Subconjunto de Treinamento e Critério de Confiabilidade para Redes Neurais Artificiais de Domínio Real, Complexo e de Clifford

Thalles de Souza Torchi

Subconjunto de Treinamento e Critério de Confiabilidade para Redes Neurais Artificiais de Domínio Real, Complexo e de Clifford

Thalles de Souza Torchi

Orientador: ***Prof. Dr. Milton Romero Romero***

Dissertação apresentada ao Departamento de Engenharia Elétrica da Universidade Federal de Mato Grosso do Sul como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.

UFMS - Campo Grande
Maio/2009

Subconjunto de Treinamento e Critério de Confiabilidade para Redes Neurais Artificiais de Domínio Real, Complexo e de Clifford

Thalles de Souza Torchi

Dissertação de Mestrado submetida e aprovada pela banca examinadora designada pelo Colegiado do Programa de Mestrado em Engenharia Elétrica da Universidade Federal de Mato Grosso do Sul, como parte dos requisitos necessários à obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 29 de maio de 2009 por:

Milton Romero Romero - Dr.
Prof. DEL/UFMS - Orientador

Maria Bernadete Zanusso - Dra.
Profa. DCT/UFMS

Rúbia Mara de Oliveira Santos - Dra.
Profa. DMT/UFMS

Roldão da Rocha Junior - Dr.
Prof. CMCC/UFABC

Agradecimentos

Agradeço:

A **Deus** pela oportunidade da vida e por esta realização.

Aos meus amados pais, **Creusa Torchi** e **Christiano Torchi**, que me auxiliaram na revisão do vernáculo deste trabalho.

À minha querida irmã **Thalita Torchi**, que, mesmo à distância, vibrou pelo meu êxito.

À amorosa **Kátia Brandão Soares**, que soube suportar as minhas ausências por conta das horas dedicadas à dissertação.

Ao colega **Guilherme Torres de Alencar**, à época graduando de Engenharia Elétrica pela UFMS, que me emprestou a sua valiosa experiência e contribuição direta na discussão e implementação das RNA de domínio real e complexo.

A todos os colegas de trabalho do **Tribunal Regional Eleitoral de Mato Grosso do Sul**, em especial aos colegas da **Secretaria de Tecnologia da Informação** e **Coordenadoria de Infra-Estrutura e Suporte**, que sempre me apoiaram em minhas iniciativas e se dispuseram a ouvir minhas explanações a respeito do tema.

Aos professores **Me. Leonardo Souza Silva**, **Dr. José Roberto Zorzatto** e **Dr. Leandro Sauer** pelo incentivo e apoio inicial aos meus estudos.

Finalmente, agradeço aos professores **Dr. Milton Romero Romero** e **Dra. Luciana Cambraia Leite**, por terem me prestado genuína orientação, guiando-me na busca de respostas aos meus questionamentos.

"Faça as coisas o mais simples que você puder,
porém não as mais simples"

Albert Einstein

Resumo

A utilização de Redes Neurais Artificiais (RNA) *Multilayer Perceptrons* (MLP) *Backpropagation* (BP) para aproximação de funções compreende três fases: treinamento, validação e utilização. Este trabalho propõe uma metodologia para abordagem de dois aspectos sobre estas fases: 1ª) estimar um parâmetro para aferir a exatidão de saídas de RNA classificando quais podem ser consideradas confiáveis e quais não podem, pois na fase de utilização as RNA não conseguem aproximar a função de interesse para 100% dos dados; e 2ª) estabelecer o número mínimo de padrões a serem utilizados na fase de treinamento que permita a convergência e a generalização dessas redes. A metodologia propõe a utilização de duas redes: a RNA Direta (RNAD), utilizada para aproximar a função de interesse, e a RNA Inversa (RNAI), utilizada para aproximar a Função Inversa (FI) da RNAD. Após o treinamento conjunto das duas redes, a diferença entre a entrada da RNAD e a saída da RNAI, que devem ser computacionalmente iguais na convergência das redes, irá definir os parâmetros apresentados. Caso a função a ser aproximada não tenha FI definida, o domínio é restringido para onde exista. A metodologia proposta será demonstrada utilizando-se dados sintéticos a partir da função quadrática $f(x) = x^2$, com o objetivo de controlar as entradas e as saídas para demonstrar experimentalmente a validade da metodologia que será aplicada para as RNA MLP nos domínios real, complexo e de Clifford. O domínio (multidimensional) de Clifford é restringido para que seja isomorfo ao domínio complexo, permitindo a visualização gráfica dos resultados e a comparação com o domínio complexo.

Abstract

The use of Artificial Neural Networks (ANN) Multilayer Perceptrons (MLP) Backpropagation (BP) for regression is done in three phases: training, validation and testing. This research proposes a methodology to approach two aspects of these phases: 1st) to estimate a parameter to confront the accuracy of the ANN output classifying those which can be considered reliable and those which are not, therefore in the testing phase of the ANN it will not be possible to evaluate the function of interest for 100% of the data; and 2nd) to establish the minimum number of patterns which allow the convergence and the generalization of those networks to be used in the training phase. The method proposes the use of two networks: Direct ANN (DANN), used to approximate the function of interest, and Inverse ANN (IANN), used to approximate the Inverse Function (IF) of DANN. After the joint training of the two networks, the difference between the input of the DANN and the output of the IANN, should be the same computerwise in the convergence of the networks, it will define the parameters presented. In case the function to be approximated does not have a defined IF, the domain is restricted for where there is one. The method proposed will be demonstrated using synthetic data starting from the quadratic function $f(x) = x^2$, with the objective of controlling the input and the output to demonstrate the validity of the method that will be applied for the ANN MLP in the Real, Complex and Clifford domain. The Clifford (multidimensional) domain is restricted so that it is isomorphous to the complex domain, allowing the graphic visualization of the results and the comparison to the complex domain.

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO

1

Introdução

Redes Neurais Artificiais (RNA) podem ser descritas como técnicas computacionais baseadas em um modelo matemático inspirado na estrutura do neurônio biológico que tem capacidade de adquirir, armazenar e utilizar conhecimento experimental (Carvalho, 2007). As RNA do tipo *Multilayer Perceptron* (MLP) são difundidas como ferramentas para a solução de problemas de reconhecimento de padrões (Travessa, 2006), fluxo de carga (W. L. Chan e Lai, 2000), aproximação de funções (Arena et al., 1995) entre outras. Uma prática estatística comum para a escolha do conjunto de treinamento dessas redes é a validação cruzada estratificada conhecida também como método *hold-out* (Kohavi, 1995), a qual reserva em torno de 70% dos dados disponíveis para treinamento e o restante para validação. Isso tem como característica não desejável a admissão de subconjuntos de treinamento maiores ou menores que o necessário, o que incide diretamente nas fases de validação e utilização, pois as redes podem convergir de acordo com o critério de parada, mas não necessariamente generalizar a função.

É uma característica inerente às RNA, na fase de utilização, que, para alguns valores de entrada, a saída atual não se aproxima corretamente da função. Em certas aplicações é necessário utilizar unicamente saídas sobre as quais o grau de exatidão, aqui denominado confiabilidade, seja alto. Em caso contrário, é melhor não utilizar. Este trabalho apresenta uma metodologia determinística como alternativa ao método *hold-out* e outros métodos estatísticos (Uchimura et al., 1995), fornecendo para tal fim dois parâmetros: o primeiro, para definir a confiabilidade na exatidão das saídas de RNA; e o segundo, para definir a quantidade mínima de padrões de treinamento necessários para permitir a convergência e a generalização.

A metodologia propõe a utilização de duas redes: a RNA Direta (RNAD), utilizada para aproximar a função de interesse, e a RNA Inversa (RNAI), utilizada para aproximar a Função Inversa (FI) da RNAD. Após o treinamento conjunto das duas redes, a diferença entre a entrada da RNAD e a saída da RNAI, que devem ser computacionalmente iguais na convergência das

redes, irá definir os parâmetros apresentados. Caso a função a ser aproximada não tenha FI definida, o domínio é restringido para onde exista. A metodologia proposta será demonstrada utilizando-se dados sintéticos a partir da função quadrática $f(x) = x^2$, com o objetivo de controlar as entradas e as saídas para demonstrar experimentalmente a validade da metodologia que será aplicada para as RNA MLP nos domínios real, complexo e de Clifford. O domínio (multidimensional) de Clifford é restringido para que seja isomorfo ao domínio complexo, permitindo a visualização gráfica dos resultados e a comparação com o domínio complexo. As equações de Clifford neste artigo refletem esta restrição.

1.1 Organização do trabalho

Os conceitos necessários para a compreensão de RNA de forma geral são apresentados no capítulo 2 os quais são necessários para a extensão dessas redes para o domínio complexo, capítulo 3. No capítulo 4 são apresentados os conceitos e definições das álgebras de Clifford necessárias para a compreensão da estrutura e funcionamento das RNA de Clifford e seu respectivo *backpropagation* no capítulo 5. A descrição da metodologia proposta neste trabalho é apresentada no capítulo 6 e os resultados e as discussões dos mesmo se encontram nos capítulos 7 e 7.4, respectivamente, e o capítulo 8 é reservado para as conclusões.

CAPÍTULO

2

Redes Neurais Artificiais

Redes Neurais Artificiais (RNA) são estruturas lógico-matemáticas para modelar a maneira como o cérebro realiza uma tarefa particular ou função de interesse. Segundo (Haykin, 1994), uma RNA é um processador maciçamente paralelo distribuído, constituído de unidades de processamento simples (neurônios artificiais), que têm a propensão natural para armazenar conhecimentos experimentais e torná-los disponíveis para o uso, normalmente são implementadas, utilizando-se componentes eletrônicos, ou é simulada por programação em um computador. Essas redes assemelham-se ao funcionamento do cérebro humano em dois aspectos: 1ª) as conexões entre os neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar o conhecimento adquirido; 2ª) o conhecimento é adquirido por meio de dados do ambiente, num processo de aprendizagem também conhecido como processo de treinamento ou "Algoritmo de Aprendizagem", que tem como finalidade ajustar os pesos sinápticos da rede de uma forma ordenada para alcançar um objetivo desejado.

Uma RNA pode possuir uma ou múltiplas camadas. O número de camadas define a capacidade de representação das relações entre o espaço de entrada e de saída. A inexistência da camada intermediária, característica do modelo *Perceptrons*, que está condicionado, por concepção, a representar bem somente relações linearmente separáveis. A existência de camadas intermediárias, característica do modelo *Multilayer Perceptrons* (MLP), retira tal limitação. Se houver apenas uma camada intermediária, a rede do tipo MLP pode representar (com determinado grau de aproximação) qualquer função contínua (Cybenko, 1988). Os conceitos acima apresentados serão discutidos com mais detalhes nas próximas seções. Por enquanto, enfatizaremos o conceito do neurônio artificial que é o ponto de partida para compreender a estrutura das RNA.

2.1 O Neurônio Artificial

O primeiro modelo matemático de um neurônio artificial foi proposto por (McCulloch e Pitts, 1943). Neste modelo, o neurônio possui n entradas $x_0, x_1, x_2, \cdots, x_i, \cdots, x_n$ (equivalentes aos dendritos) e apenas uma saída o_j (equivalente ao axônio). Para simular a sinapse, cada entrada do neurônio está associada a um peso $w_{0j}, w_{1j}, w_{2j}, ..., w_{ij}$ e um peso fixo b_j chamado *bias*, cujos valores podem ser positivos (excitatórios) ou negativos (inibitórios). Temos na Figura (2.1) a representação gráfica desse modelo[1]. Os pesos têm a finalidade de armazenar o comportamento de funções, informação esta retirada dos dados de entrada fornecidos ao neurônio.

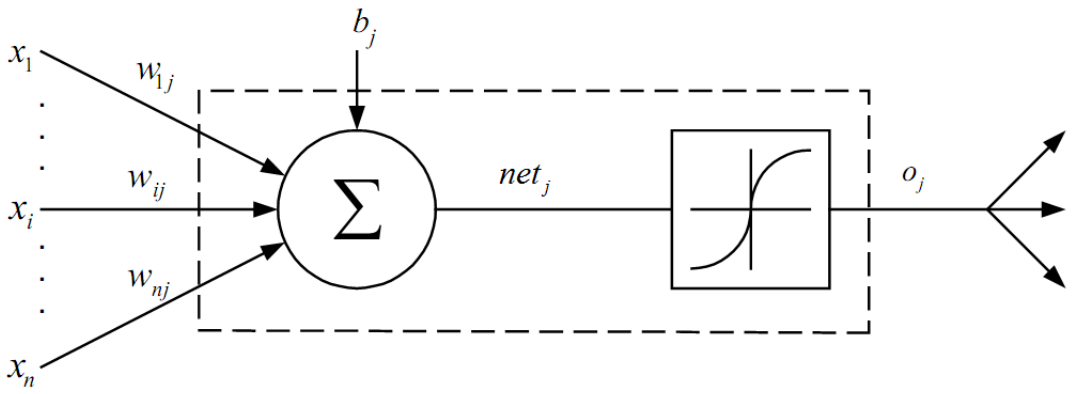


Figura 2.1: Neurônio artificial

$$net_j = \sum_i w_{ji} x_i + b_j \tag{2.1}$$

$$o_j = f(net_j) \tag{2.2}$$

O índice ji faz referência à conexão entre o j-ésimo neurônio a i-ésima entrada. Para esse modelo é comum a entrada x_0 ser fixada no valor 1, e seu peso correspondente $w_{0j} = b_j$ ser definido como *bias*. O corpo celular é emulado aplicando-se a Equação (2.1) resultando no valor net_j logo após este valor é submetido a uma função de ativação $f(\cdot)$ e a saída dessa função é representada por o_j como mostrado na Equação (2.2).

2.2 Funções de Ativação

As Funções de Ativação (FA) fornecem o valor da saída de um neurônio. Dependendo do valor de ativação net_j, existem vários tipos de funções de ativação. A escolha de uma ou

[1]Figura editada do original em (Yi, 2004)

outra depende de diversos fatores relacionados à aplicação que se quer resolver. As funções de ativação mais comuns em RNA são detalhadas a seguir.

2.2.1 Função Degrau

A Figura (2.2) mostra a função degrau. O eixo horizontal representa os valores de net_j e o eixo vertical representa os valores de o_j das Equações (2.1) e (2.2), respectivamente.

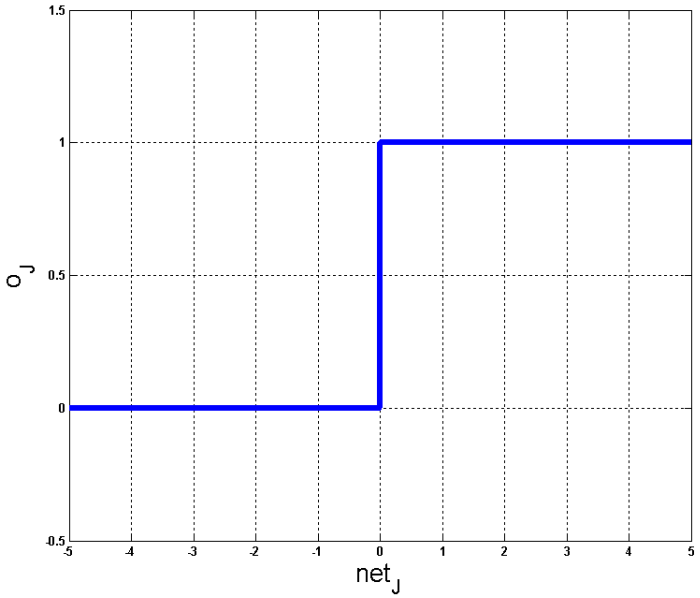


Figura 2.2: Função degrau

A função degrau descrita na Equação (2.3) é utilizada no modelo de (McCulloch e Pitts, 1943) e representa a generalização da característica binária de circuitos digitais.

$$o_j = f(net_j) = \begin{cases} 1, & \text{se } net_j \geq 0 \\ 0, & \text{se } net_j < 0 \end{cases} \tag{2.3}$$

pois só admite saídas, zero ou um. Por ter esta característica, a função não é utilizada no algoritmo *backpropagation* (que será discutido na seção 2.6) no processo de aprendizado, pois não é possível computar a derivada por toda a extensão de seu domínio.

2.2.2 Função Logística

Esta função, ao contrário da função degrau, pode assumir todos os valores entre zero e um. Esta função também é conhecida como logística e é definida pela Equação (2.4):

$$o_j = f(net_j) = \frac{1}{1 + e^{(-net_j)}} \tag{2.4}$$

A função logística mostrada na Figura (2.4) apresenta três características fundamentais para solução da maioria das aplicações práticas utilizando RNA: é continua, não linear e limitada entre zero e um. O fato de a função ser contínua permite calcular a derivada em qualquer

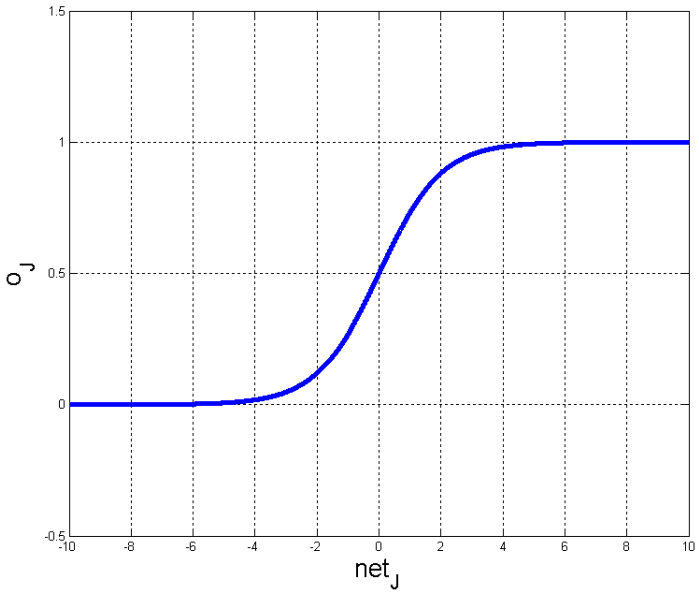


Figura 2.3: Função logística

ponto do seu domínio e, conseqüentemente, permite o processo de aprendizagem pelo algoritmo *backpropagation*. A não-linearidade permite o desenvolvimento de soluções para problemas também não lineares. E a característica de ser limitada nos garante que não importa se net_j assuma valores elevados ou ínfimos, pois o_j, por ser limitada entre entre zero e um, sempre fornecerá saídas limitadas.

2.2.3 Função Tangente Hiperbólica

A função tangente hiperbólica, definida pela Equação (2.5), possui as mesmas três características descritas na função logística descrita na Equação (2.4). No entanto, é possível observar na Figura (2.4) que essa FA possui os limites entre $[-1, 1]$.

$$o_j = f(net_j) = \frac{e^{net_j} - e^{-net_j}}{e^{net_j} + e^{-net_j}} \tag{2.5}$$

O comportamento da função tangente hiperbólica é muito semelhante ao da função logística, com a vantagem (dependendo do problema a ser abordado) de fornecer valores o_j negativos. Essa função é apresentada neste trabalho na forma de função elementar transcendental, função escrita com termos exponenciais, que permite a mestra representação matemática em diversos domínios.

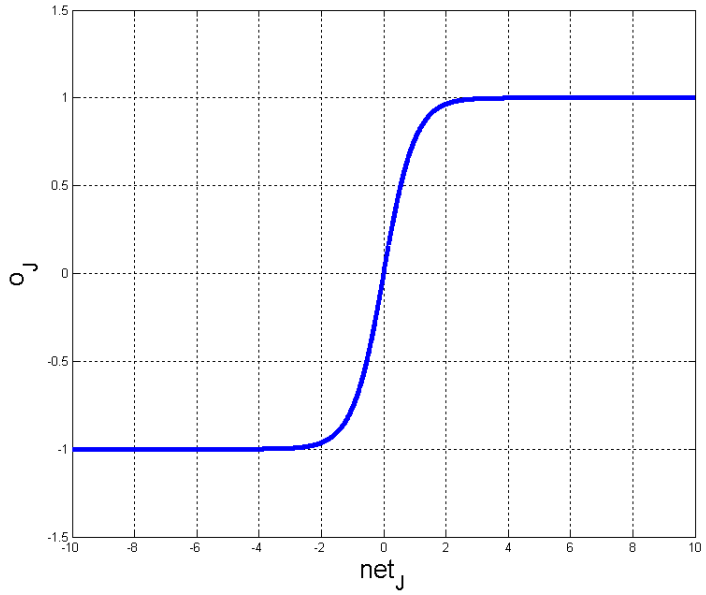


Figura 2.4: Função tangente hiperbólica

2.3 Perceptron

As Redes perceptron com uma camada são uma associação de vários neurônios de (McCulloch e Pitts, 1943) em paralelo e são consideradas o tipo mais simples de RNA. Esta rede é formada por uma camada única de neurônios de saída, os quais estão conectados por pesos w_{ij} às entradas x_i. Observe a Figura (2.5):

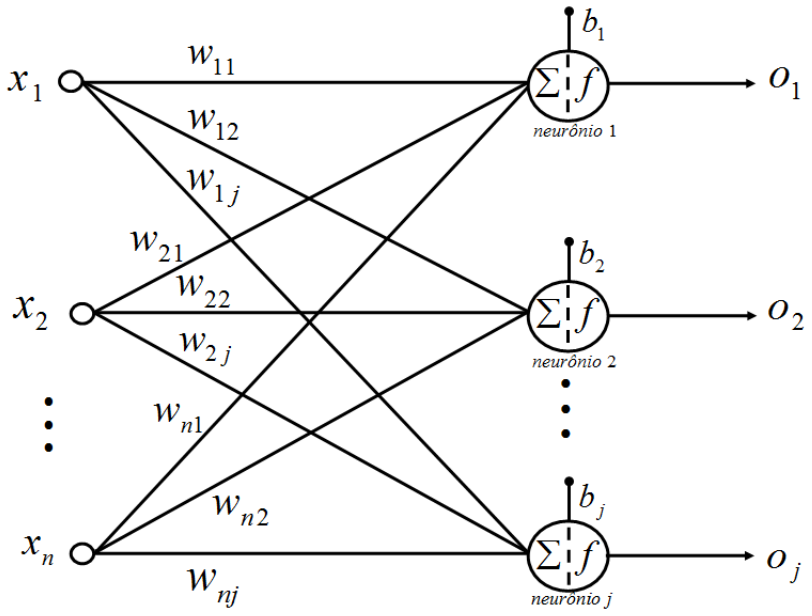


Figura 2.5: Perceptron

Cada neurônio computa o seu respectivo valor net_j, Equação (2.1), na figura representado por $\sum$. Imediatamente, net_j é aplicado na FA $f(\cdot)$, para obter a saída o_j. A Equação (2.6) abaixo resume este processo.

$$\begin{array}{rcccl} o_1 & = & f(w_{11}x_1 + w_{21}x_2 + ... + w_{i1}x_i + b_1) & = & f(net_1) \\ o_2 & = & f(w_{12}x_1 + w_{22}x_2 + ... + w_{i2}x_i + b_2) & = & f(net_2) \\ \vdots & & \vdots & & \vdots \\ o_j & = & f(w_{1j}x_1 + w_{2j}x_2 + ... + w_{ij}x_i + b_j) & = & f(net_j) \end{array} \tag{2.6}$$

De maneira a facilitar a compreensão e a implementação dos algoritmos, a Equação (2.6) será organizada na forma matricial, como mostra a Equação (2.7)

$$\begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_j \end{bmatrix} = f\left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1j} \\ w_{21} & w_{22} & \cdots & w_{2j} \\ \vdots & \vdots & \vdots & \vdots \\ w_{i1} & w_{i2} & \cdots & w_{ij} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_j \end{bmatrix} \right) \tag{2.7}$$

Observando com atenção a Equação (2.7), podemos constatar que ela nada mais é que a generalização da Equação (2.1). Assim, podendo reescrever a equação matricial como (2.8):

$$O = f(W^T X + B) \tag{2.8}$$

Em que O corresponde ao vetor de saídas da rede, f à FA, B ao vetor de *bias* e X ao vetor de entradas da rede neural e W^T, à matriz de pesos transposta.

2.4 Redes Multilayer Perceptrons

As redes neurais artificiais podem ser organizadas em mais de uma camada de neurônios, este tipo de rede é conhecido com *Multilayer Perceptrons* (MLP), Figura (2.6). Além das camadas de entrada e saída, este tipo de rede possui camadas ocultas, também chamadas de camadas intermediárias. As saídas de cada camada servem de entrada para a seguinte, e deve-se associar um peso para cada entrada em cada camada. A análise matemática de (Minsky e Papert, 1969) sobre o modelo *perceptron* provou que redes de uma camada não são capazes de solucionar problemas que não sejam linearmente separáveis. Tal restrição implica que padrões de entrada similares resultem em padrões de saída similares, o que leva o sistema à incapacidade de aprender certos mapeamentos. Dessa maneira, padrões de entrada, com estruturas similares, fornecidos do mundo externo, que levem a saídas diferentes, não são possíveis de serem mapeados por redes sem representações internas, isto é, sem camadas intermediárias.

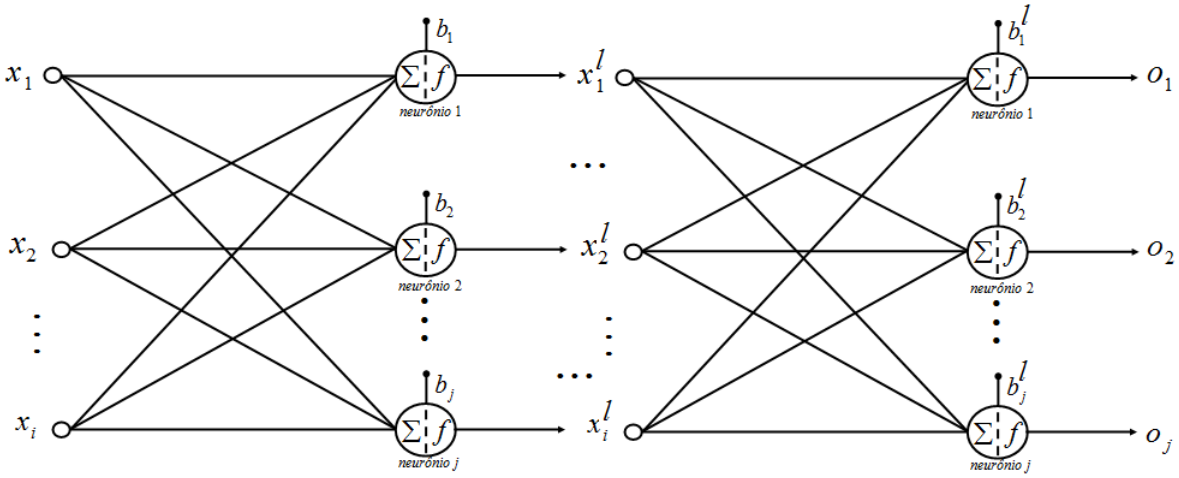


Figura 2.6: Rede MLP

Um exemplo clássico deste caso é a função ou-exclusivo (XOR). Contudo, o desenvolvimento do algoritmo de treinamento *backpropagation*, por (Rumelhart et al., 1986), mostrou que é possível treinar, eficientemente, redes com camadas intermediárias, resultando no modelo de redes neurais artificiais mais utilizado atualmente, as redes (MLP), treinadas com o algoritmo *backpropagation*. Nessas redes, cada camada tem uma função específica. A camada de saída recebe os estímulos da camada intermediária e constrói o padrão que será a resposta. As camadas intermediárias funcionam como extratoras de características. Seus pesos são uma codificação de características apresentadas nos padrões de entrada e permitem que a rede crie sua própria representação, mais rica e complexa, do problema. Todas estas afirmações são fundamentadas pelos trabalhos de (Cybenko, 1988) e (Cybenko, 1989) que demonstram que são necessárias apenas duas camadas intermediárias para a aproximação de qualquer função e uma camada intermediária para implementar qualquer função contínua.

2.5 Aprendizado Supervisionado

Para que uma RNA possa fornecer resultados convenientes a problemas específicos, é necessário que passe por uma fase de treinamento, em que seus pesos sinápticos, ou simplesmente pesos, são ajustados de forma que ela se adapte aos diferentes estímulos de entrada. A partir dessa constatação, podemos então afirmar que a propriedade mais importante de uma rede neural artificial é sua capacidade de aprendizado. Uma rede neural aprende através de um processo iterativo de ajustes aplicados aos seus pesos sinápticos e limiares, que pode ser expresso na forma de um algoritmo computacional. Não há uma definição precisa, universalmente aceita, de "aprendizado". No contexto de redes neurais artificiais, (Haykin, 1994) define aprendizado da seguinte forma: "Aprendizado é um processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estímulo pelo ambiente no qual a rede está

inserida". Um tipo particular de aprendizado será de especial interesse para este trabalho: o *aprendizado supervisionado*.

Este tipo de aprendizado é caracterizado pela presença de um "*professor*" externo. A função do "professor" durante o processo de aprendizado é suprir a rede neural com uma *resposta desejada* a um determinado estímulo apresentado pelo ambiente. Definimos um *sinal de erro* $e_j(n)$ como a diferença entre a resposta desejada e a resposta observada na saída da rede neural no instante n. Os parâmetros da rede são então ajustados de acordo com o sinal de erro $e_j(n)$. A Figura (2.7) apresenta um diagrama de blocos de um sistema com aprendizado supervisionado.

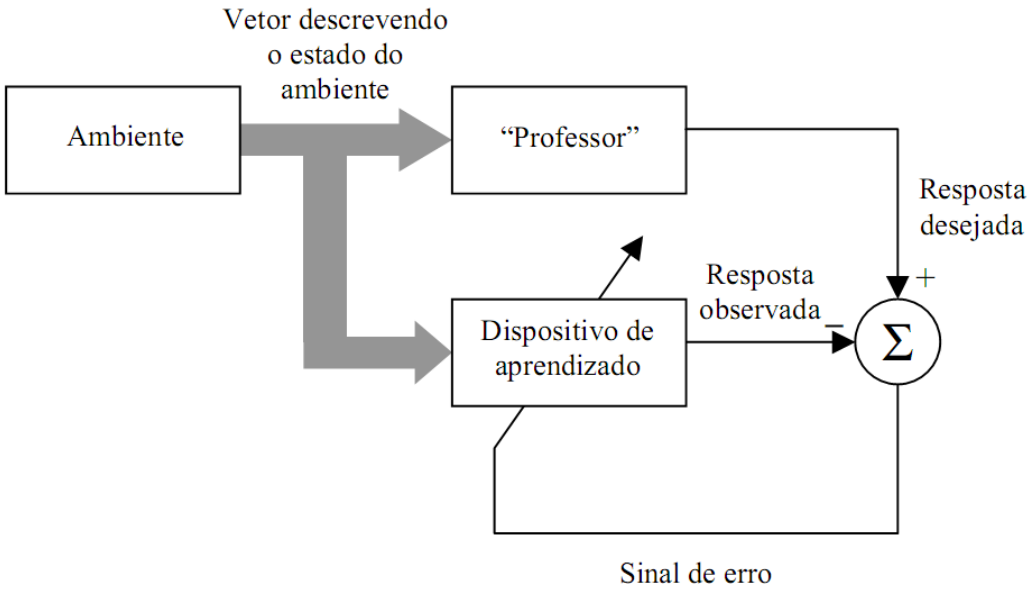


Figura 2.7: Diagrama de blocos: sistema com aprendizado supervisionado

Uma forma de implementar uma estratégia de aprendizado supervisionado em redes neurais é por meio de procedimentos iterativos de correção de erro. Durante o aprendizado, os erros vão sendo calculados sucessivamente, até que cheguem a um valor satisfatório, definido *a priori*. Seja $t_j(n)$ a resposta desejada para um neurônio j no instante n e seja $o_j(n)$ a resposta observada para este neurônio. A resposta $o_j(n)$ é produzida por um estímulo (vetor) $X(n)$ aplicado à entrada da rede da qual o neurônio j faz parte. O objetivo do aprendizado por correção de erro é minimizar alguma *função de custo* baseada no sinal de erro $e_j(n) = t_j(n) - o_j(n)$, de modo que a resposta observada de cada neurônio da rede se aproxime da resposta desejada para aquele neurônio, em algum sentido estatístico. Em outras palavras, que as respostas entre os $t_j(n)$ e $o_j(n)$ sejam tão pequenas quanto necessário. Uma função de custo comumente empregada é o *Sum of Squared Error* (SSE) definido como $E(n) = \frac{1}{2}\sum_j (e_j(n))^2$.

Devemos nos atentar que $E(n)$ define uma superfície de erro sobre o espaço dos pesos. A superfície de erro é caracterizada pela presença de um mínimo global e um ou mais mínimos locais. Os métodos de otimização utilizados na minimização de $E(n)$ usualmente recorrem à informação do gradiente do erro para ajustar os parâmetros da rede. Teoricamente, estes

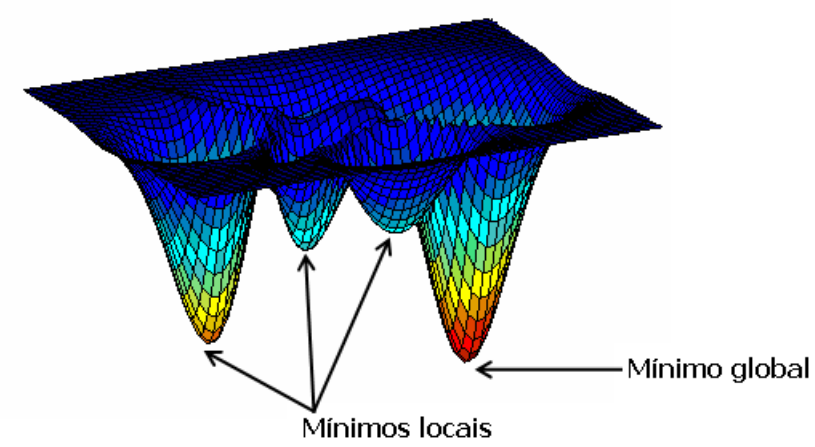


Figura 2.8: Possível superfície de erro

métodos sempre atingem um ponto de mínimo da superfície de erro, mas nada se pode afirmar sobre a natureza (local ou global) do ponto de mínimo obtido a partir de uma condição inicial arbitrária. A Figura (2.8) ilustra uma possível superfície de erro.

2.6 Algoritmo *Backpropagation*

O algoritmo *backpropagation* (BP) é utilizado no treinamento de redes neurais MLP. Basicamente, o algoritmo consiste em duas fases de computação: a fase de propagação e a de retropropagação. Na fase de propagação, Figura (2.9), uma entrada é aplicada à rede neural e o seu sinal é propagado pela rede, camada por camada. Durante a fase de propagação, os pesos da rede permanecem fixos. Na fase de retropropagação, Figura (2.10), um sinal de erro é calculado na saída da rede e é propagado no sentido reverso, camada por camada, e ao final deste processo os pesos são ajustados de acordo com uma regra de correção de erro.

