UNIVERSIDADE FEDERAL DO PAMPA

FLÁVIA MORALES ANTUNES

ABORDAGEM NEURAL PARA A DETERMINAÇÃO DE DEPENDÊNCIAS ENTRE VARIÁVEIS DO PROCESSO DE TRANSPORTE E ÍNDICE DE CONTUSÕES EM BOVINOS

Bagé
2019

FLÁVIA MORALES ANTUNES

ABORDAGEM NEURAL PARA A DETERMINAÇÃO DE DEPENDÊNCIAS ENTRE VARIÁVEIS DO PROCESSO DE TRANSPORTE E ÍNDICE DE CONTUSÕES EM BOVINOS

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Engenharia de Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientadora: Ana Paula Lüdtke Ferreira

Bagé
2019

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

A636a Antunes, Flávia Morales
ABORDAGEM NEURAL PARA A DETERMINAÇÃO DE DEPENDÊNCIAS ENTRE VARIÁVEIS DO PROCESSO DE TRANSPORTE E ÍNDICE DE CONTUSÕES EM BOVINOS / Flávia Morales Antunes.
70 p.

Trabalho de Conclusão de Curso(Graduação)-- Universidade Federal do Pampa, ENGENHARIA DE COMPUTAÇÃO, 2019.
"Orientação: Ana Paula Lüdtke Ferreira".

1. Redes neurais. 2. Pecuária de precisão. 3. Bem-estar animal. 4. Aprendizado de máquina. I. Título.

FLÁVIA MORALES ANTUNES

ABORDAGEM NEURAL PARA A DETERMINAÇÃO DE DEPENDÊNCIAS ENTRE VARIÁVEIS DO PROCESSO DE TRANSPORTE E ÍNDICE DE CONTUSÕES EM BOVINOS

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Engenharia de Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 29 de junho de 2019.

Banca examinadora:

Profª. Drª. Ana Paula Lüdtke Ferreira
Orientadora

Prof. Dr. Sandro da Silva Camargo
Universidade Federal do Pampa

Prof. Dr. Milton Roberto Heinen
Universidade Federal do Pampa

AGRADECIMENTO

Quero agradecer a minha professora orientadora Ana Paula Lüdtke Ferreira, pelo empenho dedicado ao projeto, por todo apoio e paciência ao longo da elaboração do meu trabalho de conclusão de curso.

A todos os professores, por todos os conselhos, esclarecimentos de dúvidas e ajudas durante os meus estudos e elaboração do meu trabalho de conclusão de curso.

Aos meus pais, que apesar de todas as dificuldades sempre me ajudaram e me incentivaram durante os anos de faculdade. Ao meu namorado pela compreensão e apoio todos os dias e em todos os fins de semana dedicados aos estudos.

Também gostaria de deixar um agradecimento especial ao Frigorífico Silva, de Santa Maria, e ao professor Fernando F. Silva, da Universidade Federal de Santa Maria, pela cedência dos dados para a realização deste trabalho.

“Você quer passar o resto da sua vida vendendo água com açúcar ou quer uma chance de mudar o mundo?”

— Steve Jobs

RESUMO

As exigências quanto à qualidade da carne bovina, pelos consumidores, são crescentes. Dependendo de como ocorrem, os procedimentos de transporte do gado para abate influenciam na qualidade da carne produzida pelo frigorífico e enviada para o consumidor final. As pessoas têm se preocupado não só com a textura e maciez da carne, mas também com o bem-estar animal. Durante o transporte, desde o embarque até o desembarque, a quantidade de contusões associadas ao processo de transporte representa perdas financeiras para produtores e frigoríficos, além de questões importantes referentes à ética no trato com animais. Tendo em vista a melhoria do bem-estar animal e a qualidade em que os animais estão sendo transportados, este trabalho tem como objetivo desenvolver uma abordagem neural, visando a determinação das dependências entre as variáveis no transporte de bovinos. Os dados avaliados foram coletados no Frigorífico Silva, no município de Santa Maria (RS), pelo período de um ano, por pesquisadores da Universidade Federal de Santa Maria. Esses dados estão separados por lote, com a quantidade de 3283 registros. A rede neural utilizada foi a *Perceptron* multicamadas, com o uso do algoritmo para treinamento retropropagação do erro. A acurácia obtida foi de 94,5% utilizando quatro camadas ocultas, para ativação destas camadas foi utilizada a função *Elu*, e para a camada de saída a função *softmax*.

Palavras-chave: Redes neurais. Pecuária de precisão. Bem-estar animal. Aprendizado de máquina.

ABSTRACT

The demands on the quality of beef by consumers are increasing. Depending on how they occur, the procedure of transporting cattle to slaughter influence the quality of meat produced by the slaughterhouse and sent to the final consumer. People have been concerned not only with the texture and softness of meat, but also with animal welfare. During transport, from embarkment to landing, the number of contusions associated with the transportation process results in financial losses for producers and slaughterhouses, as well as important questions regarding ethics in dealing with animals. In order to improve the animal welfare and the quality of the animals being transported, this work aims to develop a neural approach, aiming to determine the dependencies between the variables when transporting cattle. The data evaluated were collected at Frigorífico Silva, Santa Maria (RS), for a period of one year, by researchers from the Federal University of Santa Maria. These data are separated by lot, with the amount of 3283 records. The neural network used was the multilayer textit Perceptron, using the algorithm for training error backpropagation. The obtained accuracy was 94,5% using four hidden layers, for activation of these layers was used the function Elu, and for the output layer the function textit softmax.

Keywords: Neural networks. Precision agriculture. Animal welfare. Machine learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CNN	Redes neurais convolucionais
CPU	Unidade central de processamento
GPU	Unidade de processamento gráfico
IA	Inteligência artificial
IBGE	Instituto brasileiro de geografia e estatística
MLP	Redes neurais *perceptron* multicamadas
pH	Potencial de hidrogênio
PIB	Produto interno bruto
RNA	Redes neurais artificiais
WNN	Redes neurais sem pesos

SUMÁRIO

1 INTRODUÇÃO

1.1 Pecuária de corte e bem-estar animal

Segundo o Instituto Brasileiro de Geografia e Estatística (IBGE), no primeiro trimestre de 2017 foram abatidas 7,4 milhões de cabeças de bovinos. Já no mesmo período de 2018, esse número foi de 7,72 milhões, perfazendo um aumento de 4,4%. A produção brasileira de carne bovina serve tanto ao mercado interno bem como ao mercado externo. A exportação da carne bovina *in natura*, sem processamento, aumentou 31,1% do primeiro trimestre de 2017 para o mesmo período de 2018, aumentando o ingresso de divisas no país (INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA, 2018).

O Brasil possui uma das maiores extensões territoriais do mundo, mas a eficiência do setor produtivo ainda não é suficiente. Comparado com outros países de área bastante inferior, a produção brasileira ainda tem muito espaço para crescimento. Para que o título de maior produtor seja mantido e com vistas ao aumento da produtividade, estão sendo feitas diversas melhorias tecnológicas nos setores da agricultura e da pecuária, possibilitando o desenvolvimento dessas áreas no país e, consequentemente, gerando um aumento do Produto Interno Bruto (PIB) (MAGGI *et al.*, 2017).

As exigências a respeito da qualidade da carne vendida no comércio vêm aumentando ao longo dos anos. As pessoas, naturalmente, procuram por produtos de boa qualidade, com garantias de segurança alimentar. Os consumidores avaliam a textura da carne, a maciez, o sabor, entre outros fatores. Além da qualidade da carne em si, nos últimos anos os consumidores passaram a se preocupar também com as condições de vida dos animais criados para abate. O bem-estar animal passou a ser um fator diferencial, tanto em termos éticos como em termos econômicos. Os consumidores estão dispostos a pagar mais pela garantia de um produto com procedência de animais que não sofreram desnecessariamente em seu processo de criação (LAMA *et al.*, 2013).

Para que a carne produzida atenda às expectativas de qualidade e bem-estar desejadas pelos consumidores, são necessários alguns cuidados: o transporte dos animais deve ser feito em tempo e condições adequadas, para evitar o estresse decorrente do manejo e que movimentações inadequadas gerem ferimentos e contusões; o recebimento de uma alimentação adequada e suficiente ao longo de toda a sua vida; tratamentos de saúde adequados e, de forma geral, quaisquer outras condições que possam gerar estresse aos animais. A quantidade de estresse tem relação direta com a qualidade da carne no final do

ciclo de vida do animal: animais estressados geram carnes escuras, rígidas e com pH alto, que não são adequadas para o consumo humano. Estresse durante o transporte do gado da propriedade para o abatedouro faz com que os animais se debatam e gerem contusões em si próprios ou nos outros; quedas durante o transporte podem fazer com que o animal caído seja pisoteado, causando a perda parcial ou total da carcaça. Carnes com contusões não podem ser vendidas no mercado e isso acarretará em uma falha econômica, causando prejuízo tanto para os criadores como para os frigoríficos (FILHO, 2006).

Diversos problemas ocorrem no transporte do gado, desde o embarque nas suas propriedades de origem até o abate no frigorífico. No embarque já podem ocorrer contusões pelo fato da rampa ser mais baixa que o caminhão, levando a lesões nas patas do animal; o tipo de manejo no local, frequentemente feito com cães ou chicotes elétricos, também gera estresse demasiado; algumas práticas de manejo inadequado, como derrubar a porta do caminhão sobre o animal para apressar o embarque também geram contusões. Durante o deslocamento, fatores que podem influenciar na perda de carne são: densidade inadequada da carga no caminhão, o sexo dos animais embarcados, animais com procedência de propriedades distintas, a temperatura da massa de ar, as condições da estrada, o tempo de viagem, entre outros. Quando a carga chega ao frigorifico, podem haver contusões decorrentes da descida do caminhão durante o processo de desembarque. Antes do abate, os animais são colocados em um ambiente de espera, para que possam desestressar-se da viagem, minimizando os problemas já mencionados, mas que podem ser potencializados quando animais de diferentes origens e temperamentos são colocados no mesmo espaço (ATKINSON, 2000).

O entendimento das relações entre as variáveis de transporte e o resultado de aproveitamento das carcaças (perdidas total ou parcialmente, devido a contusões nos animais) permite que o manejo dos animais e as condições de transporte sejam alterados de maneira a minimizar a quantidade de contusões decorrentes do processo. Embora haja conhecimento sobre algumas questões importantes (o fato de fêmeas terem mais contusões no transporte do que machos e terneiros, por exemplo), a análise de dados reais podem ajudar a entender melhor a relação entre esses elementos e a prever a quantidade de contusões esperadas, a partir das condições de embarque, deslocamento e desembarque dos animais.

1.2 Justificativa, objetivos e organização do trabalho

Este trabalho visa construir uma rede neural (HAYKIN, 2001) para que, a partir das variáveis do processo de transporte, coletadas por pesquisadores da Universidade Federal de Santa Maria no Frigorífico Silva, consiga-se efetuar uma previsão do peso esperado de contusões da carga no final no processo de transporte. Dessa maneira, é possível modificar as variáveis do processo de transporte visando a diminuição do número e da gravidade das contusões. Pretende-se, com este trabalho, contribuir para que o processo de transporte do gado para os frigoríficos resulte em melhores indicadores no que tange a qualidade da carne vendida, os ganhos financeiros dos produtores e frigorífico e o bem-estar dos animais.

O objetivo geral deste trabalho é descrito como:

Construir uma rede neural capaz de predizer qual a gravidade das contusões nos animais embarcados, a partir das variáveis do processo de transporte, usando como base dados coletados com relação ao índice de contusões de bovinos transportados.

Os objetivos específicos deste trabalho são:

1. Buscar uma forma de prever o que acontecerá se o gado for transportado nas condições determinadas através das variáveis do processo.
2. Permitir que os frigoríficos tenham maior controle sobre o processo, ocasionando menores perdas financeiras.

O trabalho está organizado como se segue: o Capítulo 1 já apresentou, na Seção 1.1, as questões referentes ao bem-estar animal relacionado ao transporte de gado bovino para o abate; nesta Seção 1.2, a justificativa e os objetivos e na Seção 1.3 os trabalhos correlatos encontrados na literatura são descritos. O Capítulo 2 apresenta os principais elementos do problema abordado e da técnica que será utilizada. No Capítulo 3 são descritos os materiais e métodos que fundamentam o desenvolvimento deste trabalho. A arquitetura da rede neural, as escolhas de variáveis e a preparação dos dados de entrada são apresentadas no Capítulo 4 e os resultados obtidos são apresentados e discutidos no Capítulo 5. No Capítulo 6 são apresentadas as considerações finais e os trabalhos futuros. O Apêndice A apresenta o código desenvolvido para a execução deste trabalho.

1.3 Trabalhos correlatos

Para que os animais não passem por situações de estresse e sofrimento, caracterizando uma situação contínua de bem-estar, estão sendo estudadas formas de melhorar o processo que tem a tendência de gerar mais contusões no gado: o transporte. Diversos autores apresentam resultados, na literatura, sobre a questão das condições de transporte de animais e sua influência em resultados de produção.

Camargo, Ferreira e Perez (2018) apresentam uma análise de fatores que influenciam o índice de contusões de bovinos transportados entre propriedades e um frigorífico da região sul do Brasil. O trabalho utiliza o algoritmo de agrupamento *K-means* (MACQUEEN, 1967), redes Bayesianas (MARQUES; DUTRA, 2002) e algoritmos de árvore de decisão (MONARD; BARANAUSKAS, 2003b). Com a aplicação dos três métodos descritos, os autores concluíram que as variáveis mais determinantes do processo de transporte são a densidade da carga do caminhão e o sexo dos animais.

Romero *et al.* (2013) utilizam modelos de análise estatística com o estudo de 86 viagens (1179 animais), um total de 14 a 16 animais por viagem, em condições de caminhão de dois eixos com a capacidade de até 10 toneladas, ventilação, teto de lona e piso antiderrapante. O principal modelo utilizado foi o de análise de regressão logística multivariada. Para a comprovação do modelo, foi utilizado o teste estatístico Hosmer-Lemeshow. A avaliação das lesões teve como padrão de avaliação os diâmetros dos hematomas, bem como suas possíveis causas. O trabalho também efetuou análises a partir do pH muscular e os hematomas e obteve como resultado que o aumento de hematomas se dava devido à densidade carga do caminhão, número e condições de parada e o tempo de espera; e os terneiros apresentavam um risco reduzido de probabilidade de possuir um alto pH muscular.

Mendonça *et al.* (2018) utilizam modelos de análise estatística, visando encontrar as causas das contusões. A linguagem utilizada para análise dos dados foi R. Para encontrar a influência de cada uma das variáveis sobre as lesões nas carcaças foi utilizado o critério Akaike (SAKAMOTO; ISHIGURO; KITAGAWA, 1986). Mendonça *et al.* (2018) encontrou a partir de seu estudo que o sexo do animal era o fator mais importante percebendo que a fêmea possui uma maior probabilidade de apresentar hematomas na sua carcaça tendo uma porcentagem de 91% mais hematomas que o macho. Também percebeu que as maiores contusões eram geradas com a densidade inadequada de carga do caminhão e longos períodos de transporte e descarga. O problema foi resolvido com

a alteração das instalações de carregamento, deixando mais adequada para os animais e também com a alteração no manejo, melhorando os cuidados do manejador com o bovino.

Os três trabalhos acima citados concordaram que um dos fatores que mais gera contusão nos bovinos é o fator da densidade da carga do caminhão. Camargo, Ferreira e Perez (2018) e Mendonça *et al.* (2018) fizeram uso, em seus trabalhos, do mesmo conjunto de dados que evidenciou que o sexo dos animais era um fator relevante, tendo as fêmeas transportadas uma maior probabilidade de se contundir. Romero *et al.* (2013), que fez uso de uma base de dados diferentes, encontrou que o tempo de espera, o número e as condições de parada também aumentam o estresse e geram mais contusões. Todos os trabalhos utilizaram modelos de análise estatística e/ou técnicas de Inteligência Artificial para encontrar quais eram os fatores relacionados às contusões geradas nas carcaças dos bovinos de corte. O presente trabalho irá prever, a partir das condições de transporte, quais são as prováveis consequências geradas, peso de contusões na carcaça bovina, por meio da utilização de uma técnica diferente dos trabalhos citados. A técnica escolhida será uma rede neural *perceptron* multicamadas, modelo inspirado no sistema nervoso central, capaz de aprender a partir de pares de exemplos condições de trasporte/quantidade de contusões, caracterizando uma parte da área denominada *aprendizado de máquina* ou *machine learning*.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo irá apresentar os conceitos de bem-estar animal e produção de carne usados neste trabalho. Também são apresentados conceitos de inteligência artificial, aprendizado de máquina, redes neurais, redes neurais *perceptron* multicamadas e o algoritmo *backpropagation*.

