

Considerando o caráter arbitrário das conexões e disposição de neurônios em ACFNNs, o cálculo do vetor gradiente deve ser realizado *neurônio por neurônio* (WILAMOWSKI *et al.*, 2008). Isso representa uma diferença significativa quando comparado ao que é geralmente realizado em modelos de redes neurais MLP, onde o cálculo é geralmente realizado *camada por camada*, utilizando matrizes e vetores para facilitar a propagação de sinais pela rede neural e para armazenar as informações de primeira e segunda ordem dos algoritmos de otimização não-linear. Parte desta representação, envolvendo os pesos sinápticos entre duas camadas de neurônios da rede, é mostrada no lado esquerdo da Figura 4.1. As dimensões destas matrizes de pesos estão diretamente associadas com o número de entradas, saídas e neurônios por camada da rede neural. Já no caso de uma rede do tipo ACFNN, por não haver regularidade na distribuição de neurônios e conexões pela rede, a menos do fato de que todas as conexões são diretas, ou seja, não-recorrentes, outras estruturas de dados são requeridas para representar a arquitetura, mais especificamente a indexação dos neurônios e a definição de origem e destino de cada uma das conexões sinápticas, assim como os valores dos pesos sinápticos, cada qual associado a uma conexão. A parte direita da Figura 4.1 mostra uma forma de representar uma instância de rede do tipo ACFNN por meio de ponteiros e estruturas de dados mais avançadas, como um vetor de vetores, por exemplo. A estrutura de dados **A** armazena informação da arquitetura usando valores inteiros e a estrutura de dados **W** armazena os pesos sinápticos com valores reais.

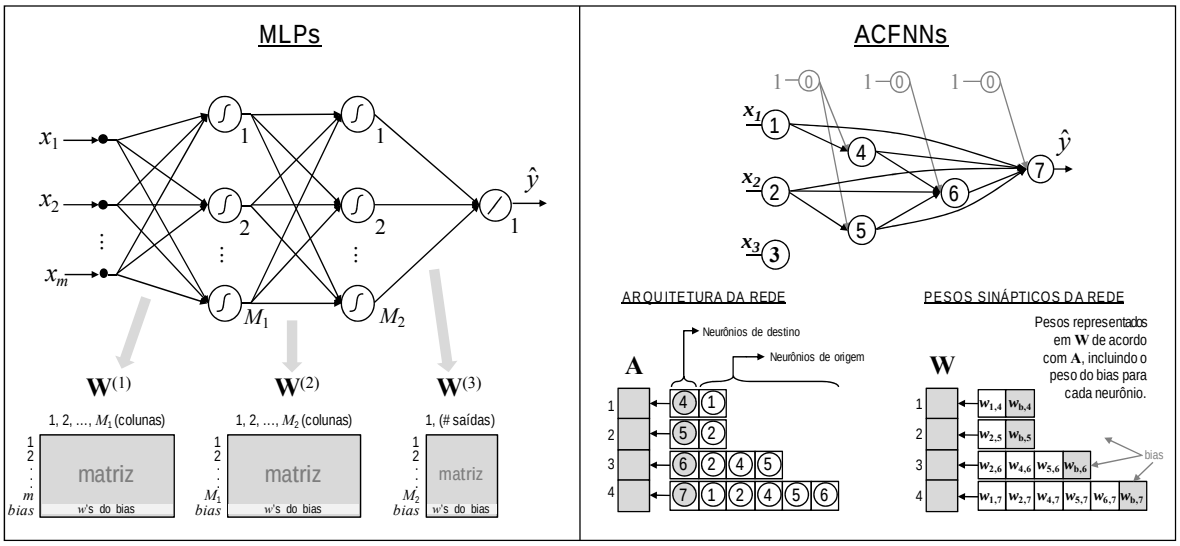


Figura 4.1 – Representação da arquitetura e pesos sinápticos de exemplos de uma rede MLP e uma ACFNN.

Na próxima subseção, será mostrada, via pseudocódigo (adaptação de WILAMOWSKI *et al.*, 2008), a forma de calcular as saídas de redes do tipo ACFNN, forma esta que será útil para o cálculo do gradiente.

4.1.1.1. Pseudocódigos para o cálculo da saída de ACFNNs

Para facilitar o entendimento dos pseudocódigos, previamente apresentam-se as descrições das principais variáveis neles contidos:

O: é uma matriz $N \times no\_neuronios$ (linhas × colunas), com N representando o número de dados de treinamento e *no_neuronios* o número total de neurônios na rede (de entrada, ocultos e de saída). Esta matriz armazena a informação de saídas de cada neurônio para cada dado de treinamento. Note que os neurônios de entrada cumprem a tarefa de apenas propagarem a informação de entrada. Logo, os valores em **O** para estas colunas seriam os dados de entrada contidos na matriz **X**.

S: matriz de $N \times no\_neuronios$, para armazenar as derivadas das funções de ativação de cada neurônio. As colunas correspondentes aos neurônios de entrada não são utilizadas,

pois não há pesos associados a eles e nenhuma das entradas é composta pela realimentação de sinais da própria rede neural.

A: estrutura de dados que armazena a arquitetura da ACFNN. A Figura 4.1, à direita, mostra um exemplo ilustrativo, no qual cada linha representa as conexões recebidas por cada neurônio (oculto ou de saída) armazenado na primeira coluna. A posição do neurônio numa dada linha indica a ordem de inserção dele na rede neural e o índice da linha é o próprio índice do neurônio. Nas outras colunas, armazenam-se os neurônios (em forma ordenada, do menor para o maior índice) de onde partem as conexões.

W: é uma estrutura de dados (ver Figura 4.1, à direita) que armazena os pesos sinápticos. Cada linha de **W** está associada a cada linha de **A**, e cada coluna está diretamente associada à aresta devidamente especificada em **A**. Há um elemento adicional, ao final de cada linha, que representa o peso da conexão associada a uma entrada constante que todo neurônio possui, também denominada de entrada de polarização ou bias do neurônio.

G: é uma estrutura de dados idêntica a **W** que armazena os valores das derivadas da função-objetivo em relação aos pesos sinápticos da rede. Portanto, os elementos de **G** compõem o vetor gradiente, a ser empregado no processo de ajuste de pesos via otimização não-linear.

Antes do cálculo dos elementos que compõem o vetor gradiente, é preciso apresentar o pseudocódigo do cálculo das saídas da ACFNN. Este procedimento permitirá também o preenchimento das matrizes **O** e **S**. A estrutura **A** servirá como base para guiar o fluxo da informação através da ACFNN. Uma vez obtidas as saídas da rede, será possível calcular o erro quadrático médio, que corresponde à função-objetivo a ser minimizada, e, finalmente, calcular o gradiente.

Pseudocódigo 4.1 – Cálculo da saída de ACFNNs

```
REPETIR para cada linha a de A,
  net =  zeros(N, 1);   // sendo N o número de amostras de treinamento
  neurons_back = neurônios com saídas ligadas ao neurônio: A(a,1);
  REPETIR para cada neurônio j em neurons_back,
    net = net +  W(a, j) * O( todas_linhas, neurons_back (j) );
```

```
  END
  net = net + W(a, última_posição) * ones(N, 1);   // considerando o peso do bias do neurônio A(a)
  O(todas_linhas, A(a,1)) = net;   // se for neurônio da camada de saída (linear), ou
                                   = tanh(net);   // se for neurônio oculto
  S(todas_linhas, A(a,1)) = net;   // se for neurônio da camada de saída
                                   = (1–O(todas_linhas, A(a,1))^2);   // se for neurônio oculto
END
```

4.1.1.2. Pseudocódigo para o cálculo do gradiente de ACFNNs

O vetor gradiente é o resultado de uma análise de sensibilidade que indica o que acontece com o valor da função-objetivo caso se varie incrementalmente cada peso da rede neural, ou seja, ele contém as derivadas parciais da função-objetivo em relação a cada peso (parâmetro ajustável) da rede neural. Para se chegar ao vetor gradiente, é necessário primeiro passar todos os sinais de entrada da rede neural até a saída, dispondo assim, do grau de ativação de cada neurônio da rede neural.

Pseudocódigo 4.2 – Cálculo da gradiente de ACFNNs

```
E = O(todas_linhas,[somente_coluna_saída]) – Y;  // calculando o erro
// Etapa 1: Processando somente para o neurônio de saída
neurons_back = neurônios com saídas ligadas ao neurônio: A(último,1);
Sig(A(último,1)) = E;
REPETIR para cada neurônio j em neurons_back,
  G(A(último,1), j) = Sig(A(último,1))^T * O(todas_linhas, neurons_back(j));
END
G(A(último,1), último) = Sig(A(último, 1))^T * ones(N,1);   // gradient do peso do bias
// Etapa 2: Processando para os neurônios ocultos
REPETIR para cada neurônio oculto h,   // em ordem descendente
   // Construindo Sig a partir de cada neurônio oculto h
   Sig = [];     temp_sig = [];     temp_ws = [];
   neurons_next = neurônios com entradas ligadas ao neurônio h;
   REPETIR para cada neurônio j em neurons_next,
      temp_sig = [ temp_sig  Sig(neurons_next(j)) ];
```

```
      temp_ws = [ temp_ws ; W(h, neurons_next(j)) ];
   END
   Sig(h) = temp_sig * temp_ws .* S(todas_linhas,h);
   // Obtendo os pesos de entrada do neurônio h
   neurons_back = neurônios com saídas ligadas ao neurônio h;
   REPETIR para cada neurônio j em neurons_back,
      G(h, j) = Sig(h)^T * O(todas_linhas, neurons_back(j));
   END
   G(h, último) = Sig(h)^T * ones(N,1);   // gradiente para o peso do bias
END
```

4.1.2. Método quasi-Newton com escalonamento automático para ajuste de pesos sinápticos

É bem conhecido na literatura de RNAs, em especial quando as funções de ativação dos neurônios são funções diferenciáveis, que o ajuste dos pesos sinápticos (aprendizado a partir dos dados (MITCHELL, 1997)) é uma tarefa de otimização não-linear de média a larga escala (VAN DER SMAGT, 1994). Por tal motivo, é requerido um algoritmo de otimização com uma alta capacidade de convergência, sendo que métodos baseados em quasi-Newton são bons candidatos a atender este requerimento (BATTITI, 1992). As principais características de métodos quasi-Newton são as seguintes:

- Não requerem o cálculo de derivadas de segunda ordem;
- Apresentam complexidade computacional de ordem quadrática, associada com multiplicação de matrizes;
- Apresentam requerimento de memória quadrática, para o armazenamento da aproximação da inversa da matriz hessiana;
- Apresentam rápida convergência;
- Precisam de, no máximo, *n* iterações para a solução de problemas quadráticos, no qual *n* é o número de variáveis a serem otimizadas.

Antes de descrever o método quasi-Newton com escalonamento automático empregado neste trabalho, primeiro são estabelecidas algumas notações auxiliares. Considere a função-objetivo, indicativa do erro quadrático obtido na saída da rede neural, dada pela seguinte equação:

$$E = \frac{1}{2}\sum_{p=1}^{P}\left[y^p - \hat{y}^p\right]^2 \tag{4.1}$$

Sem perda de generalidade, aqui supõe-se uma única saída para a rede neural. Na existência de múltiplas saídas, o somatório acima deve se estender a todas as saídas da rede, pois haverá um erro associado a cada uma delas para cada um dos P padrões de entrada, denotados por p (p=1,...,P).

Considere que todos os pesos sinápticos, numa ordem pré-definida e qualquer, embora fixa, estão armazenados no vetor $\mathbf{w} \in \Re^n$ como as variáveis a serem otimizadas, onde n corresponde ao total de pesos a serem ajustados. Baseados na função-objetivo E da equação (4.1), define-se na iteração $i > 1$ do processo de aprendizado:

- o vetor gradiente $\mathbf{g}_i = \nabla E(\mathbf{w}_i)$;
- a matriz hessiana $\mathbf{H}_i = \mathbf{H}(\mathbf{w}_i)$;
- o vetor de diferença entre dois vetores gradiente consecutivos $\Delta\mathbf{g}_i = \mathbf{g}_i - \mathbf{g}_{i-1}$; e
- o vetor de diferença entre dois vetores de pesos sinápticos consecutivos $\Delta\mathbf{w}_i = \mathbf{w}_i - \mathbf{w}_{i-1}$.

Essencialmente, os métodos quasi-Newton partem da equação iterativa padrão utilizada nos métodos baseados em Newton, a qual é mostrada a seguir (LUENBERGER, 1973):

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda_i\left[\mathbf{H}_i\right]^{-1}\mathbf{g}_i \tag{4.2}$$

onde i representa a i-ésima iteração, o escalar λ_i informa o tamanho do passo (taxa de aprendizagem) na direção $-\left[\mathbf{H}_i\right]^{-1}\mathbf{g}_i$, na i-ésima iteração. O parâmetro λ_i deve ser determinado utilizando um procedimento de busca linear visando diminuir o erro E de

forma mais eficiente. É bem conhecido que os métodos baseados em Newton são bem eficientes na vizinhança do mínimo local, onde a função-objetivo E pode ser bem aproximada por uma função quadrática e, em consequência, resulta em convergência quadrática (RAO, 2009).

Em contrapartida, existem alguns requerimentos associados a esta metodologia:

- Para produzir decrescimento na função-objetivo E, a matriz hessiana $\mathbf{H}_i$ (veja equação (4.2)) precisa ser definida positiva, para todo i. Dado que a função-objetivo E associada ao erro de treinamento é, em geral, não-convexa, $\mathbf{H}_i$ pode se tornar indefinida ou definida negativa no decorrer do procedimento de aprendizado, para algum i.
- O cálculo das segundas derivadas da função-objetivo E, na obtenção da matriz hessiana, e a inversão da matriz hessiana são ambas tarefas que demandam alto custo computacional para problemas de grande porte (como geralmente é o caso no treinamento de redes neurais artificiais). Além disso, tais operações estão sujeitas a imprecisões numéricas.

Os métodos quasi-Newton buscam superar os problemas acima mencionados através do uso sequencial de diferenças entre os vetores gradientes avaliados ao longo das iterações, para construir uma aproximação de $[\mathbf{H}_i]^{-1}$. Esta aproximação elimina a necessidade do cálculo das derivadas de segunda ordem e da inversão da matriz hessiana, ao custo de introduzir aproximações sucessivas para $[\mathbf{H}_i]^{-1}$. Os métodos quasi-Newton contam, assim, com uma taxa de convergência superlinear nas proximidades do ponto ótimo.

Se $[\mathbf{H}_i]^{-1}$ é aproximada por $\mathbf{D}_i$, então a equação (4.2) pode ser expressa na seguinte forma:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda_i \mathbf{D}_i \mathbf{g}_i \tag{4.3}$$

Várias técnicas foram propostas na literatura para o cálculo de $\mathbf{D}_i$, através de aproximações sucessivas, ou seja, cálculo de $\mathbf{D}_{i+1}$ uma vez que $\mathbf{D}_i$ é conhecido. Uma das

técnicas bem-sucedidas dentro da família de métodos quasi-Newton é a proposta por Broyden, Fletcher, Goldfarb e Shanno (BFGS). Seu desempenho foi avaliado de forma favorável através dos anos (RAO, 2009). O método BFGS produz atualizações de $\mathbf{D}_i$ com uma relativa tolerância a buscas lineares inexatas para encontrar o tamanho do passo λ_i. Por outra parte, os seguintes resultados teóricos antecipam condições que interferem no desempenho do algoritmo BFGS (LUENBERGER 1973):

- Se a função-objetivo é quadrática e se $\mathbf{H}$ é a sua matriz hessiana, a taxa de convergência do procedimento iterativo regido pela equação (4.3) é alta se a razão entre o menor e o maior autovalor da matriz $\mathbf{D}_i\mathbf{H}$ for próxima ao valor unitário.

- Seguindo a hipótese de função-objetivo quadrática, a cada iteração i do algoritmo de BFGS, a atualização de $\mathbf{D}_{i+1}$ é tal que $\mathbf{D}_{i+1}\mathbf{H}$ tem um autovalor unitário.

- Em vista das duas condições descritas acima, se $\mathbf{D}_0$ é o valor inicial atribuído a $\mathbf{D}_i$ e esta escolha é tal que todos os autovalores de $\mathbf{D}_0\mathbf{H}$ são significativamente maiores que um, então, pode-se esperar uma relação desfavorável para a razão entre o menor e o maior autovalor da matriz $\mathbf{D}_i\mathbf{H}$ nos passos intermediários do processo iterativo.

Se as três afirmações ocorrerem ao trabalhar com funções-objetivo não-quadráticas e/ou utilizando buscas lineares inexatas, isto pode levar a situações de mau-condicionamento, acompanhados de desempenhos de convergência indesejáveis (BAZARAA *et al*., 1993). Para amenizar esses aspectos de desempenho, é recomendado multiplicar cada $\mathbf{D}_i$ por um fator de escalonamento μ_i, e então aplicar a fórmula à atualização padrão de BFGS. A menos da hipótese de função-objetivo quadrática, métodos que prescrevem fatores de escala de uma forma tal que os autovalores de $\mu_i\mathbf{D}_i\mathbf{H}$ se distribuam acima e abaixo do valor unitário, são conhecidos como métodos de escalonamento automático.

Nesta tese, será utilizado um método de escalonamento automático para métodos quasi-Newton proposto por OREN (1974). Este método foi aplicado de forma bem-sucedida por BEIGI (1993) no treinamento de redes MLP tradicionais. Este algoritmo utiliza as seguintes equações para calcular a atualização da aproximação da inversa da matriz hessiana:

$$\mathbf{D}_{i+1} = \mu_i \left[\mathbf{D}_i - \frac{\mathbf{D}_i \Delta\mathbf{g}_i \Delta\mathbf{g}_i^T \mathbf{D}_i}{\Delta\mathbf{g}_i^T \mathbf{D}_i \Delta\mathbf{g}_i} + \theta_i \mathbf{v}_i \mathbf{v}_i^T \right] + \frac{\Delta\mathbf{w}_i \Delta\mathbf{w}_i^T}{\Delta\mathbf{w}_i^T \Delta\mathbf{g}_i} \tag{4.4}$$

$$\mathbf{v}_i = \left[\Delta\mathbf{g}_i^T \mathbf{D}_i \Delta\mathbf{g}_i \right]^{1/2} \left[\frac{\Delta\mathbf{w}_i}{\Delta\mathbf{w}_i{}^T \Delta\mathbf{g}_i} - \frac{\mathbf{D}_i \Delta\mathbf{g}_i}{\Delta\mathbf{g}_i{}^T \mathbf{D}_i \Delta\mathbf{g}_i} \right] \tag{4.5}$$

$$\mu_i = \phi_i \frac{\mathbf{g}_i{}^T \Delta\mathbf{w}_i}{\mathbf{g}_i{}^T \mathbf{D}_i \Delta\mathbf{g}_i} + (1 - \phi_i) \frac{\Delta\mathbf{w}_i{}^T \Delta\mathbf{g}_i}{\Delta\mathbf{g}_i{}^T \mathbf{D}_i \Delta\mathbf{g}_i} \tag{4.6}$$

Neste trabalho, a matriz $\mathbf{D}$ é inicializada com a matriz identidade e os parâmetros ajustáveis θ_i e ϕ_i foram fixados com os valores $\theta_i = 0{,}5$ e $\phi_i = 1{,}0$, respectivamente. Estes valores foram sugeridos por OREN (1974) baseado em exaustivas simulações. O tamanho do passo na equação (4.6) é determinado via procedimento de busca linear, seguindo o critério descrito por DENNIS & SCHNABEL (1996) (Capítulo 6, Seção 6.3.2 – *Step selection by backtracking*), usando interpolação cúbica somente se a interpolação parabólica se mostrar inadequada. A busca linear obedece ao critério de Wolfe e Powell para evitar que o processo de otimização deixe de progredir por causa da ocorrência de passos de otimização improdutivos (DENNIS & SCHNABEL, 1996).

4.2. Mecanismo de construção do CoACFNNA

O algoritmo construtivo proposto nesta tese para a síntese de ACFNNs é composto por procedimentos executados de forma determinística, que adicionam e removem conexões e neurônios em qualquer região da rede.

Os principais elementos de entrada e saída em cada um dos procedimentos que compõem o algoritmo proposto são os seguintes:

- A arquitetura atual da rede;
- Os valores dos pesos sinápticos das conexões;

- O erro que está sendo minimizado (*E*) associado a um subconjunto de dados de validação (segundo a equação (4.1)).

A Figura 4.2 mostra um fluxograma do algoritmo construtivo proposto. O ponto de partida é uma rede vazia com somente os neurônios das camadas de entrada e de saída, os quais representam diretamente as variáveis de entrada e de saída, respectivamente, mas nenhuma conexão está estabelecida entre estas duas camadas de neurônios. A seguir, o algoritmo entra num procedimento que calcula um índice de informação mútua de cada entrada com a saída desejada (conjunto de saídas) e usa esta informação a priori para estabelecer as primeiras conexões entre as duas camadas. Os seguintes dois procedimentos "adição de conexões" e "adição de um neurônio" buscam, completar todas as conexões necessárias e adicionar um neurônio na rede, respectivamente. Se o neurônio é adicionado, o algoritmo entra no primeiro ciclo 1 (L1), o qual é acessado somente uma vez em todo o decorrer do algoritmo. O ciclo L1 busca de forma rápida incrementar a rede com componentes (neurônios e conexões) lineares/não-lineares de acordo com a demanda do problema. Para isto, usa unicamente os seguintes procedimentos: "adição de neurônios", "adição de conexões" e "eliminação de conexões". Caso um número máximo de neurônios seja atingido ou um insucesso na adição de um novo neurônio aconteça, o algoritmo deixa o ciclo L1 e entra no ciclo L2, o qual se diferencia de L1 nos seguintes aspectos:

- Conta com o procedimento de eliminação de neurônios, ausente no ciclo L1;
- Conta com o mecanismo que controla a "relaxação do erro", ausente no ciclo L1 também.

O mecanismo de relaxação do erro tem por propósito promover alguns recursos para se escapar de mínimos locais, evitando uma convergência prematura do processo de construção da rede neural. O mecanismo atua nos quatro procedimentos do ciclo L2 que alteram a estrutura da rede (pintados com fundo de cor cinza na Figura 4.2). O procedimento "ajuste da taxa de folga do erro" regula a intensidade e duração deste mecanismo. A relaxação do erro E é efetuada antes de entrar em cada um dos procedimentos de cor cinza. Se o procedimento for bem sucedido (produz melhora no erro) o erro é atualizado com o valor proveniente do procedimento. Caso contrário, o valor do

erro assume o valor prévio à relaxação. O algoritmo termina quando os quatro procedimentos em cinza falham de forma consecutiva num mesmo ciclo L2, indicando que não houve melhora no erro, mesmo com a atuação do mecanismo de relaxação. No decorrer do algoritmo, armazena-se a cada passo a rede com menor valor do erro *E*, a qual será a solução do algoritmo construtivo proposto ao final de sua execução.

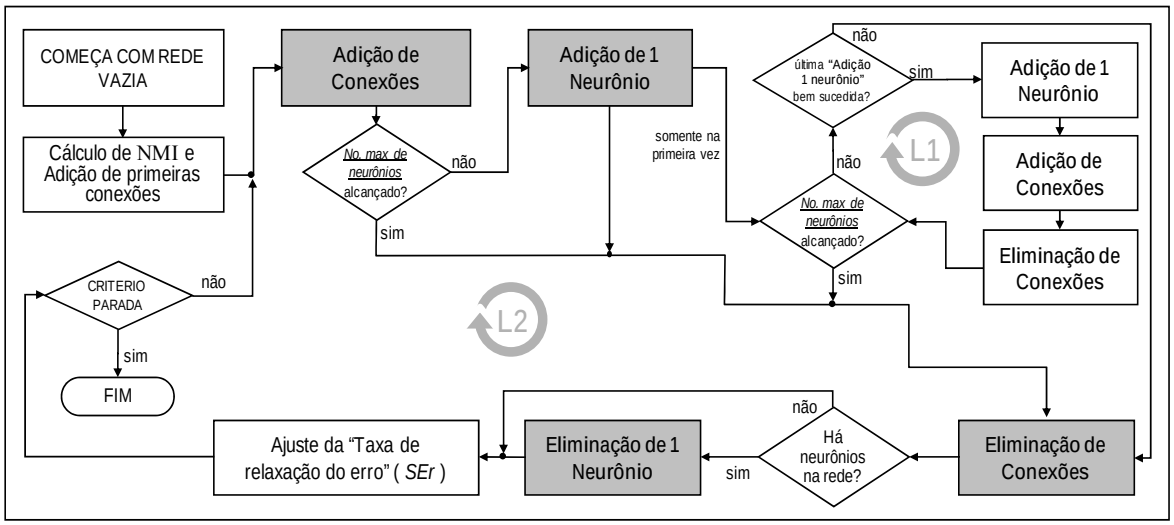


Figura 4.2 – Fluxograma do mecanismo de construção de ACFNNs.

Cada um dos procedimentos que compõem o algoritmo construtivo mostrado na Figura 4.2 será descrito detalhadamente a seguir.

4.2.1. Rede inicial com ausência de conexões

COMEÇA COM REDE VAZIA

A arquitetura da rede inicial contém apenas neurônios nas camadas de entrada e de saída. Cada um destes neurônios está associado às variáveis de entrada e saída do problema a resolver. Não existem conexões entre estas camadas de neurônios.

4.2.2. Cálculo da informação mútua e adição das primeiras conexões

Cálculo de NMI e Adição de primeiras conexões

O aspecto mais intuitivo para ser explorado neste procedimento é o fato de fazer mais sentido o estabelecimento de conexões entre pares de neurônios que apresentam um nível significativo de informação mútua. Dito de outra forma, os neurônios de entrada que carregam mais informação acerca dos neurônios de saída da rede (saída desejada) deverão ser preferidos para o estabelecimento das primeiras conexões. Cada neurônio de entrada está associado com uma variável de entrada x_i (i=1,...,ni) e o comportamento desejado (saída desejada) está associado com as variáveis y_j (j=1,...,no). A informação mútua para cada neurônio i (NMI – *neural mutual information*) de entrada é calculada em relação a todas as saídas desejadas y_j (j=1,...,no). Logo, NMI(i) representa a informação mútua acumulada para o neurônio de entrada i e é calculada da seguinte forma:

$$\mathrm{NMI}(i) = \frac{\sum_{j=1}^{no} \mathrm{MI}(x_i, y_j)}{no}, \tag{4.7}$$

sendo no o número de saídas da rede e i variando de 1 a ni.

Uma vez calculado o NMI para todos os ni neurônios de entrada, resta estabelecer as primeiras conexões da rede. Com o propósito de começar a construção da rede a partir de arquiteturas simples é que unicamente o neurônio de entrada com o maior NMI será conectado com todos os neurônios de saída da rede.

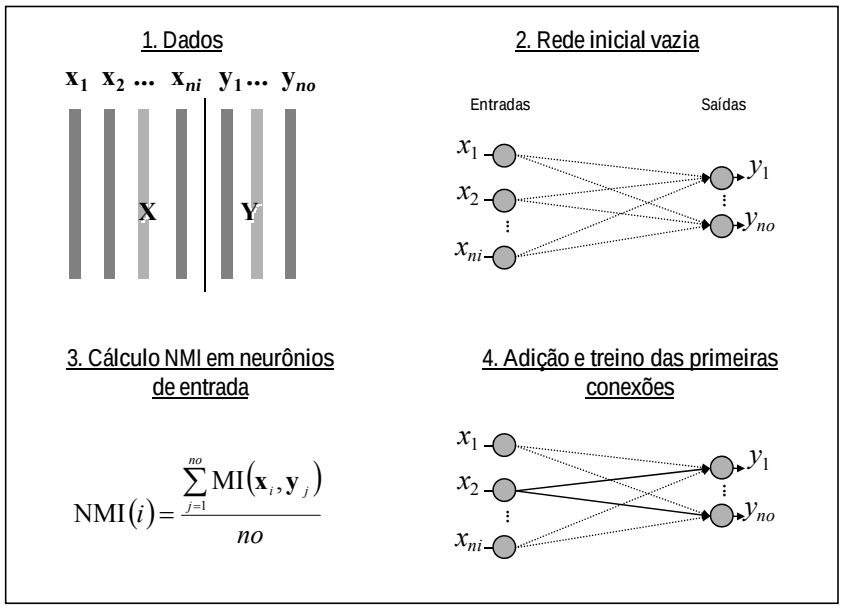


Figura 4.3 – CoACFNNA: Cálculo do NMI e adição das primeiras conexões.

Após serem estabelecidas as primeiras conexões, inicia-se a fase de ajuste de seus correspondentes pesos sinápticos, sendo que aqui pode-se adotar soluções fechadas para problemas de quadrado mínimo, pois todos os pesos se relacionam linearmente com a função-objetivo *E*. A Figura 4.3 ilustra todo o conjunto de procedimentos.

4.2.3. Adição de conexões

Adição de Conexões

Este procedimento encontra-se presente na Figura 4.2, fazendo parte dos ciclos L1 e L2. Em ambos, o propósito é o de adicionar novas conexões. A diferença fundamental está no ciclo L2, onde este módulo aparece com fundo em cor cinza, pois aqui se aplica o mecanismo de relaxação do erro. A seguir, será feita a descrição para este último caso (ciclo L2), deixando claro que no ciclo L1 a relaxação do erro não é aplicada:

- Relaxa-se o valor atual do erro *E* (ver seção 4.2.8);

- Calcule-se a NMI para os neurônios ocultos, de forma análoga ao cálculo feito para os neurônios da camada de entrada (ver equação (4.7));
- Elabora-se uma lista com as conexões candidatas (conexões ainda não estabelecidas na rede). Armazenam-se: neurônios de origem e destino, e o valor NMI para o neurônio de origem;
- Ordena-se a lista segundo os valores de NMI, do maior para o menor;
- Percorre-se a lista com o intuito de estabelecer as conexões. A cada tentativa, a conexão será temporariamente estabelecida e seu peso sináptico associado será inicializado aleatoriamente na faixa de [–0,001, +0,001]. Em seguida, a rede será re-treinada. A conexão somente será estabelecida de forma definitiva se apresentar melhoria na diminuição do erro após o re-treino. Caso contrário, descarta-se esta conexão e a próxima conexão da lista será analisada.
- Após um certo número de conexões não estabelecidas de forma consecutiva, o procedimento será concluído, mesmo que ainda existam conexões não analisadas na lista. O número consecutivo de tentativas falhas foi definido empiricamente como *nl*/4, sendo *nl* o número de conexões candidatas na lista. Esta decisão permite economizar tempo de processamento.
- Se pelo menos uma conexão foi estabelecida, este processo é etiquetado como "bem-sucedido" e o valor do erro *E* é atualizado com o valor proveniente do processo, mesmo este podendo ser maior do que o valor do erro antes do relaxação. Caso nenhuma conexão seja estabelecida, o processo é considerado como "falho" e o erro *E* recupera seu valor inicial (antes da relaxação). A Figura 4.4 ilustra este procedimento.

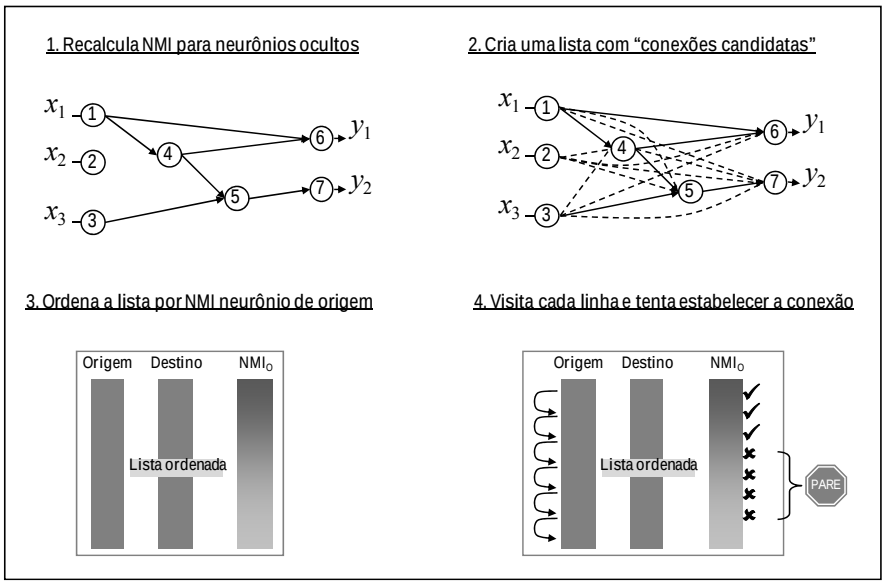


Figura 4.4 – CoACFNNA: Adição de conexões.

4.2.4. Adição de um neurônio

Adição de 1 Neurônio

O incremento em um neurônio na rede que está sendo construída requer basicamente duas informações:

- Definição do local (posição) na rede onde o neurônio será inserido;
- Definição do número de conexões iniciais de entrada/saída deste novo neurônio.

Em relação à posição do novo neurônio, existem dois casos a serem considerados:

C1: se não existem neurônios ocultos na rede, então o novo neurônio formará a primeira camada oculta da rede;

C2: se já existem neurônios ocultos formando uma ou mais camadas ocultas, então o novo neurônio pode incrementar uma das camadas ocultas, ou formar uma nova camada oculta antes da camada de saída da rede.

Para efetivar a adição de um novo neurônio, testam-se todas as considerações em C1 e C2, e escolhe-se a que conduz à maior diminuição do erro *E*. O custo computacional

permite realizar isto, pois o número de re-treinamentos da rede é igual ao número de camadas ocultas mais um.