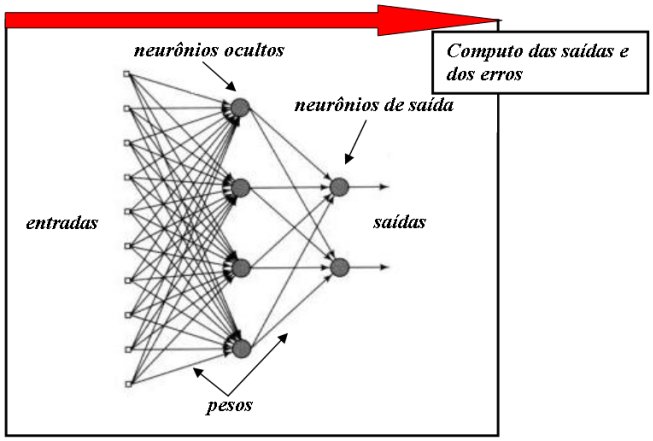


Figura 2.9: Fase de propagação

O treinamento de uma rede com o algoritmo *backpropagation* é realizado de forma supervisionada, ou seja, é apresentada à rede uma determinada entrada e também é disponibilizada

a resposta desejada para aquela entrada. A popularidade deste algoritmo resulta de sua relativa simplicidade de implementação e da capacidade de armazenar conteúdo de informação (adquirido pela rede MLP a partir do conjunto de dados) nos pesos sinápticos da rede.

Apresentados estes conceitos o algoritmo *backpropagation* pode ser descrito em detalhes. Considere o neurônio j. Na Figura (2.11), os vetores de entrada e os alvos de uma rede MLP podem ser representados por X e T, respectivamente. Onde T é a aplicação dos valores X sobre a função de interesse. Assim, define-se como padrão os vetores de entrada X e suas correspondentes saídas desejadas T utilizados no período de treinamento. Em problemas práticos, são apresentados diversos padrões de treinamento $[X, T]$ à RNA, para que, ao final do aprendizado, ela forneça respostas para entradas diferentes de X utilizadas no treinamento.

$$e_j(n) = t_j(n) - o_j(n) \tag{2.9}$$

$$E(n) = \frac{1}{2}\sum_j (e_j(n))^2 \tag{2.10}$$

$$MSE = \frac{1}{N}\sum_{n=0}^{N-1} E(n) \tag{2.11}$$

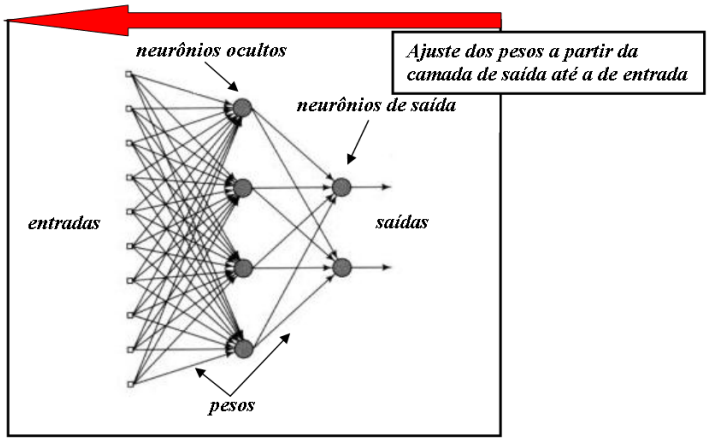


Figura 2.10: Fase de retropropagação

Seja j o indice que identifica o neurônio e n o passo de execução e considere, inicialmente, o método de treinamento em que os pesos são ajustados *padrão-a-padrão*. As mesmas etapas descritas na seção (2.1) são procedidas e mostradas nas nas Equações (2.12) e (2.13). O erro fornecido para o neurônio j é dado pela diferença entre a saída $o_j(n)$ e o alvo $t_j(n)$. Como descrito na Equação (2.9), define-se o valor instantâneo do erro quadrático para o neurônio j de acordo com a Equação (2.10). Considerando N como número total de padrões (vetores-exemplo) contidos no conjunto de treino, o *Mean Square Error* (MSE) ou erro quadrático médio é obtido somando $E(n)$ sobre todo n e então normalizando com respeito ao tamanho N do conjunto de treino, conforme a Equação (2.11).

Para um dado conjunto de treino, $E(n)$ representa a função custo do processo de minimização do erro de aprendizado, constituindo uma medida inversa do desempenho do processo de aprendizado a partir do conjunto de treino. Para minimizar $E(n)$, os pesos sinápticos são atualizados a cada apresentação de um novo padrão a MLP através do vetor de entrada até o término de uma Época.

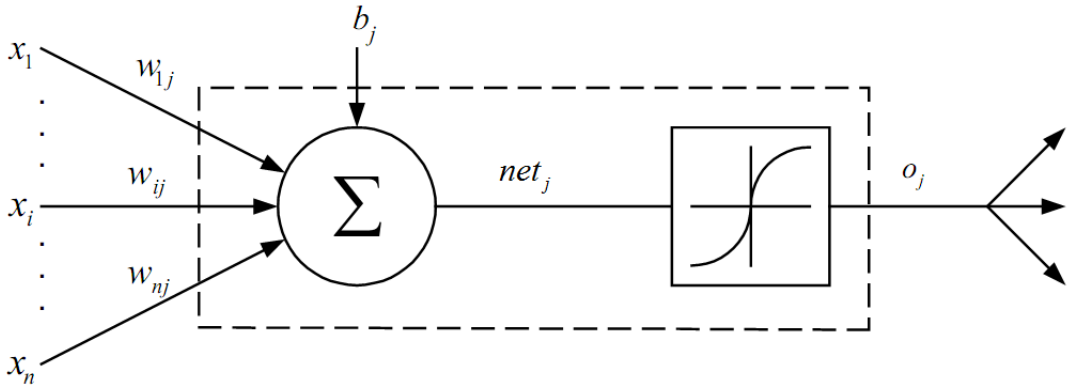


Figura 2.11: Neurônio de saída

Uma Época consiste no intervalo correspondente à apresentação de todos os N vetores-exemplo do conjunto de treino à camada de entrada da MLP. O ajuste dos pesos é feito de acordo com os respectivos erros computados para cada padrão apresentado à rede. A média aritmética destas alterações individuais nos pesos sobre o conjunto de treino é portanto uma estimativa da verdadeira alteração que resultaria a partir da alteração de pesos baseada na minimização da função de custo sobre todo o conjunto de treino. Considere a Figura (2.11), a qual descreve o neurônio j sendo alimentado por um conjunto de sinais produzidos na saída dos neurônios da camada à sua esquerda.

$$net_j(n) = \sum_i w_{ji}(n)x_i(n) + b_j \tag{2.12}$$

$$o_j(n) = f(net_j(n)) \tag{2.13}$$

A correção dos pesos é obtida a partir da regra delta, definida pelo gradiente do erro quadrático em função do próprio peso como mostra a Equação (2.14). A influência da taxa de aprendizado não está no caminho percorrido pelo algoritmo e sim na velocidade para alcançar o mínimo global (Braga et al., 2007). Uma taxa de aprendizado muito baixa torna o treinamento muito lento, ao passo que uma taxa de aprendizado muito alta provoca oscilações no treinamento e impede a convergência do processo de aprendizado. A Figura (2.8) ilustra uma possível superfície de erro que o algoritmo *backpropagation* pode percorrer. Para cada problema e aplicação, é necessário encontrar a melhor taxa de aprendizagem η por tentativa e erro, lembrando das restrições citadas há pouco.

$$\Delta w_{ji}(n) = \eta \frac{\partial}{\partial w_{ji}} E(n) \tag{2.14}$$

$$\frac{\partial}{\partial w_{ji}} E(n) = \frac{\partial E(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial o_j(n)} \cdot \frac{\partial o_j(n)}{\partial net_j(n)} \cdot \frac{\partial net_j(n)}{\partial w_{ji}} \tag{2.15}$$

Aplicando a regra da cadeia na derivada parcial da Equação (2.14), obtemos a equação em que a correção do peso é escrita em função do erro $e_j(n)$, da saída $o_j(n)$, do potencial de ativação $net_j(n)$ e do próprio peso w_{ji} do neurônio j.

Derivando cada termo da Equação (2.15), teremos os resultados parciais em (2.16):

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \tag{2.16a}$$

$$\frac{\partial e_j(n)}{\partial o_j(n)} = -1 \tag{2.16b}$$

$$\frac{\partial o_j(n)}{\partial net_j(n)} = f'(net_j(n)) \tag{2.16c}$$

$$\frac{\partial net_j(n)}{\partial w_{ji}(n)} = x_i \tag{2.16d}$$

Assim, o gradiente $\frac{\partial E(n)}{\partial w_{ji}}$ resulta em:

$$\frac{\partial E(n)}{\partial w_{ji}} = -e_j(n) \cdot f'(net_j(n)) \cdot x_i(n) \tag{2.17}$$

Pode-se observar na Equação (2.17) que o gradiente do MSE é expresso em função de três parâmetros: o erro, a derivada da FA e da entrada correspondente ao peso a corrigir. Como dito na seção 2.2, a necessidade de se utilizar funções de ativação como a linear, logística ou a tangente hiperbólica na aplicação do algoritmo *backpropagation* é mostrada claramente na Equação (2.17), em que a FA da rede deve ser necessariamente diferenciável em todo o seu domínio.

Substituindo o resultado da Equação (2.17) na Equação (2.14) e definido δ_j como (2.18), obtemos a Equação (2.19), que nada mais é do que termo de correção dos pesos em relação ao erro da rede, e na Equação (2.20) temos a regra de atualização dos pesos.

$$\delta_j(n) = -\frac{\partial E(n)}{\partial net_j(n)} = e_j(n) \cdot f'(net_j(n)) \tag{2.18}$$

$$\Delta w_{ji}(n) = -\eta \cdot \delta_j(n) \cdot x_i \tag{2.19}$$

$$w_{ji}^{new} = w_{ji}^{old} + \Delta w_{ji} \tag{2.20}$$

Salientamos que a Equação (2.18) só é válida para os neurônios da camada de saída, já que as saídas desejadas são definidas apenas para a última camada. Logo, a equação para correção dos pesos para as camadas intermediárias deve ser definida em função dos erros fornecidos pela última camada. Em outras palavras, devemos definir o gradiente δ_j para os neurônios pertencentes às camadas escondidas.

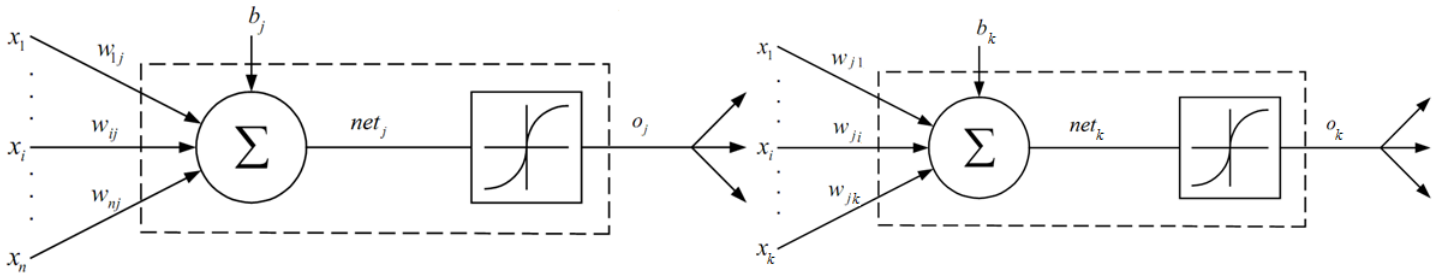


Figura 2.12: Neurônio escondido j e neurônio de saída k

Quando o neurônio j pertence a uma camada escondida, não há nenhuma saída desejada pré-especificada para o neurônio. Assim, o sinal de erro de um neurônio escondido deve ser calculado em termos dos sinais de erro de todos os neurônios aos quais o neurônio está diretamente conectado. Considere a Figura (2.12), que mostra o neurônio j como um neurônio escondido da rede.

Da Equação (2.18), podemos redefinir o gradiente local $\delta_j(n)$ para o neurônio escondido j como (2.21) usando a relação da Equação (2.16c). Para calcular a derivada parcial $\frac{\partial E(n)}{\partial o_j(n)}$, devemos proceder como segue:

$$\delta_j(n) = -\frac{\partial E(n)}{\partial o_j(n)} \cdot \frac{\partial o_j(n)}{\partial net_j(n)} = -\frac{\partial E(n)}{\partial o_j(n)} \cdot f'(net_j(n)) \tag{2.21}$$

Derivando a Equação (2.10) em relação a $o_j(n)$, obtemos a Equação (2.22a)

$$\frac{\partial E(n)}{\partial o_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial o_j(n)} \tag{2.22a}$$

$$\frac{\partial E(n)}{\partial o_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial net_k(n)} \cdot \frac{\partial net_k(n)}{\partial o_j(n)} \tag{2.22b}$$

Usando a regra da cadeia para calcular $\frac{\partial e_k(n)}{\partial o_j(n)}$, podemos rescrever (2.22a) como (2.22b). O erro na saída do neurônio k é dado por

$$e_k(n) = t_k(n) - o_k(n) = t_k(n) - f(net_k(n)) \tag{2.23}$$

sendo que $net_k(n)$ é o nível de ativação interna do neurônio k. Assim, temos

$$\frac{\partial e_k(n)}{\partial net_k(n)} = -f'(net_k(n)) \tag{2.24}$$

Da Figura (2.12) podemos ver que o nível de ativação interna de neurônio k é dado por

$$net_k(n) = \sum_{j=0}^{q} w_{kj}(n) \cdot o_j(n) \tag{2.25}$$

sendo que q é o número total de entradas (excluindo a polarização) aplicadas ao neurônio k. Derivando a Equação (2.25), em relação a $o_j(n)$, obtemos

$$\frac{\partial net_k(n)}{\partial o_j(n)} = w_{kj}(n) \tag{2.26}$$

Assim, substituindo as Equações (2.24) e (2.26) na Equação (2.22b), obtemos a derivada parcial desejada descrita por:

$$\frac{\partial E(n)}{\partial o_j(n)} = -\sum_k e_k(n) \cdot f'(net_k(n)) \cdot w_{kj}(n) \tag{2.27a}$$

$$= -\sum_k \delta_k(n) \cdot w_{kj}(n) \tag{2.27b}$$

sendo que em (2.27b) fizemos uso da definição de gradiente local dada na Equação (2.18) para exprimir $\delta_k(n)$.

Finalmente, substituindo a Equação (2.27b) na Equação (2.21), obtemos a expressão para o gradiente local $\delta_j(n)$ para o neurônio escondido j:

$$\delta_j(n) = f'(net_j(n)) \cdot \sum_k \delta_k(n) \cdot w_{kj}(n) \tag{2.28}$$

O termo $f'(net_j(n))$ na Equação (2.28) depende apenas da FA associada ao neurônio j. O termo restante, isto é, o somatório sobre k, depende de dois conjuntos de termos. O primeiro

conjunto de termos, os δ_k, exige o conhecimento dos sinais de erro $e_k(n)$, para todos os neurônios localizados na camada imediatamente posterior à camada onde se encontra o neurônio j, e que estão diretamente conectados ao neurônio j; veja a Figura (2.12). O segundo conjunto de termos, os w_{kj}, consiste nos pesos sinápticos associados a estas conexões.

2.7 Modos de Treinamento

Uma apresentação completa do conjunto de treinamento durante o processo de aprendizagem é denominada época. Dado um conjunto de treinamento, o algoritmo *backpropagation* pode ser executado em dois modos distintos:

1. Modo *padrão-a-padrão (on-line)*: os pesos são atualizados após a apresentação de cada padrão à rede neural. Neste modo de treinamento, uma época consiste na apresentação dos N padrões de treinamento organizados na ordem $\{[X(i), T(i)], i = 1, 2, \cdots, n\}$. O primeiro exemplo da época é apresentado à rede, e a seqüência de processamentos das fases de propagação e retropropação, discutidos na seção 2.6, é realizada, resultando no ajuste de pesos da rede. Repete-se este processo até que o último padrão $[X(n), T(n)]$ seja apresentado à rede, concluindo assim a época, que, após sua conclusão, o critério de parada é analisado. Caso o critério não seja satisfeito, o processo e repetido.

2. Modo *batelada (batch mode)*: todos os padrões $\{[X(i), T(i)], i = 1, 2, \cdots, n\}$ são apresentados de uma só vez e somente após a apresentação os pesos são ajustados, constituindo assim uma época. Para uma determinada época, definimos a função de custo como o erro quadrático médio, definido na Equação (2.11). A correção a ser aplicada a cada peso w_{ji} será proporcional à derivada parcial $E(n)$ em relação w_{ji} e ao termo $1/N$ que precede a Equação (2.11). Assim, temos

$$\frac{\partial MSE}{\partial w_{ji}} = \frac{1}{N} \cdot \sum_{n=1}^{N} \frac{\partial E(n)}{\partial w_{ji}} \tag{2.29}$$

   sendo que, para calcular $\frac{\partial E(n)}{\partial w_{ji}}$, procedemos da mesma forma descrita anteriormente na seção 2.6. A escolha de qual modo utilizar vai depender do desempenho que se quer atingir com a rede e da aplicação para a qual a RNA está sendo modelada (Møller, 1993).

2.8 Critérios de Parada

O algoritmo *backpropagation* executa a correção dos pesos diversas vezes e necessita de pelo menos um critério de parada (Haykin, 1994). Vamos listar aqui alguns dos critérios de parada mais usuais:

1. Quando o algoritmo de treinamento for executado um número máximo de vezes, definido a *priori*.

2. Quando a norma euclidiana do gradiente do erro for inferior a um limiar pré-especificado e arbitrariamente pequeno.

3. Quando a taxa de variação absoluta no erro quadrático médio por época for suficientemente pequena.

4. Quando o erro médio quadrático MSE atingir um valor pré-especificado e arbitrariamente pequeno.

2.9 Normalização dos Dados

As principais funções de ativação utilizadas nas aplicações práticas são a logística, Equação (2.4), e a tangente hiperbólica, Equação (2.5), apresentadas na seção 2.2. Estas funções tem suas saídas limitadas, a logística entre $[0, 1]$ e a tangente hiperbólica entre $[-1, 1]$, e, caso o conjunto de dados de treinamento e utilização do problema a ser abordado estiverem fora dos limites da função, há a necessidade destes serem normalizados de modo que o treinamento não seja comprometido. Para normalizar um elemento t do conjunto de alvos T, é empregada a Equação (2.30)

$$t_{norm} = (M - m)\frac{(t - T_{\min})}{(T_{\max} - T_{\min})} + m \tag{2.30}$$

onde:

- t: alvo;
- t_{norm}: alvo normalizado;
- M e m: valores máximo e mínimo, respectivamente, da escala normalizada;
- $T_{\max}$ e $T_{\min}$: valores máximo e mínimo, respectivamente, do vetor de alvos T.

Os valores M e m vão depender da FA empregada, mas não é aconselhável que estes valores assumam exatamente os valores de saturação das funções de ativação, pois isto pode comprometer a convergência do treinamento. O ideal é escolher um intervalo próximo às não-linearidades da FA. Após o treinamento, há a necessidade de desnormalizar os dados obtidos. Para isso, fazemos uso da Equação (2.31).

$$t = (T_{\max} - T_{\min})\frac{(t_{norm} - m)}{(M - m)} + T_{\min} \tag{2.31}$$

2.10 Inicialização dos Pesos

O primeiro passo do algoritmo de *backpropagation* é a inicialização dos pesos da rede, o que corresponde à definição de um ponto inicial da superfície de erro. Uma inicialização inadequada (ponto inicial mal localizado) pode fazer com que o algoritmo de treinamento fique preso em um mínimo local ou apresente problemas numéricos que de outra forma poderiam ser evitados. Como usualmente não temos nenhuma informação que possa ser diretamente empregada na inicialização dos pesos da rede, uma solução comumente utilizada é inicializar os pesos aleatoriamente com distribuição uniforme sobre um pequeno intervalo em torno do zero. Diversos métodos de inicialização já foram propostos com o intuito de melhorar o desempenho e convergências de RNA (Nguyen e Widrow, 1990), (Boers e Kuiper, 1992) porém, neste trabalho é adotado a inicialização aleatória dos pesos com vista a validar o desempenho da metodologia proposta sem qualquer qualquer modificação de desempenho no algoritmo *backprpagation*.

CAPÍTULO

3

Redes Neurais Artificiais Complexas

Do ponto de vista estrutural, projetar redes neurais artificiais RNA, que possam processar sinais complexos, é semelhante a se projetar suas contrapartes de domíno real. A principal diferença encontra-se na adaptação do algoritmo BP e na escolha adequada da FA para o domínio dos números complexos. Exclusas essas dificuldades, as RNA de domínio complexo adotam os mesmos conceitos de **topologia de rede**, **modos de treinamento**, **critérios de parada** e **validação do treinamento** de RNA reais.

Para que possamos adaptar satisfatoriamente o algoritmo BP, ao caso complexo, devemos compreender os conceitos relacionados com as funções contínuas, diferenciáveis, analíticas e limitadas e não-limitadas no domínio complexo, pois estes conceitos são de suma importância para se desenvolver as regras de aprendizado de domínio complexo utilizando-se o método gradiente descendente (Yi, 2004), (Nitta, 2000).

Quando bem compreendidos, estes conceitos guiam a escolha da FA a ser implementada nessas redes. Dependendo da aplicação a ser abordada, esta escolha tem grande influência na capacidade de generalização e aprendizado da rede, visto que as FA de domínio complexo podem apresentar pontos singulares.[1] A princípio, qualquer FA não-linear complexa pode ser usada no algoritmo *Complex Backpropagation* (CBP), desde que os problemas com os pontos singulares também chamados de singularidades sejam contornados (Leung e Haykin, 1991), (Benvenuto e Piazza, 1992), (Georgiou e C, 1992), (Arena et al., 1993), (Arena et al., 1995), (Kim, 2001), (Yi, 2004).

[1] Pontos onde a função de ativação toma valores no infinito (∞).

3.1 O Neurônio Artificial Complexo

O neurônio artificial complexo (Leung e Haykin, 1991) apresentado na Figura (3.1) é semelhante ao neurônio artificial convencional apresentado na seção 2.1. A diferença está nos valores de entrada, alvos, pesos que são todos números complexos. O limiar ou *bias* deste neurônio é definido com o valor 1, pois este é o elemento identidade no domínio complexo, mas do ponto de vista conceitual tal fato não impede que ele seja um número complexo (W. L. Chan e Lai, 2000).

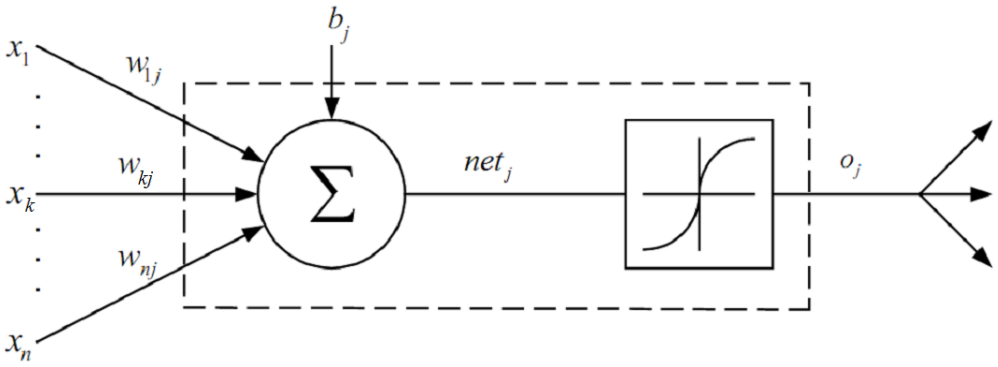


Figura 3.1: O neurônio artificial complexo

Assim, no domínio real, net_j é descrito como uma combinação linear no espaço vetorial das entradas $x_k(n)$ cujo os pesos são $w_{jk}(n)$, como mostra a Equação (3.1). De maneira a facilitar a compreensão das equações, o termo (n), que corresponde ao n-ésimo padrão de treinamento, será ausentado das expressões que seguem.

$$net_j = \sum_k w_{jk}x_k + b_j = a_j + id_j \tag{3.1}$$

$$o_j = f(net_j) = u_j + iv_j \tag{3.2}$$

A saída fornecida por cada neurônio da rede complexa é dada pela Equação (3.2), em que $f(\cdot)$ é a *função de ativação complexa*, onde u_j é a parte real da saída do neurônio e v_j é a parte imaginária da saída do neurônio.

O sinal de saída do neurônio é comparado com o alvo t_j e o erro do neurônio é obtido conforme Equação (3.3).

$$e_j = t_j - o_j \tag{3.3}$$

Para a definição do MSE complexo, devemos lembrar que o domínio complexo não é totalmente ordenado. Este fato leva ao consenso entre (Benvenuto e Piazza, 1992), (Georgiou e C, 1992), (Leung e Haykin, 1991) e outros autores de que a função custo deve ser uma função real. Esta função é apresentada na Equação (3.4).

$$E = \frac{1}{2}\sum_{j}\left(e_j e_j^*\right) = \frac{1}{2}\sum_{j}\left(e_{jR}^2 + e_{jI}^2\right) \tag{3.4}$$

Onde e_{jR} e e_{jI} correspondem, respectivamente, à parte real e imaginária do erro de cada neurônio. A próxima seção tratará dos conceitos ligados à diferenciabilidade no domínio complexo, necessários para compreensão do algoritmo CBP, e das relações e implicações com relação à escolha da FA mais adequada para este algoritmo.

3.2 Diferenciabilidade no Domínio Complexo

A diferenciabilidade no domínio complexo $\mathbb{C}$ nos ajudará a compreender melhor o comportamento das FA de variável complexa e conseqüentemente algoritmo CBP. Uma rede do tipo MLP, usando o algoritmo BP, pode ser considerada um aproximador universal de funções, quando as FA implementadas no algoritmo são não-lineares. Em outras palavras, com FA não-lineares, a rede neural se torna capaz de solucionar diversos problemas de ordem prática. (Leung e Haykin, 1991), que pode ser considerado um dos primeiros artigos sobre redes neurais artificiais complexas, os autores estenderam o conceito de FA não-linear diferenciável para o domínio complexo, usando a FA *fully* logística complexa, porém, esta função e diversas outras apresentam pontos singulares. Assim, torna-se necessário a melhor compreensão da noção de diferenciabilidade no domínio complexo para escolhermos a melhor estratégia para se contornar este problema. Uma função $G : D \subset \mathbb{C} \rightarrow \mathbb{C}$ é diferenciável em $z_0 \in D$ se para qualquer $h \in D$ a relação abaixo existir.

$$G'(z_0) = \frac{d}{dz_0}G(z_0) = \lim_{h\rightarrow 0}\frac{G(z_0+h) - G(z_0)}{h} \tag{3.5}$$

Formalmente, esta definição é semelhante ao conceito de diferenciação no domínio real, porém, devemos atentar para o fato de que condições adicionais para as derivadas parciais de $G(z_0)$ devem ser estabelecidas (Knoop, 1996a), para garantir a validade da Equação (3.5). Mais tecnicamente falando, estamos nos referindo às Equações de Cauch-Riemann (3.7).

Primeiramente devemos lembrar que uma função $G(z) = (u(x,y), v(x,y)) : D \subset \mathbb{C} \rightarrow \mathbb{C}$ é diferenciável em $z_0 \in D$, se e somente se u e v forem funções reais diferenciáveis em z_0 e as equações de Cauchy-Riemann forem satisfeitas.

$$u_x = \frac{\partial u}{\partial x}; \qquad u_y = \frac{\partial u}{\partial y}; \qquad v_x = \frac{\partial v}{\partial x}; \qquad v_y = \frac{\partial v}{\partial y} \tag{3.6}$$

$$u_x = v_y \qquad u_y = -v_x \tag{3.7}$$

3.3 Funções Inteiras e Limitadas no Domínio Complexo

Para escolhermos a FA complexa a ser implementada no algoritmo *backpropagation* complexo faz-se necessário a compreensão dos seguintes conceitos e definições: Se uma função G complexa é diferenciável na vizinhança ao redor $z_0 \in \mathbb{C}$, então G é dita ***analítica*** ou ***holomorfa*** em $\mathbb{C}$ (Knoop, 1996b). Isto equivale a dizer que todas as derivadas parciais de G descritas na Equação (3.6) são contínuas (Knoop, 1996a). O que nos leva a seguinte definição:

Uma função $f : \mathbb{C} \rightarrow \mathbb{C}$ é dita **inteira** quando esta é **holomorfa** por todo plano $\mathbb{C}$.

Esta definição foi estudada em detalhes por Liouville resultando no seu teorema sobre as propriedades das funções inteiras limitadas:

Teorema 1 (Liouville) *"Se uma função complexa f é* **inteira e limitada** *em $\mathbb{C}$, então f* ***é constante****"*.

Funções classificadas como inteiras e limitadas restringem muito o número de soluções que as RNA implementadas com ela podem solucionar. Conceitos e demonstrações sobre diferenciabilidade no domínio $\mathbb{C}$ podem ser encontrados nas referências (Lyness, 1967), (Saff e Snider, 1976). A proxima seção, apresenta as principais FA de domínio complexo suas limitações, vantagens e desvantagens.

3.4 Funções de Ativação Complexas

Segundo o **Teorema 1**, funções inteiras limitadas são todas constantes, razão pela qual esta seção restringe-se a discussão de duas classes de funções: as *inteiras não-limitadas* e as *não-inteiras limitadas* como possíveis FA para o algoritmo CBP.

3.4.1 Funções Inteiras Não-Limitadas

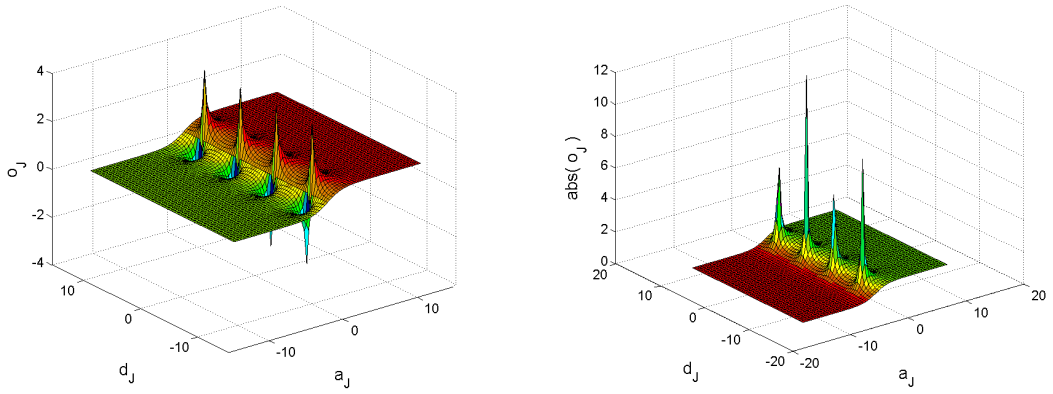
As FA mais utilizadas em redes MLP foram apresentadas na seção 2.2 e podem ser estendidas para o domínio $\mathbb{C}$, estas serão apresentadas a seguir, ao mesmo tempo em que é feita a discussão da implicação existente, no uso destas funções em RNA de domínio complexo.

Função *Fully* Logística

A função *fully* logística nada mais é que a extensão da função logística real para o domínio $\mathbb{C}$, razão por que é chamada de *fully*.

$$o_j = f(net_j) = f(a_j + id_j) = \frac{1}{1 + e^{-net_j}} \tag{3.8}$$

Esta função é classificada como não-limitada. Para entender melhor esta classificação, considere o exemplo: seja $y = e^z$ com $z = x + iy$ a função exponencial. Podemos reescrever e^z como $e^x e^{iy}$, o que torna claro que o termo e^{iy} é a identidade de Euler. Logo, temos: $e^z = e^{x+iy} = e^x e^{iy} = e^x(cos(y) + isin(y))$, porém, esta FA apresenta infinitas singularidades devido ao fato de que quando $z \in 0 \pm i(k+1)\pi$ com $k \in \mathbb{N}$ implicando: $e^{-z} = e^{-x}(cos(-y) + jsin(-y)) = -1$ é, conseqüentemente, $f(z) = \frac{1}{1+e^{-z}} \to \infty$. As singularidades de (3.8) podem ser observadas nos picos da Figura (3.2).



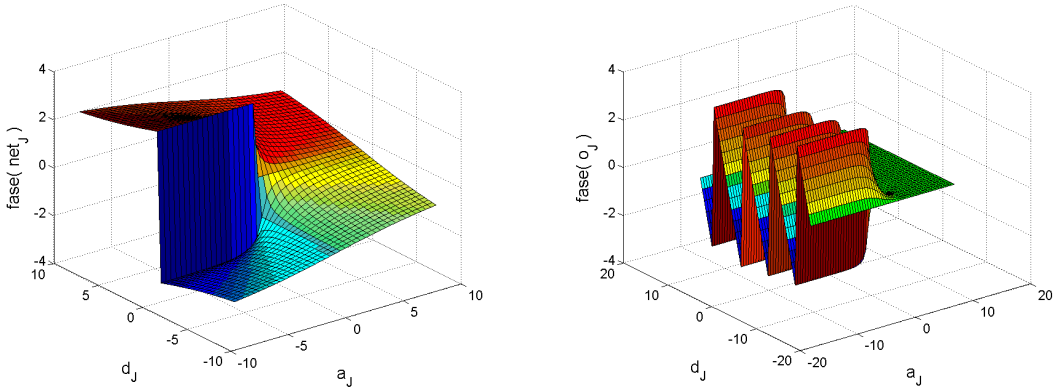
(a) FA *fully* logística (b) Magnitude *fully* logística

Figura 3.2: FA *fully* logística e respectiva magnitude

Segundo (Georgiou e C, 1992), estas singularidades são características e podem atrapalhar o treinamento de redes MLP implementadas com o algoritmo CBP, pois se a saída do neurônio situar-se em algum ponto de singularidade, a derivada da FA assumirá um valor elevado e conseqüentemente a correção dos pesos poderá ser maior que o necessário, podendo elevar o tempo de convergência ou até mesmo comprometê-lo.

