KÁSSIO MARQUES TAVARES

PREVISÃO DOS RESULTADOS DO CAMPEONATO BRASILEIRO DE FUTEBOL UTILIZANDO REDES NEURAIS ARTIFICIAIS

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
2015

KÁSSIO MARQUES TAVARES

PREVISÃO DOS RESULTADOS DO CAMPEONATO BRASILEIRO DE FUTEBOL UTILIZANDO REDES NEURAIS ARTIFICIAIS

Projeto de Fim de Curso apresentado como exigência parcial para obtenção do Diploma de **BACHAREL EM ENGENHARIA MECATRÔNICA**, pela Universidade Federal de Uberlândia.

Área de concentração: Inteligência Artificial.

Orientador: Prof. PhD. Keiji Yamanaka

Uberlândia - MG
2015

"Do your best, then don't worry, be happy"

Meher Baba

AGRADECIMENTOS

Aos meus pais, Alba Luzia Marques e José Humberto Tavares, por todo o apoio oferecido ao longo de toda minha vida.

À toda minha família, que independentemente da distância, sempre estiveram comigo quando precisei.

À Universidade Federal de Uberlândia, à Faculdade de Engenharia Mecânica e ao curso de Engenharia Mecatrônica pela oportunidade de realizar esse curso.

Ao Professor PhD. Keiji Yamanaka pela possibilidade de realizar este projeto, além de toda a orientação, paciencia e auxílio oferecidos no decorrer do trabalho.

Aos meus amigos, Alexandre Rodrigues, Bruno Pinheiro, Danilo Andrades, Julianne Rodrigues, Lucas Ferraz, Lucas Lopes, Manoel Martins e, principalmente, à minha namorada Natássia Navarro, por toda a ajuda, direta ou indireta, as quais foram essenciais para a conclusão deste projeto.

À todos que em mim depositaram sua confiança e, mesmo não colaborando diretamente, dirigiram suas boas energias, pensamentos, palavras e intensões em favor da concretização desse projeto.

TAVARES, K. M., **Previsão dos resultados do campeonato brasileiro de futebol utilizando redes neurais artificiais.** 2015. xxxxxxxxxxxxxxx f. Monografia de Conclusão de Curso (Bacharel), Universidade Federal de Uberlândia, Uberlândia.

RESUMO

Prever resultados de jogos de futebol é uma tarefa árdua e por vezes dita impossível. Os resultados são altamente instáveis e abordagens estatísticas tem sua efetividade limitada. Tendo isso em vista, o objetivo desse trabalho foi a idealização de um sistema de predição de resultados tendo como base redes neurais artificiais, cujo uso visa eliminar dois problemas da predição de jogos de futebol: A limitação dos métodos estatísticos, ao se utilizar reconhecimento de padrões; e a eliminação de resultados tendenciosos, ao se usar exclusivamente dados concretos de resultados históricos. A linguagem de programação orientada a objetos Java foi utilizada para a montagem da rede neural artificial desse projeto, a qual consiste de um perceptron multicamadas utilizando a regra de treinamento supervisionado de Backpropagation. Dois conjuntos de dados de entrada foram considerados, um visando exprimir o histórico recente dos times e o outro visando exprimir sua estabilidade e constância. Por fim, os resultados foram validados de forma a excluir alguns palpites menos confiáveis, garantindo assim uma taxa de acerto mais elevada nos restantes.

Palavras Chave: Inteligência Artificial, Redes Neurais Artificiais, Backpropagation, Futebol, Predição.

TAVARES, K. M., **Prediction of the Brazilian soccer championship results using artificial neural networks** 2015. xxxxxxxxxxxxxxx f. Monografia de Conclusão de Curso (Bacharel), Universidade Federal de Uberlândia, Uberlândia.

ABSTRACT

The result prediction of soccer games is an arduous task and sometimes said to be impossible. The results are highly unstable and statistical approaches have their effectivity limited. With this in mind, the goal of this work was the idealization of a result prediction system based on artificial neural networks, the use of which aims to eliminate two problems of the soccer games prediction: The limitation of statistical methods, when using pattern recognition; and the elimination of biased results, when using only concrete data from historical results. The object oriented programming language Java was used to assemble the artificial neural network of this project, which consists of a multilayer perceptron using the supervised Backpropagation training rule. Two sets of input data were considered, one seeking to express the recent history of the teams and the other seeking to express their stability and constancy. Finally, the results were validated to exclude some less reliable predictions, thus ensuring a higher success ratio on the others.

Keywords: Artificial Intelligence, Artificial Neural Networks, Backpropagation, Soccer, Prediction.

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

RNA	-	Rede Neural Artificial
POO	-	Programação Orientada a Objetos
IA	-	Inteligência Artificial
NI	-	Neurônio Intermediário
IDE	-	Integrated Development Environment; Ambientes de Desenvolvimento Integrado
GUI	-	Graphical User Interface; Interface Gráfica do Utilizador
API	-	Application Programming Interface; Interface de Programação de Aplicações
UFU	-	Universidade Federal de Uberlândia
$FEMEC$	-	Faculdade de Engenharia Mecânica
CBF	-	Confederação Brasileira de Futebol
α	-	Taxa de aprendizado
P	-	Probabilidade do neurônio de saída
σ	-	Desvio padrão das probabilidades dos neurônios de saída

SUMÁRIO

CAPÍTULO I

INTRODUÇÃO

O trabalho com redes neurais artificiais, também referidas simplesmente como RNA, foi motivado desde a sua criação pelo reconhecimento de que o cérebro humano processa informações de uma maneira completamente diferente dos computadores digitais modernos. O cérebro é um sistema altamente complexo, não linear e que processa paralelamente suas informações. Possui também a capacidade de organizar seus elementos estruturais, conhecidos como neurônios, de forma a realizar certas tarefas (e.g., reconhecimento de padrões, percepção e controle motor) (HAYKIN, 1999a).

Para alcançar uma boa performance, redes neurais empregam uma interconexão massiva de células computacionais simples, que são referenciadas como "neurônios"ou "unidades de processamento". Segundo HAYKIN (1999a) uma rede neural é um processador com distribuição paralela massiva, feito de unidades de processamento simples, as quais possuem uma propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso.

Redes Neurais são ferramentas úteis para a resolução de diversos problemas, os quais podem ser caracterizados como mapeamento (incluindo associação e classificação de padrões), agrupamento, e otimização restrita. Existem diversas redes neurais disponíveis para cada tipo de problema (FAUSSET, 1994). Pela perspectiva de reconhecimento de padrões, redes neurais podem ser consideradas como uma extensão das muitas técnicas convencionais que vêm sendo desenvolvidas ao longo de várias décadas (BISHOP, 1995).

O futebol, por sua vez, tem sido desde a sua criação (no século XIX pelos ingleses)

um dos esportes mais conhecidos e jogados pelo mundo. Há muito, vários países (principalmente Europeus e Sul-americanos) já fizeram do futebol o seu esporte mais popular (SILVA et al., 2002).

O futebol brasileiro é mundialmente conhecido por sua qualidade e pela paixão que os brasileiros sentem pelo esporte. Segundo DA COSTA (2005), o esporte é praticado por aproximadamente 30 milhões de brasileiros e possui 102 milhões de torcedores apenas dentre os 16 maiores clubes (em termos de tamanho de torcida).

Um fato bastante aceito entre os amantes do futebol diz respeito à imprevisibilidade dos resultados das partidas. Os jogos são disputados entre dois times e podem terminar com vitória do time da casa, vitória do time visitante e empate. Para a determinação do resultado de uma partida, encontram-se fatores esperados e imprevistos, ambos os quais têm influência no resultado da disputa (ASLAN e INCEOGLU, 2007).

A abordagem estatística para previsão de resultados já é bastante utilizada e seu aprimoramento já foi bastante estudado, como pode ser visto em ELO e SLOAN (2008), MEHREZ e HU (1995) e SILVA et al. (2002). Apesar dos esforços e estudos, a assertividade desses métodos ainda não chegou a ultrapassar a taxa de 50%.

Por sua vez, as RNAs têm se mostrado uma ferramenta muito poderosa para modelagem matemática, tanto em pesquisas quanto em aplicações práticas. Elas podem efetuar mapeamentos entre espaços de entrada e saída altamente não lineares. Com sua abordagem não-paramétrica, faz-se desnecessária a análise funcional da forma de distribuição dos dados. Assim sendo, se mostram uma ferramenta com grande potencial para a previsão dos resultados de partidas de futebol (CHENG et al., 2003).

1.1 Objetivos

O objetivo geral desse trabalho é desenvolver um programa em Java, capaz de utilizar a técnica de RNAs para realizar a previsão do resultado de uma partida do Campeonato Brasileiro de Futebol.

Os objetivos específicos são:

- Estudar sistemas computacionais estruturados em Redes Neurais Artificiais;

- Estudar a linguagem de programação Java, usada como base para a criação do algoritmo da RNA;
- Estudar dados de partidas de futebol e encontrar informações relevantes que influenciem o resultado das mesmas;
- Criação de um banco de dados estruturado e versátil com os dados de partidas passadas;
- Desenvolvimento do software responsável pela análise e tratamento dos dados e fornecimento dos resultados;
- Análise profunda de dados e da topologia da RNA criada, de forma a otimizar o seu funcionamento.

1.2 Justificativa

O resultado das partidas de futebol é hoje um fator ainda imprevisível. Os estudos já realizados para prever o desfecho de um jogo utilizam quase que exclusivamente métodos estatísticos. Estes por sua vez, apesar das diversas técnicas e abordagens já utilizadas, ainda não conseguiram acertar mais da metade dos resultados. Existem alguns poucos estudos com abordagens hibridas, utilizando estatística e redes neurais artificiais, os quais fornecem uma melhor taxa de acerto. Contudo, essa taxa ainda é baixa e gira em torno de 50%.

A proposta de utilizar somente redes neurais e reconhecimento de padrões é inovadora e promissora. Abdicando dos recursos estatísticos tem-se uma liberdade maior para o reconhecimento de anomalias, que outrora seriam tratadas como ruídos. A instabilidade da relação dados-resultados causada pelos mais diversos fatores do ambiente futebolístico, pode ser contornada pela capacidade inata das redes neurais de mapeamento, percepção e reconhecimento de padrões.

Apesar de o futebol ser o esporte mais seguido no Brasil, estudos de previsão de resultados utilizando dados de jogos disputados com times do Brasil são desconhecidos. O uso inovador de redes neurais e a utilização de dados do Campeonato Brasileiro são, portanto, inéditos. Deste modo, nos deparamos com um campo de estudos promissor e ainda não explorado.

CAPÍTULO II

FUNDAMENTAÇÃO TEÓRICA

Este capítulo trata de alguns assuntos importantes para a compreensão do trabalho, tais como: Redes Neurais Artificiais, Perceptrons, Adaline, Madaline, Regra Delta, funções de ativação, camadas intermediárias, algoritmos de treinamento, Backpropagation, programação orientada a objetos e Java.

2.1 Redes Neurais Artificiais

Toda a base para a realização desse trabalho se encontra no uso uma rede neural artificial (RNA), adequada para a realização do reconhecimento de padrões entre condições anteriores às partidas e o resultado das mesmas. Uma RNA é um sistema de processamento de informações que possui algumas características em comum com redes neurais biológicas. O desenvolvimento da teoria de redes neurais artificiais ocorreu a partir de generalizações matemáticas de modelos de percepção humana. LAURENE FAUSETT. Para isso, algumas hipóteses são levadas em conta: Assume-se que o processamento de informações ocorre em vários elementos simples chamados neurônios; Sinais são passados entre neurônios através de uma conexão; Cada conexão possui um peso associado, o qual, em redes neurais artificiais típicas, multiplicam o sinal transmitido; Cada neurônio possui uma função de ativação, a qual determina o sinal de saída com base na soma dos sinais de entrada. Um modelo completo de um neurônio, com base nas hipóteses acima, está mostrado na Figura 2.1.

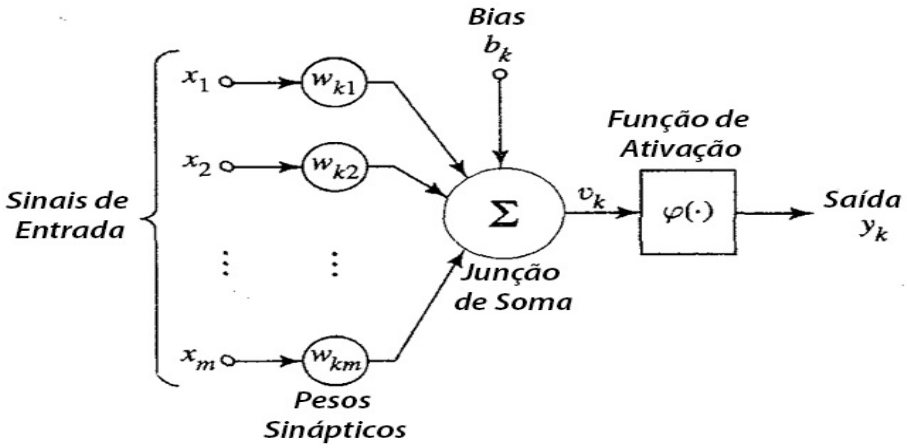

Figura 2.1 – Modelo completo de um neurônio (HAYKIN, 1999a)

2.1.1 McCulloch-Pitts

O primeiro modelo de neurônio artificial foi concebido por Warren McCulloth e Walter Pitts (MCCULLOTH e PITTS, 1943). Algumas das regras de funcionamento desses neurônios são aplicadas ainda hoje nas redes neurais modernas. Podemos descrever as regras que permeiam o funcionamento dos mesmos com base nas seguintes definições:

- A ativação do neurônio é binária. Ou seja, para cada situação de entrada existem apenas duas saídas, ativado ou desativado;
- Os neurônios são conectados diretamente por ligações, que possuem seu respectivo peso único;
- Uma ligação é excitatória se seu peso é positivo, e inibitória se seu peso é negativo. Todas conexões excitatórias possuem o mesmo peso;
- Cada neurônio possui um valor limiar, e sua ativação é função da soma dos pesos de entrada (excitatórios e inibitórios). A ativação ocorre quando a soma dos pesos ultrapassa o valor limiar;
- A definição do limiar ocorre de forma que qualquer entrada não inibitória diferente de zero faça com que o neurônio não seja ativado;
- É necessário um passo temporal para que o sinal passe por um link da conexão.

