UNIVERSIDADE FEDERAL FLUMINENSE

VITOR GAMA LEMOS

REDES NEURAIS *PERCEPTRON A*PLICADAS AO RECONHECIMENTO DE PADRÕES

Niterói

2017

VITOR GAMA LEMOS

REDES NEURAIS *PERCEPTRON* APLICADAS AO RECONHECIMENTO DE PADRÕES

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Sistemas de Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Computação.

Orientador (a):
Julliany Sales Brandão

Niterói
2017

VITOR GAMA LEMOS

REDES NEURAIS *PERCEPTRON* APLICADAS AO RECONHECIMENTO DE PADRÕES

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Sistemas de Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Computação.

Niterói, ___ de _____ de 2017.

Banca Examinadora:

_____

Julliany Sales Brandão, Dsc. – Orientadora

CEFET-RJ – Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

_____

Pedro Henrique González Silva, Dsc. – Avaliador

CEFET-RJ – Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

AGRADECIMENTOS

A Deus, que sempre iluminou a minha caminhada.

A minha Orientadora Julliany Sales Brandão, pela sua dedicação e orientação na elaboração deste trabalho.

Aos Colegas de curso pelo incentivo e troca de experiências.

A todos os meus familiares e amigos pelo apoio e colaboração.

“*Each dream you leave behind is a part of your future that will no longer exist*.”
Steve Jobs

RESUMO

As redes neurais artificiais (RNA) são modelos matemáticos inspirados no cérebro humano, mais especificamente na capacidade de aprender, processar e realizar tarefas. As RNAs são ferramentas poderosos que auxiliam na resolução de problemas complexos ligados principalmente na área de otimização combinatória e aprendizagem de máquina. Neste sentido, as redes neurais possuem as mais variadas aplicações possíveis, pois tais modelos podem adaptar-se as situações apresentadas, garantindo um aumento gradativo de desempenho sem que ocorra qualquer interferência humana.

O presente trabalho tem o intuito de demonstrar a arquitetura e os processos envolvidos no funcionamento geral de uma rede neural, tendo como foco principal, os Algoritmos *Perceptron* e suas aplicações em meio aos problemas de reconhecimento de padrões. Primeiramente, descrevem-se conceitos básicos que relacionam estas redes com a estrutura neural biológica que formam o sistema nervoso. Além disso, detalham-se diversas características, especificações, estruturas relacionadas, diferentes aplicações e como ocorre o treinamento desses modelos.

O trabalho em questão propõe uma análise da capacidade das redes *Perceptron* em diversos aspectos. Desta forma, é proposto para este trabalho a implementação e teste de tais redes em uma aplicação real. Com isso, esta abordagem possibilitou demonstrar através de um problema real que estas redes são eficientes na obtenção de uma solução satisfatória e condizentes com os resultados encontrados.

Palavras-chaves: RNA, *Perceptron*, Aprendizado de máquina, AI.

ABSTRACT

The artificial neural networks (ANNs) are mathematical models inspired by the human brain, more specifically in the ability to learn, process, and perform tasks. ANNs are powerful tools that help solve complex problems mainly related to combinatorial optimization and machine learning. In this sense, the neural networks have the most varied possible applications, because such models can adapt to the situations presented, ensuring a gradual increase of performance without any human interference occurring.

The present work has the objective to demonstrate the architecture and processes involved in the general functioning of a neural network, having as main focus the Perceptron Algorithms and their applications amid the problems of pattern recognition. Firstly, It is described the basic concepts that relate these networks to the biological neural structure that form the nervous system. In addition, several characteristics, specifications, related structures, different applications are detailed and how the training of these models occurs.

The work in question proposes an analysis of the capacity of Perceptron neural networks in several aspects. In this way, It is proposed this work the implementation and testing of such networks in a real application. Therefore, this approach made it possible to demonstrate through a real problem that these networks are efficient in obtaining a satisfactory solution and consistent with the results found.

Key words: ANN, Perceptron, Machine Learning, AI.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

IA – Inteligência Artificial (*Artificial Intelligence*)

RNA – Rede Neural Artificial (*Artificial Neural Network - ANN*)

MLP – *Perceptron* Multicamada (*Multilayer Perceptron*)

SLP – *Perceptron* de Camada Única (*Single-Layer Perceptron*)

CPU – Unidade Central de Processamento

BP – Algoritmo de retropropagação (*Backpropagation*)

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, observou-se o surgimento de novas tecnologias que permitiram a criação de mecanismos que possibilitam uma melhoria significativa na interação entre homem e máquina. Hoje, inúmeras aplicações fazem uso de sistemas inteligentes com o intuito de resolver o maior número possível de problemas nas mais diversas áreas do conhecimento. A ideia de se construir uma máquina ou mecanismo autônomo, que seja dotado de inteligência, se constitui um sonho antigo dos pesquisadores das áreas de ciências e engenharias [20]. Dito isso, as redes neurais artificiais tem um papel fundamental na construção de sistemas inteligentes.

Segundo [20], define-se uma rede neural artificial como:

Modelos computacionais inspirados no sistema nervoso de seres vivos. Possuem a capacidade de aquisição e manutenção do conhecimento (baseado em informações) e podem ser definidas como um conjunto de unidades de processamento.

As redes neurais artificiais - RNA são indispensáveis na busca por Inteligência Artificial - IA, já que são capazes de estimar soluções e extrair informações que talvez uma pessoa não identifique rapidamente. Basicamente, pode-se dizer que algumas coisas podem passar despercebidas pela mente humana, mas que as máquinas poderiam perceber em um piscar de olhos.

Como visto em [20], um dos aspectos que fazem uma RNA ser um dos métodos mais poderosos é o fato de terem habilidades de mapeamento não lineares aprendendo os comportamentos envolvidos através de amostras obtidas. Entende-se que uma rede neural pode entregar para os computadores uma nova possibilidade, isto é, uma máquina que não fique presa às regras pré-programadas e abre variadas opções de aprender com seus próprios erros, aumentar a chance de acertos e apresentar melhorias a cada execução.

Entre um dos modelos mais simples de rede neural estão às redes *Perceptron* [20]. Tais redes são modelos inspirados na retina, sendo aplicadas principalmente na identificação de padrões geométricos e atuam como um classificador binário. Basicamente, existem dois tipos de *Perceptron*, conforme mostrado a seguir:

- *Perceptron Single-Layer* (Camada única): Também denominado *Perceptron* Simples, é capaz de aprender a classificar os elementos em apenas dois grupos distintos.
- *Perceptron Multi-Layer* (Multicamadas): É semelhante ao *Perceptron* Simples, porém, são utilizados em casos não linearmente separáveis, isto é, onde apenas uma reta não consegue separar os elementos.

A Inteligência Artificial vem sendo moldada ao longo do tempo e tem sido notado um grande investimento de corporações respeitadas neste campo de pesquisa. Segundo o *New York Times* [9], a Ford investiu cerca de US$ 1 bilhão em *start-up* de Inteligência Artificial para o desenvolvimento de automóveis autônomos. Mas, mesmo com tantas pesquisas e investimentos, ainda há uma enorme dificuldade em construir máquinas capazes de imitar ações humanas. Tais habilidades, como reconhecer um rosto ou aprender uma nova música podem ser complexas demais para serem transcritas para um computador.

Diante desse cenário, a motivação deste trabalho é fazer uma análise de alguns conceitos de aprendizagem de máquina aplicados ao reconhecimento de padrões, pois, é de extrema importância entender como utilizar e aplicar tais estratégias em meio ao ambiente computacional.

Este trabalho, portanto, tem por objetivo geral demonstrar a estrutura envolvida nos modelos de redes neurais e apresentá-las de forma objetiva e com todos os processos que as regem. Em particular, tendo foco principal na construção do Algoritmo de rede neural *Perceptron*, mais especificamente, as redes do *Single-Layer*. Além disso, o assunto em questão é muito abordado em outras leituras como na obra *Neural Networks and Learning Machines* por Simon Haykin, porém, com foco em um ponto de vista mais teórico com pouco embasamento de como esses métodos são aplicados ao reconhecimento de padrões e de que maneira podem contribuir em outros ambientes, além da computação. Com isso, este trabalho foi elaborado utilizando abordagens de fácil compreensão e com exemplos práticos de como a rede neural *Perceptron* pode solucionar tais problemas, a fim de demonstrar a importância que essas redes possuem na análise de informações.

No capítulo 2, são introduzidos tópicos essenciais sobre as RNAs, a fim de explicar de forma simples todo o seu funcionamento e arquitetura. O presente capítulo pretende ilustrar os elementos básicos envolvidos na estrutura do modelo, tais como os neurônios artificiais e tópicos referentes aos processos de aprendizagem de uma rede.

No capítulo 3, são inseridas as conceituações sobre as redes *Perceptron Single-Layer*, junto da construção do Algoritmo em suas fases de treinamento, execução e classificação de amostras.

O capítulo 4 é dedicado à abordagem das redes *Perceptron* do tipo Multicamadas (MLP). Tal capítulo ilustra superficialmente as redes MLP fazendo uma abordagem de conceitos básicos e em que situações são aplicadas.

E por fim, o capítulo 5 possui as considerações finais e as propostas de trabalhos futuros.

2 REDES NEURAIS ARTIFICIAIS

As RNAs são técnicas computacionais que utilizam modelos matemáticos para simular o funcionamento dos neurônios biológicos, que por sua vez são os responsáveis por transmitir informações para o cérebro na forma de impulsos nervosos [19]. Com isso, uma RNA possui a capacidade de aprender e executar tarefas sem terem sido pré-programadas para tais ações.

Nas seções a seguir, abordam-se os principais conceitos sobre redes neurais e elementos básicos de seu funcionamento.

2.1 NEURÔNIO BIOLÓGICO

Este tópico fornece informações fundamentais sobre as células nervosas para uma melhor compreensão dos conceitos básicos que abrangem as redes neurais artificiais. Além disso, esses conceitos são essenciais para compreensão dos próximos capítulos.

O neurônio é uma das unidades fundamentais que compõe toda a estrutura cerebral do sistema nervoso central. Como pode ser visto em [10], tais células são responsáveis por transmitir informações através da diferença de potencial elétrico em sua membrana. Essas células nervosas conduzem informações através de estímulos nervosos, que por sua vez são interpretadas pelo cérebro. Pode-se observar a formação de um neurônio biológico conforme ilustradas na Figura 1.

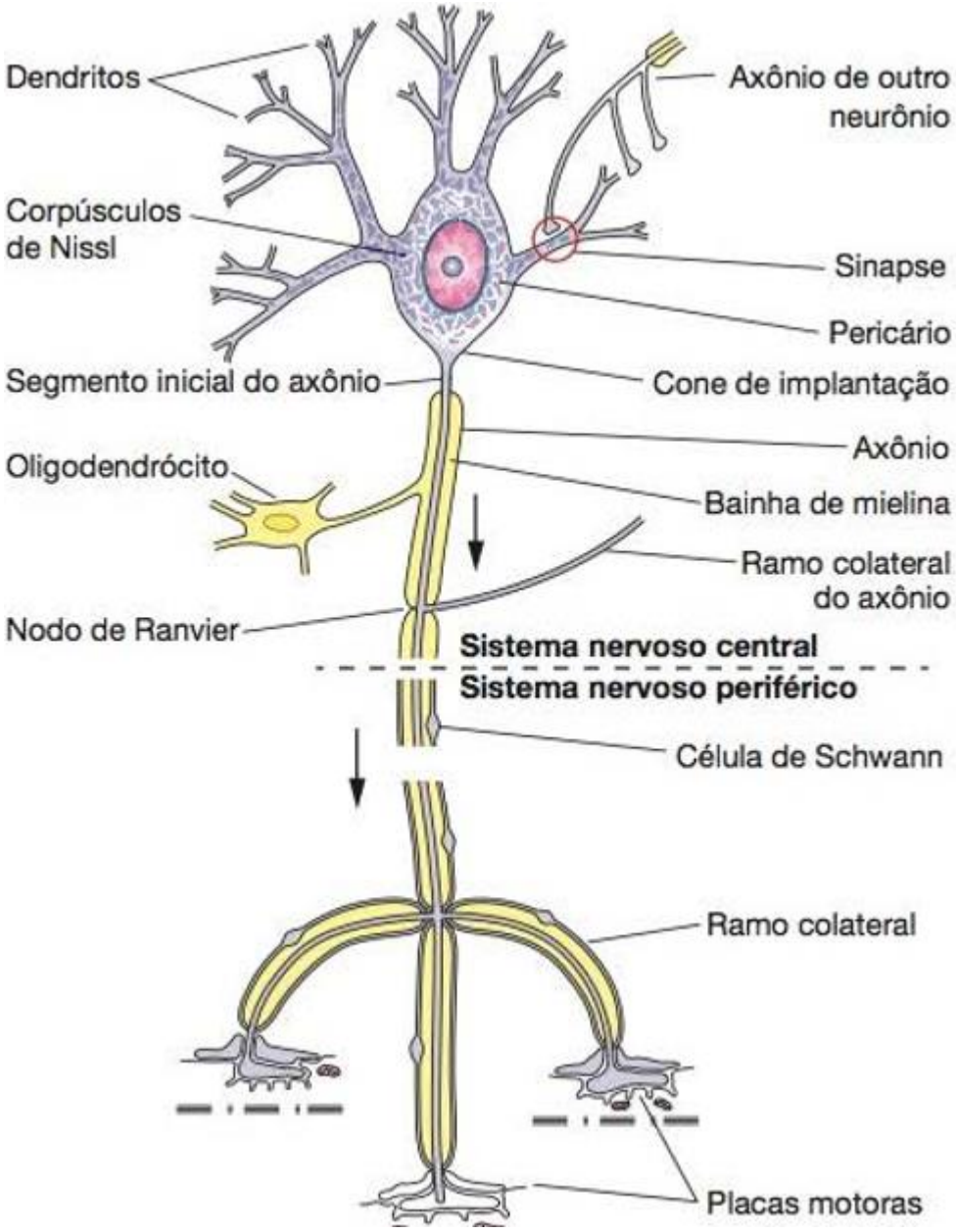


Figura 1 - Neurônio biológico [10]

Os neurônios estão interligados e distribuídos por todo o organismo, formando assim uma rede de comunicação entre as células e o sistema nervoso central. O cérebro humano possui cerca de 100 bilhões de neurônios [10] e cada um deles possui uma morfologia complexa, mas que basicamente são constituídas de três partes, ou seja:

- Os **dendritos** são as finas ramificações presentes nas extremidades ao longo de toda célula nervosa que são especializadas na recepção dos estímulos nervosos provenientes de outras partes do organismo e são transmitidos para o restante do corpo da célula.
- O **corpo celular** ou pericárdio desempenha um importante papel, já que é responsável por todas as informações originadas dos dendritos.
- Por fim, o **axônio**, formado por um único filamento prolongado ao longo do neurônio que possui a missão de enviar impulsos nervosos para o meio externo da célula.

Além disso, os neurônios possuem algumas classificações de acordo com a ausência ou presença das estruturas discutidas anteriormente. Segundo [10], pode-se classificar um neurônio de acordo com a sua morfologia, isto é:

- **Neurônios multipolares**: Estes são constituídos por um corpo celular, diversas ramificações em sua extremidade (dendritos) e um axônio. Tais tipos estão presentes em todo o tecido nervoso e são os mais encontrados no organismo.
- **Neurônios bipolares**: Estes são similares aos multipolares, porém, apresentam apenas um dendrito e um axônio. São mais comuns na estrutura sensorial, como a retina e as mucosas responsáveis pela área olfativa.
- **Neurônios pseudounipolares**: Estes apresentam um corpo celular e um único prolongamento que pode se comportar como um dendrito ou axônio.

A comunicação entre os neurônios ocorre por meio das sinapses. De acordo com [10], as sinapses são locais de contato entre os neurônios (espaço entre um neurônio e outro sem que haja qualquer contato) e tem a missão de transportar (através de substâncias neurotransmissoras) os impulsos entre cada célula nervosa, conforme ilustra a Figura 2.

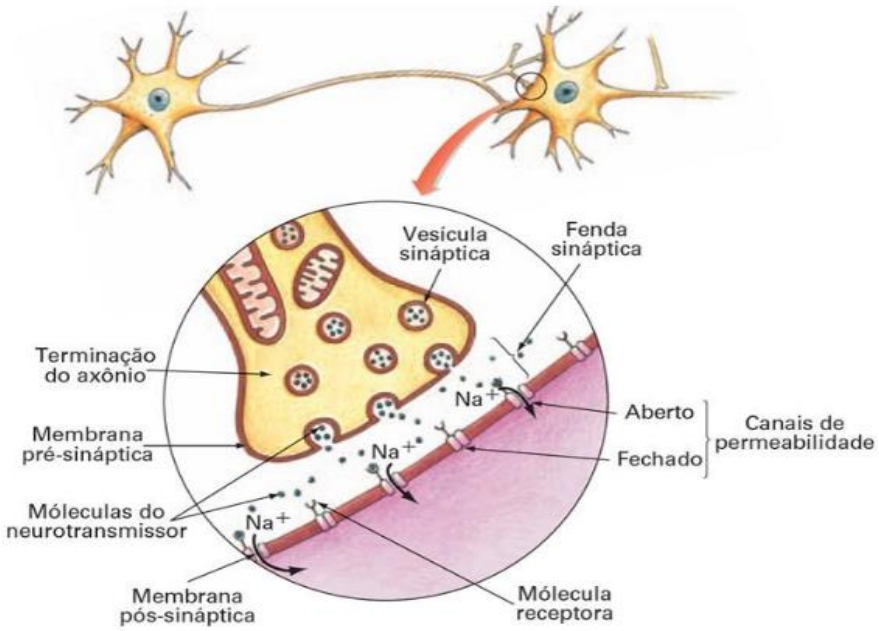


Figura 2 - Sinapse [23]

Um impulso envolve reações químicas e elétricas na célula nervosa. Um neurônio possui camadas internas e externas. Desse modo, a camada extracelular é constituída de uma alta concentração de sódio com íons positivos (Na^+), enquanto a camada intracelular é formada por íons negativos de potássio (K^-). Quando a célula nervosa se encontra no estado de “repouso”, ou melhor, quando não há qualquer movimentação de substâncias, as cargas positivas são transferidas para o lado externo e as cargas negativas para a parte interna da célula. Porém, ao ocorrer um estímulo, os pontos do lado externo da célula tornam-se íons negativos e os de dentro positivos. Esta mudança de potencial é chamada de potencial de ação.

Segundo a lei do tudo ou nada [10], um impulso nervoso só é concebido se o estimulo recebido for suficientemente intenso para estimular o neurônio. Além disso, não importa a intensidade do estímulo recebido, o impulso será sempre o mesmo em todas as ocasiões.

2.2 O NEURÔNIO ARTIFICIAL AS REDES NEURAIS

Com base nos conceitos abordados na seção anterior, os tópicos seguintes ilustram o funcionamento geral de um neurônio artificial que simulam características importantes a respeito dos neurônios biológicos.

O neurônio artificial é um modelo matemático que simula as funções dos neurônios biológicos, apresentados anteriormente. Conforme [20], define-se tal modelo como unidades processadoras que compõem a estrutura simples de uma rede neural.

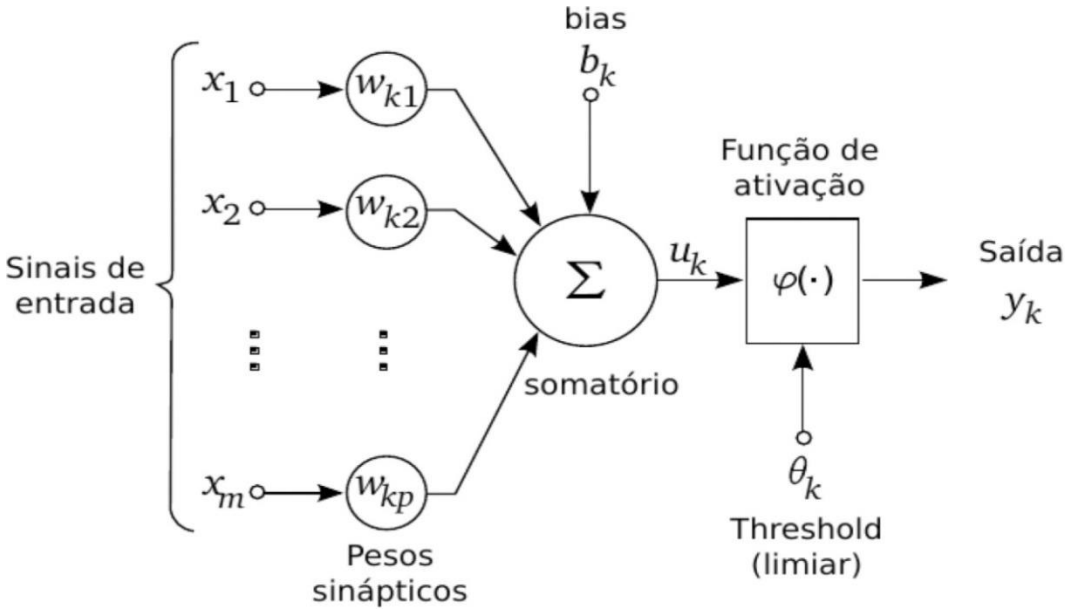


Figura 3 - Modelo de um neurônio artificial [21]

Desta maneira, pode-se observar pela Figura 3 que essencialmente um neurônio artificial é constituído de alguns elementos e funções matemáticas, isto é:

- **Sinais de entrada**: São os valores assumidos pela variável de entrada e que serão introduzidos no neurônio. As entradas também podem ser provenientes de outros neurônios constituindo assim a topologia de uma rede neural artificial.
- **Pesos sinápticos**: Os pesos sinápticos indicam a relevância que uma dada entrada possui, onde, cada uma das entradas é multiplicada ao seu peso res-

pectivo. Inicialmente, cada peso sináptico pode ter valores aleatórios, já que os mesmos serão atualizados ao decorrer da execução.