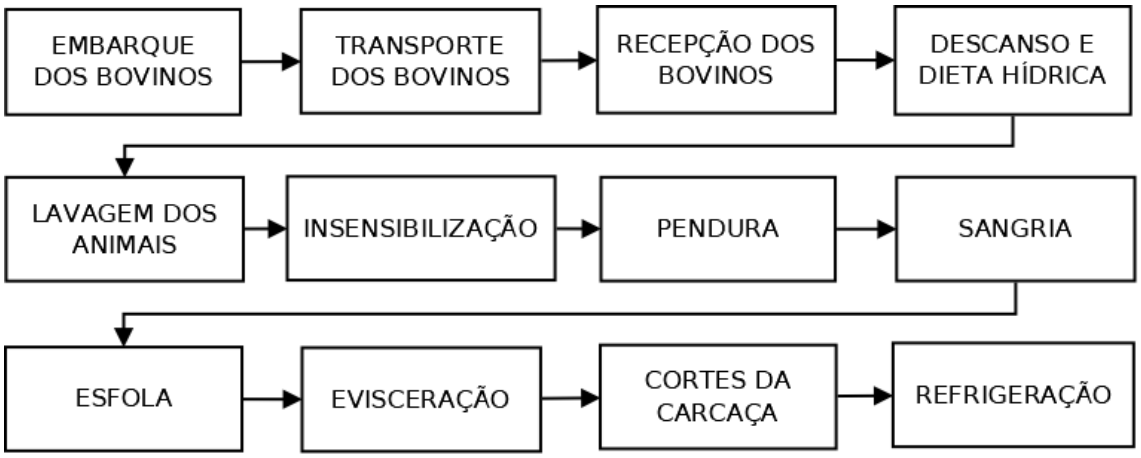
2.1 Bem-estar animal e produção da carne

Os estudos referentes ao bem-estar animal estão evoluindo crescentemente ao longo dos anos, devido ao prezar dos produtores, dos frigoríficos e da comunidade pela qualidade de vida dos animais. O bem-estar animal refere-se à qualidade de vida de um determinado animal ao longo de sua existência. Qualidade de vida, nesse caso, refere-se às condições de saúde, física e mental, traduzidas na prevenção e tratamento de doenças e na minimização do estresse produzido por condições de manejo (CAMARGO; FERREIRA; PEREZ, 2017). Os mamíferos são animais da mesma família dos humanos, com similar desenvolvimento das áreas cerebrais. Dessa forma, são capazes de produzir sentimentos como medo, prazer e alegria. A forma com que os animais são tratados pelas pessoas que os cercam também influencia seu bem-estar. Espera-se que os animais tenham afinidade com o ambiente em que se encontram e também com o manejador (BLOCK *et al.*, 2016), embora isso nem sempre seja possível ou garantido. Animais não domesticados são altamente sensíveis às modificações de seu ambiente, produzindo grandes quantidades de adrenalina quando em presença de situações estranhas que possam representar perigo. Com isso, seu nível de estresse aumenta e a tendência à fuga pode fazer com que o gado se machuque, quando em situações de contenção.

No Estado do Rio Grande do Sul, a maior parte dos sistemas produtivos agropecuários faz uso do chamado modelo extensivo, no qual o gado é deixado solto no campo, se alimentando à vontade, normalmente em campos de grande extensão. Para a produção de uma carne de qualidade, os cuidados com o gado devem ocorrer desde a criação. Se o gado for bem cuidado, com boa alimentação, sua carne será possivelmente de boa qualidade. Além dos fatores de cuidados e boa alimentação, outro fator tende a influenciar na qualidade da carne é a idade do animal: animais mais novos apresentam uma carne mais macia, clara e mais avermelhada, ao contrário dos animais mais velhos (BLOCK *et al.*, 2016).

Além dos cuidados no local de criação, os animais devem receber também todo benefício necessário para seu transporte até o frigorífico. O processo da produção da carne está ilustrado na Figura 1, que apresenta as seguintes fases: embarque dos bovinos, que inclui a forma que estão sendo colocados para dentro do caminhão; transporte dos bovinos, incluindo as condições nas quais vão ser transportados; recepção dos bovinos quando chegam no frigorífico, que também inclui a questão do manejo no desembarque dos animais; descanso e dieta hídrica, sendo relevante o local de parada no frigorífico e se o animal será deixado com outros animais desconhecidos; lavagem dos animais, para remoção das impurezas relacionadas à viagem; insensibilização, para manter animal desacordado até o seu abate final; as demais fases do processo (pendura, sangria, esfola, evisceração, cortes da carcaça e refrigeração) são particulares ao processo de abate e estão fora do escopo deste trabalho.

Figura 1 – Processo da produção da carne bovina



Fonte: Adaptado de: (KAWAGUCHI, 2000)

No embarque são necessários cuidados para o manejo do bovino. O manejador deve ter atitudes não agressivas, as rampas de subida devem ser adequadas para que os animais não batam as patas ao subirem, o modo que induzem o animal a entrar no caminhão não deve conter agressão, como o uso de choques e cachorros, entre outros fatores (CAMARGO; FERREIRA; PEREZ, 2017).

Na chegada no frigorífico, os animais devem possuir um local com condições adequadas para sua estadia, pois se o local for inadequado, eles podem ficar nervosos, estressados e agitados, prejudicando também sua adaptação (LUDTKE *et al.*, 2012). Salienta-se também que a mistura entre animais dominantes que não se conhecem não é adequada,

visto ser causa de brigas que podem envolver outros animais e produzir ferimentos com diferentes graus de gravidade, dependendo do sexo e da existência de animais aspados no grupo.

Na busca de um bom tipo de manejo de pré-abate, três elementos são essenciais: animais, instalações e pessoas. Dependendo da raça do animal, das pessoas que estão cuidando, do ambiente e do manejador, a reação desses animais pode variar. Uma boa projeção da fazenda e do frigorífico também pode facilitar o manejo. Especificamente, a forma que a mangueira possui e a impossibilidade do animal ver o local onde outros animais estão sendo conduzidos ou abatidos reduzirá o risco de agitação e estressados. O que também influencia é a forma com que as pessoas tratam estes animais e o tipo de manejo utilizado.

Quando os bovinos são originários de uma propriedade que com um bom tipo manejo no campo, eles tendem a ser mais calmos e responder melhor aos comandos dos manejadores do frigorífico. Contudo, mesmo os que foram criados de forma adequada podem apresentar problemas, seja pelo frigorífico não estar bem adequado ou pelos condutores presentes não apresentarem boa conduta, entre outros fatores. Os manejadores devem observar e ter conhecimento suficiente para entender as necessidades dos animais. Se possuírem esse conhecimento, eles irão solicitar que as instalações estejam adequadas para um melhor cuidado do animal. Além do conhecimento das necessidades, os manejadores também devem cuidar como se expressam perto do gado, procurar sempre evitar movimentos agressivos (LUDTKE *et al.*, 2012).

Após o desembarque no frigorífico, os bovinos são levados para um período de descanso. No local é essencial que haja bastante água para os animais. Se, no período do transporte a temperatura era alta, eles podem chegar estressados por estarem com muito calor, a água irá ajudar a melhorar o equilíbrio térmico e também a eliminar os conteúdos de dentro do intestino. Um bom manejo pré-abate é necessário para que os animais não tenham um maior estresse quando estão no frigorifico esperando para serem abatidos, pois o estresse prejudica a qualidade da carne e o bem-estar animal (LUDTKE *et al.*, 2012).

Após o período de descanso, os animais são levados ao box de atordoamento que é o local em que eles serão banhados para reduzir a poeira e a sujeita no local do abate. Quando chegarem no abatedouro os animais devem estar tranquilos para não prejudicar a carne. O gado é deixado inconsciente para o passo da sangria. Em alguns lugares a insensibilização é feita com marretas, arma de fogo, entre outras. Os animais são pendurados e logo após é cortada a garganta do animal para que o sangue escorra e não contamine a

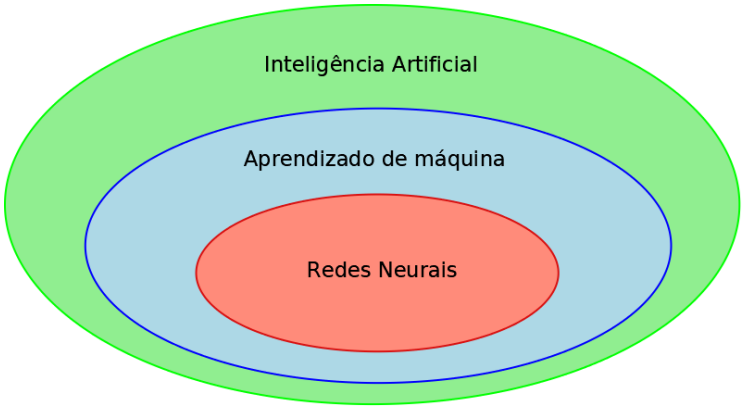
carne. A pele é retirada (processo chamado de esfola), bem como todos os órgãos internos (processo chamado de evisceração). Sua carne é dividida e lavada e direcionada aos resfriadores (BLOCK *et al.*, 2016).

As carnes após o abate devem ter uma aparência sem hematomas,resíduos, sujeiras de madeira ou metais, odores ruins, manchas de sangue, contusões ou ossos quebrados. Se alguma dessas características forem encontradas, a carne não poderá ser vendida (BLOCK *et al.*, 2016).

2.2 Redes neurais artificiais *perceptron* multicamadas

A Inteligência Artificial é uma área da computação que busca a criação de dispositivos que simulem o raciocínio humano, para que a máquina tenha um comportamento "inteligente", podendo solucionar problemas com base no seu próprio conhecimento (MUNAKATA, 1998). A inteligência artificial possui várias subáreas. A Figura 2 apresenta as duas subáreas que são relevantes para o desenvolvimento deste trabalho.

Figura 2 – Relação entre inteligência artificial, aprendizado de máquina e redes neurais



Fonte: Autora(2018)

Aprendizado de Máquina é a subárea da Inteligência Artificial que caracteriza um conjunto de técnicas que promovem o aprendizado por meio de um processo de treinamento a partir de um conjunto de dados reais. Através do aprendizado, a máquina é treinada para encontrar um correto mapeamento entre as entradas e os respectivos resultados, eliminando a necessidade de uma programação explícita. O processo de treinamento faz com que a máquina seja capaz de aprender por meio dos exemplos dados no processo de treinamento(MONARD; BARANAUSKAS, 2003a).

O aprendizado de máquina pode ser supervisionado, não supervisionado ou por reforço. O aprendizado é dito *supervisionado* quando, juntamente com os elementos de entrada de um exemplo, é fornecida a informação sobre qual é a saída correta para aquele exemplo específico (MONARD; BARANAUSKAS, 2003a). O aprendizado *não supervisionado* ocorre quando os dados de entrada são inseridos sem a informação de qual a saída desejada, e o algoritmo de aprendizado organiza as entradas recebidas de acordo com alguma métrica de similaridade (BARRETO, 2002). Na *aprendizagem por reforço* não é fornecida a resposta correta diretamente, mas o processo de aprendizado garante um esquema de punição e recompensa frente a respostas erradas ou corretas, respectivamente (FARIA; ROMERO, 1999).

As redes neurais aprendem a partir dos exemplos dados, mapeando entradas e saídas sem necessidade de uma programação que explicite a função que deverá ser construída. Por essa razão, uma rede neural pode ser vista como um construtor de funções não lineares. Esta aprendizagem só é possível devido ao aprendizado de máquina, que através de treinamentos consegue promover a aprendizagem da rede (BARRETO, 2002) e a construção da função que mapeia as entradas nas saídas.

O modelo inicial de neurônio proposto na literatura foi o modelo de McCulloch e Pitts (MCCULLOCH; PITTS, 1943), que objetivou simular o funcionamento das células do sistema nervoso central. É entendido que o modelo proposto por McCulloch e Pitts é diferente do biológico real e, por essa razão, diz-se que ele é "biologicamente inspirado". McCulloch e Pitts basearam seu modelo em cinco hipóteses: a primeira é que os neurônios só possuem atividades binárias, ou seja, o neurônio está ativado ou desativado; a segunda é que as redes neuronais são constituídas por conexões que representam as sinapses, que são regiões localizadas entre os neurônios e, pelas quais são transmitidos os impulsos nervosos para as células; a terceira é que o neurônio só é ativado se o valor de entrada for igual ou superior ao limite estabelecido para a função de ativação; a quarta é que sinapses inibitórias podem evitar o disparo do neurônio; e a quinta e última hipótese é que para um sinal trafegar entre os neurônios é exigido um certo tempo (MCCULLOCH; PITTS, 1943). A Figura 3 apresenta o modelo proposto em por McCulloch e Pitts.

O neurônio artificial possui a seguinte estrutura:

Entrada : entradas são os dados que constituem cada um dos exemplos que serão apresentados à rede e que serão utilizados na computação da saída; formalmente, a entrada é um vetor de n elementos numéricos $\mathbf{X} = (x_1, x_2, \ldots, x_n)$.

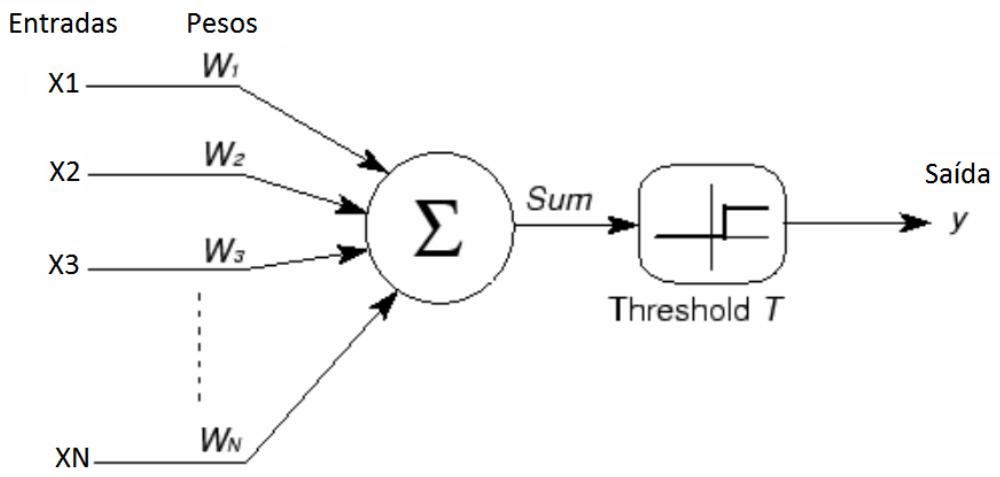
Pesos : os pesos da rede neural simulam o funcionamento das sinapses existentes no

neurônio biológico, ponderando os valores dos elementos de uma das camadas no cômputo dos valores dos elementos das camadas seguintes; formalmente, entre cada duas camadas da rede, com I e J neurônios, respectivamente, os pesos são representados por uma matriz W de dimensões $I \times J$, onde cada elemento w_{ij} da matriz representa o valor da ponderação do elemento i da primeira camada em relação ao cômputo do elemento j da camada seguinte.

Função de ativação : a função de ativação T modela o disparo entre os neurônios de uma camada e os neurônios da camada seguinte na rede, e seu valor indica se o neurônio irá conduzir o impulso para a próxima camada/saída. A função utilizada na imagem é a *Threshold*, que é uma função degrau, de saídas 0 ou 1 quando o valor do somatório fica abaixo ou atinge um determinado patamar, respectivamente, mas usualmente é uma função diferenciável similar;

Saída : os neurônios da camada de saída representam o valor de saída da função modelada pela rede e é o resultado das transmissões de neurônios que são referentes as entradas dadas; formalmente, a saída é um vetor de m elementos numéricos $\mathbf{Y} = (y_1, y_2, \ldots, y_m)$.

Figura 3 – Modelo de McCulloch e Pitts



Fonte: Adaptado de (KAWAGUCHI, 2000)

Na Equação 1 explicita-se a Figura-3 formalmente. Para que um neurônio consiga efetivar sua execução é necessário que seja realizado um somatório de todas as suas entradas **X** multiplicadas pela matriz de pesos **W**; após, a função *threshold* T irá realizar a ativação do neurônio, sendo o resultado desta função a saída y que é o resultado deste

neurônio. Cada neurônio de uma rede neural resulta em uma saída e esta saída é a entrada do neurônio da próxima camada, ou é a saída final da rede.

$$Y_j = f(\sum_{i=1}^{N} X_i W_{ij}) \tag{1}$$

onde f é a função de ativação (*threshhold*, neste caso específico).