Em relação às conexões para o novo neurônio, este recebe uma parcela (quinta parte) das conexões com os melhores neurônios anteriores, em termos de NMI, e se conecta a todos os neurônios posteriores. Esta medida tenta equilibrar o compromisso de parcimônia da rede. Se a parcela destas conexões for muito baixa, corre-se o risco do neurônio não ser inserido já que não poderá contribuir na diminuição do erro, e se a parcela for muito elevada, o princípio de parcimônia é afetado, aumentando a complexidade da rede. A parcela da quinta parte foi a que apresentou melhores resultados em testes preliminares com problemas distintos e nos quais houve crescimento gradativo e sustentável da rede.

O mecanismo de relaxação é aplicado sobre este procedimento unicamente dentro do ciclo L2 do fluxograma da Figura 4.2 (fundo de cor cinza). Assim, o erro *E* é relaxado antes de se iniciar o procedimento. Caso o neurônio seja adicionado na rede, o procedimento é considerado "bem sucedido" e o erro *E* é atualizado com o novo valor de erro proveniente do procedimento. Caso o neurônio não seja adicionado, o procedimento é etiquetado como "falho" e o erro *E* retorna ao seu valor antes da relaxação. A Figura 4.5 ilustra os passos deste procedimento.

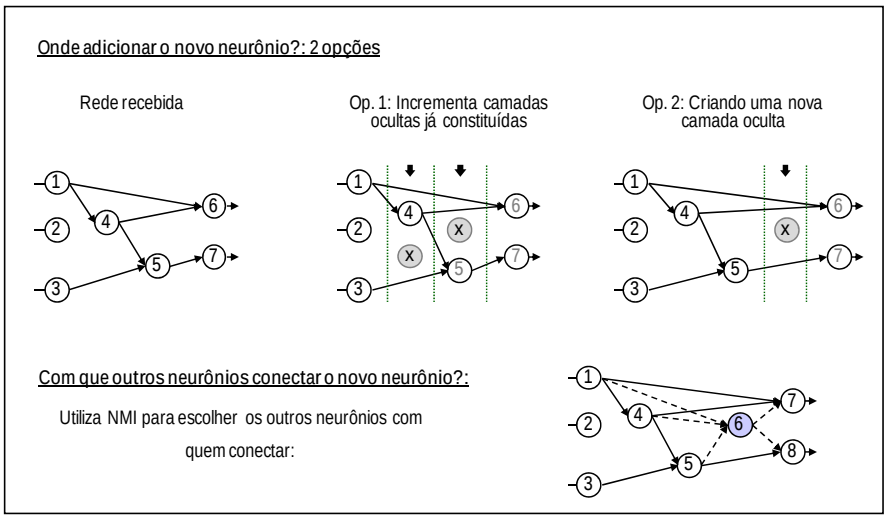


Figura 4.5 – CoACFNNA: Adição de um neurônio.

4.2.5. Eliminação de conexões

Eliminação de Conexões

Este procedimento também trabalha com o mecanismo de relaxação do erro *E* dentro do ciclo L2 da Figura 4.2. Já no ciclo L1 isto não é aplicado. A seguir, descrevem-se os aspectos básicos deste procedimento o qual é ilustrado na Figura 4.6:

- Cria-se uma lista com todas as conexões já estabelecidas na rede (consideradas candidatas a serem eliminadas). Armazena-se na lista informação de origem e destino das conexões.
- Calcula-se uma forma de ordenamento das conexões na lista baseado numa análise de sensibilidade. Anula-se a conexão associada a cada linha da lista (o valor do peso sináptico assume temporariamente o valor zero). Em seguida, é calculado o erro da rede. Assim, para cada caso de conexão anulada, calcula-se a diferença da variação (sensibilidade) no erro *E* da seguinte forma: [*E_antes* – *E_depois*] e armazena-se este valor em cada linha da lista. Note que valores positivos denotam melhora na diminuição do erro.
- Ordena-se a lista em relação ao valor da sensibilidade, de maior para menor.
- Percorre-se a lista ordenada desde o início e avalia-se a eliminação de cada conexão nela contida. Temporariamente elimina-se fisicamente a conexão, re-treina-se a rede e, caso o erro sofra diminuição, a eliminação da conexão passa a ser definitiva. Após um número consecutivo de tentativas de eliminações falhas, o procedimento é dado por concluído. Visando cuidar do tempo computacional, foi definido de forma empírica como *nl*/5 o número de tentativas falhas, onde *nl* é o número de conexões na lista.
- O procedimento é declarado "bem-sucedido" se pelo menos uma conexão for eliminada e o erro *E* atualizado com o novo valor. Caso contrário, é declarado como "falho" e o erro *E* recupera seu valor original de antes da relaxação (se estiver no ciclo L2).

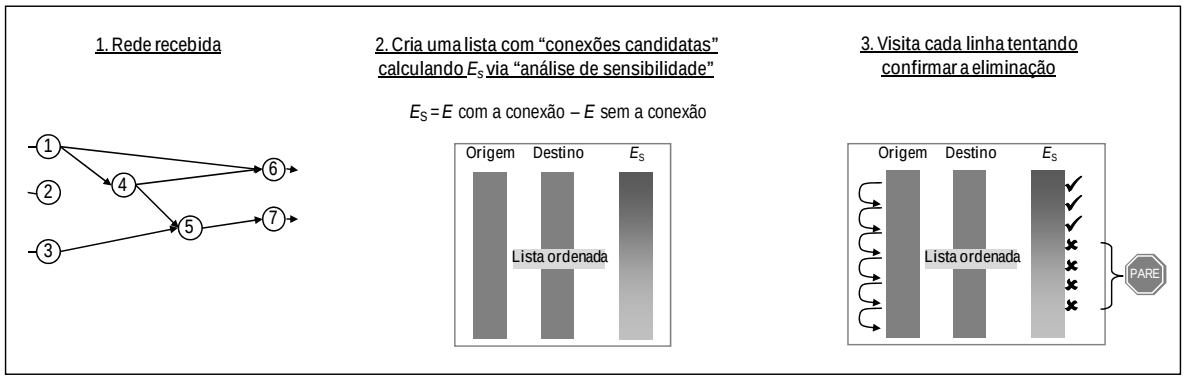


Figura 4.6 – CoACFNNA: Eliminação de conexões.

4.2.6. Eliminação de um neurônio

Eliminação de 1 Neurônio

Este procedimento trabalha somente dentro do ciclo L2 e sempre se aplica o mecanismo da relaxação. Seguem-se os seus passos também ilustrados na Figura 4.7:

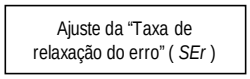
- Cria-se uma lista contendo os neurônios ocultos existentes na rede (candidatos a serem eliminados).
- Para cada neurônio da lista, calcula-se o valor de seu NMI.
- Ordena-se a lista segundo o valor de NMI, de menor para maior.
- Visita-se cada elemento da lista na tentativa de eliminar um neurônio. Temporariamente elimina-se o neurônio, re-treina-se a rede e, caso o erro mostre diminuição, a eliminação do neurônio será definitiva.
- O procedimento é concluído logo após a eliminação de um neurônio ou após se ter visitado todos os elementos da lista.
- O procedimento é declarado "bem-sucedido" caso o neurônio seja eliminado, e "falho" em caso contrário.

Figura 4.7 – CoACFNNA: Eliminação de neurônio.

4.2.7. Critério de parada

O critério de parada do algoritmo é alcançado quando os quatro procedimentos que alteram a estrutura da rede (caixas de cor cinza na Figura 4.2) são considerados como "falhos" de forma consecutiva dentro de um mesmo ciclo L2. Este cenário indica que nenhuma conexão ou neurônio, dentre aqueles que foram adicionados ou removidos da rede, trouxe ganho na diminuição do erro de treinamento *E*.

4.2.8. Ajuste da taxa de relaxação do erro



Com a intenção de reduzir a chance de que o algoritmo possa convergir para mínimos locais não desejados, foi utilizado um mecanismo de relaxação do erro. Com este mecanismo ativo, é possível degradar a qualidade do valor do erro de treinamento *E* de tal forma que se possa aceitar a aplicação de ao menos um dentre os quatro procedimentos que alteram a estrutura da rede (ciclo L2).

A heurística descrita a seguir procura controlar de forma automática o valor da taxa com que o erro *E* é alterado no decorrer do algoritmo. A equação (4.8) mostra a forma como isto é realizado:

$$E' = E \times SEr \,, \tag{4.8}$$

sendo que E' representa o novo valor de E relaxado e a taxa de relaxação do erro é representada pela variável *SEr* (*slack error rate*). Os valores que esta taxa pode assumir estão vinculados ao desempenho da rede no decorrer do processo de treinamento. Em outras palavras, à medida que o valor de E apresente diminuição, o valor de *SEr* também sofrerá diminuição, e vice versa. Com isso, enquanto o erro E tiver valores altos, o *SEr* também apresentará valores altos.

Os possíveis valores que a taxa *SEr* pode assumir estão limitados ao seguinte intervalo: [*SEr_min*; *SEr_max*]. Se *SEr* assumir o valor de *SEr_min* (que é fixado em 1), significa que o valor de E não sofrerá relaxação e manteria seu próprio valor ($E' = E$).

O valor inicial para *SEr_max* é definido pelo usuário. O valor de *SEr* começa sendo igual a *SEr_max* e, à medida que o algoritmo entra em funcionamento, estes valores são auto-ajustados visando auxiliar o processo construtivo. O valor inicial para *SEr_max* sugerido é de 1,1, o que significa que, ao iniciar, o algoritmo permitirá que o erro E sofra aumentos na ordem de 10%.

Para atualizar o valor de *SEr* de forma que seja dependente do valor do erro E, é necessário definir uma escala de valores de atuação para E da mesma forma feita para *SEr*. Em consequência, foram definidos *E_max* e *E_min* da seguinte forma:

- *E_max*: definido como o valor inicial do erro E, calculado após o procedimento que adiciona as primeiras conexões na rede.
- *E_min*: definido como o erro de treinamento desejado. Idealmente este valor seria nulo. No entanto, ao lidar com problemas reais isto não é sempre alcançado e levaria a cometer sub-estimações para *E_min*. Uma alternativa encontrada após testes preliminares envolvendo problemas diversos foi considerar *E_min* como uma fração do valor atual de E (EF – *error's fraction*). Assim, define-se EF = 0,5, o que significa que, em cada avaliação, *E_min* assumirá a metade do valor de E.

Uma vez ajustado o intervalo [*E_min*; *E_max*], pode-se calcular a atualização do valor de *SEr* da seguinte forma:

$$SEr = SEr\_max - \frac{(E\_max - E)(SEr\_max - SEr\_min)}{E\_max - E\_min}. \tag{4.9}$$

Na equação (4.9), mostra-se de forma clara que a taxa *SEr* está fortemente correlacionada com o comportamento do erro de treinamento *E*, como esperado. Se o erro *E* permanecer estacionário e distante de zero, então *SEr* não alcançará *SEr_min* e, assim, a relaxação não será atenuada, comprometendo a convergência. Levando em conta este último cenário, uma consideração final deve ser adotada: o intervalo [*SEr_min*; *SEr_max*] precisa ser reduzido no decorrer do algoritmo. A Figura 4.8 ajuda a mostrar as circunstâncias em que esta redução é recomendada. Ela mostra uma trajetória do erro *E* em um cenário ilustrativo, onde *I* representa a iteração de início do algoritmo, *G* é a iteração em que o último mínimo global foi alcançado, *C* é a iteração atual, $\overline{IG}$ é a distância entre o início do algoritmo e o último mínimo global, e $\overline{GC}$ é a distância entre o último mínimo global e o instante atual do algoritmo. Dessa forma, a cada iteração em que $\overline{GC} \geq \overline{IG}/4$, *SEr_max* deverá ser diminuído numa proporção indicada por *dec* = 0,75, da seguinte forma:

$$SEr\_max = SEr\_min + dec(SEr\_max - SEr\_min). \tag{4.10}$$

Em termos práticos, o objetivo desta equação (4.10) consiste em diminuir indiretamente a taxa de relaxação em situações em que o erro permanece constante ou entrou em um ciclo em que seu valor alterna sem tendência clara (neurônios e/ou conexões são eliminados e adicionados em forma repetitiva, já que o valor da taxa *SEr* ainda permite estas alterações). Isto pode estender o processo de convergência e até impedir a finalização do algoritmo. Os valores de *dec* e a condição $\overline{GC} \geq \overline{IG}/4$ foram definidos após testes preliminares.

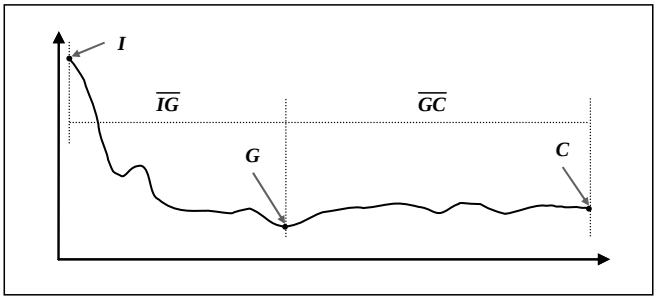


Figura 4.8 – Cenário ilustrativo usado para definir I, G, C, $\overline{IG}$ e $\overline{GC}$.

A Figura 4.9 ilustra um exemplo real do comportamento de *SEr* e *SEr_max*, de acordo com as equações (4.9) e (4.10), respectivamente. Este comportamento corresponde ao problema de classificação de dados "*Breast Tissue*".

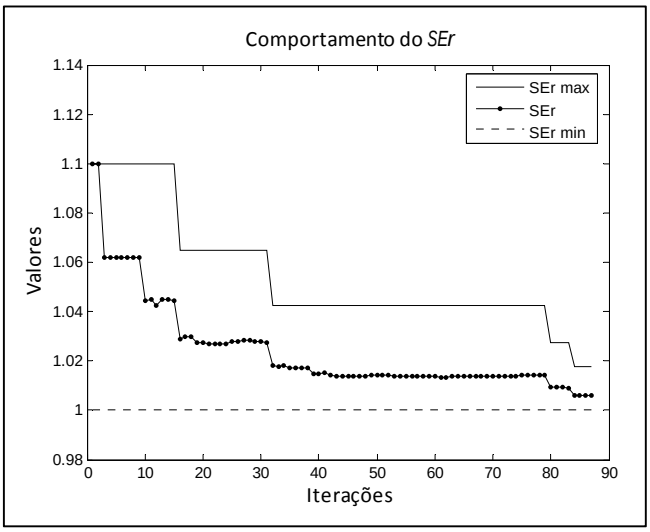


Figura 4.9 – Comportamento de *SEr* e *SEr_max* para o problema "*Breast Tissue*".

4.2.9. Aspectos de convergência do algoritmo

A convergência do algoritmo pode ser considerada levando-se em conta dois aspectos: (*i*) As redes ACFNNs como aproximadores universais e sua taxa de convergência, e (*ii*) O critério de parada da dinâmica do algoritmo. A seguir, mais detalhes:

As redes ACFNNs como aproximadores universais e sua taxa de convergência: As redes do tipo ACFNNs se constituem em aproximadores universais devido a que instâncias mais simplificadas deste tipo de arquiteturas, tais como as redes MLPs são consideradas aproximadores universais (HORNIK *et al*., 1989; CYBENKO, 1989; FUNAHASHI; 1989). Por outra parte, estudos envolvendo a taxa de convergência do algoritmo construtivo *Cascade Correlation* mostraram, num cenário de pior caso, que este algoritmo é capaz de aproximar qualquer função com uma taxa de convergência da ordem de $O(1/n_h)$, sendo n_h o número de neurônios ocultos (DRAGO & RIDELLA, 1994). Posteriormente, os mesmos autores analisaram o pior caso deste algoritmo (DRAGO & RIDELLA, 1996) e mostraram que a taxa de convergência seria incrementada se esta arquitetura aceitasse mais de um único neurônio em cada camada oculta. O CoACFNNA admite mais de um neurônio em cada camada oculta e, consequentemente, tem uma taxa de convergência, de pior caso, igual ou superior àquela obtida para o algoritmo construtivo *Cascade Correlation*, simplesmente pelo fato de que a opção de construir a topologia da rede neural seguindo a abordagem de correlação em cascata é sempre testada (usando um índice de informação mútua) e, se não for escolhida naquela iteração é porque existe uma ação construtiva mais capaz de reduzir o erro de treinamento, ou seja, de acelerar localmente a convergência em comparação com aquilo que seria produzido via um algoritmo construtivo *Cascade Correlation*.

O critério de parada da dinâmica do algoritmo: Este aspecto depende do valor inicial da variável "*SEr_max*" e do seu correspondente coeficiente de decaimento "*dec*", ambos definidos pelo usuário. *SEr_max* pode assumir qualquer valor maior ou igual a 1 e *dec* valores dentro do intervalo [0; 1]. *SEr_max* representa a intensidade nas mudanças que a topologia da rede pode sofrer e *dec* controla a duração destas mudanças. Em relação a estes parâmetros, pode-se ilustrar os seguintes cenários:

- Fixando *dec* no seu valor sugerido (0,75) e iniciando *SEr_max* com um valor alto, por exemplo igual a 2, significaria que o erro *E* pode duplicar o seu valor e o algoritmo terá alto grau de liberdade para mudanças drásticas na sua arquitetura, além de permitir uma deterioração significativa no desempenho da rede. Este

cenário pode ser interpretado como um comportamento de exploração exagerado, tornando a convergência errática e mais lenta.

- Fixando *SEr_max* no valor sugerido de 1,1 e *dec* no valor extremo superior de 1, significa que *SEr_max* nunca diminuirá seu valor e, em consequência, *SEr* será atualizado somente em proporção à diminuição do erro de aprendizado *E* no decorrer das iterações. Estas condições não garantiriam a *SEr* alcançar seu valor mínimo e, com isso, dar início ao processo de convergência do algoritmo. A desvantagem deste cenário seria o surgimento de comportamentos cíclicos na adição e remoção de neurônios e conexões, permitindo inclusive que, em algumas circunstâncias, o ciclo L2 opere indefinidamente.
- Um outro cenário seria estabelecer *dec* no seu valor limitante inferior de 0 (com *SEr_max* fixo em 1,1, por exemplo). Neste caso, *SEr_max* receberá o valor *SEr_min* logo na primeira avaliação da equação (4.10) e, consequentemente, o mecanismo de relaxação será desabilitado e a convergência do algoritmo tende a caminhar para o mínimo local mais próximo.

4.3. Aplicação do CoACFNNA junto a problemas de regressão de dados

O primeiro tipo de problema a que o CoACFNNA será submetido para observar seu desempenho é o de regressão de dados, mais especificamente, problemas de predição de séries temporais. Testes com outros tipos de problemas (classificação de dados e redução de dimensão) serão apresentados no Capítulo 5.

As séries temporais consideradas nesta seção são mostradas na Figura 4.10. A seguir, uma breve descrição de cada uma delas:

- *Clothing store*: Refere-se a uma série temporal de natureza financeira com valores de venda de roupa mensais médios do varejo nos EUA, a qual conta com 120

amostras no período de 01/1992-12/2001. Esta série se caracteriza por ser linear, com componente sazonal anual.

- IPI *Durable consumer goods*: Série temporal financeira sobre vendas de bens de consumo duráveis, a qual conta com 660 amostras. Tanto esta série como a *Clothing store* foram apresentadas por ZHANG & QI (2005) num trabalho que estuda a influência do pré-processamento da série temporal (remoção de componentes de tendência e sazonalidade) no erro de predição.
- *Sunspot*: Série temporal com 289 amostras (no período de 1700-1888). É de natureza astrofísica, com valores representando o número de manchas solares por ano.
- *Two nonlinear processes*: Série temporal sintética formada pela comutação de dois processos não-lineares, tendo sido proposta por WEIGEND *et al*. (1995) para verificar a capacidade de uma mistura de especialistas não-lineares em alocar cada especialista a cada processo que participa da comutação. Ao total, foram geradas 2000 amostras.

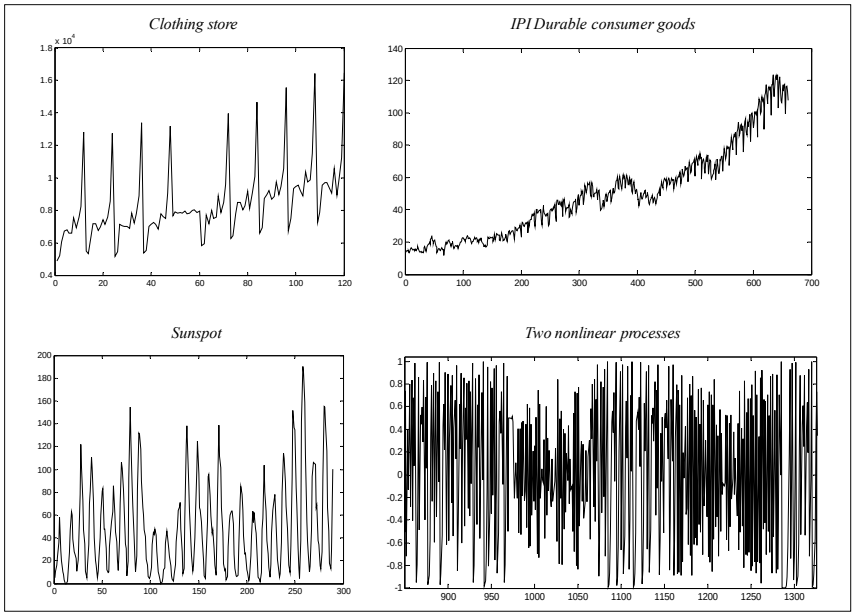


Figura 4.10 – Séries temporais consideradas para testar o CoACFNNA.

O único pré-processamento feito nas séries temporais foi a mudança de escala de seus valores para o intervalo [−1; +1] visando facilitar o uso de RNAs com funções de ativação tangente-hiperbólica. Após isto, resta a montagem dos subconjuntos de dados para treinamento, validação e teste, sempre procurando preparar os dados para a realização de predições de valores futuros, no caso, *um passo à frente*. A Figura 4.11 descreve esta etapa, onde o valor de *L* (janela de número de valores de atraso que irão compor a entrada) é estimado usando a função de auto-correlação – ACF (BOX *et al.*, 1994) ou via Informação Mútua – MI (CELLUCCI *et al.*, 2005), como exemplificado na Figura 2.16.

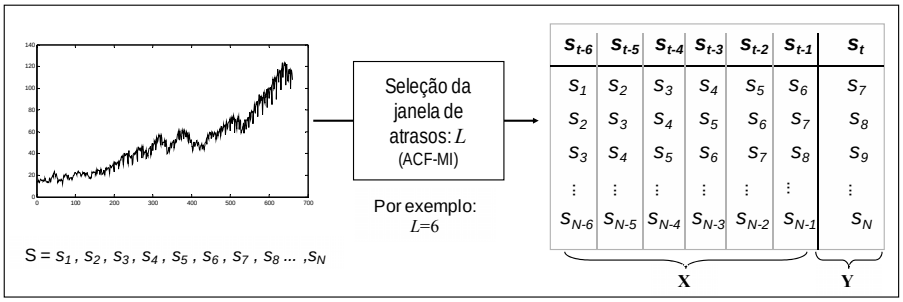


Figura 4.11 - Preparação dos dados para predição de um passo à frente com L=6 valores atrasados da série. Neste caso os N−6 padrões obtidos devem ainda ser divididos para comporem os conjuntos de treinamento, validação e teste.

Na Figura 4.11, *S* contém as amostras da série temporal, com seus valores previamente pré-processados, *L* tem valor de 6 para esta série temporal específica e na parte direita é apresentada uma tabela mostrando a montagem das matrizes **X** e **Y** que representam os padrões obtidos para o treinamento da RNA, onde cada linha de **Y** é a saída desejada associada à sua correspondente linha da matriz **X**, que representa o vetor de regressão ou padrão de entrada. Os subconjuntos de treinamento, validação e teste são extraídos a partir destas matrizes **X** e **Y**, de forma sequencial e em proporções de 50%, 25% e 25%, respectivamente. O número de atrasos *L* para as quatro séries temporais foram de L=12 para as três primeiras e L=2 para a série *Two nonlinear processes*.

As Tabelas 4.1 a 4.4 mostram os resultados numéricos comparando o desempenho do CoACFNNA com outros modelos de RNAs:

- MLP's com números distintos de neurônios ocultos;
- Rede construtiva *Cascade Correlation* (*CasCorr*), descrita no Capítulo 2.

Cada um destes modelos foram executados 30 vezes utilizando os mesmos subconjuntos de treinamento, validação e teste. As redes MLP's foram treinadas usando o mesmo algoritmo quasi-Newton utilizado pelo CoACFNNA. Já a rede neural *CasCorr* é treinada conforme os delineamentos de FAHLMAN & LEBIERE (1990), definindo como 10 o número de neurônios candidatos a serem adicionados na rede a cada iteração. As estatísticas mostradas nas Tabelas 4.1 a 4.4 são média (*mean*), desvio padrão (*std*), valor mínimo (*min*) e valor máximo (*max*) de: Erros de predição junto aos subconjuntos de treinamento, validação e teste, número de neurônios ocultos, número de conexões sinápticas utilizadas e tempo computacional demandado. Os erros de predição correspondem ao MRPE (*mean relative percentual error*). As simulações desta seção foram realizadas num notebook com CPU Intel CORE i7 Q740 de 1,73Ghz e 4Gb de memória RAM.

Por sua vez, as Figuras 4.12 a 4.15 mostram os resultados gráficos do CoACFNNA para uma das 30 execuções feitas. No caso, escolheu-se a solução com o menor erro de predição junto ao subconjunto de *teste*. As informações contidas nestas figuras estão distribuídas da seguinte forma:

- Na parte superior-esquerda das figuras, um gráfico mostra cada procedimento ou passo do fluxograma da Figura 4.2 do algoritmo, isto no eixo horizontal. Já o eixo vertical-esquerdo mostra o valor do erro *E* formando a curva de cor preta. O eixo vertical-direito mostra valores em unidades associadas às barras coloridas (uma barra a cada iteração) que representam a *adição de um neurônio* (barra vermelha), *adição de conexões* (barra verde), *eliminação de conexões* (barra azul), *eliminação de um neurônio* (barra ciano) e a barra de cor amarela (oposta a cada barra colorida) mostra o tempo gasto em segundos (sec.) ou minutos (min.) para realizar cada procedimento.
- Na parte superior-direita das figuras, ilustra-se a arquitetura da rede resultante da execução do algoritmo CoACFNNA. Os neurônios são representados por pequenas circunferências e arcos indicam a conexão entre os neurônios. É possível, assim,

visualizar a topologia resultante, sendo que em cada caso o número de camadas e de neurônios por camada depende das demandas específicas da aplicação. A camada com os neurônios de entrada é a primeira (da esquerda para a direita), a camada com os neurônios de saída é a última e os neurônios ocultos (caso existam) podem compor uma ou várias camadas entre as camadas de entrada e saída. Na ausência de camadas ocultas, o mapeamento implementado pela ACFNN é linear. As cores das conexões obedecem ao valor do seu peso sináptico resultante. Assim, os valores absolutos máximo (max|W|) e mínimo (min|W|) dos pesos (W) são mostrados abaixo do desenho da rede e as cores obedecem a um mapeamento em que o valor mínimo assume cor azul e o valor máximo assume cor vinho. A mediana dos valores absolutos dos pesos (med|W|) também é mostrada.

- Na parte inferior das figuras, são apresentados três gráficos consecutivos correspondendo ao desempenho da rede junto aos subconjuntos de treinamento, validação e teste. As curvas em azul de cada gráfico correspondem à série original e as curvas em vermelho à predição. O MRPE é mostrado no título de cada gráfico.

A Tabela 4.1 mostra os resultados numéricos para a série *Clothing store*, onde o CoACFNNA conseguiu o melhor desempenho (baseados nos erros de validação e teste). Além disso, gerou as redes mais parcimoniosas com um número médio de neurônios requeridos de 0,2, sendo 7,5 o número médio de conexões. Este experimento revela a capacidade de adaptação do CoACFNNA junto a problemas de baixa complexidade. Cabe mencionar que a grande maioria das soluções foram redes puramente lineares, descartando o uso de neurônios ocultos na sua arquitetura.

Um aspecto de grande relevância é que o CoACFNNA é capaz de realizar implicitamente um processo de seleção de variáveis, ou seja, acaba utilizando apenas um subconjunto das entradas que compõem a janela de atrasos. Isto pode ser verificado olhando o número médio de conexões, sendo que a maioria das soluções não utilizou o total de variáveis de entrada (12 atrasos da série como número de entradas). As MLP's não tiveram bom desempenho por se tratarem de modelos não-lineares perante um problema que pode ser adequadamente resolvido com base em um preditor linear. A rede neural

CasCorr também convergiu para modelos de redes lineares, como produzido pelo CoACFNNA.

Tabela 4.1 – Resultados comparativos para a série *Clothing store*.

MODELOS	*Clothing store*						
	Estatísticas	E, Treinamento	E, Validação	E, Teste	No, Neurônios	No, Conexões	Tempo
MLP 1hn	*mean ± std*	9,61 ± 2,1	10,95 ± 2,87	12,2 ± 5,12	1	15	0,1 ± 0,05
	min	7,36	6,93	5,95	1	15	0,01
	max	16,23	15,88	23,82	1	15	0,18
MLP 2hn	*mean ± std*	7,31 ± 2,48	11,22 ± 2,68	14,84 ± 5,52	2	29	0,12 ± 0,03
	min	4,31	5,76	4,15	2	29	0,03
	max	12,77	16,07	28,54	2	29	0,16
MLP 3hn	*mean ± std*	6,58 ± 1,85	10,12 ± 2,04	14,59 ± 8,3	3	43	0,37 ± 0,1
	min	3,53	5,48	6,44	3	43	0,13
	max	10,93	13,84	46,93	3	43	0,55
MLP 4hn	*mean ± std*	7,61 ± 3,28	11,31 ± 2,47	14,67 ± 7,38	4	57	0,41 ± 0,16
	min	2,77	7,43	6,58	4	57	0,02
	max	14,62	15,88	40,75	4	57	0,62
MLP 5hn	*mean ± std*	6,93 ± 2,48	10,41 ± 2,3	14,26 ± 6,22	5	71	0,48 ± 0,13
	min	3,78	7,34	7,46	5	71	0,08
	max	15,58	14,89	40,35	5	71	0,60
CasCorr	*mean ± std*	7,65 ± 0,19	10,06 ± 0,58	10,83 ± 1,43	0,63 ± 0,85	22,1 ± 12,42	3,48 ± 1,93
	min	7,10	8,83	6,98	0	13	1,87
	max	7,77	10,47	13,47	3	58	8,78
CoACFNNA	*mean ± std*	9,84 ± 2,91	6,39 ± 1,16	5,09 ± 1,62	0,2 ± 0,48	7,5 ± 7,31	3,28 ± 5,31
	min	5,96	5,07	2,80	0	2	0,41
	max	17,85	9,27	8,89	2	34	29,64

A Figura 4.12 apresenta os resultados gráficos para a série temporal *Clothing store*. O gráfico da esquerda mostra que apenas a primeira iteração (procedimento) da metodologia foi necessária (bem sucedido). Nesta primeira iteração (procedimento de *cálculo do NMI e adição de primeira conexão*) é calculada a informação mútua (NMI) dos neurônios de entrada e treinada a rede estabelecendo apenas a conexão do neurônio com maior NMI (neurônio da entrada 12). Nas iterações seguintes, nenhum outro procedimento teve sucesso e o algoritmo convergiu conforme estabelecido pelo seu critério de parada. Em consequência, a rede resultante (e que levou ao menor erro junto ao conjunto de teste) não possui nenhum neurônio oculto e apenas duas conexões, a da entrada 12 com o neurônio de saída e a conexão da entrada do *bias*, que não é mostrada no gráfico, mas é contabilizada nos resultados numéricos da Tabela 4.1, totalizando duas conexões (valor mínimo de conexões). O restante das entradas foi descartada. Olhando nos valores mínimo, mediana e máximo, dos pesos sinápticos (sem contar as conexões da entrada *bias*), observa-se

coincidência por ter apenas uma conexão. Os erros de predição foram de 8,06 para treinamento, 5,12 para validação e 2,80 para teste. O tempo computacional gasto pelo CoACFNNA foi de 1,27 segundos.

O modelo de inferência auto-regressivo linear associado com a rede produzida pelo CoACFNNA é apresentado na equação (4.11).

$$y_t = a_0 + a_{12} y_{t-12} , \tag{4.11}$$

onde, a_0 e a_{12} correspondem aos pesos sinápticos da entrada de bias e da entrada nº 12: y_{t-12}.

