UNIVERSIDADE FEDERAL DO PARANÁ

PATRICK KREZANOVSKY

REDES NEURAIS ARTIFICIAIS APLICADAS A PROBLEMAS DE PREVISÃO E CLASSIFICAÇÃO

CURITIBA

2016

PATRICK KREZANOVSKY

REDES NEURAIS ARTIFICIAIS APLICADAS A PROBLEMAS DE PREVISÃO E CLASSIFICAÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Matemática Industrial da Universidade Federal do Paraná como requisito à obtenção do título de Bacharel em Matemática Industrial

Orientadora: Prof. Dra. Mariana Kleina

CURITIBA

2016

TERMO DE APROVAÇÃO

PATRICK KREZANOVSKY

REDES NEURAIS ARTIFICIAIS APLICADAS A PROBLEMAS DE PREVISÃO E CLASSIFICAÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Matemática Industrial da Universidade Federal do Paraná como requisito à obtenção do título de Bacharel em Matemática Industrial, pela seguinte banca examinadora:

_____
Profª. Dra. Mariana Kleina
Orientadora – Departamento de Engenharia de Produção da Universidade Federal do Paraná, UFPR

_____
Prof. Dr. Volmir Eugenio Wilhelm
Departamento de Engenharia de Produção da Universidade Federal do Paraná, UFPR

_____
Prof. Dr. Luiz Carlos Matioli
Departamento de Matemática da Universidade Federal do Paraná, UFPR

_____
Profª. Ma. Alana Renata Ribeiro
Departamento de Engenharia de Produção da Faculdade Educacional Araucária, FACEAR

Curitiba, 19 de Dezembro de 2016

Aos meus pais, amigos e professores.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter me dado a vida e todas as oportunidades que tive o prazer de vivenciar.

A Universidade Federal do Paraná, pelas oportunidades oferecidas e pela honra de poder fazer parte da sua comunidade acadêmica.

Aos meus professores que ao longo desse longo período me passaram conhecimento e ajudaram na minha formação pessoal e profissional

A minha orientadora Profª. Dra. Mariana Kleina, pelo suporte no pouco tempo que tivemos, pelas suas correções e incentivos.

Aos meus pais e minha irmã, que desde o início do curso estavam ao meu lado, me apoiando, incentivando, aconselhando e me ajudando a enfrentar as dificuldades encontradas, sem eles nenhum sonho se tornaria realidade. Estendo esse agradecimento a todos os meus familiares.

A todos os meus amigos, que direta ou indiretamente me ajudaram nesse período universitário, pelos momentos de estudo, de ensino e de descontração. Tenho certeza que tanto as amizades anteriores a vida acadêmica como as feitas devido a convivência na universidade ficarão para sempre lembradas.

Deixo aqui, meu muito obrigado a todos.

RESUMO

Neste trabalho é realizado um estudo detalhado de duas redes neurais artificiais muito utilizadas em diversas aplicações: as Redes Perceptron Multicamadas (MLP) e as Redes de Funções de Bases Radiais (RBF). Tais redes podem ser aplicadas em várias áreas de conhecimento, realizando diversos procedimentos, como por exemplo, classificação, previsão, aproximação de funções, dentre outras aplicabilidades. A estrutura básica aplicada para ambas as redes estudadas consiste em uma camada de entrada, uma camada oculta e uma camada de saída, no entanto, cada modelo possui particularidades específicas, fundamentais para o perfeito mapeamento das informações. O estudo aqui realizado será baseado nas caraterísticas principais de cada modelo neural. As redes foram implementadas em *Visual Basic For Applications* (VBA) e os resultados obtidos com a aplicação em problemas reais de classificação e previsão se mostraram satisfatórios e condizentes com a realidade.

Palavras-Chave: Redes Neurais Artificiais, Redes Perceptron Multicamadas, Redes de Funções de Bases Radiais, Classificação, Previsão.

ABSTRACT

In this work, a refined study is performed about two artificial neural networks, which are very used in several applications: The Multilayer Perceptron (MLP) network and the Radial Basis Function (RBF) network. These networks can be applied in different areas of knowledge, performing various procedures, such as classification, prediction, approximation of functions, among other applications. The basic structure applied for both studied networks is defined by an input layer, a hidden layer and an output layer. However, each model has its specific peculiarities, which are extremely important for the perfect mapping of information. The study performed in this work will be based on the main characteristics of each neural model. The networks were implemented in Visual Basic For Applications (VBA) and the results obtained from the application in real situations of classification and prediction proved to be satisfactory and consistent with reality.

Keywords: Artificial Neural Networks, Multilayer Perceptron, Radial Basis Function, Classification, Prediction.

SUMÁRIO

1 INTRODUÇÃO 9

2 REDES NEURAIS ARTIFICIAIS 10

2.1 HISTÓRIA 10

2.2 NEURÔNIO BIOLÓGICO 11

2.3 NEURÔNIO ARTIFICIAL 13

2.4 FUNÇÕES DE ATIVAÇÃO 14

2.5 ARQUITETURA 18

2.6 APRENDIZADO 19

2.6.1 Aprendizado Supervisionado 20

2.6.2 Aprendizado Não-supervisionado 20

3 REDES NEURAIS UTILIZADAS 22

3.1 REDES *PERCEPTRON* MULTICAMADAS 22

3.1.1 Introdução 22

3.1.2 Topologia 23

3.1.3 Treinamento 24

3.1.3.1 Cálculo do Erro 25

3.1.3.2 Atualização dos Dados Livres 26

3.1.3.2.1 Gradiente Local para a Camada de Saída 27

3.1.3.2.2 Gradiente Local para as Camadas Ocultas 27

3.1.3.3 Taxa de Aprendizagem 28

3.1.3.4 Modos de Treinamento 28

3.1.3.5 Critérios de Parada 29

3.1.4 O Algoritmo 29

3.2 REDES DE FUNÇÕES DE BASES RADIAIS 30

3.2.1 Introdução 30

3.2.2 Treinamento 32
3.2.2.1 Primeira Fase de Treinamento 32
3.2.2.2 Segunda Fase de Treinamento 34
3.2.3 O Algoritmo 36
3.3 COMPARAÇÃO ENTRE MLP E RBF 37
4 APLICAÇÕES 39
4.1 CLASSIFICAÇÃO DE PADRÕES 39
4.1.1 O Problema XOR 39
4.1.2 Classificação Musical por Gênero 41
4.2 PREVISÃO DE SÉRIES TEMPORAIS 44
4.2.1 Previsão do Volume de Vendas de Refrigerantes em Curitiba 44
4.2.2 Previsão das Taxas Anuais de Crescimento do Ibovespa 47
5 CONCLUSÕES 53
REFERÊNCIAS 54
APÊNDICE 1 – ALGORITMO MLP 55
APÊNDICE 2 – ALGORITMO RBF 58

1 INTRODUÇÃO

Baseados na estrutura cerebral, os modelos computacionais conhecidos como redes neurais artificiais (RNA's), ou apenas redes neurais, são altamente utilizados devido à sua capacidade de aprender por meio de exemplos ou informações e fornecer respostas coerentes para dados não informados.

As redes neurais são estruturadas por unidades de processamento simples paralelamente distribuídas, conhecidos por nodos, que computam determinadas funções matemáticas (normalmente não lineares), que são interligadas por um grande número de conexões (sinapses). Tais sinapses estão ligadas a pesos, que possuem a finalidade de ponderar a entrada recebida por cada neurônio da rede, armazenando o conhecimento adquirido pelo modelo. (BRAGA *et al.*, 2000; REZENDE, 2003).

A aplicabilidade das RNA's ocorre em diversas áreas, conforme descreve Silva *et al.* (2010): áreas de química, biologia, medicina, finanças e economia, setor automotivo, dentre outras mais. Tarefas de classificação, reconhecimento de padrões, aproximação de funções e previsão de séries temporais são exemplos de como as redes neurais podem ser empregadas.

O que torna as redes neurais tão atrativas na atualidade tecnológica é a capacidade de aprender utilizando dados e retornar respostas coerentes para informações antes desconhecidas, que segundo Braga *et al.* (2000, p. 2), "é uma demonstração de que a capacidade das RNA's vai muito além do que o simples mapear relações de entrada e saída".

Para Ludwig Jr. e Costa (2007), o fato de que não há como se obter uma prova formal para um resultado obtido é uma desvantagem das RNA's. Essa afirmação parte do princípio de que só é possível confirmar se o resultado obtido pela rede neural é confiável analisando o erro dos dados iniciais propostos (usando por exemplo o erro quadrático médio). De outra forma, é impossível saber o motivo de tal resultado ser retornado, até porque, nada se sabe a respeito da relevância de um peso sináptico para uma certa resposta ou até mesmo o significado físico do mesmo.

2 REDES NEURAIS ARTIFICIAIS

2.1 HISTÓRIA

Durante os acontecimentos da segunda guerra mundial, mais precisamente em 1943, Warren McCulloch e Walter Pitts, o primeiro neurofisiologista e o segundo matemático, publicaram um artigo sobre redes neurais intitulado "*A logical Calculus of the Ideas Immament in the Nervous Activity*", McCulloch e Pitts[1] (1943, citado por Braga *et al.*, 2000, p. 2). O trabalho apresenta o primeiro modelo artificial baseado em neurônios biológicos, mostrando as relações sinápticas e as capacidades computacionais das redes. Tal modelo evidenciado por McCulloch e Pitts (1943) é até hoje a base de estudos para todos os modelos de redes neurais utilizadas. (BRAGA *et al.*, 2000).

Após o grande primeiro estudo sobre redes neurais em 1943, Donald Hebb[2] (1949, citado por Ludwig Jr. e Costa, 2007, p. 5), apresentou o primeiro modelo de treinamento via aprendizado das RNA's, visando a atualização dos pesos sinápticos.

Entre 1957 e 1958 Frank Rosenblat desenvolveu o modelo neural *perceptron*, que foi inicialmente baseado no sistema nervoso visual. (LUDWIG JR. e COSTA, 2007). O *perceptron* é uma rede composta por 2 níveis, o de entrada (retina) e o de saída. Este modelo utiliza da classificação de padrões para categorizar as entradas, é usado apenas para problemas mais simples e linearmente separáveis, ou seja, problemas em que os dados podem ser separados por uma única reta ou hiperplano. (BRAGA *et al.*, 2000).

O *perceptron* criou grandes expectativas no ambiente computacional da época, porém suas limitações despertaram o interesse de Marvin Minsky e Seymor Papert, que, em 1969 apresentaram tais limitações no livro "*Perceptrons – an*

[1] MCCULLOCH, W. S.; PITTS, W. A Logical Calculus of the Ideas Immament in the Nervous Activity. Bulletin of Mathematical Biophisycs, 5:115-133, 1943.

[2] HEBB, D. O., The Organization of Behavior. Wiley, 1949.

introduction to computational geometry". (Minsky e Papert[3], 1969 citado por Silva *et al.*, 2010, p.26). Um dos principais fundamentos era a ineficiência do *perceptron* em identificar os padrões da função lógica XOR (ou-exclusivo). (SILVA *et al.*, 2010).

$$A\ XOR\ B \iff (A \vee B) \wedge \sim(A \wedge B)$$

Em geral, o estudo de Minsky e Papert (1969) argumentava o fato de que o *perceptron* não garantia a convergência para problemas não-linearmente separáveis, principalmente devido a rede possuir apenas uma camada de neurônios. A partir dos anos 70, poucos estudos ocorreram, particularmente devido a desestimulação causada pelas limitações do *perceptron*, deixando as redes neurais estagnadas.

Apenas no decorrer dos anos 80 que o interesse pelas redes neurais veio à tona novamente, contribuída pelo avanço tecnológico e técnicas de programação mais eficientes. Em 1982, John Hopfield apresentou um sistema neurocomputacional baseado no modelo neural de uma lesma, onde reacendeu de vez o interesse pelas RNA's. (SILVA *et al.*, 2010).

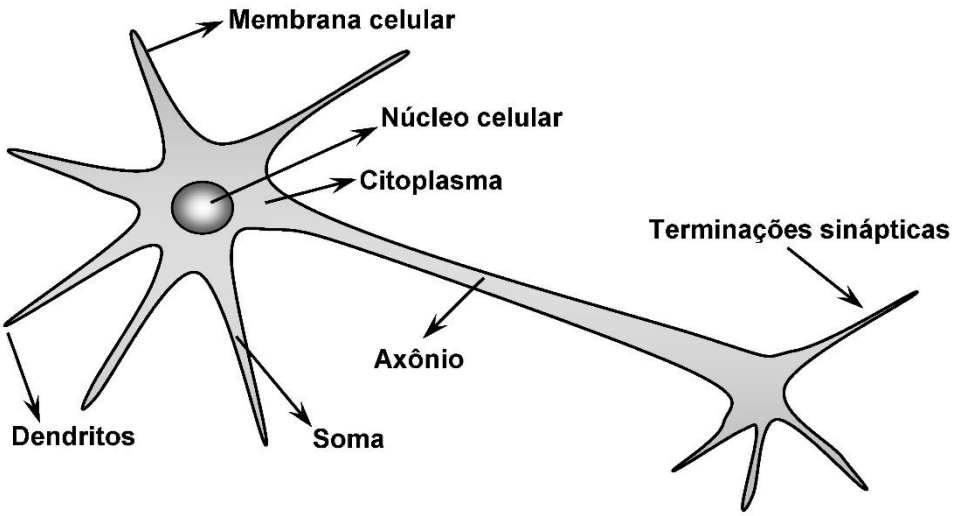
No final dos anos 80, foi proposto por Rumelhart o método *back-propagation* para atualizar os pesos sinápticos em mais de uma camada, quebrando de uma vez por todas as limitações apresentadas por Minsky e Papert (1969). (SILVA *et al.*, 2010).