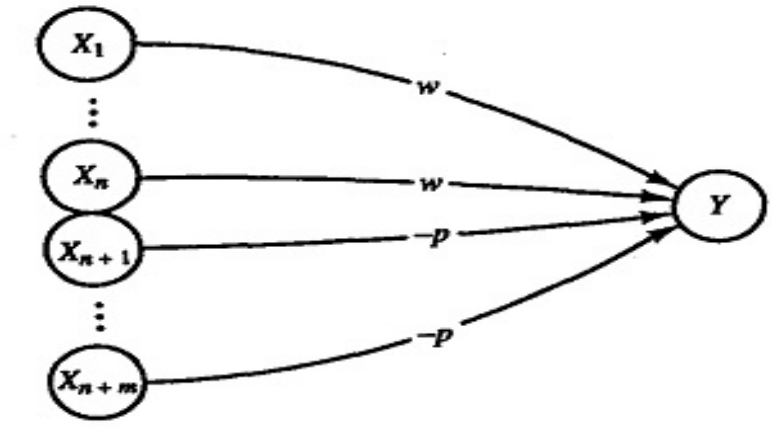

Figura 2.2 – Neurônio de McCulloch-Pitts (FAUSSET, 1994)

Um modelo completo de um neurônio, segundo as definições acima, é mostrado na Figura 2.2.

A função de ativação (degrau) desse neurônio é dado segunda a Equação 2.1, onde Yin é a soma das entradas de cada um dos neurônios, conforme Equação 2.2. O limiar do neurônio é dado pela Equação 2.3.

$$f(y_{in}) = \begin{cases} 1 & \text{se } Y_{in} > \theta \\ 0 & \text{se } Y_{in} \leq \theta \end{cases} \tag{2.1}$$

$$Y_{in} = n \cdot w + m \cdot p \tag{2.2}$$

$$\theta = n \cdot w - p \tag{2.3}$$

2.1.2 Bias

O uso de um limiar para definir a ativação do neurônio pode ser substituído pelo uso de um bias. O bias age exatamente como o peso de uma conexão cuja entrada é sempre 1 (FAUSSET, 1994), conforme mostrado na Figura 2.3.

O uso do bias ou do limiar são equivalentes. Sua importância se mostra durante a análise da separabilidade linear de um problema, conforme mostrado na Subseção 2.1.3, e durante a análise de convergência do erro, não tratada nesse trabalho.

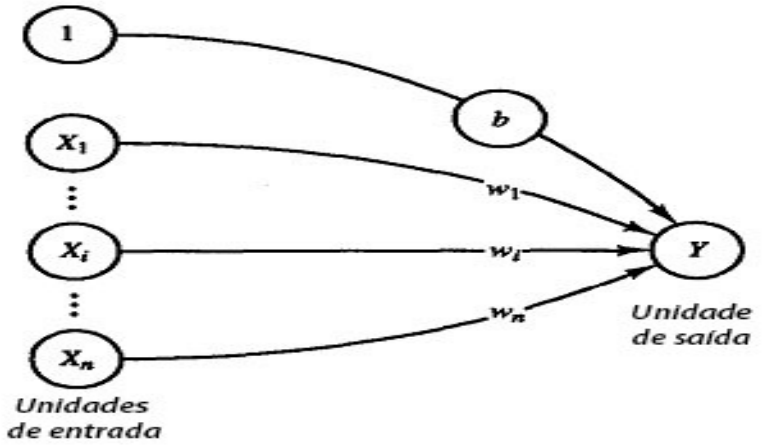

Figura 2.3 – Bias "b" em uma RNA

A diferença de aplicação entre as duas técnicas é dada pela função de ativação do neurônio, que com o uso do bias fica escrita conforme a Equação 2.4.

$$f(y_{in}) = \begin{cases} 1 & \text{se } Y_{in} > 0 \\ 0 & \text{se } Y_{in} \leq 0 \end{cases} \tag{2.4}$$

2.1.3 Separabilidade Linear

Para uma determinada rede, a resposta está confinada entre dois valores (1 ou 0, sim ou não, -1 ou 1, etc...). Existe uma fronteira, chamada *fronteira de decisão* Figura 2.4 que delimitada as regiões em que a resposta assume um ou outro valor.

A Equação 2.5 mostra a relação que determina essa fronteira para funções de ativação do tipo degrau. Dependendo do número de entradas na rede, essa equação pode representar uma linha, um plano ou um hiperplano.

$$b + \sum_i x_i \cdot w_i = 0 \tag{2.5}$$

Diz-se que um problema é linearmente separável caso existam pesos (e bias) suficientes para que todos vetores de entrada do conjunto de treinamento sejam separados pela fronteira de decisão em dois grupos, de forma que os vetores cujas saídas sejam positivas fiquem todos em um grupo, e os vetores com saídas negativas fiquem em outro. Essas duas regiões são geralmente

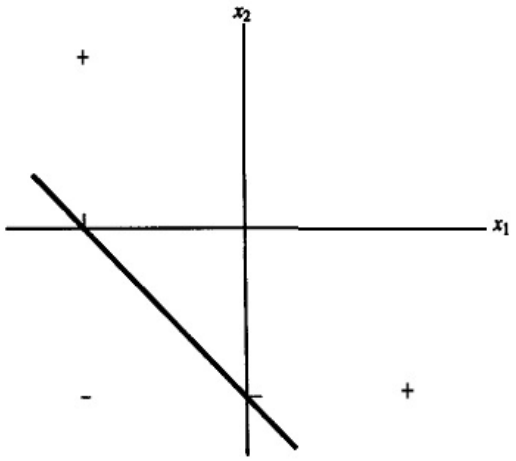

Figura 2.4 – Fronteira e Regiões de decisão para a função lógica *OU* (FAUSSET, 1994)

chamadas de *regiões de decisão* (Figura 2.4).

MINSKY e PAPERT (1969) mostraram que problemas não linearmente separáveis, são impossíveis de serem resolvidos por redes de apenas uma camada. É possível demonstrar também que redes de várias camadas, com funções de ativação lineares, também são incapazes de resolver tais problemas.

A inclusão de um bias pode transformar um problema linearmente inseparável (e portanto incapaz de ser resolvido com os métodos já apresentados) em um problema que pode ser resolvido. Isso é possível graças a adição de uma nova conexão de entrada, que será usada como valor constante para o bias. Essa nova conexão aumenta o conjunto de entradas, fornecendo mais um grau de liberdade para a rede.

2.1.4 Representação dos dados

No início dos estudos de redes neurais, a representação de dados se dava na forma binária, ou seja, utilizando 1 para reforço positivo e 0 para reforço negativo. Essa representação inicial não é, de modo geral, recomendada.

A forma bipolar é uma alternativa bastante eficiente. Essa representação utiliza 1 para reforço positivo e -1 para reforço negativo. Assim como a inclusão de um bias, a simples mudança da representação de dados para a forma bipolar pode transformar um problema sem solução em um problema que pode ser resolvido.

Caso o problema proposto requeira da rede uma generalização dos dados de entrada, o uso da forma bipolar é altamente indicado. A representação bipolar permite que dados não disponíveis sejam adotados com valor 0 de entrada, não fornecendo assim reforço positivo ou negativo.

Ao contrário do que acontece na forma binária, a representação bipolar também é mais eficiente do ponto de vista do treinamento. Por exemplo, em uma rede treinada pelo método de Hebb (Subseção 2.1.5), para conjuntos de sinais entrada cujo resultado da rede seja negativo (0 para binário ou -1 para bipolar), o uso da primeira representação impossibilita completamente um aprendizado da rede.

2.1.5 Aprendizado de Hebb

Das regras de treinamento de redes neurais, a regra de Hebb é a mais antiga, mais simples e mais conhecida de todas. Hebb propôs seu método de treinamento tendo como base um contexto neurobiológico. STENT (1973) e CHANGEUX e DANCHIN (1976) reescreveram o pensamento de Hebb de uma forma mais simples e concisa, como se segue:

- Se dois neurônios em ambos os lados de uma sinapse (conexão) são ativados simultaneamente (i.e. de forma síncrona), então a força dessa sinapse é seletivamente aumentada.
- Se dois neurônios em ambos lados de uma sinapse são ativados de forma assíncrona, então essa sinapse é seletivamente enfraquecida ou eliminada.

A afirmação original de Hebb não incluía a segunda parte. Da mesma forma, ela também comentava apenas a respeito de neurônios sendo ativados ao mesmo tempo.

McCLELLAND e RUMELHART (1988) estenderam a regra de Hebb para que o reforço positivo da sinapse também ocorra caso os neurônios sejam ambos desativados simultaneamente. Podemos, então, complementar o pensamento anterior:

- Se dois neurônios em ambos os lados de uma sinapse são DESATIVADOS simultaneamente, então a força dessa sinapse é seletivamente aumentada.

Estamos considerando inicialmente redes de apenas uma camada, onde cada conexão possui, em cada lado, apenas um neurônio de entrada e um neurônio de saída. O valor do neurônio

de entrada X é multiplicado pelo peso W dessa conexão para se obter o valor do neurônio de saída Y. Seguindo essa nomenclatura e representando os dados na forma bipolar, podemos representar a atualização do peso pela regra de Hebb segundo a Equação 2.6.

$$\Delta w_{ij} = x_i \cdot y_i \tag{2.6}$$

Se utilizarmos a representação binária, essa equação não distingue conexões que possuem uma entrada "positiva" e uma saída "negativa" de conexões onde tanto a entrada quanto a saída são "negativas". Essa é uma das vantagens mais facilmente observáveis da utilização de dados bipolares (ao invés de binários) para a representação dos dados.

Técnica de aprendizado

Existem várias técnicas de implementação das regras de aprendizado existentes. A técnica abordada aqui nada mais é do que uma das formas mais simples e genéricas de se aplicar a regra de Hebb. O entendimento dessa técnica é crucial para a posterior compreensão das outras técnicas aplicadas nesse trabalho, que podem ser tomadas como simples variações mais elaboradas desta.

O aprendizado é dado na forma iterativa e segue o seguinte algoritmo:

1. Os pesos de todas conexões são inicializados com o valor 0.

2. Para todos os conjuntos de entradas e saídas, repetem-se os passos 3-4.

    3. Para cada conexão, calculamos a variação do seu respectivo peso pela equação 6.

    4. Cada peso é atualizado conforme calculado no passo 3.

Para esclarecer um pouco mais o algoritmo acima mostrado, segue um exemplo de sua aplicação em uma rede real com representação bipolar, onde usaremos como parâmetros de entrada a função lógica E. Buscaremos com essa rede descobrir os valores dos pesos das conexões que determinam a fronteira de decisão dessa função.

A função lógica E retorna um valor "positivo" apenas se ambas as entradas possuírem também um valor "positivo". Assim sendo, montaremos a tabela verdade dessa função conforme a Tabela 2.1. A representação gráfica dessa rede está mostrada na Figura 2.5.

Tabela 2.1 – Tabela Verdade para a função lógica E

Entradas			Saída
Bias	X_1	X_2	Y
1	1	1	1
1	1	-1	-1
1	-1	1	-1
1	-1	-1	-1

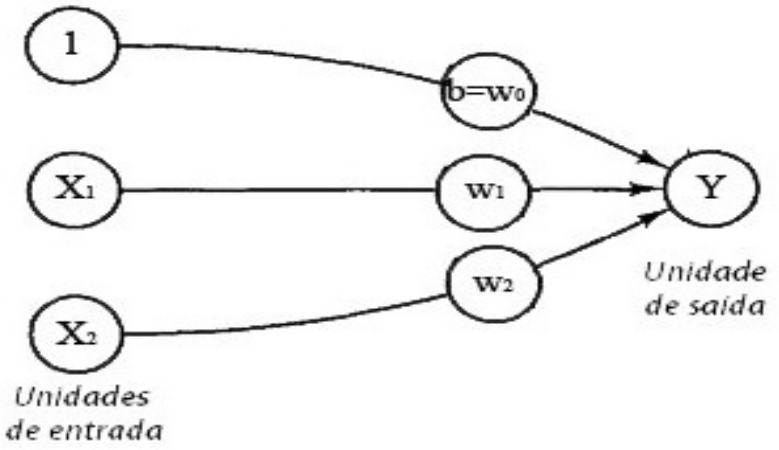

Figura 2.5 – RNA para a função lógica E

Iniciamos o treinamento conforme o passo 1 do algoritmo. Fazemos então os valores de $w_0 = 0, w_1 = 0$ e $w_2 = 0$.

Utilizaremos o primeiro par de entradas (1 ,1). Assim, calculamos o valor da variação dos pesos conforme a Equação 2.6. O resultado é mostrado na Tabela 2.2.

Tabela 2.2 – Variação dos pesos para o primeiro par de entradas

	Entrada		Saída desejada	Variação dos pesos (Δw)		
1	X_1	X_2	Y	Δw_0	Δw_1	Δw_2
1	1	1	1	1	1	1

Na sequência, atualizamos os pesos com os resultados da Equação 2.7.

$$\begin{cases} w_0 = 0 + \Delta w_0 = 1 \\ w_1 = 0 + \Delta w_1 = 1 \\ w_2 = 0 + \Delta w_2 = 1 \end{cases} \tag{2.7}$$

Repetimos o procedimento para o segundo par de entradas (1, -1). Assim, calculamos o valor da variação dos pesos conforme a Equação 2.6. O resultado é mostrado na Tabela 2.3.

Tabela 2.3 – Variação dos pesos para o segundo par de entradas

	Entrada		Saída desejada	Variação dos pesos (Δw)		
1	X_1	X_2	Y	Δw_0	Δw_1	Δw_2
1	1	-1	-1	-1	-1	1

Na sequência, atualizamos os pesos com os resultados da Equação 2.8.

$$\begin{cases} w_0 = 1 + \Delta w_0 = 0 \\ w_1 = 1 + \Delta w_1 = 0 \\ w_2 = 1 + \Delta w_2 = 2 \end{cases} \tag{2.8}$$

Repetimos o procedimento para o terceiro par de entradas (-1, 1). Calculamos o valor da variação dos pesos conforme a Equação 2.6. O resultado é mostrado na Tabela 2.4.

Tabela 2.4 – Variação dos pesos para o terceiro par de entradas

	Entrada		Saída desejada	Variação dos pesos (Δw)		
1	X_1	X_2	Y	Δw_0	Δw_1	Δw_2
1	-1	1	-1	-1	1	-1

Na sequência, atualizamos os pesos com os resultados da Equação 2.9.

$$\begin{cases} w_0 = 0 + \Delta w_0 = -1 \\ w_1 = 0 + \Delta w_1 = 1 \\ w_2 = 2 + \Delta w_2 = 1 \end{cases} \tag{2.9}$$

Repetimos uma última vez o procedimento para o quarto par de entradas (-1, -1).

