Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

SimNeural: Simulador Computacional de RNA para Aplicações em Ensino e Pesquisas Médicas e Biológicas

Bruno Bastos Neves
Jussiê Marques Martins

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof. Dr. Wilson Henrique Veneziano

Coorientador
Prof. Dr. Carlos Alberto Gonçalves

Brasília
2012

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Computação — Licenciatura

Coordenador: Prof. Dr. Flávio Barros Vidal

Banca examinadora composta por:

Prof. Dr. Wilson Henrique Veneziano (Orientador) — CIC/IE/UnB
Prof. Dr. Carlos Alberto Gonçalves (Coorientador) — CFS/IB/UnB
Prof. Dr. Joaquim Pereira Brasil Neto — CFS/IB/UnB
Prof. Dr.ª Aletéia Patrícia Favacho de Araújo — CIC/IE/UnB
Prof. Dr.ª Maristela Terto de Holanda — CIC/IE/UnB

CIP — Catalogação Internacional na Publicação

Neves, Bruno Bastos.

SimNeural: Simulador Computacional de RNA para Aplicações em Ensino e Pesquisas Médicas e Biológicas / Bruno Bastos Neves, Jussiê Marques Martins. Brasília : UnB, 2012.
193 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2012.

1. Simulação Computacional, 2. Redes Neuronais Artificiais,
3. Classificação Biológica, 4. Simulador, 5. Inteligência Artificial,
6. Algoritmo *BackPropagation*, 7. Ambiente de Simulação,
8. Reconhecimento de Padrões, 9. Aprendizagem por RNAs.

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

SimNeural: Simulador Computacional de RNA para Aplicações em Ensino e Pesquisas Médicas e Biológicas

Bruno Bastos Neves
Jussiê Marques Martins

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof. Dr. Wilson Henrique Veneziano (Orientador)
CIC/IE/UnB

Prof. Dr. Carlos Alberto Gonçalves (Coorientador)
CFS/IB/UnB

Prof. Dr. Joaquim Pereira Brasil Neto
CFS/IB/UnB

Prof. Dr.ª Aletéia Patrícia Favacho de Araújo
CIC/IE/UnB

Prof. Dr.ª Maristela Terto de Holanda
CIC/IE/UnB

Prof. Dr. Flávio Barros Vidal
Coordenador do Curso de Computação — Licenciatura

Brasília, 24 de Setembro de 2012

Dedicatória

Dedico este trabalho à minha família, que sempre esteve ao meu lado em todos os momentos maravilhosos e difíceis pelos quais passei e que ainda vou passar em minha vida, à minha noiva Kennia, com quem escolhi dividir minhas alegrias e angustias; mas dedico especialmente, para meus pais Eliane Bastos Neves e Getúlio Flores Neves, que me deram a oportunidade de estudar; algo que muitas pessoas, infelizmente, não possuem durante toda a vida.

Bruno Bastos Neves

Dedico este trabalho a Deus, aos meus pais Joaquim e Wilma, à minha namorada e à minha família, que sempre estiveram me apoiando e incentivando nessa caminhada.

"O temor ao Senhor é o princípio da sabedoria; bom entendimento têm todos os que cumprem os seus mandamentos". (Sl. 111:10)

Jussiê Marques Martins

Agradecimentos

Agradecemos aos professores Wilson Henrique Veneziano e Carlos Alberto Gonçalves que nos orientaram e nos auxiliaram durante todo o desenvolvimento deste projeto, nos transmitido valiosos ensinamentos.

Também gostaríamos de agradecer a todos do Departamento de Botânica que nos forneceram os dados durante o desenvolvimento do trabalho, à Fernanda Rodrigues da Costa, graduada em Letras, pelas correções ortográficas nos textos desta monografia e ao professor Custódio Motta da Universidade de Juiz de Fora, por nos indicar a base de dados que utilizamos para validação do trabalho.

A todos aqueles que apoiaram e contribuíram, ao longo dessa jornada, para concretização deste trabalho. Deixamos aqui a nossa gratidão.

Bruno Bastos Neves e Jussiê Marques Martins

Agradeço a Deus por todas as oportunidades que tive em minha vida. Agradeço aos meus pais, pois eles me fizeram o que sou hoje. Devo tudo a eles. Agradeço também ao meu colega e amigo Jussiê, que me acompanhou durante esta longa caminhada que durou quatro anos, desejo muita felicidade em sua vida. E por último, um agradecimento especial ao professor Wilson que nos orientou de forma exemplar em um semestre com muitas turbulências.

Bruno Bastos Neves

Agradeço a Deus, o mestre maior, que me deu a vida, à minha família, em especial aos meus pais Joaquim e Wilma, que sempre me ensinaram a vivê-la com dignidade, e à minha namorada, Fernanda, pelo carinho, companheirismo e compreensão, que sempre estiveram me apoiando e incentivando na realização dessa jornada. Agradeço também ao meu colega e amigo Bruno, por ter aceitado embarcar comigo nessa empreitada, por todo seu esforço, dedicação e comprometimento, desejo muita alegria e sucesso em sua vida sempre.

Jussiê Marques Martins

Resumo

A técnica de simulação computacional e de reconhecimento de padrões utilizando redes neuronais artificiais tem se tornado uma importante ferramenta para aprendizado e para o uso funcional. Este trabalho apresenta as atividades de concepção, desenvolvimento e validação do software SimNeural, concebido para uso amigável por profissionais das áreas de Ciências Biológicas e Médicas, em atividades de ensino e pesquisa. O software foi concebido por meio do ambiente de simulação ExtendSIM, da empresa Imagine That! Inc., utilizando o processo de desenvolvimento por prototipação e reuso de componentes. O SimNeural tem como principal objetivo ser uma ferramenta, que fornece o arcabouço necessário para a simulação de modelos configuráveis pelo usuário, para aprendizado e para uso funcional das redes neuronais artificiais, ao permitir testes de hipóteses e investigação de relações entre as variáveis.

O SimNeural implementa técnica de simulação computacional e de reconhecimento de padrões utilizando redes neuronais artificiais de múltiplas camadas treinadas com o algoritmo de retropropagação de erro (*backpropagation*). São apresentados os processos de validação obtidos com treinamento de RNAs e testes funcionais realizados com bases de dados reais, assim como os resultados da etapa de análise de amostras e dos testes experimentais de reconhecimento e classificação de padrões na aplicação dos modelos criados e treinados com o SimNeural. Os resultados do processo de validação com as base de dados reais mostraram que este software é uma ferramenta para ensino e pesquisas médicas e biológicas útil e eficaz.

Palavras-chave: Simulação Computacional, Redes Neuronais Artificiais, Classificação Biológica, Simulador, Inteligência Artificial, Algoritmo *BackPropagation*, Ambiente de Simulação, Reconhecimento de Padrões, Aprendizagem por RNAs.

Abstract

The computational simulation and the patterns recognition by artificial neuronal networks technique have become one of the most important tools to learning and functional uses. This work presents activities of conception, development and authentication of SimNeural software, designed with aapproachableinterface for professionals from different areas, such as Biological Sciences and Medicine, in several activities, as education and research. The software has been planned through the ExtendedSIM environment of simulation, from Imagine That! Inc. enterprise, using the prototyping and recycling of components process. The SimNeural main goal is to be a tool that serves as the essential framework to the simulation of models configurable by users, to learning and neuronal artificial network functional uses, allowing tests of hypothesis and investigations of the relations among variables.

The SimNeural implements techniques of computational simulation and recognition of patterns, using artificial neuronal networks of multiple layers trained withalgorithms of backpropagation of errors. Here are presented the validation procedures with RNAs training and functional trials performed based on real data, as well as the results of each stage of the analysis of the samples and experimental tests of recognition and classification of patterns in the application of the models prepared and created with the SimNeural system. The results of the whole validation process, with the real databases, showed that this software actually is an useful and effective tool for biological and medical researchand learning.

Keywords: Computational Simulation, Artificial Neuronal Networks, Biological Classification, Simulator, Artificial intelligence, Algorithm of BackPropagtion, Environment of Simulation, Recognition of Patterns, Learning by RNAs.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Apresentação do problema

Em diversas situações o uso de simulação computacional é uma alternativa viável e com obtenção de ótimos resultados, seja para verificar a viabilidade de um projeto, para evitar riscos existentes em atividades perigosas ou para pesquisas e aquisição de conhecimento. O custo com uso de técnicas de simulação é baixo quando comparado ao que poderia ser despendido com a criação de um laboratório completo para experimentação por alunos de um curso de biologia, por exemplo.

O grande problema é que a criação dessas simulações exige um grande conhecimento, não apenas do ambiente e do problema que se deseja simular, mas também de linguagens de programação, formando uma barreira de difícil transposição pelas outras disciplinas.

1.2 Hipóteses

Existem softwares que foram desenvolvidos para reduzir estas dificuldades por meio de interfaces com usuário, os quais permitem a criação de simulações com um conhecimento mínimo de linguagens de programação. Com tais ferramentas é possível criar ricas simulações em 2D ou até mesmo 3D, mesmo para pessoas leigas em linguagens de programação.

Neste contexto, com o uso dessas ferramentas, talvez a tarefa de criar e executar simulações computacionais possa ser simplificada para atingir este público alvo.

1.3 Motivação

Verificar a potencialidade de um dos ambientes de simulação disponíveis no mercado, o ExtendSim, no que se refere a fornecer ao usuário uma maneira de simplificar a criação e a execução de simulação. Para realizar tal tarefa, em conjunto com o Instituto de Ciências Biológicas da Universidade de Brasília (UnB), pretende-se desenvolver um modelo, o Projeto SimNeural, que consistirá em um simulador de Redes Neurais Artificiais que possa ser aplicado em aulas e atividades de pesquisa do instituto.

1.4 Objetivo

Desenvolver um ambiente de simulação de Redes Neuronais Artificiais, no pacote de simulação ExtendSim. O software deve facilitar a criação de simulação de RNA's por disciplinas acadêmicas diversas que não possuem grande afinidade com a codificação de sistemas.

1.5 Objetivos específicos

Os objetivos específicos são:

- apreender conceitos relativos a simulações;
- analisar o software ExtendSim;
- dominar completamente a linguagem ModL, usada no ambiente ExtendSim;
- compreender os fundamentos de Redes Neurais Artificiais e o algoritmo de aprendizagem *Backpropagation*;
- tornar o ambiente de simulação o mais amigável possível para usuários leigos em Redes Neurais;
- validar o ambiente com o cliente, o qual é o Instituto de Ciências Biológicas.

1.6 Resultados esperados

Ao final deste trabalho pretende-se construir um simulador de Redes Neurais Artificiais que utilize algumas das ferramentas disponíveis no pacote de simulação ExtendSim. Este simulador deverá ser de fácil operação, quer seja operado por pessoas da área de computação ou por usuários de outras áreas do conhecimento.

1.7 Metodologia

Visando os objetivos específicos deste trabalho, realizou-se uma pesquisa em busca de comparar ferramentas similares ao ExtendSim. Tal pesquisa constitui a revisão da literatura. Posteriormente, os seguintes tópicos foram definidos para constituição do embasamento teórico necessário para o projeto:

- conceitos em simulação computacional;
- ambiente de simulação ExtendSim;
- Redes Neurais Artificiais;
- engenharia de software.

Com esses conceitos apreendidos, foi construído um simulador no ExtendSim, para avaliar as possibilidades que o desenvolvimento no pacote de simulação ExtendSim pode oferecer. O software construído deverá, então, ser validado junto ao Instituto de Ciências Biológicas da UnB (IB) para comprovar a hipótese levantada.

1.8 Organização deste trabalho

Para o estudo proposto, este trabalho está organizado conforme segue nos próximos parágrafos.

O Capítulo 2 é a revisão da literatura que busca mostrar outras ferramentas desenvolvidas para simulação de Redes Neurais Artificiais, além de mostrar os pontos fortes e fracos das ferramentas desenvolvidas, vamos mostrar outros ambientes além do ExtendSim, que podem ser usados em trabalhos deste tipo.

No Capítulo 3 são abordados conceitos fundamentais da simulação computacional, tais como simulações de eventos contínuos e discretos.

O Capítulo 4 explora o ambiente da ferramenta ExtendSim e as potencialidades que ela oferece aos seus usuários. São abordados também os principais elementos necessários para construção de modelos de simulação.

No Capítulo 5, abordamos os conceitos fundamentais de Redes Neurais Artificiais, que é justamente o modelo que implementa-se na criação do ambiente que foi feito para avaliar o uso do ExtendSim.

No Capítulo 6 está descrito todo o processo por trás do desenvolvimento do Projeto SimNeural, bem como o modelo criado utilizando os conceitos de Redes Neurais Artificiais discutidos no Capítulo 5.

O Capítulo 7 descreve o projeto SimNeural em dtalhes. São descritas as principais características emergentes do ambiente de simulação desenvolvido neste trabalho.

O Capítulo 8 mostra o processo e os resultados de validação do SimNeural.

Finalmente, o Capítulo 9 contém todas as conclusões a respeito do uso do SimNeural e do ExtendSim para fins didáticos e de pesquisa, além de uma avaliação deste trabalho, quanto aos objetivos propostos.

Capítulo 2

Revisão da literatura

O desenvolvimento de ferramentas computacionais de simulação para modelos biológicos, em especial, usando redes neuronais artificiais, tem sido abordado sob perspectivas distintas, em trabalhos nacionais e internacionais. As ferramentas diferem quanto a metodologia, técnica e complexidade permitida para a simulação de modelos. Para cada uma é possível elencar vantagens e desvantagens, dependendo do objetivo da pesquisa.

Este capítulo fornece uma revisão de diferentes tipos de ferramentas, destacando abordagens distintas. As ferramentas escolhidas para a revisão foram definidas conforme os trabalhos mais citados na literatura, além de algumas ferramentas recentes que apontam novas tendências e abordagens. Para todas as ferramentas investigadas, foram analisados trabalhos de modelos que as aplicam, de modo que fossem comparadas apenas aquelas que já foram testadas e consideradas eficazes dentro do seu objetivo.

2.1 Ferramentas desenvolvidas para simulações de Redes Neuronais Artificiais

Existem muitos pacotes de aplicativos computacionais para simulação disponíveis no mercado. Nesta seção serão abordadas, algumas das ferramentas de simulação cujas características estão relacionadas aos conceitos abordados no conjunto de palavras chave deste projeto.

2.1.1 SNNS (*Stuttgart Neural Network Simulator*)

Dentre as opções de simuladores de redes neuronais artificiais encontradas na literatura, o projeto SNNS (*Stuttgart Neural Network Simulator*), que foi desenvolvido no *Institute for Parallel and Distributed High Performance Systems* (IPVR) da Universidade de Stuttgart, na Alemanha, apresenta maior compatibilidade e semelhança com os objetivos propostos pelo SimNeural. O objetivo do SNNS foi criar um ambiente de simulação eficiente e flexível para a pesquisa de aplicações de rede neuronal [33]. O SNNS apresenta uma interface de utilização bastante amigável, incluindo uma série de funcionalidades que possibilitam a solução de uma grande quantidade de problemas por meio de reconhecimento de padrões. Quatro componentes formam a base do SNNS: o núcleo do simulador,

a interface gráfica com o usuário (veja a Figura 2.1), a interface de execução em lote e o compilador de rede snns2c.

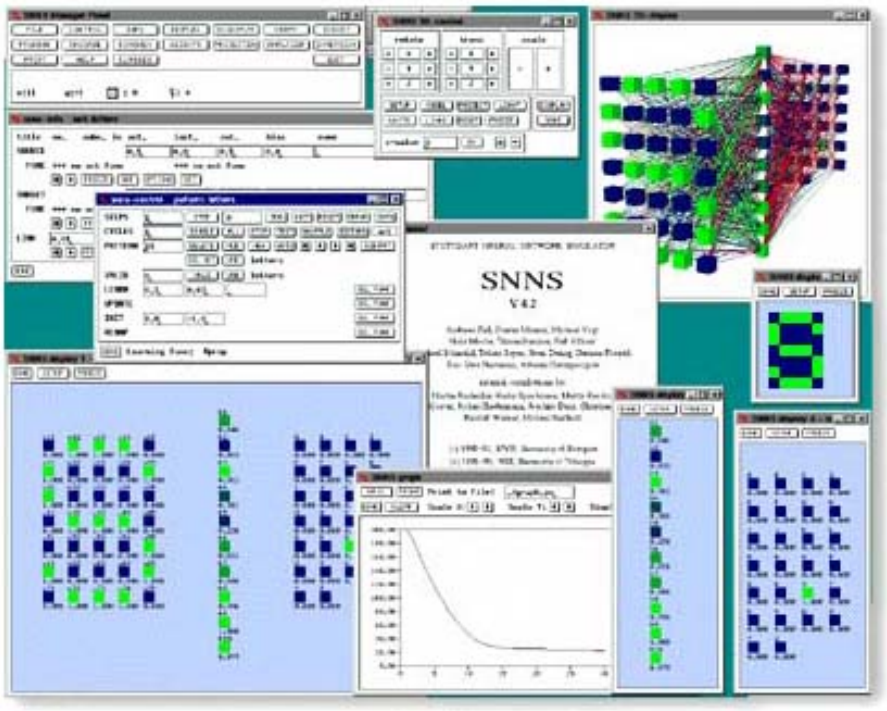


Figura 2.1: Interface gráfica do SNNS [32].

O enfoque do projeto SNNS é ser robusto em termos de consistência, e as funções de aprendizado disponíveis no SNNS são numerosas e variadas, podendo atender um número significativo de alternativas topológicas de redes neurais. Porém, sua interface não oferece uma apresentação didática e de fácil compreensão para usuários iniciantes nos conceitos de RNA e simuladores computacionais.

2.1.2 JavaNNS (*Java Neural Network Simulator*)

O software SNNS apresenta uma versão em Java, o JavaNNS v4.2, que é o sucessor do SNNS. O *Java Neural Network Simulator* (JavaNNS) é um simulador para redes neurais artificiais, desenvolvido pela *Wilhelm-Schickard-Institute for Computer Science* (WSI), na Universidade de Tübingen, Alemanha. O núcleo do SNNS foi mantido nesta versão, sendo que sua principal vantagem, é ter uma interface gráfica mais agradável e amigável escrita em Java [6]. Na Figura 2.2, é possível ver como é a nova interface, que mantém o poder de simulação do antecessor SNNS.

O JavaNNS permite o uso de redes predefinidas, ou seja, com arquitetura já pronta para realização de treinamentos e análises. Esse simulador difere do SIMNeural por trabalhar com a criação da RNA por camadas e usar quadrados para representar cada neurônio. Depois de criar todas as camadas e neurônios o usuário deve ligá-las. No SIMNeural o usuário define a arquitetura, e o software gera a RNA, automaticamente. Para criar ou usar uma rede no JavaNNS é necessário que o usuário, ao contrário do SIMNeural, tenha

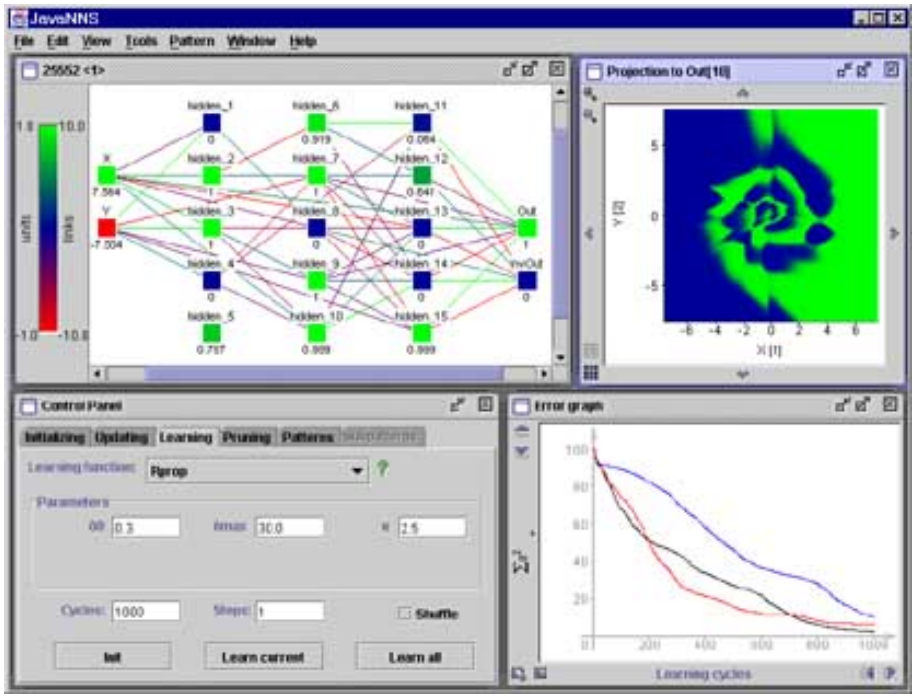


Figura 2.2: Interface gráfica do JavaNNS [31].

conhecimento do funcionamento dos algoritmos de aprendizagens que desejar implementar na RNA.

2.1.3 MATLAB (*Matrix Laboratory*)

Outro sistema que também oferece suporte para simulação de redes neurais artificiais é o MATLAB, por meio do seu pacote *Neural Network Toolbox*.

O MATLAB foi desenvolvido no início da década de 80 por Cleve Moler, no Departamento de Ciência da Computação da Universidade do Novo México, EUA. Tem por objetivo ser um ambiente de alto nível com linguagem interativa que permite ao usuário executar tarefas de computação intensiva mais rápido do que com linguagens de programação tradicionais como C, C++ e Fortran [21].

Ele é um "software" interativo de alta performance voltado para o cálculo numérico. Integra análise numérica, cálculo com matrizes, processamento de sinais e construção de gráficos em ambiente fácil de usar, onde problemas e soluções são expressos somente como eles são escritos matematicamente, ao contrário da programação tradicional [21].

O *Neural Network Toolbox* inclui e fornece linha de comando, funções e ferramentas gráficas para a criação, treinamento, visualização e simulação de redes neurais. As ferramentas gráficas facilitam o desenvolvimento de redes neurais para tarefas, tais como dados de encaixe (incluindo dados de séries temporais), reconhecimento de padrões, e *clustering*.

As redes neurais criadas com a *Neural Network Toolbox* (Figura 2.3) são usadas para aplicações nas quais a análise formal seria difícil ou impossível, tais como reconhecimento de padrões e de identificação do sistema não-linear. Depois de criar as redes nesta ferramenta, pode-se gerar automaticamente código MATLAB para capturar o trabalho e automatizar tarefas.

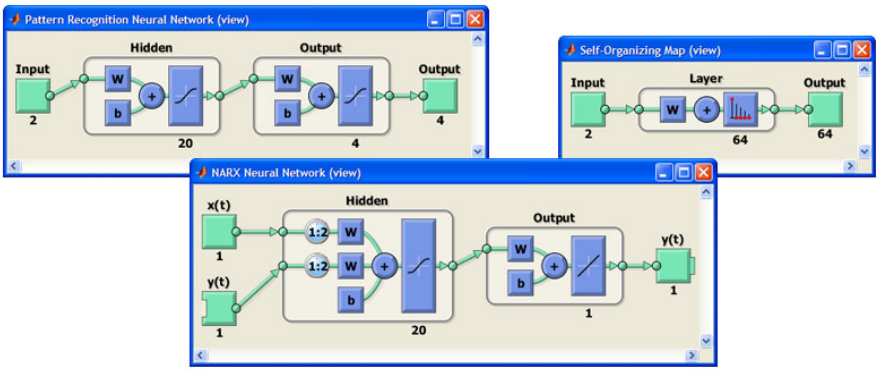


Figura 2.3: Exemplo de rede neuronal criada no MATLAB [21].

2.1.4 Software ARENA

O Arena é um software de simulação de processos, voltado para estudo de sistemas utilizando modelos matemáticos. Ele fornece uma interface gráfica que permite a elaboração de modelos de simulação baseado na linguagem SIMAN. Essa linguagem basicamente enxerga o sistema como uma sequência de eventos aleatórios que causam mudanças no estado do modelo (semelhante às redes de Petri ou aos autômatos) [34].

No Arena os modelos são construídos usando blocos de modelagem, conforme demonstrado na Figura 2.4. Existem dois grupos de blocos, os quais são o grupos de fluxo que são os blocos interconectados e que formam uma rede de informações e comandos por onde as entidades seguirão, e o grupo de dados, no qual os blocos ficam ocultos no modelo e têm a função de inserir a especificação de cada elemento do fluxo.

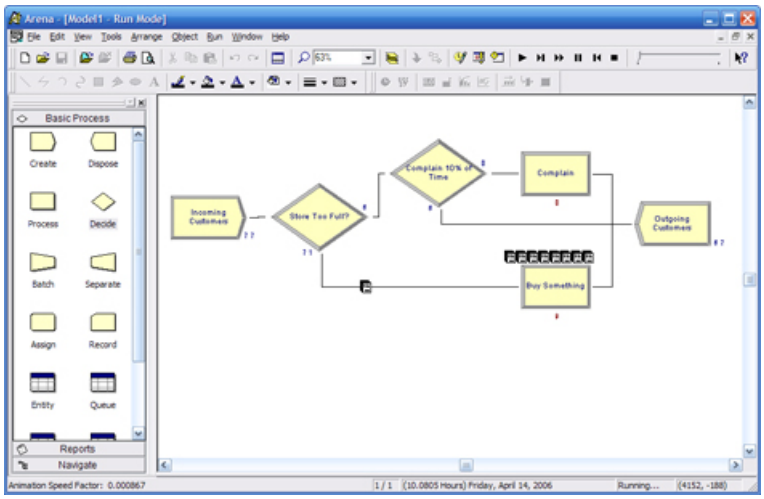


Figura 2.4: Visão gráfica do software Arena 9.0, com um modelo de sistema.

Na pesquisa realizada, o Arena mostrou ser muito usado para modelagem na área de engenharia de redes, mas não foram encontrados projetos utilizando esta ferramenta para desenvolver simulação de redes neurais artificiais.

2.1.5 Pacote de simulação STELLA©

O pacote de simulação STELLA© é baseado na metodologia das dinâmicas de sistemas. O software STELLA© foi concebido com o objetivo de simular um sistema ao longo do tempo, fazer modelagem da teoria para o mundo real, mapear as entradas de um sistema e demonstrar os resultados [15].

A interface baseada em ícones facilitando a construção de modelos é uma das principais características deste simulador (veja a Figura 2.5). Além disso, possui diagramas de estoque e fluxo em sua linguagem que apoia o pensamento sistêmico, e fornece informações sobre como os sistemas funcionam e as estruturas de suas funções, equações e *arrays* facilitam as operações matemáticas, estatísticas e lógicas, além de permitir visualizações da estrutura do modelo [15].

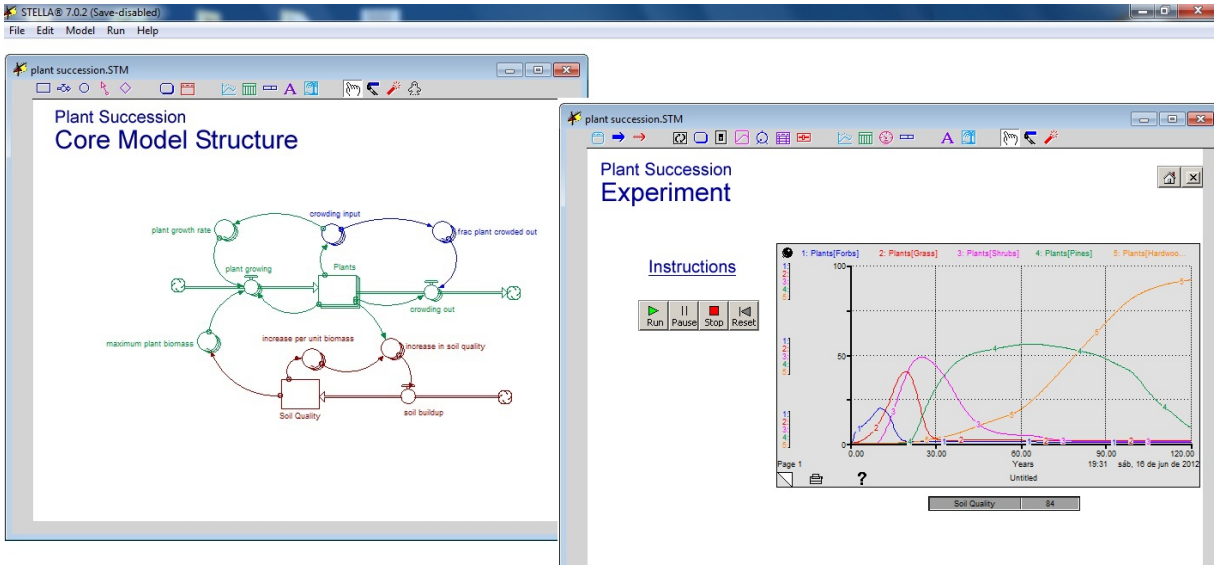


Figura 2.5: Visão gráfica do software STELLA© 7.0, com o modelo do sistema *Plaint Succession*.

O uso principal do STELLA© é o educacional como estimulador na aprendizagem por meio da modelagem e simulação de sistemas. Sendo adotado por educadores e pesquisadores para estudar, desde a economia à física, literatura de cálculo, química, entre outros. Na pesquisa realizada não foi encontrado projeto que usasse o STELLA© como ambiente de simulação para redes neuronais artificiais

Capítulo 3

Conceitos básicos em simulação computacional

Este capítulo aborda conceitos básicos de simulação de modelos e sistemas. Além disso, elenca vantagens e desvantagens no uso da simulação como forma de analise de problemas, cita as sete etapas em um estudo de simulação.

3.1 O histórico da simulação

A simulação de sistemas é tão antiga quanto a humanidade. Existem evidências, nas pinturas rupestres, que os homens das cavernas desenhavam as caçadas em paredes planejando resultados de suas ações. Durante muito tempo a predição esteve limitada a lógica e métodos dedutivos dos filósofos gregos. *Francis Bacon* foi considerado o criador do método científico ao reconhecer que a razão, por si só, não tem nenhuma capacidade de predição, é necessário utilizá-la em conjunto com meticulosas técnicas de observação [22].

O uso moderno do vocábulo simulação remonta ao Método de Monte Carlo desenvolvido por volta de 1940. Esse método proporcionou a solução de problemas matemáticos não probabilísticos através da simulação de um processo estocástico, sendo aplicado em diversos problemas matemáticos [22].

Com o aumento da capacidade computacional, a simulação cresceu como técnica utilizada para experimentação e estudos pelas mais diversas áreas de conhecimento. Com este crescimento surgiram também novas alternativas para o campo de simulação computacional.

As linguagens de formato geral foram as primeiras a serem utilizadas para a simulação. Essas linguagens, como C, FORTRAN e PASCAL, podem ser utilizadas na implementação de qualquer problema computacional. Inicialmente, as simulações utilizavam estas linguagens e eram restritas aos trabalhos em universidades e centros de pesquisas militares. Cada simulação tinha um propósito específico e execuções muito problemáticas devido ao baixo poder computacional no início da revolução da informação.

Contudo, observou-se que os programas, apesar de objetivos diferentes, possuíam muitas características comuns, dando origem ao desenvolvimento de linguagens de simulação, mais poderosas e mais eficientes. Estas linguagens têm a vantagem de reduzir o tempo necessário para implementação de um modelo, pois apresentam estruturas voltadas para

este fim, ao contrário das linguagens de propósito geral. Dentre as linguagens de simulação destacaram-se o GPSS, SIMAN, SLAM e o SIMSCRIPT [22].

O salto dado foi grande, mas o tempo gasto com aprendizado e eliminação de erros ainda era enorme. Durante os anos de 1980, juntamente com um exponencial aumento na capacidade computacional, a simulação estendeu-se a muitas indústrias e com isso programas direcionados a elas foram criados (por exemplo: Extend, Arena, etc.). Estes programas são chamados pacotes de simulação e eram passíveis de uso por profissionais de diversas áreas e com diferentes níveis de conhecimento sobre simulação.

Os pacotes de simulação reduziram o tempo gasto na criação de modelos, liberando para concentração de esforços nas análises de resultados. O pacote usado no Projeto SimNeural foi o ExtendSim, software sucessor do pacote de simulação Extend.

Tabela 3.1: Evolução da simulação e das ferramentas ao longo do tempo [16].

