



Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciência

Instituto de Matemática e Estatística

Raphael Silva de Figueiredo


**Perceptron Multicamadas com Backpropagation
para Análise de Crédito**

Rio de Janeiro

2019

Raphael Silva de Figueiredo

**Perceptron Multicamadas com Backpropagation
para Análise de Crédito**



Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Orientadora: Prof.^a Dra. Roseli Suzi Wedemann

Rio de Janeiro

2019

CATALOGAÇÃO NA FONTE
UERJ / REDE SIRIUS / BIBLIOTECA CTC-A

F475 Figueiredo, Raphael Silva de .
 Perceptron multicamadas com backpropagation para análise de
 crédito / Raphael Silva de Figueiredo. - 2019.
 84f. : il.

Orientadora: Roseli Suzi Wedemann
Dissertação (Mestrado em Ciências Computacionais) - Universidade
do Estado do Rio de Janeiro. Instituto de Matemática e Estatística.

1. Redes neurais - Teses. 2. Inteligência artificial - Teses. I.
Wedemann,, Roseli Suzi. II. Universidade do Estado do Rio de Janeiro.
Instituto de Matemática e Estatística. III. Título.

CDU 004.032.26

Patricia Bello Meijinhos CRB7/5217 - Bibliotecária responsável pela elaboração da ficha catalográfica

Autorizo, apenas para fins acadêmicos e científicos, a reprodução total ou parcial
desta dissertação, desde que citada a fonte.

Assinatura

Data

Raphael Silva de Figueiredo

**Perceptron Multicamadas com Backpropagation
para Análise de Crédito**

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Aprovada em 13 de fevereiro de 2019.

Banca Examinadora:

Prof.^a Dra. Roseli Suzi Wedemann (Orientadora)
Instituto de Matemática e Estatística - UERJ

Prof.^a Dra. Patrícia Nunes da Silva
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Luís Alfredo Vidal de Carvalho
Universidade Federal do Rio de Janeiro - UFRJ

Rio de Janeiro

2019

DEDICATÓRIA

Dedico esta dissertação e meus esforços aos meus pais Jorge e Arlete Figueiredo.

AGRADECIMENTOS

Primeiramente a Deus que permitiu que tudo isso acontecesse, ao longo de minha vida, e não somente nestes anos como estudante, mas que em todos os momentos é o maior mestre que alguém pode conhecer. Aos meus pais Jorge e Arlete Figueiredo que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa da minha vida. Ao programa de pós-graduação em ciências computacionais, e às pessoas com quem convivi nesse ambiente ao longo desses anos. A experiência de aprender na comunhão de amigos foram a melhor experiência da minha formação acadêmica. Agradeço a todos os professores que não mediram esforços para que obtivéssemos o melhor rendimento possível, seja com aulas, colóquios, e-mails, textos e etc. A minha orientadora Roseli Wedemann, pelo empenho e dedicação a elaboração desse trabalho. A Nereida Rezende, pela grande ajuda e por ter cedido da base de dados do BNDES. A Universidade do Estado do Rio de Janeiro, pela oportunidade de realizar o curso. E por fim a todos os amigos que fizeram parte da minha formação, em especial Vitor Tocci, Hannibal Escobar, Juliana de Souza, Lívia Medeiros e Leonardo Cavadas. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Só se pode alcançar um grande êxito
quando nos mantemos fiéis a nós mesmos.

Friedrich Nietzsche

RESUMO

FIGUEIREDO, R. S. *Perceptron Multicamadas com Backpropagation para análise de crédito*. 2019. 84 f. Dissertação (Mestrado em Ciências Computacionais) - Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2019.

O uso de ferramentas que utilizam inteligência artificial já está bastante presente em nossas vidas. Técnicas como a rede neuronal artificial e outras formas de aprendizado de máquina vêm crescendo ano após ano, nos ajudando em diversas áreas do conhecimento, como na medicina, astronomia, economia, engenharia, entre outras. Neste trabalho estudaremos a rede neuronal do tipo perceptron, tendo como foco o perceptron multicamadas com o algoritmo de backpropagation, mostrando passo a passo como ela foi desenvolvida, sua implementação em linguagem C e aplicando-a a um problema prático realizando a classificação de devedores do BNDES (Banco Nacional do Desenvolvimento Econômico e Social).

Palavras-chave: Rede neuronal. Perceptron multicamadas. Backpropagation.

ABSTRACT

FIGUEIREDO, R. S. *Multilayer Perceptron with Backpropagation for Credit Analysis*. 2019. 84 f. Dissertação (Mestrado em Ciências Computacionais) - Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2019.

The use of tools that use artificial intelligence is already quite present in our lives. Techniques such as the artificial neural network and other forms of machine learning have been growing year after year, helping us in many areas of knowledge, such as medicine, astronomy, economics, engineering, among others. In this work, we will study multilayer perceptron neural networks with the backpropagation learning algorithm, showing step by step how it was developed, its implementation in C language and applying it to a practical problem of classifying debtors of the BNDES (Banco Nacional do Desenvolvimento Econômico e Social).

Keywords: Neural network. Multilayer perceptron. Backpropagation.

LISTA DE ILUSTRAÇÕES

Figura 1 - Neurônio biológico e seus principais constituintes.	18
Figura 2 - Primeiro modelo de neurônio artificial criado por McCulloch e Pitts. . .	19
Figura 3 - Rede feedforward de apenas uma camada.	21
Figura 4 - Plano perpendicular ao vetor peso \mathbf{w} que separa os círculos sólidos $\zeta_i = +1$ dos abertos $\zeta_i = -1$	24
Figura 5 - Plano perpendicular ao vetor de pesos \mathbf{w} , onde, de acordo com a equação (7), todas as saídas devam estar do mesmo lado do plano. . . .	25
Figura 6 - Plano no espaço (ξ_1, ξ_2) , que separa o círculo sólido ($\zeta^\mu = +1$) dos círculos abertos ($\zeta^\mu = -1$).	27
Figura 7 - Problema com a não separabilidade linear. Caso onde não é possível separar os círculos sólidos ($\zeta^\mu = +1$) dos círculos abertos ($\zeta^\mu = -1$) com uma única reta.	28
Figura 8 - Superfície de erro no espaço de peso.	30
Figura 9 - Procedimento de descida do gradiente em uma superfície quadrática simples (as partes da esquerda e da direita são cópias da mesma superfície). Quatro trajetórias são mostradas, cada uma para 20 iterações a partir do círculo aberto. O valor mínimo é no ponto + e a elipse mostra o contorno de erro constante. A única diferença significativa entre as trajetórias é o valor de η , que foi de 0.02, 0.0476, 0.049 e 0.0505 da esquerda para a direita.	32
Figura 10 - Função sigmóide para diversos valores do parâmetro β , que determina a inclinação da função próximo do valor $h = 0$	35
Figura 11 - A função tangente hiperbólica definida no intervalo de -1 a $+1$	37
Figura 12 - Rede feedforward de duas camadas mostrando as notações para as unidades e conexões.	38
Figura 13 - Como a retropropagação dos erros é realizada em uma rede feedforward de três camadas.	40
Figura 14 - Comportamento da curva da regressão logística.	46
Figura 15 - Acurácia em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui uma camada intermediária com 4 neurônios.	55
Figura 16 - Sensibilidade em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui uma camada intermediária com 4 neurônios.	56

Figura 17 - Especificidade em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui uma camada intermediária com 4 neurônios.	56
Figura 18 - Acurácia em função do número de neurônios, para diferentes valores da taxa de aprendizagem. As curvas $\eta = 0.2$, $\eta = 0.3$ e $\eta = 0.4$ são semelhantes à curva $\eta = 0.1$	57
Figura 19 - Sensibilidade em função do número de neurônios, para diferentes valores da taxa de aprendizagem. As curvas $\eta = 0.2$ e $\eta = 0.3$ são semelhantes à curva $\eta = 0.1$	58
Figura 20 - Especificidade em função do número de neurônios, para diferentes valores da taxa de aprendizagem. As curvas $\eta = 0.2$, $\eta = 0.3$ e $\eta = 0.4$ são semelhantes à curva $\eta = 0.1$	58
Figura 21 - Acurácia em função da taxa de aprendizagem, para diferentes valores do número de neurônios na camada intermediária.	59
Figura 22 - Sensibilidade em função da taxa de aprendizagem, para diferentes valores do número de neurônios na camada intermediária.	60
Figura 23 - Especificidade em função da taxa de aprendizagem, para diferentes valores do número de neurônios na camada intermediária.	60
Figura 24 - Acurácia em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui duas camadas intermediárias, com 10 neurônios na primeira camada intermediária e 6 neurônios na segunda camada intermediária.	62
Figura 25 - Sensibilidade em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui duas camadas intermediárias, com 10 neurônios na primeira camada intermediária e 6 neurônios na segunda camada intermediária.	63
Figura 26 - Especificidade em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui duas camadas intermediárias, com 10 neurônios na primeira camada intermediária e 6 neurônios na segunda camada intermediária.	63
Figura 27 - Acurácia em função do número de neurônios da segunda camada intermediária, para diferentes valores da taxa de aprendizagem.	64
Figura 28 - Sensibilidade em função do número de neurônios da segunda camada intermediária, para diferentes valores da taxa de aprendizagem.	65
Figura 29 - Especificidade em função do número de neurônios da segunda camada intermediária, para diferentes valores da taxa de aprendizagem.	65
Figura 30 - Acurácia em função da taxa de aprendizagem, para diferentes valores do número de neurônios na segunda camada intermediária.	66

Figura 31 - Sensibilidade em função da taxa de aprendizagem, para diferentes valores do número de neurônios na segunda camada intermediária. . . .	67
Figura 32 - Especificidade em função da taxa de aprendizagem, para diferentes valores do número de neurônios na segunda camada intermediária. . . .	67

LISTA DE TABELAS

Tabela 1 - Tabela verdade que define a função AND, onde duas entradas resultam em uma saída positiva, somente se todas as entradas são positivas, caso contrário a saída é negativa.	26
Tabela 2 - Tabela verdade da função XOR, onde a saída será positiva, se exclusivamente uma ou a outra entrada for positiva.	27
Tabela 3 - Matriz de confusão ou tabela de contingência em que na linha está o valor previsto e na coluna o valor real.	52
Tabela 4 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 5, 0, 1), $\eta = 0.03$ em 100 épocas. Obtivemos uma acurácia de 99,42%, sensibilidade de 99,84% e especificidade de 97,84%. 61	61
Tabela 5 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 3, 0, 1), $\eta = 0.15$ em 100 épocas. Obtivemos uma acurácia de 92,81%, sensibilidade de 100,00% e especificidade de 65,84%. 61	61
Tabela 6 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 6, 0, 1), $\eta = 0.005$ em 100 épocas. Obtivemos uma acurácia de 97,92%, sensibilidade de 97,37% e especificidade de 100,00%. 61	61
Tabela 7 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 10, 6, 1), $\eta = 0.05$ em 100 épocas. Obtivemos uma acurácia de 99,70%, sensibilidade de 99,86% e especificidade de 99,07%. 68	68
Tabela 8 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 10, 3, 1), $\eta = 0.65$ em 100 épocas. Obtivemos uma acurácia de 87,13%, sensibilidade de 100,00% e especificidade de 38,89%. 68	68
Tabela 9 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 10, 1, 1), $\eta = 0.01$ em 100 épocas. Obtivemos uma acurácia de 98,29%, sensibilidade de 97,83% e especificidade de 100,00%. 68	68
Tabela 10 - Coeficientes estimados por máxima verossimilhança no software R, para a regressão logística	69
Tabela 11 - Matriz de confusão resultante do teste da capacidade de generalização, após o treinamento da regressão logística. Obtivemos uma acurácia de 97,94%, sensibilidade de 98,49% e especificidade de 95,88%.	70

LISTA DE ABREVIATURAS E SIGLAS

MLP	Multi Layer Perceptron
RNA	Rede Neuronal Artificial
BNDES	Banco Nacional do Desenvolvimento Econômico e Social
CNPJ	Cadastro Nacional da Pessoa Jurídica
IBGE	Instituto Brasileiro de Geografia e Estatística
CEP	Código de Endereçamento Postal
CNAE	Classificação Nacional de Atividades Econômicas
GLM	Generalized Linear Models
VP	Verdadeiro Positivo
VN	Verdadeiro Negativo
FP	Falso Positivo
FN	Falso Negativo
Nadimp	Número de clientes adimplentes
Ninadimp	Número de clientes inadimplentes

LISTA DE SÍMBOLOS

i	Refere-se a unidade de saída
j	Refere-se a unidade escondida ou intermediária
k	Refere-se a unidade de entrada
O	Unidade de saída
V	Unidade escondida ou intermediária
ξ	Terminal de entrada
ζ	Saída desejada
w	Conexões da unidade de entrada para a unidade escondida ou intermediária
W	Conexões da unidade escondida para a unidade de saída
h	Soma dos sinais vindo de outros neurônios
g	Função de ativação
θ	Limiar de ativação
μ	Padrão atual
p	Número de padrões
η	Taxa de aprendizagem
β	Parâmetro de inclinação
δ	Deltas ou erros

SUMÁRIO

	INTRODUÇÃO	15
1	REDES NEURONAIS ARTIFICIAIS	16
1.1	História	16
1.2	Primeiro Modelo de Neurônio Artificial	18
2	PERCEPTRON SIMPLES	20
2.1	Redes Feedforward	20
2.2	Tipos de Aprendizado	23
2.3	Unidades com Limiar de Disparo	23
2.4	Separabilidade Linear	25
2.5	Unidades Lineares	28
2.5.1	<u>Gradiente Descendente</u>	29
2.5.2	<u>Convergência do Gradiente Descendente</u>	31
2.6	Unidades Não-Lineares	33
2.6.1	<u>Função Sigmóide</u>	34
2.6.2	<u>Função Tangente Hiperbólica</u>	35
3	PERCEPTRON MULTICAMADAS	38
3.1	Backpropagation	39
3.1.1	<u>Função Custo</u>	41
3.1.2	<u>Ajuste dos Pesos</u>	41
3.1.3	<u>O Algoritmo</u>	43
4	REGRESSÃO LOGÍSTICA	45
4.1	Transformação Logit	45
4.2	Estimação dos Parâmetros	46
5	O PROBLEMA DE DEVEDORES DO BNDES	48
5.1	Descrição das Variáveis	48
5.2	Normalização dos dados de entrada	50
6	DESEMPENHO DOS MÉTODOS DE CLASSIFICAÇÃO	52
7	RESULTADOS	54
7.1	Perceptron Multicamadas	54
7.1.1	<u>Rede neuronal com 1 camada intermediária</u>	55
7.1.2	<u>Rede neuronal com 2 camadas intermediárias</u>	62
7.2	Regressão Logística	69
	CONCLUSÃO	71
	REFERÊNCIAS	73
	APÊNDICE A – Programas Numéricos	75

INTRODUÇÃO

Desde o início da civilização, o homem buscou ferramentas que o auxiliavam nas tarefas cotidianas, como flechas, lanças, facas e etc. Porém para garantir a manutenção da espécie, sua principal ferramenta e um fator bastante decisivo para o sucesso da espécie foi seu cérebro, com uma enorme capacidade de absorver e analisar informações ao seu redor. O cérebro humano é formado por aproximadamente 86 bilhões de neurônios conectados entre si por aproximadamente 100 trilhões de conexões, de forma que cada neurônio é responsável pela absorção da informação e pela propagação do estímulo ao qual foi submetido.

Vários pesquisadores vêm tentando simular o funcionamento do neurônio biológico por meio de algoritmos e suporte computacional, com o propósito de reduzir a intervenção humana em algumas tarefas. Este estudo ganhou força no século XX com o surgimento do modelo do neurônio artificial idealizado pelos pesquisadores McCulloch e Pitts, posteriormente com o modelo de Hopfield e, na década de 50, com a rede neuronal perceptron proposto por Rosenblatt. Um perceptron é uma rede neuronal artificial capaz de realizar aprendizado de máquina, sendo composto por várias camadas de neurônios, que simulam algumas funções cognitivas, características do cérebro humano. Porém algumas limitações fizeram o perceptron de Rosenblatt cair no esquecimento por um longo tempo. Somente no final da década de 80, com o surgimento do algoritmo de backpropagation proposto por Rumelhart, Hinton e Williams a pesquisa sobre este tipo de técnica de inteligência artificial voltou a ganhar força, pois mostrou que é possível treinar eficientemente redes com camadas intermediárias, resultando no modelo de redes neuronais artificiais mais utilizado atualmente, a rede neuronal do tipo perceptron multicamadas que será o tema central desta dissertação. Desenvolvemos um algoritmo em linguagem C, baseado no algoritmo de backpropagation descrito no livro “Introduction to the theory of neural computation” de (HERTZ; KROGH; PALMER, 1991), onde simulamos a rede neuronal do tipo perceptron multicamadas com o algoritmo de backpropagation e como função de ativação utilizamos a função tangente hiperbólica.

1 REDES NEURONAIS ARTIFICIAIS

As redes neuronais artificiais (RNA) são modelos computacionais inspirados no sistema nervoso dos seres vivos. Estas redes têm a capacidade de aprender padrões complexos de informações, sendo que sua estrutura de conexões massivamente paralela é representada matematicamente, por uma matriz de pesos sinápticos. A ideia desse tipo de rede é criar um paradigma computacional para tentar simular alguns modos de processamento do cérebro humano, baseado em uma estrutura que seja parecida com o modelo biológico do cérebro. As redes neuronais artificiais se baseiam na interpretação e representação do funcionamento dos neurônios e possuem a capacidade de adaptação e auto-aprendizagem que lhes permitem produzir melhores resultados à medida que mais informações tornam-se disponíveis. As redes neuronais artificiais se diferenciam umas das outras basicamente pela sua arquitetura, ou seja, a maneira como seus neurônios e suas conexões estão organizadas.

Segundo (HAYKIN, 1999), o cérebro é um computador altamente complexo e não-linear e tem a capacidade de armazenar informações e certas formas de realizar processamento. No momento do nascimento de um indivíduo, o cérebro tem a capacidade de desenvolver sua aprendizagem através da experiência do indivíduo no meio em que vive, sendo que as regras de aprendizagem são dadas pela biologia. Esse processo ocorre nos neurônios e é também chamado de plasticidade cerebral, uma vez que permite que o sistema nervoso se desenvolva e se adapte ao meio em que está. De acordo com (CHENG; TITTERINGTON, 1994), as redes neuronais artificiais são formadas por um conjunto de unidades computacionais, interligadas por um sistema de conexões e o objetivo deste tipo de inteligência artificial é construir um sistema que possa computar, aprender, lembrar e otimizar da mesma maneira que um cérebro humano. Haykin diz em (HAYKIN, 1999), que uma rede neuronal é algo similar a uma máquina, projetada para modelar a maneira como o cérebro realiza uma tarefa particular ou função de interesse, sendo normalmente implementada com o uso de componentes eletrônicos ou simulada por programação computacional.

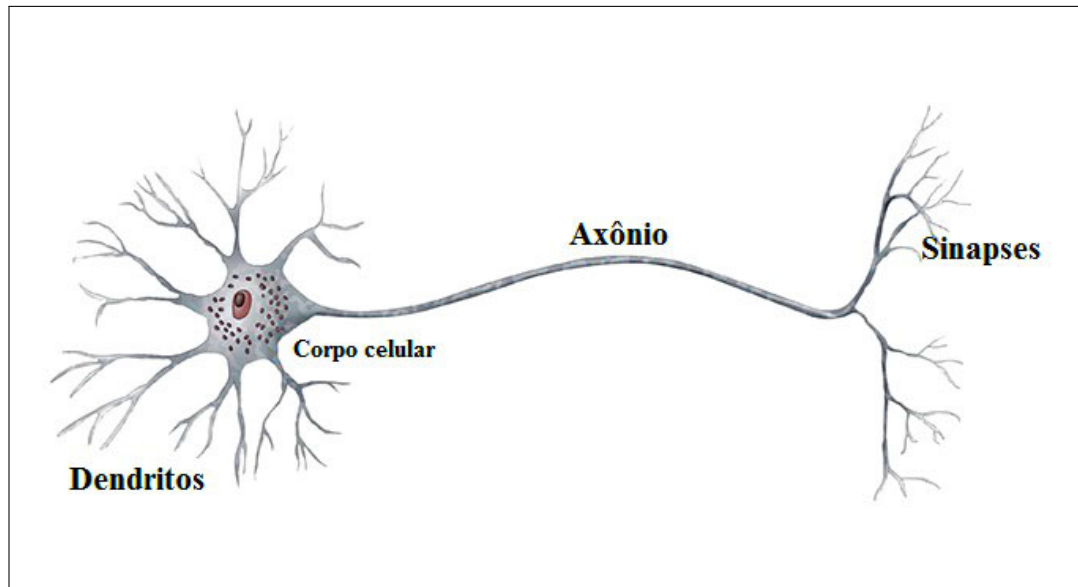
1.1 História

Para entendermos como funcionam e como se originaram os modelos mais recentes de redes neuronais artificiais, devemos voltar ao passado e entender como surgiu a motivação para o desenvolvimento dos primeiros modelos de rede neuronal artificial. O homem sempre procurou meios para simplificar seu trabalho, torná-lo mais fácil ou até automatizá-lo, seja com ferramentas mais rudimentais ou, como no momento atual, com

computadores com grande poder de processamento. Porém o homem aspira a ter algo para o qual ele não precise dar as coordenadas, algo que aprenda, processe e resolva questões como o cérebro humano resolve, de forma paralela. Assim a principal inspiração para o desenvolvimento das redes neuronais artificiais é o cérebro humano. Podemos citar as principais características que permitiram que o cérebro humano tenha feito tanto sucesso, como sua robustez e tolerância a falhas, flexibilidade, seu alto grau de paralelismo, seu tamanho pequeno e compacto, e o fato de que dissipa pouco calor durante sua operação. Tudo o que o ser humano faz, pensa ou sente, é resultado da operação das unidades básicas da estrutura cerebral que são os neurônios. Os neurônios são células nervosas que desempenham o papel de conduzir os impulsos nervosos. Estas células especializadas são, portanto, as unidades básicas do sistema que processa as informações e estímulos no corpo humano. Estima-se que existam cerca de 86 bilhões de neurônios no sistema nervoso humano, conectados uns aos outros através das sinapses, formando assim uma extensa rede. As sinapses transmitem estímulos através da variação das concentrações de sódio e potássio na junção sináptica e no meio intracelular, e desta forma processam e armazenam uma gama muito grande de informações. Listamos a seguir alguns constituintes dos neurônios biológicos.

- Os dendritos são as extensões de uma célula nervosa, através dos quais os impulsos oriundos de outras células são recebidos nas sinapses e transmitidos ao corpo celular.
- O corpo celular é responsável pela integração da informação recebida pelo neurônio de seus vizinhos.
- O axônio é responsável por propagar o impulso nervoso produzido pelo corpo celular até as sinapses que conectam o neurônio aos seus vizinhos.
- Uma sinapse é o local onde se dá a comunicação entre um neurônio e outro. A transferência do impulso nervoso nas sinapses ocorre através dos neurotransmissores.

Figura 1 - Neurônio biológico e seus principais constituintes.



Fonte: O autor, 2019.

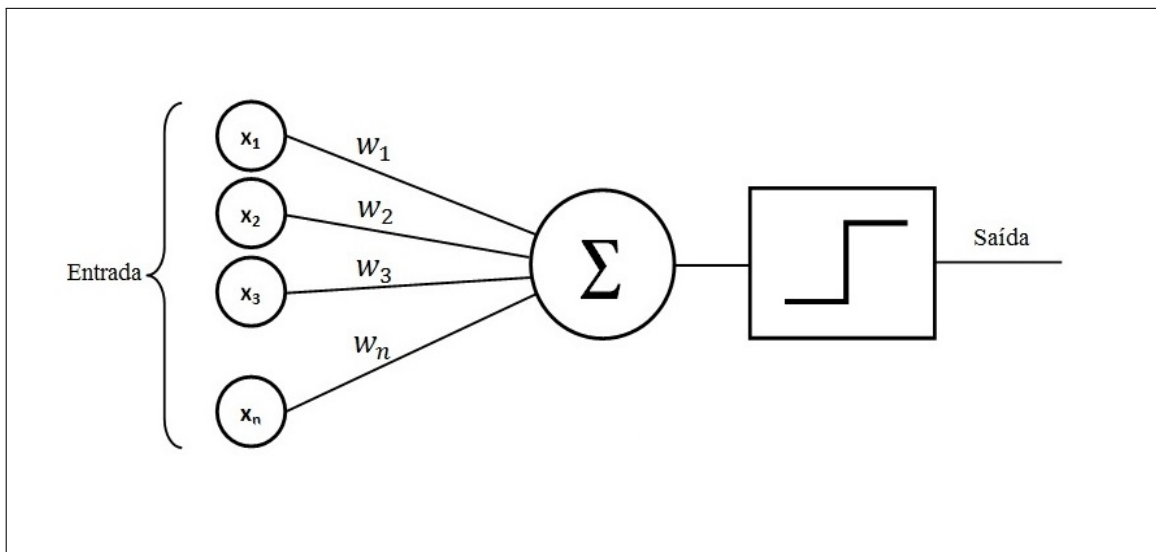
1.2 Primeiro Modelo de Neurônio Artificial

A pesquisa na área de redes neuronais artificiais teve início com dois pesquisadores americanos, Warren Sturgis McCulloch (1898 – 1969), psiquiatra e neuroanatomista que dedicou cerca de 20 anos refletindo e estudando sobre a representação do sistema nervoso humano, e Walter Pitts (1923 – 1969), era matemático e foi trabalhar com McCulloch na Universidade de Chicago. Ambos fizeram parte de um dos primeiros grupos do mundo dedicado ao estudo da biofísica teórica, criado por Nicolas Rashevsky (1899 – 1972). Em 1943, McCulloch e Pitts escreveram juntos um artigo chamado “*A logical calculus of the ideas immanent in nervous activity*” (MCCULLOCH; PITTS, 1943), onde descrevem que os eventos neurais e as relações entre eles podem ser tratados por meio da lógica proposicional. Assim criaram as bases para o primeiro modelo de redes neuronais artificiais. O grande impacto do modelo de McCulloch e Pitts deu-se na Ciência da Computação. O pai dos modernos computadores digitais, John von Neumann, foi bastante influenciado pelo trabalho de McCulloch e Pitts, pois ele percebeu o grande poder computacional que um sistema composto por unidades lógicas simples possui. O modelo de McCulloch e Pitts é baseado em cinco hipóteses.