Calculamos o valor da variação dos pesos conforme a Equação 2.6. O resultado é mostrado na Tabela 2.5.

Tabela 2.5 – Variação dos pesos para o quarto par de entradas

	Entrada		Saída desejada	Variação dos pesos (Δw)		
1	X_1	X_2	Y	Δw_0	Δw_1	Δw_2
1	-1	-1	-1	-1	1	1

Na sequência, atualizamos os pesos com os resultados da Equação 2.10.

$$\begin{cases} w_0 = -1 + \Delta w_0 = -2 \\ w_1 = 1 + \Delta w_1 = 2 \\ w_2 = 1 + \Delta w_2 = 2 \end{cases} \tag{2.10}$$

Terminamos assim o treinamento e a rede resultante é mostrada na Figura 2.6.

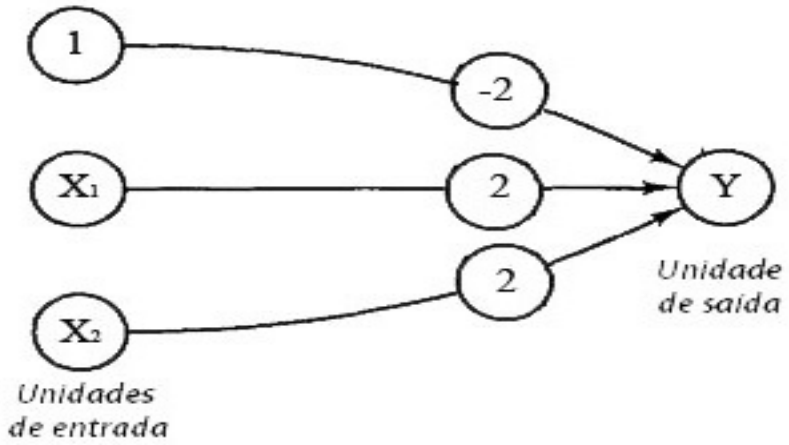

Figura 2.6 – RNA resolvida para a função lógica *E*

Essa configuração de pesos nos fornece uma fronteira de decisão em forma de reta, que pode ser encontrada utilizando a Equação 2.11. A equação da reta resultante do exemplo acima é dada pela Equação 2.12 e está mostrada na Figura 2.7.

$$1 + X_1 + X_2 = 0 \tag{2.11}$$

$$X_2 = -X_1 + 1 \tag{2.12}$$

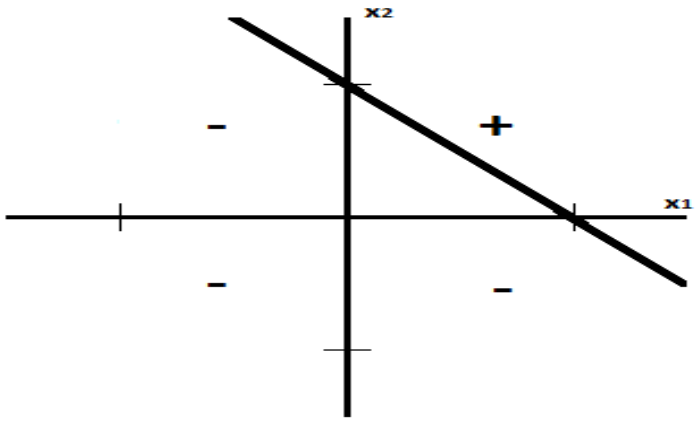

Figura 2.7 – Fronteira de decisão da função lógica *E*

Podemos notar que apesar de os pesos terem sido alterados pelo último loop, a equação da reta (e consequentemente a fronteira de decisão) não é alterada após o mesmo. Isso significa que o treinamento já havia encontrado a solução do problema antes do fim do algoritmo. Como será mostrado mais adiante, outras técnicas de treinamento podem continuar indefinidamente, repetindo os vetores de entrada, refazendo os cálculos sem fim. Nesses casos, uma condição de parada deverá ser adotada.

2.1.6 Perceptron

Redes com um modelo de treinamento iterativo e supervisionado foram propostas por Frank Rosenblatt em 1962. Ele as chamou de Perceptrons e elas tiveram um grande impacto nos estudos de redes neurais artificiais. Isso se deve ao fato de que a regra de convergência dos perceptrons é mais poderosa do que sua antecessora, a regra de Hebb. Além disso, sob determinadas condições, é possível provar matematicamente a convergência de seu procedimento de aprendizado iterativo. Basicamente, ela consiste de um simples neurônio com pesos sinápticos e bias ajustáveis.

Um perceptron particularmente simples utilizava valores binários de entrada e saída, contudo a função de ativação do neurônio de saída poderia retornar valores de -1, 0 ou 1 durante o aprendizado. A possibilidade de 3 diferentes valores só ocorre graças ao novo tipo de função de ativação, onde o valor do limiar passou a delimitar uma região intermediária, e não apenas um valor limítrofe entre saídas positivas e negativas. Essa função é descrita pela Equação 2.14.

$$Y_{in} = \sum_i x_i \cdot w_i \tag{2.13}$$

$$f(Y_{in}) = \begin{cases} 1 & \text{se } Y_{in} > \theta \\ 0 & \text{se } -\theta \leq Y_{in} \leq \theta \\ -1 & \text{se } Y_{in} < \theta \end{cases} \tag{2.14}$$

Para se atualizar os pesos das ligações durante o treinamento usamos a equação 11. Tal equação apresenta um novo parâmetro, chamado de taxa de aprendizado, que é um valor constante determinado antes do início do treinamento.

$$\Delta w_i = \alpha \cdot x_i \cdot t \tag{2.15}$$

Observando a função de ativação de saída, que retorna valores de -1, 0 ou 1, e a saída real (Target t) que deve ter valores de -1 ou 1, podemos concluir alguns pontos importantes relacionados a esse perceptron.

- A rede não diferencia situações onde, por exemplo, a saída real é -1 e a saída calculada é 0 ou 1. Em ambos os casos os pesos são atualizados, de maneira igual, e o sinal do erro informa se os mesmos devem aumentar ou diminuir.
- Apenas conexões de entrada com valores diferentes de 0 possuem seus pesos atualizados.
- Padrões de entrada que não produzem erro na saída também não têm seus pesos atualizados.
- Como o treinamento ocorre até que os pesos sejam tais que a saída seja correta para quaisquer padrões de entrada, e padrões sem erro não resultam em treinamento da rede, quanto mais treinada a rede estiver, menor será o seu aprendizado.
- O valor do limiar não mais fornece uma equação, delimitando uma fronteira de decisão, mas sim uma inequação, fornecendo uma região de "indecisão", separando a região de respostas positivas e a região de respostas negativas. Dessa forma, o valor do limiar se torna mais significativo para o processo de aprendizagem.

O treinamento do perceptron continua até que os pesos das conexões sejam tais que para todos os padrões de entrada a resposta obtida pela rede seja correta. Segundo o teorema de convergência dos perceptrons, caso tais pesos existam, eles serão encontrados em um número finito de passos do treinamento.

Ao contrário das redes mostradas até agora, no perceptron o uso do limiar e do bias não são equivalentes. Como agora eles possuem funções diferentes, ambos são necessários para que o treinamento funcione corretamente.

Técnica de aprendizado

O algoritmo de aprendizado do perceptron está apresentado a seguir. Sua estrutura é semelhante à do treinamento de Hebb, porém já mais avançado e mais parecido com o algoritmo utilizado para a realização desse trabalho.

1. Inicializar pesos e bias (podem ser setados em 0, por simplicidade)
2. Definir taxa de aprendizado $0 < \alpha < 1$
3. Enquanto a condição de parada não for atingida, repetem-se os passos 4-7
    4. Para todos os conjuntos de entradas e saídas, repetem-se os passos 5-6
        5. Calcula-se o valor da saída de acordo com a Equação 2.13 e a Equação 2.14
        6. Caso a saída calculada seja diferente da saída correta, atualizam-se os pesos de acordo com a Equação 2.15
        7. Caso nenhum peso tenha sido alterado no passo 6, pare. Caso contrário, continue

2.1.7 Adaline

WIDROW e HOFF Jr. (1960) apresentaram um novo modelo de redes chamado Adaline, que é uma redução para Adaptive Linear Neuron (Neurônio Linear Adaptativo, em tradução livre). A inovação apresentada por eles se destaca pela regra de treinamento utilizada, a Regra Delta, também conhecida por Regra dos Mínimos Quadrados Médios (LMS) ou Regra de

Widrow-Hoff. A Regra Delta se mostra como uma inovação diante das opções anteriores à ela devido ao fato de que é capaz de fornecer correções pequenas e variáveis ao longo do treinamento. Seu desenvolvimento se deu com a ideia de que a mudança dos pesos da conexão deve minimizar a diferença entre a saída calculada Yin e o valor alvo t, minimizando o erro ao longo de todos os padrões de entrada. Isso é obtido se reduzindo o erro em cada padrão de entrada, um por vez. Para se atualizar os pesos das ligações usamos a Equação 2.17. Tal equação, assim como nos perceptrons, apresenta um parâmetro chamado de taxa de aprendizado, que é um valor constante determinado antes do início do treinamento.

$$Y_{in} = \sum_i x_i \cdot w_i \tag{2.16}$$

$$\Delta w_i = \alpha \cdot x_i \cdot (t - Y_{in}) \tag{2.17}$$

Semelhantemente as redes já apresentadas, um Adaline tipicamente usa ativação bipolar para os sinais de entrada e saída, possui pesos ajustáveis nas conexões e também possui bias. Apesar da Regra Delta poder ser utilizada em redes com múltiplas saídas e/ou camadas, um Adaline é um caso especial onde existe apenas uma saída e uma camada.

A arquitetura de uma rede Adaline é mostrada na Figura 2.8.

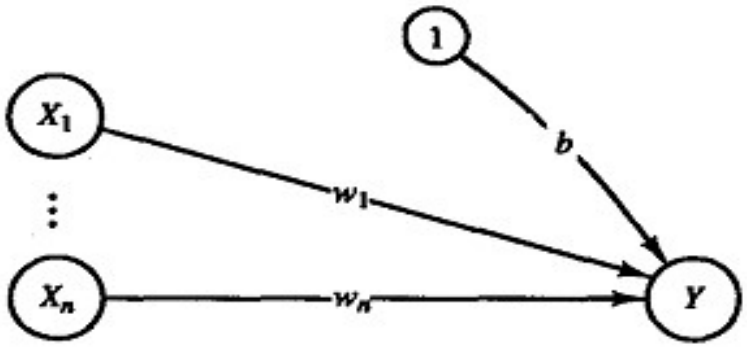

Figura 2.8 – Arquitetura de uma Adaline (FAUSSET, 1994)

Técnica de aprendizado

O algoritmo de aprendizado do Adaline está apresentado a seguir. Sua estrutura não é muito diferente à do perceptron, contudo, possui uma mudança singela: Os pesos aqui são alterados em todos loops de treinamento e o controle de parada se dá por um limiar máximo de alteração dos pesos. Como pode ser visto abaixo.

1. Inicializar pesos e bias (podem ser setados em 0, por simplicidade)
2. Definir taxa de aprendizado $0 < \alpha < 1$
3. Enquanto a condição de parada não for atingida, repetem-se os passos 4-7
    4. Para todos os conjuntos de entradas e saídas, repetem-se os passos 5-6
        5. Calcula-se o valor da saída de acordo com a Equação 2.16
        6. Atualizam-se os pesos de acordo com a Equação 2.17
    7. Caso a maior alteração de peso que ocorreu no passo 6 seja menor que um valor determinado, pare. Caso contrário, continue.

A taxa de aprendizado deve ser determinada com cuidado. Um valor muito pequeno pode fazer com que a rede seja treinada muito lentamente, e de forma oposta, um valor muito grande pode fazer com que o treinamento não convirja. HECHT-NIELSEN (1990) chegou a uma equação para limitar o valor da taxa de aprendizado. Porém, é mais comum simplesmente a escolha de um valor inicial pequeno (0,1 por exemplo).

Um outro ponto importante desse treinamento, se deve ao fato de que a aplicação da rede em situações que a saída deve ser bivalente não pode ser apenas uma repetição de parte do algoritmo de treinamento, pois o mesmo retornaria valores quebrados. Assim, a aplicação do Adaline em tais situações requer uma função de ativação para poder ser usada. Comumente a função degrau é utilizada para tais aplicações.

2.1.8 Redes Multicamadas

Muitas vezes, uma rede neural de apenas uma camada, como as mostradas até agora, não é capaz de solucionar o problema proposto. Tais limitações foram um dos fatores que reduziram o interesse nas redes neurais artificiais nos anos 70. Para tentar solucionar esses casos, foram desenvolvidas as redes multicamadas (Figura 2.9). Nesse trabalho serão expostas as redes Madaline e Multilayer Perceptron com BackPropagation, que são extensões multicamadas das redes Adaline e Perceptron, respectivamente.

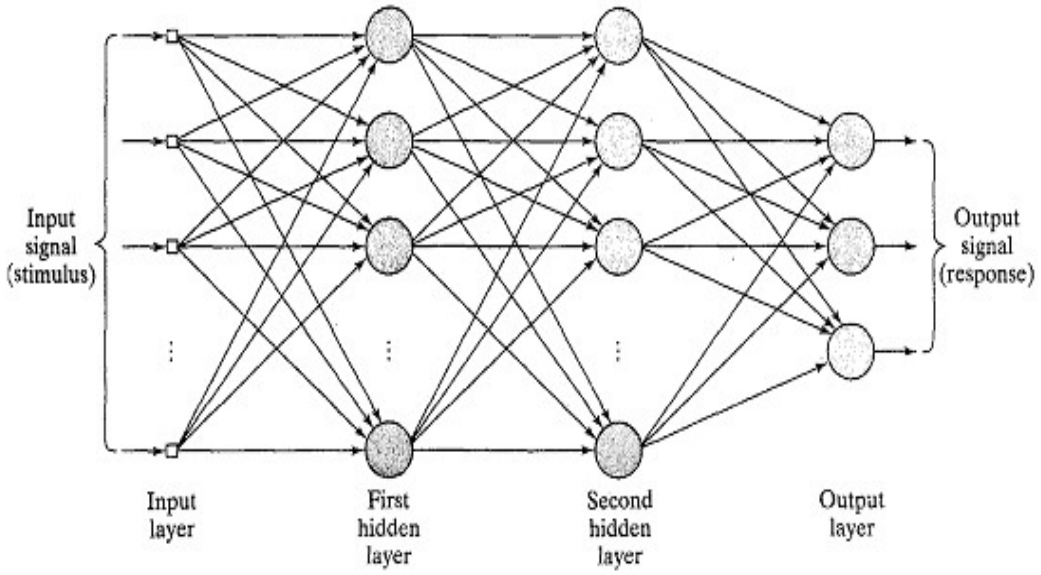

Figura 2.9 – Arquitetura de uma rede multicamadas com duas camadas intermediárias (HAYKIN, 1999a)

2.1.9 Madaline

O uso das redes Adaline em uma configuração multicamadas é possível, e chamado Madaline - Many Adaptive Linear Neurons (Muitos Neurônios Lineares Adaptativos, em tradução livre).