A partir do modelo inicial proposto por McCulloch e Pitts, outras arquiteturas foram propostas. Embora os estudos sobre redes neurais tenham sido comprometidos devido às críticas de Minsky e Papert, provando que as um perceptron não conseguia aprender a função XOR (MINSKY; PAPERT, 2017), os estudos sobre as redes retornaram na década de 1980 (OSÓRIO; BITTENCOURT; OSóRIO, 2000). Em particular, os primeiros modelos consistentes de redes neurais começaram a ser construídos a partir dos trabalhos dos pesquisadores D. Parker em 1982 e D. Rumelhart em 1985. Porém, a partir do trabalho de Rumelhart e Hinton, que eliminou a objeção em relação a questão das funções linearmente separáveis, apresentado em “Parallel Distributed Processing - PDP"em (RUMELHART *et al.*, 1987), que utilizava uma rede neural multicamadas com o algoritmo de treinamento retropropagação de erro, resolvendo o problema da XOR, as pesquisas em redes neurais tomaram novo fôlego.

Para este trabalho foram estudados três modelos de redes neurais, para verificação da viabilidade de uso no problema abordado.

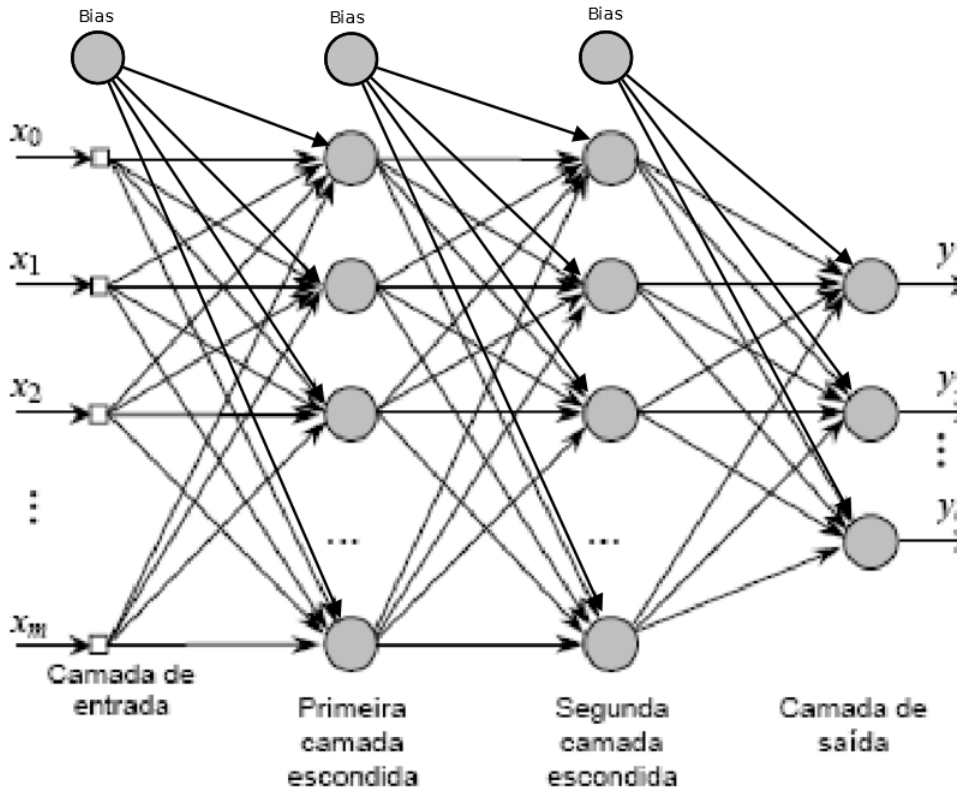
As redes estudadas foram escolhidas a partir de uma pesquisa bibliográfica sobre quais as redes mais utilizadas atualmente. As redes neurais sem pesos (*weightless neural networks*, (LUDERMIR; OLIVEIRA, 1994)) não utilizam as conexões (sinapses) para armazenar o estado de conhecimento da rede, mas elementos da memória. Cada dado de entrada está sendo direcionado a diferentes endereços da memória. As redes neurais sem pesos são baseadas em memória RAM e admitem que os dados de entrada e saída sejam apenas valores binários. Dessa forma, conseguem gerar um impacto positivo no tempo de resposta. A falta de documentação e de exemplos de uso deste tipo de rede, bem como a falta de ferramentas para sua construção, causou a eliminação desse método na aplicação deste trabalho.

As redes neurais convolucionais (*Convolutional Neural Network*, ou CNN) estão sendo bastante utilizadas para classificação de imagens, sendo uma extensão do modelo perceptron de múltiplas camadas, com adição de sucessivas camadas de convolução e de *pooling*, que compactam as características mais relevantes da imagem, gerando benefícios no aprendizado da rede (KOUSHIK, 2016). Um CNN assimila os filtros (camadas convo-

lucionais) e, de certa forma, aprende a reconhecer padrões (STUTZ, 2014). A quantidade de neurônios da camada convolutiva é referente ao número de saídas da camada anterior. Cada neurônio contém uma matriz de *pixels* que é multiplicada por uma matriz de valores denominada de *kernel* ou filtro de convolução. Os *pixels* utilizados dentro do neurônio são conduzidos das saídas da camada anterior, cada neurônio após a aplicação dos filtros gera saídas para as camadas subsequentes. A camada de *pooling*, também conhecida como camada de agrupamento, agrupa os dados existentes para facilitar o treinamento da rede. Ela tem como função principal a diminuição dos dados (VARGAS; PAES; VASCONCELOS, 2016). Apesar das vantagens mencionadas na literatura, a quantidade de dados disponível para esse trabalho fica muito aquém da necessária para o uso de uma CNN. Sendo assim, essa rede também não foi escolhida.

As redes neurais *perceptron* multicamadas (*multilayer perceptron neural networks*, ou MLP) são uma generalização do modelo de *perceptron* apresentado na Figura 3. As redes de *perceptrons* são constituídas por uma camada de neurônios com seus respectivos pesos sinápticos, com adição de um*bias*, que é um valor utilizado para melhorar os ajustes de pesos da rede. Por possuir apenas uma camada sua eficiência é mais restrita, porém, com mais neurônios e camadas é possível não só solucionar problemas lineares, mas também não lineares. A rede *perceptron* multicamadas é constituída por um vetor de valores de entrada, por um ou mais neurônios, por uma ou mais camadas intermediárias e suas interligações são definidas como *feedforward*, ou seja, cada neurônio de uma camada possui uma conexão com cada um dos neurônios da camada seguinte, como ilustrado na Figura 4. Este vetor de entrada passa pela camada inicial e por suas camadas subjacentes até que seja gerada a sua saída, que pode ser de um ou mais elementos (HAYKIN, 2001).

Figura 4 – Modelo de Rede Neural *Perceptron* Multicamadas



Fonte: Adaptado de: (HAYKIN, 2001)

As redes neurais *perceptron* multicamadas podem possuir quantos neurônios e camadas ocultas forem necessários. Porém, quanto mais neurônios e camadas existirem maior será o tempo de treinamento e processamento desta rede (HAYKIN, 2001). Quando existir um maior o conjunto de dados, a rede tende a demorar um tempo maior para ser treinada. Com isso são necessárias algumas otimizações para a redução do tempo de treinamento. Para a melhora no processamento e poder computacional podem ser utilizadas as Unidades de Processamento Gráfico (GPU) que é capaz de acelerar o processamento de tarefas específicas, em apoio à Unidade Central de Processamento (CPU) (GURGEL *et al.*, 2014).

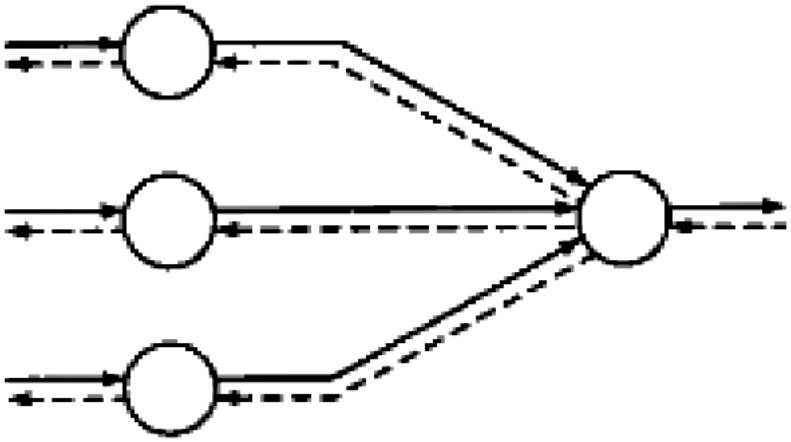
Os neurônios ocultos, existentes nas camadas escondidas, dão a capacidade da rede aprender problemas lineares ou não lineares, de forma que estes neurônios extraiam padrões das entradas resultando em uma solução (HAYKIN, 2001). O algoritmo de treinamento estabelece os pesos das conexões entre os neurônios, codificando dessa forma o aprendizado da rede. Neste trabalho, o algoritmo de treinamento usado é o de retropropagação de erro ou *backpropagation*.

A aprendizagem do algoritmo é realizada por meio da alteração dos pesos da rede

em relação à parte correspondente dos pesos no erro global. O erro é uma subtração da resposta desejada pela saída obtida da rede. Esse procedimento propaga o erro dentro da rede, da saída em direção à entrada e, por essa razão, chama-se *backpropagation*. A execução do backpropagation é composta por duas etapas: a propagação e a retropropagação. Na etapa que está ocorrendo a propagação, em que todos os pesos sinápticos são fixos, sem alterações e na etapa da retropropagação os pesos sinápticos são ajustados pela regra de correção do erro, para que os resultados reais se aproximem dos resultados desejados. Para que o valor do erro seja encontrado, ocorre uma subtração da resposta obtida pela resposta desejada, caracterizando um processo de treinamento supervisionado. O valor do erro é, então, retropropagado pela rede (HAYKIN, 2001).

Utilizando o algoritmo retropropagação de erro em redes perceptron multicamadas vão existir dois fluxos de sinais, os sinais funcionais e os sinais de erro, como mostra a Figura 5.

Figura 5 – Modelo de Rede Neural *Perceptron* Multicamadas com sinais funcionais e sinais de erro



Fonte: (HAYKIN, 2001)

O sinal funcional, ilustrado pelas linhas não pontilhadas da Figura 5, é o sinal que se propaga para frente na rede, neurônio por neurônio, ao chegar no final se torna um sinal de saída. Este sinal é calculado com base nas entradas e pesos dos neurônios, Equação 1.

O sinal de erro, ilustrado pelas linhas pontilhadas da Figura 5, é o sinal que se propaga para trás na rede que, ao contrário do sinal funcional, não se origina das entradas e sim da saída da rede.

A fórmula para o cálculo do erro é representada pela Equação 2, que dá-se pela

subtração da resposta desejada (d_k) pela saída obtida da rede (y_k).

$$e_k = d_k - y_k \tag{2}$$

O valor da energia total do erro é definido pela Equação 3, sendo e_k o sinal de erro e C o conjunto total de neurônios da camada de saída da rede.

$$\varepsilon = \frac{1}{2} \sum_{k \in C} e_k^2 \tag{3}$$

Quando o neurônio é alimentado pelas suas entradas é obtido um campo local induzido (v_j), que é gerado na entrada da função de ativação. Equação 4.

$$v_j = \sum_{i=1}^{m} Y_i W_{ij} \tag{4}$$

y_j é o sinal que aparece na saída do neurônio, é denominado de sinal funcional. É a aplicação da função de ativação ao campo local induzido(v_j), está representado na Equação 5.

$$y_j = f(v_j) \tag{5}$$

O algoritmo *backpropagation* aplica correção nos pesos sinápticos, que é chamado de fator de sensibilidade, que através dele é determinado a direção da busca dos pesos, para o peso sináptico $\omega_i j$. Esta equação é definida pela derivada parcial da energia do erro em função dos pesos, Equação 6.

$$\frac{\partial \varepsilon}{\partial w_{ij}} \tag{6}$$

Utilizando esta derivada com a regra da cadeia, o gradiente é expressado como a Equação 7.

$$\frac{\partial \varepsilon}{\partial W_{ij}} = \frac{\partial \varepsilon}{\partial e_j} \frac{\partial e_j}{\partial y_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_j}{\partial w_{ij}} \tag{7}$$

Para ser encontrado o gradiente local que irá gerar as modificações nos pesos sinápticos, é necessária a aplicação de várias diferenciações. A primeira é diferenciação da Equação 3 em relação ao sinal de erro (e_j), Equação 8.

$$\frac{\partial \varepsilon}{\partial w_{ij}} = e_j \tag{8}$$

A segunda é diferenciação da Equação 2 com relação o sinal funcional (y_j), Equação 9.

$$\frac{\partial \varepsilon}{\partial w_{ij}} = -1 \tag{9}$$

A terceira diferenciação é da Equação 5 com relação a v_j, Equação 10

$$\frac{\partial y_j}{\partial v_j} = f'(v_j) \tag{10}$$

A ultima é a diferenciação da Equação 4 em relação a $_{ij}$, Equação 11

$$\frac{\partial v_j}{\partial w_{ij}} = y_i \tag{11}$$

A aplicação da Equações 8 com a Equação 11 na Equação 7 e aplicando uma correção Δw_{ij} é gerada a Equação 12.

$$\Delta \omega_{ij} = -\eta \frac{\partial \varepsilon}{\partial w_{ij}} \tag{12}$$

A taxa de aprendizado do algoritmo de treinamento retropropagação do erro é definida pela variável η. O gradiente local (δ) é definido pela diferenciação da energia do erro (ε) pelo campo local induzido (v_j), Equação 13.

$$\delta = \frac{\partial \varepsilon}{\partial v_j} \tag{13}$$

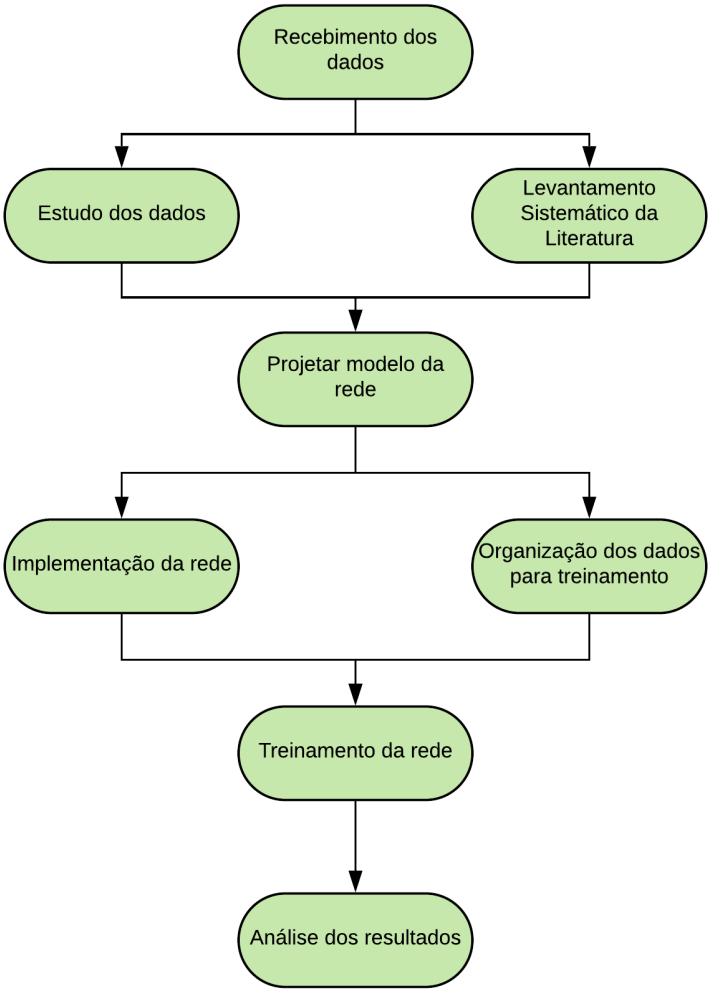
A rede a ser utilizada será a rede neural *perceptron* multicamadas. A escolha desta rede ocorreu pelo estudo das redes selecionadas anteriormente. As redes neurais sem pesos não foram utilizadas, devido a escassez de bibliografias e conteúdos compreensíveis. E as redes neurais convolucionais não foram utilizadas pela desnecessidade da utilização de um alto processamento para solução deste problema.

3 METODOLOGIA

3.1 Etapas do trabalho

Este trabalho tem como objetivo a criação de uma rede neural para prever qual a gravidade esperada das contusões dos animais embarcados em uma propriedade, a partir das variáveis do processo de transporte. Para execução dos objetivos a metodologia do trabalho está estruturada em etapas, apresentadas pelo diagrama da Figura 6 e descritas na sequência.

Figura 6 – Diagrama de etapas utilizadas para a Metodologia



Fonte: Autor(2019)

Recebimento dos dados - a primeira etapa foi o recebimento dos dados que foram co-

letados por pesquisadores da Universidade Federal de Santa Maria no Frigorífico Silva. Os dados, que foram disponibilizados em planilhas do Excel, são do período de 1 ano.

Estudo dos dados obtidos - a segunda etapa foi verificar quais eram os dados recebidos e a importância deles para o transporte, para ser possível encontrar as relações entre as variáveis. Isto foi feito com base em pesquisas na literatura, que informavam quais eram os impactos de algumas variáveis específicas para a geração de menos contusões e um melhor bem-estar animal.