Em contrapartida, (Leung e Haykin, 1991) dizem que o fato de a função ter singularidades não é um grande problema, pois este pode ser contornado apenas escalando ou normalizando os dados de entrada para alguma região satisfatória do plano complexo onde as singularidades não tenham tanta influência, visto que os pontos singulares são periódicos.

A Figura (3.3) mostra uma outra característica importante sobre esta FA. A fase $\theta = \arg(net_j)$ da entrada Figura [3.3(a)] e a fase $\theta = \arg(o_j)$ da saída Figura [3.3(b)], como podemos observar, *são distintas e não-lineares*. Estas características são desejáveis para aplicações em que há necessidade de variação da fase (Buchholz e Sommer, 2000).



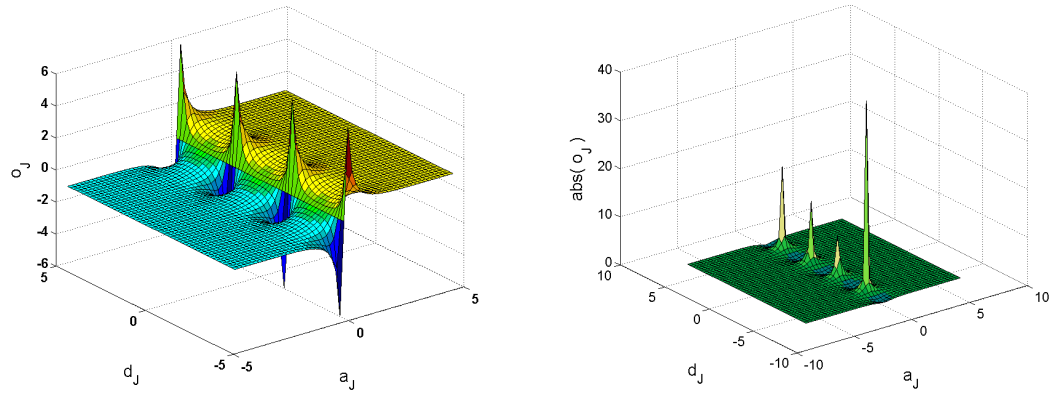
(a) Fase do espaço de entrada (b) Fase de saída: *fully* logística

Figura 3.3: Fase do espaço de entrada e da função *fully* logística

Função *Fully* Tangente Hiperbólica

Da mesma maneira que a função logística pode ser estendida para o domínio complexo, a função tangente-hiperbólica de domínio real também pode. No domínio complexo, ela recebe o nome de *fully* tangente hiperbólica.

$$o_j = f(net_j) = f(a_j + id_j) = \frac{e^{net_j} - e^{-net_j}}{e^{net_j} + e^{-net_j}} \tag{3.9}$$



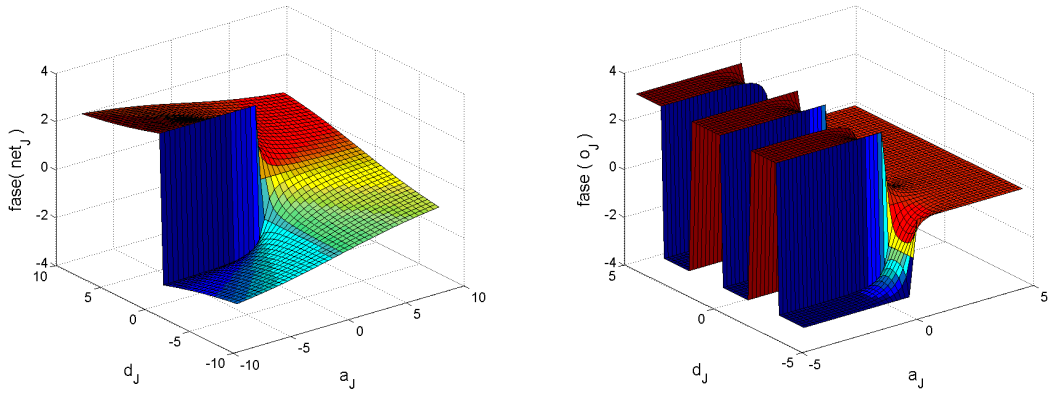
(a) FA *fully* tangente hiperbólica (b) Magnitude *fully* tangente hiperbólica

Figura 3.4: FA *fully* tangente hiperbólica e respectiva magnitude

Esta FA também possui pontos singulares quando $net_j \in 0 \pm i\left(2k + \frac{1}{2}\right)\pi$ com $k \in \mathbb{N}$, que podem ser visualizados na Figura (3.4). (Kim, 2001) validou diversas funções inteiras não-limitadas entre elas a função *fully* tangente hiperbólica enquanto (Kim e Adali, 2003) demonstram que esta FA é um bom discriminante não-linear e satisfaz as condições de generalização impostas pelo algoritmo CBP em redes MLP complexas.

A FA *fully* tangente hiperbólica também guarda a característica de não-linearidade na fase, tal como a *fully* logística, como pode ser visto na Figura (3.5), e as condições citadas por

(Leung e Haykin, 1991), para o contorno dos problemas das singularidades, são reforçadas por (Kim, 2001), (Kim e Adali, 2003) com a adição da ressalva, a qual diz que, apesar de essa função ser classificada como não-limitada, ela é limitada por quase todo seu domínio, salvo os pontos singulares, mas isto é suficiente para resolvermos a maioria das aplicações práticas em engenharia. Mais adiante discutiremos com mais detalhes este assunto.



(a) Fase do espaço de entrada (b) Fase de saída: *fully* tangente hiperbólica

Figura 3.5: Fase do espaço de entrada e da função *fully* tangente hiperbólica

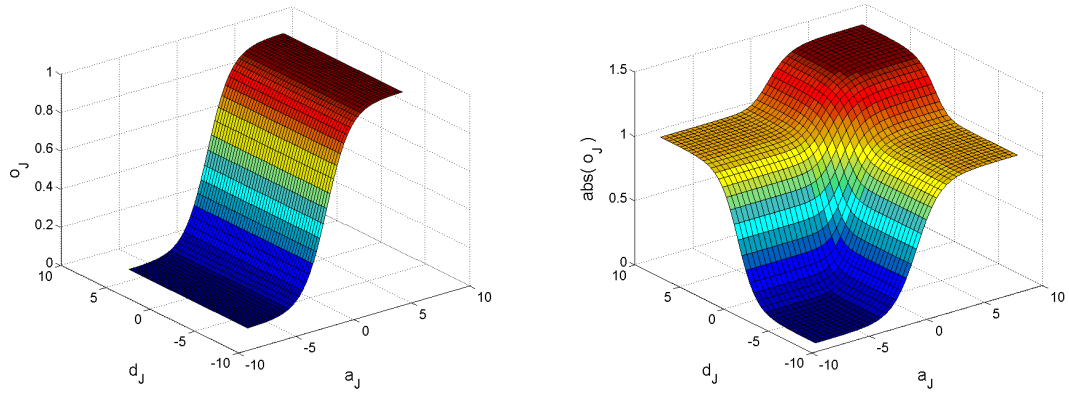
3.4.2 Funções Não-Inteiras Limitadas

Esta classe de funções não possui derivadas totais, pois u_j e v_j da saída $f(net_j)$ são calculados separadamente. No entanto, pode-se calcular as derivadas parciais em todo seu domínio. Este fato implica que, quando se definir a regra de ajuste dos pesos sinápticos no algoritmo CBP, devemos fazê-lo usando as derivadas parciais. Nesta seção, vamos apresentar alguns exemplos deste tipo de FA e suas principais características.

Função *Split* Logística

Esta função *split* simplesmente consiste em aplicar a função logística, descrita na Equação 2.4, na parte real e na parte imaginária de $net_j = a_j + id_j$ (e por este motivo é chamada *split*) como mostra a Equação (3.10). Esta FA é dita limitada devido a ausências de pontos singulares, como pode ser observado na Figura (3.6). Para as FA dessa classe, a característica de ser limitada é vista por (Georgiou e C, 1992) e (Benvenuto e Piazza, 1992) como ideal para se projetar redes MLP complexas usando o algoritmo CBP.

$$o_j = f(net_j) = f(a_j + id_j) = \frac{1}{1+e^{-a_j}} + i\left(\frac{1}{1+e^{-d_j}}\right) \tag{3.10}$$



(a) FA *split* logística (b) Magnitude *split* logística

Figura 3.6: FA split logística e respectiva magnitude

Porém para escrevermos o algoritmo CBP, usando FA da mesma classe que (3.10), devemos usar suas derivadas parciais, já que estas não são analíticas por toda a extensão do domínio complexo $\mathbb{C}$.

Logo, necessitamos derivar as componentes u_j e v_j de o_j em (3.10) em função das componentes a_j e d_j do sinal de entrada net_j. Em (3.11) seguem estes resultados:

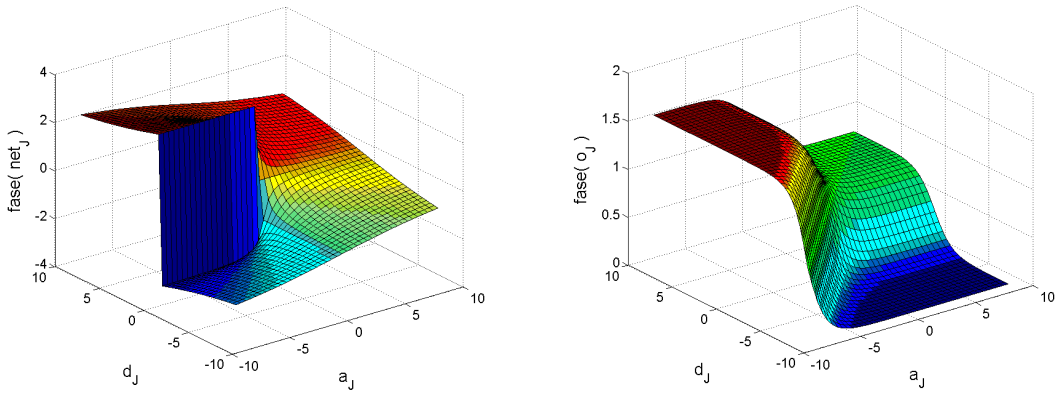
$$u_{a_j} = \frac{\partial u_j}{\partial a_j} = \frac{1}{1+e^{-a_j}}\left(1-\frac{1}{1+e^{-a_j}}\right) \tag{3.11a}$$

$$u_{d_j} = \frac{\partial u_j}{\partial d_j} = 0 \tag{3.11b}$$

$$v_{a_j} = \frac{\partial v_j}{\partial a_j} = 0 \tag{3.11c}$$

$$v_{d_j} = \frac{\partial v_j}{\partial d_j} = \frac{1}{1+e^{-d_j}}\left(1-\frac{1}{1+e^{-d_j}}\right) \tag{3.11d}$$

Podemos notar, nas Equações (3.11a) e (3.11d), a derivada de primeira ordem da função logística aplicada na parte real e imaginária do sinal de entrada net_j. Esta FA também tem a característica desejável de não-linearidade na fase, como pode ser obsevado na Figura (3.7).



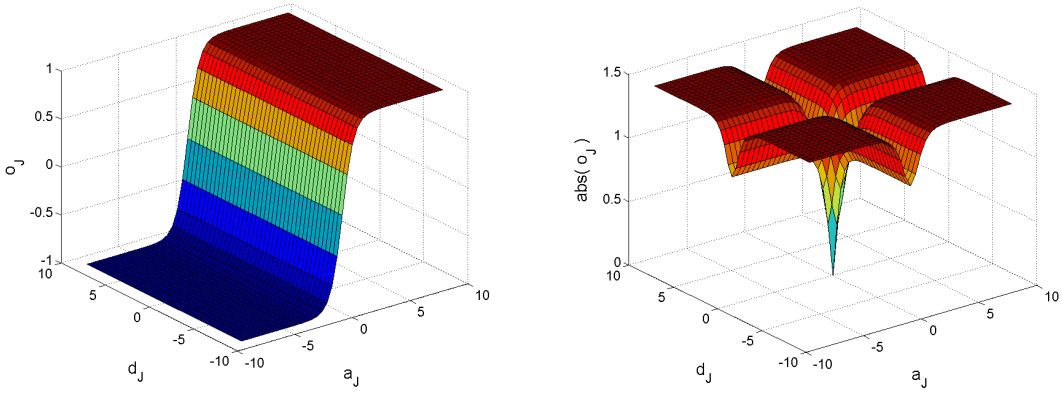
(a) Fase do espaço de entrada (b) Fase de saída: *split* logística

Figura 3.7: Fase do espaço de entrada e da função *split* logística

Função *Split* Tangente Hiperbólica

Assim como a FA logística pode ser separada em parte real e imaginária, a função tangente hiperbólica também pode ser separada e recebe o nome de *split* tangente hiperbólica, Equação (3.12). Os gráficos referentes a ela são mostrados na Figura (3.8). Ela também se encaixa no grupo das funções preferidas por (Georgiou e C, 1992) por ser não-inteira e limitada.

$$o_j = f(net_j) = f(a_j + id_j) = \frac{e^{a_j} - e^{-a_j}}{e^{a_j} + e^{-a_j}} + i\left(\frac{e^{d_j} - e^{-d_j}}{e^{d_j} + e^{-d_j}}\right) \tag{3.12}$$



(a) FA *split* tangente hiperbólica (b) Magnitude *split* tangente hiperbólica

Figura 3.8: FA *split* tangente hiperbólica e respectiva magnitude

Esta FA, assim como a *split* logística, igualmente não tem derivadas totais e faz-se necessário então calcular as derivadas parciais dos componentes u_j e v_j de o_j em (3.12), em função das componentes a_j e d_j do sinal de entrada net_j. Para se implementar o CBP, estes resultados podem ser observados na Equações (3.13a) a (3.13d). Esta FA também guarda a característica da não-linearidade entre a fase do espaço de entrada e a fase da saída, como pode ser observado

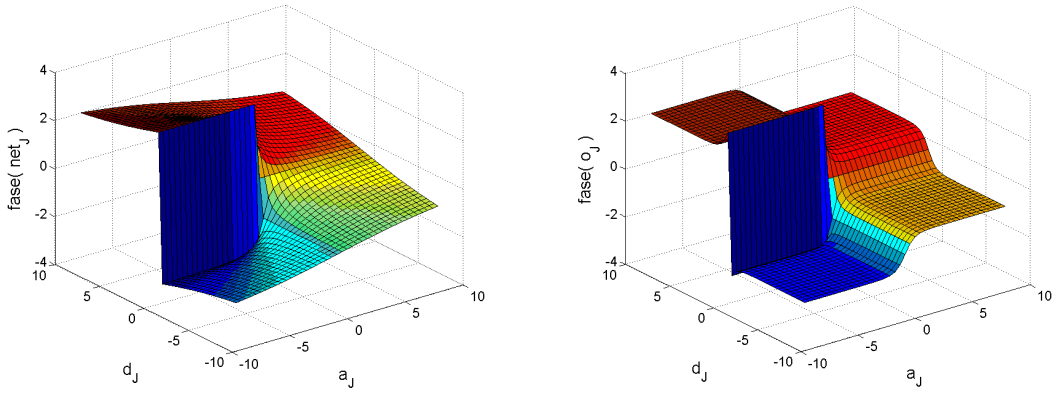
na Figura (3.9).

$$u_{a_j} = \frac{\partial u_j}{\partial a_j} = \left(\frac{2}{e^{a_j} + e^{-a_j}}\right)^2 \tag{3.13a}$$

$$u_{d_j} = \frac{\partial u_j}{\partial d_j} = 0 \tag{3.13b}$$

$$v_{a_j} = \frac{\partial v_j}{\partial a_j} = 0 \tag{3.13c}$$

$$v_{d_j} = \frac{\partial v_j}{\partial d_j} = \left(\frac{2}{e^{d_j} + e^{-d_j}}\right)^2 \tag{3.13d}$$



(a) Fase do espaço de entrada (b) Fase de saída: *split* tangente hiperbólica

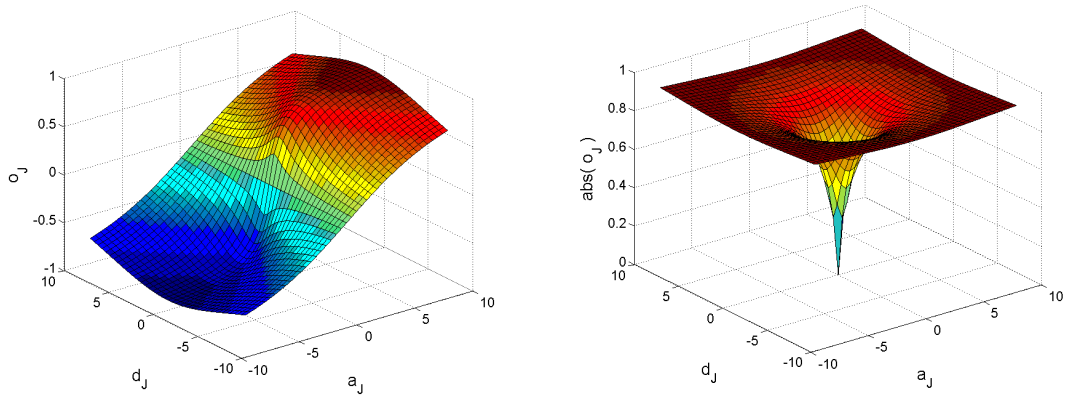
Figura 3.9: Fase do espaço de entrada e da função *split* tangente hiperbólica

Função *fully* de Georgiou

A função proposta por (Georgiou e C, 1992) é classificada como não-inteira e limitada, mas como se pode ver na Equação (3.14), a função não divide a parte real e imaginária do sinal net_j de entrada. Assim, não pode ser classificada como uma função *split*, apesar de possuir características comuns a esta. A Figura (3.10) mostra o comportamento gráfico da FA e da magnitude.

$$o_j = f(net_j) = f(a_j + id_j) = \frac{net_j}{c + \frac{1}{r}|net_j|} \tag{3.14}$$

Na Equação (3.14), temos as constantes $\{\ c \quad r\ \} \in \mathbb{R}$ onde c controla a declividade da função e r controla o limite da saída da mesma.



(a) FA *fully* de Georgiou (b) Magnitude da FA *fully* de Georgiou

Figura 3.10: FA *fully* de Georgiou e respectiva magnitude

Tal como as funções *split*, esta FA não é analítica em todo domínio $\mathbb{C}$ e exige a necessidade de se conhecer as derivadas parciais para a construção do algoritimo CBP. As derivadas parciais são apresentadas em (3.15), lembrando que $net_j = a_j + id_j$ e $o_j = u_j + iv_j$.

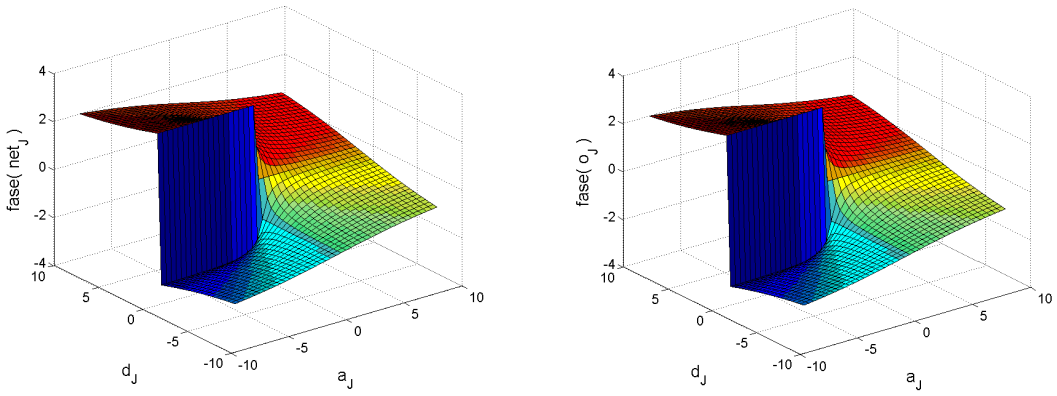
$$u_{a_j} = \frac{\partial u_j}{\partial a_j} = \begin{cases} \frac{r\left(d_j^2 + c \cdot r|net_j|\right)}{|net_j|(c \cdot r + |net_j|)^2} & se\ |net_j| \neq 0 \\ \frac{1}{c} & se\ |net_j| = 0 \end{cases} \tag{3.15a}$$

$$u_{d_j} = \frac{\partial u_j}{\partial d_j} = \begin{cases} \frac{r \cdot a_j \cdot d_j}{|net_j|(c \cdot r + |net_j|)^2} & se\ |net_j| \neq 0 \\ 0 & se\ |net_j| = 0 \end{cases} \tag{3.15b}$$

$$v_{a_j} = \frac{\partial v_j}{\partial a_j} = \begin{cases} \frac{r \cdot a_j \cdot d_j}{|net_j|(c \cdot r + |net_j|)^2} & se\ |net_j| \neq 0 \\ 0 & se\ |net_j| = 0 \end{cases} \tag{3.15c}$$

$$v_{d_j} = \frac{\partial v_j}{\partial d_j} = \begin{cases} \frac{r\left(a_j^2 + c \cdot r|net_j|\right)}{|net_j|(c \cdot r + |net_j|)^2} & se\ |net_j| \neq 0 \\ \frac{1}{c} & se\ |net_j| = 0 \end{cases} \tag{3.15d}$$

Na Figura (3.11), podemos ver que a FA proposta pelo autor não consegue implementar uma importante característica que uma FA complexa deveria possuir, que é a não-linearidade entre a fase de entrada e a de saída (Kim e Adali, 2003). Este fato reduz o número de classes de funções que a MLP complexa, implementada com esta FA, pode aproximar (Buchholz e Sommer, 2000). Em outras palavras, se a aplicação de interesse a ser resolvida não tem o comportamento da fase como informação relevante para a solução, esta FA é uma boa opção para ser implementada no algoritmo CBP, caso contrário, não.



(a) Fase do espaço de entrada (b) Fase de saída: FA *fully* de Georgiou

Figura 3.11: Fase do espaço de entrada e fase da FA *fully* de Georgiou

3.4.3 Funções de Ativação Complexas Satisfatórias

A característica natural de funções analíticas complexas de serem **não-limitadas** foi o ponto de partida de vários autores para a busca de propriedades desejáveis para funções de ativação complexas (Kim e Adali, 2003). O primeiro autor a abordar esta idéia foi (Georgiou e C, 1992), que identificou cinco propriedades que uma FA complexa deve possuir para ser considerada satisfatória. Vamos agora citá-las, assumindo antes que $z = x + iy$ e a FA sejam $f(z) = u(x, y)+iv(x, y)$ e as derivadas parciais de $f(z)$ sejam, respectivamente, u_x, u_y, v_x e v_y. Assim:

1. $f(z)$ deve ser não-linear em x e y, pois, caso seja linear, a rede *feed-forward* treinada com esta função só é capaz de resolver problemas linearmente separáveis.

2. $f(z)$ deve ser limitada. Esta afirmação só é verdadeira quando ambas funções u e v são limitadas. Caso uma delas não seja limitada durante o treinamento, há possibilidade da ocorrência de um eventual *overflow*.

3. As derivadas parciais u_x, u_y, v_x e v_y devem existir e devem ser limitadas. O motivo é o mesmo do item anterior.

4. $f(z)$ não deve ser inteira, pois, segundo o **Teorema 1**, toda função inteira e limitada é uma função constante. Como a característica limitada é fundamental para maioria dos problemas, a FA desejável deve ser não-inteira, para que isto ocorra.

5. $u_x v_y \neq v_x u_y$. A última afirmativa vem do fato que, para que a correção de peso seja efetuada, o termo δ usado na correção dos pesos sinápticos deve ser não-nulo.

Observando o **Teorema 1**, (You e Hong, 1998) perceberam que a segunda e quarta condições de (Georgiou e C, 1992) são redundantes, isto é, uma função não-linear limitada no domínio $\mathbb{C}$ não pode ser inteira. Usando este fato, os autores reduziram as condições anteriores para apenas quatro e as estendeu para as FA limitadas *split*. Assim, as condições são:

1. $f(z)$ deve ser não-linear em x e y.

2. Para garantir a estabilidade do sistema, $f(z)$ não deve possuir singularidades e ser limitada para todo z pertencente a um subconjunto limitado.

3. As derivadas parciais u_x, u_y, v_x e v_y devem ser contínuas e limitadas.

4. $u_x v_y \neq v_x u_y$. Caso contrário, $f(z)$ não é uma FA satisfatória, exceto quando:

$$\begin{aligned} u_x &= v_x = 0, \qquad e \qquad u_y \neq 0, v_y \neq 0 \\ u_y &= v_y = 0, \qquad e \qquad u_x \neq 0, v_x \neq 0 \end{aligned}$$

Estes dois conjuntos de condições apresentados enfatizam as FA limitadas e suas respectivas derivadas parciais. (Kim e Adali, 2003) notaram que estas condições nada mais são do que casos especiais do *Fully Complex Backpropagation* que utilizam *Funções Elementares Transcendentais* (FET)[2] como FA. Esta generalização só é possível porque as FET **são limitadas e analíticas por quase todo o domínio** $\mathbb{C}$.

Contudo, em seu trabalho (Kim e Adali, 2003) lembram que as FET violam claramente as condições 2 e 3, tanto no trabalho de (Georgiou e C, 1992) quanto no de (You e Hong, 1998), onde ambas as condições tratam das características FA limitadas e suas derivadas parciais também limitadas. Todavia, estas violações somente ocorrem nos pontos singulares que por natureza são todos periódicos em FET complexas. Isto torna mais clara a afirmação de que as FET são analíticas e limitadas por quase todo seu domínio, e uma das conclusões do autor é que, para a maioria dos problemas práticos em engenharia, os resultados convergem com probabilidade igual a 1 quando utilizamos FET complexas como FA.

(Kim e Adali, 2003) ainda fazem a observação sobre a restrição $u_x v_y \neq v_x u_y$, que é imposta para garantir o aprendizado contínuo, isto é, prevenir a situação onde um sinal de entrada diferente de zero para a FA force o gradiente da função de erro se tornar nulo. Esta condição, segundo o autores, é desnecessária para FET complexas *fully*, porque todas satisfazem as equações de Cauchy-Riemann (3.7), o que leva a concluir que a restrição $u_x v_y \neq v_x u_y$ só faz sentido para funções complexas não-analíticas *split*.

Uma das importantes conclusões do trabalho dos autores é resumida na proposição a seguir que, segundo suas próprias palavras, "relaxa e reduz" as condições e características desejáveis de FA complexas do tipo *fully*:

Proposição 1 (Taehwan Kim e Tülay Adali) *"Em um domínio limitado do plano complexo* $\mathbb{C}$*, uma FA* fully *complexa não-linear* $f(z)$ *necessita ser analítica e limitada, para ser considerada satisfatória."*

[2]Funções que podem ser reescritas, utilizando-se o termo exponencial e^z.

3.5 Algoritmo *Backpropagation* Complexo

A estrutura de uma rede MLP complexa é semelhante à MLP real discutida na seção 2.4 e a diferença principal se encontra no valores dos sinais de entrada, pesos sinápticos, sinais de saída e alvos, que são todos números complexos. Para podermos definir com precisão o algoritmo de aprendizado complexo, é necessário escolhermos uma FA adequada, pois esta escolha influência diretamente, dependendo da aplicação para a qual ela é destinada a resolver.

A correção dos pesos complexos baseia-se na regra delta, modificada para valores complexos, onde a soma dos erros quadráticos E agora é descrita em função da parte real e da parte imaginária do peso complexo. Logo, a correção é expressa por:

$$\Delta w_{ji} = \eta \left(\frac{\partial E}{\partial w_{jiR}} + i \frac{\partial E}{\partial w_{jiI}} \right) \tag{3.16}$$

Onde os subíndices R e I indicam a contribuição das partes real e imaginária dos pesos complexos e η a taxa de aprendizado. O algoritmo CBP é executado da mesma maneira como é feito no domínio real: duas fases (propagação e retropropagação) em que primeiramente são obtidos os sinais de saída da rede e, a partir do erro de comparação, é feita a correção da última camada até a primeira.

No domínio real, a correção dos pesos está em função da derivada da FA $f'(net_j)$. Entretanto, no domínio complexo, o cálculo da variação do peso pode ser feito de duas maneiras: pelas derivadas parciais, caso a FA seja uma das não-inteiras limitadas, apresentadas na seção 3.4.2, ou pela derivada total, caso a FA seja uma das inteiras não-limitadas, apresentadas na seção 3.4.1.

3.5.1 Correção Pelas Derivadas Parciais

A correção dos pesos de uma RNA complexa pode ser feita a partir das derivadas parciais de uma FA não-inteira limitada, desde que as condições estabelecidas na seção 3.4.3, para este tipo de função, sejam satisfeitas. Considere o neurônio da Figura (3.1) pertencente à camada de saída de uma rede MLP complexa e a Equação (3.1). Derivando-se as variáveis a_j e d_j em função da parte real e imaginária dos pesos, obtemos:

$$\frac{\partial a_j}{\partial w_{jiR}} = X_{iR}; \qquad \frac{\partial d_j}{\partial w_{jiR}} = X_{iI}; \qquad \frac{\partial a_j}{\partial w_{jiI}} = -X_{iI}; \qquad \frac{\partial d_j}{\partial w_{jiI}} = X_{iR} \tag{3.17}$$

O MSE está em função de $u_j(a_j, d_j)$ e $v_j(a_j, d_j)$, assim como a_j está em função de w_{jiR} e d_j de w_{jiI}. Então, usando a regra da cadeia e de posse das Equações (3.4) e (3.17), a parte real

e a imaginária do gradiente podem ser escritas conforme mostram as Equações (3.18).

$$\frac{\partial E}{\partial w_{jiR}} = \frac{\partial E}{\partial u_j}\left(\frac{\partial u_j}{\partial a_j}\frac{\partial a_j}{\partial w_{jiR}} + \frac{\partial u_j}{\partial d_j}\frac{\partial d_j}{\partial w_{jiR}}\right) + \frac{\partial E}{\partial v_j}\left(\frac{\partial v_j}{\partial a_j}\frac{\partial a_j}{\partial w_{jiR}} + \frac{\partial v_j}{\partial d_j}\frac{\partial d_j}{\partial w_{jiR}}\right) \quad (3.18a)$$

$$\frac{\partial E}{\partial w_{jiI}} = \frac{\partial E}{\partial u_j}\left(\frac{\partial u_j}{\partial a_j}\frac{\partial a_j}{\partial w_{jiI}} + \frac{\partial u_j}{\partial d_j}\frac{\partial d_j}{\partial w_{jiI}}\right) + \frac{\partial E}{\partial v_j}\left(\frac{\partial v_j}{\partial a_j}\frac{\partial a_j}{\partial w_{jiI}} + \frac{\partial v_j}{\partial d_j}\frac{\partial d_j}{\partial w_{jiI}}\right) \quad (3.18b)$$

Definindo $\delta_j = -\frac{\partial E}{\partial u_j} - i\frac{\partial E}{\partial v_j}$ e com as equações (3.18), obtemos as Equações (3.19).

$$\partial E/\partial w_{jiR} = -\delta_{jR}\left(u_{a_j}x_{iR} + u_{d_j}x_{iR}\right) + \delta_{jI}\left(v_{d_j}x_{iR} + v_{d_j}x_{iR}\right) \quad (3.19a)$$

$$\partial E/\partial w_{jiI} = -\delta_{jR}\left(u_{a_j}\left(-x_{iI}\right) + u_{d_j}x_{iI}\right) + \delta_{jI}\left(v_{d_j}\left(-x_{iI}\right) + v_{d_j}x_{iI}\right) \quad (3.19b)$$

Desta maneira, o gradiente é definido como a Equação (3.20).

$$\nabla_{w_{ji}}E = -x_i\left(\left(u_{a_j} + iu_{d_j}\right)\delta_{jR} + \left(v_{a_j} + iv_{d_j}\right)\delta_{jI}\right) \quad (3.20)$$

Nota-se que, analogamente ao domínio real, em que o gradiente era calculado em função da entrada do neurônio, do termo δ_j e da derivada $f'(net_j)$, o gradiente no domínio complexo, para FA não-inteiras limitadas, é calculado em função: da entrada do neurônio x_i, o termo δ_j e as derivadas parciais u_{a_j}, u_{d_j}, v_{a_j}, v_{d_j}. Assim, para minimizar o erro E, cada peso complexo w_{ji} deve ser alterado pela quantidade Δw_{ji}, que é proporcional ao gradiente negativo, como mostra a Equação (3.21), onde o sobrescrito $(*)$ indica o complexo conjugado.

$$\Delta w_{ji} = \eta x_i^*\left[\left(u_{a_j} + iu_{d_j}\right)\delta_{jR} + \left(v_{a_j} + iv_{d_j}\right)\delta_{jI}\right] \quad (3.21)$$

(Buchholz, 2005) Logo, a regra de atualização dos pesos complexos é análoga à mostrada na Equação (2.20). Como mostrado na seção 2.6, o termo δ_j pode pertencer a um neurônio na saída da rede ou a um neurônio escondido. (Yi, 2004) sugere que sobre a definição $\delta_j = -\frac{\partial E}{\partial u_j} - i\frac{\partial E}{\partial v_j}$ seja desenvolvida a regra da cadeia, de tal forma, a podermos isolar os termos da derivada da FA e ficar evidente o produto *termo-a-termo*[3], pois dessa maneira obteremos uma expressão semelhante à regra comum do domínio real, que é nitidamente mais compreensível. Assim, listamos os δ_j para o neurônio e saída e intermediário, nesta ordem, e por último a simplificação evidente da Equação (3.21) decorrente da simplificação dos δ_j propostos.

$$\delta_j = (t_j - o_j) \odot f'(net_j) \quad (3.22)$$

$$\delta_j = \left(\sum_k \delta_k w_{jk}^*\right) \odot f'(net_j) \quad (3.23)$$

$$w_{ji}^{new} = w_{ji}^{old} + \eta \cdot \delta_j \cdot x_i^* \quad (3.24)$$

[3] $\odot$ denota o produto termo-a-termo. Ex: $(x + iy) \odot (x' + iy') = xx' + iyy'$.

3.5.2 Correção Pela Derivada Total

Como evidenciado na seção 3.5.1, as Equações (3.22) e (3.23) são utilizadas para FA que não possuem derivadas totais. Para as equações análogas, usando a derivada total, simplesmente substituiremos o produto termo-a-termo pelo produto usual. Assim, as equações δ_j para o neurônio de saída e intermediário serão, respectivamente:

$$\delta_j = (t_j - o_j) f'(net_j^*) \tag{3.25}$$

$$\delta_j = \left(\sum_k \delta_k w_{jk}^* \right) f'(net_j^*) \tag{3.26}$$

A expressão para a regra de atualização dos pesos, usando a derivada total, é semelhante à Equação (3.24), mudando apenas o valor de δ_j correspondente às Equações (3.25) e (3.26). Detalhes sobre as demonstrações e relações apresentadas nesta seção podem ser encontradas em (Georgiou e C, 1992) e (Yi, 2004).