2.2 NEURÔNIO BIOLÓGICO

Os neurônios são células nervosas que desempenham o papel de conduzir os impulsos nervosos. O processamento de informações e estímulos do corpo humano, são, então, responsabilidades dos neurônios. Essas células são compostas de três partes principais: o corpo da célula, os dendritos e o axônio (FIGURA 1).

[3] MINSKY, M.; PAPERT, W. Perceptrons – an introduction to computacional geometry. MIT Press, Massachusetts, 1969.

FIGURA 1 - NEURÔNIO BIOLÓGICO



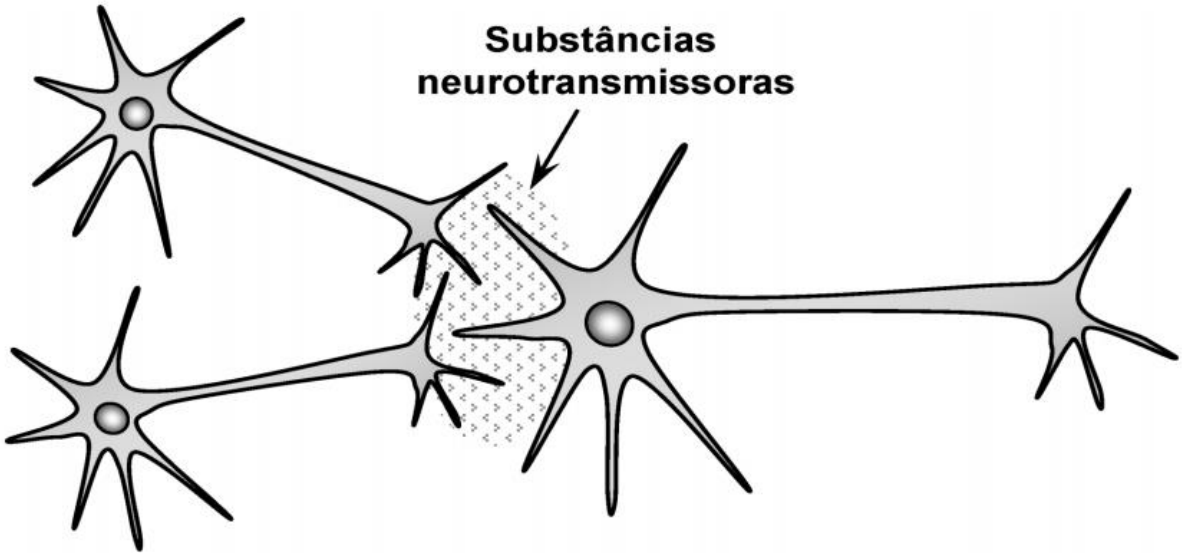
FONTE: SILVA *et al.* (2010, p. 29).

Os dendritos, segundo Braga *et al.* (2000), possuem poucos milímetros de comprimento e são responsáveis pelo recebimento e condução das informações oriundas de outros neurônios ou do meio externo onde os mesmos podem estar em contato.

O corpo do neurônio, também chamado de soma, é composto de várias organelas citoplasmáticas, como núcleo, mitocôndria, lisossomo, etc. O corpo celular é encarregado de coletar e processar as informações enviadas pelos dendritos. Aqui, um potencial de ativação é produzido e indicará se os impulsos nervosos serão ou não disparados pelo axônio.

O axônio é composto de uma fibra tubular onde os impulsos nervosos são conduzidos para outros neurônios. A terminação dos axônios é ramificada e recebe a denominação de terminações sinápticas, que se conectam, mesmo que sem contato físico, com os dendritos de outros neurônios (FIGURA 2). Esse contato que se encarrega de transferir os impulsos nervosos de um neurônio para o outro é chamado de sinapse. (SILVA *et al.*, 2010, KOVÁCS, 2006).

FIGURA 2 - CONEXÕES SINÁPTICAS ENTRE NEURÔNIOS



FONTE: SILVA *et al.* (2010, p. 30).

Segundo Braga *et al.* (2000) e Silva *et al.* (2010), o cérebro humano é composto por aproximadamente 10 bilhões (10^{11}) de neurônios interligados por conexões sinápticas, que, trabalhando em conjunto, são responsáveis pela maioria das funções executadas pelo cérebro.

2.3 NEURÔNIO ARTIFICIAL

O neurônio matemático, inicialmente proposto por McCulloch e Pitts é até hoje o modelo mais utilizado nas diferentes arquiteturas das redes neurais. Os modelos matemáticos são uma forma simplificada dos neurônios biológicos e se assemelham em dois aspectos principais de acordo com Haykin (2001, p. 28):

1. O conhecimento é adquirido pela rede a partir de seu ambiente através de um processo de aprendizagem. (HAYKIN, 2001, P. 28).
2. Forças de conexão entre neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar conhecimento adquirido. (HAYKIN, 2001, P. 28).

O neurônio computacional recebe um ou mais sinais de entrada e retorna somente um sinal de saída, podendo este ser um retorno final da rede ou um dado

de entrada para outro neurônio de outra camada. Podem-se caracterizar as redes neurais como estruturas de unidades de processamento paralelamente distribuídas devido ao fato dos neurônios receberem as informações simultaneamente. (LUDWIG JR. e COSTA, 2007).

O corpo do neurônio realiza a soma da multiplicação termo a termo entre os *n* sinais de entrada $\{x_1, x_2, \dots, x_n\}$, analogamente representando os impulsos elétricos captados pelos dendritos, e os *n* pesos sinápticos $\{w_1, w_2, \dots, w_n\}$ que informam o quão relevante é cada uma das entradas no neurônio.

Uma possibilidade a mais é a inclusão de uma polarização ou *bias*, caracterizado pela letra *b*. O *bias* proporciona uma liberdade mais ampla na ponderação dos dados de entrada, além de uma maior capacidade de aproximação da rede. Ainda mais, o *bias* permite que um neurônio apresente uma saída não nula, caso todos os dados de entrada sejam nulos. Essa variável é somada na função ativação e ajustada da mesma forma que os pesos sinápticos. (LUDWIG JR. e COSTA, 2007).

$$v = \sum_{i=1}^{n} w_i \cdot x_i + b. \qquad 1$$

A função da Equação (1) é denominada pela letra v representando a saída do corpo celular artificial. Tal valor é apresentado a uma função de ativação, responsável por limitar a saída do neurônio evitando o acréscimo progressivo dos valores de saída ao longo das camadas da rede. (LUDWIG JR. e COSTA, 2007; SILVA *et al.*, 2010).

2.4 FUNÇÕES DE ATIVAÇÃO

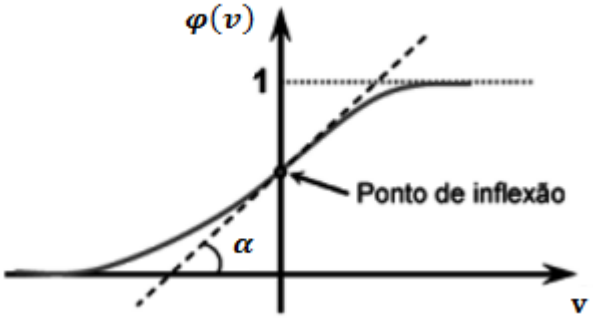
Dentre todas as funções de ativação, as mais utilizadas e que possuem maior destaque são:

- Função Sigmoidal Logística: É uma função semilinear, monotônica e limitada, onde seus valores de saída serão sempre reais e restritos entre zero e um.

$$\varphi(v) = \frac{1}{1 + e^{-\alpha v}}, \qquad (2)$$

onde α é uma constante real que determina a suavidade da curva.

FIGURA 3 – FUNÇÃO SIGMOIDAL LOGÍSTICA



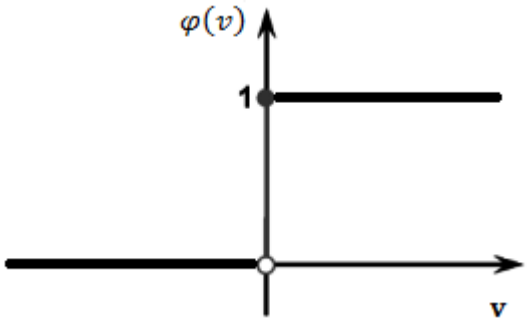
FONTE: SILVA *et al.* (2010, p. 39).

- Função Degrau: função não diferenciável assumindo resultados diferentes conforme os valores de v.

$$\varphi(v) = \begin{cases} 1, & se\ v \geq 0 \\ 0, & se\ v < 0. \end{cases} \qquad (3)$$

A Figura 4 ilustra o gráfico da função degrau.

FIGURA 4 – FUNÇÃO DEGRAU



FONTE: SILVA *et al.* (2010, p. 36).

Outra adaptação da função degrau chamada função degrau bipolar se dá pela Equação (4)

$$\varphi(v) = \begin{cases} 1, & se\ v > 0 \\ 0, & se\ v = 0 \\ -1, & se\ v < 0. \end{cases} \qquad (4)$$

Uma aproximação da função degrau bipolar muito utilizada em problemas de classificação é dada por:

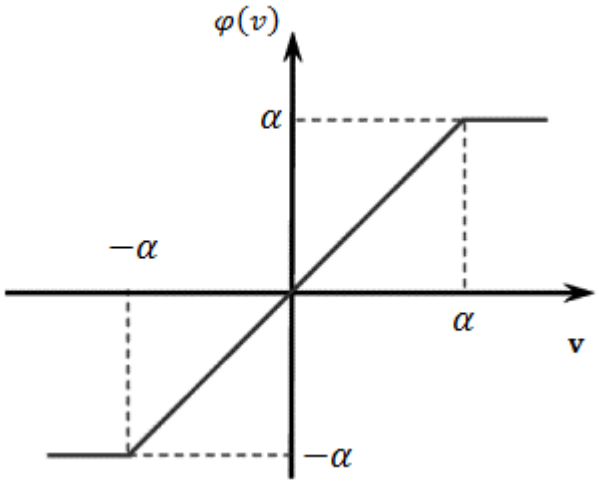
$$\varphi(v) = \begin{cases} 1, & se\ v \geq 0 \\ -1, & se\ v < 0. \end{cases} \quad (5)$$

- Função rampa simétrica: Também não diferenciável, a função rampa simétrica estabelece valores constantes mínimo e máximo para as saídas, conforme intervalo $[-\alpha,\ \alpha]$, conforme representação matemática abaixo:

$$\varphi(v) = \begin{cases} \alpha, & se\ v > \alpha \\ v, & se - \alpha \leq v \leq \alpha \\ -\alpha, & se\ v < -\alpha. \end{cases} \quad (6)$$

A representação gráfica da função rampa simétrica está ilustrada na Figura 5.

FIGURA 5 – FUNÇÃO RAMPA SIMÉTRICA



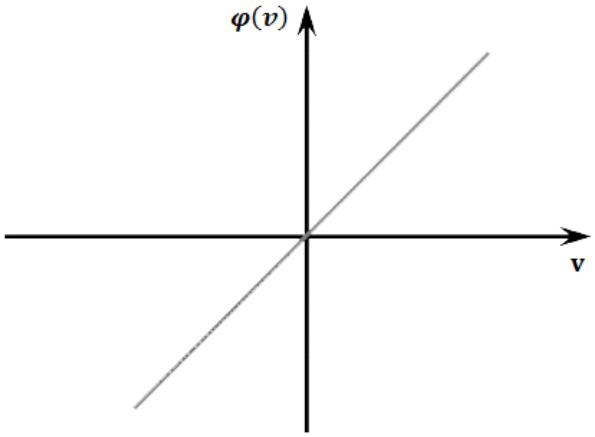
FONTE: SILVA *et al.* (2010, p. 38).

- Função linear: Totalmente diferenciável, a função linear é definida pela equação:

$$\varphi(v) = \alpha v, \quad (7)$$

onde α é um número real.

FIGURA 6 – FUNÇÃO LINEAR



FONTE: SILVA *et al.* (2010, p. 42).

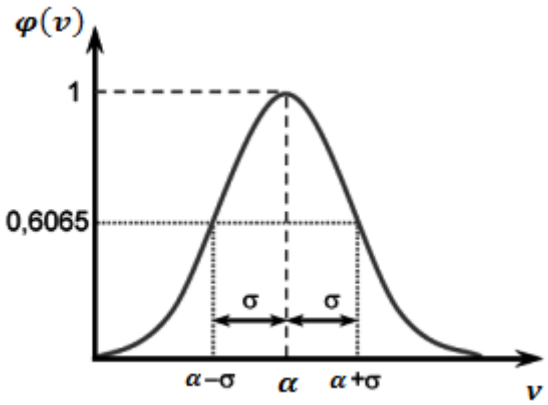
- Função gaussiana: A função gaussiana é definida por:

$$\varphi(v) = e^{-\frac{(v-\alpha)^2}{2\sigma^2}}, \qquad 8$$

tal que α define o centro da função ativação e σ o desvio padrão de v em relação a α, onde σ^2 determina a variância. Geralmente, $||v - \alpha||$ representa a distância euclidiana entre o potencial de ativação v e o centro da função gaussiana α. (SILVA *et al.*, 2010).

A Figura 7 ilustra a função gaussiana:

FIGURA 7 – FUNÇÃO GAUSSIANA



FONTE: SILVA *et al.* (2010, p. 41).

Segundo descrito em Silva *et al.* (2010), a função gaussiana é dita de base radial pois possui uma base de centro α, onde o raio da circunferência é denotado pelo desvio padrão.

2.5 ARQUITETURA

As Redes Neurais Artificiais (RNA's) podem ser caracterizadas como uma ligação de neurônios contidos ou não em camadas por meio de sinapses. A escolha da quantidade de camadas, pesos sinápticos e funções ativação pode alterar não só a sua estrutura, mas também sua funcionalidade.