1. A atividade de um neurônio é binária, ou seja, a cada instante, o neurônio ou está disparando (atividade = 1), ou não está disparando (atividade = 0).
2. As conexões entre pares de neurônios são bidirecionais. Estas conexões do modelo são inspiradas nas sinapses biológicas e podem ser excitatórias ou inibitórias.

3. Um neurônio possui um limiar de disparo fixo, sendo que ele só dispara se o sinal total que ele recebe, num dado instante, for maior ou igual ao valor do limiar.
4. A recepção de um sinal através de uma sinapse inibitória, em um dado instante, impede que o neurônio dispare.
5. Um sinal leva uma unidade de tempo para passar de um neurônio da rede para outro.

Figura 2 - Primeiro modelo de neurônio artificial criado por McCulloch e Pitts.



Fonte: O autor, 2019.

A Figura 2 representa a estrutura lógica do modelo de neurônio artificial criado por McCulloch e Pitts, formado por um vetor de entradas e as respectivas sinapses representadas por pesos numéricos. A soma ponderada das entradas é submetida à uma função de ativação, que determina se a soma é maior ou menor que o valor do limiar do neurônio. Se for maior, o neurônio dispara e sua variável de estado recebe o valor binário 1, caso contrário o neurônio não dispara e sua variável de estado recebe o valor binário 0.

2 PERCEPTRON SIMPLES

¹ A rede neuronal artificial do tipo perceptron foi proposta inicialmente por Frank Rosenblatt (1928 – 1971), entre os anos de 1957 a 1962. Rosenblatt era um psicólogo americano e sua pesquisa inspirou muitos engenheiros, físicos e matemáticos a dedicar seus esforços em diferentes aspectos das redes neurais. É interessante notar que o perceptron é tão reconhecido nos dias atuais, quanto no final da década de 50, quando o modelo foi publicado no artigo (ROSENBLATT, 1958). O perceptron foi desenvolvido para lidar com problemas de classificação e reconhecimento de padrões. Esta é uma tarefa que os seres humanos fazem sem nenhum esforço aparente e de forma quase instantânea sendo, no entanto, um problema difícil de ser resolvido por uma máquina. A palavra perceptron deriva do latim “percipio”, que quer dizer compreender, e sua forma supina é “perceptum”, ou seja, a rede deve ser capaz de compreender o mundo exterior.

De acordo com Haykin (HAYKIN, 1999), mesmo sendo de grande importância e inspiração para muitos pesquisadores o perceptron recebeu duras críticas, e a primeira crítica real ao perceptron de Rosenblatt foi apresentada no estudo (MINSKY; SELFRIDGE, 1961), onde os autores mostraram que o perceptron definido por Rosenblatt não poderia generalizar nem em relação à noção de paridade (parecença, semelhança) e muito menos fazer abstrações genéricas. As limitações computacionais do perceptron de Rosenblatt foram mostradas com uma sólida fundamentação matemática, no famoso livro “Perceptrons: an introduction to computational geometry” (MINSKY; PAPERT, 1969). Após a apresentação de uma análise matemática brilhante e bem detalhada do perceptron, Minsky e Papert provaram que o perceptron definido por Rosenblatt só poderia ser aplicado em problemas linearmente separáveis. Esta conclusão de Minsky e Papert foi a grande responsável por lançar sérias dúvidas sobre as capacidades computacionais, não apenas do perceptron, mas das redes neurais em geral, ocasionando um grande desinteresse pela área, até meados dos anos 80, quando surgiu o algoritmo de backpropagation, fazendo com que muitos pesquisadores voltassem a pesquisar sobre as redes neurais artificiais.

2.1 Redes Feedforward

As redes feedforward foram chamadas de perceptrons quando estudadas em detalhe por Rosenblatt e seus colaboradores no artigo “Principles of neurodynamics: perceptrons and the theory of brain mechanisms” (ROSENBLATT, 1962). As redes feedforward têm,

¹ Este capítulo é baseado na referência (HERTZ; KROGH; PALMER, 1991).

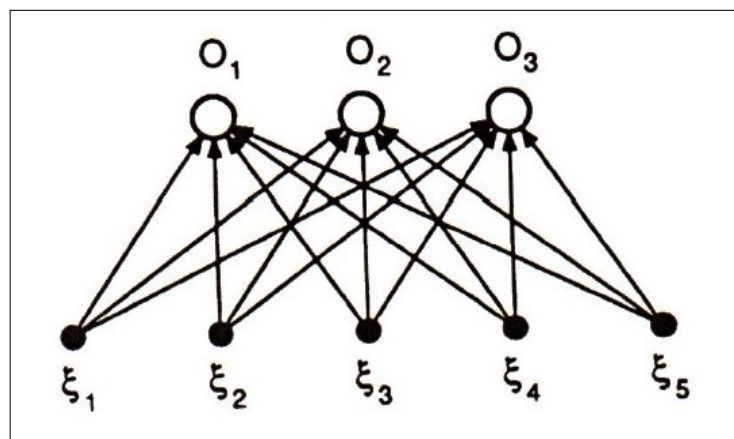
por definição, conexões w_{ij} assimétricas, de forma que $w_{ij} \neq w_{ji}$. Todas as conexões são unidirecionais e é possível enumerar as unidades, de modo que a matriz de pesos w_{ij} seja triangular, na qual todas as entradas acima ou abaixo da diagonal principal são zero. Como as conexões são assimétricas, não é possível definir uma função de energia associada às interações entre os neurônios da rede, como no caso da rede de Hopfield. Somente com conexões simétricas pode-se garantir a existência deste tipo de função de energia. Assim, não podemos empregar aqui métodos de mecânica estatística de equilíbrio.

As redes do tipo feedforward possuem um conjunto de terminais de entrada cujo o único papel é alimentar os padrões de entrada para o restante da rede. Depois disso, pode surgir uma ou mais camadas intermediárias de unidades, seguidas por uma camada de saída final, na qual o resultado da computação é lido. Nesta classe restrita de redes feedforward, não há conexões de um neurônio de camada superior para um neurônio de camada inferior, nem a outras unidades na mesma camada, nem a unidades com mais de uma camada à frente. Cada unidade (ou entrada) alimenta apenas os neurônios da próxima camada. As unidades nas camadas intermediárias são frequentemente chamadas de unidades escondidas.

De acordo com (GOODFELLOW; BENGIO; COURVILLE, 2016), as redes feedforward são de extrema importância para os profissionais que atuam com aprendizado de máquina. Elas formam a base de muitas aplicações comerciais importantes, por exemplo, as redes convolucionais, usadas para reconhecimento de objetos a partir de fotos. O conceito de redes feedforward abre caminho para a construção de redes recorrentes, que estão presentes em muitas aplicações mais complexas.

As arquiteturas de redes neuronais mais utilizadas são: feedforward de uma camada, feedforward de múltiplas camadas, recorrente e estrutura reticulada. Nesta seção, nos restringimos apenas às redes feedforward de uma camada, as quais chamamos de perceptron simples.

Figura 3 - Rede feedforward de apenas uma camada.



Fonte: HERTZ, 1991.

A Figura 3 mostra como um perceptron simples é organizado, com suas respectivas unidades de entrada ξ_k e suas unidades de saída O_i . As conexões de um perceptron simples são assimétricas e unidirecionais, partindo de uma unidade de entrada para uma unidade de saída. A computação realizada por uma unidade de saída O_i é representada por uma função de ativação $g(h)$, geralmente não-linear. O resultado do cálculo de uma unidade de saída é uma função explícita das entradas, isso vale para todas as redes feedforward. A entrada é propagada pela rede e produz a saída, de acordo com a equação

$$O_i = g(h_i) = g\left(\sum_k w_{ik}\xi_k\right). \quad (1)$$

Os limiares de ativação foram omitidos de nossa descrição, porque estes podem ser tratados através da inclusão de conexões para uma unidade de entrada, cujo estado tem o valor permanente -1 . Mais especificamente, podemos fixar $\xi_0 = -1$ e escolher as conexões $w_{i0} = \theta_i$ para obter a equação com o limiar θ_i

$$O_i = g\left(\sum_{k=0}^N w_{ik}\xi_k\right) = g\left(\sum_{k=1}^N w_{ik}\xi_k - \theta_i\right). \quad (2)$$

A tarefa geral de associação pode ser modelada demandando que um padrão de saída específico ζ_i^μ seja calculado em resposta a um padrão de entrada ξ_k^μ . Ou seja, queremos que o padrão de saída calculado O_i^μ seja igual ao padrão de saída desejado ζ_i^μ , para cada i e μ , de forma que

$$O_i^\mu = \zeta_i^\mu \quad (\text{desejado}). \quad (3)$$

Para a rede neuronal perceptron simples, a saída calculada O_i^μ é dada pela equação (1), quando a entrada ξ_k assume o valor do padrão ξ_k^μ :

$$O_i^\mu = g(h_i^\mu) = g\left(\sum_k w_{ik}\xi_k^\mu\right). \quad (4)$$

Definimos p como sendo o número de pares entrada-saída no conjunto de treinamento, de forma que $\mu = (1, 2, \dots, p)$. As entradas, saídas calculadas e saídas desejadas assumir valores binários ± 1 ou contínuos. Para as saídas calculadas O_i^μ , isso depende da natureza da função de ativação $g(h)$. A função de ativação tanto como a rede estão relacionados com o tipo de problema a ser resolvido.

2.2 Tipos de Aprendizado

Segundo Haykin (HAYKIN, 1999), a habilidade de aprender é de grande importância e ajuda a melhorar o desempenho de uma RNA. Este processo de aprendizagem ocorre através de diversas iterações, e em cada iteração são aplicados ajustes aos pesos sinápticos. Idealmente a rede torna-se melhor preparada a cada iteração deste processo. Este processo é conhecido como algoritmo de aprendizagem. Em redes neurais artificiais a aprendizagem pode ser realizada de duas formas:

- O aprendizado supervisionado corresponde a uma forma de aprendizado que utiliza um conjunto de pares ordenados (entrada, saída), chamado de conjunto teste, onde o valor correto da saída, para cada elemento do conjunto teste, é previamente conhecido. Os pesos da rede são ajustados em diversas iterações do algoritmo, onde um agente indica se a resposta da rede está correta para cada padrão de entrada do conjunto teste. Haykin (HAYKIN, 1999), descreve o aprendizado supervisionado como sendo um aprendizado sobre a tutela de um professor. Na rede neuronal artificial do tipo perceptron utilizamos o aprendizado supervisionado.
- No aprendizado não-supervisionado, não existe o conjunto teste com pares ordenados (entrada, saída) previamente conhecidos. A rede deve classificar os padrões apresentados a ela de acordo com critérios de semelhança entre os padrões. Desta maneira o sistema extrai as características do conjunto de padrões, agrupando-os em classes inerentes aos dados. Este tipo de aprendizado é muito utilizado em problemas de clusterização. Um exemplo de aprendizagem não-supervisionada são os mapas auto-organizáveis de Kohonen.

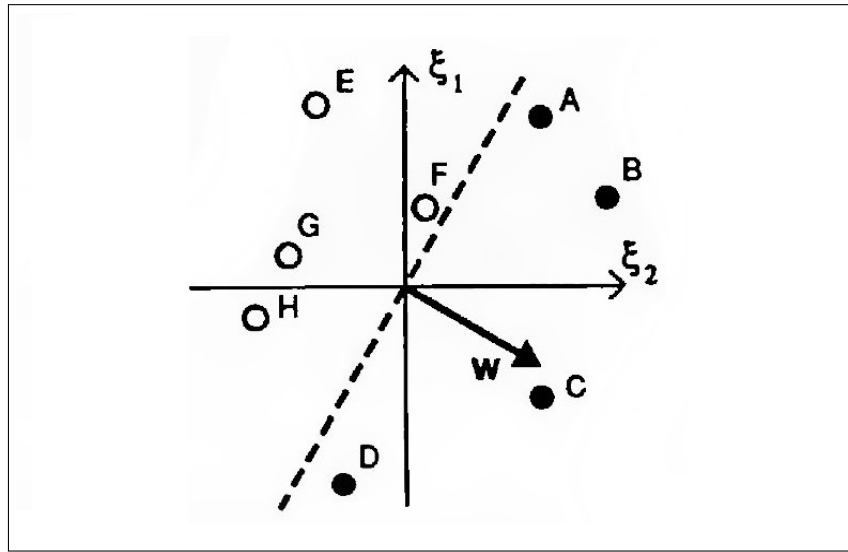
2.3 Unidades com Limiar de Disparo

Para entendermos como funcionam as redes com unidades com limiar de disparo, partiremos de um caso simples de unidades com limiar determinístico, $g(h) = \text{sgn}(h)$, e assumimos que as saídas desejadas ζ_i^μ assumem valores binários ± 1 . Portanto, o que importa é que o sinal de entrada ponderado do neurônio i , h_i^μ , seja igual a ζ_i^μ , para cada i e μ . As unidades de saída são independentes, e portanto é conveniente considerar apenas uma unidade de saída por vez e descartar os índices i . Então os pesos w_{ik} tornam-se um vetor de pesos $\mathbf{w} = (w_1, w_2, \dots, w_N)$ com um elemento para cada entrada. Cada padrão de entrada ξ_k^μ também pode ser considerado como um vetor padrão $\boldsymbol{\xi}^\mu$ neste mesmo espaço N -dimensional. Então, a condição da equação (3) se torna, para cada μ ,

$$\text{sgn}(\mathbf{w} \cdot \boldsymbol{\xi}_\mu) = \zeta^\mu \quad (\text{desejado}). \quad (5)$$

A equação (5) implica que o vetor de pesos \mathbf{w} deve ser escolhido de modo que a projeção do padrão ξ^μ sobre ele tenha o mesmo sinal que ζ^μ . Mas a fronteira entre as projeções positivas e negativas sobre \mathbf{w} é o plano $\mathbf{w} \cdot \xi = 0$, que passa através da origem e é perpendicular à \mathbf{w} . Assim, a condição para a operação correta da rede é que este plano deve dividir as entradas que possuem saídas positivas daquelas que possuem saídas negativas, conforme ilustrado na Figura 4.

Figura 4 - Plano perpendicular ao vetor peso \mathbf{w} que separa os círculos sólidos $\zeta_i = +1$ dos abertos $\zeta_i = -1$.



Fonte: HERTZ, 1991.

Frequentemente, é conveniente usar uma representação alternativa. Se definimos

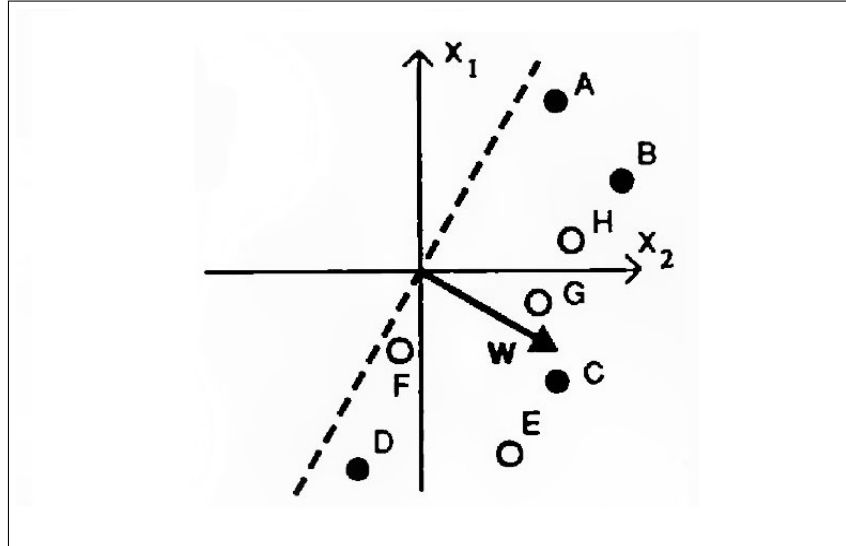
$$\mathbf{x}^\mu \equiv \zeta^\mu \xi^\mu, \quad (6)$$

transformamos a condição (5), para cada μ , em

$$\mathbf{w} \cdot \mathbf{x}^\mu > 0 \quad (\text{desejado}). \quad (7)$$

Isto significa que os vetores \mathbf{x} (cada um dos quais depende de um par (entrada, saída) do conjunto de treinamento) devem estar do mesmo lado do plano perpendicular a \mathbf{w} , como ilustrado na Figura 5.

Figura 5 - Plano perpendicular ao vetor de pesos \mathbf{w} , onde, de acordo com a equação (7), todas as saídas devam estar do mesmo lado do plano.



Fonte: HERTZ, 1991.

2.4 Separabilidade Linear

Caso não seja possível encontrar um vetor de pesos \mathbf{w} que separe o plano, o problema não pode ser resolvido, e a rede não poderá executar a tarefa de reconhecimento de padrões não importa o quanto ela é treinada. Portanto, a condição principal para que exista uma solução para um problema, utilizando a rede neuronal do tipo perceptron simples com unidades limiares é saber se esse problema é ou não linearmente separável. Um problema linearmente separável é aquele para o qual um plano pode ser encontrado no espaço ξ que separa os padrões $\zeta^\mu = +1$ dos padrões $\zeta^\mu = -1$. Se houver várias unidades de saída, devemos ser capazes de encontrar um tal plano, para cada saída. Se o limiar de disparo for igual a zero, o plano de separação deve passar pela origem, como vimos na Figura 4. No entanto, é interessante restabelecer um limite explícito temporariamente. Isso transforma o cálculo realizado pela rede em

$$O_i = \text{sgn} \left(\sum_{k>0} w_{ik} \xi_k - w_{i0} \right), \quad (8)$$

ou, para apenas uma unidade em notação vetorial,

$$O = \text{sgn}(\mathbf{w} \cdot \boldsymbol{\xi} - w_0). \quad (9)$$

Assim, as regiões no espaço de entrada N -dimensional $(\xi_1, \xi_2, \dots, \xi_N)$ com diferentes decisões (± 1) para a saída calculada O , são separadas por um plano $(N - 1)$ -dimensional, com distância w_0 da origem

$$\mathbf{w} \cdot \boldsymbol{\xi} = w_0. \quad (10)$$

Portanto, o efeito de adicionar um limiar explícito é simplesmente o de permitir que o plano separador não passe pela origem. Novamente, existe um desses planos para cada unidade de saída.

Alguns exemplos são apropriados para o estudo da separabilidade linear. Consideramos primeiro um exemplo simples onde a função é linearmente separável. A função AND, é uma função que possui duas variáveis binárias que assumem valores ± 1 , e portanto é necessário uma rede neuronal do tipo perceptron simples com duas unidades de entrada ξ_1 e ξ_2 .

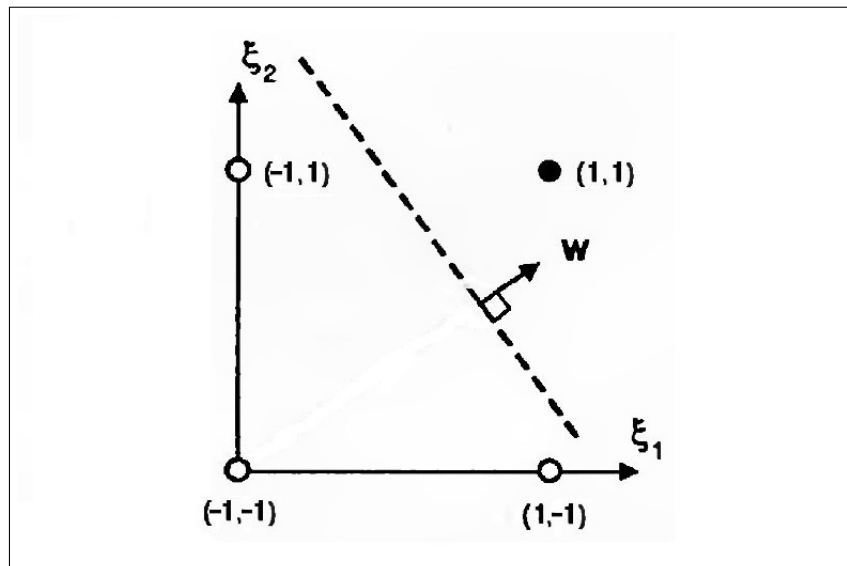
Tabela 1 - Tabela verdade que define a função AND, onde duas entradas resultam em uma saída positiva, somente se todas as entradas são positivas, caso contrário a saída é negativa.

ξ_1	ξ_2	ζ
-1	-1	-1
-1	+1	-1
+1	-1	-1
+1	+1	+1

Fonte: HERTZ, 1991.

Na Figura 6 podemos encontrar o plano separador, logo o problema é linearmente separável e uma rede neuronal perceptron simples pode resolvê-lo.

Figura 6 - Plano no espaço (ξ_1, ξ_2) , que separa o círculo sólido ($\zeta^\mu = +1$) dos círculos abertos ($\zeta^\mu = -1$).



Fonte: HERTZ, 1991.

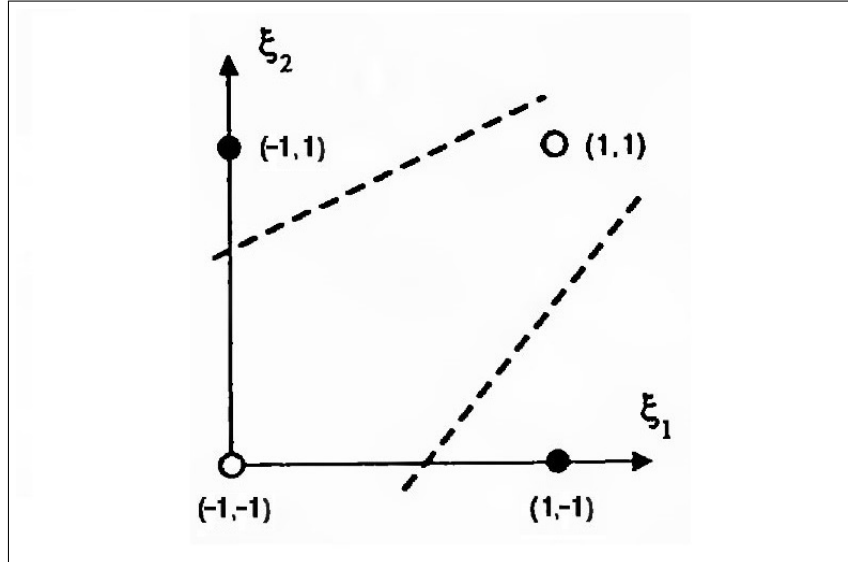
Consideraremos agora um caso onde a função não é linearmente separável, a função XOR (ou-exclusivo). Os pesquisadores Minsky e Papert (MINSKY; PAPERT, 1969), mostraram em seu estudo que o perceptron simples não poderia resolver esse tipo de problema. Como pode ser observado na Figura 7, não é possível traçar uma única reta que divide o plano em dois, de forma a separar os dois tipos de pontos, e portanto não é possível resolver este problema com uma rede neuronal do tipo perceptron simples.

Tabela 2 - Tabela verdade da função XOR, onde a saída será positiva, se exclusivamente uma ou a outra entrada for positiva.

ξ_1	ξ_2	ζ
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	-1

Fonte: HERTZ, 1991.

Figura 7 - Problema com a não separabilidade linear. Caso onde não é possível separar os círculos sólidos ($\zeta^\mu = +1$) dos círculos abertos ($\zeta^\mu = -1$) com uma única reta.



Fonte: HERTZ, 1991.

2.5 Unidades Lineares

Até agora em nosso estudo de rede neuronal do tipo perceptron simples, consideramos apenas unidades de limiar, com $g(h) = \text{sgn}(h)$. Voltaremos agora para as unidades de valor contínuo, com $g(h)$ sendo uma função contínua e diferenciável de u . A grande vantagem destas unidades é que elas nos permitem construir uma função custo $E[\mathbf{w}]$ que mede o erro de desempenho do sistema, como uma função diferenciável dos pesos $\mathbf{w} = \{w_{ik}\}$. Podemos então usar uma técnica de otimização, como o gradiente descendente, para minimizar esta medida de erro.