3.6 Normalização de Números Complexos

A normalização de um número complexo pode ser feita de duas maneiras: normalizando a parte real e a imaginária, separadamente, ou normalizando pelo módulo. O problema de se normalizar separadamente a parte real e a imaginária é que a fase θ do número normalizado é diferente da fase do número original, o que dificulta a recuperação dessa informação no processo de desnormalização. Então, optamos pela normalização dos dados complexos, usando o módulo, pois este não muda a informação da fase entre o número original e o normalizado.

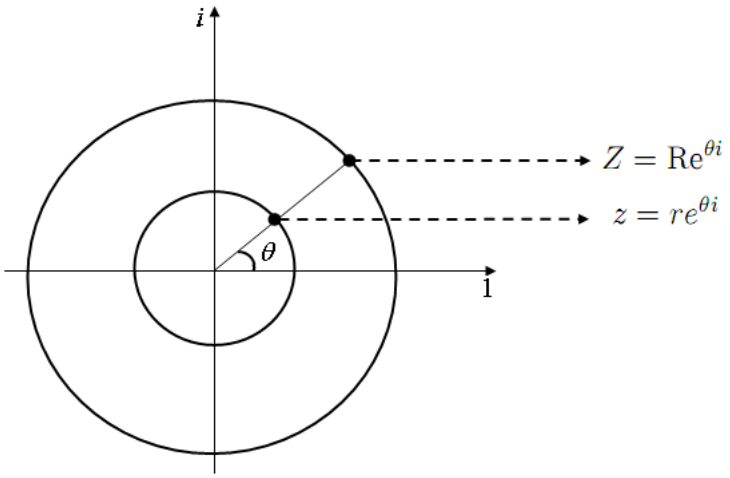


Figura 3.12: Normalização de números complexos com preservação da fase

A idéia da normalização pelo módulo, que pode ser visualizada na Figura (3.12), consiste simplesmente em levar o número complexo que está definido no raio R para o raio r. Esta

idéia também permite o caminho inverso, ou seja, levar um número do raio r para o R. Assim, podemos considerar:

- $\mathbf{Z} = Re^{i\theta}$ é o número a ser normalizado e R seu respectivo raio;
- $z = re^{i\theta}$ é o número normalizado e r seu respectivo raio.

Assim, seguem as respectivas equações de normalização e desnormalização:

$$z = r\frac{\mathbf{Z}}{|\mathbf{Z}|} \tag{3.27a}$$

$$\mathbf{Z} = R\frac{z}{|z|} \tag{3.27b}$$

Os experimentos de (W. L. Chan e Lai, 2000) e (Travessa, 2006) sugerem que o raio de normalização r fique em torno de 0.1, pois neste raio os autores, em seus trabalhos, garantiram o treinamento das redes, evitando o *overflow*. Já (Kim, 2001) e (Benvenuto e Piazza, 1992) recomendam o raio de normalização r unitário. Todavia, é consenso que tanto a parte real quanto a parte complexa sejam valores pequenos, situados no intervalo $(0, 1]$.

CAPÍTULO

4

Álgebras Geométricas ou de Clifford

Sejam $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_i, \ldots, \mathbf{e}_n$ vetores pertencentes ao espaço vetorial V_n n-dimensional sobre os reais e λ qualquer escalar real. A álgebra geométrica $Cl_{p,q} = Cl(V_n)$ com $n = p + q$ é gerada a partir do espaço vetorial V_n onde é definido o produto geométrico $\otimes$ o qual guarda as seguintes propriedades sobre todos os vetores:

$$\mathbf{e}_1 \otimes (\mathbf{e}_2 \otimes \mathbf{e}_3) = (\mathbf{e}_1 \otimes \mathbf{e}_2) \otimes \mathbf{e}_3 \tag{4.1a}$$

$$\mathbf{e}_1 \otimes (\mathbf{e}_2 + \mathbf{e}_3) = \mathbf{e}_1 \otimes \mathbf{e}_2 + \mathbf{e}_1 \otimes \mathbf{e}_3 \tag{4.1b}$$

$$(\mathbf{e}_2 + \mathbf{e}_3) \otimes \mathbf{e}_1 = \mathbf{e}_2 \otimes \mathbf{e}_1 + \mathbf{e}_3 \otimes \mathbf{e}_1 \tag{4.1c}$$

$$\mathbf{e}_i \lambda = \lambda \mathbf{e}_i \tag{4.1d}$$

$$(\mathbf{e}_i)^2 = \pm |\mathbf{e}_i|^2 \tag{4.1e}$$

onde $|\mathbf{e}_i|$ é um escalar positivo associado a $\mathbf{e}_i$. O axioma (4.1e) é chamado *contraction rule*. Diz-se que vetor $\mathbf{e}_i$ tem assinatura positiva (ou negativa) quando o sinal em (4.1e) é especificado como positivo (ou negativo), e o vetor $\mathbf{e}_i$ é chamado de vetor nulo se $|\mathbf{e}_i| = 0$ quando $\mathbf{e}_i \neq 0$. As álgebras de Clifford têm diversas assinaturas $Cl_{p,q}$, onde: p é a quantidade de vetores de base $\mathbf{e}_i$ que tem assinatura positiva $(+1)$ e q é a quantidade de vetores de base $\mathbf{e}_i$ que tem assinatura negativa (-1).

$$\begin{aligned} \overbrace{\mathbf{e}_i \otimes \mathbf{e}_j}^{\text{produto geométrico}} &= \overbrace{\mathbf{e}_i \vee \mathbf{e}_j}^{\text{produto interno}} + \overbrace{\mathbf{e}_i \wedge \mathbf{e}_j}^{\text{produto exterior}} \\ &= \tfrac{1}{2}(\mathbf{e}_i \mathbf{e}_j + \mathbf{e}_j \mathbf{e}_i) + \tfrac{1}{2}(\mathbf{e}_i \mathbf{e}_j - \mathbf{e}_j \mathbf{e}_i) \end{aligned} \tag{4.2}$$

O produto geométrico, Equação (4.2), é a combinação linear de dois produtos: o *produto interno* $(\vee)$ e o *produto exterior* $(\wedge)$. O produto interno, devido ao axioma (4.1e), retorna ape-

nas valores escalares e é definido como a parte simétrica do produto geométrico, enquanto o segundo é definido como a parte anti-simétrica do produto geométrico e apenas retorna valores definidos como k-*blades* (Hestenes e Ziegler, 1991). Qualquer elemento gerado pelo produto exterior $(\wedge)$ de qualquer quantidade de vetores $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_k$ é chamado de k-*blade* ou blade de grau k, *grade* k do inglês, e é expresso de acordo com a Equação (4.3). Quais quer combinação linear de k-*blades* é chamada de k-*vector* ou multivetor. Todos os k-*blades* de quaisquer assinatura satisfazem a relação (4.4).

$$\langle \mathbf{e}_1 \mathbf{e}_2 \cdots \mathbf{e}_k \rangle_k = \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \cdots \wedge \mathbf{e}_k \tag{4.3}$$

$$\langle \mathbf{e}_1 \mathbf{e}_2 \cdots \mathbf{e}_k \rangle_k = -\langle \mathbf{e}_k \cdots \mathbf{e}_2 \mathbf{e}_1 \rangle_k \tag{4.4}$$

As álgebras de Clifford de assinatura $Cl_{p,q}$, com $n = p + q$, geram exatamente 2^n combinações linearmente independentes entre a unidade escalar e os vetores de base através do produto geométrico. Tomemos como exemplo a assinatura $Cl_{0,3}$, esta assinatura tem três vetores de base $\mathbf{e}_i$ que têm assinatura negativa, pois $q = 3$, e os elementos de base desta assinatura são $\{1, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ conseqüentemente esta assinatura é composta por $2^3 = 8$ termos linearmente independentes os quais são:

$$\{1,\ \mathbf{e}_1,\ \mathbf{e}_2,\ \mathbf{e}_3,\ \mathbf{e}_1 \wedge \mathbf{e}_2,\ \mathbf{e}_1 \wedge \mathbf{e}_3,\ \mathbf{e}_2 \wedge \mathbf{e}_3,\ \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3\}$$

Os k-*blades* nada mais são do que subespaços orientados (Dorst e Mann, 2002a), onde o produto geométrico permite a modelagem dos espaços vetoriais conhecidos através destes objetos (Dorst et al., 2007). Estes elementos são os **escalares** (0-*blades*), **vetores** (1-*blades*), segmentos de plano orientados ou **bivetores** (2-*blades*) e volumes orientados ou **trivetores** (3-*blades*), como mostra a Fig.4.1, a qual exibe uma possível representação destes subespaços. Dependendo da assinatura da álgebra de Clifford, esta pode ter a álgebra dos números com-

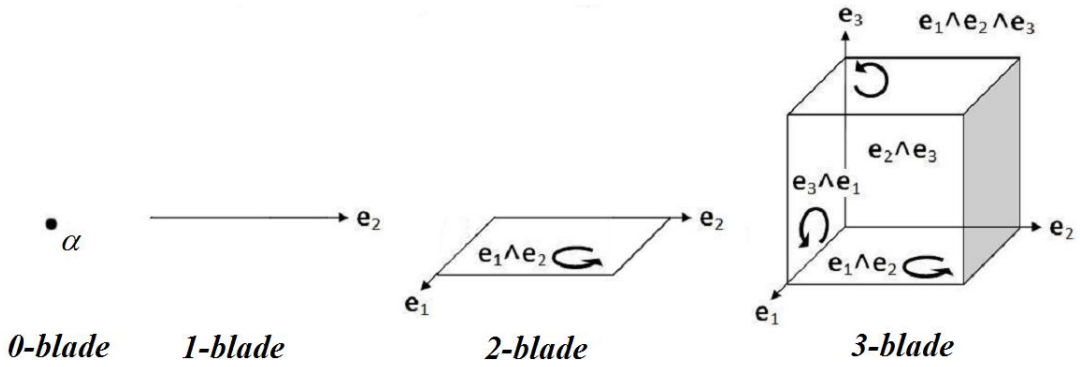
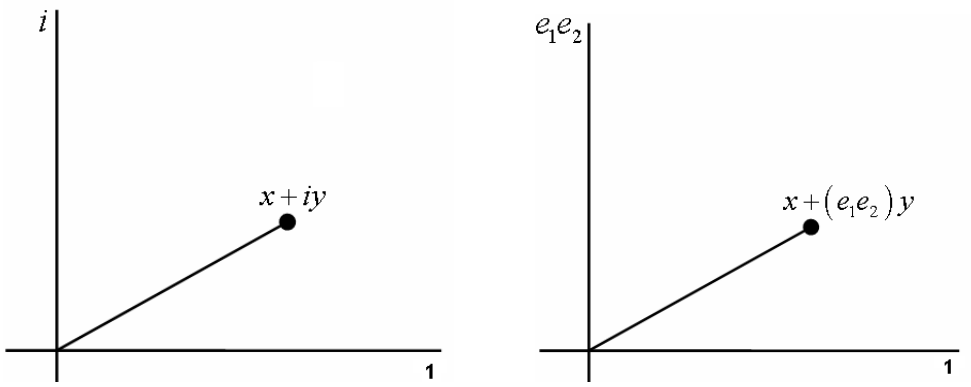


Figura 4.1: k-*blades*: subespaços orientados

plexos como subespaço (Dorst et al., 2007), como é o exemplo da assinatura $Cl_{2,0}$ que tem a álgebra dos números complexos expandida sobre $\{1, \mathbf{e}_1 \wedge \mathbf{e}_2\}$ onde $(\mathbf{e}_1 \wedge \mathbf{e}_2)^2 = -1$ é equivalente à unidade imaginária do plano complexo, que é o objeto deste trabalho para efeitos de

comparação com o domínio complexo. A Figura (4.2) exemplifica esta comparação. Para facilitar a representação dos k-*blades* iremos utilizar a seguinte notação $\mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \cdots \wedge \mathbf{e}_n = \mathbf{e}_1\mathbf{e}_2 \cdots \mathbf{e}_n$.



(a) Plano complexo gerado pela base $\{1, i\}$ (b) Plano complexo gerado pela base $\{1, \mathbf{e}_1\mathbf{e}_2\}$

Figura 4.2: Plano complexo gerado por duas bases distintas

4.1 Segmentos de Área e Volume Orientados

Esta seção se dedica a explanar um pouco sobre os elementos básicos das álgebras de Clifford apresentados na seção anterior bem como apresentar as características geométricas que desses elementos trazem por definição.

4.1.1 Vetores e Escalares

O conceito de **vetor** foi criado para descrever grandezas que agregam a si propriedades geométricas tais como: velocidade de um corpo que por definição tem sentido, direção e magnitude. Sua representação se faz da seguinte maneira: o comprimento do vetor informa a magnitude da grandeza, a direção desta é determinada pela reta-suporte e o seu sentido por uma flecha colocada em uma das suas extremidades, como pode ser observado na Figura (4.3). Em sentido oposto, temos as grandezas adimensionais que não necessitam de uma representação vetorial, e são chamadas **escalares** (Vieira, 2005).

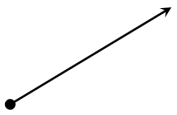
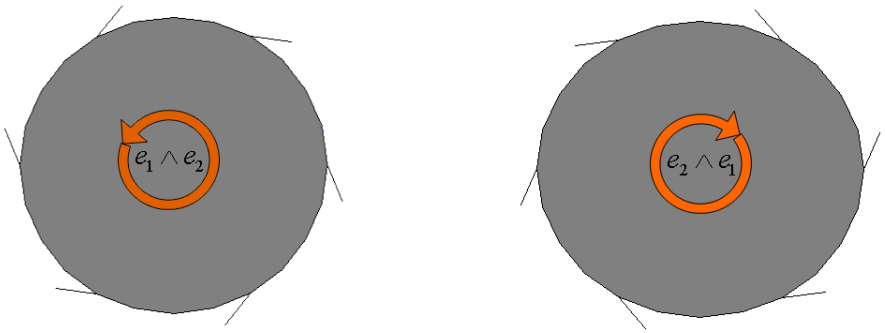


Figura 4.3: Um vetor

4.1.2 Bivetores e Trivetores

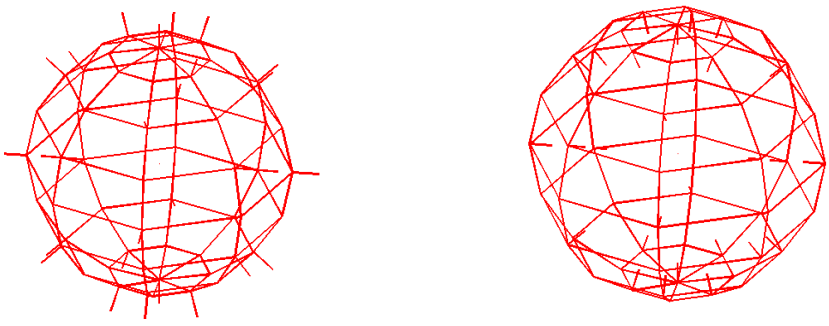
Tais como os vetores, os **bivetores** e **trivetores** foram pensados para representar grandezas que têm características geométricas associadas a elas por definição. Um bom exemplo é a grandeza campo elétrico que se propaga em três dimensões. Esta grandeza fica melhor representada por volumes do que por vetores. O mesmo acontece para grandezas de superfície. Assim, os bivetores são definidos como fragmentos de área orientados, onde o valor de sua área informa a magnitude da grandeza por ele representada, a direção da grandeza é determinada pela direção do plano-suporte do bivetor e admite dois possíveis sentidos: horário ou anti-horário, conforme demonstra a Figura (4.4).



(a) Bivetor $\mathbf{e}_1 \wedge \mathbf{e}_2$ anti-horário (b) Bivetor $\mathbf{e}_2 \wedge \mathbf{e}_1$ horário

Figura 4.4: Possíveis orientações do bivetor $\mathbf{e}_1 \wedge \mathbf{e}_2$

Já o **trivetor** é um volume orientado onde o volume determina a magnitude e também admite duas orientações, como pode ser observado na Figura (4.5). As formas desses objetos podem variar: os fragmentos de plano podem ser representados por paralelogramos ou discos; os volumes, por paralelepípedos ou esferas, ou seja, podemos escolher a forma que desejarmos para estes elementos. Segundo (Suter, 2003) e (Dorst et al., 2007), a forma pouco importa. O que se deve dar atenção é à capacidade desses elementos de representar todo o espaço de interesse.



(a) Trivetor $\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$ com orientação para fora da esfera. (b) Trivetor $-\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$ com orientação para dentro da esfera.

Figura 4.5: Possíveis orientações do trivetor $\mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3$

4.1.3 Multivetores

Um **multivetor** é uma combinação linear de diferentes k-blades como mostra o exemplo do multivetor definido em uma álgebra 3-dimensional $\mathbb{R}^3 \equiv Cl_{3,0}$ o qual necessita de $2^3 = 8$ termos linearmente independentes para a sua representação completa como mostra a representação abaixo onde λ_i são os coeficientes reais de cada k-*blade*:

$$\underbrace{\lambda_1}_{\text{escalar}} + \underbrace{\lambda_2\mathbf{e}_1 + \lambda_3\mathbf{e}_2 + \lambda_4\mathbf{e}_3}_{\text{vetores}} + \underbrace{\lambda_5\mathbf{e}_1\mathbf{e}_2 + \lambda_6\mathbf{e}_2\mathbf{e}_3 + \lambda_7\mathbf{e}_3\mathbf{e}_1}_{\text{bivetores}} + \underbrace{\lambda_8\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3}_{\text{trivetores}}$$

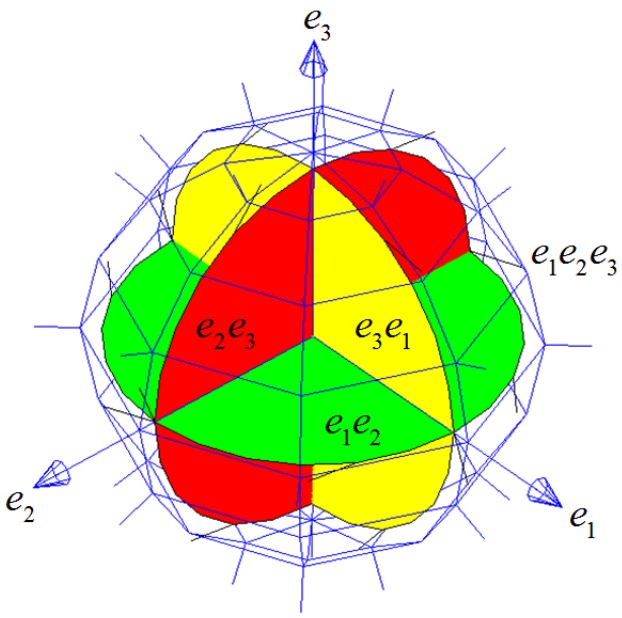


Figura 4.6: Multivetor

O multivetores também são conhecidos como **números de Clifford**. De maneira análoga, podemos ter multivetores de álgebra de maiores dimensões, porém a visualização de todos os objetos já não é possível. A Figura (4.6) mostra uma possível representação gráfica de um multivetor na álgebra $Cl_{3,0}$ onde o mesmo nada mais é do que uma combinação linear dos escalares,que não tem representação gráfica, e dos possíveis k-*blades* que compoem a álgebra em questão.

Definidos estes objetos básicos, a proxima seção irá discutir os principais operadores que atuam sobre esses objetos. Mais adiante, outros operadores úteis e necessários para implementar o *BackPropagation* de Clifford serão apresentados. Informações mais detalhadas sobre os diversos elementos das álgebras de Clifford podem ser encontradas em (Hestenes, 2003), (Dorst e Mann, 2002a), (Dorst e Mann, 2002b), (Mann et al., 2001), (Arthan, 2006), (Paiva, 2007).

4.2 Operadores Geométricos

Esta seção se dedica a apresentar e caracterizar os operadores das álgebras de Clifford necessários para a implementação computacional das RNA de Clifford, caso o leitor tenha familiaridade com os mesmos é sugerido seguir com a leitura para o proximo capítulo.

4.2.1 Inversa Geométrica

O produto geométrico de Clifford é invertível e único. Seja $\mathbf{A}$ um multivetor **não-nulo** e seu inverso dado por $\mathbf{A}^{-1}$, onde:

$$\mathbf{A}^{-1} = \frac{\tilde{\mathbf{A}}}{\mathbf{A} \vee \tilde{\mathbf{A}}} \tag{4.5}$$

O termo $\tilde{\mathbf{A}}$ é chamado de **reverso** do multivetor $\mathbf{A}$ e é obtido, invertendo-se a ordem dos fatores dos k-*blades* com *grade* $k \geq 2$. Assim, se $\mathbf{A} = \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \ldots \wedge \mathbf{e}_k$, o seu reverso será $\tilde{\mathbf{A}} = \mathbf{e}_k \wedge \cdots \wedge \mathbf{e}_2 \wedge \mathbf{e}_1$. Esta inversa satisfaz $\mathbf{A}^{-1}\mathbf{A} = 1 = \mathbf{A}\mathbf{A}^{-1}$ para qualquer elemento não-nulo da álgebras de Clifford (Dorst e Mann, 2002a).

4.2.2 Norma de Multivetores

A norma ou valor absoluto de multivetores é uma extensão do valor absoluto de números complexos. Tomemos como exemplo o multivetor $\mathbf{A}$:

$$\mathbf{A} = \lambda_1 + \lambda_2\mathbf{e}_1 + \lambda_3\mathbf{e}_2 + \lambda_4\mathbf{e}_3 + \lambda_5\mathbf{e}_1\mathbf{e}_2 + \lambda_6\mathbf{e}_2\mathbf{e}_3 + \lambda_7\mathbf{e}_3\mathbf{e}_1 + \lambda_8\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$$

A norma $||\mathbf{A}||$ deste multivetor é dada pela expressão:

$$||\mathbf{A}|| = \sqrt{\lambda_1^2 + \lambda_2^2 + \lambda_3^2 + \lambda_4^2 + \lambda_5^2 + \lambda_6^2 + \lambda_7^2 + \lambda_8^2}$$

que nada mais é que a raiz quadrada da soma de cada coeficiente ao quadrado dos termos do multivetor $\mathbf{A}$. No caso geral, para qualquer álgebra de assinatura $Cl_{p,q}$, a norma de um multivetor $\mathbf{A}$ é dada pela raiz quadrada do produto geométrico entre o multivetor e seu reverso[1], tomando do resultado do produto a soma dos escalares, como mostra a Equação 4.6.

$$||\mathbf{A}|| = \sqrt{\left\langle \mathbf{A} \otimes \tilde{\mathbf{A}} \right\rangle_0} \tag{4.6}$$

[1] Veja a definição de reverso na seção 4.2.1.

4.2.3 Conjugado Geométrico

As álgebras de Clifford são uma extensão n-dimensional da álgebra dos números complexos (Arena et al., 1997). A definição do complexo conjugado de um multivetor é de suma importância para se estender o algoritmo BP para o domínio das álgebras de Clifford (Yi, 2004). No domínio complexo, para encontrarmos o complexo conjugado z^* de um número z, basta que se inverta o sinal da parte imaginária de z. Nas álgebras de Clifford, os multivetores são combinações lineares de diferentes k-*blades* e o complexo conjugado de cada um deles depende do *grade* k ao qual o blade pertence (Mann et al., 1999). Dado o multivetor:

$$\mathbf{A} = \underbrace{\lambda_1}_{0\text{-}blade} + \underbrace{\lambda_2\mathbf{e}_1 + \lambda_3\mathbf{e}_2 + \lambda_4\mathbf{e}_3}_{1\text{-}blades} + \underbrace{\lambda_5\mathbf{e}_1\mathbf{e}_2 + \lambda_6\mathbf{e}_2\mathbf{e}_3 + \lambda_7\mathbf{e}_3\mathbf{e}_1}_{2\text{-}blades} + \underbrace{\lambda_8\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3}_{3\text{-}blades}$$

Para computarmos o complexo conjugado $\mathbf{A}^*$ do multivetor $\mathbf{A}$, a inversão do sinal de cada um dos k-*blades* é dada pela expressão $(-1)^{k_m \frac{k_m+1}{2}}$, onde $k_m = k \bmod 4$. A Tabela (4.1) descreve esta relação.

k-*blade*	$k_m = k \bmod 4$	*sinal do conjugado*
0-*blade*	0	$+$
1-*blade*	1	$-$
2-*blade*	2	$-$
3-*blade*	3	$+$
4-*blade*	0	$+$
5-*blade*	1	$+$
$\vdots$	$\vdots$	$\vdots$

Tabela 4.1: Sinal do conjugado do *blade* de grade k

Logo, o conjugado do multivetor $\mathbf{A}$ será:

$$\mathbf{A}^* = \underbrace{\lambda_1}_{0\text{-}blade} - \underbrace{\lambda_2\mathbf{e}_1 - \lambda_3\mathbf{e}_2 - \lambda_4\mathbf{e}_3}_{1\text{-}blade} - \underbrace{\lambda_5\mathbf{e}_1\mathbf{e}_2 - \lambda_6\mathbf{e}_2\mathbf{e}_3 - \lambda_7\mathbf{e}_3\mathbf{e}_1}_{2\text{-}blade} + \underbrace{\lambda_8\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3}_{3\text{-}blade}$$

4.2.4 Função Exponencial de Multivetores

Para um multivetor $\mathbf{A}$, qualquer função exponencial $\exp(\mathbf{A})$ é definida como a expansão polinomial (de Sabbata & Bidyut Kumar Datta, 2007):

$$\begin{aligned} \exp(\mathbf{A}) &= 1 + \tfrac{\mathbf{A}}{1!} + \tfrac{\mathbf{A}^2}{2!} + \tfrac{\mathbf{A}^3}{3!} + \cdots \\ &= \sum_{k=0}^{\infty} \tfrac{\mathbf{A}^k}{k!} \end{aligned} \tag{4.7}$$

Se o quadrado do multivetor **A** é um escalar, então a expansão polinomial em (4.7) pode ser computada, usando-se as funções trigonométricas usuais:

$$\exp(\mathbf{A}) = \begin{cases} \cos(\alpha) + \mathbf{A}\frac{\text{sen}(\alpha)}{\alpha}, & \text{se } \mathbf{A}^2 = -\alpha^2 \\ 1 + \mathbf{A}, & \text{se } \mathbf{A}^2 = 0 \\ \cosh(\alpha) + \mathbf{A}\frac{\text{senh}(\alpha)}{\alpha}, & \text{se } \mathbf{A}^2 = \alpha^2 \end{cases} \tag{4.8}$$

onde α é um escalar real. Informações adicionais sobre implementação computacional da Equação (4.7) podem ser encontradas em (Dorst et al., 2007), (Mann et al., 1999) e (Girard, 2007).

4.2.5 Diferenciação de Multivetores

A definição da derivada multivetorial está associada às definições de diferenciação escalar e diferenciação vetorial. Mais precisamente, ela engloba as duas definições e é dada pela expressão:

$$(\mathbf{A} \cdot \partial_{\mathbf{X}}) F(\mathbf{X}) \equiv \lim_{\varepsilon \to 0} \frac{F(\mathbf{X} + \varepsilon \mathbf{A}) - F(\mathbf{X})}{\varepsilon} \tag{4.9}$$

onde $F(X)$ é a função a ser derivada sobre o multivetor X e A é o multivetor de mesmo grade de X que dá a orientação da derivada. Como este operador tem característica semelhante à derivada escalar e vetorial, (Dorst et al., 2007) e (Hestenes e Sobczyk, 1984) mostram que a soma das derivadas parciais em cada dimensão de atuação de $F(X)$ nos fornece a informação da variação do multivetor na direção de A. Podemos citar como exemplo o trabalho de (Yi, 2004) que usou a função logística $F(X) = 1/(1 + \exp(-X))$ e sua respectiva derivada $F'(X) = F(X)[1 - F(X)]$, ambas definidas na álgebra tridimensional $Cl_{3,0}$ para construir o algoritmo BP de Clifford, lembrando que neste caso $A = 1$ é um escalar ou um multivetor de *grade* igual a zero.

4.2.6 Rotações Geométricas

Dependendo da assinatura da álgebra de Clifford, esta pode ter a álgebra dos números complexos como um subespaço (Yi, 2004) e (Kim e Adali, 2003). A vantagem de se fazer tal generalização é a de podermos definir a função exponencial de um bivetor como um objeto que nos permitirá fazer rotações com quaisquer elementos da álgebra de Clifford.

Este objeto chama-se **rotor** e é definido como o produto em ***sandwich*** do elemento x que se quer rotacionar (Dorst e Mann, 2002b). Sua Equação é dada por (4.10) .

$$x \mapsto \begin{cases} \overrightarrow{R} x \overleftarrow{R}, & \text{rotação no sentido anti-horário} \\ \overleftarrow{R} x \overrightarrow{R}, & \text{rotação no sentido horário} \end{cases} \tag{4.10}$$

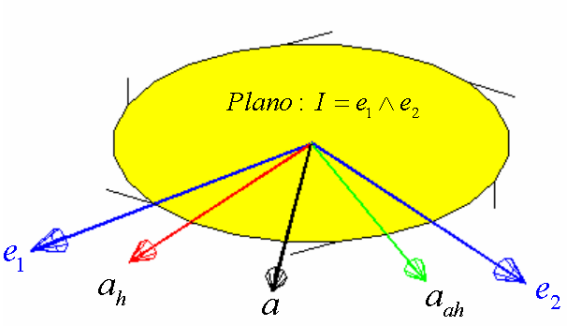


Figura 4.7: Rotações do vetor a sobre o plano I

onde x pode ser qualquer elemento da álgebra de Clifford e $\overrightarrow{\overleftarrow{R}}$ é igual:

$$\overrightarrow{\overleftarrow{R}} = \exp\left(\pm I\frac{\theta}{2}\right) = \cos\left(\frac{\theta}{2}\right) \pm I\text{sen}\left(\frac{\theta}{2}\right) \tag{4.11}$$

Observando a Equação (4.11), podemos notar que ela é semelhante à equação de rotação no domínio complexo. A diferença está no termo $\pm I = \pm\mathbf{e}_i \wedge \mathbf{e}_j$, que é um bivetor de sentido anti-horário caso I ou horário $-I$. Na Figura (4.7), temos como exemplo o plano $I = \mathbf{e}_1 \wedge \mathbf{e}_2$ e um vetor $\mathbf{a}$ neste plano de fase igual a 45 graus. Os dois vetores restantes são, respectivamente, rotações de 30 graus no sentido horário (vetor $\mathbf{a}_h$) e anti-horário (vetor $\mathbf{a}_{ah}$) do vetor $\mathbf{a}$ e são computados respectivamente: $\mathbf{a}_h = \overleftarrow{R}\mathbf{a}\overrightarrow{R}$, $\mathbf{a}_{ah} = \overrightarrow{R}\mathbf{a}\overleftarrow{R}$.

CAPÍTULO

5

Redes Neurais Artificiais de Clifford

As RNA de Clifford têm em comum com as RNA reais a **topologia de rede**, **modos de treinamento**, **critérios de parada** e **validação do treinamento**. Para essas redes, os valores das entradas, pesos, *bias* e saídas das FA são todos **multivetores** da álgebra de Clifford. A escolha da FA de domínio de Clifford também é um ponto a ser considerado, visto que esta escolha afeta diretamente a adaptação do algoritmo BP para as álgebras de Clifford. Nesta seção, apresentaremos os principais conceitos de RNA do domínio de Clifford, seus elementos básicos e suas respectivas características.

5.1 O Neurônio Artificial de Clifford

O modelo do neurônio artificial de Clifford, Figura (5.1), é muito semelhante ao modelo real e complexo. A diferença está nos valores das suas componentes x_i, b_j, net_j, o_j, que agora são todos números de Clifford ou multivetores.

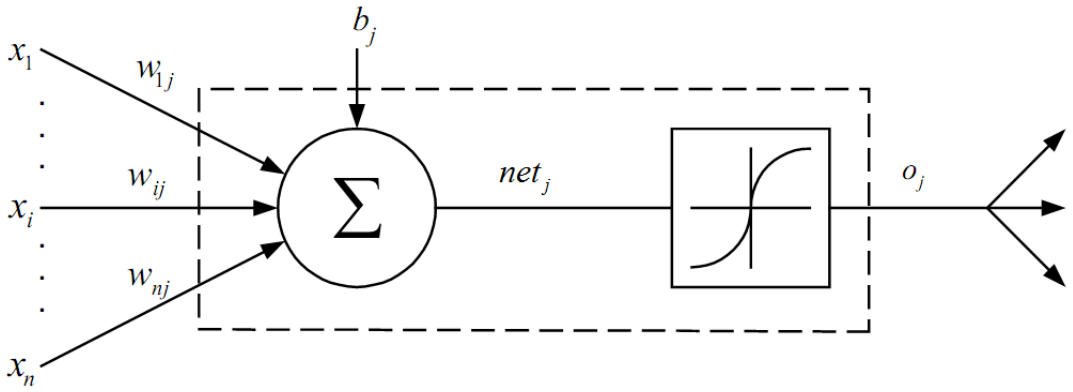


Figura 5.1: O neurônio artificial de Clifford

E assim, no domínio real e complexo, o valor o_j é computado como aplicação da FA sobre o valor net_j. Contudo, no lugar do produto usual temos o produto geométrico $\otimes$. Assim, as Equações (2.1) e (2.2) no domínio de Clifford são definidas como:

$$net_j = \sum_i w_{ji} \otimes x_i + b_j \tag{5.1}$$

$$o_j = f(net_j) \tag{5.2}$$

5.2 Funções de Ativação de Clifford

As FA de Clifford, tais como as complexas, também são divididas em **funções inteiras não-limitadas** e **funções não-inteiras limitadas**, e as definições, características e propriedades dessas funções discutidas na seção 3.4 também são válidas para o domínio de Clifford (Yi, 2004) e (Laville e Lehman, 2005). Aqui, apresentaremos uma representante de cada tipo, visto que quaisquer outras funções acarretam características semelhantes.

5.2.1 Funções Inteiras Não-Limitadas

Função *Fully* Tangente Hiperbólica de Clifford

Esta função pode ser escrita na forma exponencial, tal como suas contrapartes de domínio real e complexo, porém, o valor net_j e conseqüentemente o_j agora se tratam de multivetores de Clifford de qualquer assinatura $Cl_{p,q}$.

$$o_j = f(net_j) = \frac{e^{net_j} - e^{-net_j}}{e^{net_j} + e^{-net_j}} \tag{5.3}$$

E, de maneira semelhante, podemos definir a FA *fully* logística em Clifford e diversas FA do tipo *fully*.