Período	**Ferramenta**	**Características do estudo de simulação**	**Exemplos**
1950 a 1960	Linguagens de propósito geral	Aplicações em grandes corporações; Grupos de desenvolvimento de modelos com 6 a 12 pessoas; Geram programas a serem executados em grandes computadores; Grandes investimentos em capital; Aplicáveis a qualquer contexto; Exigem conhecimento profundo da linguagem; Exigem muito tempo de desenvolvimento; Não são totalmente reutilizáveis.	FORTRAN, PASCAL e C
1970 a início de 1980	Linguagens de simulação	Utilização em um maior número de corporações; desenvolvimento e uso dos pacotes de linguagens; surgem linguagens de simulação baseadas em *System Dynamics*; comandos projetados para tratar lógica de filas e demais comandos comuns; Mais amigáveis, mas ainda requerem programador especializado.	SIMSCRIPT, GPSS, GASP, IV, DYNAMO, SIMAN, SLAM
1980 a início de 1990	Simuladores de Alto nível	Introdução do PC e da animação; Simulação realizada antes do início da produção; Facilidade de uso; Menos flexível que as linguagens de propósito geral e de simulação; Projetados para permitir modelagem rápida; Dispõem de elementos específicos para representar filas, transportadores, etc.; Restringem-se à sistemas de certos tipos.	Simfactory e XCell
1990 em diante	Pacotes de Simulação	Melhor animação e facilidade de uso; Fácil integração com outras linguagens de programação; Usada em fase de projeto; Grande uso em serviços; Uso para controle de sistemas reais; Grande integração com outros pacotes (bases de dados e processadores de texto); aprimoramento dos simuladores, permitindo rápida modelagem; Integram a flexibilidade das linguagens de simulação com a facilidade de uso dos pacotes de simulação.	ARENA, Extend, Stella, ProModel

3.2 Sistemas

A natureza pode ser vista como um conjunto de sistemas que interagem constantemente. Até mesmo nosso corpo é um sistema composto de diversos subsistemas. As simulações são criadas sobre modelos de sistemas devidamente delimitados [26]. Algumas definições a respeito de sistemas segundo alguns autores de diversas áreas:

- Sistema é um conjunto de coisas ou combinações de coisas ou partes, formando um todo complexo ou unitário ;
- Um sistema pode ser definido como um grupo de objetos que são reunidos em alguma interação ou interdependência regular, a fim de alcançar algum propósito [2].

Dessas definições é possível notar que um sistema é algo abstrato. Em um dado instante, um conjunto de objetos pode ser uma pequena parte de um sistema maior, ou seja, um subsistema, em outras, pode ser um sistema completo. Portanto, todo sistema deve ter estabelecido seus limites, para que possa ser estudado. Determinar estes limites depende, fundamentalmente, do estudo realizado e dos objetivos a serem alcançados.

3.3 Modelos

Modelos são abstrações de fenômenos, objetos ou entidades complexas, e seu valor reside no fato de melhorar nossa compreensão a respeito das características de comportamento de um sistema real. Em comparação com a observação do sistema real, um modelo deste sistema trabalha com um menor custo para obtenção das informações, gerando conhecimento mais rapidamente e em condições, algumas vezes, não observáveis em condições naturais [8].

3.3.1 Tipos de modelos

Primeiramente, existe uma distinção que deve ser feita entre um modelo matemático e um modelo físico. Modelos físicos são, geralmente, réplicas em tamanho reduzido daquilo que eles representam, como maquetes, mapas, fotografias, etc. Os modelos matemáticos, usam símbolos em lugar de objetos físicos; eles representam um sistema em termos de relacionamentos lógicos e quantitativos, os quais, por meio de de manipulação, alteram o comportamento do modelo tentando demonstrar como o sistema reagiria. Um modelo de simulação é um tipo de modelo matemático. São apresentados a seguir alguns exemplos.

Linear ou não-linear

Em modelos lineares, o sistema representado segue uma lei linear, isto é, sempre há uma mesma resposta de uma variável endógena (interna do sistema) a uma variável exógena (externa ao sistema). Nos modelos não lineares esta relação não ocorre de forma linear.

Estático ou dinâmico

Em modelos estáticos o tempo não desempenha nenhum papel importante, enquanto que nos modelos dinâmicos o tempo apresenta papel fundamental na evolução do modelo.

Estável ou instável

Em modelos, esta característica aplica-se apenas aos modelos dinâmicos, que têm seu comportamento variável no tempo. Diz-se que o modelo é estável quando este retorna a sua condição inicial após alguma perturbação. Já os modelos dinâmicos instáveis são aqueles em que, depois de determinada ativação, há uma tendência ao desenvolvimento de sua amplitude afastando-se da condição inicial.

Discretos ou contínuos

As alterações nas variáveis de estado do sistema podem ser contínuas ou discretas conforme a evolução do tempo. Na alteração discreta, as variáveis modificam-se discretamente em pontos específicos no tempo. Já na alteração contínua, as variáveis podem ser alteradas continuamente ao longo do tempo simulado.

Determinísticos ou Estocásticos

Modelos que não contêm variáveis aleatórias são classificados como determinísticos. Nestes, um conjunto conhecido de entradas pode ser mapeado em um conjunto definido de saídas. Em um modelo estocástico este mapeamento não é possível, pois existe a presença de variáveis aleatórias que levam a saídas imprevisíveis.

Analíticos ou de simulação

Os modelos analíticos podem ser definidos pela sua estrutura formada por equações matemáticas, por meio das quais o comportamento do sistema pode ser obtido pela atribuição de valores aos parâmetros do modelo e solução das equações. Modelos de simulação possuem uma estrutura lógica/matemática e podem ser exercitados de forma a imitar o comportamento do sistema. Através de experiências, observações podem ser realizadas para que conclusões sejam tiradas a respeito do sistemas.

3.3.2 Escolha das variáveis do sistema

A construção de um modelo envolve a correta escolha das principais variáveis de um sistema, pois apenas um pequeno número de variáveis pode, geralmente, explicar grande parte do comportamento de um sistema. Disso decorre o fato de que podem coexistir diversos modelos, com diversos níveis de detalhamento de um mesmo sistema [1].

3.4 Tipos de simulação

Foram comentados os tipos de modelos existentes. Assim, como tipos de modelos, os tipos de simulação são análogos a estes modelos. Entretanto, no que diz respeito a modelos discretos e contínuos, é preciso esclarecer que a utilização de um modelo discreto não implica que será usada uma simulação discreta e o mesmo vale para modelos contínuos.

3.5 A simulação de sistemas

Em essência, a simulação consiste em realizar um modelo matemático de situação real, e nele levar a cabo experiências. É uma definição abrangente e poderia incluir, por exemplo jogos militares ou de negócios, que não muito raramente, também são chamados de simuladores [22].

Contudo, outros autores da área restringem mais essa definição, acrescentando que na simulação o modelo deve ser manipulado e os resultados devem ser analisados para que sejam tiradas conclusões sobre como os mais diversos fatores afetarão o desempenho do sistema simulado.

O grande volume de dados e a complexidade de cálculos de uma simulação exigem o uso intensivo do computador. O uso desta ferramenta na criação e solução dos modelos reduziu drasticamente o tempo despendido nos trabalhos e, por este motivo, as técnicas de simulação estão cada vez mais presentes em pesquisa e desenvolvimento [27].

Existem três perguntas que devem ser respondidas antes de iniciar o uso de simulação para solucionar um determinado problema [22]. As perguntas são:

1. é o processo de mais baixo custo para solução do problema?
2. há segurança de se obter uma solução satisfatória?
3. a técnica a ser usada permitirá uma interpretação relativamente fácil por parte do usuário?

A primeira pergunta é muito levantada no meio empresarial ou de mercado. Já as duas últimas são completamente aplicáveis a projetos deste tipo, pois ao utilizar a simulação de forma didática, precisa-se que ela permita uma interpretação fácil pelos usuários e a solução deve ser segura o suficiente para satisfazer os objetivos propostos.

Diversos autores citam algumas atividades onde a simulação em computador digital poderia ser utilizada [20] [22] [27]:

- experimentação e avaliação, com vista a prever resultados de mudanças sem implementá-las de fato em um sistema real, evitando gastos excessivos e riscos;
- forma de estudar novos sistemas, buscando o refinamento destes;
- compreensão de sistemas reais legados;
- familiarizar equipes com equipamentos ou sistemas;
- ensino, como material pedagógico para estudantes ou profissionais;
- aquisição de conhecimento por meio das etapas de uma simulação, principalmente na formulação do problema, na construção do modelo e na análise dos resultados;
- projeções de futuro, como ,por exemplo, mercado de ações;
- situações de risco para saúde humana, como ambientes radioativos.

Um modelo de simulação tem as seguintes propriedades [3]:

- intenção de representar a totalidade ou parte de um sistema;

- possibilidade de ser executado e manipulado;
- o tempo ou contador de repetições é uma de suas variáveis;
- proposta de auxiliar no entendimento do sistema.

Quanto ao auxílio referido no item 4, pode significar uma descrição de uma parte do sistema, explicação do comportamento do sistema, predição do comportamento do sistema ou ensino da teoria por trás do sistema modelado.

3.5.1 Características do modelo de simulação usado no projeto SimNeural

O Projeto SimNeural, que consiste em um simulador de Redes Neurais Artificiais, foi desenvolvido com base em um modelo de simulação não-linear, dinâmico, instável, discreto e estocástico. Estas características são completamente atendidas pelo pacote de simulações ExtendSim.

3.6 Vantagens e desvantagens da simulação

Segundo diversos autores da área, os benefícios mais citados das técnicas de simulação são [3] [20] [27]:

- modelos mais realistas;
- processo de modelagem evolutivo: os modelos iniciam-se simples e crescem em complexidade com o tempo à medida que são compreendidas as peculiaridades dos sistemas;
- "e se?" (*"What if?"*): a eterna pergunta que nos fazemos pode ser realizada livremente com simulação de modelos;
- aplicação em problemas mal estruturados: muitos problemas reais ainda não foram totalmente estruturados. A simulação é uma ferramenta para estudos destes tipos de problemas;
- facilidade de comunicação: simulações podem ser usadas no ensinamento de modelos matemáticos ou de sistemas complexos, por terem uma linguagem de melhor compreensão;
- soluções rápidas;
- grande flexibilidade.

Todavia, há também algumas desvantagens:

- tempo e experiência necessários para construção de modelos;
- em alguns casos, a simulação apresenta resultados de difícil interpretação;
- uso inadequado em situações mais simples, quando uma solução analítica é possível;

- tempo grande de processamento quando os recursos computacionais são muito limitados.

As desvantagens da simulação estão sendo combatidas com avanços tanto no poder computacional quanto no desenvolvimento de pacotes de simulações mais rápidos, amigáveis e robustos. O uso de animações 3D nos pacotes é uma clara pretensão de facilitar a visualização de alguns resultados de difícil interpretação.

3.7 Etapas em um estudo com simulação

A programação do modelo representa somente 25 a 50 por cento do trabalho em um estudo de simulação [19]. Esse autor ainda cita que muitas pessoas que executam estudos com simulação não possuem um treinamento formal em simulação além do uso de produtos de simulação em particular.

Dessa forma, conduzir um estudo utilizando simulação requer que o analista tenha, no mínimo, conhecimentos em validação de modelos, distribuição das probabilidades de entrada, *design* e análise de experimentos de simulação, teoria da probabilidade, etc. Além destes conhecimentos, é preciso conhecer profundamente o sistema que será simulado.

Law [19] propôs sete passos para condução com sucesso de um estudo com simulação. Eles podem ser visualizados em forma de um esquema na Figura 3.1.

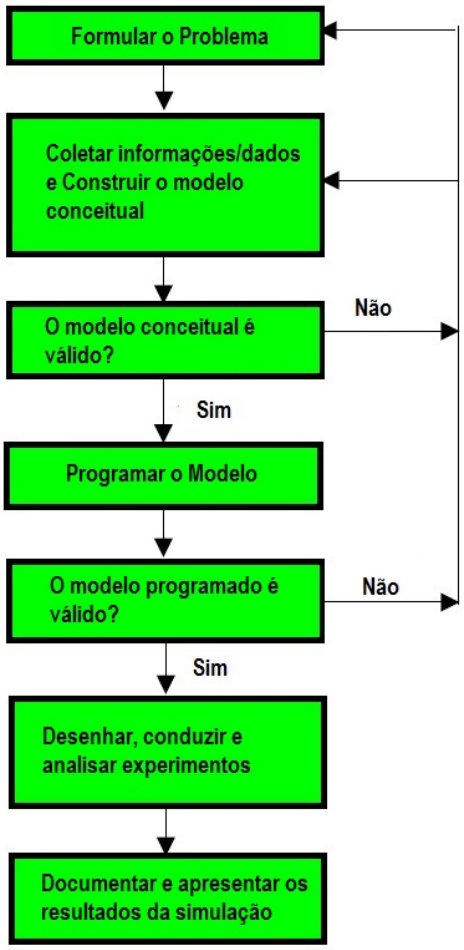


Figura 3.1: Os sete passos de um estudo utilizando simulação [19].

Passo 1. Formular o problema

O problema de interesse do estudo deve ser formulado pela pessoa que toma decisões no projeto. Nem sempre o problema está completamente definido e, muitas vezes, com o avanço dos estudos pontos obscuros são esclarecidos e devem ser comunicados ao tomador de decisões que pode, com eles, reformular o problema. Os seguintes aspectos devem ser definidos:

- objetivos gerais do estudo;
- objetivos (questões) específicos a serem respondidos pelo estudo, para determinar o correto nível de detalhamento do modelo que será desenvolvido;
- medidas de desempenho que serão utilizadas para avaliar a eficácia das diferentes configurações do sistema;
- escopo do modelo, também ligado ao nível de detalhamento necessário;
- as configurações do sistema a serem modeladas;
- janela de tempo e recursos necessários para o estudo. Muitas vezes ocorre subestimação do tempo e recursos necessários para o estudo.

Passo 2. Coleta de informações/dados e construção do modelo conceitual

Envolve a coleta de informações sobre a estrutura do sistema e dos procedimentos executados por ele. As informações coletadas, como algoritmos, procedimentos e parâmetros devem ser documentadas, coisa que muitas vezes é esquecida. O nível de detalhamento do modelo vai depender, além dos aspectos citados anteriormente, da disponibilidade de dados do sistema, credibilidade que se deseja alcançar.

Não existe uma correspondência exata (*one-to-one*). O ideal é começar com um modelo simples e incrementá-lo progressivamente, pois o excesso de complexidade pode causar um aumento desnecessário no tempo de simulação do modelo ou pior, obscurecer fatores do sistema que realmente são importantes.

Passo 3. Validação do modelo conceitual

Law [19] propõe um percorrimento estruturado do modelo conceitual. Ele defende que isso ajuda a garantir que as suposições do modelo estão corretas e completas e promove um domínio sobre o problema. Este percorrimento deve ser executado antes da programação para evitar re-trabalho.

Passo 4. Programar o modelo

Programar o modelo conceitual em alguma linguagem de propósito geral (C, C++, FORTRAN) ou em um pacote de simulação (ARENA, Extend, STELLA). A vantagem do uso de linguagens de propósito geral é a familiaridade que pode já existir com a linguagem, garantindo um melhor controle do programa, e menor custo com a produção do software. Por outro lado, o uso de pacotes de simulação reduz o tempo de programação e até mesmo o custo em projetos grandes.

Passo 5. Validação do programa

A validação do programa consiste em verificar se o programa representa corretamente o modelo e se o mesmo funciona corretamente. É importante a realização de testes e *debugging* nesta etapa. Algumas técnicas de validação são:

- Validação de resultados: se existir um sistema, compare as medidas de performance com o modelo de simulação programado. É a mais importante técnica de validação, pois se os resultados forem positivos, fatalmente aumenta a credibilidade do programa;
- Revisão de resultados: mesmo que não exista um sistema real para comparação, analistas devem revisar os resultados para procurar inconsistências;
- Análise de sensibilidade: deve ser executada para verificar, no programa do modelo, quais fatores/variáveis têm um maior impacto no comportamento e na performance, visando descobrir quais devem ser mais cuidadosamente modeladas.

Passo 6. Desenhar, conduzir e analisar experimentos de simulação

Para cada configuração do sistema, decidir os principais aspectos como o tempo de execução, número de replicações independentes, etc. Planejar com uma espécie de *script* de experiência como serão realizados os testes e obtidos os resultados.

Passo 7. Documentar e apresentar os resultados da simulação

A documentação deve conter o modelo conceitual, uma descrição detalhada do programa computacional, bem como resultados e conclusões dos estudos. Uma apresentação final deve ser realizada, mostrando animações ou o programa sendo executado, além de discussão sobre a construção e validação do modelo de forma a promover sua credibilidade.

Os passos acima são um guia para estudo por meio de simulação, mas nada impede que outras técnicas de gestão de projetos, ou mesmo de outras técnicas de estudo sejam utilizadas nos experimentos com simulações.

Capítulo 4

Ambiente de simulação ExtendSim

Neste capítulo é explorado o ambiente de simulação ExtendSim, da empresa Imagine That! Inc., no intuito de demonstrar o potencial da ferramenta e as possibilidades que ele oferece aos usuários. Também serão mostrados os principais elementos necessários para a construção e/ou alteração de modelos na ferramenta, inclusive os modelos elaborados no Projeto SimNeural.

4.1 Principais características

Algumas características interessantes do ambiente [14]:

- construção de modelos através de blocos programáveis e de GUI;
- GUI customizável, os blocos podem ser alterados livremente, através de sistema semelhante ao VisualBasic da Microsoft;
- criação de níveis hierárquicos nos modelos, permitindo uma estruturação em camadas dos modelos;
- animações em 2D e 3D;
- permite alteração de parâmetros durante a execução das simulações;
- editor de equações;
- gráficos customizáveis para exibir resultados das simulações;
- integração com outras linguagens de programação, como C++, Visual C++, etc.

O ambiente pode ser usado tanto para simulação de modelos contínuos ou discretos. É possível criar bibliotecas com blocos de simulação customizáveis, permitindo um compartilhamento entre usuários do ambiente.

4.2 Versões do ExtendSim

O ExtendSim é uma ferramenta sob licenciamento *copyright*, que proíbe a execução de uma parte da obra ou ela no todo, por terceiros não autorizados. No *copyright*, o autor

não permite a modificação, alteração, distribuição e nem a criação de obra derivada, sem prévia autorização. Contudo, a empresa forneceu uma cópia para desenvolvimento de projetos acadêmicos, do qual deriva este trabalho.

A versão cedida para fins acadêmicos foi a ExtendSim 7 OR. Na Tabela 4.1 tem-se um comparativo entre as versões disponíveis [14]:

Tabela 4.1: Versões existentes do ExtendSim.

Continuous Processes (CP)	ExtendSim CP é o produto base da suite, a pedra fundamental. Ele possui uma série de funções núcleo que estão incluídas em todas as outras versões, além de construções especializadas na modelagem de processos contínuos. Nesta versão, apenas a animação 2D está disponível.
Operations Research (OR)	As ferramentas da versão OR permitem a pesquisa de desempenho operacional, através de um sistema de eventos discretos baseado em trocas de mensagens. Com esta versão é possível acompanhar e analizar o comportamento de entidades físicas ou lógicas quando os eventos levá-los a mudar de estado.
Advanced Technology (AT)	A versão AT apresenta funcionalidades voltadas para processamentos em lotes, que buscam simular taxas de fluxo para sistemas de grande volume ou de grande velocidade. É flexível suficiente para acomodar qualquer escala de processo industrial. Além disso, inclui todas as capacidades da versão OR.
ExtendSim Suite (Total Simulation Project Support)	A principal funcionalidade adicionada por esta versão é o uso da tecnologia de animação 3D. O ambiente 3D é independente, mas integrado com os modelos lógicos, sendo a animação em 3D, uma adição à modelos lógicos já validados. Além disso, inclui todas as funcionalidades das versões anteriores.

Já existe no mercado a versão 8 do ExtendSim, com aprimoramentos em relação à versão utilizada neste trabalho. Contudo, os autores tiveram acesso apenas a versão 7 disponível. A única coisa a observar é que os modelos criados na versão 7 podem ser recompilados, sem problemas, para a versão atual.

4.3 Construção e execução de modelos no ambiente ExtendSim OR

Nesta sessão serão mostradas as ferramentas que o ambiente de simulação em sua versão OR coloca nas mãos de seus usuários: interface GUI, controles do fluxo da simulação, principais bibliotecas, blocos etc.

4.3.1 Interface GUI

A interface GUI, que será denominada a partir desta seção de área de trabalho, é onde os usuários irão construir seus modelos através da inserção de blocos prontos que realizam determinadas ações durante as simulações. O usuário necessita apenas abrir

bibliotecas, escolher os blocos que deseja utilizar e arrastá-los para a área de trabalho. A comunicação entre os blocos é realizada, via de regra, por conexões de entrada e saída, que são conectadas durante a criação do modelo.

A Figura 4.1 demonstra a área de trabalho de um dos modelos de exemplo que estão incluídos no ExtendSim. Destaca-se em vermelho os conectores de entrada e saída, que podem ser interligados através de simples comandos de clique e arraste do *mouse*.

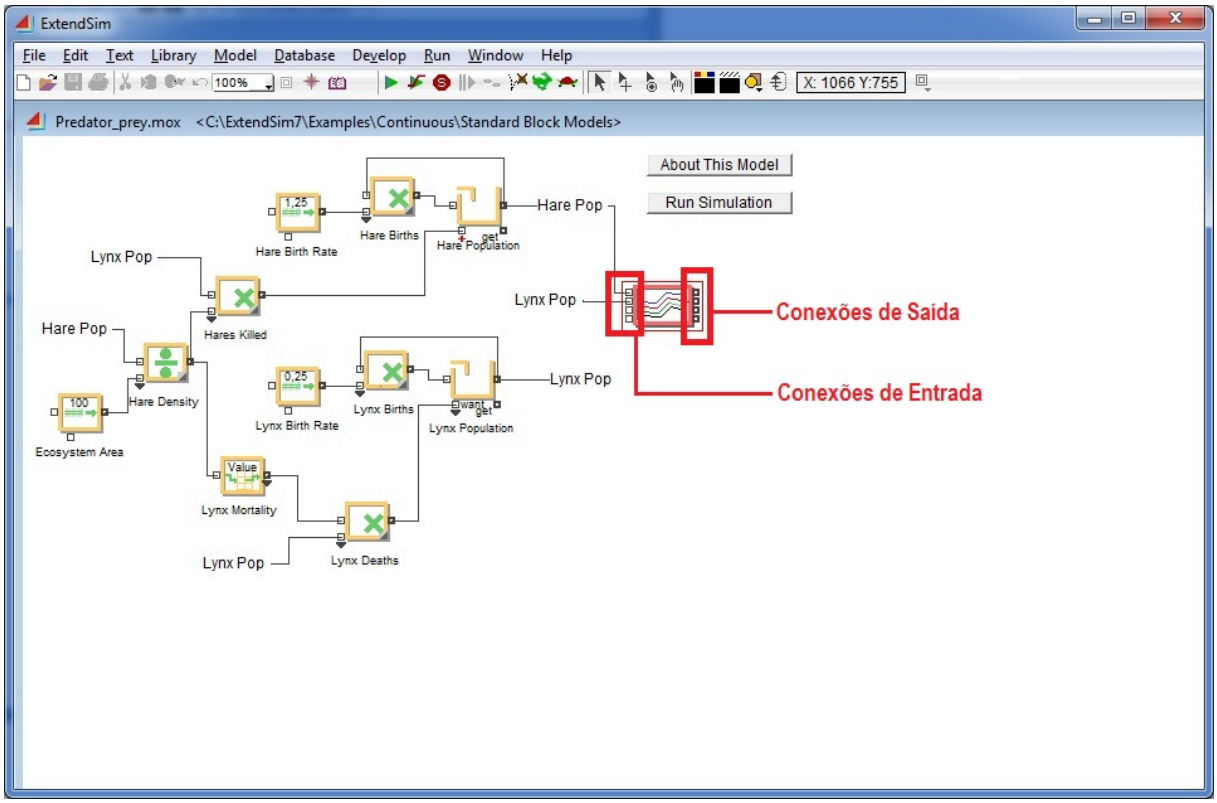


Figura 4.1: Área de trabalho do modelo presa-predador do ExtendSim.

Os ícones dos blocos e as ferramentas de descrição das informações contidas nos blocos possibilitam a criação de modelos de fácil entendimento por outras pessoas.

4.3.2 Bibliotecas e blocos

Já foi dito que os modelos são construídos por meio da inserção e conexão de blocos na área de trabalho do ExtendSim. Agora será mostrado o que de fato é um bloco e como podem ser acessados, operados e construídos.

Como exemplo será usado o bloco denominado *Math*, um dos blocos prontos disponibilizados na biblioteca *Value*. No ambiente ExtendSim, as bibliotecas são arquivos de extensão .lix que podem ser criadas, abertas ou fechadas através do menu *Library* disponível no ambiente ExtendSim. Elas possuem a função de armazenar de forma organizada os blocos.

Ao colocar um bloco na área de trabalho, um clique duplo sobre ele permite acessar um diálogo de configuração que, geralmente, atua sobre as operações realizadas pelo bloco.

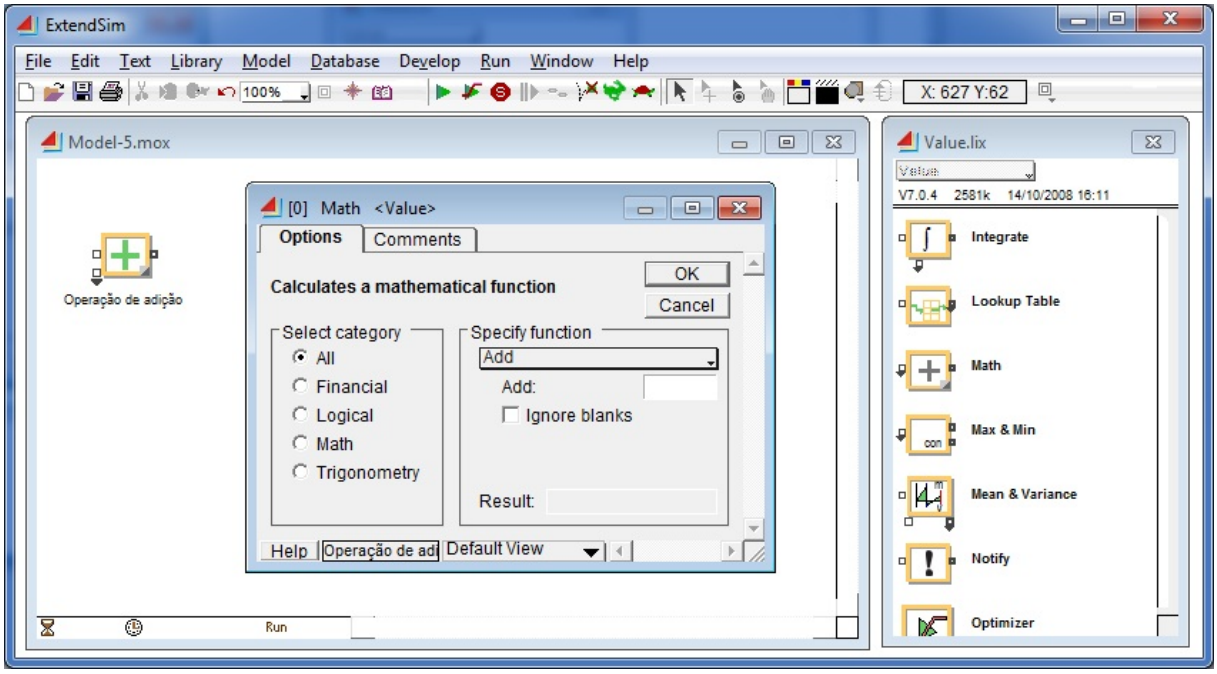


Figura 4.2: Janela mostrando o uso de um bloco no ExtendSim.

Na Figura 4.2 o diálogo do bloco *Math* está aberto, juntamento com a biblioteca Value.lix. Este bloco possui um série de funções matemáticas que podem ser usadas durante uma simulação. Por padrão a operação selecionada é a de adição (Add), uma alteração na operação, através da seleção da categoria e da operação em si, altera totalmente o comportamento do bloco, inclusive seu ícone.

Note-se que para realizar alterações das operações implementadas pelos blocos ou mesmo alterar a quantidade de operandos, o usuário não necessita de nenhum conhecimento em programação. Tudo é realizado por uma interface gráfica e esta é sem dúvida uma das grandes vantagens da utilização deste ambiente de simulação.

As bibliotecas e blocos podem ser compartilhadas por modelos diferentes. Um mesmo bloco de uma mesma biblioteca terá o mesmo comportamento em todos os projetos nos quais forem utilizados. Esta característica propicia um desenvolvimento de blocos reusáveis. Tal modelo se encaixa no desenvolvimento baseado em reuso de componentes [29].

Olhando em detalhes o relacionamento entre modelos, blocos, e bibliotecas, pode-se dizer que os blocos usados em um modelo são armazenados nas bibliotecas. Os blocos são formados pelos componentes descritos na Tabela 4.2. A definição completa de um bloco (seu código, ícone, itens GUI, etc) é armazenada em uma biblioteca.

Ao incluir um bloco em um modelo, ele não é de fato copiado para o modelo, mas apenas uma referência a determinada biblioteca e ao bloco são inseridas no modelo, de forma que ao abrir um modelo salvo ele buscará pelas referências. Isso é o que permite um fácil reuso e manutenção dos blocos, pois alterando uma definição de um bloco em uma biblioteca, todos os modelos que referenciam este bloco são automaticamente atualizados. Além disso, este método reduz o gasto com memória de computador e processamento dos modelos.

Tabela 4.2: Componentes de um bloco.

Ícone	Representação gráfica do bloco. Ela pode ser estática ou dinâmica, como é o caso do bloco *Math* que muda seu ícone de acordo com a operação selecionada.
Rótulo	Ao lado do botão *help* na caixa de diálogo do bloco, existe um campo que pode ser usado para rotular o bloco. O texto do rótulo é exibido junto ao bloco na área de trabalho.
Conectores	Elementos fundamentais dos blocos, pois é através deles que são estabelecidas as conexões entre blocos, bem como o fluxo de execução dos modelos. Os conectores podem ser de entrada ou saída, sendo que cada um deles possui um modo específico de uso.
Código	É transparente para o usuário, sendo a lógica interna do bloco. É escrita em uma linguagem semelhante ao C chamada ModL
Caixa de diálogo	Interface gráfica que permite aos usuários alterar o comportamento do bloco

4.3.3 Camadas hierárquicas

Em modelos complexos é possível a criação de diversas camadas que podem agrupar blocos para formar subsistemas dentro da simulação de um sistema complexo. Eles são chamados de blocos hierárquicos e têm como principal função organizar os modelos. Um bloco hierárquico pode ser criado vazio ou a partir de blocos já existentes no modelo. Sua aparência é semelhante a de um bloco comum, mas um duplo clique sobre ele não abre uma caixa de diálogo, mas uma área semelhante a área de trabalho do projeto, como se fosse sua extensão, contendo os blocos que estão dentro dele.

A ideia é literalmente colocar blocos dentro de outros blocos. Entretanto, os blocos criados para realizar esta organização não são armazenados nas bibliotecas, mas no próprio modelo.

Muitas vezes o usuário do ExtendSim pode deparar-se com modelos que necessitam de n camadas. Visando simplificar o acesso aos parâmetros dos blocos internos, existe uma ferramenta que permite ao usuário copiar elementos das caixas de diálogos de blocos internos para uma camada de controle criada por ele.

Assim como blocos comuns, ícones e animações 2D podem ser adicionados aos blocos hierárquicos, unindo o uso deles com ferramentas para desenho de formas e de escrita em modelos, é possível criar modelos complexos ao mesmo tempo em que não se abre mão de uma fácil compreensão por quem for executá-los.

4.3.4 Executando simulações

Com o modelo construído, o usuário deve configurar a execução da simulação. No caso do Projeto SimNeural, todos os parâmetros internos do modelo já estarão configurados, restando ao usuário alterar parâmetros do SimNeural.

Simulation Setup

Por meio do menu *Run* > *Simulation Setup* o usuário tem acesso ao painel para configuração dos parâmetros da simulação (veja a Figura 4.3), bem como das animações 3D, caso elas existam; É possível alterar quantas vezes a simulação será executada, o modo de interação com o ambiente 3D, etc. Estas configurações valem para cada execução de uma simulação, não sendo preciso realizar novas configurações para o mesmo modelo antes de uma nova execução.