Um modelo simples de uma Madaline pode ser visto na Figura 2.10. Nela, temos dois neurônios de entrada (X1 e X2), duas Adalines resultando dois neurônios intermediários (Z1 e Z2) e uma terceira Adaline, que utiliza os neurônios intermediários (aqui chamados NI) como entradas e resulta a saída Y. Podemos ver que cada uma das três Adalines intermediarias precisam de um bias.

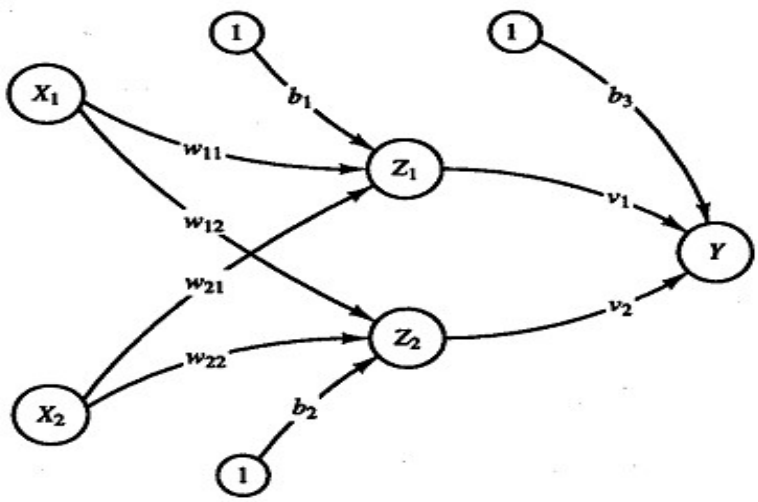

Figura 2.10 – Arquitetura de uma Madaline com uma camada intermediária de dois neurônios (FAUSSET, 1994)

O processo de treinamento de uma Madaline não sofre grandes alterações em relação à sua versão mais simples, de uma camada. Basicamente, a rede é uma combinação de várias Adalines, assim como seu processo de treinamento também é a combinação do processo de treinamento de várias Adalines. Assim sendo, o mesmo algoritmo já mostrado é apenas repetido para cada pedaço da rede multicamadas.

Devemos, entretanto, levar em conta que as entradas do sistema são utilizadas apenas pelas Adalines que resultam a primeira camada intermediária. Por sua vez, a primeira camada intermediária fornece as entradas para as Adalines que resultam a segunda camada intermediária, e assim sucessivamente, até que se chegue na saída do sistema.

2.1.10 Backpropagation

O uso de perceptrons multicamadas, com a regra de treinamento supervisionado de Backpropagation (retro propagação de erros) ou Regra Delta Generalizada, tem sido amplo e eficaz na resolução de problemas, por vezes complexos, em diversas áreas. Apesar de sua importância atual, não se diz que o Backpropagation possui um criador, pois foi desenvolvido simultaneamente por diversos pesquisadores. Esse método visa reduzir o gradiente do erro quadrático total da saída da rede ao longo do treinamento, ou seja, minimizando o erro da saída calculado pelo sistema.

O algoritmo utilizado nesse trabalho usa a regra de Backpropagation. O objetivo é treinar a rede para alcançar um balanço entre a habilidade de responder corretamente a padrões de entrada (utilizados para treinamento) e a habilidade de responder bem a entradas similares, mas não necessariamente idênticas, às utilizadas no treinamento.

Podemos dizer que o Backpropagation utiliza o melhor de cada um dos modelos de redes neurais já apresentados. Em sua base temos uma arquitetura multicamadas, baseada em Perceptrons (Figura 2.11). A regra de treinamento utilizada é a Regra Delta Generalizada, que é uma modificação da Regra Delta. Além disso, possui a retro propagação do erro, que visa uma redução gradual do erro ao longo do treinamento.

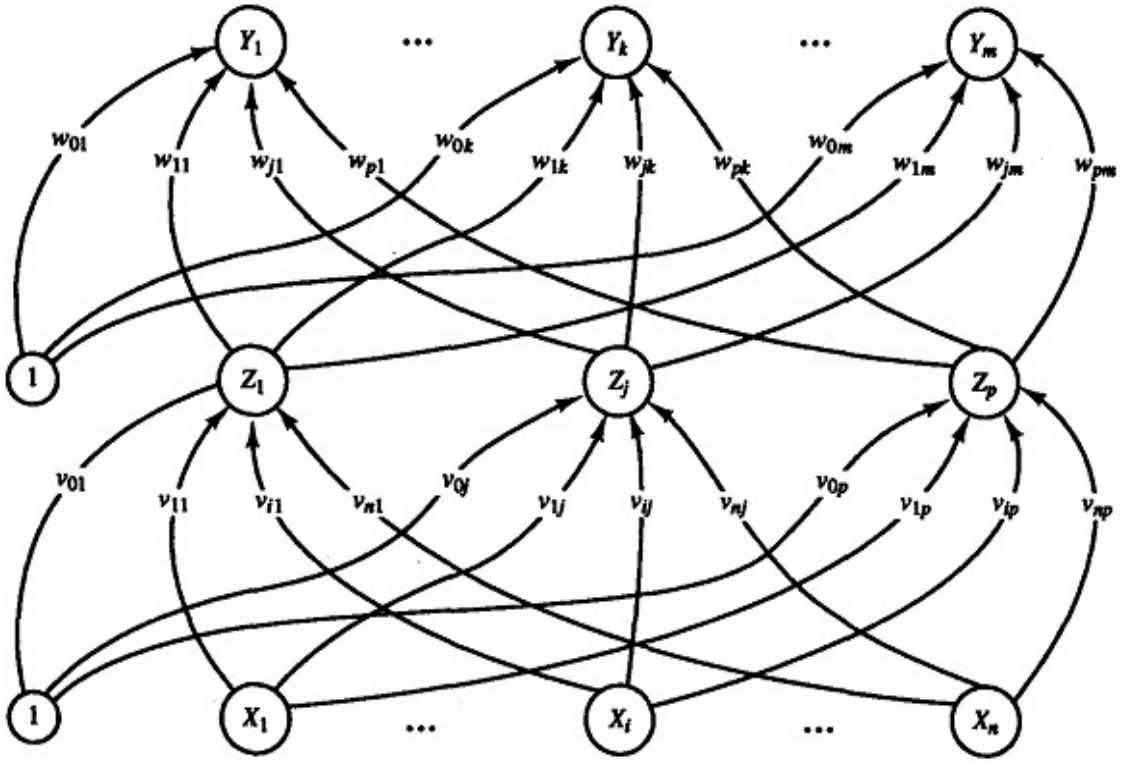

Figura 2.11 – Arquitetura de uma rede Backpropagation com uma camada intermediária de três neurônios. (FAUSSET, 1994)

Treinamento e Aplicação

Treinar uma rede com Backpropagation pode ser uma tarefa lenta mas, uma vez treinada, os resultados da rede são produzidos rapidamente. Para realizar o treinamento temos 3 estágios: Feedforward (Alimentação); Backpropagation (Retro Propagação); Ajuste de Pesos.

Na Alimentação, cada neurônio da primeira camada recebe seu valor de entrada e reproduz seu resultado para todos neurônios da camada seguinte, de acordo com sua função de ativação. Os neurônios das camadas intermediárias seguem o mesmo comportamento, e propagam o resultado da rede através de todas camadas existentes até os neurônios de saída.

Na Retro Propagação, é calculado o erro obtido entre o resultado da etapa de alimentação e o valor correto esperado. Esse erro é então retro propagado para toda a rede de acordo com um fator específico.

No Ajuste de Pesos, o erro que foi retro propagado é então utilizado para se calcular o ajuste que deve ser feito no peso de cada uma das conexões. Com as correções calculadas, todos os pesos das conexões da rede são atualizados simultaneamente.

A aplicação da rede, por sua vez, se torna tarefa simples após o treinamento. Com os valores finais dos pesos das conexões, basta aplicarmos a primeira etapa do treinamento (Feedforward) com os valores de entrada desejados.

Técnica de Aprendizado

A técnica de aprendizado Backpropagation está apresentada a seguir. Sua estrutura é a mais complexa e avançada presente nesse trabalho. Também é importante citar que o algoritmo mostrado a seguir foi utilizado para se montar a parte prática desse trabalho.

As equações utilizadas no algoritmo mostrado abaixo são descritas a seguir. A nomenclatura utilizada por ser vista na Figura 2.11, para uma melhor ilustração de sua representação. Uma explicação do que cada equação representa é dada em sequência.

$$Y_{in} = \sum_i x_i \cdot w_i \tag{2.18}$$

$$Y_{out} = f(Y_{in}) \tag{2.19}$$

$$\delta_z = (t - Z_{out}) \cdot f'(Z_{in}) \tag{2.20}$$

$$\Delta w_{yz} = \alpha \cdot Y_{out} \cdot \delta_z \tag{2.21}$$

$$\delta_{in} = \sum_i \delta_i \cdot w_i \tag{2.22}$$

$$\delta_y = \delta_{in} \cdot f'(Y_{in}) \tag{2.23}$$

$$\Delta w_{xy} = \alpha \cdot X_{out} \cdot \delta_y \tag{2.24}$$

$$w_{novo} = w_{velho} + \Delta w \tag{2.25}$$

A Equação 2.18 fornece o valor que chega a cada um dos neurônios, sendo o resultado da somatória dos valores fornecidos por cada conexão que "chega" no neurônio. O valor fornecido por cada conexão é o resultado do valor de seu neurônio de origem multiplicado pelo peso da conexão.

A Equação 2.19 fornece o valor que "sai" de cada neurônio. Tal valor é obtido ao se aplicar a função de ativação utilizada ao valor obtido na Equação 2.18.

A Equação 2.20 fornece o "fator de informação do erro" dos neurônios da última camada. Esse fator é obtido a partir da diferença entre o valor de saída esperado "t" e o valor de saída obtido "Z_{out}". Tal resultado é então multiplicado pelo valor de entrada do neurônio aplicado à derivada da função de ativação utilizada.

A Equação 2.21 fornece o "fator de correção do erro" das conexões que chegam aos neurônios da última camada. Esse valor é obtido através da multiplicação da taxa de aprendizado "α" com o valor de saída do neurônio anterior da conexão "Y_{out}" com o fator de informação do erro do neurônio posterior da conexão "δ_z".

A Equação 2.22 fornece o valor retro propagado de erro que "chega" ao neurônio intermediário. Esse valor é obtido fazendo a somatória da multiplicação do fator de informação do erro "δ_i" com o fator de correção do erro "w_i" através de todas as conexões posteriores ao neurônio.

A Equação 2.23 fornece o "fator de informação do erro" dos neurônios intermediários. Esse fator é obtido a partir da multiplicação do valor retro propagado de erro que "chega" ao neurônio "δ_{in}" e o valor de entrada do neurônio "Y_{in}" aplicado à derivada da função de ativação utilizada.

A Equação 2.24 fornece o "fator de correção do erro" das conexões que chegam aos neurônios intermediários. Esse valor é obtido através da multiplicação da taxa de aprendizado "α" com o valor de saída do neurônio anterior da conexão "X_{out}" com o fator de informação do erro do neurônio posterior da conexão "δ_y".

A Equação 2.25 fornece as diretrizes de atualização dos pesos de cada conexão. O valor do peso atualizado w_{novo} corresponde à soma do peso antigo w_{velho} com o fator de correção do erro da conexão.

Segue o algoritmo:

1. Inicializar pesos e bias

2. Enquanto a condição de parada não for atingida, repetem-se os passos 3-10.

    3. Para todos pares de entradas e saídas, repetem-se os passos 4-9

        (FeedForward - Alimentação)

    4. Os neurônios da primeira camada recebem o valor de entrada e os propagam para as camadas intermediárias.

    5. Os neurônios da segunda camada recebem os valores da camada anterior e calcula-se seus valores de entrada de acordo com a Equação 2.18.

    6. A saída dos neurônios da segunda camada é então calculada de acordo com sua função de ativação, conforme a Equação 2.19. O cálculo segue para todas as camadas intermediárias seguintes até que se compute a saída do sistema.

        (Backpropagation - Retro Propagação)

    7. Cada neurônio de saída recebe um fator de informação do erro, calculado de acordo com a Equação 2.20. Recebe também um fator de correção do erro, calculado

de acordo com a Equação 2.21.

8. Cada neurônio intermediário calcula seu valor de entrada do erro retro propagado de acordo com a Equação 2.22. Tal valor é multiplicado pela derivada de sua função de ativação para obter seu fator de informação de erro, conforme Equação 2.23. Seu fator de correção do erro é calculado de acordo com a Equação 2.24.

(Ajuste de Pesos)

9. Cada conexão tem seu peso atualizado, de acordo com a Equação 2.25.

10. Testar condição de parada

Mais detalhes sobre os parâmetros utilizados e as escolhas que devem ser feitas para o treinamento, dentre eles o número de NIs e a função de ativação utilizada, serão mostrados no capítulo de Materiais e Métodos.

2.2 Programação Orientada a Objetos

Para a realização desse trabalho, foi necessária a criação de um programa de computador capaz de realizar os cálculos determinados pelo algoritmo de Backpropagation. A Programação Orientada a Objetos (POO) foi escolhida como paradigma para se escrever o código. Nela, nos expressamos no código em termos de objetos, suas propriedades, métodos e estados.

Essa forma ou paradigma de programação foi escolhido por fornecer algumas vantagens essenciais necessárias ao longo do desenvolvimento do projeto. Entre elas podemos citar:

- Possui uma base conceitual no campo de estudo da cognição, ou seja, tenta representar o mundo da forma mais real possível. Da mesma forma, a Inteligência Artificial também busca representar a inteligência humana da forma mais real possível. Criando assim um vínculo conceitual entre ambas, o que facilita o desenvolvimento.