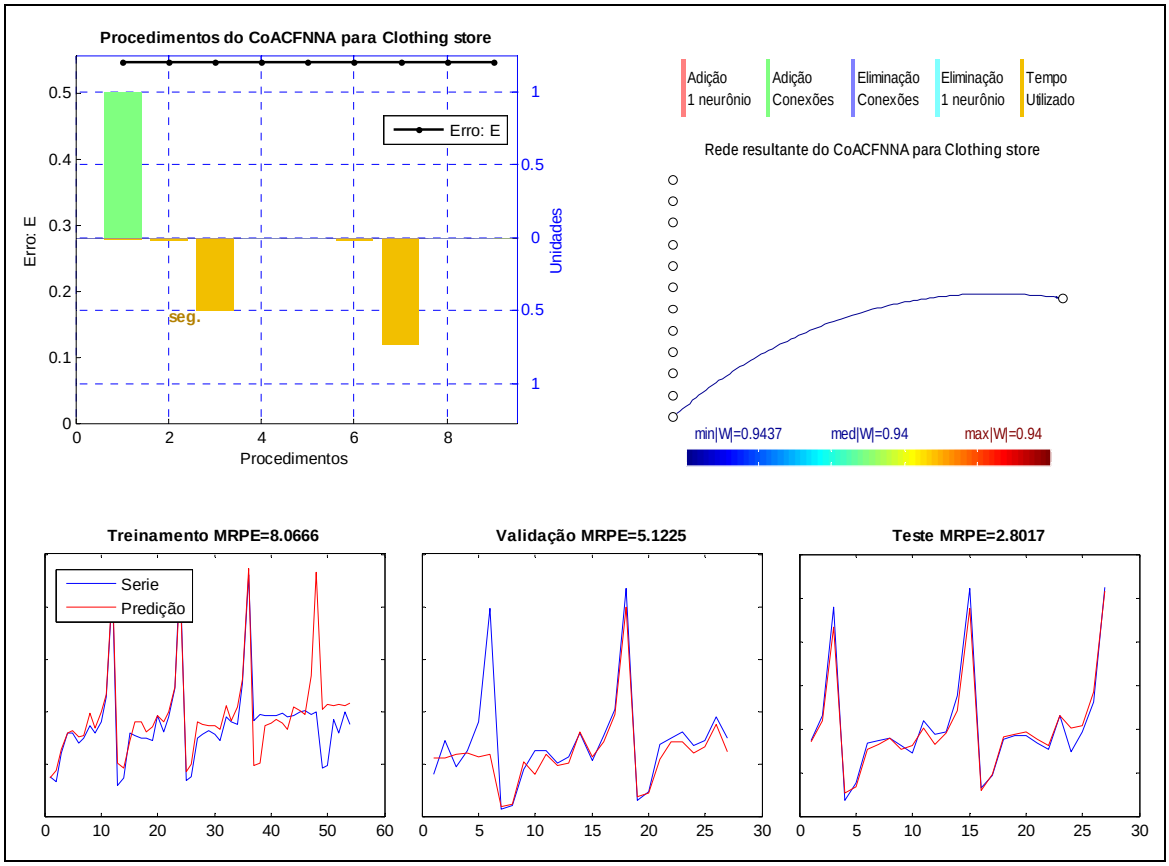


Figura 4.12 – Solução com o menor erro de *teste* produzida pelo CoACFNNA para a série *Clothing store*.

Os resultados numéricos para a série *IPI Durable consumer goods* são mostrados na Tabela 4.2. Nela, os dois algoritmos construtivos (*CasCorr* e CoACFNNA) obtiveram os

menores erros junto aos subconjuntos de validação e teste, com uma ligeira vantagem para o CoACFNNA no subconjunto de teste. As redes MLP's, mesmo tendo apresentado erros de validação não muito distantes dos algoritmos construtivos, tiveram erros de teste mais elevados, o que pode indicar a ocorrência de sobre-ajuste (*over-training*) inclusive quando se observam os erros de treinamento, os quais foram, na maioria deles, menores do que os produzidos pelos algoritmos construtivos. Novamente, a capacidade adaptativa dos algoritmos construtivos, respondendo às demandas específicas de cada aplicação, está retratada neste experimento.

Tabela 4.2 – Resultados comparativos para a série *IPI Durable consumer goods*.

MODELS	*IPI Durable consumer goods*						
	Estatísticas	E, Treinamento	E, Validação	E, Teste	No, Neurônios	No, Conexões	Tempo
MLP 1hn	*mean ± std*	5,59 ± 0,39	3,46 ± 0,21	10,58 ± 2,92	1	15	0,11 ± 0,05
	min	5,31	3,04	5,45	1	15	0,02
	max	6,59	3,70	16,53	1	15	0,24
MLP 2hn	*mean ± std*	5,31 ± 0,5	3,55 ± 0,38	10,23 ± 3,02	2	29	0,16 ± 0,05
	min	4,63	2,99	6,87	2	29	0,03
	max	6,57	5,19	18,03	2	29	0,21
MLP 3hn	*mean ± std*	4,93 ± 0,64	3,41 ± 0,37	10,49 ± 5,4	3	43	0,37 ± 0,15
	min	4,01	2,88	5,02	3	43	0,11
	max	6,13	4,51	28,66	3	43	0,63
MLP 4hn	*mean ± std*	4,53 ± 0,58	3,28 ± 0,31	12,85 ± 6,95	4	57	0,47 ± 0,18
	min	3,79	2,84	6,43	4	57	0,16
	max	5,83	4,30	38,01	4	57	0,75
MLP 5hn	*mean ± std*	4,43 ± 0,53	3,34 ± 0,37	12,37 ± 6,69	5	71	0,6 ± 0,22
	min	3,85	2,88	5,38	5	71	0,21
	max	5,49	4,48	34,48	5	71	0,89
CasCorr	*mean ± std*	5,36 ± 0,01	3,24 ± 0,01	5,15 ± 0,02	0,13 ± 0,43	14,9 ± 6,23	2,59 ± 1,01
	min	5,30	3,20	5,14	0	13	2,12
	max	5,38	3,24	5,26	2	42	6,87
CoACFNNA	*mean ± std*	5,2 ± 0,16	3,03 ± 0,14	4,73 ± 0,51	0,87 ± 0,78	20,93 ± 8,09	9,64 ± 8,83
	min	4,91	2,67	3,86	0	12	1,16
	max	5,57	3,21	6,11	2	37	36,58

Os resultados gráficos associados a uma das 30 execuções para a série *IPI Durable consumer goods* são mostrados na Figura 4.13. Apenas os dois primeiros procedimentos da metodologia do CoACFNNA foram utilizados para esta solução, que convergiu rapidamente para um modelo de rede linear com conexões entre as camadas de entrada e saída. A maioria das entradas foi utilizada, sendo que apenas a entrada 11 foi descartada. O número total de conexões estabelecidas foi de 12, considerando a conexão da entrada *bias*.

Os erros de predição foram os seguintes: treinamento 5,41, validação 3,17 e teste 3,86. O tempo computacional gasto foi de 1,84 segundos.

O modelo de inferência auto-regressivo linear associado com a rede produzida pelo CoACFNNA é apresentado na equação (4.12) a seguir:

$$y_t = a_0 + a_1 y_{t-1} + a_2 y_{t-2} + a_3 y_{t-3} + a_4 y_{t-4} + a_5 y_{t-5} + a_6 y_{t-6} + a_7 y_{t-7} + a_8 y_{t-8} + a_9 y_{t-9} + a_{10} y_{t-10} + a_{12} y_{t-12}. \quad (4.12)$$

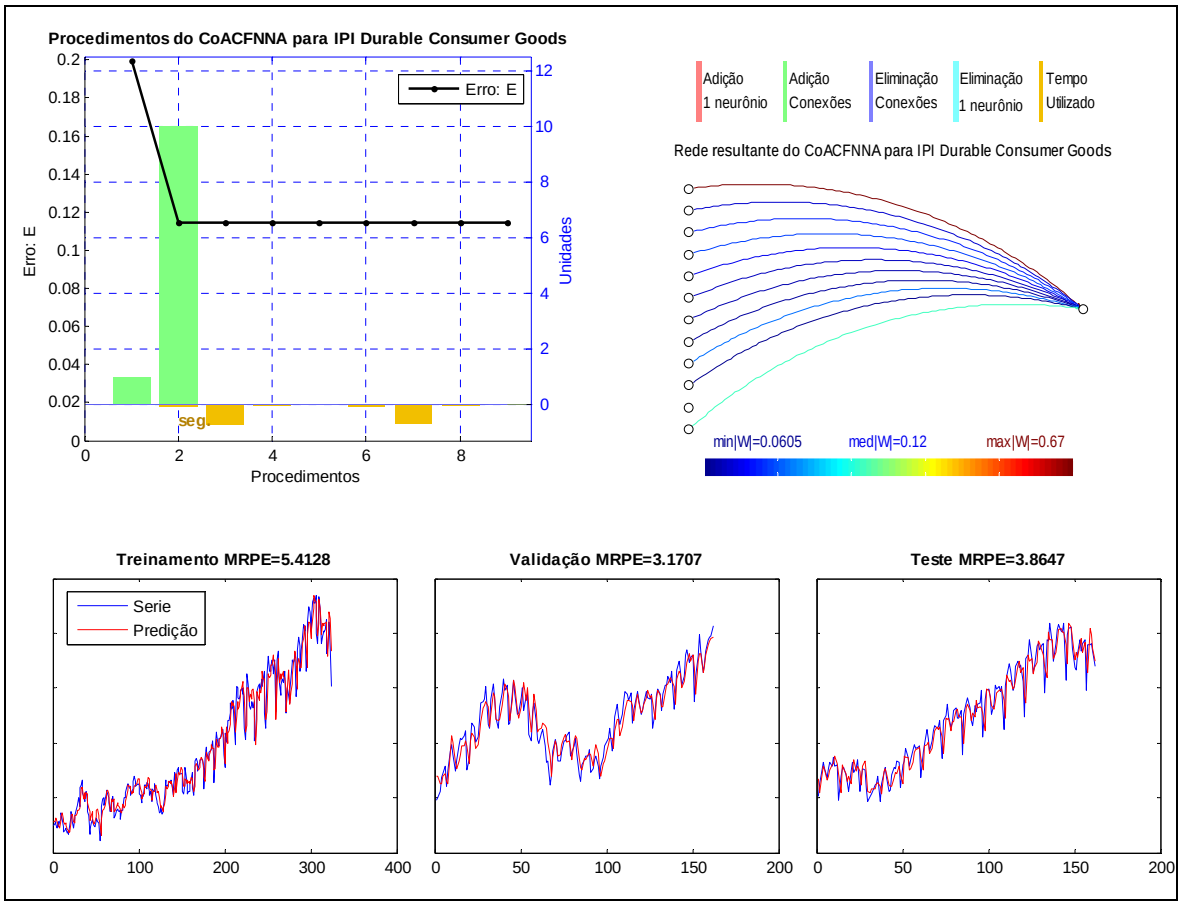


Figura 4.13 – Solução com o menor erro de *teste* produzida pelo CoACFNNA para a série *IPI Durable consumer goods*.

Analisando os resultados para a série *Sunspot* (Tabela 4.3), observa-se que a rede MLP com 5 neurônios ocultos (MLP 5hn) conseguiu uma diferença marcante em relação à rede construtiva *CasCorr* em ambos os subconjuntos de validação e de teste. No entanto, a diferença de desempenho do CoACFNNA nestes subconjuntos foi ainda mais marcante.

Embora o erro de treinamento do CoACFNNA seja maior do que aquele produzido pela MLP 5hn, seus erros médios de validação e teste foram bem inferiores, indicando que as arquiteturas de rede geradas por esta metodologia têm melhore capacidade de generalização, apresentando inclusive menor quantidade de neurônios ocultos e conexões.

Tabela 4.3 – Resultados comparativos para a série *Sunspot*.

MODELOS	*Sunspot*						
	Estatísticas	E, Treinamento	E, Validação	E, Teste	No, Neurônios	No, Conexões	Tempo
MLP 1hn	*mean ± std*	76,4 ± 12,57	88,62 ± 9,31	43,1 ± 4,4	1	15	0,04 ± 0,03
	min	49,54	74,83	35,57	1	15	0,02
	max	111,00	117,44	51,90	1	15	0,14
MLP 2hn	*mean ± std*	72,82 ± 10,7	85,51 ± 9,53	47,83 ± 29,18	2	29	0,13 ± 0,03
	min	51,90	68,67	32,93	2	29	0,08
	max	89,17	100,34	197,31	2	29	0,22
MLP 3hn	*mean ± std*	68,56 ± 12,9	80,3 ± 10,75	42,38 ± 13,51	3	43	0,34 ± 0,12
	min	46,71	47,59	32,67	3	43	0,14
	max	96,94	104,38	111,08	3	43	0,51
MLP 4hn	*mean ± std*	65,43 ± 21,09	80,58 ± 13,42	43,83 ± 19,11	4	57	0,43 ± 0,15
	min	43,55	57,49	29,31	4	57	0,08
	max	163,79	133,49	140,62	4	57	0,63
MLP 5hn	*mean ± std*	65,57 ± 12,18	80,25 ± 9,95	40,29 ± 4,72	5	71	0,6 ± 0,14
	min	45,44	57,47	33,64	5	71	0,13
	max	108,43	104,62	53,00	5	71	0,75
CasCorr	*mean ± std*	73,37 ± 4,47	95,13 ± 2,21	45,38 ± 1,18	1,23 ± 1,87	32,1 ± 31,19	4,8 ± 4,39
	min	59,04	92,69	41,36	0	13	1,97
	max	78,11	102,65	49,15	8	153	21,37
CoACFNNA	*mean ± std*	67,07 ± 9,68	62,75 ± 6,28	29,56 ± 2,62	0,43 ± 0,68	12,13 ± 7,71	4,71 ± 5,1
	min	47,78	48,62	25,23	0	4	0,55
	max	85,27	74,09	33,85	2	31	18,99

Os resultados gráficos para a execução com o menor erro de teste relativos à série *Sunspot* (Figura 4.14) mostram um maior número de procedimentos realizados pelo CoACFNNA. Mas os que tiveram sucesso foram somente os que envolviam adição e eliminação de conexões, consequentemente a rede resultante é de natureza linear. A motivação da implementação do mecanismo de relaxação do erro no CoACFNNA é verificada na iteração 8, onde o algoritmo decide aceitar a eliminação de uma conexão em detrimento da qualidade do erro *E*. Esta decisão é recompensada logo na iteração 10, pois após a adição de 4 conexões o erro experimenta uma melhora significativa a um nível melhor do que aquele existente por ocasião da iteração 8, mostrando a utilidade deste mecanismo e a sua capacidade de promover a fuga de eventuais mínimos locais. A rede gerada nesta simulação é linear e usou apenas 6 das 12 variáveis de entrada, totalizando 7

conexões sinápticas. Os erros de predição foram de 59,36 para treinamento, 58,13 para validação e 25,23 para teste. O tempo computacional gasto foi de 3,04 segundos.

O modelo de inferência auto-regressivo linear associado com a rede produzida pelo CoACFNNA é apresentado na equação (4.13).

$$y_t = a_0 + a_1 y_{t-1} + a_2 y_{t-2} + a_5 y_{t-5} + a_8 y_{t-8} + a_9 y_{t-9} + a_{10} y_{t-10} \,. \qquad (4.13)$$

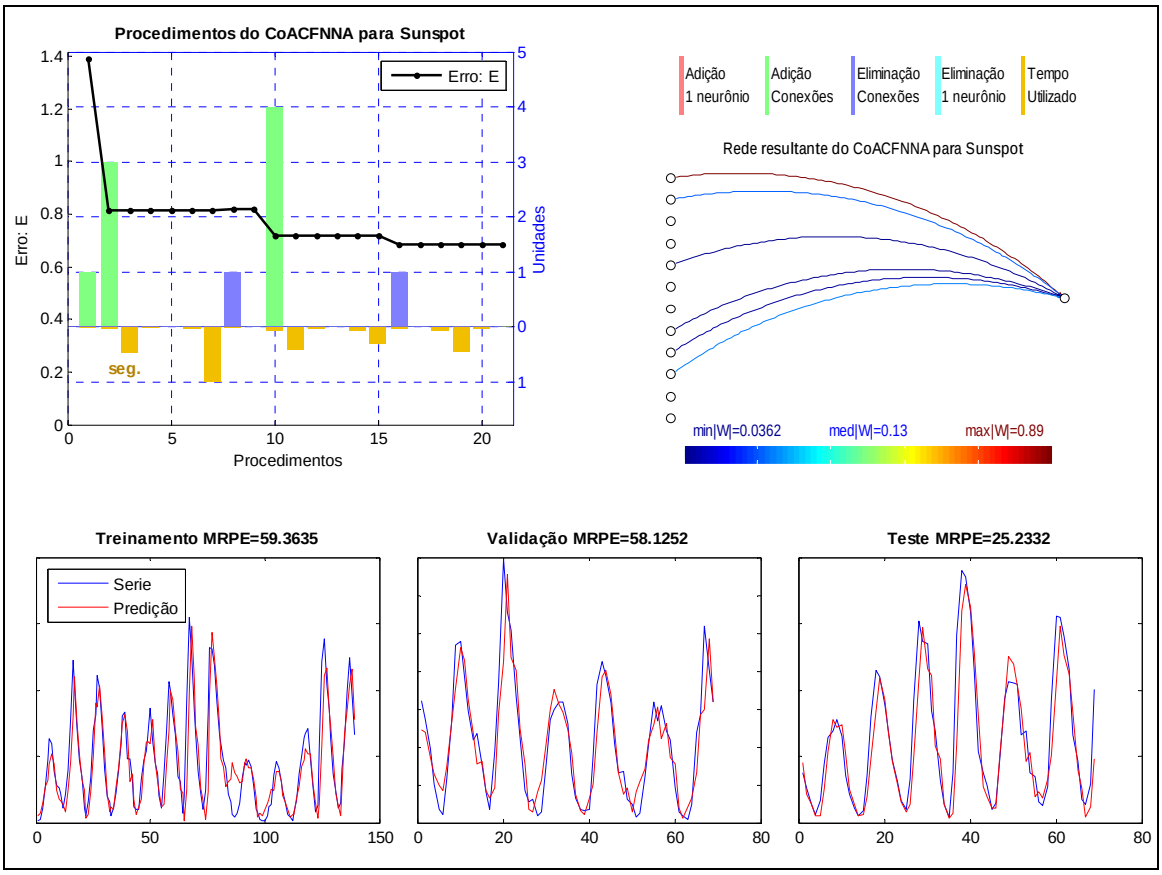


Figura 4.14 – Solução com o menor erro de *teste* produzida pelo CoACFNNA para a série *Sunspot*.

As três primeiras séries temporais, representando problemas reais de predição, mostraram requerer apenas redes neurais lineares. Já para o caso da quarta série temporal, a *Two nonlinear processes*, as redes terão que possuir neurônios ocultos na sua arquitetura para poderem expressar desempenhos competentes. A Tabela 4.4 mostra os resultados,

confirmando que o CoACFNNA gerou redes com 6,1 neurônios ocultos em média e obteve os melhores resultados de predição. Por outro lado, a melhor configuração de rede MLP foi a que apresentava 10 neurônios ocultos. A *CasCorr* mostrou-se pouco competente para este problema. A explicação pode ser encontrada observando o número de neurônios ocultos médios que foi de apenas 2,87, revelando claramente que este algoritmo convergiu prematuramente e não conseguiu incorporar a quantidade de neurônios que o problema requeria. Uma hipótese alternativa para explicar este desempenho inferior da rede neural *CasCorr* pode estar fundamentada no fato de que este problema parece requerer redes com neurônios ocultos formando camadas e não em forma de cascata, como é a única possibilidade existente para a rede neural *CasCorr*.

Tabela 4.4 – Resultados comparativos para a série *Two nonlinear processes*.

MODELOS	*Two nonlinear processes*						
	Estatísticas	E, Treinamento	E, Validação	E, Teste	No, Neurônios	No, Conexões	Tempo
MLP 2hn	*mean ± std*	128,48 ± 13,18	237,76 ± 33,34	156,76 ± 26,22	2	9	0,1 ± 0,05
	min	107,04	106,84	110,08	2	9	0,04
	max	147,90	261,41	208,95	2	9	0,26
MLP 6hn	*mean ± std*	98,18 ± 6,61	118,16 ± 26,31	106,16 ± 15,2	6	25	0,7 ± 0,16
	min	85,60	71,32	86,01	6	25	0,33
	max	110,84	165,29	147,94	6	25	0,92
MLP 10hn	*mean ± std*	84,66 ± 5,55	81,09 ± 17,69	93,78 ± 6,8	10	41	1,77 ± 0,24
	min	72,42	53,33	84,57	10	41	0,62
	max	96,48	122,21	112,26	10	41	1,93
MLP 14hn	*mean ± std*	85,16 ± 4,89	81,88 ± 17,04	96,81 ± 6,02	14	57	1,83 ± 0,21
	min	76,14	50,76	86,96	14	57	1,03
	max	92,95	121,31	109,29	14	57	2,08
MLP 18hn	*mean ± std*	82,29 ± 4,58	79,16 ± 14,32	101,76 ± 5,61	18	73	1,9 ± 0,24
	min	72,27	54,76	86,23	18	73	0,81
	max	91,01	108,52	111,43	18	73	2,12
CasCorr	*mean ± std*	129,01 ± 10,83	200,44 ± 27,97	158,68 ± 37,21	2,87 ± 2,33	19,77 ± 16,7	6,3 ± 4,4
	min	110,60	165,62	98,85	0	3	1,34
	max	153,27	322,95	218,97	8	63	17,06
CoACFNNA	*mean ± std*	86,71 ± 8,69	81,19 ± 27,33	87,31 ± 6,37	6,1 ± 1,32	36,13 ± 9,34	107,36 ± 59,46
	min	71,38	46,59	72,55	3	15	23,13
	max	112,46	159,57	99,42	9	55	290,82

A Figura 4.15 apresenta a execução com o melhor erro de predição junto ao subconjunto de teste para a série *Two nonlinear processes*. Por se tratar de uma série não-linear, a rede gerada recebeu vários neurônios ocultos e conexões, convergindo para apenas 5 neurônios ocultos distribuídos em 3 camadas. No gráfico, observa-se como o erro *E* é diminuído a cada adição de neurônios. O número de conexões final foi de 31 e os erros de predição de

71,38 para treinamento, 84,64 para validação e 72,55 para teste. O tempo gasto foi de 45,44 segundos. O modelo de inferência auto-regressivo associado com a rede produzida pelo CoACFNNA é do tipo não-linear e mais complexo de ser representado em uma equação.

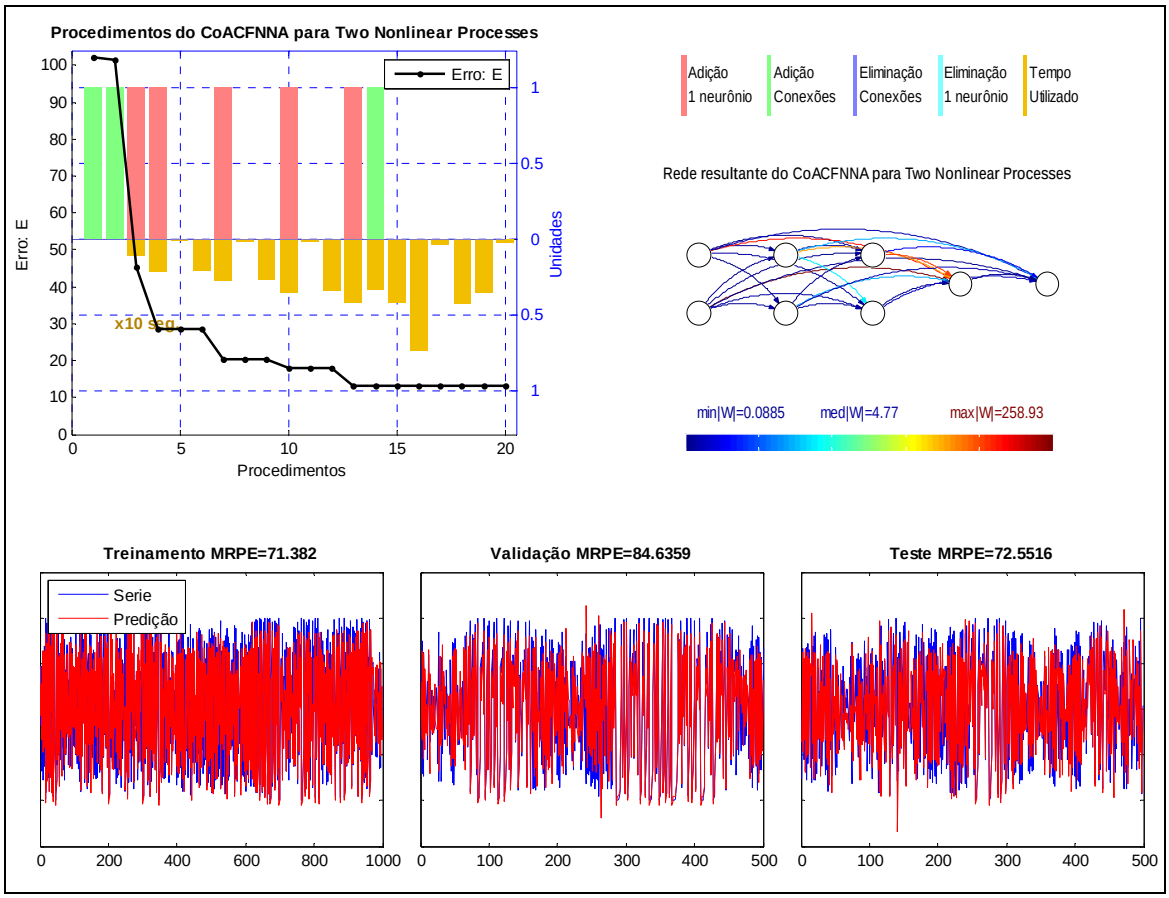


Figura 4.15 – Solução com o menor erro de *teste* produzida pelo CoACFNNA para a série *Two nonlinear processes*.

A seguir, as Figuras 4.16 e 4.17 mostram resultados para dois casos extremos a partir das 30 execuções feitas com o CoACFNNA e a série *Two nonlinear processes*. Referem-se às execuções que geraram redes com a maior e menor quantidade de neurônios ocultos e que, coincidentemente, demandaram o maior e menor tempo de processamento. Para o primeiro caso, se requereu até 45 iterações do algoritmo, sendo que a rede gerada conta com 9 neurônios ocultos distribuídos em 3 camadas, 55 conexões, com erros de treinamento no valor de 79,28, de validação no valor de 59,55, e de teste no valor de 82,74.

O tempo computacional foi de 290,81 segundos. Para o segundo caso (Figura 4.17), foram 13 iterações do algoritmo, 3 neurônios ocultos em duas camadas, 15 conexões, erros de treinamento no valor de 112,46, de validação no valor de 159,57 e de teste no valor de 90,90. O tempo computacional foi de 23,12 segundos.

Além de se mostrar uma série temporal bastante desafiadora, por envolver o chaveamento de dois processos distintos, este último experimento permite verificar, com mais ênfase que os três experimentos anteriores, que, mesmo com o emprego do mecanismo de relaxação do erro, o CoACFNNA ainda assim pode convergir para topologias de redes neurais que representam mínimos locais. A razão para que as decisões de inserção e poda de neurônios podem ser distintas a cada execução do algoritmo está na forma aleatória com que se inicializam as conexões recém-inseridas, o que pode representar um ponto de investigação futura, visando tornar o algoritmo ainda mais consistente e robusto às demandas de cada aplicação. Cabe mencionar que os experimentos envolvendo problemas de classificação de padrões, a serem apresentados no Capítulo 5, exibem mais consistência nos resultados, com menos variação nas topologias resultantes.

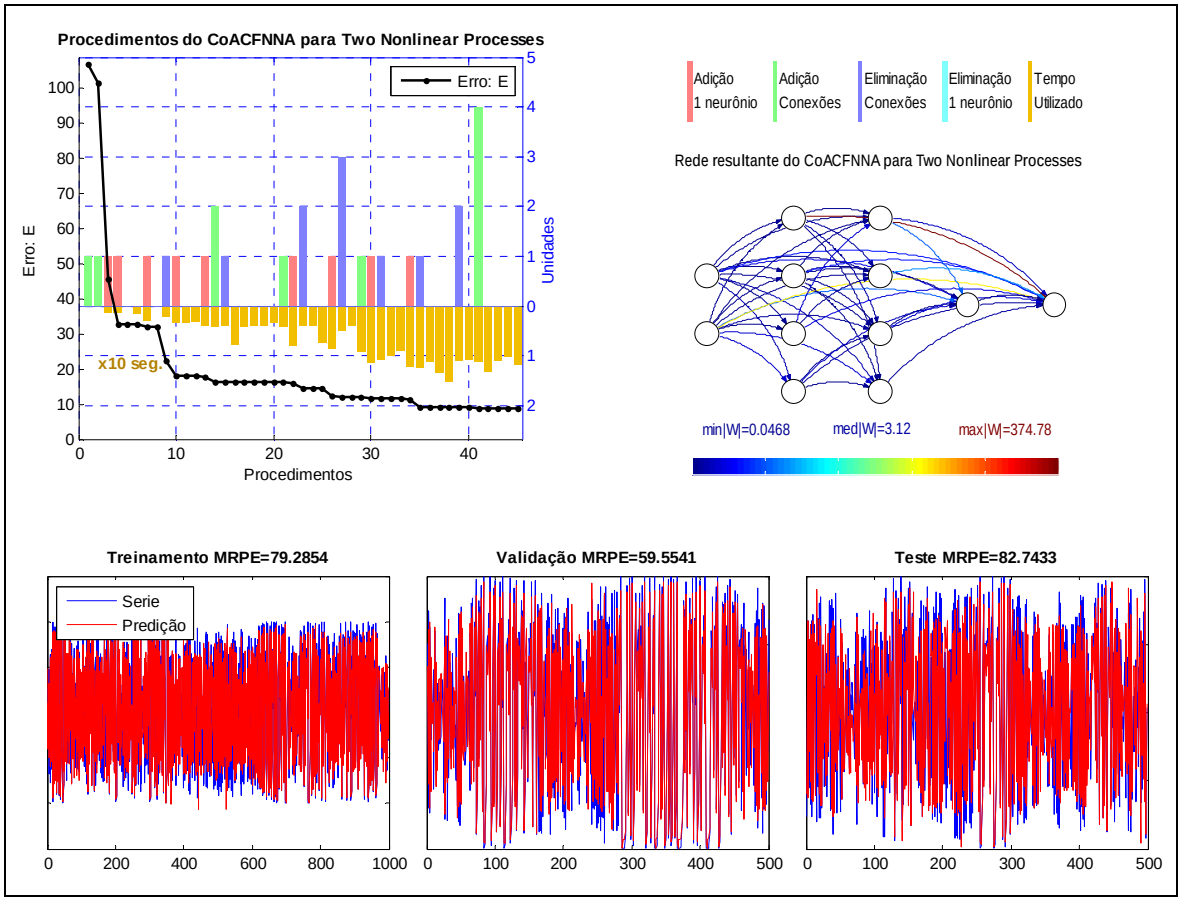


Figura 4.16 – Solução com o maior número de neurônios e conexões produzida pelo CoACFNNA para a série *Two nonlinear processes*.

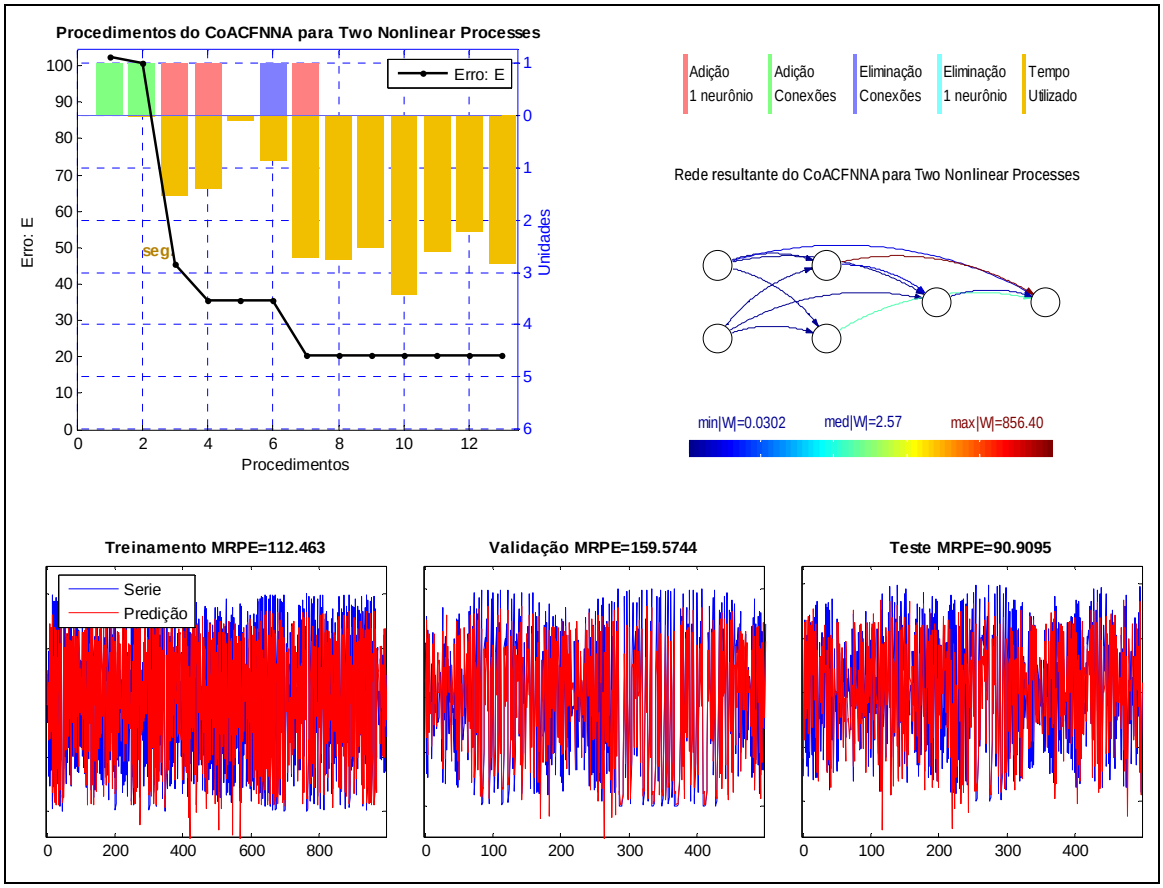


Figura 4.17 – Solução com o menor número de neurônios e conexões produzida pelo CoACFNNA para a série *Two nonlinear processes*.