Começamos nesta seção com as unidades lineares, para as quais $g(h) = h$. Estas não são tão úteis na prática quanto as redes não-lineares, que consideraremos a seguir, mas são mais simples e permitem uma análise mais detalhada. A saída de uma rede neuronal do tipo perceptron simples linear, submetido a um padrão de entrada ξ_k^μ , é dada por

$$O_i^\mu = \sum_k w_{ik} \xi_k^\mu, \quad (11)$$

e a associação desejada é $O_i^\mu = \zeta_i^\mu$, como na equação (3), ou

$$\zeta_i^\mu = \sum_k w_{ik} \xi_k^\mu \quad (\text{desejado}). \quad (12)$$

Agora as saídas O_i^{μ} s assumem valores contínuos, embora ainda possamos restringir os valores desejados de ζ_i^{μ} a ± 1 .

2.5.1 Gradiente Descendente

O método do gradiente descendente é um método de otimização numérica que busca o mínimo local de uma função de forma iterativa. Em cada passo é tomada a direção negativa do gradiente, que corresponde à direção de declive máximo. Estamos interessados em encontrar a regra de aprendizado que nos permitirá encontrar um conjunto de pesos por incrementos sucessivos, a partir de um ponto inicial escolhido arbitrariamente, que produzem as saídas desejadas a partir de cada padrão de entrada. Para isso, definiremos uma medida de erro ou função custo, por

$$E[\mathbf{w}] = \frac{1}{2} \sum_{i\mu} (\zeta_i^{\mu} - O_i^{\mu})^2 = \frac{1}{2} \sum_{i\mu} (\zeta_i^{\mu} - \sum_k w_{ik} \xi_k^{\mu})^2. \quad (13)$$

A grande vantagem desta função custo é que ela mede o erro de desempenho do sistema como uma função diferenciável dos pesos \mathbf{w} . Podemos então usar a técnica de otimização do gradiente descendente, para minimizar essa medida de erro. A função E é sempre positiva ou igual a zero, e se aproxima de zero quando chegamos próximo da solução ótima. A função custo depende somente dos pesos w_{ik} e dos padrões que representam o problema em questão. Dada nossa função custo $E[\mathbf{w}]$, podemos melhorar o conjunto dos pesos w_{ik} , deslizando para baixo na superfície que esta função define no espaço \mathbf{w} . Especificamente, o algoritmo do gradiente descendente incrementa cada peso w_{ik} por um número Δw_{ik} proporcional ao gradiente de E , na posição atual. O valor do incremento é dado pela taxa de aprendizagem denotada pelo parâmetro η e pela derivada da função custo em relação ao peso w_{ik} , de forma que

$$\begin{aligned} \Delta w_{lm} &= -\eta \frac{\partial E}{\partial w_{lm}} \\ &= -\eta \frac{\partial}{\partial w_{lm}} \left[\frac{1}{2} \sum_{i\mu} (\zeta_i^{\mu} - O_i^{\mu})^2 \right] \\ &= 2 \cdot \frac{1}{2} \cdot (-\eta) \sum_{\mu} (\zeta_l^{\mu} - O_l^{\mu}) \frac{\partial}{\partial w_{lm}} \left[\left(\zeta_l^{\mu} - \sum_k w_{lk} \xi_k^{\mu} \right) \right] \\ &= -\eta \sum_{\mu} (\zeta_l^{\mu} - O_l^{\mu}) (-\xi_m^{\mu}) \\ \Delta w_{lm} &= \eta \sum_{\mu} (\zeta_l^{\mu} - O_l^{\mu}) \xi_m^{\mu}. \end{aligned} \quad (14)$$

Se realizarmos mudanças individualmente para cada padrão de entrada ξ_k^μ , teremos

$$\Delta w_{ik} = \eta (\zeta_i^\mu - O_i^\mu) \xi_k^\mu, \quad (15)$$

correspondendo às mudanças em resposta ao padrão μ . Ainda podemos simplificar a equação (15) se definirmos os erros ou deltas como

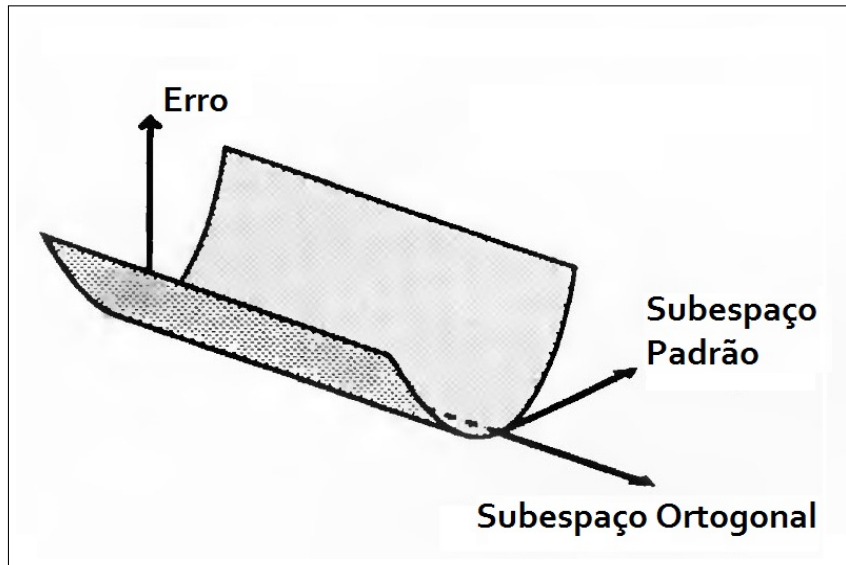
$$\delta_i^\mu = \zeta_i^\mu - O_i^\mu, \quad (16)$$

transformando a equação (15) em

$$\Delta w_{ik} = \eta \delta_i^\mu \xi_k^\mu. \quad (17)$$

Este resultado (equações (15), (16) e (17)) é chamado de regra delta, ou regra adaline, ou regra Widrow-Hoff, ou método dos mínimo quadrados, como é chamado em inglês (LMS - least mean square) de acordo com (RUMELHART; MCCLELLAND, 1986). A função custo (13) é simplesmente uma forma quadrática nos pesos. No subespaço gerado pelos padrões, a superfície é uma bacia parabólica. Assumindo que os vetores padrão são linearmente independentes, de modo que exista solução para a equação (3), o valor mínimo é encontrado em $E = 0$. Nas direções (se houver) do espaço \mathbf{w} ortogonal a todos os vetores padrão, o erro é constante. Em outras palavras, a superfície de erro no espaço dos pesos é como uma “calha de chuva”, como mostrado na Figura 8, com infinitos vales nas direções ortogonais aos vetores padrão.

Figura 8 - Superfície de erro no espaço de peso.



Fonte: HERTZ, 1991.

A regra do gradiente descendente (14), produz alterações nos vetores de peso $\mathbf{w}_i = (w_{i1}, \dots, w_{iN})$ apenas nas direções dos vetores padrão $\boldsymbol{\xi}^\mu$. Dentro do subespaço gerado pelos vetores padrão, a regra do gradiente descendente necessariamente diminui o erro, se a taxa de aprendizagem η for pequena o suficiente, porque nos leva na direção descendente do gradiente. Assim, com iterações suficientes, nos aproximamos do fundo do vale da “calha de chuva”, a partir de qualquer ponto de partida. Além disso, qualquer ponto no fundo do vale resolve o problema dado pelas equações (3) e (4) de forma exata.

2.5.2 Convergência do Gradiente Descendente

O argumento apresentado na seção anterior para a convergência do gradiente até atingir o fundo do vale é intuitivamente razoável, mas precisa de uma análise mais detalhada. O primeiro passo é diagonalizar a forma quadrática da função custo (13). Se os vetores padrão são linearmente independentes, isso nos permite escrever $E[\mathbf{w}]$ na forma

$$E = \sum_{\lambda=1}^M \alpha_\lambda (w_\lambda - w_\lambda^0)^2, \quad (18)$$

onde M é o número total de pesos, igual a N vezes o número de unidades de saída, e os w_λ 's são combinações lineares dos w_{ik} 's. Os α_λ 's e w_λ^0 's são constantes dependendo apenas dos vetores padrão. Os autovalores α_λ são necessariamente positivos ou zero, por causa da soma dos quadrados da função custo, sua forma quadrática é positiva semi-definida. Observamos que se alguns dos α_λ 's são zero então E é independente dos w_λ 's correspondentes. Isso é equivalente às calhas de chuva mencionadas anteriormente; as direções dos autovetores correspondentes no espaço \mathbf{w} são ortogonais a todos os vetores padrão.

Agora vamos realizar o procedimento de gradiente descendente em (18). Sabendo que a transformação de w_{ik} para w_λ é linear, isso é inteiramente equivalente ao gradiente descendente na base original. No entanto, a base diagonal torna isso muito mais fácil:

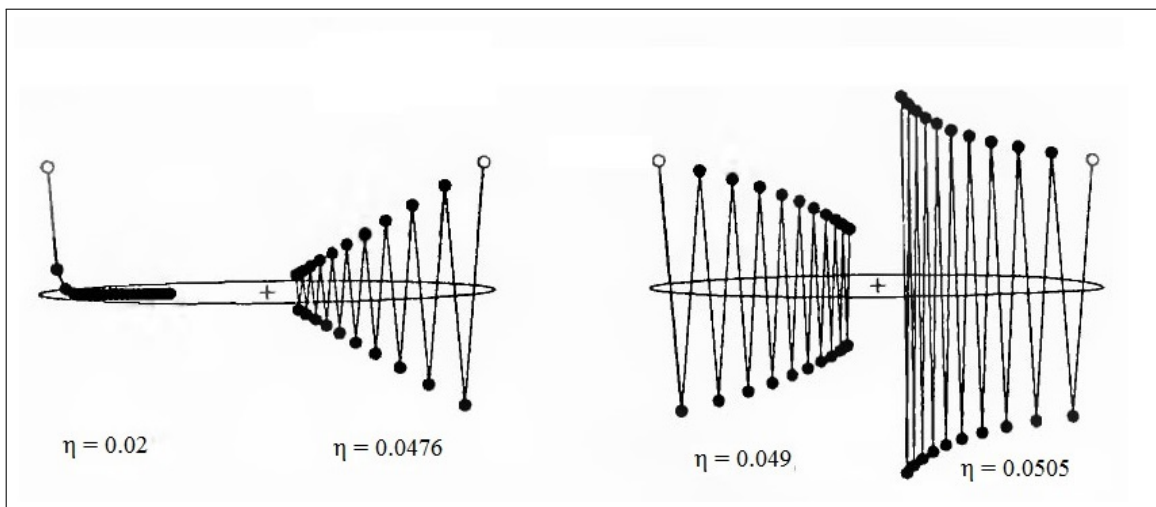
$$\Delta w_\lambda = -\eta \frac{\partial E}{\partial w_\lambda} = -\eta \frac{\partial}{\partial w_\lambda} \left[\sum_{\gamma=1}^M \alpha_\gamma (w_\gamma - w_\gamma^0)^2 \right] = -2\eta \alpha_\lambda (w_\lambda - w_\lambda^0). \quad (19)$$

Portanto, a distância $\delta w_\lambda = w_\lambda - w_\lambda^0$ a partir da solução ótima na direção de λ é transformada de acordo com

$$\delta w_\lambda^{new} = \delta w_\lambda^{old} + \Delta w_\lambda = (1 - 2\eta \alpha_\lambda) \delta w_\lambda^{old}. \quad (20)$$

Nas direções em que $\alpha_\lambda > 0$ nós nos aproximamos da solução ótima, desde que $|1 - 2\eta\alpha_\lambda| < 1$. A abordagem é de primeira ordem, cada distância δw_λ é multiplicada por um fator fixo em cada interação. O valor de η é limitado pelo maior autovalor α_λ^{\max} , correspondente à direção da curvatura mais íngreme da superfície de erro; devemos ter $\eta < 1/\alpha_\lambda^{\max}$ ou vamos acabar extrapolando (pulando) o mínimo, e assim nunca chegaremos ao valor mínimo do vale onde a solução é ótima. A taxa de aproximação até chegar à solução ótima é geralmente limitada pelo menor autovalor não nulo α_λ^{\min} , correspondente à direção de curvatura mais rasa da “calha de chuva”. Se $\alpha_\lambda^{\max}/\alpha_\lambda^{\min}$ for muito grande, a convergência pode ser bastante lenta.

Figura 9 - Procedimento de descida do gradiente em uma superfície quadrática simples (as partes da esquerda e da direita são cópias da mesma superfície). Quatro trajetórias são mostradas, cada uma para 20 iterações a partir do círculo aberto. O valor mínimo é no ponto + e a elipse mostra o contorno de erro constante. A única diferença significativa entre as trajetórias é o valor de η , que foi de 0.02, 0.0476, 0.049 e 0.0505 da esquerda para a direita.



Fonte: HERTZ, 1991.

A Figura 9 ilustra o procedimento de gradiente descendente com um caso simples. Mostramos o gradiente descendente na superfície $E = x^2 + 20y^2$ para 20 iterações em diferentes valores de η . Esta forma quadrática já é diagonal, com $a_1 = 1$ e $a_2 = 20$. Com $\eta = 0.02$ alcançamos $y \approx 0$ rapidamente, mas, em seguida, progredimos lentamente no eixo x . No outro extremo, se $\eta > 1/20 = 0.05$, o algoritmo produz uma oscilação divergente no eixo y . A abordagem mais rápida é aproximadamente quando os coeficientes x e y são iguais a $|1 - 2\eta|$ e $|1 - 40\eta|$, fornecendo $\eta = 1/21 = 0,0476$ (segunda ilustração).

2.6 Unidades Não-Lineares

De acordo com Glorot e Bengio (GLOROT; BENGIO, 2010), as funções de ativação são essenciais para dar capacidade representativa às redes neurais artificiais, introduzindo uma componente de não linearidade. Particularmente, ao introduzir uma função de ativação não linear, a superfície de custo da rede neuronal deixa de ser convexa, como a calha de chuva descrita anteriormente. Isto torna a otimização mais complicada, mas permite que a rede trate outros tipos de problemas que não podem ser resolvidos com unidades lineares. Desta forma, generalizamos a aprendizagem do gradiente descendente a partir do caso linear $g(h) = h$.

A função custo com a forma quadrática para a função de ativação não-linear, $g(h)$, torna-se

$$E[\mathbf{w}] = \frac{1}{2} \sum_{i\mu} \left(\zeta_i^\mu - O_i^\mu \right)^2 = \frac{1}{2} \sum_{i\mu} \left[\zeta_i^\mu - g \left(\sum_k w_{ik} \xi_k^\mu \right) \right]^2. \quad (21)$$

Tomando a derivada da soma dos quadrados da função custo (21),

$$\begin{aligned} \frac{\partial E}{\partial w_{lm}} &= \frac{\partial}{\partial w_{lm}} \left[\frac{1}{2} \sum_{i\mu} \left[\zeta_i^\mu - g \left(\sum_k w_{ik} \xi_k^\mu \right) \right]^2 \right], \\ \frac{\partial E}{\partial w_{lm}} &= 2 \cdot \frac{1}{2} \sum_{\mu} \left[\zeta_l^\mu - g(h_l^\mu) \right] \frac{\partial}{\partial w_{lm}} \left[\zeta_l^\mu - g \left(\sum_k w_{lk} \xi_k^\mu \right) \right], \end{aligned}$$

onde,

$$(h_l^\mu) = \sum_k w_{lk} \xi_k^\mu.$$

Desta forma

$$\begin{aligned} \frac{\partial E}{\partial w_{lm}} &= \sum_{\mu} \left[\zeta_l^\mu - g(h_l^\mu) \right] \cdot \left[-g'(h_l^\mu) \right] \xi_m^\mu \\ \frac{\partial E}{\partial w_{lm}} &= - \sum_{\mu} \left[\zeta_l^\mu - g(h_l^\mu) \right] g'(h_l^\mu) \xi_m^\mu. \end{aligned} \quad (22)$$

A correção dada pelo gradiente descendente $-\eta \partial E / \partial w_{ik}$ para w_{ik} após a apresentação do padrão μ tem a mesma forma que a equação (17). A medida de erro delta (δ) agora adquire um fator $g'(h_i^\mu)$, que é a derivada da função de ativação $g(h)$, e se apresenta como

$$\delta_i^\mu = [\zeta_i^\mu - O_i^\mu] g'(h_i^\mu) . \quad (23)$$

Existem diversos tipos de funções de ativação, tais como: função degrau, função linear, sigmóide, tangente hiperbólica, ReLU, Leaky ReLU, Softmax. Neste trabalho consideraremos a função sigmóide e a função tangente hiperbólica.

2.6.1 Função Sigmóide

De acordo com Haykin (HAYKIN, 2007), a função sigmóide é a forma mais comum de função de ativação utilizada na construção de redes neurais artificiais. Ela é definida como uma função estritamente crescente que exibe um balanceamento adequado entre o comportamento linear e não-linear. A função sigmóide é contínua no intervalo de 0 a 1 e é diferenciável. Para encontrar o fator $g'(h)$ da função sigmóide, devemos calcular a derivada de $g(h)$ da seguinte forma. Seja

$$g(h) = \left[1 + e^{(-2\beta h)}\right]^{-1} . \quad (24)$$

Sua derivada é dada por

$$g'(h) = \frac{\partial}{\partial h} \left[1 + e^{(-2\beta h)}\right]^{-1} = -\left[1 + e^{(-2\beta h)}\right]^{-2} \cdot e^{(-2\beta h)} \cdot (-2\beta) .$$

De forma que

$$\begin{aligned} g'(h) &= \frac{1}{\left(1 + e^{(-2\beta h)}\right)^2} \cdot e^{(-2\beta h)} \cdot 2\beta , \\ &= \frac{e^{(-2\beta h)}}{\left(1 + e^{(-2\beta h)}\right)^2} \cdot 2\beta , \\ &= \frac{1}{\left(1 + e^{(-2\beta h)}\right)} \cdot \frac{e^{(-2\beta h)}}{\left(1 + e^{(-2\beta h)}\right)} \cdot 2\beta . \end{aligned} \quad (25)$$

Sendo

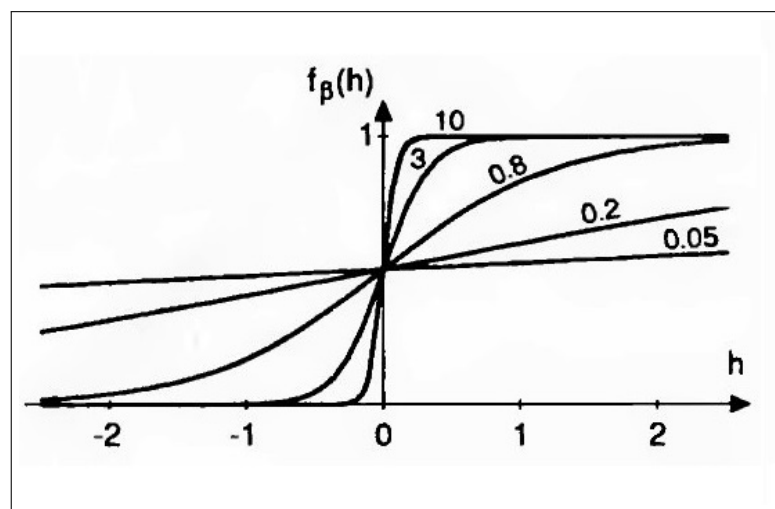
$$\frac{e^{(-2\beta h)}}{\left(1 + e^{(-2\beta h)}\right)} = 1 - \frac{1}{\left(1 + e^{(-2\beta h)}\right)} = (1 - g) , \quad (26)$$

a equação (25) pode ser reescrita como

$$g'(h) = 2\beta g(1 - g). \quad (27)$$

O comportamento da função sigmóide pode ser verificado também no gráfico apresentado na Figura 10.

Figura 10 - Função sigmóide para diversos valores do parâmetro β , que determina a inclinação da função próximo do valor $h = 0$.



Fonte: HERTZ, 1991.

2.6.2 Função Tangente Hiperbólica

A função tangente hiperbólica é contínua no intervalo de -1 a $+1$, é diferenciável e pode ser definida a partir do seno e cosseno hiperbólicos, como vemos a seguir

$$\tanh(\beta h) = \frac{\sinh(\beta h)}{\cosh(\beta h)}, \quad (28)$$

ou ainda

$$\tanh(\beta h) = \frac{e^{(\beta h)} - e^{-(\beta h)}}{e^{(\beta h)} + e^{-(\beta h)}}. \quad (29)$$

Para encontrar $g'(h)$ quando $g(h)$ é dado pela função tangente hiperbólica, devemos calcular a derivada de $g(h)$ da seguinte forma. Sendo

$$g(h) = \tanh(\beta h), \quad (30)$$

tomamos a derivada desta função, usando a forma dada pela equação (29)

$$\begin{aligned} \frac{\partial}{\partial(\beta h)} \left(\frac{e^{(\beta h)} - e^{-(\beta h)}}{e^{(\beta h)} + e^{-(\beta h)}} \right) &= \frac{(e^{(\beta h)} + e^{-(\beta h)})(e^{(\beta h)} + e^{-(\beta h)}) - (e^{(\beta h)} - e^{-(\beta h)})(e^{(\beta h)} - e^{-(\beta h)})}{(e^{(\beta h)} + e^{-(\beta h)})^2}, \\ &= \frac{(e^{(\beta h)} + e^{-(\beta h)})^2 - (e^{(\beta h)} - e^{-(\beta h)})^2}{(e^{(\beta h)} + e^{-(\beta h)})^2}, \\ &= \frac{(e^{(\beta h)} + e^{-(\beta h)})^2}{(e^{(\beta h)} + e^{-(\beta h)})^2} - \frac{(e^{(\beta h)} - e^{-(\beta h)})^2}{(e^{(\beta h)} + e^{-(\beta h)})^2}. \end{aligned} \quad (31)$$

Considerado a equação (29), a equação (31) pode ser reescrita como

$$g'(\beta h) = 1 - \tanh(\beta h)^2, \quad (32)$$

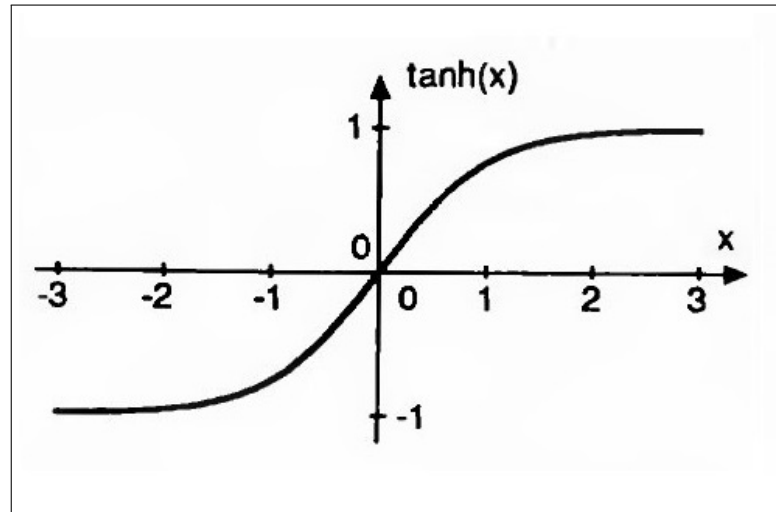
de forma que

$$\frac{\partial}{\partial h} (1 - \tanh(\beta h)^2) = \beta \cdot (1 - g^2), \quad (33)$$

ou ainda

$$g'(h) = \beta(1 - g^2). \quad (34)$$

Figura 11 - A função tangente hiperbólica definida no intervalo de -1 a $+1$.



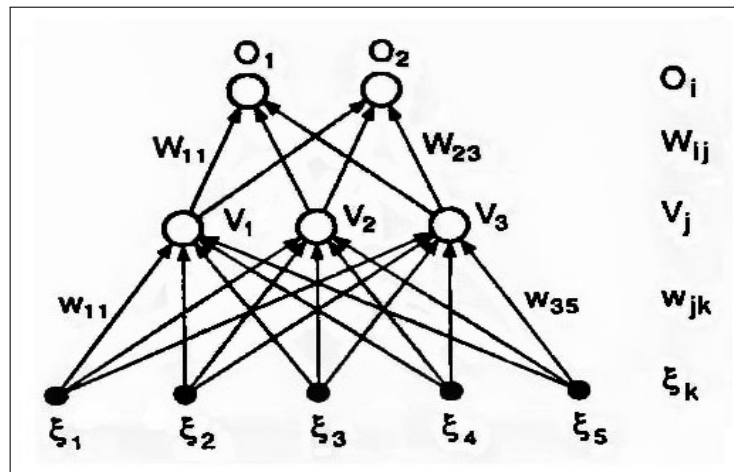
Fonte: HERTZ, 1991.

Observamos que a função tangente hiperbólica $g(h) = \tanh(\beta h)$, para unidades com saída entre -1 e $+1$ é particularmente conveniente, pois sua derivada é dada por $g'(h) = \beta(1 - g^2)$. Assim, não é necessário recalculá-la em (23), uma vez que já se calculou $O_i^\mu = g(h_i^\mu)$. O mesmo se aplica à função sigmóide $g(h) = [1 + \exp(-2\beta h)]^{-1}$ para unidades com saídas entre 0 e 1, sendo sua derivada dada por $g'(h) = 2\beta g(1 - g)$. Na rede neuronal do tipo perceptron simples, a principal vantagem da função de ativação não-linear é poder manter a saída entre limites fixos, como ± 1 para uma função tangente hiperbólica. O estudo do caso não-linear ganha mais notoriedade nas redes feedforward de múltiplas camadas, onde a restrição de que os padrões de entrada precisam ser linearmente independentes não se aplica, possibilitando a solução de problemas que não são possíveis com as redes feedforward de uma camada.