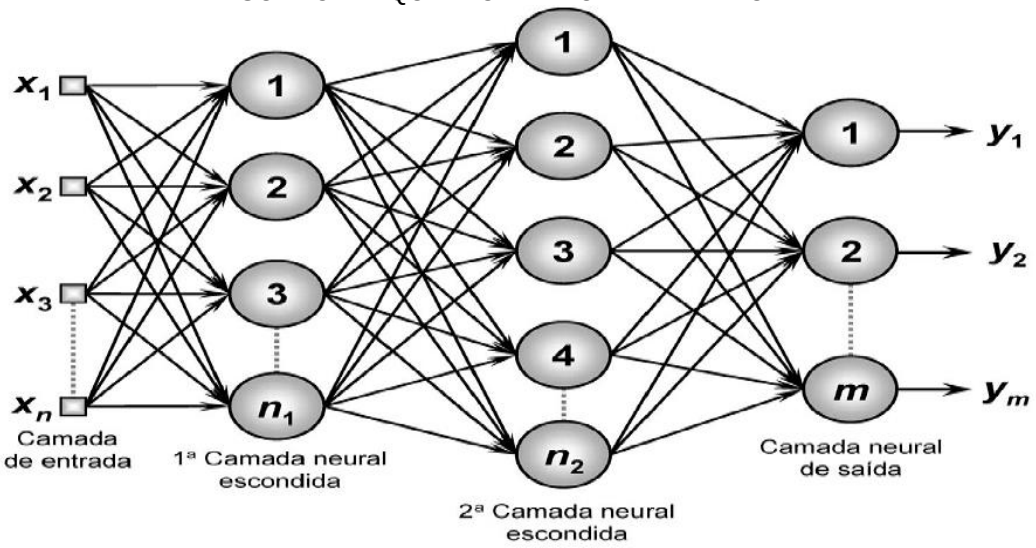
Segundo Silva *et al.* (2010, p. 45), "a arquitetura de uma rede neural artificial, define a forma como diversos neurônios estão arranjados, ou dispostos, uns em relação aos outros". As RNA's proporcionam a liberdade de se escolher a quantidade de neurônios, camadas e demais parâmetros que a compõem prezando a necessidade de cada problema.

Sobretudo, pode-se separar as redes neurais em três principais partes, denominadas camadas:

- Camada de Entrada: É responsável pelo recebimento dos dados externos bem como sua distribuição. Esta camada não possui neurônios e não realiza qualquer cálculo.
- Camadas Ocultas: Também chamada de camada escondida, pode ser composta por uma, nenhuma, ou várias camadas de neurônios responsáveis por extrair as características do processo. Basicamente, quase todo o trabalho interno da rede é realizado nestas camadas.
- Camada de Saída: Encarregada de receber os dados enviados pelas camadas anteriores e produzir e apresentar os sinais de saída da rede. Cada neurônio desta camada retorna um resultado padrão para o problema inicial.

A Figura 8 ilustra a arquitetura de uma rede neural formada pela camada de entrada, duas camadas escondidas e a camada de saída.

FIGURA 8 – ARQUITETURA DE UMA REDE NEURAL



FONTE: SILVA *et al.* (2010, p. 48).

2.6 APRENDIZADO

Um dos motivos pelos quais as redes neurais são tão atrativas deve-se a sua capacidade de aprendizado. Amostras padrões são inicialmente apresentadas a rede, de onde a mesma extrai informações relevantes para a identificação da relação entre entrada e saída. O conhecimento então adquirido é utilizado para retornar saídas aproximadas para quaisquer sinais de entradas expostos à rede. (SILVA *et al.*, 2010).

A obtenção de tais respostas generalizadas a serem produzidas pelas saídas da rede é adquirida por intermédio de um algoritmo de aprendizado. O algoritmo é um conjunto de passos ordenados para que uma RNA possa adquirir e guardar tal conhecimento realizando atualizações iterativas de parâmetros, conhecidos por pesos sinápticos e *bias*. (LUDWIG JR. e COSTA, 2007; SILVA *et al.*, 2010).

Geralmente, do conjunto amostral submetido à rede, cerca de 60 a 90% dos dados, escolhidos aleatoriamente, será utilizado para o processo de aprendizagem, denominado de subconjunto de treinamento. Já o subconjunto de teste será utilizado para a comprovação de que o conhecimento adquirido está dentro dos patamares esperados, tal subconjunto utilizará cerca de 10 a 40% da amostra total como base.

O processo de aprendizado pode ser caracterizado de duas formas: aprendizado supervisionado e aprendizado não-supervisionado.

2.6.1 Aprendizado Supervisionado

O aprendizado supervisionado consiste em apresentar à rede além de entradas, também suas saídas correspondentes. Sendo assim, a partir dessas informações as estruturas neurais encontraram um padrão entre as entradas e saídas.

A cada etapa do treinamento a rede possui sua saída calculada comparada com a saída indicada inicialmente, onde os pesos sinápticos e *bias* são ajustados gradativamente com a finalidade de minimizar o erro obtido com tal comparação de saídas (BRAGA *et al.*, 2000). O processo é repetido até que a taxa de erro esteja dentro de valores aceitáveis.

Segundo Braga *et al.* (2000), uma desvantagem do aprendizado supervisionado é que caso novas entradas sejam apresentadas sem suas respectivas saídas, a rede será apenas capaz de reproduzir as saídas conforme conhecimento já adquirido, não conseguindo obter novos conhecimentos.

Basicamente, o aprendizado supervisionado é um caso de interferência indutiva externa, onde os parâmetros neurais são ajustados devido a se conhecer antecipadamente os dados desejados de saída. (SILVA *et al.*, 2010).

2.6.2 Aprendizado Não-supervisionado

Ao Contrário do aprendizado supervisionado, o aprendizado não-supervisionado não recebe as saídas desejadas. A rede recebe o conjunto de dados iniciais (apenas entradas) e trabalha em cima da divisão desses grupos em subgrupos. Basicamente, o aprendizado é feito por reconhecimento de padrões. (LUDWIG JR. e COSTA, 2007).

O conjunto neural se auto organiza para distinguir as particularidades dos dados iniciais, separando-os em classes conforme suas características em comum. Segundo Braga *et al.* (2000) tal processo só é possível caso haja redundância entre os dados, pois sem esse fator, seria impossível definir padrões entre as informações de entrada.

De acordo com Silva *et al.* (2010), a quantidade máxima de classes pode ser definida conforme necessidade de adaptação do problema a ser resolvido.

3 REDES NEURAIS UTILIZADAS

Neste capítulo serão apresentadas as duas redes neurais estudadas no trabalho: As Redes *Perceptron* Multicamadas (MLP) e as Redes de Funções de Bases Radiais (RBF). Para cada uma das redes, serão apresentados os métodos de treinamento, bem como os algoritmos. Por fim, é realizada uma comparação entre as duas redes neurais, sendo apontadas as principais diferenças entre elas.

3.1 REDES *PERCEPTRON* MULTICAMADAS

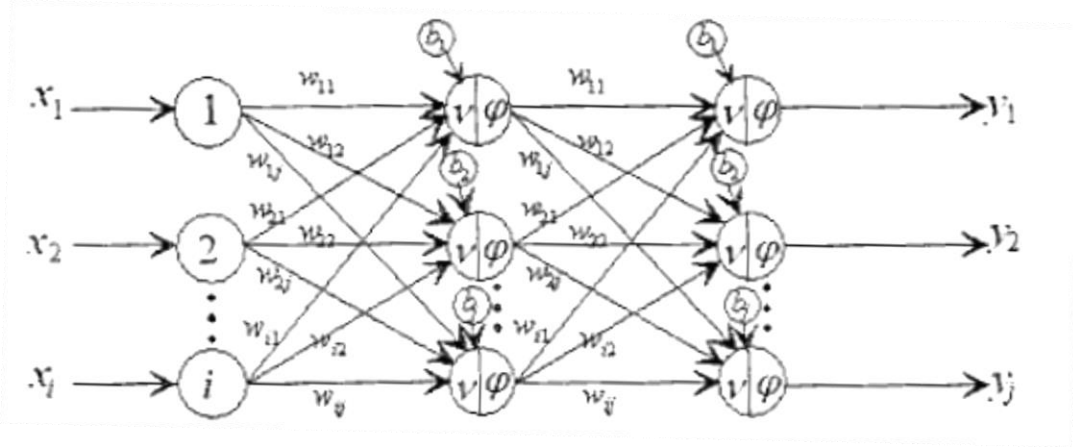
3.1.1 Introdução

As redes *perceptron* multicamadas ou *multilayer perceptron* (MLP) possuem pelo menos uma camada oculta em sua estrutura e, devido a este motivo, conseguem buscar soluções de problemas não linearmente separáveis. (SILVA *et al.*, 2010).

Um ponto forte das redes MPL é sua versatilidade em atender a diversos problemas, podendo ser empegada em reconhecimento de padrões, previsão de séries temporais, otimização de sistemas, identificação e controle de processos, dentre outras aplicações. (SILVA *et al.*, 2010).

Seu funcionamento se inicia na camada de entrada, passando para as camadas ocultas e finalizando na camada de saída, fazendo com que os dados de entrada sempre se propaguem para frente, de camada para camada, conforme observa-se na Figura 9. (LUDWIG JR. e COSTA, 2007).

FIGURA 9 - REDE *PERCEPTRON* MULTICAMADAS



FONTE: LUDWIG JR. e COSTA (2007, p. 46).

3.1.2 Topologia

A distribuição das camadas, bem como da quantidade de neurônios num sistema MLP varia muito e depende de vários fatores, como por exemplo, dos valores iniciais apresentados, da complexidade do problema, dos pesos sinápticos atribuídos inicialmente, dentre outros.

Segundo Cybenko[4] (1997 citado por Braga *et al.*, 2000, p. 53), o número de camadas ocultas necessárias pode ser considerada:

- Uma camada oculta é suficiente para aproximação de funções continuas;
- Duas camadas ocultas são suficientes para aproximar qualquer função matemática.

Em alguns casos, duas ou mais camadas podem facilitar no treinamento da rede, entretanto, a única camada que possui o valor exato do erro é a camada de saída, a camada anterior recebe apenas uma estimativa do valor, a anterior a esta recebe a estimativa da estimativa e assim sucessivamente. Logo, quando o número de camadas aumenta em excesso, pode-se perder demais a precisão do erro e o mesmo se torna menos útil. (BRAGA *et al.*, 2000).

Para a quantidade de neurônios em cada camada oculta, duas linhas podem ser seguidas: definir o número de neurônios em função do número de entradas e saídas, ou considerar a quantidade de neurônios dez vezes menor que o número de

[4] CYBENKO, G. Approximation by superpositions of a sigmoid funeration process. *Applied artificial Intelligence*, 11(1):33-57, 1997.

padrões, em que cada padrão possui um conjunto de entradas e saídas. Entretanto, outras possibilidades podem ser consideradas, já que essa escolha depende de vários fatores conforme já mencionado. (BRAGA *et al.*, 2000).

Também de acordo com Braga *et al.* (2000), deve ser levado em consideração o fato de que, caso sejam alocados neurônios em excesso, pode ocorrer *overfitting*, em que a rede aprende o padrão original de treinamento e não as características principais que lhe permitirão a generalização dos resultados. Todavia, neurônios em quantidades muito reduzidas podem causar *underfitting*, em que a rede demorará muito tempo para reconhecer os padrões ou não terá sucesso na convergência.

A quantidade de neurônios nas camadas de entrada e saída irá depender do número de entradas e saídas, respectivamente, em que a rede será submetida. Por exemplo, se a rede receber n valores de entrada, a mesma deverá possuir n neurônios na camada de entrada, o mesmo se aplica para a camada de saída. (SILVA *et al.*, 2010).

3.1.3 Treinamento

Dentre as várias maneiras de se treinar uma rede MLP, a mais usual é a denominada *back-propagation*. Tal algoritmo, por meio do aprendizado supervisionado, calcula o erro entre a saída apresentada inicialmente e a saída calculada para atualizar seus pesos sinápticos e *bias*. (BRAGA *et al.*, 2000).

Duas fases fazem parte deste processo de treinamento. A chamada *forward* é encarregada de apresentar as saídas calculadas pela rede, nesta fase, os pesos sinápticos e *bias* não sofrem alterações. Já a fase denominada *backward* é responsável, após o cálculo do erro, pelas atualizações dos pesos sinápticos e *bias*. A cada iteração, as duas fases são aplicadas, visando a redução do erro. (BRAGA *et al.*, 2000; SILVA *et al.*, 2010).

A função ativação para este processo deve ser contínua e diferenciável em qualquer ponto, logo, uma função que satisfaz esses requisitos é a função sigmoidal logística, conforme retratada em Braga *et al.* (2000) e Haykin (2001):

$$\varphi_j(v_j) = \frac{1}{1 + e^{-\alpha v_j}}, \quad 9$$

onde α é uma constante real que determina a suavidade da curva e v_j é a soma ponderada de todas as entradas acrescidas do *bias*, dado por

$$v_j(t) = \sum_{i=1}^{n} w_{ij}(t) \cdot y_j(t) + b_j(t), \quad 10$$

sendo t a iteração atual do treinamento, $w_{ij}(t)$ e $b_j(t)$ o peso sináptico do neurônio j associado a saída do neurônio i da camada anterior e o *bias* associado ao neurônio j respectivamente. A variável $y_j(t)$ é a entrada recebida pelo neurônio j na iteração t. (HAYKIN, 2001).

Quando o neurônio j está situado na primeira camada oculta, $y_j(t)$ assume o valor das entradas x_i apresentadas inicialmente à rede, logo

$$y_j(t) = x_i. \quad 11$$

Já quando o neurônio j está localizado a partir da segunda camada oculta ou está na camada de saída, $y_j(t)$ assume o valor resultante da função ativação, que será a saída da camada anterior:

$$y_j(t) = \varphi_i(v_i(t)), \quad 12$$

onde i representa o neurônio da camada anterior. (HAYKIN, 2001).