5.2.2 Funções Não-Inteiras Limitadas

Função *Split* Logística de Clifford

Tal como a função *split* logística complexa, a função *split* logística de Clifford nada mais é do que a aplicação da função logística real definida na Equação (2.4) em cada coeficiente dos termos do multivetor de entrada da função. Tomemos como exemplo multivetor $\mathbf{A}$:

$$\mathbf{A} = \lambda_1 + \lambda_2\mathbf{e}_1 + \lambda_3\mathbf{e}_2 + \lambda_4\mathbf{e}_3 + \lambda_5\mathbf{e}_1\mathbf{e}_2 + \lambda_6\mathbf{e}_2\mathbf{e}_3 + \lambda_7\mathbf{e}_3\mathbf{e}_1 + \lambda_8\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$$

Logo, o valor da função *split* logística de Clifford em A será:

$$f(\mathbf{A}) = \begin{array}{ccccccccc} \frac{1}{1+e^{-\lambda_1}} & + & \frac{1}{1+e^{-\lambda_2}}\mathbf{e}_1 & + & \frac{1}{1+e^{-\lambda_3}}\mathbf{e}_2 & + & \frac{1}{1+e^{-\lambda_4}}\mathbf{e}_3 & + \\ \frac{1}{1+e^{-\lambda_5}}\mathbf{e}_1\mathbf{e}_2 & + & \frac{1}{1+e^{-\lambda_6}}\mathbf{e}_2\mathbf{e}_3 & + & \frac{1}{1+e^{-\lambda_7}}\mathbf{e}_3\mathbf{e}_1 & + & \frac{1}{1+e^{-\lambda_8}}\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 \end{array} \tag{5.4}$$

E, de forma análoga, podemos definir a FA *split* tangente hiperbólica e diversas outras da mesma categoria.

5.3 Algoritmo *Backpropagation* de Clifford

O algoritmo BP de Clifford é muito semelhante ao BP complexo, discutido na seção 3.5, tanto no quesito estrutural quanto na escolha da FA a ser utilizada. As principais diferenças repousam nos valores das entradas, pesos sinápticos, *bias* e saídas da rede. Todavia, este novo domínio de atuação, tal como no domínio complexo, nos leva a adaptar o BP para tratar os sinais da rede que agora são todos multivetores de Clifford. Assim, nesta seção apresentaremos as devidas alterações no BP para processar multivetores de Clifford, levando em consideração a escolha da FA que pode ser inteira não-limitada ou não-inteira limitada.

O erro na saída da rede é dado pela Equação (5.5) e semelhante à Equação (2.9) no domínio real, porém os enésimos alvos $t_j(n)$ e as enésimas saídas $o_j(n)$ agora são multivetores de Clifford. Para facilitar a escrita das equações a seguir, o termo (n), que se refere a enésimo padrão de treinamento, será suprimido de algumas equações que se seguem.

$$e_j = t_j - o_j \tag{5.5}$$

O somatório dos erros quadráticos instantâneos para o BP em Clifford é definido como o quadrado da norma do erro na saída que, por sua vez, pode ser escrito em termos do conjugado de Clifford, como mostra a Equação (5.6):

$$\begin{aligned} E &= \tfrac{1}{2}\sum_j |e_j|^2 \\ &= \tfrac{1}{2}\sum_j \sum_{k=1}^{2^n} \left(\lambda_k^{t_j} - \lambda_k^{o_j}\right)^2 \\ &= \tfrac{1}{2}\left\langle (t_j - o_j) \otimes (t_j - o_j)^* \right\rangle_0 \end{aligned} \tag{5.6}$$

onde o $*$ sobrescrito representa o conjugado de Clifford e $2^n = 2^{p+q}$ é o total de elementos da álgebra de assinatura $Cl_{p,q}$ e $\lambda_k^{t_j}$ e $\lambda_k^{o_j}$ são, respectivamente, os coeficientes dos multivetores t_j e o_j. Assim, a função real que representa o MSE para o domínio de Clifford é definida como:

$$MSE = \frac{1}{N}\sum_{n=0}^{N-1} E(n) \tag{5.7}$$

Com o MSE definido, podemos encontrar as regras de aprendizado para o BP de Clifford de forma análoga às feitas nas seções 2.6 e 3.5. Detalhes sobre estas demonstrações podem ser encontradas em (Yi, 2004), (Buchholz e Sommer, 2000), (Arena et al., 1998) e (Arena et al., 1997). Estas regras são resumidas na Tabela (5.1), onde $\otimes$ representa o produto geométrico, $\odot$ o produto termo-a-termo[1] e o * sobrescrito representa o conjugado de Clifford.

	FA *inteira não-limitada*	FA *não-inteira limitada*
w_{ji}^{new}	$= w_{ji}^{old} + \eta\delta_j \otimes x_i^*$	$= w_{ji}^{old} + \eta\delta_j \otimes x_i^*$
δ_j saída	$= (t_j - o_j) \otimes f'(net_j^*)$	$= (t_j - o_j) \odot f'(net_j)$
δ_j oculto	$= \left(\sum_k w_{jk}^* \otimes \delta_k\right) \otimes f'(net_j^*)$	$= \left(\sum_k w_{jk}^* \otimes \delta_k\right) \odot f'(net_j)$

Tabela 5.1: Regras de aprendizado *backpropagation* de Clifford

[1] $\odot$ denota o produto termo-a-termo Ex: $(x + y\mathbf{e}_2 + z\mathbf{e}_1\mathbf{e}_2) \odot (x' + y'\mathbf{e}_2 + z'\mathbf{e}_1\mathbf{e}_2) = xx' + yy'\mathbf{e}_2 + zz'\mathbf{e}_1\mathbf{e}_2$.

CAPÍTULO

6

Confiabilidade de Redes Neurais Artificiais

O método mais comum para determinar a convergência de uma RNA para um conjunto de treinamento é estabelecer um MSE mínimo e encerrar a correção dos pesos, quando este valor for alcançado. No entanto, a convergência de uma rede neural não garante a sua generalização, pois a rede é utilizada para entradas diferentes daquelas usadas no conjunto de treinamento. Nesta seção, vamos discutir a análise de confiabilidade de RNA, bem como a convergência e generalização dessas redes e propor um novo método para a análise de confiabilidade.

6.1 Quantidade de Padrões vs. Generalização de RNA

Para verificar a generalização de uma RNA, a validação cruzada estratificada é o método mais utilizado. Este método consiste em dividir parte do conjunto de entradas e alvos, reservando 70% a 80% para o treinamento e o restante para a validação (Kohavi, 1995). Após o treinamento, aplica-se à rede neural o conjunto de simulação e comparam-se os valores fornecidos pela RNA com os alvos desse conjunto de simulação. Caso a rede convergir para o MSE especificado e as entradas de simulação respondam dentro da especificação, o treinamento é considerado válido. Um dos problemas neste tipo de validação é determinar o conjunto de padrões de treinamento mais adequado para usar o mapeamento aprendido para qualquer valor de entrada.

Para mostrar a influência da escolha dos padrões de treinamento, suponha que uma RNA seja utilizada para aproximar uma função $y = f(x)$ qualquer. Para que a convergência seja alcançada, esta rede possui um neurônio na camada de entrada correspondente ao domínio da

função, um neurônio na camada de saída correspondente ao contradomínio da função e ao menos uma camada oculta, como demonstrou (Baum e Haussler, 1987). No treinamento desta RNA, são executados cinco passos essenciais:

1. escolha do intervalo das entradas de treinamento;
2. escolha dos padrões de treinamento;
3. especificação do MSE desejado e critérios de parada adicionais, caso seja necessário;
4. escolha de uma topologia mais adequada para a RNA;
5. execução do algoritmo BP para o MSE especificado.

Primeiramente, determina-se um intervalo de treinamento $[x_{min}; x_{max}]$, no qual as entradas nunca poderão estar fora, tanto no teste quanto na utilização. Essa condição é necessária, pois as redes neurais artificiais do tipo MLP não são capazes de extrapolar o conjunto de treinamento. Em seguida, é feita a escolha dos padrões a serem utilizados no aprendizado da RNA. Esta escolha é feita de maneira que o conjunto de entradas e alvos represente a função $y = f(x)$ no intervalo especificado.

O próximo passo é determinar um critério de parada. Como o problema é representar a função por meio de redes neurais, supervisionadas, o melhor critério é especificar um MSE mínimo, pois o algoritmo encerrará apenas quando as saídas fornecidas forem suficientemente próximas do conjunto de comparação, onde o MSE representa o erro a ser tolerado na saída da rede no treinamento. Com relação à topologia ideal para uma RNA, é de conhecimento que ela varia de acordo com a aplicação. Então, cabe ao projetista de redes determinar a melhor topologia para a solução do seu problema. Por último, é implementado o algoritmo BP para treinar uma RNA com os padrões de treinamento escolhidos.

Vamos supor que o MSE foi alcançado para duas situações diferentes:

a) com poucos padrões de treinamento;

b) com muitos padrões de treinamento.

É intuitivo dizer que na situação *b* a RNA responde melhor para entradas diferentes do treino comparado à primeira situação, pois mais exemplos lhe foram apresentados; logo, ela foi capaz de mapear melhor a função, apesar de, nos dois casos, o MSE especificado ter sido atingido, mas, dependendo do tamanho do conjunto a ser analisado, reservar 80% dos dados pode ser dispendioso demais. Assim, uma pergunta se faz necessária, neste âmbito: existe uma forma de se determinar a quantidade suficiente de padrões de treinamento de tal forma a garantir a generalização da RNA? Esta pergunta será alvo de discussão na próxima seção, onde vamos sugerir uma metodologia de abordagem para este problema.

Além do número de padrões influenciar na generalização da RNA, é importante determinar a distribuição adequada deles, pois podemos escolher um número grande de padrões, mas, se concentrarmos a maioria deles em um extremo do intervalo $[x_{min}; x_{max}]$, as entradas distantes destes valores não responderão adequadamente à função $y = f(x)$. Pode-se contornar este problema simplesmente escolhendo valores uniformemente distribuídos entre os extremos x_{min} e x_{max}.

6.2 Metodologia Direta-Inversa

O método proposto neste trabalho para verificar se as saídas fornecidas durante a utilização da rede neural artificial são corretas de acordo com a especificação do problema é denominado **Metodologia Direta-Inversa**. Este método baseia-se no treinamento de duas RNA: uma rede para aproximar a função de interesse (RNAD) e uma segunda para aproximar a função inversa da primeira (RNAI) sendo, respectivamente:

- X, T e O: o vetor de entradas, alvos e saídas da RNAD;
- O, X e X^{α}: o vetor de entradas, alvos e saídas da RNAI;
- N: o número de padrões de treinamento.

Na fase de treinamento, a RNAD é treinada com o conjunto de treinamento $[X \;\; T]$ com N pequeno até se atingir o critério de parada e os pesos representativos deste treino são salvos e a rede fornece como conjunto de saídas atuais o vetor O. Logo após, a RNAI é treinada com o conjunto $[O \;\; X]$ até se atingir o critério de parada e os pesos representativos deste treino são salvos e a rede fornece como conjunto de saídas atuais o vetor X^{α}. O fluxo de treinamento pode ser visualizado na Figura (6.1).

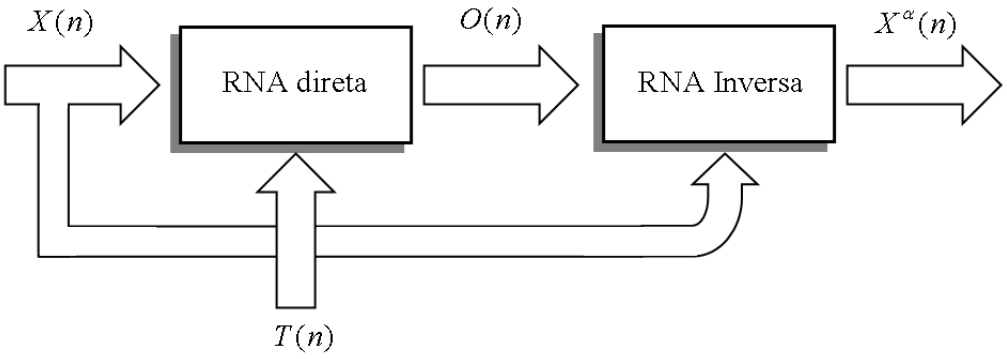


Figura 6.1: Fluxo de dados no treinamento e utilização das redes direta e inversa

Após o treinamento conjunto da RNAD e RNAI, a diferença entre cada componente do vetor X e X^{α} é calculada de acordo com o erro e_{inv}, Equação (6.1), e é computada a média desses

erros E_{inv}, Equação (6.2).

$$e_{inv}(t) = |x(t) - x^{\alpha}(t)|, \quad 1 \leq t \leq N \tag{6.1}$$

$$E_{inv} = \frac{1}{N}\sum_{t=1}^{N} e_{inv}(t) \tag{6.2}$$

Para se obter a quantidade de padrões mínima que permitirá à RNAD generalizar, aumenta-se gradativamente a quantidade de padrões N de treinamento. Efetua-se um novo treinamento conjunto da RNAD e RNAI e avalia-se o novo valor para o parâmetro E_{inv}. Este novo valor, com a convergência das redes, será menor que o do treinamento anterior. Este processo é repetido até que E_{inv} não tenha mais mudanças significativas em seu valor caracterizando a generalização da rede e, portanto, o valor de N associado a ele é a quantidade mínima de padrões que permitirá à RNAD generalizar a função de interesse.

6.3 Confiabilidade das Saídas de RNA

Na fase de utilização, uma saída confiável é aquela em que a diferença entre o valor da entrada da RNAD e a saída da RNAI, Equação (6.1), seja menor que o erro médio de treinamento E_{inv}. Portanto, na fase de utilização, um conjunto de utilização X é aplicado à RNAD e o fluxo da Fig. 6.1 é seguido para fornecer a saída X^{α}. Logo após, o erro e_{inv} é computado para cada componente dos vetores X e X^{α} de utilização. Estes são classificados de acordo com a desigualdade da Equação (6.3). Para qualquer quantidade N de padrões de treinamento, o parâmetro E_{inv} consegue classificar os valores confiáveis e não-confiáveis associados a quantidade N do treinamento corrente.

$$o(t) \text{ será } = \begin{cases} \text{não-confiável}, & \text{se } e_{inv}(t) > E_{inv} \\ \text{confiável}, & \text{se } e_{inv}(t) \leq E_{inv} \end{cases} \tag{6.3}$$

Nos domínios complexo ou de Clifford, é necessária a utilização de outro parâmetro para aferir a quantidade mínima de padrões de treinamento devido ao fato de estes dois domínios não serem totalmente ordenados. Este parâmetro é dado como módulo da diferença dos valores absolutos entre a entrada da RNAD e a saída da RNAI, $Einv_{dm}$, como mostra a Equação (6.4). Neste artigo, o cômputo do módulo na álgebra de Clifford, no subespaço desejado, é idêntico ao domínio complexo, e a generalização da definição de módulo para qualquer álgebra de Clifford pode ser encontrada em (Dorst et al., 2007). O parâmetro E_{inv}, Equação (6.2), permanece sendo o valor que fará a classificação entre os valores confiáveis e não-confiáveis na fase de utilização nestes domínios.

$$Einv_{dm} = \frac{1}{N}\sum_{t=1}^{N} ||x(t)| - |x^{\alpha}(t)|| \tag{6.4}$$

CAPÍTULO

7

Resultados

Nesta seção, apresentaremos os resultados dos treinamentos e utilização das redes direta e inversa nos domínios dos números reais, complexos e de Clifford, que demonstram o funcionamento da metodologia proposta. As funções escolhidas para a demonstração da metodologia foram a quadrática real, complexa e de Clifford, cada qual limitada a certos intervalos de domínio para garantir a sua função inversa, o que é fundamental para a aplicação da metodologia.

Os resultados foram obtidos utilizado-se o *software* **Matlab**® na **versão 7.1.0.246 (R14)**. Para implementar as RNA, com o algoritmo BP, foram desenvolvidos *scripts* específicos, sem utilização do *toolbox* de redes neurais disponibilizado pelo *software*, para cada domínio. No domínio real e complexo, utilizamos funções básicas nativas do Matlab®, enquanto no domínio de Clifford utilizamos, em conjunto, com as funções básicas, rotinas do toolbox **Geometric AlgeBra Learning Environment (GABLE)** (Mann et al., 2001), que possibilitam o cômputo dos principais operadores das álgebras geométricas de Clifford de assinatura $Cl_{p,q}$, para o caso específico das álgebras onde $p + q = 3$.

Com o objetivo de encontrar a melhor topologia de rede em cada domínio, tanto para a RNAD quanto para RNAI, treinaram-se diversas redes, variando-se o número de neurônios e de camadas, para encontrar o melhor modelo para cada rede, em cada domínio de atuação. A FA escolhida para se implementar nas redes foi a tangente hiperbólica, porque esta função permite que as RNA complexas e de Clifford guardem informação sobre o comportamento da fase (Yi, 2004), cada qual com sua respectiva representação no domínio de atuação.

7.1 Resultados no Domínio Real

A função quadrática $f(x) = x^2$ real tem o seu comportamento bem conhecido, porém, esta função não tem sua função inversa garantida por todo seu domínio, razão pela qual o seu domínio e restringido entre $[1,\ 5]$.

7.1.1 Parâmetros de Treinamento: Domínio Real

Os parâmetros de treinamento para a RNAD e RNAI de domínio real são mostrados na Tabela (7.1). As entradas e alvos da RNAI são dependentes da RNAD de acordo com as definições da seção 6.2. A topologia, critérios de parada e modo de treinamento da RNAI são os mesmos da RNAD.

Algoritmo de treinamento:	*backpropagation*
Modo de treinamento:	padrão a padrão
Entradas RNAD:	entre $[1,\ 5]$
Taxa de aprendizado:	10^{-1}
MSE desejado:	10^{-4}
Número máximo de epócas:	20.000
Topologia RNAD e RNAI:	1-4-4-1

Tabela 7.1: Parâmetros de treinamento redes direta e inversa: domínio real

Os valores do intervalo de entrada da RNAD foram normalizados entre $\left[\frac{1}{4}, \frac{3}{4}\right]$, usando-se a Equação (2.30) e, após o treinamento e utilização das redes, os dados são retomados ao seu intervalo original de acordo com a Equação (2.31).

7.1.2 Análise da Confiabilidade: Domínio Real

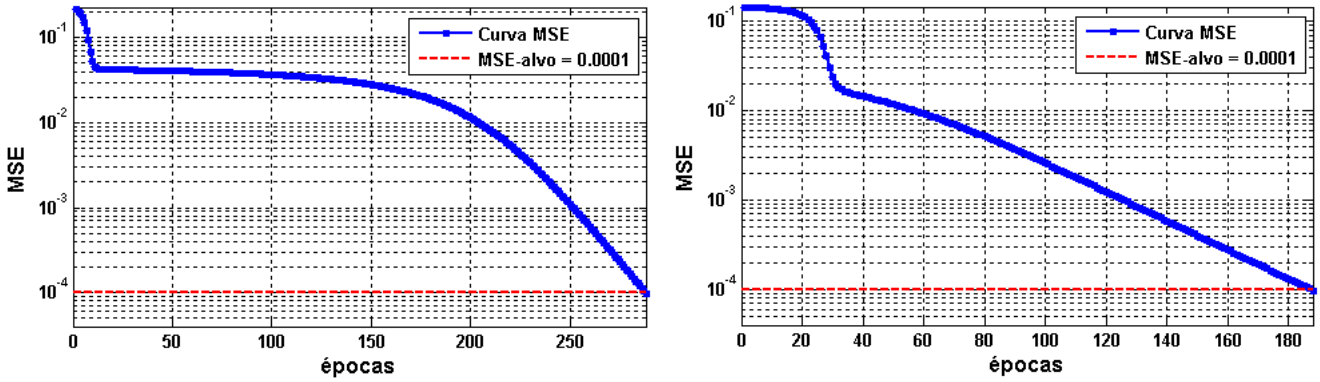
Treinamento com 2 Padrões: Domínio Real

Nesta etapa, foram utilizados 2 padrões de treinamento: entradas $X = [1\ ,5]$ e alvos $T = [1\ ,25]$. As duas redes convergiram para o MSE desejado, como mostra a Figura (7.1). Os alvos e as saídas das redes RNAD e RNAI, após o treinamento, podem ser observados na Tabela (7.2).

RNAD		RNAI	
alvos-treino	saídas-treino	alvos-treino	saídas-treino
1	1.5128	1	1.0667
25	24.4335	5	4.8761

Tabela 7.2: Alvos e saídas de treinamento RNAD e RNAI para 2 padrões: domínio real

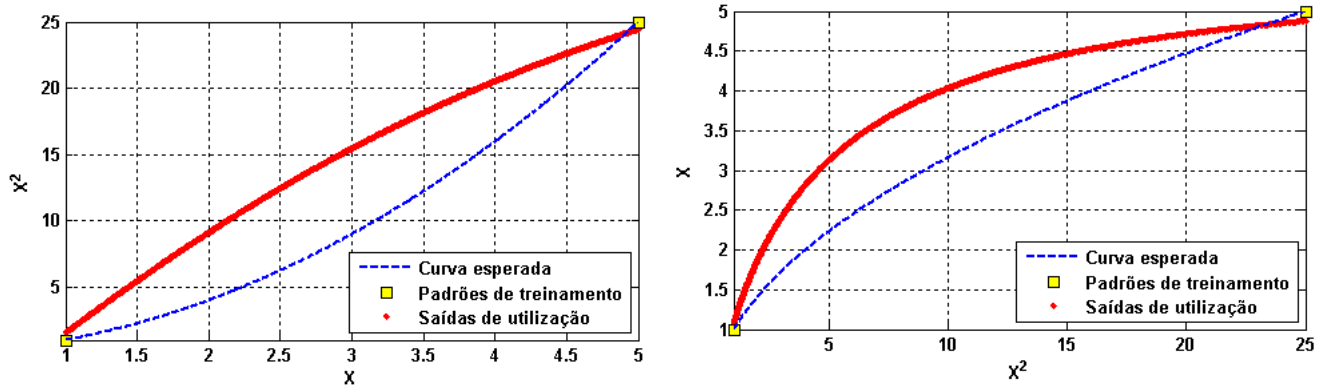
Para testar a qualidade do treinamento, foi criado um conjunto de teste com 500 entradas, todas entre $[1\ ,5]$. O gráfico das saídas em função das suas respectivas entradas é mostrado na



(a) MSE RNAD (b) MSE RNAI

Figura 7.1: MSE RNAD e RNAI para 2 padrões de treinamento: domínio real

Figura (7.2), onde os pontos em vermelho representam as saídas fornecidas pela RNA treinada mediante a apresentação do conjunto de teste com 500 padrões. Nota-se que a rede direta não foi capaz de aprender a função $f(x) = x^2$, já que a maioria dos valores de utilização desvia muito da curva esperada, como mostra a Figura [7.2(a)]. O mesmo se pode concluir da rede inversa, observando a Figura [7.2(b)].



(a) Utilização RNAD (b) Utilização RNAI

Figura 7.2: Utilização da RNAD e RNAI treinadas com 2 padrões: domínio real

O parâmetro E_{inv}, apresentado na seção 6.2, permite averiguar a qualidade das saídas de utilização da RNAD. Para o treinamento com 2 padrões, ele assumiu o valor 0,0119. As saídas confiáveis e não-confiáveis podem ser visualizadas na Figura (7.3), onde as saídas não-confiáveis estão representadas por **x** e as confiáveis pelos pontos em vermelho. Observando as saídas confiáveis, podemos notar que elas estão próximas aos padrões de treinamento e as saídas não-confiáveis próximas aos valores intermediários. Este comportamento já era esperado, visto que não foi fornecido, no treinamento da rede, informação suficiente para aproximar os valores intermediários.

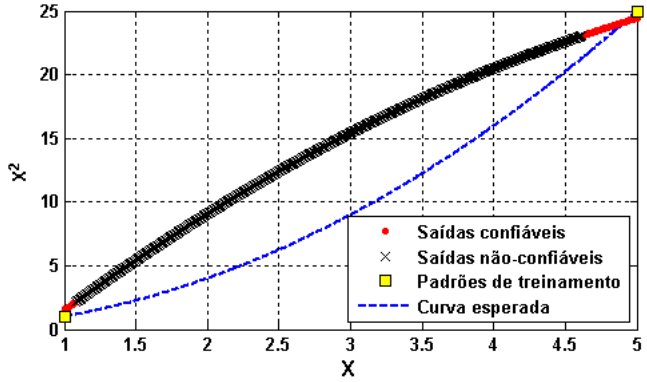
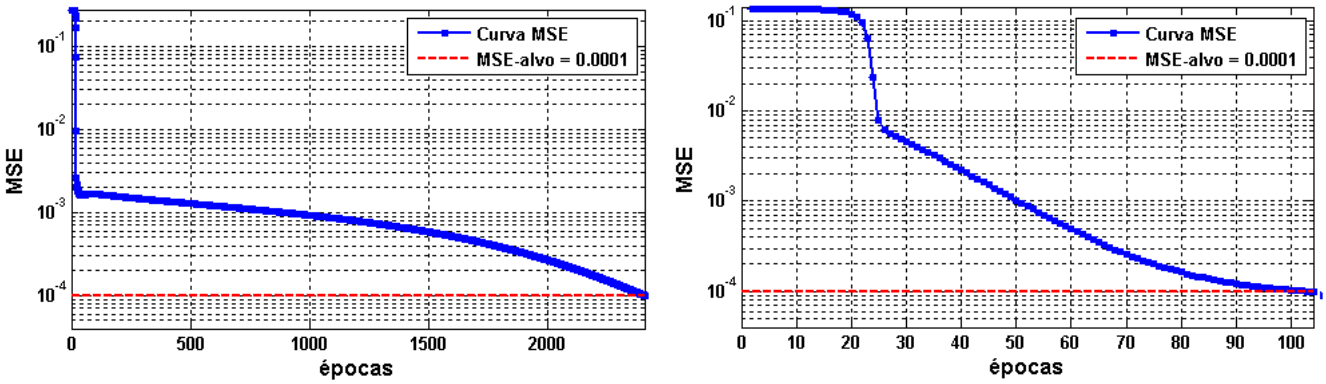


Figura 7.3: Confiabilidade das saídas da RNAD treinada com 2 padrões: domínio real

Treinamento com 5 padrões: Domínio Real

Para o treinamento com 5 padrões, o MSE desejado também foi alcançado pelas duas redes, como mostra a Figura (7.4).



(a) MSE RNAD (b) MSE RNAI

Figura 7.4: MSE RNAD e RNAI para 5 padrões de treinamento: domínio real

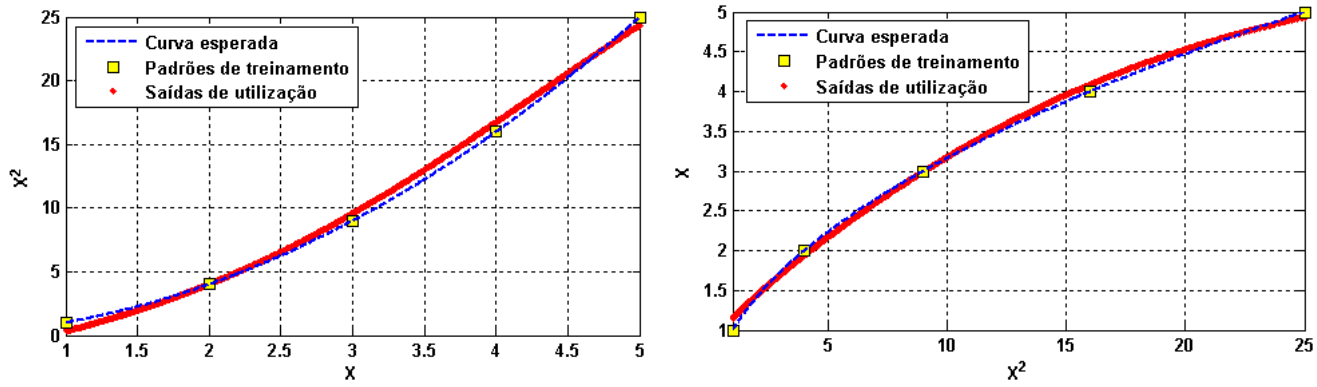
Na Tabela (7.3) podemos observar os alvos e as saídas das duas redes para o treinamento com 5 padrões.

RNAD		RNAI	
alvos-treino	saídas-treino	alvos-treino	saídas-treino
1	0.3183	1	1.1401
4	4.0060	2	1.9326
9	9.5351	3	2.9872
16	16.7059	4	4.0842
25	24.3185	5	4.9325

Tabela 7.3: Alvos e saídas de treinamento RNAD e RNAI para 5 padrões: domínio real

Para testar as redes treinadas com 5 padrões, fornecemos o mesmo conjunto, com 500 padrões de utilização, usados para testar as redes treinadas com 2 padrões. Observando a Figura

(7.5), podemos notar que, fornecendo mais padrões de treinamento, a rede melhorou a interpolação de $f(x) = x^2$ e também da sua respectiva inversa, o que também já era esperado, uma vez que neste treinamento com 5 padrões há uma quantidade maior de informação que pode ser armazenada nos pesos sinápticos da rede. Para o treinamento das redes com 5 padrões, o erro E_{inv} foi de 0,0093, que é menor, comparado com o mesmo erro obtido com o treinamento com 2 padrões.



(a) Utilização RNAD (b) Utilização RNAI

Figura 7.5: Utilização da RNAD e RNAI treinadas com 5 padrões: domínio real

Observando a Figura (7.6), podemos constatar que existe uma quantidade maior de padrões de utilização da RNAD que podem ser classificados com confiáveis de acordo com a comparação com o erro e_{inv} dos padrões de utilização com o erro E_{inv} do treinamento com 5 padrões.

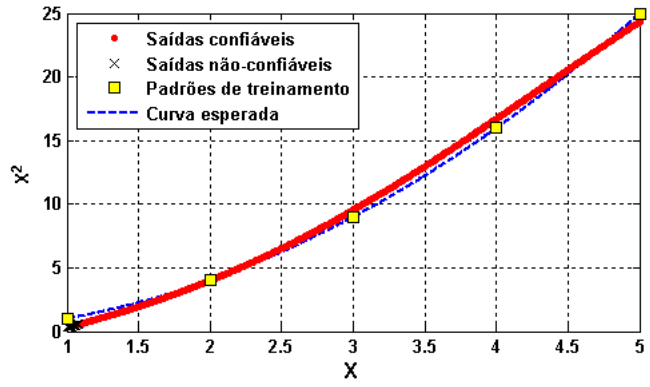


Figura 7.6: Confiabilidade das saídas da RNAD treinada com 5 padrões: domínio real

7.1.3 Amostragem Suficiente para Treinamento: Domínio Real

Com o intuito de encontrar um número mínimo suficiente de padrões, para garantir a generalização da RNA que representa a função $f(x) = x^2$, a RNA em questão foi treinada e testada diversas vezes para diferentes quantidades de padrões, processo este necessário também para

poder avaliar o comportamento do erro E_{inv}, com o aumento do número de padrões, de acordo com a metodologia proposta. Os resultados são resumidos na Tabela (7.4).

		épocas de treino	
padrões	E_{inv}	RNAD	RNAI
2	$0,0119$	457	155
3	$0,0108$	2688	222
4	$0,0108$	3028	533
5	$0,0093$	2406	104
6	$0,0099$	2107	58
7	$0,0098$	1437	214
8	$0,0093$	2350	261

Tabela 7.4: Comportamento do erro E_{inv} com o aumento de padrões: domínio real

A partir de 5 padrões, o erro E_{inv} não tem mudanças significativas, razão pela qual podemos concluir que este número de amostras é suficiente para generalizar a RNA para o problema proposto. Dados adicionais podem ser encontrados em (Alencar, 2007).

7.2 Resultados no Domínio Complexo

Pelo fato de o domínio complexo não ser um conjunto totalmente ordenado, como os números reais sobre a reta real, optamos pela representação de um número complexo na forma $z = re^{\theta i}$ para construir os conjuntos de treinamento e de utilização. Esta forma nos permite gerar números complexos igualmente espaçados pela fase θ sobre um determinado raio r de interesse. E tal como a função quadrática real, a função complexa z^2 não tem sua função inversa garantida por todo seu domínio. Assim, optamos por uma faixa de valores para a fase no raio unitário onde a função inversa exista.

7.2.1 Parâmetros de Treinamento: Domínio Complexo

Os parâmetros de treinamento para a RNAD e RNAI de domínio complexo são mostrados na Tabela (7.5). As entradas e alvos da RNAI são dependentes da RNAD de acordo com as definições da seção 6.2. A topologia, critérios de parada e modo de treinamento da RNAD são os mesmos da RNAI.

Algoritmo de treinamento:	*complex backpropagation*
Modo de treinamento:	padrão a padrão
Entradas RNAD:	$exp(\theta i)$: com $5^\circ \leq \theta \leq 40^\circ$
Taxa de aprendizado:	10^{-1}
MSE desejado:	$7 * 10^{-5}$
Número máximo de epócas:	20000
Topologia RNAD e RNAI:	1-5-1

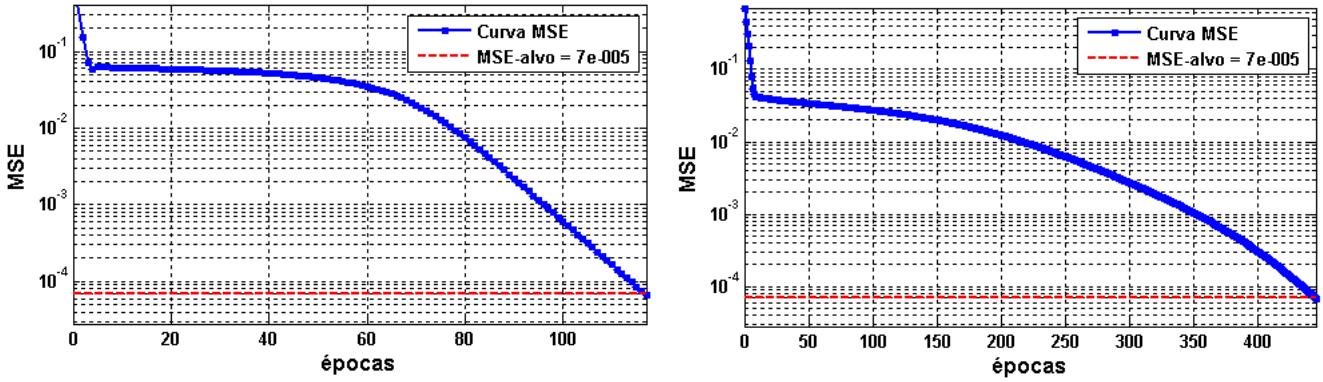
Tabela 7.5: Parâmetros de treinamento redes direta e inversa: domínio complexo

Os valores de entrada das redes complexas não foram normalizados, pois escolhemos o raio unitário para criar os conjuntos de treinamento. Para raios maiores que 1, a normalização é necessária (Leung e Haykin, 1991) e pode ser efetuada, utilizando-se as equações discutidas na seção 3.6.