Levantamento sistemático da literatura - a terceira etapa foi a pesquisa de referências bibliográficas, que foi feita através da busca por palavras chaves em diversos sites, para que os trabalhos encontrados pudessem ser utilizados como referência no trabalho e como base para a escrita da fundamentação teórica.

Projetar modelo de rede - na quarta etapa, foi escolhida qual era a rede a ser utilizada e foi feito o mapeamento de todas entradas e saídas, bem como um desenho para exemplificar como seria a visualização da rede.

Implementação da rede - na quinta etapa, foi construída a rede neural *perceptron* multicamada utilizando o modelo da Google DNNClassifier.

Organização dos dados para treinamento - na sexta etapa, foi feita uma análise de quais dados eram necessários para o treinamento da rede, para serem utilizados como entrada na rede. Dados que pudessem prejudicar o treinamento da rede foram retirados.

Treinamento da rede - na sexta etapa foi feito o treinamento da rede com uso do algoritmo retropropagação do erro. Foram testados além do algoritmo de retropropagação do erro outros três otimizadores, *AdamOptimizer*, *AdagradOptimizer* e *RMSPropOptimizer*. Para função de ativação, foram testados quatro tipos, são eles: *Elu*, *Relu*, *Sigmoide* e *Tangente hiperbólica*.

Análise dos resultados - Com a rede concluída, foram utilizados métodos para avaliação e validação da rede, o método de validação utilizado foi o de validação cruzada, os dados foram divididos em dois conjuntos: de treinamento e de teste. Para avaliar a rede foi utilizada a acurácia.

A revisão da literatura foi feita, utilizando a metodologia de buscas bibliográficas, por meio de palavras chaves em fontes previamente escolhidas. Primeiramente foi feita a busca por palavras-chaves em inglês. Os resultados obtidos estão apresentados na

Tabela 1. Nesta tabela estão representados por um traço (“–”) os resultados que corresponderam a um número muito alto de artigos retornados.

Tabela 1 – Contagem sistemática da literatura - Inglês

	Palavra-chave/Fonte	Precision Agriculture	47CAI	IEEE Xplorer Digital Library
1	Neural Network	72	0	-
2	Transport Cattle	4	0	16
3	Transport Cattle Neural Network	0	0	0
4	Livestock	-	0	546
5	Transport Livestock	-	0	13

Fonte: Autora (2019)

Foram analisados todos resultados obtidos das pesquisas por palavras chaves em inglês. O método de análise utilizado deu-se através da leitura dos resumos de cada trabalho para verificação da possível relação com este trabalho. Para a pesquisa, utilizou-se as palavras com o operador *and* e também sem. Mesmo com o operador *and*, a filtragem de resultados buscava por apenas uma das palavras digitadas no campo de texto. Nenhum dos trabalhos encontrados pôde ser utilizado como correlato.

O próximo passo foi efetuar a busca de palavras-chave em português, em revistas e em um congresso que recebe artigos em português. Os resultados obtidos estão representados na Tabela 2.

Tabela 3 – Contagem sistemática da literatura - Português

	Palavra-chave/Fonte	47CAI	Pesquisa Agropecuária Brasileira	Revista Brasileira de Ciências Agrárias
1	Redes Neurais	0	15	4
2	Transporte Gado	1	1	0
3	Transporte Gado Redes neurais	0	0	0
4	Pecuária	0	77	16

Fonte: Autora (2019)

Os resultados em português foram obtidos e analisados, sendo encontrado um único trabalho correlato no 10º *Congreso Argentino de Agroinformática* (47CAI) . A partir do artigo correlato encontrado, foi feita uma análise de todos os artigos referenciados pelo autor, sendo encontrados mais dois artigos correlatos para utilização neste trabalho.

3.2 Material e métodos

Os dados utilizados foram coletados por pesquisadores da Universidade Federal de Santa Maria no Frigorífico Silva, em Santa Maria, durante um período de um ano. Os dados recebidos foram disponibilizados em uma planilha de Excel e foram estudados, observando quais as variáveis relevantes e quais as relações entre elas.

O estudo da fundamentação teórica sobre redes neurais artificiais foi realizado com ênfase nas redes neurais convolucionais, redes neurais *perceptron* multicamadas e as redes neurais sem pesos, para avaliar qual a melhor forma para solucionar o problema.

O motivo da escolha destas três redes para estudo foi que as convolucionais estão sendo bastante utilizadas atualmente possuindo um número relevante de referencial teórico, as *perceptron* multicamadas foram escolhidas por ser uma rede clássica muito utilizada para dados e solução de problemas não lineares, já as redes neurais sem pesos foram pesquisadas para verificar o possível uso dela com dados.

Foi necessário realizar um estudo sobre a linguagem Python, linguagem de progra-

mação utilizada para a implementação da rede neural, pois não é uma linguagem abordada no curso. A linguagem Python foi escolhida por ser uma linguagem bastante ativa, são disponibilizados cada vez mais novos pacotes e atualizações. Esta linguagem também é bastante utilizada para construção de redes neurais, possuindo vários pacotes específicos para a construção das redes, como o pacote que será utilizado, *Tensorflow* (GOOGLE BRAIN, 2019).

A partir dos estudos feitos na literatura, houve uma dificuldade de encontrar referências precisas sobre as redes neurais sem pesos, devido serem bem restritas, esta rede não será utilizada neste trabalho. Também observou-se que a rede neural convolucional não se aplica a esses dados, pois seus filtros de convolução e *pooling* são utilizados para diminuição de grandes números de dados, normalmente em imagens, por possuírem uma grande quantidade de *pixels*. Esta rede também demanda de um grande poder de processamento, sendo assim desnecessária devido ao pouco volume de variáveis. Visto que as variáveis de entrada que serão utilizadas não são de grande quantidade e são atributos de uma tabela, a Rede Neural Perceptron Multicamadas é a mais adequada para ser utilizada. Por possuir mais de uma camada de neurônios, esta rede é capaz de solucionar não apenas problemas lineares, mas também problemas não lineares. Os dados recebidos estão representados nas Tabelas 5 e 6.

Tabela 5 – Variáveis recebidas - planilha 1

-	Variáveis	Tipo	Intervalo
1	Data do abate	String	01/01/2014 - 01/01/2015
2	Mês do abate	Numérico	1 - 12
3	Lote	Numérico	1 - 43
4	Produtor	String	Nome do produtor
5	Cidade de origem	String	Nome da cidade
6	Microrregião	String	Nome da microrregião
7	Mesorregião	String	Nome da mesorregião
8	Preço base	Numérico	0 - 20
9	Quantidade	Numérico	0 - 513
10	Rendimento do Lote	Numérico	0 - 100 (%)
11	Sexo do animal	String	M, F
12	Sequencial abate	Numérico	1 - 730
13	Peso da carcaça fria	Numérico	8,62 - 615,34

14	Gordura	Numérico	3 - 5
15	ID Comprador	Numérico	30 - 142
16	Destino	String	Sigla do destino
17	Preço do kg	Numérico	0 - 20
18	Raça	String	Nome da raça
19	Programa	String	Nome do programa

Fonte: Autora (2019)

Tabela 6 – Variáveis recebidas- planilha 2

	Variáveis	Tipo	Intervalo
1	Placa do caminhão	String	Digitos da placa
2	Data de saída	String	31/08/2014 - 29/07/2015
3	Data + 1	String	01/08/2014 - 30/07/2015
4	Produtor	String	Nome do Produtor
5	Município	String	Nome do município
6	Quantidade de animais	Numérico	0 - 386
7	Quantidade de bois	Numérico	0 - 104
8	Quantidade de Vacas	Numérico	0 - 70
9	Categoria	String	M, F
10	Tipo de mangueira	String	Nome do tipo
11	Tipo de veículo	String	Nome
12	Km em estrada de chão	Numérico	0 - 800
13	Km total	Numérico	6 - 612
14	Pessoas no carregamento	Numérico	0 - 15
15	Condições da mangueira	String	B, RE, R
16	Condições do tempo	String	B, RE, R
17	Condições da estrada	String	B, RE, R
18	Manejo na fazenda	String	B, RE, R
19	Caminhão atolou	String	S ou N
20	Único embarcador	String	S ou N
21	Horário de saída	String	00:00 - 23:59
22	Horário de chegada	String	00:00 - 23;59
23	Tempo de viagem	Numérico	0 - 34550

24	Hora de entrada no frigorífico	String	00:00 - 23:59
25	Hora de saída	String	00:00 - 23:59
26	Único embarcador	String	S ou N
27	Motorista	String	Nome do motorista
28	Peso líquido	Numérico	0 - 123000
29	Horário de Conferência 1	String	00:00 - 23:59
30	Condição 1	String	B, RE, R
31	Horário de Conferência 2	String	00:00 - 23:59
32	Condição 2	String	B, RE, R
33	Horário de Conferência 3	String	00:00 - 23:59
34	Condição 3	String	B, RE, R
35	Horário de Conferência 4	String	00:00 - 23:59
36	Condição 4	String	B, RE, R
37	Horário de Conferência 5	String	00:00 - 23:59
38	Condição 5	String	B, RE, R
39	Horário de Conferência 6	String	00:00 - 23:59
40	Condição 6	String	B, RE, R
41	Horário de Conferência 7	String	00:00 - 23:59
42	Condição 7	String	B, RE, R
43	Horário de Conferência 8	String	00:00 - 23:59
44	Condição 8	String	B, RE, R
45	Horário de Conferência 9	String	00:00 - 23:59
46	Condição 9	String	B, RE, R
47	Horário de Conferência 10	String	00:00 - 23:59
48	Condição 10	String	B, RE, R
49	Horário de Conferência 11	String	00:00 - 23:59
50	Condição 11	String	B, RE, R
51	Horário de Conferência 12	String	00:00 - 23:59
52	Condição 12	String	B, RE, R
53	Horário de Conferência 13	String	00:00 - 23:59
54	Condição 13	String	B, RE, R
55	Condição 14	String	B, RE, R
56	Condição 15	String	B, RE, R
57	Data do abate	Numérico	01/09/2014 - 30/06/2015

58	Produtor 2	String	Nome do segundo produtor
59	Lote	Numérico	1 - 43
60	Data Lote	Numérico	códigos
61	Quantidade final	Numérico	0 - 513
62	Sexo	String	M, F
63	Peso médio da carcaça	String	149,72 - 615,34

Fonte: Autora (2019)

Para a organização dos dados necessários para produzir os exemplos de aprendizado, primeiramente foram analisados quais eram os dados que apresentavam inexistência ou incorreção de informação em uma ou mais colunas das tabelas. A solução encontrada foi a exclusão destes valores, para não prejudicar o treinamento e aprendizagem da rede. Após esse processo, foi feita a redução da quantidade de colunas: os horários de conferência foram reduzidos por meio da contagem de todos em cada viagem e acrescentado o somatório em uma nova coluna com apenas o valor total de conferências e também uma nova coluna constando o calculo de quantas conferência foram feitas por cada quilômetro andado.

Após avaliação e estudo sobre as variáveis restantes, mais algumas foram removidas:

- Placa
- Data
- Data +1
- Categoria
- Hora de saída
- Hora de chegada
- Hora de entrada
- Condições (1-15)
- Conferências (1-13)
- Data do abate
- Lote

A variável Placa é uma variável que possui muitos valores diferentes, isso poderia prejudicar o treinamento da rede. A rede poderia dar mais importância para essa variável

do que para outras que possivelmente seriam de mais importância. Para não prejudicar o aprendizado da rede esta variável foi removida.

As variáveis de Data, Data+1, hora de chegada, hora de entrada e Data do abate teriam influência sobre o processo de transporte se estivessem relacionadas ao clima. Se os dados de clima estivessem disponível, essas variáveis citadas poderiam ser utilizadas para verificar qual a temperatura em determinada data ou hora. Pela ausência dos dados de clima, essas variáveis foram removidas.

A variável categoria possui se os animais são machos ou fêmeas, mas em alguns carregamentos constava que os animais eram machos, mas possuía tanto fêmeas como machos. Nas variáveis boi e vaca, contém a quantidade de cada um existente no lote. Por esse motivo a variável categoria foi removida.

A variável lote foi removida por não ser relevante, cada registro é de um lote diferente, então não é necessário uma variável a mais para referenciar o lote em questão. Por isso essa variável foi removida.

A rede foi organizada com um total de 28 variáveis de entrada e três possíveis saídas. Visto que as redes neurais aceitam como entrada apenas dados numéricos, os dados existentes nas tabelas serão convertidos para valores numéricos e escalonados para utilizá-los na entrada da rede. A biblioteca utilizada para conversão dessas variáveis foi a *feature columns* do pacote *Tensorflow*.

A saída à ser utilizada na rede neural *perceptron* multicamadas será a Contusão, que possui três possíveis saídas, mostradas na Tabela 5.

Tabela 7 – Variáveis usadas na saída

	Variáveis	Tipo	Intervalo
1	Peso da Contusão	String	Médio/Leve/Pesado

Fonte: Autora (2019)

Após a implementação da rede concluída, deu-se início ao treinamento da rede, que foi feito a partir do algoritmo de retropropagação do erro. Este algoritmo retropropaga o erro, encontrando os pesos adequados para a rede. A quantidade de vezes em que este algoritmo será executado depende da quantidade de épocas definidas, um número

baixo/alto de épocas pode não resultar em uma aprendizagem elevada, por esse motivo deve ser encontrado o número de épocas adequados para melhor valor do erro.

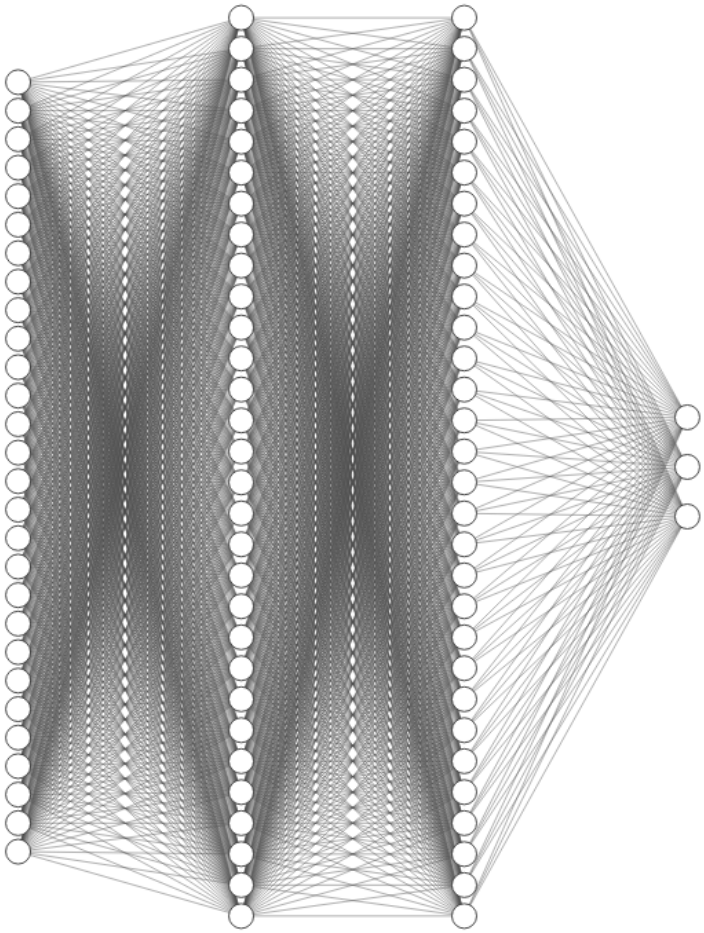
Com a rede já treinada, foi feito o cálculo da acurácia que é a proximidade entre os dados reais e os dados previstos pela rede, ou seja, a porcentagem de acerto. Para obter esta acurácia foi utilizada uma função chamada *accuracy* da biblioteca *Tensorflow.metrics*. Esta função faz o cálculo da precisão verificando o valor de saída y verdadeiro e o valor previsto pela rede e retorna a taxa de acerto.

4 ARQUITETURA E DESENVOLVIMENTO

4.1 Arquitetura da Rede

A implementação da rede foi feita com a utilização da biblioteca da Google, Tensorflow. Esta biblioteca foi criada pela equipe do *Google Brain* com o objetivo de facilitar a implementação de modelos de aprendizado de máquina. Um modelo de visualização da rede está apresentado na Figura 7.

Figura 7 – Modelo de Rede Neural *Perceptron* Multicamadas - MLP



Fonte: Autora(2019)

A arquitetura da rede está composta por 28 váriaveis de entradas **X**, definidas na Tabela 9 e três saídas **Y**, correspondentes à gravidade esperada das contusões e definidas categoricamente como: *leve*, *média* e *pesada*, apresentadas na Tabela 10. O tratamento da saída foi feito utilizando três índices, o índice 0 para contusões leves, 1 para contusões

médias e 2 para contusões pesadas.