- Suas representações são facilmente visíveis no mundo real. Como todas as representações são feitas em forma de objetos, a compreensão dos mesmos se torna mais clara e perceptível, tanto para o programador quanto para um usuário.

- Fácil manutenção. A POO possui a capacidade de encapsulamento dos objetos, tornando-os entidades únicas, com propriedades e funções únicas. O encapsulamento no código reduz drasticamente a chance de que uma parte do sistema interfira no funcionamento de outra, reduzindo assim uma das mais comuns fontes de erros em programas.

- Facilidade de evolução. O encapsulamento fornece também modularidade, facilitando o desenvolvimento do código através de reuso e extensão de componentes (módulos) já existentes.

2.2.1 Java

Java é uma linguagem de programação desenvolvida pela Sun Microsystems em 1995. Existem atualmente inúmeras aplicações, em incontáveis plataformas, utilizando essa linguagem, que é definida pela sua desenvolvedora da seguinte forma: "Java é uma linguagem simples, orientada a objetos, distribuída, interpretada, robusta, segura, independente de arquitetura, portável, de alto desempenho, suportando multithreads e dinâmica."

Das características citadas acima, podemos destacar algumas vantagens que foram determinantes para que Java fosse escolhida como a linguagem de programação a ser usada nesse trabalho.

- Simplicidade: Java é considerada uma linguagem simples, com uma sintaxe baseada em C e C++, porém com diversas vantagens e melhorias em relação a essas.

- Orientada a Objetos: Utiliza o paradigma da POO, o que encaixa com o objetivo do projeto e permite uma analogia do mundo real muito mais clara e perceptível.

- Robusto: Por ser destinada para escrever programas que devem ser confiáveis numa variedade de maneiras, a linguagem Java coloca muita ênfase na verificação precoce de possíveis problemas, em checagens dinâmicas em tempo de execução e na eliminação de situações que são propensas a erros.

- Multithread: Multithreading é um modo de criação de aplicativos com vários segmentos ou processos em execução simultânea. Tal capacidade melhora a capacidade de resposta

interativa do sistema e seu comportamento em tempo real, resultando em uma velocidade de execução aprimorada.

2.3 Futebol

O futebol é um esporte de grande popularidade no mundo em geral, e principalmente no Brasil. Trata-se de um esporte coletivo com partidas que duram 90 minutos entre dois times de 11 jogadores cada.

Ele foi escolhido como tema desse trabalho por sua popularidade e por se tratar de um esporte considerado imprevisível.

Contudo, um certo padrão pode ser observado nos times mais vencedores. A quantidade de gols marcados e sofridos, o retrospecto jogando em "casa" e "fora de casa", a relação entre número de vitórias, empates e derrotas, entre outros. Nesse trabalho busca-se reconhecer esses padrões e, a partir deles, determinar com certa assertividade o resultado de jogos futuros.

2.3.1 Campeonato Brasileiro de Futebol

O Brasileirão, como também é conhecido o Campeonato Brasileiro de Futebol, é o principal torneio entre clubes de futebol do Brasil. O torneio foi iniciado em 1971 e ocorre anualmente desde então.

O campeonato historicamente não tinha uma padronização no sistema de disputa, o que gerava mudanças frequentes de regras e de número de participantes. Contudo, a partir de 2003 o sistema de pontos corridos foi adotado e a partir de 2006 o número de times participantes foi reduzido a 20. Esse formato de pontos corridos com 20 times foi determinado como o "formato definitivo" pela Confederação Brasileira de Futebol (CBF) e assim vem sendo utilizado ao longo dos anos.

O formato da competição é determinado de forma que durante o decorrer da temporada (de maio a dezembro), cada clube joga duas vezes contra os outros (em um sistema de pontos corridos), uma vez em seu estádio e a outra no de seu adversário, em um total de 38 jogos.

As equipes recebem três pontos por vitória e um por empate. Não são atribuídos pontos para derrotas.

As equipes são classificadas pelo total de pontos, depois pelo saldo de gols e, em seguida, pelos gols marcados. Em caso de empate entre dois ou mais clubes, os critérios de desempate são os seguintes: maior número de vitórias; maior saldo de gols; maior número de gols pró; confronto direto; menor número de cartões vermelhos recebidos; menor número de cartões amarelos recebidos.

O Campeonato Brasileiro de Futebol foi escolhido como tema de estudo por algumas de suas características, entre as quais podemos destacar:

- Longa duração. Por se tratar de um campeonato longo, é possível obter uma maior quantidade de informações para serem usadas no treinamento e/ou aplicação da Rede Neural Artificial, resultando em uma aplicação mais eficiente e confiável.

- Pontos corridos. O sistema de classificação em pontos permite uma melhor comparação entre as performances que os times vêm apresentando.

- Duas rodadas. Além de aumentar a quantidade de jogos para análise, a dupla rodada permite adicionarmos mais um parâmetro de entrada, sendo esse a condição de partida "em casa" ou "fora de casa".

- Vários Times. A quantidade de times permite diversas opções de confrontos, o que eleva as oportunidades de análise e aprendizado que a Rede Neural realiza.

- Proximidade e familiaridade. Por ser um campeonato de conhecimento geral, com regras simples e conhecidas, e fontes de dados facilmente acessíveis, o Campeonato Brasileiro de Futebol se mostra como um campo de estudo "sem mistérios".

- Sistema de disputa padronizado. Desde 2006 o formato do campeonato é o mesmo, sem previsões de alterações, o que facilita a extensão da abordagem de um ano para o outro e a comparação dos resultados entre anos diferentes.

CAPÍTULO III

MATERIAIS E MÉTODOS

Neste capítulo será apresentado todo o material (software) utilizado e o desenvolvimento do projeto, que foi dividido em cinco partes. A primeira foi o desenvolvimento do programa de computador responsável por executar a RNA. A segunda parte consiste da criação do banco de dados a ser utilizado para treinar e rodar a RNA. Na terceira parte temos as escolhas relativas ao funcionamento da RNA em si. A quarta parte consiste nas estratégias utilizadas para se analisar o resultado fornecido pela RNA. Por fim, na quinta parte temos a criação de uma Interface Gráfica do Utilizador (Graphical User Interface - GUI) para melhor utilização e experimentação do programa.

3.1 Desenvolvimento do programa

Para a criação do programa, foi escolhida a linguagem de Programação Orientada a Objetos (POO) Java. A aplicação desenvolvida foi nomeada Soccer Wizard e será assim referenciada na sequencia desse trabalho.

3.1.1 IDEs

Existem diversos Ambientes de Desenvolvimento Integrado (Integrated Development Environment - IDE) especializados em Java, que são programas utilizados para a criação de

códigos, que visam facilitar o desenvolvimento de aplicações.

Para esse trabalho foram escolhidas duas IDEs. Eclipse, desenvolvido pela Eclipse Foundation, e Netbeans, desenvolvido pela Oracle Corporation. Ambas foram usadas em conjunto para desenvolver partes distintas do código com funções diferentes entre si.

O Eclipse (Figura 3.1) foi escolhido devido à prévia experiência pessoal, às facilidades e comodidades apresentadas durante a escrita do código, e à extensa gama de plugins existentes.

Figura 3.1 – Eclipse Logo

O Netbeans (Figura 3.2) foi escolhido por fornecer uma ferramenta de desenvolvimento de GUIs muito versátil, completa e fácil de utilizar. Sem tal ferramenta seria praticamente impossível criar uma GUI completa, utilizável e visualmente agradável.

Figura 3.2 – Netbeans Logo

3.1.2 Plugins

Apesar de ambas IDEs serem bastante completas, alguns complementos foram necessários. Os plugins utilizados no projeto são citados a seguir.

Testar e debugar o código escrito é tarefa fundamental em qualquer desenvolvimento de software. Para essa função foi utilizado o JUnit. O JUnit é um framework de código aberto, criado por Erich Gamma e Kent Beck, com suporte à criação de testes automatizados na linguagem de programação Java.

O recurso de importações de dados de arquivos gerados no Excel também se mostrou essencial. Isso se deve ao fato de que o banco de dados a ser utilizado na rede neural artificial seria montado em uma planilha Excel. Para isso, foi utilizado o Java Excel API. Essa API (Interface de Programação de Aplicações) permite ao usuário de Java ler, escrever e modificar dados de planilhas Excel dinamicamente.

Para que o código gerasse os gráficos necessários, também foi necessário um complemento à IDE. O framework de código aberto JFreeChart foi escolhido. Seu intuito é facilitar a exibição de gráficos de alta qualidade em aplicações Java.

3.1.3 Sistema de Controle de Versão

Visando também uma evolução saudável do código foi utilizado um sistema controle de versão. Como o programa se tratava de um software a ser desenvolvido em várias etapas e consequentemente possuindo diversas versões ao longo de seu desenvolvimento, tal sistema de versionamento é essencial para que informações não se percam e que a evolução do código seja bem controlada. Pela facilidade de uso, velocidade, gratuidade e experiência prévia do usuário, foi escolhido o sistema de versionamento GitHub (Figura 3.3).

Figura 3.3 – GitHub Logo

3.1.4 Organização de Classes e Pacotes

Para se criar o programa, aproveitou-se das vantagens e facilidades oferecidas pela linguagem POO Java. Dessa forma, buscou-se uma estrutura que representasse entidades em forma de objetos encapsulados em blocos modularizados.

Tal organização visou a montagem de uma base sólida, onde possíveis pequenos erros

em trechos escondidos de código não passassem desapercebidos e se propagassem ao longo do desenvolvimento, prejudicando o programa como um todo.

Da mesma forma, tal organização facilita o desenvolvimento da aplicação, pois módulos já completos podem ser reaproveitados em trechos futuros do código. Assim, com o projeto terminado, a estrutura de pacotes e classes ficou como mostrado na Figura 3.4.

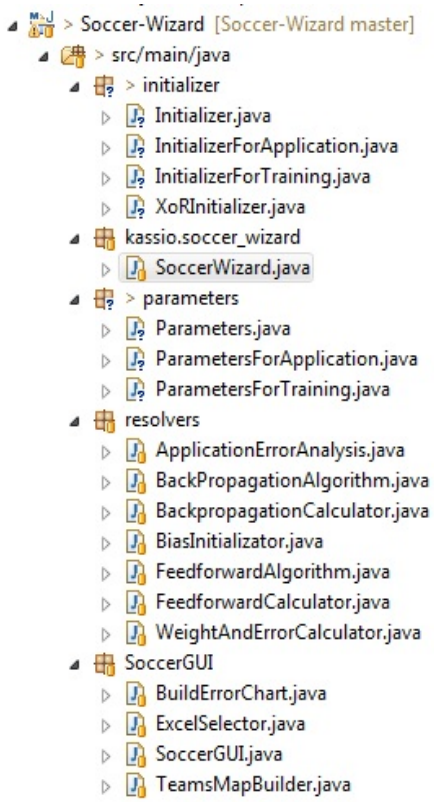

Figura 3.4 – Estruturação de Pacotes e Classes no Soccer Wizard

Cada um dos 5 pacotes finais possuem as classes responsáveis por um pedaço da aplicação correspondente.

No pacote "kassio.soccer_wizard" temos a classe principal, que é responsável por todo funcionamento do programa.

O pacote "initializer" é responsável pela importação dos dados do excel e pela correta configuração dos padrões de entrada para uso do programa. Ou seja, inicializa os dados que serão usados no programa.

O pacote "parameters" é responsável por incorporar na execução do programa os parâmetros informados pelo usuário.

O pacote "resolvers" compreende o algoritmo em si da RNA. Realiza todas as contas

e fornece todos os resultados com base nos dados obtidos do "initializer" e do "parameters".

Já o pacote "SoccerGUI" compreende a GUI do programa e todo o código que envolva a mesma.

3.2 Base de dados

Um banco de dados é necessário para que a rede neural obtenha os dados utilizados durante o seu treinamento e execução. Para criar esse banco de dados foi escolhido o editor de planilhas Microsoft Office Excel. Tal editor foi escolhido pelo seu grande e difundido uso, pela facilidade de criação e edição de aplicações, pela possibilidade de integração dos dados com Java, pelos diversos recursos de ajuda online e pela experiência prévia do usuário com o mesmo.

3.2.1 Organização

Para formular um banco de dados de fácil entendimento e boa manutenabilidade ficou decidido que o mesmo seria organizado da seguinte maneira:

- Uma aba da planilha é destinada para os resultados não tratados de todos os jogos do campeonato, denominada "Resultados" (Figura 3.5).
- Uma aba da planilha é destinada para auxiliar a composição das fórmulas utilizadas nas demais planilhas, denominada "Referências" (Figura 3.6).
- Cada um dos times do campeonato possui sua própria aba, com seus resultados históricos importados automaticamente da aba de resultados (Figura 3.7).
- Em cada aba de time, é destinada uma linha da planilha para cada rodada do campeonato. Nessa linha são apresentados o histórico do time pré-rodada e o resultado da mesma.

Essa organização traz diversas vantagens.

Com a concentração dos resultados em apenas uma aba, podemos facilmente verificar a consistência do banco de dados, encontrar um resultado específico, checar possíveis falhas de

	1ª Rodada							2ª Rodada							3ª Rodada						
	Home Team		x		Away Team			Home Team		x		Away Team			Home Team		x		Away Team		
	19	Time 19	1	x	0	Time 16	16	2	Time 2	2	x	2	Time 8	8	4	Time 4	2	x	1	Time 8	8
	20	Time 20	2	x	2	Time 13	13	4	Time 4	2	x	1	Time 17	17	2	Time 2	2	x	2	Time 9	9
	5	Time 5	1	x	1	Time 4	4	18	Time 18	5	x	1	Time 19	19	17	Time 17	1	x	1	Time 12	12
	12	Time 12	2	x	0	Time 14	14	9	Time 9	0	x	2	Time 15	15	20	Time 20	2	x	0	Time 19	19
	15	Time 15	0	x	2	Time 18	18	3	Time 3	0	x	0	Time 6	6	11	Time 11	1	x	1	Time 6	6
	7	Time 7	3	x	1	Time 3	3	14	Time 14	0	x	3	Time 20	20	5	Time 5	1	x	0	Time 15	15
	17	Time 17	0	x	0	Time 9	9	11	Time 11	1	x	1	Time 5	5	10	Time 10	3	x	0	Time 7	7
	10	Time 10	2	x	1	Time 2	2	13	Time 13	2	x	0	Time 7	7	13	Time 13	1	x	2	Time 3	3
	8	Time 8	5	x	0	Time 11	11	1	Time 1	2	x	0	Time 12	12	14	Time 14	2	x	2	Time 16	16
	6	Time 6	2	x	1	Time 1	1	16	Time 16	2	x	1	Time 10	10	1	Time 1	0	x	0	Time 18	18

Figura 3.5 – Trecho da aba "Resultados"

informação no banco de dados, assim como verificar visualmente de forma rápida se os resultados importados pelo Soccer Wizard estão corretos.