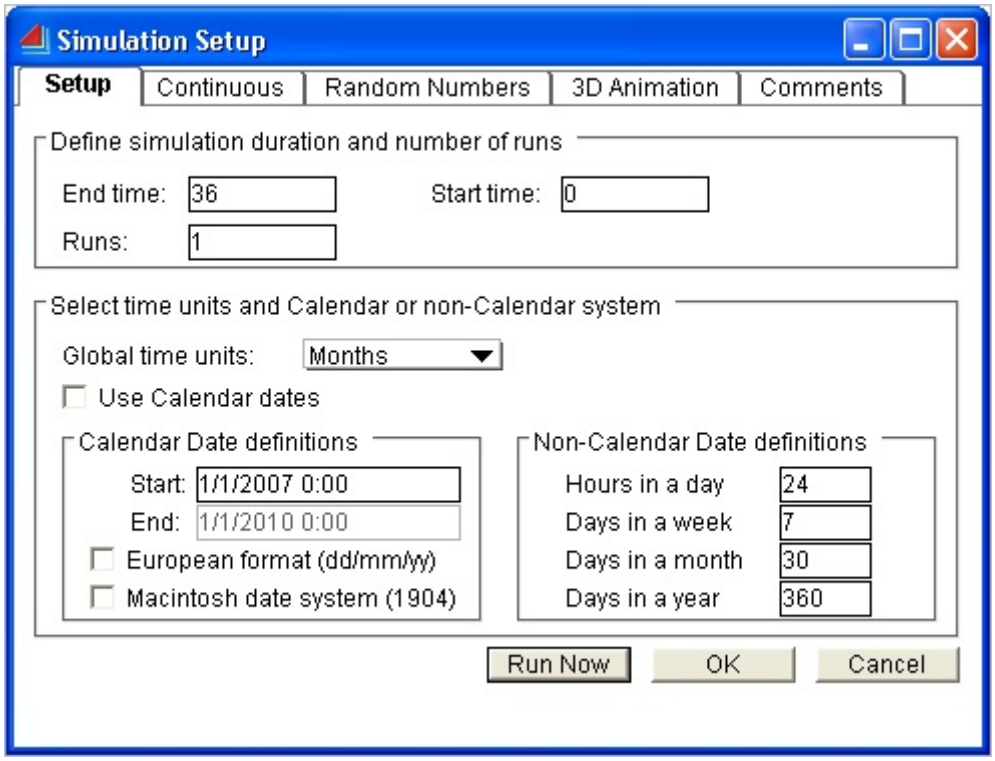


Figura 4.3: Painel *Simulation Setup*, responsável por configurar os parâmetros de execução da simulação.

Após a configuração da simulação, é possível realizar a execução de diversas formas: por meio do botão *Run Now*, localizado no próprio diálogo de configuração; através do atalho de teclado (Ctrl+R); ou através da barra de ferramentas do ambiente.

4.3.5 Ambiente de programação no ExtendSim

Existe um ambiente de desenvolvimento dentro do ExtendSim. Ao acessar a estrutura de um bloco duas janelas serão abertas: uma com a estrutura propriamente dita e outra com o editor de dialogbox (veja a Figura 4.4).

Edição da estrutura do bloco

A janela de estrutura é dividida em quatro frames: ícone, ajuda, código, variáveis.

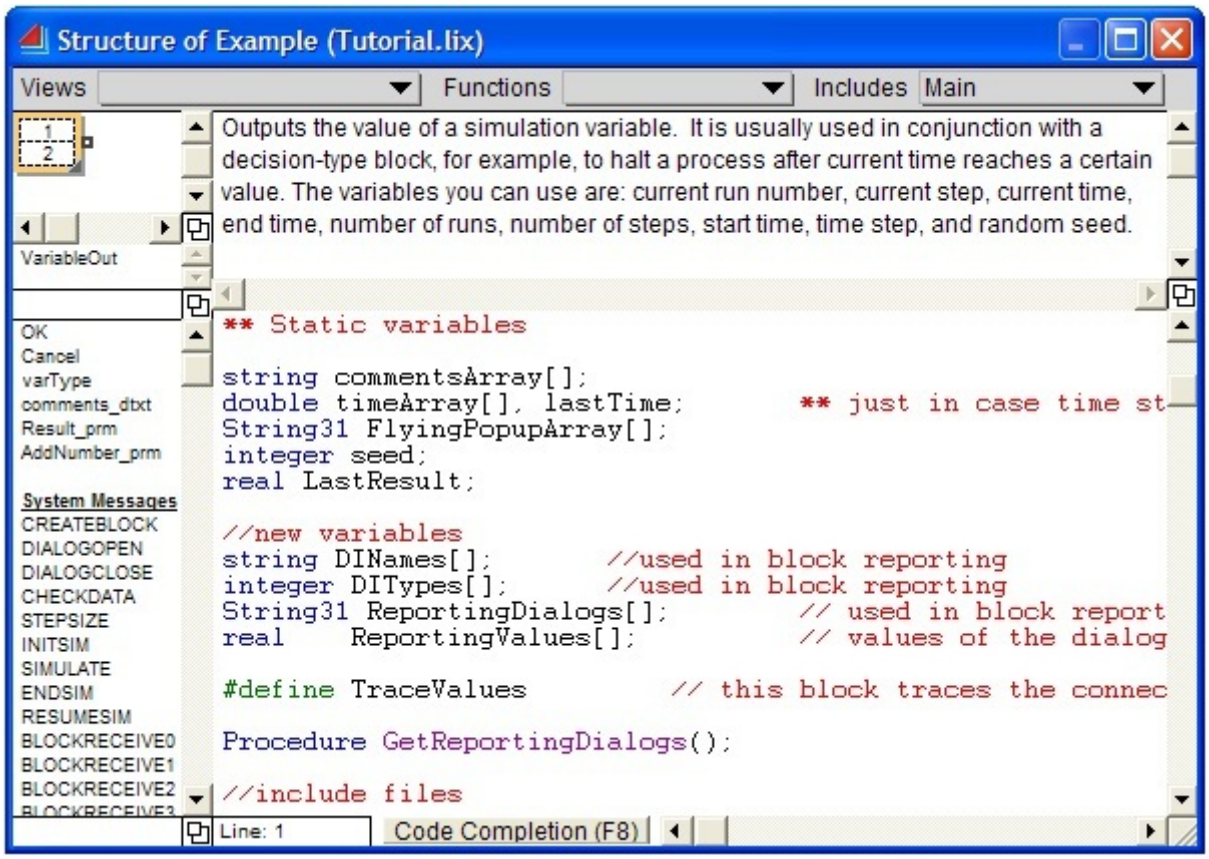


Figura 4.4: Janela de edição da estrutura de blocos do ExtendSim.

- O Ícone, neste *frame* é realizada a edição do ícone que representa o bloco, bem como os conectores e objetos de animação. Existem ferramentas de desenho para criação de ícones, mas, além delas, podem ser coladas figuras existentes.
- O Ajuda, usado para inserir ou editar uma espécie de documentação de ajuda para o bloco. O texto escrito aqui pode ser acessado através do botão *Help* presente no canto inferior esquerdo da caixa de diálogo do bloco.
- O Variáveis exibe todas as variáveis definidas nos itens inseridos na caixa de diálogo do bloco. Como referência, este painel exibe também todas as variáveis de sistema e nomes de mensagens.
- O Código, usado para inserir e editar o código ModL do bloco. Possui a funcionalidade de realçar os códigos a medida que são digitados para torná-los mais compreensíveis. À medida que o código é escrito, existe uma funcionalidade que sugere possíveis funções para uso, como um auto-complete. No momento da compilação, a janela exibe a posição dos erros de sintaxe. Serão abordados os aspectos da ModL mais adiante.

Edição do diálogo do bloco

Como já explicado, a caixa de diálogo ou, simplesmente, *dialogbox* do bloco é a interface com o usuário. Ao contrário do que poderia se pensar, alterá-lo é a parte mais simples

da criação/alteração de um bloco. O ExtendSim possui um poderoso editor WYSIWYG *(What you see is what you get)*.

Todos os elementos visíveis no *dialogbox* como textos, tabelas, botões, etc. são conhecidos como *dialog itens* ou itens de diálogo. Cada um destes itens possui suas definições, seus formatos e um nome de variável que os identifica.

Para inserção de novos *dialog itens* existem ferramentas agrupadas em no menu *Develop*, que armazena diversas ferramentas para trabalhar com o desenvolvimento de blocos. Após inserção, os *dialog itens* podem ser usados no código fonte do bloco e são exibidos na janela de edição da estrutura.

Um aspecto negativo do pacote ExtendSim é que a formatação dos elementos das caixas de diálogo se limitam ao posicionamento dos itens na área da caixa. Não é possível alterar a fonte ou o tamanho dela, o que prejudica a visualização de dados ou a customização para pessoas com problemas de visão ou de idades mais avançadas, forçando a diminuição da resolução do monitor para tentar visualizar melhor as informações na tela, o que acarreta prejuízos para a visualização dos modelos como um todo.

4.3.6 A linguagem ModL

No Ambiente de programação ExtendSim, a linguagem utilizada é a ModL, que possui uma estrutura muito semelhante ao C. Além disso, DLLs e bibliotecas compartilhadas provêm um método para ligar o ExtendSim com outras linguagens de programação, como C++, Java, etc.

Por ter uma estrutura similar ao C, o aprendizado da linguagem para o desenvolvimento deste trabalho não foi de grande dificuldade. Houve maiores problemas com as estruturas de dados que tiveram que ser criadas internamente, pois basicamente não existiam estruturas compatíveis com o trabalho realizado.

Por meio da linguagem Modl é possível acessar todos os elementos do bloco criados a partir do *dialogbox editor*, bastando apenas usar a variável que foi definida no momento da criação do *dialog item*. Existem *message handlers* que interpretam mensagens vindas da simulação, de outros blocos ou da interação com o *dialogbox* do próprio bloco, para executar um bloco de instruções definidas, de forma muito semelhante com o *Visual Basic*.

As declarações no corpo do *message handler* têm o mesmo formato das demais declarações do Modl, que são extremamente parecidas com C. Na Figura 4.5 é mostrado um exemplo de *message handler* usando Modl no ExtendSim [13].

```
on messagename
  {
   one or more statements;
  }
```

Figura 4.5: *Massage handler* usando ModL no ExtendSim.

Como observado durante esta seção, o ModL é muito semelhante ao C. Realmente e a sintaxe das linguagens se comporta de maneira semelhante até mesmo na questão de ignorar espaços em branco, e terminar sentenças com ponto-e-virgula. Contudo, segue uma tabela demonstrando algumas diferenças entre ModL e C:

Tabela 4.3: Alguns diferenças entre Modl e C.

ModL	**C**
Case insensitive	Case sensitive
real ou double (Mac OS e Windows: 16 dígitos significatívos	double
integer ou long (32bits)	long
String ou Str255 (255 caracteres no máximo)	typdef struct para definir a estrutura equivalente
Subscripts que verificam limites dos arrays, exibindo mensagens de erros em caso de buffer overflow	Não há tratamento quanto aos limites do array
Funções declaradas apenas usando padrão ANSI (declarações de protótipos)	Funções podem ser declaradas nos padrões KeR ou ANSI
Funções e *message handlers* podem ser sobrescritos	Não disponível em C, mas disponível em C++
O laço *for* tem a seguinte sintaxe: for(declaração;boleano;declaração)	O laço *for* tem a seguinte sintaxe: for(expressão;expressão;expressão)
é operador de exponenciação (como usado em BASIC e em planilhas eletrônicas)	Em C, é usado somente como operador OU exclusivo em extressões lógicas
Concatenaçao de *strings* ocorre utilizando o operador +	Em C é preciso utilizar a função strcat

Para um conhecimento mais aprofundado sobre o ExtendSim, sugeri-se a consulta aos livros/manuais do ExtendSim [13, 14].

Capítulo 5

Redes Neuronais Artificiais (RNA's)

As redes são usadas para modelar inúmeros fenômenos em física, ciência da computação, biologia, bioquímica, etologia, matemática, sociologia, economia, telecomunicações e várias outras áreas. Isto ocorre porque muitos sistemas podem ser vistos como uma rede: células, computadores, população, etc.

Neste capítulo aborda-se os conceitos fundamentais de redes neuronais artificiais. Apresentando os fundamentos biológicos e historicos, as caracteristicas estruturais, de aprendizagem e de treinamento, usando o algoritmo de aprendizagem *backpropagation*, das redes neuronais artificiais.

5.1 Introdução as Redes Neuronais

5.1.1 Fundamentos biológicos

O sistema nervoso humano é responsável por gerar e transmitir estímulos que chegam ou partem do cérebro. Constituído de células denominadas neurônios (como mostrado na Figura 5.1), o cérebro é responsável pela decisão, integração dos pensamentos e sensações e pela adaptação do organismo e do próprio ser, sendo esta última função realizada através do aprendizado.

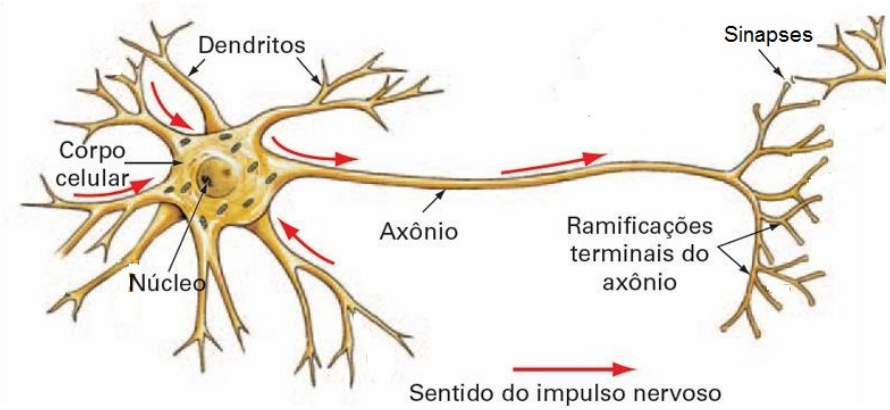


Figura 5.1: Neurônio natural.

O cérebro é um computador (sistema de processamento de informação) altamente complexo, não-linear e paralelo. Ele tem a capacidade de organizar suas células (neurônios) estruturais de forma a realizar certos processamentos de reconhecimento de padrões, percepção e controle motor inúmeras vezes mais rápido que os computadores digitais existentes atualmente [12].

O neurônio é delimitado por uma fina membrana celular que possui determinadas propriedades, essenciais ao funcionamento da célula. A partir do corpo celular projetam-se extensões filamentares, os dendritos e o axônio. Os neurônios são definidos como células polarizadas capazes de receber sinais em seus dendritos e transmitir informações por seus axônios. Ao ser excitado, um neurônio transmite informações, através de impulsos, chamados potenciais de ação para outros neurônios. Estes sinais são propagados como ondas pelo axônio da célula e convertidos para sinais químicos nas sinapses.

O neurônio biológico pode ser visto como o dispositivo computacional elementar do sistema nervoso, composto de muitas entradas e uma saída. As entradas são formadas através das conexões sinápticas que conectam os dendritos aos axônios de outras células nervosas. Os sinais que chegam por estes axônios são pulso elétricos conhecidos como impulsos nervosos ou potenciais de ação e constituem a informação que o neurônio processa para produzir como saída um impulso nervoso no seu axônio [18].

5.1.2 Histórico das Redes Neuronais Artificiais

As Redes Neuronais Artificiais ou também denominadas Redes Neurais Artificiais (RNA's) surgiram por volta de 1943, quando o neurofisiologista Warren McCulloch e o matemático Walter Pitts, da Universidade de Illinois, fizeram uma analogia entre as células nervosas e o processo eletrônico num artigo publicado no *Bulletin of Mathematical Biophysics* com o título: *A Logical Calculus of the Ideas Immanent in Nervous Activity* [12].

Em 1949, o biólogo e psicólogo Donald Hebb, estudando o comportamento dos animais, reforçou as teorias do condicionamento psicológico pela propriedade de neurônios individuais. O estudo de Hebb propôs um princípio de aprendizado em sistemas nervosos complexos, ou seja, uma lei que descreve o funcionamento quantitativo da sinapse e do processo de treinamento humano, publicado no seu livro *The Organization of Behavior*.

Em 1951, Marvin Minsky, co-fundador do Laboratório de Inteligência Artificial do MIT, construiu o SNARC, o primeiro simulador de cadeia neural. O SNARC podia ajustar seus pesos sinápticos automaticamente. Apesar de não chegar a executar alguma função de processamento de informação interessante, ele serviu de fator motivador para ideias que surgiram posteriormente. Em 1956, na Primeira Conferência Internacional de Inteligência Artificial, o pesquisador da IBM, Nathaniel Rochester, apresentou um modelo de rede neural artificial que consistia numa simulação de centenas de neurônios interconectados através de um sistema que verificaria como a rede responderia aos estímulos ambientais.

Já em 1959, Frank Rosenblatt, na Universidade de Cornell, criou uma rede de múltiplos neurônios do tipo discriminadores lineares e a batizou de rede perceptron. Rosenblatt baseou-se nas linhas de pensamento de McCulloch para desenvolver o seu modelo matemático de sinapse humana. Devido as suas complexas pesquisas e inúmeras contribuições técnicas, muitos o consideram como fundador da neurocomputação.

No final da década de 1950, Minsky e Seymour Papert lançaram em uma obra chamada *Perceptron*, a qual demonstrava que o modelo apresentado por Rosenblatt não era muito promissor, devido ao uso de técnicas empíricas, das grandes dificuldades da matemática envolvida e dos poucos recursos computacionais disponíveis na época. A publicação de Minsky e Papert acabou esfriando as pesquisas e praticamente todo o investimento financeiro nesta área foi cancelado.

Nos anos seguintes, muitos artigos foram publicados, e várias previsões exageradas e pouco confiáveis para a época foram anunciadas [12]. Enquanto Rosenblatt trabalhava no *Perceptron*, Bernard Widrow e alguns de seus estudantes da Universidade de Stanford desenvolveram um novo modelo de processamento de redes neurais chamado de Adaline (*ADAptive LINear Elemtent*), a qual se destacava pela sua poderosa lei de aprendizado. O princípio de treinamento para as redes Adalines ficou conhecido como a Regra Delta, que foi mais tarde generalizada para redes com modelos neurais mais sofisticados.

No início da década de 1980, o físico e biólogo de reputação mundial John Hopfield também se interessou pela neurocomputação e escreveu vários artigos, criticando fortemente as teorias apresentadas por Minsky e Papert na década de 1950, que levaram vários cientistas a se unirem nesta nova área emergente. O administrador de programas da DARPA (*Defense Advanced Research Projects Agency*), Ira Skurnick, criou em 1983 as pesquisas em neurocomputação da DARPA. Estes fatos acabaram abrindo novos horizontes para a neurocomputação.

Em 1986, o professor de psicologia da Universidade de Stanford, David E. Rumelhart, e seu colega James L. McClelland, professor de psicologia da Universidade de Carnegie-Mellon, publicaram o livro *Parallel Distributed Processing: Explorations in the Microstructure of Cognition (vol.1: Foundations, vol.2: Psychological and Biological Models)*, expandindo o campo de pesquisa da neurocomputação. Nesse livro, eles apresentam um modelo matemático e computacional que propicia o treinamento supervisionado dos neurônios artificiais. Surgia, então, o algoritmo *backpropagation*, um algoritmo de otimização global sem restrições.

Em 1987 ocorreu a Primeira Conferência de Redes Neurais. Também foi formada a Sociedade Internacional de Redes Neurais (*International Neural Networks Society - INNS*) juntamente com o *INNS Journal* em 1989, do *Neural Computation* e do IEEE *Transactions on Neural Networks* em 1990. A partir desses acontecimentos, muitas instituições formaram institutos de pesquisa e programas de educação em neurocomputação.

5.2 Características das Redes Neurais Artificiais

Neste projeto foram usados para modelar problemas biológicos e médicos as Redes Neurais Artificiais (RNA's) de múltiplas camadas. Um tipo de rede que vê os nós como "neurônios artificiais" (*Perceptrons*). Um neurônio artificial (Figura 5.2) é um modelo computacional inspirado nos neurônios naturais.

Neurônios naturais recebem sinais por meio de sinapses situadas nos dendritos ou membrana do neurônio. Quando os sinais recebidos são fortes o suficiente (ultrapassam um certo limite), o neurônio é ativado e emite um sinal para o axônio. Este sinal pode ser enviado para outro sinapse e pode ativar outros neurônios, conforme Figura 5.1.

Tendo a estrutura fisiológica das redes neurais biológicas e o funcionamento básico dos neurônios, publicados por McCULLOCH e PITTS (1943), pode-se descrever matemati-

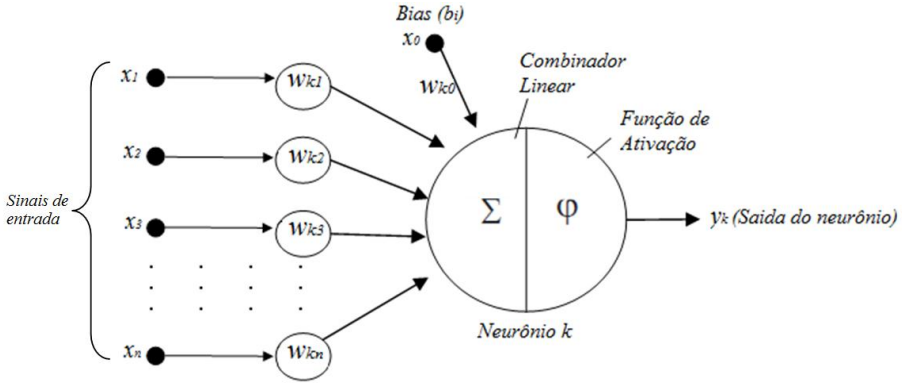
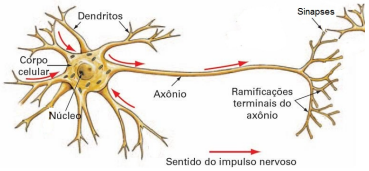
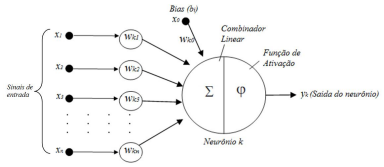


Figura 5.2: Representação gráfica de um *perceptron* (neurônio artificial)

camente um modelo para o neurônio biológico conforme mostrado na Figura 5.2, com as equivalências descritas na Tabela 5.1 [28].

Tabela 5.1: Comparação entre o neurônio biológico (natural) e o neurônio artificial.

Parâmetro	**Neurônio Biológico**	**Neurônio Artificial**
Modelo		
Entrada	Dendritos	$x_1, x_2, x_3, ..., x_n$
Saída	Axônio	y_k
Pesos	Comportamento das sinapses	$W_{k1}, W_{k2}, W_{k3}, ..., W_{kn}$
Efeitos da sinapse	Neurônio pós-sináptico	$S_k = \sum_{i=0}^{n} x_i \cdot W_i$

Os dendritos de um determinado neurônio k são representados pelos n sinais de entrada $x_1, x_2, ..., x_n$, enquanto que a saída y_k representa o axônio deste neurônio. Para representar o efeito gerado pelas sinapses, cada sinal de entrada é associado a um determinado peso $(W_{k1}, W_{k2}, W_{k3}, ..., W_{kn})$ denominado de pesos sinápticos. Que podem assumir tanto valores positivos quanto negativos, representando, desta forma, os efeitos excitatórios ou inibitórios sobre o neurônio. O núcleo de processamento do neurônio (o corpo celular) é representado matematicamente pela junção de um combinador linear com a função de ativação. Basicamente, este combinador linear faz o somatório de todas as entradas do neurônio, sendo que estas entradas são ponderadas pelos respectivos pesos sinápticos. Matematicamente, o combinador linear deste neurônio (S_k) é representado pela Equação 5.1:

$$S_k = \sum_{i=0}^{n} x_i \cdot W_i \quad \text{onde } x_0 \text{ representa a entrada do bias e } w_{k0} \text{ o peso associado a ele.} \tag{5.1}$$

A função de ativação funciona como uma espécie de compressor, ou seja, ela restringe o intervalo dos valores de saída do neurônio para uma faixa de valores previamente escolhida. Além de agir sobre os valores de saída do neurônio, a função de ativação também age sobre o valor bias. O bias é uma espécie de excitador/inibidor dos neurônios da rede e tem a função de aumentar ou reduzir a entrada líquida de sinais, tendo, geralmente, um valor de entrada fixo igual a 1 ou -1. Dessa forma, pode-se inclusive, entender o bias como se fosse uma entrada adicional ao neurônio, porém com um valor fixo.

Em um neurônio biológico, o disparo de um impulso nervoso acontece se o percentual de sinais recebidos fica acima de um determinado limiar de excitação. No primeiro modelo de neurônio matemático proposto por McCulloch e Pitts, a função de ativação usada, denominada função sinal, simula exatamente este comportamento. Se o somatório das entradas do neurônio (S_k) mais o valor do bias (b_k) excederem um determinado limiar (geralmente 0), o valor de saída do neurônio será igual a 1; caso contrário será igual a -1. Matematicamente, isto é representado pela Equação 5.2:

$$y_k = \begin{Bmatrix} 1, & se & S_k+b_k \geq 0 \\ -1, & se & S_k+b_k < 0 \end{Bmatrix} \tag{5.2}$$

A partir deste primeiro modelo de neurônio artificial, surgiram vários outros modelos que passaram a permitir a produção de uma saída com valores que não fossem necessariamente -1 ou 1, utilizando outras funções de ativação.

Uma das funções mais usadas nas implementações das RNA's é a função sigmoide. Ela é uma função monotonamente crescente e seu gráfico tem a forma de S, sendo que existem vários tipos de funções sigmoides, podendo variar o grau de curvatura e a faixa de valores de saída da função. A função de ativação adotada na implementação da RNA apresentada mais à frente nesta monografia é um tipo de função sigmoide denominada função logística. Na Equação 5.3 a seguir é apresentada a formulação matemática e na Figura 5.3, o gráfico da função logística:

$$y_k = \frac{1}{1 + e^{-S_k}} \tag{5.3}$$

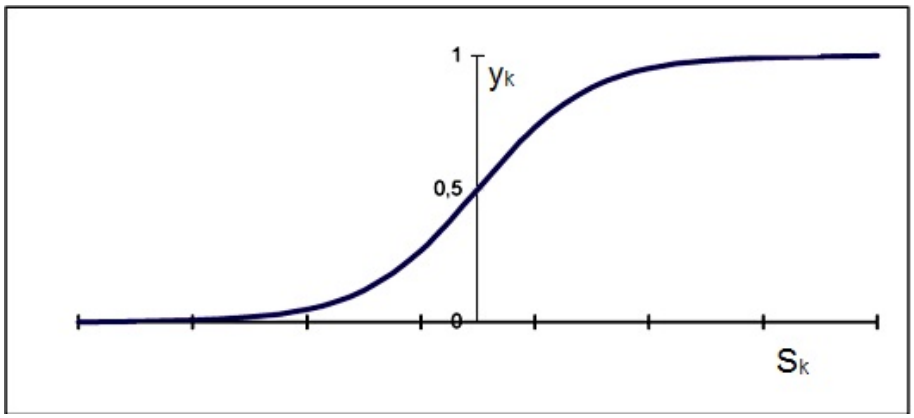


Figura 5.3: Função logística.

As RNA's de múltiplas camadas ou também denominadas *Multilayer Perceptrons*

(MLPs) têm sido aplicadas com sucesso em uma variedade de áreas, desempenhando tarefas como: classificação de padrões (reconhecimento), controle e processamento de sinais [18].

Árvores de decisão e as redes neurais artificiais são as técnicas mais utilizadas nos processos relacionados com mineração de dados [23].

A mineração de dados (*data mining*) é uma das etapas do processo de *Knowledge Discovery in Databases* (KDD) - descoberta de conhecimento em banco de dados, o qual é um processo muito mais complexo que busca extrair conhecimento de uma base de dados. Nesta etapa, são utilizadas certas técnicas, ou um conjunto delas, para extrair os dados mais significativos de uma base de dados e, deles, informações [11].

As redes neurais artificiais são as que despertam um maior interesse dentro da tecnologia de *data mining* e dos processos de descoberta de conhecimento. Contudo, esta técnica possui algumas limitações no que diz respeito à sua facilidade de uso e desenvolvimento, mas, mesmo assim, possui vantagens significativas sobre outras técnicas de mineração de dados. A maior destas vantagens é, sem dúvida, a alta precisão dos modelos de predição, que podem ser aplicados sobre uma extensa gama de problemas reais e de negócios.

Esta técnica implementa padrões de detecção e algoritmos de aprendizado de máquina para construir modelos de predição para bases de dados históricos em larga escala.

Uma rede neural é constituída de três tipos de camadas de neurônios [4], as quais são:

- camada de entrada: onde os padrões e dados são apresentados à rede neural;
- camadas intermediárias (ou escondidas): nestas camadas é feito todo o processamento da rede neural;
- camada de saída: nesta camada o resultado da rede é apresentado ao observador.

Na Figura 5.4 pode ser visto o esquema de uma rede neural artificial.

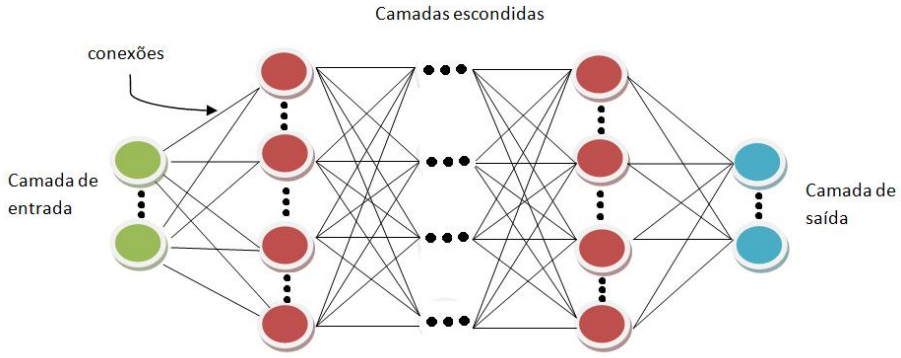


Figura 5.4: Topologia de uma rede neural artificial de múltiplas camadas.

A técnica de redes neurais surgiu da área de inteligencia artificial e parte da premissa de que as máquinas podem "pensar" se houver uma maneira de simular a estrutura e o funcionamento do cérebro humano nos computadores. Apesar de os cientistas ainda estarem longe de entenderem o cérebro humano, as técnicas de redes neurais que são executadas nos computadores já podem simular de certa forma o comportamento do cérebro ou a maneira de pensar do ser humano, em determinados aspectos.

Por sua origem e o sucesso recente que vem alcançando, as técnicas de rede neurais artificiais têm despertado um grande interesse no contexto de construção de modelos preditivos em *data mining*. Para entender melhor como as redes neurais podem detectar padrões em bases de dados, uma analogia é feita com o comportamento humano, ou seja, as redes neurais podem "aprender" a detectar tais padrões e realizar melhores e mais precisas predições da mesma forma que um humano o faz.

As redes neurais realizam este aprendizado, em sentido bastante real da palavra, porém, por trás deste aprendizado, os algoritmos e técnicas que estão sendo desenvolvidos não são totalmente diferentes das técnicas encontradas na estatística ou em outros algoritmos de mineração de dados. Por exemplo, seria injusto afirmar que as redes neurais poderiam superar outras técnicas porque elas podem "aprender" e melhorar no decorrer do tempo, enquanto que as outras técnicas permanecem estáticas [23].

Uma característica marcante das redes neurais é que elas são extremamente automatizadas. Seu usuário final, ou desenvolvedor, não precisa conhecer muito extensamente sobre como elas trabalham, sobre a modelagem de predição, ou mesmo sobre a base de dados para utilizá-las [23]. Naturalmente, isto é uma aproximação grosseira da realizada, mas define bem a facilidade de construção de redes neurais utilizando-se aplicações de mercado. Note-se que o entendimento desses fatores facilita o desenvolvimento dos modelos preditivos e, principalmente, a análise dos resultados. Ainda assim, o desconhecimento dos mesmos não é fator de obstrução do uso desta técnica.