3 PERCEPTRON MULTICAMADAS

² Após a demonstração de Minsky e Papert, de que apenas funções linearmente separáveis podem ser representados por uma rede neuronal artificial do tipo perceptron (MINSKY; PAPERT, 1969), muitos pesquisadores perderam o interesse nas pesquisas sobre redes feedforward, ocasionando um declínio da atividade de pesquisa na área de redes neurais como um todo. Embora o maior poder das redes feedforward de múltiplas camadas tenha sido percebido há muito tempo, apenas no final da década de 80, foi demonstrado como se pode fazê-las aprender uma função específica, usando a retropropagação dos erros (RUMELHART; HINTON; WILLIAMS, 1986). As limitações de uma rede neuronal do tipo perceptron simples não se aplicam à rede feedforward com camadas intermediárias ou “escondidas”. As camadas escondidas ficam localizadas entre a camada de entrada e a camada de saída. Uma rede com apenas uma camada escondida pode representar qualquer função binária. A rede neuronal do tipo perceptron multicamadas possui um grande poder computacional, devido a suas camadas internas. É importante ressaltar que a camada de entrada apenas distribui os sinais pela rede, e não realiza nenhuma operação matemática ou computacional.

Figura 12 - Rede feedforward de duas camadas mostrando as notações para as unidades e conexões.



Fonte: HERTZ, 1991.

² Este capítulo é baseado na referência (HERTZ; KROGH; PALMER, 1991).

A Figura 12 mostra uma rede onde, um determinado neurônio em qualquer camada da rede está conectado a todos os neurônios da camada anterior, por uma conexão (aresta) eferente. O fluxo do sinal através da rede progride para frente, de cada camada para a seguinte.

Consideraremos inicialmente a rede a rede de duas camadas, ilustrada na Figura 12. Nossas convenções notacionais estão ilustradas na figura; as unidades de saída são denotadas por O_i , as unidades escondidas por V_j , e os terminais de entrada por ξ_k . Os pesos w_{jk} fazem a conexão das unidades de entrada com as unidades escondidas, e os pesos W_{ij} a conexão das unidades escondidas com as unidades de saída. Podemos observar que o índice i sempre se refere a uma unidade de saída, j a uma unidade escondida, e k a uma unidade de entrada. As entradas ξ_k são sempre fixadas em valores específicos, e cada padrão diferente é rotulado por um sobrescrito μ , portanto a entrada k é definida como ξ_k^μ quando o padrão μ é apresentado. Os ξ_k^μ s podem assumir valores binários (0/1 ou ± 1) ou valores contínuos. Usamos N para o número de unidades de entrada e p para o número de padrões de entrada ($\mu = 1, 2, \dots, p$).

Dado o padrão μ , a unidade escondida j recebe a entrada ponderada h_j^μ

$$h_j^\mu = \sum_k w_{jk} \xi_k^\mu, \quad (35)$$

e produz a saída

$$V_j^\mu = g(h_j^\mu) = g\left(\sum_k w_{jk} \xi_k^\mu\right). \quad (36)$$

A unidade de saída i então recebe

$$h_i^\mu = \sum_j W_{ij} V_j^\mu = \sum_j W_{ij} g\left(\sum_k w_{jk} \xi_k^\mu\right), \quad (37)$$

e produz como saída final

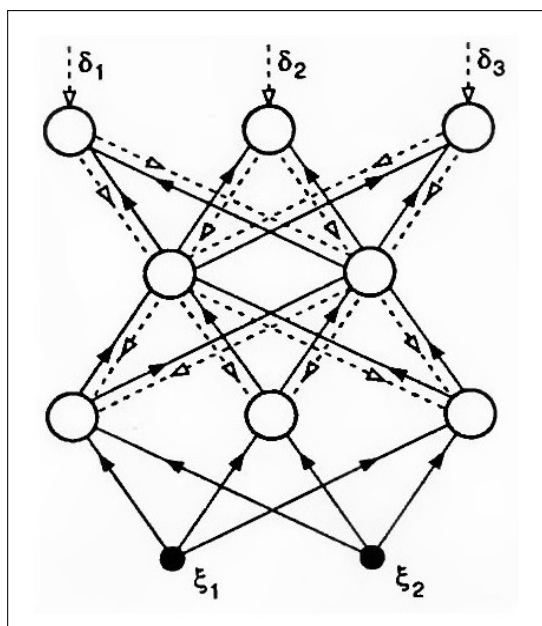
$$O_i^\mu = g(h_i^\mu) = g\left(\sum_j W_{ij} V_j^\mu\right) = g\left(\sum_j W_{ij} g\left(\sum_k w_{jk} \xi_k^\mu\right)\right). \quad (38)$$

3.1 Backpropagation

De acordo com Rumelhart e colaboradores (RUMELHART; HINTON; WILLIAMS, 1986), o algoritmo de backpropagation é um procedimento que ajusta repetidamente os pesos das conexões da rede, de modo a minimizar a diferença entre a saída calculada da rede e a saída desejada. Como resultado dos ajustes de peso, as unidades

das camadas escondidas que não fazem parte nem da entrada e nem saída, passam a representar características importantes do domínio da tarefa, e as regularidades na tarefa são capturadas pelas interações dessas unidades. Lecun e colaboradores (LECUN et al., 1998), afirmam que o backpropagation é um algoritmo de aprendizado de rede neuronal artificial muito popular porque ele é conceitualmente simples, computacionalmente eficiente, e porque muitas vezes funciona. No entanto, fazê-lo funcionar bem pode parecer mais uma arte do que uma ciência. Projetar um treinamento em uma rede, usando backpropagation exige a realização de escolhas aparentemente arbitrárias, como o número de neurônios, número de camadas, taxa de aprendizado, conjunto de treinamento, conjunto de teste e assim por diante. Essas escolhas podem ser críticas, mas não há uma receita infalível para realizá-las, porque são em grande parte dependentes do problema e dos dados em questão. No entanto, existem heurísticas e algumas teorias subjacentes que podem ajudar a orientar um praticante a fazer as melhores escolhas.

Figura 13 - Como a retropropagação dos erros é realizada em uma rede feedforward de três camadas.



Fonte: HERTZ, 1991.

Durante o treinamento da rede com o algoritmo de backpropagation, a rede opera em uma sequência de duas etapas, como mostra a Figura 13. A primeira etapa envolve a propagação da informação para cima (para frente), cujo primeiro passo consiste em apresentar um padrão à camada de entrada da rede. A informação resultante da atividade de processamento flui através da rede, camada por camada, até que a resposta seja produzida pela camada de saída. A segunda etapa corresponde à descida, onde a saída calculada é

comparada à saída desejada para o padrão em questão. Se a saída calculada for diferente da saída desejada, o erro é calculado e propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões dos neurônios das camadas intermediárias vão sendo modificados através da regra delta generalizada, conforme o erro é retropropagado.

3.1.1 Função Custo

A função custo (13) utilizada na a rede neuronal do tipo perceptron simples, é modificada de acordo com a nova forma de calcular a saída O_i^μ e torna-se, para a rede neuronal do tipo perceptron multicamadas,

$$E[\mathbf{w}] = \frac{1}{2} \sum_{\mu i} \left[\zeta_i^\mu - g \left(\sum_j W_{ij} g \left(\sum_k w_{jk} \xi_k^\mu \right) \right) \right]^2. \quad (39)$$

A nova função custo obtida através da equação (39) é claramente uma função diferenciável e contínua de cada peso, de forma que podemos usar o algoritmo do gradiente descendente para aprender os pesos apropriados para cada conexão. Isso é tudo o que o algoritmo de backpropagation propõe, mas existe uma grande importância prática na forma das regras de atualização resultantes.

3.1.2 Ajuste dos Pesos

A regra de atualização para o ajuste dos pesos das conexões da camada escondida para a camada de saída é dada pela taxa de aprendizagem η e pela derivada da função custo obtida na equação (39), em relação aos pesos das conexões da camada escondida para a camada de saída, como mostramos a seguir.

$$\begin{aligned} \Delta W_{lm} &= -\eta \frac{\partial E}{\partial W_{lm}} \\ &= -\eta \frac{\partial}{\partial W_{lm}} \frac{1}{2} \sum_{\mu i} \left[\zeta_i^\mu - g \left(\sum_j W_{ij} g \left(\sum_k w_{jk} \xi_k^\mu \right) \right) \right]^2 \\ &= -\eta \cdot 2 \cdot \frac{1}{2} \sum_{\mu} \left[\zeta_l^\mu - O_l^\mu \right] \frac{\partial}{\partial W_{lm}} \left[\zeta_l^\mu - g \left(\sum_j W_{lj} g \left(\sum_k w_{jk} \xi_k^\mu \right) \right) \right] \\ &= -\eta \sum_{\mu} \left[\zeta_l^\mu - O_l^\mu \right] \cdot \left(-g' \left(h_l^\mu \right) \right) V_m^\mu \\ &= \eta \sum_{\mu} \left[\zeta_l^\mu - O_l^\mu \right] g' \left(h_l^\mu \right) V_m^\mu, \end{aligned}$$

ou, definindo $\delta_i^\mu = [\zeta_i^\mu - O_i^\mu] g'(h_i^\mu)$,

$$\Delta W_{ij} = \eta \sum_{\mu} \delta_i^\mu V_j^\mu. \quad (40)$$

Podemos observar que o delta δ_i^μ da equação (40) é idêntico ao obtido anteriormente para a rede neuronal do tipo perceptron simples, na equação (23). A equação (40) é bastante similar à equação (17), porém com a saída da unidade escondida V_j^μ desempenhando o papel da entrada do perceptron, que na rede neuronal do tipo simples é desempenhado por ξ_k^μ .

A regra de atualização para o ajuste dos pesos das conexões das unidades de entrada para as camadas escondidas é realizada diferenciando a função custo (39), em relação aos $w_{jk}'s$, usando a regra da cadeia

$$\begin{aligned} \Delta w_{jk} &= -\eta \frac{\partial E}{\partial w_{jk}} \\ &= -\eta \sum_{\mu} \frac{\partial E}{\partial V_j^\mu} \frac{\partial V_j^\mu}{\partial w_{jk}} \\ &= \eta \sum_{\mu i} [\zeta_i^\mu - O_i^\mu] g'(h_i^\mu) W_{ij} g'(h_j^\mu) \xi_k^\mu \\ &= \eta \sum_{\mu i} \delta_i^\mu W_{ij} g'(h_j^\mu) \xi_k^\mu \\ \Delta w_{jk} &= \eta \sum_{\mu} \delta_j^\mu \xi_k^\mu \end{aligned} \quad (41)$$

com,

$$\delta_j^\mu = g'(h_j^\mu) \sum_i W_{ij} \delta_i^\mu. \quad (42)$$

Podemos notar que a equação (41) possui a mesma forma da equação (40), mas com uma definição diferente dos $\delta's$. A regra geral para atualização dos pesos, para um número arbitrário de camadas, tem a forma

$$\Delta w_{pq} = \eta \sum_{\text{padrões}} \delta_{\text{saída}} \mathbf{x} V_{\text{entrada}}, \quad (43)$$

onde saída e entrada referem-se às duas extremidades p e q da conexão em questão, e V significa a ativação da extremidade de entrada de uma unidade escondida ou o valor de uma unidade da camada de entrada. O significado de δ depende da camada em questão, para a última camada de conexões é dada pela equação (23), enquanto para todas as outras camadas é dada pela equação (42).

3.1.3 O Algoritmo

Como o backpropagation é um procedimento bastante importante para a rede neuronal do tipo perceptron multicamadas, apresentaremos os passos do algoritmo, tomando um padrão μ de cada vez. Consideramos uma rede com M camadas $m = (1, 2, \dots, M)$ e usaremos V_i^m para a saída da i -ésima unidade na m -ésima camada. Desse modo, V_i^0 será sinônimo de ξ_i para a i -ésima entrada. O peso w_{ij}^m significa a conexão da unidade V_j^{m-1} para a unidade V_i^m . Assim o procedimento de backpropagation é:

1. Inicializar os pesos com valores aleatórios pequenos.
2. Escolher um padrão ξ_k^μ e aplicá-lo à camada de entrada ($m = 0$) de modo que

$$V_k^0 = \xi_k^\mu \quad (\text{para todo } k). \quad (44)$$

3. Propagar o sinal para frente através da rede, usando a regra

$$V_i^m = g(h_i^m) = g\left(\sum_j w_{ij}^m V_j^{m-1}\right), \quad (45)$$

para cada neurônio i e cada camada m , até que as saídas finais V_i^M tenham sido calculadas.

4. Calcular os deltas para a camada de saída

$$\delta_i^M = g'(h_i^M) [\zeta_i^\mu - V_i^M], \quad (46)$$

comparando as saídas atuais V_i^M com as saídas desejadas ζ_i^μ para o padrão μ em questão.

5. Calcular os deltas para as camadas anteriores através da retropropagação dos erros

$$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m, \quad (47)$$

para $m = (M, M-1, \dots, 2)$ até que todos os deltas já tenham sido calculados para cada unidade. Calculamos os deltas até a camada 2, pois a camada 1 é entendida como a entrada dos sinais.

6. Usar a equação

$$\Delta w_{ij}^m = \eta \delta_i^m V_j^{m-1}, \quad (48)$$

para atualizar todas os pesos sinápticos de acordo com $w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$.

7. Voltar para o passo 2 e repetir para o próximo padrão.

4 REGRESSÃO LOGÍSTICA

A regressão logística é um dos algoritmos mais conhecidos e mais utilizados para classificação de padrões. A função logística foi descoberta no século XIX por Pierre-François Verhulst (1804 – 1849) para descrever o crescimento das populações. A ideia básica da função logística é simples e usada nos dias atuais para descrever a chance de ocorrência de um determinado evento (CRAMER, 2002). O método da regressão logística ganhou maior notoriedade após a segunda metade do século XX e obteve um grande avanço a partir das publicações (COX; SNELL, 1989) e (HOSMER; LEMESHOW, 2000). O método da regressão logística caracteriza-se por descrever a relação entre uma variável dependente binária e qualitativa, e um conjunto de variáveis independentes. A regressão logística além de explicar a relação entre as variáveis, possui algumas características que em conjunto tornam o método diferente dos demais modelos de regressão tradicionais, pois a variável dependente é qualitativa binária e segue uma distribuição de Bernoulli. Como se trata de uma sequência de eventos com distribuição de Bernoulli, a soma do número de sucessos ou fracassos nessa experiência terá distribuição Binomial com parâmetros n e π_i , onde n é o número de eventos, π_i é a probabilidade de sucesso e $1 - \pi_i$ é a probabilidade de fracasso. Nesse método de classificação, a probabilidade de ocorrência de um evento pode ser estimada diretamente. A variável dependente Y_i assume apenas dois possíveis estados, sucesso com probabilidade $P(Y_i = 1) = \pi_i$ e fracasso com probabilidade $P(Y_i = 0) = 1 - \pi_i$, para todo $i = 1, 2, \dots, m$ e para valores específicos de p variáveis independentes $x_1^i, x_2^i, x_3^i, \dots, x_p^i$. A função logística pode então ser escrita da seguinte forma

$$\pi(\mathbf{x}) = \frac{e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}}. \quad (49)$$

4.1 Transformação Logit

A transformação logit é fundamental no estudo do modelo da regressão logística e seu objetivo é linearizar o modelo, aplicando o logaritmo. Essa transformação define-se como:

$$f(\mathbf{x}) = \ln \left(\frac{\pi_i}{1 - \pi_i} \right),$$

$$f(\mathbf{x}) = \ln \left(\frac{\frac{e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}}}{1 - \frac{e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}}} \right) = \ln \left(\frac{\frac{e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}}}{1} \right),$$

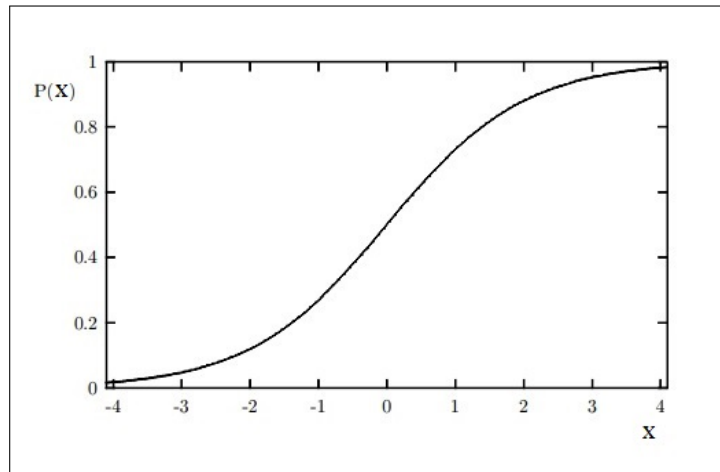
$$\begin{aligned} f(\mathbf{x}) &= \ln(e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}), \\ f(\mathbf{x}) &= \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p. \end{aligned} \tag{50}$$

O modelo com essa transformação possui diversas propriedades similares às do modelo de regressão linear. Por exemplo, $f(\mathbf{x})$ depende linearmente das variáveis x_i , pode ser contínua e pode variar no intervalo $(-\infty, +\infty)$ (HOSMER; LEMESHOW, 2000).

4.2 Estimação dos Parâmetros

Os coeficientes $\beta_0, \beta_1, \dots, \beta_p$, são estimados a partir do conjunto de treinamento, pelo método da máxima verossimilhança, no qual encontramos uma combinação de coeficientes que maximiza a probabilidade da amostra ter sido observada (MOOD; GRAYBILL; BOES, 1974). Considerando valores específicos dos coeficientes $\beta_0, \beta_1, \dots, \beta_p$, em função das variáveis explicativas, observa-se que a curva logística tem um comportamento probabilístico no formato da letra *S*, como pode ser observado na Figura 14 (HOSMER; LEMESHOW, 2000).

Figura 14 - Comportamento da curva da regressão logística.



Fonte: CRAMER, 2002.

A regra para discriminar dois grupos diferentes, utilizando o método da regressão logística é mostrado a seguir.

- Se $\pi_i \geq 0.5$, então o evento é classificado como $Y_i = 1$.
- Se $\pi_i < 0.5$, então o evento é classificado como $Y_i = 0$.

Como no perceptron de múltiplas camadas, para uma melhor classificação utilizando o método da regressão logística, recomenda-se separar a amostra em duas partes, uma parte para o treinamento do modelo e a outra parte para testar a eficiência da classificação.

5 O PROBLEMA DE DEVEDORES DO BNDES

O Banco Nacional do Desenvolvimento Econômico e Social (BNDES) é uma empresa pública federal, vinculada ao Ministério do Desenvolvimento, Indústria e Comércio Exterior, que apoia e financia a longo prazo investimentos em diversos segmentos econômicos como agricultura, indústria, infraestrutura, comércio e serviços, além de investimentos sociais nas áreas de educação, saúde, agricultura familiar e outras. Podem solicitar financiamento empresas sediadas no Brasil, pessoas físicas residentes no país e entidades da administração pública.

Desde o início dessa dissertação, abrimos caminho para discutir a rede neuronal do tipo perceptron simples e posteriormente o perceptron multicamadas, a fim de utilizar essas técnicas de uma forma prática. Utilizando os métodos previamente descritos baseados no algoritmo de backpropagation, criamos em linguagem C um programa para simular a rede neuronal do tipo perceptron multicamadas e fizemos a comparação dos resultados obtidos com outro método de classificação baseado na regressão logística.

Nos foi cedida pelo BNDES, uma base de dados com informações sobre 6601 clientes que adquiriram financiamento para investimentos. O banco de dados fornece informações sobre estes devedores e, em particular, revela a situação de adimplência ou inadimplência de cada cliente. A cada cliente que realizou algum tipo de financiamento para investimentos são atribuídos valores para oito parâmetros que o caracterizam como devedor. Utilizamos estes parâmetros para realizar a classificação de cada devedor como adimplente ou inadimplente, com o auxílio da rede neuronal do tipo perceptron multicamadas, com a finalidade de reduzir a inadimplência. Sabendo que aproximadamente 80% dos devedores são adimplentes e 20% são inadimplentes, e obedecendo a essas proporções, construímos um conjunto de treinamento e um conjunto de teste, com interseção nula, a fim de proporcionar um melhor treinamento para a rede neuronal do tipo perceptron multicamadas e para a regressão logística.

5.1 Descrição das Variáveis

A base de dados do BNDES foi construída com clientes que realizaram financiamento para investimentos a partir do ano de 2012, e incrementada ano a ano até o ano de 2016. No ano de 2012, a base continha 4526 clientes, em 2013 foram adicionados 1031 clientes, em 2014 foram adicionados mais 784 clientes, em 2015 foram adicionados mais 50 clientes e em 2016 foram adicionados mais 210 clientes. Do total de 6601 clientes que compõem a base do ano de 2012 até 2016, 5223 são clientes adimplentes e 1378 são clientes inadimplentes. Oito variáveis foram selecionadas para descrever a situação des-

ses clientes e serão utilizadas como as entradas do modelo de classificação. As variáveis são: porte, finalidade, CEP, setor, idade, pontualidade, restrições e refin. Ao fazer a análise exploratória de dados sobre a base do BNDES, observamos que em alguns casos clientes distintos apresentavam os mesmos valores de suas variáveis de entrada, apresentando no entanto valores distintos de sua variável de saída, ou seja, alguns clientes tinham os mesmos valores para as oito variáveis que caracterizavam o cliente, sendo que alguns destes eram classificados como adimplentes e outros como inadimplentes, desta forma introduzindo bastante incerteza para o modelo de classificação. Para reduzir os efeitos da incerteza causada pela variabilidade de respostas, adicionamos mais duas variáveis. Uma variável relacionada com a probabilidade do cliente ser adimplente e outra variável relacionada com a probabilidade do cliente ser inadimplente, ambas as variáveis foram estimadas considerando as proporções observadas no conjunto de treinamento. Desta forma, totalizamos dez variáveis que caracterizam cada cliente do BNDES que resultou ser adimplente ou inadimplente.

1. Porte: Esta variável pode assumir três valores discretos, correspondendo às três categorias: micro, pequena e média empresa. Devido à quantidade pequena de empresas com valor de porte pequeno e médio, estas duas categorias foram fundidas. Quando o cliente é de uma microempresa, pertence à categoria 1 e quando o cliente é de uma pequena ou média empresa, pertence à categoria 2.
2. Finalidade: Esta variável se refere à finalidade do financiamento. Observou-se dentro da amostra, grandes diferenças nas frequências de inadimplência em diferentes tipos de finalidades de financiamento. Quando a finalidade do financiamento é de capital de giro, investimento e inovação, o cliente pertence à categoria 1. Pertence à categoria 2, os clientes cuja finalidade do financiamento é para a aquisição de máquinas, ônibus, caminhões e clientes sem informação do tipo de investimento.
3. CEP: Esta variável foi construída com os dois primeiros números do CEP (Código de Endereçamento Postal) da localização da empresa. Pertencem à mesma categoria os clientes que possuem os dois primeiros dígitos de CEP iguais. Inicialmente, foram criadas oito categorias de CEP. Posteriormente, foi necessário fazer agrupamentos, de forma que restaram três categorias.
4. Setor: A variável setor foi construída com os dois primeiros dígitos do CNAE (Classificação Nacional de Atividades Econômicas), conforme a classificação do IBGE (Instituto Brasileiro de Geografia e Estatística). A metodologia é muito similar à utilizada no caso da variável CEP. Inicialmente foram criadas oito categorias, mas houve necessidade de fundir algumas categorias, restando três categorias.
5. Idade: A variável idade foi construída com base na diferença entre a data da consulta e a data de quando o cliente foi registrado nos bancos de dados da Serasa. A

categoria 1 representa os clientes que estão há mais de cinco anos registrados na Serasa. A categoria 2 representa os clientes que estão há menos de cinco anos registrados na Serasa em relação à data da consulta.