Outra vantagem de existir apenas uma aba com os resultados das partidas é a facilidade de criação de novos bancos de dados de campeonatos diferentes, que não necessariamente possuem a mesma quantidade de times ou rodadas.

Nesse ponto se torna de grande valia a aba "Referências". Nela, as referências para as células de "Resultados" ficam organizadas de forma ordenada e organizada, de forma que as células das abas dos times não precisem ser alteradas quando um novo banco de dados (outro campeonato, por exemplo) é inserido. Dessa forma, o banco de dados se torna robusto e versátil, pois as células mais importantes e detalhistas (as que são importadas para uso do Soccer Wizard) nunca precisam ser alteradas.

Rodada	Start Column	Start Cell 1	Final Cell 1	Full Range 1	Column Range 1	Start Cell 2	Final Cell 2	Full Range 2
1	1	$A$1	$A$11	Resultados!$A$1:$A$11	Resultados!$A$1:$A$11	$B$1	$B$11	Resultados!$A$1:$B$11
2	8	$H$1	$H$11	Resultados!$H$1:$H$11	Resultados!$H$1:$H$11	$I$1	$I$11	Resultados!$H$1:$I$11
3	15	$O$1	$O$11	Resultados!$O$1:$O$11	Resultados!$O$1:$O$11	$P$1	$P$11	Resultados!$O$1:$P$11
4	22	$V$1	$V$11	Resultados!$V$1:$V$11	Resultados!$V$1:$V$11	$W$1	$W$11	Resultados!$V$1:$W$11
5	29	$AC$1	$AC$11	Resultados!$AC$1:$AC$11	Resultados!$AC$1:$AC$11	$AD$1	$AD$11	Resultados!$AC$1:$AD$11
6	36	$AJ$1	$AJ$11	Resultados!$AJ$1:$AJ$11	Resultados!$AJ$1:$AJ$11	$AK$1	$AK$11	Resultados!$AJ$1:$AK$11
7	43	$AQ$1	$AQ$11	Resultados!$AQ$1:$AQ$11	Resultados!$AQ$1:$AQ$11	$AR$1	$AR$11	Resultados!$AQ$1:$AR$11
8	50	$AX$1	$AX$11	Resultados!$AX$1:$AX$11	Resultados!$AX$1:$AX$11	$AY$1	$AY$11	Resultados!$AX$1:$AY$11
9	57	$BE$1	$BE$11	Resultados!$BE$1:$BE$11	Resultados!$BE$1:$BE$11	$BF$1	$BF$11	Resultados!$BE$1:$BF$11
10	64	$BL$1	$BL$11	Resultados!$BL$1:$BL$11	Resultados!$BL$1:$BL$11	$BM$1	$BM$11	Resultados!$BL$1:$BM$11
11	71	$BS$1	$BS$11	Resultados!$BS$1:$BS$11	Resultados!$BS$1:$BS$11	$BT$1	$BT$11	Resultados!$BS$1:$BT$11

Figura 3.6 – Trecho da aba "Referências"

Por fim, a organização dos times de forma que cada um possua sua aba e que cada linha represente uma rodada é invariavelmente necessária quando pensamos na integração necessária

com o Soccer Wizard. Tal disposição dos dados facilita a busca das informações de forma automática e iterativa, principalmente quando se definem números para os times, para serem usados no lugar de seus nomes. O uso de números facilita a busca dos times em loops de código e garante de forma simples que as informações sejam buscadas nas abas corretas.

3.2.2 Parâmetros de entrada

Buscando encontrar a melhor forma de representar os dados para a rede neural funcionar, decidiu-se testar duas formas distintas. A primeira, mais simples, considera apenas o fator campo, e foi inspirada pela publicação "A Comparative Study on Neural Network based Soccer Result Prediction", de Burak Galip Aslan e Mustafa Murat Inceoglu (ASLAN e INCEOGLU, 2007). A segunda leva em conta uma maior quantidade de parâmetros, que foram escolhidos por se acreditar que suas informações eram relevantes para a determinação do resultado da partida.

Primeira representação (simples)

Buscando uma forma mais simples e concisa de se representar as condições de entrada da rede neural artificial, chegou-se a representação da Figura 3.7. Aqui, existem apenas duas condições:

- Taxa de aproveitamento em casa do time mandante.
- Taxa de aproveitamento fora de casa do time visitante.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Time 1															
2	Rodada	Time	Home/Away	Index	Adversário	GP	GC	Result	Win	Draw	Lose	Home Rating	Away Rating	Home Normalized	Away Normalized	Home/Away
3	1	1	Away	5	4	2	2	Draw	0	1	0	0	0	0	0	0
4	2	1	Home	11	6	5	2	Win	1	0	0	0	0	0	0	0
5	3	1	Away	11	16	2	1	Win	1	0	0	1	0	0.058823529	0	0
6	4	1	Home	11	9	2	0	Win	1	0	0	1	1	0.058823529	0.058823529	0.058823529
7	5	1	Away	9	12	1	3	Lose	0	0	1	2	1	0.117647059	0.058823529	0.058823529
8	6	1	Home	9	17	1	1	Draw	0	1	0	2	0	0.117647059	0	0.117647059
9	7	1	Away	11	7	4	2	Win	1	0	0	2	0	0.117647059	0	0
10	8	1	Home	11	3	0	0	Draw	0	1	0	2	1	0.117647059	0.058823529	0.117647059
11	9	1	Away	11	15	0	1	Lose	0	0	1	2	1	0.117647059	0.058823529	0.058823529
12	10	1	Home	8	13	0	0	Draw	0	1	0	2	0	0.117647059	0	0.117647059

Figura 3.7 – Trecho de uma aba de "time", mostrando a primeira representação de dados

A taxa de aproveitamento em casa/fora de casa é provavelmente o fator mais relevante dentre os que serão citados na segunda representação. Por isso, foi escolhido como foco nessa primeira representação.

Para se calcular a taxa de aproveitamento utiliza-se o seguinte racional:

- Todos times começam o campeonato com taxas iguais a 0.
- Após uma vitória em casa/fora a taxa de aproveitamento em casa/fora é acrescida de uma unidade.
- Após um empate em casa/fora a taxa de aproveitamento em casa/fora não é alterada.
- Após uma derrota em casa/fora a taxa de aproveitamento em casa/fora é reduzida em uma unidade.

Essa primeira forma de representação reduz drasticamente a quantidade de cálculos necessários para se treinar a rede neural artificial, se comparada com a segunda representação (apresentada a seguir). Da mesma forma a análise das informações é bem mais simples e direta, pois o número de entradas utilizadas no Soccer Wizard é de apenas duas, uma para o time mandante do jogo e uma para o time visitante.

Nessa representação, se perde o fator histórico recente. A taxa de aproveitamento é construída ao longo de todo o campeonato, desde seu início. Dessa forma, a análise deixa de ser prioritariamente sobre o retrospecto recente dos times e passa a ser mais abrangente temporalmente. Ou seja, resultados de rodadas muito antigos e de rodadas muito recentes tem igual peso sobre a taxa de aprendizagem.

Dessa forma focamos mais na estabilidade e constância do time ao longo do campeonato ao invés de seu momento atual.

Segunda representação (completa)

A segunda forma de se representar os dados para análise levou em conta fatores que geralmente são utilizados para fazer análises de performance e aproveitamento dos times. Tais informações são constantemente exibidas juntamente com as estatísticas do campeonato em sites e jornais esportivos. As informações escolhidas foram:

- Aproveitamento de pontos dos últimos 5 jogos.
- Mando de campo.
- Aproveitamento de pontos dos últimos 3 jogos dentro/fora de casa.
- Média de gols marcados nos últimos 5 jogos.
- Média de gols sofridos nos últimos 5 jogos.
- Classificação do time no campeonato.

O aproveitamento de pontos dos últimos 5 jogos foi escolhido pois representa o momento atual do time. Uma sequência de bons resultados pode representar um bom momento do time, uma reação ou um momento de decisão do campeonato onde resultados são necessários. Uma análise análoga pode ser feita para resultados ruins.

Mando de campo representa se o time irá jogar em casa ou fora de casa. Tal fator é vastamente reconhecido como sendo influente no resultado de uma partida.

Seguindo a ideia do mando de campo, o aproveitamento de pontos dos últimos 3 jogos dentro/fora de casa mostra o retrospecto recente do time jogando em casa ou fora de casa. Caso o jogo seja em casa, utiliza-se o retrospecto dos jogos correspondentes. Da mesma forma se faz com jogos fora de casa.

O resultado de uma partida é definido pela quantidade de gols marcados por ambos os times. Assim, a média de gols marcados e sofridos por ambos os times também é um indicador influente sobre o resultado da partida.

A classificação do campeonato entra como um fator menos momentâneo e mais abrangente temporalmente, pois indica a eficiência do time desde o início do campeonato. Contudo, da mesma forma, é influente sobre o resultado das partidas, visto que times mais bem classificados exibem resultados melhores ao longo do campeonato.

É importante ressaltar que nenhum fator atua sozinho. Todos foram escolhidos pois acredita-se que cada um desempenha um papel importante na determinação do resultado de uma partida. Por mais que um indicador qualquer aponte tendência de vitória, a rede neural artificial irá tratar e analisar todo o conjunto de dados na busca pela predição correta do resultado da partida.

Por se tratar de um campeonato longo em que os times passam por períodos de alta e baixa eficiência, uma análise dos dados em um período mais curto e recente é mais fiel ao momento atual de cada time e, assim, pode representar melhor as chances do mesmo em um próximo confronto.

Outro ponto de destaque é o fato que cada um dos fatores citados é utilizado duas vezes. Uma para o time mandante da partida e outra para o time visitante, formando assim um conjunto de doze informações que serão usadas como entrada (input) do Soccer Wizard.

Um ponto negativo dessa forma de representação se encontra na quantidade de informações de entrada (12, ao invés de 2, como na representação anterior). Tal número dificulta a análise dos resultados e a correlação de determinada informação com a capacidade da rede de realizar a predição correta.

3.2.3 Normalização dos dados

Conforme mostrado no capítulo de Fundamentação Teórica, os algoritmos utilizados em redes neurais artificiais geralmente possuem como entrada valores bivalentes (bipolares ou binários). O algoritmo de backpropagation utilizado nesse trabalho segue a mesma tendência binária, entradas com valor 1 para reforço positivo e com valor -1 para reforço negativo. Contudo, o backpropagation também aceita entradas intermediárias entre -1 e 1, para reforços que não são totalmente positivos e nem totalmente negativos.

Dessa forma, foi necessária uma normalização dos dados de forma que o Soccer Wizard fosse capaz de tratar as informações encontradas no banco de dados.

Para realizar a normalização, definimos um valor máximo e mínimo, representando o maior reforço positivo e o maior reforço negativo respectivamente. O maior reforço positivo recebe o valor 1. O maior reforço negativo recebe o valor -1. Para quaisquer valores entre o máximo e o mínimo, encontramos o valor normalizado correspondente ao realizar uma regressão linear através da reta que passa pelos valores máximo e mínimo.

As informações da primeira e segunda representação são normalizadas segundo a Tabela 3.1, onde também pode ser visto um exemplo.

Tabela 3.1 – Normalização dos dados de entrada

			Exemplo	
Entrada	Valor mínimo (-1)	Valor máximo (1)	Resultado	Valor Normalizado
Aproveitamento últimos 5 jogos	0 pontos	15 pontos	10 pontos	0.33
Mando de campo	Fora	Casa	Casa	1
Aproveitamento últimos 3 jogos dentro/fora	0 pontos	9 pontos	3 pontos	-0.33
Média de gols marcados 5 jogos	0 gols	3 gols	2.25 gols	0.5
Média de gols sofridos 5 jogos	3 gols	0 gols	1.5 gols	0
Classificação do time	Último	Primeiro	Oitavo	0.2 (total de 20 times)
Taxa de aproveitamento casa/fora	-(Número de rodadas/2)	+(Número de rodadas/2)	2	0.1 (total de 20 rodadas)

3.3 Parâmetros de treinamento

Uma RNA precisa de várias informações para realizar sua tarefa de forma ideal. Desde dados de entrada até parâmetros de controle de loop, são diversas as variáveis que influenciam no correto funcionamento da RNA. A escolha dessas informações, a coerência dos dados de entrada e parâmetros adaptados para a aplicação que se está utilizando são fatores fundamentais para o sucesso do projeto.

3.3.1 Quantidade de padrões de entrada

Em 1989, BAUM e HAUSSLER chegaram em uma relação que exprime a exatidão esperada da aplicação, e, de acordo com o número de padrões de entrada disponíveis, P, e o número de pesos a serem treinados, W. Tal relação está mostrada na Equação 3.1.

$$W/P = e \tag{3.1}$$

Para exemplificar, uma RNA com 20 pesos ($W = 20$) precisará de 200 padrões de entrada ($P = 200$) para que se tenha segurança que irá classificar 90% ($e = 0,1$) dos padrões de entrada corretamente.

É bom lembrar que essa relação é apenas uma referência, pois o resultado real da rede depende de inúmeros outros fatores. Além disso, o problema real está em fazer com que a RNA classifique corretamente padrões alheios ao conjunto de treinamento.

Como o conjunto de neurônios de entrada e saída do Soccer Wizard é relativamente extenso, utilizamos um campeonato inteiro para treinamento, ou seja, 620 padrões de entrada.

Da mesma forma, utilizamos outro campeonato inteiro para testar a eficácia do treinamento realizado. Nesse trabalho utilizamos o Campeonato Brasileiro de 2012 e 2013, respectivamente.

Essa escolha visa melhorar a eficiência do treinamento, tendo como referência a relação da equação 22. Além disso, o uso de campeonatos de diferentes anos mostra a capacidade da rede de reconhecer padrões alheios ao conjunto de dados de entrada.

3.3.2 Representação do dados

As formas binária e bipolar são as mais comuns para se representar os dados em uma RNA. Contudo, como já foi dito, a forma bipolar possui limitações que podem fazer com que a rede não reconheça certos tipos de padrões ou que faça com que o treinamento gaste mais iterações para ser realizado.