Apesar da facilidade de construção e desenvolvimento de modelos preditivos baseados em redes neurais, especialmente quando utilizadas ferramentas de mercado, algumas decisões de projeto necessitam ser tomadas para se utilizar esta técnica de forma efetiva, como os descritos a seguir.

- quantos nós na rede serão conectados?
- quantas unidades de processamento neuronais devem ser utilizadas?
- quando o treinamento da rede neural deve ser interrompido para evitar um superajuste?

Além disso, as redes neurais são auto-adaptativas, isto é, elas aprendem informações sobre um problema específico. Elas trabalham bem nas tarefas de classificação, sendo muito úteis em mineração de dados [11].

5.3 Aprendizado nas RNA's

Embora tenham sido criadas várias topologias e formas de organização relacionadas às RNA's, que possibilitam a uma rede específica resolver um determinado tipo de problema, a propriedade principal e de importância primordial é a sua capacidade de aprender a partir de seu ambiente e de melhorar o seu desempenho através da aprendizagem [12].

Utilizando a adaptação de [12], da definição dada por Mendel e McClaren (1970), define-se aprendizagem no contexto de redes neurais como:

> *"Aprendizagem é um processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estimulação pelo ambiente no qual a rede está*

inserida. O tipo de aprendizagem é determinado pela maneira pela qual a modificação dos parâmetros ocorre."

Existem várias formas diferentes de aprendizado, que também estão relacionadas à característica topológica da rede e ao aspecto do problema a ser resolvido. Tendo em vista esta variedade de formas de aprendizado, nesta monografia são enfocados os principais conceitos relacionados à forma de aprendizado utilizada na implementação da RNA do software SIMNeural. Ou seja, o aprendizado supervisionado por meio da retropropagação de erros da camada de saída, com a correção dos erros pela regras do algoritmo *Backpropagation*.

As Redes neurais Artificiais podem ser classificadas em duas categorias principais de acordo com o paradigma de redes: redes supervisionadas e não supervisionadas. Os métodos adaptativos que tentam reduzir o erro de saída são os métodos de aprendizado supervisionado, enquanto aqueles que desenvolvem representações internas sem amostras de saída são chamados métodos não supervisionados de aprendizado.

5.3.1 Aprendizagem por correção de erros

A retropropagação de erros da camada de saída pelas camadas escondidas para modificar os pesos das ligações é o método mais comum para se construir um modelo de rede neural. Como as únicas mudanças que podem ocorrer, uma vez que a arquitetura do modelo foi decidida, são os pesos das ligações, estes pesos são alterados para melhorar a precisão do modelo.

Basicamente pela alteração destes pesos é que as redes neurais tendem a não cometer os mesmos erros em utilizações futuras. Se um erro é cometido pela rede neural, este erro pode ser mensurado no nó de saída como sendo a diferença entre o valor do nó de saída e o valor desejado. A saída desejada, porém, só é conhecida quando se está na fase de treinamento, quando os dados históricos são utilizados.

A taxa de aprendizado controla o quão rápido as redes neurais reagirão para um erro em particular [4]. Se esta taxa é alta, cada erro será considerado como muito importante, e os pesos serão modificados significativamente. Caso contrário, se a taxa é baixa, os pesos só serão alterados para a direção correta em cada ocorrência do erro. Se um alto valor para a taxa de aprendizado é utilizado, a rede agirá rapidamente para corrigir qualquer ocorrência de um eventual erro, porém, o próximo erro pode requerer uma grande alteração nos pesos, o que poderá acarretar na reversão das mudanças feitas quando da ocorrência do último erro.

Uma taxa de aprendizado alta pode levar a construção de uma rede neural de forma muito mais rápida, porém, ela pode resultar em uma rede neural que não consiga atingir uma convergência, e os pesos podem mudar rapidamente de um valor para outro com cada ocorrência de um novo erro.

5.3.2 Aprendizagem supervisionada

Ao ser criada uma rede neural são atribuídos os pesos sinápticos (representados por cada conexão entre dois neurônios) de forma aleatória e diz-se que a rede neural é burra, pois ainda não consegue distinguir corretamente nenhum conjunto de entrada que for apresentado a ela.

Para que os pesos sejam ajustados é necessário realizar o treinamento da rede. No momento de treinamento da rede, um subconjunto do conjunto de dados do problema é apresentado a RNA para que ela possa, utilizando um algoritmo de aprendizagem (também chamados de algoritmos de treinamento), ajustar os pesos para obter uma redução no erro entre a saída apresentada pela rede para cada elemento do subconjunto e o valor que era esperado para aquele elemento do subconjunto. Isso significa que, o conjunto de treinamento é composto por valores de entradas e as respectivas saídas desejadas por este conjunto de dados.

Nesta forma de aprendizagem (Figura 5.5) imagina-se a existência de um supervisor (ou professor) externo, que tem o conhecimento sobre o ambiente (o problema em questão). A rede neural não possui nenhum conhecimento deste ambiente.

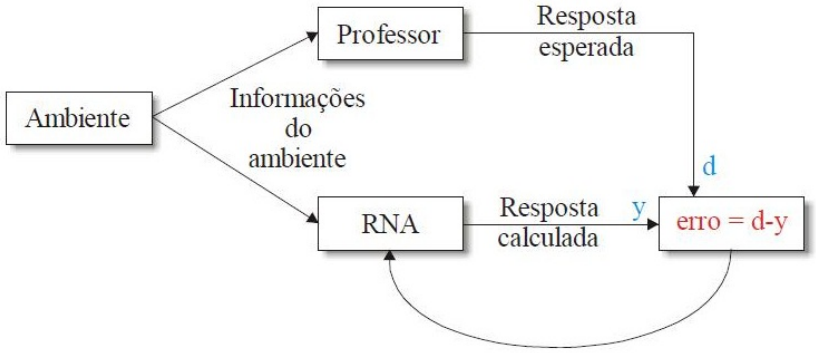


Figura 5.5: Esquema do paradigma de aprendizagem supervisionada.

Por esses motivos, o aprendizado é dito supervisionado. Existe o caso em que a rede é treinada apenas com um conjunto de dados de entrada sem a saída desejada. Neste caso, o qual não é abordado pelo Projeto SIMNeural, o aprendizado é dito não supervisionado.

Os algoritmos de aprendizagem supervisionada são aqueles que objetivam construir um modelo de distribuição de classes (categorias) em função das características dos dados em questão. Esse modelo é formado a partir de um conjunto de treinamento, no qual as classes de suas instâncias são previamente conhecidas. A partir daí, o algoritmo é capaz de extrair características importantes para futuras classificações [17].

No aprendizado supervisionado, são dados para o objeto um, dois ou mais conjuntos de valores: o conjunto de valores de entrada e o conjunto de valores de saída esperados para cada entrada, por exemplo. Desta forma, o treinamento consiste em um problema de otimização dos parâmetros do objeto para que possam responder às entradas conforme esperado e extrapolar o mesmo comportamento para outras entradas não previstas no treinamento.

5.4 Algoritmo de aprendizado para Redes Neurais

O algoritmo *backporpagation* foi estabelecido como o mais popular algoritmo utilizado no contexto do aprendizado de RNAs MLP [18].

A popularidade do algoritmo *backpropagation* resulta de sua relativa simplicidade de implementação e do fato de ser um poderoso dispositivo para armazenar o conteúdo de informação (adquirido pela rede MLP a partir do conjunto de dados) nos pesos sinápticos.

5.4.1 Algoritmo baseado em retropropagação de erro (*Backpropagation*)

O algoritmo de treinamento baseado em retropropagação, também denominado na literatura como *Backpropagation*, consiste no ajuste dos pesos da rede através da retropropagação do erro encontrado na saída. A minimização é conseguida realizando-se, a cada iteração, a atualização dos pesos da rede no sentido oposto ao da função. Trata-se, portanto, de um algoritmo de treinamento supervisionado, determinístico, de computação local, e que implementa o método do gradiente descendente na soma dos quadrados dos erros.

A topologia da arquitetura da rede que utiliza esta regra de aprendizagem é formada, geralmente, por uma ou mais camadas escondidas de neurônios não-lineares e uma camada de saída de neurônios lineares. Devido à grande difusão da arquitetura da rede a que esta regra de aprendizagem se aplica, é comum referir-se a ela com o nome da própria regra de aprendizagem, ou seja, rede *Backpropagation*.

Antes de ser descrito o algoritmo *backpropagation*, é necessário que sejam feitas algumas considerações quanto à notação que será utilizada:

- *unidades de limite (Bias):* O bias é uma espécie de excitador/inibidor dos neurônios da rede e tem a função de aumentar ou reduzir a entrada líquida de sinais, tendo, geralmente, um valor de entrada fixo igual a 1 ou -1.
- *função de ativação:* $y_k = \frac{1}{1+e^{-S_k}}$, onde y_k será substituído por h_j (camada oculta) e O_j (camada de saída)
- *delta erros* $\delta(k)_j$*:* O delta erro está relacionada à derivada da função de ativação.
- *coeficiente de aprendizagem* η*:* O objetivo deste parâmetro é manter a estabilidade do processo de minimização dos erros.
- *termo de momentum* α*:* aumenta a velocidade de aprendizado da rede sem que ela se torne instável.
- *época:* uma época consiste no intervalo correspondente à apresentação de todos os n vetores (entrada-saída) do conjunto de treino.

O algoritmo *Backpropagation* para um treinamento incremental pode ser descrito pelos seguintes passos [25]:

Dado um conjunto de pares de vetor entrada-saída. Computa-se um conjunto de pesos para uma rede de três camadas que mapeia entradas nas saídas correspondentes.

1. Seja A o número de unidades da camada de entrada, conforme determinado pelo comprimento dos vetores de entrada de treinamento. Seja C o número de unidades da camada de saída. Agora escolha B, o número de unidades da camada oculta. Conforme mostra a Figura 5.6, as camadas de entrada e oculta têm, cada uma,

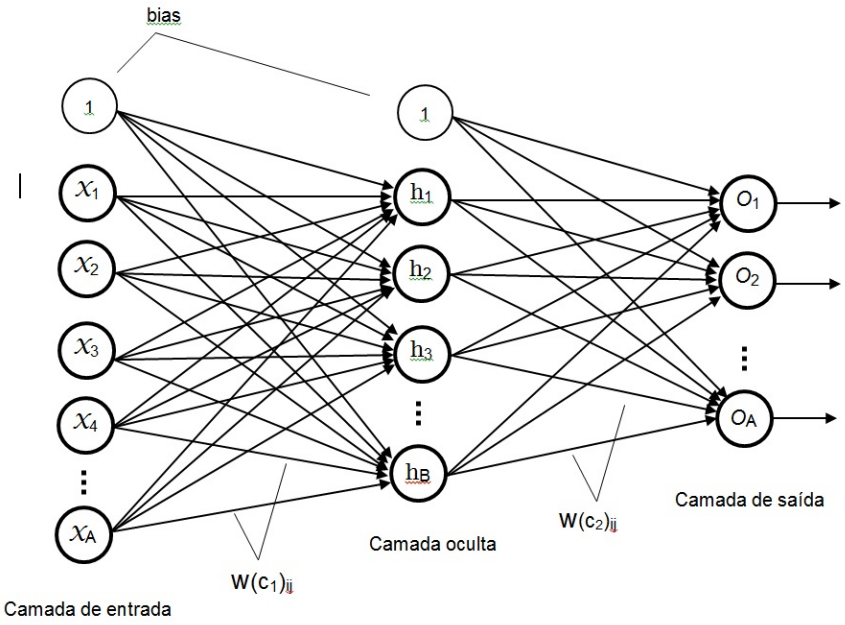


Figura 5.6: Uma rede multicamada.

uma unidade extra usada como limite; portanto, as unidades dessas camadas às vezes serão indexadas pelos intervalos $(0, ..., A)$ e $(0, ..., B)$. Denota-se os níveis de ativação das unidades da camada de entrada por x_j, da camada oculta por h_j e da camada de saída por O_j. Os pesos que conectam a camada de entrada à camada oculta são denotados por $W_{(C_1)ij}$, onde o subscrito i indexa as unidades de entrada e o j, as unidades ocultas. Da mesma maneira, os pesos que conectam a camada oculta à camada de saída são denotados por $W_{(C_2)ij}$, onde i indexa as unidades ocultas e j, as unidades de saída.

2. Inicialize os pesos da rede. Cada peso deve ser ajustado aleatoriamente para um número entre 0 e 1.

$$\begin{aligned} W_{(C_1)ij} &= aleatório(0,1) \quad \text{para todo } i = 0, ..., A, j = 1, ..., B \\ W_{(C_2)ij} &= aleatório(0,1) \quad \text{para todo } i = 0, ..., B, j = 1, ..., C \end{aligned}$$

3. Inicialize as ativações das unidades de limite. Seus valores são fixos.

$$\begin{aligned} x_0 &= 1,0 \\ h_0 &= 1,0 \end{aligned}$$

4. Escolha um para entrada-saída. Suponha que o vetor de entrada seja x_i e que o vetor de saída objeto seja y_i.

5. Propague a ativação das unidades da camada de entrada para as unidades da camada oculta usando a função de ativação:

$$h_j = \frac{1}{1 + e^{-\sum_{i=0}^{A} W_{(C_1)ij} x_i}} \quad \text{para todo } j = 1, ..., B \tag{5.4}$$

6. Propague as ativações das unidades da camada oculta para as unidades da camada de saída.

$$O_j = \frac{1}{1 + e^{-\sum_{i=0}^{B} W_{(C_2)ij} h_i}} \quad \text{para todo } j = 1, ..., C \tag{5.5}$$

7. Compute os erros das unidades da camada de saída, denotada por $\delta 2_j$. Os erros baseiam-se na saída real da rede (O_j), e na saída esperada (y_j).

$$\delta 2_j = O_j \left(1 - O_j\right) \left(y_j - O_j\right) \quad \text{para todo } j = 1, ..., C \tag{5.6}$$

8. Compute os erros das unidades da camada oculta, denominada por $\delta 1_j$.

$$\delta 1_j = h_j \left(1 - h_j\right) \sum_{i=1}^{C} \delta 2_i \cdot W_{(C_2)ji} \quad \text{para todo } j = 1, ..., B \tag{5.7}$$

9. Ajuste os pesos entre a camada oculta e a camada de saída. o coeficiente de aprendizagem é denotado por η; Um valor razoável para η é $0,35$ [25].

$$\Delta W_{(C_2)ij} = \eta \cdot \delta 2_j \cdot h_j \quad \text{para todo } i = 0, ..., A; j = 1, ..., B \tag{5.8}$$

10. Ajuste os pesos entre a camada de entrada e a camada oculta.

$$\Delta W_{(C_1)ij} = \eta \cdot \delta 1_j \cdot x_i \quad \text{para todo } i = 0, ..., A; j = 1, ..., B \tag{5.9}$$

11. Vá para a etapa 4 e repita. Quando todos os pares entrada-saída tiverem sido apresentados à rede, uma época terá sido completada. Repita as etapas de 4 a 10 para tantas épocas quantas forem desejadas.

O algoritmo, usado acima, é generalizado diretamente para redes com mais de três camadas. Para cada camada oculta extra, insira uma etapa de propagação para frente entre as etapas 6 e 7, uma etapa para a computação de erros entre as etapas 8 e 9 e uma etapa para o ajuste dos pesos entre as etapas 10 e 11. A computação de erros para unidades ocultas deve usar a Equação 5.7 da etapa 8, mas i pode abranger as unidades da camada seguinte, não necessariamente a camada de saída.

A velocidade da aprendizagem pode aumentar se alterarmos as etapas 9 e 10, de modificação dos pesos, para que elas incluam um termo de momentum (α). As fórmulas para atualização dos pesos passam a ser:

$$\begin{aligned} &\Delta W_{(C_2)ij}\left(t+1\right) = \eta \cdot \delta 2_j \cdot h_j + \alpha \cdot \Delta W_{(C_2)ij}\left(t\right) \quad \text{para todo } i = 0, ..., A; j = 1, ..., B \\ &\Delta W_{(C_1)ij}\left(t+1\right) = \eta \cdot \delta 1_j \cdot x_i + \alpha \cdot \Delta W_{(C_1)ij}\left(t\right) \quad \text{para todo } i = 0, ..., A; j = 1, ..., B \end{aligned} \tag{5.10}$$

5.5 Treinamento

5.5.1 Preparação dos dados

A maneira que o dado na base de dados de treinamento é alimentado para dentro da rede neural pode gerar um grande impacto no momento em que ele é levado para o treinamento na rede e, da mesma forma, na precisão da rede neural como um todo [23]. Normalmente, o dado é pré-processado a um alto grau para alimentar a rede na forma de um dado cru, que é de tratamento mais fácil. Por exemplo, se considerarmos uma previsão de série temporal, como o próximo preço final de uma ação no pregão da bolsa de valores, o dado cru, aquele sem nenhum pré-processamento, é armazenado na base de dados como uma série de preços no decorrer do tempo em intervalos fixos. Poderia-se imaginar que o preço da ação é capturado de hora em hora (por exemplo, R$30,00 às 11:00h, R$31,00 às 12:00h, R$28,00 às 13:00h, R$30,00 às 14:00h, assim por diante). Este dado pode ser apresentado por uma rede neural por meio da alocação de um nó por ponto de dado a cada hora. Uma forma extremamente melhor de fornecer o mesmo dado para a rede seria informar não só o preço atual da ação, mas também a diferença entre o preço da ação de hora para hora (por exemplo, +R$1,00, -R$3,00, +R$2,00 para os preços finais da ação mencionada anteriormente). O mesmo dado seria apresentado à rede neural para treinamento, mas a codificação de um dado como diferença em preços produzirá um modelo de predição bastante superior.

O mapeamento de previsores categóricos, como a cor do olho, por exemplo, é um pouco mais complexo do que o mapeamento de previsores numéricos [23]. é intuitivamente fácil mapear a renda ou a idade para um valor numérico entre 0.0 e 1.0. Porém, não é tão claro realizar o mapeamento para um previsor categórico com um valor fixo de valores e não determináveis entre diferentes valores. Por exemplo, não faz o menor sentido a pergunta se “loiro” é maior do que “ruivo”. Faz sentido perguntar se uma pessoa é mais velha que a outra. Para resolver este problema, existem algumas técnicas que podem ser utilizadas para codificar previsores categóricos para redes neurais:

- codificação numérica: cada previsor é associado randomicamente a um número, por exemplo, “loiro” = 1, “ruivo” = 2, “preto”= 3 e “castanho” = 4. Estes valores então é que serão escalonados para se ajustar na faixa entre 0.0 e 1.0;
- codificação um-de-n: neste caso, cada valor categórico diferente recebe seu próprio nó de entrada. Por exemplo, existirão quatro nós de entrada para o previsor de cor de cabelo no caso mencionado anteriormente;
- codificação binária: Cada valor de previsor é randomicamente associado a um número e, então, estes números são convertidos em representação binária. Por exemplo, “loiro” = 001, “ruivo” = 010, “preto” = 011 e “castanho” = 100. Um nó de entrada é associado a cada posição na cadeia binária. Por exemplo, três nós são necessários para a codificação binária da cor do cabelo.

5.5.2 Processo de treinamento

O processo a seguir descreve o treinamento de uma rede neural multicamada usando o algoritmo *backpropagation*. Para ilustrar este processo, foi utilizado uma RNA com duas

entradas e uma saída, que é mostrado na Figura 5.7, onde: cada neurônio é composto de duas unidades (Figura 5.2). A primeira unidade, denominada de combinador linear, faz o somatório do produto dos pesos pelos sinais de entrada, conforme Equação 5.1. A segunda unidade recebe o valor de saída do combinador e aplica a função de ativação do neurônio (Equação 5.3).

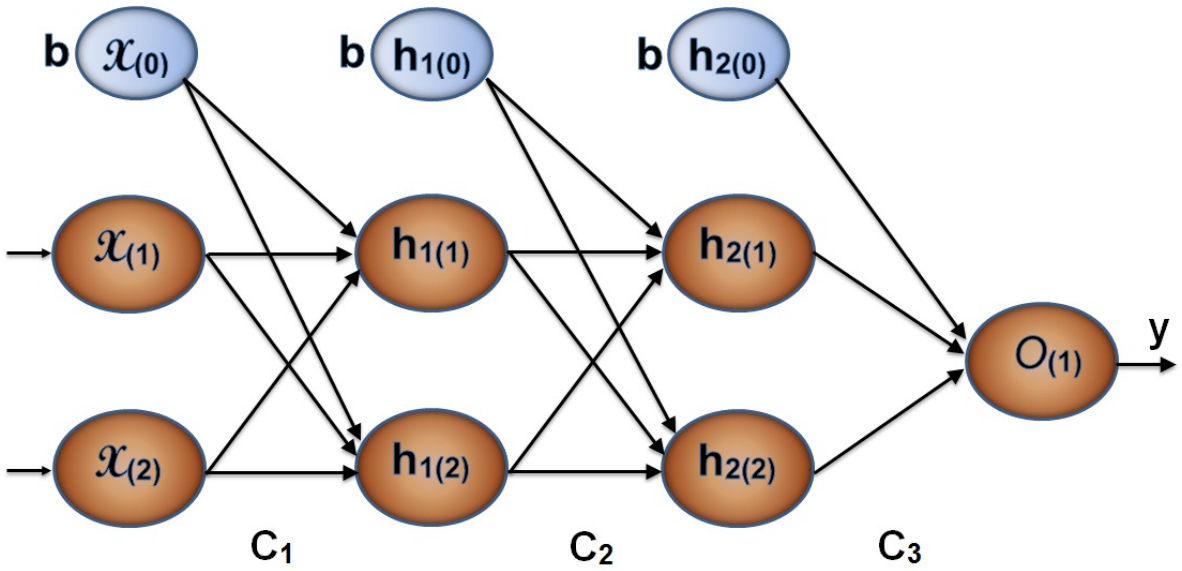


Figura 5.7: RNA com 2 neurônios na camada de entrada, 2 camadas ocultas com 2 neurônios cada e 1 neurônio na camada de saída.

Para ensinar a rede neural precisa-se do conjunto de treinamento. Este conjunto de dados consiste nos valores de entrada (x_1 e x_2) atribuídos a um correspondente valor de saída (desejado) d. O treinamento da rede é um processo iterativo. Usando novos dados do conjunto de treinamento, em cada iteração os valores dos pesos de cada ligação dos nós (neurônios) são modificados. A modificação é calculada usando o algoritmo descrito da seguinte forma:

Propagação: Os dois valores de entrada obtidos a partir do conjunto de treino, a cada passo de ensino, são propagados pela rede. Após esta etapa, pode-se determinar os valores de saída de cada neurônio em cada camada de rede. As figuras a seguir ilustram como os valores de entrada (sinal de *input*) se propagam através da primeira camada da rede. O símbolo $W_{(C_1)ij}$, onde $C_1 = \textit{conexão entre as camadas}\ x_{(i)}$ e $h_{1(j)}$ com $i = \{0, 1, 2\}$ e $j = \{0, 1, 2\}$, representa os pesos das conexões entre a camada de entrada e a primeira camada oculta da rede. O Símbolo $y_{1(n)}$ representa o sinal de saída do neurônio $h_{1(n)}$, onde $n = \{1, 2\}$.

$$y_{1(1)} = \frac{1}{1 + e^{-S_{1(1)}}} \quad \text{onde: } S_{1(1)} = \sum_{i=0}^{2} W_{(C_1)i1} \cdot x_i \tag{5.11}$$

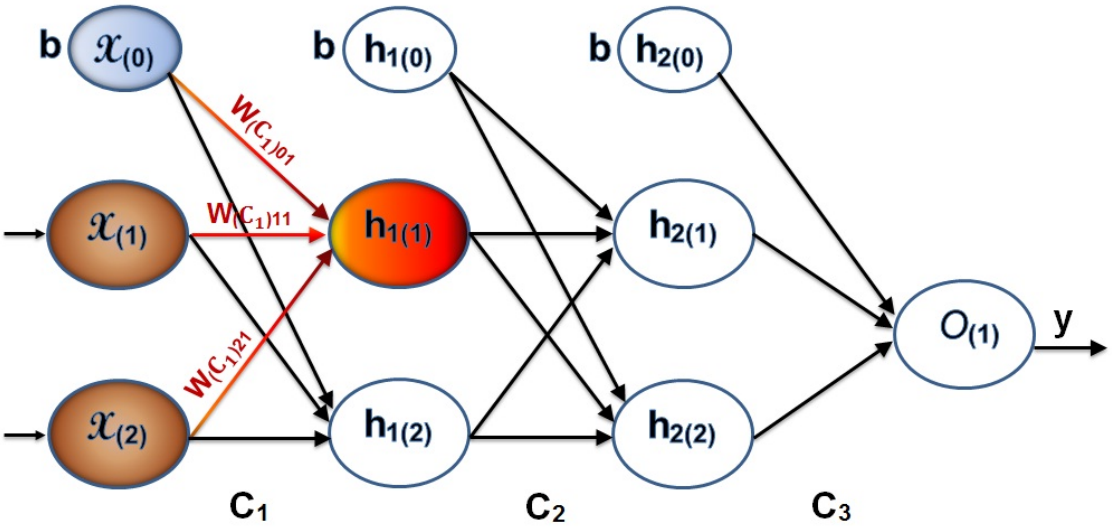


Figura 5.8: Representação gráfica da equação 5.11.

$$y_{1(2)} = \frac{1}{1 + e^{-S_{1(2)}}} \quad \text{onde: } S_{1(2)} = \sum_{i=0}^{2} W_{(C_1)i2} \cdot x_i \tag{5.12}$$

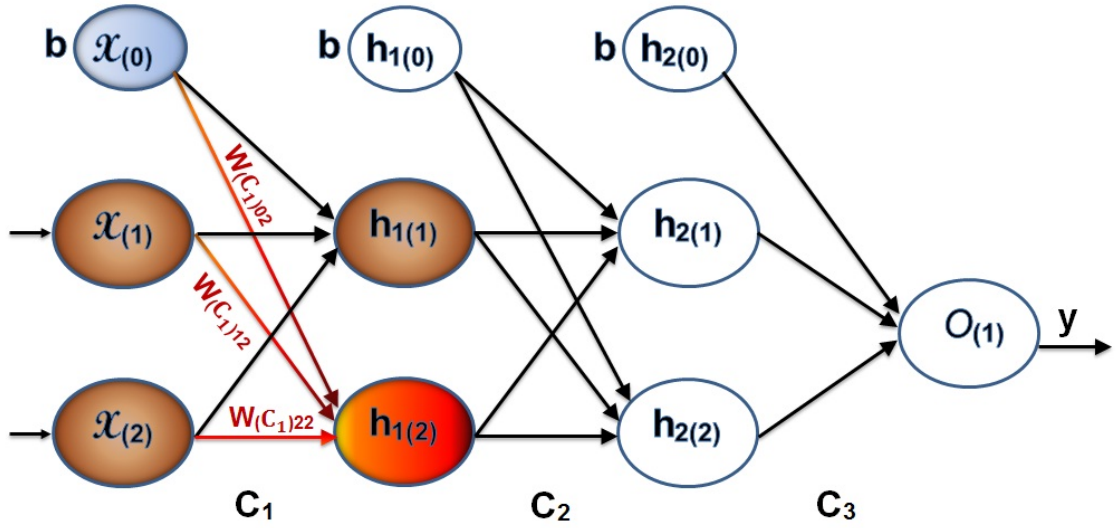


Figura 5.9: Representação gráfica da Equação 5.12.

Propagação de sinais através da camada oculta. O Símbolo $W_{(C_1)ij}$, onde $C_1 = \textit{conexão entre as camadas } h_{1(i)}$ e $h_{2(j)}$ com $i = \{0, 1, 2\}$ e $j = \{0, 1, 2\}$, representa os pesos das conexões entre a saída do neurônio $h_{1(i)}$ e da entrada do neurônio $h_{2(i)}$ na camada seguinte.

$$y_{2(1)} = \frac{1}{1 + e^{-S_{2(1)}}} \quad \text{onde: } S_{2(1)} = \sum_{i=0}^{2} W_{(C_2)i2} \cdot y_{1(i)}, \text{ para } i = 0 \text{ temos } y_{1(i)} = h_{1(i)}. \quad (5.13)$$

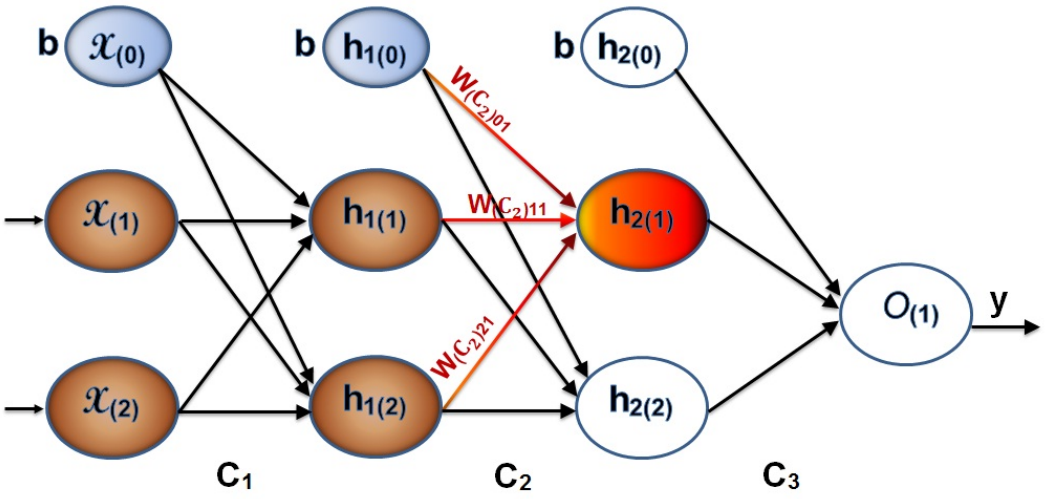


Figura 5.10: Representação gráfica da Equação 5.13.

$$y_{2(2)} = \frac{1}{1 + e^{-S_{2(2)}}} \quad \text{onde: } S_{2(2)} = \sum_{i=0}^{2} W_{(C_1)i2} \cdot y_{1(i)}, \text{ para } i = 0 \text{ temos } y_{1(i)} = h_{1(i)}. \quad (5.14)$$

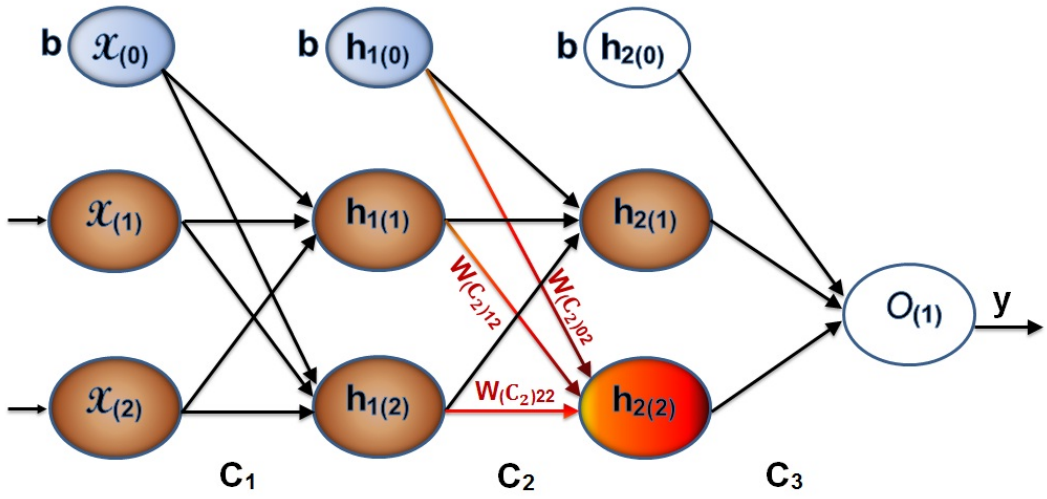


Figura 5.11: Representação gráfica da Equação 5.14.