3.1.3.1 Cálculo do Erro

O erro quadrático total calculado na camada de saída para a iteração t é dado, segundo Silva *et al.* (2010), Haykin (2001) e Ludwig Jr. e Costa (2007), por

$$\varepsilon_{med}(t) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j=1}^{k} (d_j^n(t) - y_j^n(t))^2, \quad 13$$

tal que N é a quantidade de valores de entrada apresentado a rede, k é a quantidade de neurônios na camada de saída e consequentemente a quantidade de

saídas da rede, d_j a *j*-ésima saída desejada apresentada inicialmente e y_j é a *j*-ésima saída calculada pela rede.

Pode-se, para fins de simplificação, assumir $\varepsilon_{med}(t)$ por

$$\varepsilon_{med}(t) = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j=1}^{k}(e_j^n(t))^2, \quad 14$$

onde $e_j(t)$ é a diferença entre a saída desejada e a saída calculada pela rede, escrita da forma

$$e_j(t) = d_j - y_j(t), \quad 15$$

Em geral, $\varepsilon_{med}(t)$ representa o quanto as saídas calculadas para um determinado padrão de entrada difere da real resposta para o mesmo. Logo, o objetivo principal do método MLP é a minimização de $\varepsilon_{med}(t)$, atualizando a cada iteração seus dados livres (pesos sinápticos e *bias*). (BRAGA *et al.*, 2000).

3.1.3.2 Atualização dos Dados Livres

Segundo Braga *et al.* (2000), por mais que o erro total é definido pela formula de $\varepsilon_{med}(t)$ onde considera-se a média do erro de todas as saídas para todas as entradas iniciais, para fins de minimização, utiliza-se apenas o erro calculado com base nas saídas de um padrão de entrada (x_n, d_n) denotado por

$$\varepsilon(t) = \frac{1}{2}\sum_{j=1}^{k}(e_j(t))^2. \quad 16$$

Para a atualização dos pesos sinápticos ($w_{ij}(t)$), o algoritmo de retro-propagação faz uso de um fator correção ($\Delta w_{ij}(t)$) por meio da derivada parcial $\partial\varepsilon(t)/\partial w_{ij}(t)$. Tal derivada, segundo Haykin (2001, p. 189), “representa um fator sensibilidade, determinando a direção de busca no espaço pesos, para o peso sináptico w_{ij}”.

Deste modo, a correção para o peso sináptico w_{ij} é descrita como

$$\Delta w_{ij}(t) = \eta \delta_j(t) y_j(t), \quad (17)$$

onde η representa a taxa de aprendizagem do algoritmo de retro-propagação. (HAYKIN, 2001).

Uma modificação na Equação (17) é feita para a atualização dos *bias*. Como não se tem valores $y_j(t)$ associados, tal incógnita é igualada a um. Desta forma, define-se $\Delta b_j(t)$ por

$$\Delta b_j(t) = \eta \delta_j(t). \quad (18)$$

O gradiente local ($\delta_j(t)$) pode ser definido de duas formas diferentes, de acordo com camada em que o neurônio j esta situado.

3.1.3.2.1 Gradiente Local para a Camada de Saída

O gradiente local calculado na camada de saída é definido como

$$\delta_j(t) = e_j(t)\varphi_j'\left(v_j(t)\right). \quad (19)$$

Quando j pertence a camada de saída, o valor do erro $e_j(t)$ é calculado, possibilitando então, a determinação de $\delta_j(t)$. (HAYKIN, 2001).

3.1.3.2.2 Gradiente Local para as Camadas Ocultas

Caso j esteja localizado nas camadas ocultas, o valor do erro para estas camadas não é possível de calcular, pois não existe uma saída desejada para que se possa comparar. Logo, o sinal do erro para um neurônio oculto é feito recursivamente, utilizando-se os sinais dos erros das camadas anteriores. Tem-se então, de acordo com Haykin (2001), a seguinte definição para gradiente local:

$$\delta_j(t) = \varphi_j'\left(v_j(t)\right)\sum_k \delta_k(t) w_{jk}(t), \quad (20)$$

tal que j é o neurônio oculto atual e k o neurônio anterior a j.

3.1.3.3 Taxa de Aprendizagem

A taxa de aprendizagem (η), definida inicialmente na equação (17), informa o quão rápido uma rede irá identificar e gravar os padrões de entrada. Uma taxa de aprendizagem alta, resultará em grandes variações nos pesos sinápticos e *bias*, porém, tais modificações tornam a atualização muito oscilatória, podendo não obter padrões eficientes. Caso η seja um valor baixo demais, o aprendizado se tornará mais estável, porém em um processo mais lento. (HAYKIN, 2001).

Afim de se garantir a estabilidade e mesmo assim manter uma alta taxa de aprendizagem, um termo denominado momento é incluído na Equação (17), conforme descrito em Haykin (2001) e Silva *et al.* (2010):

$$\Delta w_{ij}(t) = \eta \delta_j(t) y_j(t) + \alpha \Delta w_{ij}(t'), \qquad 21$$

onde $\Delta w_{ij}(t')$ representa a variação dos pesos sinápticos anterior ao atual e α sendo a constante de momento. O parâmetro momento tem por objetivo ponderar a alteração dos valores livres entre a iteração anterior e a atual. (SILVA *et al.*, 2010).

3.1.3.4 Modos de Treinamento

Os valores de entrada são apresentados em N padrões na forma $((x_1, d_1);\ (x_2, d_2); \ldots;\ (x_N, d_N))$. O treinamento então pode ser realizado de duas maneiras: sequencial ou por lote, sendo que é considerado uma iteração do processo quando a apresentação completa de tal conjunto é executada. (HAYKIN, 2001).

No modo sequencial, para cada subconjunto apresentado à rede, os pesos sinápticos e *bias* são atualizados, realizando assim as fases *forward* e *backward* N vezes a cada iteração t. (HAYKIN, 2001).

Já no modo por lote, os pesos sinápticos e *bias* são atualizados após a apresentação de todo o conjunto de uma única vez, portanto, em uma iteração, as fases *forward* e *backward* são realizadas apenas uma vez. (HAYKIN, 2001).

Segundo Haykin (2001), o aprendizado sequencial é preferível em relação ao por lote, porque é computacionalmente mais rápido e armazena menos informação para cada neurônio.

Considerando o uso do modo sequencial, a apresentação aleatória de cada subconjunto em cada iteração também é muito relevante para o treinamento, pois proporciona mais dinamismo para o aprendizado além de evitar o estacionamento do algoritmo *back-propagation* e um mínimo local. (HAYKIN, 2001).

3.1.3.5 Critérios de Parada

O objetivo do algoritmo *back-propagation* é a minimização de $\epsilon_{med}(t)$. Um critério de parada adotado, é quando o erro quadrático médio for suficientemente pequeno, tendo então uma aproximação considerada aceitável entre as saídas desejadas e as calculadas pela rede. O processo de treinamento para quando $\epsilon_{med}(t) < \theta$, onde θ é um valor suficientemente pequeno. (BRAGA *et al.*, 2000).

Outros critérios de parada também considerados são: encerrar o processo de treinamento após t iterações, quando se atingir um percentual aceitável de saídas consideradas satisfatórias ou também de acordo com o tempo de processamento do algoritmo. Conforme também descrito por Braga *et al.* (2010), a combinação desses três critérios, também pode ser considerada.

3.1.4 O Algoritmo

Obter um conjunto de entradas (x^N, d^N);

Iniciar os pesos sinápticos, *bias*, taxa de aprendizado (η), constante de momento (α) e precisão θ;

Enquanto critério de parada não é satisfeito, fazer:

Para cada subconjunto de entrada, fazer:

Passo *forward*

Obter y_j e φ_j para cada camada da rede;

Calcular ε_j;

Passo *backward*

Determinar δ_j, Δw_{ij} e Δb_j;

Atualizar w_{ij} e b_j;

Calcular ε_{med}.

Fim

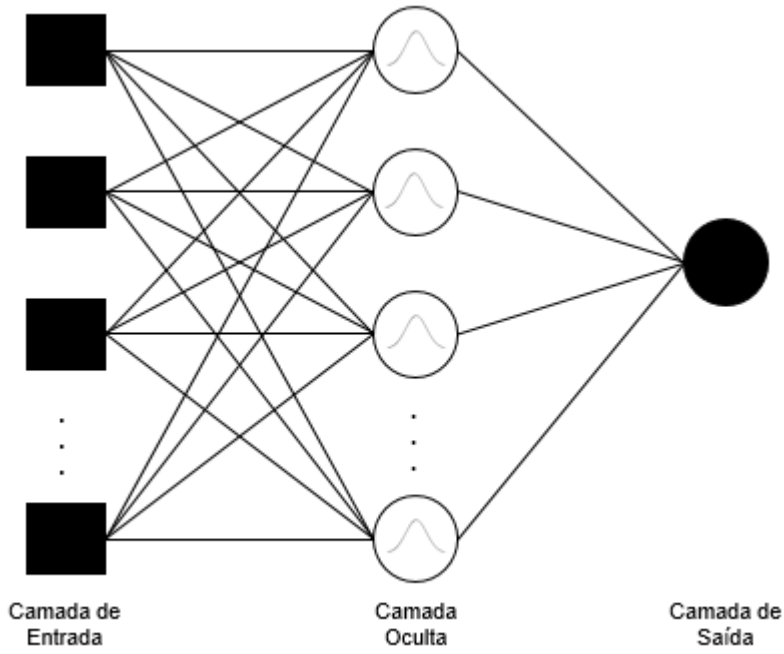
3.2 REDES DE FUNÇÕES DE BASES RADIAIS

3.2.1 Introdução

Ao contrário das redes MLP, onde funções são calculadas considerando o produto interno entre os pesos sinápticos e as entradas, as redes de bases radiais, também conhecidas como RBF (*Radial Basis Function*), fazem uso de um ajuste de curvas (aproximação) em um espaço de alta dimensão. (HAYKIN, 2001).

Semelhante as redes MPL, as redes RBF são constituídas por basicamente três camadas principais, sendo elas a camada de entrada, camada oculta e camada de saída, conforme Figura 10, onde cada camada executa tarefas específicas fundamentais para o funcionamento da rede. (BRAGA *et al.*, 2000).

FIGURA 10 - REDE DE FUNÇÕES DE BASE RADIAL



FONTE: ADAPTADO DE BRAGA *et al.* (2000, p. 194)

É usual as redes RBF conterem apenas uma camada oculta, mesmo que redes com mais de uma camada tenham sido estudas segundo Xiangdong[5] (1991, citado por Braga *et al.*, 2000, p. 194). A camada de saída, representada na Figura 10, pode conter mais de um neurônio, dependendo da necessidade da rede. (BRAGA *et al.*, 2000).

O nome da rede se dá pela utilização de funções de bases radiais, como por exemplo a função gaussiana (Equação (8)). Tal função é aplicada na distância entre os vetores de entrada e seus pontos centrais para a camada oculta e na ponderação dos dados internos para a camada de saída. (BRAGA *et al.*, 2000).

[5] XIANGDONG, H.; LAPEDES, A. Nonlinear Modelling and Prediction by Successive Approximation Using Radial Basis Function. Technical report, Los Alamos National Laboratory, 1991.

3.2.2 Treinamento

Dos vários métodos utilizados para treinar as redes RBF, os mais comuns são os híbridos, pois são realizados em duas fases. A primeira consiste em um modelo não-supervisionado, classificando os dados de entrada em subgrupos de acordo com suas principais características. (SILVA *et al.*, 2010; BRAGA *et al.*, 2000).

A segunda fase é responsável por atualizar os pesos sinápticos da camada de saída mediante modelos lineares similares aos utilizados para o treinamento da última camada das redes MLP. (BRAGA *et al.*, 2000).

O treinamento das redes de funções de bases radiais é feito apenas considerando o processo *forward*, ou seja, uma vez que o processo de treinamento é finalizado na camada oculta e os valores y_j são obtidos, os dados não são atualizados novamente, passa-se apenas para a camada de saída e o processo é então finalizado. (SILVA *et al.*, 2010).

3.2.2.1 Primeira Fase de Treinamento

Na primeira fase de treinamento, os neurônios da camada oculta representam os padrões de treinamento em que os parâmetros de entrada serão submetidos. Para tal, é utilizado o método K-médias. (BRAGA *et al.*, 2000).

O algoritmo K-médias, seleciona as K primeiras entradas como sendo seus pontos centrais (w_j), por consequência, a camada oculta terá K neurônios. A quantidade de pontos centrais deverá ser menor ou igual a quantidade de entradas iniciais propostas, entretanto, caso sejam mapeados todos os sinais de entrada, pode ocorrer *overfitting*. (BRAGA *et al.*, 2000).

A distância euclidiana entre x_k e w_j é utilizada para separar as N entradas entre os K padrões da rede, onde $k = 1,2,\dots,N$ e $j = 1,2,\dots,K$, para todo $K \leq N$. Cada conjunto Ω_j é formado pelos m pontos mais próximos ao centro w_j. (SILVA *et al.*, 2010). A distância euclidiana é então calculada conforme (22):

$$d_{jk} = \sqrt{(x_{k1} - w_{j1})^2 + (x_{k2} - w_{j2})^2 + \cdots + (x_{kn} - w_{jn})^2} \quad 22$$

onde cada N entrada é representada da forma $(x_1, x_2, \dots, x_n)$.

A cada iteração t, todos os N parâmetros de entradas são classificados entre os K grupos Ω_j. Os centros w_j são atualizados com base em cada grupo, visando a minimização da distância euclidiana de cada grupo. Desta forma, Silva *et al.* (2010) apresenta a equação (23):

$$w_{ji}(t) = \frac{1}{m_j(t)} \sum_{x_i \in \Omega_j(t)} x_i, \quad 23$$

onde $m_j(t)$ é a quantidade de pontos alocados no grupo $\Omega_j(t)$ para a iteração t.