- **Função Soma**: Tal função é o somatório da multiplicação dos pesos pelas suas entradas com o intuito de resultar em um potencial de ativação, representado pela expressão:

$$SOMA = \sum_{i=1}^{n} X_i \cdot W_i - \theta \quad (1)$$

Ou seja:

$$SOMA = X_1 \cdot W_1 + X_2 \cdot W_2 + \cdots + X_n \cdot W_n - \theta \quad (2)$$

- **Função de ativação**: É constituída por uma determinada função para limitar em intervalos uma saída *y*. A função de ativação possui algumas características em particular que serão abordadas de forma mais detalhada na seção seguinte.
- **Limiar de ativação – *Bias***: Este termo especifica qual será o patamar apropriado para que o resultado produzido pela função soma possa gerar um valor de disparo em direção à saída do neurônio [20].
- **Saída**: Valor de resultado final produzido pelo neurônio após a realização de todas as atividades anteriores.

Então, resume-se o funcionamento de um neurônio artificial de acordo com as seguintes etapas:

- **Etapa 1**: Inicialmente, um conjunto correspondente aos valores entradas e saídas desejadas é introduzido no neurônio artificial. Além disso, os pesos são iniciados com valores randômicos.
- **Etapa 2**: Deve ser feita a multiplicação do peso W_i a sua respectiva entrada X_i.
- **Etapa 3**: Os dados são passados para a função soma, vista anteriormente. Além disso, não se deve esquecer a subtração do limiar de ativação (*Bias*).

- **Etapa 4**: A variável *u* recebe o valor do resultado proveniente da função soma e aplica uma função de ativação para limitar a saída do neurônio.
- **Etapa 5**: Por fim, a saída *y* recebe o valor final produzido pelo neurônio.

Basicamente, o neurônio artificial funciona como um neurônio biológico, ou seja, as entradas passam por toda sua estrutura como os impulsos nervosos e saem até a saída do neurônio se propagando para outros neurônios da rede. Seguindo os conceitos descritos, os neurônios artificiais podem ter suas saídas interligadas com outros, formando assim uma rede neural totalmente conectada. Por sua vez, essas redes podem ser constituídas por uma ou mais camadas neurais. Segundo [20], tais camadas podem ser divididas em três partes principais, conforme ilustra a Figura 4.

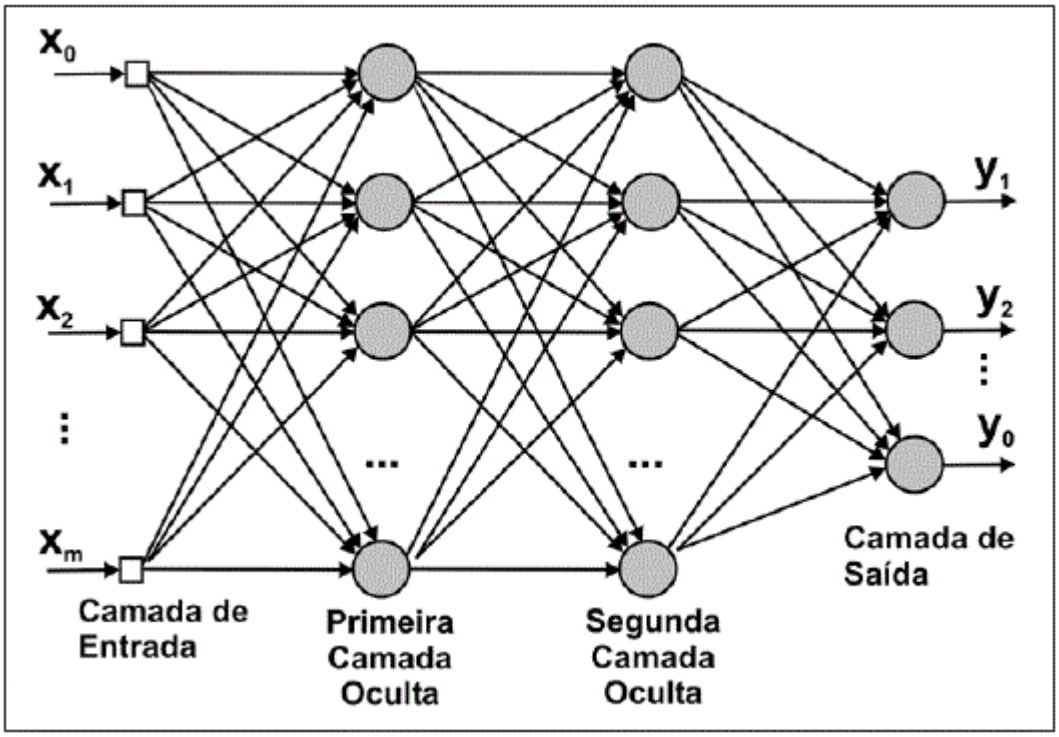


Figura 4 - Redes Neurais Multicamadas [1]

A **Camada de entrada** é responsável por recebe todas as informações de entrada da rede neural e os envia para a camada seguinte. Logo, essas informações são passadas para as **camadas intermediárias**, cuja missão é realizar o processa-

mento interno da rede, além de extrair as características vindas dos processos e informações da camada de entrada. Por fim, a **camada de saída** deve apresentar os resultados finais da rede, onde, estão prontos e já processados pelas camadas anteriores.

2.3 FUNÇÃO DE ATIVAÇÃO

A função de ativação tem o papel de limitar a saída de um neurônio artificial dentro de um intervalo pré-definido. O exemplo a seguir faz uma analogia para melhor entendimento das funções de ativação.

Hoje, um computador possui processadores potentes que podem realizar muitos cálculos em um curto intervalo de tempo, porém, tais componentes aquecem rapidamente. Então, para evitar que a CPU seja danificada existem softwares que identificam a temperatura atual do hardware, permitindo que o computador desligue em caso de superaquecimento. Isso significa que há um limiar para determinar se o computador deve ou não ser desligado, ou seja, se a temperatura x for maior que tal limite, então o computador deve ser desligado imediatamente.

Basicamente, existem diversos tipos de função de ativação, sendo as mais utilizadas em redes neurais [5]:

- **Função degrau (*Heavyside*)**: Pode assumir valor 1, caso o potencial de ativação de um neurônio *u* seja maior ou igual a zero. Se caso for negativo, então assume valor zero. Tal função é representada pela expressão:

$$g(u) = \begin{cases} 1, & Se\ u \geq 0 \\ 0, & Se\ u < 0 \end{cases} \quad (3)$$

A representação gráfica desta função pode ser conferida na Figura 5.

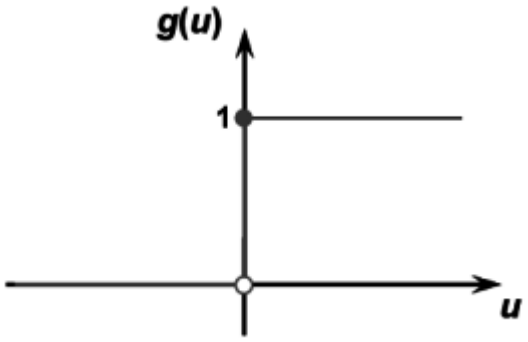


Figura 5 - Função de ativação degrau [20]

- **Função sigmóide**: O resultado de saída do neurônio na função sigmoide sempre assume valores reais entre 0 e 1. Tal função é representada pela expressão:

$$g(u) = \frac{1}{1 + e^{-\beta u}} \tag{4}$$

e equivale a constante de Euler, onde, *e* = 2,71828...;

β é uma constante real que está associada ao nível de inclinação da função.

A representação gráfica desta função pode ser conferida na Figura 6.

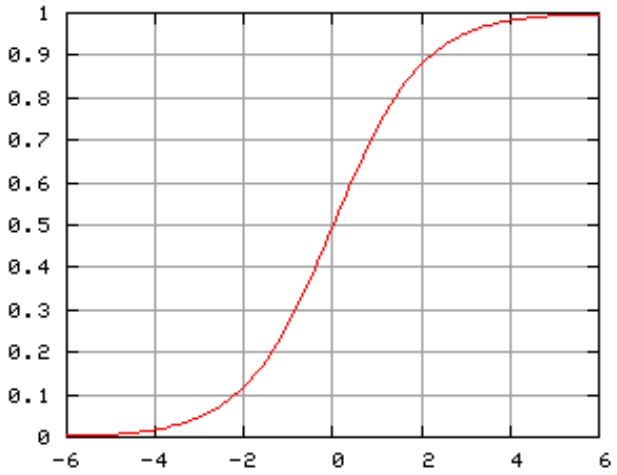


Figura 6 - Função de ativação sigmóide [20]

- **Função linear**: O resultado produzido pela saída é igual aos valores do potencial de ativação *u*. Tal função é representada pela expressão:

$$g(u) = u \tag{5}$$

A representação gráfica desta função pode ser conferida na Figura 7.

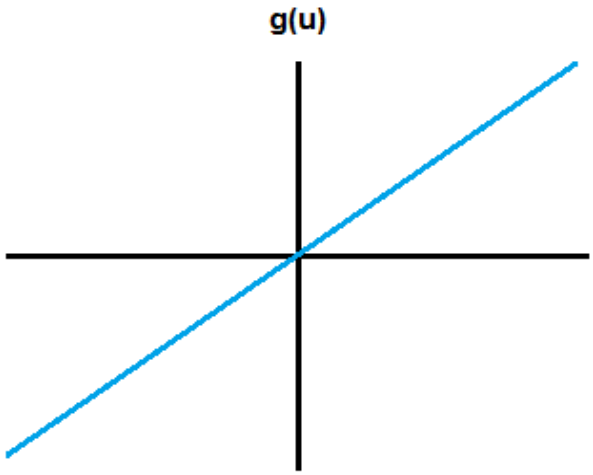


Figura 7 – Função de ativação Linear [20]

- **Função tangente hiperbólica**: O resultado produzido pelas saídas assumirá valores entre -1 e +1. Tal função é representada por:

$$g(u) = \frac{1 - e^{-\beta \cdot u}}{1 + e^{-\beta \cdot u}} \tag{6}$$

e equivale a constante de Euler, onde, *e* = 2,71828...;

β é uma constante real que está associada ao nível de inclinação da função.

A representação gráfica desta função pode ser observada na Figura 8.

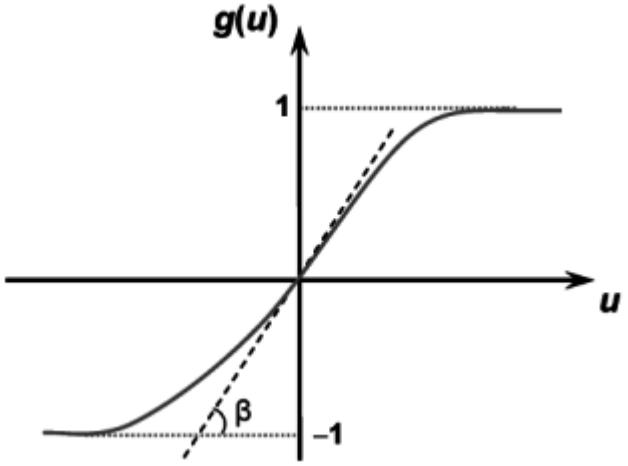


Figura 8 - Função de ativação tangente hiperbólica [20]

2.4 ARQUITETURA GERAL DE UMA REDE NEURAL

Conforme mencionado anteriormente, uma rede neural pode ser constituída por diversos neurônios artificiais. Desta maneira, a arquitetura trata a organização que cada neurônio se encontra na estrutura interna da rede e a forma que estão dispostas às conexões sinápticas neurais entre elas.

Antes de iniciar os próximos capítulos, devem-se entender os tipos de arquitetura de RNA existentes. Com isso, os parágrafos seguintes irão tratar apenas das arquiteturas relacionadas à rede *Perceptron,* foco principal deste trabalho. Primeiramente, as redes *Perceptron* tem sua arquitetura do tipo *feedforward*. Nesta arquitetura, o fluxo de informações neurais está sempre direcionado do início até o fim sem qualquer retorno, ou seja, da camada de entrada até a camada de saída (ver Figura 9).

Segundo [20], uma definição mais formal para esta arquitetura é*:*

Em uma rede com essa arquitetura, uma camada está interligada a outra de forma que o fluxo de informações em sua estrutura reside em apenas um sentido,

ou seja, partindo da camada de entrada em direção à camada de saída, inexistindo-se qualquer tipo de realimentação de valores produzidos.

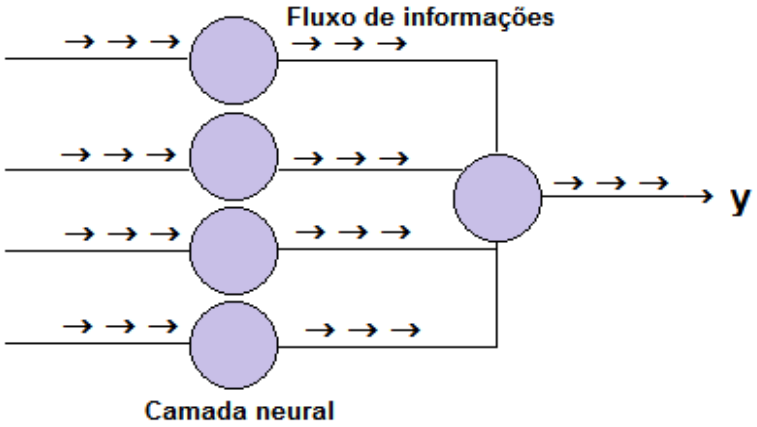


Figura 9 - Fluxo de informações
Fonte: Elaborado pelo autor

Ainda nesta discussão, as redes *feedforward* podem ser divididas de acordo com a presença de uma ou mais camadas, ou seja, *feedforward* de camada única e *feedforward* de camadas múltiplas. Baseado nisso, as redes *Perceptron Single-Layer* estão inserida no contexto *feedforward* de camada única, já que, possuem apenas uma camada em sua estrutura neural. Por outro lado, as redes do tipo *feedforward* de camada múltiplas apresentam uma ou mais camadas intermediárias, característica principal das redes *Perceptron Multiplayer.* Com base neste contexto, a Figura 10 ilustra a diferença estrutural entre as redes SLP e MLP.

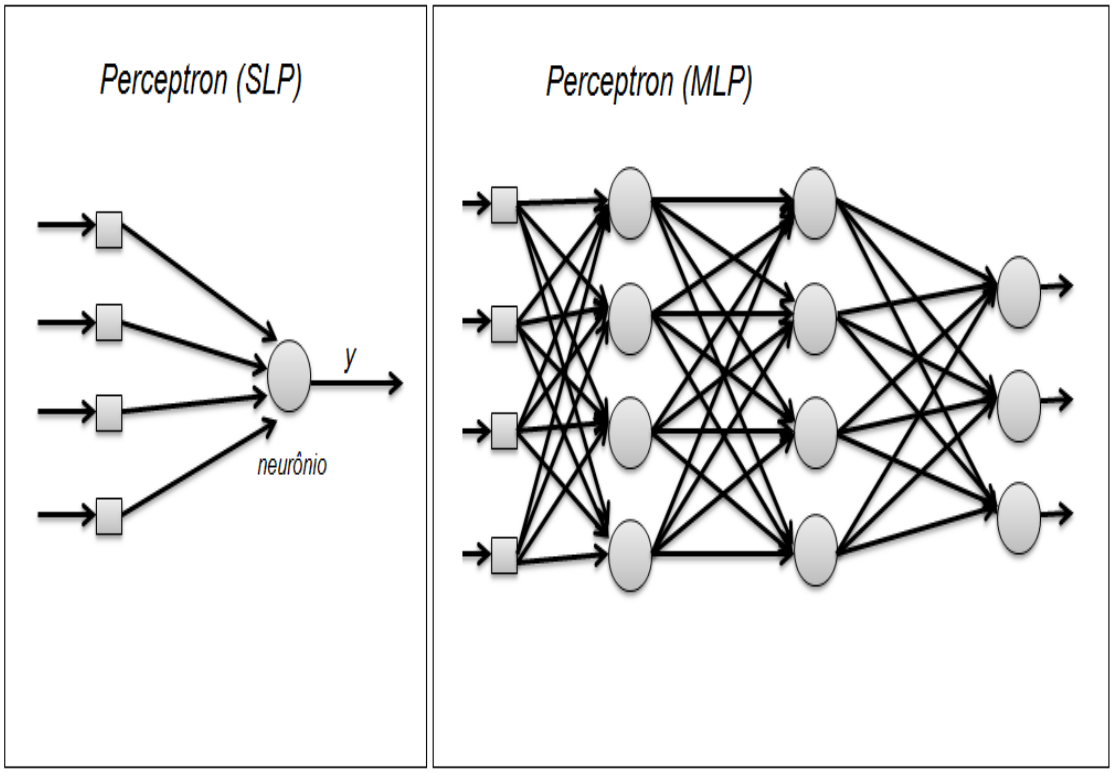


Figura 10 - Arquitetura de redes *Perceptron*

Fonte: Elaborado pelo autor

Para desenvolver uma rede neural e aplicá-las, deve-se realizar uma análise prévia do problema a ser resolvido. Certamente, um tipo de rede neural pode ser mais adequado para dado problema do que outros. Assim, na construção de uma rede neural é necessário identificar os dados que serão nela introduzidos e também fazer uma análise mais detalhada do problema descrito para identificar qual é a topologia e modelo neural mais conveniente neste caso. Com isso, o projeto de criação de uma rede neural pode ser efetuado conforme as etapas demonstradas na Figura 11.

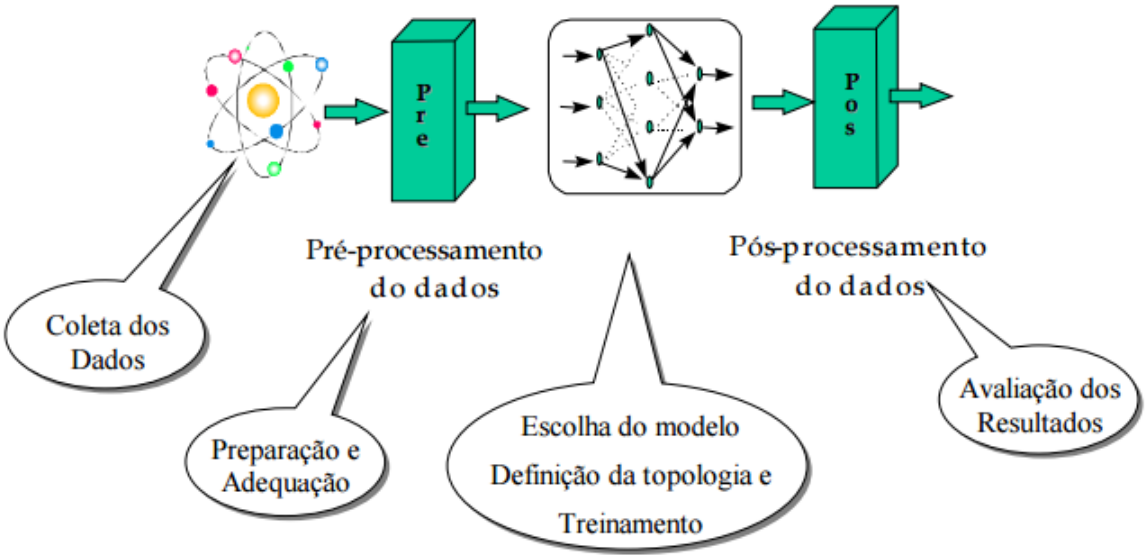


Figura 11 - Projeto de uma RNA [18]

2.5 FORMAS DE APRENDIZADO DE UMA RNA

Um dos destaques mais relevantes das redes neurais artificiais está na capacidade de aprender [20]. Baseado nisso, é importante entender os processos inseridos dentro do contexto de aprendizagem. Logo, as seções seguintes visam explicar a respeito das formas existentes para treinar uma rede neural.