Propagação de sinais por meio da camada de saída.

$$y = \frac{1}{1 + e^{-S_{O(1)}}} \qquad \text{onde: } S_{O(1)} = \sum_{i=0}^{2} W_{(C_3)i1} \cdot y_{2(i)}, \text{ para } i = 0 \text{ temos } y_{2(i)} = h_{2(i)}. \quad (5.15)$$

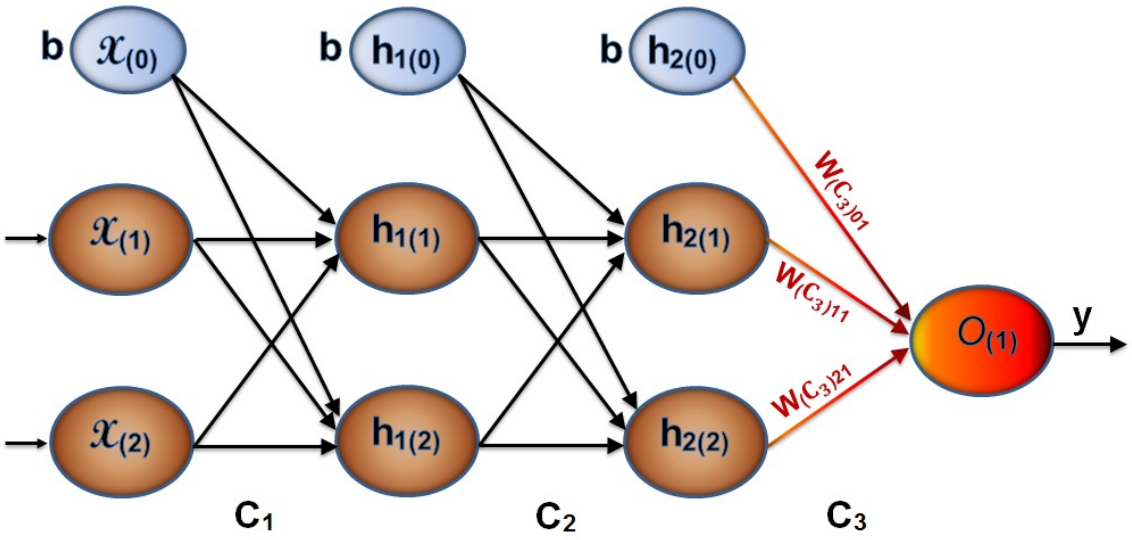


Figura 5.12: Representação gráfica da Equação 5.15.

No passo seguinte, o valor de saída da rede (y) é comparado com o valor de saída desejado (d), que é um dado fornecido pelo conjunto de treinamento supervisionado. A diferença é chamado de sinal de erro de neurônio da camada de saída (δ).

$$\delta = y - d \quad (5.16)$$

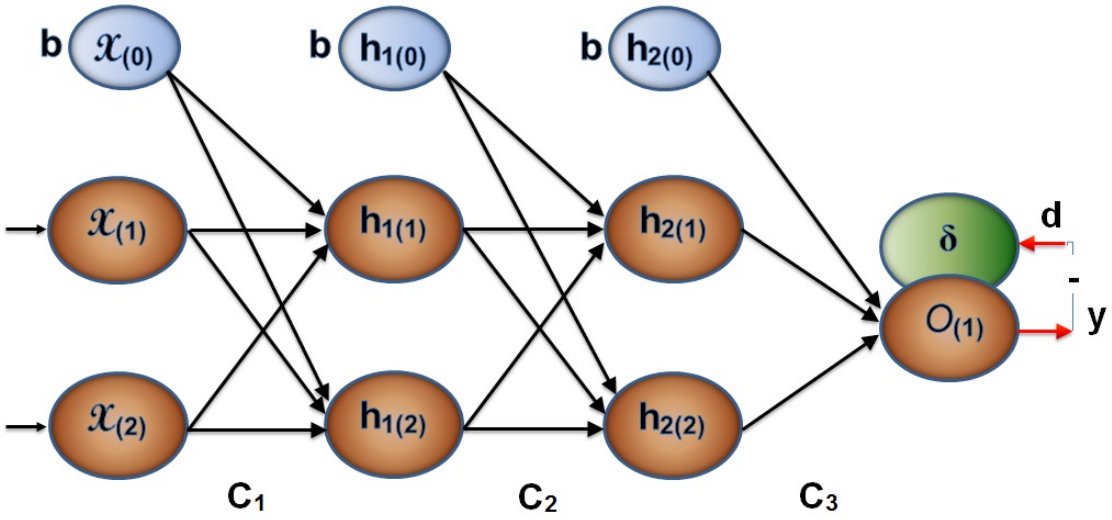


Figura 5.13: Representação gráfica da Equação 5.16.

Retropropagação: É impossível o calcular sinal de erro dos neurônios internos diretamente, porque os valores desejados de saída destes neurônios são desconhecidos. Por muitos anos um método eficaz para fazer o treinamento das redes de multicamadas era desconhecido. Apenas em meados da década de oitenta esse problema foi resolvido com o algoritmo backpropagation. A idéia é propagar o erro de sinal δ (calculado na última camada) de volta para todos os neurônios, que emitiram sinais para os neurônio da camada de saída.

$$\delta_{2(1)} = W_{(C_3)11} \cdot \delta \tag{5.17}$$

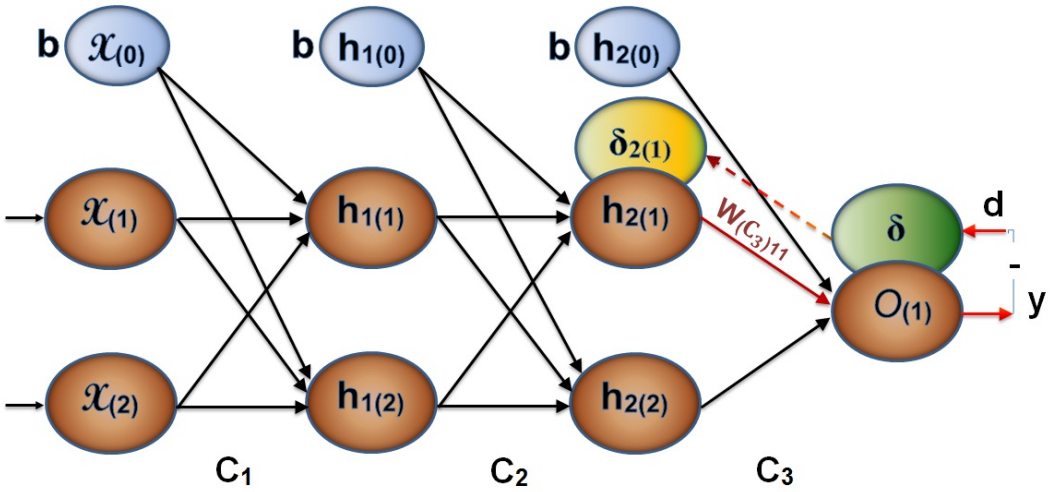


Figura 5.14: Representação gráfica da retropropagação do erro (Equação 5.17).

$$\delta_{2(2)} = W_{(C_3)21} \cdot \delta \tag{5.18}$$

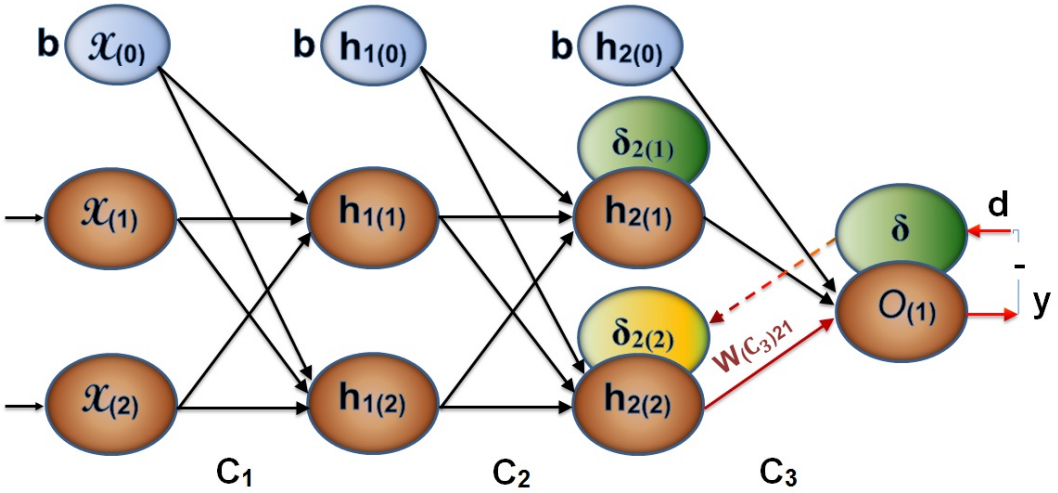


Figura 5.15: Representação gráfica da retropropagação do erro (Equação 5.18).

$$\delta_{1(1)} = W_{(C_2)11} \cdot \delta_{2(1)} + W_{(C_2)12} \cdot \delta_{2(2)} \tag{5.19}$$

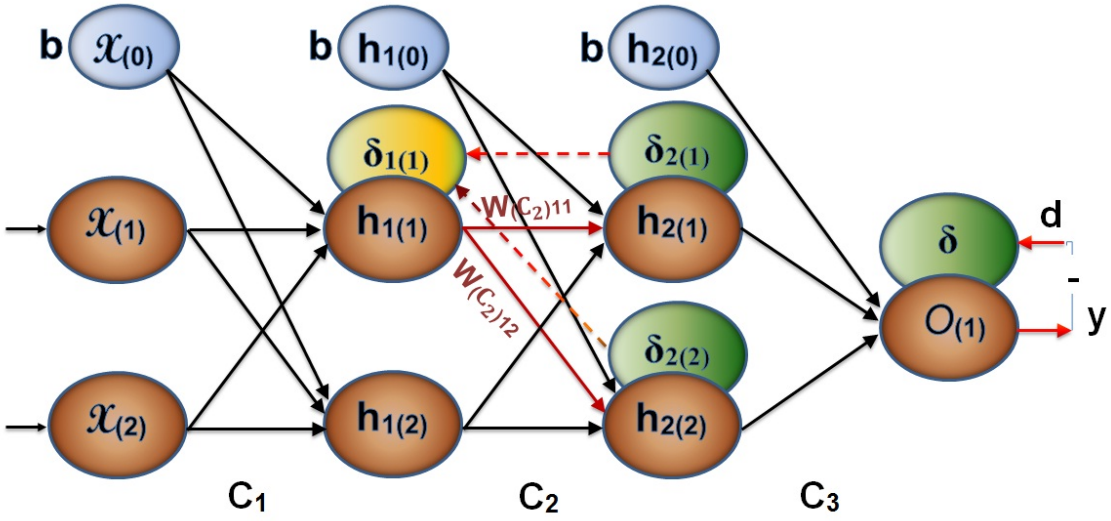


Figura 5.16: Representação gráfica da retropropagação do erro (Equação 5.19).

$$\delta_{1(2)} = W_{(C_2)21} \cdot \delta_{2(1)} + W_{(C_2)22} \cdot \delta_{2(2)} \tag{5.20}$$

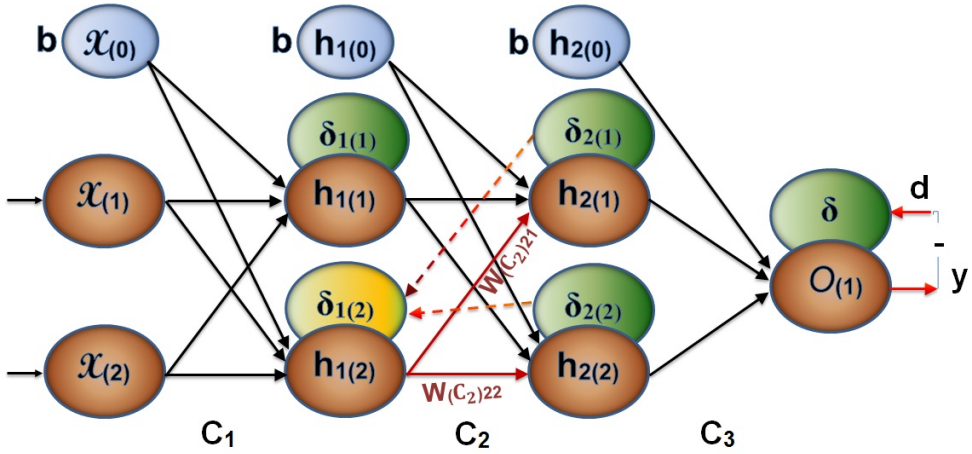


Figura 5.17: Representação gráfica da retropropagação do erro (Equação 5.20).

Quando o sinal de erro de cada neurônio é computado, os coeficientes de pesos das conexões de entrada do neurônio podem ser modificados. Notem-se os cálculos nas fórmulas a seguir:

$$
\begin{aligned}
W'_{(C_1)01} &= \alpha \cdot W'^{-1}_{(C_1)01} + \eta \cdot \delta_{1(1)} \cdot x_{(0)} \\
W'_{(C_1)11} &= \alpha \cdot W'^{-1}_{(C_1)11} + \eta \cdot \delta_{1(1)} \cdot x_{(1)} \\
W'_{(C_1)21} &= \alpha \cdot W'^{-1}_{(C_1)21} + \eta \cdot \delta_{1(1)} \cdot x_{(2)}
\end{aligned}
\tag{5.21}
$$

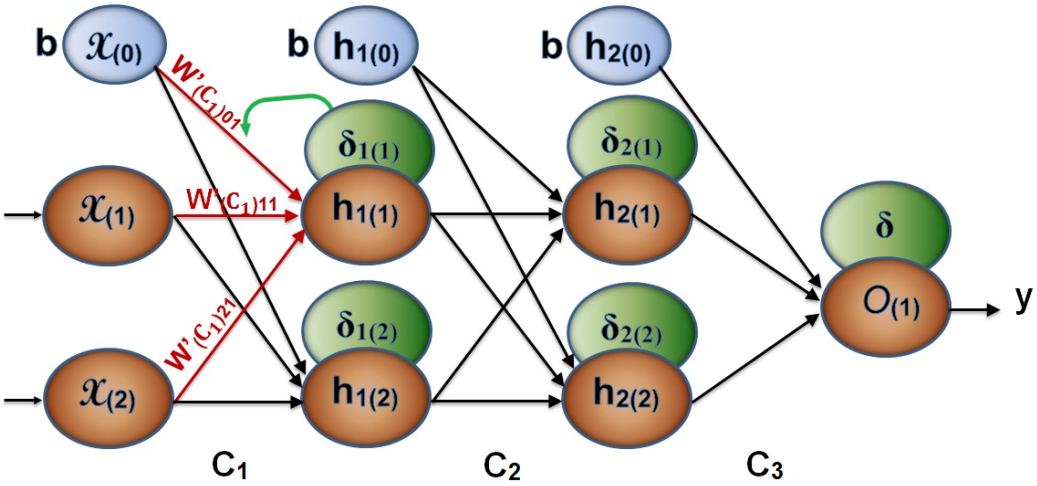


Figura 5.18: Representação gráfica da alteração dos pesos (Equação 5.21).

$$
\begin{aligned}
W'_{(C_1)02} &= \alpha \cdot W'^{-1}_{(C_1)02} + \eta \cdot \delta_{1(2)} \cdot x_{(0)} \\
W'_{(C_1)12} &= \alpha \cdot W'^{-1}_{(C_1)12} + \eta \cdot \delta_{1(2)} \cdot x_{(1)} \\
W'_{(C_1)22} &= \alpha \cdot W'^{-1}_{(C_1)22} + \eta \cdot \delta_{1(2)} \cdot x_{(2)}
\end{aligned}
\tag{5.22}
$$

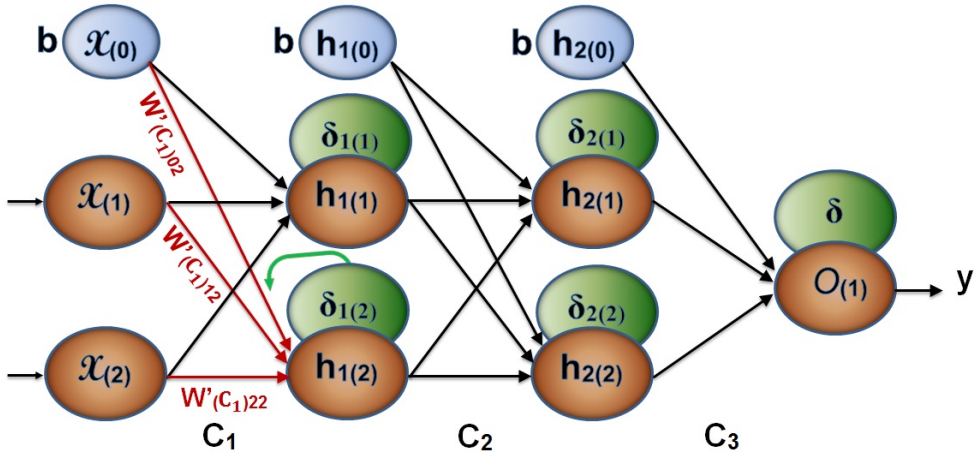


Figura 5.19: Representação gráfica da alteração dos pesos (Equação 5.22).

$$
\begin{aligned}
W'_{(C_2)01} &= \alpha \cdot W'^{-1}_{(C_2)01} + \eta \cdot \delta_{2(1)} \cdot y_{1(0)} \\
W'_{(C_2)11} &= \alpha \cdot W'^{-1}_{(C_2)11} + \eta \cdot \delta_{2(1)} \cdot y_{1(1)} \\
W'_{(C_2)21} &= \alpha \cdot W'^{-1}_{(C_2)21} + \eta \cdot \delta_{2(1)} \cdot y_{1(2)}
\end{aligned}
\tag{5.23}
$$

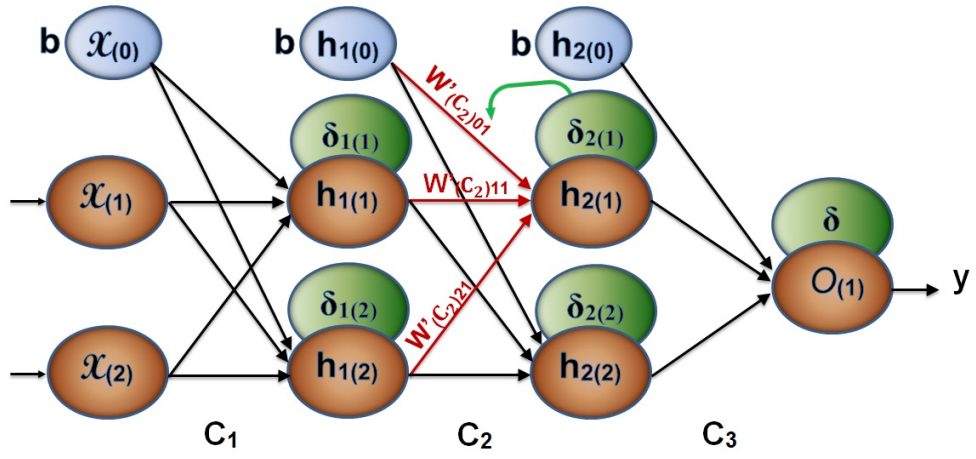


Figura 5.20: Representação gráfica da alteração dos pesos (Equação 5.23).

$$
\begin{aligned}
W'_{(C_2)02} &= \alpha \cdot W'^{-1}_{(C_2)02} + \eta \cdot \delta_{2(2)} \cdot y_{1(0)} \\
W'_{(C_2)12} &= \alpha \cdot W'^{-1}_{(C_2)12} + \eta \cdot \delta_{2(2)} \cdot y_{1(1)} \\
W'_{(C_2)22} &= \alpha \cdot W'^{-1}_{(C_2)22} + \eta \cdot \delta_{2(2)} \cdot y_{1(2)}
\end{aligned}
\tag{5.24}
$$

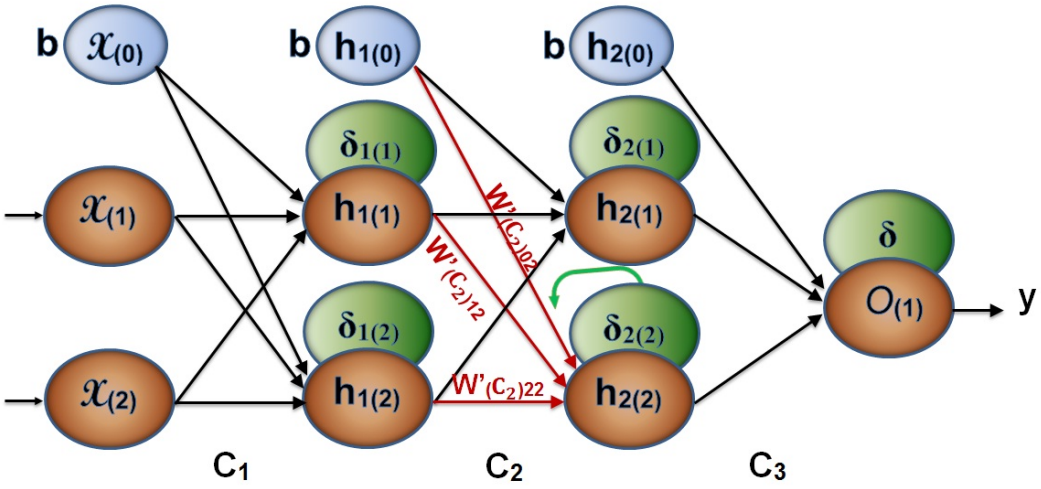


Figura 5.21: Representação gráfica da alteração dos pesos (Equação 5.24).

$$
\begin{aligned}
W'_{(C_3)01} &= \alpha \cdot W'^{-1}_{(C_3)01} + \eta \cdot \delta \cdot y_{2(0)} \\
W'_{(C_3)11} &= \alpha \cdot W'^{-1}_{(C_3)11} + \eta \cdot \delta \cdot y_{2(1)} \\
W'_{(C_3)21} &= \alpha \cdot W'^{-1}_{(C_3)21} + \eta \cdot \delta \cdot y_{2(2)}
\end{aligned}
\tag{5.25}
$$

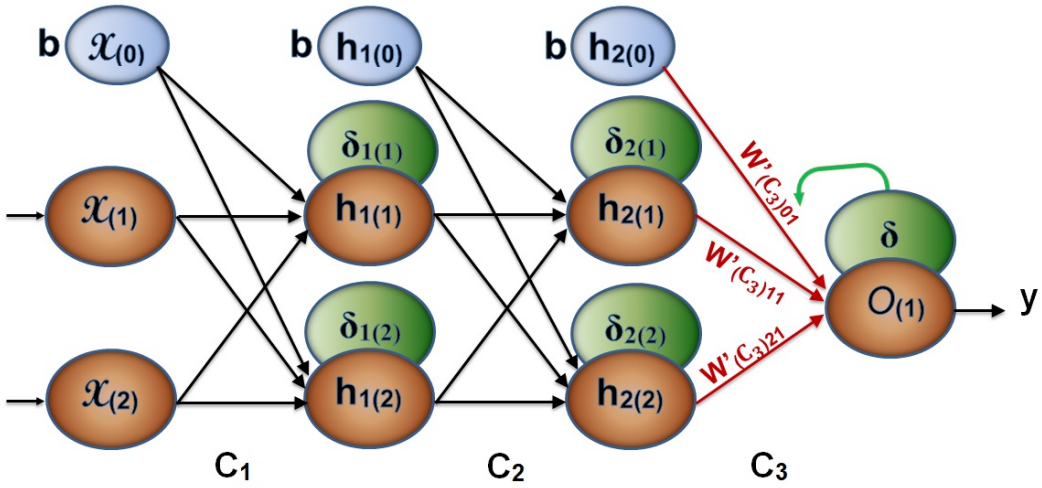


Figura 5.22: Representação gráfica da alteração dos pesos (Equação 5.25).

Finalmente, no processo de treinamento, deve-se considerar a capacidade de generalização da rede, quando treinada com o algoritmo *backpropagation*. Basicamente, diz-se

que a rede tem uma boa generalização quando o mapeamento de entrada-saída feito por ela for correto (ou aproximado) para dados de entrada que não foram utilizados na fase de treinamento. Isto significa que a rede deve fornecer respostas próximas ou iguais das respostas desejadas fornecidas pelo treinador da rede, para dados ainda não utilizados.

À medida em que o treinamento da rede avança, de uma forma geral, ela tende a fornecer respostas cada vez mais próximas das respostas desejadas. No entanto, se a rede for exposta a um treinamento excessivo, que é chamado de *overtraining*, a sua capacidade de generalização irá diminuir. Isto se deve ao fato de que a rede começa a memorizar os padrões de treinamento e desta forma grava todas as peculiaridades destes dados. O *overtraining* faz com que a taxa de acerto para padrões não utilizados no treinamento comece a cair a partir de um determinado ponto do treinamento.

Com isso mostra-se a parte de treino. Para fazer a parte de teste da rede seria necessário pegar um conjunto de dados (possíveis entradas), que não foram usados no treino, e passar pela rede propagando até o cálculo dos δ (Equação 5.16). Se, para cada padrão de entrada, a diferença entre o valor desejado e o calculado pela rede fosse pequena, então ele seria classificado como certo ou pertencente a classe que a saída representa.

5.6 Considerações e aplicações de Redes Neurais

A aplicação de redes neurais em bases de dados novas é relativamente inexpressiva. Cada registro pode ser percorrido por um série de multiplicações, adições e ações de filtragem que compõem cada nó da rede neural. Na verdade, as redes neurais podem ser aplicadas à base de dados em paralelo, na qual cada processador contém uma cópia da rede neural e os pesos das ligações treinadas, além de um subconjunto do total de registros a serem manipulados.

O modelo atual é justamente o peso da ligação e o padrão de conexão entre os nós. Como a maioria das redes neurais hoje em dia é totalmente conectada entre camadas, a estrutura da arquitetura da rede pode ser armazenada pela simples noção da camada na qual um determinado nó está e da camada onde os pesos da ligação que o conecta ao próximo nível estão.

O treinamento de redes neurais pode ser extremamente caro no aspecto computacional [4] [23]. Normalmente, o treinamento de redes neurais envolve os seguintes passos:

1. criar uma rede inicial com pesos randômicos associados às ligações;
2. executar cada registro da base de dados de treinamento através da rede e utilizar o algoritmo de aprendizado para corrigir os erros;
3. verificar o critério de parada. Preferencialmente, utilizando validação cruzada para checar os ajustamentos. O treinamento também pode ser paralisado quando um mínimo pré-estabelecido de erros de treinamento for alcançado;
4. se o critério de parada não for alcançado, retornar ao passo 2 e repetir o processo por toda a base de dados de treinamento.

Esta técnica de treinamento pode seguir indefinidamente [23]. Cada aplicação da base de dados é chamada de época de treinamento, ou janela de treinamento, e não é absolutamente incomum para uma rede necessitar de centenas de janelas de treinamento

para alcançar resultados aceitáveis ou expressivos. O tempo de treinamento depende de uma série de fatores, como as dificuldades do problema e da própria base de dados. Grandes bases de dados heterogêneas levarão um tempo maior do que pequenas bases de dados homogêneas, porém, aquelas podem conseguir resultados superiores. A taxa de aprendizado que deve ser especificada pode ter um efeito significativo no total do tempo para transformar a rede em um modelo consistente. Se a taxa for muito grande, a rede pode oscilar entre valores para os pesos das ligações e nunca convergir. Uma taxa muito baixa pode tornar o processo proibitivamente lento.

Um ponto contrário às redes neurais é a dificuldade de entendimento do modelo que ela construirá, assim como a maneira pela qual os dados originais afetam as respostas dos previsores de saída. Os complexos modelos das redes neurais são capturados somente pelos pesos das ligações na rede, que representam equações matemáticas extremamente complexas.

Existem várias tentativas de se aliviar esses problemas básicos provenientes das redes neurais. A metodologia mais simples é olhar de fato as redes neurais e tentar criar explicações plausíveis para os significados dos nós escondidos.

A compreensão do comportamento dos nós escondidos é provavelmente a parte mais difícil das redes neurais. A questão mais simples que pode ser levantada é qual dos muitos previsores é na verdade o mais importante para uma determinada predição. Uma possibilidade seria a análise dos pesos das ligações de um dado previsor para verificação se ele é grande ou pequeno. Se todos os pesos oriundos de um nó de entrada estão próximos de zero para um dado previsor, então, este previsor em particular poderia resultar em um baixo impacto. Porém, se algumas das ligações forem importantes e outras não, este previsor poderá resultar em um grande impacto em determinadas situações e em outras não.

Quando existem camadas escondidas, é extremante complicado se determinar o impacto global de um dado previsor, já que uma ligação forte poderia conectar um nó a uma camada escondida, que possuirá um forte impacto em todos os nós de saída.

Devido a esta complexidade, o impacto de vários previsores é determinado por meio de experimentações iterativas. Estas experimentações podem ser realizadas de duas formas [23]. A primeira, através da construção de uma rede de nós de entrada para o previsor, e verificando-se a precisão da predição para a rede com o previsor. A segunda é através de ligeiras alterações nos valores de um dado previsor, verificando-se posteriormente se o valor de saída é alterado.

As redes neurais artificias são utilizadas em uma grande variedade de aplicações. Elas podem ser usadas em diferentes segmentos de negócios, desde a detecção de fraudes no segmento de cartões de crédito até a predição de riscos de investimentos nos segmentos de mercado de capitais. Estas são aplicações mais atuais, dada a alta probabilidade de retorno financeiro associada a mercados dessa natureza. Porém, a aplicabilidade das redes neurais é mais antiga, e vai desde o uso militar, para a direção automática de veículos não tripulados, até simulações biológicas, como o aprendizado correto da pronúncia de uma língua a partir de um texto escrito.

Capítulo 6

Desenvolvimento de software

Para concretizar este estudo com o pacote de simulação SimNeural, utilizou-se a simulação do modelo de RNA que foi proposta no Capítulo 5. A escolha deste modelo ocorreu após o encontro realizado com o professor Carlos Gonçalves, do Instituto de Ciências Biológicas, e Wilson H. Veneziano, do Departamento de Ciência da Computação, no início do projeto. A escolha pela simulação de RNAs se deu pelo fato de ser um tema de grande interesse das duas áreas.

Ao analisar o pacote de simulação ExtendSim, observam-se algumas características interessantes que se correlacionavam com processos de desenvolvimento de software evolutivo exploratório através de prototipação e com o desenvolvimento de software baseado em componentes.

Neste capítulo são esclarecidos conceitos de engenharia de software e relacioná-los com o desenvolvimento do projeto SimNeural.

6.1 Plataforma

O ExtendSim é um software executado sobre a plataforma Windows da Microsoft. Segundo a empresa Imagine That! Inc., o ExtendSim pode ser executado em versões 32 bits e 64 bits do Windows XP, Vista e Seven, além do MacOSx 10.4. No desenvolvimento deste trabalho, utilizamos um computador com as seguintes configurações durante o desenvolvimento:

- sistema operacional Windows 7 64 bits;
- processador Intel core I7 Q2670m 2.2-2.9 GHz;
- 8 GB RAM DDR 3 1333 MHz;
- 750 GB HD;
- placa de vídeo HD 6990m.

Os testes de sistema foram executados em diversas plataformas de Windows XP e Seven, para testar a estabilidade dos modelos desenvolvidos.

6.1.1 Ferramentas de desenvolvimento

Neste projeto, utilizou-se apenas o ambiente ExtendSim para realizar todo o desenvolvimento do ambiente de simulação de Redes Neurais Artificiais, que foi chamado de Projeto SimNeural. Outras ferramentas utilizadas foram o o Microsoft Word 2010 para criação do manual da aplicação e, por último, utilizamos o *Inno Setup* para gerar o *release* do Projeto SimNeural.

6.2 Processo de software

O processo de software é um arcabouço para as tarefas que são necessárias para construção de softwares de alta qualidade. O processo de software define a abordagem que é adotada quando o software é elaborado [24].

Outra definição encontrada na literatura diz que um processo de software é um conjunto de atividades que leva à produção de um produto de software. Estas atividades poderiam envolver o desenvolvimento de um novo software ou com a ampliação e a modificação dos sistemas existentes [29].

Os dois autores fazem a mesma observação de que os processos de software variam de acordo com o tamanho e complexidade do software que será produzido. Contudo, citam atividades que estão presentes em todos os processos de software, as chamadas atividades guarda-chuva:

- *especificação de software:* a funcionalidade do software e as restrições sobre sua operação devem ser definidas;
- *projeto e implementação de software:* software que atenda à especificação deve ser projetado e desenvolvido;
- *validação de software:* o software deve ser validado para garantir o cumprimento dos requisitos do cliente;
- *evolução do software:* a evolução do software cobre as mudanças que surgem da necessidade do cliente.