4.3.1. Análise de sensibilidade para o parâmetro *SEr_max*

As Figuras 4.18 e 4.19 apresentam resultados de execução do CoACFNNA visando realizar um análise de sensibilidade do parâmetro *SEr_max* (ver seção 4.2.8). Para isso, foram escolhidos dois problemas, o *Clothing store* e *Two nonlinear processes*, os quais foram executados 30 vezes com valores distintos de *SEr_max* = [1,00 1,05 1,10 1,15 1,20 1,25 1,30]. Nas figuras: o gráfico da esquerda mostra os erros médios de treinamento, validação e teste; o gráfico central mostra a média do número de neurônios e de conexões; e o gráfico à direita mostra a média do tempo computacional, do número de iterações do erro mínimo e do término de execução do algoritmo (parada).

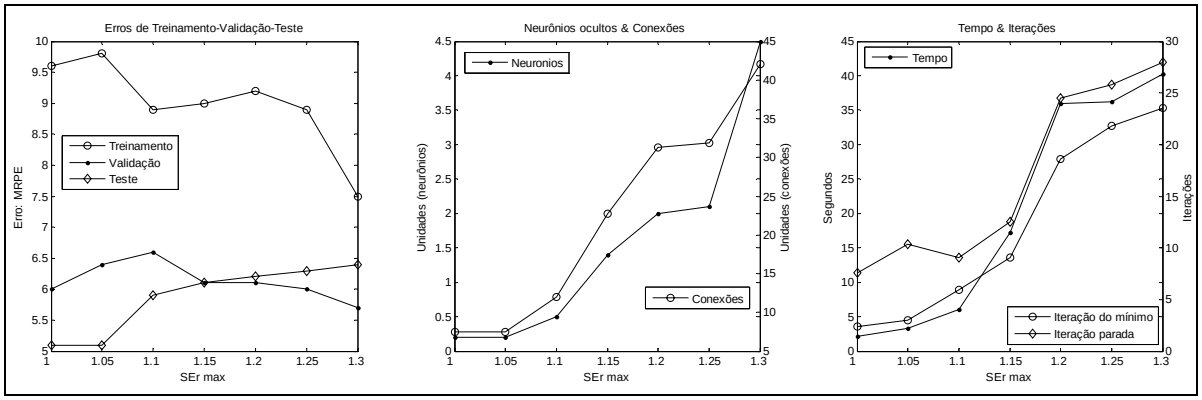


Figura 4.18 – Análise de sensibilidade do *SEr_max* do CoACFNNA para o problema *Clothing store*.

Para ambos os problemas observa-se uma correlação direta entre o valor do *SEr_max* e o número de neurônios, conexões, tempo computacional e o número de iterações do algoritmo. O desempenho das redes em termos de minimização dos erros (gráfico à esquerda) não sofreu variações consideráveis. No *Clothing store*, observa-se um leve aumento do erro de teste e diminuição dos erros de treinamento e validação com valores altos de *SEr_max*, indicando um possível caso de sobre-ajuste, pois as redes vão ganhando mais neurônios e conexões. No *Two nonlinear processes*, verificou-se que o *SEr_max* com valor de 1,00 (anulação do mecanismo de relaxação do erro) levou a resultados menos interessantes. Logo, evidencia-se a necessidade de se considerar um certo grau de relaxação do erro visando escapar de mínimos locais. Valores recomendados para *SEr_max* podem ser estabelecidos no intervalo [1,05; 1,10].

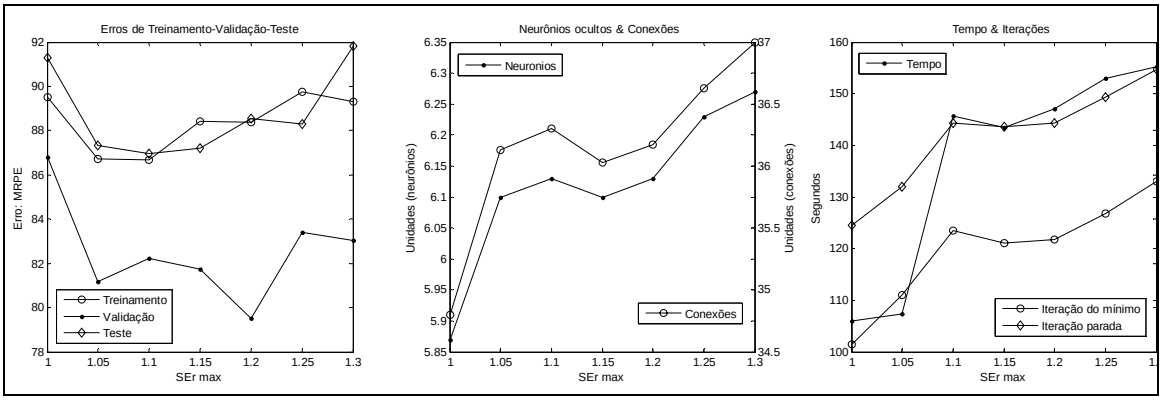


Figura 4.19 – Análise de sensibilidade do *SEr_max* do CoACFNNA para o problema *Two nonlinear processes*.

4.4. Considerações finais do capítulo

Já se sabia, na literatura, que as duas primeiras séries testadas (*Clothing store* e *IPI Durable consumer goods*) eram séries de natureza linear (ZHANG & QI, 2005) e o algoritmo CoACFNNA mostrou-se competente para se adaptar às exigências mínimas destas séries, construindo propostas de redes lineares e descartando entradas pouco relevantes. Já a natureza da terceira série temporal (*Sunspot*) era uma incógnita. No entanto, o CoACFNNA convergiu para um modelo linear e com bom desempenho perante as redes não-lineares MLP's. A quarta série testada exibia maior flexibilidade por ser um produto da composição de duas séries diferentes, ambas não-lineares, e o algoritmo proposto nesta tese conseguiu obter soluções de boa qualidade e parcimônia, embora com alguma variabilidade alta junto às execuções com resultados extremos.

Outro ponto a destacar é a capacidade dos algoritmos de aprendizado construtivo para gerar soluções parcimoniosas e melhor ajustadas às demandas de cada aplicação. Adicione-se a esta afirmação o baixo número de parâmetros que precisam ser especificados pelo usuário para permitir a execução desses algoritmos. No caso da rede neural *CasCorr*, o único parâmetro a ser definido pelo usuário foi o número de neurônios ocultos candidatos a serem treinados a cada iteração para fazer parte da rede, que foi de 10 para os 4 problemas de predição. No caso do CoACFNNA, o único parâmetro dependente do usuário foi a taxa de folga do erro (*SEr_max*) que foi definido com o valor de 1,05 para os 4 problemas.

Os dois aspectos mais relevantes ao se analisarem os resultados produzidos pelo CoACFNNA são a sua capacidade implícita de seleção de variáveis e o seu potencial de detecção e exploração de aspectos lineares presentes nos mapeamentos a serem sintetizados, ambos derivados diretamente da análise dos padrões disponíveis para treinamento.

O Capítulo 5 traz uma maior quantidade de teste comparativos para o CoACFNNA em problemas de classificação de dados e redução de dimensão.

Capítulo 5

CoACFNNA em classificação de dados e redução de dimensão

Resumo: Este capítulo tem dois propósitos: (*i*) Avaliar o desempenho do CoACFNNA na resolução de problemas de classificação de dados; e (*ii*) Apresentar as modificações no CoACFNNA padrão para permitir a sua aplicação junto a problemas de redução de dimensão. Para ambos os casos, foram selecionados problemas de natureza real e artificial. No caso dos problemas de classificação de dados, os resultados do CoACFNNA são confrontados com: (1) Modelos tradicional de redes neurais MLP com uma e duas camadas ocultas; (2) Mistura heterogênea de especialistas; (3) Modelo construtivo *Cascade Correlation*; e (4) EPNet (*Evolutionary Programming Network*). Para os problemas de redução de dimensão, os resultados obtidos via o CoACFNNA foram confrontados com aqueles produzidos por: (1) PCA; (2) Algoritmo de Sammon; e (3) CCA (*Curvilinear Component Analysis*).

5.1 Considerações iniciais

Com a finalidade de avaliar o desempenho do CoACFNNA junto a problemas de classificação e de redução de dimensão, foram escolhidos um elenco de problemas, tanto de natureza real como artificial. Os problemas reais possuem desafios como, por exemplo, *outliers* (observações não esperadas), ruído nos dados, classes superpostas, ou poucas amostras disponíveis (YAO & LIU, 1997). Por outra parte, os problemas artificiais foram

escolhidos por representarem situações específicas e, assim, exigirem habilidades como alta flexibilidade, ou seja, capacidade de sintetizar mapeamentos de entrada-saída dotados de forte não-linearidade.

O mérito de cada proposta de solução será medido com base nos seguintes critérios:

- Melhor desempenho (medido via uma função-objetivo) junto aos subconjuntos de validação e teste (dados que não foram utilizados no processo de aprendizado); e
- Maior parcimônia do modelo obtido, no sentido deste empregar uma menor quantidade de recursos computacionais (tempo de processamento e memória). No contexto de RNAs, isto consistiria em um menor número de neurônios e conexões na arquitetura da rede neural.

5.2 Problemas de classificação de dados

Foram selecionados oito problemas de classificação de dados, os quais estão compostos por números distintos de variáveis de entrada (atributos) e de saída (classes). Três deles são problemas artificiais, usados frequentemente como *benchmarks* em problemas de agrupamento e também de classificação de dados (ver Figura 5.1), sendo caracterizados por requererem muita flexibilidade dos modelos. Os outros cinco problemas são de natureza real, foram extraídos do UCI *Machine Learning Repository* (FRANK & ASUNCION, 2010) e pertencem às áreas da biologia, indústria e saúde. Em seguida, foram considerados outros dois problemas de classificação, compostos por um elevado número de amostras, para avaliar a escalabilidade do CoACFNNA.

A Tabela 5.1 mostra aspectos principais associados aos oito problemas iniciais.

Tabela 5.1 – Principais aspectos dos oito problemas de classificação de dados.

Problemas	Tipo	No. Entradas	No. Classes	No. amostras por classe	Total de amostras
1 *Two Spirals 2D*	Artificilal	2	2	C1=472, C2=472	944
2 *Two Donuts 3D*	Artificial	3	2	C1=500, C2=500	1000
3 *Three Spirals 3D*	Artificial	3	3	C1=590, C2=590, C3=590	1770
4 *Iris*	Real - Biologia	4	3	C1=50, C2=50, C3=50	150
5 *Wine*	Real - Indústria	13	3	C1=59, C2=71, C3=48	178
6 *Breast Cancer Wisconsin*	Real - Saúde	9	2	C1=444, C2=239	683
7 *Breast Tissue*	Real - Saúde	6	6	C1=21, C2=15, C3=18, C4=16, C5=14, C6=22	106
8 *Wall-Following Robot Navigation*	Real - Indústria	4	4	C1=2205, C2=826, C3=2097, C4=328	5456

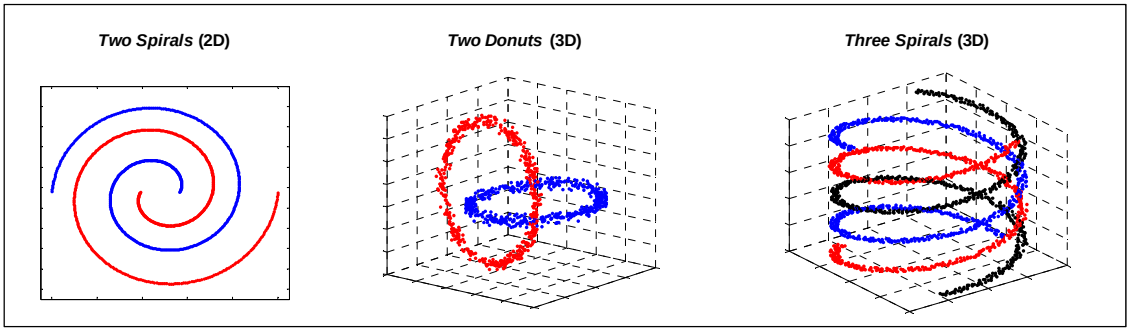


Figura 5.1 – Ilustração dos três problemas artificiais de classificação de dados.

Os experimentos de classificação estão divididos da seguinte forma:

- Primeiro estudo de caso: o CoACFNNA foi comparado com modelos tradicionais, tais como MLPs, MHEs e *Cascade Correlation* (*CasCorr*).
- Segundo estudo de caso: o CoACFNNA foi comparado com o EPNet (YAO & LIU, 1997), que é baseado em computação evolutiva.
- Terceiro estudo de caso: o CoACFNNA foi empregado junto a problemas com maior quantidade de amostras, tendo por finalidade avaliar o incremento em tempo computacional do algoritmo. Para isto, foram utilizados outros dois problemas do

UCI - *Machine Learning Repository* (o *MAGIC Gamma Telescope* e o *StatLog–Shuttle*) que serão melhor apresentados na Seção 5.2.3.

Para todos os problemas de classificação de dados, foram levadas em conta as seguintes condições experimentais:

- Os algoritmos foram implementados utilizando o ambiente de programação Matlab e rodados num computador com processador Intel Core 2 Quad 32 bits de 2,83 MHz e com 4 Gb de memória RAM, com exceção do EPNet (no segundo estudo).
- Os resultados numéricos de desempenho dos modelos apresentados nas Tabelas 5.3, 5.4 e 5.5 foram produzidos a partir de 30 execuções independentes de cada modelo.
- Para o CoACFNNA, o parâmetro a ser definido pelo usuário é o *SEr_max* (descrito na subseção 4.2.8) e cujo valor inicial foi definido como *SEr_max* = 1,10 para todos os problemas de classificação.
- Os classificadores apresentam tantas saídas quanto o número de classes, sendo que a classe é definida pela saída de maior valor numérico.
- Durante o processo de treinamento, a saída desejada é um vetor binário que contém apenas um bit de valor 1 (correspondente à classe), com os demais bits assumindo o valor 0.

5.2.1. Primeiro estudo de caso: CoACFNNA e modelos tradicionais

Os modelos que fazem parte deste primeiro estudo de caso são os seguintes:

- Redes MLPs com uma e duas camadas de neurônios ocultos;
- Mistura Heterogênea de Especialistas (MHE);
- O algoritmo construtivo *Cascade Correlation* (FAHLMAN & LEBIERE, 1990).

Cabe indicar que todos os modelos do tipo MLP e de Mistura Heterogênea de Especialistas utilizam o mesmo algoritmo de treinamento (baseado no método quasi-Newton e descrito na Subseção 4.1.2) empregado pelo CoACFNNA.

Os principais parâmetros associados a cada modelo classificador são definidos a seguir:

- Para a MLP com uma camada oculta, o parâmetro é o número de neurônios nesta camada. Para tal, várias configurações foram consideradas ao variar este número com os seguintes valores: 1, 2, ..., 15, neurônios ocultos.
- Para a MLP com 2 camadas ocultas, foram exploradas várias configurações variando o número de neurônios para ambas as camadas nos valores de 1, 2, ..., 10 e, finalmente, que atendem a alguma restriçao adicional se não toma-se 1-1, os quais são chamados de: MLP2HL-A e MLP2HL-B para cada um dos problemas. A Tabela 5.2 mostra os números de neurônios para cada uma destas duas redes.
- Para os modelos de MHE, a configuração foi determinada com 2 especialistas controlados por uma rede *gating*, para todos os problemas. Os especialistas e a rede *gating* são compostos por redes MLP, de uma camada oculta quando representando modelos não-lineares e sem nenhuma camada oculta quando representando modelos lineares. A escolha dos tipos de especialistas e rede *gating* (lineares ou não-lineares) foi feita via testes considerando as mesmas opções de número de neurônios feita para as redes MLPs de uma camada oculta, para representar os especialistas não-lineares, e redes sem neurônios ocultos para a representação dos especialistas lineares. Ao final, foram escolhidas as duas propostas com melhor desempenho (MHE-A e MHE-B), cujo detalhamento é apresentado na Tabela 5.2. Nesta tabela, o formato da representação mostra o número de neurônios ocultos de cada especialista e rede *gating*, da seguinte forma: "(especialista1,especialista2)*gating*". Caso o modelo seja puramente linear, então mostra-se L (de linear) em vez de valores numéricos. Por exemplo: para o problema Iris com MHE-A, a configuração é "(3,L)L" o que significa que o primeiro especialista é um modelo não-linear MLP com 3 neurônios ocultos, enquanto que o segundo especialista e a rede *gating* são modelos lineares.

- Para o modelo *Cascade Correlation*, a sua implementação se baseou no trabalho de FAHLMAN & LEBIERE (1990). O único parâmetro opcional que pode ser definido pelo usuário refere-se ao número de neurônios ocultos candidatos, a serem treinados e avaliados antes da inserção de um deles na rede, a cada iteração. Foi definido como 10 o número de neurônios candidatos.

O conjunto de dados disponível foi dividido de forma aleatória em 3 subconjuntos: *treinamento* (50% das amostras), *validação* (25% das amostras) e *teste* (25% das amostras). Uma vez definida cada partição das amostras nesses três subconjuntos, a mesma partição foi utilizada em todas as execuções de todos os modelos que fazem parte do estudo comparativo.

Tabela 5.2 – Descrição das MLPs com duas camadas ocultas e das MHEs.

MLP2HL & MHEs	*Two Spirals (2D)*	*Two Donuts (3D)*	*Three Spirals (3D)*	*Iris*	*Wine*	*Breast Cancer Wisconsin*	*Breast Tissue*	*Wall-Following Robot Navigation*
MLP2HL-A	6 - 6	6 - 6	4 - 4	2 - 2	2 - 2	1 - 1	3 - 3	5 - 5
MLP2HL-B	7 - 7	7 - 7	5 - 5	3 - 3	3 - 3	2 - 2	4 - 4	6 - 6
MHE-A	(10,10)7	(6,6)6	(8,8)5	(3,L)L	(L,L)L	(L,L)L	(L,L)L	(11,L)L
MHE-B	(11,11)7	(7,7)6	(9,9)5	(2,2)2	(2,L)L	(L,L)2	(4,L)L	(12,L)L

As Tabelas 5.3 e 5.4 apresentam os resultados numéricos em termos de taxa de classificação correta para os subconjuntos de *Treinamento*, *Validação* e *Teste*. A informação corresponde ao valor médio e desvio-padrão (média ± desvio-padrão), calculados após 30 execuções independentes de cada um dos modelos.

Tabela 5.3 – Taxas de classificação correta para os primeiros 4 problemas.

Modelos	*Two Spirals (2D)*			*Two Donuts (3D)*			*Three Spirals (3D)*			*Iris*		
	Treinamento	*Validação*	*Teste*	*Treinamento*	*Validação*	*Teste*	*Treinamento*	*Validação*	*Teste*	*Treinamento*	*Validação*	*Teste*
MLP 1hn	65,3±2,1	62,0±1,9	64,1±2,4	67,6±4,6	68,0±3,7	70,0±3,8	43,4±2,0	45,8±1,4	37,8±1,6	72,5±6,8	86,8±7,6	63,6±5,4
MLP 2hn	65,0±3,4	60,0±4,0	64,0±3,4	93,1±5,1	92,2±6,3	92,1±5,1	49,1±4,3	51,6±2,5	44,9±4,9	98,4±0,8	96,8±1,1	98,9±1,3
MLP 3hn	68,5±3,4	63,8±4,3	67,2±3,2	97,4±1,9	97,4±2,5	97,0±1,8	64,6±5,8	63,0±4,4	61,8±7,0	98,8±0,6	97,2±0,7	98,5±1,4
MLP 4hn	72,7±4,4	69,4±5,5	70,4±4,6	98,6±0,9	98,6±0,9	98,4±1,3	76,9±5,3	74,0±4,8	75,4±6,5	98,8±0,3	97,2±0,7	99,2±1,3
MLP 5hn	78,1±3,7	75,3±5,9	76,1±3,3	99,3±0,9	99,1±1,0	99,1±1,1	85,7±3,2	83,2±3,3	84,4±3,5	98,7±0,0	96,9±1,0	98,9±1,5
MLP 6hn	79,9±4,9	76,6±6,1	78,0±4,5	99,7±0,6	99,5±0,9	99,5±0,9	94,4±2,2	93,3±2,8	93,7±2,2	98,5±0,4	97,2±0,7	99,0±1,3
MLP 7hn	85,2±6,4	82,7±7,4	82,6±7,0	99,8±0,5	99,8±0,5	99,8±0,5	98,9±1,8	98,3±2,4	98,3±2,4	98,6±0,2	97,0±0,9	98,9±1,3
MLP 8hn	93,0±5,5	90,4±7,0	91,0±6,4	99,8±0,5	99,8±0,6	99,8±0,7	99,8±0,4	99,8±0,5	99,7±0,5	98,7±0,0	97,0±0,9	98,9±1,3
MLP 9hn	96,2±5,9	94,5±6,7	94,1±6,9	100	100	100	99,9±0,4	99,8±0,5	99,8±0,5	98,6±0,2	96,8±1,1	99,0±1,3
MLP 10hn	98,4±3,7	97,5±4,4	97,4±4,3	100	100	100	100	100	100	98,6±0,2	97,2±0,7	98,9±1,3
MLP 11hn	99,2±2,4	98,4±3,0	98,2±3,4	100	100	100	100	100	100	98,5±0,4	97,2±0,7	99,5±1,0
MLP 12hn	99,7±1,2	99,2±2,0	99,1±2,2	100	100	100	100	100	100	98,5±0,4	97,4±0,0	99,3±1,2
MLP 13hn	99,1±4,7	98,9±4,3	98,7±5,5	100	100	100	100	100	100	98,5±0,5	97,2±0,7	99,6±0,9
MLP 14hn	99,9±0,5	99,6±1,2	99,7±1,4	100	100	100	100	100	100	98,4±0,5	97,4±0,0	99,6±0,9
MLP 15hn	100	99,9±0,2	100	100	100	100	100	100	100	98,6±0,3	97,2±0,7	99,5±1,1
MLP2HL-A	100±0,1	99,8±0,4	99,7±0,8	99,8±0,4	99,4±0,9	99,5±0,7	98,2±1,8	97,6±2,5	97,2±2,4	98,5±0,5	96,4±1,3	98±1,4
MLP2HL-B	100±0	100±0,1	100±0,1	100±0	99,9±0,4	100±0,2	100±0,1	100±0,2	99,9±0,3	98,5±0,6	97±0,9	99,1±1,3
MHE-A	99,9±0,2	99,4±0,6	99,6±0,5	99,7±0,5	99,5±0,7	99,3±0,8	100	99,9±0,4	99,8±0,3	99,3±1,5	94,2±2,3	94,1±1,9
MHE-B	100	99,9±0,2	100	100	100	100	100	100	100	99,6±0,6	97,4±1,3	99,2±0,9
CasCorr	97,3±3,5	96,9±3,9	94,8±5,6	99,9±0,2	100	99,8±0,3	99,3±0,8	99,1±0,9	98,3±1,3	96,4±2	96,8±1,3	96±2
CoACFNNA	99,7±0,4	99,7±0,5	99,6±0,7	100	100	99,8±0,5	100	100	99,9±0,1	98,6±0,8	100	99,1±0,5

Tabela 5.4 – Taxas de classificação correta para os 4 problemas restantes.

Modelos	*Wine*			*Breast Cancer Wisconsin*			*Breast Tissue*			*Wall-following Robot Navigation*		
	Treinamento	*Validação*	*Teste*	*Treinamento*	*Validação*	*Teste*	*Treinamento*	*Validação*	*Teste*	*Treinamento*	*Validação*	*Teste*
MLP 1hn	72,2±4,7	67,9±7,1	71,1±6,1	96,9±0,5	96,3±0,7	97,7±0,4	37,8±1,2	36,8±3,4	36,0±3,1	78,3±3,5	76,5±3,4	79,2±3,3
MLP 2hn	100	95,3±1,6	96,4±2,8	97,0±0,3	96,0±0,7	97,6±0,2	55,1±2,9	48,5±10,3	41,4±4,6	81,7±5,1	79,8±5,3	82,1±4,7
MLP 3hn	100	95,2±1,8	97,1±1,9	96,9±0,3	96,0±0,7	97,6±0,3	70,9±3,2	61,6±5,1	46,2±4,2	87,9±6,0	86,4±6,2	87,8±5,6
MLP 4hn	100	95,3±2,1	97,1±2,1	96,9±0,2	96,3±0,5	97,7±0,2	67,4±11,3	59,5±4,1	47,8±7,0	92,3±5,9	90,5±6,1	91,5±5,6
MLP 5hn	100	95,0±1,3	97,7±1,8	96,8±0,2	96,4±0,6	97,7±0,2	65,0±9,5	57,2±4,8	46,0±6,3	94,5±4,1	92,6±4,4	93,5±4,0
MLP 6hn	100	95,3±1,5	97,8±1,6	96,8±0,2	96,4±0,5	97,8±0,3	65,3±9,6	58,0±4,2	46,5±4,4	96,6±1,2	94,7±1,7	95,5±1,4
MLP 7hn	100	94,8±1,3	97,2±1,3	96,8±0,2	96,4±0,5	97,6±0,3	64,8±7,9	58,0±5,2	44,2±4,5	97,1±1,1	94,9±1,6	95,7±1,4
MLP 8hn	100	95,0±1,4	97,3±1,3	96,8±0,2	96,6±0,6	97,6±0,2	66,2±6,8	55,4±5,1	46,5±5,6	97,0±0,7	94,9±1,0	95,6±0,9
MLP 9hn	100	95,3±1,1	97,7±1,2	96,8±0,2	96,6±0,6	97,7±0,3	61,6±6,9	58,5±3,9	44,7±3,8	97,3±0,9	95,3±1,3	96,1±1,2
MLP 10hn	100	95,3±1,1	97,3±1,4	96,8±0,2	96,6±0,6	97,7±0,3	63,3±8,4	58,1±4,3	45,9±4,5	97,7±1,1	95,8±1,5	96,5±1,3
MLP 11hn	100	95,3±1,1	97,3±0,9	96,7±0,2	96,6±0,6	97,6±0,4	63,6±6,9	55,9±5,2	45,0±4,2	97,5±1,0	95,6±1,4	96,3±1,3
MLP 12hn	100	95,2±1,3	97,7±1,0	96,8±0,2	96,6±0,5	97,6±0,3	62,1±5,8	58,1±5,6	46,4±4,4	98,0±0,9	96,0±1,4	96,7±1,3
MLP 13hn	100	95,3±1,4	97,4±1,4	96,8±0,2	96,5±0,7	97,6±0,4	61,4±5,6	58,3±5,3	45,6±4,8	97,8±0,7	95,8±1,2	96,5±1,1
MLP 14hn	100	94,9±1,2	97,5±1,1	96,7±0,2	96,5±0,5	97,6±0,3	62,6±6,1	57,5±5,3	46,5±3,8	98,2±0,8	96,5±1,3	97,1±1,1
MLP 15hn	100	95,0±1,2	97,6±1,0	96,8±0,2	96,6±0,6	97,7±0,4	63,0±7,7	57,0±4,7	44,1±5,2	98,4±0,7	96,8±1,2	97,4±1,1
MLP2HL-A	100±0	94,4±2,5	97,1±1,8	96,7±0,4	96,9±1	98±0,6	63,5±9,3	56,9±8,9	45,5±4,5	95±8,2	94,1±8,8	94,9±7,8
MLP2HL-B	100±0,2	94,2±2,4	97,7±1,8	96,8±0,6	96,6±0,7	98±0,6	68±13,2	57,5±11	46,2±8,5	97,8±4,9	96,9±5,2	97,3±4,8
MHE-A	99,9±0,3	95,3±3	97±2	96,8±0,4	95±1,3	97,2±0,8	83,3±5,8	52,8±5,4	55,3±4,6	97,4±1,3	95,5±1,7	96,2±1,4
MHE-B	100	93,3±2,5	96,2±3,1	97±0,3	95,1±1,5	97,3±0,6	80,1±7,5	57,7±8,7	52,1±7,5	98±1,6	96,4±2,1	96,9±1,7
CasCorr	100±0,2	95±2,6	97,1±1,3	97,2±0,3	95,9±0,8	97,6±0,3	75,5±0	54,7±3,3	47,4±2,3	85,6±2,5	83,6±2,7	84,8±2,4
CoACFNNA	99,4±1,1	100	97,2±1,9	97±0,6	98,7±0,4	97,8±0,6	67,6±8,1	66,4±5,9	56,1±5,7	98,8±0,8	98,4±0,9	98,6±0,9

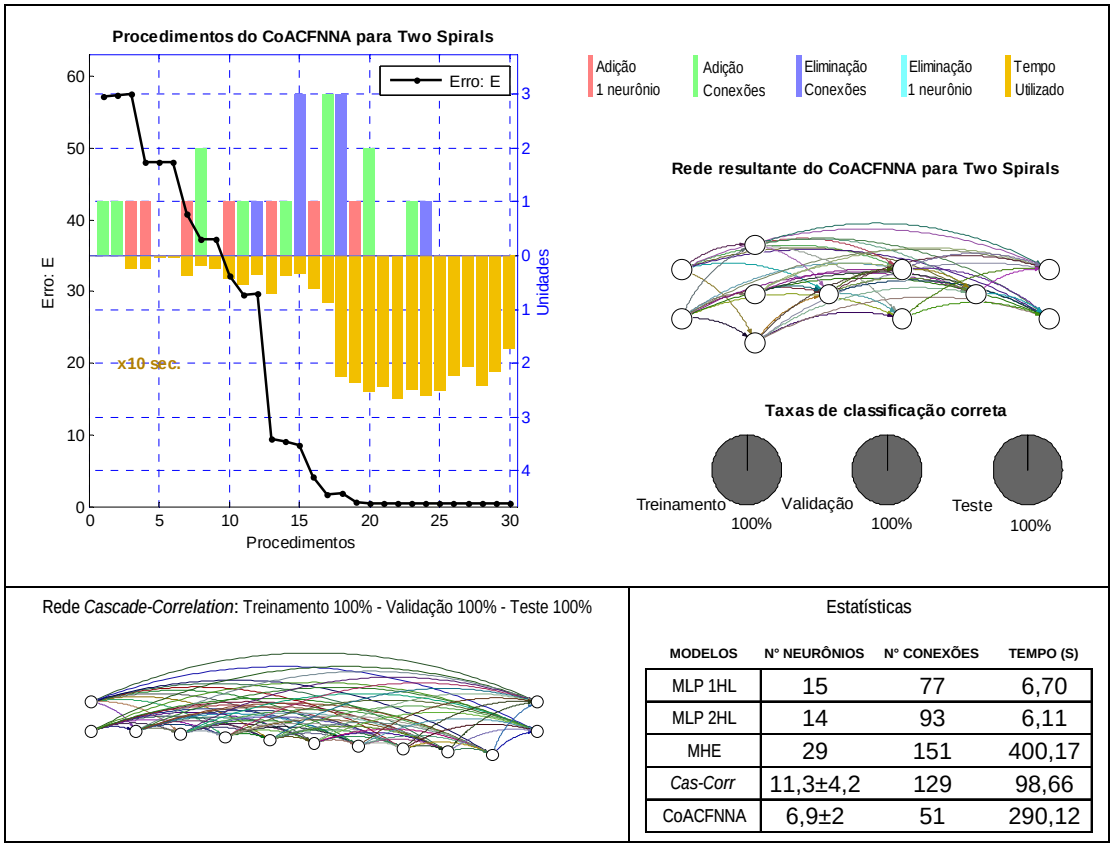
As Figuras 5.2 a 5.9, que são apresentadas a seguir, mostram informações gráficas em termos de arquiteturas das redes resultantes, tanto para o *CasCorr* como para o CoACFNNA. A informação em cada uma destas figuras está organizada da seguinte forma:

- Na parte superior-esquerda, mostra-se a sequência inteira das ações executadas ao longo do processo construtivo (aprendizado) realizado pelo algoritmo proposto nesta tese, o CoACFNNA. No gráfico, no eixo das ordenadas, a curva de cor preta corresponde ao erro de aprendizado *E*, que é obtido após a execução de cada um dos procedimentos que compõem o algoritmo construtivo (segundo o diagrama de fluxo do algoritmo ilustrado na Figura 4.2). O sucesso de cada procedimento é mostrado no eixo das abscissas, onde as barras de cor *vermelha* e *ciano* representam a adição e eliminação de neurônios, respectivamente, e as barras de cor *verde* e *azul* indicam o número de conexões inseridas e eliminadas da rede, respectivamente. Repare que as conexões inseridas e removidas junto com a adição e remoção dos neurônios não são ilustradas. O tempo gasto por cada

um destes procedimentos é ilustrado com barras de cor *dourado* em posição oposta às outras barras;

- Na parte superior-direita, mostra-se a arquitetura da rede neural resultante do CoACFNNA. As taxas de classificação correta também são apresentadas. Estas informações correspondem ao gráfico da parte da esquerda da figura e trata-se da proposta com maior taxa de classificação correta e com a arquitetura de maior parcimônia obtida dentre as 30 execuções do algoritmo;
- Na parte inferior-esquerda, mostra-se uma rede resultante via o algoritmo *CasCorr*. Esta rede corresponde à melhor instância das 30 execuções para este modelo. As taxas de classificação correta associadas a esta rede também são apresentadas;
- Na parte inferior-direita, mostra-se uma tabela reportando algumas estatísticas de interesse colhidas das 30 execuções dos modelos. A informação nela contida é a seguinte: valores de média e desvio-padrão do número de neurônios ocultos, o valor médio do número de conexões estabelecidas e o valor médio do tempo computacional (em segundos) demandado pelos modelos.