6. Pontualidade: A pontualidade é uma variável que descreve hábitos de pagamento, e se refere a contas já pagas, mas que podem ter sido pagas com pontualidade ou com atraso. Na categoria 1, estão as empresas que pagaram suas contas com 100% de pontualidade no último ano. Na categoria 2, estão os clientes que tiveram qualquer atraso de pagamentos no último ano.
7. Restrições: A variável restrição revela se a empresa possui qualquer tipo de apontamento, observação ou restrição em aberto no momento da consulta. Na categoria 1, estão as empresas que não possuem nenhuma restrição. Na categoria 2, estão as empresas que possuem algum tipo de restrição.
8. Refin: A variável refin representa o fato da empresa ter dívidas vencidas e não pagas e que são fornecidas pelas empresas participantes do convênio REFIN (bancos e financeiras). Na categoria 1, estão as empresas que não possuem dívidas com bancos e financeiras na data da consulta. Na categoria 2, estão as empresas com qualquer dívida em aberto com bancos e financeiras, de qualquer valor.
9. Probabilidade de adimplência: Quando um determinado vetor de entradas está associado a uma única saída, se a saída é adimplente, o elemento probabilidade de adimplência recebe o valor 1. Quando um determinado vetor de entradas está associado a uma única saída, se a saída é inadimplente, o elemento probabilidade de adimplência recebe o valor 0. Quando um mesmo vetor está associado à adimplente (Nadimp) para alguns clientes e à inadimplente (Ninadimp) para outros, esta variável corresponde a razão $\frac{Nadimp}{(Nadimp + Ninadimp)}$.
10. Probabilidade de inadimplência: Quando um determinado vetor de entradas está associado a uma única saída, se a saída é inadimplente, o elemento probabilidade de inadimplência recebe o valor 1. Quando um determinado vetor de entradas está associado a uma única saída, se a saída é adimplente, o elemento probabilidade de inadimplência recebe o valor 0. Quando um mesmo vetor está associado à adimplente (Nadimp) para alguns clientes e à inadimplente (Ninadimp) para outros, esta variável corresponde a razão $\frac{Ninadimp}{(Ninadimp + Nadimp)}$.

5.2 Normalização dos dados de entrada

A normalização dos dados de entrada é o processo pelo qual todos os dados são projetados nos intervalos $[0, 1]$ ou $[-1, 1]$. Caso a normalização não seja realizada, os

dados de entrada terão efeitos desproporcionais sobre os neurônios, levando a decisões erradas, pois não é possível comparar valores com diferentes ordens de magnitude. No conjunto de dados do BNDES, as variáveis descritivas, que são as variáveis de entrada do nosso modelo, representam grandezas qualitativas, e algumas variáveis como CEP e setor possuem três categorias, enquanto as demais possuem apenas duas categorias. A fórmula de normalização, é a seguinte:

$$y = \frac{(x - x_{min})(d2 - d1)}{x_{max} - x_{min}} + d1. \quad (51)$$

Onde x é o valor a ser normalizado, x_{min} é o limite inferior do intervalo do domínio do valor x , x_{max} é o limite superior do intervalo do domínio do valor x , $d1$ é o intervalo inferior ao qual o valor x será projetado e $d2$ é o intervalo superior ao qual o valor x será projetado.

No banco de dados do BNDES as variáveis que representam duas categorias (porte, finalidade, idade, pontualidade, restrições e refin), após a normalização, a categoria 1 terá o valor -1 e a categoria 2 terá o valor 1 . Nas variáveis que representam três categorias (CEP e setor), após a normalização, a categoria 1 terá o valor -1 , a categoria 2 terá o valor 0 e a categoria 3 terá o valor 1 . A variável dependente (que é a resposta do modelo) representa duas categorias: adimplente e inadimplente. Para realizar o cálculo no perceptron multicamadas (onde a variável resposta pertence ao intervalo $[-1, 1]$), a categoria adimplente será representada pelo valor 1 e a categoria inadimplente será representada pelo valor -1 . Já para o modelo da regressão logística (onde a variável resposta pertence ao intervalo $[0, 1]$), a categoria adimplente será representada pelo valor 1 e a categoria inadimplente será representada pelo valor 0 .

6 DESEMPENHO DOS MÉTODOS DE CLASSIFICAÇÃO

Para avaliarmos o desempenho dos métodos de classificação utilizamos os testes diagnósticos. Eles são bastante utilizados na estatística e na medicina. Os testes diagnósticos referem-se a quanto, em termos quantitativos ou qualitativos, um teste é útil para diagnosticar uma característica da população ou para realizar uma previsão (AKOBENG, 2007). Para determinar a validade, compara-se os resultados do teste com os de um padrão já conhecido. O teste diagnóstico ideal deve fornecer, sempre, a resposta correta, ou seja, em nosso caso na rede neuronal do tipo perceptron multicamadas, um resultado positivo para os clientes que são adimplentes e um resultado negativo para os clientes que são inadimplentes. Para uma melhor avaliação do desempenho, é importante calcular a acurácia, a sensibilidade e a especificidade. Na Tabela 3, $Y = +1$ corresponde aos clientes adimplentes e $Y = -1$ corresponde aos clientes inadimplentes. O número de acertos, para cada classe, se localiza na diagonal principal da matriz, sendo VP (verdadeiro positivo) o caso onde o modelo classifica corretamente o cliente como adimplente e VN (verdadeiro negativo) o caso onde o modelo classifica corretamente o cliente como inadimplente. O número de erros, para cada classe, se localiza na diagonal secundária da matriz, sendo FP (falso positivo) o caso onde o modelo classifica erradamente o cliente como adimplente, quando na verdade o cliente era inadimplente e FN (falso negativo) o caso onde o modelo classifica erradamente o cliente como inadimplente, quando na verdade o cliente era adimplente.

Tabela 3 - Matriz de confusão ou tabela de contingência em que na linha está o valor previsto e na coluna o valor real.

		Valor Real	
		$Y = +1$	$Y = -1$
Valor Previsto	$Y = +1$	VP	FP
	$Y = -1$	FN	VN

Fonte: O autor, 2019.

Utilizamos os testes diagnósticos relacionados nesta seção para avaliar o desempenho dos métodos de classificação, para o perceptron multicamadas e para o método da regressão logística. As grandezas que caracterizam o desempenho dos métodos são as seguintes.

- Acurácia

Acurácia é a capacidade que o método possui para detectar os clientes verdadeiramente positivos e os clientes verdadeiramente negativos. Em nosso caso, classificar corretamente os clientes adimplentes e os clientes inadimplentes. A acurácia é definida por:

$$ACC = \frac{VP+VN}{VP+FP+FN+VN} . \quad (52)$$

- Sensibilidade

Sensibilidade é a capacidade que o método possui para detectar os clientes verdadeiramente positivos. Em nosso caso, classificar corretamente os clientes adimplentes. A sensibilidade é definida por:

$$S = \frac{VP}{VP+FN} . \quad (53)$$

- Especificidade

Especificidade é a capacidade que o método possui para detectar os clientes verdadeiramente negativos. Em nosso caso, classificar corretamente os clientes inadimplentes. A especificidade é definida por:

$$E = \frac{VN}{VN+FP} . \quad (54)$$

7 RESULTADOS

Com o auxílio do programa construído em linguagem C para simular a rede neuronal do tipo perceptron multicamadas, e treinada com o algoritmo de backpropagation e do software estatístico R para simular a regressão logística, realizamos diversos treinamentos para diferentes valores dos parâmetros que especificam os modelos, com a finalidade de reduzir a inadimplência entre os clientes que possuem acesso ao crédito, para obter financiamento no BNDES. Para isso, a base de dados do BNDES foi dividida em duas, aproximadamente 30% dos clientes para o conjunto de treinamento e aproximadamente 70% dos clientes para o conjunto de teste, escolhidos de forma aleatória utilizando de uma distribuição uniforme. O perceptron multicamadas foi treinado com o conjunto de treinamento e sua capacidade de generalização foi verificada com o conjunto de teste. Utilizando o software estatístico R, treinamos uma regressão logística com o mesmo conjunto de treinamento e verificamos a capacidade de generalização com o mesmo conjunto de teste utilizado para o perceptron multicamadas. Nas Seções 7.1 e 7.2, mostramos os diferentes resultados obtidos com o perceptron multicamadas e com a regressão logística respectivamente.

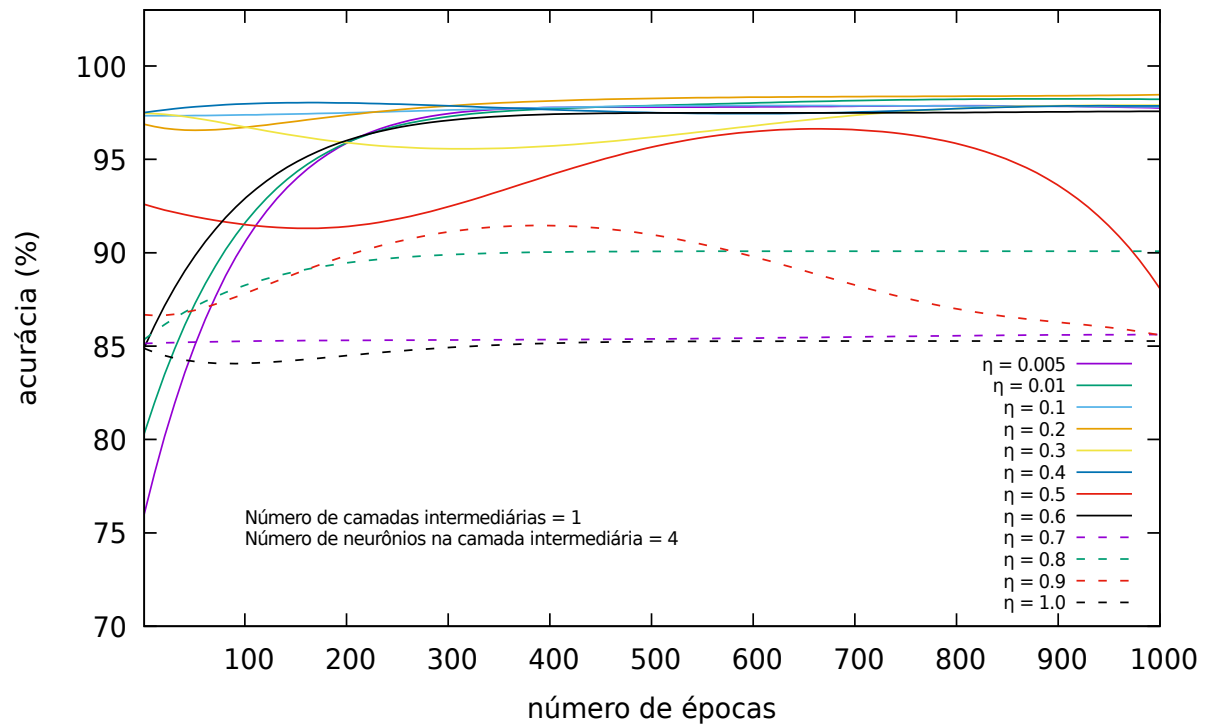
7.1 Perceptron Multicamadas

O perceptron multicamadas possui diversos parâmetros que podem ser ajustados com a finalidade de se obter o melhor resultado para a classificação de padrões. Para o perceptron multicamadas, realizamos medidas de desempenho para dois tipos de topologias. Utilizamos uma rede com uma camada intermediária e posteriormente uma rede com duas camadas intermediárias. Para cada tipo de topologia da rede, variamos o número de neurônios por camada, o número de épocas e a taxa de aprendizagem da rede neuronal. As diferentes topologias são descritas na forma (e, n_1, n_2, s) , onde e representa a quantidade de entradas do problema, n_1 representa o número de neurônios na primeira camada intermediária, n_2 o número de neurônios na segunda camada intermediária, e s representa o número de neurônios na camada de saída. Utilizamos as grandezas, acurácia, sensibilidade e especificidade para escolher quais configurações dos modelos garante o melhor resultado, na classificação dos padrões. Os resultados dos experimentos realizados são mostrados na Subseções 7.1.1 e 7.1.2.

7.1.1 Rede neuronal com 1 camada intermediária

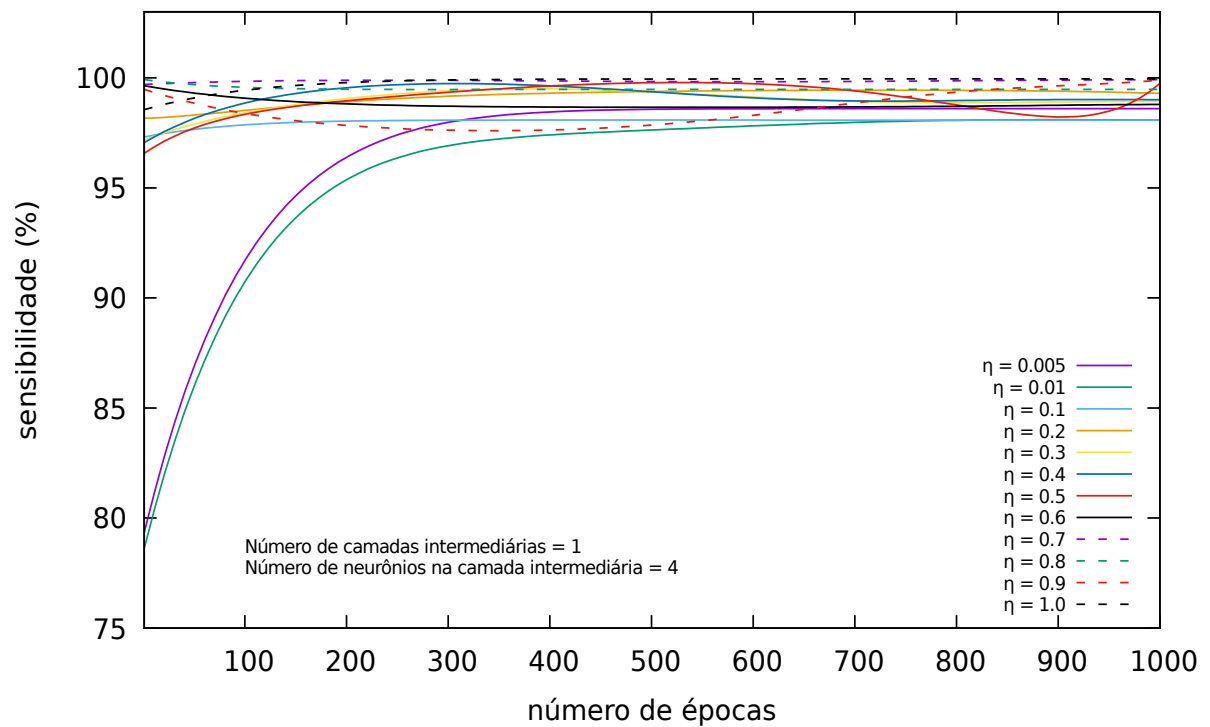
As Figuras 15, 16 e 17, são referentes à acurácia, sensibilidade e especificidade respectivamente, para a topologia (10, 4, 0, 1), que possui dez entradas, quatro neurônios na camada intermediária e um neurônio na camada de saída, para diferentes números de épocas, e cada curva do gráfico corresponde a uma taxa de aprendizagem diferente. Podemos observar na Figura 15, que a acurácia é bastante elevada para $\eta \leq 0.4$, enquanto para $\eta > 0.4$, a acurácia é bastante menor. A sensibilidade, que é a capacidade da rede classificar corretamente os clientes adimplentes, mostrada na Figura 16, atinge sua estabilidade a partir da época 300 e se mantém alta para qualquer valor da taxa de aprendizagem. Isto se deve ao fato de que o nosso conjunto de dados possui aproximadamente 80% de clientes adimplentes, sendo mais fácil para a rede aprender tais padrões. A especificidade, que é a capacidade da rede classificar corretamente os clientes inadimplentes, mostrada na Figura 17, possui bons resultados para taxas de aprendizagem próximas de zero e, por outro lado, taxas de aprendizagem próximas de $\eta = 1$ não conseguem captar bem os padrões dos clientes inadimplentes, resultando em valores ruins para a especificidade.

Figura 15 - Acurácia em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui uma camada intermediária com 4 neurônios.



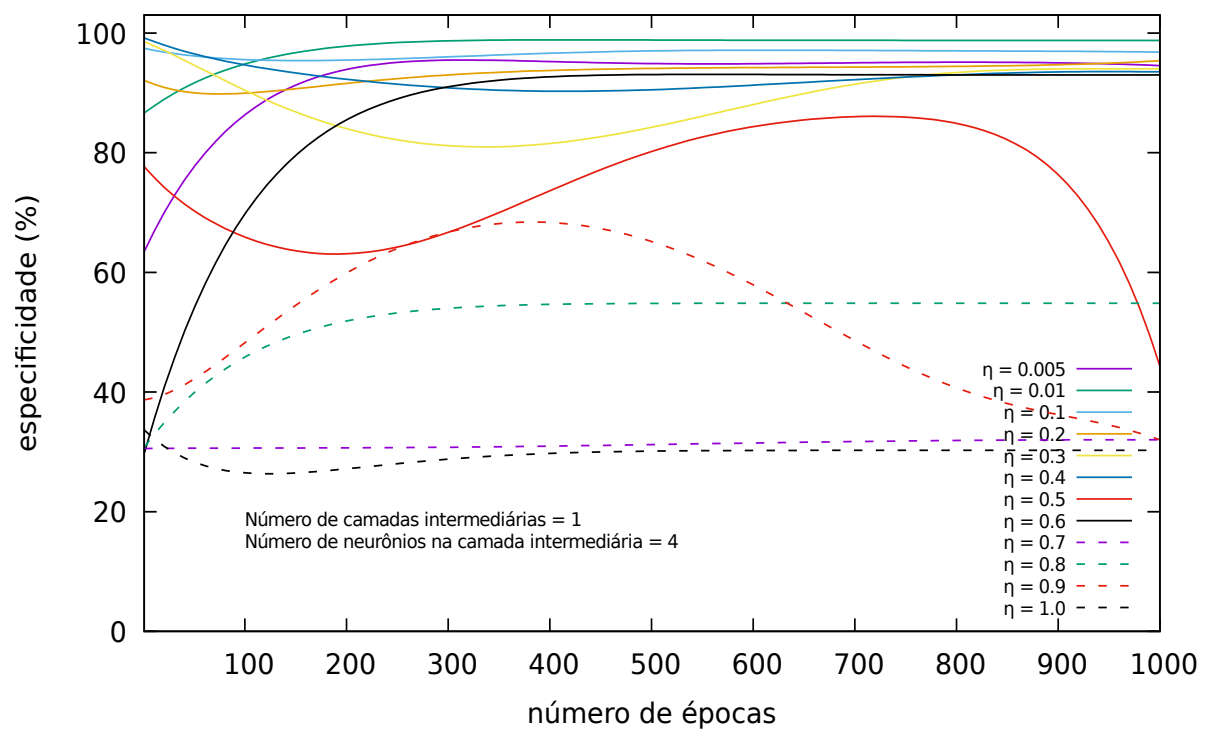
Fonte: O autor, 2019.

Figura 16 - Sensibilidade em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui uma camada intermediária com 4 neurônios.



Fonte: O autor, 2019.

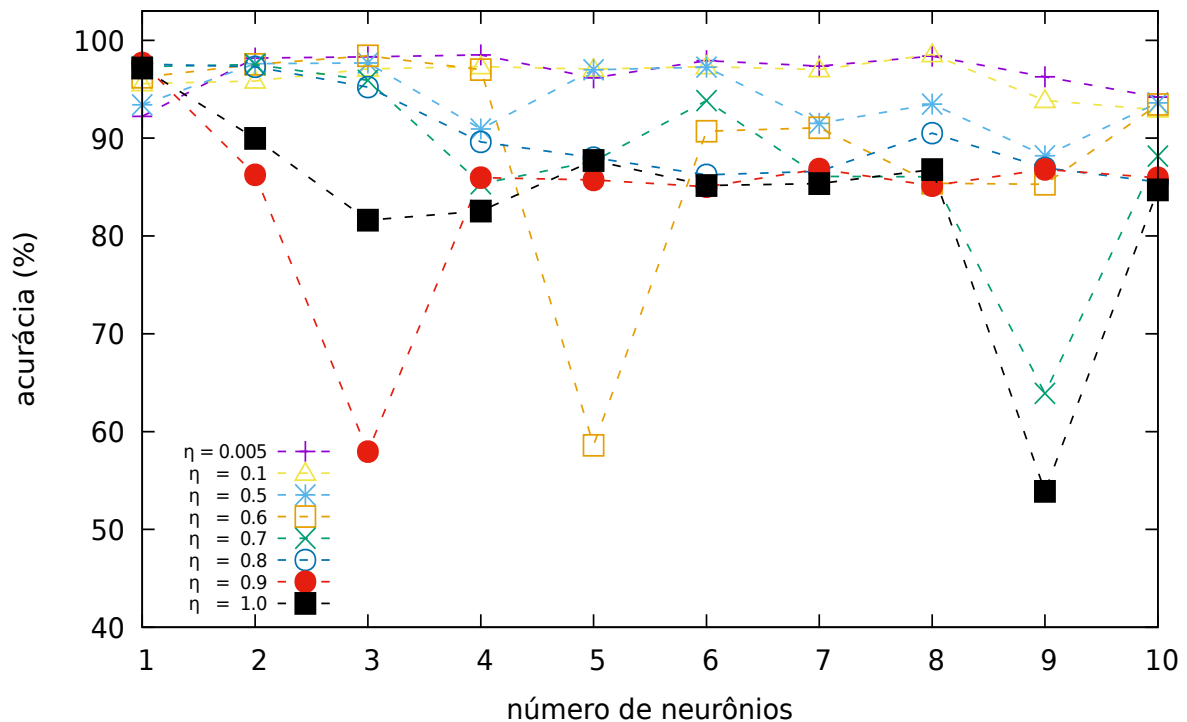
Figura 17 - Especificidade em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui uma camada intermediária com 4 neurônios.



Fonte: O autor, 2019.

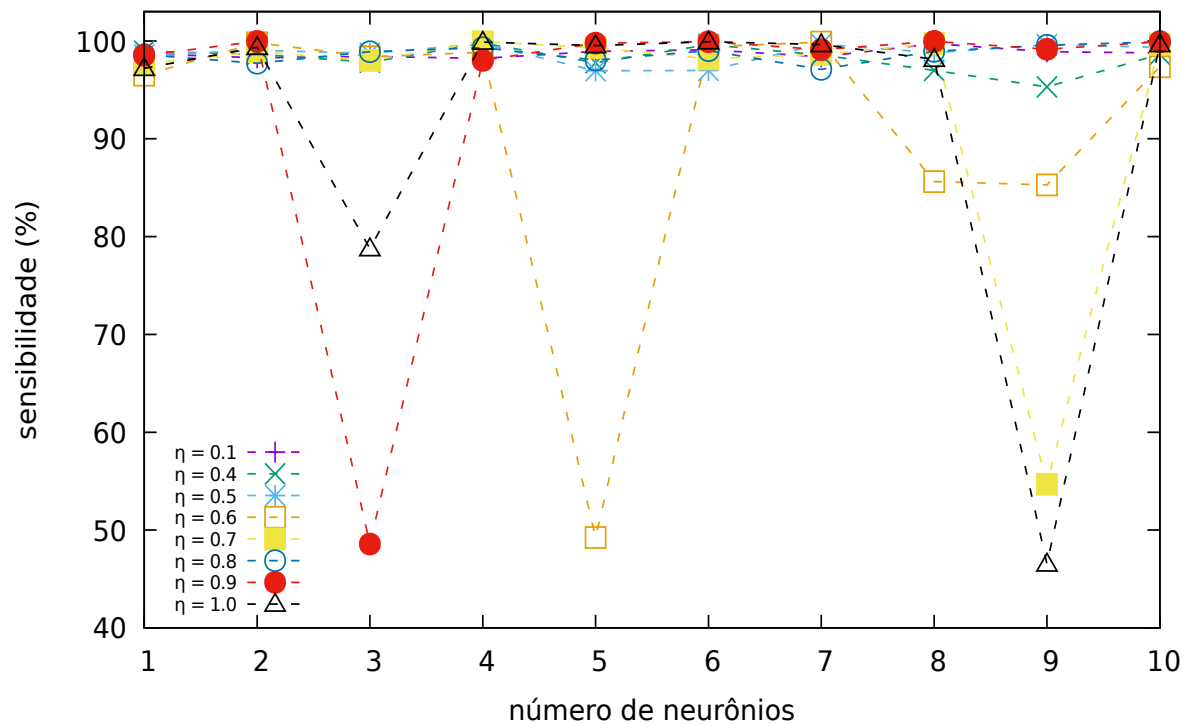
As Figuras 18, 19 e 20, são referentes à acurácia, sensibilidade e especificidade respectivamente, para uma topologia com dez entradas, um número diferente de neurônios na camada intermediária e um neurônio na camada de saída, e cada curva do gráfico corresponde a uma taxa de aprendizagem diferente. Como podemos observar na Figura 18, a acurácia se mantém alta para $\eta \leq 0.5$, mesmo com um número baixo de neurônios da camada intermediária. Já para taxas de aprendizagem próximas de $\eta = 1$, para diversos números de número de neurônios da camada intermediária, a acurácia do modelo diminui sensivelmente. A sensibilidade, como pode ser vista na Figura 19, se mantém alta para $\eta \leq 0.5$, para todos os valores de número de neurônios da camada intermediária. Já para $\eta > 0.5$, com exceção de $\eta = 0.8$, as curvas apresentam grande instabilidade conforme variamos o número de neurônios da camada intermediária, causando uma queda na sensibilidade do modelo, para diversas topologias. A especificidade, como podemos observar na Figura 20, é bastante sensível à variação do número de neurônios da camada intermediária, para taxas de aprendizagem $\eta < 0.5$, obtivemos resultados bastante satisfatórios, para diversos valores do número de neurônios da camada intermediária. Já para taxas de aprendizagem próximas de $\eta = 1$, a especificidade sofre uma redução, com o aumento do número de neurônios da camada intermediária.

Figura 18 - Acurácia em função do número de neurônios, para diferentes valores da taxa de aprendizagem. As curvas $\eta = 0.2$, $\eta = 0.3$ e $\eta = 0.4$ são semelhantes à curva $\eta = 0.1$.