6.3 Modelos de processo de software

Os modelos de processo de software são abstrações que podem ser usadas para explicar diferentes abordagens para o desenvolvimento de software, como *frameworks*. Eles podem ser adaptados para criar processos mais específicos, que se adequem melhor ao tamanho e complexidade do software. Sommerville [29] apresentou a seguinte definição para três tipos de modelos:

- *modelo em cascata:* etapas rigidamente separadas e sequenciais, seguindo um fluxo de atividades;
- *modelo evolucionário:* intercala atividades de especificação, desenvolvimento e validação. As atividades são repetidas em ciclos, sendo que o sistema inicial é desenvolvido rapidamente baseado em especificações abstratas e refinado com as entradas do cliente.

- *engenharia de software baseada em componentes:* enfoque na integração de componentes pré-existentes, ao invés de desenvolvê-los completamente.

O ExtendSim com suas bibliotecas de bloco é um ambiente propício ao reuso de componentes. Considerando o tempo para execução do projeto, utilizaram-se, também, aspectos da modelagem evolucionária.

6.4 Detalhando o modelo de desenvolvimento evolucionário

O modelo evolucionário é baseado na ideia de se desenvolver uma implementação de software inicial e expor os resultados aos comentários do usuário para que, com o *feedback* fornecido por ele, a aplicação possa ser refinada até atingir a maturidade do sistema. Como dito anteriormente, as atividades do processo de software são intercaladas à medida que os protótipos do sistema são refinados/desenvolvidos.

Existem dois tipos de desenvolvimento evolucionário [29]:

- desenvolvimento exploratório, no qual o objetivo do processo é trabalhar com o cliente para explorar os requisitos, até a entrega de um sistema final.
- prototipação *Throwaway*, na qual o objetivo do processo é compreender requisitos e permitir uma melhor definição destes. O protótipo é descartado.

No desenvolvimento exploratório, um protótipo do sistema é criado nos estágios iniciais, porém, ao contrário da prototipação *throwaway*, que descarta os protótipos, o protótipo é incrementado à medida que os requisitos são compreendidos. Utilizou-se esta abordagem no ExtendSim para desenvolver o projeto SimNeural.

Uma abordagem evolucionária costuma ser, frequentemente, mais eficaz do que a abordagem em cascata na produção de sistemas que atendam às necessidades imediatas dos clientes. A vantagem, de fato, é que a especificação pode ocorrer de forma incremental, à medida que os usuários compreendem melhor o problema. Contudo, essa abordagem apresenta dois problemas: o processo não é visível (pouca documentação) e o sistemas são, frequentemente, mal estruturados.

Para softwares de pequeno e médio porte, de até 500 mil linhas de código, como é o caso do SimNeural, a abordagem evolucionária seria o melhor método de desenvolvimento, pois os problemas começam a surgir em sistemas de maior porte e mais complexos [29].

O desenvolvimento evolucionário envolve métodos ágeis de desenvolvimento, como o XP, e está incorporado ao *Rational Unified Process*(RUP).

6.4.1 Prototipação

Um protótipo é uma versão inicial de um sistema de software usado para demonstrar conceitos, experimentar interfaces, funcionalidades, etc. Eles permitem aos usuários e desenvolvedores experimentarem como o sistema resolverá o problema [29].

Tradicionalmente, a prototipação tinha a finalidade exclusiva de valiar os requisitos, apenas apoiando o desenvolvimento. A vantagem de seu uso na elucidação dos requisitos

é de que, tendo a possibilidade de visualizar, na prática, o funcionamento/interface do sistema os usuários e a equipe de desenvolvimento podem avaliar erros e mudanças que deverão ser realizadas nos requisitos.

Atualmente, pela demanda do mercado, cada vez maior, o tempo para o desenvolvimento diminuiu e os limites entre a prototipação e o desenvolvimento normal do sistema, muitas vezes, são indefinidos e muitos sistemas são desenvolvidos com uma abordagem evolucionária de protótipos.

O custo inicial de utilização de protótipos é elevado, contudo, o benefício trazido pela redução do retrabalho, a longo prazo, reduz os custos globais do processo de software. O ExtendSim e suas ferramentas de interface com o usuário, permitiram o uso desta abordagem para elucidar requisitos e realizar a evolução de protótipos do sistema, até uma versão final.

Convém observar que a prototipação evolucionária causa problemas em projetos de maior porte a saber:

- pode ser impossível ajustar o protótipo para atender aos requisitos não funcionais (desempenho, confiabilidade, segurança, etc.);
- mudanças rápidas durante o desenvolvimento implicaram em protótipos não documentados, dificultando manutenções futuras;
- mudanças futuras tendem a degradar o sistema;
- padrões de qualidade são, geralmente, relegados a segundo plano no desenvolvimento.

Como o projeto SimNeural não se trata de um projeto de grande porte e tive-se disponibilidade do cliente *on-site*, utilizou-se o modelo de prototipagem evolucionária.

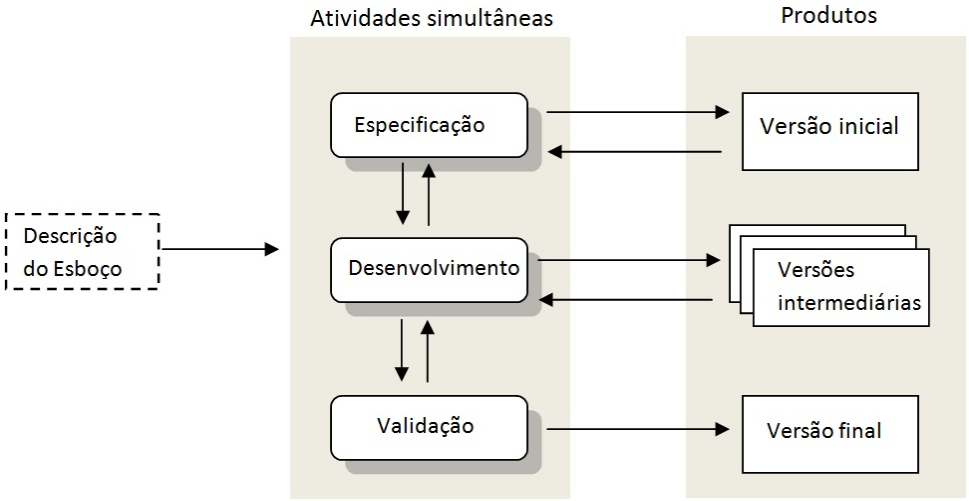


Figura 6.1: Desenvolvimento evolucionário prototipado [29].

6.5 Modelo de desenvolvimento baseado em componentes

A engenharia de software baseada em componentes (*Component-Based Software Enginnering* - CBSE) surgiu no final da década de 1990 como uma abordagem baseada no reuso para desenvolvimento de sistemas de software, motivada pela frustração do fato que somente a orientação a objetos não conduziu ao reuso adequado. Na abordagem evolucionária de desenvolvimento, o reuso é, frequentemente, essencial para o desenvolvimento rápido de sistemas.

A ideia por trás da CBSE é reduzir o acoplamento entre os componentes do sistema, permitindo, de forma mais facilitada, o reuso destes componentes ou mesmo a substituição em manutenções evolutivas. Esta abordagem de desenvolvimento tornou-se importante à medida que a complexidade e tamanho dos sistemas cresceram, e entregar um software melhor e mais rapidamente requer que se reutilize componentes ao invés de realizar novas implementações em cada novo projeto.

Os pontos essenciais da engenharia de software baseada em componentes são [29]:

- componentes independentes completamente especificados por suas interfaces. Deve haver uma clara separação entre a interface do componente e a sua implementação, permitindo a substituição da implementação sem afetar outros componentes que utilizam seus serviços;
- padrões de componentes que facilitam a integração de componentes. Estes padrões podem definir como serão as interfaces dos componentes, de forma a independer de linguagens de programação;
- *Middleware* que fornece apoio de software para integração dos componentes;
- processo de desenvolvimento voltado à CBSE.

Mesmo quando os componentes reusáveis não estão disponíveis, as bases da CBSE são usadas por muitas organizações, por serem princípios seguros de projeto que se apoiam na construção de software compreensível e fácil de manter. A *componentização* de um sistema pode garantir que independência funcional de subsistemas e detalhes de implementação são ocultados, permitindo alterações em um componente sem alterar o funcionamento do sistema.

Assim, embora a CBSE tenha muitas vantagens, ela ainda possui alguns entraves [29]:

- confiabilidade de componentes: existem componentes caixa preta, isto é, quando realizamos reuso de algum componente do qual não conhecemos o código-fonte. Neste caso a confiabilidade fica comprometida;
- certificação de componentes: relacionada estreitamente com a confiabilidade, seria a solução para o primeiro problema, contudo, a indústria não está muito interessada neste assunto;
- previsão de propriedade emergente: ao realizar a integração de componentes é difícil prever como as propriedades emergentes do sistema serão afetadas (desempenho, erros inesperados, etc.).

6.5.1 Processo de desenvolvimento adaptado à CBSE

Conforme pontuado na Seção 6.3, o processo de software deve ser adaptado à CBSE. A abordagem orientada a reuso depende de uma grande base de componentes de software reusáveis e de algum *framework* de integração desses componentes. Estes componentes podem ser sistemas comerciais independentes que fornecem alguma funcionalidade específica.

Nos modelos baseados em CBSE, embora os estágios de especificação e de validação sejam comparáveis a outros modelos de processos de desenvolvimento de software, como o modelo Cascata, os estágios intermediários são diferentes, a saber:

1. Análise de componentes: dada uma especificação de requisitos, é realizada uma busca pelos componentes para implementar essa especificação. Geralmente, não existe uma correspondência exata e os componentes que podem ser usados fornecem apenas parte da funcionalidade necessária.

2. Modificação de requisitos: Analisados os componentes, os requisitos são confrontados para verificar se os componentes encontrados suprem as necessidades do sistema. Caso positivo, parte-se para o projeto de sistema com reuso, caso contrário, podem ser realizadas mudanças nos requisitos para refletir os componentes disponíveis. Quando tais modificações são impossíveis, a atividade de análise de componentes deve ser novamente realizada até esgotar todas as possibilidades.

3. Projeto de sistema com reuso: com requisitos definidos e componentes escolhidos, o *framework* do sistema é projetado ou um *framework* existente é reusado. No caso de falha na etapa de modificação de requisitos, isto é, não encontrar de forma alguma conciliação entre componentes reusáveis disponíveis e requisitos, pode ser necessário construir um software (componente) novo.

4. Desenvolvimento e integração: O software que não pode ser reusado é desenvolvido e os componentes são integrados. A integração do sistema pode ser parte do desenvolvimento.

A Figura 6.2 é ilustrativa do modelo de desenvolvimento baseado em componentes. É possível observar as etapas intermediárias que pertencem a abordagem CBSE, e os processos de especificação de requisitos e de validação de requisitos.

6.6 Processo de desenvolvimento do SimNeural

As características do projeto SimNeural e do pacote de simulação ExtendSim permitiram aplicar alguns dos conceitos de processo de software discutidos nas seções anteriores deste capítulo. Pelo curto espaço de tempo necessário para desenvolver o sistema e pelo tamanho relativamente pequeno do sistema, resolve-se adotar a abordagem de desenvolvimento evolucionário.

Foram cerca de seis reuniões para definição de requisitos do sistema. Entre as reuniões, protótipos (versões intermediárias) foram desenvolvidos e utilizados para validação e incremento de requisitos e software do sistema. Com esta abordagem, criou-se uma interface que é amigável ao usuário, dentro das possibilidades oferecidas pelo pacote ExtendSim.

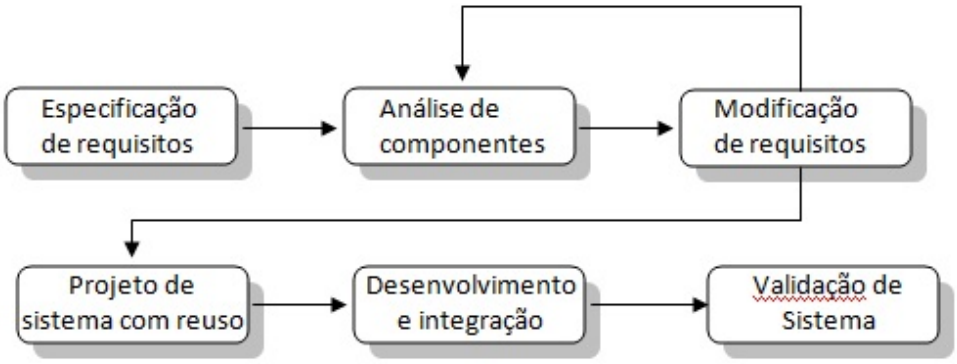


Figura 6.2: Desenvolvimento evolucionário prototipado [29].

Outro processo de desenvolvimento mesclado com o evolucionário foi o de engenharia de software baseada em componentes. Nos estágios iniciais de desenvolvimento, procurou-se utilizar os componentes disponíveis nas bibliotecas distribuídas juntamente com o ExtendSim. Conforme descrito na Seção 4.3.2 do Capítulo 4, as bibliotecas e blocos presentes no ExtendSim o tornam uma ferramenta de desenvolvimento voltada ao reuso de componentes. A geração de gráficos e os botões usados nas telas principais do SimNeural foram todos componentes reaproveitados das bibliotecas *Plotter* e *Utilities*

Além do supracitado, é preciso destacar que grande parte dos requisitos do sistema são inerentes ao modelo que foi utilizado para avaliação do ExtendSim neste projeto: Redes Neurais Artificiais (Capítulo 5).

6.7 Arquitetura de software

O SimNeural é dividido em quatro camadas lógicas, mostradas na (Tabela 6.1).

Tabela 6.1: Camadas do SimNeural.

Camada	**Responsabilidade**
Camada de definição de dados	Responsável pela geração, e armazenamento de dados das RNAS criadas.
Camada de importação e exportação de dados	Responsável por lidar com leitura/escrita de arquivos externos ao SimNeural.
Camada de interface com usuário	Possui todos os controles responsáveis pela alteração da estrutura, da importação e da execução das RNAs.
Camada neural	Definida a estrutura por meio da camada da interface, os neurônios são criados. Os três tipos de neurônios representam a camada neural, responsável pela execução dos cálculos do *backpropagation*.

Existem algumas fronteiras entre as camadas que não estão bem definidas, devido à impossibilidade de se trabalhar com um menor nível de acoplamento, por parte do pacote de simulações ExtendSim. Por exemplo, na camada de Importação, o bloco SalvarComo, responsável por salvar o modelo de trabalho atual no SimNeural, apresenta uma interface

com o usuário, mesmo não estando na camada lógica de interface com o usuário. No próximo capítulo serão especificados os blocos pertencentes a cada camada e sua respectiva função.

A linguagem ModL não é orientada a objetos, o que impossibilitou uso de padrões de projeto que tornariam o reuso de componentes muito facilitado, através do uso de *Design Patterns* como *commander* e *observer*.

Outra característica do Projeto SimNeural é que ele é composto por diversos modelos unidos por um modelo central (Figura 6.3). Essa organização foi escolhida por permitir ao usuário trabalhar com vários modelos ao mesmo tempo sem ter que recriar toda a lógica que permitiria tais ações simultaneamente.

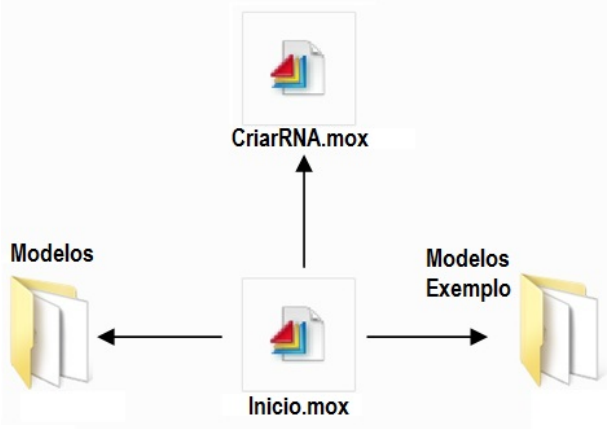


Figura 6.3: Composição do Projeto SimNeural.

6.8 Geração do *release*

Após conclusão do projeto, resultaram diversos arquivos que compõem o SimNeural, vários modelos que são unidos por meio de um modelo central (Figura 6.3), formando algo parecido com uma arquitetura em estrela. Nesse contexto, foi necessário agrupar estes arquivos para permitir uma rápida e fácil instalação pelo usuário. Primeiro pensou-se em utilizar um *self-extract* gerado pelo WinRar, porém, após uma pesquisa pela internet, encontra-se o software *Inno Setup*.

O *Inno Setup*, desenvolvido em Borland Delphi, permite a criação de instaladores para programas em Windows. Ele compacta os arquivos mantendo a estrutura que o programador desejar, bastando a configuração através de um simples e poderoso *script* de instalação. O programa de instalação gerado pelo *Inno Setup* possui visual padrão de programas de instalação Windows disponíveis no mercado e facilita aos usuários que, muitas vezes, já estão acostumados com este padrão.

Capítulo 7

O Projeto SimNeural

7.1 Requisitos técnicos

O projeto SimNeural foi desenvolvido sobre o pacote de simulação ExtendSim e, portanto, necessita que este esteja instalado na máquina do usuário. Além desses requisitos, pode ser citado, ainda, os requisitos de sistema exigidos pelo próprio ExtendSim (Tabela 7.1):

Tabela 7.1: Requerimentos do ExtendSim [14]

	Windows	**Mac**
Processador	Pentium 4 ou superior	Power PC (nativo) ou processador Intel
Sistema operacional	Windows XP, Vista, 7 ou 2000	OSX 10.4-6
Disco rígido	300 MB	300 MB
RAM	512 MB; 2GB recomendado	512 MB; 2GB recomendado
Requisitos adicionais para a versão ExtendSim Suite	100 MB adicionais no HD e um acelerador gráfico de no mínimo 64 MB de RAM	100 MB adicionais no HD e um acelerador gráfico de no mínimo 64 MB de RAM

Neste trabalho de desenvolvimento e testes, executa-se o SimNeural em plataformas XP e 7, 32 e 64 bits, utilizando as mais diversas configurações de hardware. O comportamento do sistema mostrou-se com poucas variações, apenas durante a execução dos treinamentos e validação de dados das RNAs.

7.2 Visão geral do projeto SimNeural

Pensado para uso didático, o Projeto SimNeural tentou aproveitar as ferramentas de interface disponíveis no pacote de simulação ExtendSim. Na verdade, o software é composto de vários modelos interligados: Inicio.mox, CriarRNA.mox, além de modelos que foram usados na validação do SimNeural e são distribuídos como modelos exemplos.

Para o usuário, foi criada um interface amigável que evita o trabalho de abrir modelo por modelo por meio de pesquisas no gerenciador de arquivos do sistema operacional. Na Figura 7.1 pode ser visto os botões que mapeiam e abrem os modelos que compõem o software.

Figura 7.1: Tela inicial do projeto SimNeural.

Funções dos botões na tela principal:

- Criar Rede Neural: abre o modelo CriarRna.mox, que nada mais é que um modelo, aparentemente em branco, pois toda a estrutura necessária para gerar as redes neurais estão lá.
- Abrir Modelo: abre a janela do gerenciador de arquivos diretamente na pasta onde os modelos são salvos pelo SimNeural.
- Exemplos de Modelo: o usuário é levado ao diretório contendo os exemplos que são distribuídos junto com o Projeto SimNeural (detalhes no Item 7.3).
- Ajuda: abre o manual do Projeto SimNeural, elaborado para auxiliar a operação do software.
- Sair: finaliza a execução do SimNeural, juntamente com qualquer modelo que esteja aberto. Os modelos que sofreram alterações solicitam ação do usuário quanto a salvar ou não as mudanças.

7.2.1 Criando uma rede neural

Ao clicar em "Criar Rede Neural" o modelo CriarRna.mox é automaticamente carregado pelo pacote de simulação ExtendSim. O painel de interação com o usuário é muito semelhante ao da tela inicial, pois garante uma padronização da interface com o usuário. No painel exibido na criação de uma nova RNA estarão presentes as seguintes funcionalidades:

- Definir Arquitetura: abre o painel de controle da Rede Neural que será construída. Através desse painel o usuário poderá definir a quantidade de neurônios e forma de treinamento, importar dados, definir pesos, etc., enfim, tudo que for preciso para gerar e operar a RNA pode ser acessado por meio desse painel;
- Salvar Como: funcionalidade usada para salvar o modelo atual no diretório de modelos do Projeto SimNeural;
- Ajuda: abre o manual do Projeto SimNeural.

Conforme explicado na Seção 4.3.5 do Capítulo 4, o ExtendSim possui uma interface própria para que o usuário interaja com os blocos (componentes) do modelo de simulação, que são chamadas de *dialogbox*. Estas caixas de diálogos são facilmente identificadas, pois apresentam muitas semelhanças umas com as outras (observa-se um exemplo na Figura 7.2).

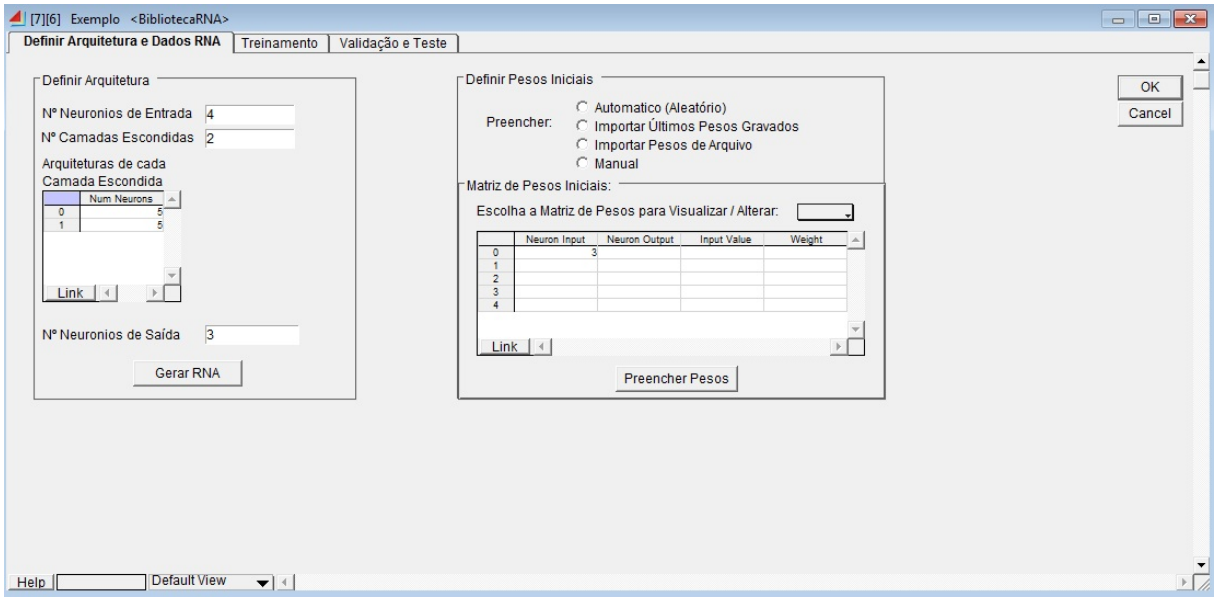


Figura 7.2: Painel central de definição da arquitetura e controle das RNAS.

Pode-se observar que no painel de controle da RNA tem-se três abas: Definir arquitetura e dados RNA, Treinamento e Validação e teste. A seguir será mostrado como seria o fluxo necessário a criação de uma RNA no Projeto SimNeural, os detalhes de cada etapa desta criação e configuração estão melhor descritos no manual do Projeto SimNeural.

Definição da estrutura da rede neural

A primeira aba do painel da Figura 7.2 possui todos os controles que precisam ser usados para o desenho e a definição dos pesos da rede neural. O usuário deverá indicar, nos respectivos campos, a quantidade de camadas e neurônios da RNA e mandar gerá-la. A Figura 7.3 exibe a área de trabalho após geração da RNA.

Observa-se que existem três tipos de neurônios: os da camada de entrada, os da camada escondida e os da camada de saída da RNA. As representações gráficas são distintas para que o usuário consiga identificar facilmente onde começa e termina cada camada da RNA. As conexões entre os neurônios são geradas automaticamente pela lógica interna do SimNeural, utilizando a API do ExtendSim.

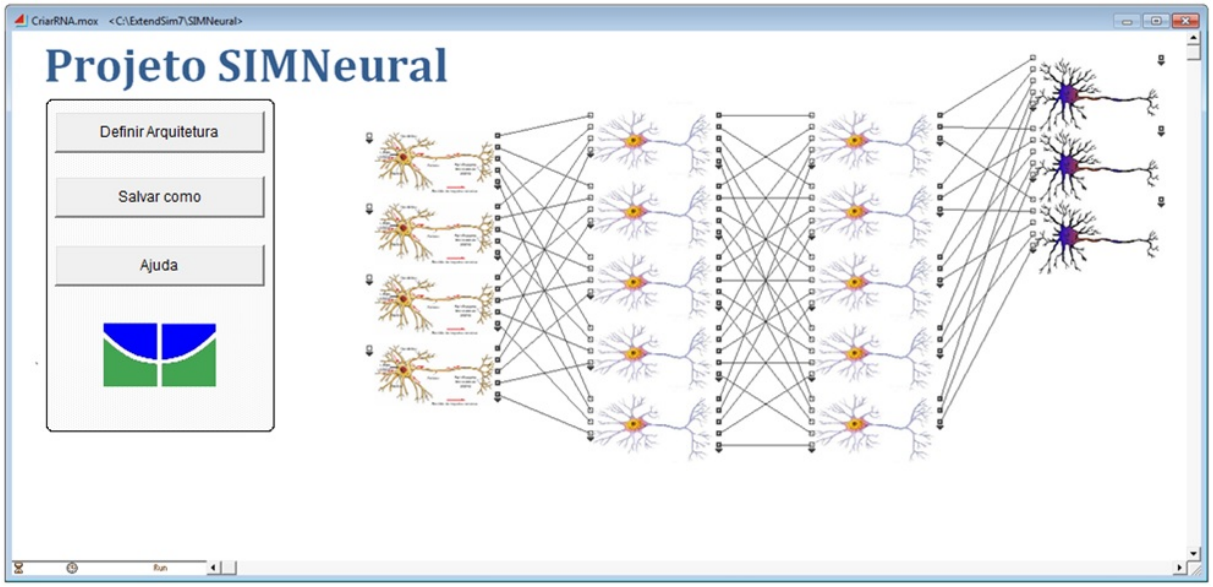


Figura 7.3: Área de trabalho de um novo modelo no SimNeural, após criação de uma rede neural de quatro camadas, sendo uma de entrada, duas intermediárias e uma de saída.

Ao mandar gerar a RNA, será exibida uma caixa de diálogo solicitando informações sobre os dados que serão processados. Estas informações são irrelevantes para os cálculos, contudo são interessantes para visualização do treinamento e resultados obtidos. Com a RNA gerada, o usuário poderá, ainda no painel de definição da estrutura, definir os pesos da RNA. Existem quatro possibilidades para definição dos pesos sinápticos:

- Automático: os pesos serão definidos aleatoriamente com valores entre 0 e 1;
- Importar últimos pesos gravados: ao gravar pesos em um treinamento e executar novas épocas de treinamento, esta opção permite retornar os pesos ao ponto da gravação;
- Importar pesos de Arquivo: permite a importação de pesos gravados a partir de arquivo de dados gravado de um treinamento realizado anteriormente;
- Manual: o usuário pode alterar manualmente cada peso da RNA.

Ao selecionar a opção de importar a partir de arquivo, uma caixa de diálogo, solicitando o arquivo para importação, será exibida. Após a seleção da opção desejada, o usuário deve clicar em "Preencher Pesos" para que todos os pesos sinápticos sejam definidos, conforme a escolha realizada. Com isso a estrutura da RNA estará completa.

7.2.2 Treinando a rede neural

Pode-se dizer que esta é a principal e mais complicada etapa de todo o processo. Terminada a definição da RNA, clicando na aba Treinamento, o usuário é levado à tela da Figura 7.4, onde possui todas as configurações e variáveis possíveis para análise do treinamento que for realizado.

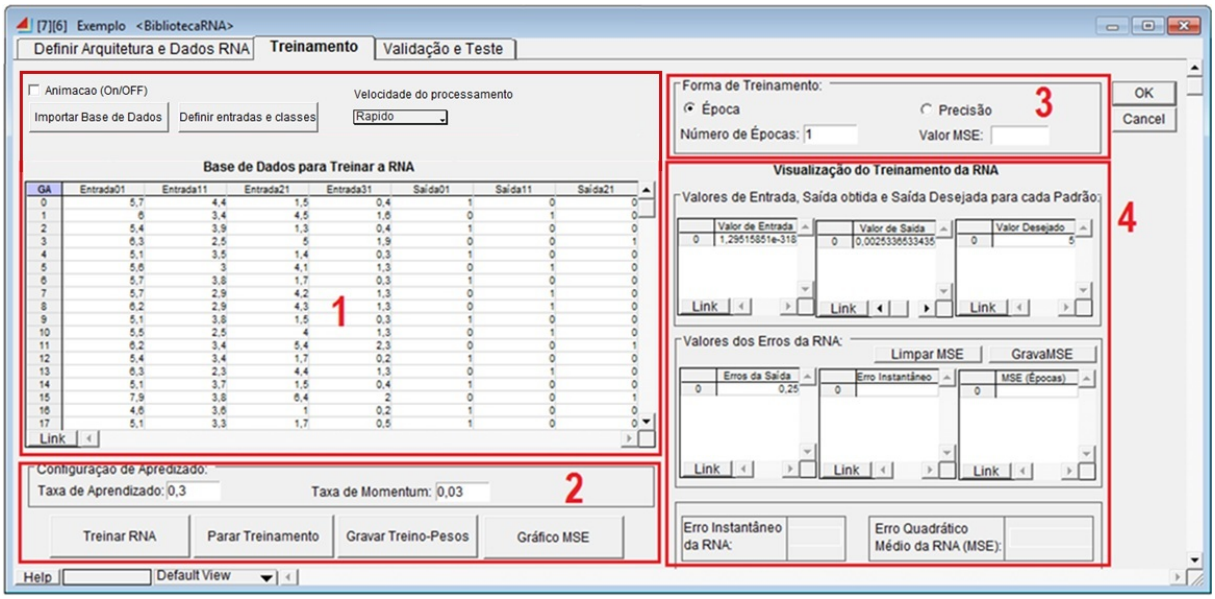


Figura 7.4: Aba treinamento no painel de controle da RNA.

Para fins de melhor entendimento, foi separado a tela em quatro partes que são:

1. dados para o treinamento: aqui o usuário pode visualizar, importar, definir rótulos (definir nomes das colunas nas tabelas), selecionar a velocidade de processamento na execução da RNA;

2. configurações de aprendizado: o usuário define as taxas de aprendizado e momentum do treinamento e ainda dispõe dos botões: Treinar RNA, Parar Treinamento, Gravar Treino-Pesos e Gráfico MSE. As funções de cada um serão explicadas a seguir;

3. forma de Treinamento: é nestes dois campos que o usuário vai definir se o treinamento será por épocas ou por limiar de MSE (até atingir determinado valor de MSE);

4. visualização do treinamento da RNA: seis tabelas que exibem a evolução do treinamento. Em ordem: Valores de entrada, Saída obtida, Saída desejada, Erros de saída, Erro instantâneo e MSE. Existem dois botões que controlam o que será gravado da execução da rede.

Os botões destacados na configuração de aprendizado (marcação 2 na Figura 7.4), realizam as seguintes operações:

- Treinar RNA: inicia a execução do treinamento da RNA usando os parâmetros e a forma de treinamento estabelecidos em 2 e 3. O algoritmo usado no treinamento é o *backpropagation*;
- Parar Treinamento: interrompe o treinamento que estiver em execução (os dados das tabelas não são apagados e o treinamento pode ser continuado depois);
- Gravar Treino-Pesos: armazena todos os pesos de cada conexão da RNA em um arquivo TXT. Permite restaurar os pesos a partir da aba Definir Arquitetura e dados RNA;
- Gráfico MSE: ferramenta que permite ao usuário gerar gráficos a partir de um ou mais MSEs gravados. O MSE (*Medium Square Error* - Erro quadrático médio) é obtido somando os erros instantâneos de todos os elementos do conjunto de teste em uma Época e realizando uma normalização com respeito ao tamanho do conjunto de treino. O MSE representa a função de custo do processo de minimização do erro de aprendizado.