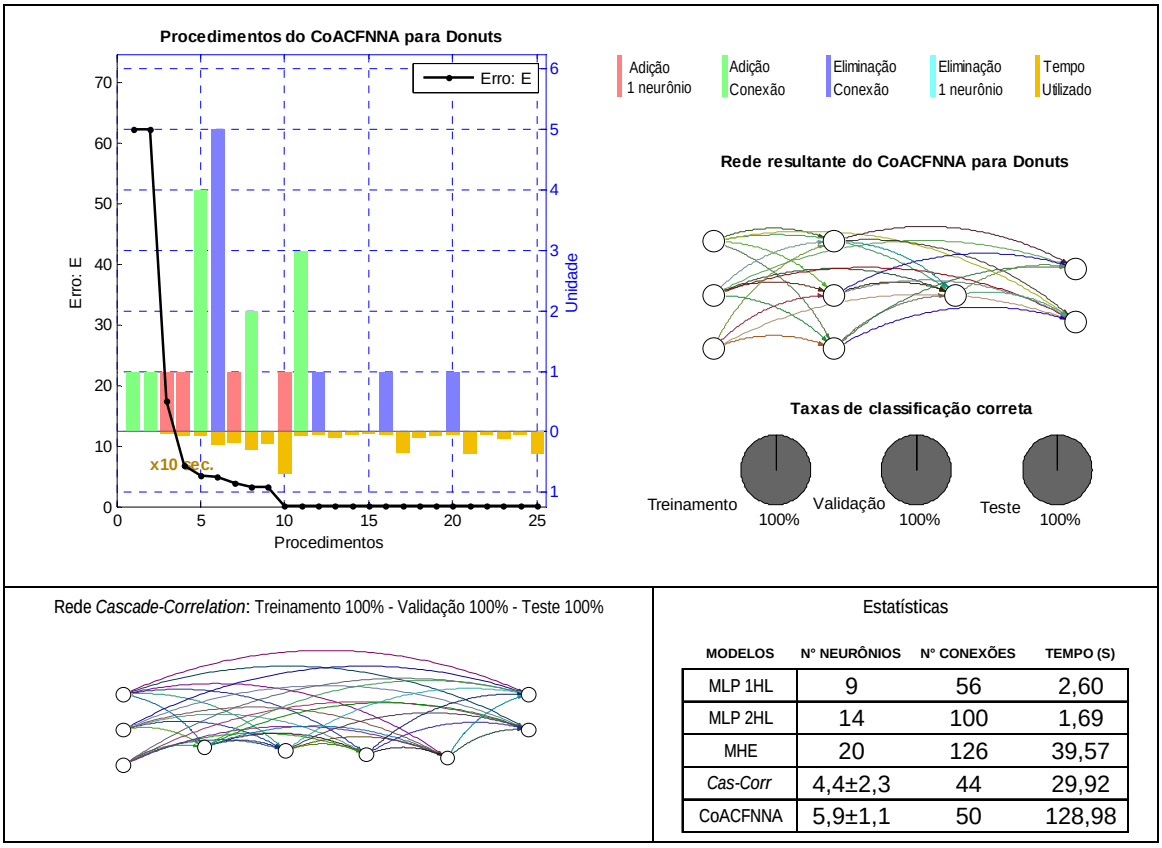
Na Figura 5.2, para o problema *Two Spirals 2D*, no lado esquerdo da figura, observa-se que, junto com a adição do quinto neurônio na rede, o erro de treinamento *E* experimenta sua maior queda em valor e que, após a adição do sétimo neurônio, poucas melhorias foram registradas. O processo inteiro parou após todos os procedimentos mostrarem insucesso no mesmo ciclo, isto na iteração 30. Todos os modelos participantes deste estudo comparativo foram competentes para resolver o problema em pelo menos uma das 30 execuções. No entanto, em média, *CasCorr* apresentou o pior desempenho (segundo a Tabela 5.3). A partir da tabela de estatísticas da parte direita inferior da Figura 5.2, pode-se observar que a melhor MLP com uma camada oculta (MLP1HL) necessita de 15 neurônios ocultos para mostrar competitividade, a rede MLP com duas camadas ocultas (MLP2HL) necessita de um total de 14 neurônios (7 em cada camada oculta), o modelo de MHE necessita de um total de 29 neurônios (11 em cada especialista e 7 na rede *gating*). Já o modelo *CasCorr* precisa, em média, de 11,3 neurônios ocultos e 129 conexões. O CoACFNNA produziu as soluções com maior parcimônia, pois, em média, usou 6,9 neurônios e 51 conexões.



MODELOS	N° NEURÔNIOS	N° CONEXÕES	TEMPO (S)
MLP 1HL	15	77	6,70
MLP 2HL	14	93	6,11
MHE	29	151	400,17
Cas-Corr	11,3±4,2	129	98,66
CoACFNNA	6,9±2	51	290,12

Figura 5.2 – Melhores soluções do CoACFNNA e *CasCorr* para o problema *Two Spirals 2D*.

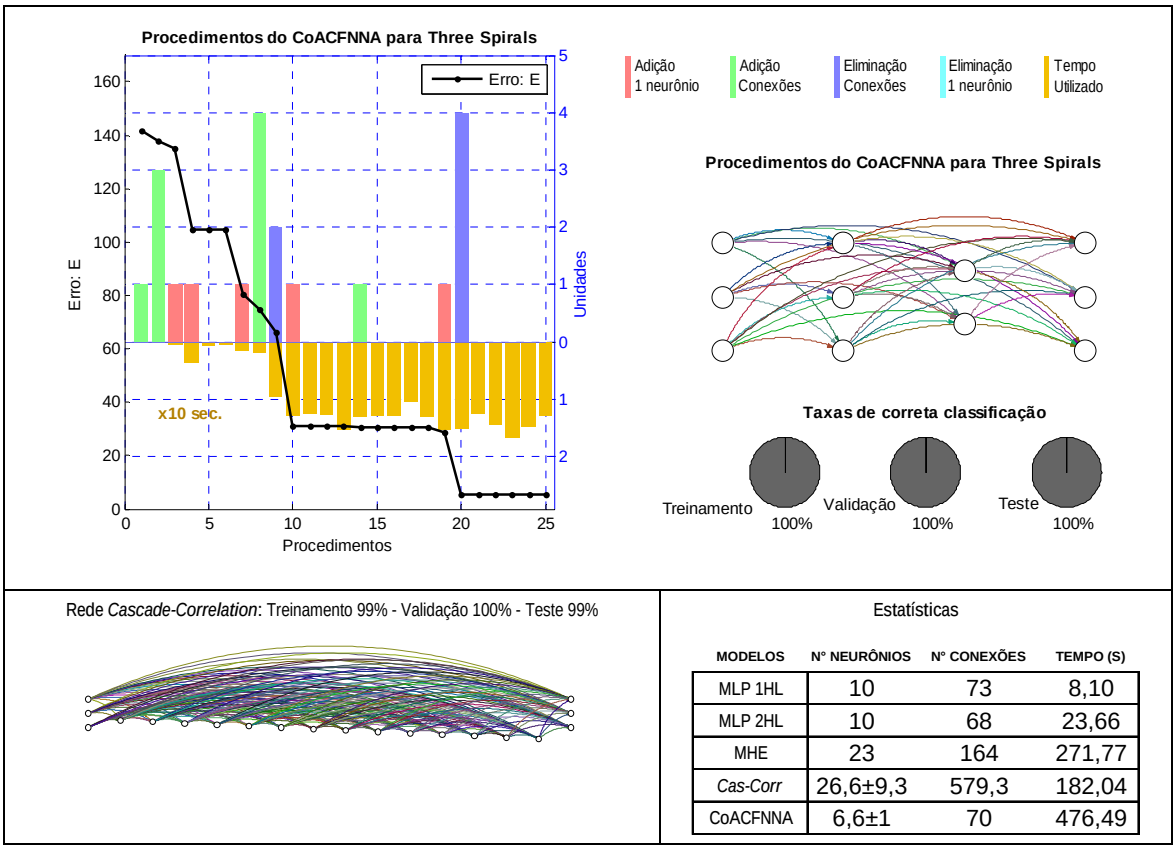
Para o problema *Two Donuts 3D*, todos os modelos conseguem apresentar desempenhos similares (como mostrado na Tabela 5.3). Na Figura 5.3, a melhor MLP1HL requer 9 neurônios ocultos, a MLP2HL precisa de um total de 14 neurônios (7 em cada camada oculta), o MHE utiliza um total de 20 neurônios (7 em cada especialista e 6 na rede *gating*), o *CasCorr* precisa, em média, 4,4 neurônios e as redes obtidas via CoACFNNA precisam de 5,9 neurônios, em média. Para este problema, o *CasCorr* produziu redes com maior parcimônia, seguido de perto pelas redes produzidas pelo CoACFNNA.



MODELOS	N° NEURÔNIOS	N° CONEXÕES	TEMPO (S)
MLP 1HL	9	56	2,60
MLP 2HL	14	100	1,69
MHE	20	126	39,57
Cas-Corr	4,4±2,3	44	29,92
CoACFNNA	5,9±1,1	50	128,98

Figura 5.3 – Melhores soluções do CoACFNNA e *CasCorr* para o problema *Two Donuts 3D*.

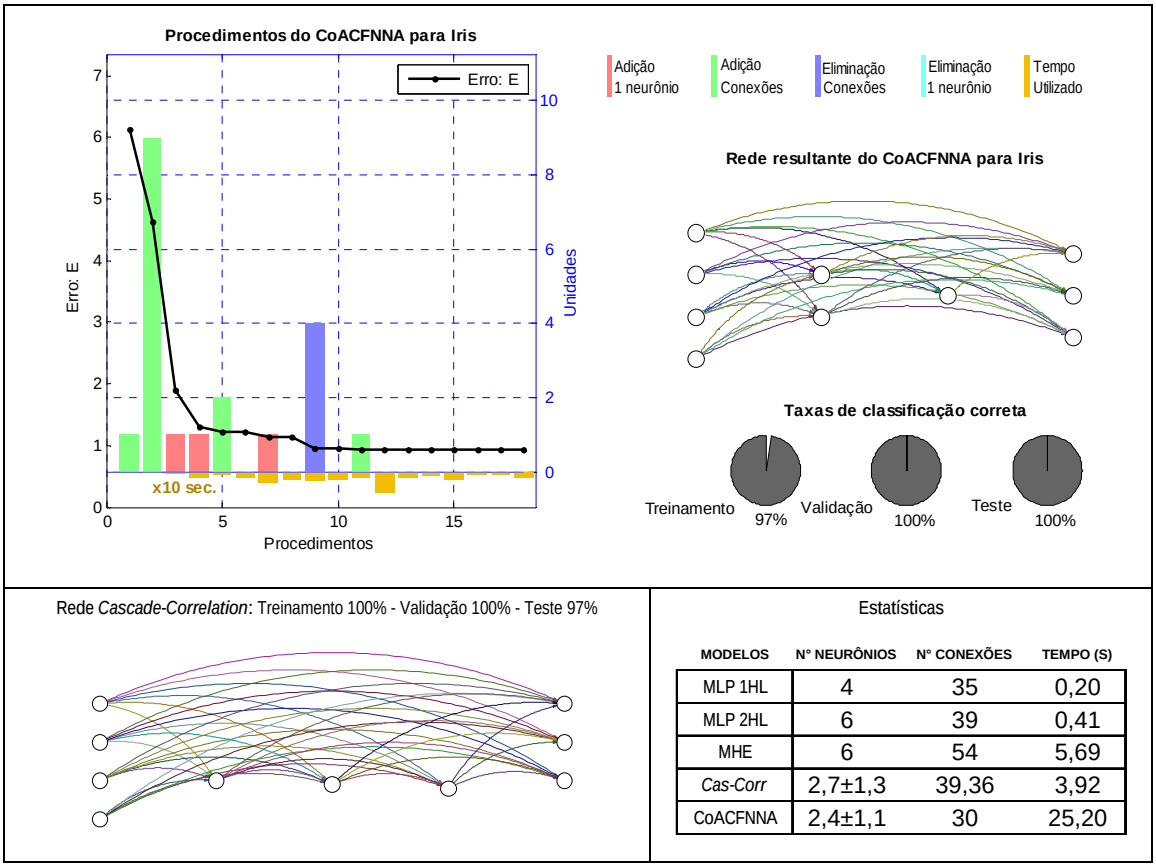
O problema *Three Spirals 3D* foi solucionado por todos os modelos em estudo, com uma pequena dificuldade para o *CasCorr*, como pode ser verificado na Tabela 5.3. No aspecto de parcimônia, pode-se apreciar (ver Figura 5.4) que a melhor configuração de MLP1HL utiliza 10 neurônios ocultos, a MLP2HL usa 5 neurônios em cada camada, totalizando 10 neurônios, a MHE precisa de um total de 23 (9+9+5) neurônios. O *CasCorr*, desta vez, produziu as redes com menor parcimônia, já que, em média, usou 26 neurônios. O CoACFNNA liderou em termos de parcimônia, ao usar, em média, apenas 6,6 neurônios ocultos. No entanto, o número médio de conexões (70) é equivalente aquele obtido com as MLPs, pois são permitidas nas ACFNNs conexões diretas com os neurônios de saída, mesmo para neurônios que não fazem parte da última camada oculta, antes da camada de saída.



MODELOS	Nº NEURÔNIOS	Nº CONEXÕES	TEMPO (S)
MLP 1HL	10	73	8,10
MLP 2HL	10	68	23,66
MHE	23	164	271,77
Cas-Corr	26,6±9,3	579,3	182,04
CoACFNNA	6,6±1	70	476,49

Figura 5.4 – Melhores soluções do CoACFNNA e *CasCorr* para o problema *Three Spirals 3D*.

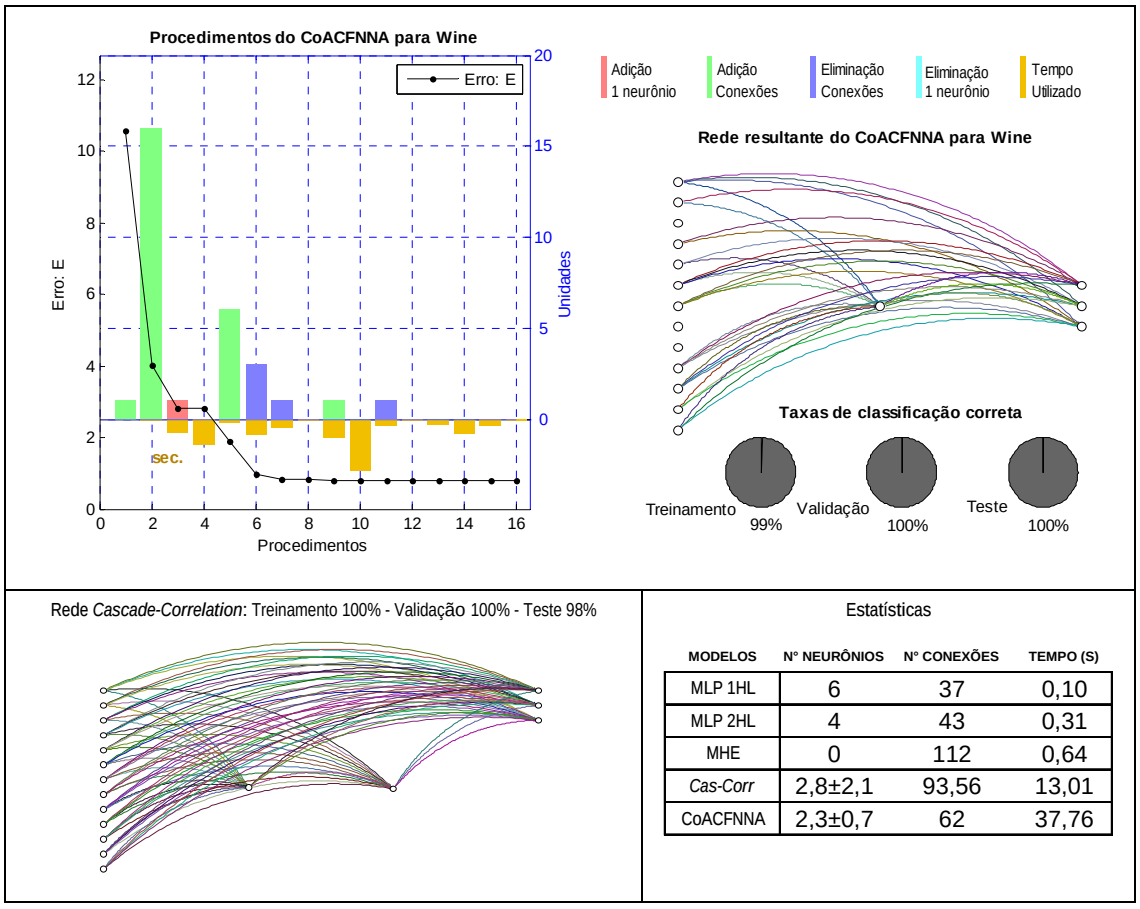
Para o problema *Iris*, somente o *CasCorr*, em média, não foi capaz de apresentar desempenho acima de 99% de classificação correta (como mostrado na Tabela 5.3), isso junto ao subconjunto de teste. No entanto, as redes do *CasCorr* e do CoACFNNA solicitaram os menores números de neurônios, como ilustrado na tabela da parte inferior direita da Figura 5.5.



MODELOS	N° NEURÔNIOS	N° CONEXÕES	TEMPO (S)
MLP 1HL	4	35	0,20
MLP 2HL	6	39	0,41
MHE	6	54	5,69
Cas-Corr	2,7±1,3	39,36	3,92
CoACFNNA	2,4±1,1	30	25,20

Figura 5.5 – Melhores soluções do CoACFNNA e *CasCorr* para o problema *Iris*.

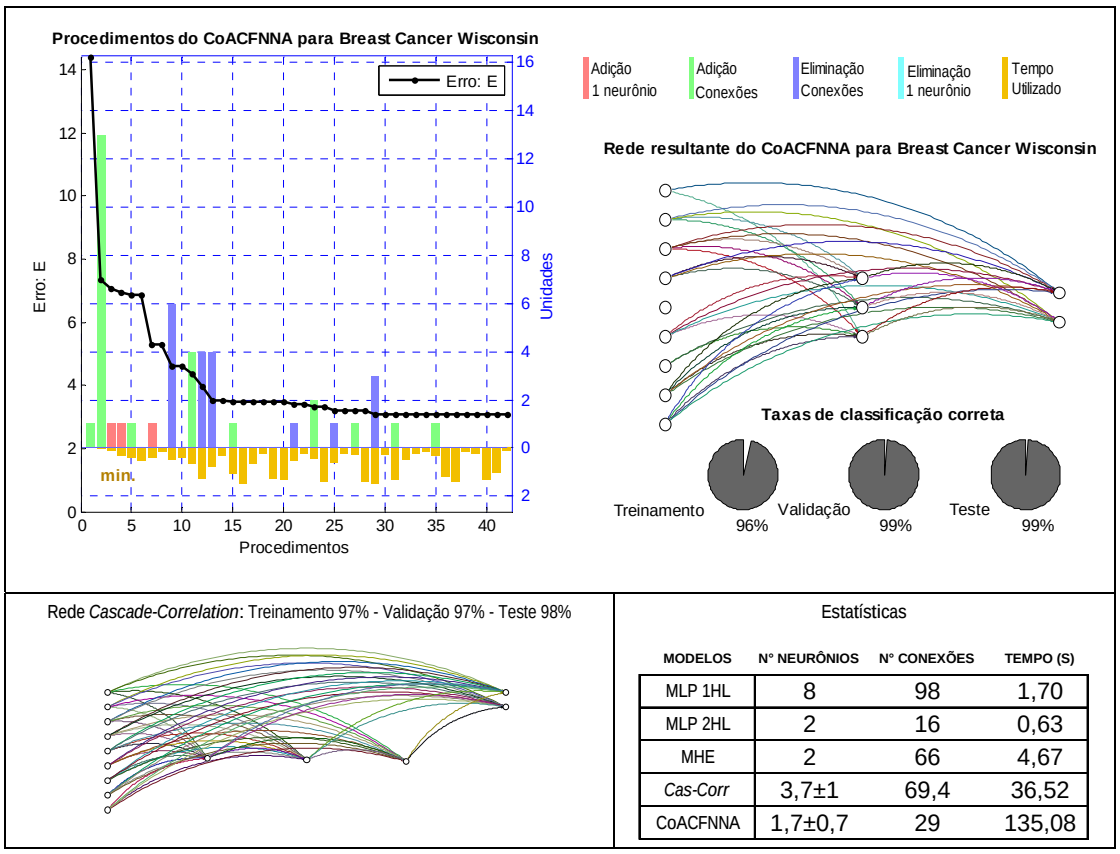
A Figura 5.6 apresenta resultados para o problema *Wine*. A melhor configuração de MLP1HL precisa de 6 neurônios ocultos para alcançar desempenho competitivo no subconjunto de *teste*, enquanto que a MLP2HL precisa de um total de 4 neurônios (dois em cada camada oculta). Já a MHE não precisa de modelos não-lineares, pois tanto as duas redes especialistas quanto a *gating* foram modelos lineares. O *CasCorr* precisa, em média, de 2,8 neurônios e o CoACFNNA requer, em média, 2,3 neurônios. Este problema ajuda a exemplificar, na prática, a capacidade intrínseca do CoACFNNA de realizar implicitamente a tarefa de *seleção de variáveis*, já que a rede resultante desconsiderou 3 das 13 variáveis de entrada do problema, sendo elas x_3, x_8 e x_9 (como ilustrado na parte superior-direita da Figura 5.6).



MODELOS	Nº NEURÔNIOS	Nº CONEXÕES	TEMPO (S)
MLP 1HL	6	37	0,10
MLP 2HL	4	43	0,31
MHE	0	112	0,64
Cas-Corr	2,8±2,1	93,56	13,01
CoACFNNA	2,3±0,7	62	37,76

Figura 5.6 – Melhores soluções do CoACFNNA e *CasCorr* para o problema *Wine*.

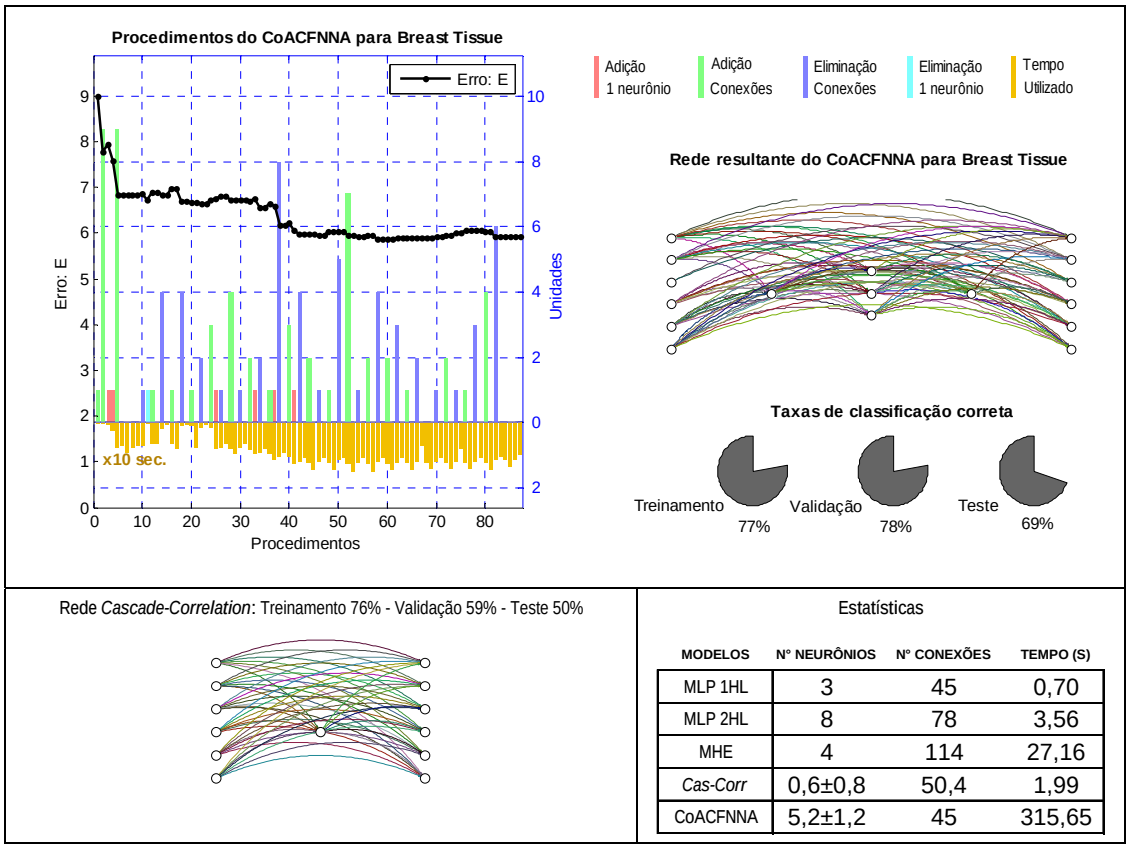
A Figura 5.7 mostra os resultados para o problema *Breast Cancer Wisconsin*. Aqui, pode-se dizer que, em média, todos os modelos conseguem desempenhos parecidos, com uma pequena vantagem no subconjunto de *teste* para a MLP2HL, a qual precisa de 2 neurônios ocultos (um neurônio em cada camada). A melhor MLP1HL usa 8 neurônios, enquanto que a MHE utiliza os dois especialistas lineares e a rede *gating* com 2 neurônios ocultos. O *CasCorr*, em média, emprega 3,7 neurônios, e o CoACFNNA produz redes que, em média, requerem 1,7 neurônios ocultos. A rede neural produzida pelo CoACFNNA, ilustrada na Figura 5.7, desconsiderou a entrada x_5 do problema.



MODELOS	N° NEURÔNIOS	N° CONEXÕES	TEMPO (S)
MLP 1HL	8	98	1,70
MLP 2HL	2	16	0,63
MHE	2	66	4,67
Cas-Corr	3,7±1	69,4	36,52
CoACFNNA	1,7±0,7	29	135,08

Figura 5.7 – Melhores soluções do CoACFNNA e *CasCorr* para o problema *Breast Cancer Wisconsin*.

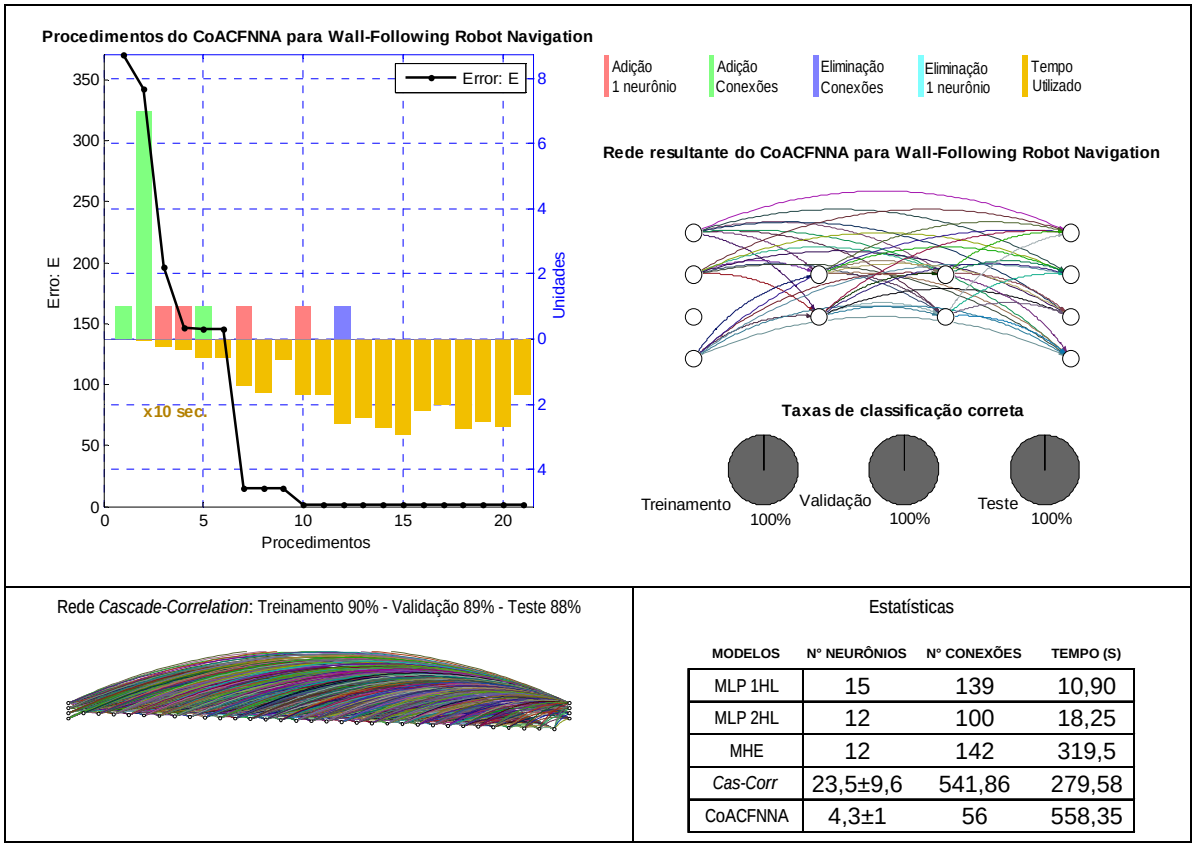
Para o problema *Breast Tissue*, o desempenho médio do CoACFNNA é superior ao produzido pelos outros modelos (como indicado na Tabela 5.4), sempre considerando o subconjunto de *teste*. O número médio de neurônios ocultos foi 5,2 (ver Figura 5.8). O *CasCorr* usa menos neurônios, em média 0,6, mas seu desempenho é bem inferior ao dos outros modelos. Por ser o problema de classificação mais desafiador, foi aqui que o CoACFNNA apresentou a maior variabilidade nos resultados, embora em níveis compatíveis ao obtido pelos demais modelos de bom desempenho (ver Tabela 5.4).



MODELOS	Nº NEURÔNIOS	Nº CONEXÕES	TEMPO (S)
MLP 1HL	3	45	0,70
MLP 2HL	8	78	3,56
MHE	4	114	27,16
Cas-Corr	0,6±0,8	50,4	1,99
CoACFNNA	5,2±1,2	45	315,65

Figura 5.8 – Melhores soluções do CoACFNNA e *CasCorr* para o problema *Breast Tissue*.

Por último, quanto aos resultados para o problema *Wall-following Robot Navigation*, o desempenho do CoACFNNA é superior ao produzido pelos demais modelos. O *CasCorr* apresenta dificuldades junto a este problema, tanto em termos de parcimônia como de taxa de classificação correta. A parcimônia das redes produzidas pelo CoACFNNA é muito superior à obtida com os demais modelos, precisando, em média, apenas de 4,3 neurônios ocultos. Já o *CasCorr* usa, em média, 23 neurônios, o MLP1HL usa 15, o MLP2HL usa 12 ao todo, e a MHE precisa de um total de 12 neurônios. A Figura 5.9 ilustra estes resultados e mostra uma instância de rede com apenas 4 neurônios ocultos e com 100% de acerto junto aos 3 subconjuntos de dados, tendo sido descartada a variável de entrada x_3.



MODELOS	Nº NEURÔNIOS	Nº CONEXÕES	TEMPO (S)
MLP 1HL	15	139	10,90
MLP 2HL	12	100	18,25
MHE	12	142	319,5
Cas-Corr	23,5±9,6	541,86	279,58
CoACFNNA	4,3±1	56	558,35

Figura 5.9 – Melhores soluções do CoACFNNA e *CasCorr* para o problema *Wall-following Robot Navigation*.

A seguir, apresentam-se resultados de simulações realizadas visando acompanhar a influência do parâmetro *SEr_max*, entendido como a taxa inicial máxima de relaxação do erro (descrito na seção 4.2.8). Foram considerados os seguintes valores para este parâmetro: 1,00; 1,05; 1,10; 1,15; 1,20; 1,25 e 1,30. Para cada um destes valores, foram realizadas 30 execuções do CoACFNNA, junto aos problemas *Wine* e *Two Spirals*. Os resultados são ilustrados nas Figuras 5.10 e 5.11. No gráfico da esquerda de cada figura, apresentam-se as taxas médias de classificação correta para os subconjuntos de treinamento, validação e teste. No gráfico do meio da figura, os valores médios do número de neurônios e de conexões. E no gráfico à direita, os valores médios do tempo computacional, número de iterações para atingir o erro mínimo e as iterações totais utilizadas.

Analisando os resultados para o problema *Wine* da Figura 5.10, observa-se que valores altos para *SEr_max* tendem a elevar o número de neurônios e conexões das soluções

produzidas. O tempo computacional e o número de iterações do algoritmo também, em média, sofrem incremento, comprometendo a parcimônia das soluções. Já o desempenho, em termos de taxa de classificação correta, manteve-se, em média, estável a partir de *SEr_max* = 1,10.

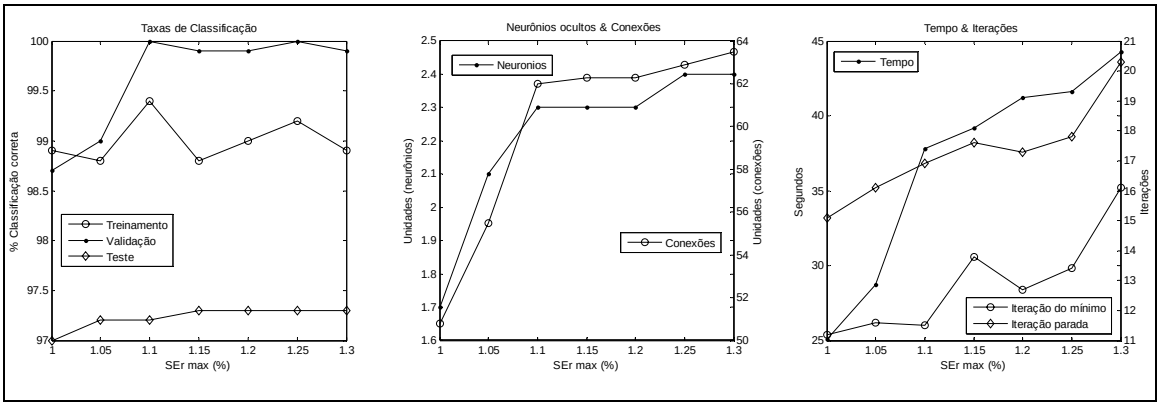


Figura 5.10 – Análise de sensibilidade para o *SEr_max* do CoACFNNA no problema *Wine*.