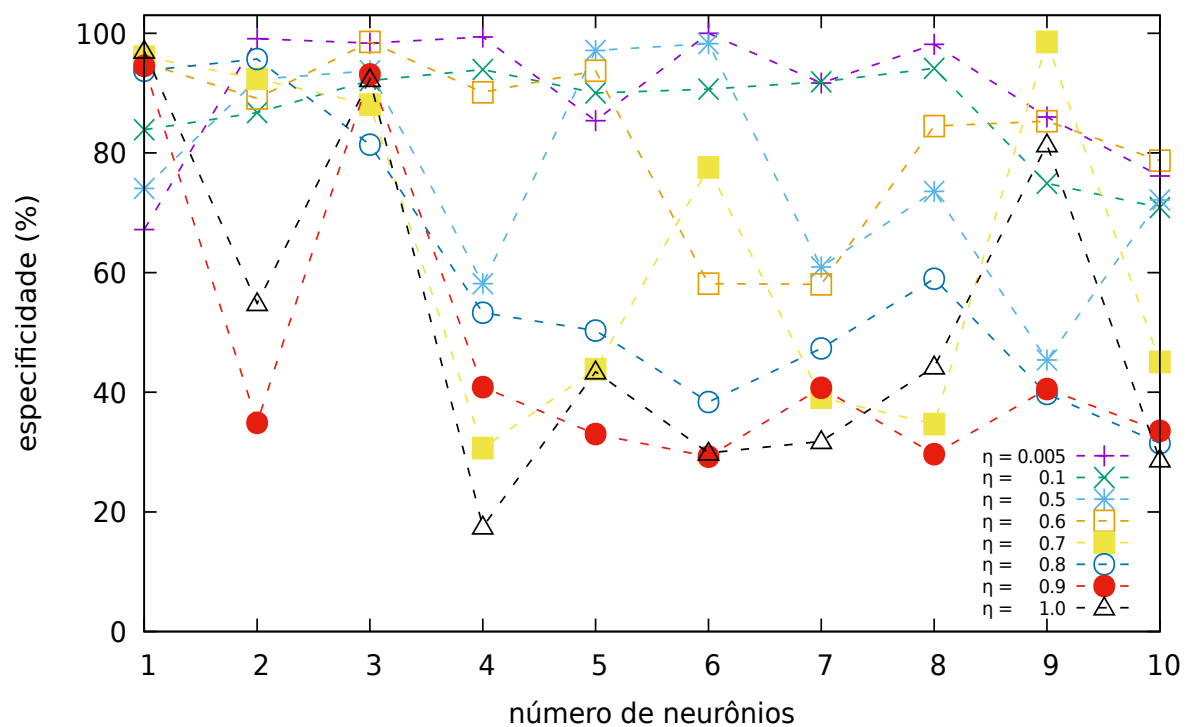
Fonte: O autor, 2019.

Figura 19 - Sensibilidade em função do número de neurônios, para diferentes valores da taxa de aprendizagem. As curvas $\eta = 0.2$ e $\eta = 0.3$ são semelhantes à curva $\eta = 0.1$.



Fonte: O autor, 2019.

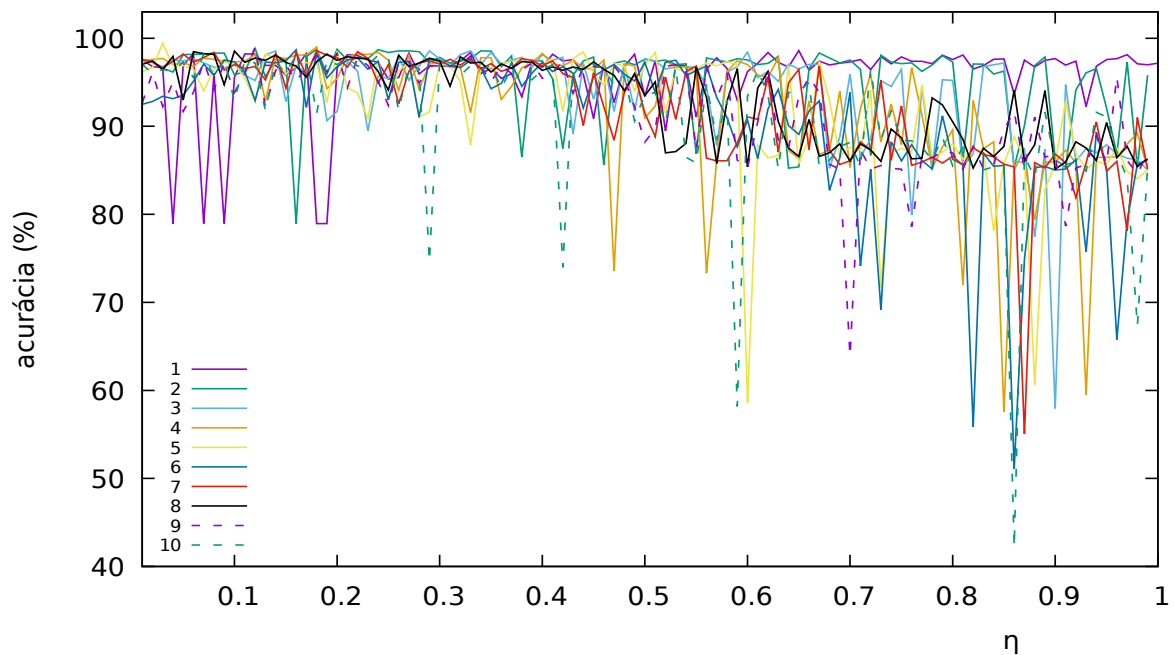
Figura 20 - Especificidade em função do número de neurônios, para diferentes valores da taxa de aprendizagem. As curvas $\eta = 0.2$, $\eta = 0.3$ e $\eta = 0.4$ são semelhantes à curva $\eta = 0.1$.



Fonte: O autor, 2019.

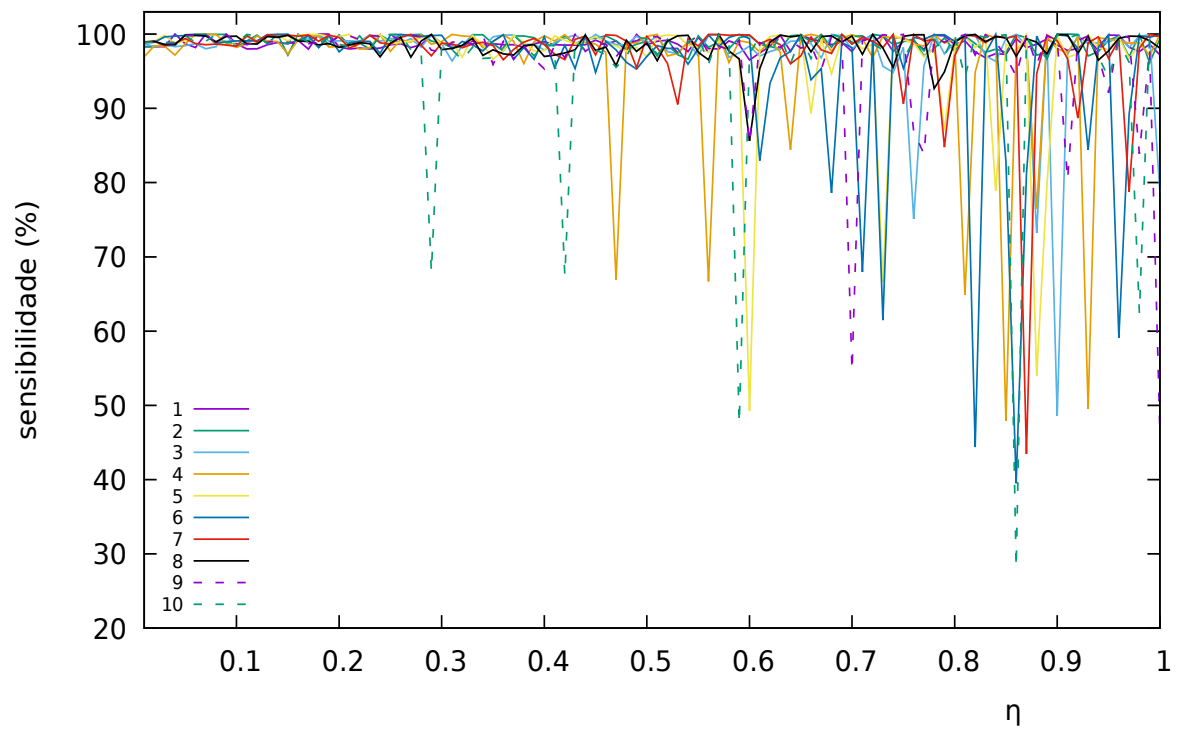
As Figuras 21, 22 e 23, são referentes à acurácia, sensibilidade e especificidade em função da variação da taxa de aprendizagem, e cada curva do gráfico corresponde a uma topologia com dez entradas, um número diferente de neurônios na camada intermediária e um neurônio na camada de saída. A acurácia como podemos observar na Figura 21, apresenta grande oscilação para taxas de aprendizagem próximas a zero e para topologias com poucos neurônios na camada intermediária, como é o caso da topologia (10, 1, 0, 1) e (10, 2, 0, 1). No geral, podemos observar que a acurácia é melhor para $\eta \leq 0.4$, independente do número de neurônios da camada intermediária. As topologias com um número maior de neurônios na camada intermediária e com $\eta > 0.4$ apresentam regiões de configurações de grande instabilidade com fortes quedas da acurácia do modelo. A sensibilidade, como pode ser vista na Figura 22, é bastante alta para $\eta \leq 0.3$, independente da topologia da rede, e para $\eta > 0.3$ a sensibilidade fica bastante instável, tendendo a ser pior para valores mais altos do número de neurônios, n_1 . A especificidade, como pode ser vista na Figura 23, fica bastante instável para $\eta \leq 0.2$ e para as topologias que possuem poucos neurônios na camada intermediária, como é o caso da topologia (10, 1, 0, 1) e (10, 2, 0, 1) que apresentam uma grande oscilação na especificidade, chegando a errar todas as classificações dos clientes inadimplentes. No geral, podemos observar que a especificidade é melhor para $0.2 \leq \eta \leq 0.4$, independente do número de neurônios da camada intermediária. Topologias com um número maior de neurônios na camada intermediária e $\eta > 0.4$ apresentam bastante redução na especificidade alcançada pelo modelo.

Figura 21 - Acurácia em função da taxa de aprendizagem, para diferentes valores do número de neurônios na camada intermediária.



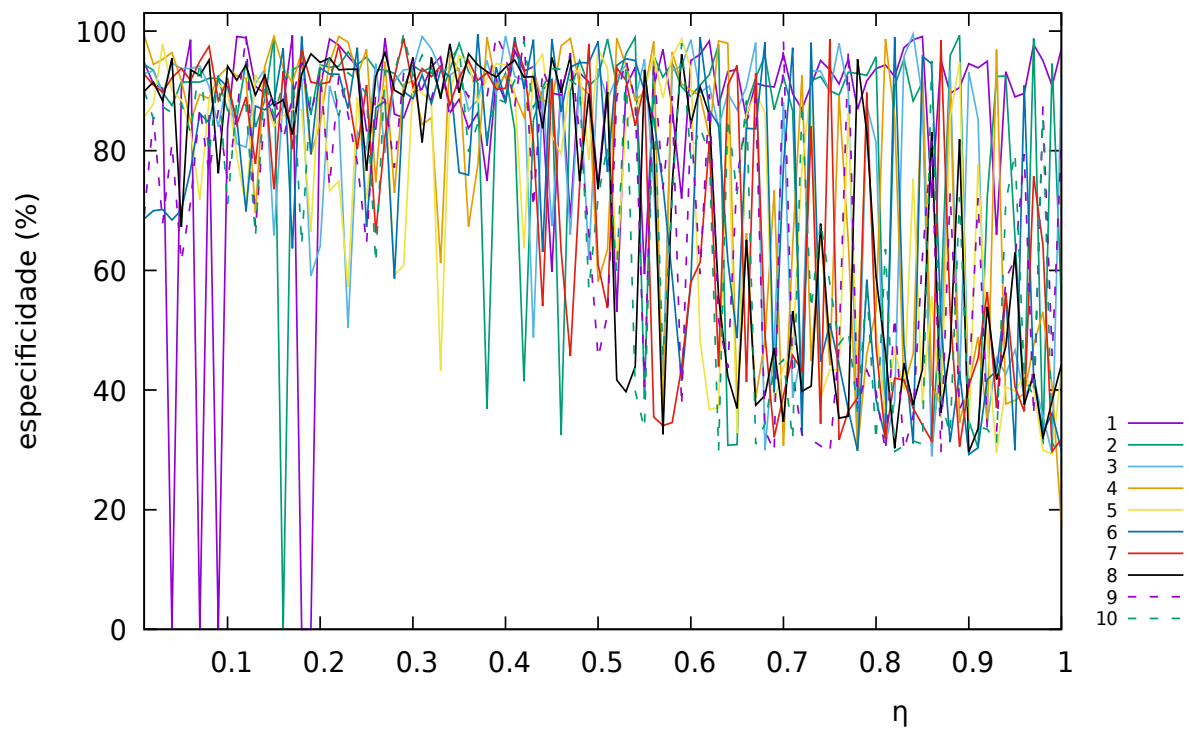
Fonte: O autor, 2019.

Figura 22 - Sensibilidade em função da taxa de aprendizagem, para diferentes valores do número de neurônios na camada intermediária.



Fonte: O autor, 2019.

Figura 23 - Especificidade em função da taxa de aprendizagem, para diferentes valores do número de neurônios na camada intermediária.



Fonte: O autor, 2019.

Nas Tabelas 4, 5 e 6, podemos observar a matriz de confusão com os três melhores resultados encontrados para a acurácia, sensibilidade e especificidade respectivamente. Os resultados correspondem à testes da capacidade de generalização, após o treinamento da rede neuronal, com uma camada intermediária.

Tabela 4 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 5, 0, 1), $\eta = 0.03$ em 100 épocas. Obtivemos uma acurácia de 99,42%, sensibilidade de 99,84% e especificidade de 97,84%.

		Valor Real	
		+1	-1
Valor Previsto	+1	3638	21
	-1	6	951

Fonte: O autor, 2019.

Tabela 5 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 3, 0, 1), $\eta = 0.15$ em 100 épocas. Obtivemos uma acurácia de 92,81%, sensibilidade de 100,00% e especificidade de 65,84%.

		Valor Real	
		+1	-1
Valor Previsto	+1	3644	332
	-1	0	640

Fonte: O autor, 2019.

Tabela 6 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 6, 0, 1), $\eta = 0.005$ em 100 épocas. Obtivemos uma acurácia de 97,92%, sensibilidade de 97,37% e especificidade de 100,00%.

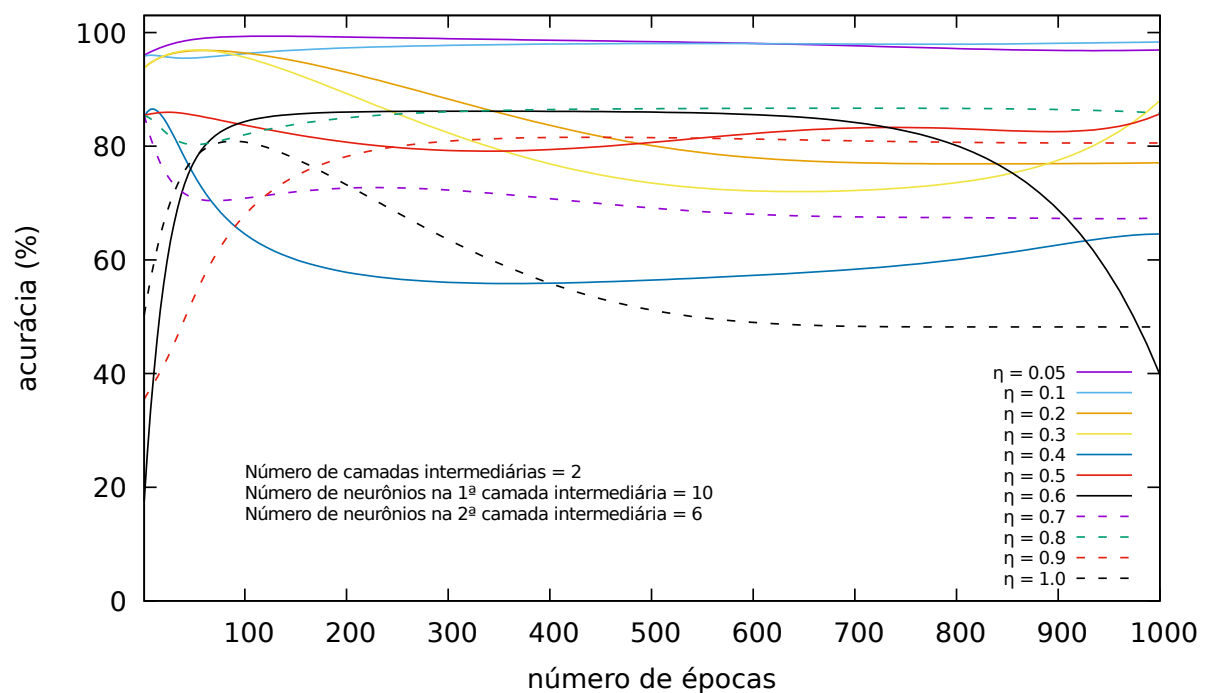
		Valor Real	
		+1	-1
Valor Previsto	+1	3548	0
	-1	96	972

Fonte: O autor, 2019.

7.1.2 Rede neuronal com 2 camadas intermediárias

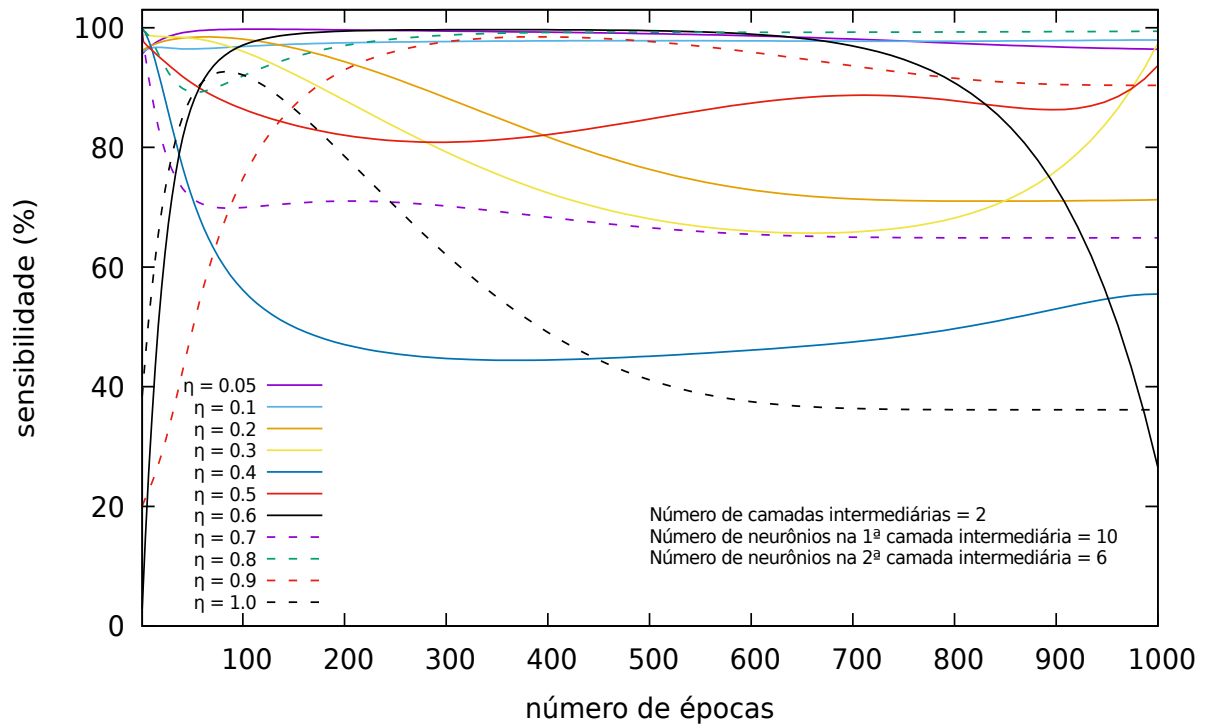
As Figuras 24, 25 e 26, são referentes à acurácia, sensibilidade e especificidade para a topologia (10, 10, 6, 1), essa topologia possui dez entradas, dez neurônios na primeira camada intermediária, seis neurônios na segunda camada intermediária e um neurônio na camada de saída, para diferentes números de épocas, e cada curva do gráfico corresponde a uma taxa de aprendizagem diferente. Na Figura 24, podemos notar que as taxas de aprendizagem próximas de zero apresentam uma melhor acurácia e as taxas de aprendizagem a partir de $\eta > 0.2$ apresentam menores valores de acurácia. A sensibilidade, como podemos ver na Figura 25, assim como a acurácia apresenta comportamento variável, dependendo bastante dos valores de eta e do número de épocas. Algumas taxas de aprendizagem como $\eta = 0.05$, $\eta = 0.1$ e $\eta = 0.8$ atingem, com poucas épocas, a estabilidade e geram bons resultados, outras taxas de aprendizagem, como $\eta = 0.3$, $\eta = 0.5$ e $\eta = 0.9$, precisam de um número maior de épocas para atingir resultados satisfatórios. Na Figura 26, vemos bons valores de especificidade, para taxas de aprendizagem baixas, como $\eta = 0.05$, $\eta = 0.1$, $\eta = 0.2$ e $\eta = 0.4$, necessitando de poucas épocas para atingir tal resultado. Outras taxas de aprendizagem como $\eta = 0.6$ e $\eta = 1.0$, necessitam de um número maior de épocas para atingir um resultado satisfatório e ainda outras taxas de aprendizagem como $\eta = 0.3$, $\eta = 0.5$, $\eta = 0.7$, $\eta = 0.8$ e $\eta = 0.9$, não atingem um resultado satisfatório.

Figura 24 - Acurácia em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui duas camadas intermediárias, com 10 neurônios na primeira camada intermediária e 6 neurônios na segunda camada intermediária.



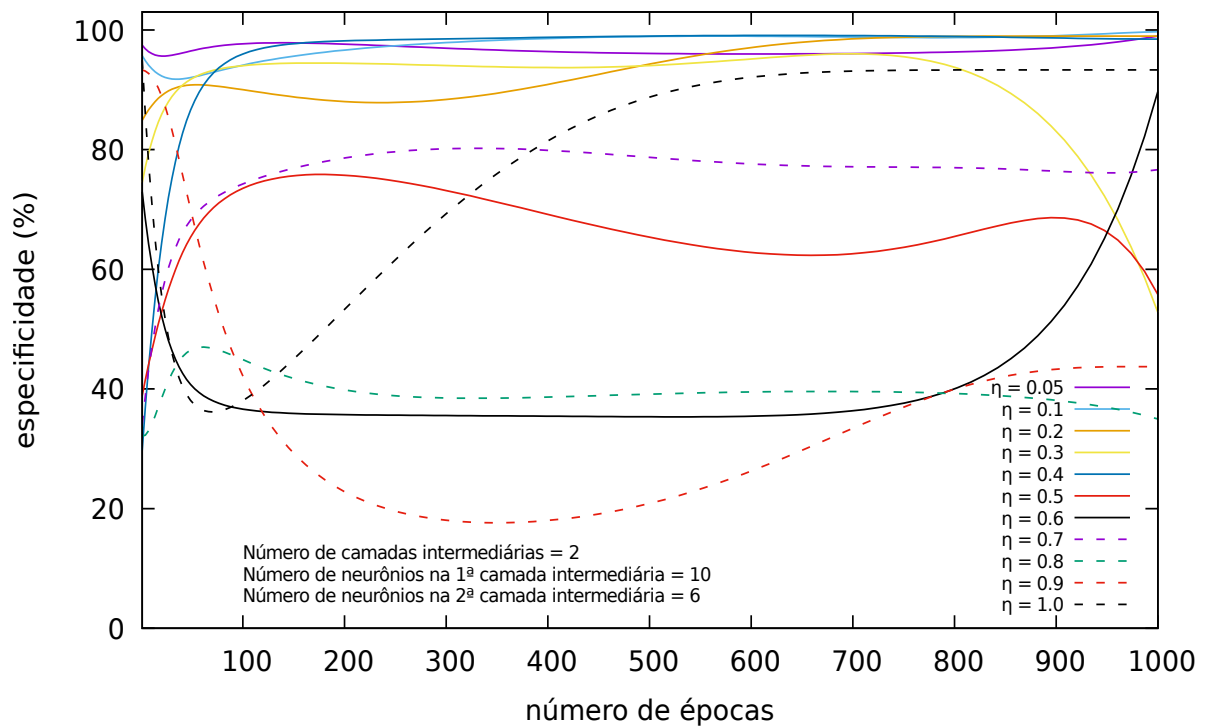
Fonte: O autor, 2019.

Figura 25 - Sensibilidade em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui duas camadas intermediárias, com 10 neurônios na primeira camada intermediária e 6 neurônios na segunda camada intermediária.



Fonte: O autor, 2019.