Os dados utilizados nessa aplicação também não são, em sua maioria, discretos. Ou seja, seus valores são variáveis contínuos entre um valor máximo e um valor mínimo. Podemos citar como exemplo a média de gols feitos. Tal valor varia entre $+3$ e 0, não possuindo intervalos discretos.

Dito isso, a escolha da forma binária se torna clara para o problema apresentado. Além de não apresentar limitações claras, permite que possamos escolher valores contínuos entre -1 e $+1$, representando respectivamente reforço negativo e positivo. Da mesma forma, podemos representar um reforço neutro (empate, por exemplo) como uma entrada nula, de valor 0.

3.3.3 Aleatoriedade dos parâmetros de entrada

Para se treinar a rede, utilizamos os 620 padrões contidos em um campeonato (já excluindo 5 rodadas iniciais e 2 rodadas finais).

Por se tratar de um campeonato longo, podem ocorrer períodos em que haja algum tipo de tendência ou viés. Caso o treinamento fosse executado sequencialmente da primeira para a última rodada (que é forma mais simples de montar o algoritmo), tais tendências poderiam ter um impacto negativo no processo de treinamento, piorando a eficiência do mesmo.

Visando evitar tendências e melhorar como um todo o processo de treinamento, os

padrões de entrada foram misturados de forma que o treinamento não fosse feito sequencialmente, mas sim aleatoriamente. Ou seja, a iteração entre as entradas do sistema não obedece nenhuma ondem específica.

Para se embaralhar o conjunto de entradas, foi usada uma função geradora de números pseudo-aleatórios, própria do Java.

3.3.4 Função de ativação

A função de ativação, que é utilizada para se fornecer o resultado de saída um neurônio em função do seu valor de entrada, deve ter algumas características importantes em uma RNA de backpropagation. Essas características garantem que o treinamento ocorra de forma correta, rápida, eficiente e com menos peso computacional.

Uma função de ativação precisa ser contínua, diferenciável e monotonicamente não decrescente (não pode inverter sua relação de ordem). Esses pontos são necessários para o correto treinamento.

Para rapidez e eficiência do treinamento, é desejado que a função de ativação sature. Por exemplo, se aproximando de valores finitos máximos e mínimos assintóticamente.

Por fim, para redução do peso computacional, uma função com derivada fácil de se computar também é desejada. Geralmente usa-se funções cujos valores das derivadas podem ser expressos em termos da própria função.

Uma função que une todas essas exigências dentro do alcance dos nossos dados (-1, 1), é a Função Sigmoidal Bipolar. Essa função está expressa na Equação 3.2, sua derivada na Equação 3.3 e sua ilustração está na Figura 3.8.

$$f(x) = \frac{2}{1 + e^{-x}} - 1 \tag{3.2}$$

$$f'(x) = \frac{1}{2} \cdot [1 + f(x)] \cdot [1 - f(x)] \tag{3.3}$$

Por atender todos os requerimentos, a Função Sigmoidal Bipolar é a função de ativação utilizada nesse trabalho.

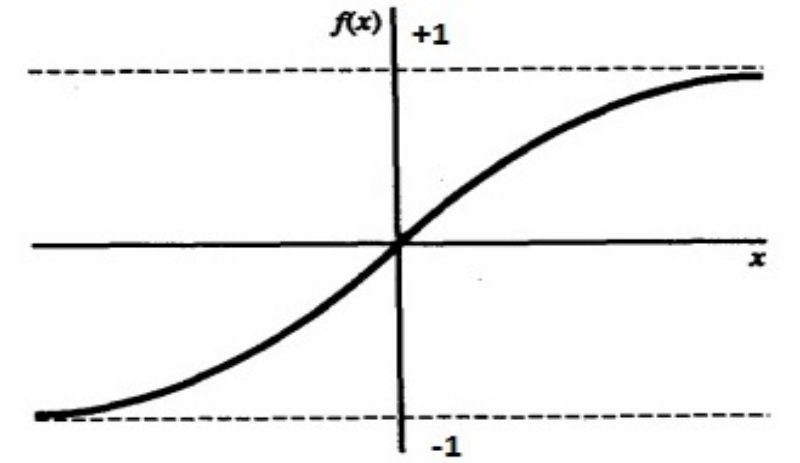

Figura 3.8 – Função Bipolar Sigmoidal

3.3.5 Inicialização de pesos e bias

A escolha dos pesos iniciais e bias (que é o peso de uma entrada constante igual a 1), tem fundamental importância quando levamos em conta a velocidade de convergência do erro da RNA.

A atualização dos pesos depende tanto da função de ativação utilizada (Bipolar Sigmoidal) quanto da derivada da mesma. Dessa forma, é conveniente escolher pesos que reduzam as possibilidades de que esses valores sejam nulos, ou aproximadamente nulos. Um procedimento comum é simplesmente inicializar os valores dos pesos entre -0.5 e $+0.5$ aleatoriamente.

Existem modificações dessa inicialização dos pesos. Em 1990, NGUYEN e WIDROW desenvolveram uma simples modificação que geralmente fornece um aprendizado muito mais rápido. A análise feita por eles é baseada em uma função de ativação bem próxima da função bipolar sigmoidal utilizada nesse trabalho.

Basicamente, a modificação fornecida por eles começa inicializando os pesos da mesma forma: aleatoriamente entre -0.5 e $+$ 0.5. Com esses pesos definidos, utiliza-se o número de neurônios iniciais e intermediários para se encontrar um fator que, juntamente com os pesos aleatórios iniciais, definem quais serão os pesos iniciais definitivos.

A inicialização de pesos de Nguyen e Widrow é utilizada neste trabalho.

3.3.6 Taxa de aprendizado

A taxa de aprendizado "α" é um importante fator durante o treinamento da RNA e a escolha de um valor adequado requer cuidado. Um valor muito pequeno pode fazer com que a rede seja treinada muito lentamente, e de forma oposta, um valor muito grande faz com que o treinamento possa não convergir.

De acordo com (HECHT-NIELSEN, 1990), um limite superior para seu valor pode ser determinado a partir do maior autovalor da matrix de correlação R (Equação 3.4) dos vetores de entrada $x(p)$, conforme Equação 3.5.

$$R = \frac{1}{P} \cdot \sum_{P=1}^{P} x(p)^T \cdot x(p) \tag{3.4}$$

$$\alpha < \frac{maiorautovalordeR}{2} \tag{3.5}$$

Contudo, por simplicidade, é mais comum escolher um valor pequeno ($\alpha < 1$) para a taxa de aprendizado. Caso o treinamento não convirja, esse valor deve ser reduzido. Caso o treinamento esteja muito lento, esse valor pode ser aumentado para se tentar obter uma melhor velocidade.

Nesse trabalho foi determinado, experimentalmente, que uma taxa de aprendizado entre $\alpha = 0.01$ e $\alpha = 0.02$ se mostrou apropriada para a aplicação, fazendo com que houvesse convergência e da mesma forma oferecendo uma boa velocidade de treinamento.

3.3.7 Quantidade de neurônios intermediários

Se olharmos os resultados da RNA de forma gráfica, podemos determinar sua fronteira de decisão, conforme dito na Subseção 2.1.3. Isso acontece caso os dados de entrada possam ser delimitados por um número finito de linhas ou planos. Dessa forma, é possível construir uma rede com $2P$ NIs que irão aprender perfeitamente P padrões de entrada bipolares. No caso apresentado nesse trabalho, temos 620 padrões de entrada bipolares.

Caso um aprendizado perfeito fosse desejado, devido ao grande número de padrões de entrada aqui utilizados, a RNA teria um número de interconexões e um tempo de treinamento

desnecessariamente grandes.

Contudo, o objetivo final da RNA aqui montada não é realizar esse aprendizado perfeito, mas sim saber generalizar e fornecer a resposta mais adequada para quaisquer tipos de padrões entrada (e não apenas os fornecidos para treinamento). Além disso, essa generalização necessária não seria particularmente boa com uma quantidade tão alta de NIs.

De qualquer forma, $2P$ nos dá uma ideia do limite superior de NIs a se considerar.

De forma a buscar o balanço mais adequado entre aprendizado perfeito e generalização, vários testes foram feitos, com diferentes quantidades de NIs. As quantidades mais promissoras foram utilizadas no estudo desse projeto.

3.3.8 Duração do treinamento

Geralmente, para RNAs usando backpropagation, existem duas condições possíveis para se terminar um treinamento, por valor do erro e por número de iterações.

No programa criado para esse trabalho existem essas duas opções.

Contudo, durante o desenvolvimento do mesmo, observou-se que terminar o treinamento com base no erro obtido ao final das iterações era pouco eficiente e muitas vezes difícil de se controlar com qualidade. Tal condição reduz a confiabilidade dos resultados da rede, pois é difícil saber se o treinamento foi finalizado quando deveria.

Por sua vez, determinar o fim do treinamento pelo número de iterações realizadas se mostrou muito mais simples, estável e confiável.

O erro médio ao longo das iterações tende a cair quanto mais se treina a rede, com uma tendência de se estabilizar após um certo número de iterações. Além disso, o treinamento pode ser considerado rápido, pois não dura mais do que alguns minutos.

Com esse conjunto de fatores, podemos simplesmente finalizar o treinamento após um número significativamente grande de iterações (10000, por exemplo). Assim teremos uma rede bem treinada (erro praticamente estabilizado) sem grande demora no treinamento e sem dificuldades para analisar qual valor limitador devemos escolher.

3.3.9 Remoção de padrões de entrada não representativos

O Campeonato Brasileiro de Futebol, assim como qualquer disputa que envolve pessoas e diversos fatores externos, possui resultados inesperados e que fogem ao padrão conhecido. Com isso em mente, fica claro que alguns padrões de entrada utilizados não são confiáveis e representativos do comportamento do restante da amostra.

Apesar disso, não é possível excluir tais padrões indesejados pré-treinamento por não ser possível identifica-los.

Visando contornar esse problema, a seguinte estratégia foi utilizada:

1. O treinamento ocorre normalmente, com todos padrões de entrada.
2. Com a rede treinada, verifica-se quais foram os erros resultantes de cada um dos padrões.
3. Os piores resultados (5% do total) são excluídos do conjunto de treinamento.
4. A rede é treinada uma segunda vez, com o novo conjunto de treinamento.
5. Com a rede treinada novamente, verifica-se quais foram os erros resultantes de cada um dos padrões.
6. Os piores resultados (5% do total) são excluídos do novo conjunto de treinamento.
7. A rede é treinada uma terceira e última vez, com o mais novo conjunto de treinamento.

Dessa forma os padrões de entrada que mais se afastaram do comportamento da amostra são excluídos. O treinamento final da rede, portanto, não inclui tais padrões de entrada indesejados.

A decisão de se excluir 10% dos resultados, assim como a separação dessa operação em duas etapas de 5% cada, foi puramente experimental. Várias quantidades de exclusão diferentes e várias formas de se realizar esse procedimento foram testadas. A estratégia apresentada foi a que se mostrou mais efetiva no resultado final da RNA.

3.4 Estratégias de análise pós processamento

Após o treinamento, a RNA nos fornece um conjunto de pesos das conexões. Com os pesos determinados, executamos a rede fornecendo novos dados de entrada. O resultado então pode ser entendido como o valor de saída dos neurônios da última camada. Para se retirar alguma informação válida desse conjunto de dados de saída é preciso entende-lo e analisa-lo.

O resultado da RNA consiste em um conjunto de três neurônios (representando vitória, empate e derrota), onde cada um pode ter seu valor entre -1 e 1. A partir desses valores, determinamos qual foi o resultado fornecido pela rede. As estratégias utilizadas nesse trabalho para determinar esses resultados e a validade dos mesmos estão explicadas na sequência.

3.4.1 Determinação do resultado

A configuração de três neurônios de saída, representando cada um vitória, empate e derrota, nos permite analisar a resposta da rede para cada possível resultado independentemente dos outros. Ou seja, podemos entender o resultado de cada neurônio de saída como a possibilidade que tal resultado correspondente venha a ocorrer dado certo conjunto de entradas.

Nos neurônios de entrada, ao se configurar reforços positivos como $+1$ e reforços negativos como -1, determinamos que os resultados de saída serão analisados da mesma forma. Assim, quanto mais próximo de $+1$ for o resultado do neurônio de saída, maior será o reforço positivo do mesmo. Por exemplo, caso o neurônio representante de vitória tenha resposta $+0.93$ (próximo de $+1$) e o neurônio representante de derrota tenha resposta -0.81 (próximo de -1), entendemos que para esse conjunto de entradas, existe um reforço muito positivo para que o resultado seja vitória e um reforço muito negativo para que o resultado seja derrota.

Como os neurônios são independentes entre si, a rede pode fornecer conjuntos de saída de todas formas possíveis, o que pode dificultar a análise dos resultados. Alguns exemplos que podem gerar confusão na hora de analisar os resultados estão mostrados na Tabela 3.2.

Como mostrado na Tabela 3.2, podem existir respostas onde a simples análise da magnitude do reforço dos neurônios não seja suficiente para se determinar o resultado da rede. Resultados onde todos os reforços sejam negativos ou positivos, ou simplesmente todos reforços

Tabela 3.2 – Possíveis resultados dos neurônios de saída

	Conjunto de respostas						
Neurônio	1	2	3	4	5	6	7
Vitória	0	-0,5	0,93	-1	0,97	-0,12	-0,99
Empate	0,1	-0,7	0,92	0,03	-0,5	0,01	-0,95
Derrota	0,05	-0,8	0,91	0,02	1	-0,13	-0,999

sejam muito parecidos entre si podem levar a um mal entendimento do resultado da mesma.

Contudo, cada conjunto de saída deve fornecer uma, e somente uma, resposta para o conjunto de entradas analisado. Apesar do valor do neurônio indicar a magnitude do reforço para tal resultado, e do fato de que cada neurônio ser independente dos demais, podemos estabelecer uma relação entre eles. Independentemente dos valores, sempre haverá um neurônio com resultado maior que os demais, e por menor que seja seu reforço, ainda será maior que os outros.

Para se determinar o resultado da rede, decidiu-se então, a partir dessa análise, que o maior resultado seria tomado como a resposta final da rede, independentemente do valor do neurônio. Ou seja, o neurônio de saída com maior valor determinará se o resultado será vitória, empate ou derrota.