De acordo com Silva *et al.* (2010), o processo de atualização dos centros w_j é feita enquanto houver modificação nos pontos de cada grupo Ω_j.

As saídas da camada oculta são retornadas pela função de ativação gaussiana, sendo essencial para seu cálculo o valor da variância. A variância de cada grupo para com seu respectivo centro é realizada pelo critério da distância quadrática média, conforme equação (24), descrita em Silva *et al.* (2010):

$$\sigma_j{}^2 = \frac{1}{m_j} \sum_{x_k \in \Omega_j} \sum_{i=1}^{n} (x_{ki} - w_{ji})^2 \text{, onde } j = 1,2,\dots,K. \quad 24$$

Quando então as entradas iniciais são aplicadas na função de ativação gaussiana para cada neurônio, obtêm-se as entradas para a última camada da rede RBF, a camada de saída. (SILVA *et al.*, 2010).

3.2.2.2 Segunda Fase de Treinamento

O processo supervisionado é considerado para a segunda fase do treinamento das redes RBF, onde para cada entrada x_k, um valor correspondente d_k é apresentado como sendo a saída desejada.

O segundo processo de treinamento é semelhante ao apresentado para a camada de saída das redes MLP, com a diferença que a atualização é feita apenas considerando os pesos sinápticos e *bias* entre a camada oculta e camada de saída, não realizando a retro-propagação do erro para as camadas anteriores. (SILVA *et al.*, 2010).

Mais precisamente, a segunda fase do treinamento das redes RBF é uma rede *perceptron* de camada única, sendo que a limitação para resolução de problemas não-lineares é sanada com a aplicação da primeira fase do treinamento.

Conforme descreve Silva *et al.* (2010), para cada neurônio da única camada oculta da estrutura neural, pesos sinápticos u_j são atribuídos ligando a camada oculta à camada de saída. Tais pesos são utilizados para ponderar as entradas da camada de saída.

Após a conclusão da primeira fase de treinamento, uma resposta de tamanho N, retornada pela camada oculta, é obtida apresentando-se cada entrada inicial x_k para a função gaussiana:

$$y_{kj} = \varphi_j(x_k - w_j) = \exp\left(-\frac{\sum_{i=1}^{n}(x_{ki} - w_{ji})^2}{2{\sigma_j}^2}\right), \qquad 25$$

em que w_j e ${\sigma_j}^2$ são, respectivamente, o centro e a variância associados ao neurônio j. Cada resposta então é composta por pontos (y_k, d_k), onde d_k é a saída desejada para o parâmetro de entrada inicial x_k. O ponto y_k será da forma $(\varphi_1(x_k - w_1), \varphi_2(x_k - w_2), \dots, \varphi_K(x_k - w_K))$, na qual $k = 1, 2, \dots, N$. (SILVA *et al.*, 2010).

Assim sendo, as respostas são recebidas pela camada oculta e ponderadas, a cada iteração, de acordo com os pesos sinápticos e *bias* associados a cada neurônio da camada de saída:

$$v_i(t') = \sum_{j=1}^{K} u_{ij}(t') \cdot y_j + b_i(t'), \quad 26$$

onde $u_{ij}(t')$ é o peso sináptico que liga o neurônio j da camada oculta ao neurônio i da camada de saída e $b_i(t')$ o *bias* associado ao neurônio i da camada de saída, ambos na iteração t', conforme descrito em Silva *et al.* (2010). A Equação (26) é comparada com a saída desejada, resultando no sinal de erro:

$$e_{ki}(t') = d_{ki} - v_i(t'). \quad 27$$

De acordo com Haykin (2001), o sinal erro é utilizado no controle dos pesos sinápticos e *bias* do modelo neural. Utiliza-se de métodos de otimização irrestrita com a finalidade de minimizar o erro $\varepsilon(t)$ pelo método da descida mais íngreme. O método realiza ajustes nos pesos sinápticos e *bias* em uma direção oposta ao vetor gradiente $\nabla\varepsilon(t')$. Desta forma, a correção é realizada por:

$$\Delta u_{ij}(t') = -\eta y_{kj}(t') e_{ki}(t'), \quad 28$$

tal que η é a taxa de aprendizado. Para o *bias*, utiliza-se da mesma equação, porém, como não se tem valores y_{kj} associados, tal incógnita é substituída por um.

Uma iteração é concluída após todo o conjunto y_N ser apresentado à rede, preferencialmente de modo sequencial, devido ao fato de ser computacionalmente mais rápido, além de ocupar menos memória.

Assim como para o *perceptron* de multicamadas, depois do término de cada iteração, o erro quadrático médio $(\varepsilon_{med}(t'))$ é calculado baseado na diferença entre a saída calculada pela rede e a saída desejada apresentada inicialmente. Uma condição de parada é quando não se tenha mudanças expressivas da rede comparado-se o erro atual com o erro da iteração anterior, ou seja, $\epsilon_{med}(t'-1) - \epsilon_{med}(t') < \theta$, onde θ é um valor suficientemente pequeno. Outros critérios também podem ser considerados conforme apresentado na Seção 3.1.3.5.

Para a resposta final da rede, funções de ativação devem ser consideradas, sejam elas função degrau, sigmoidal logística, linear, dentre outras.

3.2.3 O Algoritmo

Obter um conjunto de entradas (x^N, d^N);

Iniciar os centros, pesos sinápticos, *bias*, taxa de aprendizado (η) e precisão θ;

Primeira fase

Enquanto critério de parada não é satisfeito, fazer:

Para cada subconjunto de entrada, fazer:

Calcular a distância euclidiana entre x_k e w_j, para todo $k = 1, 2, \dots, N$ e $j = 1,2, \dots, K$;

Atribuir cada x_k para um grupo Ω_j;

Ajustar w_j, para todo $j = 1,2, \dots, K$;

Calcular ${\sigma_j}^2$, para todo $j = 1,2, \dots, K$;

Obter os conjuntos $y_{kj} = (\varphi_1(x_k - w_1), \varphi_2(x_k - w_2), \dots, \varphi_K(x_k - w_K)),\ j = 1,2, \dots, K$;

Segunda fase

Enquanto critério de parada não é satisfeito, fazer:

Para cada subconjunto de entrada, fazer:

Obter v_j para cada neurônio da camada de saída;

Calcular ε_j;

Determinar Δu_{ij} e Δb_j;

Atualizar u_{ij} e b_j;

Calcular ε_{med}.

Fim.

3.3 COMPARAÇÃO ENTRE MLP E RBF

Tanto as redes *perceptron* de multicamadas (MLP) quanto as redes de funções de bases radiais (RBF) são consideradas aproximadoras universais. Por mais que os resultados obtidos das redes sejam similares, nota-se certas diferenças entre uma rede e outra.

O fato dessas redes serem classificadas como multicamadas se refere a terem, além da camada de entrada e saída, uma estrutura neural oculta. Isso as difere do *perceptron* por exemplo, tornando-as capazes de resolver problemas não linearmente separáveis. Entretanto, a camada oculta das redes MLP pode ser composta por várias subcamadas de neurônios, enquanto que as redes RBF estão limitadas geralmente a apenas uma camada de neurônios na camada oculta. (HAYKIN, 2001; BRAGA *et al.*, 2000).

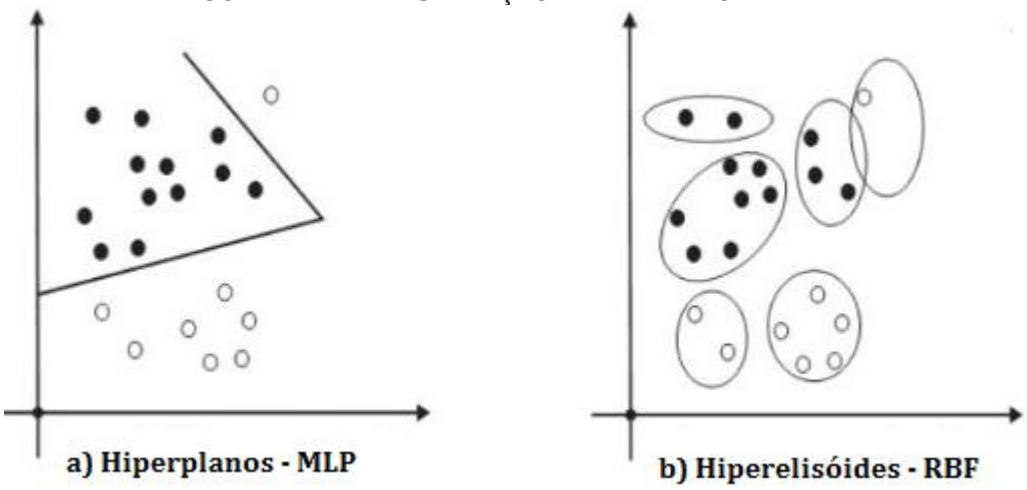
O modelo neural utilizado para treinar as redes *perceptron* de multicamadas é o mesmo para todas as camadas do sistema neural, sendo que geralmente são camadas não-lineares. Já para as redes com funções de bases radiais, tal processo difere de uma camada para outra, em que a camada oculta é não-linear e a camada de saída é linear. (HAYKIN, 2001).

A linearidade da camada de saída das redes RBF mostra uma relação da rede com o modelo neural *perceptron*, na realidade, a semelhança da RBF com o *perceptron* é muito maior do que com a MLP. Ao contrário do *perceptron*, incapaz de resolver problemas não-lineares, a RBF faz uso de funções de bases radiais (por exemplo a função gaussiana) na camada oculta, possibilitando o retorno de respostas aproximadas pela camada de saída linear. (HAYKIN, 2001).

O mapeamento do modelo MLP é feito por meio de hiperplanos, particionando o espaço de entradas em retas contínuas, enquanto que o modelo

RBF divide os parâmetros de entrada por meio de hiperelipsóides conforme Figura 11.

FIGURA 11 – REPRESENTAÇÃO MAPEAMENTO MLP E RBF



FONTE: SILVA *et al.* (2010, p. 177)

As redes MLP realizam uma aproximação global do problema inicial com maiores capacidades de generalização, enquanto que as redes RBF fazem apenas aproximações locais, pois o mapeamento é feito por meio de elipsoides baseados apenas nos dados de entrada. (BRAGA *et al.*, 2000).

Quanto mais dispersos forem os parâmetros de entrada, maior será a instabilidade dos retornos das redes. Para a MLP, dados muitos distantes podem ser classificados erroneamente, ao tempo que para a RBF, quanto mais distantes forem os dados dos centros das elipsoides, mais incertos serão os retornos. (BRAGA *et al.*, 2000).

4 APLICAÇÕES

As duas redes neurais apresentadas no trabalho (MLP e RBF) foram implementadas em *Visual Basic for Applications* (VBA) e os códigos são apresentados no Apêndice 1 e Apêndice 2. Adotou-se para ambas o número máximo de iterações em 2.000 e taxa de aprendizado (α) de 0,7 para a rede MLP.

A aplicações foram feitas em dois segmentos. O primeiro sendo em classificações de padrões e o segundo para previsão de séries temporais.

4.1 CLASSIFICAÇÃO DE PADRÕES

4.1.1 O Problema XOR

O problema XOR, também chamado de ou-exclusivo, é um operador lógico de base binária que pode ser considerado como um detector de diferenças entre dois operandos lógicos, conforme Tabela 1.

TABELA 1 – MODELO XOR

x_1	x_2	d
1	1	0
1	0	1
0	1	1
0	0	0

FONTE: O AUTOR (2016)

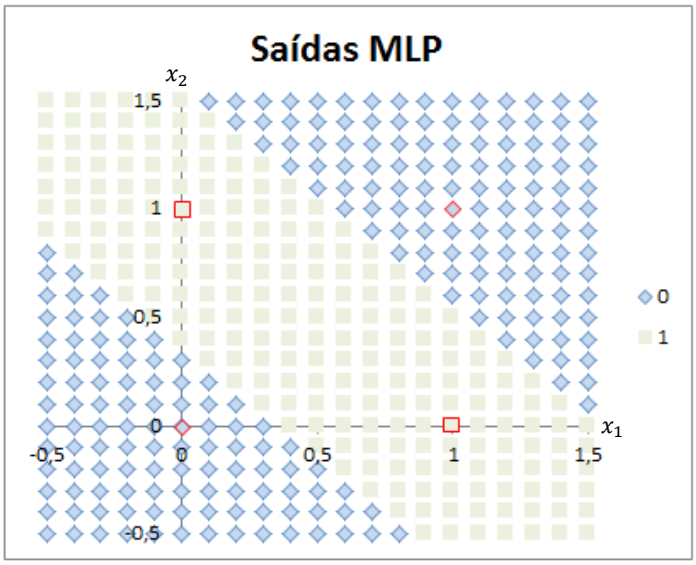
Conforme descrito em Haykin (2000), o mapeamento do problema XOR consiste em classificar pontos no hipercubo unitário. Quando as entradas x_1 e x_2 possuem o mesmo valor, a saída d é zero, caso contrário, é um. Redes de camada única não são capazes de realizar a classificação devido à falta de linearidade das entradas (x_1, x_2). Tal limitação não é encontrada nas redes com camadas ocultas, como é o caso da MLP e RBF.

Os testes foram realizados, a fim de padronização, com pesos sinápticos e *bias* inicializados randomicamente e precisão (θ) de $1\text{x}10^{-6}$, ambas as redes com 2 neurônios na camada oculta.

A convergência ocorreu para as duas redes antes de se alcançar o número máximo de iterações. Para fins de comparação, a rede MLP executou 609 iterações para alcançar a convergência, enquanto a RBF 161 iterações.