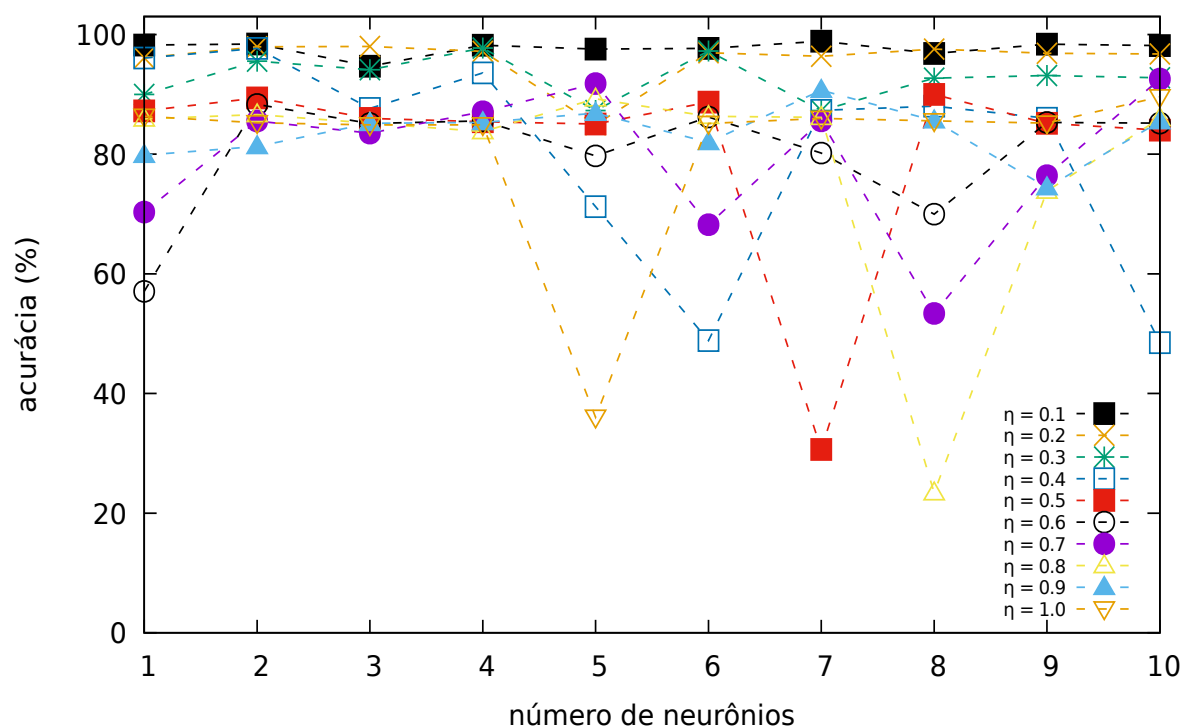
Figura 26 - Especificidade em função do número de épocas, para diferentes valores da taxa de aprendizagem. A rede possui duas camadas intermediárias, com 10 neurônios na primeira camada intermediária e 6 neurônios na segunda camada intermediária.



Fonte: O autor, 2019.

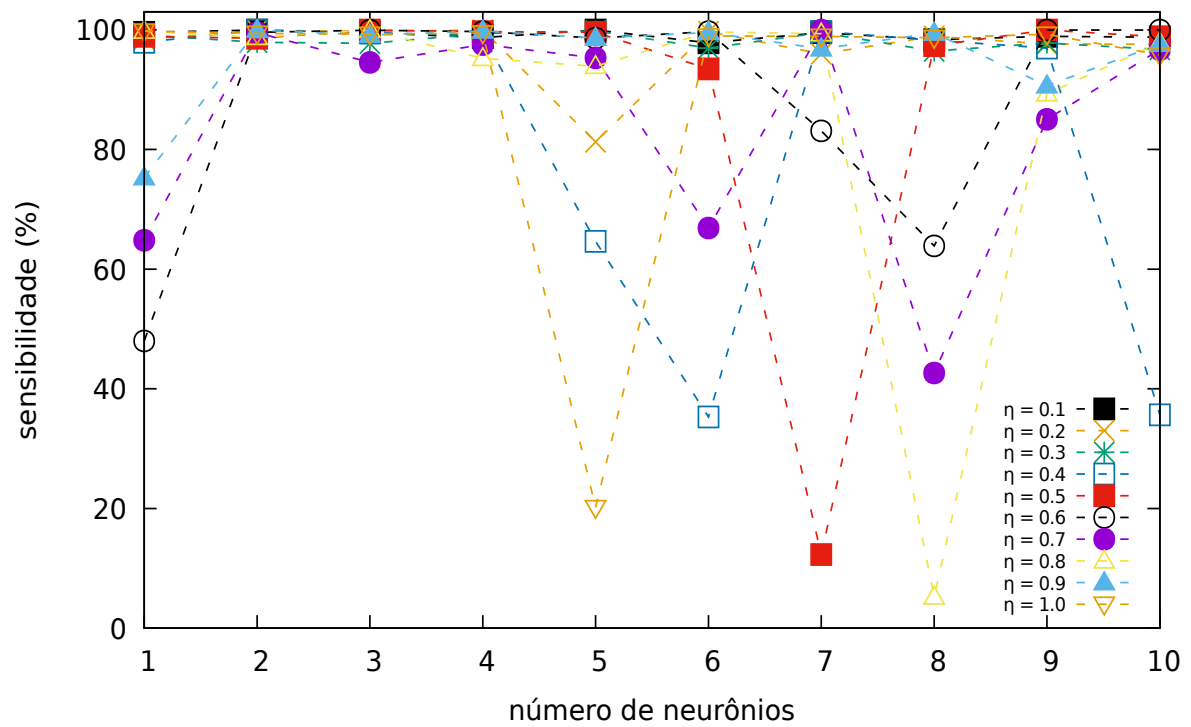
As Figuras 27, 28 e 29, são referentes à acurácia, sensibilidade e especificidade, para a topologia $(10, 10, n_2, 1)$, que possui dez entradas, dez neurônios na primeira camada intermediária, um número diferente de neurônios na segunda camada intermediária e um neurônio na camada de saída, e cada curva do gráfico corresponde a uma taxa de aprendizagem diferente. Como podemos observar na Figura 27, a acurácia se mantém alta para $\eta \leq 0.3$, e para qualquer variação de neurônios da segunda camada intermediária, já para $\eta > 0.3$, a medida que aumentamos o número de neurônios da segunda camada intermediária o modelo perde acurácia. A sensibilidade, como pode ser vista na Figura 28, é alta quando a rede possui 2, 3 e 4 neurônios na segunda camada intermediária para qualquer taxa de aprendizagem, a partir de 4 neurônios na segunda camada intermediária, a rede apresenta grande instabilidade para $\eta > 0.3$, ocasionando perda de sensibilidade. A especificidade, como podemos observar na Figura 29, possui grande instabilidade, porém para $\eta \leq 0.2$, a rede apresenta uma elevada especificidade, independente do número de neurônios da segunda camada intermediária, a partir de $\eta > 0.2$, a rede apresenta grande instabilidade e alterna entre resultados bons e ruins para a especificidade.

Figura 27 - Acurácia em função do número de neurônios da segunda camada intermediária, para diferentes valores da taxa de aprendizagem.



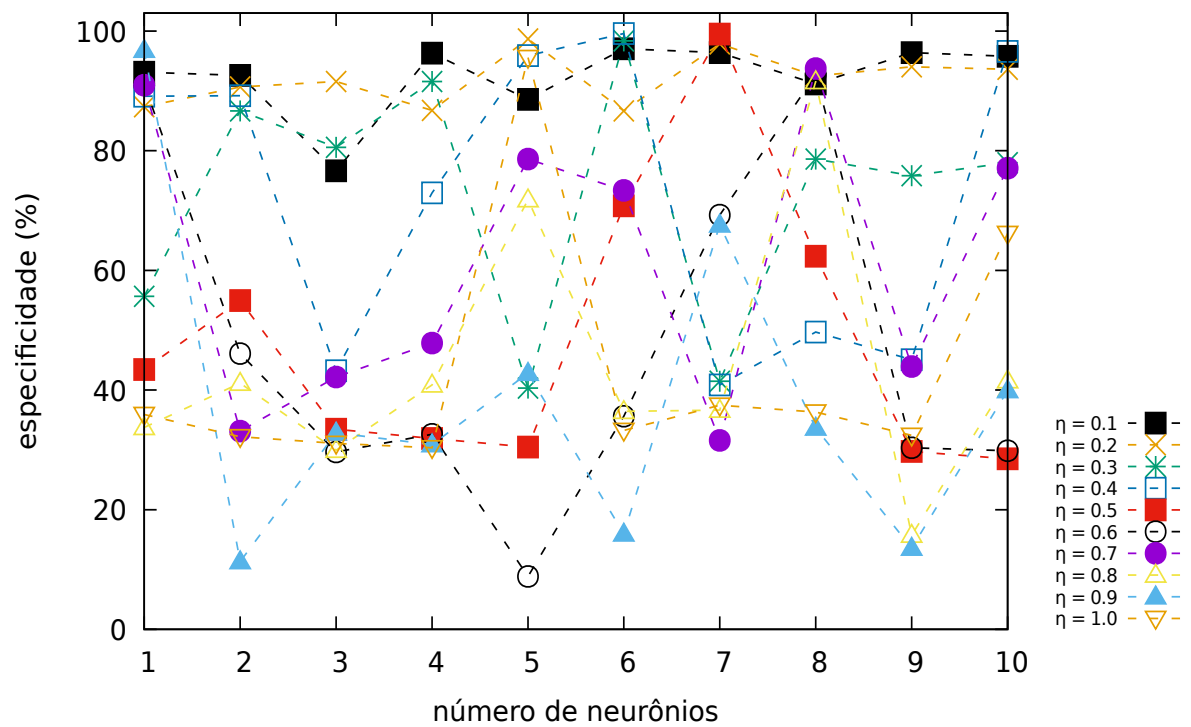
Fonte: O autor, 2019.

Figura 28 - Sensibilidade em função do número de neurônios da segunda camada intermediária, para diferentes valores da taxa de aprendizagem.



Fonte: O autor, 2019.

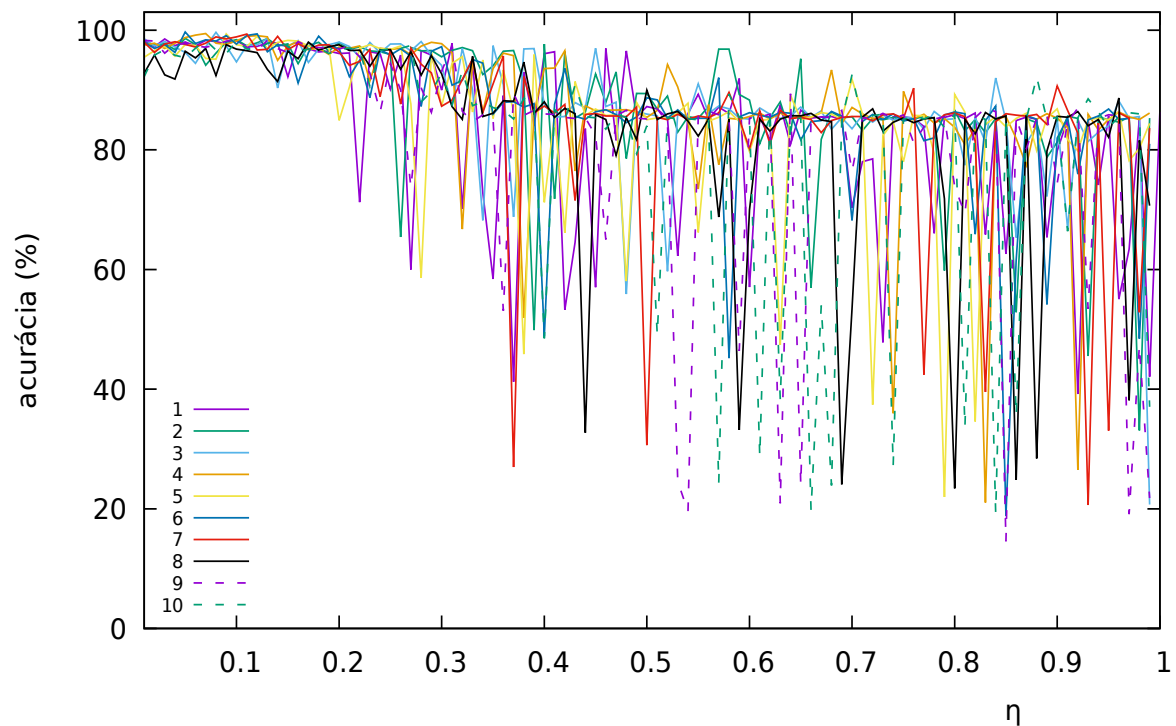
Figura 29 - Especificidade em função do número de neurônios da segunda camada intermediária, para diferentes valores da taxa de aprendizagem.



Fonte: O autor, 2019.

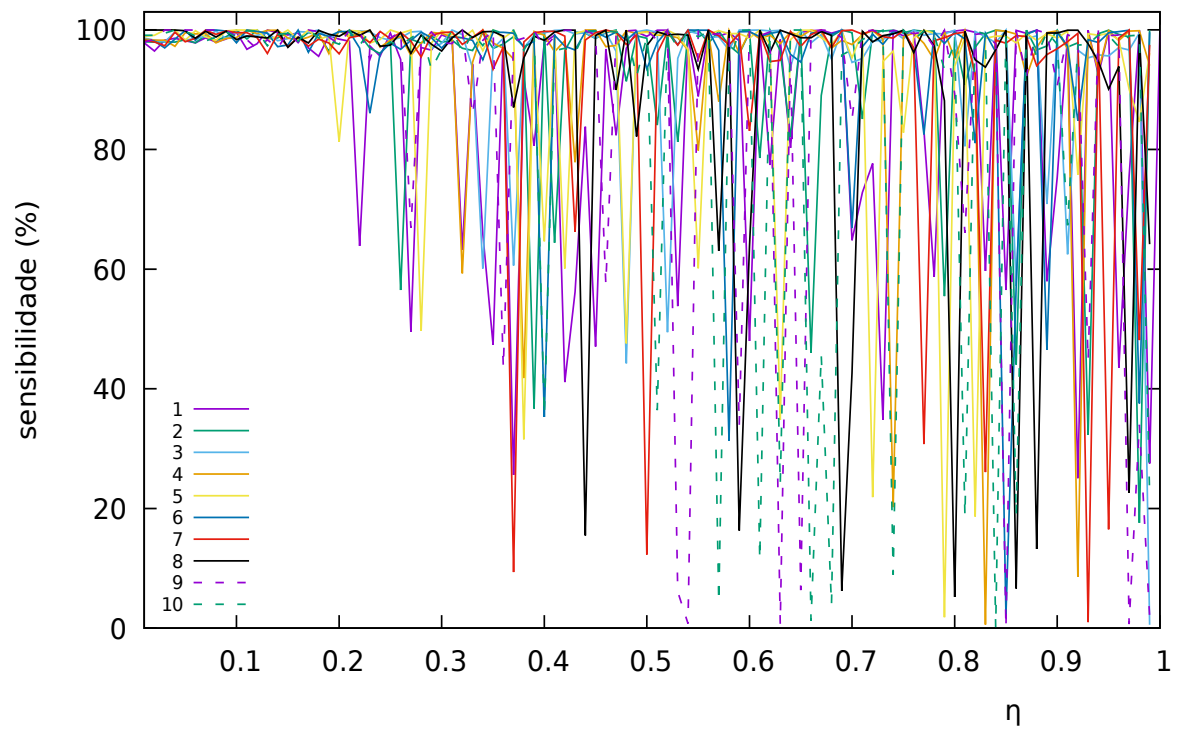
As Figuras 30, 31 e 32, são referentes à acurácia, sensibilidade e especificidade respectivamente, variando a taxa de aprendizagem, e cada curva do gráfico corresponde a uma topologia $(10, 10, n_2, 1)$, que possui dez entradas, dez neurônios na primeira camada intermediária, um número diferente de neurônios na segunda camada intermediária e um neurônio na camada de saída. Como podemos observar nas Figuras 30, 31 e 32, para $\eta \leq 0.2$, independente da topologia utilizada a acurácia, a sensibilidade e a especificidade são bastante satisfatórias, já para $\eta > 0.2$, o modelo começa a apresentar uma grande instabilidade com configurações que geram perdas grandes de acurácia, sensibilidade e especificidade.

Figura 30 - Acurácia em função da taxa de aprendizagem, para diferentes valores do número de neurônios na segunda camada intermediária.



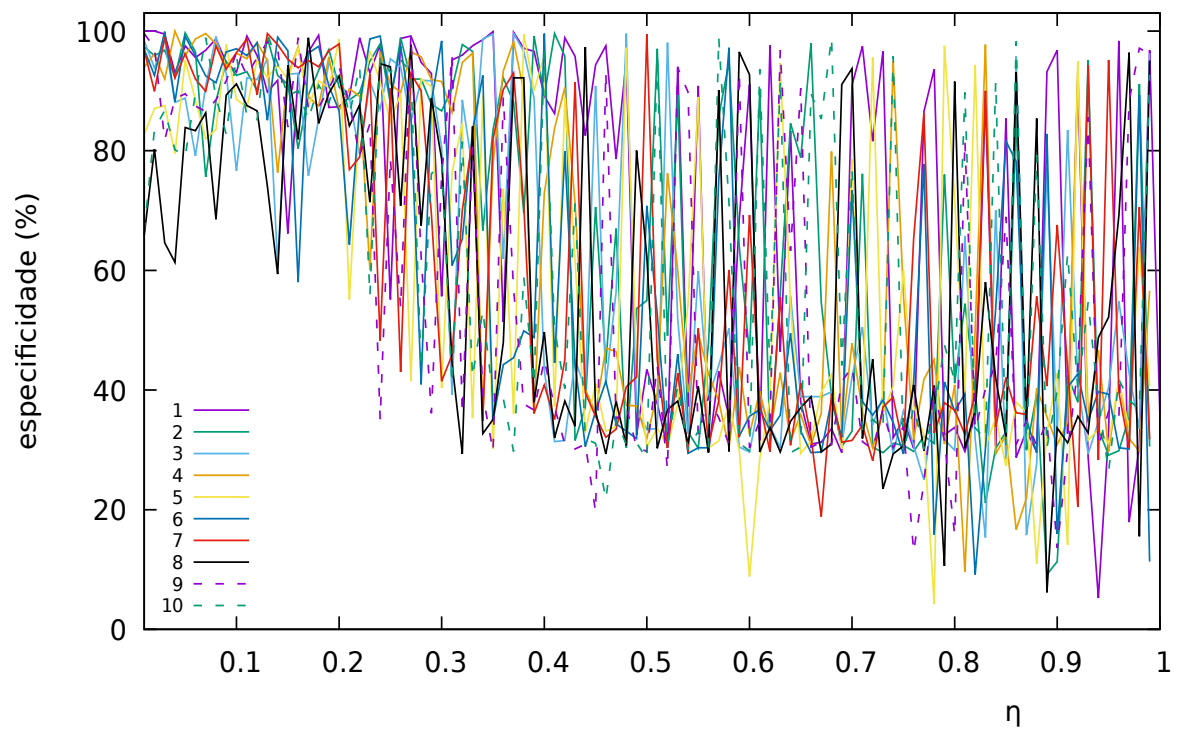
Fonte: O autor, 2019.

Figura 31 - Sensibilidade em função da taxa de aprendizagem, para diferentes valores do número de neurônios na segunda camada intermediária.



Fonte: O autor, 2019.

Figura 32 - Especificidade em função da taxa de aprendizagem, para diferentes valores do número de neurônios na segunda camada intermediária.



Fonte: O autor, 2019.

Nas Tabelas 7, 8 e 9, podemos observar a matriz de confusão com os três melhores resultados encontrados para a acurácia, sensibilidade e especificidade respectivamente. Os resultados correspondem à testes da capacidade de generalização, após o treinamento da rede neuronal, com duas camadas intermediárias.

Tabela 7 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 10, 6, 1), $\eta = 0.05$ em 100 épocas. Obtivemos uma acurácia de 99,70%, sensibilidade de 99,86% e especificidade de 99,07%.

		Valor Real	
		+1	-1
Valor Previsto	+1	3639	9
	-1	5	963

Fonte: O autor, 2019.

Tabela 8 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 10, 3, 1), $\eta = 0.65$ em 100 épocas. Obtivemos uma acurácia de 87,13%, sensibilidade de 100,00% e especificidade de 38,89%.

		Valor Real	
		+1	-1
Valor Previsto	+1	3644	594
	-1	0	378

Fonte: O autor, 2019.

Tabela 9 - Matriz de confusão resultante do teste da capacidade de generalização, com a topologia (10, 10, 1, 1), $\eta = 0.01$ em 100 épocas. Obtivemos uma acurácia de 98,29%, sensibilidade de 97,83% e especificidade de 100,00%.

		Valor Real	
		+1	-1
Valor Previsto	+1	3565	0
	-1	79	972

Fonte: O autor, 2019.

7.2 Regressão Logística

Realizamos o treinamento da regressão logística no software estatístico R, com o mesmo conjunto de treinamento utilizado na rede neuronal do tipo perceptron multicamadas. Após o treinamento da regressão logística, testamos sua capacidade de generalização, com o mesmo conjunto de teste utilizado na rede neuronal perceptron multicamadas, com a finalidade de comparar os resultados dos dois métodos de classificação. Na regressão logística como já mencionado no Capítulo 4, utilizamos o modelo Logit, e seus parâmetros são estimados por máxima verossimilhança. No software estatístico R, utilizamos a função GLM (Modelos Lineares Generalizados), para treinar a regressão logística. A Tabela 10 apresenta os coeficientes estimados para a regressão logística após o treinamento no software R.

Tabela 10 - Coeficientes estimados por máxima verossimilhança no software R, para a regressão logística .

Coeficientes	Estimativa
Intercepto	1.107e+06
β_1	-1.790e+00
β_2	9.169e+01
β_3	1.331e+01
β_4	-9.407e+00
β_5	-3.788e-01
β_6	7.144e+00
β_7	1.283e-01
β_8	4.965e+02
β_9	-1.106e+06
β_{10}	-1.109e+06

Fonte: O autor, 2019.

Na Tabela 11, podemos observar a matriz de confusão com o resultado encontrado, para o teste da capacidade de generalização, após o treinamento da regressão logística.

Tabela 11 - Matriz de confusão resultante do teste da capacidade de generalização, após o treinamento da regressão logística. Obtivemos uma acurácia de 97,94%, sensibilidade de 98,49% e especificidade de 95,88%.

		Valor Real	
		1	0
Valor Previsto	1	3589	40
	0	55	932

Fonte: O autor, 2019.

CONCLUSÃO

Iniciamos este trabalho estudando o primeiro modelo de neurônio artificial criado por McCulloch e Pitts, que serviu de base para a formulação de outros modelos de redes neurais. No Capítulo 2, vimos como o perceptron simples foi desenvolvido por Rosenblatt, e suas limitações computacionais. No Capítulo 3, mostramos que as limitações do perceptron simples não se aplicavam à rede feedforward com camadas intermediárias, o que deu um grande incentivo para novas pesquisas sobre redes neurais. Observamos que a rede neuronal do tipo perceptron multicamadas possui um grande poder computacional devido às suas camadas intermediárias. No treinamento de uma rede neuronal do tipo perceptron multicamadas, podemos destacar a grande liberdade de ajuste dos parâmetros, como a taxa de aprendizagem, a escolha da função de ativação, o intervalo dos pesos, o número de épocas, o número de camadas e o número de neurônios em cada camada. Todos esses parâmetros podem interferir positivamente ou negativamente no resultado do treinamento. Por isso, uma etapa bastante importante no treinamento da rede é a escolha dos parâmetros que calibram o modelo.

Realizamos diversos treinamentos para diferentes topologias e parâmetros da rede neuronal e comparamos os resultados obtidos pelo perceptron multicamadas com os resultados obtidos pela regressão logística. Podemos destacar que os dois modelos apresentaram resultados bastante satisfatórios em relação à acurácia, a sensibilidade e a especificidade. O objetivo do trabalho em questão é reduzir a inadimplência ocorrida no financiamento para investimentos do BNDES. Como a base do BNDES possui aproximadamente 80% de clientes adimplentes e 20% de clientes inadimplentes, constatamos que a acurácia pode nos levar a decisões errôneas, pois mesmo com uma acurácia elevada podemos ter uma situação com bastante inadimplência. O teste diagnóstico que nos revela a capacidade de classificar corretamente os clientes inadimplentes é a especificidade. Com isso, o teste que possui a maior especificidade foi escolhido como o melhor resultado desse experimento. O treinamento do perceptron multicamadas com a topologia (10, 10, 1, 1), com $\eta = 0.01$ em 100 épocas, representado na Tabela 9, obteve 100,00% de especificidade, 98,29% de acurácia e 97,83% de sensibilidade. O modelo de regressão logística obteve 95,88% de especificidade, 97,94% de acurácia e 98,49% de sensibilidade. Com isso, concluímos que um bom modelo de classificação para reduzir a inadimplência de financiamentos realizados pelo BNDES é o modelo da rede neuronal do tipo perceptron multicamadas.