2.5.1 APRENDIZAGEM SUPERVISIONADA

A aprendizagem supervisionada é definida como qualquer algoritmo que necessita de amostras para realizar o aprendizado com base nos dados recebidos como exemplos. Dessa maneira, utiliza-se um conjunto amostras para que seja feito o treinamento prévio de uma rede neural [20]. Diante desta definição, considere o seguinte exemplo:

Suponha que uma aplicação necessite realizar o reconhecimento numérico em um determinado texto. Para isso, uma rede neural pode ser utilizada para este processo. Inicialmente, necessita-se que essa rede aprenda apenas o que é o número 2. Então, foi fornecido um conjunto de fotos com o número dois, retirado de panfletos, textos em livros e algumas páginas da internet. Dessa maneira, tal algoritmo encontra características semelhantes em cada um dos exemplos fornecidos, assim identifica-se um padrão que possibilite reconhecer futuramente o número dois em qualquer texto que lhe seja fornecido.

Em geral, com a aprendizagem supervisionada é possível que seja formulada uma regra específica, com base em exemplos apresentados anteriormente, de modo a prever e classificar. Essa é uma das técnicas mais empregadas para treinamento de redes neurais e árvores de decisão[1]. O *Perceptron* é um exemplo de rede neural que faz uso dessa técnica de aprendizado. O método de funcionamento da aprendizagem supervisionada pode ser observado na Figura 12.

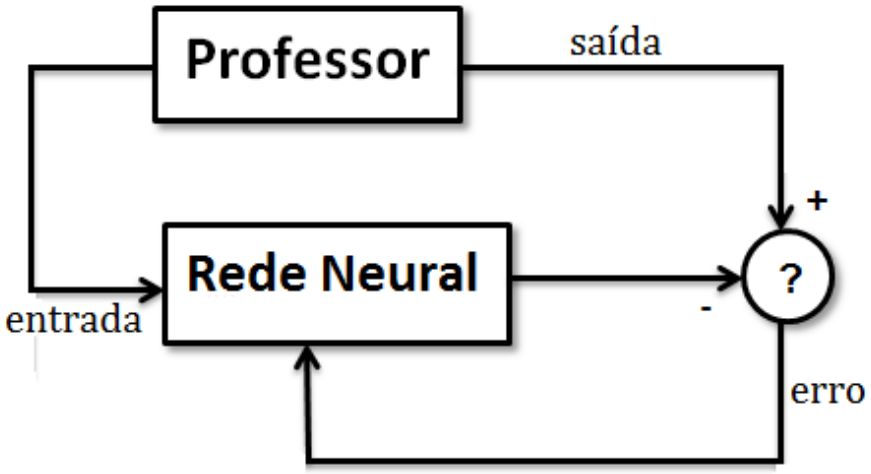


Figura 12 – Mecanismo do aprendizado supervisionado [6]

[1] Segundo [14], uma **árvore de decisão** é uma estrutura muito usada na implementação de sistemas especialistas e em problemas de classificação. Essas estruturas tomam como entrada uma situação descrita por um conjunto de atributos e retorna uma decisão.

Conforme [17], os estágios a seguir descrevem as tarefas da aprendizagem supervisionada com base no que foi visto em definições anteriores. Desta forma, podem-se descrever tais tarefas do seguinte modo:

- **Etapa 1:** Dado um conjunto formado por *n* elementos, onde, cada um de seus componentes possui o formato $\{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\}$, sendo x_i a variável correspondente as entradas e y_i representando as saídas;
- **Etapa 2:** Sabe-se que cada saída y_i foi gerado por uma função até então desconhecida. Estas informações são introduzidas no Algoritmo para que o treinamento da rede seja realizado e encontre um padrão nestes elementos.
- **Etapa 3:** Finalmente, o Algoritmo deve encontrar uma função *f* que se aproxime com a função geradora das amostras fornecidas.

2.5.2 APRENDIZAGEM NÃO SUPERVISIONADA

Diferentemente da aprendizagem supervisionada, neste método não é fornecido qualquer exemplo com as saídas desejadas. Deste modo, o treinamento prévio é inexistente. A classificação dos dados ocorre mediante técnicas estatísticas para extração de características similares entre o conjunto de informações fornecidas. Essa abordagem é muito utilizada em Mineração de Dados (*Data Mining*).

A Mineração de Dados explora e analisa informações utilizando técnicas provenientes de áreas como aprendizado de máquina, estatística e banco de dados. Além disso, pode-se dizer que o conjunto dessas técnicas fornece ferramentas de análise extremamente importantes que possibilitam a descoberta de relações presentes entre os dados e com base nelas fazer previsões de futuras tendências em meio a diversos campos de pesquisa e empresarial.

Então, pode-se citar a associação e o *clustering* como sendo técnicas que utilizam aprendizado não supervisionado compreendidos dentro do campo de Mineração de dados. Em Mineração de dados, a associação tem por objetivo utilizar regras específicas para encontrar a similaridade e os relacionamentos presentes entre

elementos em meio a um conjunto de dados. Já as técnicas de *clustering* agrupam os dados com base no grau de similaridade dos mesmos para formar conjuntos de elementos com características parecidas.

2.6 APLICAÇÕES REAIS DE UMA REDE NEURAL

Em aplicações reais, as RNAs podem ser treinadas para resolver uma infinidade de problemas em diversas áreas do conhecimento. De um modo geral, as redes neurais podem ser aplicadas principalmente no reconhecimento de padrões com a finalidade de identificar características semelhantes em um conjunto de dados. Na maior parte, as redes neurais são inseridas em áreas onde há problemas complexos que não podem ser resolvidos apenas utilizando a programação baseado em regras.

Ainda neste contexto, as redes neurais também podem ser aplicadas no campo de visão computacional. Resumidamente, a visão computacional é a área que desenvolve técnicas para extração de informações de imagens, dessa forma, as RNAs são partes fundamentais deste processo. Além das aplicações citadas, de acordo com [20], também é possível encontrar aplicações nos mercado financeiro com o objetivo de prever ações financeiras antecipadamente.

As redes neurais também têm sido muito utilizadas nas séries temporais. Uma série temporal pode ser descrita como um conjunto de observações feitas em alguma informação específica ao longo de um determinado período de tempo. Então, o número de compradores semanais de uma companhia, valores de investimento mensal na bolsa de valores ou até mesmo a quantidade de chuva diária de um determinado local são exemplos de séries temporais.

Hoje, muitas outras áreas usufruem do uso de sistemas inteligentes. De certa forma, as redes neurais estão contidas nesses ambientes, já que são ferramentas poderosas que permitem uma manipulação mais aprofundada de um grande volume de dados. Enfim, compreender essas ferramentas é primordial no projeto de sistemas que possuem como ponto principal a análise de informações. Baseado nos

conceitos vistos no presente capítulo, os assuntos a seguir irão tratar especificamente as redes do tipo *Perceptron* e suas diversas aplicações em meio ao âmbito computacional.

3 REDES NEURAIS *PERCEPTRON – SINGLE LAYER*

O presente capítulo tem por objetivo apresentar detalhadamente o funcionamento e os conceitos a respeito das redes neurais *Perceptron Single-Layer*. Para uma melhor organização das ideias e garantir a maior absorção possível do assunto abordado, tal capítulo abrange respectivamente a arquitetura básica dessas redes e também cada etapa responsável pelo seu treinamento. Além disso, o algoritmo foi elaborado e seu pseudocódigo encontra-se dividido em partes ao longo de todo o capítulo.

3.1 INTRODUÇÃO E CONCEITOS BÁSICOS

O *Perceptron* foi idealizado por Rosenblatt em 1958, com o intuito de implementar um modelo computacional inspirado na retina, objetivando-se um elemento de percepção eletrônica de sinais [20]. Em geral, esse modelo possui uma estrutura bem simples, em razão de serem constituídos por um único neurônio artificial. Além disso, é importante ressaltar que este tipo de rede neural está apto a resolver apenas problemas considerados linearmente separáveis, característica que pode ser vista nos tópicos a seguir. Além disso, é importante ressaltar que as redes *Perceptron Single-Layer* resolvem apenas problemas de classificação padrões descritos como linearmente separáveis, característica que pode ser vista nos tópicos logo a seguir.

3.1.1 PROBLEMAS LINEARMENTE SEPARÁVEIS

Basicamente, um problema linearmente separável pode ser descrito como um problema de classificação de padrões que é solucionado utilizando uma única reta linear para dividir duas classes distintas em um dado hiperplano (ver Figura 13).

Com base nesse conceito, uma alternativa para solucionar estes problemas é utilizar redes *Perceptron Single-Layer* que atuam como um classificador binário, por isso só podem definir apenas soluções para problemas descritos como sendo linearmente separável [20].

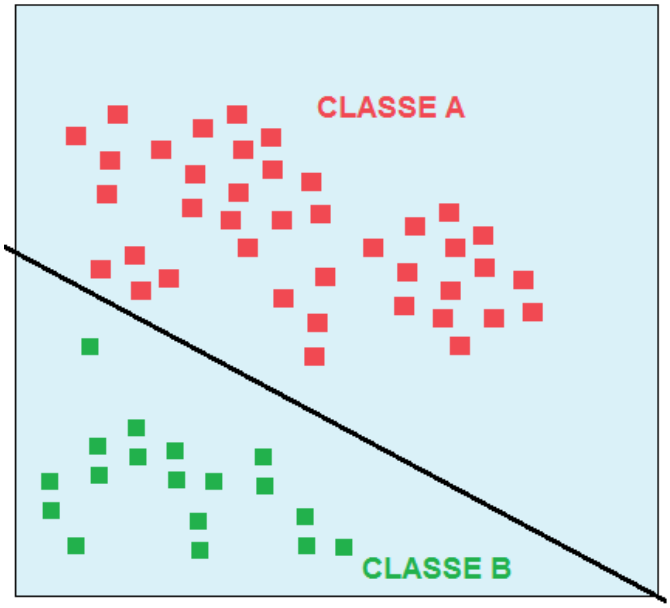


Figura 13 - Linearmente separáveis

Fonte: Elaborado pelo autor

3.1.2 PROBLEMAS NÃO LINEARMENTE SEPARÁVEIS

Diferentemente do problema anterior, nos problemas não linearmente separáveis uma reta linear pode não ser capaz de separar corretamente os elementos no hiperplano. Em alguns casos, é necessário o uso de mais de uma reta classificadora para a separação dos elementos (ver Figura 14). Portanto, para resolver estes problemas é necessário o uso de uma rede neural com mais de uma camada neural, como por exemplo, o *Perceptron Multilayer* que será apresentado mais detalhadamente no capítulo seguinte.

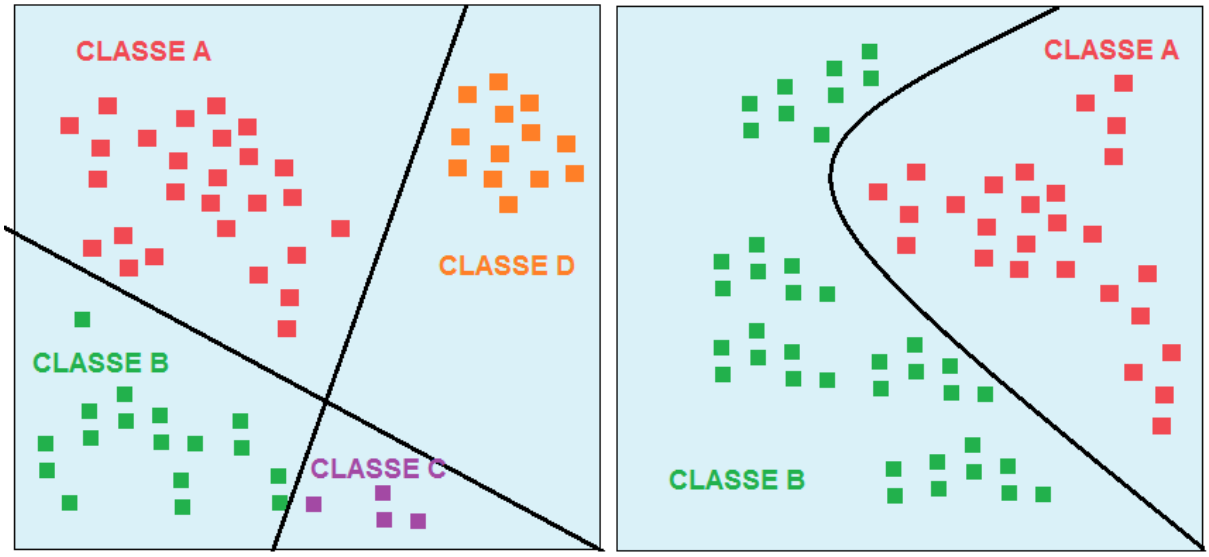


Figura 14 - Não linearmente separáveis

Fonte: Elaborado pelo autor

3.2 ARQUITETURA DO *PERCEPTRON* DE CAMADA ÚNICA

Antes de partir para uma análise mais aprofundada dos aspectos principais envolvidos nos métodos de aprendizagem do *Perceptron*, devem-se compreender os componentes básicos da estrutura dessas redes. Conforme mencionado no capítulo anterior, o *Perceptron Single-Layer* faz parte da arquitetura *feedforward* de camada única, já que possuem basicamente um neurônio artificial e apenas uma camada neural, como se observa na Figura 15.

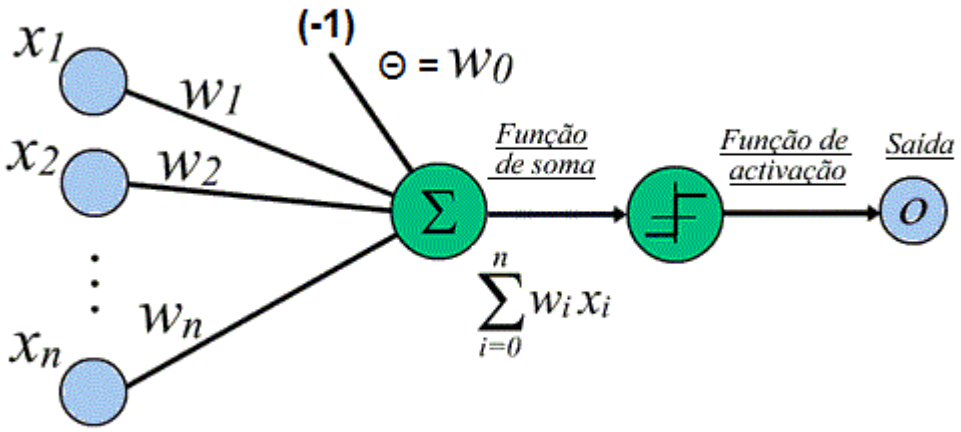


Figura 15 – Perceptron Single-Layer [7]

Diante da representação vista, partindo para um contexto mais formal, o funcionamento geral de uma rede *Perceptron* está relacionado com um conjunto de tarefas que se pode resumir em expressões matemáticas, conforme denotado a seguir:

$$y = g\left(u = \sum_{i=1}^{n} X_i \,.\, W_i - \theta\right) \tag{7}$$

Onde, y é a saída do neurônio;
g é a função de ativação;
u é o potencial de ativação;
x_i é o conjunto de entradas;
w_i é o conjunto de pesos;
θ é o limiar de ativação (*Bias*).

Analisando minunciosamente as variáveis descritas, percebe-se que o valor do limiar de ativação representado por θ(*Bias*), pode ser acrescentado no neurônio como sendo parte dos pesos sinápticos, sendo assim $\theta = W_0$, que por sua vez assume um valor negativo. Seguindo este raciocínio, se os conjuntos de elementos

são formados por $X = (x_1, x_{2,\dots,}x_n)$ e $W = (w_1, w_{2,\dots,}w_n)$, então, o potencial de ativação(u) será dado por:

$$u = ((-1) \cdot W_0 + X_1 \cdot W_1 + X_2 \cdot W_2 + \cdots + X_n \cdot W_n) \quad (8)$$

Ainda neste contexto, deve-se estabelecer uma função de ativação para o problema, de modo que essa escolha esteja de acordo com as limitações da rede neural, isto é, como a saída do neurônio só assumirá dois valores possíveis, deste modo, é necessário optar pelo uso de uma função com essa particularidade. Com base nisso, as funções mais adequadas para esses casos são (acompanhadas de seus respectivos códigos-fonte):

- **Função de ativação bipolar**: Relembrando que a função de ativação bipolar está compreendida pela seguinte notação.

$$g(u) = \begin{cases} 1, se\ u \geq 0 \\ 0, se\ u < 0 \end{cases} \quad (9)$$

Tabela 1 – Pseudocódigo – Função de ativação bipolar

```
1.  Procedimento G_Bipolar(u) retorna valor
2.  entradas : u
3.
4.  variáveis locais: valor
5.
6.  início
7.
8.   valor ← 0
9.   Se  u ≥ 0  então
10.          valor ← 1
11.     Senão
12.          valor ← 0
13.  Fim-se
14.
15.   retornar valor
16. FimAlgoritmo
```

Fonte: Desenvolvido pelo autor

- **Função de ativação degrau**: Relembrando que a função de ativação degrau é bem semelhante a bipolar e está compreendida pela seguinte notação.

$$g(u) = \begin{cases} 1, \ se \ u \ \geq 0 \\ -1, se \ u \ < 0 \end{cases} \quad (10)$$

Tabela 2 - Pseudocódigo – Função de ativação degrau

```
1.  Procedimento G-Degrau(u) retornar valor
2.  entradas: u
3.
4.  variáveis locais: valor
5.
6.  início
7.
8.    valor ← 0
9.    Se  u ≥ 0 então
10.           valor ← 1
11.         Senão
12.           valor ← -1
13.   Fim-se
14.    retornar valor
15. FimAlgoritmo
```

Fonte: Desenvolvido pelo autor

3.3 FASE DE TREINAMENTO DO *PERCEPTRON SLP*

3.3.1 VARIÁVEIS PRINCIPAIS

Nos tópicos que seguem, aborda-se apenas uma conceituação simples das variáveis que devem ter seus valores pré-estabelecidos antes da execução do treinamento da rede.

Em geral, no treinamento da rede devem-se encontrar os valores corretos para os pesos de modo que haja o menor erro possível na classificação das amostras. Mas, antes de discutir sobre o Algoritmo de treinamento do *Perceptron*, é de

suma importância conceituar detalhadamente o papel que cada variável desempenha, a fim de compreender alguns aspectos indispensáveis que podem resultar em uma maior eficácia na execução da aplicação. Logo, tais variáveis estão dispostas como mostra a Tabela 3, onde, cada uma delas está relacionada com sua representação e os valores que podem ser assumidos pelas mesmas.

Tabela 3 - Lista de variáveis

Nome	Variável	Valores (Tipo)
Sinais entrada	X_i	Real
Pesos sinápticos	W_i	Real
Saídas desejadas	d_i	Real
Taxa de aprendizado	η	$(0 < \eta \leq 1)$
Potencial ativação	*u*	Real
Saída neural	*Y*	Binário
Função Soma	*Soma*	Real
Erro de Saída	*erro*	Real
Número Max épocas	*EpocaMax*	Inteiro > 0
Contador de épocas	*ContEpoca*	Inteiro≥ 0

Fonte: Adaptado de [20]

Inicialmente, é preciso inserir na rede as amostras para o treinamento. Como alternativa, as informações de entrada e saída podem ser representadas através de vetores ou matrizes, desde que tenham os dados necessários para o treinamento. Além disso, é importante ressaltar que os pesos sinápticos são iniciados com valores randômicos e atualizados no decorrer da aplicação.

Em seguida, especifica-se a taxa de aprendizagem e o número máximo de épocas. A taxa de aprendizagem define o quão rápido é o processo de treinamento. O valor de escolha tem de ser feito cuidadosamente, pois um valor muito pequeno pode provocar uma demora considerável no processo de estabilização da rede e um valor muito grande talvez impeça que o treinamento chegue a uma solução. Baseado nisso, se estabelece um intervalo compreendido em $0 < \eta \leq 1$.

Por fim, o número de épocas indica cada apresentação completa de todas as amostras pertencentes ao subconjunto de treinamento [20]. Na realidade, o nú-

mero de épocas é uma condição de parada para a rede, já que, se a rede não encontrar uma solução aceitável, então o seu treinamento é interrompido. Desse modo, a variável correspondente ao contador de épocas irá incrementar-se a cada uma dessas apresentações (ciclos) até que a condição de parada seja satisfeita. Além disso, os valores de saída são binários, ou seja, só existem dois resultados possíveis que serão os responsáveis por classificar uma determinada amostra em um grupo ou outro.

3.3.2 TREINAMENTO DO *PERCEPTRON SINGLE-LAYER*

Esta seção consiste em mostrar como ocorre o treinamento de uma rede *Perceptron*. Neste contexto, para uma abordagem simples, o funcionamento da rede foi dividido em etapas para facilitar a compreensão deste processo. Além disso, as variáveis descritas são as mesmas apresentadas previamente. Baseado nisso, o treinamento da rede é feito através de um conjunto de passos, conforme ilustra a Figura 16 [20].