3.4.2 Validação de resultados

Apesar da estratégia de determinação do resultado mostrada na Subseção 3.4.1 ser eficiente, ainda não é suficiente para uma predição dos resultados otimizada. Com isso, uma segunda estratégia foi adotada, complementar à primeira.

Nessa estratégia, buscamos validar os resultados obtidos inicialmente, excluindo resultados não entendidos como confiáveis.

Para se chegar em uma estratégia eficiente, utilizamos um racional aqui chamado de "probabilidade do resultado". Esse racional é calculado para cada neurônio de saída de acordo com a Equação 3.6.

$$P_i = \frac{1 + n_i}{(1 + n_1) + (1 + n_2) + (1 + n_3)} \tag{3.6}$$

Na Equação 3.6, P_i é a probabilidade do resultado do neurônio i. Já n_1, n_2 e n_3 são os resultados dos 3 neurônios de saída. Soma-se 1 no resultado de cada neurônio para que todos valores sejam necessariamente positivos.

Durante a aplicação da rede, a probabilidade do resultado de cada um dos neurônios de cada um dos padrões de entrada é calculada. Da mesma forma, calcula-se a média e o desvio padrão das probabilidades de cada um dos 3 neurônios de saída ao longo de todos padrões de entrada. Com esses valores podemos determinar a confiabilidade dos resultados.

Para que um resultado seja considerado válido ele deve passar por duas condições, mostradas na Equação 3.7 e na Equação 3.8, cujas variáveis são explicadas na sequência.

$$P_M \geq P_L \tag{3.7}$$

$$P_M \geq \bar{P}_i + \sigma_L \cdot \sigma_i \tag{3.8}$$

P_M é a probabilidade do neurônio com maior valor de saída. P_L é a probabilidade limiar, determinada pelo usuário. $\bar{P}_i$ é a média das probabilidades do neurônio i ao longo de todos padrões de entrada. σ_L é o desvio padrão limiar, determinado pelo usuário. σ_i é o desvio padrão das probabilidades do neurônio i ao longo de todos padrões de entrada.

A condição mostrada na Equação 3.7 determina se a probabilidade do neurônio analisado está acima de determinado valor P_L fornecido pelo usuário. Essa condição visa excluir resultados onde não exista uma clara definição de qual neurônio possui maior reforço positivo.

A condição mostrada na Equação 3.8 visa eliminar resultados tendenciados pela RNA. Muitas vezes as condições de treinamento favorecem um ou outro neurônio, fornecendo para o mesmo, valores de saída maiores (em média). Essa condição confere a média e o desvio padrão da probabilidade do neurônio ao longo de todos padrões de entrada utilizados na aplicação. Um resultado só é considerado válido se sua probabilidade for maior que a probabilidade média acrescida de uma quantidade σ_L(determinada pelo usuário) de desvios padrão.

Caso um resultado se aplique às duas condições acima, o mesmo é aceito como válido.

3.5 Interface Gráfica do Usuário

Ao se criar um programa, a forma de se executá-lo pode ser, por vezes, bastante complexa e pouco intuitiva para quem não participou do desenvolvimento do mesmo. Dessa forma, a criação de uma GUI se faz necessária para o bom uso do programa e visualização de seus resultados.

Nesse projeto, a interface gráfica ficou dividida em uma janela com duas abas, correspondentes à parte de teste da RNA e da execução em si do Soccer Wizard. Uma visualização da janela, com foco na aba de testes está mostrada na Figura 3.9.

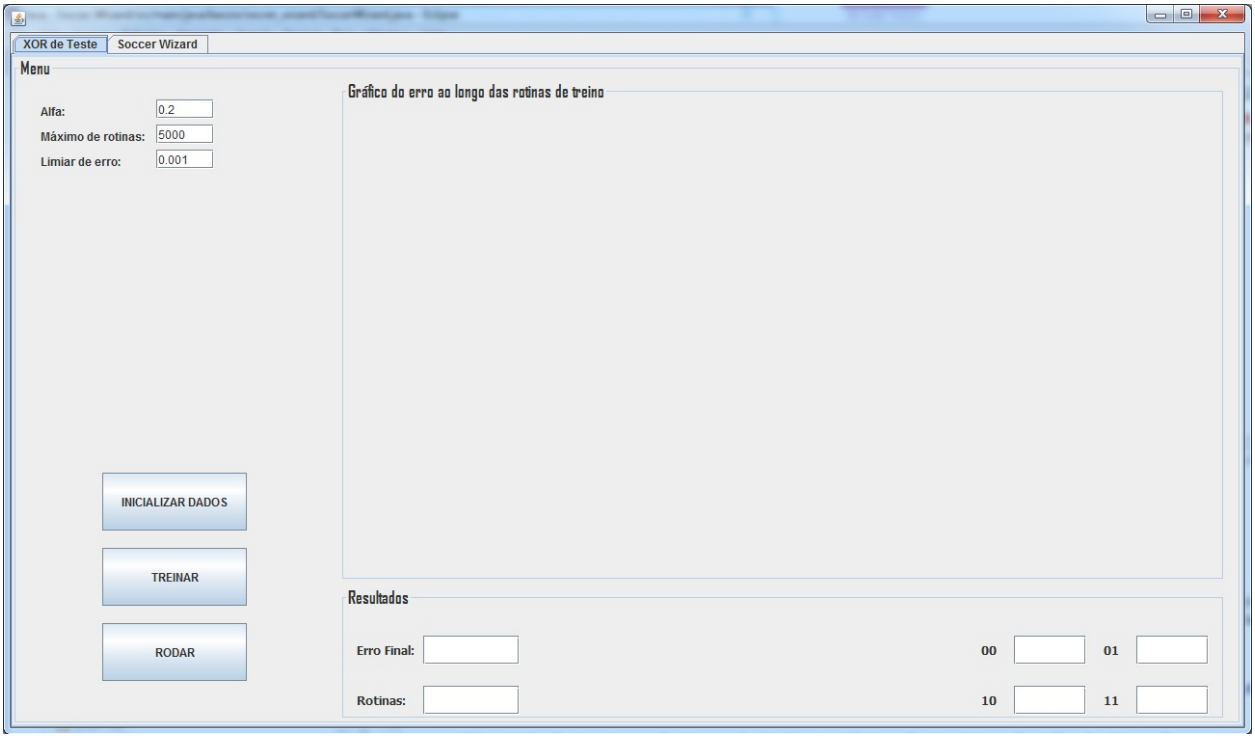

Figura 3.9 – Aba de testes da GUI

A função da GUI é fornecer todos comandos necessários para a execução do programa, permitir a entrada dos parâmetros utilizados e mostrar ao usuário todas as informações pertinentes a respeito da execução do mesmo.

Os componentes da GUI inclusos na aba de execução do Soccer Wizard (Figura 3.10) estão explicados na sequência.

1. Taxa de aprendizado utilizada no treinamento.

2. Número máximo de iterações executadas durante o treinamento.

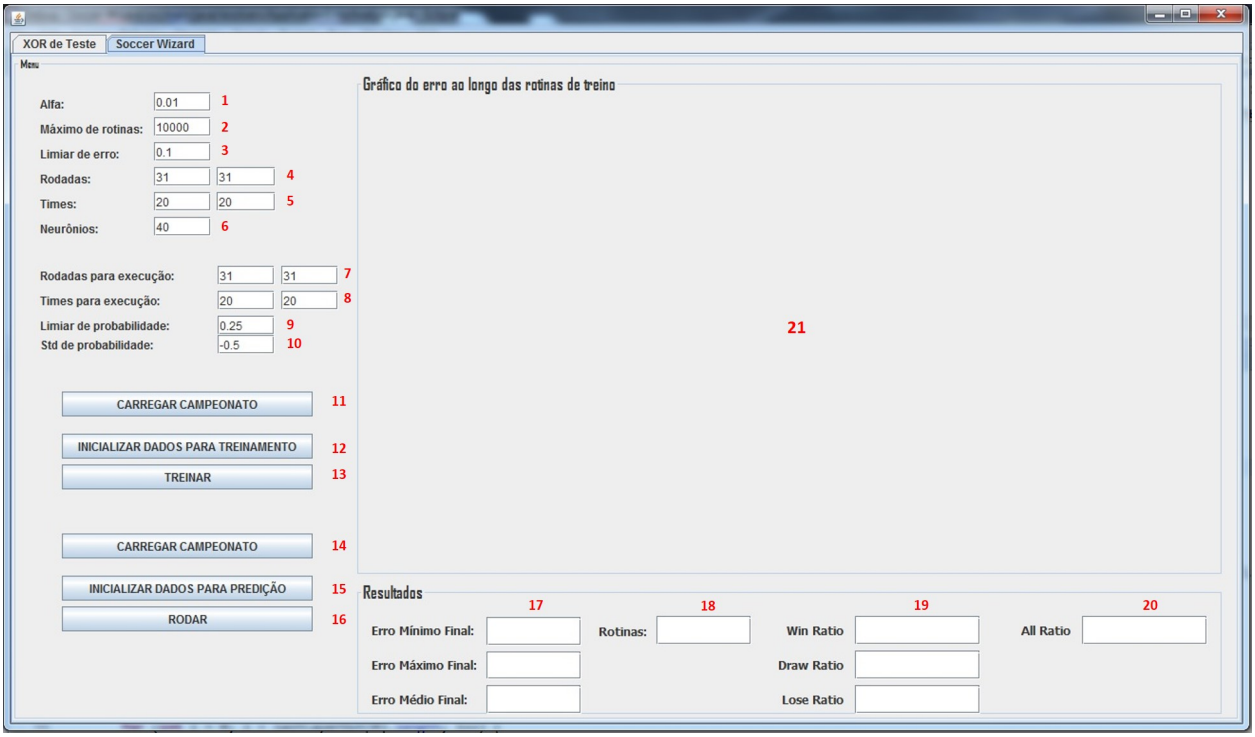

Figura 3.10 – Aba Soccer Wizard da GUI

3. Limiar do erro médio, abaixo do qual o treinamento é terminado.

4. Número de rodadas utilizadas no treinamento.

5. Número de times utilizados no treinamento (que jogam em cada uma das rodadas).

6. Número de neurônios na camada intermediária (NI).

7. Número de rodadas utilizadas na execução da RNA.

8. Número de times utilizados na execução da RNA (que jogam em cada uma das rodadas).

9. Parâmetro de usuário "Probabilidade Limiar" P_L.

10. Parâmetro de usuário "Desvio Padrão Limiar" $?_L$.

11. Procura e carrega no programa o campeonato desejado para treinamento.

12. Carrega os dados informados nos campos 1 a 6 no programa.

13. Executa o treinamento da RNA.

14. Procura e carrega no programa o campeonato desejado para execução da RNA treinada.

15. Carrega os dados informados nos campos 7 a 9 no programa.

16. Executa a RNA treinada.

17. Apresenta os valores do erro máximo, médio e mínimo, dentre todos padrões de entrada da última iteração do treinamento.

18. Apresenta o número de iterações executadas no treinamento.

19. Apresenta a taxa de acerto dentre resultados de vitória, empate e derrota.

20. Apresenta a taxa de acerto incluindo todos resultados.

21. Apresenta graficamente a evolução dos erros ao longo das iterações de treinamento.

Referências Bibliográficas

S. R. ALPERT, S. W. WOYAK, H. J. SHROBE, and L. F. ARROWOOD. Guest editor's introduction: Object-oriented programming in ai. *IEEE Expert: Intelligent Systems and Their Applications*, 5:6–7, 12 1990.

B. G. ASLAN and M. M. INCEOGLU. A comparative study on neural network based soccer result prediction. In *Seventh International Conference on Intelligent Systems Design and Applications*. IEE Computer Society, 2007.

E.B. BAUM and D. HAUSSLER. What size net gives valid generalization? Technical report, Computer Research Laboratory, University of California, 1988.

C. M. BISHOP. *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.

J. P. CHANGEUX and A. DANCHIN. Selective stabilization of developing synapses as a mechanism for the specification of neuronal networks. *Nature*, 264:705–712, 1976.

T. CHENG, D. CUI, Z. FAN, J. ZHOU, and S. LU. A new model to forecast the results of matches based on hybrid neural networks in the soccer rating system. In *Fifth International Conference on Computational Intelligence and Multimedia Application*. IEE Computer Society, 2003.

L. P. DA COSTA. *Atlas do esporte no Brasil: Atlas do esporte, educação física e atividades de saúde e lazer no Brasil*. Shape Editora e Promoções Ltda., 2005.

A. E. ELO and S. SLOAN. *The Rating of Chess Players, Past and Present*. Ishi Press International, 2008.

L. V. FAUSSET. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall, 1994.

A. M. FLITMAN. Towards probabilistic footy tipping: a hybrid approach utilising genetically defined neural networks and linear programming. *Computers & Operations Research*, 33:2003–2022, 7 2006.

S. S. HAYKIN. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999a.

S. S. HAYKIN. *Neural Networks and Learning Machines*. Prentice Hall, 1999b.

R. HECHT-NIELSEN. *Neurocomputing*. Addison-Wesley Publishing Company, 1990.

J. L. McCLELLAND and D. E. RUMELHART. *Explorations in Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1988.

W. S. MCCULLOTH and W. PITTS. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

A. MEHREZ and M. Y. HU. Predictors of the outcome of a soccer game - a normative analysis illustrated for the israeli soccer league. *Zeitschrift für Operations Research*, 42:361–372, 10 1995.

W. T. MINSKY and S. A. PAPERT. *Perceptrons, Expanded Edition*. Cambridge, MA: MIT Press, 1969.

D. NGUYEN and B. WIDROW. Improving the learning speed of two-layer neural networks by choosing initial values of the adaptive weights. In *International Joint Conference on Neural Networks*, volume 3, pages 21–26, 1990.

M. C. PURUCKER. Neural network quarterbacking: How different training methods perform in calling the games. *Potentials, IEEE*, 15:9–15, 8 1996.

F. ROSENBLATT. *Principles of Neurodynamics*. New York: Spartan, 1962.

C. F. SILVA, E. S. GARCIA, and E. SALIBY. Soccer championship analysis using monte carlo simulation. In *2002 Winter Simulation Conference*, volume 1-2, pages 2011–2016, 2002.

G. S. STENT. A physiological mechanism for hebb's postulate of learning. *Proceedings of the National Academy of Science of the U.S.A.*, 70:997–1001, 1973.

M. WATSON. *Practical Artificial Intelligence Programming With Java*. Mark Watson, 2008.

B. WIDROW and M. E. HOFF Jr. Adaptive switching circuits. In *IRE WESCON Convention Record*, volume 4, pages 96–104, 1960.