7.2.2 Análise da Confiabilidade: Domínio Complexo

Treinamento com 2 Padrões: Domínio Complexo

Para 2 padrões de treinamento, tanto a rede direta quanto a rede inversa convergiram para o MSE desejado, como mostra a Figura (7.7).



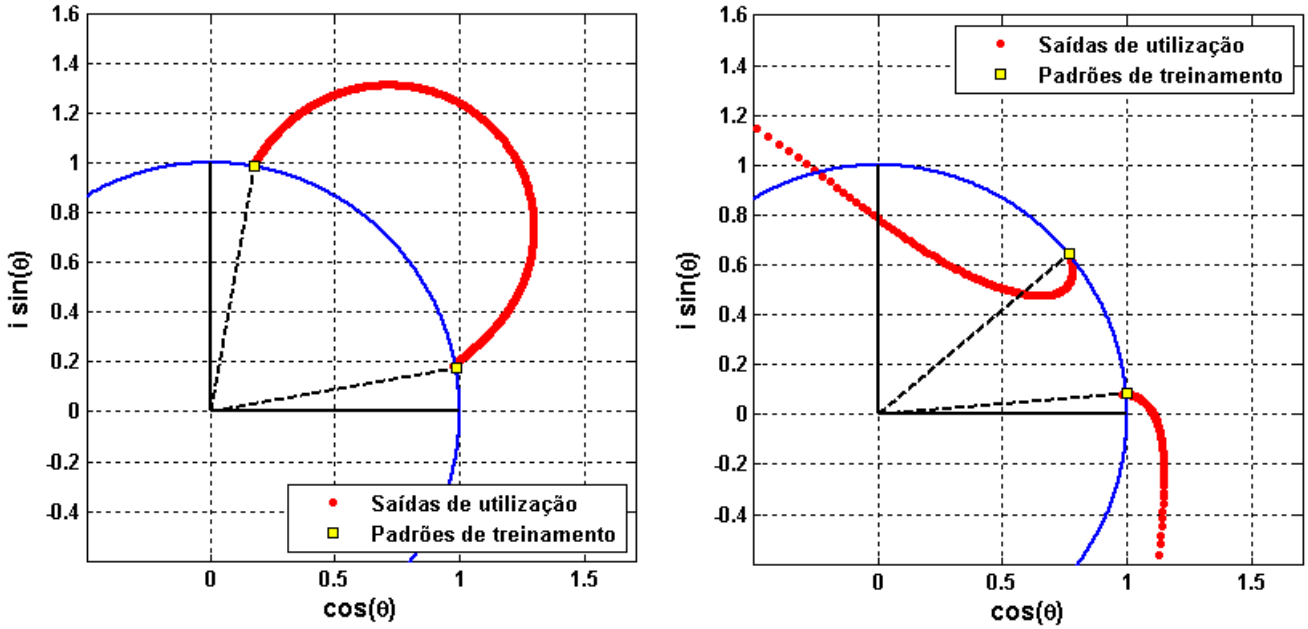
(a) MSE RNAD (b) MSE RNAI

Figura 7.7: MSE RNAD e RNAI para 2 padrões de treinamento: domínio complexo

RNAD			RNAI		
2θ	alvos-treino $= \exp(2\theta i)$	saídas-treino	θ	alvos-treino $= \exp(\theta i)$	saídas-treino
$10°$	$0,9848 + 0,1736i$	$0,9738 + 0,1817i$	$5°$	$0,9962 + 0,0872i$	$0,9834 + 0,0813i$
$80°$	$0,1736 + 0,9848i$	$0,1722 + 0,9873i$	$40°$	$0,7660 + 0,6428i$	$0,7653 + 0,6445i$

Tabela 7.6: Alvos e saídas de treinamento RNAD e RNAI para 2 padrões: domínio complexo

Contudo, 2 padrões de treinamento são insuficientes para generalizar a função quadrática complexa, como pode ser observado na Figura (7.8), onde submetemos 500 padrões de teste às redes treinadas, e somente os pontos próximos aos padrões de treinamento se aproximaram dos alvos desejados. Na Tabela (7.6), podemos observar acurácia dos treinamentos para 2 padrões. O parâmetro E_{inv}, calculado para averiguar a confiabilidade das saídas da rede direta, para o treinamento com 2 padrões, assumiu o valor de 0,00946 e filtrou satisfatoriamente os dados não-confiáveis dos confiáveis, como mostra a Figura (7.9), onde também é possível ver que os valores aceitos como confiáveis estão próximos dos valores fornecidos para o treinamento da rede.



(a) Utilização RNAD (b) Utilização RNAI

Figura 7.8: Utilização da RNAD e RNAI treinadas com 2 padrões: domínio complexo

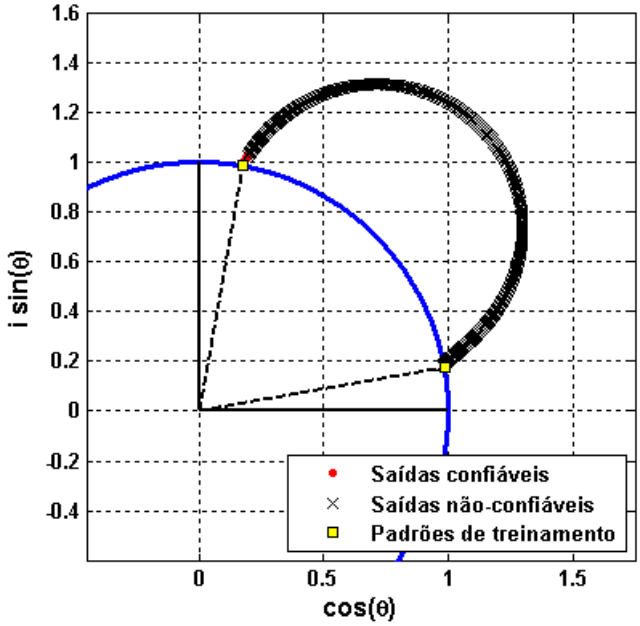
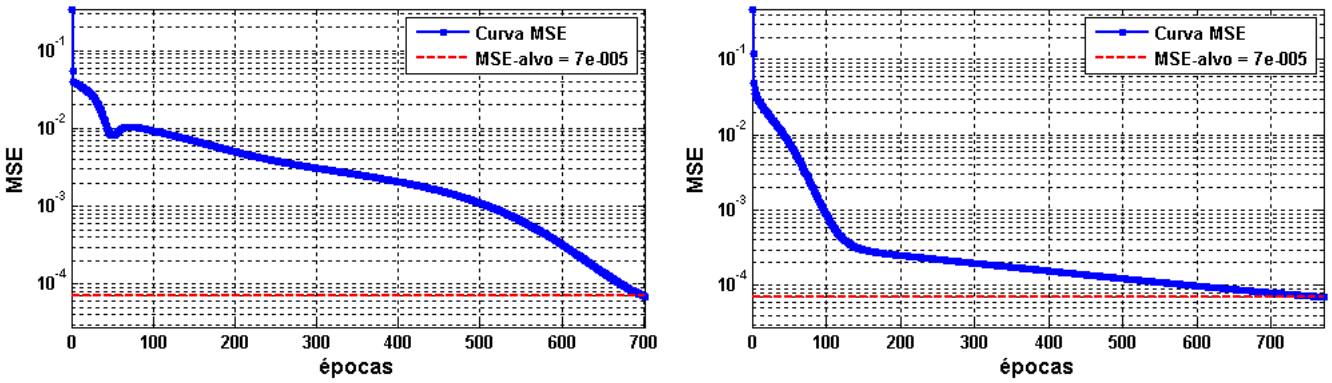


Figura 7.9: Confiabilidade das saídas da RNAD treinada com 2 padrões: domínio complexo

Treinamento com 6 Padrões: Domínio Complexo

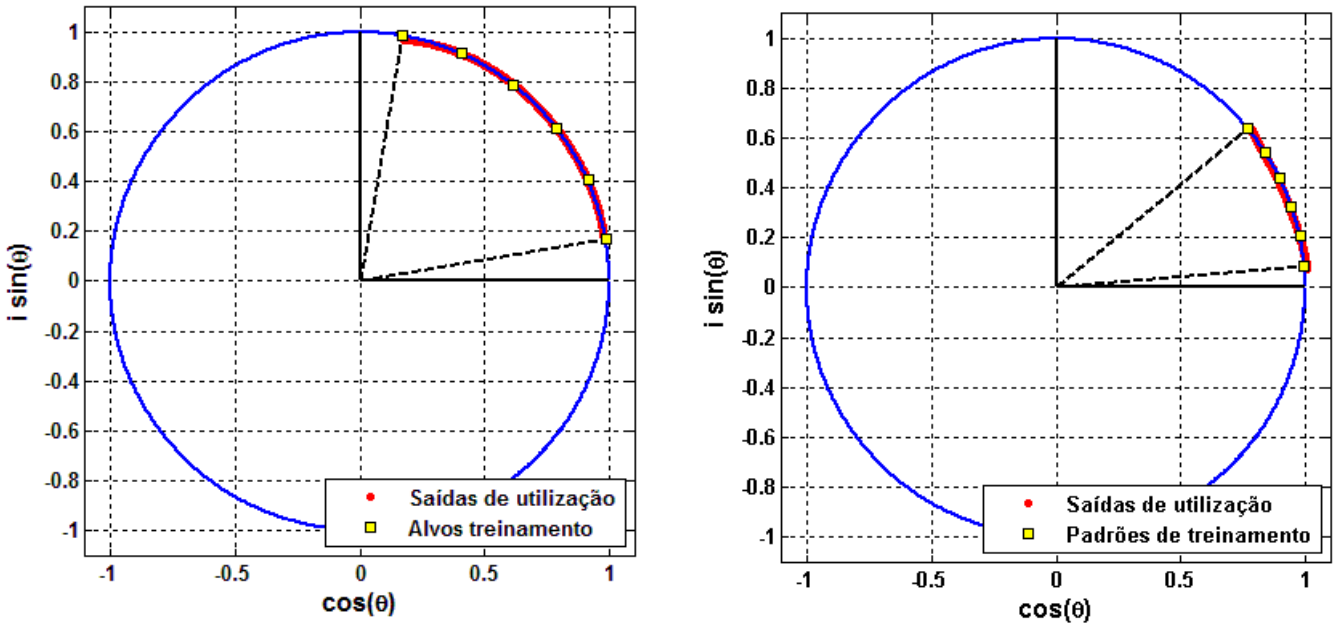
Como se pode observar na Figura (7.10), os treinamentos das redes com 6 padrões também ocorreram com sucesso, pois tanto a rede direta quanto a inversa atingiram o MSE desejado.



(a) MSE RNAD (b) MSE RNAI

Figura 7.10: MSE RNAD e RNAI para 6 padrões de treinamento: domínio complexo

Com 6 padrões de treinamento, as redes conseguem representar melhor a função complexa z^2 e sua respectiva inversa, como pode ser observado na Figura (7.11). Este fato se deve ao aumento do número de padrões de treinamento, o que torna possível aos pesos das redes armazenarem mais informações sobre a função de interesse no domínio especificado.



(a) Utilização RNAD (b) Utilização RNAI

Figura 7.11: Utilização da RNAD e RNAI treinadas com 6 padrões: domínio complexo

O erro E_{inv} atingiu o valor 0,01141 para o treinamento com 6 padrões e também se mostrou uma boa estimativa para filtrar os dados confiáveis dos não-confiáveis, como pode ser visto na Figura (7.12).

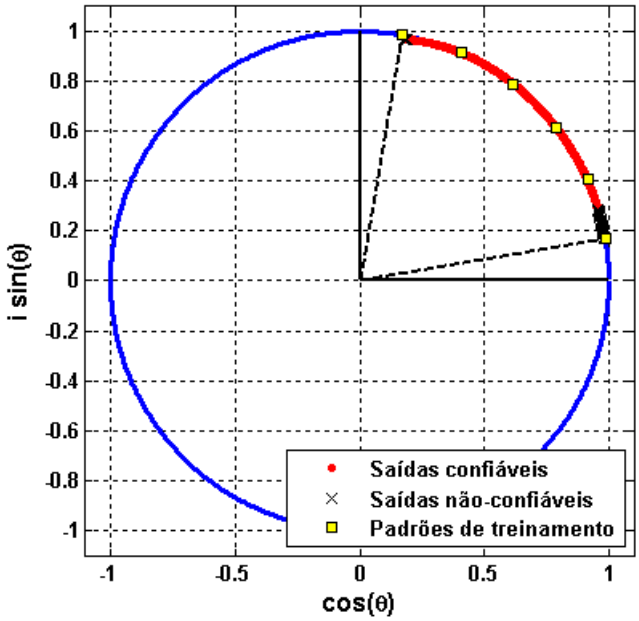


Figura 7.12: Confiabilidade das saídas da RNAD treinada com 6 padrões: domínio complexo

No entanto, não houve muita alteração no valor do erro E_{inv} do treinamento de 2 para 6 padrões, o que o torna um parâmetro não confiável para amostragem da quantidade miníma de padrões necessários para generalizar a rede direta, o que nos compeliu a procurar um outro parâmetro que nos forneça a informação desejada. Este novo parâmetro será discutido na próxima seção. Na Tabela (7.7), podemos visualizar os alvos e as saídas atuais das redes direta e inversa para o treinamento com 6 padrões.

RNAD			RNAI		
2θ	alvos-treino $= \exp(2\theta i)$	saídas-treino	θ	alvos-treino $= \exp(\theta i)$	saídas-treino
$10°$	$0,9848 + 0,1736i$	$0,9789 + 0,1732i$	$5°$	$0,9962 + 0,0872i$	$1,0056 + 0,0774i$
$24°$	$0,9135 + 0,4067i$	$0,9092 + 0,4119i$	$12°$	$0,9781 + 0,2079i$	$0,9776 + 0,2003i$
$38°$	$0,7880 + 0,6157i$	$0,7862 + 0,6173i$	$19°$	$0,9455 + 0,3256i$	$0,9361 + 0,3226i$
$52°$	$0,6157 + 0,7880i$	$0,6317 + 0,7991i$	$26°$	$0,8988 + 0,4384i$	$0,8932 + 0,4441i$
$66°$	$0,4067 + 0,9135i$	$0,4164 + 0,9105i$	$33°$	$0,8387 + 0,5446i$	$0,8255 + 0,5470i$
$80°$	$0,1736 + 0,9848i$	$0,1646 + 0,9715i$	$40°$	$0,7660 + 0,6428i$	$0,7816 + 0,6468i$

Tabela 7.7: Alvos e saídas de treinamento RNAD e RNAI para 6 padrões: domínio complexo

7.2.3 Amostragem Suficiente para Treinamento: Domínio Complexo

O erro E_{inv} no domínio real tende a diminuir e se estabilizar em um determinado valor, com o aumento gradativo da quantidade de padrões de treinamento, fornecendo, assim, um

parâmetro para estimar a quantidade de padrões mínima para generalizar uma RNA. Este parâmetro e utilizado também para filtrar as saídas confiáveis, das não-confiáveis, na fase de utilização dessa RNA. Todavia, no domínio complexo, o erro E_{inv} não tem o mesmo comportamento que no domínio real, devido ao domínio complexo não ser ordenado, como pode ser observado na coluna E_{inv} na Tabela (7.8).

			épocas de treino	
padrões	E_{inv}	$Einv_{dm}$	RNAD	RNAI
2	$0,00946$	$0,00770$	117	445
3	$0,01042$	$0,00517$	1485	1559
4	$0,01002$	$0,00495$	3114	215
5	$0,01010$	$0,00538$	2134	545
6	$0,01141$	$0,00412$	701	771
7	$0,01128$	$0,00423$	923	703
8	$0,01119$	$0,00431$	959	562

Tabela 7.8: Comportamento do erro E_{inv} com o aumento de padrões: domínio complexo

Este fenômeno confirma que, no domínio complexo, E_{inv} se torna uma estimativa não confiável para aferir a quantidade mínima de padrões necessários para generalizar uma RNA, mas tal como no domínio real, E_{inv} continua sendo um bom parâmetro para filtrar os dados confiáveis dos não-confiáveis, na fase de utilização da RNAD, como mostrado na Figura (7.12). Assim, se faz necessário o uso do parâmetro $Einv_{dm}$, Equação (6.4), definido na seção 6.2.

A principal diferença entre E_{inv} e $Einv_{dm}$ é a conotação geométrica que o primeiro tem para o campo complexo, pois ele consegue distinguir o quão iguais são os fasores[1], mesmo aqueles que têm valores absolutos iguais, enquanto o segundo simplesmente tem a média dos valores absolutos da diferença entre os módulos de dois números complexos como parâmetro. Em outras palavras, mede a diferença entre dois números reais sem implicação geométrica alguma, fato que justifica o uso de E_{inv} para averiguar a qualidade das saídas da RNAD e $Einv_{dm}$ é qualificado para averiguar a quantidade de padrões necessários para generalizar o problema, pois este erro $Einv_{dm}$ tem o mesmo comportamento que o erro E_{inv} no domínio real, como pode ser verificado na Tabela (7.8). E, observando esta mesma tabela, podemos afirmar que 6 padrões de treino são suficientes para generalizar a função quadrática complexa.

[1]Representação senoidal (teorema de Euler) de um vetor de rotação no plano complexo.

7.3 Resultados no Domínio de Clifford

Conforme discutido no capítulo 4, as álgebras de Clifford podem ter, dependendo da sua assinatura, a álgebra dos números complexos como subespaço. Afim de facilitar a ponderação dos resultados e estabelecer parâmetros observáveis, vamos fornecer conjuntos de treinamento e utilização para as RNA de Clifford de assinatura $Cl_{0,3}$[2], o subespaço gerado pelos elementos de base $\{1, \mathbf{e}_1 \wedge \mathbf{e}_2\}$, que é um subespaço equivalente ou isomórfico à álgebra dos números complexos (Denker, 2007). Escolhemos a assinatura $Cl_{0,3}$, pois esta é utilizada por padrão no *toolbox* GABLE para **Matlab**® que fornece as rotinas dos operadores das álgebras de Clifford.

7.3.1 Parâmetros de Treinamento: Domínio De Clifford

Os parâmetros de treinamento para as redes RNAD e RNAI são fornecidos na Tabela (7.9).

Algoritmo de treinamento:	*Clifford backpropagation*
Modo de treinamento:	padrão a padrão
Entradas RNAD:	$exp(\theta \mathbf{e}_1 \wedge \mathbf{e}_2)$: com $5^\circ \leq \theta \leq 40^\circ$
Taxa de aprendizado:	10^{-1}
MSE desejado:	$7 * 10^{-5}$
Número máximo de epócas:	20000
Topologia RNAD e RNAI:	1-3-1

Tabela 7.9: Parâmetros de treinamento para as redes direta e inversa: domínio de Clifford

Tais como os resultados do domínio real e complexo, as entradas e alvos da RNAI são dependentes da RNAD de acordo com as definições da seção 6.2. A topologia, critérios de parada e modo de treinamento da RNAD são os mesmos da RNAI. Os intervalos dos valores de θ e a utilização do raio unitário são idênticos aos das redes complexas.

Os padrões de treinamento e utilização no domínio de Clifford foram construídos de forma muito semelhante ao domínio complexo, utilizando-se da relação exponencial $re^{\theta \mathbf{e}_1 \mathbf{e}_2}$ que agora é definida para bivetores, veja seção 4.2.4. Como exemplo, tomemos $r = 1$ e $\theta = 5^\circ$. Logo,

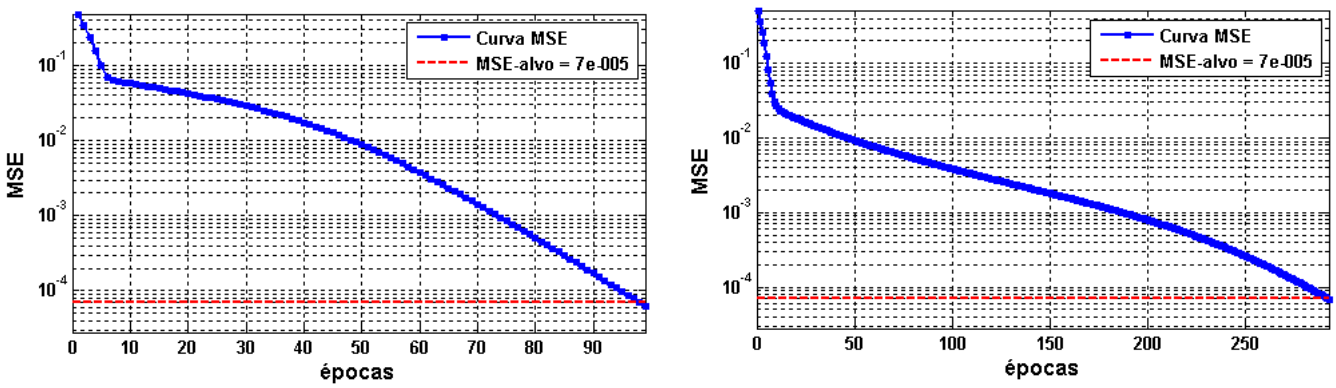
$$\begin{aligned} re^{\theta \mathbf{e}_1 \mathbf{e}_2} &= 0.9962 + 0\mathbf{e}_1 + 0\mathbf{e}_2 + 0\mathbf{e}_3 + 0.0872\mathbf{e}_1\mathbf{e}_2 + 0\mathbf{e}_2\mathbf{e}_3 + 0\mathbf{e}_3\mathbf{e}_1 + 0\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 \\ &= 0.9962 + 0.0872\mathbf{e}_1\mathbf{e}_2 \end{aligned}$$

7.3.2 Análise da Confiabilidade: Domínio de Clifford

Treinamento com 2 Padrões: Domínio de Clifford

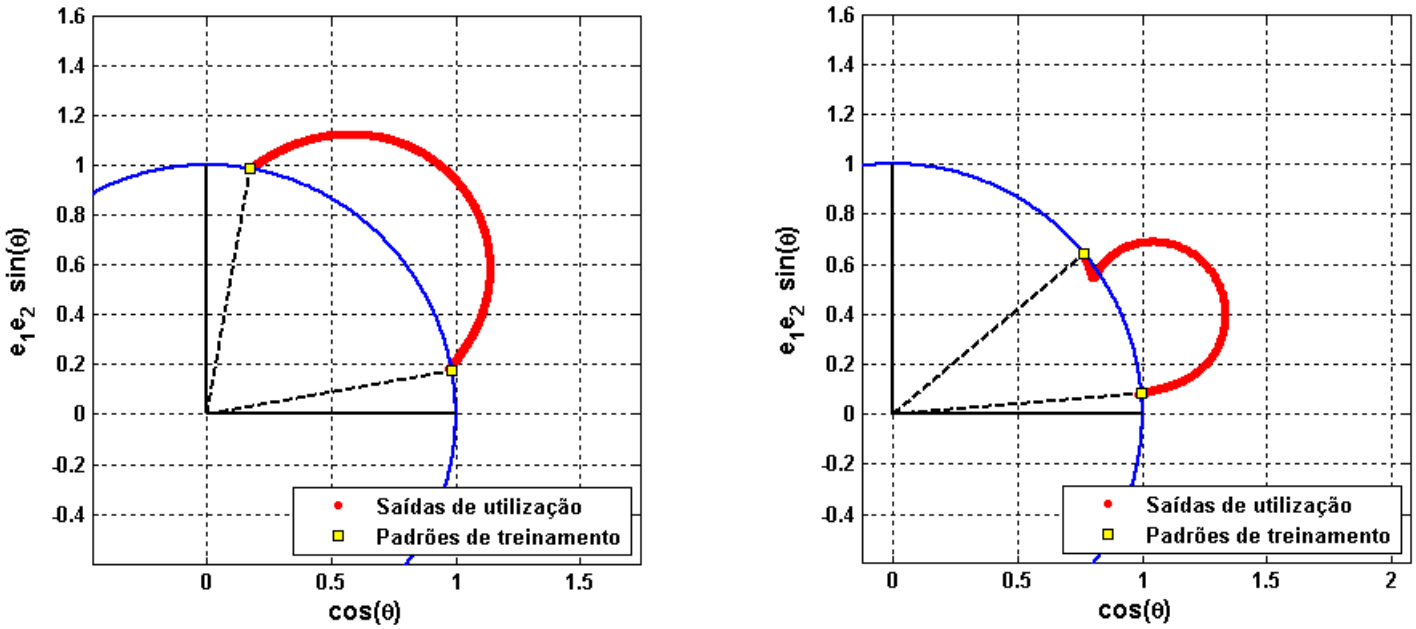
Na Figura (7.13), podemos observar a convergência do treinamento das redes RNAD e RNAI para dois padrões. Ambas atingiram o MSE-alvo do treinamento.

[2]Esta assinatura tem todos os seus elementos de base e_i^2 iguais -1. Veja capítulo 4, para detalhes.



(a) MSE RNAD (b) MSE RNAI

Figura 7.13: MSE RNAD e RNAI para 2 padrões de treinamento: domínio de Clifford



(a) Utilização RNAD (b) Utilização RNAI

Figura 7.14: Utilização da RNAD e RNAI treinadas com 2 padrões: domínio de Clifford

Tal como no domínio complexo, submetemos as redes treinadas a 500 padrões de utilização, para averiguar a qualidade da generalização das redes e, como esperado, devido ao isomorfismo desta álgebra no domínio especificado com a álgebra dos números complexos, dois padrões de treinamento foram insuficientes para generalizar a função quadrática de Clifford $f(c) = c^2$ e sua respectiva inversa. Como mostra a Figura (7.14).

Esta característica de isomorfismo permite que usemos também o erro E_{inv} para medir a qualidade da saída da RNAD no domínio de Clifford. Este erro, para o treinamento com dois padrões, atingiu o valor de 0,00984 e somente filtrou como valores adequados aqueles que se encontravam próximos dos valores de treinamento, como pode ser visto na Figura (7.15).

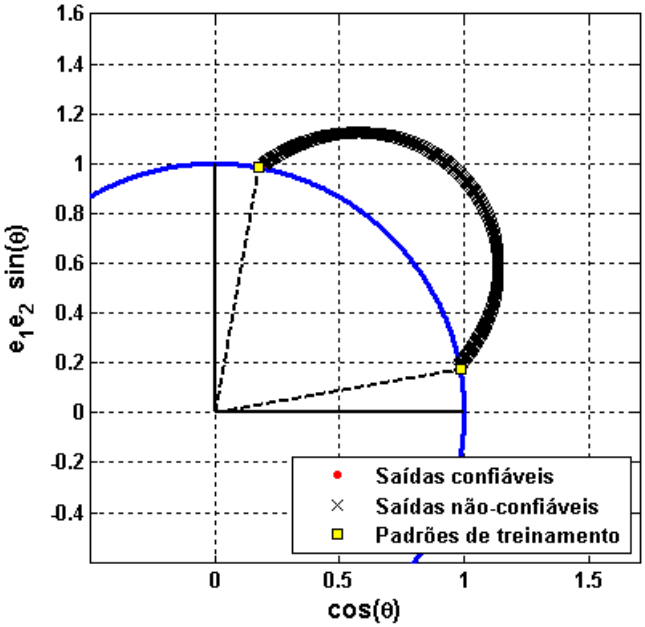


Figura 7.15: Confiabilidade das saídas da RNAD treinada com 2 padrões: domínio de Clifford

E na Tabela (7.10) podemos observar a acurácia das saídas do treinamento com 2 padrões no domínio de Clifford.

RNAD			RNAI		
2θ	alvos-treino $= \exp(2\theta e_1 e_2)$	saídas-treino	θ	alvos-treino $= \exp(\theta e_1 e_2)$	saídas-treino
$10°$	$0,9848 + 0,1736 e_1 e_2$	$0,9716 + 0,1813 e_1 e_2$	$5°$	$0,9962 + 0,0872 e_1 e_2$	$0,9850 + 0,0755 e_1 e_2$
$80°$	$0,1736 + 0,9848 e_1 e_2$	$0,1720 + 0,9888 e_1 e_2$	$40°$	$0,7660 + 0,6428 e_1 e_2$	$0,7638 + 0,6455 e_1 e_2$

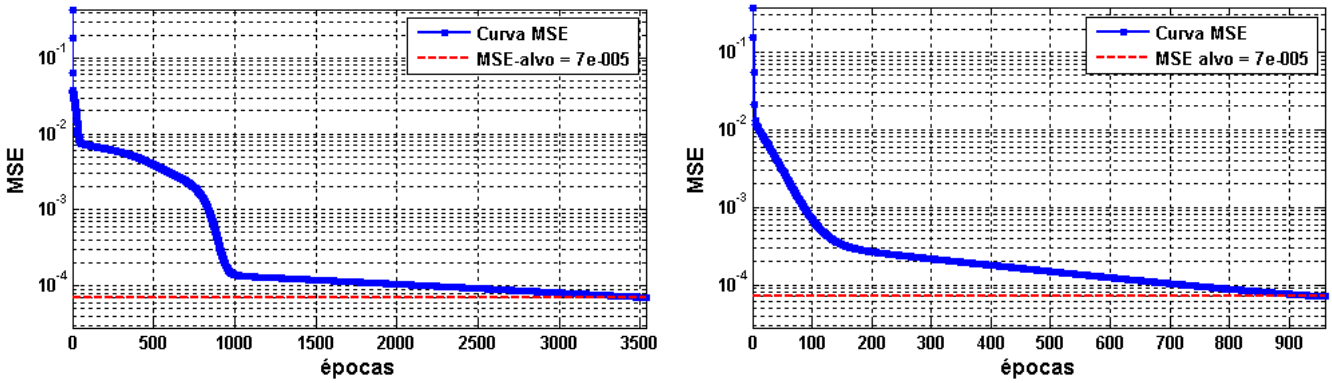
Tabela 7.10: Alvos e saídas de treinamento RNAD e RNAI para 2 padrões:domínio de Clifford

Treinamento com 6 Padrões: Domínio de Clifford

Fica evidente, na Figura (7.16), a convergência dos treinamentos das redes direta e inversa. Ambas atingiram o MSE-alvo. Em comparação ao treinamento com 6 padrões no domínio complexo, o desempenho no treinamento foi mais custoso, em termos computacionais.

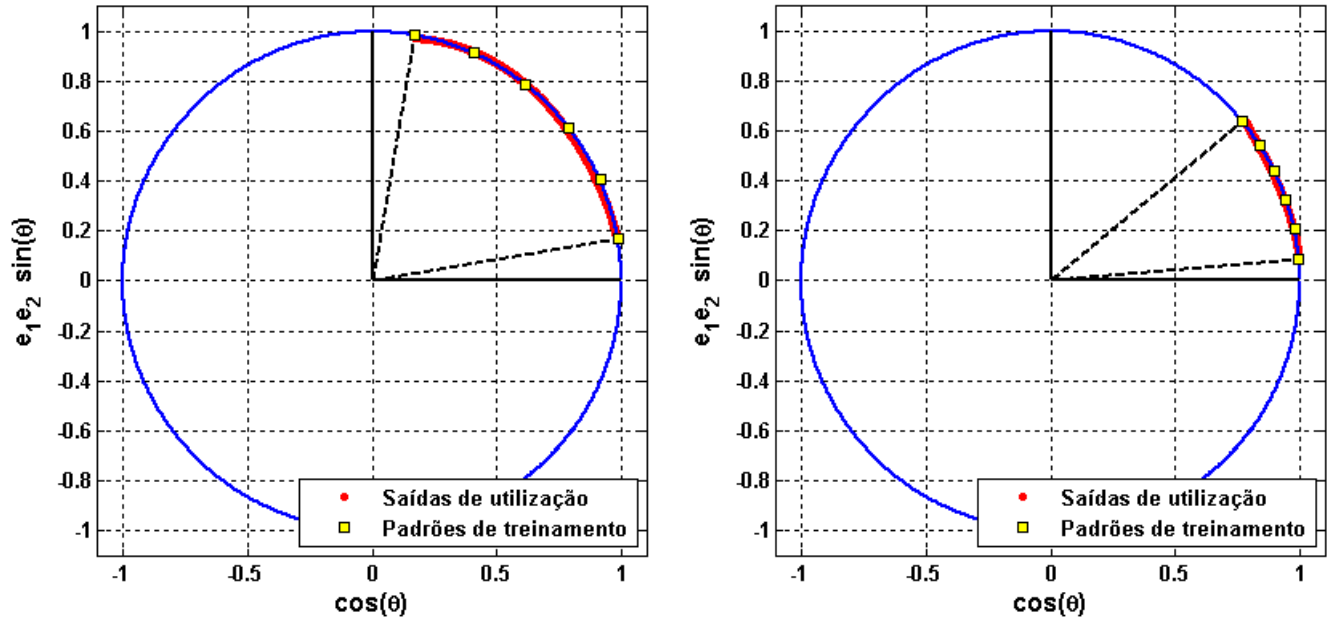
Estes resultados vão de encontro com os obtidos por (Buchholz e Sommer, 2000), que verificaram que as redes de Clifford levam mais épocas para terminar o treinamento do que as redes complexas, porém, evidenciando que o resultado na generalização é melhor do que as redes complexas. Na fase de utilização, Figura (7.17), também fica patente a melhora da representação da função $f(c) = c^2$ pela RNAD, que agora se ajusta melhor à curva ideal da função quadrática, e o mesmo se pode dizer da sua função inversa.

E_{inv}, que atingiu para este treino o valor de 0,00955, também se mostrou um bom parâmetro para filtrar os dados confiáveis dos não-confiáveis, como evidenciado na Figura (7.18).



(a) MSE RNAD (b) MSE RNAI

Figura 7.16: MSE RNAD e RNAI para 6 padrões de treinamento: domínio de Clifford



(a) Utilização RNAD (b) Utilização RNAI

Figura 7.17: Utilização da RNAD e RNAI treinadas com 6 padrões: domínio de Clifford

E, como era de se esperar, E_{inv} no domínio de Clifford também não teve o comportamento esperado como no domínio real. Na Tabela (7.11) é possível observar a qualidade das saídas no treinamento com 6 padrões.

7.3.3 Amostragem Suficiente para Treinamento: Domínio de Clifford

Contudo, tal como no domínio complexo, o valor E_{inv} não sofreu um decréscimo significativo no seu valor, em comparação com o treinamento com 2 padrões, condição necessária para podermos avaliar a quantidade mínima necessária para a generalização da função quadrática.