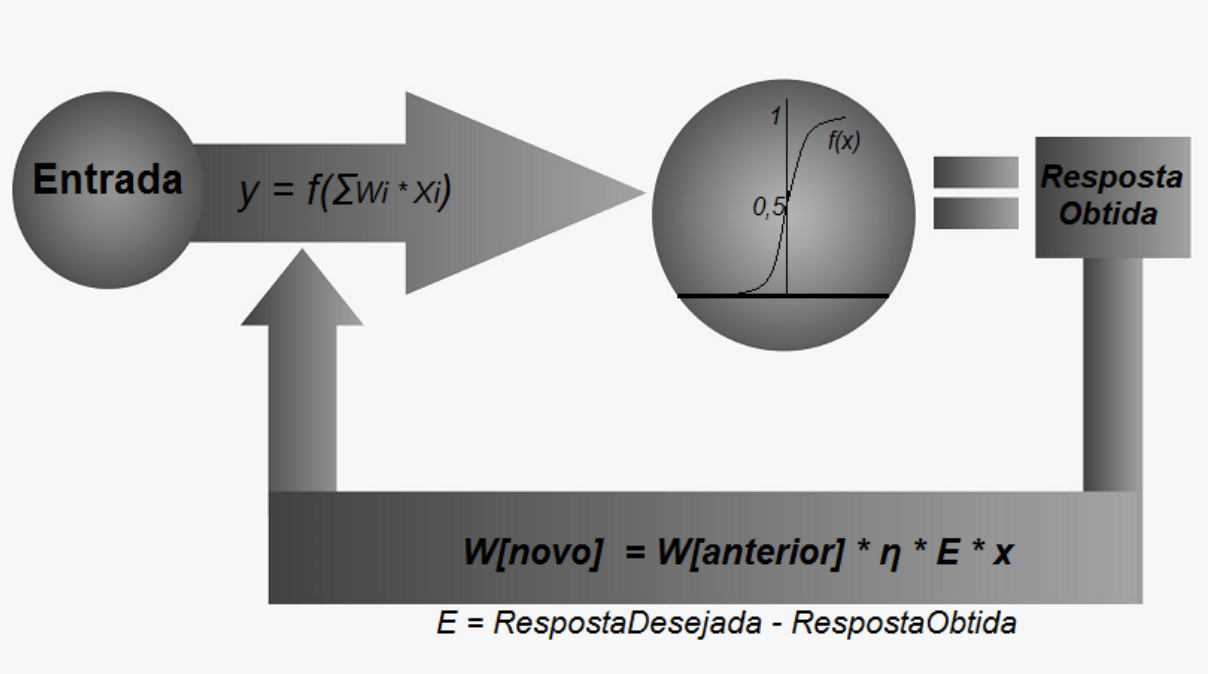


Figura 16 - Esquema de treinamento de uma rede neural [24]

- **ETAPA 1**: Inicializar os pesos randomicamente e as variáveis iniciais da rede . Além disso, é de extrema importância respeitar o intervalo definido para a taxa de aprendizagem, como foi visto anteriormente.
- **ETAPA 2**: Apresenta-se para o Algoritmo uma matriz ou vetor com as entradas e saídas desejadas das amostras.
- **ETAPA 3**: Nesta etapa, as informações da matriz aprendizado são conduzidas para a função somatório e depois para a função de ativação.
- **ETAPA 4**: Comparar o resultado de *y* com a saída desejada pela rede. Se tais resultados forem diferentes, então é necessário fazer o ajuste de pesos e do valor *Bias* (O ajuste de pesos pode ser visto mais detalhadamente na seção seguinte).
- **ETAPA 5**: Retornar a etapa 3 e repetir todo o procedimento até que o erro seja inexistente ou o contador de épocas seja maior ou igual ao máximo de épocas.

Como se pode notar, o processo de treinamento é caracterizado pela sua simplicidade e sua facilidade de implementação algorítmica. Baseado nos passos descritos, o treinamento do *Perceptron Single-Layer* pode ser expresso por meio do

pseudocódigo descrito na Tabela 4. O código produzido utiliza as técnicas e componentes mostrados em tópicos introdutórios deste trabalho. Além disso, a função de correção de pesos está descrita e desenvolvida na seção seguinte.

Tabela 4 - Pseudocódigo – Procedimento de treinamento da rede

```
1.  Procedimento Treinamento-SLP ( exemplo , pesos)
2.  entradas: exemplos {Conjunto de amostras de treinamento}
3.            pesos {Conjunto de pesos com valores pequenos e aleatório
4.            SaidaDesejada^K {Resposta desejada para a k-ésima amostra}
5.
6.  variáveis globais: MaxEpocas{Número máximo de épocas}
7.  variáveis locais: Contepoca {Contador de épocas},
8.                    saídaObtida {Saída Obtida no treinamento da rede}
9.                    Soma {Combinador Linear}
10.                   erro{O treinamento está concretizado se não existir erro}
11. Início
12.
13.  Contepoca ← 0 {Iniciar contador de épocas}
14.  saídaObtida ← 0
15.  Soma  ← 0
16.  Repetir todas as instruções:
17.   erro ←"não existe"
18.   Para todos os exemplo de treinamento i faça
19.    Para todos os exemplos de treinamento j faça
20.      Se j = 0 então
21.            soma ← soma + (-1) * pesos[ j ]
22.         Senão
23.            soma ← soma + exemplos[ i , j ] * pesos[ j ]
24.      Fim-Se
25.   Fim-Para
26.         Se soma ≥ 0 então
27.               saídaObtida ← 1
28.           Senão
29.               saídaObtida ← 0
30.         Fim-se
31.         Se (saídaObtida  ≠ saidaDesejada^K) então
32.           Realizar Ajuste de Pesos( )
33.           erro ← "existe"
34.         Fim-se
35.   Fim-Para
36.   Contepoca ← Contepoca + 1
37.    Até que (erro = “não existe”)
38. FimAlgoritmo
```

Fonte: Desenvolvido pelo autor

3.3.3 AJUSTES DE PESO DO *PERCEPTRON*

O ajuste dos pesos do *Perceptron* visa determinar uma combinação correta de pesos baseado nas saídas desejadas das amostras. Nesta etapa, o processo de ajuste é repetido até que a saída obtida seja igual a saída desejada para cada uma das amostra. É importante perceber que o reconhecimento de padrões ocorre através da modificação dos pesos até que tais valores se estabilizem e a rede finalmente esteja treinada.

Da descrição acima, pode-se dizer que o treinamento consiste em realizar constantes ajustes nos pesos e no limiar de ativação (*Bias*) para obter valores com base nas amostras de exemplo inicialmente fornecidas. Com base no contexto computacional, utilizando vetores como forma de representação para as expressões que se seguem, a equação seguinte é definida como a regra para o ajuste dos pesos:

$$W_{[atual]} = W_{[anterior]} + \eta \cdot (d_{[i]} - y) \cdot x_{[i]} \quad (11)$$

$$\theta_{[atual]} = \theta_{[anterior]} + \eta \cdot (d_{[i]} - y) \cdot x_{[i]} \quad (12)$$

Assim, como o Θ(Bias) possui a mesma fórmula de ajuste. Então, tal variável pode ser incorporada como pertencente aos pesos, onde, $W_0 = (-1) * \Theta$. De forma complementar, a expressão pode ser resumida como mostra a seguir:

$$W_{[0]} = W_{[0]} + \eta \cdot (d[i] - y) \cdot (-1) \quad (13)$$

$$W_{[i]} = W_{[i]} + \eta \cdot (d[i] - y) \cdot x[i] \quad (14)$$

Ainda em relação a isso, a fim de oferecer um maior entendimento do assunto e uma futura aplicação dos conhecimentos adquiridos. O algoritmo presente na Tabela 5 denota uma função para a realização do ajuste de peso, onde, a primeira posição do vetor peso foi definida como sendo o termo *Bias*. O valor *Bias* também

pode ser inserido na posição final do vetor W_i, desde que as fórmulas de ajuste estejam de acordo com o que foi discutido na equação (14).

Tabela 5 - Pseudocódigo – Algoritmo para ajuste de pesos

```
1.  Procedimento Ajuste-Pesos(i, saidaObtida)
2.
3.  entradas: i , saídaObtida
4.
5.  variáveis locais : erro
6.
7.  variáveis globais: pesos {Conjunto de pesos}
8.                     entradaxi {Conjunto de entradas}
9.
10.
11. Início
12.
13.   erro ← saidaDesejada[ i ] – saídaObtida
14.   Para todos os pesos j faça
15.       Se j = 0 então
16.           Pesos[ j ] ← pesos[ j ] + ( taxadeAprendizado * erro * (-1));
17.       Senão
18.           Pesos[ j ] ← pesos[ j ] + ( taxadeAprendizado * erro * entradaxi)
19.       Fim-Se
20.   Fim-Para
21. FimAlgoritmo
```

Fonte: Desenvolvido pelo autor

3.4 FASE DE CLASSIFICAÇÃO DE AMOSTRAS

Esta seção pretende demonstrar como ocorre a fase de classificação das amostras no *Perceptron*. É durante esta etapa que as amostras são classificadas, com base no padrão encontrado no treinamento da rede. Segundo [20], uma rede será considerada treinada quando inexistir erro entre os valores desejados em comparação com aqueles produzidos em suas saídas. Portanto, os pesos devem estar devidamente estabilizados. Diante disso, o algoritmo da Tabela 6 foi fundamentado dos conceitos e funções apresentadas anteriormente.

Tabela 6 - Pseudocódigo – Procedimento de classificação de amostras

```
1.  Procedimento CLASSIFICAR-AMOSTRAS (amostra, pesos-ajustados):
2.  entradas: amostra { Amostras a serem classificadas }
3.  pesos-ajustados { Conjunto de pesos já ajustados após o treinamento}
4.  variáveis locais: u {Potencial de ativação}
5.
6.  início
7.    u ← 0
8.    Para cada uma das amostras i faça
9.      u ← u + amostras[ i ] * pesos-ajustados[ i ]
10.   Fim-para
11.     y ← g(u), onde g é a função de ativação escolhida (degrau ou bipolar)
12.     Se y > 0 então
13.           "A amostra pertence a classe A"
14.       Senão
15.           "A amostra pertence a classe B"
16.     Fim-se
17.
18. FimAlgoritmo
```

Fonte: Desenvolvido pelo autor

De forma resumida, com base no que foi desenvolvida, a etapa de classificação pode ser descrita como mostra os passos seguintes:

- **Passo 1**: Primeiramente, deve-se introduzir nos parâmetros da função as novas amostras e pesos pós-treinamento. Nesse momento, é importante entender que as novas amostras também contem o valor de *Bias*. Como exemplo, considere as seguintes amostras com valores aleatórios, onde, a última posição do vetor contém o termo *Bias* apresentado com o valor *Bias* = -1.

  Amostra1 = [0.93345, 0.4543, 0.3345, -1];

  Amostra2 = [0.43461, 0.9443, 0.5845, -1];

- **Passo 2**: Encontra-se o valor o potencia de ativação *u*, de acordo com a função somatório vista no início do capítulo. Além disso, a função somatório está indicada pela variável *soma* definida como uma variável global do Algoritmo.
- **Passo 3**: Agora, determina-se o valor da saída *y*, segundo a função de ativação *g(u).*

- **Passo 4**: Por fim , separar as classes com base na saída encontrada. Relembrando, o *Perceptron* de camada única só pode classificar os elementos em apenas duas classes.

3.5 RECONHECIMENTO DE PADRÕES UTILIZANDO RNAS

A fim de utilizar os conhecimentos teóricos sobre o *Perceptron*, esta parte do trabalho visa uma abordagem mais prática na tentativa de garantir um entendimento mais amplo do funcionamento do Algoritmo. A partir disso, como exemplo de aplicação, serão utilizadas algumas informações obtidas por meio de pesquisas científicas em outras áreas para elaborar um problema de classificação de padrões. Como exemplo, considere um conjunto de informações de uma análise físico-química de polpas de frutas congelada obtidas em [13]. Tais informações podem ser observadas de acordo com a Tabela 7.

Tabela 7 – Análise físico-química de frutas congeladas.

Polpa	Ácido Cítrico	Ácido Tartárico	Ácido Málico	Ácido Láctico	Ácido Acético
Mamão	0,4477g	0,5247g	0,4687g	0,6296g	0,4197g
Mamão	0,4477g	0,5247g	0,4687g	0,6296g	0,4197g
Mamão	0,4680g	0,5485g	0,4900g	0,6582g	0,4388g
Pêssego	0,6129g	0,7183g	0,6416g	0,8619g	0,5746g
Pêssego	0,6286g	0,7367g	0,6581g	0,8840g	0,5893g
Pêssego	0,6129g	0,7183g	0,6416g	0,8619g	0,5746g
Abacaxi	0,7544g	0,8840g	0,7897g	1,0608g	0,7072g
Abacaxi	0,7701g	0,9024g	0,8062g	1,0829g	0,7219g
Abacaxi	0,7544g	0,8840g	0,7897g	1,0608g	0,7072g

Fonte: Congresso Brasileiro de Química, 2014.

Como a rede *Perceptron Single-Layer* pode apenas separar amostras em duas classes distintas, então os dados vistos anteriormente tiveram que ser adaptados para o problema, conforme se observa na Tabela 8.

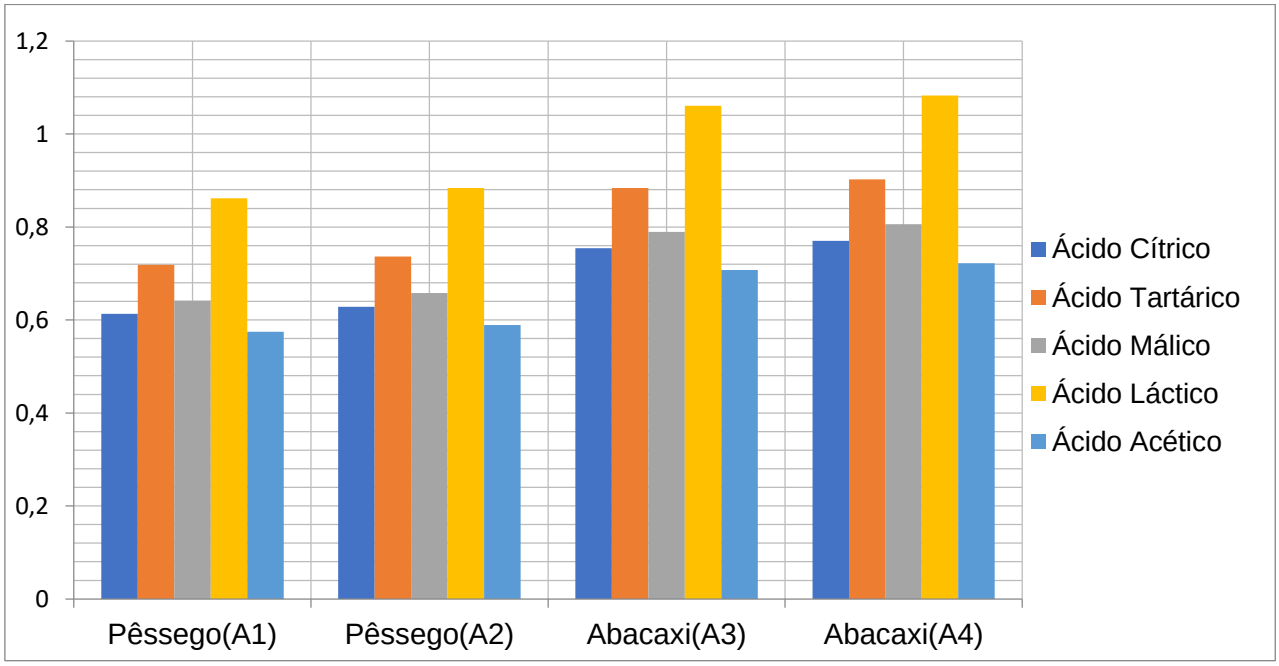
Tabela 8 - Amostras para treinamento

Amostras	Ácido Cítrico	Ácido Tartárico	Ácido Málico	Ácido Láctico	Ácido Acético
Pêssego	0,6129g	0,7183g	0,6416g	0,8619g	0,5746g
Pêssego	0,6286g	0,7365g	0,6581g	0,8840g	0,5893g
Abacaxi	0,7544g	0,8840g	0,7897g	1,0608g	0,7072g
Abacaxi	0,7701g	0,9024g	0,8062g	1,0829g	0,7219g

Fonte: Adaptado de [13]

Tais amostras foram devidamente selecionadas para que o *Perceptron* seja capaz de realizar a separação após o treinamento da rede. Para facilitar a visualização das informações e de que forma os tipos de ácidos estão distribuídos em cada uma das frutas, os dados presentes na Tabela 8 estão representados graficamente como demonstra o Gráfico 1.

Gráfico 1 - Amostras para treinamento



Fonte: Elaborado pelo autor

Com base nos dados apresentados, supondo agora que os tipos de ácido presente nas frutas são as entradas de um neurônio artificial representada por X_i e as saídas as frutas do tipo “pêssego” e “abacaxi”. Então, pode-se utilizar tanto um vetor ou matriz para representação dessas informações. Neste caso, para simplificar o desenvolvimento do projeto foi utilizada uma matriz com quatro linhas e cinco colunas, ou melhor:

$$X = \begin{pmatrix} x_{0,0} & \cdots & x_{0,4} \\ \vdots & \ddots & \vdots \\ x_{3,0} & \cdots & x_{3,4} \end{pmatrix}, d = (1, 1, 0, 0) \tag{13}$$

Obs.: O valor um (1) corresponde a fruta pêssego.

O valor zero (0) corresponde a fruta abacaxi.

O vetor *d* corresponde as saídas esperadas.

A matriz *X* corresponde a Matriz de aprendizado da rede.

.

Assim, a matriz treinamento com *n* entradas pode ser expressa matematicamente por:

$$X = \begin{bmatrix} 0{,}6129 & 0{,}7193 & 0{,}6416 & 0{,}8619 & 0{,}5756 \\ 0{,}6286 & 0{,}7365 & 0{,}6581 & 0{,}8840 & 0{,}5893 \\ 0{,}7544 & 0{,}8040 & 0{,}7897 & 1{,}0608 & 0{,}7072 \\ 0{,}7701 & 0{,}9024 & 0{,}8062 & 1{,}0829 & 0{,}7219 \end{bmatrix} \tag{14}$$

Partindo para o contexto computacional, deve-se utilizar o algoritmo implementado anteriormente para a realização do treinamento da rede. Primeiramente, todos os pesos sinápticos estão definidos aleatoriamente com o valor zero e a taxa de aprendizado com o valor η = 0,05. Além dessas atribuições, para o número máximo de épocas foi estabelecido 30 ciclos (*MaxEpocas = 30*).

Além do mais, foi definida a posição (W_0) do vetor como sendo o limiar de ativação da rede (*Bias*). Assim, ao decorrer da execução do treinamento do *Perceptron*, os pesos devem ser ajustados se a saída obtida for diferente da saída desejada. Com os parâmetros já estabelecidos, a rede pode ser treinada com os dados estipulados. O resultado de cada passo do treinamento pode ser conferido na Figura 17 que mostra o valor final com os pesos já ajustados e também de todos os ajustes de pesos feitos no decorrer da execução da aplicação.