Como foi abordado nesta dissertação, realizamos o projeto em linguagem C, porém algumas sugestões podem ajudar outros pesquisadores em projetos futuros a realizar um estudo com a rede neuronal do tipo perceptron multicamadas, tais como: Realizar o modelo da rede neuronal perceptron multicamadas em linguagem Python; Utilizar outras

funções de ativação; Automatizar a escolha dos parâmetros e da topologia da rede neuronal, para que a rede responda com o melhor resultado encontrado dentre um intervalo predefinido de parâmetros e topologias.

REFERÊNCIAS

- AKOBENG, Anthony K. Understanding diagnostic tests 1: sensitivity, specificity and predictive values. *Acta paediatrica*, v. 96 3, p. 338–41, 2007.
- CHENG, Bing; TITTERINGTON, D. M. [neural networks: A review from statistical perspective]: Rejoinder. *Statist. Sci.*, The Institute of Mathematical Statistics, v. 9, n. 1, p. 49–54, 02 1994.
- COX, D.R.; SNELL, E.J. *Analysis of Binary Data, Second Edition*. London, United Kingdom: Chapman & Hall, 1989.
- CRAMER, J.S. *The Origins of Logistic Regression*. Amsterdam, Netherlands: Tinbergen Institute, 2002.
- GLOROT, Xavier; BENGIO, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*. Québec, Canada: Université de Montréal, 2010. v. 9, p. 249–256.
- GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. Massachusetts, EUA: The MIT Press, 2016.
- HAYKIN, S.S. *Neural Networks: A Comprehensive Foundation*. New Jersey, USA: Prentice Hall, 1999. (International edition).
- HAYKIN, S. *Redes Neurais: Princípios e Prática*. Rio de Janeiro, Brasil: Artmed, 2007.
- HERTZ, John; KROGH, Anders; PALMER, Richard G. *Introduction to the Theory of Neural Computation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1991.
- HOSMER, David W.; LEMESHOW, Stanley. *Applied logistic regression*. New Jersey, EUA: John Wiley and Sons, 2000.
- LECUN, Yann et al. Efficient backprop. In: *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK: Springer-Verlag, 1998. p. 9–50.
- MCCULLOCH, Warren; PITTS, Walter. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, v. 5, p. 127–147, 1943.
- MINSKY, Marvin.; PAPERT, Seymour. Perceptrons : An introduction to computational geometry. 1969.
- MINSKY, Marvin.; SELFRIDGE, Oliver. Learning in Random Nets. Butterworths, 1961.
- MOOD, A.M.F.; GRAYBILL, F.A.; BOES, D.C. *Introduction to the Theory of Statistics*. [S.l.]: McGraw-Hill, 1974.
- ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, p. 65–386, 1958.

_____. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Washington, D.C: Spartan Books, 1962. (Report (Cornell Aeronautical Laboratory)).

RUMELHART, David E.; HINTON, Geoffrey E.; WILLIAMS, Ronald J. Learning representations by back-propagating errors. *Nature*, Nature Publishing Group, v. 323, p. 533–, out. 1986.

RUMELHART, David E.; MCCLELLAND, James L. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986.

APÊNDICE A – Programas Numéricos

Segue abaixo o procedimento *Backstruct* escrito em linguagem C, para estruturar a *Rede Neuronal Perceptron Multicamadas*.

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <math.h>
4  #include <string.h>
5  #include <malloc.h>
6
7  typedef struct NEURONIO_CONTENT{
8
9      double V;
10     double V2;
11     double *peso;
12     double delta;
13     double h_in;
14
15 } neuronio_content;
16
17 typedef struct CAMADA_CONTENT{
18
19     int num_neuronios;
20     struct NEURONIO_CONTENT *neuronio;
21
22 } camada_content;

```

Segue abaixo o procedimento *MLP* escrito em linguagem C, retirado da referência (HERTZ; KROGH; PALMER, 1991), que executa a *Rede Neuronal Perceptron Multicamadas*.

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <math.h>
5  #include <time.h>
6  #include <string.h>
7  #include <malloc.h>
8  #define MAX 10
9  #include "back_struct.h"
10
11
12 double eta, momentum, beta, error, threshold, ran3(), **ksi, **ksi_teste, **desired_out,
13 **desired_teste, delta_w, erro, g_ativ, dg_ativ, sum;
14 int i, j, k, p, e, t, num_padroes, num_camadas, epocas, cam[MAX], epoca, num_teste,
15 iseed = -17841;
16 float acerto, sensibilidade, especificidade, vpp, vpn, contador_vp = 0.0,
17 contador_fp = 0.0, contador_fn = 0.0, contador_vn = 0.0;
18
19 FILE *fp, *dados_entrada, *dados_saida, *dados_teste, *dados_teste_saida, *impressao_leitura,
20 *impressao_parametros, *impressao_leitura_teste, *impressao_erro, *impressao_output,
21 *impressao_pesos, *impressao_teste, *confusao, *resultados_testes, *errorf;
22
23 camada_content *camada;

```

```

24
25 /*
26 ****
27     Global Functions
28 ****
29 */
30
31 void cabecalho(), parametros_iniciais_printar(), impressao_dos_parametros_arquivo(),
32 leitura_dos_parametros(), check_allocation(), arquivos_leitura_escrita(),
33 alocao_e_leitura_conj_train_test(), treinamento_do_backpropagation(),
34 impressao_dos_pesos_finais(), teste_da_rede(), respostas_da_rede_backpropagation(),
35 fechar_programa();
36
37 /*
38 ****
39     Main Multilayer Perceptron Simulator Program
40 ****
41 */
42
43 void main()
44 {
45     /*   Leitura dos parametros da rede */
46     leitura_dos_parametros ();
47     /*   Alocar a rede e ler os conjuntos de train e test */
48     alocao_e_leitura_conj_train_test();
49     /*   Treinamento da Rede Backpropagation */
50     treinamento_do_backpropagation();
51     /*   Pesos finais */
52     impressao_dos_pesos_finais();
53     /*   Teste da Rede Backpropagation */
54     teste_da_rede();
55     /*   Respostas da Rede */
56     respostas_da_rede_backpropagation()
57     fechar_programa();
58 } /* end main() */
59
60 /*
61 ****
62     Functions
63 ****
64 */
65
66 void leitura_dos_parametros()
67 {
68     int i;
69     arquivos_leitura_escrita();
70     cabecalho();
71     fscanf(fp,
72 "%s%d%s_%s%d%s_%s%lf%s_%s%lf%s_%s%lf%s_%s%lf%s_%s%d%s_%s%lf%s",
73 &num_padroes, &num_camadas, &epocas, &eta, &momentum, &beta, &threshold, &num_teste, &error);
74
75     for (i = 0; i < num_camadas; i++) fscanf(fp, "%s%d%s", &(cam[i]));
76     fclose(fp);
77     parametros_iniciais_printar();
78     impressao_dos_parametros_arquivo();
79 }
80
81 void arquivos_leitura_escrita()
82 {

```

```

83     fp = fopen("in_parametros.dat", "r");
84     impressao_parametros = fopen("./Respostas_da_rede/parametros_entrada.txt", "w");
85     resultados_testes = fopen("./Respostas_da_rede/respostas_testes.txt", "a");
86     impressao_leitura = fopen("./Respostas_da_rede/leitura_conj_treinamento.txt", "w");
87     impressao_leitura_teste = fopen("./Respostas_da_rede/leitura_conj_teste.txt", "w");
88     confusao = fopen("./Respostas_da_rede/respostas_da_rede_backpropagation.txt", "w");
89     errorf = fopen("./Respostas_da_rede/erro_alocacao.txt", "w");
90     impressao_erro = fopen("./Respostas_da_rede/erro_da_rede.txt", "w");
91     impressao_output = fopen("./Respostas_da_rede/resultado_treinamento.txt", "w");
92     impressao_pesos = fopen("./Respostas_da_rede/pesos_iniciais_finais.txt", "w");
93     impressao_teste = fopen("./Respostas_da_rede/resultado_teste.txt", "w");
94     dados_entrada = fopen("./conjunto_de_treinamento/BNDES_30percent_var9_var10_entrada.csv", "r");
95     dados_saida = fopen("./conjunto_de_treinamento/BNDES_30percent_var9_var10_saida.csv", "r");
96     dados_teste = fopen("./conjunto_de_teste/BNDES_70percent_var9_var10_entrada.csv", "r");
97     dados_teste_saida = fopen("./conjunto_de_teste/BNDES_70percent_var9_var10_saida.csv", "r");
98 }
99
100 void cabecalho()
101 {
102     printf("\n\t_UERJ");
103     printf("\n\t_Mestrado_de_Ciencias_Computacionais");
104     printf("\n\t_Aluno:_Raphael_Silva_de_Figueiredo");
105     printf("\n\t_Perceptron_Multicamadas_-_Backpropagation\n");
106 }
107
108 void parametros_iniciais_printar()
109 {
110     int i;
111     printf("\n\t_numero_de_padroes_do_conj_treinamento: %d", num_padroes);
112     printf("\n\t_numero_de_epocas: %d", epocas);
113     printf("\n\t_taxa_de_aprendizagem: %0.3lf", eta);
114     printf("\n\t_termo_de_momento: %0.1lf", momentum);
115     printf("\n\t_valor_de_beta: %0.1lf", beta);
116     printf("\n\t_valor_do_threshold: %0.1lf", threshold);
117     printf("\n\t_numero_de_padroes_do_conj_teste: %d", num_teste);
118     printf("\n\t_valor_de_erro_maximo: %lf\n", error);
119     printf("\n\t_numero_de_camadas: %d\n\n", num_camadas);
120
121     for (i = 0; i < num_camadas; i++)
122     {
123         printf("\t_neuronios_da_camada %d: %d\n", i, cam[i]);
124     }
125 }
126
127 void impressao_dos_parametros_arquivo()
128 {
129     int i;
130     fprintf(impressao_parametros, "\n\t_UERJ");
131     fprintf(impressao_parametros, "\n\t_Mestrado_de_Ciencias_Computacionais");
132     fprintf(impressao_parametros, "\n\t_Aluno:_Raphael_Silva_de_Figueiredo");
133     fprintf(impressao_parametros, "\n\t_Perceptron_Multicamadas_-_Backpropagation\n");
134     fprintf(impressao_parametros, "\n\t_numero_de_padroes_do_conj_treinamento: %d", num_padroes);
135     fprintf(impressao_parametros, "\n\t_numero_de_epocas: %d", epocas);
136     fprintf(impressao_parametros, "\n\t_taxa_de_aprendizagem: %0.3lf", eta);
137     fprintf(impressao_parametros, "\n\t_termo_de_momento: %0.1lf", momentum);
138     fprintf(impressao_parametros, "\n\t_valor_de_beta: %0.1lf", beta);
139     fprintf(impressao_parametros, "\n\t_valor_do_threshold: %0.1lf", threshold);
140     fprintf(impressao_parametros, "\n\t_numero_de_padroes_do_conj_teste: %d", num_teste);
141     fprintf(impressao_parametros, "\n\t_valor_de_erro_maximo: %lf\n", error);

```

```

142 fprintf(impressao_parametros, "\n_numero_de_camadas: %d\n\n", num_camadas);
143
144 for (i = 0; i < num_camadas; i++)
145     fprintf(impressao_parametros, "\n_neuronios_da_camada %d: %d\n", i, cam[i]);
146     fclose(impressao_parametros);
147 }
148
149
150 /* Gerador de numeros aleatorios */
151 #define MBIG 1000000000
152 #define MSEED 161803398
153 #define MZ 0
154 #define FAC (1.0/MBIG)
155
156 int inext, inextp;
157 long ma[56];
158 int iff = 0;
159
160 double
161 ran3(idum)
162 int *idum;
163 {
164     /* static int inext, inextp;
165     static long ma[56];
166     static int iff = 0; */
167     long mj, mk;
168     int i, ii, k;
169
170     if (*idum < 0 || iff == 0)
171     {
172         iff = 1;
173         mj = MSEED - (*idum < 0 ? -*idum : *idum);
174         mj %= MBIG;
175         ma[55] = mj;
176         mk = 1;
177         for (i = 1; i <= 54; i++)
178         {
179             ii = (21 * i) % 55;
180             ma[ii] = mk;
181             mk = mj - mk;
182             if (mk < MZ) mk += MBIG;
183             mj = ma[ii];
184         } /* end for */
185
186         for (k = 1; k <= 4; k++)
187             for (i = 1; i <= 55; i++)
188             {
189                 ma[i] -= ma[1+(i + 30) % 55];
190                 if (ma[i] < MZ) ma[i] += MBIG;
191             } /* end for */
192         inext = 0;
193         inextp = 31;
194         *idum = 1;
195     } /* end if */
196
197     if (++inext == 56) inext = 1;
198     if (++inextp == 56) inextp = 1;
199     mj = ma[inext] - ma[inextp];
200     if (mj < MZ) mj += MBIG;

```



```

201     ma[inext] = mj;
202     if (mj * FAC == 1.0) return (0.0);
203     else return (mj * FAC);
204 } /* end ran3 */
205
206
207 /* Check allocation */
208 void check_allocation (pt, in_funct, in_var)
209 void *pt;
210 char in_funct[];
211 char in_var[];
212 {
213     if (pt == NULL)
214     {
215         fprintf(errorf, "Run-time_memory_allocation_error...\n");
216         fprintf(errorf, "In_function: %s, variable: %s\n", in_funct, in_var);
217         fprintf(errorf, "...now_exiting_to_system...\n");
218         exit(0);
219         fclose(errorf);
220     }
221 } /* check_allocation () */
222
223 void alocação_e_leitura_conj_train_test()
224 {
225     /* Alocar a Rede Neuronal */
226     camada = (camada_content *) malloc (num_camadas * sizeof(camada_content));
227     check_allocation( (void *)camada, "main()", "camada");
228
229     for ( i = 0 ; i < num_camadas ; i++ )
230         camada[i].num_neuronios = cam[i];
231
232     for ( i = 0 ; i < num_camadas ; i++ )
233     {
234         camada[i].neuronio = (neuronio_content *) malloc ((camada[i].num_neuronios + 1)
235             * sizeof(neuronio_content));
236         check_allocation( (void *)camada[i].neuronio, "main()", "camada[i].neuronio");
237         for (j = 0 ; j <= camada[i].num_neuronios ; j++ )
238         {
239             camada[i].neuronio[j].peso = (double *) malloc (camada[i+1].num_neuronios
240                 * sizeof(double));
241             check_allocation( (void *)camada[i].neuronio[j].peso, "main()",
242                 "camada[i].neuronio[j].peso");
243             for (k = 0; k < camada[i+1].num_neuronios; k++)
244                 camada[i].neuronio[j].peso[k] = ran3(&iseed);
245         }
246     } /* end for num_camadas */
247
248     /* Alocar conjunto treinamento */
249     ksi = (double **) malloc ( num_padroes * sizeof(double));
250     check_allocation( (void *)ksi, "main()", "ksi");
251     for (i = 0; i < num_padroes; i++)
252         ksi[i] = (double *) malloc ((camada[0].num_neuronios + 1) * sizeof(double));
253
254     desired_out = (double **) malloc ( num_padroes * sizeof(double));
255     check_allocation( (void *)desired_out, "main()", "desired_out");
256     for (i = 0; i < num_padroes; i++)
257         desired_out[i] = (double *) malloc (camada[num_camadas - 1].num_neuronios
258             * sizeof(double));
259

```

```

260  /* Leitura do conjunto de treinamento */
261  fprintf(impressao_leitura, "\n");
262  for (p = 0 ; p < num_padroes ; p++)
263  {
264      fprintf(impressao_leitura, "%-5d", p);
265      for (j = 0; j < camada[0].num_neuronios ; j++)
266      {
267          fscanf(dados_entrada, "%lf", &ksi[p][j]);
268          fprintf(impressao_leitura, "%+.3lf", ksi[p][j]);
269      }
270      for (k = 0; k < camada[num_camadas - 1].num_neuronios ; k++)
271      {
272          fscanf(dados_saida, "%lf", &desired_out[p][k]);
273          fprintf(impressao_leitura, "%+lf", desired_out[p][k]);
274      }
275      fprintf(impressao_leitura, "\n");
276  }
277
278  /* Alocar conjunto teste */
279  ksi_teste = (double **) malloc ( num_teste * sizeof(double));
280  check_allocation( (void *)ksi_teste, "main()", "ksi_teste");
281  for (i = 0; i < num_teste; i++)
282      ksi_teste[i] = (double *) malloc ((camada[0].num_neuronios + 1) * sizeof(double));
283
284  desired_teste = (double **) malloc ( num_teste * sizeof(double));
285  check_allocation( (void *)desired_teste, "main()", "desired_teste");
286  for (i = 0; i < num_teste; i++)
287      desired_teste[i] = (double *) malloc (camada[num_camadas - 1].num_neuronios
288          * sizeof(double));
289
290  /* Leitura do conjunto de teste */
291  fprintf(impressao_leitura_teste, "\n");
292  for (t = 0 ; t < num_teste ; t++)
293  {
294      fprintf(impressao_leitura_teste, "%-5d", t);
295      for (j = 0; j < camada[0].num_neuronios ; j++)
296      {
297          fscanf(dados_teste, "%lf", &ksi_teste[t][j]);
298          fprintf(impressao_leitura_teste, "%+.3lf", ksi_teste[t][j]);
299      }
300      for (k = 0; k < camada[num_camadas - 1].num_neuronios ; k++)
301      {
302          fscanf(dados_teste_saida, "%lf", &desired_teste[t][k]);
303          fprintf(impressao_leitura_teste, "%+lf", desired_teste[t][k]);
304      }
305      fprintf(impressao_leitura_teste, "\n");
306  }
307 }
308
309 void treinamento_do_backpropagation()
310 {
311     /* Impressao dos pesos iniciais */
312     fprintf(impressao_pesos, "\nPesos Iniciais\n\n");
313     for ( i = num_camadas - 1; i > 0; i-- )
314     {
315         for (j = 0 ; j < camada[i].num_neuronios ; j++ )
316         {
317             for (k = 0 ; k < camada[i-1].num_neuronios +1 ; k++ )
318             {

```



```

378         dg_ativ = beta * (1 - (g_ativ * g_ativ));
379         camada[i].neuronio[j].delta = ( dg_ativ * sum );
380     } /* end else */
381 } /* end for num_neuronios camada i */
382 } /* end for num_camadas */
383
384 /* Atualizacao dos pesos apos o calculo dos deltas */
385 for ( i = num_camadas - 1; i > 0; i-- )
386 {
387     for ( j = 0 ; j < camada[i].num_neuronios ; j++ )
388     {
389         for ( k = 0 ; k < camada[i-1].num_neuronios + 1 ; k++ )
390         {
391             delta_w = eta * (camada[i].neuronio[j].delta * camada[i-1].neuronio[k].V);
392             camada[i-1].neuronio[k].peso[j] = camada[i-1].neuronio[k].peso[j]
393                 + momentum * delta_w;
394         }
395     }
396 } /* end for num_camadas */
397
398 /* Impressao do output */
399
400 for ( j = 0 ; j < camada[num_camadas-1].num_neuronios ; j++ )
401     fprintf(impressao_output, "\n_padrao_%d-__desired_%+12lf:___output_=%+1f",
402         p+1, desired_out[p][j],
403         camada[num_camadas-1].neuronio[j].V);
404
405 /* Impressao dos erros */
406
407 for ( j = 0 ; j < camada[num_camadas-1].num_neuronios ; j++ )
408     erro += 0.5 * ( (desired_out[p][j] - camada[num_camadas-1].neuronio[j].V)
409         * (desired_out[p][j] -
410             camada[num_camadas-1].neuronio[j].V) );
411
412 }/* end for (p) num_padroes */
413
414 fprintf(impressao_erro, "\nepoca_=%d:___erro_=%lf", e+1, erro);
415
416 if ( erro < error ) break;
417
418 /* Impressao do progresso de epocas */
419 printf("\nb\b\b\b\b\b\b_d", e+1); fflush(stdout);
420
421 }/* end for (e) */
422     printf("\n");
423 }
424
425 void impressao_pesos_finais()
426 {
427     /* Impressao dos pesos finais */
428     fprintf(impressao_pesos, "\nPesos_Finais\n\n");
429     for ( i = num_camadas - 1; i > 0; i-- )
430     {
431         for ( j = 0 ; j < camada[i].num_neuronios ; j++ )
432         {
433             for ( k = 0 ; k < camada[i-1].num_neuronios +1 ; k++ )
434             {
435                 fprintf(impressao_pesos, "%+1f\t", camada[i-1].neuronio[k].peso[j]);
436             }

```

```

437         fprintf(impressao_pesos, "\n");
438     }
439 }
440 }
441
442 void teste_da_rede()
443 {
444     for (t = 0 ; t < num_teste ; t++ )
445     {
446         for ( i = 0 ; i < num_camadas ; i++ )
447         {
448             for ( j = 0 ; j < camada[i].num_neuronios ; j++ )
449             {
450                 camada[i].neuronio[j].h_in = 0.0;
451                 if (i == 0)
452                 {
453                     camada[i].neuronio[j].V = ksi_teste[t][j];
454                 }
455                 else
456                 {
457                     for (k = 0 ; k < camada[i-1].num_neuronios ; k++ )
458                         camada[i].neuronio[j].h_in += (camada[i-1].neuronio[k].peso[j]
459                             * camada[i-1].neuronio[k].V);
460                     camada[i].neuronio[j].V = tanh(beta * camada[i].neuronio[j].h_in);
461                 }
462             }
463         }
464         for ( j = 0 ; j < camada[num_camadas-1].num_neuronios ; j++ )
465         {
466             if (camada[num_camadas-1].neuronio[j].V >= 0.0)
467             {
468                 camada[num_camadas-1].neuronio[j].V2 = 1.0;
469             }
470             else
471             {
472                 camada[num_camadas-1].neuronio[j].V2 = -1.0;
473             }
474
475             if (desired_teste[t][j] == 1.0 && camada[num_camadas-1].neuronio[j].V2 == 1.0)
476             {
477                 contador_vp = contador_vp + 1;
478             }
479             else if (desired_teste[t][j] == 1.0 && camada[num_camadas-1].neuronio[j].V2 == -1.0)
480             {
481                 contador_fn = contador_fn + 1;
482             }
483             else if (desired_teste[t][j] == -1.0 && camada[num_camadas-1].neuronio[j].V2 == 1.0)
484             {
485                 contador_fp = contador_fp + 1;
486             }
487             else if (desired_teste[t][j] == -1.0 && camada[num_camadas-1].neuronio[j].V2 == -1.0)
488             {
489                 contador_vn = contador_vn + 1;
490             }
491
492             fprintf(impressao_teste, "\npadrao_teste_%-7d_-desired_%+12lf:output=_%+lf", t+1,
493                 desired_teste[t][j], camada[num_camadas-1].neuronio[j].V);
494         }
495     }

```

```

496 }
497
498 void respostas_da_rede_backpropagation()
499 {
500     acerto = ((contador_vp + contador_vn)/
501 (contador_vp + contador_vn + contador_fn + contador_fp))*100;
502     sensibilidade = (contador_vp/(contador_vp + contador_fn))*100;
503     especificidade = (contador_vn/(contador_vn + contador_fp))*100;
504     vpp = (contador_vp/(contador_vp + contador_fp))*100;
505     vpn = (contador_vn/(contador_vn + contador_fn))*100;
506
507     fprintf(confusao, "\n_VP_%-5.0f_FP_%.0f" , contador_vp, contador_fp);
508     printf("\n_VP_%-5.0f_FP_%.0f" , contador_vp, contador_fp);
509     fprintf(confusao, "\n_FN_%-5.0f_VN_%.0f" , contador_fn, contador_vn);
510     printf("\n_FN_%-5.0f_VN_%.0f" , contador_fn, contador_vn);
511
512     fprintf(confusao, "\n\n_acuracia: _%.2f_%%" , acerto);
513     printf("\n\n_acuracia: _%.2f_%%" , acerto);
514     fprintf(confusao, "\n_erro: _%.2f_%%" , 100 - acerto);
515     printf("\n_erro: _%.2f_%%" , 100 - acerto);
516
517     fprintf(confusao, "\n\n_sensibilidade: _%.2f_%%" , sensibilidade);
518     printf("\n\n_sensibilidade: _%.2f_%%" , sensibilidade);
519     fprintf(confusao, "\n_especificidade: _%.2f_%%" , especificidade);
520     printf("\n_especificidade: _%.2f_%%" , especificidade);
521
522     fprintf(confusao, "\n\n_vpp: _%.2f_%%" , vpp);
523     printf("\n\n_vpp: _%.2f_%%" , vpp);
524     fprintf(confusao, "\n_vpn: _%.2f_%%" , vpn);
525     printf("\n_vpn: _%.2f_%%\n\n" , vpn);
526
527     fprintf(resultados_testes, "\n%d_%.2f_%.2f_%.2f_%.2f_%.2f_%.3f_ _d", cam[1], acerto,
528 sensibilidade, especificidade, vpp, vpn, eta, epocas);
529 }
530
531 void fechar_programa()
532 {
533     fclose(resultados_testes);
534     fclose(impressao_leitura);
535     fclose(impressao_leitura_teste);
536     fclose(confusao);
537     fclose(impressao_erro);
538     fclose(impressao_output);
539     fclose(impressao_pesos);
540     fclose(impressao_teste);
541     fclose(dados_entrada);
542     fclose(dados_saida);
543 }

```