Para o problema *Two Spirals*, conforme apresentado na Figura 5.11, observa-se que, quando o mecanismo de relaxação do erro é anulado (*SEr_max* = 1,0), as redes experimentaram, em média, desempenhos inferiores. As taxas de classificação correta sofreram melhora considerável ao usar valores de *SEr_max* a partir de 1,05, e à medida que se incrementa este valor incrementaram-se também o número de neurônios, conexões, tempo e iterações, embora o número de neurônios e conexões tenha exibido um comportamento de saturação em seu crescimento.

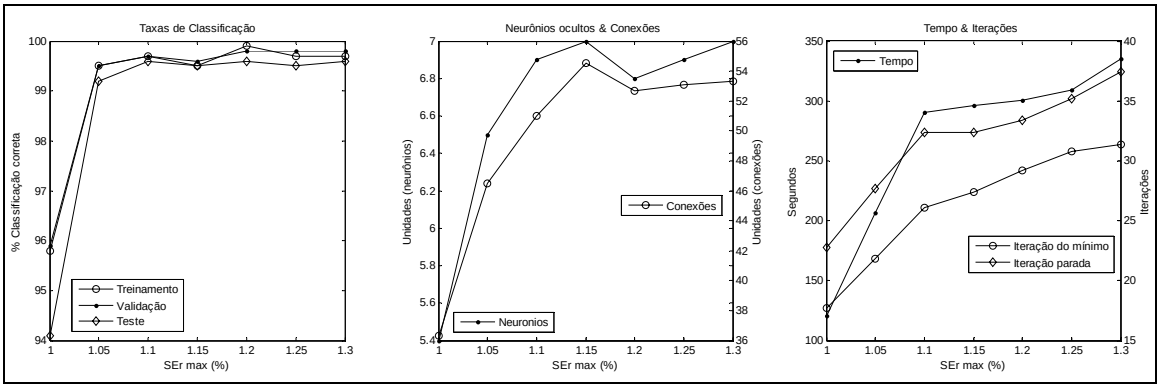


Figura 5.11 – Análise de sensibilidade para o *SEr_max* do CoACFNNA no problema *Two Spirals*.

5.2.2. Segundo estudo de caso: CoACFNNA versus EPNet

Visto que os experimentos comparativos da seção anterior não incluíam modelos com capacidade de produzir redes neurais com arquiteturas arbitrárias, do tipo ACFNN, a presente seção visa complementar estes estudos. O CoACFNNA é comparado aqui com o EPNet, que é um algoritmo bem conhecido, referenciado na literatura e que também produz redes do tipo ACFNN, mas via computação evolutiva. As principais características do algoritmo EPNet foram detalhadas no Capítulo 3, seção 3.3.

No trabalho de YAO & LIU (1997), o EPNet foi testado em vários problemas, dentre os quais foram escolhidos os quatro problemas reais de classificação adotados pelos autores. Estes problemas pertencem ao UCI *Machine Learning Repository* e referem-se à área de saúde. São eles: *Breast Cancer*, *Diabetes*, *Heart Disease* e *Thyroid*.

Visando realizar uma comparação com as mesmas condições referente à divisão dos conjuntos de dados (*treinamento*, *validação* e *teste*), a separação destes subconjuntos foi realizada da mesma forma adotada pelos autores do EPNet. Isto foi possível porque os autores forneceram esta informação. Consequentemente, os resultados obtidos via o CoACFNNA foram comparados com os resultados publicados pelos autores do EPNet.

A Tabela 5.5 apresenta os resultados obtidos pelo EPNet, extraídos das Tabelas VI e VII de YAO & LIU (1997), seguido dos resultados obtidos via o CoACFNNA. Em ambos os casos, realizaram-se 30 execuções para gerar valores estatísticos de média e desvio-padrão das seguintes medidas: taxas de classificação correta para os subconjuntos de treinamento, validação e teste, números de neurônios ocultos e conexões utilizadas (incluem os valores mínimos e máximos). Uma vez que ambas as metodologias produzem redes do tipo ACFNN, estes valores permitem ter um maior alcance do grau de parcimônia que cada proposta pode oferecer.

Analisando os resultados (ver Tabela 5.5), pode-se notar a superioridade das redes produzidas pelo CoACFNNA, em especial nos três últimos problemas, sendo que para o primeiro problema (*Breast Cancer*) os desempenhos podem ser considerados equivalentes, tanto em termos de taxa de acerto quanto no tocante à parcimônia das redes produzidas.

Para os outros três problemas, *Diabetes*, *Heart Disease* e *Thyroid*, as redes do CoACFNNA apresentam melhores taxas de acerto e com graus de parcimônia superiores. Por exemplo, fazendo uma análise do número de conexões para os problemas de *Diabetes* e *Heart Disease*, as redes do CoACFNNA requerem em média um número de conexões próximo da metade das requeridas pelas redes do EPNet, sendo que para o problema *Thyroid* esta diferença mostrou-se maior ainda, quase a terça parte do requerido pelo EPNet (219 conexões do EPNet perante a 75 do CoACFNNA).

Um aspecto que vale mencionar, nesta parte da análise dos resultados, está relacionado com uma etapa de refinamento adicional que os autores do EPNet efetuaram após a evolução da rede tomada como solução. Aqui, antes de calcular o desempenho no subconjunto de teste, os autores combinaram os subconjuntos de treinamento e validação para um ajuste final dos pesos sinápticos via os algoritmos MBP (*modified backpropagation*) e SA (*simulated annealing*).

Tabela 5.5 – Resultados numéricos comparativos entre o CoACFNNA e o EPNet.

Modelos		*Breast Cancer*					*Diabetes*				
	Estatísticas	Treinamento	Validação	Teste	No. Neuroniôs	No. Conexões	Treinamento	Validação	Teste	No. Neuroniôs	No. Conexões
EPNet	*MEAN±STD*	96,23±0,69	99,41±0,24	98,62±0,94	2±1,1	41±14,7	75,95±0,01	81,15±0,01	77,62±0,01	3,4±1,3	52,3±16,1
	MIN	95,42	98,86	96,00	0	15	73,96	79,69	75,00	1	27
	MAX	98,28	100,00	100,00	5	84	78,13	83,33	80,73	6	87
CoACFNNA	*MEAN±STD*	96,09±0,63	98,87±0,4	98,92±0,27	2,5±0,8	40,9±11,4	76,14±1,08	81,56±0,87	79,13±1,33	2,2±0,6	34,3±7,0
	MIN	94,74	98,25	98,24	1	22	74,22	80,21	76,56	1	18
	MAX	97,08	99,42	99,41	4	63	78,39	83,85	81,77	4	50
		Heart Disease					*Thyroid*				
EPNet	*MEAN±STD*	86,37±1,52	82,7±2	83,24±2,03	4,1±2,1	92,6±40,8	99,18±0,15	98,83±0,24	97,89±0,22	5,9±2,4	219,6±74,4
	MIN	83,58	79,41	80,88	1	34	98,85	98,25	97,38	3	128
	MAX	90,30	86,77	86,77	10	213	99,48	99,36	98,37	12	417
CoACFNNA	*MEAN±STD*	84,54±2,49	90,99±1,39	85,30±3,16	2,4±0,7	50,2±13,6	99,26±0,15	99,09±0,17	98,42±0,15	2,5±1	75,1±27,8
	MIN	79,59	89,19	78,08	2	34	98,89	98,81	97,93	1	28
	MAX	91,16	94,59	90,41	5	91	99,56	99,36	98,66	6	148

A Figura 5.12 ilustra as melhores redes em termos de parcimônia produzidas pelo CoACFNNA para cada um dos quatro problemas utilizados neste estudo comparativo. Vale destacar que, nos quatro casos, a seleção de variáveis de entrada esteve presente, sendo que para o problema *Thyroid*, 5 das 21 variáveis de entrada foram desconsideradas pela rede resultante, que possui apenas 1 neurônio oculto e 28 conexões, e com melhor taxa de acerto que qualquer das 30 produzidas pelo EPNet, em que as redes com maior parcimônia utilizaram 3 neurônios (analisando unicamente pelo número de neurônios ocultos) e 128 conexões (analisando unicamente pelo número de conexões).

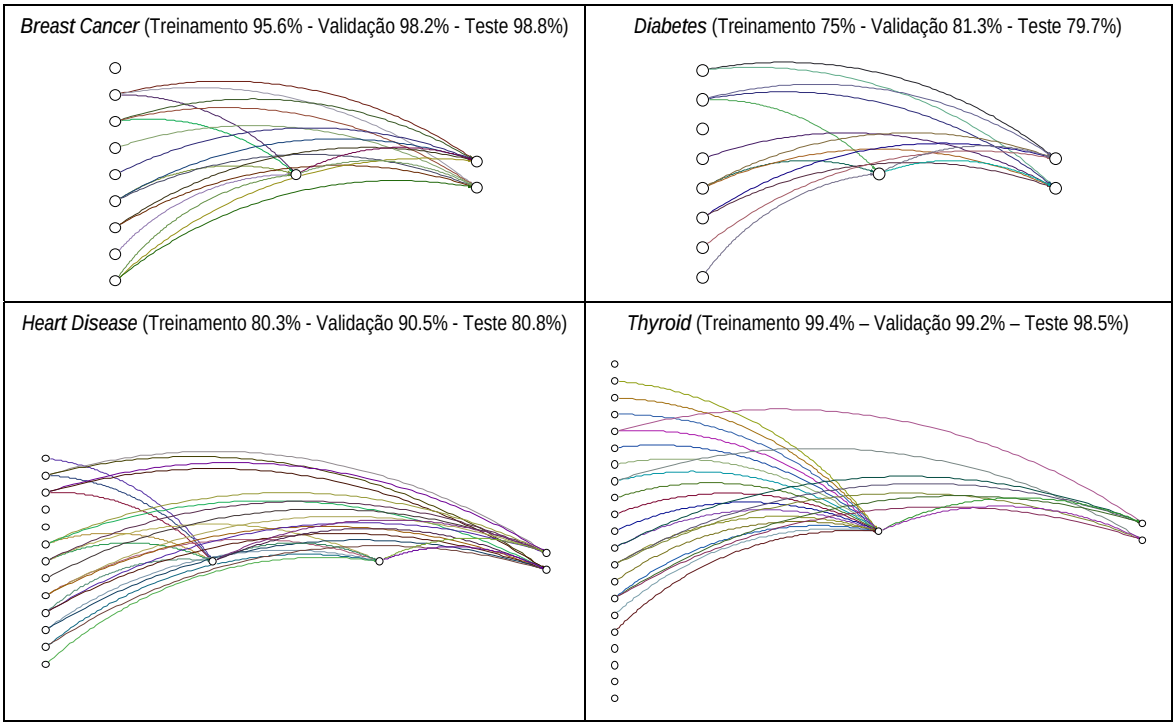


Figura 5.12 – As redes com maior parcimônia produzidas pelo CoACFNNA.

5.2.3. Problemas com grande quantidade de amostras

Com o propósito de observar o impacto em termos de tempo computacional, conforme se aumenta o número de amostras, foram considerados dois problemas com uma quantidade de amostras bem maior que aquele encontrado junto aos problemas já trabalhados nas seções anteriores. Estes problemas foram extraídos do UCI *Machine Learning Repository*. A descrição dos problemas e os resultados são apresentados a seguir.

Problema do *MAGIC Gamma Telescope*: Problema da área da física com dados gerados por um programa baseado no método de Monte Carlo para simular o registro de alta energia de partículas gamma na atmosfera da base terrestre do telescópio gamma Cherenkov, usando técnicas de tratamento de imagens. Os dados possuem 19020 amostras e duas classes. A primeira classe possui 12332 amostras e corresponde ao sinal Gamma e a outra classe possui 6688 e representa a Hadron (*background*). O número de variáveis de entrada

(atributos) é de 10. A melhor rede obtida pelo CoACFNNA, depois de 10 execuções do algoritmo, tem a seguinte configuração: 5 neurônios ocultos em três camadas ocultas, 85 conexões e requereu, em média, 9,44 minutos para realizar todos os procedimentos construtivos. A Figura 5.13 ilustra todo o procedimento de construção para esta rede. Note que, dessa vez, a unidade de tempo é minutos (barras de cor dourada representando o tempo gasto em cada procedimento). As taxas de classificação correta foram de 88,22%, 87,19% e 86,92% para os subconjuntos de treinamento, validação e teste, respectivamente. O modelo MLP precisou pelo menos de 9 neurônios ocultos para obter um desempenho similar e o *CasCorr* alcançou uma taxa de acerto máxima de 79,81% junto ao subconjunto de teste, utilizando 6 neurônios ocultos.

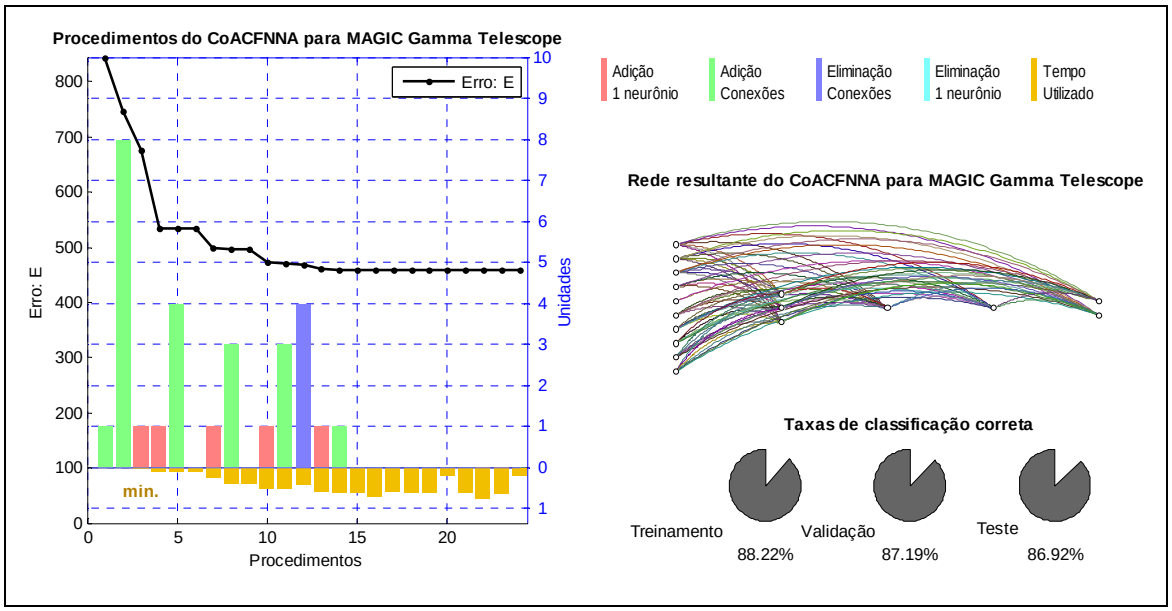


Figura 5.13 – A melhor rede do CoACFNNA para o problema *MAGIC Gamma Telescope*.

Problema do *StatLog–Shuttle*: Este problema também é de natureza física e possui 58000 amostras, 9 variáveis de entrada e 7 classes. Aproximadamente 80% das amostras pertencem à classe 1. Os restantes 20% das amostras contêm as outras 6 classes. Com isso, classificar todas as amostras como pertencentes à classe predominante já produziria uma taxa de acerto em torno de 80%. De acordo com os provedores dos dados, o objetivo é obter um classificador com taxa de acerto de 99% a 99,9%. A melhor rede produzida pelo

CoACFNNA, após 10 execuções, possui as seguinte características: 8 neurônios ocultos dispostos em 3 camadas, 160 conexões e requereu 99,37 minutos (ver Figura 5.14). As taxas de classificação correta foram de 99,72%, 99,75% e 99,73%, para os subconjuntos de treinamento, validação e teste, respectivamente. O modelo MLP precisou pelo menos 14 neurônios para igualar este desempenho, enquanto que o *CasCorr* convergiu para 6 neurônios e uma taxa de acerto de apenas 93,27% junto ao subconjunto de teste.

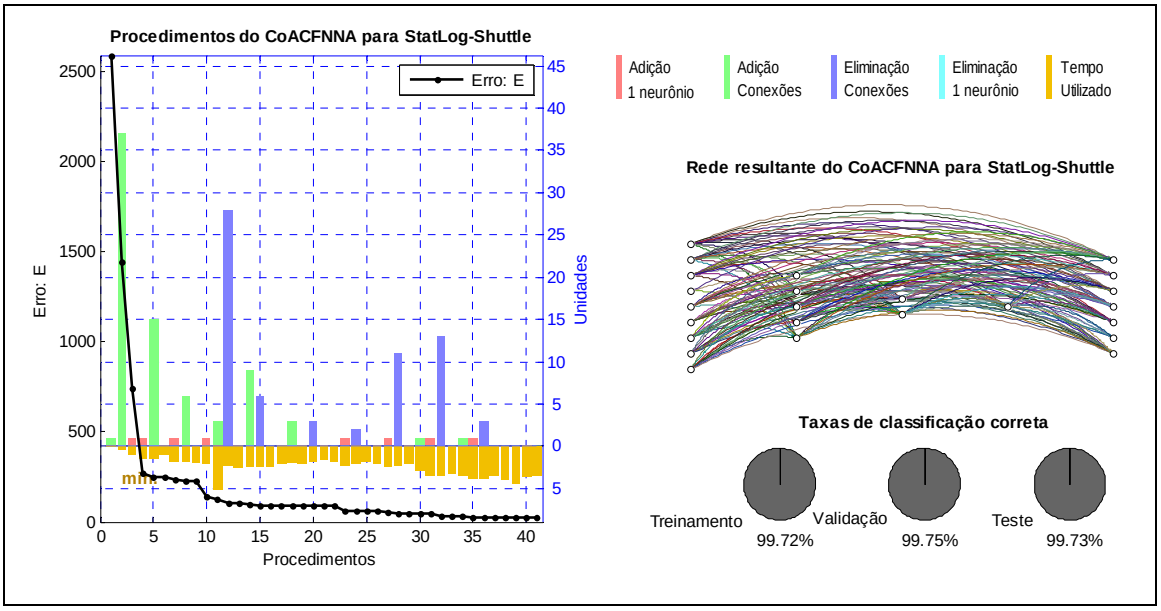


Figura 5.14 – A melhor rede do CoACFNNA para o problema *Statlog-Shuttle*.

5.3 Problema de redução de dimensão

Esta seção tem como objetivo descrever modificações realizadas no algoritmo padrão do CoACFNNA visando a sua aplicação junto a problemas de redução de dimensão.

Na grande maioria de vezes, e principalmente ao lidar com problemas do mundo real, surge a necessidade de reduzir a dimensão em que se encontra um determinado conjunto de dados. Esta necessidade pode estar associada a demandas de visualização dos dados ou à expectativa de melhora do desempenho de certos algoritmos que executam tarefas de

reconhecimento de padrões e regressão de dados, por exemplo. A redução da dimensão é justificada nos seguintes termos:

- Do ponto de vista de *visualização dos dados*, em dimensões compatíveis com a capacidade de percepção do ser humano, por exemplo, em 2D ou 3D, a redução de dimensão permitiria coletar informações a priori (com um certo grau de aproximação) da estimativa da distribuição dos dados no espaço real. Essas informações poderiam alimentar processos de tomada de decisão em: (*i*) Agrupamento de dados, na estimativa do número de grupos existentes, da densidade e da forma dos grupos; (*ii*) Classificação de padrões, ao promover uma aproximação da real distribuição das classes, ao permitir averiguar a existência de sobreposição de classes e ao possibilitar uma análise de separabilidade linear das amostras. O sucesso desta etapa poderia influenciar na escolha dos algoritmos de agrupamento e classificação, assim como na definição de seus parâmetros.

- Do ponto de vista de redução da dimensão, onde o objetivo não é necessariamente poder visualizar os dados em 2D ou 3D e sim reduzi-los a uma dimensão que beneficie o desempenho de algoritmos de aprendizado de máquina, seja em termos de acurácia ou de custo computacional. Dependendo do algoritmo de redução de dimensão dos dados, se for um que combina os atributos originais, então está-se tratando de Extração de Características (*Feature Extraction*), onde um novo conjunto de dados (de preferência com menor dimensão) é gerado a partir dos dados originais. Por outra parte, se a redução da dimensão é feita apenas pela seleção (filtro) dos atributos mais informativos, então está-se tratando de Seleção de Variáveis (*Variable Selection* (GUYON & ELISSEEFF, 2003)).

5.3.1 Algoritmos de redução de dimensão

Dependendo da forma como os algoritmos conduzem o processo de redução de dimensão, eles podem ser classificados como lineares ou não-lineares. Dentre os algoritmos

lineares mais conhecidos na literatura tem-se: PCA – *Principal component analysis* (DEMARTINES & HÉRAULT, 1995), ICA – *Independent component analysis* (JUTTEN & HÉRAULT, 1991; COMON, 1994), SVD – *Singular value decomposition* (LEE & VERLEYSEN, 2007).

Em relação a algoritmos não-lineares, são muitas as propostas na literatura, e eles são ajustados de forma a aprender um modelo interno com dimensão de saída menor do que aquela exibida pelo conjunto original de dados. Dentre os algoritmos mais conhecidos, pode-se citar *Sammon's Mapping* (SAMMON, 1969) como o primeiro proposto na literatura, ainda utilizado e que serve de base para novas propostas; CCA – *Curvilinear Component Analysis* (DEMARTINES & HÉRAULT, 1995); *nonlinear PCA* (KRAMER, 1991; SCHOLZ *et al*., 2005), que é o nome comumente utilizado para quando se emprega uma rede neural MLP com, no mínimo, três camadas ocultas, sendo que a *nova dimensão* é extraída das saídas dos neurônios especificados para compor a camada oculta central da rede. Neste caso, a saída desejada da rede é dada pelos sinais de entrada. As funções de ativação dos neurônios da camada oculta da *nova dimensão* e da camada de saída podem ser lineares ou não-lineares (KRAMER, 1991) e as funções de ativação para os outros neurônios ocultos são não-lineares. A Figura 5.15 ilustra um exemplo de arquitetura de rede *nonlinear PCA*.

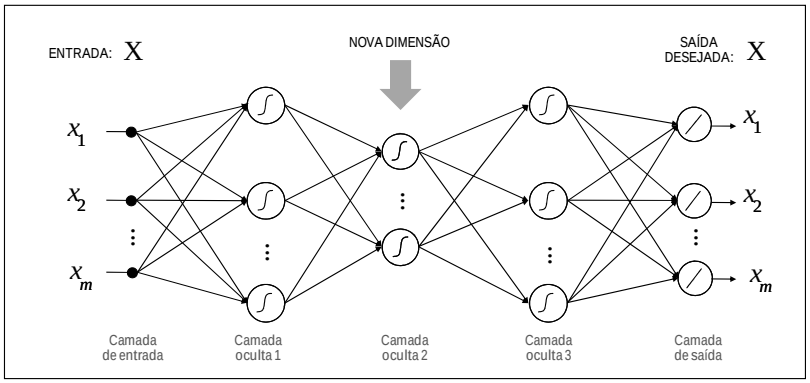


Figura 5.15 – Exemplo do *nonlinear PCA* via uma RNA MLP com 3 camadas ocultas.

Se todos os neurônios da rede neural da Figura 5.15 tiverem função de ativação linear, o ajuste dos pesos vai reproduzir na camada oculta 2 os componentes principais do PCA

linear. Esta é uma justificativa para a denominação de *nonlinear PCA* para o que é feito pela rede neural da Figura 5.15.

5.3.2 Modificações no algoritmo padrão CoACFNNA

A motivação para modificar o CoACFNNA padrão e permitir seu uso em problemas de redução de dimensão está nos benefícios do aprendizado construtivo, o qual tende a oferecer redes neurais parcimoniosas e melhor adaptadas aos requerimentos do problema, pontos que foram verificados nas simulações envolvendo problemas de regressão e classificação de dados. Um exemplo didático do que seria uma rede PCA-não-linear-ACFNN é mostrado na Figura 5.16.

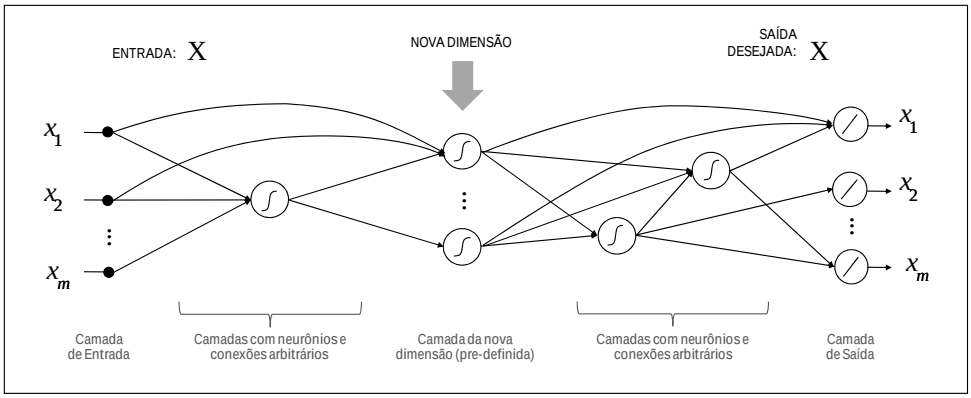


Figura 5.16 – Exemplo didático de *PCA* não-linear via uma rede do tipo ACFNN.

As considerações e modificações feitas nos procedimentos do algoritmo padrão CoACFNNA (apresentado no Capítulo 4) são descritas a seguir.

COMEÇA COM REDE VAZIA

Neste caso, considera-se como rede vazia a arquitetura com apenas neurônios nas camadas de entrada, de saída e na camada da *nova dimensão* (ver Figura 5.16). O número de neurônios da camada da *nova dimensão* é definido pelo usuário. Nenhuma conexão é estabelecida neste primeiro procedimento.

Cálculo de NMI e Adição de primeiras conexões

De forma análoga ao procedimento padrão do CoACFNNA, calcula-se o

NMI (segundo equação 4.7) para cada neurônio da camada de entrada. Em seguida, cada neurônio da camada da *nova dimensão* é conectado com o neurônio de entrada com maior NMI e com todos os neurônios da camada de saída. Finalmente, a rede é treinada.

Adição de Conexões

A modificação a ser realizada neste procedimento está associada unicamente à eliminação, da lista de conexões candidatas a serem inseridas (ver Figura 4.4), daquelas conexões que unem, de forma direta, qualquer par de neurônios antes e depois da camada da *nova dimensão*.

Adição de 1 Neurônio

Seguindo as ideias do procedimento padrão do CoACFNNA (ver Figura 4.5), se tentará inserir um neurônio *antes* e *depois* da camada da *nova dimensão*. Se for *antes*, pode contemplar os casos de incrementar cada camada de neurônios ocultos já existente ou criar uma nova camada antes da camada da *nova dimensão*. Se for *depois*, também se tentará incrementar cada camada oculta já existente ou criar uma nova camada antes da camada de saída. O caso que apresentar maior melhora na redução do erro será finalmente estabelecido. Em relação às conexões de entrada/saída deste novo neurônio, foram adotadas as seguintes considerações:

Eliminação de Conexões

A principal modificação realizada neste procedimento consiste em retirar da lista de conexões candidatas a serem eliminadas (ver Figura 4.6) aquelas conexões que levam ao caso de *desconectar* alguns dos neurônios da camada da *nova dimensão* (sem conexão de entrada ou saída).

Eliminação de 1 Neurônio

De forma análoga à eliminação de conexões, neste procedimento deve-se cuidar para não eliminar neurônios pertencentes à camada da *nova dimensão*. Em consequência, todos estes neurônios devem ser retirados da lista de neurônios a eliminar, como mostrado na Figura 4.7 para o caso do algoritmo CoACFNNA padrão.

5.3.3 Problemas de teste para redução de dimensão

Para testar o CoACFNNA em problemas de redução de dimensão, foram considerados os primeiros oito problemas das simulações envolvendo classificação de dados (ver Tabela

5.1 e Figura 5.1). O objetivo para estes experimentos é de reduzir a dimensão de entrada destes problemas para apenas duas dimensões (2D), com exceção do problema *Two Spirals* que será reduzido para uma dimensão, dado que sua dimensão original já é 2.

No CoACFNN, o parâmetro a ser definido pelo usuário, o *SEr_max*, foi inicializado em 1,05 para todos os problemas envolvendo redução de dimensão. Por outra parte, em relação à divisão dos conjuntos de treinamento, visto que os outros 3 algoritmos trabalham com o total de amostras, se optou por usar o total de amostras para compor o subconjunto de *treinamento*. O subconjunto de *validação* foi gerado a partir da inserção de ruído de até 20% sobre o total de amostras, visando evitar o sobre-ajuste (*over-training*).

Para o algoritmo de Sammon, foi estabelecido 25000 como o número máximo de iterações e parada antecipada se o erro variar abaixo de 1e-9, de uma iteração para outra. O número máximo de iterações para o CCA foi estabelecido em 500. Todas as simulações computacionais destes problemas foram realizadas num notebook com CPU Intel CORE i7 Q740 de 1,73Ghz e 4Gb de memória RAM, sendo que os algoritmos foram implementados em Matlab e fazem parte do toolbox SOM (http://www.cis.hut.fi/somtoolbox/).

As Figuras 5.18 a 5.24 apresentam os resultados de redução de dimensão para os problemas acima mencionados, utilizando: a nova versão de CoACFNNA, PCA, Sammon e CCA. Em cada figura, no gráfico do lado superior mostram-se as iterações (procedimentos) do algoritmo junto ao erro de aprendizado, seguindo o mesmo padrão adotado junto aos problemas de classificação de dados. Na parte superior-direita, mostra-se a rede do tipo ACFNN gerada e, na parte inferior da figura, a visualização das amostras na nova dimensão (2D) com cores representando cada classe: da esquerda para a direita, tem-se CoACFNNA, PCA, Sammon e CCA. Os tempos computacionais (em minutos) também são mostrados nos títulos de cada um destes gráficos. Para os resultados em 2D gerados pelo CoACFNNA, mostram-se também no título dos gráficos o MSE (*mean square error*) calculado a partir da saída desejada (os próprios dados de entrada) e a saída da rede gerada. Este cálculo não é possível obter a partir dos resultados dos outros algoritmos, já que eles

seguem outras diretivas para obter a redução de dimensão. A seta na figura da rede resultante indica a camada da nova dimensão.

Por outra parte, visando avaliar os resultados de uma forma quantitativa, optou-se por calcular um índice de erro de aproximação *EA* a partir das amostras na dimensão original e das amostras na dimensão reduzida. A partir de cada um destes conjuntos de amostras, calculam-se as matrizes de distâncias euclidianas ponto a ponto: **M** e **m**, respectivamente, normalizam-se ambas as matrizes numa mesma faixa de valores ([0 1]) e, em seguida, calcula-se o somatório dos erros quadráticos dos elementos da triangular superior, como indicado na Figura 5.17. Um cenário ideal seria aquele em que as amostras mantêm todas as suas distâncias (ponto a ponto) proporcionalmente iguais, tanto na dimensão real quanto na reduzida, resultando num *EA* nulo.

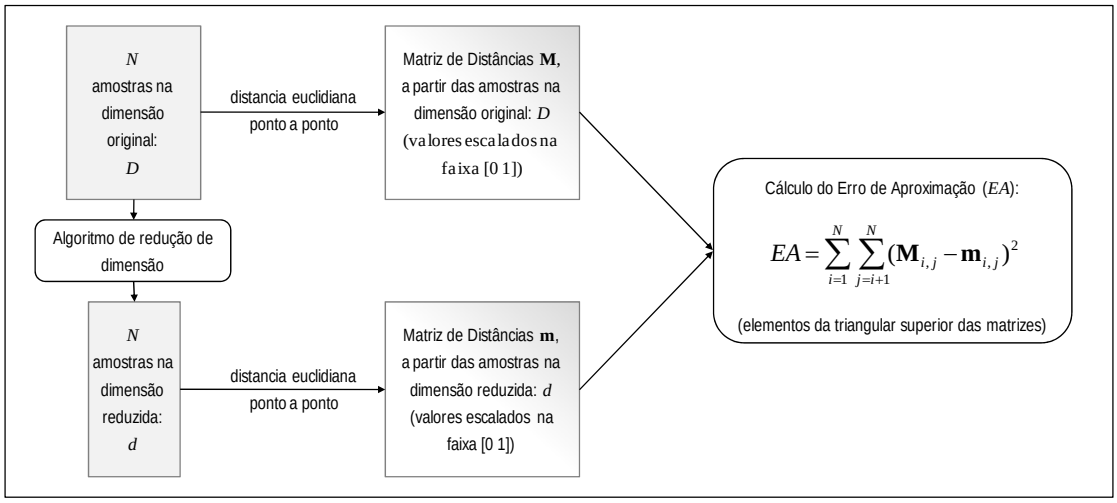


Figura 5.17 – Cálculo do erro de aproximação *EA*.

A seguir, a Tabela 5.6 mostra os erros de aproximação obtidos em cada caso.