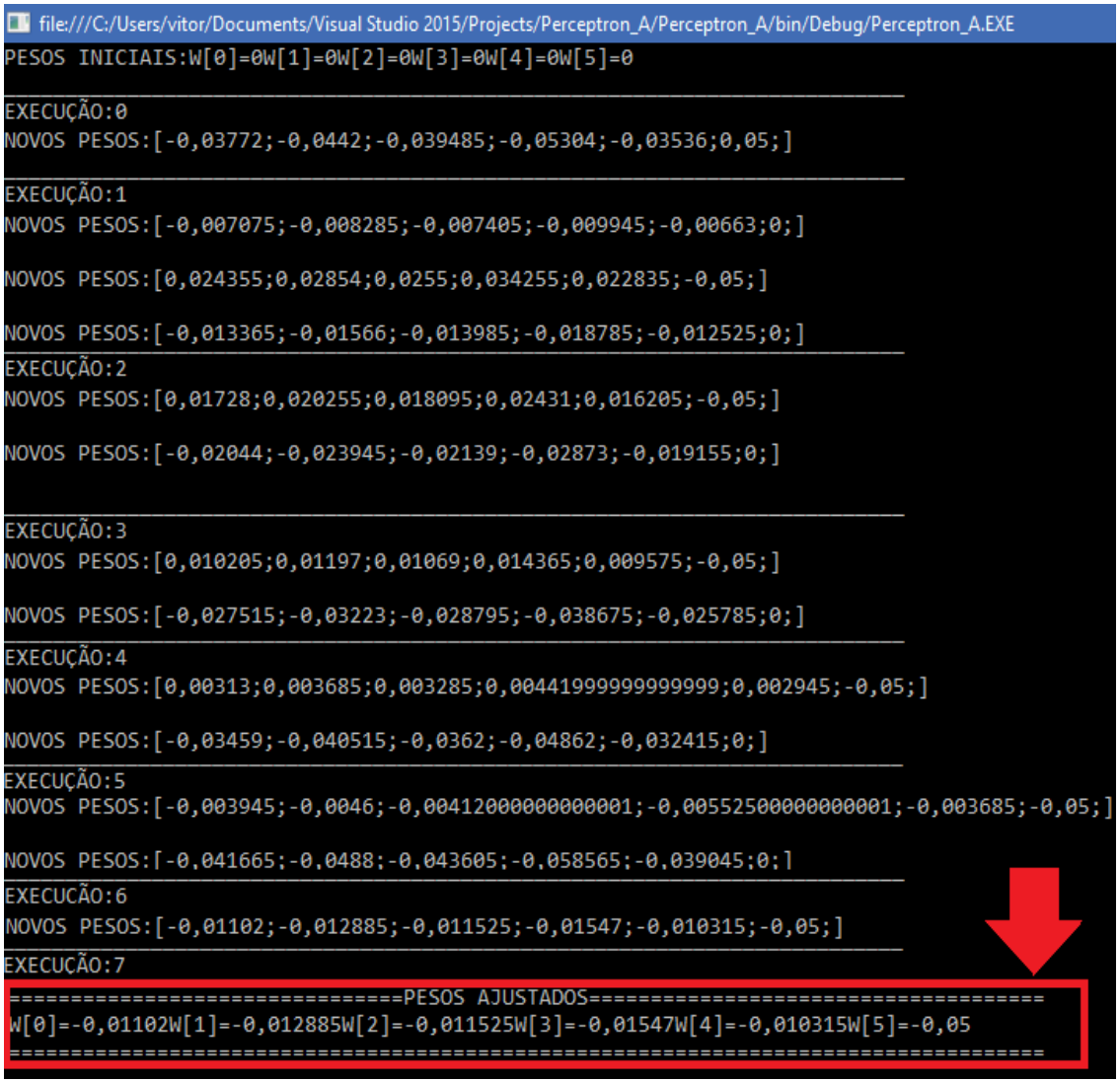


Figura 17 - Resultados das amostras

Fonte: Elaborado pelo autor

Agora, com o treinamento finalizado e os dados finais obtidos, pode-se passar para a etapa de classificação de amostras. Os dados contidos na Figura 17 demonstram detalhadamente cada execução e os novos ajustes de pesos que são estabelecidos com base nas equações vistas, onde, o algoritmo tem sua condição de parada satisfeita quando os pesos estão devidamente ajustados e o erro é inexistente (indicado em vermelho). Para testar a confiabilidade dos resultados exibidos, as amostras seguintes possuem valores muito aproximados dos dados reais da tabela de frutas congeladas. Baseado neste contexto, A Tabela 10 ilustra as amostras

criadas para serem classificadas pelo *Perceptron*. Além disso, o termo *Bias* está destacado em vermelho, pois é um valor indispensável e não deve ser esquecido nas amostras para classificação na rede já treinada.

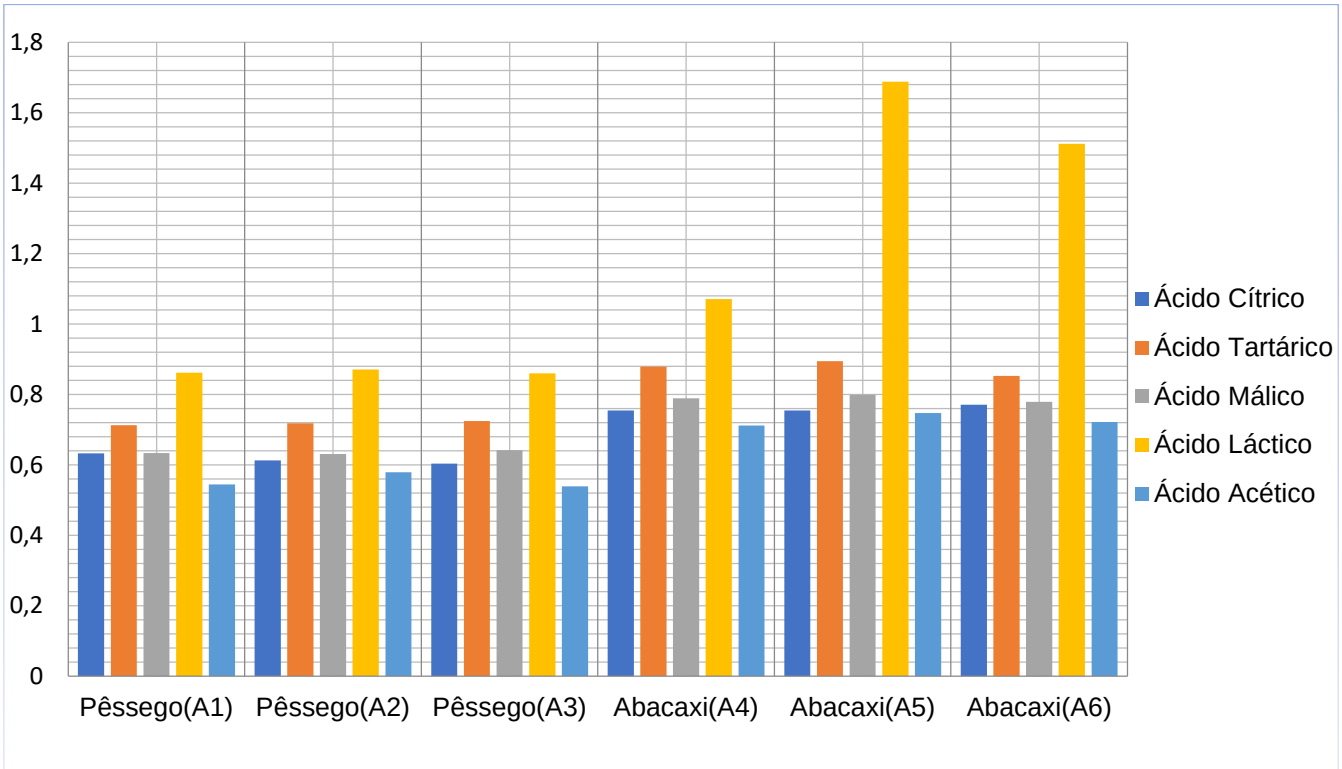
Tabela 9 - Amostras para classificação

RESULTADOS ESPERADOS	Ácido Cítrico	Ácido Tartárico	Ácido Málico	Ácido Láctico	Ácido Acético	*BIAS*
Pêssego	0,6329g	0,7133g	0,6336g	0,8619g	0,5446g	-1
Pêssego	0,6127g	0,7180g	0,6315g	0,8715g	0,5789g	-1
Pêssego	0,6034g	0,7245g	0,6422g	0,8599g	0,5389g	-1
Abacaxi	0,7544g	0,8790g	0,7891g	1,0712g	0,7122g	-1
Abacaxi	0,7544g	0,8945g	0,7997g	1,688g	0,7472g	-1
Abacaxi	0,7712g	0,8530g	0,7791g	1,5124g	0,7222g	-1

Fonte: Elaborado pelo autor

Para melhor entendimento e visualização dos dados apresentados, o Gráfico 2 é a representação dos dados contidos na Tabela 10, onde, cada uma das amostras de frutas *Ai = {A1, A2, ... , An }* está relacionada com os valores e tipos de ácidos encontrados em cada uma delas. Analisando o Gráfico 2 mais detalhadamente é possível estabelecer alguns padrões, como por exemplo a presença de uma maior quantidade de ácido láctico nas amostras referentes a fruta abacaxi do que na fruta pêssego. Com isso, uma rede neural observa tais padrões encontrados em uma variedade de dados de treinamento e os classifica. Sendo assim, fazendo a escolha correta de uma RNA e utilizando os parâmetros corretos para a sua configuração, é mais do que notável as vantagens que tal mecanismo pode entregar para campos relacionados a análise dados.

Gráfico 2 - Amostras para classificação



Fonte: Elaborado pelo autor

Com base nessas informações, esses dados devem ser introduzidos na função de classificação de amostras com o objetivo de apresentar a classificação correta para cada uma das amostras. Deste modo, a função de classificação tem por objetivo separar cada uma dos exemplos nas suas classes pertencentes com base nos valores ajustados dos pesos sinápticos no treinamento da rede.

```
_____________________________________________________________________
================================PESOS AJUSTADOS=================================
 W[0] = -0,01102 W[1] = -0,012885 W[2] = -0,011525 W[3] = -0,01547 W[4] = -0,010315 W[5] = -0,05
================================================================================
_____________________CLASSIFICAÇÃO DE AMOSTRAS_____________________
      A amostra[1] pertence a CLASSE PESSEGO
      A amostra[2] pertence a CLASSE PESSEGO
      A amostra[3] pertence a CLASSE PESSEGO
      A amostra[4] pertence a CLASSE ABACAXI
      A amostra[5] pertence a CLASSE ABACAXI
      A amostra[6] pertence a CLASSE ABACAXI
```

Figura 18 - Classificação das amostras

Fonte: Elaborado pelo autor

Por fim, o resultado de todas as amostras (Conforme apresentado na Figura 18) condiz com o que foi apresentado na tabela anterior. Logo, cada uma das amostras foi separada em suas devidas classes. Sendo assim, pode-se dizer que o treinamento da rede foi realizado com sucesso para um número pequeno de amostras. Além disso, a rede pode ser treinada para resolver outros problemas de classificação, porém alguns problemas podem não ser resolvidos por uma rede SLP. Com isso, o capítulo seguinte trata um pouco sobre as redes MLP, a fim de estabelecer um maior conhecimento e uma comparação com a rede abordada no presente capítulo.

4 REDES NEURAIS *PERCEPTRON – MULTILAYER*

Com base no que já foi visto anteriormente, este capítulo trata o funcionamento do *Perceptron* multicamadas de forma a compreender sua aplicação na resolução de problemas de reconhecimento de padrões. Primeiramente, o principal objetivo do presente capítulo é a abordagem de conceitos e comparações com o *Perceptron* de camada única, cintando suas vantagens e desvantagens no que diz respeito a sua complexidade de implementação e o seu funcionamento no geral.

4.1 ESTRUTURA DAS REDES MULTICAMADAS

As redes *Perceptron Multilayer* são caracterizadas pela presença de pelo menos uma camada intermediária (escondida) em sua estrutura neural, situada entre a camada de entrada e a sua respectiva camada de saída [20]. Com isso, tais redes possuem a vantagem de conseguirem realizar a classificação de mais de duas classes distintas e também resolver problemas não linearmente separáveis, diferentemente do *Perceptron Single-Layer*.

De acordo com os conceitos sobre a arquitetura das redes neurais apresentados no capítulo 2, as redes *Perceptron* de múltiplas camadas estão introduzidas dentro da arquitetura do tipo *feedforward*. Além disso, as unidades de processamento dessas redes (neurônios) estão interligadas um com os outros, formando as camadas neurais ilustrada na Figura 19.

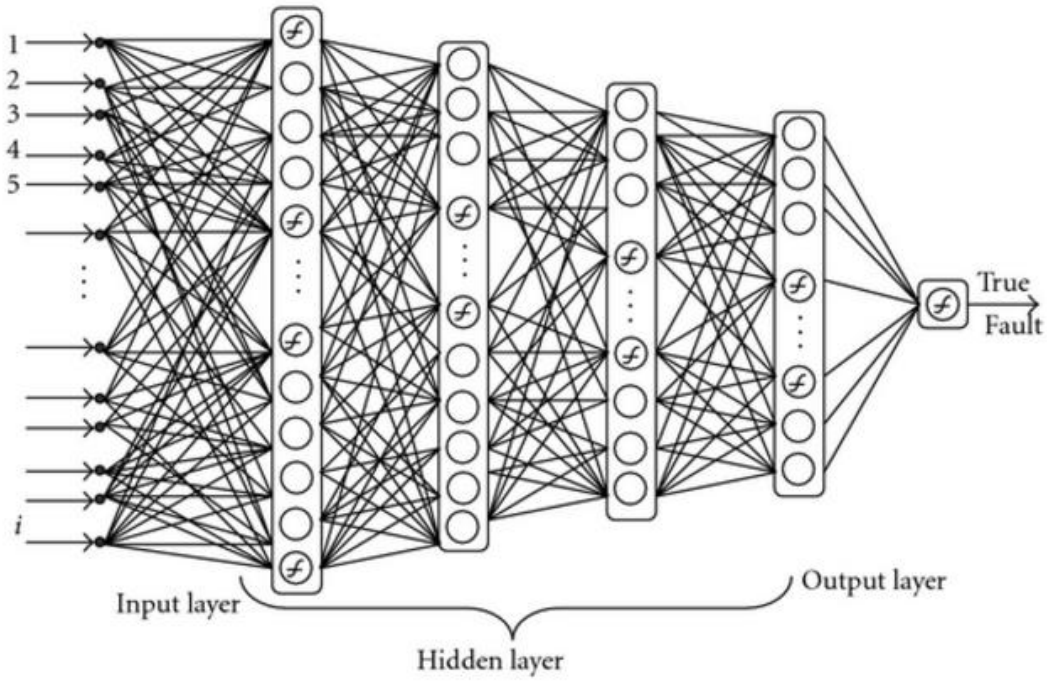


Figura 19 - Rede Neural Multicamada [2]

Como pode se observar na Figura 19, todo o fluxo de informações se inicia nas camadas de entrada (*Input Layer*) e percorrem por todas as camadas neurais ocultas (*Hidden Layer*) até terminarem nas camadas de saída (*Output Layer*). As redes *Perceptron Multilayer* também são caracterizadas pelas elevadas possibilidades de aplicações em diversos tipos de problemas, sendo considerada uma das arquiteturas mais versáteis quanto à aplicabilidade [17].

Outra característica importante dessas redes está na possibilidade de acrescentar mais de um neurônio na camada de saída, onde cada um desses neurônios artificiais é responsável por representar uma das suas respectivas saídas na resolução de um dado problema. Além desse aspecto, o processo de treinamento deste tipo de rede é diferente das redes com apenas uma camada, pois é necessário utilizar o chamado algoritmo de retropropagação para ajuste de pesos das camadas intermediárias no qual pode ser visto na seção a seguir.

4.2 CAMADAS ESCONDIDAS DA REDE MLP

As redes MLP são compostas de diversas camadas em sua estrutura que se ligam uma com as outras. No geral, tais redes são compostas de uma camada de entrada, várias camadas ocultas e uma camada de saída como já abordado. É importante ressaltar que cada camada possui certa quantidade de neurônios, que por sua vez devem ser definidos cuidadosamente para que não haja qualquer interferência que dificulte o treinamento da rede.

Primeiramente, não há um valor pré-definido para a quantidade de neurônios em uma camada, mas deve-se considerar a seguinte situação. Se uma camada escondida possuir um número muito pequeno de unidades processador (neurônios), então a rede estará limitada e não poderá trabalhar com dados complexos, tendo como consequência uma baixa generalização. Conforme [3], pode-se definir formalmente a generalização em aprendizagem de máquina da seguinte maneira:

Generalização refere-se a quão bem os conceitos aprendidos por um modelo de aprendizagem de máquina se aplicam a exemplos específicos não vistos pelo modelo quando estava aprendendo.

Ainda nesta questão, uma quantidade exagerada de neurônios pode provocar problemas no processo de treinamento da rede tendo como consequência uma demora significativa no treinamento e a não convergência da rede com os dados nela apresentados. Portanto, o número de neurônios não pode ser nem tão grande e nem tão pequeno.

Além disso, a definição de um número de camadas escondidas na rede MLP é necessária. Em geral, para a maioria dos problemas que tais redes resolvem uma ou duas camadas escondidas são suficientes. Com isso, acrescentar muitas camadas em uma rede pode trazer efeitos negativos, desde um longo tempo para treinamento da rede e erros no treinamento.

4.3 O ALGORITMO *BACKPROPAGATION*

Basicamente o treinamento de uma rede *Perceptron* com múltiplas camadas é dado pelo algoritmo conhecido como *backpropagation* (retropropagação). A execução do processo de retropropagação de erros é muito importante, pois é responsável por organizar os neurônios que estão presentes nas camadas ocultas ou intermediárias da rede. Com base em [17], o método de *backpropagation* é dividido em duas etapas, ou seja:

- **Fase 1 - Propagação adiante (*forward*):** Nesta etapa os sinais de entrada na rede percorrem o caminho passando por todas as camadas neurais até o processo gerador das respectivas saídas do *Perceptron*. Outro fato importante é que na execução deste processo os pesos sinápticos *W* e o limiar de ativação (*Bias*) não sofrem qualquer alteração em seus valores. Para um melhor entendimento, a Figura 20 demonstra a propagação das entradas pela rede, além das funções matemáticas que serão discutidas em tópicos futuros.

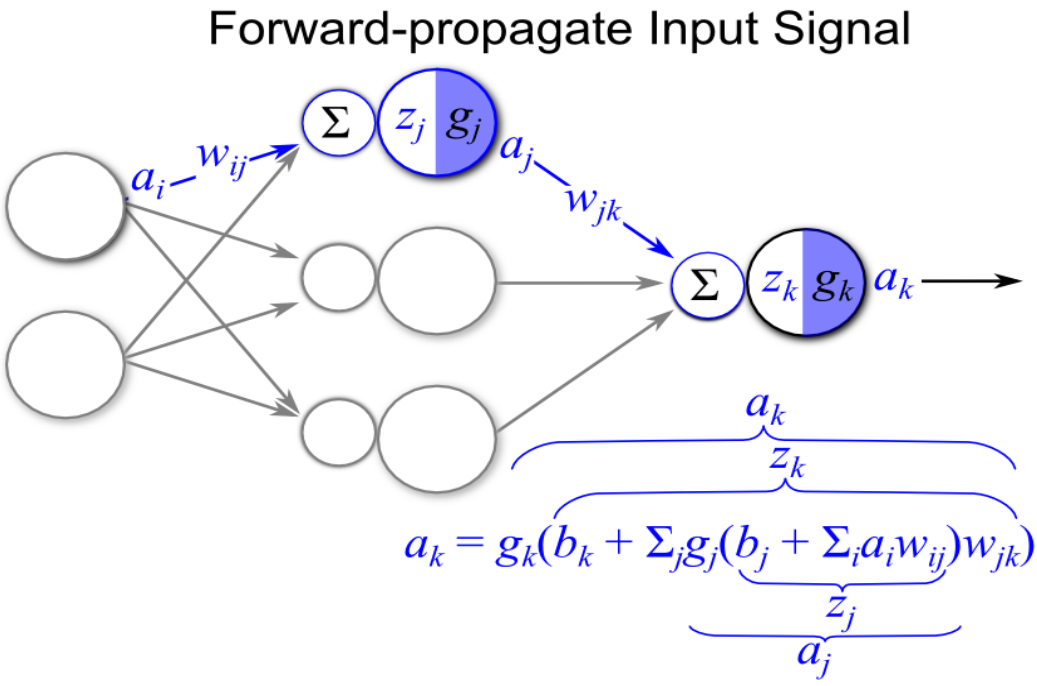


Figura 20 - Propagação dos sinais de entrada – Forward [22]

- **Fase 2 - Propagação reversa (*backward*):** Esta etapa é realizada após o processo *forward* e tem o objetivo de realizar a atualização dos pesos sinápticos e limiares dos neurônios presentes em todas as camadas da rede. Tal processo está baseado nos erros obtidos entre os resultados esperados e as geradas pelas saídas dos neurônios. A Figura 21 demonstra visualmente o sentido de propagação dos sinais de erro na rede, onde se inicia nas camadas de saída e percorrem a rede até as camadas de entrada.