Tabela 9 – Dados de entrada da rede neural

	Dados	Categoria
1	Produtor	Categórica
2	Município	Categórica
3	Quantidade de Bois no carregamento	Numérica
4	Quantidade de Vacas no carregamento	Numérica
5	Quantidade de Touros no carregamento	Numérica
6	Quantidade de Búfalos (Macho)	Numérica
7	Quantidade de Búfalos (Fêmea)	Numérica
8	Mangueira	Categórica
9	Tipo de Veículo	Categórica
10	Km de chão	Numérica
11	Km final	Numérica
12	Pessoas no carregamento	Numérica
13	Condições da mangueira	Categórica
14	Condições do tempo	Categórica
15	Condições da estrada	Categórica
16	Manejo na fazenda	Categórica
17	Caminhão atolou	Categórica
18	Único embarcador	Categórica
19	Tempo de viagem	Numérica
20	Único embarcador 2	Categórica
21	Motorista	Categórica
22	Peso total embarcado	Numérica
23	Produtor 2	Categórica
24	Quantidade total embarcada	Numérica
25	Sexo	Categórica
26	Peso médio carcaça fria	Numérica
27	Numero de conferências	Numérica
28	Número de conferências por km	Numérica

Fonte: Autora(2019)

Tabela 10 – Dados de saída da rede neural

	Dados	Categoria
1	Leve	0
2	Médio	1
3	Pesado	2

Fonte: Autora(2019)

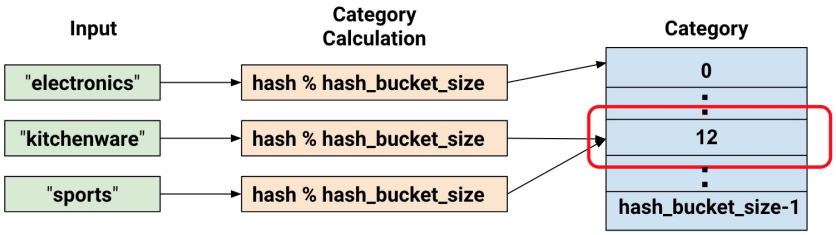
Os dados extraídos da Tabela 9 foram salvos em um vetor X e os dados extraídos da Tabela 10 salvos em um vetor Y.

Os dados de entrada contém dados numéricos e também dados categóricos. Para que os dados pudessem ser utilizados como entrada no modelo de rede DNNClassifier foi utilizada a função *feature_columns* da biblioteca TensorFlow, para padronização dos dados. Esta função pode usada tanto para dados numéricos quanto para dados categóricos e verifica a variação dos dados. A função mapeia cada tipo de argumento em um endereço de uma tabela hash utilizando o pseudocódigo representado na Equação 14.

$$feature\_id = hash(raw\_feature)\%hash\_bucket\_size \tag{14}$$

Após a execução da Equação 14, as entradas são mapeadas para seus respectivos espaços na tabela hash, representado na Figura 8

Figura 8 – Mapeamento colunas categóricas para tabela hash



Fonte: (GOOGLE BRAIN, 2019)

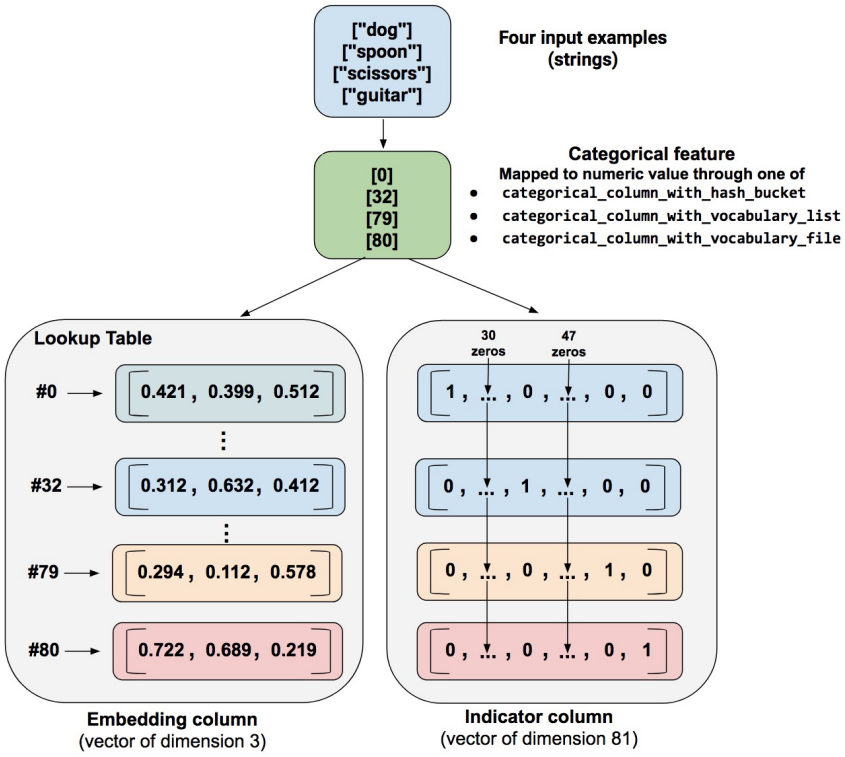
Para melhorar a separação dos dados podem ser utilizados dois tipos de função, a de indicador (*indicator columns*) e a função de incorporação (*embedding columns*). Essas

duas funções mapeiam cada entrada categórica em um dado numérico, denominado numérico categórico. A *Indicator columns* gera um vetor para cada categoria, de dimensão correspondente ao número de categorias totais. Supondo que a Fórmula 14 foi aplicada em um dado e o resultado foi de 79, a posição 79 do vetor será preenchida com o valor um e o restante das posições do vetor serão preenchidos com zero. A *embedding columns* gera um vetor de três dimensões para cada categoria. Segundo (GOOGLE BRAIN, 2019) os vetores de três dimensões são gerados durante o treinamento da rede:

> "*As atribuições acontecem durante o treinamento. Ou seja, o modelo aprende a melhor maneira de mapear seus valores categóricos numéricos de entrada para o valor do vetor de incorporação para resolver seu problema. Colunas de incorporação aumentam os recursos do seu modelo, pois um vetor de incorporação aprende novos relacionamentos entre as categorias a partir dos dados de treinamento*" (GOOGLE BRAIN, 2019).

A representação destas duas funções está representada na Figura 9.

Figura 9 – Função indicador e Função incorporação



Fonte: (GOOGLE BRAIN, 2019)

A utilização da função indicador nesta abordagem seria inviável, por existir uma grande variação de dados dentro de algumas colunas categóricas. Essa grande variação dos dados tornaria a dimensão dos vetores demasiado grande, causando uma sobrecarga de memória para o treinamento. A função de incorporação, armazena os mesmos da-

dos em um vetor de três dimensões, dissipando a sobrecarga, por esse motivo a função indicador será utilizada.

Os dados numéricos foram convertidos para o tipo numérico *float*, ou de ponto flutuante, e aplicados na padronização (*standardization*), para deixar os valores na mesma escala. A padronização x_i^*, para cada variável x_i feita, está apresentada na Equação 15, onde $E(\mathbf{X})$ é o valor esperado, ou média, e $\sigma(\mathbf{X})$ é o desvio padrão.

$$x_i^* = \frac{x_i - E(\mathbf{X})}{\sigma(\mathbf{X})} \tag{15}$$

Com a padronização concluída, foi utilizada a mesma função utilizada nos dados categóricos, a *feature_columns* para mapear os dados numéricos para uma tabela hash.

Os dados utilizados foram separados em dois conjuntos, um conjunto de dados para testes e outro conjunto para dados de treinamento. Para a execução do treinamento da rede foi utilizado o método de envio de amostra de dados (*batch_size*). O tamanho das amostras utilizadas de 32, ou seja, a cada execução 32 dados aleatórios do conjunto de treinamento são enviados para treino. Cada envio do *batch_size* para o treino é denominado de *step*. Uma época da rede dá-se quando todos os dados existentes já foram treinados. Uma rede com o *batch_size* de valor 32 e o total de dados de 3200, demora 100 steps para conclusão de 1 época. O método de envio por amostras foi utilizado para prevenir uma possível sobrecarga de memória.

O modelo de classificador utilizado foi o *DNNClassifier*. *DNNClassifier* significa que é uma rede neural densa, ou seja, os neurônios de cada camada estão interligados com todos neurônios da próxima camada.

O método aplicado para avaliação da perda foi entropia cruzada (Cross-Entropy). Este método verifica a variância entre a previsão feita pelo modelo(YP) e o valor real(Y), utilizando a Equação 16.

$$CE(Y,YP) = -\sum_i Y_i \log(YP_i) \tag{16}$$

4.2 Otimizadores

No processo de treino das redes neurais são feitas atualizações nos pesos para a minimização do erro. Para melhorar este processo são utilizados otimizadores. O de uso mais comum é o de descida do gradiente, sendo o próprio *backpropagation*, os outros oti-

mizadores são o algoritmo de retropropagação do erro com algumas modificações. Foram testados quatro otimizadores: *GradientDescendentOptimizer*, *AdamOptimizer*, *AdagradOptimizer* e *RMSPropOptimizer*.

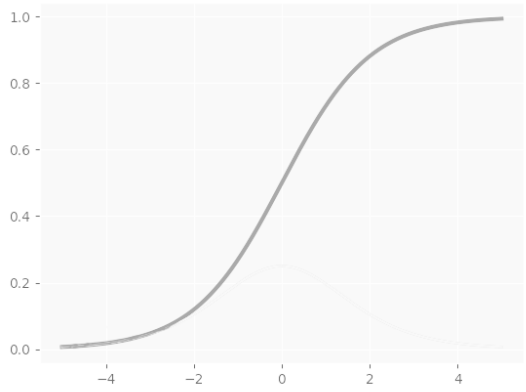
- Método de descida do gradiente (*GradienteDescendentOptimizer*) é o uso direto do algoritmo de retropropagação do erro. Os pesos são inicializados aleatoriamente, todas as camadas são executadas(propagação) e é calculado o erro. No sentido inverso(retropropagação), são calculados novos pesos para cada neurônio e é verificado se o resultado está adequado, se não estiver, é calculada novamente a propagação.
- Método de estimação adaptativa de momento (*Adam*) - este método calcula para cada parâmetro a taxa de aprendizado, utilizando o gradiente de primeira ordem. Por calcular a taxa de aprendizado por parâmetro, não é necessário um gradiente fixo, ele utiliza gradientes esparsos e também calcula a média decadente dos últimos gradientes (JANGID; SRIVASTAVA, 2018).
- Método de gradiente adaptativo (*Adagrad*) - é feita a divisão da taxa de aprendizado pela soma do gradiente quadrado, resultando em uma pequena taxa de aprendizado (JANGID; SRIVASTAVA, 2018).
- Método *RMSProp* - conserva a média do gradiente quadrado de cada peso e divide o gradiente pela raiz do valor médio quadrático (RMS) (JANGID; SRIVASTAVA, 2018).

4.3 Funções de ativação

Para ativação das camadas ocultas foram testadas quatro tipos de funções de ativação, a sigmoide (Equação 17, Figura 10), a tangente hiperbólica (Equação 18, Figura 11), a relu (Equação 19, Figura 12) e a elu (Equação 20, Figura 13).

$$Y = \frac{1}{1+e^{-x}} \tag{17}$$

Figura 10 – Função de ativação - Sigmóide

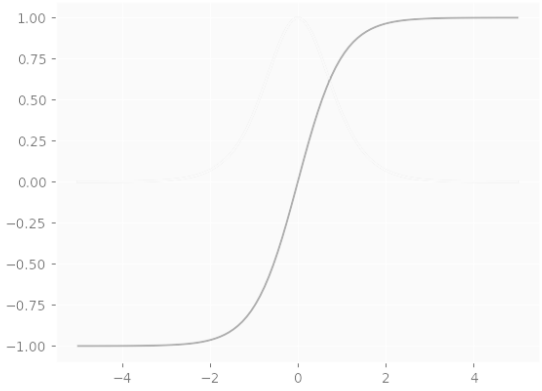


Fonte: Adaptado de:(FACURE, 2017)

Na função sigmoide os valores altos tendem a 1, já os valores baixos, que são negativos tendem a 0 (NIELSEN, 2015).

$$Y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{18}$$

Figura 11 – Função de ativação - Tangente Hiperbólica



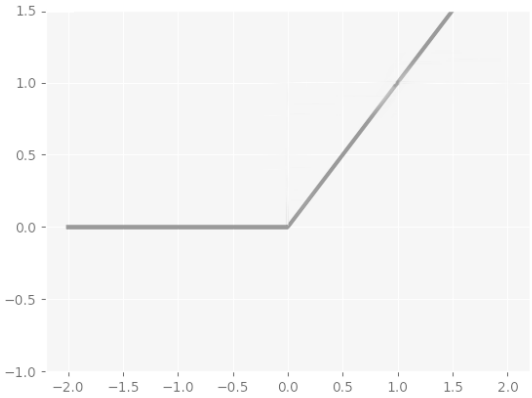
Fonte: Adaptado de:(FACURE, 2017)

A função tangente hiperbólica retorna valores entre -1 e 1. As entradas negativas são mapeadas para valores negativos, as entradas positivas são mapeadas para valores

positivos e as entradas próximas de zero irão tender à zero (NIELSEN, 2015).

$$Y = max(0, x) \tag{19}$$

Figura 12 – Função de ativação - Relu

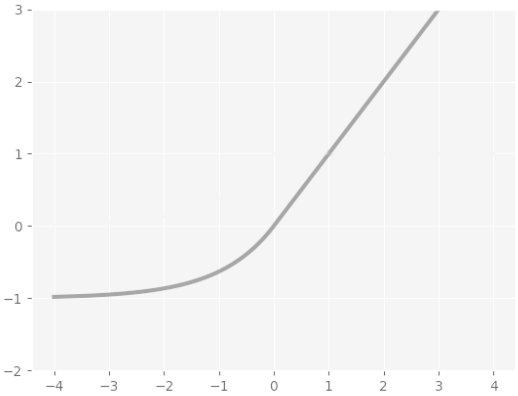


Fonte: Adaptado de:(FACURE, 2017)

A função Relu (Unidades Lineares Retificadas) retorna o valor zero ou maior do que zero (NIELSEN, 2015).

$$Y = \begin{cases} x, & \text{x} \geq 0 \\ \alpha(e^x - 1), & \text{x} < 0 \end{cases} \tag{20}$$

Figura 13 – Função de ativação - Elu



Fonte: Adaptado de:(FACURE, 2017)

α - Constante inteira positiva

Os resultados da função Elu (Unidade Linear Exponencial) alcançam de -α até ∞. Ao contrário da função Relu, a Elu pode retornar números negativos (NIELSEN, 2015).

Foram feitos testes na rede para verificar qual das funções tornaria a solução mais eficiente.

Para a camada de saída é utilizada a função *SoftMax* representada na Equação 21 (NIELSEN, 2015).

$$Y = \frac{e_K^x}{\sum_{k=1}^{K} e_k^x} \tag{21}$$

A função *SoftMax* retorna a probabilidade de qual será o rótulo de saída. Como a rede possui três saídas, o resultado da *SoftMax* será a probabilidade de cada uma das saídas. Ela transforma as saída para valores entre 0 e 1 e divide pela soma de todas as saídas (NIELSEN, 2015).

4.4 Avaliação e Validação

A validação da rede foi feita por meio da validação cruzada com o método *hold-dout*. Os dados foram divididos em dois conjuntos, o conjunto para teste e o conjunto para treinamento. Para definição da quantidade exata para treinamento e testes a rede foi trenada inicialmente com 10% de dados para teste e 90% para treinamento, depois foi sendo aumentado o número de teste, porém a divisão de dados que obteve o melhor resultado foi de 20% dos dado existentes para teste e o conjunto para treinamento com os 80% restantes.

Para avaliação dos resultados da rede foi utilizado o método de acurácia. A função utilizada para calcular a acurácia da rede foi a *accuracy*, que calcula a frequência em que as previsões combinam com as respostas reais a partir da Equação 22 (LEAL, 2017).

$$\text{acurácia} = \frac{\text{Verdadeiros Positivos} + \text{Verdadeiros Negativos}}{\text{Total}} \tag{22}$$

Verdadeiros positivos são os valores que resultam conforme era esperado pela aplicação(SOUZA, 2019).

Verdadeiros negativos são quando os testes efetuados resultam em uma falha na aplicação(SOUZA, 2019).