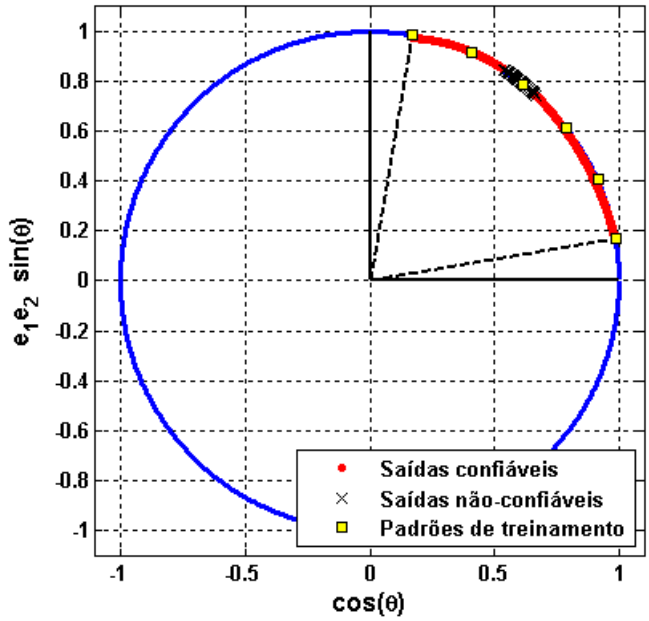


Figura 7.18: Confiabilidade das saídas da RNAD treinada com 6 padrões: domínio de Clifford

RNAD			RNAI		
2θ	alvos-treino $= \exp(2\theta e_1 e_2)$	saídas-treino	θ	alvos-treino $= \exp(\theta e_1 e_2)$	saídas-treino
$10°$	$0,9848 + 0,1736 e_1 e_2$	$0.9865 + 0.1703 e_1 e_2$	$5°$	$0,9962 + 0,0872 e_1 e_2$	$0.9994 + 0.0821 e_1 e_2$
$24°$	$0,9135 + 0,4067 e_1 e_2$	$0.9068 + 0.4110 e_1 e_2$	$12°$	$0,9781 + 0,2079 e_1 e_2$	$0.9819 + 0.1989 e_1 e_2$
$38°$	$0,7880 + 0,6157 e_1 e_2$	$0.7941 + 0.6163 e_1 e_2$	$19°$	$0,9455 + 0,3256 e_1 e_2$	$0.9463 + 0.3248 e_1 e_2$
$52°$	$0,6157 + 0,7880 e_1 e_2$	$0.6053 + 0.7822 e_1 e_2$	$26°$	$0,8988 + 0,4384 e_1 e_2$	$0.8765 + 0.4431 e_1 e_2$
$66°$	$0,4067 + 0,9135 e_1 e_2$	$0.4215 + 0.9285 e_1 e_2$	$33°$	$0,8387 + 0,5446 e_1 e_2$	$0.8375 + 0.5494 e_1 e_2$
$80°$	$0,1736 + 0,9848 e_1 e_2$	$0.1696 + 0.9736 e_1 e_2$	$40°$	$0,7660 + 0,6428 e_1 e_2$	$0.7788 + 0.6426 e_1 e_2$

Tabela 7.11: Alvos e saídas de treinamento RNAD e RNAI para 6 padrões:domínio de Clifford

Conseqüentemente, a solução é a mesma dada para o domínio complexo. Tomamos o valor $Einv_{dm}$, Equação (6.4), como parâmetro para averiguar a quantidade mínima de padrões para a generalização do problema. Os comportamentos destes parâmetros podem ser observados na Tabela (7.12). Aqui podemos notar que, a partir de 6 padrões, o erro $Einv_{dm}$ já não sofre alterações significativas, o que implica que 6 padrões já são suficientes para a generalização da função quadrática de Clifford.

7.4 Discussão Dos Resultados

7.4.1 Domínio Real

No domínio real, a metodologia proposta mostrou-se eficiente para validar a quantidade mínima necessária de padrões para generalizar a função quadrática $f(x) = x^2$, no intervalo de domínio especificado. A evolução do erro E_{inv}, ver Tabela (7.4), evidencia que 5 padrões de treinamento são suficientes para generalizar a função. Este mesmo parâmetro também se

			épocas de treino	
padrões	E_{inv}	$Einv_{dm}$	RNAD	RNAI
2	$0,00984$	$0,00610$	99	293
3	$0,01132$	$0,00776$	1677	1033
4	$0,01158$	$0,00723$	2527	2136
5	$0,01097$	$0,00496$	3418	549
6	$0,00955$	$0,00470$	3533	960
7	$0,01100$	0.00463	2430	1028
8	0.01056	0.00468	9052	2864

Tabela 7.12: Comportamento do erro E_{inv} com o aumento de padrões: domínio de Clifford

mostrou eficiente para filtrar os dados confiáveis dos não-confiáveis, ver Figuras (7.6) e (7.3).Na realização dos experimentos pôde perceber-se que, ao tentar encontrar a topologia ideal para esta função, é que uma camada intermediária não era suficiente para que as redes convergissem em seus treinamentos. Assim, foi necessário acrescentar mais uma camada, onde se verificou que a topologia $1-4-4-1$, para as redes BP trabalhando em modo padrão-a-padrão, foi suficiente para que as redes convergissem para o MSE desejado nos treinamentos, salientando-se que o algoritmo BP desenvolvido não tem nenhum adicional de performance para garantir a convergência. Medida esta nescessária para avaliar a metodologia sem a intervenção desses métodos de performance.

7.4.2 Domínio Complexo

No domínio complexo, a metodologia também mostrou eficácia para validar a quantidade mínima necessária de padrões para generalização da função quadrática complexa $f(z) = z^2$ e para filtrar os dados confiáveis dos não-confiáveis. Este último pode ser visualizado nas Figuras (7.9) e (7.12), porém, neste domínio o parâmetro E_{inv}, Equação (6.2), não teve o comportamento esperado, que era de diminuir e estabilizar seu valor conforme o aumento do número de padrões de treinamento como no domínio real, fato este explicado pela falta de ordem nos elementos do domínio em questão. Verificou-se experimentalmente que ele é um bom parâmetro para filtrar os dados confiáveis dos não-confiáveis, porém, um parâmetro de baixa relevância para aferir a quantidade mínima de padrões necessários para garantir a generalização da função quadrática complexa, já que seu valor não diminuiu significantemente com o aumento do número de padrões de treinamento: ver Tabela (7.8).

Assim, foi necessário o uso de outro parâmetro derivado de E_{inv}, para que fosse possível encontrar a quantidade de padrões que garantiria a generalização do problema, parâmetro este chamado $Einv_{dm}$. Enquanto E_{inv} é a média dos valores absolutos da diferença entre dois vetores complexos, $Einv_{dm}$, Equação (6.4), é a média dos valores absolutos da diferença dos valores absolutos dos vetores complexos. Na prática, o primeiro tem uma conotação geométrica e o segundo é apenas um escalar, porém, $Einv_{dm}$ tem o mesmo comportamento de E_{inv} no domínio real: ele diminui seu valor conforme se aumenta a quantidade de padrões de treinamento, e se estabiliza a partir de 6 padrões de treinamento, ver Tabela (7.8). Para as redes

complexas, notou-se que a topologia $1-5-1$, com apenas uma camada intermediária para as redes *complex backpropagation* padrão-a-padrão, foi suficiente para que as redes convergissem para o MSE desejado, o que, em comparação ao domínio real, é um resultado expressivo, já que no domínio complexo as redes foram capazes de guardar informações do comportamento dos padrões complexos para a função quadrática e ao mesmo tempo guardar informação do comportamento da fase dessa função.

7.4.3 Domínio de Clifford

No domínio de Clifford, escolhemos o subespaço gerado pelos elementos de base $\{1, \mathbf{e}_1 \wedge \mathbf{e}_2\}$, que é isomorfo à álgebra dos números complexos, para validar a metodologia. Neste domínio, os resultados foram muito semelhantes aos do domínio complexo. O erro E_{inv} continuou sendo um boa métrica para validar os padrões de utilização confiáveis dos não-confiáveis e, tal como no domínio complexo, este parâmetro não foi suficiente para aferir a quantidade mínima de padrões para garantir a generalização da função quadrática de Clifford $f(c) = c^2$ no espaço de domínio especificado. Assim, utilizamos o parâmetro $Einv_{dm}$ para averiguar qual a quantidade mínima de padrões necessários para generalizar o problema.

No domínio de Clifford, a partir de 5 padrões, as redes conseguem generalizar a função quadrática satisfatoriamente. A topologia utilizada foi $1-3-1$, a qual garantiu a convergência dos treinamentos das redes, o que, em comparação ao treinamento das redes complexas, mostra, experimentalmente, que são necessários menos neurônios para representar o mesmo problema com resultados análogos ao do domínio complexo, evidenciando, assim, que os neurônios de Clifford têm maior facilidade para guardar o comportamento desta função em relação ao neurônio de domínio complexo.

CAPÍTULO

8

Conclusões

Este trabalho propôs, validou e demonstrou a metodologia Direta-Inversa como uma solução para: 1ª) estimar a exatidão de saídas de RNA classificando quais podem ser consideradas confiáveis e quais não podem, na fase de utilização; e 2ª) estabelecer o número suficientemente pequeno de padrões (neste trabalho chamado de número mínimo) a serem utilizados na fase de treinamento, de acordo com esta metodologia, para garantir a convergência e a generalização das redes.

Ressalve-se, apenas, que, enquanto no domínio real o erro E_{inv} conseguiu ser um parâmetro para estimar a quantidade de padrões suficientes para generalizar uma função quadrática, ao mesmo tempo, este erro, pôde ser usado como critério para filtrar as saídas confiáveis das não-confiáveis, na fase de utilização. Todavia, no domínio complexo e de Clifford, o erro E_{inv} conseguiu filtrar satisfatoriamente os dados confiáveis dos não-confiáveis, porém como não teve o mesmo comportamento que no domínio real, que é diminuir e estabilizar seu valor com o aumento gradativo do número de padrões de treinamento, foi necessário, nestes dois domínios, fazer uma adaptação deste erro, à qual chamamos de $Einv_{dm}$, para poder averiguar a quantidade mínima de padrões necessários para garantir a convergência e a generalização destas redes nos domínios em específico.

Com relação à topologia das redes, as RNA complexas e de Clifford se mostraram mais robustas para aproximar a função quadrática, necessitando de uma topologia simplificada em relação à do domínio real, isso porque, ao invés das duas camadas necessárias no domínio real, nos domínios complexo e de Clifford foi necessária somente uma e com menos neurônios para uma representação eficiente.

A metodologia foi desenvolvida para a abordagem de problemas para aproximar funções que possuam inversa ou que tenham inversa em domínios restritos, como foi demonstrado na aproximação da função quadrática $f(x) = x^2$ para os domínios real, complexo e de Clifford.

Os resultados mostram a eficácia da metodologia nestes três domínios com a determinação dos parâmetros que permitem tomar decisões na fase de utilização sobre a confiabilidade da aproximação de funções. Deixa-se como sugestão de trabalhos futuros a aplicação da metodologia para problemas práticos em engenharia, bem como modelagem da metodologia com outros algoritmos de aprendizado de RNA bem e verificar o desempenho desses algoritmos utilizando as técnicas otimização de convergência entre outras.

Referências Bibliográficas

ALENCAR, G. T. *Análise da confiabilidade de redes neurais artificiais complexas*. Departamento de Engenharia Elétrica CCET/UFMS: Trabalho de Conclusão de Curso, 2007.

ARENA, P.; FORTUNA, L.; RE, R.; XIBILIA, M. G. On the capability of neural networks with complex neurons in complex valued functions approximation. In: *ISCAS*, p. 2168–2171, 1993.

ARENA, P.; FORTUNA, L.; RE, R.; XIBILIA, M. G. Multilayer perceptrons to approximate complex valued functions. *Int. J. Neural Syst*, v. 6, n. 4, p. 435–446, 1995.

ARENA, P.; FORTUNA, L.; RE, R.; XIBILIA, M. G. Multilayer perceptrons to approximate quaternion valued functions. *Neural Networks*, v. 10, n. 2, p. 335–342, 1997.

ARENA, P.; FORTUNA, L.; RE, R.; XIBILIA, M. G. *Neural networks in multidimensional domains: Fundamentals and new trends in modeling and control*. London: Springer-Verlag, 1998.

ARTHAN, R. D. A minimalist construction of the geometric algebra. [*On-line*]. Disponível em `http://arxiv.org/abs/math/0607190` (Acessado em 21/05/2009)

BAUM, E. B.; HAUSSLER, D. What size net gives valid generalisation? *NIPS-89, Denver*, 1987.

BENVENUTO, N.; PIAZZA, F. On the complex backpropagation algorithm. *IEEE Transactions on Signal Processing*, v. 40, n. 4, p. 967–969, 1992.

BOERS, E. J. W.; KUIPER, H. *Biological metaphors and the design of artificial neural networks*. Dissertação de Mestrado, Leiden University, Niels Bohrweg 1, 2333 CA, Leiden, The Netherlands, 1992.

BRAGA, A. P.; CARVALHO, A. P. L. F.; LUDERMIR, T. B. *Livro redes neurais artificiais - teoria e aplicações*. Editora LTC, 2007.

BUCHHOLZ, S. *A theory of neural computation with clifford algebra*. Tese de Doutoramento, Inst. f. Informatik u. Prakt. Math. der Christian-Albrechts-Universität zu Kiel, 2005.

BUCHHOLZ, S.; SOMMER, G. Learning geometric transformations with clifford neurons. In: *AFPAC*, Springer, p. 144–153, 2000 (*Lecture Notes in Computer Science*, v.1888).

CARVALHO, A. P. Redes neurais artificiais. [*On-line*].
Disponível em `www.icmc.usp.br/~andre/research/neural/index.htm`
(Acessado em 21/05/2009)

CYBENKO, G. *Continuous valued neural networks with two hidden layers are sufficient*. Relatório Técnico, Department of Computer Science, Tufts University, Medford, MA, 1988.

CYBENKO, G. *Approximations by superpositions of a sigmoidal function*. Technical Report CSRD-856, Center for Supercomputing Research and Development, University of Illinois, 1989.

DENKER, J. Comparing complex numbers to clifford algebra. [*On-line*].
Disponível em `http://www.av8n.com/physics/complex-clifford.htm`
(Acessado em 21/05/2009)

DORST, L.; FONTIJNE, D.; MANN, S. *Geometric algebra for computer science: An object-oriented approach to geometry (the morgan kaufmann series in computer graphics)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

DORST, L.; MANN, S. Geometric algebra: A computational framework for geometrical applications (part 1). *IEEE Computer Graphics and Applications*, v. 22, n. 3, p. 24–31, 2002a. Disponível em `http://computer.org/cga/cg2002/g3024abs.htm`

DORST, L.; MANN, S. Geometric algebra: A computational framework for geometrical applications (part 2). *IEEE Computer Graphics and Applications*, v. 22, n. 4, p. 58–67, 2002b. Disponível em `http://computer.org/cga/cg2002/g4058abs.htm`

GEORGIOU, G. M.; C, K. Complex domain backpropagation. *IEEE Transactions on Circuits and Systems. II: Analog and Digital Signal Processing*, v. 39, n. 5, p. 330–334, 1992.

GIRARD, P. R. *Quaternions, clifford algebras and relativistic physics*. Springer, 2007.

HAYKIN, S. *Neural networks. A comprehensive foundation*. New York: Macmillan College Publishing, 1994.

HESTENES, D. Oersted medal lecture 2002: Reforming the mathematical language of physics. *American Journal of Physics*, v. 71, n. 2, p. 104–121, 2003.

HESTENES, D.; SOBCZYK, G. *Clifford algebra to geometric calculus: A unified language for mathematics and physics*. Springer, 1984.

HESTENES, D.; ZIEGLER, R. Projective geometry with cli®ord algebra. Acta Applicandae Mathematicae vol.23, p. 25–63, 1991.

KIM, T. Complex backpropagation neural network using elementary transcendental activation functions. 2001.

KIM, T.; ADALI, T. Approximation by fully complex multilayer perceptrons. *Neural Computation*, v. 15, n. 7, p. 1641–1666, 2003.

KNOOP, K. *Theory of functions, parts i and ii, two volumes bound as one*, cáp. The Cauchy-Riemann Differential Equations New York: Dover: Dover Publications, p. 28–31, 1996a.

KNOOP, K. *Theory of functions, parts i and ii, two volumes bound as one*, cáp. Analytic Continuation and Complete Definition of Analytic Functions New York: Dover: Dover Publications, p. 83–111, 1996b.

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. Morgan Kaufmann, p. 1137–1143, 1995.

LAVILLE, G.; LEHMAN, E. Analytic cliffordian functions. 2005. Disponível em `www.citebase.org/abstract?id=oai:arXiv.org:math/0502090`

LEUNG, H.; HAYKIN, S. The complex backpropagation algorithm. *IEEE Transactions on Signal Processing*, v. 39, n. 9, p. 2101–2104, 1991.

LYNESS, J. N. Numerical algorithms based on the theory of complex variable. In: *Proceedings of the 1967 22nd national conference*, New York, NY, USA: ACM, p. 125–133, 1967.

MANN, S.; DORST, L.; BOUMA, T. *The making of a geometric algebra package in matlab*. Relatório Técnico CS-99-27, University of Waterloo, Canada, 26 pages, 1999.

MANN, S.; DORST, L.; BOUMA, T. The making of GABLE: a geometric algebra learning environment in Matlab. In: *Geometric algebra with applications in science and engineering (Ixtapa-Zihuatanejo, 1999)*, Boston, MA: Birkhäuser Boston, p. 491–511, 2001.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biophysics*, v. 5, p. 115–133, 1943.

MINSKY, M.; PAPERT, S. *Perceptrons*. Cambridge, Mass: MIT Press, 1969.

MØLLER, M. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, v. 6, n. 4, p. 525–533, 1993.

NGUYEN, D.; WIDROW, B. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In: *IEEE International Joint Conference on Neural Networks (4th IJCNN'90)*, San Diego: IEEE, stanford, p. III–21–III–26, 1990.

NITTA, T. An analysis of the fundamental structure of complex-valued neurons. *Neural Processing Letters*, v. 12, n. 3, p. 239–246, 2000.

PAIVA, C. R. Álgebra geométrica do espaço: Introdução. [*On-line*].
Disponível em `https://dspace.ist.utl.pt/bitstream/2295/172011/1/IntroAG.pdf` (Acessado em 21/05/2009)

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning internal representations by error propagation. In: RUMELHART, D. E.; MCCLELLAND, J. L., eds. *Parallel Distributed Processing*, v. 1, cáp. 8, Cambridge, Massachusetts: MIT Press, p. 318–362, 1986.

DE SABBATA & BIDYUT KUMAR DATTA, V. *Geometric algebra and applications to physics*. CRC Press - Taylor & Francis Group, 2007.

SAFF, E. B.; SNIDER, A. D. *Fundamentals of complex analysis for mathematics, science, and engineering*. New Jersey: Prentice-Hall, 1976.

SUTER, J. Geometric algebra primer. [*On-line*].
Disponível em `http://www.jaapsuter.com/paper/ga_primer.pdf` (Acessado em 21/05/2009)

TRAVESSA, S. S. *Análise das redes neurais complexas na detecção de espículas e piscadas em sinais de eeg*. Dissertação de Mestrado, Universidade Federal de Santa Catarina, 2006.

UCHIMURA, S.; HAMAMOTO, Y.; TOMITA, S.; OSHIMA, N. Effects of the sample size in artificial neural network classifierdesign. In: *IEEE International Conference on*, p. 2126–2129, 1995.

VIEIRA, R. S. Tópicos de Álgebra geométrica. [*On-line*].
Disponível em `http://br.geocities.com/rickrsv` (Acessado em 21/05/2009)

W. L. CHAN, A. T. P. S.; LAI, L. L. Initial applications of complex artificial neural networks to load-flow analysis. *IEE Proc. - Gener. Transm. Distrib.*, v. 147, n. 6, p. 361–366, 2000.

YI, Q. *Learning rules for low-dimensional clifford neural networks*. Dissertação de Mestrado, Portland State University, 2004.

YOU, C.; HONG, D. Nonlinear blind equalization schemes using complex-valued multilayer feedforward neural networks. *IEEE-NN*, v. 9, n. 6, p. 1442, 1998.

APÊNDICE

A

Códigos Fonte *Backpropagation* Real

A.1 MainX2real.m

```
clear;
clc;
format long;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Entradas e Alvos
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Ct=8;
X=linspace(1,5,Ct);
T=X.^2;
[X minX maxX]=premnmx(X);
X=(.75-.25)*(X-(-1))/(1-(-1))+(.25);
[T minT maxT]=premnmx(T);
T=(.75^2-.25^2)*(T-(-1))/(1-(-1))+(.25^2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Rede Direta
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net=redemlp(X,T,[4 4 1],{'tans','tans','tans'});
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Treinar Rede Direta
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net.parametros.n=0.1;
net.parametros.Erro=1e-4;
alvo.d = net.parametros.Erro;
net.parametros.m=0;
net.parametros.Nmax=10^4;
netd=treinar(net,X,T);
```

```
Yt=simular(netd,X);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Rede Inversa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net=redemlp(Yt,X,[4 4 1],{'tans','tans','tans'});
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Treinar Rede Inversa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net.parametros.n = 0.1;
net.parametros.Erro = 1e-4;
alvo.i = net.parametros.Erro;
net.parametros.m=0;
net.parametros.Nmax=10^4;
neti=treinar(net,Yt,X);
Yti=simular(neti,Yt);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plota o comportamento do Erro Global da Redes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clf; subplot(2,2,1);
semilogy(netd.Eplot,'-ob','LineWidth',1.5,'MarkerEdgeColor','b',...
    'MarkerFaceColor','b','MarkerSize',1.5);
hold on
plot([0 size(netd.Eplot,2)],[alvo.d alvo.d],'--r','LineWidth',2);
axis([0 size(netd.Eplot,2) alvo.d/2.5  max(netd.Eplot) ]);
title('\fontsize{12}Rede Direta'); xlabel('\fontsize{12}épocas');
ylabel('\fontsize{12}MSE');
legend('Curva MSE',['MSE alvo = ' num2str(alvo.d)]);
grid on;
subplot(2,2,2);
semilogy(neti.Eplot,'-ob','LineWidth',1.5,'MarkerEdgeColor','b',...
    'MarkerFaceColor','b','MarkerSize',1.5);
hold on
plot([0 size(neti.Eplot,2)],[alvo.i alvo.i],'--r','LineWidth',2);
axis([0 size(neti.Eplot,2) alvo.i/2.5  max(neti.Eplot) ]);
title('\fontsize{12}Rede Inversa'); xlabel('\fontsize{12}épocas');
ylabel('\fontsize{12}MSE');
legend('Curva MSE',['MSE alvo = ' num2str(alvo.i)]);
grid on; pause;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Validar
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Cs=500; S=linspace(1,5,Cs);
[S minS maxS]=premnmx(S);
S=(.75-.25)*(S-(-1))/(1-(-1))+(.25);
Ys=simular(netd,S); Ysi=simular(neti,Ys);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Computo do erro da inversa -> Einv
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Einv = 0;
for k = 1 : size(X,2)
    Einv = Einv + abs( X(1,k)-Yti(1,k) );
end
Einv = Einv/size(X,2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Confiabilidade das saidas da REDE DIRETA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for a = 1 : size(Ysi,2)
    erro_inv_vl(1,a)= abs( S(1,a) - Ysi(1,a));
    if erro_inv_vl(1,a) ≤ Einv
        aceite(1,a) = 1;
    else
        aceite(1,a) = 0;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Desnormalizando os Dados
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
X   =(X-.25)*(maxX-minX)/(.75-.25)+minX;
Yti =(Yti-.25)*(maxX-minX)/(.75-.25)+minX;
S   =(S-.25)*(maxS-minS)/(.75-.25)+minS;
Ysi =(Ysi-.25)*(maxX-minX)/(.75-.25)+minX;
T   =(T-.25^2)*(maxT-minT)/(.75^2-.25^2)+minT;
Yt  =(Yt-.25^2)*(maxT-minT)/(.75^2-.25^2)+minT;
Ys  =(Ys-.25^2)*(maxT-minT)/(.75^2-.25^2)+minT;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Erro da Inversa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MSEi    = neti.parametros.Eg;
beta    = sqrt(2*MSEi);
eit     = X-Yti;
falfa   = sum(abs(eit - beta))/Ct;
ei      = S-Ysi;
tol     = beta+falfa;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Gráficos do Treinamento
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clf;
a=linspace(1,5,500);
b=a.^2;
subplot(2,2,1)
plot(a,b,'--b','LineWidth',2);
grid on;
```

```
title('Utilização Rede Direta')
axis([1 5 1 25])
xlabel('X'); ylabel('X^2');
hold on
plot(X,T,'s','MarkerEdgeColor','k','MarkerFaceColor','y','MarkerSize',7);
pause
plot(S,Ys,'*r','LineWidth',2,'MarkerSize',4);
plot(X,T,'s','MarkerEdgeColor','k','MarkerFaceColor','y','MarkerSize',7);
plot(a,b,'--b','LineWidth',2);
legend('Curva esperada','Padrões de treinamento','Saídas de utilização')
subplot(2,2,2)
plot(b,a,'--b','LineWidth',2);
grid on;
axis([1 25 1 5]);
ylabel('X'); xlabel('X^2');
title('Utilização Rede Inversa');
hold on
plot(T,X,'s','MarkerEdgeColor','k','MarkerFaceColor','y','MarkerSize',7);
pause
plot(S.^2,Ysi,'*r','LineWidth',2,'MarkerSize',4)
plot(T,X,'s','MarkerEdgeColor','k','MarkerFaceColor','y','MarkerSize',7);
plot(b,a,'--b','LineWidth',2);
legend('Curva esperada','Padrões de treinamento','Saídas de utilização');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Erro da Inversa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause;
clf;
stem(S,ei,'.','MarkerSize',5);
hold on
plot(Yti,zeros(1,size(Yt,2)),'s','MarkerEdgeColor','k',...
    'MarkerFaceColor','y','MarkerSize',5);
pause
plotarConf(S,Ys,Ys,Ysi,Yt,Yti,X,T,S,S.^2,aceite);
format short
NETD.T = T;
NETD.O = Yt;
NETI.T = X;
NETI.O = Yti;
vld.O  = Ys;
vli.O  = Ysi;
```

A.2 redemlp.m

```
function [net]=redemlp(X,T,neu,fativ)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%configuração da rede
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net.arquitetura.nent=size(X,1);
net.arquitetura.nsai=size(T,1);
net.arquitetura.ncam=length(neu);
net.arquitetura.fativ=fativ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%pesos iniciais
%pesos da camada de entrada
%pesos das camadas intermediarias
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
IW{1,1}=rand(neu(1,1),size(X,1));
b{1}=rand(neu(1,1),1);
if length(neu)≤1
    LW={};
else
    for cam=2:length(neu)
        LW{cam,cam-1}=rand(neu(1,cam),(neu(1,cam-1)));
        b{cam}=rand(neu(1,cam),1);
    end
end
net.parametros.n=.1;
net.parametros.Erro=1;
net.parametros.N=0;
net.parametros.E=1;
net.parametros.Eg=1;
net.parametros.m=0;
net.parametros.Nmax=10000;
net.tempo=0;
net.pesos.IW=IW;
net.pesos.b=b;
net.pesos.LW=LW;
clear E IW N Ta Z cam db j nent neu Ea IWm Na X ans dIW ∆ m nsai Eg...
      LW Nmax Xa b dLW e n Erro LWm T Y bm dY fativ ncam pad Erroa
```

A.3 treinar.m

```
function [net]=treinar(net,X,T)
n=net.parametros.n;
Erro=net.parametros.Erro;
N=net.parametros.N;
Eg=net.parametros.Eg;
m=net.parametros.m;
Nmax=net.parametros.Nmax;
IW=net.pesos.IW;
b=net.pesos.b;
LW=net.pesos.LW;
nent=net.arquitetura.nent;
nsai=net.arquitetura.nsai;
ncam=net.arquitetura.ncam;
fativ=net.arquitetura.fativ;
Ct=size(X,2);
epocass = 0;
tic;
while Eg>Erro
    Erroa(1,1)=Erro;
    Ea(1,1)=Eg;
    Na(1,1)=N;
    for pad=1:Ct
        Z{pad}{1}=IW{1}*X(:,pad) + b{1};
        if fativ{1,1}=='line';
            Y{pad}{1}=purelin(Z{pad}{1});
            dY{pad}{1}=dpurelin(Z{pad}{1},Y{pad}{1});
        elseif fativ{1,1}=='logs';
            Y{pad}{1}=logsig(Z{pad}{1});
            dY{pad}{1}=dlogsig(Z{pad}{1},Y{pad}{1});
        elseif fativ{1,1}=='tans';
            Y{pad}{1}=tansig(Z{pad}{1});
            dY{pad}{1}=dtansig(Z{pad}{1},Y{pad}{1});
        end
        for cam=2:ncam
            Z{pad}{cam}=LW{cam,cam-1}*Y{pad}{cam-1} + b{cam};
            if fativ{1,cam}=='line';
                Y{pad}{cam}=purelin(Z{pad}{cam});
                dY{pad}{cam}=dpurelin(Z{pad}{cam},Y{pad}{cam});
            elseif fativ{1,cam}=='logs';
                Y{pad}{cam}=logsig(Z{pad}{cam});
                dY{pad}{cam}=dlogsig(Z{pad}{cam},Y{pad}{cam});
            elseif fativ{1,cam}=='tans';
                Y{pad}{cam}=tansig(Z{pad}{cam});
                dY{pad}{cam}=dtansig(Z{pad}{cam},Y{pad}{cam});
```

```
            end
        end
        e{pad}=T(:,pad)-Y{pad}{ncam};   E{pad}=.5*(sum(e{pad}.^2));
        Δ{pad}{ncam}=e{pad}.*dY{pad}{ncam};
        cam=ncam-1;
        for j=1:ncam-1
            Δ{pad}{cam}=(LW{cam+1,cam}'*Δ{pad}{cam+1})...
                .*dY{pad}{cam};
            cam=cam-1;
        end
        dIW{1}=n*Δ{pad}{1}*(X(:,pad)');
        IWm{1}=IW{1};
        IW{1}=IW{1}+dIW{1};
        IW{1}=IW{1}+m*(IW{1}-IWm{1});
        db{1}=n*Δ{pad}{1};
        bm{1}=b{1};
        b{1}=b{1}+db{1};
        b{1}=b{1}+m*(b{1}-bm{1});
        for cam=1:ncam-1
            dLW{cam+1,cam}=n*Δ{pad}{cam+1}*(Y{pad}{cam}');
            LWm{cam+1,cam}=LW{cam+1,cam};
            LW{cam+1,cam}=LW{cam+1,cam}+dLW{cam+1,cam};
            LW{cam+1,cam}=LW{cam+1,cam}+m*(LW{cam+1,cam}-LWm{cam+1,cam});
            db{cam+1}=n*Δ{pad}{cam+1};
            bm{cam+1}=b{cam+1};
            b{cam+1}=b{cam+1}+db{cam+1};
            b{cam+1}=b{cam+1}+m*(b{cam+1}-bm{cam+1});
        end
    end
    Eg=E{1};
    for pad=1:Ct-1
        Eg=Eg+E{pad+1};
    end
    Eg=Eg/Ct
    N=N+1;
    epocass = epocass + 1;
    Eplot(1,epocass) = Eg;
    Ea(1,2)=Eg;
    Na(1,2)=N;
    Erroa(1,2)=Erro;
    semilogy(Na,Ea,'-')
    hold on
    plot(Na,Erroa,'-k')
    hold on
    if N==Nmax
        break
    end
```

```
end
net.tempo =toc;
axis([0 N 10e-6 2]);
xlabel('Épocas')
ylabel('Erro')
net.parametros.n=n;
net.parametros.Erro=Erro;
net.parametros.N=N;
net.parametros.E=E;
net.parametros.Eg=Eg;
net.parametros.m=m;
net.Eplot = Eplot;
net.pesos.IW=IW;
net.pesos.b=b;
net.pesos.LW=LW;
clear E IW N Ta Z cam db j nent neu Ea IWm Na X ans dIW ∆ m nsai Eg...
      LW Nmax Xa b dLW e n Erro LWm T Y bm dY fativ ncam pad Erroa
```

A.4 simular.m

```
function [Ys]=simular(net,S)
n=net.parametros.n;
Erro=net.parametros.Erro;
N=net.parametros.N;
Eg=net.parametros.Eg;
m=net.parametros.m;
Nmax=net.parametros.Nmax;
IW=net.pesos.IW;
b=net.pesos.b;
LW=net.pesos.LW;
nent=net.arquitetura.nent;
nsai=net.arquitetura.nsai;
ncam=net.arquitetura.ncam;
fativ=net.arquitetura.fativ;
Cs=size(S,2);
for pad=1:Cs
    Z{pad}{1}=IW{1}*S(:,pad) + b{1};
    if fativ{1,1}=='line';
        Y{pad}{1}=purelin(Z{pad}{1});
    elseif fativ{1,1}=='logs';
        Y{pad}{1}=logsig(Z{pad}{1});
    elseif fativ{1,1}=='tans';
        Y{pad}{1}=tansig(Z{pad}{1});
    end
    for cam=2:ncam
        Z{pad}{cam}=LW{cam,cam-1}*Y{pad}{cam-1} + b{cam};
```

```
        if fativ{1,cam}=='line';
            Y{pad}{cam}=purelin(Z{pad}{cam});
        elseif fativ{1,cam}=='logs';
            Y{pad}{cam}=logsig(Z{pad}{cam});
        elseif fativ{1,cam}=='tans';
            Y{pad}{cam}=tansig(Z{pad}{cam});
        end
    end
end
for pad=1:Cs
    Ys(:,pad)=Y{pad}{ncam};
end
```