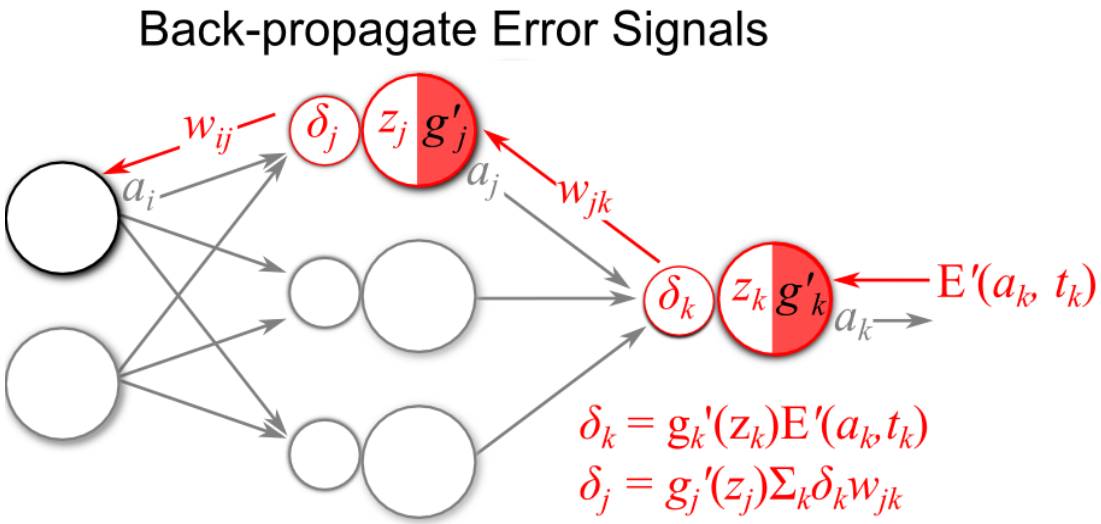


Figura 21 - Sinais de erro – backward [22]

Logo, para detalhar o funcionamento de tal método, considere uma rede neural com uma camada de entrada, duas camadas intermediárias e uma camada de saída dispostas como denota a Figura 22. O exemplo a seguir ilustra cada passo que o algoritmo realiza no treinamento da rede neural MLP.

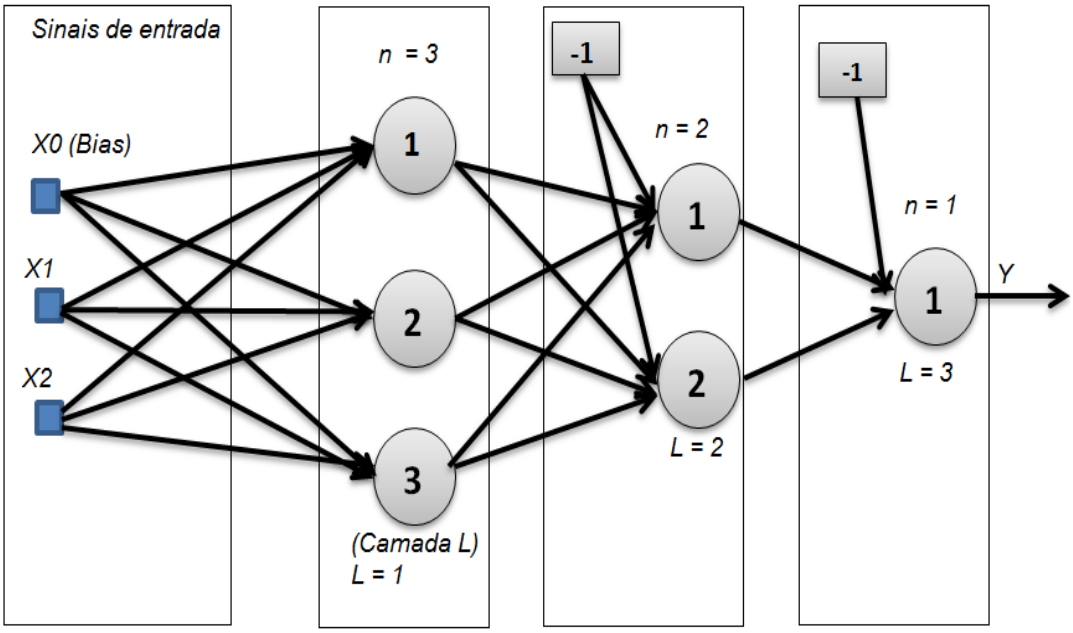


Figura 22 - Rede neural MLP

Fonte: Adaptado de [20]

Utilizando a rede neural apresentada na Figura 22, os processos realizados na execução do *forward* tendo três sinais de entrada, sendo o termo x_0 correspondente ao limiar (*Bias*) são observados na Figura 23. Além disso, leve em conta que a primeira camada possui três neurônios (*n = 3*), a segunda possui dois neurônios (*n = 2*) e a camada de saída apenas um neurônio (*n = 1*).

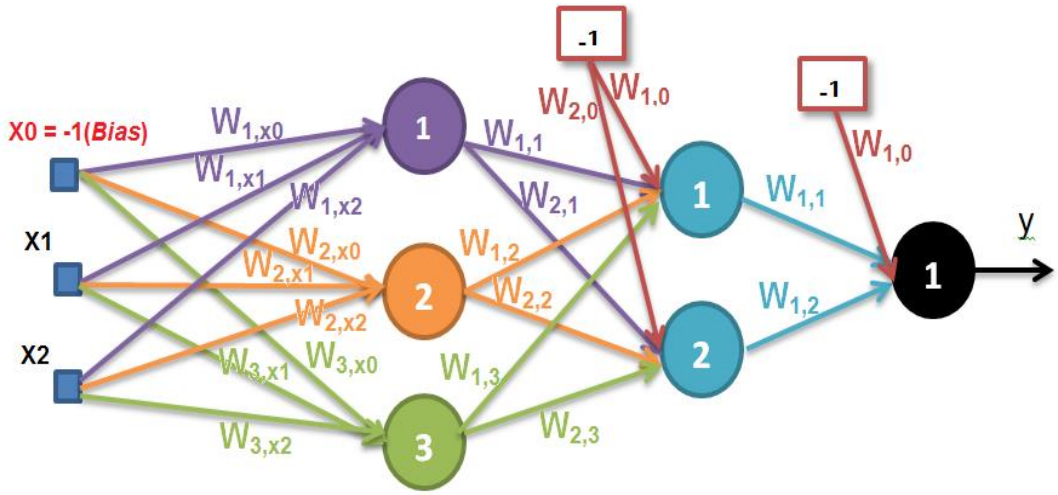


Figura 23 - Processo de funcionamento forward

Fonte: Adaptado de [20]

Com base na Figura 23, considere que cada neurônio presente na rede é definido por *j*, onde, cada um deles está contido em uma dada camada neural *L*. Sendo assim, cada neurônio utiliza uma função de ativação representada por *g(l)* e que neste caso é igual para todos os neurônios pertencentes a uma mesma camada neural. Para uma rede multicamadas, faz-se o uso de funções de ativação não lineares como, por exemplo, a função sigmóide ou a tangente hiperbólica.

Sendo assim, a variável $W_{j,i}$ representa a matriz de pesos sinápticos correspondentes as entradas, onde o valor de cada peso está conectado com o *j*-ésimo neurônio de uma camada *L* e ao *i*-ésimo neurônio dentro desta mesma camada. Desta forma, a primeira etapa para a execução do *backpropagation* é dado pelo conjunto de fórmulas a seguir:

$$l_j^{(L)} = \sum_{i=0}^{n} W_{j,i}^{(L)} \cdot X_i \Leftrightarrow \left(W_{j,0}^{(L)} \cdot X_0 + W_{j,1}^{(L)} \cdot X_1 + \cdots + W_{j,n}^{(L)} \cdot X_n \right) \tag{15}$$

Além disso, a saída dos neurônios *y* com relação à camada *L* está definida por:

$$y_j^{(L)} = g(l_j^{(L)}) \tag{16}$$

Logo, considerando a Figura 24, o cálculo da saída final (camada de saída) é denotado da seguinte maneira:

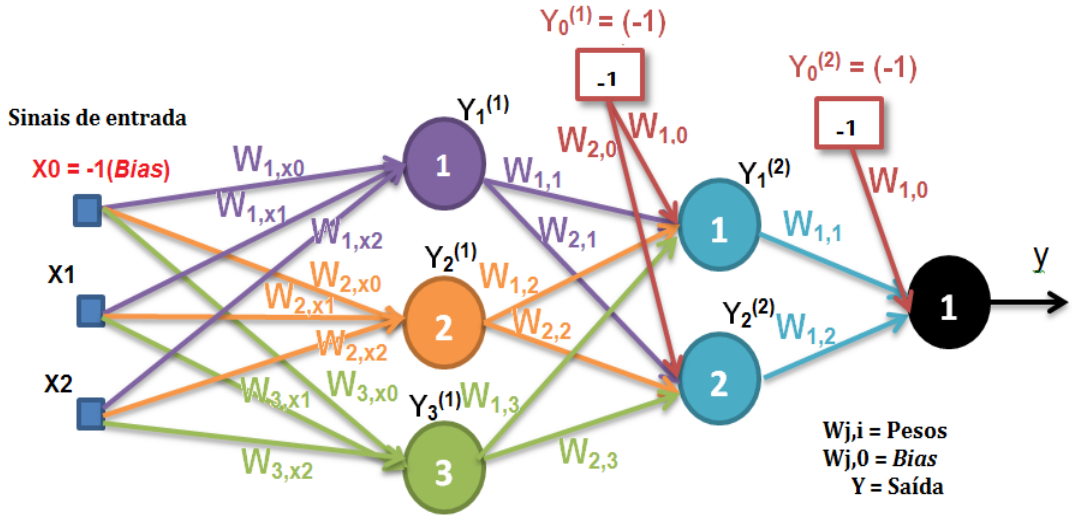


Figura 24 - Etapa de execução Algoritmo forward

Fonte: Adaptado de [20]

- **Camada intermediária 1 (Três neurônios):**

$$Y_1^{(1)} = g\left(W_{1,x0}^{(1)} \cdot X_0 + W_{1,x1}^{(1)} \cdot X_1 + W_{1,x2}^{(1)} \cdot X_2\right) \quad (17)$$

$$Y_2^{(1)} = g\left(W_{2,x0}^{(1)} \cdot X_0 + W_{2,x1}^{(1)} \cdot X_1 + W_{2,x2}^{(1)} \cdot X_2\right) \quad (18)$$

$$Y_3^{(1)} = g\left(W_{3,x0}^{(1)} \cdot X_0 + W_{3,x1}^{(1)} \cdot X_1 + W_{3,x2}^{(1)} \cdot X_2\right) \quad (19)$$

- **Camada intermediária 2 (Dois neurônios):**

$$Y_1^{(2)} = g\left(W_{1,0}^{(2)} \cdot Y_0^{(1)} + W_{1,1}^{(2)} \cdot Y_1^{(1)} + W_{1,2}^{(2)} \cdot Y_2^{(1)} + W_{1,3}^{(2)} \cdot Y_3^{(1)}\right) \quad (20)$$

$$Y_2^{(2)} = g\left(W_{2,0}^{(2)} \cdot Y_0^{(1)} + W_{2,1}^{(2)} \cdot Y_1^{(1)} + W_{2,2}^{(2)} \cdot Y_2^{(1)} + W_{2,3}^{(2)} \cdot Y_3^{(1)}\right) \quad (21)$$

- **Camada de saída (Um neurônio):**

$$Y = g\left(W_{1,0}^{(3)} \cdot Y_0^{(2)} + W_{1,1}^{(3)} \cdot Y_1^{(2)} + W_{1,2}^{(3)} \cdot Y_2^{(2)}\right) \quad (22)$$

Por fim, esta etapa do processo está concluída (Etapa de propagação adiante). Agora, deve-se realizar o calculo do sinal do erro de saída para que o algoritmo comece a etapa de propagação reversa ou *backward*. Neste passo, deve-se definir uma função representativa dos desvios (erros), cujo tem a função de obter o desvio existente entre uma resposta obtida pelo neurônio e a saída desejada *y*. O sinal de erro *δ* na saída neural de um neurônio é dado da seguinte maneira:

$$\delta = d_j - y_j \quad (23)$$

Onde, *δ* representa o sinal de erro;

d_j representa a saída desejada no *j*-ésimo neurônio de saída;

y_j representa a saída produzida no *j*-ésimo neurônio de saída.

Sendo assim, a função de erro quadrático é utilizada para medir o desempenho local associado aos resultados produzidos pelos neurônios de saída frente à referida amostra [20], isto é:

$$E(k) = \frac{1}{2}\sum_{j=1}^{n}(\delta)^2 \Leftrightarrow E(K) = \frac{1}{2}\sum_{j=1}^{n}(d_j(k) - y_j^{(L)}(k))^2 \quad (24)$$

Onde, *E(k)* é o erro quadrático considerando a *k*-ésima amostra de treinamento, enquanto δ é o sinal de erro obtido anteriormente.

Com base na equação, o valor médio do erro quadrático assumindo a existência de um conjunto para treinamento da rede constituído por *p* amostras é definido da seguinte forma:

$$E_{médio} = \frac{1}{p}\sum_{k=1}^{n}(E(K)) \tag{25}$$

Onde, *E(K)* é definido como a função do erro quadrático.

Estes foram os conceitos básicos para entender os primeiros passos para compreensão do algoritmo de retropropagação. Além disso, o aprendizado dessas redes é baseado no método do gradiente descendente que visa determinar um conjunto de pesos para minimizar o erro de forma que a cada execução os pesos são atualizados na direção que produzem a maior queda ao longo da superfície de erro. Com isso, a seção seguinte aborda detalhadamente cada etapa da execução dos ajustes de pesos sinápticos em uma rede MLP.

4.4 AJUSTE DE PESOS DE UMA REDE MLP

Baseado no que foi trabalhada anteriormente, esta seção tem por objetivo explicitar de forma clara o funcionamento do ajuste de pesos e do limiar de acordo com o algoritmo *backpropagation*. Como esta etapa é mais complexa do que as das redes SLP, os tópicos seguintes foram separados de acordo com [20], ou seja, os passos necessários para o ajuste de peso foram divididos segundo as camadas que constituem a rede MLP. A Figura 25 representa o sentido de propagação na fase *backward* do algoritmo, onde, a propagação ocorre de modo inverso visando o ajuste de peso de todas as camadas do MLP.

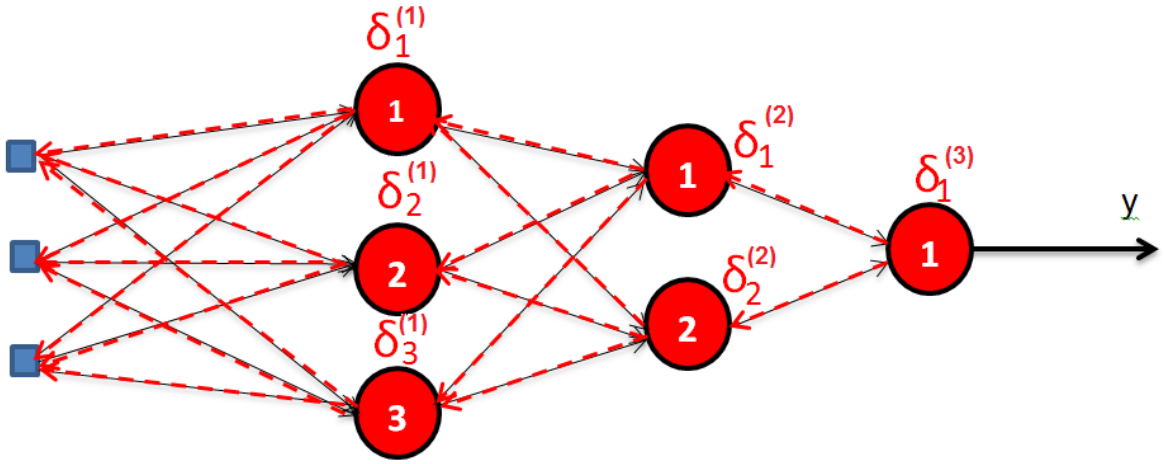


Figura 25 - Retropropagação na rede MLP

Fonte: Adaptado de [20]

O algoritmo de retropropagação é baseado em uma função matemática denominada gradiente descendente [20]. Tal função indica um ponto máximo de custo que uma função assume para alguns valores. Para exemplificar melhor o funcionamento do gradiente, observe a Figura 26. Além disso, considere um conjunto de pesos W_1 e W_2, onde, E indica o erro associado a tais pesos.

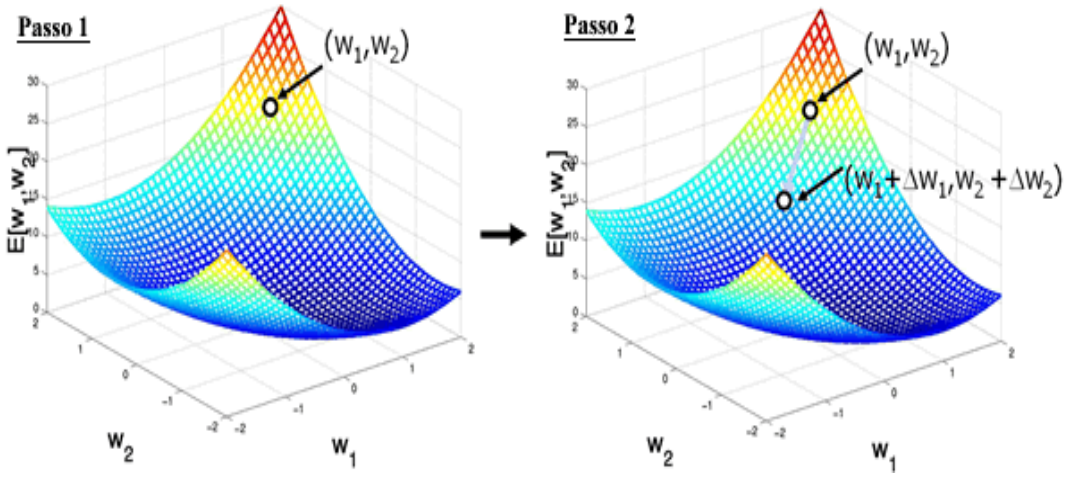


Figura 26 - Descida do gradiente [7]

No passo 1, os pesos encontram-se em uma região com maior erro possível (quanto mais próximo da região em vermelho, maior é o custo). No passo 2, com a execução do algoritmo e constantes ajustes na direção oposta em que *E* é máximo, os pesos movimentam-se para a região onde o custo do erro é menor (indicada em azul), ou seja, o objetivo do *backpropagation* é convergir (descer) os pesos até que seja encontrado o menor erro possível. Em resumo, o processo de aprendizado visa a minimização da existência do erro médio total, no qual, o ajuste de pesos é feito de acordo com os erros em cada respectiva saída. Assim, como passo inicial do treinamento deve-se empregar a uma correção para ∇E que é calculado através do conceito de derivada parcial como mostra a equação a seguir:

$$\nabla E^{(L)} = \frac{\partial E}{\partial W_{j,i}^{(L)}} = \frac{\partial E}{\partial Y_j^{(L)}} \cdot \frac{\partial Y_j^{(L)}}{\partial l_j^{(L)}} \cdot \frac{\partial l_j^{(L)}}{\partial W_{j,i}^{(L)}} \tag{26}$$

Onde ∇E é definida como a derivada parcial do erro associado a um peso $W_{j,i}$

Como a fase *backward* percorre a rede de modo inverso, ou seja, saída até a entrada, então os primeiros pesos a serem atualizados são os que correspondem aos neurônios da camada de saída. Deste modo, o ajuste dos pesos sinápticos da camada de saída é feito utilizando os seguintes cálculos:

- **Ajuste de pesos da camada de saída (*L* = 3)**

De acordo com a equação (28) e os conceitos nelas trabalhados, o primeiro passo é realizar o cálculo para ajuste de pesos da camada de saída, isto é:

$$\nabla E^{(3)} = \frac{\partial E}{\partial W_{ij}^{(3)}} = -\left(d_j - Y_j^{(3)}\right) \cdot g'(l_j^{(3)}) \cdot (Y_i^{(2)}) \tag{27}$$

Onde, o gradiente local δ é definido com base no *j*-ésimo neurônio e *g'(.)* é a derivada de primeira ordem da função de ativação [20], sendo assim:

$$\delta^{(L)} = \left(d_j - Y_j^{(L)}\right) \cdot g'\left(l_j^{(L)}\right) \tag{28}$$

Por fim, utilizando as notações vistas e considerando que η é a taxa de aprendizagem, então a expressão abaixo indica em notação computacional a equação para o ajuste de pesos da camada de saída da rede:

$$W_{j,i}{}^{(3)} = W_{j,i}{}^{(3)} + \eta \cdot (\delta_j^{(3)} \cdot Y_i^{(2)}) \tag{29}$$

- **Ajuste de pesos das camadas intermediárias ou escondidas (*L* = 2):**

Como a rede utilizada como exemplo possui duas camadas escondidas, então o ajuste de pesos deve ser feito para todas essas camadas. Logo, este tópico refere-se à segunda camada intermediária (escondida) da rede. De acordo com as equações vistas anteriormente, nesta camada o cálculo é desenvolvido da seguinte maneira:

$$\nabla E^{(2)} = \frac{\partial E}{\partial W_{ij}^{(2)}} = \frac{\partial E}{\partial Y_j^{(2)}} \cdot \frac{\partial Y_j^{(2)}}{\partial l_j^{(2)}} \cdot \frac{\partial l_j^{(2)}}{\partial W_{ij}^{(2)}} \tag{30}$$

Já neste caso, o gradiente local é definido baseado na quantidade de neurônios dessa mesma camada e com os ajustes feitos na camada de saída, isto é:

$$\delta^{(2)} = \left(\sum_{k=1}^{n_3} \delta_k^{(3)} \cdot W_{k,j}^{(3)} \right) \cdot g'\left(l_j^{(2)}\right) \tag{31}$$

Onde, *n* representa os neurônios desta mesma camada.