Tabela 5.6 – Erro de aproximação quantitativo para os problemas de redução de dimensão.

Problemas	*fator*	CoACFNNA	PCA	Sammon	CCA
Two Spirals 2D	1,00E+04	2,1818	4,7919	4,7041	4,6949
Two Donuts 3D	1,00E+04	3,7693	5,6323	5,6763	5,5474
Three Spirals 3D	1,00E+05	0,7914	1,5102	1,6044	1,3965
Iris	1	159,4570	177,5955	36,9208	32,4161
Wine	1,00E+03	0,8521	1,5608	1,5237	1,5182
Breast Cancer Wisconsin	1,00E+03	7,4894	8,6025	1,1916	3,5375
Breast Tissue	1	103,1348	251,7770	348,3174	370,6843
Wall-following Robot Navigation	1,00E+05	1,1760	0,9935	0,4889	0,5144

O índice de erro de aproximação mostra que em 5 dos 8 problemas o CoACFNNA produziu redes do tipo ACFNNs que reduziram a dimensão dos dados gerando um *EA* menor do que aquele produzido pelos outros algoritmos. O algoritmo Sammon obteve o melhor índice em 2 casos e o CCA em 1 caso. Já o algoritmo PCA foi inferior a pelo menos um dos seus concorrentes em todos os casos.

A seguir, exibem-se os resultados para análise visual junto a cada um dos problemas considerados.

- *Two Spirals*: Com resultados mostrados na Figuras 5.18, ao reduzir sua dimensão para 1, nenhum dos algoritmos conseguiu isolar as classes. O CoACFNNA mostra dois grupos com amostras de ambas as classes em ambos os grupos e um MSE de 0,0073 e demorou 0,62 minutos. O algoritmo mais rápido foi o PCA com 0,02 minutos e o mais demorado o CCA, com 1,36 minutos. A rede gerada pelo CoACFNNA usou 4 neurônios ocultos antes da camada da nova dimensão e 1 neurônio oculto após esta camada.

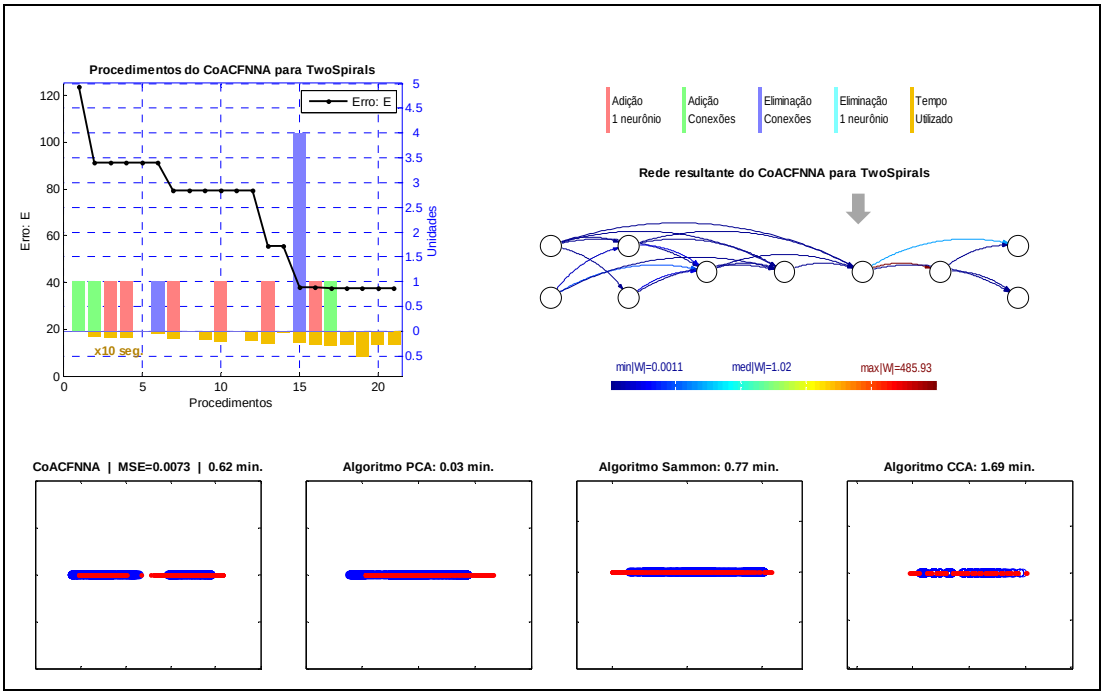


Figura 5.18 – Redução de 2D para 1D do problema *Two Spirals*.

- *Two Donuts*: O CoACFNNA gerou uma rede (ver Figura 5.19) sem nenhum neurônio oculto antes da camada da *nova dimensão* e três neurônios após ela, com MSE de 0,0014 em 0,24 minutos e cujas amostras visualizadas em 2D mostram-se mais descritivas da distribuição dos dados na dimensão original 3D (ver Figura 5.1). PCA e Sammon também mostram representações descritivas próximas do original: duas circunferências que se cruzam. Já o CCA mostra as classes em forma de "C", descrevendo com menor fidelidade as classes originais.

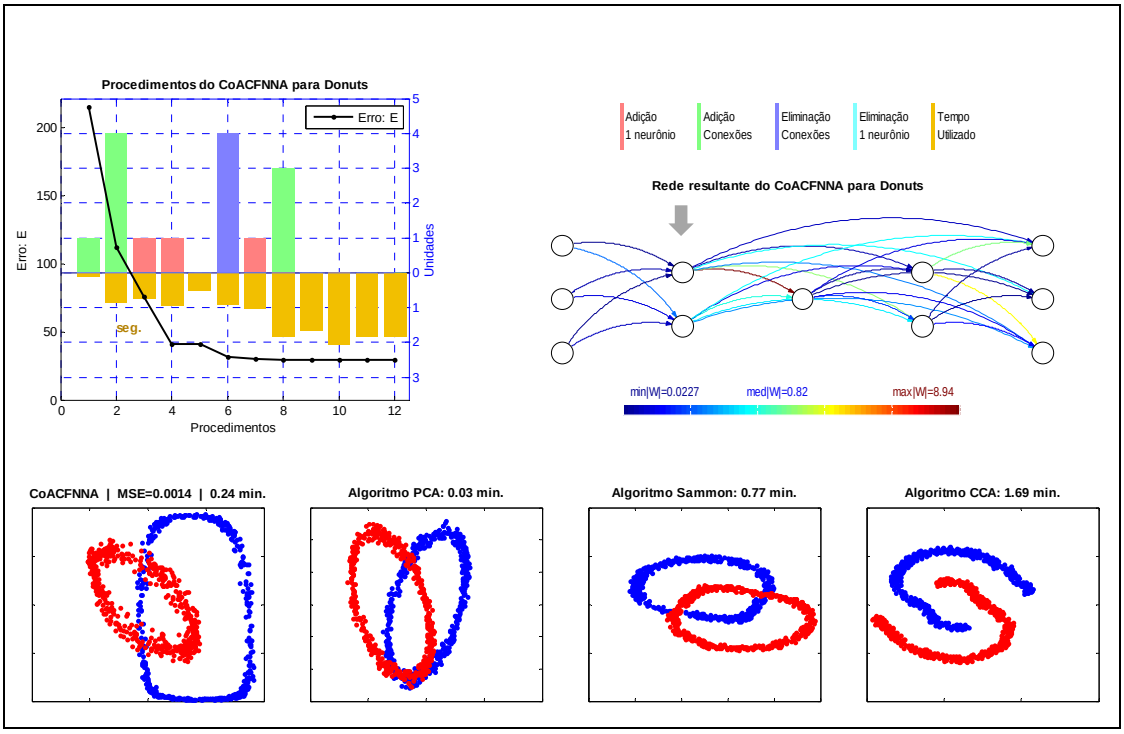


Figura 5.19 – Redução de 3D para 2D do problema *Donuts*.

- *Three Spirals*: Para este experimento, a rede produzida pelo CoACFNNA possui unicamente dois neurônios ocultos após a camada da nova dimensão, o MSE foi de 0,0245 e demorou 1,07 minutos. O resultado visual na nova dimensão (2D) é mais representativo do que os produzidos pelos outros algoritmos (como mostrado na Figura 5.20). O CCA foi o mais demorado com 5,99 minutos, seguido do Sammon com 2,95 minutos. O mais rápido foi o PCA, com 0,2 minutos.

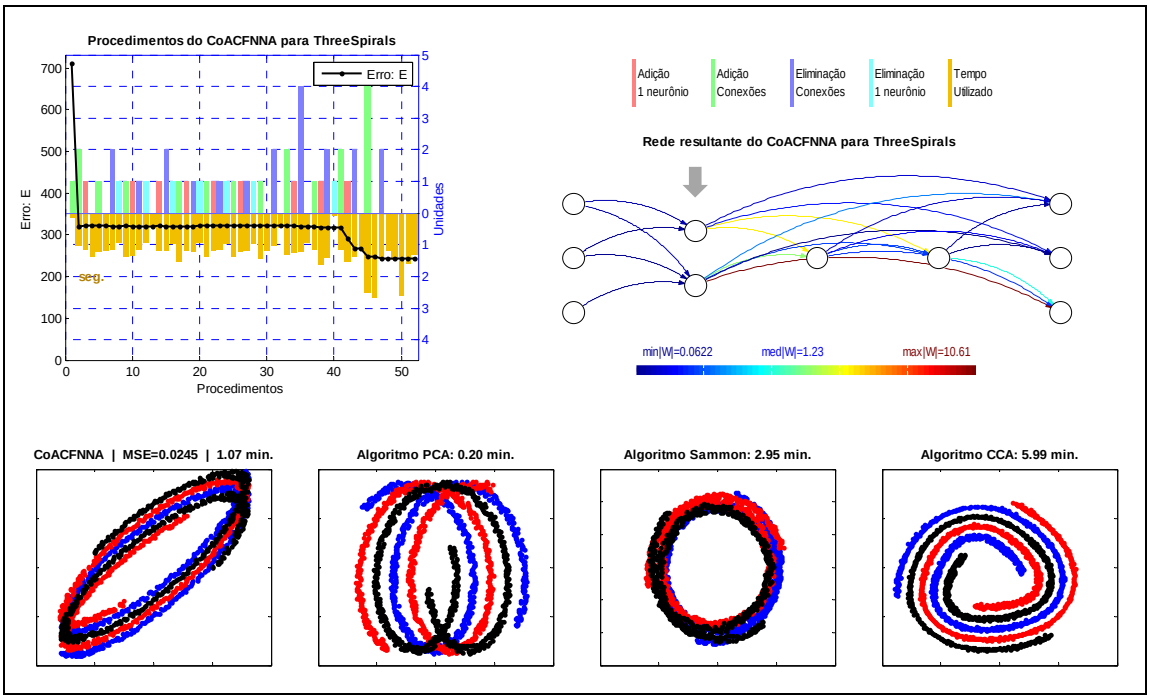


Figura 5.20 – Redução de 13 para 2 dimensões do problema *Three Spirals*.

- *Iris*: Todos os algoritmos mostraram em 2 dimensões resultados semelhantes para este problema, cuja dimensão original é 4, como mostrado na Figura 5.21. Neste problema, o CoACFNNA demorou mais que os outros competidores, com 1,29 minutos, utilizou 2 e 4 neurônios ocultos antes e depois da camada da nova dimensão, respectivamente, e o MSE foi de 0,0217.

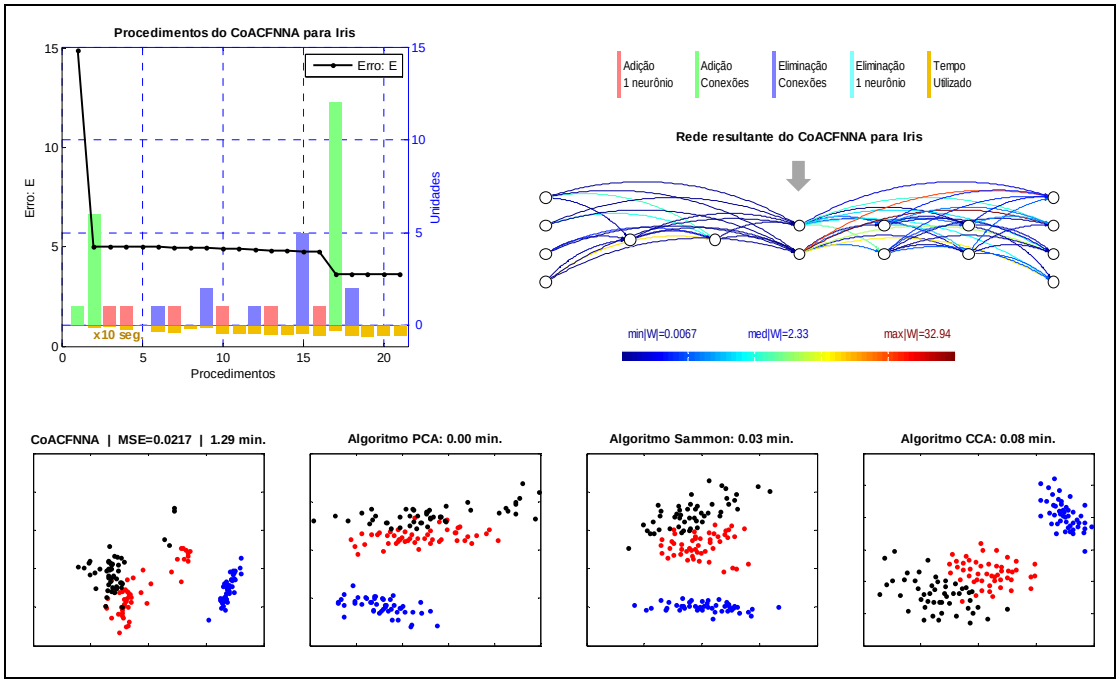


Figura 5.21 – Redução de 4D para 2D do problema *Iris*.

- *Wine*: Ao visualizar os resultados para este problema (Figura 5.22) pode-se notar que os algoritmos Sammon e CCA não conseguiram mostrar uma boa visualização dos dados em duas dimensões, pois ambos representaram os dados quase que formando uma linha de pontos. O desafio de reduzir o problema de 13 dimensões para 2 dimensões pode explicar este fato. O PCA e o CoACFNNA representam melhor os dados. No entanto, vale notar que o CoACFNNA mostra as classes com amostras em azul distante da classe com amostras em preto, colocando a classe em vermelho no meio das outras duas. Já os outros três algoritmos mostram as três classes sobrepostas e com a classe em preto no meio das outras duas classes. A rede ACFNN produzida possui 6 e 3 neurônios ocultos antes e depois da camada da *nova dimensão*, respectivamente, MSE de 1092,64 e demorou 5,48 minutos. Os outros algoritmos envolveram tempo computacional significativamente menor.

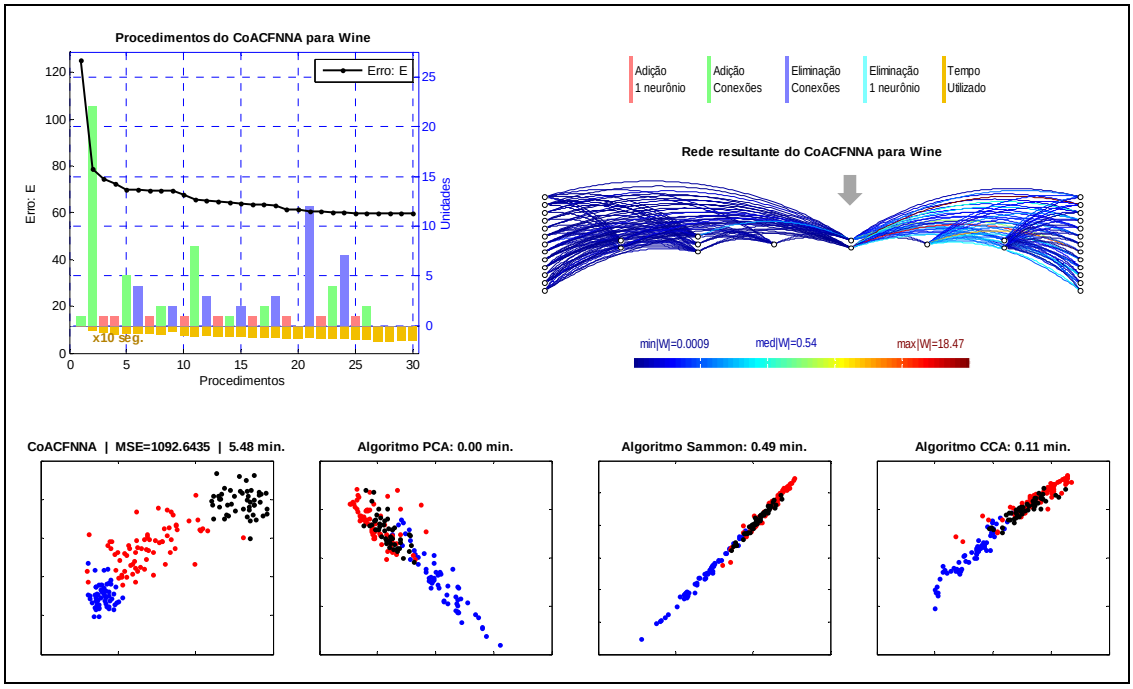


Figura 5.22 – Redução de 13D para 2D do problema *Wine*.

- *Breast Cancer Wisconsin*: Este problema é também desafiador por envolver uma redução de dimensão de 9 para 2. Os resultados são mostrados na Figura 5.23. Todos os algoritmos representaram a classe em azul mais densa e a classe em vermelho mais dispersa. PCA, Sammon e CCA representaram as duas classes com distribuições de formas arredondadas, sendo que o PCA mostrou maior sobreposição entre as duas classes. A rede produzida pelo CoACFNNA escalou 1 e 2 neurônios ocultos antes e depois da camada da *nova dimensão*, MSE de 1,69 e demandou maior tempo que os outros algoritmos, com 1,44 minutos.

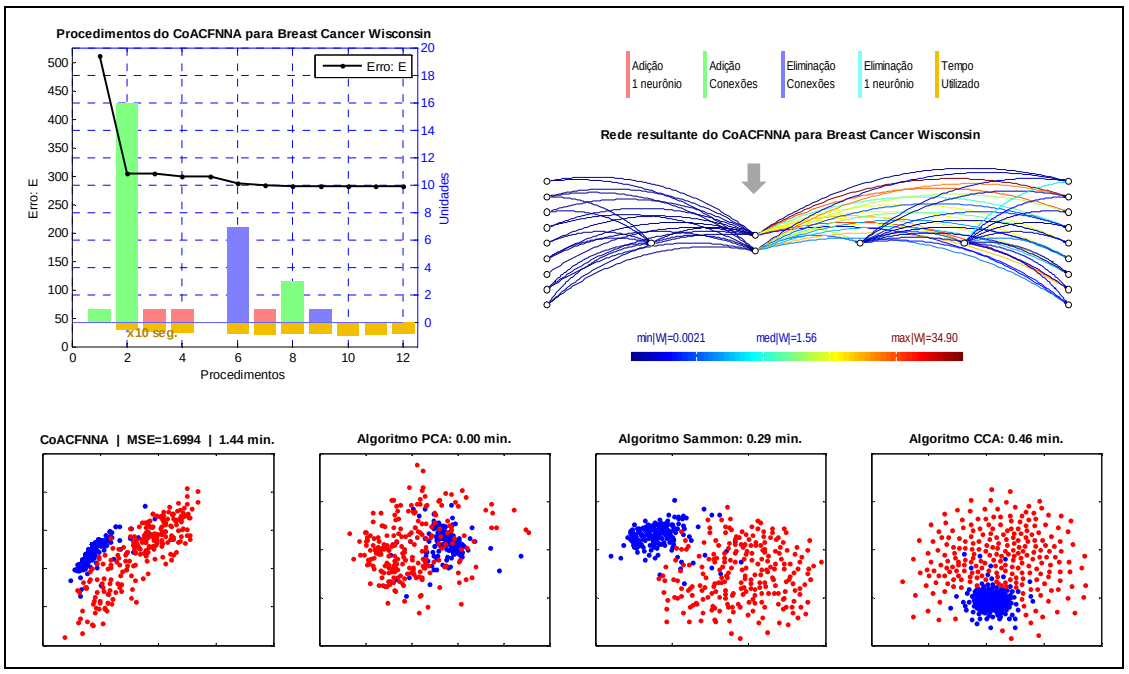


Figura 5.23 – Redução de 9D para 2D do problema *Breast Cancer Wisconsin*.

- *Breast Tissue*: Este problema já gerou uma série de dificuldades para os modelos classificadores e mostra também desafiador na redução da dimensão original 6 para 2. Aqui tem-se 6 classes e apenas 106 amostras. A Figura 5.24 mostra os resultados da redução de dimensão e apenas na representação obtida via o CoACFNNA consegue-se visualizar as 6 cores de amostras correspondentes às 6 classes do problema. Nos outros 3 algoritmos, ocorre forte sobreposição, sendo o pior caso o resultado do algoritmo de Sammon. A rede ACFNN produzida conta com 2 e 4 neurônios antes e depois da camada da nova dimensão, MSE de 1.535.410,15 e demandou 2,58 minutos.

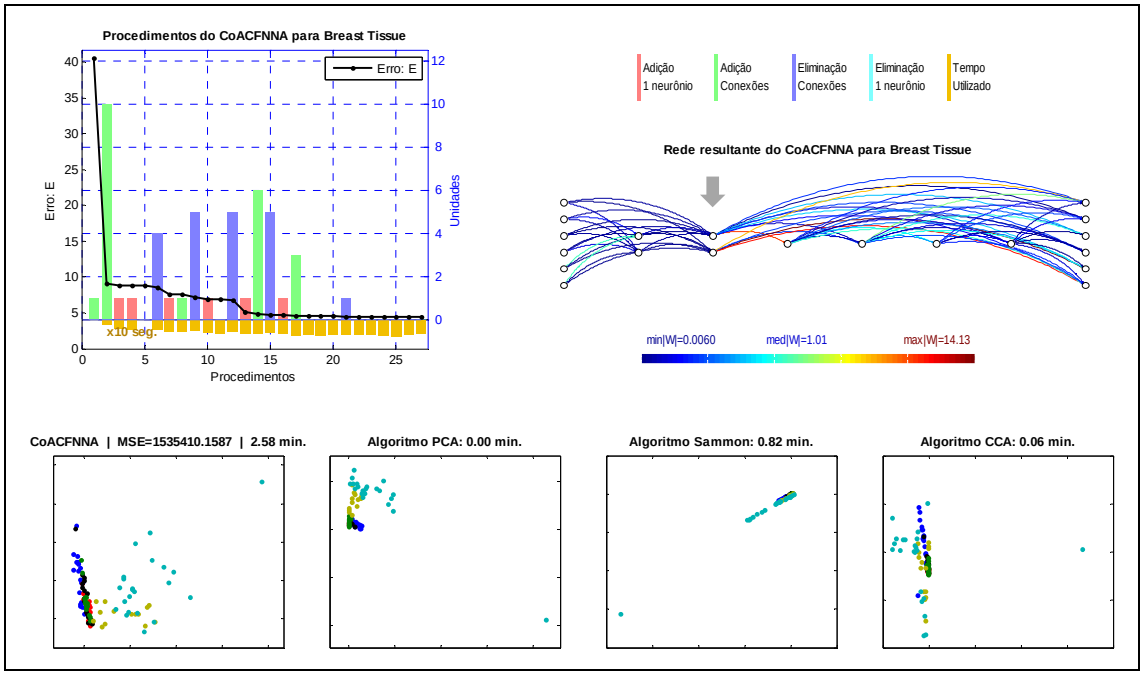


Figura 5.24 – Redução de 6D para 2D do problema *Breast Tissue*.

- *Wall-following Robot Navigation*: As informações visuais que se pode extrair dos resultados gerados pelos 4 algoritmos são similares. Por exemplo, todos mostram a classe em preto formando um grupo maior e outros 4 grupos pequenos. Um ponto que pode ser destacado é o tempo computacional demandado neste problema, que conta com 5456 amostras, o CoACFNNA demandou 1,10 minutos, PCA 5,13 minutos, Sammon 58,51 minutos e o CCA 52,35 minutos. Fica claro que os algoritmos Sammon e CCA são sensíveis, em maior grau, ao número de amostras do problema. A rede ACFNN produzida possui apenas dois neurônios após a camada da nova dimensão e MSE de 0,0991.

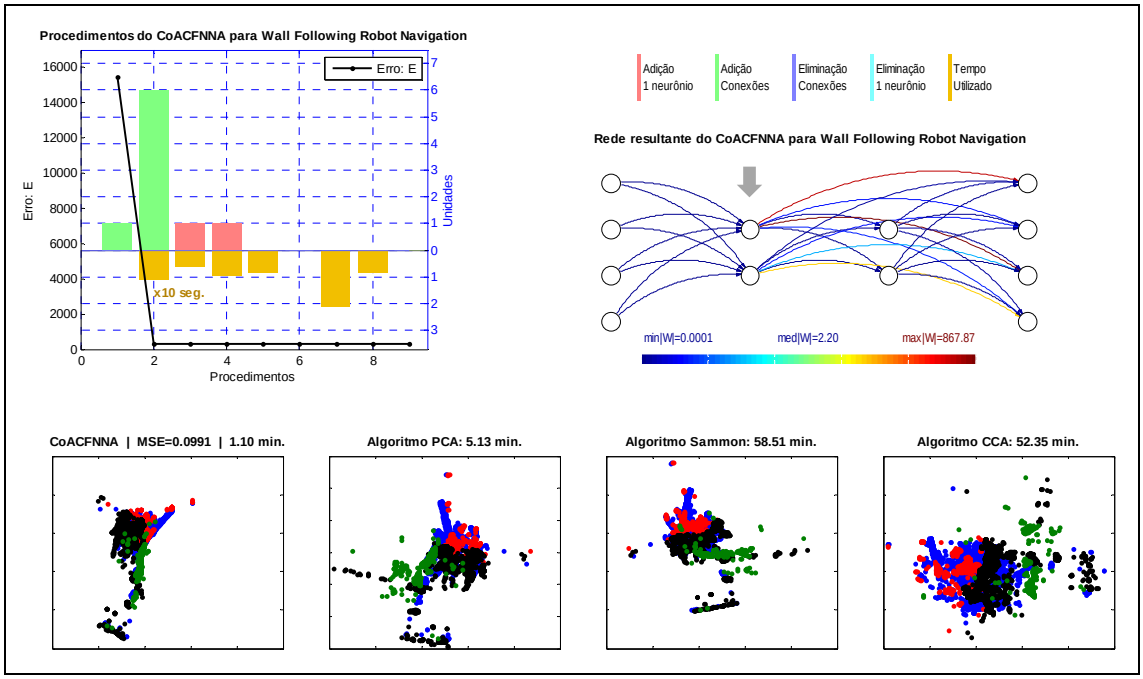


Figura 5.25 – Redução de 4D para 2D do problema *Wall-following Robot Navigation.*

5.4 Considerações finais do capítulo

Dentre as principais considerações em relação aos problemas de classificação de dados, tem-se que:

- Na maioria dos problemas testados, o CoACFNNA mostrou eficiência para produzir redes com as melhores taxas de classificação e parcimônia em suas arquiteturas.

- O CoACFNNA mostrou que o problema de sintetizar redes do tipo ACFNN pode ser bem resolvido sem precisar de propostas de busca populacionais, que exigem maior custo computacional, como, por exemplo, o EPNet, baseado em programação evolutiva.

Considerações finais em relação aos problemas de redução de dimensão:

- A abordagem de redução de dimensão usando RNAs deve continuar sendo mais explorada e utilizada, por permitir lidar de forma flexível com grandes quantidades de dados. Cabe salientar que o número de parâmetros a ajustar (pesos sinápticos) não depende do número de amostras do problema, o que não se aplica aos algoritmos de mapeamento tais como Sammon e CCA que, por exemplo, demandaram altos tempos computacionais no problema *Wall-following Robot Navigation*, o qual tem associado a si 5456 amostras.

- Em termos de avaliação visual dos resultados e avaliação quantitativa (via o índice de erro de aproximação *EA*) os resultados via o CoACFNNA mostraram melhor desempenho na maioria das vezes, o que indica que é uma boa proposta a ser considerada em estudos mais avançados.

- O emprego de conexões diretas, antes e depois da camada oculta da *nova dimensão*, permite explorar conjuntamente a ocorrência de associações lineares e não-lineares para se chegar a uma representação de dimensão reduzida. Logo, uma interpretação da arquitetura de rede neural obtida pode conduzir a novos conhecimentos qualitativos acerca da distribuição original dos dados.

Capítulo 6

Conclusões e perspectivas futuras

6.1 Conclusões

Levando em conta as considerações finais já levantadas ao final dos capítulos 3 ao 5, os quais descrevem as metodologias e estudos comparativos realizados, cabe salientar nas próximas subseções apenas os aspectos mais relevantes referentes à arbitrariedade da arquitetura de rede neural artificial e às duas metodologias propostas para explorar este grau de flexibilidade na modelagem.

6.1.1 Em relação às ACFNNs

Foi verificado o potencial das arquiteturas ACFNNs em relação ao que se obtém quando se consideram redes neurais MLP, mistura heterogênea de especialistas e *Cascade Correlation*, tanto em termos de desempenho quanto no uso de recursos computacionais para sintetizar e implementar a rede neural obtida.

Uma primeira linha de argumentação para explicar estes resultados se sustenta na arbitrariedade destas arquiteturas, o que introduz um grau de flexibilidade que pode ser devidamente moldado às demandas específicas de cada aplicação. Para alguns problemas, a ACFNN produzida apresentou poucas camadas ocultas e mais neurônios por camada. Em outros problemas, o resultado foi de mais camadas com poucos neurônios em cada uma.

Além disso, as conexões diretas entre as camadas de entrada e saída estiveram sempre presentes nas simulações, promovendo mapeamento linear de entrada-saída.

A segunda linha de argumentação se desprende da primeira nos seguintes aspectos:

(*i*) As MLP's e o *Cascade Correlation* são "casos particulares" de arquiteturas do tipo ACFNNs. Logo, se a demanda do problema assim indicar, a ACFNN pode convergir para uma arquitetura MLP ou CasCorr. E quando não convergem para uma delas é porque as decisões de inserção e poda de neurônios e conexões indicou caminhos mais promissores para se chegar à topologia final da rede neural.

(*ii*) No caso da mistura heterogênea de especialistas, sua arquitetura é modular e as funções de ativação dos neurônios de saída da rede *gating* são do tipo *softmax* (realizam uma combinação convexa dos valores das suas saídas). Sendo assim, não se trata de um caso particular de ACFNN, pois a estrutura hierárquica envolvendo especialistas e rede *gating* não representa blocos construtivos para as duas propostas de algoritmos voltados para a síntese de ACFNNs. No entanto, ambas as propostas trabalham com o princípio de dividir-para-conquistar, ao menos durante o processo de treinamento.

6.1.2 Em relação à metodologia baseada em computação evolutiva

Esta tese mostra que o problema de sintetizar uma rede neural do tipo ACFNN, com resultados promissores, pode não requerer todo o poder computacional demandado por metodologia baseadas em busca populacional. Visto que esta conclusão é embasada apenas em experiências comparativas envolvendo os algoritmos ACFNN-GA e EPNet, descritos no Capítulo 3, ela não poderia ser conclusiva. Além disso, há a possibilidade de surgir alguma nova proposta baseada neste tipo de abordagem que se mostre mais eficiente em termos de geração de ACFNNs com alta capacidade de generalização e parcimônia do modelo resultante. Já quanto a custo computacional, as abordagens baseadas em meta-

heurísticas e população de soluções candidatas tendem a ser mais custosa que abordagens construtivas, onde uma única solução é aprimorada.

O mais recente algoritmo GAME (KORDÍK *et al.*, 2010), descrito na seção 3.4, pode ser visto como uma das propostas mais elaboradas na literatura, pois aborda o problema de síntese de redes neurais com uma estratégia de *meta-aprendizado*, onde vários algoritmos de otimização (analíticos e aproximados) são executados e guiados por um algoritmo genético dotado de operadores avançados e com estratégia de *niching*. Embora os autores do GAME tenham testado o algoritmo junto a vários problemas, não realizaram um estudo comparativo com outras propostas da literatura, por exemplo, o EPNet, e não forneceram informações em relação ao tempo computacional demandado. Um dos problemas de teste em comum entre os utilizados para avaliar o desempenho do GAME e os problemas testados nesta tese é o *Breast Cancer Wisconsin*, no qual o GAME conseguiu uma média em torno de 96,3% de classificação correta, como produto de 10 execuções de validação do tipo *cross-fold* (*k-folds*: com $k = 10$), perante os 97,8% de classificação correta em média de 30 execuções do CoACFNNA, via validação cruzada com subconjuntos de *treinamento*, *validação* e *teste* divididos aleatoriamente em porcentagens de 50%-25%-25%, respectivamente.

6.1.3 Em relação à metodologia construtiva CoACFNNA

A estratégia de partir de uma rede vazia e ir adicionando, pouco a pouco, mais componentes (neurônios e conexões), buscando atender à demanda específica de um determinado problema, permite que o algoritmo possa convergir de forma consistente para arquiteturas mais parcimoniosas. Uma análise de pior caso, baseada em estudos realizados junto a redes do tipo *CasCorr*, indica que a taxa de redução do erro com a adição de novos nós é ao menos linear com o número de nós.