A.5 validacao.m

```
clear;
clc;
Ct=3;
X=linspace(.25,.75,Ct);
T=X.^2;
net=redemlp(X,T,[5 5 1],{'tans','tans','tans'});
net.parametros.n=.1;
net.parametros.Erro=10^-5;
net.parametros.m=0;
net.parametros.Nmax=5000;
netd=treinar(net,X,T);
clf;
Yt=simular(netd,X);
net=redemlp(T,X,[5 5 1],{'tans','tans','tans'});
net.parametros.n=.1;
net.parametros.Erro=10^-5;
net.parametros.m=0;
net.parametros.Nmax=5000;
neti=treinar(net,Yt,X);
XI1=simular(neti,T);
XI2=simular(neti,Yt);
Cs=500;
S=linspace(.25,.75,Cs);
Ys=simular(netd,S);
SI=simular(neti,Ys);
alfa = Yt - T;
beta = XI1 - X;
falfa = XI2 - beta - X;
Malfa = sum(.5*alfa.^2)/Ct;
Mbeta = sum(.5*beta.^2)/Ct;
Mfalfa = sum(.5*falfa.^2)/Ct;
```

```
talfa = sqrt(2*Malfa);
tbeta = sqrt(2*Mbeta);
tfalfa = sqrt(2*Mfalfa);
tol = tbeta + tfalfa;
ei = S - SI;
a=linspace(.25,.75,500);
b=a.^2;
subplot(1,2,1)
plot(a,b,'-k')
hold on
plot(X,Yt,'x','LineWidth',2,'MarkerSize',10)
pause
subplot(1,2,2)
plot(b,a,'-k')
hold on
plot(Yt,XI1,'x','LineWidth',2,'MarkerSize',10)
pause
clf
subplot(1,2,1)
plot(a,b,'-k')
hold on
plot(S,Ys,'x','LineWidth',2,'MarkerSize',10)
pause
subplot(1,2,2)
plot(b,a,'-k')
hold on
plot(Ys,SI,'x','LineWidth',2,'MarkerSize',10)
pause
clf
stem(S,SI - S,'.')
hold on
plot([min(S) max(S)],[0 0],'-k')
pause
clf
for pad=1:Cs
    if abs(ei(pad))≤tol
        plot(S(pad),Ys(pad),'.')
        hold on
    else
        plot(S(pad),Ys(pad),'xk','LineWidth',2,'MarkerSize',10)
        hold on
    end
end
```

A.6 plotarConf.m

```
function [ out ] = plotarConf(Tvld,Ovld,Tvli,Ovli,Od,Oi,Xd,Td,Xs,Ts,aceit)
subplot(2,2,1)
hold on
if length(find(aceit==1))≠0
    for a=1:size(aceit,2)
        if aceit(a)==1
            vetorOK.X(1,a) = Xs(1,a);
            vetorOK.Y(1,a) = Ovld(1,a);
        else
            vetorNO.X(1,a) = Xs(1,a);
            vetorNO.Y(1,a) = Ovld(1,a);
        end
    end
    % remove os valores nulos
    vp1 = find(vetorOK.X≠0);
    for a = 1:length(vp1)
        vOKX(1,a)=vetorOK.X(1,vp1(a));
    end
    vp2 = find(vetorOK.Y≠0);
    for a = 1:length(vp2)
        vOKY(1,a)=vetorOK.Y(1,vp2(a));
    end
    clear vp1 vp2 a
    vp1 = find(vetorNO.X≠0);
    for a = 1:length(vp1)
        vNOX(1,a)=vetorNO.X(1,vp1(a));
    end
    vp2 = find(vetorNO.Y≠0);
    for a = 1:length(vp2)
        vNOY(1,a)=vetorNO.Y(1,vp2(a));
    end
    clear vp1 vp2 a
    plot(vOKX,vOKY, 'or','MarkerEdgeColor','r','MarkerFaceColor','r',...
        'MarkerSize',3);
    plot(vNOX,vNOY, 'xk','MarkerEdgeColor','k','MarkerFaceColor','k',...
        'MarkerSize',7);
    plot(Xd,Xd.^2,'s','MarkerEdgeColor','k','MarkerFaceColor','y',...
        'MarkerSize',7);
    a=linspace(1,5,500);
    b=a.^2;
    plot(a,b,'--b','LineWidth',2);
    legend('Saídas confiáveis','Saídas não confiáveis',...
        'Alvos treinamento','Curva esperada','Location','best');
    plot(vOKX,vOKY, 'or','MarkerEdgeColor','r','MarkerFaceColor','r',...
```

```
        'MarkerSize',3);
    plot(vNOX,vNOY, 'xk','MarkerEdgeColor','k','MarkerFaceColor','k',...
        'MarkerSize',7);
    plot(Xd,Xd.^2,'s','MarkerEdgeColor','k','MarkerFaceColor','y',...
        'MarkerSize',7);
    xlabel('\fontsize{12}X')
    ylabel('\fontsize{12}Y = X^2')
    %title('Confiabilidade das saídas rede Direta')
    grid on
else
    plot(Xs,Ovld,'xk','MarkerEdgeColor','k','MarkerFaceColor','k',...
        'MarkerSize',7);
    plot(Xd,Xd.^2,'s','MarkerEdgeColor','k','MarkerFaceColor','y',...
        'MarkerSize',7);
    legend('Saídas não confiáveis','Alvos treinamento')
    a=linspace(1,5,500);
    b=a.^2;
    plot(a,b,'-b','LineWidth',2);
    plot(Xs,Ovld,'xk','MarkerEdgeColor','k','MarkerFaceColor','k',...
        'MarkerSize',7);
    plot(Xd,Xd.^2,'s','MarkerEdgeColor','k','MarkerFaceColor','y',...
        'MarkerSize',7);
    xlabel('\fontsize{12}X')
    ylabel('\fontsize{12}Y = X^2')
    %title('\fontsize{12}Confiabilidade das saídas rede Direta')
    grid on
end
subplot(2,2,2)
hold on
plot(Ts,Ovli,'or','MarkerEdgeColor','r','MarkerFaceColor','r',...
    'MarkerSize',3);
plot(Td,Xd,'s','MarkerEdgeColor','k','MarkerFaceColor','y',...
    'MarkerSize',7);
legend('Saídas de utilização rede inversa','Alvos treinamento');
plot(b,a,'--b','LineWidth',2);
grid on;
ylabel('\fontsize{12}X')
xlabel('\fontsize{12}Y = X^2')
title('\fontsize{12}Utilização Rede Inversa')
```

APÊNDICE B

Códigos Fonte *Backpropagation* Complexo

B.1 MainX2complex.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Gera os dados de treinamento
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
clc;
npt=input('insira a Qtd de padrões de treinamento = ');
R=1;
teta = deg2rad(linspace(5,40,npt));
for a=1:npt
    X = R*exp(teta*i);
    T = X.^2;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Gera os dados de utilização
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nps=500;
teta = deg2rad(linspace(5,40,nps)); for a=1:nps
    Xs = R*exp(teta*i);
    Ts = Xs.^2;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Configuração/Parametros da rede DIRETA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net.hiden.nw  = 5;
net.alvo      = 7e-5;
net.u         = 0.1;
net.maxepocas = 20000;
format long
%Treinamento e utilização: REDE DIRETA
Xp =randperm(length(X));
X = X(Xp);
T = T(Xp);
netd  = treinar(X,T,net);
vld=validar(Xs,Ts,netd.pesos);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Configuração/Parametros da rede INVERSA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net.hiden.nw  = 5;
net.alvo      = 7e-5;
net.u         = 0.1;
net.maxepocas = 20000;
%Treinamento utilização: REDE INVERSA
netdp =randperm(length(netd.O));
netd.O = netd.O(netdp);
X = X(netdp);
neti  = treinar(netd.O,X,net);
vli=validar(vld.O,Xs,neti.pesos);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plota o comportamento do Erro Global da Redes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clf;
subplot(2,2,1)
semilogy(netd.Eplot,'-ob','LineWidth',1.5,'MarkerEdgeColor','b',...
    'MarkerFaceColor','b','MarkerSize',1.5);
hold on
plot([0 size(netd.Eplot,2)],[netd.alvo netd.alvo],'--r','LineWidth',2);
axis([0 size(netd.Eplot,2) netd.alvo/2.5  max(netd.Eplot) ]);
title('\fontsize{12}Rede Direta')
xlabel('\fontsize{12}épocas')
ylabel('\fontsize{12}MSE')
legend('Curva MSE',['MSE alvo = ' num2str(netd.alvo)]);
grid on; subplot(2,2,2);
semilogy(neti.Eplot,'-ob','LineWidth',1.5,'MarkerEdgeColor','b',...
    'MarkerFaceColor','b','MarkerSize',1.5);
hold on
plot([0 size(neti.Eplot,2)],[neti.alvo neti.alvo],'--r','LineWidth',2);
axis([0 size(neti.Eplot,2) neti.alvo/2.5  max(neti.Eplot) ]);
title('\fontsize{12}Rede Inversa')
```

```
xlabel('\fontsize{12}épocas')
ylabel('\fontsize{12}MSE')
legend('Curva MSE',['MSE alvo = ' num2str(neti.alvo)]);
grid on;
pause;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Computo do vetor "aceite": (1) - saída confiável (0) - não confiável
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Einv  = sum(abs(X-neti.O))/length(X);
Einv2 = sum(abs(abs(X)-abs(neti.O)))/length(X);
erro_inv_vl =abs(Xs-vli.O);
for a=1:length(vli.O)
    if erro_inv_vl(1,a)≤Einv
        aceite(1,a)=1;
    else
        aceite(1,a)=0;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Chama o scritp PlotarConf
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
plotarConf(vld.T,vld.O,vli.O,T,X,Ts,Xs,aceite);
```

B.2 treinar.m

```
function [ result ] = treinar(X,T,net)
nw = net.hiden.nw;
ne = 1;
ns = 1;
rand('state', sum(100*clock));
a=-0.45;
b=0.45;
p.w=(a+(b-a))*(rand(1,nw)+i*rand(1,nw));
p.h=(a+(b-a))*(rand(1,nw)+i*rand(1,nw));
Eg=1;
EgPad=0;
tmp=0;
epocas=0;
tic;
while (Eg>net.alvo)
    for pad=1:size(X,2)
        for a=1:nw
            net.w(1,a)=p.w(1,a)*X(1,pad);
            o.w(1,a)=afun(net.w(1,a));
        end
        for a = 1:nw
```

```
            tmp=tmp+p.h(1,a)*o.w(1,a);
        end
        net.k=tmp;
        o.k=afun(tmp);
        O(1,pad)= o.k;
        netj.O(1,pad)=tmp;
        tmp=0;
        erro=T(1,pad)-o.k;
        EgPad(1,pad)=0.5*abs(erro)^2;
        Δ.h=erro*conj(dfun(net.k));
        for a=1:nw
            Δ.w(1,a)=conj(p.h(1,a))*Δ.h*conj(dfun(net.w(1,a)));
        end
        for a=1:nw
            tmp=p.w(1,a);
            p.w(1,a)=tmp+net.u*Δ.w(1,a)*conj(X(1,pad));
        end
        for a=1:nw
            tmp=p.h(1,a);
            p.h(1,a)=tmp+net.u*Δ.h*conj(o.w(1,a));
        end
        tmp=0;
    end
    epocas=epocas+1;
    Eg=sum(EgPad)/size(X,2)
    Eplot(1,epocas)=Eg;
    if epocas≥net.maxepocas;
        break
    end
end
result.tempo=toc;
result.pesos=p;
result.Eg=Eg;
result.epocas=epocas;
result.T=T;
result.O=O;
result.Eplot=Eplot;
result.alvo=net.alvo;
result.netj.O=netj.O;
result.maxepocas=net.maxepocas;
```

B.3 validar.m

```
function [result] = validar(Xs,Ts,P);
p.w = P.w;
p.h = P.h;
nw = size(p.w,2);
tmp=0;
for pad=1:size(Xs,2)
    for a=1:nw
        net.w(1,a) = p.w(1,a) * Xs(1,pad);
        o.w(1,a)= afun(net.w(1,a));
    end
    for a = 1:nw
        tmp = tmp + p.h(1,a) * o.w(1,a);
    end
    net.k = tmp ;
    o.k = afun(tmp);
    O(1,pad)= o.k ;
    netj.O(1,pad) = tmp;
    tmp=0;

    erro = Ts(1,pad) - o.k;
    EgPad(1,pad)= 0.5 * abs(erro)^2;
end
Eg = sum(EgPad)/size(Xs,2);
result.Eg = Eg;
result.T = Ts;
result.O = O;
result.netj.O = netj.O;
```

B.4 afun.m

```
% Função de Ativação Complexa
function [ B ] = afun(b)
B = tanh(b);
```

B.5 dfun.m

```
% derivada da função de ativação complexa
function [ C ] = dfun(s)
C = 1 - tanh(s).^2;
```

B.6 plotarConf.m

```
function [ out ] = plotarConf(Tvld,Ovld,Ovli,Td,Xd,Ts,Xs,aceit)
clf;
hold on;
if length(find(aceit==0))≥1 & length(find(aceit==1))≥1
    vp = find(aceit==1);
    for i=1:length(vp)
        vOK(1,i) = Ovld(vp(i));
    end
    vp = find(aceit==0);
    for i=1:length(vp)
        vNO(1,i) = Ovld(vp(i));
    end
    plot(vOK,'or','MarkerEdgeColor','r','MarkerFaceColor','r',...
        'MarkerSize',2);
    plot(vNO,'xk','MarkerEdgeColor','k','MarkerFaceColor','k',...
        'MarkerSize',7);
    plot(Td,'s','MarkerEdgeColor','k','MarkerFaceColor','y'...
        ,'MarkerSize',5);
    legend('Saídas confiáveis','Saídas não confiáveis',...
        'Alvos treinamento');
    t = deg2rad(linspace(0,360,1000));
    t1 = exp(t*j);
    plot(t1,'-b','LineWidth',2.5);
    hold on;
    axis([-1.1 1.1 -1.1 1.1]);
    axis square;
    title('\fontsize{12}Confiabilidade das Saídas Rede Direta');
    xlabel('\fontsize{12}cos(\theta)');
    ylabel('\fontsize{12}i sin(\theta)'); grid on;
    plot([0 1],[0 0],'-k','LineWidth',1.5);
    tt = exp(deg2rad(90)*j);
    plot([0 real(tt)],[0 imag(tt)],'-k','LineWidth',1.5);
    tt = exp(deg2rad(2*5)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
    tt = exp(deg2rad(2*40)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
    plot(Td,'s','MarkerEdgeColor','k','MarkerFaceColor',...
        'y','MarkerSize',5);
    plot(vOK,'or','MarkerEdgeColor','r','MarkerFaceColor',...
        'r','MarkerSize',2);
    plot(vNO,'xk','MarkerEdgeColor','k','MarkerFaceColor','k',...
        'MarkerSize',7);
    plot(Td,'s','MarkerEdgeColor','k','MarkerFaceColor','y',...
        'MarkerSize',5); pause;
```

```
elseif length(find(aceit==1))== length(aceit)
    plot(Ovld,'or','MarkerEdgeColor','r','MarkerFaceColor','r',...
        'MarkerSize',2);
    plot(Td,'s','MarkerEdgeColor','k','MarkerFaceColor','y',...
        'MarkerSize',5);
    legend('Saídas confiáveis','Alvos treinamento');
    t = deg2rad(linspace(0,360,1000));
    t1 = exp(t*j);    plot(t1,'-b','LineWidth',2.5);
    hold on;    axis([-1.1 1.1 -1.1 1.1]);
    axis square
    title('\fontsize{12}Confiabilidade das Saídas Rede Direta')
    xlabel('\fontsize{12}cos(\theta)')
    ylabel('\fontsize{12}i sin(\theta)')
    grid on;    plot([0 1],[0 0],'-k','LineWidth',1.5);
    tt = exp(deg2rad(90)*j);
    plot([0 real(tt)],[0 imag(tt)],'-k','LineWidth',1.5);
    tt = exp(deg2rad(2*5)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
    tt = exp(deg2rad(2*40)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
    plot(Ovld,'or','MarkerEdgeColor','r','MarkerFaceColor','r',...
        'MarkerSize',2);
    plot(Td,'s','MarkerEdgeColor','k','MarkerFaceColor','y',...
        'MarkerSize',5); pause;
else
    plot(Ovld,'xk','MarkerEdgeColor','k','MarkerFaceColor','k',...
        'MarkerSize',7);
    plot(Td,'s','MarkerEdgeColor','k','MarkerFaceColor','y',...
        'MarkerSize',5);
    t = deg2rad(linspace(0,360,1000));
    t1 = exp(t*j);    plot(t1,'-b','LineWidth',2.5);
    hold on;    axis([-1.1 1.1 -1.1 1.1]);    axis square;
    title('\fontsize{12}Confiabilidade das Saídas Rede Direta');
    xlabel('\fontsize{12}cos(\theta)');
    ylabel('\fontsize{12}i sin(\theta)');    grid on;
    plot([0 1],[0 0],'-k','LineWidth',1.5);
    tt = exp(deg2rad(90)*j);
    plot([0 real(tt)],[0 imag(tt)],'-k','LineWidth',1.5);
    tt = exp(deg2rad(2*5)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
    tt = exp(deg2rad(2*40)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
    legend('Saídas não confiáveis','Alvos treinamento');
    plot(Ovld,'xk','MarkerEdgeColor','k','MarkerFaceColor','k',...
        'MarkerSize',7);
    plot(Td,'s','MarkerEdgeColor','k','MarkerFaceColor','y',...
        'MarkerSize',5);    pause;
```

```
end
clf;
plot(Ovli,'or','MarkerEdgeColor','r','MarkerFaceColor','r',...
    'MarkerSize',2);
hold on
plot(Xd,'s','MarkerEdgeColor','k','MarkerFaceColor','y',...
    'MarkerSize',5);
t = deg2rad(linspace(0,360,1000));
t1 = exp(t*j);
plot(t1,'LineWidth',2);
axis([-1.1 1.1 -1.1 1.1]);
axis square
title('\fontsize{12}Rede Direta')
xlabel('\fontsize{12}cos(\theta)')
ylabel('\fontsize{12}i sin(\theta)')
grid on
plot([0 1],[0 0],'-k','LineWidth',1.5);
tt = exp(deg2rad(90)*j);
plot([0 real(tt)],[0 imag(tt)],'-k','LineWidth',1.5);
tt = exp(deg2rad(5)*j);
plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
tt = exp(deg2rad(40)*j);
plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
plot(Xd,'s','MarkerEdgeColor','k','MarkerFaceColor','y','MarkerSize',5);
grid on
legend('Saídas de utilização','Alvos treinamento','Location','best');
title('\fontsize{12}Saídas de utilização Rede Inversa')
plot(Xd,'s','MarkerEdgeColor','k','MarkerFaceColor','y','MarkerSize',5);
pause;
clf;
plot(Ovld,'or','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',2);
hold on
plot(Td,'s','MarkerEdgeColor','k','MarkerFaceColor','y','MarkerSize',5);
t = deg2rad(linspace(0,360,1000));
t1 = exp(t*j);
plot(t1,'LineWidth',2);
axis([-1.1 1.1 -1.1 1.1]);
axis square
title('\fontsize{12}Rede Direta')
xlabel('\fontsize{12}cos(\theta)')
ylabel('\fontsize{12}i sin(\theta)')
grid on
plot([0 1],[0 0],'-k','LineWidth',1.5);
tt = exp(deg2rad(90)*j);
plot([0 real(tt)],[0 imag(tt)],'-k','LineWidth',1.5);
tt = exp(deg2rad(2*5)*j);
plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
```

```
tt = exp(deg2rad(2*40)*j);
plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
plot(Td,'s','MarkerEdgeColor','k','MarkerFaceColor','y','MarkerSize',5);
grid on
legend('Saídas de utilização','Alvos treinamento','Location','best');
title('\fontsize{12}Saídas de utilização Rede Direta')
plot(Td,'s','MarkerEdgeColor','k','MarkerFaceColor','y','MarkerSize',5);
```

APÊNDICE

C

Códigos Fonte *Backpropagation* de Clifford

C.1 MainX2Clifford.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Gera os dados de treinamento
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
clc;
npt=input('insira a Qtd de padrões de treinamento = ');
R=1;
teta = deg2rad(linspace(5,40,npt));
for a=1:npt
    X(:,a) = m(R*gexp(teta(1,a)*e1^e2));
    T(:,a) = m(GA(X(:,a))*GA(X(:,a)));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Gera os dados de utilização
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nps=500;
teta = deg2rad(linspace(5,40,nps));
for a=1:nps
Xs(:,a) = m(R*gexp(teta(1,a)*e1^e2));
Ts(:,a) = m(GA(Xs(:,a))*GA(Xs(:,a)));
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Configuração/Parametros da rede DIRETA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net.hiden.nw  = 3;
net.alvo      = 7e-5;
net.u         = 0.1;
net.maxepocas = 20000;
format long;
%Treinamento e utilização: REDE DIRETA
Xp =randperm(size(X,2));
X = X(:,Xp);
T = T(:,Xp);
netd=treinar(X,T,net);
vld=validar(Xs,Ts,netd.pesos);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Configuração/Parametros da rede INVERSA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net.hiden.nw  = 3;
net.alvo      = 7e-5;
net.u         = 0.1;
net.maxepocas = 20000;
%Treinamento utilização: REDE INVERSA
netdp =randperm(size(netd.O,2));
netd.O = netd.O(:,netdp);
X = X(:,netdp);
neti=treinar(netd.O,X,net);
vli=validar(vld.O,Xs,neti.pesos);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plota o comportamento do Erro Global da Redes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clf;    subplot(2,2,1);
semilogy(netd.Eplot,'-ob','LineWidth',1.5,'MarkerEdgeColor','b',...
    'MarkerFaceColor','b','MarkerSize',1.5);
hold on
plot([0 size(netd.Eplot,2)],[netd.alvo netd.alvo],'--r','LineWidth',2);
axis([0 size(netd.Eplot,2) netd.alvo/2.5  max(netd.Eplot) ]);
title('\fontsize{12}Rede Direta')
xlabel('\fontsize{12}épocas')
ylabel('\fontsize{12}MSE')
legend('Curva MSE',['MSE alvo = ' num2str(netd.alvo)]);
grid on
subplot(2,2,2)
semilogy(neti.Eplot,'-ob','LineWidth',1.5,'MarkerEdgeColor','b',...
    'MarkerFaceColor','b','MarkerSize',1.5);
hold on
plot([0 size(neti.Eplot,2)],[neti.alvo neti.alvo],'--r','LineWidth',2);
axis([0 size(neti.Eplot,2) neti.alvo/2.5  max(neti.Eplot) ]);
```

```
title('\fontsize{12}Rede Inversa')
xlabel('\fontsize{12}épocas')
ylabel('\fontsize{12}MSE')
legend('Curva MSE',['MSE alvo = ' num2str(neti.alvo)]);
grid on
pause
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Computo do erro do erro: Einv
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Einv=0;
Einv2=0;
for k=1:size(X,2)
    Einv=Einv+norm(GA(neti.O(:,k))-GA(X(:,k)));
    Einv2=Einv2+abs(norm(GA(neti.O(:,k)))-norm(GA(X(:,k))));
end
Einv=Einv/(size(X,2));
Einv2=Einv2/(size(X,2));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Computo do vetor "aceite": (1) - saída confiável (0) - não confiável
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for a=1:size(vli.O,2)
    erro_inv_vl(1,a)=norm(GA(Xs(:,a))-GA(vli.O(:,a)));
    if erro_inv_vl(1,a)≤Einv
        aceite(1,a)=1;
    else
        aceite(1,a)=0;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Chama o scritp PlotarConf
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clf;
plotarConf(vld.T,vld.O,vli.O,T,X,aceite);
```

C.2 treinar.m

```
function [ result ] = treinar(X,T,net)
nw = net.hiden.nw;
ne = 1;
ns = 1;
rand('state', sum(100*clock));
p.w=zeros(8,ne*nw);
a=-.5;
b =.5;
p.w(1,:)=a+(b-a).*rand(1,size(p.w,2));
p.w(5,:)=a+(b-a).*rand(1,size(p.w,2));
```

```
p.h=zeros(8,nw*ns);
a=-.5;
b =.5;
p.h(1,:) = a + (b-a).*rand(1,size(p.w,2));
p.h(5,:) = a + (b-a).*rand(1,size(p.w,2));
Eg=1;
EgPad=0;
tmp=0;
epocas=0;
while (Eg>net.alvo)
    for pad=1:size(X,2)
        for a=1:nw
            net.w(:,a)=m(GA(p.w(:,a))*GA(X(:,pad)));
            o.w(:,a)=m(afun(GA(net.w(:,a))));
        end
        for a=1:nw
            tmp=tmp+GA(p.h(:,a))*GA(o.w(:,a));
        end
        net.k=m(tmp);
        netj.O(:,pad) = m(tmp);
        o.k=afun(tmp);
        O(:,pad)=m(o.k);
        tmp=0;
        erro = GA(T(:,pad)) - o.k;
        EgPad(1,pad)= 0.5*(sum(m(erro).^2));
        Δ.h = erro* conjugate(dfun(GA(net.k)));
        for a=1:nw
            Δ.w(:,a)= m(conjugate(GA(p.h(:,a))) * Δ.h *...
                conjugate(dfun(GA(net.w(:,a)))));
        end
        for a=1:nw
            tmp=GA(p.w(:,a));
            p.w(:,a)=m(tmp+net.u*GA(Δ.w(:,a))*conjugate(GA(X(:,pad))));
        end
        for a=1:nw
            tmp=GA(p.h(:,a));
            p.h(:,a)=m(tmp+net.u*Δ.h*conjugate(GA(o.w(:,a))));
        end
        tmp=0;
    end
    epocas=epocas+1;
    Eg=sum(EgPad)/size(X,2)
    Eplot(1,epocas)=Eg;
    if epocas≥net.maxepocas;
        break
    end
end
```

```
result.pesos=p;
result.Eg=Eg;
result.epoc=epocas;
result.T=T;
result.O=O;
result.Eplot=Eplot;
result.netj=netj.O;
result.alvo=net.alvo;
```

C.3 validar.m

```
function [result] = validar(Xs,Ts,P);
p.w = P.w;
p.h = P.h;
nw = size(p.w,2);
tmp=0;
for pad=1:size(Xs,2)
    %FASE DE FORWARD
    for a=1:nw
        net.w(:,a)=m(GA(p.w(:,a))*GA(Xs(:,pad)));
        o.w(:,a)=m(afun(GA(net.w(:,a))));
    end
    for a=1:nw
        tmp=tmp+GA(p.h(:,a))*GA(o.w(:,a));
    end
    net.k=m(tmp);
    o.k=afun(tmp);
    O(:,pad)=m(o.k);
    tmp=0;
    erro = GA(Ts(:,pad)) - o.k;
    EgPad(1,pad)= 0.5*(sum(m(erro).^2));
end
Eg = sum(EgPad)/size(Xs,2);
result.Eg = Eg;
result.T = Ts;
result.O = O;
```

C.4 afun.m

```
% FUNÇÃO DE ATIVAÇÃO TANH, EM CLIFFORD
function [ B ] = afun(b)
B = GAdivide(GAgexp(b) - GAgexp(-b),GAgexp(b) + GAgexp(-b) );
```

C.5 dfun.m

```
% DERIVADA DA FUNÇÃO TANH EM CLIFFORD.
function [ C ] = dfun(s)
 C = (2/(GAgexp(s)+GAgexp(-s)))*(2/(GAgexp(s)+GAgexp(-s)));
```

C.6 plotarConf.m

```
function [ out ] = plotarConf(Tvld,Ovld,Ovli,Td,Xd,aceit)
clf;
hold on;
if length(find(aceit==0))≥1 & length(find(aceit==1))≥1
    vp = find(aceit==1);
    for a=1:length(vp)
        vxOK(1,a)=Ovld(1,vp(a));
        vyOK(5,a)=Ovld(5,vp(a));
    end
    vp = find(aceit==0);
    for a=1:length(vp)
        vxNO(1,a)=Ovld(1,vp(a));
        vyNO(5,a)=Ovld(5,vp(a));
    end
    plot(vxOK(1,:),vyOK(5,:),'or','MarkerEdgeColor','r',...
        'MarkerFaceColor','r','MarkerSize',2);
    plot(vxNO(1,:),vyNO(5,:),'xk','MarkerEdgeColor','k',...
        'MarkerFaceColor','k','MarkerSize',7);
    plot(Td(1,:),Td(5,:),'s','MarkerEdgeColor','k',...
        'MarkerFaceColor','y','MarkerSize',5);
    legend('Saídas confiáveis','Saídas não confiáveis',...
        'Alvos treinamento');
    t = deg2rad(linspace(0,360,1000));
    t1 = exp(t*j);
    plot(t1,'-b','LineWidth',2.5);
    hold on
    axis([-1.1 1.1 -1.1 1.1]);
    axis square
    title('\fontsize{12}Confiabilidade das Saídas Rede Direta')
    xlabel('\fontsize{12}cos(\theta)')
    ylabel('\fontsize{12}e_1e_2  sin(\theta)')
    grid on
    plot([0 1],[0 0],'-k','LineWidth',1.5);
    tt = exp(deg2rad(90)*j);
    plot([0 real(tt)],[0 imag(tt)],'-k','LineWidth',1.5);
    tt = exp(deg2rad(2*5)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
```

```
    tt = exp(deg2rad(2*40)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
    plot(vxOK(1,:),vyOK(5,:),'or','MarkerEdgeColor','r',...
        'MarkerFaceColor','r','MarkerSize',2);
    plot(vxNO(1,:),vyNO(5,:),'xk','MarkerEdgeColor','k',...
        'MarkerFaceColor','k','MarkerSize',7);
    plot(Td(1,:),Td(5,:),'s','MarkerEdgeColor','k',...
        'MarkerFaceColor','y','MarkerSize',5);
    pause;
elseif length(find(aceit==1))== length(aceit)
    plot(Ovld(1,:),Ovld(5,:),'or','MarkerEdgeColor','r',...
        'MarkerFaceColor','r','MarkerSize',2);
    plot(Td(1,:),Td(5,:),'s','MarkerEdgeColor','k',...
        'MarkerFaceColor','y','MarkerSize',5);
    legend('Saídas confiáveis','Alvos treinamento');
    t = deg2rad(linspace(0,360,1000));
    t1 = exp(t*j);
    plot(t1,'-b','LineWidth',2.5);
    hold on
    axis([-1.1 1.1 -1.1 1.1]);
    axis square
    title('\fontsize{12}Confiabilidade das Saídas Rede Direta')
    xlabel('\fontsize{12}cos(\theta)')
    ylabel('\fontsize{12}e_1e_2  sin(\theta)')
    grid on
    plot([0 1],[0 0],'-k','LineWidth',1.5);
    tt = exp(deg2rad(90)*j);
    plot([0 real(tt)],[0 imag(tt)],'-k','LineWidth',1.5);
    tt = exp(deg2rad(2*5)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
    tt = exp(deg2rad(2*40)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
    plot(Ovld(1,:),Ovld(5,:),'or','MarkerEdgeColor','r',...
        'MarkerFaceColor','r','MarkerSize',2);
    plot(Td(1,:),Td(5,:),'s','MarkerEdgeColor','k',...
        'MarkerFaceColor','y','MarkerSize',5);
    pause;
else
    plot(Ovld(1,:),Ovld(5,:),'xk','MarkerEdgeColor','k',...
        'MarkerFaceColor','k','MarkerSize',7);
    plot(Td(1,:),Td(5,:),'s','MarkerEdgeColor','k',...
        'MarkerFaceColor','y','MarkerSize',5);
    t = deg2rad(linspace(0,360,1000));
    t1 = exp(t*j);
    plot(t1,'-b','LineWidth',2.5);
    hold on
    axis([-1.1 1.1 -1.1 1.1]);
```

```
    axis square
    title('\fontsize{12}Confiabilidade das Saídas Rede Direta')
    xlabel('\fontsize{12}cos(\theta)')
    ylabel('\fontsize{12}e_1e_2  sin(\theta)')
    grid on
    plot([0 1],[0 0],'-k','LineWidth',1.5);
    tt = exp(deg2rad(90)*j);
    plot([0 real(tt)],[0 imag(tt)],'-k','LineWidth',1.5);
    tt = exp(deg2rad(2*5)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
    tt = exp(deg2rad(2*40)*j);
    plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
    legend('Saídas não confiáveis','Alvos treinamento');
    plot(Ovld(1,:),Ovld(5,:),'xk','MarkerEdgeColor','k',...
        'MarkerFaceColor','k','MarkerSize',7);
    plot(Td(1,:),Td(5,:),'s','MarkerEdgeColor','k',...
        'MarkerFaceColor','y','MarkerSize',5);
    pause;
end
clf;
plot(Ovli(1,:),Ovli(5,:),'or','MarkerEdgeColor','r',...
    'MarkerFaceColor','r','MarkerSize',2);
hold on
plot(Xd(1,:),Xd(5,:),'s','MarkerEdgeColor','k',...
    'MarkerFaceColor','y','MarkerSize',5);
t = deg2rad(linspace(0,360,1000));
t1 = exp(t*j);
plot(t1,'LineWidth',2);
axis([-1.1 1.1 -1.1 1.1]);
axis square
title('\fontsize{12}Rede Direta')
xlabel('\fontsize{12}cos(\theta)')
ylabel('\fontsize{12}e_1e_2  sin(\theta)')
grid on
plot([0 1],[0 0],'-k','LineWidth',1.5);
tt = exp(deg2rad(90)*j);
plot([0 real(tt)],[0 imag(tt)],'-k','LineWidth',1.5);
tt = exp(deg2rad(5)*j);
plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
tt = exp(deg2rad(40)*j);
plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
plot(Xd(1,:),Xd(5,:),'s','MarkerEdgeColor','k',...
    'MarkerFaceColor','y','MarkerSize',5);
grid on
legend('Saídas de utilização','Padrões de treinamentos','Location','best');
title('\fontsize{12}Saídas de utilização Rede Inversa');
pause
```

```
clf;

plot(Ovld(1,:),Ovld(5,:),'or','MarkerEdgeColor','r',...
    'MarkerFaceColor','r','MarkerSize',2);
hold on
plot(Td(1,:),Td(5,:),'s','MarkerEdgeColor','k',...
    'MarkerFaceColor','y','MarkerSize',5);
t = deg2rad(linspace(0,360,1000));
t1 = exp(t*j);
plot(t1,'LineWidth',2);
axis([-1.1 1.1 -1.1 1.1]);
axis square
title('\fontsize{12}Rede Direta')
xlabel('\fontsize{12}cos(\theta)')
ylabel('\fontsize{12}e_1e_2  sin(\theta)')
grid on
plot([0 1],[0 0],'-k','LineWidth',1.5);
tt = exp(deg2rad(90)*j);
plot([0 real(tt)],[0 imag(tt)],'-k','LineWidth',1.5);
tt = exp(deg2rad(2*5)*j);
plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
tt = exp(deg2rad(2*40)*j);
plot([0 real(tt)],[0 imag(tt)],'--k','LineWidth',1.5);
plot(Td(1,:),Td(5,:),'s','MarkerEdgeColor','k',...
    'MarkerFaceColor','y','MarkerSize',5);
grid on
legend('Saídas de utilização','Padrões de treinamentos','Location','best');
title('\fontsize{12}Saídas de utilização Rede Inversa');
```