Por fim, a notação para o ajuste de pesos da segunda camada neural (intermediária) é definida da seguinte maneira:

$$W_{j,i}{}^{(2)} = W_{j,i}{}^{(2)} + \eta \cdot (\delta_j^{(2)} \cdot Y_i^{(1)}) \tag{32}$$

- **Ajuste de pesos das camadas intermediárias ou escondidas (*L* = 1):**

Esta etapa é responsável por realizar a atualização dos pesos sinápticos provenientes da primeira camada intermediária que está ligada diretamente com os sinais de entrada da rede. Sendo assim, o cálculo é similar ao da segunda camada, porém com alguns pequenos detalhes como mostra a seguir:

$$\nabla E^{(1)} = \frac{\partial E}{\partial W_{ij}^{(1)}} = \frac{\partial E}{\partial Y_j^{(1)}} \cdot \frac{\partial Y_j^{(1)}}{\partial l_j^{(1)}} \cdot \frac{\partial l_j^{(1)}}{\partial W_{ij}^{(1)}} \tag{33}$$

Já neste caso, o gradiente local é definido baseado na quantidade de neurônios dessa mesma camada e com os ajustes feitos na segunda camada intermediária da rede. Com base nisso, tem-se que:

$$\delta^{(1)} = \left(\sum_{k=1}^{n_2} \delta_k^{(2)} \cdot W_{k,j}^{(2)} \right) \cdot g'\left(l_j^{(1)}\right) \tag{34}$$

Finalmente, considerando x_i como a variável referente aos sinais de entrada, a notação para o ajuste de pesos da segunda camada neural (intermediária) é definida da seguinte maneira:

$$W_{j,i}{}^{(1)} = W_{j,i}{}^{(1)} + \eta \cdot (\delta_j^{(1)} \cdot x_i) \tag{35}$$

4.5 TREINAMENTO DA REDE *PERCEPTRON MULTILAYER*

Os tópicos a seguir têm por objetivo apresentar passo a passo de como funciona o treinamento de uma rede MLP com base nos conceitos e cálculos apresentados. Com isso, pode-se resumir o processo de treinamento de uma rede *Perceptron* multicamadas utilizando o algoritmo *backpropagation* da seguinte maneira [20]:

- **ETAPA 1 – *FORWARD:*** Devem-se acrescentar os sinais de entrada na rede e também realizar a inicialização dos pesos *W* e do limiar (*Bias*) com valores aleatórios. O ideal é que a representação dos pesos e entradas sejam inseridas na rede através de uma matriz.
- **ETAPA 2 – *FORWARD:*** Obter as saídas $Y_j^{(L)}$ para todas as camadas neurais *L* correspondentes ao *j*-ésimo neurônio conforme as equações previamente vistas (Equação 12 e 13).

- **ETAPA 3 – *BACKWARD*:** Realizar o cálculo da derivada parcial do erro ∇E primeiramente na camada de saída.
- **ETAPA 4 – *BACKWARD*:** Repetir todo o procedimento anterior para todas as camadas existentes na rede MLP até a camada *L = 1*.
- **ETAPA 5 – *BACKWARD*:** Realizar a propagação de ∇E de volta a camada anterior e os pesos respectivos entre as duas camadas.
- **ETAPA 6 – *BACKWARD*:** O algoritmo encerra até que sua condição de parada seja satisfeita, ou seja, quando a rede estiver treinada. Desta forma, uma rede MLP é considerada totalmente treinada quando o erro quadrático médio entre duas épocas sucessivas for inferior à precisão, requerida ao problema a ser mapeado [20].
- **CRITÉRIO DE PARADA:** Como um critério de parada para a execução do método pode-se assumir algumas alternativas. Uma delas é utilizar um número de épocas ou ciclos que define a quantidade de vezes que um conjunto de treinamento é inserido na rede. É importante definir corretamente o valor dos ciclos da rede. Segundo [19], sugere-se que o valor esteja entre o intervalo 500 e 3.000 ciclos.

  Além disso, pode-se definir como critério de parada o erro quadrático médio, ou seja, o algoritmo deve convergir quando o erro quadrático médio entre duas épocas sucessivas for suficientemente pequeno [19], dizendo isso em notação matemática:

$$\left| E_M^{(ATUAL)} - E_M^{(ANTERIOR)} \right| \leq \varepsilon \tag{36}$$

Onde, E_M é o erro quadrático médio e ε denota a precisão requerida para o processo de convergência especificado em função do tipo de aplicação a ser mapeada pela rede. Conforme os conceitos vistos sobre o algoritmo *backpropagation*, o processo de treinamento de uma rede MLP pode ser descrita em pseudocódigo como visto nas tabelas a seguir. Para uma melhor compreensão das ideias, o algoritmo foi dividido em duas fases (*forward e backward*), como pode ser observada através Tabela 10.

Tabela 10 - Pseudocódigo (Fase *forward* e *Backward*)

```
1.  Procedimento Fase_Forward(entrada)
2.  entradas: entrada
3.
4.  variáveis globais : Wji {Pesos sinápticos},dk {Saídas desejadas}
5.                      L  {número de camada}, taxaAprendizado.
6.
7.  Início
8.      Para cada um dos pesos Wji faça
9.          Iniciar valores pequenos e aleatórios para todos os pesos Wji
10.     Fim-Para
11.
12.     Para cada camada L presente na rede faça
13.         Para cada neurônio j presente na camada L faça
14.             Determinar I_J (L), com base na equação(17)
15.             Y_J (L) ← g(I_J (L)), onde g(I) é definido como função de ativação
16.         Fim-para
17.     Fim-para
18.
19. FimAlgoritmo
```

```
1.  Procedimento Fase_Backward(entrada)
2.  entradas: entrada
3.
4.  variáveis globais: Wji {Pesos sinápticos} , dk {Saídas desejadas}
5.   L {número de camadas}, taxaAprendizado
6.
7.  Início
8.      Repetir todas as instruções
9.          Para i = L até a primeira camada(i = 1) faça
10.             Para cada neurônio j presente na camada L faça
11.                 Determinar δ_J(L), com base na fórmula do gradiente local
12.                 Ajustar peso W_J,I(L), com base na equação de ajuste de peso
13.             Fim-para
14.         Fim-para
15.     Até que critério de parada seja satisfeito.
16.
17. FimAlgoritmo
```

Fonte: Elaborado pelo autor

4.6 APLICAÇÃO PRÁTICA DO *PERCEPTRON MULTILAYER*

Com base nos conceitos vistos, esta seção tem por objetivo apresentar um exemplo prático utilizando os conhecimentos adquiridos em tópicos anteriores. Sendo assim, considere o problema das frutas congeladas, onde, desta vez as amostras apresentam três classes distintas. Logo, para que seja possível separar tais elementos é necessário o uso de redes neurais com múltiplas camadas. Para o problema em questão, foi definido um conjunto de amostras para realizar o treinamento do *Perceptron* MLP (Conforme pode ser visto na Tabela 11).

Tabela 11 – Frutas congeladas (Amostras de treinamento da rede).

Polpa	Ácido Cítrico	Ácido Tartárico	Ácido Málico	Ácido Láctico	Ácido Acético
Mamão	0,4477g	0,5247g	0,4687g	0,6296g	0,4197g
Mamão	0,4680g	0,5485g	0,4900g	0,6582g	0,4388g
Pêssego	0,6129g	0,7183g	0,6416g	0,8619g	0,5746g
Pêssego	0,6286g	0,7367g	0,6581g	0,8840g	0,5893g
Abacaxi	0,7544g	0,8840g	0,7897g	1,0608g	0,7072g
Abacaxi	0,7701g	0,9024g	0,8062g	1,0829g	0,7219g

Fonte: Congresso Brasileiro de Química, 2014.

O próximo passo é definir os parâmetros e a estrutura principal da rede. Sabe-se que uma rede MLP é constituída por diversos neurônios distribuídos entre as várias camadas neurais. Com isso, a estrutura das camadas neurais foi definida da seguinte maneira:

- **Camada de entrada:** É constituída de cinco neurônios ($n = 5$), pois possui 5 entradas correspondente aos tipos de ácidos presente em cada uma das frutas descritas.
- **Camada intermediária**: É constituída de três neurônios ($n = 3$).

- **Camada de saída**: É constituída de apenas um neurônio (*n = 1*), com três saídas possíveis (frutas). Conforme a estrutura descrita, a Figura 27 ilustra a configuração da rede neural em questão.

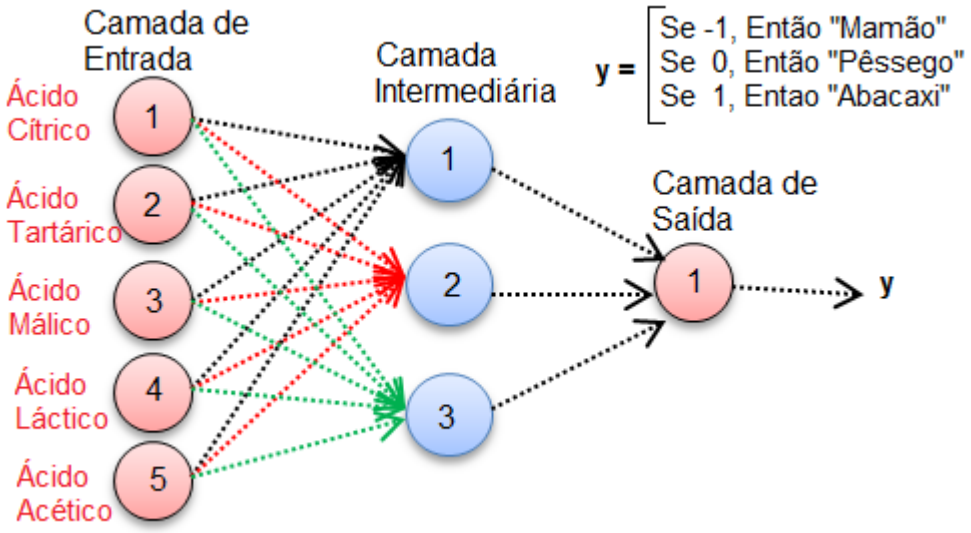


Figura 27- Configuração da rede MLP para o problema das frutas congeladas
Fonte: Elaborado pelo autor

Antes de iniciar os testes, foram definidos alguns parâmetros para esta rede neural. Para cada neurônio, foi utilizada a função de ativação do tipo bipolar sigmóide. Além disso, a taxa de aprendizado foi definida com o valor de $\eta = 0,05$. Para o critério de parada do algoritmo, foi escolhido um máximo de 5000 iterações (*MaxEpocas = 5000*). Vale ressaltar que o número máximo de épocas foi definido com base em execuções anteriores do código, onde, os resultados mostraram-se mais satisfatórios para tal valor. Logo, os resultados finais obtidos na fase de treinamento da rede neural podem ser observados conforme ilustra a Figura 28.

Figura 28 - Resultados do treinamento da rede

Fonte: Elaborado pelo autor

As informações mostradas na Figura 28 demonstram a minimização do erro total médio ao longo das iterações do algoritmo. Com isso, o objetivo principal do treinamento dessas redes é encontrar a região onde a presença do erro é mínima. Para melhor entendimento deste conceito, o Gráfico 3 fornece a representação gráfica do decaimento do erro total ao longo das iterações (épocas) do algoritmo.

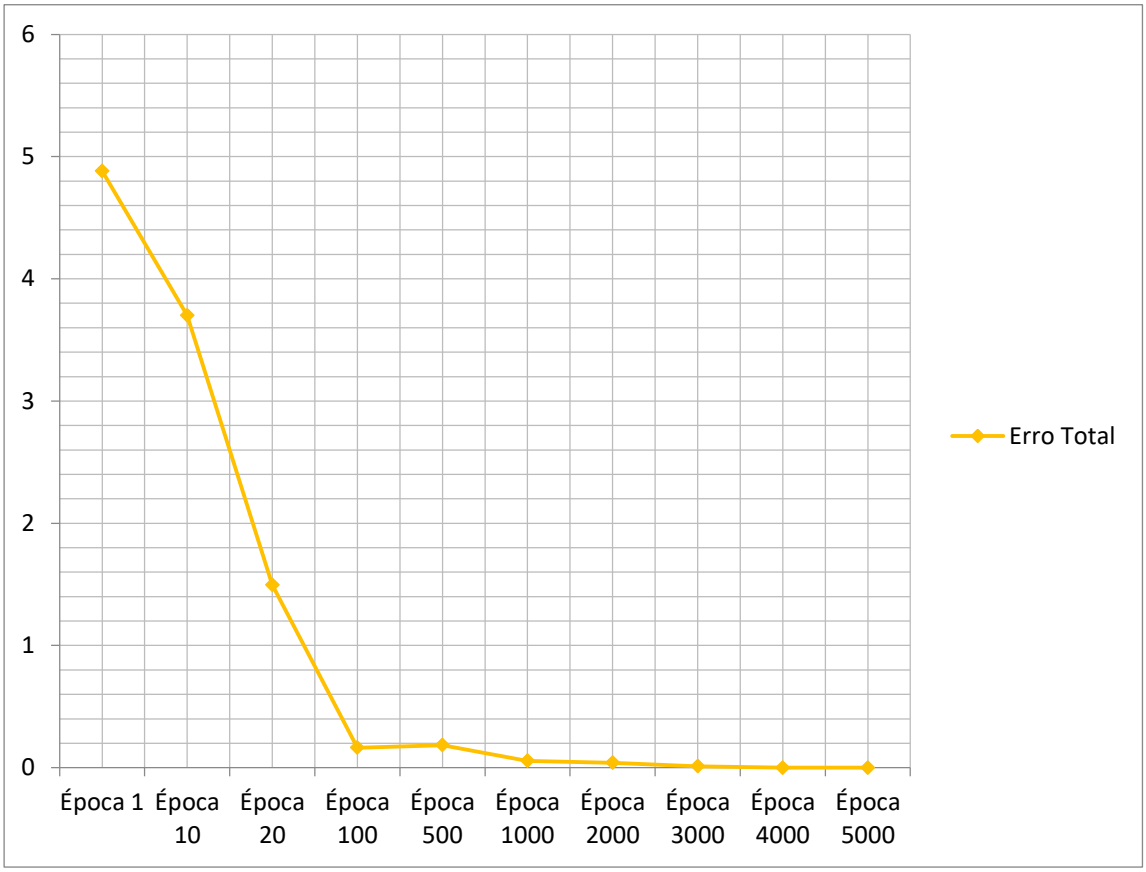


Gráfico 3 - Erro Médio Total

Fonte: Elaborado pelo autor

Para testar se a rede está devidamente treinada, foi elaborado um conjunto de amostras para que sejam classificadas conforme os dados de exemplo inseridos no processo de treinamento. Deste modo, se a rede conseguir separar corretamente as amostras em suas devidas classes, então o treinamento foi bem sucedido. Sendo assim, considere as seguintes amostras criadas para serem classificadas pelo *Perceptron Multilayer*.

Tabela 12 - Amostras para classificação (Rede MLP)

Amostra	RESULTADOS ESPERADOS	Ácido Cítrico	Ácido Tartárico	Ácido Málico	Ácido Láctico	Ácido Acético
0	Mamão	0,4680g	0,5485g	0,4900g	0,6582g	0,4388g
1	Mamão	0,4780g	0,5385g	0,4900g	0,6482	0,4288g
2	Pêssego	0,6286g	0,7497g	0,6581g	0,8840g	0,5893g
3	Pêssego	0,6186g	0,7267g	0,6381g	0,8530g	0,5893g
4	Pêssego	0,6286g	0,7467g	0,6499g	0,8740g	0,5793g
5	Abacaxi	0,7675g	0,8900g	0,7858g	1,0608g	0,7172g

Fonte: Elaborado pelo autor

Baseado nos dados vistos na Tabela 12, estas informações foram inseridas no algoritmo por meio de uma matriz que será classificada com base nos ácidos presentes em cada uma das frutas. Deste modo, a fase de classificação deve realizar a separação das frutas de acordo com o aprendizado da rede. Logo, as informações apresentadas podem ser observadas a seguir (ver Figura 29):

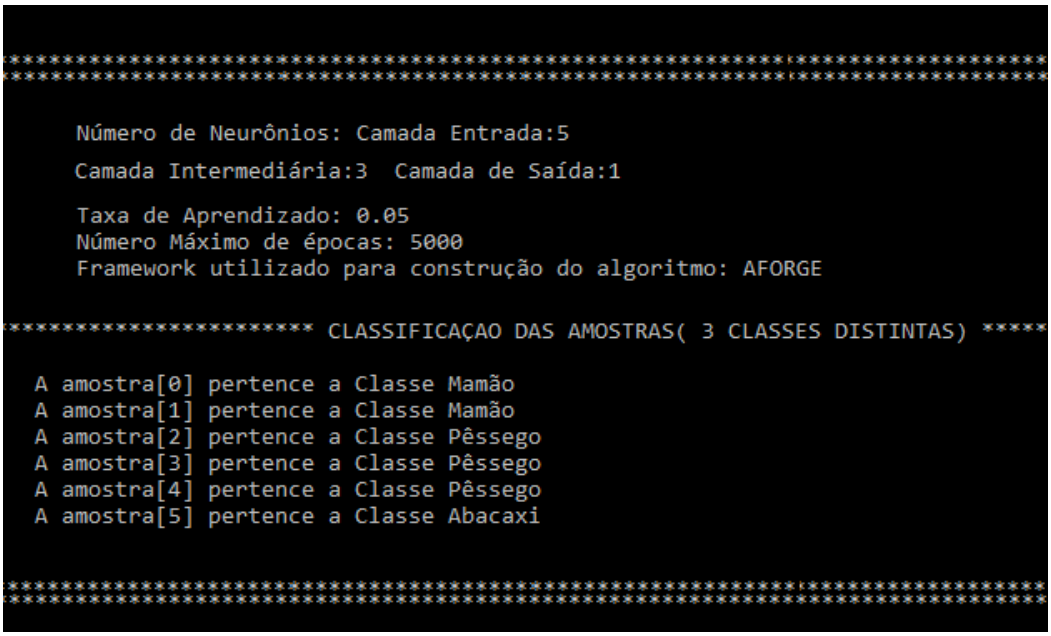


Figura 29 - Resultados encontrados pela rede MLP

Fonte: Elaborado pelo autor

Por fim, é possível observar que os resultados apresentados correspondem com resultados esperados (Conforme apresentado na Figura 29). Logo, pode-se dizer que a rede teve um aprendizado bem sucedido, possibilitando a separação das frutas corretamente com base nos parâmetros utilizados na elaboração do *Perceptron Multilayer*.

4.7 REDES MLP X REDES SLP

Com relação nos conceitos vistos, as redes neurais são mecanismos importantes e solucionam problemas que algoritmos convencionais são incapazes de lidar. Com isso, o *Perceptron* é uma rede neural poderosa principalmente em problemas que envolvem o reconhecimento de padrões e previsão de séries temporais, já abordadas em capítulos anteriores.

As redes SLP possuem uma estrutura muito mais simplificada do que as MLP. As redes de camada única possuem pouca complexidade na sua implementação, já que não há muitos parâmetros e o ajuste dos pesos sinápticos não é tão elaborado ou complexo. Porém, tais redes possuem uma atuação limitada por resolver problemas apenas linearmente separáveis.

Já as redes MLP possuem mais camadas neurais, o que permite uma maior variedade de classificações de padrões. O fato de terem camadas escondidas às possibilitam agir como detectores de características e também aumentar as suas respectivas para gerar mais resultados para a classificação de elementos. Em contrapartida, o uso de camadas escondidas dificulta a visualização da aprendizagem, ou seja, como a rede neural chegou a esta resposta final. Assim, esta não transparência na visualização de resultados pode impossibilitar a justificação do comportamento que uma MLP toma em uma determinada situação.

5 CONCLUSÕES E TRABALHOS FUTUROS

Na elaboração deste trabalho, foram observadas aplicações importantes para o assunto abordado. Com isso, um estudo mais aprofundado no que se diz respeito às redes neurais artificiais fornece mecanismos de extrema importância que podem ajudar na elaboração de soluções para problemas de otimização e aprendizado de máquina. Além disso, tais redes trazem muitos benefícios quando utilizadas com os parâmetros corretos.

Através das pesquisas e testes realizados foram percebidos que os parâmetros utilizados no treinamento da rede neural podem ser fatores decisivos, já que, valores muito inferiores ou superiores do estabelecidos podem gerar erros e dificuldades na busca por uma solução. Sendo assim, realizar uma pré-análise do problema é crucial para a seleção da RNA que mais se adequa ao que foi proposto.