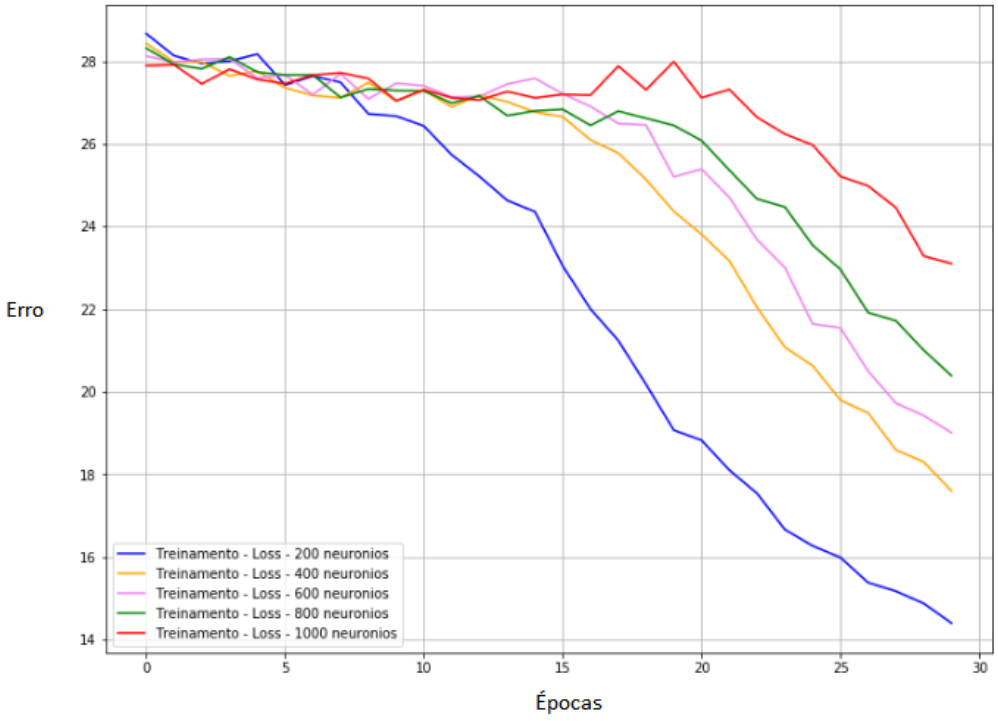
Outro método de avaliação utilizado foi a matriz de confusão. A matriz de confu-

são apresenta a quantidade em que cada resultado real foi corretamente ou incorretamente previsto (SOUZA, 2019). Na diagonal principal da matriz estão os resultados que foram classificados corretamente, nos locais externos à diagonal principal estão os resultado que a rede previu incorretamente.

5 RESULTADOS E DISCUSSÃO

Para encontrar a quantidade mais adequada de camadas ocultas e neurônios, inicialmente foram feitos treinamentos com apenas uma camada oculta de 200 elementos. Após cada teste, foram aumentados 200 neurônios nesta mesma camada em cada treinamento até a quantidade de 1000 neurônios. Os dados utilizados possuem uma grande variação de tipos diferentes, por este motivo a rede de apenas uma camada não trouxe os resultados esperados. Foram feitos testes com arquiteturas de 1 até 4 camadas, e variação de neurônios de 50 até 1000 neurônios. Serão apresentados os resultados de 1 camada oculta e de 4 camadas ocultas (extremos treinados). No gráfico da Figura 14, está representado o resultado da evolução do erro feitos com a rede de apenas uma camada oculta possuindo 5 tipos de arquiteturas.

Figura 14 – Gráfico do erro - 1 camada oculta



Fonte: Autora (2019)

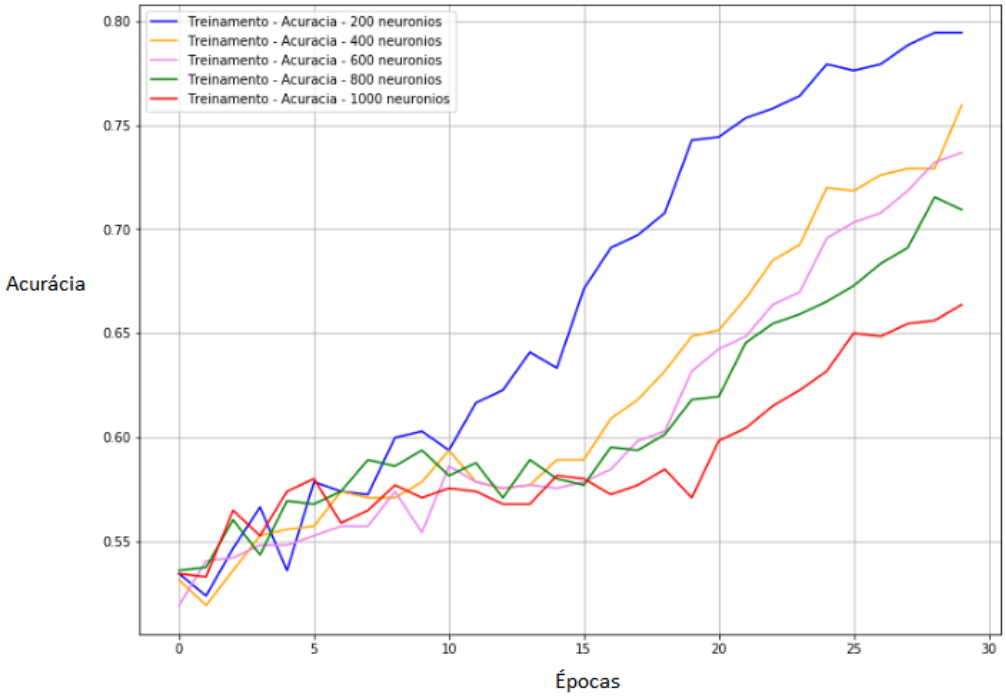
Cada cor de linha existente no gráfico da Figura 14 representa um tipo de arquitetura diferente.

- A de cor azul é composta por 1 camada oculta com 200 neurônios.
- A de cor amarelo é composta por 1 camada oculta com 400 neurônios.

- A de cor rosa é composta por 1 camada oculta com 600 neurônios.
- A de cor verde é composta por 1 camada oculta com 800 neurônios.
- A de cor vermelha é composta por 1 camada oculta com 1000 neurônios.

No gráfico da Figura 15, estão representados os valores de acurácia resultante das mesmas arquiteturas utilizadas na Figura 14.

Figura 15 – Gráfico da acurácia - 1 camada oculta



Fonte: Autora (2019)

Os treinamentos apresentados nos gráficos das Figuras 15 e Figura 14 foram interrompidos na época 30 pelo fato de que, após essa época a rede interrompeu sua aprendizagem e passou a decorar os dados, não sendo capaz de classificar corretamente os dados de teste. Também pode-se observar que a arquitetura de 200 neurônios gerou um erro baixo e uma acurácia relativamente alta. Para verificar se a diminuição na quantidade de neurônios poderia gerar em um aumento da acurácia e diminuição do erro, foram feitos testes com 50 e 100 neurônios nesta mesma camada, e os resultados foram que: o erro aumentou e a acurácia diminuiu.

Mais camadas ocultas foram adicionadas no intuito de melhoramento dos resultados. No gráfico da Figura 16, estão apresentados os treinamentos feitos com a utilização de 4 camadas ocultas. Além da utilização da métrica de adição de 200 neurônios a cada

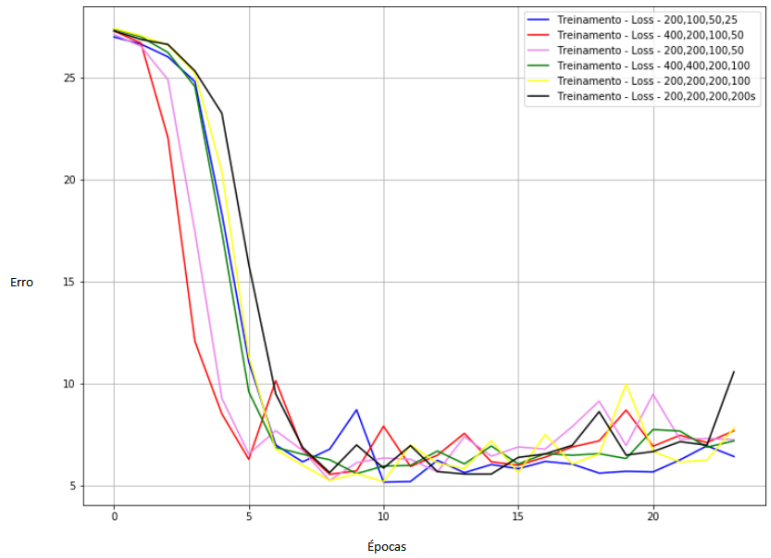
treino, foi utilizado que, em cada camada seguinte diminuiria pela metade a quantidade de neurônios utilizados.

- A de cor azul é composta por 4 camadas ocultas: a primeira com 200 neurônios, a segunda com 100 neurônios, a terceira com 50 neurônios e a quarta com 25 neurônios.
- A de cor vermelho é composta por 4 camadas ocultas: a primeira com 400 neurônios, a segunda com 200 neurônios, a terceira com 100 neurônios e a quarta com 50 neurônios.
- A de cor rosa é composta por 4 camadas ocultas: a primeira com 200 neurônios, a segunda com 200 neurônios, a terceira com 100 neurônios e a quarta com 50 neurônios.
- A de cor verde é composta por 4 camadas ocultas: a primeira com 400 neurônios, a segunda com 400 neurônios, a terceira com 200 neurônios e a quarta com 100 neurônios.
- A de cor amarela é composta por 4 camadas ocultas: a primeira com 200 neurônios, a segunda com 200 neurônios, a terceira com 200 neurônios e a quarta com 100 neurônios.
- A de cor preta é composta por 4 camadas ocultas: a primeira com 200 neurônios, a segunda com 200 neurônios, a terceira com 200 neurônios e a quarta com 200 neurônios.

Como observado no gráfico da Figura 16, as arquiteturas que resultaram em um menor erro foram as representadas pelas cores azul e amarela.

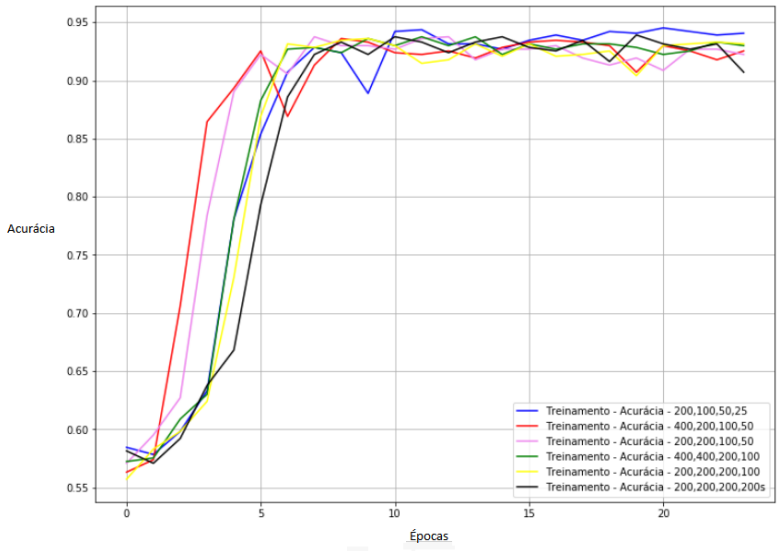
No gráfico da Figura 17, estão representados os valores de acurácia resultante das mesmas arquiteturas utilizadas na Figura 16.

Figura 16 – Gráfico do erro - 4 camadas ocultas



Fonte: Autora (2019)

Figura 17 – Gráfico da acurácia - 4 camadas ocultas

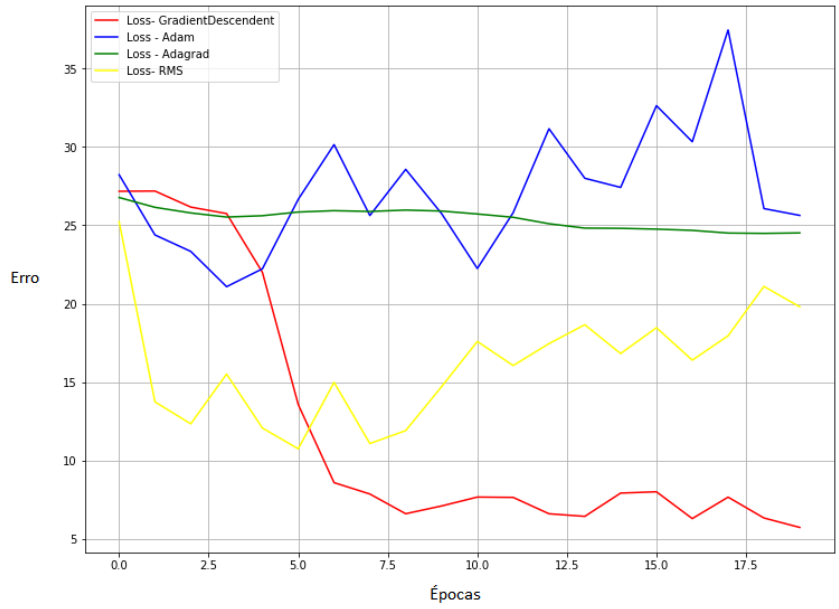


Fonte: Autora (2019)

Como observado no gráfico da Figura 17, a arquitetura representada pela cor azul se sobressaiu comparada com as outras , atingindo uma acurácia de 94,5%, resultando em uma rede eficiente. Buscando um melhoramento da acurácia da rede foram testadas a substituição do otimizador gradiente descendente pelos otimizadores *AdamOptimizer*, *RMSPropOptimizer* e *AdagradOptimizer*. Os resultados do erro com os diferentes otimi-

zadores estão representados no gráfico da Figura 18.

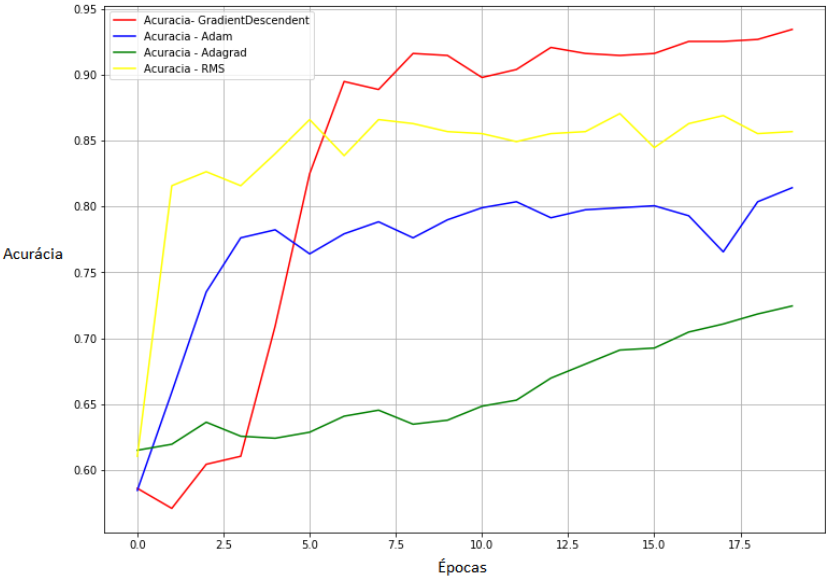
Figura 18 – Gráfico do erro com diferentes otimizadores



Fonte: Autora (2019)

Os resultados da acurácia com os diferentes otimizadores estão representados no gráfico da Figura 19.

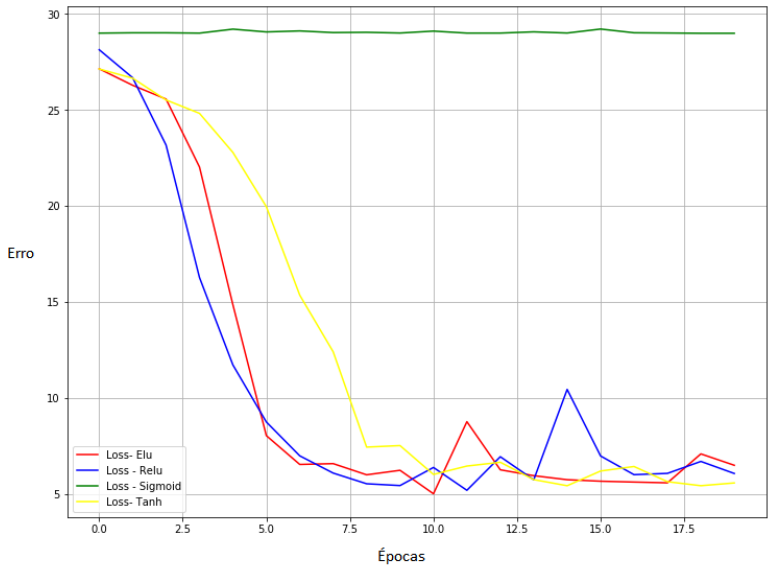
Figura 19 – Gráfico da acurácia com diferentes otimizadores



Fonte: Autora (2019)

Como pode-se observar o otimizador com melhor desempenho foi o *GradientDescendentOptimizer*, representado pela linha vermelha do gráfico ilustrado na Figura 18. Para verificar qual a melhor função de ativação a ser utilizada, foram feitos testes da arquitetura escolhida com quatro funções de ativação: *Elu*, *Relu*, *Sigmoid* e *Tanh*. Os resultados obtidos estão representados no gráfico da Figura 20.