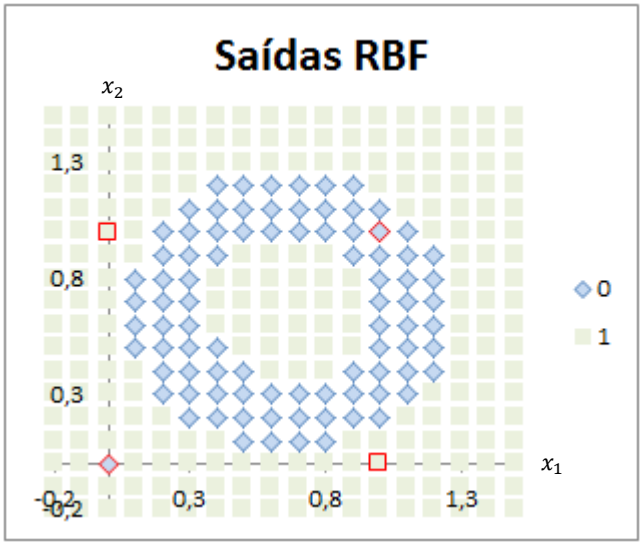
As fronteiras de limitações das classes mapeadas de ambas as redes são apresentada na Figura 12 e Figura 13.

FIGURA 12 - HIPERPLANOS REDE MLP



FONTE: O AUTOR (2016)

FIGURA 13 - HIPERESFERAS RBF



FONTE: O AUTOR (2016)

4.1.2 Classificação Musical por Gênero

A indústria musical nos dias atuais é de tamanho imenso, da mesma forma como o gosto musical dos usuários também o é. Classificar músicas de acordo com o gênero musical é um trabalho que, feito manualmente, demanda tempo e paciência tanto de quem realiza produções musicas quanto do usuário final. O problema consiste em uma classificação de padrões com o objetivo de indentificar o gênero do áudio por meio de características de um conjunto de dados. (JEREMIĆ, 2015).

O conjunto de dados possui 100 músicas distintas classificadas igualmente em 4 gêneros musicais: Rock, Clássico, Jazz e Música popular. Cada dado possui 8 características específicas, sendo: 1) a duração da música em segundos; 2) o tempo em batimentos por segundo (BPM); 3) a amplitude da média quadrática (RMS), ou seja, o poder contínuo ou musical que o amplificador pode fornecer; 4) a frequência de amostragem em kHz; 5) a taxa de amostragem (44,1 kHz ou 48 kHz); 6) a faixa dinâmica (DR ou DNR); 7) a tonalidade, podendo ser no formado C, C #, D, D #, E, F, F #, G, G #, A, Bb ou B, designados por números de 1 a 11; 8) número de erros digitais, que podem ser do tipo *glitches* ou *clipping*, neste caso, foi utilizado a soma dos dois erros obtida por meio do *software* Wavelab 7. (JEREMIĆ, 2015).

A rede terá então 8 neurônios na camada de entrada para absorver as 8 características de cada música e 4 neurônios na camada de saída, produzindo o padrão de cada gênero musical: Rock (1 0 0 0), Clássico (0 1 0 0), Jazz (0 0 1 0) e Música Popular (0 0 0 1).

Devido ao mau comportamento das redes MLP e RBF para conjuntos muito dispersos, o conjunto de dados passou por um processo de normalização, da forma:

$$x_n = \frac{x - x_{min}}{x_{max} - x_{min}} \qquad 29$$

onde x é o valor atual a ser nomalizado, x_{min} o menor valor e x_{max} o maior valor do conjunto amostral.

Escolhida aleatoriamente, 70% da amostra foi separada para o processo de treinamento. Os outros 30% do conjunto foram utilizados para o processo de teste. A taxa de aprendizado, para ambas as redes, foi variada entre 0,1 e 0,7 ($\eta \in [0{,}1, 0{,}7]$).

O treinamento foi realizado para três quantidades diferentes de neurônios na camada oculta (7, 15 e 34 neurônios). A Tabela 3 exibe os resultados obtidos para a rede MLP.

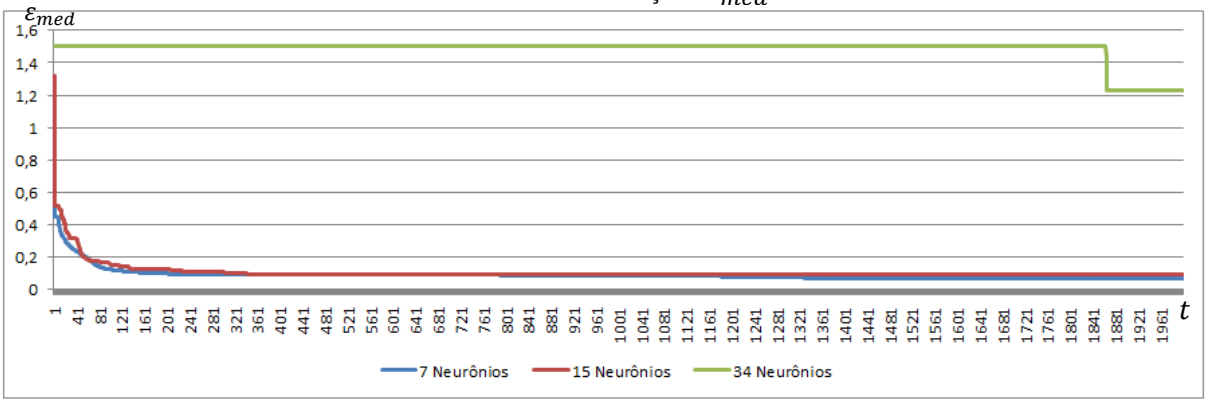
TABELA 3 - RESULTADOS PARA MLP

Quantidade de Neurônios	Quantidade de classificações Errôneas	Tempo de Processamento (s)	Erro Quadrático Médio (ε_med)
7	4	14	0,07
15	11	26	0,09
34	15	56	1,23

FONTE: O AUTOR (2016)

Nota-se que o melhor resultado é quando a quantidade de neurônios equivale a 7 unidades. Para o número de neurônios menor que 7, o algoritmo não apresentou melhor desempenho. A Figura 14 apresenta a evolução do algoritmo em minimizar ε_{med} a cada iteração.

FIGURA 14 – COMPARAÇÃO ε_{med} MLP



FONTE: O AUTOR (2016)

A tabela de resultados para a rede RBF, também considerando 3 quantidades diferentes de neurônios para camada oculta, é dada pela Tabela 4.

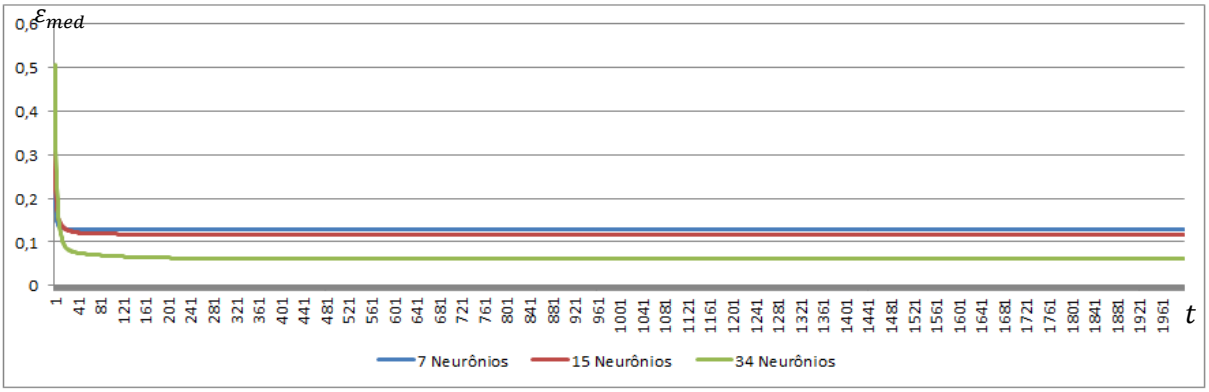
TABELA 4 – RESULTADOS PARA RBF

Quantidade de Neurônios	Quantidade de classificações Errôneas	Tempo de Processamento (s)	Erro Quadrático Médio (ε_med)
7	12	5	0,13
15	13	8	0,12
34	4	12	0,06

FONTE: O AUTOR (2016)

Neste caso, quando a quantidade de neurônios é muito baixa, o mapeamento da função gaussiana torna-se muito disperso, implicando em retornos menos satisfatórios. A comparação entre os erros de cada teste pelo número de iterações é dado pela Figura 15.

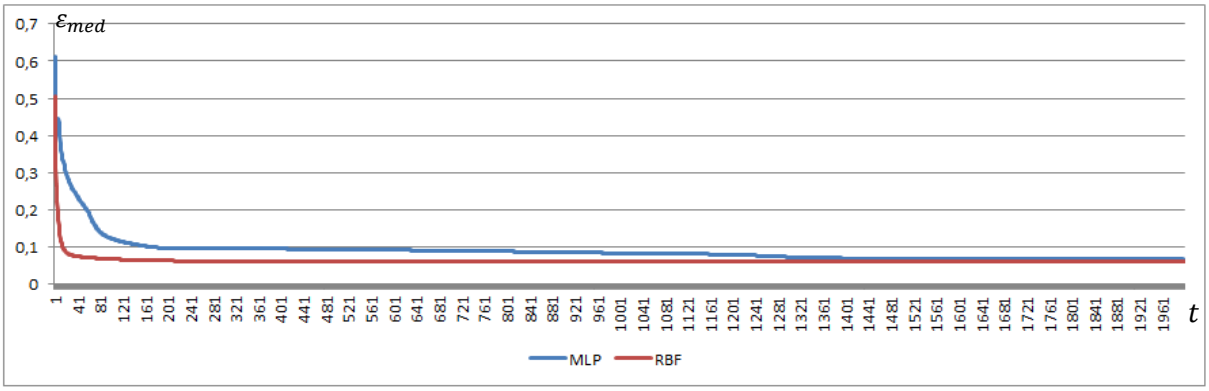
FIGURA 15 - COMPARAÇÃO ε_{med} RBF



FONTE: O AUTOR (2016)

Os melhores resultados se deram com 7 neurônios na camada oculta para a MLP, com $\eta = 0{,}7$, e com 34 para a RBF, com $\eta = 0{,}1$. A Figura 16 apresenta a comparação entre os dois melhores resultados obtidos. Percebe-se que o erro da rede RBF cai mais rapidamente do que a MLP, isso devido ao fato de que o algoritmo *back-propagation* converge mais lentamente.

FIGURA 16 – COMPARAÇÃO ε_{med} MLP E RBF



FONTE: O AUTOR (2016)

O teste é então realizado de acordo com o melhor treinamento de cada modelo neural. A rede é responsável por apresentar uma saída para o problema de acordo com o modelo neural já treinado, sem alterar nenhum peso sináptico ou *bias*. Desta forma, os resultados são apresentados pela Tabela 5.

TABELA 5 – RESULTADOS TESTE

Modelo Neural	Quantidade de Classificações Errôneas	Erro Quadrático Médio (ε_med)
MLP	12	0,21
RBF	20	0,18

FONTE: O AUTOR (2016)

Os principais erros são entre as classificações do gênero Rock e Música popular. A variância dos dados em geral é de 0,12, o que indica uma alta dispersão dos parâmetros de entrada.

4.2 PREVISÃO DE SÉRIES TEMPORAIS

4.2.1 Previsão do Volume de Vendas de Refrigerantes em Curitiba

As redes neurais também são empregadas em sistemas variantes no tempo. O que se deseja é o mapeamento das informações conforme o tempo decorre para, então, realizar previsões de novas informações.

O caso estudado em questão é em relação ao volume de vendas de refrigerantes em Curitiba. O problema consiste em mapear dados de vendas de um período de dois anos e prever possíveis volumes para os próximos meses. Os valores originais foram normalizados por meio da equação (29) com o objetivo de torná-los menos dispersos.

Foram utilizados um neurônio na camada de entrada representando o mês a ser mapeado e um para a camada de saída, ficando responsável pelo retorno do volume de vendas para o mês em questão. A quantidade de neurônios da camada oculta variou de uma rede para outra, sendo a quantidade máxima de neurônios igual à quantidade de amostras, ou seja, 31 neurônios. Os dados foram apresentados de forma sequencial, ajustando os pesos sinápticos e *bias* a cada dado apresentado. Taxa de momento $\alpha = 0{,}7$ e número máximo de iterações $t_{max} = 2000$ fixados para ambas as estruturas neurais.

O processo de treinamento da rede MLP recebeu as 31 entradas e convergiu antes de 2000 iterações. Devido a inclusão da taxa de momento, a rede MLP fez uso de uma taxa de aprendizado de 0,7, obtendo os resultados da Tabela 6.

TABELA 6 – RESULTADOS MLP

Quantidade de Neurônios	Quantidade Iterações	Erro Quadrático Médio (ε_med)
5	28	1,52E-02
12	79	9,18E-03
20	902	8,28E-03

FONTE: O AUTOR (2016)

O resultado com o menor erro é quando a rede foi treinada com 20 neurônios, porém necessitou de mais iterações, enquanto que quando treinada com 12 neurônios teve um erro apenas 10% maior porém convergiu mais rapidamente.

A Tabela 7 apresenta os resultados obtidos do treinamento da rede RBF considerando valor de 0,1 para a taxa de aprendizagem.