Analisando dados dos treinamentos

No SimNeural, desenvolve-se uma forma dinâmica de acompanhar os treinamentos das RNAs: por meio das tabelas destacadas pelo número quatro da Figura 7.4. Nelas, os valores são alterados a cada apresentação de um vetor de entrada. Chama-se de vetor de entrada o conjunto de dados de entrada e saída desejada. A exceção a esta regra é a última tabela: MSE, na qual os valores são atualizados cada época do treinamento (apresentação de todos os vetores de entrada).

Observa-se que existem dois botões “Limpar MSE” e “Gravar MSE”. O primeiro botão limpa completamente a tabela de MSE atual, já o segundo, permite salvar o MSE do treinamento realizado em um de até cinco “espaços” de memória, para posterior comparação em um gráfico. Tanto o botão “Gravar MSE” quanto “Gráfico MSE” exibem a mesma janela para o usuário (Figura 7.5).

Esta janela permite escolher em qual espaço de memória será armazenado o conjunto de valores MSE do treinamento atual. Clicando em Exibir Gráfico, um gráfico comparando todos os MSEs gravados nos cinco respectivos espaços de memória é exibido para o usuário. O gráfico (Figura 7.6) pode ser alterado através de ferramentas que não serão especificadas aqui, mas sim no manual do Projeto SimNeural, que é instalado junto com o aplicativo.

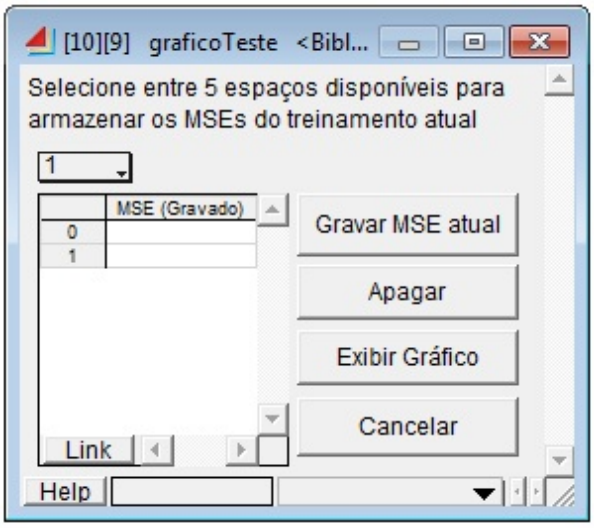


Figura 7.5: Janela de configuração do gráfico e do armazenamento dos MSEs de até cinco treinamentos realizados.

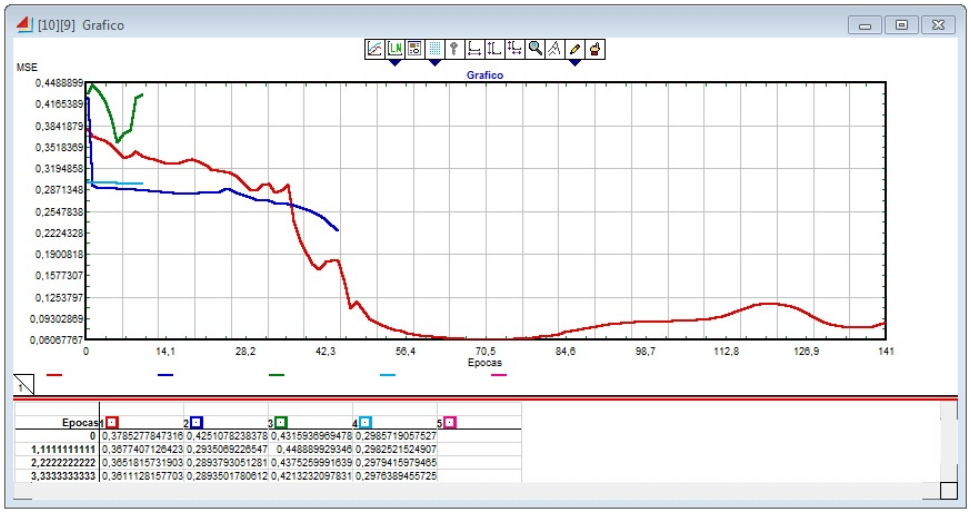


Figura 7.6: Gráfico gerado após salvar os MSEs de quatro treinamentos em espaços diferentes de memória.

Utilizando o atalho de teclado Ctrl+C é possível copiar uma imagem do gráfico para a área de transferência para uso em outros aplicativos, como editores de texto ou planilhas.

7.2.3 Validação e teste

Com a rede treinada, é possível testar o resultado obtido no treinamento com outro conjunto de dados. Esta última etapa visa certificar que a rede de fato aprendeu e não, simplesmente, memorizou os dados apresentados durante o treinamento. A ideia é que os vetores de entrada apresentados aqui sejam diferentes dos apresentados no treinamento, para testar a capacidade de generalização da RNA.

Por exemplo, se houver 150 amostras, apresentar 50 amostras no treinamento e 100 na etapa de validação e testes é uma boa opção. O número de amostras para realizar o

treinamento varia, pois é definido empiricamente. Um conjunto menor tende a diminuir a velocidade do aprendizado, mas pode aumentar a capacidade de generalização (capacidade classificar padrões que não foram utilizados durante o treinamento).

As deficiências na capacidade de generalização podem ser atribuídas aos seguintes fatores:

- *Overfitting*: ocorre quando a RNA memoriza os dados de treinamento, em vez de extrair as características gerais que permitem a generalização. Geralmente é causado por um número muito grande de camadas escondidas ou de neurônios nestas;
- *Underfitting*: A utilização de um número de neurônios inferior ao número necessário pode fazer com que a rede gaste muito tempo para aprender ou até mesmo que ela não consiga convergir ou generalizar demais padrões de entrada.

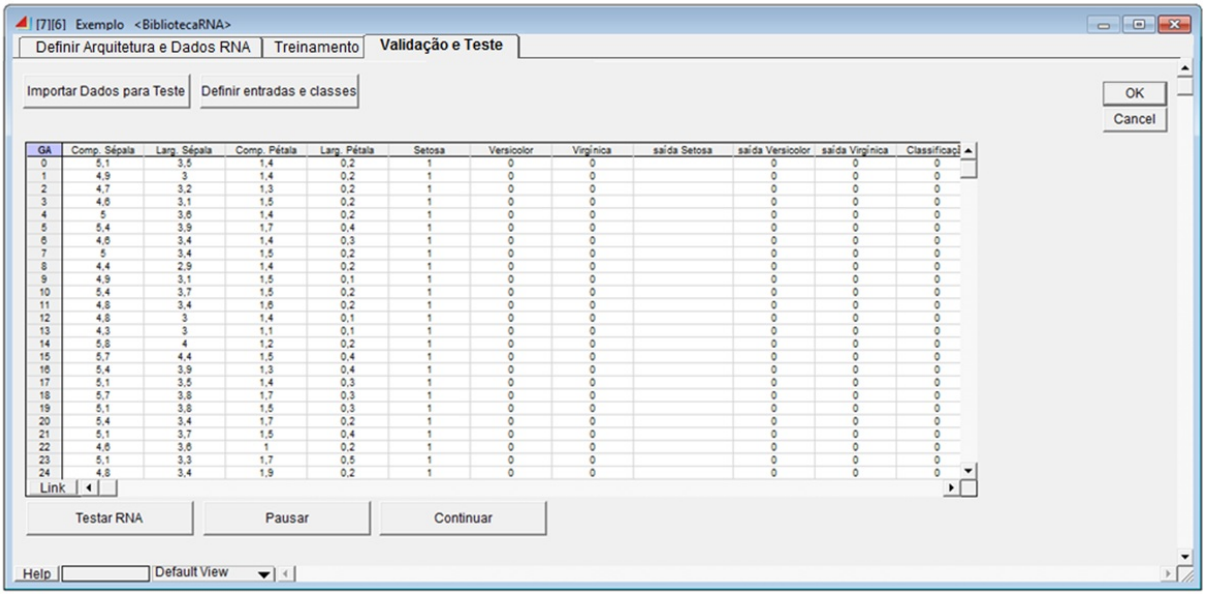


Figura 7.7: Painel de validação e teste.

O painel de validação e teste (Figura 7.7) permite a importação de dados para teste (semelhante a importação dos dados do painel de treinamento), definição dos rótulos das colunas da tabela e das classes que serão identificadas. Além disso, abaixo da planilha, existem os controles que permitem iniciar o teste ("Testar RNA"), pausá-lo ou continuá-lo. A tabela é gerada na importação e de acordo com a quantidade de neurônios da rede (acrescentando as colunas que receberão os dados da execução da RNA). Maiores detalhes podem ser vistos no manual do Projeto SimNeural.

7.3 Exemplos de modelos

O SimNeural possui três exemplos de RNAs incluídos na instalação, são estes:

- Flores de íris: neste modelo, o usuário tem uma rede que classifica três espécies de flores de Íris (setosa, versicolor e virgínica);

- RNA passo a passo: modelo utilizado na aprendizagem do algoritmo *backpropagation*;
- Câncer: a RNA deste modelo classifica as amostras de tumor em benigno ou maligno, de acordo com nove características distintas.

7.3.1 Flores de íris

O modelo flor de íris apresenta algumas mudanças em relação aos modelos que podem ser criados a partir do botão Criar Rede Neural (Figura 7.1). Nele as colunas já estarão devidamente rotuladas e a rede treinada para reconhecer padrões da base dados de flores fornecidas junto com o modelo. Outra peculiaridade, para ajudar no uso didático do modelo é que os usuários podem acrescentar amostras "visualmente" em vez de importação de arquivos de dados para validação e teste.

A RNA, neste modelo, é fixa em quatro neurônios de entrada, pois as flores são classificadas de acordo com o comprimento da pétala, com a largura da pétala, com o comprimento da sépala e com a largura da sépala. Além disso, o número de neurônios de saída também é fixo em três neurônios, pois são apenas três classes possíveis: setosa, versicolor e virgínica. Portanto, sobram apenas as camadas escondidas, que permitem a variação deste modelo, de fins estritamente didáticos.

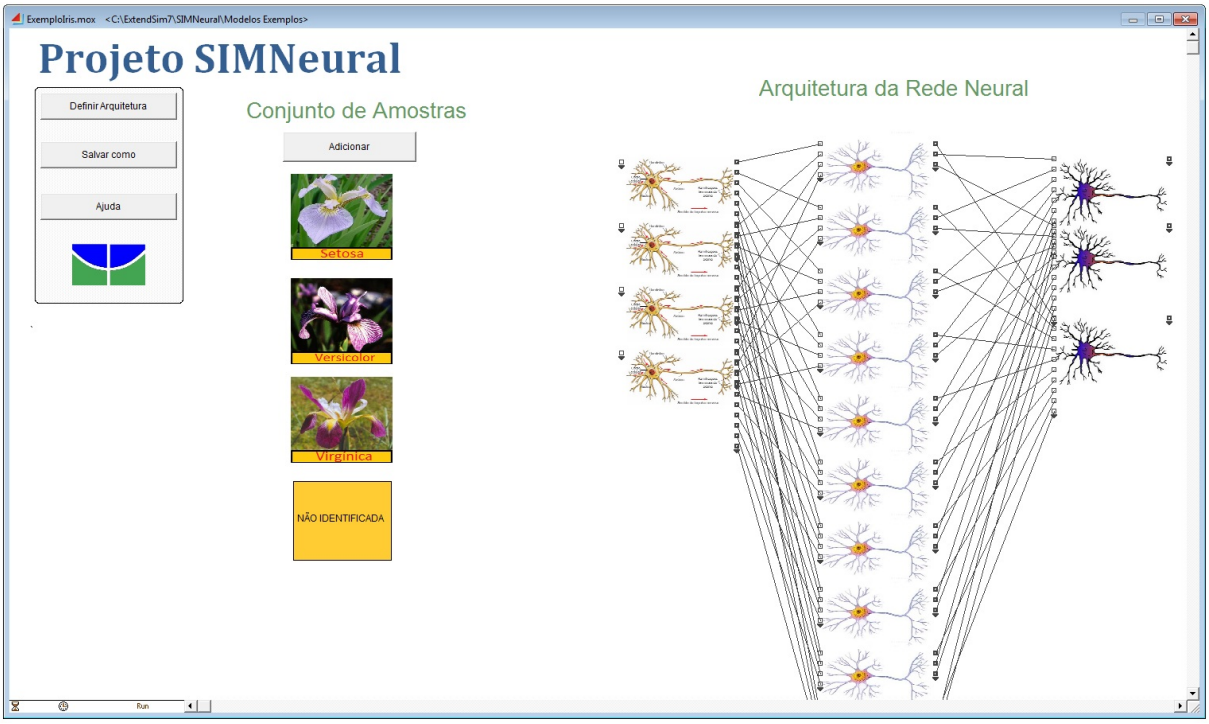


Figura 7.8: Área de trabalho do modelo exemplo flores de íris.

Um duplo clique sobre a amostra (Figura 7.8) permite inserir/alterar os dados dela e realizar a identificação em uma das três espécies disponíveis. Ao realizar a identificação, uma imagem representando a espécie identificada é colocada na amostra. A identificação é realizada com base nos pesos atuais da RNA e pode ser alterada através de novos treinamentos.

7.3.2 RNA passo a passo

O modelo RNA passo a passo foi pensado a partir de um dos primeiros protótipos do sistema, quando ainda não era dominado o algoritmo *backpropagation*. Consiste em um modelo focado exclusivamente na execução passo a passo do algoritmo, sendo inútil para uso real em classificação de algum tipo de padrão, pois não trata conjuntos de dados.

Nesse modelo, tudo se resume a um único painel acessado a partir de um botão chamado "RNA Passo a Passo". Observa-se na Figura 7.9 que todos os passos responsáveis pela execução de um treinamento de RNA estão contemplados, os quais são:

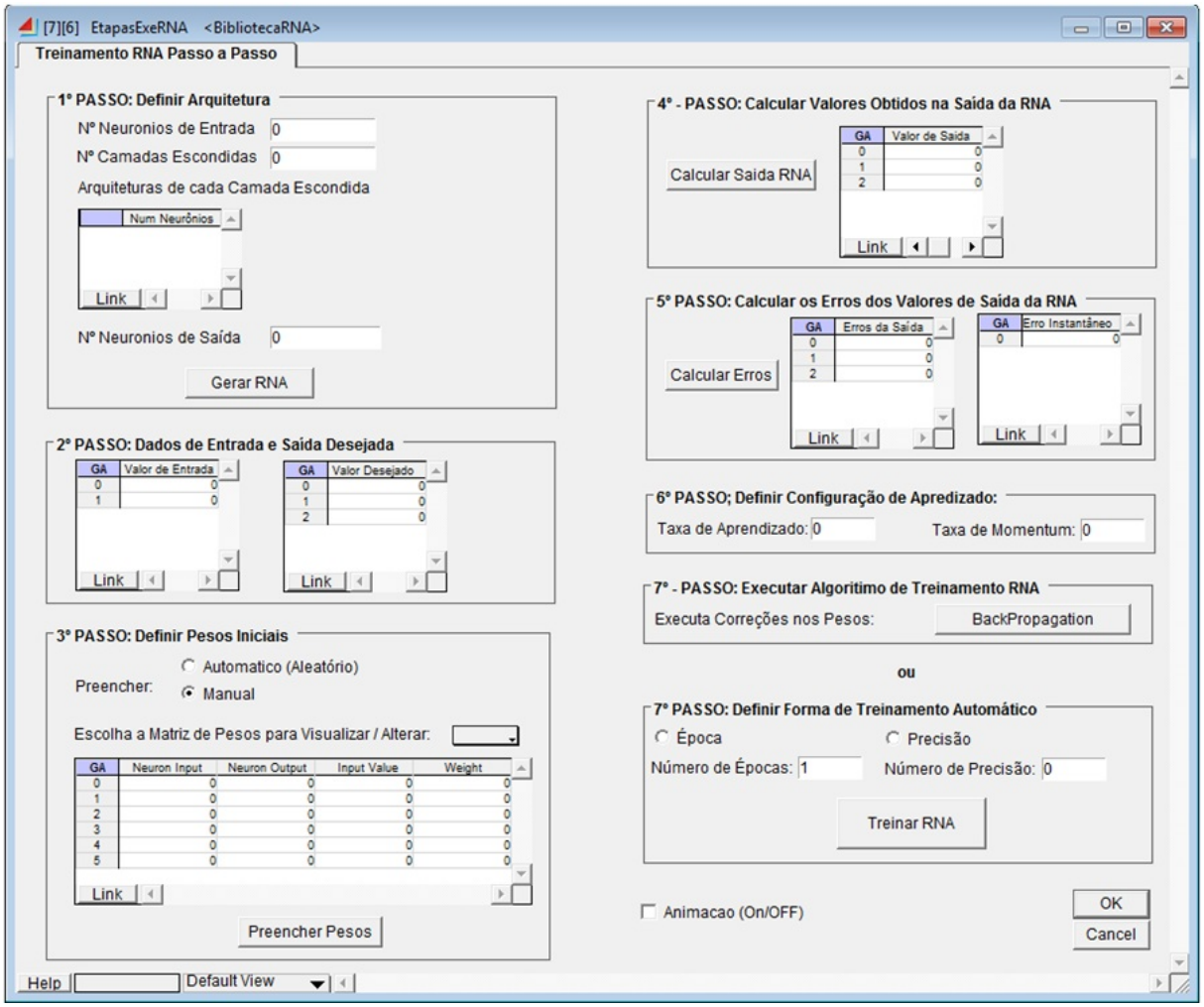


Figura 7.9: Painel de execução do algoritmo passo a passo no modelo RNA passo_a_passo.

1. Definir a arquitetura da rede: o usuário deverá definir o número de camadas da rede e seus respectivos neurônios em um painel semelhante ao da Figura 7.2;
2. Dados de entrada e saída desejada: Aqui, diferentemente da importação de dados realizada no exemplo das flores que utilizamos para demonstrar a criação e treinamento de uma RNA, os valores que serão usados para treino e teste desta RNA são inseridos pelo próprio usuário de acordo com a estrutura da RNA;

3. Definir pesos iniciais: nesta etapa serão preenchidos os pesos com duas possibilidades: manual e aleatório, de forma semelhante com a etapa de preenchimento de pesos do nosso exemplo das flores de íris;

4. Calcular valores obtidos na saída da RNA: os dados de entrada definidos no segundo passo são apresentados à rede para cálculo através do passo *forward* do *backpropagation*.

5. Calcular os erros dos valores obtidos pela RNA: os valores de saída da RNA são comparados com os valores de saída desejados definidos no segundo passo e os valores são exibidos para o usuário;

6. Definir a configuração de aprendizado: o usuário deverá fixar as taxas de aprendizado e de momentum para o cálculo de ajuste de pesos; e

7. Este passo é subdividido em:

   a) Executar ajuste dos pesos manualmente: clicando no botão "Executar", os pesos da rede neural serão ajustados para que o quarto passo possa ser executado novamente de forma a se compararem os resultados;

   b) Executar ajuste dos pesos automaticamente: o usuário irá definir o número de épocas que serão processadas ou definirá à medida de precisão que pretende alcançar, para, depois, clicar em "Executar, fazendo com que a rede ajuste os pesos até atingir a condição definida.

7.3.3 RNA para classificação de tumores

O último modelo exemplo distribuído com o Projeto SimNeural é o do modelo que chamamos de Câncer, juntamente com o modelo de flores de íris, utilizam dados de um repositório de dados para aprendizado de máquina da universidade de *Wisconsin*, EUA. O modelo foi criado seguindo os padrões de todos os modelos que podem ser desenvolvidos no Projeto SimNeural (exceto os últimos dois modelos exemplos citados acima).

A rede foi modelada com nove neurônios na camada de entrada, apenas uma camada escondida contendo dez neurônios e uma camada de saída com dois neurônios (classifica em benigno e maligno), sendo que esta estrutura pode ser alterada inteiramente pelo usuário. A RNA já vem treinada com uma taxa de acerto superior a 90%, mas novos treinamentos podem ser executados.

7.4 Manual do Projeto SimNeural

Durante o desenvolvimento deste projeto, dada a complexidade do software, surgiu a necessidade de que fosse desenvolvido um manual de uso do SimNeural. No manual, não foram abordados detalhes sobre o pacote de simulação ExtendSim, mas sim conceitos fundamentais de Redes Neurais Artificiais e detalhes acerca do uso do SimNeural e todas as etapas e funcionalidades presentes no software.

Quanto aos conceitos de RNAs, não foi apresentado o algoritmo completo do *backpropagation*, para não tornar o manual cansativo, mas apenas seus conceitos principais, de forma que o usuário saiba o que está por trás do aprendizado das redes neurais.

O manual foi estruturado em oito grandes partes: Introdução, Objetivos do Projeto SimNeural, Instalação, Conceitos básicos sobre redes neurais, Ambiente do projeto SimNeural, Ambiente do ExtendSim, Considerações Finais e Glossário; de modo que o usuário tenha total controle sobre seu conteúdo e consiga achar facilmente a informação que precisar, pois a API do ExtendSim não permitiu inserir uma ajuda *on-line* (i.e *tool tips*) para o usuário.

7.5 Sugestões de aplicação do Projeto SimNeural

Avaliando as características do Projeto SimNeural, foram elaboradas algumas sugestões de uso para o SimNeural:

- Iniciação científica em ensino médio: o software pode servir para despertar o "querer pesquisar" em jovens alunos do ensino médio, através do uso de métodos de pesquisa utilizando os exemplos do SimNeural ou modelos criados pelo professor;
- Uso em aulas de biologia: o poder de classificação das RNAs pode ser usado em aulas de taxonomia em biologia. A ferramenta pode ser utilizada até mesmo em atividades de educação a distância (EAD);
- Utilização em iniciação em Redes Neurais: o exemplo RNA passo a passo e mesmo os outros exemplos podem ser úteis em aulas para alunos iniciantes em inteligência artificial;
- O professor deve assistir os alunos durante o aprendizado na operação do SimNeural, para que os conceitos possam ser corretamente aplicados na ferramenta;
- Novos modelos podem ser produzidos pelo professor e pela turma, de forma que as aulas não fiquem limitadas ao simples uso dos exemplos de modelos distribuídos juntamente com o SimNeural.

Destaca-se que todas as sugestões de uso são permeadas por explicação de conceitos de Inteligência Artificial e Neurobiologia, e todas as práticas podem ser precedidas de aulas teóricas sobre os assuntos.

Capítulo 8

Validação do SimNeural

Este capítulo apresenta a validação da implementação SimNeural como um simulador e modelador de RNA. Primeiramente, são feitas análises utilizando bases de dados reais, a fim de verificar o comportamento de uma RNA, modelada pelo SimNeural, em diversas situações. Em seguida, é apresentada a visão pedagógica do software, mostrando suas funcionalidades como uma ferramenta para ensino e aprendizado de redes neurais artificiais. Por fim, são apresentados exemplos de uso do software para desenvolvimento e simulação de modelos para classificação de padrões.

8.1 Validação do SimNeural como simulador de modelos para classificação de padrões usando RNAs

Para validar o reconhecimento de padrões com o uso do SimNeural, foram implementadas redes neuronais artificiais do tipo perceptrons de múltiplas camadas treinadas com o algoritmo de retropropagação do erro (*backpropagation*). Com o intuito de avaliar o desempenho das redes neuronais implementada em algumas situações práticas, foram escolhidos alguns problemas de classificação bem distintos, com a utilização de bases de dados reais. As bases de dados a seguir foram modeladas no processo de validação.

8.1.1 Identificação das flores de íris

A Íris é uma planta da família das Iridáceas, sendo que existem mais de 1000 espécies em todo o mundo. Existem três espécies deste tipo de planta que são muito utilizadas para o estudo de classificação de padrões: setosa, versicolor e virgínica. O conjunto de dados das flores Íris foi popularizado por Fisher [7], que fez observações em 50 exemplares de cada uma destas três espécies, utilizando como parâmetros de medição o comprimento e largura da pétala e comprimento e largura da sépala de cada flor. A partir destas características, é possível avaliar se uma determinada flor é da espécie setosa, virgínica ou versicolor.

Para validar este exemplo de modelo usaremos a base de dados mantida pela University of California - Irvine (http://archive.ics.uci.edu/ml/datasets/Iris). Esta base de dados foi disponibilizada por FRANK e ASUNCION [9] para fins acadêmicos.

O banco de dados é composto por 150 amostras, sendo 50 de cada espécie. Cada uma destas amostras possui 5 atributos, sendo 4 deles características numéricas (comprimento da sépala, largura da sépala, comprimento da pétala e largura da pétala de cada flor) e um sendo a classificação com base nos quatro primeiros atributos. A vírgula (,) é o separador de cada atributo, conforme mostrado na Figura 8.1.

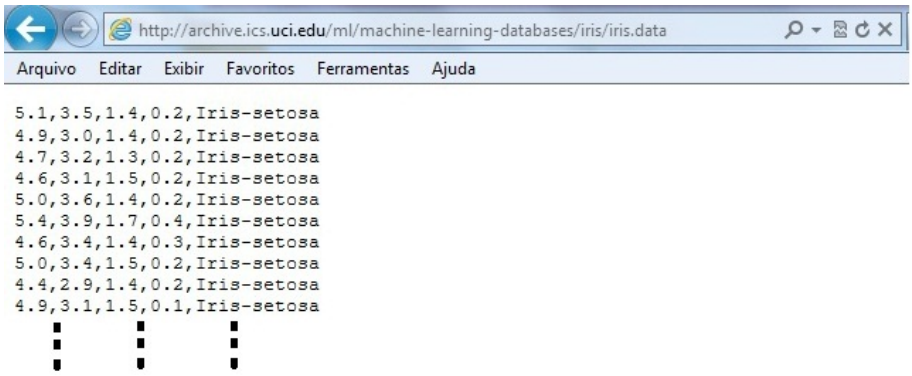


Figura 8.1: Dados retirados do repositório disponibilizado por FRANK e ASUNCIO [9].

O primeiro passo para validar foi definir a arquitetura e os pesos da rede neural, através da aba "Definir Arquitetura e Dados RNA". A arquitetura escolhida foi uma RNA com 1 camada oculta de 10 neurônios e como padrão, deste modelo, a camada de entrada tem 4 e a de saída 3 neurônios. O número de neurônios da camada de entrada da rede (número de entradas) e o número de neurônios da camada de saída (número de classes) são parâmetros configurados automaticamente e fixos, ao contrário do número de camadas e neurônios escondidos, que podem ser alterados livremente. Os pesos iniciais da RNA foram preenchidos com a opção automáticos (que gera um novo conjunto de pesos aleatórios).

Os próximos passos foram o treinamento e o teste da RNA através das abas "Treinamento" e "Validação e Teste". Fez-se a importação das amostras para treinamento e teste, além da configuração dos parâmetros do processo de aprendizagem da RNA.

Para o treinamento e teste de validação que descreveremos abaixo foram usados as 10 primeiras amostras de cada espécie do banco de dados original, formando uma base de dados para treinamento da RNA com 30 amostras dispostas aleatoriamente, conforme apresentado na Figura 8.2. E portanto, sobraram 40 amostras de cada espécie de Íris para formar o conjunto de dados para teste, que será usado para verificar a capacidade de classificação da RNA depois de treinada.

Os parâmetros da taxa de aprendizado(η) e da taxa de momento (α) da RNA foram configurados com os valores respectivamente iguais a 0,2 e 0,01, que, segundo Silva [28], em seus estudos, obteve um treinamento mais rápido e sem muitas oscilações, para RNAs com apenas uma camada interna.

Utilizando a base de dados, as configurações de arquitetura e a taxas de aprendizagem e momentum descritas acima, a RNA apresentou o comportamento de Aprendizagem mostrado na Figura 8.3 pela linha vermelha. Nota-se que depois de 2000 épocas de treinamento o MSE é igual a 0,0895787. O gráfico mostrado nesta figura possibilita um acompanhamento detalhado da variação do erro quadrático médio(MSE) no decorrer das iterações (épocas), ou seja, à medida que a RNA é treinada os erros das 3 saídas da rede

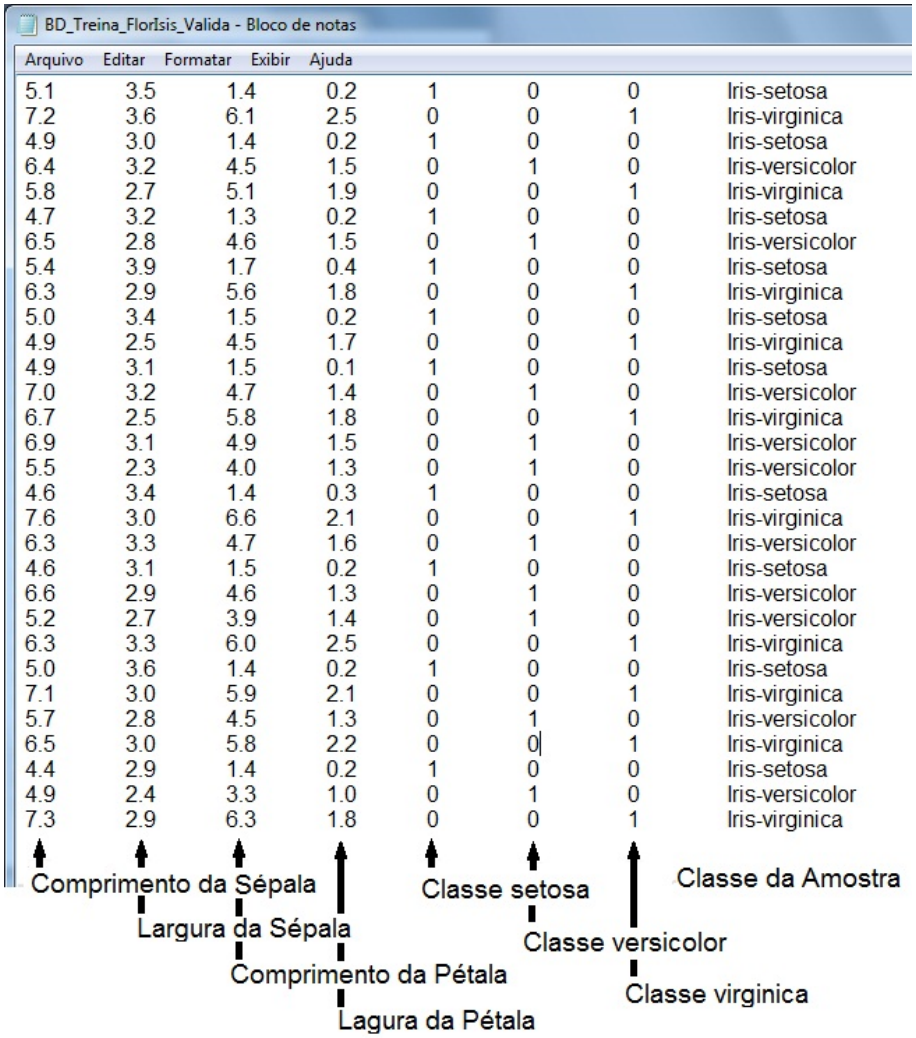


Figura 8.2: Base de dados das flores íris no formato de importação pelo SimNeural para treinamento da RNA.

(valor obtido menos valor desejado) diminuem, mostrando que a rede está ajustando os pesos (aprendendo) para alcançar o valor real.

Uma característica interessante do SimNeural, também disponível no modelo ExemploIris, para uso pedagógico, é que este gráfico permite a comparação de vários treinamentos realizados. É possível, por exemplo, treinar a rede configurada inicialmente com a taxa de aprendizado ($\eta = 0,2$) e taxa de momento ($\alpha = 0,01$), mostrada na Figura 8.3 pela linha vermelha e comparar o desempenho com a rede configurada para a taxa de aprendizado ($\eta = 0,5$) e taxa de momento ($\alpha = 0,3$), representada pela linha azul, em uma mesma base de dados. Assim, é posível analisar e verificar o funcionamento e desempenho da RNA com três configurações distintas, conforme a Tabela 8.1.