Figura 20 – Gráfico do erro com diferentes funções de ativação



Fonte: Autora (2019)

Como pode-se observar a função de ativação que obteve o melhor resultado foi a função Elu, que gerou o erro mais baixo. A partir da Época 10, a rede começou a decorar os dados.

Após os testes com a substituição de otimizadores e funções de ativação a configuração do melhor resultado obtido foi de 4 camadas ocultas, a primeira com 200 neurônios, a segunda com 100, a terceira com 50 e a quarta com 25. Utilizando a função de ativação Elu, otimizador *GradientDescendentOptimizer*, taxa de aprendizagem de 0.001 e 2100 steps. A acurácia foi de 94,5%. Para melhor visualização da taxa de acerto foi feita a matriz de confusão dos resultados representada na Tabela 12.

Tabela 12 – Matriz de confusão

		Previsto		
		Leve	Médio	Pesado
Valor Real	Leve	48	8	0
	Médio	8	262	17
	Pesado	0	3	311

Na matriz de confusão, as colunas são a representação da quantidade de valores previstos pela rede em cada uma das categorias e as linhas a representação dos valores reais. Os valores localizados na diagonal principal da matriz são os valores do número acerto gerado pela rede e nos demais espaços estão o número de vezes que a rede errou. Os maiores erros são em relação aos valores médios, devido ao fato da separação entre os pesos leve/médio/pesado, ser feita arbitrariamente pelo responsável da coleta dos dados.

Como pode se observar na Tabela 12, dos resultados obtidos através do teste, das três saída possíveis a que ocorreu mais erros na sua previsão foi o peso médio, quando o valor real da saída era médio, a rede previu em 8 casos que a saída era leve e em 17 que a saída era pesado, mas em 262 saídas ocorreu acerto.

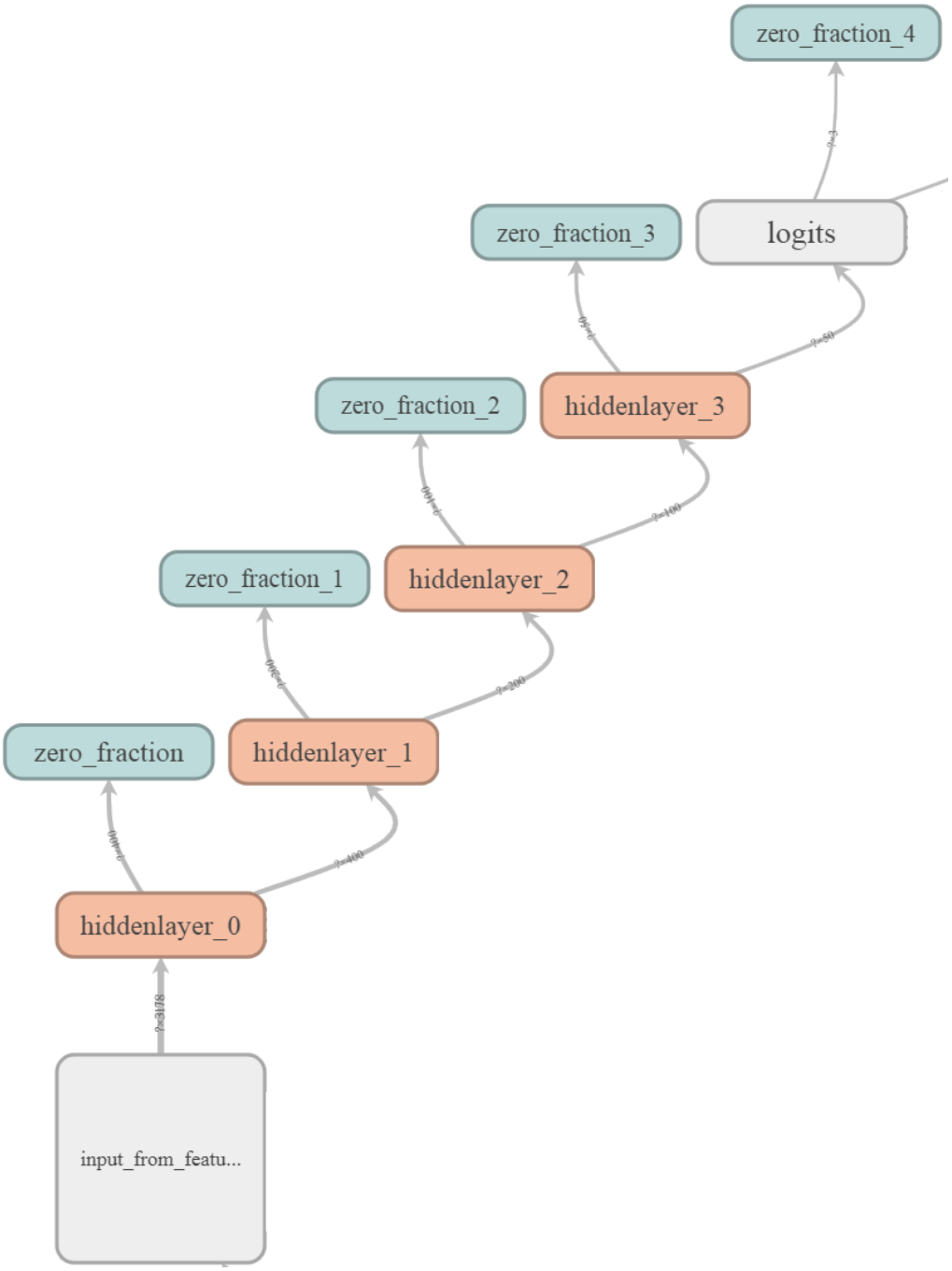
Quando o resultado da saída era leve, a rede não previu em nenhum caso que sua saída era pesada, apenas em 8 casos encontrou que era média.

Quando o resultado da saída era pesado, em nenhum caso a rede previu que a rede era leve, apenas em 3 casos encontrou que era média.

Mesmo com alguns erros na sua classificação, a rede não errou nos extremos, nunca considerou uma saída que era leve sendo como pesada, e nem uma saída pesada como se fosse leve, sendo um ponto positivo da rede.

Através da ferramenta *Tensorboard* foi gerada uma ilustração do modelo final para teste, constituído por 4 camadas ocultas, uma camada de saída e 1 bias por camada, como ilustrado na Figura 21.

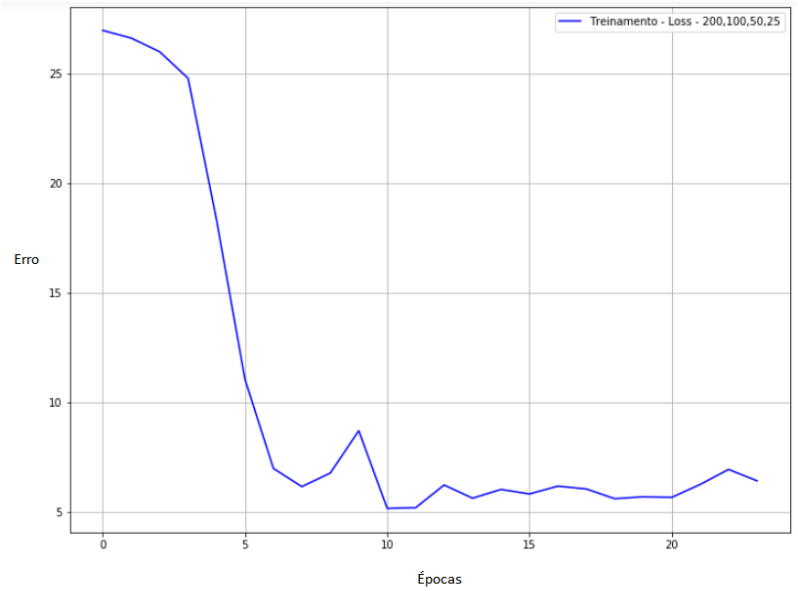
Figura 21 – Modelo de teste



Fonte: Autora (2019)

Para verificação da evolução do erro na rede da arquitetura escolhida foi gerado o gráfico ilustrado na Figura 22. Quando a linha está na posição 10 no eixo das épocas, a rede alcança seu máximo de aprendizagem, após, a rede passou a decorar os dados.

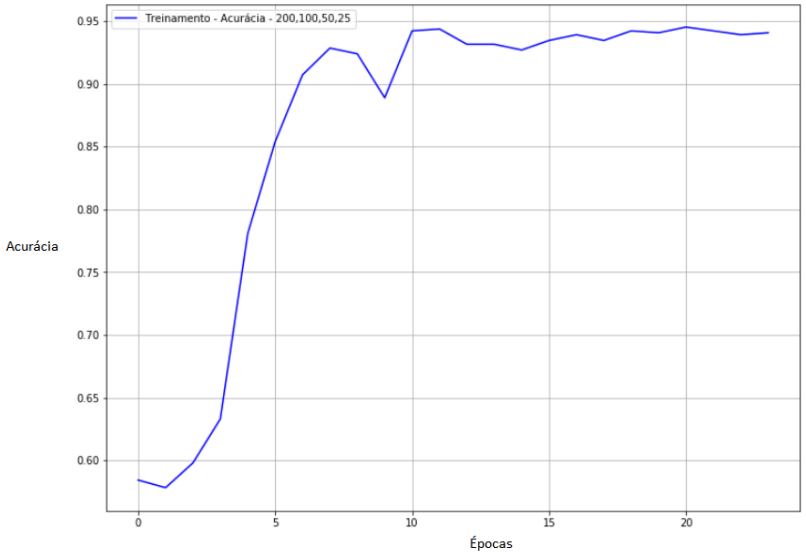
Figura 22 – Gráfico do Erro



Fonte: Autora (2019)

Para ilustrar a evolução da acurácia de teste do modelo escolhido, foi gerado o gráfico da acurácia, ilustrado na Figura 23.

Figura 23 – Gráfico da acurácia da rede



Fonte: Autora (2019)

6 CONSIDERAÇÕES FINAIS

Dependendo de como ocorre, os fatores do transporte do gado para abate influenciam na qualidade da carne produzida pelo frigorífico e enviada para o consumidor final. A quantidade de contusões associadas ao processo de transporte representa perdas financeiras significativas aos produtores e ao frigorífico, que nem sempre consegue repassá-las ao produtor ou à empresa de transporte.

Um animal contundido durante o transporte, além das questões importantes referentes à ética no trato com animais de produção, pode ter um percentual significativo de sua carcaça não aproveitada ou destinada a finalidades menos nobres que o consumidor humano. A mudança no processo condições de embarque, condições de transporte e espera pós-desembarque podem ter alterações benéficas para os animais e para os demais atores envolvidos. Adicionalmente, o produto dos frigoríficos pode ter maior valor de mercado para os consumidores conscientes em relação ao bem-estar animal, além de produzir uma carne mais clara, menos rígida e com pH baixo.

Este trabalho fez uso de uma rede neural *perceptron* multicamadas densa para mapeamento das variáveis do processo de transporte, com dados coletados durante o período de um ano, em peso esperado de contusões decorrentes do processo nos animais. Os resultados mostram uma acurácia de 94,5% na predição da gravidade de contusões a partir da entrada das variáveis. A análise de sensibilidade da rede não foi realizada neste trabalho, mas pretende-se que ela seja usada para o que o produtor possa inferir como alterações no seu processo de embarque e deslocamento ao frigorífico podem diminuir ou agravar as perdas decorrentes do não aproveitamento da carne. Da mesma forma, os frigoríficos podem fazer uso da rede para calcular o quanto precisam pagar para o produtor, em decorrência da perda esperada de seus animais, dadas as condições de embarque e as condições de deslocamento.

O motivo pelo qual a rede não atingiu uma acurácia mais próxima a 100% pode ser tanto pela limitação dos dados utilizados quanto por fatores ausentes dos dados. As condições meteorológicas do dia do transporte podem ter influência no fator estresse, especialmente se estiver muito quente e as paradas não forem feitas em local adequado. Como trabalhos futuros estão a inserção de dados climáticos como variáveis do modelo e a construção de métodos de coleta de dados mais confiáveis e automatizados, que permitam a escala da aquisição de dados para construção de melhores modelos de inferência.

6.1 Trabalhos Futuros

Como trabalhos futuros, propõe-se efetuar a extração das regras da rede neural construída com o objetivo de encontrar quais são as variáveis de transporte com mais dependências em relação ao peso da contusão.

E também a criação de um algoritmo com interface gráfica que efetue a leitura do modelo criado, receba os dados inseridos pelos funcionários do frigorífico e possa ser de fácil utilização para melhorar as condições do transporte resultando em uma melhora do bem-estar animal e da qualidade do produto final.

REFERÊNCIAS

ATKINSON, S. **Farm animal transport, welfare and meat quality**. [S.l.: s.n.], 2000.

BARRETO, J. M. Introdução as redes neurais artificiais. **V Escola Regional de Informática. Sociedade Brasileira de Computação, Regional Sul, Santa Maria, Florianópolis, Maringá**, p. 5–10, 2002.

BLOCK, N. C. da S. *et al.* Processo de produção da carne bovina: dos animais ao produto final. **X Encontro de engenharia de produção agroindustrial**, 2016.

CAMARGO, M. da S.; FERREIRA, A. P. L.; PEREZ, N. B. Avaliação do bem-estar animal no transporte de gado de corte. **Congresso Brasileiro de Agroinformática - SBIAGro, Campinas - São Paulo**, 2017.

CAMARGO, M. da S.; FERREIRA, A. P. L.; PEREZ, N. B. Identificação de variáveis de relevância no Índice de contusões associadas ao transporte de gado de corte. 2018.

FACURE, M. **Funções de ativação - Entendendo a importância da ativação correta nas redes neurais**. 2017. Acesso em: 17 maio 2019. Disponível em: https://matheusfacure.github.io/2017/07/12/activ-func/.

FARIA, G.; ROMERO, R. F. Explorando o potencial de algoritmos de aprendizado com reforço em robôs móveis. In: **Proceedings of the IV Brazilian Conference on Neural Networks**. [S.l.: s.n.], 1999. p. 237–242.

FILHO, A. L. Produção De Carne Bovina No Brasil. **II Simpósio sobre desafios e novas tecnologias na bovinocultura de corte**, n. JANUARY 2006, 2006.

GOOGLE BRAIN. **Tensorflow**. 2019. Acesso em: 29 maio 2019. Disponível em: https://www.tensorflow.org/.

GURGEL, S. T. A. *et al.* Análise de técnicas de implementação paralela para treinamento de redes neurais em gpu. Universidade Federal da Paraíba, 2014.

HAYKIN, S. **Redes Neurais - 2ed.** BOOKMAN COMPANHIA ED, 2001. ISBN 9788573077186. Disponível em: https://books.google.com.br/books?id=lBp0X5qfyjUC.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. Indicadores ibge - estatística da produção pecuária. **Instituto Brasileiro de Geografia e estatística**, 2018.

JANGID, M.; SRIVASTAVA, S. Handwritten devanagari character recognition using layer-wise training of deep convolutional neural networks and adaptive gradient methods. **Journal of Imaging**, Multidisciplinary Digital Publishing Institute, v. 4, n. 2, p. 41, 2018.

KAWAGUCHI, K. **The McCulloch-Pitts model of neuron**. 2000. Acesso em: 04 outubro 2018. Disponível em: http://wwwold.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node12.html.

KOUSHIK, J. Understanding convolutional neural networks. **arXiv preprint arXiv:1605.09081**, 2016.

LAMA, M. de la *et al.* Atitudes dos retalhistas de carne ao bem-estar animal em espanha. **Meat Science**, v. 95, n. 3, p. 569–575, 2013.

LEAL, R. dos S. **Métricas Comuns em Machine Learning: como analisar a qualidade de chat bots inteligentes—métricas (3 de 4)**. 2017. Acesso em: 17 maio 2019. Disponível em: https://medium.com/as-m\%C3\%A1quinas-que-pensam/m\%C3\%A9tricas-comuns-em-machine-learning-como-analisar-a-qualidade-de-chat-bots-inteligentes-m\%C3\%A9tricas-1ba580d7cc96.

LUDERMIR, T. B.; OLIVEIRA, W. R. de. Weightless neural models. **Computer standards & interfaces**, North-Holland, v. 16, n. 3, p. 253–263, 1994.

LUDTKE, C. B. *et al.* Abate humanitario de bovinos. **WSPA BRASIL - SOCIEDADE MUNDIAL DE PROTEÇÃO ANIMAL, Rio de Janeiro**, p. 148, 2012.