Com base no assunto discutido, o *Perceptron* SLP fornece diversas vantagens, principalmente quando se trata de reconhecer padrões. Além disso, tal rede possui uma estrutura simples que facilita sua implementação e compreensão dos resultados obtidos pela mesma. Deste modo, esta rede se mostrou eficiente na análise e classificação de dados para o problema apresentado no presente trabalho. Para o *Perceptron* MLP, a pesquisa foi satisfatória para compreender o funcionamento e o algoritmo de treinamento que as envolve, além de ser um ponto inicial para entender quando e como devem ser utilizadas. Com base no problema abordado neste trabalho, a rede *Perceptron* MLP também se mostrou eficaz no reconhecimento de padrões envolvendo mais de duas classes para separação, que por sua vez não poderia ser resolvido utilizando a rede *Perceptron* SLP.

De um modo geral, as redes neurais são técnicas de alto potencial e adaptação, ou seja, podem ser modeladas para aplicações em inúmeras áreas do conhecimento. Portanto, este trabalho possibilitou adquirir um maior conhecimento sobre assunto discutido, além de suas vantagens e como realizar a implementação de tais redes para a resolução de problemas específicos.

Por fim, pretendo continuar estudos nas áreas ligadas a aprendizagem de máquina e aprimorar os conhecimentos com cursos dentro do campo de ciência de

dados. Além disso, no âmbito acadêmico tenho a intenção de realizar pós-graduação nos campos de Inteligência Artificial.

REFERÊNCIAS BIBLIOGRÁFICAS

1. BARBOSA, Anderson Henrique; FREITAS, Marcílio Sousa da Rocha; NEVES, Francisco de Assis das. **Confiabilidade estrutural utilizando o método de Monte Carlo e redes neurais**. Rem: Rev. Esc. Minas, Ouro Preto, v. 58, n. 3, p. 247-255, set. 2005. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0370-44672005000300011&lng=pt&nrm=iso>. Acesso em: 23 abr. 2017.

2. BOHLOULI, Reza; ROSTAMI, Babak; KEIGHOBADI, Jafar. ***Application of Neuro-Wavelet Algorithm in Ultrasonic-Phased Array Nondestructive Testing of Polyethylene Pipelines***. Journal Of Control Science And Engineering**,** [s.l.], v. 2012, p.1-9, 2012.

3. BROWNLEE, Jason. ***Overfitting and Underfitting With Machine Learning Algorithms*.** 2016. Machine Learning Mastery. Disponível em: <www.machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>. Acesso em: 07 abr. 2017.

4. CARDON, André; MÜLLER, Daniel Nehme. **Introdução Às Redes Neurais Artificiais.** 1994. 32 f. Monografia (Especialização) - Curso de Pós-graduação em Ciências da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 1994. Disponível em: <http://www.dainf.ct.utfpr.edu.br/~myriam/PastaWeb/RNGrad/tutorial_I.pdf>. Acesso em: 11 abr. 2017.

5. FERREIRA, Marco Antônio. **Noções de Redes Neurais Artificiais.** 2014. Universidade Tecnológica Federal do Paraná. Disponível em: <http://paginapessoal.utfpr.edu.br/mafinocchio/labsi-laboratorio-de-seguranca-e-iluminacao/redes-neurais-artificiais/NOCaO%20DE%20REDES%20NEURAIS%20ARTIFICIAIS.pdf/ >. Acesso em: 04 abr. 2017.

6. FIORIN, Daniel V. et al . **Aplicações de redes neurais e previsões de disponibilidade de recursos energéticos solares**. Rev. Bras. Ensino Fís., São Paulo, v. 33, n. 1, p. 01-20, Mar. 2011. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1806-11172011000100009&lng=en&nrm=iso>. Acesso em: 23 abr. 2017.

7. FREITAS, T. A. **Perceptrão**. Universidade Técnica de Lisboa - IST.Disponivel em:<http://web.tecnico.ulisboa.pt/ana.freitas/bioinformatics.ath.cx/bioinformatics.ath.cx/index79ff.html?id=112>. Acesso em: 02 Mar. 2017.
8. HAYKIN, Simon. **Neural Networks and Learning Machines.** 3. ed. Hamilton: Pearson, 2009. 938 p.

9. ISAAC, Mike; BOUDETTE, Neal E.. ***Ford to Invest $1 Billion in Artificial Intelligence Start-Up*.** 2017. The New York Times. Disponível em: <https://www.nytimes.com/2017/02/10/technology/ford-invests-billion-artificial-intelligence.html?_r=0>. Acesso em: 20 mar. 2017.

10. JUNQUEIRA, Luiz Carlos Uchoa; CARNEIRO, José. **Histologia básica.** 12. ed. Rio de Janeiro: Guanabara Koogan, 2013.

11. KOJIć, Nenad; RELJIN, Irini; RELJIN, Branimir. A Neural Networks-Based Hybrid Routing Protocol for Wireless Mesh Networks. **Sensors,** [s.l.], v. 12, n. 12, p.7548-7575, 7 jun. 2012.

12. MCCULLOCH, Warren S.; PITTS, Walter. ***A logical calculus of the ideas immanent in nervous activity***. Bulletin Of Mathematical Biology, [s.l.], v. 52, n. 1-2, p.99-115, jan. 1990.

13. MONTEIRO, L. SATIN, J. **Análise Físico-Química de polpas de frutas congeladas.** CBQ- Congresso Brasileiro de Química. Disponível em: <http://www.abq.org.br/cbq/2014/trabalhos/3/5495-18827.html>. Acesso em: 27 fev. 2017.

14. POZZER, Cesar Tadeu. **Aprendizado por Árvores de Decisão.** 2006. Disponível em: <http://www.usr.inf.ufsm.br/~pozzer/disciplinas/pj3d_decisionTrees.pdf>. Acesso em: 05 abr. 2017.

15. PUC-RIO; **Redes Neurais Artificiais.** Disponível em: <https://www.maxwell.vrac.puc-rio.br/7335/7335_5.PDF>. Acesso em: 28 fev. 2017.

16. RABELO, Prof. Ricardo José. **O neurônio artificial.** Disponível em: <http://www.gsigma.ufsc.br/~popov/aulas/rna/neuronio_artificial/index.html>. Acesso em: 27 fev. 2017.

17. RUSSELL, Stuart; NORVING, Peter. **Inteligência Artificial.** 3. ed. Rio de Janeiro: Elsevier, 2013. Tradução da Terceira Edição.

18. SENECHAL, Ana Carolina Le. **Análise e pré-processamento de dados utilizando técnicas de mineração de dados educacionais para o Moodle.** 2013. 120 f. Monografia (Graduação) - Curso de Ciências da Computação, Universidade Federal de Lavras, Lavras, 2013. Disponível em: <http://www.bcc.ufla.br/wp-content/uploads/2013/10/ANÁLISE-E-PRÉ-PROCESSAMENTO-DE-DADOS-UTILIZANDO-TÉCNICAS-DE-MINERAÇÃO-DE-DADOS-EDUCACIONAI-PARA-O-MOODLE.pdf>. Acesso em: 02 fev. 2017.

19. SILVA, Eugênio; OLIVEIRA, Anderson Canêdo de. **Dicas para a configuração de redes neurais.** Disponível em:

<http://equipe.nce.ufrj.br/thome/grad/nn/curso/..\mat_didatico\dicas_configuracao_rna.pdf>. Acesso em: 10 abr. 2017.

20. SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. **Redes Neurais Artificiais para engenharia e ciências aplicadas.** São Paulo: Artliber, 2010. 399 p.

21. SILVA, Saulo Rodrigues e; SCHIMIDT, Fernando. **Redução de variáveis de entrada de Redes Neurais Artificiais a partir de dados de análise de componentes principais na modelagem de oxigênio dissolvido**. Quím. Nova, São Paulo , v. 39, n. 3, p. 273-278, Abr. 2016 Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0100-40422016000300273&lng=en&nrm=iso>. Acesso em: 23 mar. 2017.

22. STANSBUR, Dustin. ***A Gentle Introduction to Artificial Neural Networks*.** 2014. Disponível em: <https://theclevermachine.wordpress.com/tag/backpropagation/>. Acesso em: 04 abr. 2017.

23. TABORDA, Anna. **Aula ao Vivo: Tecido e Sistema Nervoso.** 2015. Descomplica (Material de Aula). Disponível em: <https://descomplica.com.br/blog/biologia/aula-ao-vivo-tecido-e-sistema-nervoso/>. Acesso em: 03 fev. 2017.

24. Universidade de São Paulo, USP. **Redes Neurais Artificiais.** Disponível em: <http://conteudo.icmc.usp.br/pessoas/andre/research/neural/>. Acesso em: 02 mar. 2017.

APÊNDICE

APÊNDICE A – REDE *PERCEPTRON* SLP EM C# - VISUAL STUDIO 2015

```
1.  using System;
2.
3.  namespace perceptron
4.  {
5.  //Autor: Vítor Gama Lemos
6.  // Rede Perceptron SLP para análise de padrões em frutas congeladas
7.  // TCC - 2017.1
8.  public class PerceptronSingleLayer
9.  {
10.     private double[,] MatrizTreinar;
11.     private double[] saidaDesejada;
12.     private double CombinadorLinear;
13.     private double[] pesos;
14.     private double taxa_Aprendizado;
15.     private int MaxEpoca = 30;
16.     private static int contador = 0;
17. public PerceptronSingleLayer()
18. {
19.
20. MatrizTreinar = new double[,] {
21.             {0.6129, 0.7183, 0.6416, 0.8619, 0.5746},//Pessego
22.             { 0.6286, 0.7365, 0.6581, 0.8840, 0.5893},//Pessego
23.             { 0.7544, 0.8840, 0.7897, 1.0608, 0.7072},//Abacaxi
24.             { 0.7701, 0.9024, 0.8062, 1.0829, 0.7219}};//Abacaxi
25.
26.             pesos = new double[] { 0, 0, 0, 0, 0, 0 };
27.             saidaDesejada = new double[] { 1, 1, 0, 0 };
28.             taxa_Aprendizado = 0.05;
29.
30. }
31. private void Inicializar()
32. {
33.   Console.Write(" Pesos Iniciais :");
34.             for (int i = 0; i < pesos.Length; i++)
35.             {
36.                Console.Write(" W({0})={1} ,", i, pesos[i]);
37.             }
38.   Console.WriteLine();
39. }
40.
41. private static int g(double u)
42. {
43.     int valor = 0;
44.     valor = (u >= 0) ? 1 : -1;
45.     return valor;
46. }
47. private void TreinarRede()
48. {
49.     int epoca = 0;
50.     double saida = 0;
51.     string erro = "existe";
52.
53. while ((erro == "existe") || (epoca <= MaxEpoca))
54. {
55.     erro = "inexiste";
56.     Console.WriteLine("************************************");
57.     Console.WriteLine(" Época:{0}", epoca);
```

```
58.                 for (var i = 0; i < MatrizTreinar.GetLength(0); i++)
59.                 {
60.                 CombinadorLinear = 0;
61.                    for (int j = 0; j < pesos.Length; j++)
62.                    {
63.                       if (j != pesos.Length - 1)
64.                       {
65.                       CombinadorLinear += MatrizTreinar[i, j] * pesos[j];
66.                       }
67.                       else
68.                       {
69.                       CombinadorLinear += (-1) * pesos[j];
70.                       }
71.                    }
72.                    saida = (CombinadorLinear >= 0) ? 1 : 0;
73.                    if (saida != saidaDesejada[i])
74.                    {
75.                    AjustePeso(i, saida);
76.                    erro = "existe";
77.                    }
78.                }
79.             epoca++;
80.          }
81.    }
82.    public void AjustePeso(int i, double saida)
83.    {
84.    double erro = saidaDesejada[i] - saida;
85.    Console.WriteLine();
86.    Console.Write(" Novos Pesos :[");
87.
88.        for (int j = 0; j < pesos.Length; j++)
89.        {
90.         if (j != pesos.Length - 1)
91.         {
92.          pesos[j] = pesos[j] + (taxa_Aprendizado * erro * MatrizTreinar[i, j]);
93.         }
94.         else
95.         {
96.          pesos[j] = pesos[j] + (taxa_Aprendizado * erro * (-1)); //BIAS
97.         }
98.         Console.Write(" w({0})={1} ;", j, pesos[j]);
99.         }
100.   Console.Write("]");
101.   Console.WriteLine();
102.   }
103.
104.   class FaseClassificacao:PerceptronSingleLayer
105.   {
106.       public void ClassificarAmostra(double[] amostra, double[] pesos_ajustados)
107.       {
108.       contador++;
109.       double u = 0;
110.            for (int i = 0; i < amostra.Length; i++)
111.            {
112.               u += amostra[i] * pesos_ajustados[i];
113.            }
114.       double y = g(u);
115.
116.       string resposta = (y > 0) ? "A amostra({0}) pertence a classe Pêssego" :
117.                                   "A amostra({0}) pertence a classe Abacaxi";
118.       Console.WriteLine(resposta, contador);
```

```
119.        }
120.    public void MostrarPesosAjustados()
121.    {
122.        Console.WriteLine("===============PESOS AJUSTADOS=====================");
123.           for (int i = 0; i < pesos.Length; i++)
124.           {
125.           Console.Write(" W({0})={1} ,", i, pesos[i]);
126.           }
127.        Console.WriteLine();
128.        Console.Write("================================================");
129.        }
130.    }
131.
132.    public static void Main(string[] args)
133.    {
134.
135.        PerceptronSingleLayer Perceptron = new PerceptronSingleLayer();
136.        FaseClassificacao Classificar = new FaseClassificacao();
137.        Perceptron.Inicializar();
138.        Perceptron.TreinarRede();
139.        Classificar.MostrarPesosAjustados();
140.
141.        double[] amostra1 = { 0.6329, 0.7133, 0.6336, 0.8619, 0.5446, -1};//pessego
142.        double[] amostra2 = { 0.6127, 0.7180, 0.6315, 0.8715, 0.5789, -1};//pessego
143.        double[] amostra3 = { 0.6034, 0.7245, 0.6422, 0.8599, 0.5389, -1};//pessego
144.        double[] amostra4 = { 0.7544, 0.8790, 0.7891, 1.0712, 0.7122,-1}; //abacaxi
145.        double[] amostra5 = { 0.7544, 0.8945, 0.7997, 1.688, 0.7472, -1};//abacaxi
146.        double[] amostra6 = { 0.7712, 0.8530, 0.7791, 1.5124, 0.7222, -1};//abacaxi
147.
148.        Console.WriteLine("**********CLASSIFICAÇÃO DE AMOSTRAS*****************");
149.
150.        Classificar.ClassificarAmostra(amostra1, Perceptron.pesos);
151.        Classificar.ClassificarAmostra(amostra2, Perceptron.pesos);
152.        Classificar.ClassificarAmostra(amostra3, Perceptron.pesos);
153.        Classificar.ClassificarAmostra(amostra4, Perceptron.pesos);
154.        Classificar.ClassificarAmostra(amostra5, Perceptron.pesos);
155.        Classificar.ClassificarAmostra(amostra6, Perceptron.pesos);
156.
157.        Console.ReadKey();
158.    }
159.    }
160.    }
```

APÊNDICE B – REDE *PERCEPTRON* MLP EM C# - VISUAL STUDIO 2015

```
1.  using System;
2.  using System.Collections.Generic;
3.  using AForge;
4.  using AForge.Neuro;
5.  using AForge.Neuro.Learning;
6.
7.  namespace PerceptronMLP
8.  {
9.
10. //Autor: Vítor Gama Lemos
11. // Rede Perceptron MLP para análise de padrões em frutas congeladas
12. // TCC - 2017.1
13.
14. class PerceptronMLP
15. {
16.       private double[][] MatrizTreinar;
17.       private double[][] SaidasDesejadas;
18.       private int nNeuronioEntrada;
19.       private int nNeuronioIntermediaria;
20.       private int nNeuronioSaida;
21.       private int MaxEpoca;
22.
23. public PerceptronMLP()
24. {
25.        nNeuronioEntrada = 5;
26.        nNeuronioIntermediaria = 3;
27.        nNeuronioSaida = 1;
28.        MaxEpoca = 5000;
29.
30. MatrizTreinar = new double[][] {
31.             new double[]{ 0.4477, 0.5247, 0.4687, 0.6296, 0.4197}, //(Mamão)
32.             new double[]{ 0.4680, 0.5485, 0.4900, 0.6582, 0.4388}, //(Mamão)
33.             new double[]{ 0.6129, 0.7183, 0.6416, 0.8619, 0.5746}, //(Pêssego)
34.             new double[]{ 0.6286, 0.7467, 0.6581, 0.8840, 0.5893}, //(Pêssego)
35.             new double[]{ 0.77, 0.892, 0.7944, 1.0700, 0.7072}, //(Abacaxi)
36.             new double[]{ 0.7675, 0.89, 0.7858, 1.0608, 0.7172}, //(Abacaxi)
37. };
38.
39. SaidasDesejadas = new double[][]
40. {
41.        new double[]{ -1 }, // -1 = MAMÃO
42.        new double[]{ -1 }, // -1 = MAMÃO
43.        new double[]{ 0 }, // 0 = PÊSSEGO
44.        new double[]{ 0 }, // 0 = PÊSSEGO
45.        new double[]{ 1 }, // -1 = ABACAXI
46.        new double[]{ 1 }, // -1 = ABACAXI
47. };
48.
49.     TreinarRedeMLP();
50. }
51.
52. private void TreinarRedeMLP()
53. {
54.        ActivationNetwork PerceptronMLP = new ActivationNetwork(
55.                                          new BipolarSigmoidFunction(),
56.                                          nNeuronioEntrada,
57.                                          nNeuronioIntermediaria,
```

```
58.                                                 nNeuronioSaida);
59.
60.     BackPropagationLearning BackPropagation = new BackPropagationLearn-
        ing(PerceptronMLP);
61.
62.     BackPropagation.LearningRate = 0.5;
63.
64.     Console.WriteLine(" ********* FASE DE TREINAMENTO DA REDE ************** \n");
65.
66.     int ContEpoca = 1;
67.     double ErroInicial = 0;
68.     double ErroQuadratico = 0;
69.
70.      while(ContEpoca <= MaxEpoca)
71.       {
72.         ErroQuadratico = BackPropagation.RunEpoch(MatrizTreinar, SaidasDesejadas);
73.
74.          if(ContEpoca == 1)
75.          {
76.            ErroInicial = ErroQuadratico;
77.          }
78.
79.     Console.WriteLine(" Iteracao[{0}] Erro Quadratico:{1}", ContEpoca,
        Math.Round(ErroQuadratico,6));
80.
81.        ContEpoca = ContEpoca + 1;
82.     }
83.
84.     Console.WriteLine("********* REDE PERCEPTRON TREINADA ****************** \n");
85.
86.     Console.WriteLine("Número de Neurônios: C.Entrada:{0}   C.Intermediária:{1} Ca-
        mada de Saída:{2}", nNeuronioEntrada, nNeuronioIntermediaria, nNeuronioSaida);
87.
88.     Console.WriteLine("  Taxas = Erro inicial:{0}  Erro Final:{1}",
        Math.Round(ErroInicial,6), Math.Round(ErroQuadratico,6));
89.
90.       ClassificarAmostras(PerceptronMLP);
91.     }
92.
93.     private void ClassificarAmostras(ActivationNetwork PerceptronMLP)
94.     {
95.           double[][] AmostrasClassificacao =
96.           {
97.                new double[]{ 0.4680, 0.5485, 0.4900, 0.6582, 0.4388},//Mamao
98.                new double[]{ 0.4780, 0.5385, 0.4900, 0.6482, 0.4288},//Mamao
99.                new double[]{ 0.6286, 0.7467, 0.6581, 0.8840, 0.5893},//Pessego
100.               new double[]{ 0.6186, 0.7267, 0.6381, 0.8530, 0.5893},//Pessego
101.               new double[]{ 0.6286, 0.7467, 0.6499, 0.8740, 0.5793},//Pessego
102.               new double[]{ 0.7675, 0.89, 0.7858, 1.0608, 0.7172},//Abacaxi
103.          };
104.
105.    Console.WriteLine(" ****** CLASSIFICAÇÃO DAS AMOSTRAS(3 CLASSES DISTINTAS)
        ***** \n");
106.
107.    for (int i = 0; i < AmostrasClassificacao.Length; i++)
108.     {
109.
110.       double[] SaidasEsperadas = PerceptronMLP.Compute(AmostrasClassificacao[i]);
111.
112.       if (Math.Round(SaidasEsperadas[0]) < 0)
113.       {
```

```
114.         Console.WriteLine("    A amostra[{0}] pertence a Classe Mamão",i);
115.
116.       }else if ((Math.Round(SaidasEsperadas[0]) >= 0) &&
       (Math.Round(SaidasEsperadas[0]) < 1))
117.       {
118.         Console.WriteLine("    A amostra[{0}] pertence a Classe Pêssego", i);
119.
120.       }
121.         else if ((Math.Round(SaidasEsperadas[0])) >= 1)
122.         {
123.         Console.WriteLine("    A amostra[{0}] pertence a Classe Abacaxi", i);
124.         }
125.       }
126.    }
127.    }
```