Para compreender o comportamento e avaliar o desempenho de uma RNA é importante fazer as seguintes análises [28]:

1. Observar a divisão dos conjuntos de treinamento e de teste, pois a escolha de uma quantidade de elementos muito grande para o conjunto de treinamento (mais de 80%) pode dificultar uma avaliação do desempenho da rede (generalização), já que

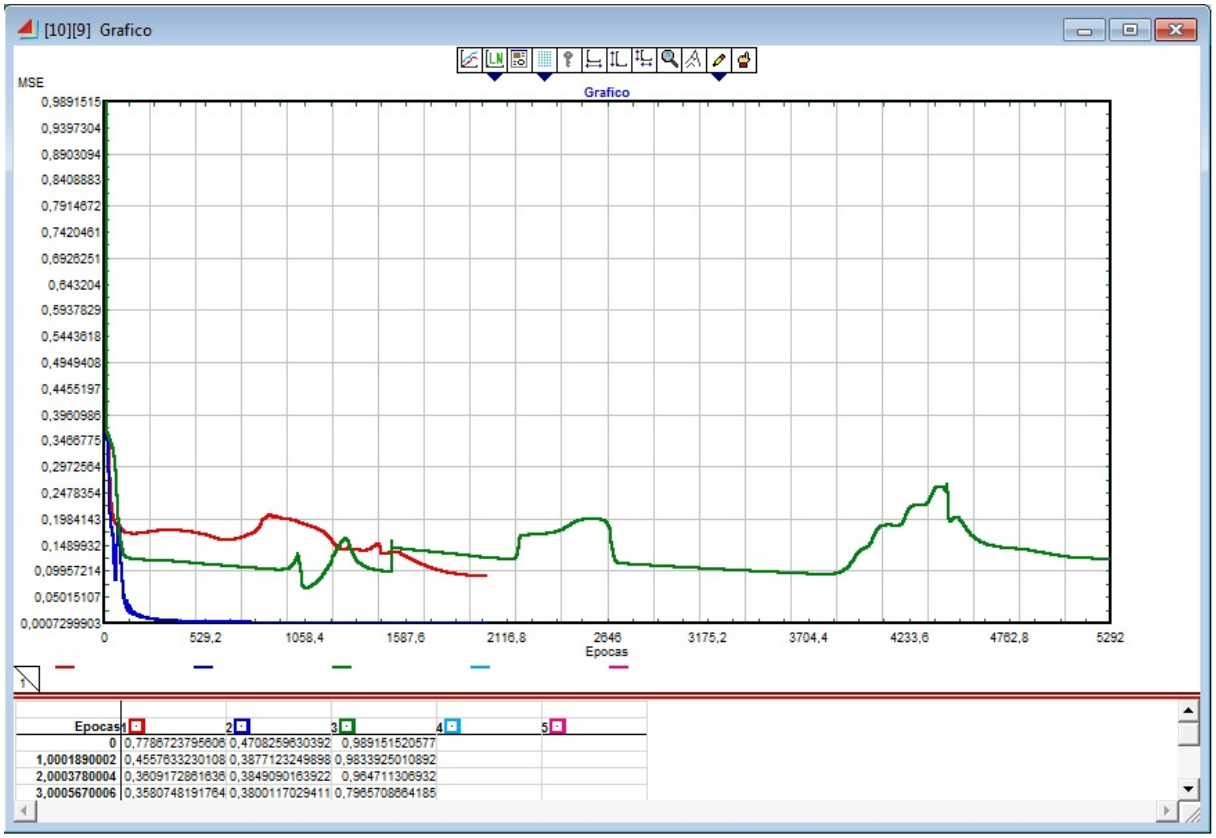


Figura 8.3: Gráfico do MSE no treinamento (curva de aprendizagem).

poucos dados serão destinados ao conjunto de teste. Portanto, necessita testar, para uma mesma configuração de rede, conjuntos de treinamento e de teste com quantidade de elementos diferentes. E analisar a tendência da rede quando aumentamos ou diminuimos o conjunto de treinamento, ou seja, como se comporta o erro quadrático médio (aumenta ou diminui).

2. Verificar as taxas de aprendizado e de momento. Para uma mesma configuração de rede e conjuntos de treinamento e de teste, devem ser analisadas algumas combinações dessas duas taxas. De acordo com os valores escolhidos para essas taxas pode-se ter grandes diferenças no desempenho da rede.

3. A configuração da arquitetura da rede. A escolha do número de camadas internas e o número de neurônios influenciam na capacidade de aprendizagem e desempenho de uma RNA.

A partir de todas estas análises, é possível avaliar uma boa configuração de rede para o problema da classificação das flores Íris ou outro problema qualquer. Obviamente, não existe uma fórmula para calcular exatamente a quantidade de neurônios da rede, as taxas de aprendizado e de momento, o número de épocas de treinamento etc. Isto varia de acordo com cada problema e esta determinação de valores só é possível por meio de simulações.

Tabela 8.1: Configurações da RNA para treinamento e teste do modelo flores de íris.

Parâmetro	**RNA-1**	**RNA-2**	**RNA-3**
Nº Neurônios Entrada	4	4	4
Nº Neurônios Saída	3	3	3
Nº Camadas Ocultas	1	1	1
N° Neurônios Camadas Ocultas	10	10	20
N° Amostras para Treinamento	30	30	12
Taxa de Aprendizagem	0,2	0,5	Variável
Taxa de Momento	0,01	0,3	Variável
N° Amostras para Teste	120	120	120
Percentual de acerto	82%	89%	66%
Linha no Gráfico (Figura 8.3)	Vermelha	Azul	Verde

Para verificar o potencial de classificação da rede neural simulada no modelo ExemploIris, utiliza-se a RNA com a configuração 2, mostrada na Tabela 8.1. Essa configuração obteve um melhor desempenho na aprendizagem, com relação à configuração 1, chegando a um erro quadrático médio aproximadamente igual a 0,00072961 para o conjunto de treinamento após 2000 interações. A base de dados para teste usada contém as 40 amostras de cada espécie de Íris não usada no treinamento. A classificação atribuída ao conjunto de testes pela a rede chegou a um erro de aproximadamente 0,1833, resultando em apenas 22 classificações errôneas (22 espécies virgínicas classificadas como versicolores). Desta forma, o percentual de acerto da rede para as espécies setosa, versicolor e virgínica foi de, respectivamente, 100%, 100% e 45%, gerando um percentual médio igual a 81,67%.

Apesar da configuração 2 ter se mostrado eficiente, ainda estava distante o objetivo para este modelo, em que desejava-se obter uma taxa de acerto superior à 90% para o grupo de 120 amostras de teste. Para atingir tal meta, resolve-se utilizar uma última abordagem, diminuindo o conjunto de dados de treinamento, para doze amostras, sendo três de cada espécie; aumentando o número de neurônios da camada oculta para dez; e alterando as taxas de aprendizado e momentum durante o decorrer do treinamento, conforme pode ser visto na configuração 3 da Tabela 8.1.

Com o número reduzido de amostras, o processamento do treinamento ficou mais rápido, permitindo um número maior de iterações. Primeiro tenta-se um treinamento durante 15.463 épocas com taxas de aprendizagem e de momentum, respectivamente, de 0,3 e 0,03. O MSE ficou alto e o número de acertos não passou de 80%. Resolve-se tentar diminuir a taxa de aprendizado para 0,01 e deixamos executar um treinamento por 16.260 épocas, sem efeitos satisfatórios. Por último, realiza-se um teste variando as taxas. Nesse caso, começa-se com taxas nos valores de 0,25 e 0,3 durante 1.032 épocas, onde foi obtido um MSE igual a 0,089916613 e a partir da época 1.032, reduz-se as taxas de aprendizado e momentum para 0,08 e 0,03, respectivamente, pois o MSE estava baixando lentamente. Com a alteração, deixa-se a rede executar até a época de número 5.292, onde finalmente consegui-se atingir uma taxa de acerto superior a 92% na identificação das 120 amostras de teste, apesar de um MSE final de 0,12194521.

Após os experimentos com flores de íris, conclui-se que o MSE é apenas um dos indicadores de desempenho, sendo que a fase de validação com os dados de teste é a que realmente deve ser realizada para validar o treinamento. Além disso, ver-se que, para o problema proposto, 20 neurônios e uma variação de taxa de aprendizado durante o treinamento se mostrou uma abordagem mais eficaz.

8.1.2 Modelo para classificação de tumores de mama

O problema da classificação dos tumores de mama, disponibilizado na base de dados [9], foi testado em quatro configurações de RNAs distintas. O objetivo da rede neste problema é tentar classificar um padrão de entrada que contém 9 características de um paciente (valores inteiros entre 1 e 10, que representam características específicas das células) em duas classes, uma indicando que o tumor é maligno ou é benigno.

Esta base de dados é formada por 683 padrões, que formam um conjunto com classes não linearmente separáveis. Para a compreensão do comportamento e avaliação do desempenho de uma RNA na resolução do problema, foram analisadas quatro configurações de rede e treinamento diferentes, conforme descritos na Tabela 8.2.

Tabela 8.2: Configurações de RNA para treinamento e teste de classificação do tumor de mama.

Parâmetro	**RNA - 1**	**RNA - 2**	**RNA - 3**	**RNA - 4**
Nº Neurônios Entrada	9	9	9	9
Nº Neurônios Saída	2	2	2	2
Nº Camadas Ocultas	1	1	2	1
N° Neurônios Camadas Ocultas	20	20	5 e 5	10
N° Amostras para Treinamento	50	50	50	50
Taxa de Aprendizagem	0,5	0,25	0,5	0,25
Taxa de Momento	0,3	0,7	0,3	0,3
N° Amostras para Teste	683	683	683	683
Percentual de acerto	82%	89%	66%	91%
Gráfico de MSE - Linhas	Vermelha	Azul	Verde	Azul Claro

As RNA's, descritas na Tabela 8.2, apresentaram o gráfico de Aprendizagem mostrado na Figura 8.4 pela linha vermelha (RNA-1), azul (RNA-2) e, verde (RNA-3) e ciano (RNA-4), onde há a relação dos MSEs ao longo das épocas de treinamento.

A rede RNA - 1 que foi configurada com uma camada interna, com 20 neurônios, e treinada durante 1000 épocas, com a utilização de uma taxa de aprendizado igual a 0,5 e uma constante de momento igual a 0,3, obteve um erro quadrático médio aproximadamente igual a 0,0687806 para o conjunto de treinamento. O percentual de acertos obtidos foi de 72% para o conjunto de 414 amostras do tumor benigno e 99% para as 219 amostras do maligno, representando 117 classificações errôneas, de um total de 633 padrões do conjunto de teste.

A rede RNA - 2 que foi configurada com uma camada interna, com 20 neurônios, e treinada durante 1000 épocas, com a utilização de uma taxa de aprendizado igual a 0,25

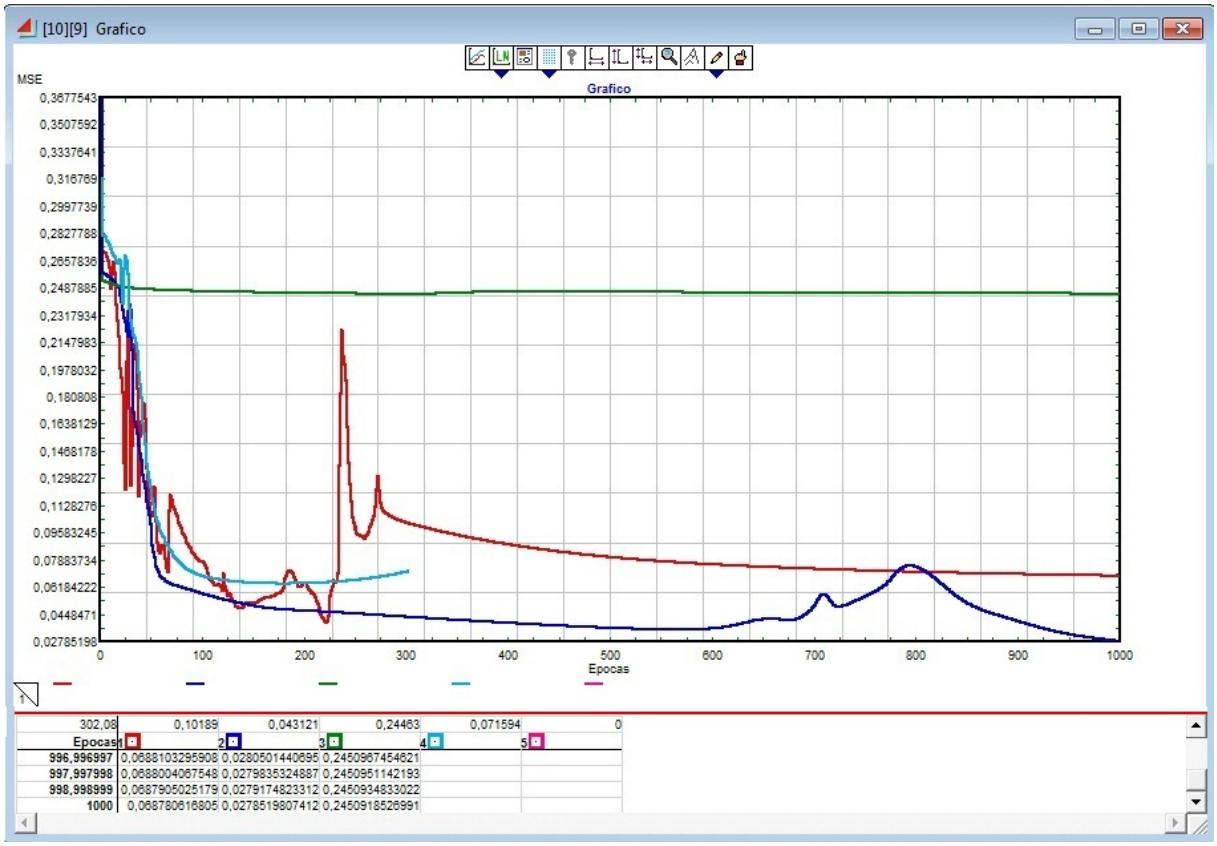


Figura 8.4: Gráfico do MSE no treinamento (curva de aprendizagem) do problema de classificação do tumor de mama.

e uma constante de momento igual a 0,7, obteve um erro quadrático médio aproximadamente igual a 0,027852 para o conjunto de treinamento. O percentual de acertos obtidos foi de 91% para o conjunto de 414 amostras do tumor benigno e 87% para as 219 amostras do maligno, representando 67 classificações errôneas, de um total de 633 padrões do conjunto de teste.

A rede RNA - 3 que foi configurada com 2 camadas internas, com 5 neurônios cada, e treinada durante 1000 épocas, com a utilização de uma taxa de aprendizado igual a 0,5 e uma constante de momento igual a 0,3, obteve um erro quadrático médio aproximadamente igual a 0,245092 para o conjunto de treinamento. O percentual de acertos obtidos foi de 100% para o conjunto de 414 amostras do tumor benigno e 0% para as 219 amostras do maligno, representando 219 classificações errôneas, de um total de 633 padrões do conjunto de teste. Como a rede apresentou um MSE alto e não foi capaz de reconhecer nenhuma amostra da classe maligno, podemos concluir que a RNA-3 não conseguiu aprender as características do problema. Isso pode ocorrer devido a arquitetura não ser suficiente para extrair as características do problema ou o número de épocas de treinamento serem insuficientes para ocorrer a minimização dos erros [28].

A rede RNA - 4 que foi configurada com uma camada interna, com 10 neurônios, e treinada durante 302 épocas, com a utilização de uma taxa de aprendizado igual a 0,25 e uma constante de momento igual a 0,3, obteve um erro quadrático médio aproximada-

mente igual a 0,071758 para o conjunto de treinamento. O percentual de acertos obtidos foi de 87% para o conjunto de 397 amostras do tumor benigno e 98% para as 286 amostras do maligno, representando 57 classificações errôneas, de um total de 683 padrões do conjunto de teste. Foi observado um bom percentual de acerto para duas classes de saída do problema para essa configuração da RNA.

8.1.3 Validação com outros dados

Uma outra base de dados validada foi a fornecida por Claudenir Simões Caires (doutorando pelo Programa de Pós-graduação em Botânica - Departamento de Botânica - Herbário da Universidade de Brasília - UnB), com 105 amostras de 16 espécies de plantas do gênero botânico Phoradendron que pertencente à família das Santalaceae. A base de dados especificava 14 atributos de cada amostra. O dados foram preparados conforme a subseção 5.5.1 e submetidos ao processo de classificação, através do treinamento de uma RNA. Esse treinamento tinha por objetivo encontrar uma configuração de RNA, que simulada pelo SimNeural, fosse capaz de determinar a qual espécie uma determinada amostra contendo os 14 atributos pertence.

Entretanto, depois de vários treinamentos e testes, nenhuma RNA conseguiu extrair características do banco de amostragem para realizar a classificação. Ou seja, não foi possível obter aprendizagem durante os treinamentos. Baseando-se nas observações durante o processo de treinamento, testes e nos critérios de análises propostos por Silva [28], para a compreensão do comportamento e avaliação do desempenho de uma RNA, conclui-se que o conjunto de amostras não fornece elementos suficientes para o aprendizado da rede e os atributos não delineiam padrões separáveis fortes, que caracterizam a espécie, capaz de ser detectada pelas RNAs criadas e simuladas pelo SimNeural.

Ao trabalhar com conjuntos de dados reais deve sempre ser considerado o fato de que as amostras a serem utilizadas estão sujeitas a ruídos, obtidos principalmente, durante os procedimentos de amostragem [5]. Além disso, pode haver dados redundantes e um número de amostras insuficientes para representar o problema desejado. Estes fatores tornam necessária a utilização de procedimentos de validação cruzada, que não é abordada por este projeto, ou regularização da base de dados, para que se possa obter uma rede com melhor capacidade de generalização, ou seja, uma rede que responda corretamente a dados que não foram utilizados durante o processo de treinamento.

8.2 Validação pedagógica

O SimNeural disponibiliza no menu "Exemplos de Modelo", os modelos "RNA Passo_a_ Passo.mox" e "ExemploIris". Esses modelos tem por finalidade ser um ambiente para o processo de ensino-aprendizagem dos conceitos, do algoritmo e da simulação das redes neuronais artificiais de múltiplas camadas que usam o algoritmo de retropropagação para treinamento. Além desses, o SimNeural permite a um professor ou tutor criar modelos específicos e personalizados para uso didático.

O modelo "RNA Passo_a_ Passo.mox" apresenta todos os elementos para criação de uma RNA (Figura 7.9) e seu treinamento. Esse modelo permite uma execução passo a passo e de forma simplificada do *backpropagation*, em sete passos conforme mostra a

Seção 7.3.2 do Capítulo 7, porém não executa uma rede depois de treinada para fazer classificação de padrões.

O modelo "ExemploIris" apresenta um rede pré-treinada que permite a taxinomia de flores iris, conforme detalhada na Seção 7.3.1, no Capítulo 7, e validada na Seção 8.1.1 deste capítulo.

A validação do uso pedagógico do SimNeural pode ser demonstrada respondendo ao seguinte questionamento: *Como o uso do software de simulação de redes neurais, o SimNeural, pode favorecer a prática docente e contribuir para a aprendizagem e para uma pedagogia dinâmica?*

O educador Paulo Freire [10] propõe que: *"Aprender para nós é construir, reconstruir, constatar para mudar ..."*. Ou seja, o processo de aprendizagem ocorre com maior facilidade quando se vivencia e constrói a teoria.

A simulação possibilita ao usuário o desenvolver cenários, testá-los, analisar os resultados obtidos e assim, complementar os conceitos, pois *"... ensinar não é transferir conhecimento, mas criar as possibilidades para a sua própria produção ou a sua construção"* [10]. O SimNeural fornece ferramenta para que o usuário construa seu conhecimento dentro do contexto abordado pelo software.

Nesse sentido, o uso do SimNeural como ferramenta na educação (ensino-aprendizagem) pode ser visto como um processo de descoberta, exploração e de observação, além de possibilitar a construção do conhecimento. Diante disso, o simulador pode transformar o processo de ensino-aprendizagem num instrumento versátil e de grande eficácia. O uso de gráficos e a possibilidade de comparar diversos treinamentos realizados, além de contar com um modelo específico para demonstração do algoritmo *backpropagation*, oferece ao professor uma poderosa ferramenta que pode ser vista como um objeto virtual de aprendizagem no ensino de Redes Neurais Artificiais ou de conceitos de outras áreas, bastando que sejam realizadas adaptações ou aplicações em outros modelos além dos sugeridos e realizados neste projeto.

Um objeto virtual de aprendizagem é [30]:

> "um recurso de digital reutilizável que auxilie na aprendizagem de algum conceito e, ao mesmo tempo, estimule o desenvolvimento de capacidades pessoais, como, por exemplo, imaginação e criatividade. Dessa forma, um objeto virtual de aprendizagem pode tanto contemplar um único conceito quanto englobar todo o corpo de uma teoria. Pode ainda compor um percurso didático, envolvendo um conjunto de atividades, focalizando apenas determinado aspecto do conteúdo envolvido, ou formando, com exclusividade, a metodologia adotada pa determinado trabalho."

8.2.1 Validação junto ao Instituto de Ciências Biológicas da Unb

O Projeto SimNeural foi exposto aos professores, doutorandos e mestrandos do Instituto de Ciências Biológicas da UnB (IB). O tempo médio de apresentação do software para cada pesquisador foi de cerca de uma hora. O *feedback* fornecido por eles foi excelente, sendo que em um primeiro contato com o software, já sugeriram diversas aplicações, tanto pedagógicas quanto nas pesquisas realizadas. Alguns dos pontos levantados foram:

- uso nas pesquisas de doutorando, para identificar espécies da flora;

- integração com software de processamento de imagens para identificação de células com problemas de reprodução por meio de fotos tiradas por microscópios digitais;
- utilização no ensino à distância ou semipresencial.

8.3 Análise dos resultados

Em comparação aos sistemas semelhantes estudados durante a execução deste projeto, as taxas de acerto alcançadas com o sistema podem ser consideradas satisfatórias, pois chegam aos mesmos patamares.

Entretanto, devido a uma característica intrínseca às redes neurais, não é possível fazer considerações sobre quais seriam as melhores configurações para simular a classificação de padrões em base de dados reais. A característica mencionada refere-se à robustez das redes neurais quanto a desvios no padrão do modelo. Assim, por definição, as redes neurais devem ser treinadas e testadas até que ela encontre o aprendizado satisfatório, capaz de reconhecer atributos como pertencentes a um padrão. Entretanto, o que não desvirtua o projeto que teve êxito na sua meta principal, ser uma ferramenta para aprendizado e para uso funcional das redes neuronais artificiais.

O Projeto SimNeural é amigável para profissionais de ciências biológicas, médicas e demais áreas do conhecimento que não possuem familiaridade com a programação, por ser flexível e configurável via interface gráfica, tanto para uso em ensino como em pesquisa. Além disso, um manual de, uso que documenta todo o software, desde sua instalação até ao ajuste dos mais finos parâmetros, contando com cerca de 40 páginas foi escrito para auxiliar o uso didático.

Capítulo 9

Conclusões

Os estudos realizados, o software desenvolvido e as validações realizadas permitem dizer que, de fato, o uso de pacotes de simulação e mais especificamente do pacote de simulação ExtendSim pode auxiliar a criação de modelos de simulação por pessoas das mais diversas áreas do conhecimento, até mesmo distantes da computação.

Conforme visto no desenvolvimento dos Capítulos 2, 3 e 4, a simulação é ferramenta não apenas para planejar, mas para gerar e analisar conhecimento sobre sistemas reais. Há diversas ferramentas que poderiam ser utilizadas; e conseguiu-se entrar a fundo no pacote de simulação ExtendSim, que se mostrou uma ferramenta robusta e ousada ao oferecer ferramentas poderosas para os usuários finais.

Assim, embora haja muitos aspectos positivos neste pacote, é preciso destacar que sua interface apresenta alguns problemas, como a dificuldade para se trabalhar com múltiplas janelas, já que todas as janelas não podem ultrapassar os limites da janela pai impossibilitando o uso de múltiplos monitores. Outro grave problema dos ambientes que podem ser gerados por meio do pacote ExtendSim e que afetou duramente o Projeto SimNeural é que os elementos das caixas de diálogo, como por exemplo um *text label*, não podem ter sua aparência modificada (fonte, tamanho, cor) o que deixa a interface engessada e até mesmo ilegível para pessoas com maiores dificuldades de visão. Neste aspecto, o ExtendSim, como ferramenta didática, deixa a desejar.

Contudo, esses problemas são superados, em muito, pelo poder que a ferramenta confere ao usuário na criação de modelos. O reuso de blocos (componentes) é interessante e a curva de aprendizado do software é exponencial, ainda mais para aqueles que têm conhecimento da linguagem C ou que se aventurarem pelos códigos-fonte distribuídos livremente na maioria dos blocos que acompanham as bibliotecas padrões do ambiente. Já para os usuários que não possuem qualquer conhecimento em linguagens de programação, o reuso dos blocos já construídos permitem o uso do ExtendSim para construção de modelos complexos de simulação.

O modelo escolhido pelos autores deste trabalho, Redes Neurais Artificiais, para testar as funcionalidades do ExtendSim mostrou-se de grande valia para tal fim, pois forçou a criar novos blocos e reutilizar blocos existentes. A interface do SimNeural foi refinada o máximo possível para atender aos objetivos do projeto e mostrou-se clara, segundo avaliações de pessoas ligadas ao Instituto de Ciências Biológicas. Apesar de aspectos positivos, alguns blocos do SimNeural precisam ser refinados em trabalhos futuros, para permitir

um reúso de forma facilitada, pois algumas coisas ainda estão fortemente acopladas aos modelos criados.

Espera-se que o trabalho realizado neste projeto possa oferecer novas possibilidades no ensino de Redes Neuronais Artificiais, e que a aplicação das redes criadas no SimNeural possam ser utilizadas em outras áreas de conhecimento.

9.1 Trabalhos futuros

Durante o projeto percebeu-se que existem muitos algoritmos de aprendizado para redes neurais, até mesmo variantes do próprio *backpropagation*. Novas versões poderiam acrescentar estes novos algoritmos para aumentar a utilidade do SimNeural.

O Projeto SimNeural ainda pode ser refinado para um menor acoplamento dos blocos criados no decorrer deste trabalho. Atualmente, alterar o modelo em baixo nível requer alteração de algumas referências nos blocos e isso pode ser corrigido em trabalhos futuros.

Por último, durante o projeto, surgiu a possibilidade de utilizar o SimNeural para processar dados para uso na classificação de células reprodutivas bovinas. Porém, seria necessária uma integração ou desenvolvimento de software para processamento de imagens destas células, para gerar os dados processáveis por uma RNA. Tal trabalho poderia ser feito em conjunto com os veterinários da UnB.

Referências

[1] R.L. ACKOFF and *et al. Pesquisa operacional.* Livros técnicos e científicos, Rio de, 1977. 12

[2] J. BANKS. *Discrete event simulation, 2 ed.* Prentice Hall, New Jersey, 1996. 11

[3] R.F. BARTON. *A primer on simulation and gaming.* Prentice-Hall, New Jersey, 1970. 13, 14

[4] C. M. BISHOP. *Pattern Recognition and Machine Learning.* Springer Science, Cambridge, USA, 2006. 33, 35, 50

[5] L. N. CASTRO SILVA. Análise e síntese de estratégias de aprendizado para redes neurais artificiais. Dissertação de mestrado, Departamento de Engenharia de Computação e Automação Industrial - Faculdade de Engenharia Elétrica e de Computação - Universidade Estadual de Campinas, 1998. 79

[6] I. FISCHER and *et al. JavaNNS - Java Neural Network Simulator. User Manual, Version 1.1.* Department of Computer Architecture - WILHELM-SCHICKARD-INSTITUTE FOR COMPUTER SCIENCE - UNIVERSITY OF TÜBINGEN, Germany, 2002. 5

[7] R. A. FISHER. The use of multiple measurements in taxonomic problems. *Anais de Eugenics, 7, p. 179-188*, 1936. 72

[8] J.W. FORRESTER and *et al. System Dynamics, 2 ed.* North-Holland Publishing Company, Amsterdam, 1980. 11

[9] A. FRANK and A. ASUNCION. Uci machine learning repository. acessado em 22/06/2012: http://archive.ics.uci.edu/ml. In *UC Irvine Machine Learning Repository.* University of California, Irvine, School of Information and Computer Sciences, 2010. ix, 72, 73, 77

[10] P. FREIRE. *Pedagogia da autonomia: saberes necessários à prática educativa.* 39. Ed. Coleção Leitura. editora: Paz e Terra, São Paulo, 2009. 80

[11] J. HAN and M. KAMBER. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2001. 33, 34

[12] S. HAYKIN. *Redes Neurais : princípios e prática 2. ed.* Traduzido por Paulo Martins Engel. Bookman, Porto Alegre-RS, 2001. 29, 30, 34

[13] Inc. Imagine That. *ExtendSim Developer Reference*. Imagine That Inc., 2007. 26, 27

[14] Inc. Imagine That. *ExtendSim User Guide*. Imagine That Inc., 2007. x, 19, 20, 27, 60

[15] isee systems inc. Stella systems thinking for education and research. *acessado em 16/06/2012. http://www.iseesystems.com/softwares/Education/StellaSoftware.aspx*, 2012. 8

[16] W.D KELTON and *et al. Simulation with Arena*. Mc-Graw-Hill, New York, 1998. x, 10

[17] S. B. KOTSIANTIS. *Supervised machine learning: A review of classification techniques. Informatica, vol. 31, no. 3, pp. 249-268.* Slovene Society Informatika, Greece, 2007. 36

[18] Z. L. KOVACS. *Redes neurais artificiais: fundamentos e aplicações. 4 edição*. Editora Livraria da Fisica, São Paulo, 2006. 29, 33, 36

[19] A.M. LAW. How to conduct a successful simulation study. *Proceedings of the 2003 Winter Simulation Conference S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.*, pages p. 66–70, 2003. viii, 15, 16, 17

[20] A.M. LAW and W.D. KELTON. *Simulation modeling and analysis, 3 ed.* McGraw-Hill, Boston, 2000. 13, 14

[21] Inc. MathWorks. Matlab-product documentation. *acessado em 25/05/2012. http://www.mathworks.com/help/techdoc/learn_matlab/bs98aqv.html*, 2012. viii, 6, 7

[22] T.H. NAYLOR. *Computer simulation experiments with models of economic systems*. John WIley & Sons, New York, 1971. 9, 10, 13

[23] C. A. R. PINHEIRO. *Inteligência Analítica - Mineração de Dados e Descoberta de Conhecimento*. Editora Ciência Moderna Ltda, Rio de Janeiro, 2008. 33, 34, 40, 50, 51

[24] R PRESSMAN. *Software Enginnering: A Practitioner's Approach, 7ed*. Mc-Graw-Hill, 2009. 53

[25] E. RICH and K. KNIGHT. *Inteligência artificial; tradução Maria Cláudia Santos Ribeiro Ratto; revisão técnica Alvaro Antunes*. Makron Books, São Paulo, 1993. 37, 39

[26] R.E. SHANNON. *Systens Simulation: the art and science,*. Prentice Hall, 1975. 11

[27] T. SHIMIZU. *Simulação em computador digital*. Edgard Blucher, São Paulo, 1975. 13, 14

[28] L. F. C. SILVA. Modelo de rede neural artificial treinada com o algoritmo back-propagation. Graduação - monografia, Departamento de Ciência da Computação - Universidade Federal de Juiz de Fora Minas Gerais, UFJF, 2003. 31, 73, 74, 78, 79

[29] I. SOMMERVILLE. *Software Enginnering, 8 edição.* Editora Pearson Addison-Wesley, 2007. ix, 22, 53, 54, 55, 56, 58

[30] W. SPINELLI. Aprendizagem matemática em contextos significativos: Objetos virtuais de aprendizagem e percursos temáticos. Dissertação de mestrado, Universidade de São Paulo - USP, 2005. 80

[31] UNIVERSITY OF TÜBINGEN. Javanns - java neural network simulator. *Disponível em: http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/welcome_e.html Acesso em: 24 de maio de 2012*, 2012. viii, 6

[32] UNIVERSITY OF TÜBINGEN. Snns - stuttgart neural network simulator. *Disponível em: http://www.ra.cs.uni-tuebingen.de/software/snns/welcome_e.html , acessado em 24/05/2012*, 2012. viii, 5

[33] A. ZELL and *et al. SNNS Stuttgart Neural Network Simulator User Manual Version 4.2.* Stuttgart: University of Stuttgart/University of Tübingen, Germany, 2008. 4

[34] E. L. Zílio and U. B. Pinto. 7