MACQUEEN, J. Some methods for classification and analysis of multivariate observations. In: **Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics**. Berkeley, Calif.: University of California Press, 1967. p. 281–297. Disponível em: https://projecteuclid.org/euclid.bsmsp/1200512992.

MAGGI, B. *et al.* Revista de política agrícola. **Secretaria de politica agrícola do ministério da agricultura, pecuáia e abastecimento**, 2017.

MARQUES, R. L.; DUTRA, I. Redes bayesianas: o que são, para que servem, algoritmos e exemplos de aplicações. **Coppe Sistemas–Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil**, 2002.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

MENDONÇA, F. S. *et al.* Pre-slaughtering factors related to bruises on cattle carcasses. **Animal Production Science**, CSIRO, v. 58, n. 2, p. 385–392, 2018.

MINSKY, M.; PAPERT, S. A. **Perceptrons: An introduction to computational geometry**. [S.l.]: MIT press, 2017.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. **Sistemas inteligentes-Fundamentos e aplicações**, v. 1, n. 1, p. 32, 2003.

MONARD, M. C.; BARANAUSKAS, J. A. Indução de regras e árvores de decisão. **Sistemas Inteligentes. Rezende, SO Editora Manole Ltda**, p. 115–140, 2003.

MUNAKATA, T. **Fundamentals of the new artificial intelligence**. [S.l.]: Springer, 1998. v. 2.

NIELSEN, M. A. **Neural networks and deep learning**. [S.l.]: Determination press San Francisco, CA, USA:, 2015. v. 25.

OSÓRIO, F. S.; BITTENCOURT, J. R.; OSóRIO, F. S. Sistemas inteligentes baseados em redes neurais artificiais aplicados ao processamento de imagens. In: **I Workshop de inteligência artificial**. [S.l.: s.n.], 2000.

ROMERO, M. *et al.* Risk factors influencing bruising and high muscle ph in colombian cattle carcasses due to transport and pre-slaughter operations. **Meat science**, Elsevier, v. 95, n. 2, p. 256–263, 2013.

RUMELHART, D. E. *et al.* **Parallel distributed processing**. [S.l.]: MIT press Cambridge, MA, 1987. v. 1.

SAKAMOTO, Y.; ISHIGURO, M.; KITAGAWA, G. Akaike information criterion statistics. **Dordrecht, The Netherlands: D. Reidel**, Taylor & Francis, v. 81, 1986.

SOUZA, E. G. de. **Entendendo o que é Matriz de Confusão com Python**. 2019. Acesso em: 19 maio 2019. Disponível em: https://medium.com/data-hackers/entendendo-o-que-\%C3\%A9-matriz-de-confus\%C3\%A3o-com-python-114e683ec509.

STUTZ, D. Understanding convolutional neural networks. **InSeminar Report, Fakult'at fur Mathematik, Informatik und Naturwissenschaften Lehr-und Forschungsgebiet Informatik VIII Computer Vision**, 2014.

VARGAS, A. C. G.; PAES, A.; VASCONCELOS, C. N. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. In: **Proceedings of the XXIX Conference on Graphics, Patterns and Images**. [S.l.: s.n.], 2016. p. 1–4.

APÊNDICE A — CÓDIGO FONTE EM PYTHON: REDE NEURAL PERCEPTRON MULTICAMADAS

```
@author: flavia morales antunes

import pandas as pd
import tensorflow as tf
import numpy as np
import openpyxl as op
import matplotlib.pyplot as plt
%matplotlib inline
from datetime import datetime
from datetime import date


#Leitura da matriz em excel com os dados
planilha = pd.read_excel('dadosteste.xlsx')
planilha.head()

# Conversao das saidas (medio, leve e pesado) por rotulos (1,2 e 3)
def converter_classe(rotulo):
    if rotulo == 'leve':
        return 0
    elif rotulo == 'medio':
        return 1
    else:
        return 2

planilha.Peso_contusao = planilha.Peso_contusao.apply(converter_classe)

# Extracao da coluna "Peso_contusao" e salvando-a em y, e as colunas
    restantes em x.
x = planilha.drop('Peso_contusao', axis=1)
y = planilha.Peso_contusao

#Separac o dos dados para treinamento e teste 20% para teste e 80%
    para treinamento
from sklearn.model_selection import train_test_split
x_treinamento, x_teste, y_treinamento, y_teste = train_test_split(x,y,
    test_size=0.2)
```

```
35
37 # Mapeamento de colunas categoricas para tabela hash
   produtor = tf.feature_column.categorical_column_with_hash_bucket(key =
       'Produtor', hash_bucket_size = 1056)
39 municipio = tf.feature_column.categorical_column_with_hash_bucket(key =
       'municipio', hash_bucket_size = 129)
   mangueira = tf.feature_column.categorical_column_with_hash_bucket(key =
       'Mangueira', hash_bucket_size = 757)
41 tipo_veiculo = tf.feature_column.categorical_column_with_hash_bucket(
       key = 'tipo_veiculo', hash_bucket_size = 12)
   cond_mangueira = tf.feature_column.categorical_column_with_hash_bucket(
       key = 'condicoes_mangueira', hash_bucket_size = 4)
43 cond_tempo = tf.feature_column.categorical_column_with_hash_bucket(key
       = 'condicoes_tempo', hash_bucket_size = 4)
   cond_estrada = tf.feature_column.categorical_column_with_hash_bucket(
       key = 'condicoes_estrada', hash_bucket_size = 5)
45 manejo_fazenda = tf.feature_column.categorical_column_with_hash_bucket(
       key = 'manejo_fazenda', hash_bucket_size = 6)
   caminhao_atolou = tf.feature_column.categorical_column_with_hash_bucket
       (key = 'caminhao_atolou', hash_bucket_size = 7)
47 motorista = tf.feature_column.categorical_column_with_hash_bucket(key =
       'Motorista', hash_bucket_size = 183)
   produtor2 = tf.feature_column.categorical_column_with_hash_bucket(key =
       'Produtor2', hash_bucket_size = 1010)
49 sexo = tf.feature_column.categorical_column_with_hash_bucket(key = '
       sexo', hash_bucket_size = 4)
   unico_embarcador = tf.feature_column.
       categorical_column_with_hash_bucket(key = 'unico_embarcador',
       hash_bucket_size = len(planilha['unico_embarcador'].unique()))
51 unico_embarcador2 = tf.feature_column.
       categorical_column_with_hash_bucket(key = 'unico_embarcador2',
       hash_bucket_size = len(planilha['unico_embarcador2'].unique()))

53 # Gerando vetores tridmensionais para cada coluna categorica
   embedded_produtor = tf.feature_column.embedding_column(produtor,
       dimension = len(planilha['Produtor'].unique()))
55 embedded_municipio = tf.feature_column.embedding_column(municipio,
       dimension = len(planilha['municipio'].unique()))
   embedded_mangueira = tf.feature_column.embedding_column(mangueira,
       dimension = len(planilha['Mangueira'].unique()))
```

```
embedded_tipo_veiculo = tf.feature_column.embedding_column(tipo_veiculo, dimension = len(planilha['tipo_veiculo'].unique()))
embedded_cond_mangueira = tf.feature_column.embedding_column(cond_mangueira, dimension = len(planilha['condicoes_mangueira'].unique()))
embedded_cond_tempo = tf.feature_column.embedding_column(cond_tempo, dimension = len(planilha['condicoes_tempo'].unique()))
embedded_cond_estrada = tf.feature_column.embedding_column(cond_estrada, dimension = len(planilha['condicoes_estrada'].unique()))
embedded_manejo_fazenda = tf.feature_column.embedding_column(manejo_fazenda, dimension = len(planilha['manejo_fazenda'].unique()))
embedded_caminhao_atolou = tf.feature_column.embedding_column(caminhao_atolou, dimension = len(planilha['caminhao_atolou'].unique()))
embedded_motorista = tf.feature_column.embedding_column(motorista, dimension = len(planilha['Motorista'].unique()))
embedded_produtor2 = tf.feature_column.embedding_column(produtor2, dimension = len(planilha['Produtor2'].unique()))
embedded_sexo = tf.feature_column.embedding_column(sexo, dimension = len(planilha['sexo'].unique()))
embedded_unico_embarcador = tf.feature_column.embedding_column(unico_embarcador, dimension = len(planilha['unico_embarcador'].unique()))
embedded_unico_embarcador2 = tf.feature_column.embedding_column(unico_embarcador2, dimension = len(planilha['unico_embarcador2'].unique()))

# Conversao de todas colunas numericas para float
planilha['Boi'] = planilha['Boi'].astype(float)
planilha['Touro'] = planilha['Touro'].astype(float)
planilha['Vaca'] = planilha['Vaca'].astype(float)
planilha['bufalo_f'] = planilha['bufalo_f'].astype(float)
planilha['bufalo_m'] = planilha['bufalo_m'].astype(float)
planilha['km_chao'] = planilha['km_chao'].astype(float)
planilha['km_final'] = planilha['km_final'].astype(float)
planilha['pessoas_carregamento'] = planilha['pessoas_carregamento'].astype(float)
planilha['tempo_viagem'] = planilha['tempo_viagem'].astype(float)
planilha['peso_liquido'] = planilha['peso_liquido'].astype(float)
```

```
planilha['peso_med_carc_fria'] = planilha['peso_med_carc_fria'].astype(
    float)
planilha['quantidade'] = planilha['quantidade'].astype(float)
planilha['NumCond'] = planilha['NumCond'].astype(float)
planilha['numporkm'] = planilha['numporkm'].astype(float)

# Padronizacao das colunas numericas para a mesma escala
def padroniza_boi(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.Boi.mean())), tf.constant(planilha.Boi.std())
        )

def padroniza_vaca(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.Vaca.mean())), tf.constant(planilha.Vaca.std
        ()))

def padroniza_touro(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.Touro.mean())), tf.constant(planilha.Touro.
        std()))

def padroniza_bufalo_m(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.bufalo_m.mean())), tf.constant(planilha.
        bufalo_m.std()))

def padroniza_bufalo_f(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.bufalo_f.mean())), tf.constant(planilha.
        bufalo_f.std()))

def padroniza_km_chao(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.km_chao.mean())), tf.constant(planilha.
        km_chao.std()))
def padroniza_km_final(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.km_final.mean())), tf.constant(planilha.
        km_final.std()))

```

```
def padroniza_pessoas_carregamento(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.pessoas_carregamento.mean())), tf.constant(
        planilha.pessoas_carregamento.std()))

def padroniza_tempo_viagem(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.tempo_viagem.mean())), tf.constant(planilha.
        tempo_viagem.std()))

def padroniza_peso_liquido(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.peso_liquido.mean())), tf.constant(planilha.
        peso_liquido.std()))

def padroniza_quantidade(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.quantidade.mean())), tf.constant(planilha.
        quantidade.std()))

def padroniza_peso_med(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.peso_med_carc_fria.mean())), tf.constant(
        planilha.peso_med_carc_fria.std()))
def padroniza_numcond(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.NumCond.mean())), tf.constant(planilha.
        NumCond.std()))

def padroniza_numporkm(valor):
    return tf.divide(tf.subtract(tf.cast(valor, tf.float64), tf.
        constant(planilha.numporkm.mean())), tf.constant(planilha.
        numporkm.std()))


# Mapeamento de colunas numericas para tabela hash
boi = tf.feature_column.numeric_column(key = 'Boi', normalizer_fn =
    padroniza_boi)
vaca = tf.feature_column.numeric_column(key = 'Vaca', normalizer_fn =
    padroniza_vaca)
```

```
touro = tf.feature_column.numeric_column(key = 'Touro', normalizer_fn =
    padroniza_touro)
bufalo_m = tf.feature_column.numeric_column(key = 'bufalo_m',
    normalizer_fn = padroniza_bufalo_m)
bufalo_f = tf.feature_column.numeric_column(key = 'bufalo_f',
    normalizer_fn = padroniza_bufalo_f)
km_chao = tf.feature_column.numeric_column(key = 'km_chao',
    normalizer_fn = padroniza_km_chao)
km_final = tf.feature_column.numeric_column(key = 'km_final',
    normalizer_fn = padroniza_km_final)
pessoas_carregamento = tf.feature_column.numeric_column(key = '
    pessoas_carregamento', normalizer_fn =
    padroniza_pessoas_carregamento)
tempo_viagem = tf.feature_column.numeric_column(key = 'tempo_viagem',
    normalizer_fn = padroniza_tempo_viagem)
peso_liquido = tf.feature_column.numeric_column(key = 'peso_liquido',
    normalizer_fn = padroniza_peso_liquido)
quantidade = tf.feature_column.numeric_column(key = 'quantidade',
    normalizer_fn = padroniza_quantidade)
peso_med_carc_fria = tf.feature_column.numeric_column(key = '
    peso_med_carc_fria', normalizer_fn = padroniza_peso_med)
numcond = tf.feature_column.numeric_column(key = 'NumCond',
    normalizer_fn = padroniza_numcond)
numporkm = tf.feature_column.numeric_column(key = 'numporkm',
    normalizer_fn = padroniza_numporkm)

# Gerando um vetor com todas colunas j  padronizadas para insercao na
    rede
colunas = [embedded_produtor, embedded_municipio, embedded_mangueira,
    embedded_tipo_veiculo,embedded_cond_mangueira,
            embedded_cond_tempo, embedded_cond_estrada,
                embedded_manejo_fazenda,
            embedded_caminhao_atolou, embedded_motorista,
                embedded_produtor2, embedded_sexo,
            boi,vaca,touro,bufalo_m,bufalo_f,km_chao,km_final,
                pessoas_carregamento,
            embedded_unico_embarcador, tempo_viagem,
                embedded_unico_embarcador2, peso_liquido,
            quantidade, peso_med_carc_fria, numcond,numporkm]


```

```
# Criacao da funcao de treinamento
funcao_treinamento = tf.estimator.inputs.pandas_input_fn(x =
    x_treinamento, y = y_treinamento,batch_size = 32, num_epochs = None
    , shuffle = True)

# Criacao da funcao de teste
funcao_teste = tf.estimator.inputs.pandas_input_fn(x = x_teste, y =
    y_teste, batch_size = 32,num_epochs = 1, shuffle = False)

# Criacao do classificador utilizando o modelo DNNClassifier
#Hidden_units = numero de camadas
#Feature_columns = valores de entrada
#Model_dir = local para salvamento do modelo construido
#n_classes = quantidade de possiveis saidas
#activation_fn = funcao de ativacao a ser utilizada
#Optimizer = otimizador utilizado
#Learning_rate = taxa de aprendizado
classificador = tf.estimator.DNNClassifier(hidden_units =
    [400,200,100,50], feature_columns=colunas,model_dir='train/linreg',
     n_classes=3, activation_fn=tf.nn.elu, optimizer = tf.train.
    GradientDescentOptimizer(learning_rate=0.001))
resultados = []

#Treino da rede
for i in range(20):
    treino = classificador.train(input_fn = funcao_treinamento, steps =
        200)
    resultados.append(classificador.evaluate(input_fn=funcao_teste))

#Salvando resultados em um arquivo de texto
now = datetime.now()
teste = str(now).split()
arq_name = teste[0]+'_'+teste[1][0:2]+'-'+teste[1][3:5]
arq = open(arq_name+'.txt','w')
arq.write('hidden_units = 400,200,100,50, activation_fn=tf.nn.elu,
    optimizer = tf.train.GradientDescendentOptimizer-lr = 0.001\n')
for i in range(len(resultados)):
    arq.write(str(resultados[i])+"\n")
arq.close()

#Funcao de predicao
```

```
185 funcao_predict = tf.estimator.inputs.pandas_input_fn(x = x_teste,
        shuffle = False)
    pred = list(classificador.predict(input_fn=funcao_predict))
187 predi = []
    for i in range (len(pred)):
189     predi.append(pred[i]['logits'].argmax()) #adicionando previs es a
            um vetor

191 #Plotando acuracia da rede
    plot_resultado = []
193 for i in range(len(resultados)):
        plot_resultado.append(resultados[i]['accuracy'])
195 plt.rcParams['figure.figsize'] = (12,9)
    plt.plot(plot_resultado, '-', color = 'red',label='Treinamento -
        Acuracia')
197 plt.legend()
    plt.grid(True)
199 plt.show()

201 #Plotando resultados previstos e resultados reais
    plt.rcParams['figure.figsize'] = (12,9)
203 plt.plot(list(predi), '.', color = 'red',label='Resultado Previsto')
    plt.plot(list(y_teste),'', color = 'green',label='Resultado Correto')
205 plt.legend()
    plt.grid(True)
207 plt.show()

209 #Calculando a correla  o da sa da prevista com a sa da real
    y_teste_array = np.asarray(y_teste)
211 pred_array = np.asarray(predi)
    y_pd = pd.Series(y_teste_array)
213 predi_pd = pd.Series(pred_array)
    y_pd.corr(predi_pd)
215

    #Gerando matriz de confusao
217 from sklearn.metrics import confusion_matrix
    confusion_matrix(y_teste_array, pred_array)
```