A capacidade de evitar a convergência prematura (fugir de mínimos locais não desejados) é alcançada não só pelo mecanismo de controle por relaxação do erro, mas também pelo uso de procedimentos de poda de conexões e de neurônios, sempre visando

ampliar a capacidade de generalização. Por exemplo, as conexões que são introduzidas junto com a adição de um novo neurônio podem não ser todas necessárias. Procedimentos de poda de conexões retiram da rede tanto as conexões redundantes (aquelas que na sua ausência não influenciam no erro, mas aumentam a parcimônia) quanto as conexões "conflitantes" (aquelas que ao serem removidas levam a uma redução do erro).

A escolha do método quasi-Newton para o ajuste dos pesos sinápticos das redes neurais no CoACFNNA mostrou-se acertada. Isto foi verificado no trabalho associado ao algoritmo GAME, onde os autores utilizaram 18 alternativas para ajustar os pesos de um único neurônio destinado a compor a rede. Dentre eles, um baseado em quasi-Newton e outro baseado em Estratégias Evolutivas CMA-ES (HANSEN & OSTERMEIER, 2001) foram os que produziram melhores resultados, sendo o método quasi-Newton mais rápido na maioria dos experimentos.

6.2 Perspectivas futuras

Antes de tratar dos novos desafios advindos das contribuições vinculadas a esta tese, cabe uma interpretação conceitual dos aspectos mais relevantes presentes nas metodologias propostas. Para tanto, a Figura 6.1 servirá de referência.

Na Figura 6.1, mostra-se, na parte A, a circunferência maior representando de forma ilustrativa o espaço de todas as soluções possíveis para mapeamento linear/não-linear de entrada-saída. Na parte inferior deste espaço, está indicada a parcela representando as soluções lineares e, na parte superior, as não-lineares. O círculo menor de cor preta, com a letra P, representa uma instância de solução de um problema qualquer, localizado na região de não-linearidade.

Na parte B da Figura 6.1, aparece um modelo (rede neural) representado pela letra M, que tenta resolver o problema P. Este modelo tem a capacidade de resolver de forma bem sucedida todos os problemas dentro de seu raio de cobertura (circulo cinza no entorno dele), ou seja, dentro de seu espaço de representação. Na prática, no entanto, é possível

cometer o que se convencionou denominar de erro de estimação, impedindo que o menor erro possível, denominado de erro de aproximação, seja atingido. O erro total obtido é então denominado de erro de generalização. Seguem definições mais formais para esses três tipos de erro:

- Erro de aproximação: Este erro surge inevitavelmente quando o projetista define a arquitetura do modelo M. Por exemplo, uma rede MLP com uma arquitetura com baixo número de neurônios na(s) camada(s) oculta(s) levaria a uma redução na flexibilidade do modelo, e problemas que requerem mapeamentos mais complexos podem não ser tratados de forma adequada, por mais eficaz que seja o método de treinamento da rede neural. Na parte B da Figura 6.1, o espaço de representação não contém a solução do problema, induzindo, assim, um erro de aproximação que não pode ser eliminado, por mais bem-sucedido que seja o treinamento da rede neural.
- Erro de estimação: Este erro está associado ao ajuste dos parâmetros do modelo (treinamento da rede), o qual pode não ser uma tarefa trivial, dado que a função-objetivo que guia esta etapa pode ser multimodal. A ocorrência de mínimos locais no processo de treinamento aumenta o risco de se cometer este erro, mesmo acertando na escolha do espaço de representação (o qual é determinado pela arquitetura da rede neural).
- Erro de generalização: Ou também erro de classificação ou de predição, dependendo do tipo de problema, é dependente de ambos os erros citados anteriormente.

Voltando à parte B da Figura 6.1, após a etapa de treinamento do modelo M, este conseguiu se aproximar da solução do problema P até o ponto em forma de estrela, cometendo um erro de generalização equivalente à somatória dos erros de estimação e aproximação.

A parte C da Figura 6.1 representa as propostas de comitês de máquinas (ensemble e mistura de especialistas), onde se ilustra a combinação de vários modelos, alguns deles possivelmente até de baixa capacidade de generalização, mas que juntos podem conduzir a

resultados que superam o desempenho individual de qualquer um dos componentes, reduzindo também a variância. No entanto, para esta abordagem ser promissora ela depende da participação do projetista na definição dos parâmetros do comitê, tanto do número e tipo de componentes, como dos parâmetros internos de cada um deles, pois podem estar presentes erros de aproximação que não promovem ganho de desempenho quando os componentes do comitê são devidamente combinados.

Finalmente, a parte D da Figura 6.1 representa os modelos com aprendizado construtivo, por exemplo, o CoACFNNA proposto nesta tese. Neste cenário, o aprendizado do modelo M parte da região de linearidade e o espaço de representação (representado pelos círculos em cinza a cada instante t_i, i=1,…,n) varia a cada iteração, ao contrário dos casos anteriores. E a redefinição do espaço de representação é feita iterativamente, sempre buscando reduzir de forma sistemática os erros de estimação e de aproximação. Nos casos anteriores, o processo de aprendizado levava em conta unicamente o erro de estimação.

No caso das metodologias baseadas em busca populacional (ACFNN-GA, EPNet e GAME), a sua representação seria parecida com aquela presente no gráfico da parte D da Figura 6.1, mas com vários modelos M inicializados em pontos distintos e geralmente aleatórios do espaço de todas as soluções possíveis. Novas gerações de soluções candidatas são então geradas a partir das propostas atuais na população, sempre visando reduzir os erros de estimação e aproximação.

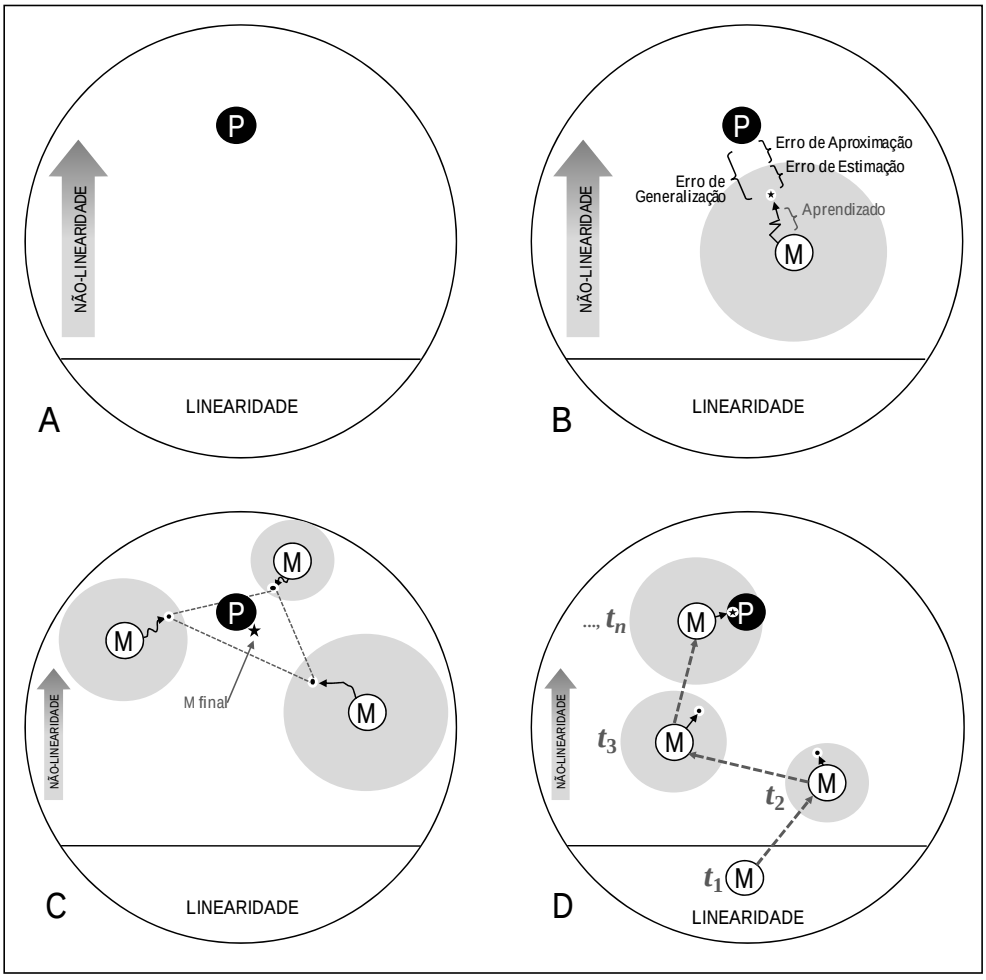


Figura 6.1 – Representações das metodologias no espaço dos problemas.

Os cenários considerados como perspectivas futuras deste trabalho são apresentados em duas partes, levando em conta o investimento que podem demandar para serem atendidos: perspectivas de curto prazo e perspectivas de médio a longo prazo.

6.2.1 Perspectivas de curto prazo

- Inclusão e avaliação de neurônios com outras funções de ativação não-lineares, desde que sejam diferenciáveis. Isto é possível porque o cálculo do vetor gradiente é feito

neurônio a neurônio. A motivação está na possibilidade de adotar melhores decisões locais durante o processo construtivo.

- Ampliação da flexibilidade da ACFNN ao permitir a existência de conexões recorrentes. Para isso, o passo fundamental está em propor uma nova forma de cálculo do vetor gradiente da função-objetivo, em relação aos pesos das conexões da rede neural.
- Extensão da proposta de algoritmo de treinamento de RNA baseado em regularização bayesiana (FORESEE & HAGAN, 1997) para treinar redes do tipo ACFNNs, junto a problemas de pequeno a médio porte (poucas amostras de treinamento). Em termos práticos, a principal vantagem desta proposta é a de não precisar um subconjunto de *validação* para controlar o problema de sobre-ajuste no treinamento. Os autores adotam a abordagem que insere um termo de penalização dos pesos sinápticos na função-objetivo, e os coeficientes de penalização são parte do processo de otimização. Em consequência, diminui-se o risco de sobre-ajuste e maximiza-se a capacidade de generalização. A desvantagem é que, por ser um método Levenberg-Marquardt, requer o cálculo da matriz hessiana, a qual é aproximada via Gauss-Newton utilizando a matriz jacobiana. Se o número de amostras for muito elevado, esta tarefa pode se tornar muito custosa computacionalmente, ou até inviável.

6.2.2 Perspectivas de médio a longo prazo

- Treinamento *on-line* no CoACFNNA. Este tipo de treinamento supervisionado é utilizado de forma bem-sucedida em redes com arquitetura previamente definida (SAAD, 1998). No entanto, no contexto de algoritmos construtivos, constitui um desafio em aberto, porque este tipo de treinamento se caracteriza por apresentar as amostras de treinamento à rede apenas uma única vez. Consequentemente, os procedimentos de agregação e poda de componentes da rede e os valores dos pesos sinápticos devem sofrer alterações apenas com a chegada de uma única amostra. As aplicações deste tipo de treinamento estão focadas em problemas com mudanças de regime e que exigem

respostas imediatas, para os quais não há tempo e/ou memória disponível para executar o treinamento *off-line* (em batelada).

- Ajuste dos pesos sinápticos de modelos não-lineares com otimalidade global. Em termos práticos, isto significaria que, independente da complexidade do problema e da inicialização dos pesos sinápticos de uma arquitetura de rede previamente definida, o treinamento deverá conduzir a um mesmo resultado (ótimo global). Na literatura, este desafio não foi superado ainda. Há propostas utilizando meta-heurísticas de busca populacional que demandam recursos computacionais elevados e que não garantem o cumprimento das premissas levantadas neste desafio.
- Estudo mais formal da metodologia de redução de dimensão proposta no capítulo 5. Diferente de outras abordagens, a representação compacta das variáveis de entrada pode se dar por um mapeamento linear ou não-linear, e a recuperação da informação original, com alguma perda, pode se dar por mapeamento linear ou não-linear. Se ambos forem mapeamentos lineares, então converge-se automaticamente para o PCA. Se ambos forem não-lineares, converge-se automaticamente para o *nonlinear PCA*. Mas se apenas um deles for não-linear, não existem modelos equivalentes na literatura, representando uma nova forma de concepção do processo de redução de dimensão, a qual emerge do próprio processo construtivo de síntese da topologia da rede neural.

Referências Bibliográficas

ALMEIDA, L. B. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. *Proceeding of the IEEE First International Conference on Neural Networks*, pp. 609-618, 1987.

ANGELINE, P. J.; SAUDERS, G. M. & POLLACK, J. B. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transaction on Neural Networks*, vol. 5, pp. 54-65, 1994.

BÄCK, T., FOGEL, D. B. & MICHALEWICZ, Z. (eds.) **Handbook of Evolutionary Computation**. Institute of Physics Publishing, Bristol, Philadelphia and Oxford University Press, 1997.

BARTO, A. G. & ANANDAN, P. Pattern-recognizing stochastic learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 15, no. 3, pp. 360-375, 1985.

BATTITI, R. First and second-order methods for learning: between steepest descent and Newton's method. *Neural Computation*, vol. 4, no. 2, pp. 141-166, 1992.

BAZARAA, M. S.; SHERALI, H. D. & SHETTY, C. M. **Nonlinear Programming - Theory and Algorithms**, John Wiley & Sons, 1993.

BEIGI, H. S. M. Learning algorithms for neural networks based on quasi-Newton methods with self-scaling. *Transactions of the ASME*, vol. 115, pp. 38-43, 1993.

BELEW, R. K.; MCINERNEY, J. & SCHRAUDOLPH, N. N. Evolving networks: Using genetic algorithm with connectionist learning, Computer Sci. Eng. Dept., Univ. California – San Diego, Tech. Rep. CS90-174 revised, Feb. 1991.

BOSCOLO, R.; PAN, H. & ROYCHOWDHURY, V. P. Non-parametric ICA. *Proceedings of the Third International Conference on Independent Component Analysis and Blind Signal Separation*, pp. 13-18, 2001.

BOX, G. E. P.; JENKINS, G. M. & REINSEL, G. C. **Time Series Analysis: Forecasting and Control**. 3nd ed. Holden-Day, 1994.

BROOMHEAD, D. S. & LOWE, D. Multivariate functional interpolation and adaptive networks. *Complex Systems*, vol. 2, pp. 321-355, 1988.

CARPENTER, G. A. & GROSSBERG, S. The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network. *IEEE Computer*, vol. 21, no. 3, pp.77-88, 1988.

CAUCHY, A. Méthode générale pour la résolution des systèms d'equations simultanées, Comp. Rend. Sci. Paris, vol. 25, pp. 46-89, 1847.

CELLUCCI, C. J.; ALBANO, A. M. & RAPP, P. E. Statistical validation of mutual information calculations: Comparison of alternative numerical algorithms. *Physical Review E*, vol. 71, issue 6, id. 066208, pp. 1-14, 2005.

CHAPELLE, O.; SCHÖLKOPF, B. & ZIEN, A. **Semisupervised learning: Adaptive computation and machine learning**. Cambridge, Mass., USA: MIT Press, 2006.

CHERKASSKY, V. & MULIER, F. **Learning from Data: Concepts, Theory, and Methods**, Wiley-IEEE, 2007.

COMON, P. Independent component analysis – A new concept? *Signal Processing*, vol. 36, pp. 287–314, 1994.

CONNOR, J. T. & MARTIN, R. D. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, vol. 5, pp. 240-254, 1994.

COVER, T. & THOMAS, J. **Elements of Information Theory**. John Wiley & Sons, New York, NY, 1991.

CYBENKO, G. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control Signals and Systems*, vol. 2, pp. 303-314, 1989.

DEMARTINES, P. & HÉRAULT, J. CCA: Curvilinear component analysis. In *15th Workshop GRETSI*, Juan-les-Pins (France), September, 1995.

DEMPSTER, A. P.; LAIRD, N. M. & RUBIN, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B, vol. 39, no 1, pp. 1-38, 1977.

DENNIS, J. E. & SCHNABEL, R. B. **Numerical Methods for Unconstrained Optimization and Nonlinear Equations**. Prentice-Hall, Englewood Cliffs, NJ, USA, 1983. Reprinted as Classics in Applied Mathematics 16, SIAM, Philadelphia, USA, 1996.

DIETTERICH, T. G. Ensemble methods in machine learning. In *Proceedings of the International Workshop on Multiple Classifier Systems (MCS), LNCS 1857*, pp. 1-15, Italy, Springer, 2000.

DOS SANTOS, E. P. & VON ZUBEN, F. J. Efficient Second-Order Learning Algorithms for Discrete-Time Recurrent Neural Networks. In: L. R. Medsker; L. C. Jain. (Org.). Recurrent Neural Networks Design and Applications. CRC Press International, vol. 1, pp. 47-75, 2000.

DRAGO, G. P. & RIDELLA, S. Convergence properties of Cascade Correlation in function approximation. *Neural Computing & Applications*, vol. 2, pp. 142-147, 1994.

DRAGO, G. P. & RIDELLA, S. On the convergence of a growing topology neural algorithm. *Neurocomputing*, vol. 12, pp. 171-185, 1996.

FAHLMAN, S. E. Faster-learning variations on backpropagation: an empirical study. In: Touretzky, D.S., Hinton, G.E., Sejnowski, T.J. (eds). *Proceedings of the 1988 Connectionst Models Summer School*, pp. 38-51. Morgan Kaufmann, San Mateo, 1988.

FAHLMAN, S. E. & LEBIERE, C. The cascade correlation architecture. Advances in Neural Information Processing Systems 2, Morgan Kaufman, San Mateo, pp. 524-532, 1990.

FIESLER, E. Comparative bibliography of ontogenic neural networks. *Proceedings of the International Conference on Artificial Neural Networks* (ICANN), Sorrento, vol. 94, pp. 793-796, 1994.

FINNOFF, W.; HERGENT, F. & ZIMMERMANN, H. G. Improving model selection by nonconvergent methods. *Neural Networks*, vol. 6, pp. 771-783, 1993.

FOGEL, D. B. An Introduction to Simulated Evolutionary Optimization. *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 3-14, 1994.

FOGEL, D. B. **Evolutionary Computation: Toward a New Philosophy of Machine Intelligence**. IEEE Press, Piscataway, USA, 1995.

FOGEL, L. J. Autonomous Automata. *Industrial Research*, vol. 4, pp. 14-19, 1962.

FOGEL, L. J. **On the Organization of Intellect**, Ph.D. Thesis, University of California, CA, USA, 1964.

FORESEE, F. D. & HAGAN, M. T. Gauss-Newton approximation to Bayesian regularization. Proceedings of *IEEE International Joint Conference on Neural Networks*, vol. 3, pp. 1930-1935, 1997.

FRANCO, L.; ELIZONDO, D. A. & JEREZ, J. M. **Constructive Neural Networks**. Studies in Computational Intelligence, vol. 258, Springer, 2009.

FRANK, A. & ASUNCION, A. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2010.

FRASER, A. M. & SWINNEY, H. L. Independent coordinates for strange attractors from mutual information. *Physical Review A*, vol. 33, pp. 1134-1140, Feb. 1986.

FREAN, M. The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation*, vol. 2, pp. 198-209, 1990.

FRITZKE, B. A. Growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems* vol. 7, MIT Press, Cambridge MA, pp. 625-632, 1995.

FUNAHASHI, K. On the approximate realization of the continuous mappings by neural networks. *Neural Networks*, vol. 2, pp. 183-192, 1989.

GALLANT, S. I. Optimal linear discriminants. *Proceedings of the Eighth International Conference on Pattern Recognition*, pp. 849-852, 1986.

GALLANT, S. I. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, vol. 1 no. 2, pp. 179-191, 1990.

GALLANT, S. I. **Neural Network Learning & Expert Systems**. The MIT Press, England, 1994.

GEMAN, S.; BIENENSTOCK, E. & DOURSAT, R. Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, vol. 4 , no. 1, pp. 1-58, 1992.

GHOSH, J. & TUMER, K. Structural adaptation and generalization in supervised feed-forward networks. *Journal of Artificial Neural Networks*, vol. 1, no. 4, pp. 431-458, 1994.

GOLDBERG, D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, MA, 1989.

GOMES, L. C. T. & VON ZUBEN, F. J. Vehicle routing based on self-organization with and without fuzzy inference. Proceedings *of the IEEE International Conference on Fuzzy Systems* (FUZZ-IEEE'2002), in the 2002 IEEE World Congress on Computational Intelligence (WCCI'2002), Honolulu, Hawaii, vol. 2, pp. 1310-1315, May 12-17, 2002.

GONÇALVES, R.; VON ZUBEN, F. J. & GOMIDE, F. Using Constructive Learning in Embedded Systems Engineering. *Proceedings of the IEEE International Joint Conference on Neural Networks*, Anchorage, Alaska, vol. 1, pp. 251-255, 1998.

GUYON, I. & ELISSEEFF, A. An introduction to variable and feature selection. *Journal of Machine Learning Research*, vol. 3, pp. 1157-1182, 2003.

HAGAN, M. T. Training Feedforward Networks with the Marquardt Algorithm, *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989-993, 1994.

HAGAN, M. T.; DEMUTH, H. B. & BEALE, M. H. **Neural Network Design**. PWS Publishing, Boston, 1996.

HANCOCK, P. J. B. Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification. *Proceedings of*

International Workshop of Combinations of Genetic Algorithms and Neural Networks (COGANN-92), Los Alamitos, CA: IEEE Computer Soc. Press, pp. 108-122, 1992.

HANSEN, N. & OSTERMEIER, A. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, vol. 9, no. 2, 159-195, 2001.

HAYKIN, S. **Unsupervised Adaptive Filtering, Volume 1: Blind Source Separation**, John Wiley & Sons, 2000.

HAYKIN, S. **Neural Networks and Learning Machines**. Prentice-Hall, 3rd Edition, 2008.

HEBB, D. O. **Organization of Behaviour: A Neuropsychological Theory**. Wiley, New York, 1949.

HINTON, G. E. & SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science Magazine*, vol. 313. no. 5786, pp. 504-507, 2006.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. University of Michigan Press, 1975.

HOLSCHUH, L. M. Contribuições para o Aprendizado por Busca de Projeção. Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 2008.

HOPFIELD, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, vol. 79. pp. 2554-2558, April, 1982.

HORNIK, K.; STINCHOMBE, M. & WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, vol. 2, no. 5, pp. 359-366, 1989.

HWANG, J.-N.; LAY, S.-R.; MAECHLER, M.; MARTIN, R. D. & SCHIMERT, J. Regression Modeling in Back-Propagation and Projection Pursuit Learning, *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 342-353, 1994.

JACOBS, R. A.; JORDAN, M. I.; NOWLAN, S. J. & HINTON, G. E. Adaptive mixtures of local experts. *Neural Computation*, vol. 3, no 1, pp. 79-87, 1991.

JAIN, A. K. & MAO, J. Artificial Neural Networks: A Tutorial. *IEEE Computer*, pp. 31-44, 1996.

JORDAN, M. I. & JACOBS, R. A. Hierarchical Mixtures of experts and the EM algorithm. *Neural Computation*, MIT Press , vol. 6, pp. 181-214, 1994.

JOYA, G.; ATENCIA, M. A. & SANDOVAL, F. Hopfield neural network applied to optimization problems: Some theoretical and simulation results. Biological and Artificial Computation: From Neuroscience to Technology, Lecture Notes in Computer Science, vol. 1240/1997, pp. 556-565, 1997.

JUTTEN, C. & HÉRAULT, J. Blind separation of sources, Part I: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, vol. 24, pp. 1-10, 1991.

KIRKPATRICK, S.; GELATT, C. D. & VECCHI, M. P. Optimization by Simulated Annealing. *Science*, vol. 220, no. 4598, pp. 671-680, 1983.

KOHONEN, T. **Self-Organization and Associative Memory**. Springer-Verlag New York, 3rd edition, New York, 1989.

KORDÍK, P.; KOUTNÍK, J.; DRCHAL, J.; KOVÁRÍK, O.; CEPEK, M. & SNOREK, M. Meta-learning approach to neural network optimization. *Neural Networks*, vol. 23, no. 4, pp. 568-582, 2010.

KRAMER, M. A. Nonlinear principal component analysis using autoassociative neural networks. *AIChE J.*, vol. 37, pp. 233-243, 1991.

KUNCHEVA, L. I. Classifier ensembles for changing environments. *5th International Workshop on Multiple Classifier Systems*, Lecture Notes in Computer Science, vol. 3077, pp. 1-15, 2004.

LAHNAJÄRVI, J. J. T.; LEHTOKANGAS, M. I. & SAARINEN, J. P. P. Evaluation of constructive neural networks with cascade architectures. *Neurocomputing*, vol. 48, pp. 573-607, 2002.

LEE, J. A. & VERLEYSEN, M. **Nonlinear Dimensionality Reduction**. Springer, 2007.

LI, C.; LI, S.; ZHANG, D. & CHEN, G. Cryptanalysis of a Chaotic Neural Network Based Multimedia Encryption Scheme. *Advances in Multimedia Information Processing - PCM 2004 Proceedings*, Part III, volume 3333 of Lecture Notes in Computer Science, pp. 418-425, 2004.

LIMA, C. A. M. Comitê de máquinas: Uma abordagem unificada empregando máquinas de vetores-suporte. Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 2004.

LUENBERGER, D. G. **Introduction to Linear and Nonlinear Programming**. Addison-Wesley, New-York, 1973.

MAHFOUD, S. W. Niching methods for genetic algorithms. *Technical Report* 95001. Illinois Genetic Algorithms Laboratory (IlliGaL), University of Illinois at Urbana-Champaign, 1995.

MANIEZZO, V. Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions Neural Networks*, vol. 5, pp. 39-53, 1994.

MCLACHLAN, G. J. & BASFORD, K. E. **Mixture Models: Inference and Applications to Clustering**. Marcel Dekker, Inc., New York, 1988.

MENDEL, J. M. Tutorial on Higher-Order Statistics (Spectra). In Signal Processing and System Theory: Theoretical Results and Some Applications. *Proceedings of the IEEE*, vol. 79, pp. 278-305, 1991.

MICHALEWICZ, Z. **Genetic algorithms + Data Structures = Evolution Programs**. Springer-Verlag, 3rd edition, 1996.

MITCHELL, T. **Machine Learning**. New York: McGraw-Hill, 1997.

MUSELLI, M. Sequential constructive techniques. In: Leondes, C. (ed.) Neural Network Systems Techniques and Applications, vol. 2, pp. 81-144. Academic, San Diego, 1998.

NARENDRA, K. S. & THATHACHAR, M. A. L. Learning automata: A survey. *IEEE Transactions in Systems, Man and Cybernetics*, vol. SMC-4, no. 4, 1974.

NARENDRA, K. S. & PARTHASARATHY, K. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, vol. 1, pp. 4-27, 1990.

NARENDRA, K. S. & PARTHASARATHY, K. Gradient methods for the optimisation of dynamical systems containing neural networks. *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 252-262, 1991.

NØRGAARD, M.; RAVN, O.; POULSEN, N. K. & HANSEN, L. K. **Neural Networks for Modelling and Control of Dynamic Systems**. Springer-Verlag, London, 2000.

OREN, S. S. On the selection of parameters in self scaling variable metric algorithms. *Mathematical Programming*, vol. 7, pp. 351-367, 1974.

PEARLMUTTER, B. A. Fast Exact Multiplication by the Hessian. *Neural Computation*, vol. 6, no. 1, pp. 147-160, 1994.

PERRONE, M. P. & COOPER, L. N. When networks disagree: Ensemble methods for hybrid neural networks. In R. J. Mammone, editor, **Neural Networks for Speech and Image Processing**, chapter 10. Chapman-Hall, 1993.

PINEDA, F. J. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, vol. 59, no. 19, pp. 2229-2232, 1987.

POULARD, H. & LABRECHE, S. A new threshold unit learning algorithm. *Technical Report* 95504, LAAA-CNRS, Toulouse, France, 1995.

PRECHELT, L. Proben1 – A set of neural network benchmark problems and benchmarking rules. Fakultät für Informatik, Univ. Karlsruhe, Karlsruhe, Germany, Tech. Rep. 21/94, Sept. 1994.

PRECHELT, L. Early Stopping – but when?, *Technical Report*, URL: http://wwwipd.ira.uka.de/~prechelt/Biblio/stop_tricks1997.ps.gz, 1997.

PRECHELT, L. Automatic early stopping using cross validation: Quantifying the criteria. *Neural Networks*, vol. 11, no 4, pp. 761-767, 1998.

PRINCIPE, J. C. **Information Theoretic Learning: Renyi's Entropy and Kernel Perspectives**. Springer, 2010.

PUMA-VILLANUEVA, W. J.; LIMA, C. A. M.; DOS SANTOS, E. P. & VON ZUBEN, F. J. Mixture of Heterogeneous Experts Applied to Time Series: A Comparative Study. *Proceedings of the IEEE International Joint Conference on Neural Networks - IJCNN*, vol. 1, pp. 1160-1165, Montreal, 2005.

PUMA-VILLANUEVA, W. J. & VON ZUBEN, F. J. Evolving arbitrarily connected feedforward neural networks via genetic algorithms. *Brazilian Symposium on Artificial Neural Networks* (SBRN), São Bernardo do Campo - SP, October 23-28, 2010.

PUMA-VILLANUEVA, W. J.; DOS SANTOS, E. P. & VON ZUBEN, F. J. A constructive algorithm to synthesize arbitrarily connected feedforward neural networks. *Neurocomputing*, vol. 75, pp. 14-32, 2012.

RAMAMURTI, V. & GHOSH, J. Structural adaptation in mixture of experts. *Proceedings of the 13th International Conference on Pattern Recognition*, vol. 4, pp. 704-708, 1996.

RAO, S. S. **Engineering Optimization Theory and Practice**. John Wiley & Sons, Inc., Hoboken, New Jersey, 4th edition, 2009.

RAVIV, Y. & INTRATOR, N. Variance reduction via noise and bias constraints. In *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems* (ed. A. J. C. Sharkey). London, Springer-Verlag, pp. 163-175, 1999.

REED, R. Pruning algorithms - a survey. *IEEE Transaction on Neural Networks*, vol. 4, no. 5, pp. 740-747, 1993.

RUMELHART, D. E. & MCCLELLAND, J. L. **Parallel distributed processing: Exploration in the microstructure of cognition**. *MIT Press*, Cambridge, Massachussetts, vol. 1, 1986.

SAAD, D. **On-line learning in neural networks**. Publications of the Newton Institute, Cambridge University Press, 1998.

SAMMON, J. W. A nonlinear mapping algorithm for data structure analysis. *IEEE Transactions on Computers*, vol. CC-18, no. 5, pp. 401-409, 1969.

SCHAFFER, J. D.; WHITLEY, D. & ESHELMAN, L. J. Combinations of genetic algorithms and neural networks: A survey of the state of the art. *Proceedings of the International*

Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92), D. Whitley and J. D. Schaffer, Eds. Los Alamitos, CA: IEEE Computer Soc. Press, pp. 1-37, 1992.

SCHOLZ, M.; KAPLAN, F.; GUY, C. L.; KOPKA, J. & SELBIG, J. Non-linear PCA: a missing data approach. *Bioinformatics*, vol. 21, no. 20, pp. 3887-3895, Oxford University Press, 2005.

SHANNON, C. E. & WEAVER, W. **The mathematical theory of communication**. Urbana IL: University of Illinois Press, 1949.

SHEWCHUK, J. R. An introduction to the conjugate gradient method without the agonizing pain. *Technical Report*. School of Computer Science Carnegie Mellow University, Pittsburgh, PA 15213, 1994.

SRINIVAS, M. & PATNAIK, L. M. Genetic algorithms: a survey. *Computer*, vol. 27, no. 6, p. 17-26, June, 1994.

VAN DER SMAGT, P. Minimization methods for training feedforward neural networks. *Neural Networks*, vol. 7, no. 1, pp. 1-11, 1994.

VON ZUBEN, F. J. Modelos Paramétricos e Não-Paramétricos de Redes neurais Artificiais e Aplicações, Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 1996.

WADE, J. G. Convergence properties of the conjugate gradient method. Available at www-math.bgsu.edu/-gwade/tex_examples/example2.txt, 2006.

WEIGEND, A. S.; MANGEAS, M. & SRISTAVA, A. N. Nonlinear gated experts for time series: Discovering regimes and avoiding over fitting. *International Journal of Neural Systems*, vol. 6, pp. 373-399, 1995.

WERBOS, P. **Beyond regression: New tools for prediction and analysis in the behavioral sciences**. Ph.D. Thesis, Harvard University, 1974.

WHITLEY, D.; STARKWEATHER, T. & BOGART, C. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, vol. 14, pp. 347-361, 1990.

WIDROW, B. & STEARNS, S. D. **Adaptive Signal Processing**. New York: Prentice-Hall, 1985.

WIENER, N. **Extrapolation, Interpolation, and Smoothing of Stationary Time Series with Engineering Applications**. MIT Press, 1949.

WILAMOWSKI, B. M.; COTTON, N. J.; KAYNAK, O. & DUNDAR, G. Computing gradient vector and Jacobian matrix in arbitrarily connected neural networks. *IEEE Transactions on Industrial Electronics*, vol. 55, no. 10, pp. 3784-3789, 2008.

XU, L.; JORDAN, M. I. & HINTON, G. E. An Alternative model for mixtures of experts. *Advances in Neural Information Processing Systems 7*, eds., Cowan; J. D., Tesauro, G.; Alspector, J., MIT Press, Cambridge MA, pp. 633-640, 1995.

YAO, X. & SHI, Y. A preliminary study on designing artificial neural networks using coevolution. *Proceedings of IEEE Singapore International Conference on Intelligent Control and Instrumentation*, Singapore, pp 149-154. June, 1995.

YAO, X. & LIU, Y. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694-713, 1997.

ZHANG, G. P., PATUWO, B. E. & HU, M. Y. Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, vol. 14, no. 1, pp. 35-62, 1998.

ZHANG, G. P. & QI, M. Neural network forecasting for seasonal and trend time series. *European Journal of Operation Research*, vol. 160, pp. 501-514, 2005.