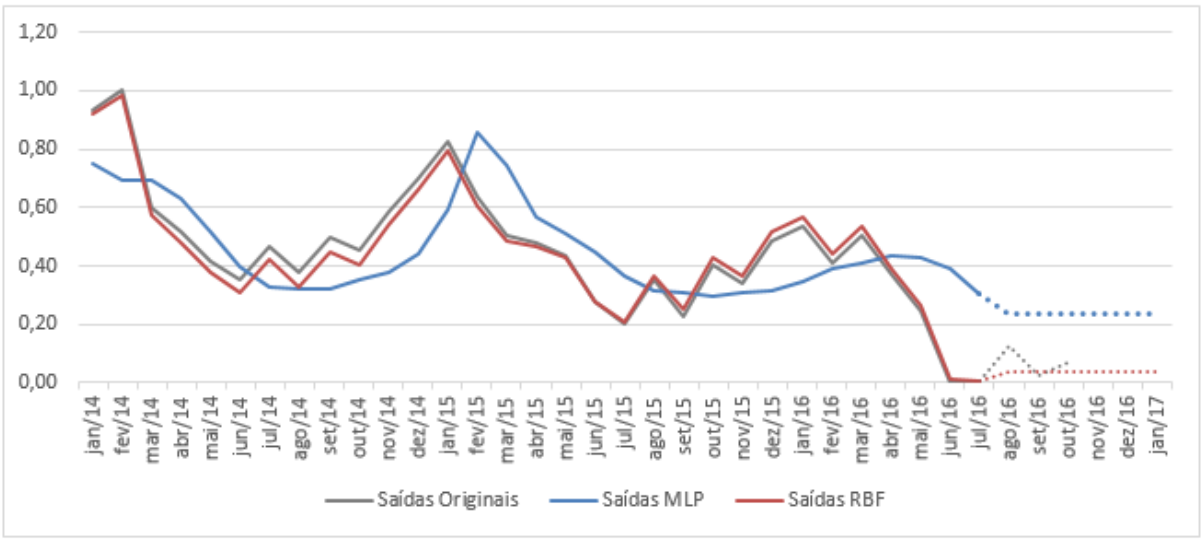
TABELA 7 – RESULTADOS RBF

Quantidade de Neurônios	Quantidade Iterações	Erro Quadrático Médio (ε_med)
5	95	3,14E-02
20	39	3,57E-03
34	69	3,17E-05

FONTE: O AUTOR (2016)

Todos os treinamentos apresentaram bons desempenhos. A Figura 17 apresenta os dados originais, bem como as saídas calculadas pelas rede MLP (com 12 neurônios) e a rede RBF (com 34 neurônios) incluindo também a previsão do volume de vendas para seis meses.

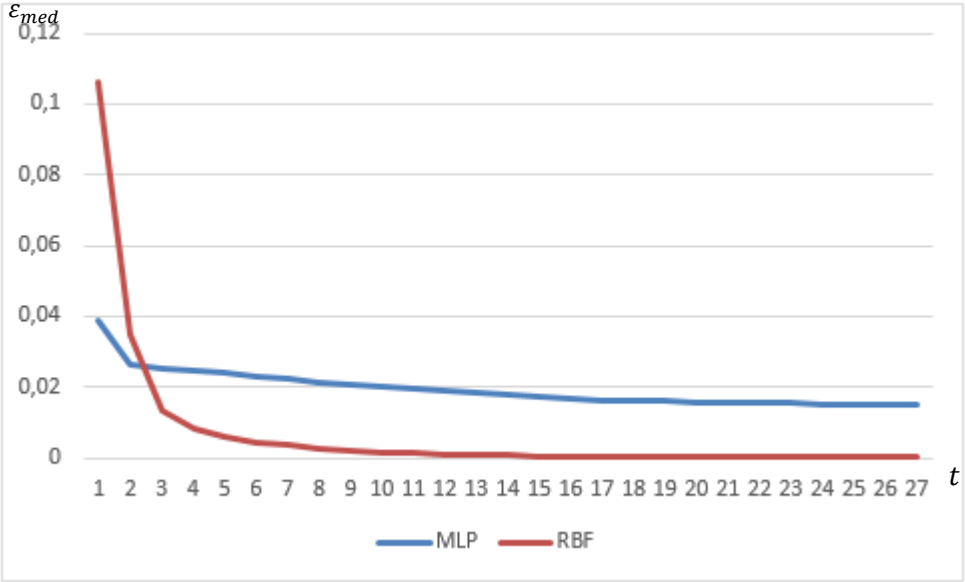
FIGURA 17 – COMPARAÇÃO SAÍDAS DESEJADAS COM SAÍDAS CALCULADAS



FONTE: O AUTOR (2016)

A Figura 18 apresenta a comparação de ε_{med} durante cada iteração entre as redes MLP e RBF.

FIGURA 18 – COMPARAÇÃO ε_{med} MLP E RBF



FONTE: O AUTOR (2016)

4.2.2 Previsão das Taxas Anuais de Crescimento do Ibovespa

Chama-se de taxa de crescimento a taxa média da variação do índice Bovespa composta anualmente entre dois períodos, ou seja, o quanto o preço das ações do Ibovespa cresceu ou decresceu durante determinado período. (IBOVESPA, 2016).

A base de dados contém as taxas de crescimento do Ibovespa desde o ano de 1987 até o ano de 2016, sendo este último atualizado até o mês de novembro de 2016. A taxa pode ser observada de períodos de um a vinte e nove anos. A Figura 19 apresenta parte da tabela extraída do site da Ibovespa.

FIGURA 19 – TAXAS PARCIAIS DE CRESCIMENTO APRESENTADA NO SITE DA IBOVESPA

Ano	1987	1988	1989	1990
1987				
1988	2.549,5			
1989	2.121,4	1.762,5		
1990	1.163,0	772,0	308,3	
1991	1.385,3	1.124,8	893,2	2.315,9

FONTE: IBOVESPA (2016)

Analisando a Figura 19, nota-se, por exemplo, que a taxa de crescimento do índice entre o final de 1987 e 1991 foi de 1.385%.

A tabela foi organizada de forma que as redes neurais pudessem ter dois neurônios na camada de entrada: o primeira sendo com o ano de início do período e o segunda com o ano final do período. O retorno da rede será a taxa de crescimento aproximada para o período desejado.

O treinamento foi feito com base nos anos de 1987 a 2014, deixando o ano de 2015 para teste e o ano de 2016 como uma nova previsão, devido ao fato de os dados de 2016 estarem atualizados até o mês de novembro.

Devido à grande dispersão dos valores das taxas de crescimento, os dados foram normalizados por meio da Equação (29), obtendo assim, valores dentro do conjunto $[0,1]$. Os anos também foram dispostos em números de 1 a 30, sendo 1 o ano de 1987 e 30 o ano de 2016 para uma melhor adaptação das redes.

Para os resultados de treino da rede MLP, foram utilizados três variações de tamanhos para a camada oculta, taxa de momento $\alpha = 0{,}5$ e taxa de aprendizagem $\eta = 0{,}7$. Para esta aplicação os resultados foram considerados satisfatórios quando a alteração do erro quadrático médio fosse menor que $1 \cdot 10^{-8}$, ou seja $\varepsilon_{med}(t-1) - \varepsilon_{med}(t) < 1 \cdot 10^{-8}$, t representando a iteração na qual o erro está sendo calculado.

A Tabela 8 apresenta os resultados obtidos com o treinamento da rede MLP.

TABELA 8 – RESULTADOS MLP

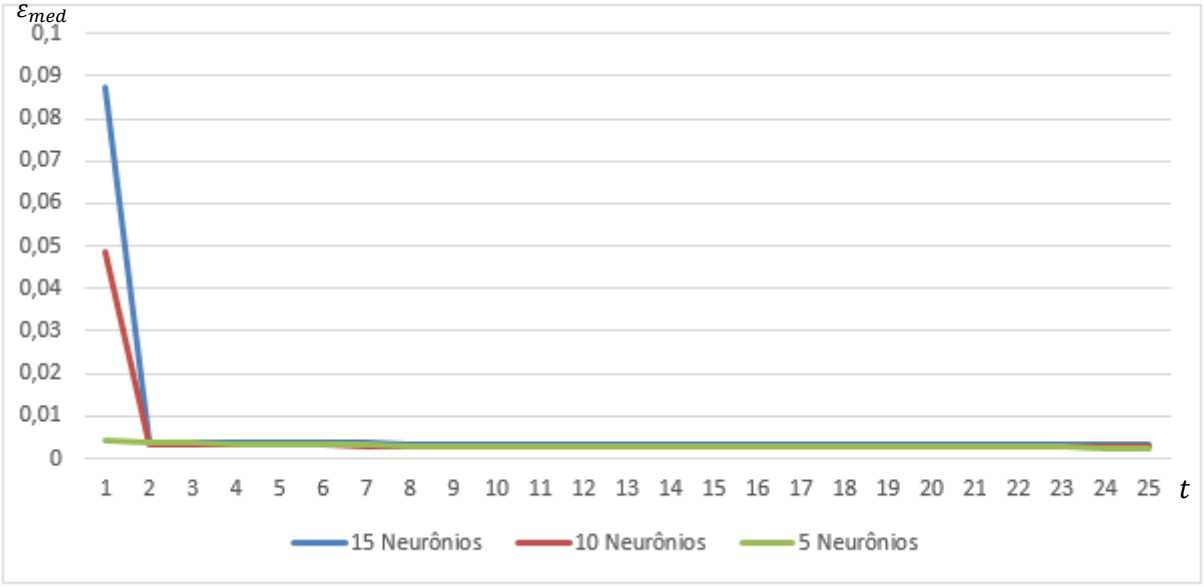
Quantidade de Neurônios	Número de Iterações	Erro Quadrático Médio ε_med
5	1570	9,19347E-06
10	1099	3,39209E-06
15	1793	3,00557E-06

FONTE: O AUTOR (2016)

Nota-se que por mais que o erro para a rede treinada com 15 neurônios tenha um menor erro quadrático médio, ela levou quase 700 iterações a mais para conseguir a convergência quando comparada com a rede treinada com 10 neurônios na camada oculta.

A comparação da evolução do processo de minimizar o erro quadrático médio no decorrer das iterações é apresentado para a rede MLP na Figura 20.

FIGURA 20 – COMPARAÇÃO ε_{med} MLP



FONTE: O AUTOR (2016)

A rede RBF foi treinada usando a taxa de aprendizagem $\eta = 0{,}7$ e a mesma metodologia para o cálculo de ε_{med}. Os resultados são apresentados na Tabela 9.

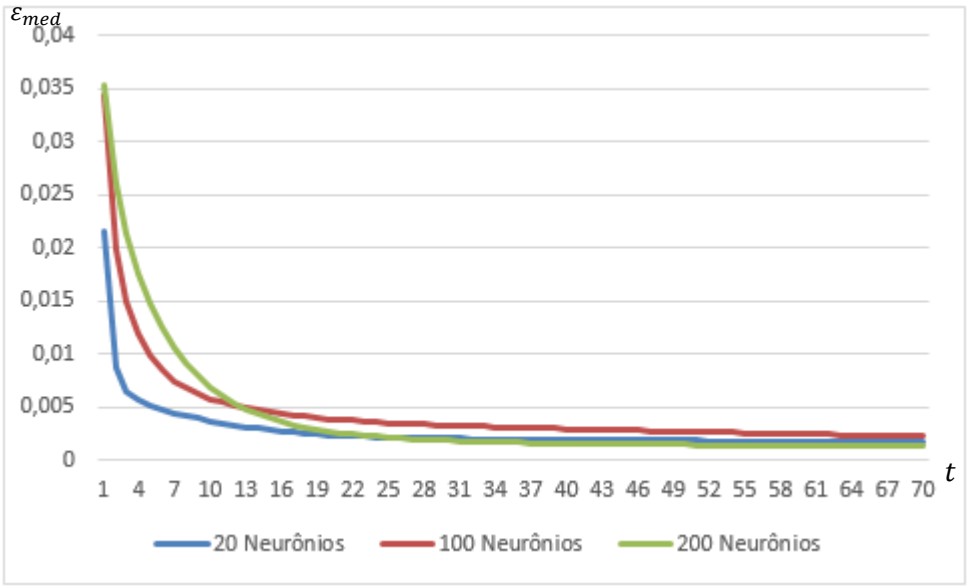
TABELA 9 – RESULTADOS RBF

Quantidade de Neurônios	Número de Iterações	Erro Quadrático Médio ε_med
20	122	1,83E-03
100	470	1,08E-03
200	1378	1,18E-04

FONTE: O AUTOR (2016)

Da mesma forma com para a rede MLP, o treinamento que possui o melhor valor para o erro quadrático médio também é o que precisou de mais iterações para convergir. A evolução da convergência da rede com relação a ε_{med} é apresentada na Figura 20.

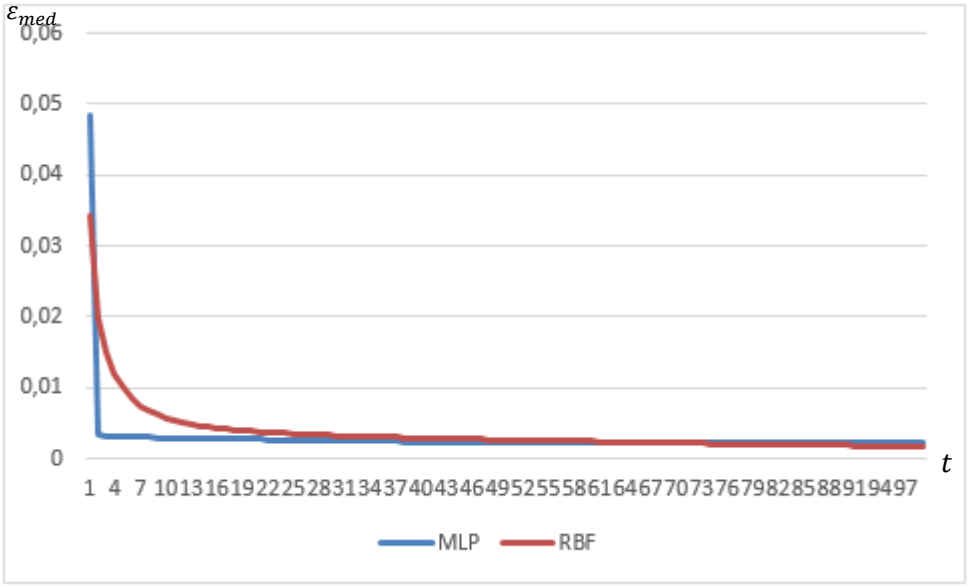
Figura 20 - COMPARAÇÃO ε_{med} RBF



FONTE: O AUTOR (2016)

Considerando a rede MLP treinada com 10 neurônios e a RBF treinada com 100, tem-se a seguinte comparação do erro na Figura 20.

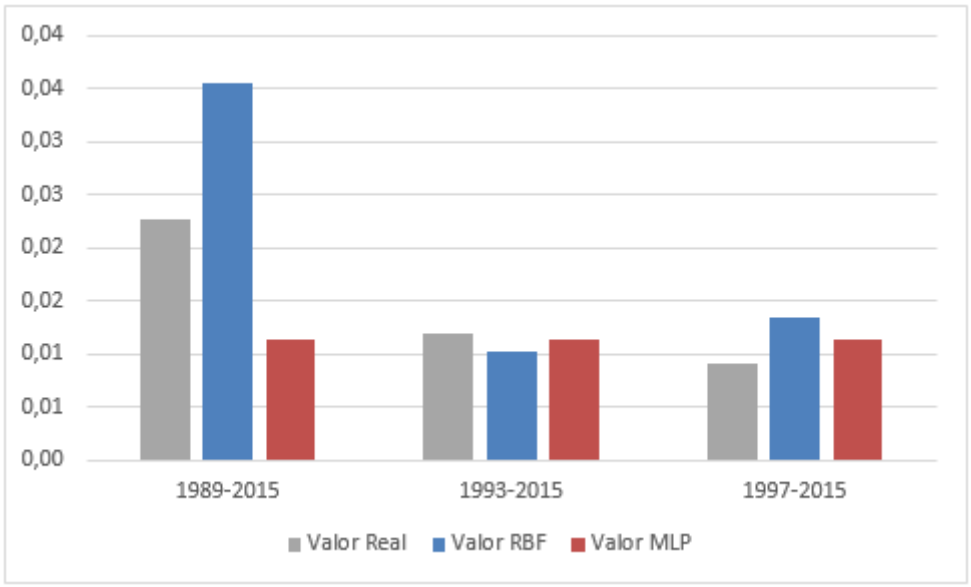
FIGURA 20 - COMPARAÇÃO ε_{med} MLP E RBF



FONTE: O AUTOR (2016)

Considerando as mesmas redes comparadas na Figura 20, o teste foi realizado obtendo um erro quadrático médio de $\varepsilon_{med} = 3{,}39\mathrm{E} - 06$ para a rede MLP, enquanto que para RBF, o erro foi de $\varepsilon_{med} = 5{,}98\mathrm{E} - 05$. Considerando três períodos aleatórios, observa-se a diferença entre o valor real e o valor de teste apresentado pela Figura 21.

FIGURA 21 – COMPARAÇÃO TESTE



FONTE: O AUTOR (2016).

Nota-se que há uma divergência mínima entre os valores, entretanto, as redes neurais trabalham como aproximadoras, logo, valores exatos serão muito raros. Chegou-se a valores satisfatórios tanto para os treinamentos quanto para os testes.

5 CONCLUSÕES

O estudo baseado em redes neurais abordou suas estruturas de forma geral aprofundando mais especificadamente em dois modelos principais, Redes *perceptron* multicamadas e redes de funções de bases radiais.

Além das características específicas de cada modelo, existem variáveis que podem mudar significativamente os retornos das redes, como por exemplo o número de neurônios da camada oculta, a taxa de aprendizado e o critério de parada.

A taxa de aprendizagem teve que ser relativamente baixa para a rede RBF, uma vez que valores altos desestabilizavam as redes, retornando saídas não condizentes com a realidade. Entretanto, para as redes MLP, a taxa de aprendizado foi considerada alta, sendo que a instabilidade da rede foi combatida com a taxa de momento.

Para as redes RBF, a quantidade de neurônios igual ou muito próximo à quantidade de amostras, implica em um perfeito mapeamento do problema inicial, porém, quando dados antes desconhecidos são apresentados, a rede não é capaz de retornar um valor condizente, devido ao fato do raio da função gaussiana ser muito pequeno, não abrangendo outras possibilidades de respostas. Quando se define uma quantidade de neurônios muito baixa em relação ao tamanho das amostras, o mapeamento torna-se muito genérico, fazendo com que respostas menos confiáveis sejam retornadas.

Caso uma quantidade muito alta de neurônios na camada oculta seja definida para as redes MLP, além do processo de treinamento se tornar muito longo, um tempo excessivo será gasto para se obter dados aceitáveis. Mesmo assim, a rede com um número excessivo de neurônios na camada oculta pode resultar em *overfitting*, onde a rede perde sua capacidade de aprender.

Em geral, ambas as redes possuem capacidades altas de aprendizado, a MLP no entanto, devido ao seu modelo de aprendizagem, acaba se tonando mais lenta que a RBF, ou seja, convergindo com menos intensidade.

REFERÊNCIAS

BRAGA *ET AL.*, A. P.; CARVALHO, A. C. P. L. F.; LUDERMIR, T. B. **Redes Neurais Artificiais** – Teoria e Aplicações. Rio de Janeiro: LTC, 2000.

HAYKIN, S. **Redes Neurais** – Princípios e Prática. Porto Alegre: Bookman, 2ª Edição, 2001.

IBOVESPA, Taxa de Crescimento. Disponível em: <http://www.bmfbovespa.com.br/pt_br/produtos/indices/indices-amplos/indice-ibovespa-ibovespa-estatisticas-historicas.htm>. Acesso em 10 de dez. 2016.

JEREMIĆ, M. **Music Classification by Genre Using Neural Networks** – An Example of a Multivariate Data Type Classification Problem Using Neuroph Framework. An Experiment for Intelligent System Couse - University of Belgrade, Serbia, 2015.

KOVÁCS, Z. L. **Redes Neurais Artificiais** – Fundamentos e Aplicações. São Paulo: Editora Livraria da Física, 2006.

LUDWIG, O.; COSTA, E. M. M. **Redes Neurais:** Fundamentos e Aplicações com Programas em C. Rio de Janeiro: Editora Ciencia Moderna, 2007.

REZENDE, S. O. **Sistemas inteligentes:** Fundamentos e aplicações. São Paulo: Manole, 2003.

SILVA *ET AL.*, I. N.; SPATTI, D. H.; FLAUZINO, R. A. **Redes Neurais Artificiais para Engenharia e Ciências Aplicadas** curso prático. Artliber, 2010.

APÊNDICE 1 – ALGORITMO MLP

```
For iter = 1 To iter_max 'Controle das iterações #####
   somaerro = 0
   For na = 1 To N 'na controla o ponto de treinamento #####

      'Forward #####

      'Calculo de yj para cada neurônio j da camada oculta #####
      For j = 1 To K
         v(j) = 0
         For i = 1 To Ni
            v(j) = v(j) + w(i, j) * entrada(na, i)
         Next
         v(j) = v(j) + b(j)
         y(1, j) = (1 / (1 + Exp(-(v(j)))))
      Next

      'Calculo de yj para cada neurônio j da camada de saída #####
      For js = Ni + 1 To Ns
         v(js) = 0
         For j = 1 To K
            v(js) = v(js) + w(js, j) * y(1, j)
         Next
         v(js) = v(js) + b(js)
         y(na + 1, js) = (1 / (1 + Exp(-(v(js)))))
      'Fim Forward #####

      'Calculo erro para o padrão na #####
         e(js) = entrada(na, js) - y(na + 1, js)
         somaerro = somaerro + (e(js)) ^ 2 'Somatório do erro para cada sinal de
saída #####
```

```
'Backward #######

    deltas(js) = e(js) * (Exp(v(js)) / ((Exp(v(js)) + 1) ^ 2)) 'Calcula o delta do neuronio da camada de saída #####

    deltab(js) = eta * deltas(js) + alfa * deltab(js) 'Calcula a correção do bias da camada de saída #####

    b(js) = b(js) + deltab(js) 'Atualiza o bias da camada de saída #####

    For j = 1 To K

      deltaw(js, j) = eta * y(1, j) * deltas(js) + alfa * deltaw(js, j) 'Calcula a correção dos pesos sinápticos entre a camada oculta e a camada de saída #####

      w(js, j) = w(js, j) + deltaw(js, j) 'Atualiza os pesos sinápticos entre a camada oculta e a camada de saída #####

    Next

  Next

  For j = 1 To K

    deltao = 0

    For js = Ni + 1 To 12

      deltao = deltao + deltas(js) * w(js, j) 'Ponderação dos deltas da camada oculta #####

    Next

    For i = 1 To Ni

      deltao = (Exp(v(j)) / ((Exp(v(j)) + 1) ^ 2)) * deltao 'Calcula o delta de cada neurônio da camada oculta #####

      deltaw(i, j) = eta * entrada(N, i) * deltao + alfa * deltaw(i, j) 'Calcula a correção dos pesos sinápticos entre a camada de entrada e a camada oculta #####

      w(i, j) = w(i, j) + deltaw(i, j) 'Atualiza os pesos sinápticos entre a camada de entrada e camada de saída #####

    Next

    deltab(j) = eta * deltao + alfa * deltab(j) 'Calcula a correção dos bias de cada neurônio da camada oculta #####

    b(j) = b(j) + deltab(j) 'Atualiza o bias de cada neurônio da camada oculta #####

  Next
```

```
        'Fim da fase backward #####

    Next
    e_med = (somaerro) / (2 * Nn) 'Ponderação do erro #####

    If e_min > e_med Then
        e_min = e_med
    End If

    'Se erro médio for menor que parâmetro de parada, altera para que iter fique na iteração máxima aceitável e pare o processo #####
    If e_med < teta Then
        iter = iter_max
    End If
Next
```

APÊNDICE 2 – ALGORITMO RBF

```
'Primeira fase #####
For iter = 1 To iter_max
   For j = 1 To Nj
      For i = 1 To Ni
         d(j, i) = 0 'Inicializa as distâncias #####
         omega(j, 0) = 0
         omega(j, i) = 0 'Inicializa os conjuntos #####
      Next
   Next
   somaw = 0
   For k = 1 To N
      For j = 1 To Nj
         somad = 0
         For i = 1 To Ni
            somad = somad + ((entrada(k, i) - w(j, i)) ^ 2)
         Next
         d(j, k) = (somad) ^ (1 / 2) 'Distância euclidiana do ponto k com o centro j #####
      Next
      min_d = d(Nj, k)
      jd = Nj
      For j = 1 To Nj - 1 'Verifica a menor distância entre o ponto k e os centros e aloca no grupo #####
         If min_d > d(j, k) Then
            min_d = d(j, k)
            jd = j
         End If
      Next
      omega(jd, 0) = omega(jd, 0) + 1 'Indica a próxima posição livre do conjunto #####
      omega(jd, omega(jd, 0)) = k
```

```
    Next


    For j = 1 To Nj
        For i = 1 To Ni
            somax = 0
            For m = 1 To omega(j, 0)
                somax = somax + entrada(omega(j, m), i) 'Soma xi para cálculo da média do ponto central #####
            Next
            want(j, i) = w(j, i)
            w(j, i) = somax / omega(j, 0) 'Media ponto central #####
            If want(j, i) = w(j, i) Then
                somaw = somaw + 1
            End If
        Next
    Next


    If somaw = Nj * Ni Then 'Se todos os pontos wji forem iguais aos anteriores, não teve modificação dos grupos, acaba iteração #####
        t = iter_max
    End If
Next


For j = 1 To Nj 'Calculo da variancia #####
    somax = 0
    For m = 1 To omega(j, 0)
        somad = 0
        For i = 1 To Ni
            somad = somad + (entrada(omega(j, m), i) - w(j, i)) ^ 2
        Next
        somax = somax + somad
    Next
    variancia(j) = somax / omega(j, 0)
```

```
Next
'Fim primeira fase #####

'Atualização das entradas para a segunda fase #####
For k = 1 To N
   For j = 1 To Nj
      somax = 0
      For i = 1 To Ni
         somax = somax + ((entrada(k, i) - w(j, i)) ^ 2)
      Next
      If variancia(j) = 0 Then
         variancia(j) = 1E-20
      End If
      y(k, j) = Exp(-(somax / (2 * variancia(j))))
   Next
Next
'Fim atualização #####

'Segunda Fase #####
For t = 1 To iter_max 'Controla as iterações #####
   somae = 0
   For k = 1 To N 'Controla o ponto de treinamento #####
      For js = Ni + 1 To 12
       v(k, js) = 0
       For j = 1 To Nj
          v(k, js) = v(k, js) + (u(js, j) * y(k, j)) 'Ponderação da saída da camada oculta
#####
       Next
       v(k, js) = v(k, js) + b(js) 'Saída calculada #####

      'Calculo erro para o padrão k #####
      e(js) = entrada(k, js) - v(k, js)
```

```
        'Atualização dos pesos e bias #####
        b(js) = b(js) + eta * e(js) 'Atualiza o bias da camada de saída #####

        For j = 1 To Nj
            deltau(js, j) = eta * y(k, j) * (e(js)) 'Calcula a correção dos pesos sinápticos entre a camada oculta e a camada de saída #####
            u(js, j) = u(js, j) + deltau(js, j) 'Atualiza os pesos sinápticos entre a camada oculta e a camada de saída #####
        Next
        'fim segunda fase #####

        somae = somae + (e(js)) ^ 2 'Somatório do erro para cada entrada #####
        Next
    Next
    e_med_ant = e_med
    e_med = (somae) / (2 * N) 'Ponderação do erro #####

    'Se a diferença entre o erro da iteração anterior e da iteração atual for menor parâmetro de parada, altera para que iter fique na iteração máxima aceitável e pare o processo #####
    If Abs(e_med - e_med_ant) < teta Then
        t = iter_max
    End If
Next
```