Everton Victor de Vilas Bôas Júnior, Enrico Salemi Faria

Comparação de performance entre diferentes configurações de redes neurais

Itajubá - MG

2020

Everton Victor de Vilas Bôas Júnior, Enrico Salemi Faria

Comparação de performance entre diferentes configurações de redes neurais

Trabalho Final de Graduação apresentado à Universidade Federal de Itajubá como requisito para à obtenção do grau em Bacharel em Engenharia de Computação.

Universidade Federal de Itajubá – UNIFEI

Instituto de Engenharia de Sistemas e Tecnologia da Informação

Engenharia da Computação

Orientador: Maurílio Pereira Coutinho

Coorientador: Hanneli Carolina Andreazzi Tavante

Itajubá - MG

2020

Everton Victor de Vilas Bôas Júnior, Enrico Salemi Faria
Comparação de performance entre diferentes configurações de redes neurais/ Everton Victor de Vilas Bôas Júnior, Enrico Salemi Faria. – Itajubá - MG, 2020-
57p. : il. (algumas color.) ; 30 cm.

Orientador: Maurílio Pereira Coutinho

Tese (Graduação) – Universidade Federal de Itajubá – UNIFEI
Instituto de Engenharia de Sistemas e Tecnologia da Informação
Engenharia da Computação, 2020.

1. Rede neural. 2. Aprendizagem profunda. 3. Python. I. Maurílio Pereira Coutinho. II. Universidade Federal de Itajubá. III. Engenharia de Computação. IV. Comparação de performance entre diferentes configurações de redes neurais

Everton Victor de Vilas Bôas Júnior, Enrico Salemi Faria

Comparação de performance entre diferentes configurações de redes neurais

Trabalho Final de Graduação apresentado à Universidade Federal de Itajubá como requisito para à obtenção do grau em Bacharel em Engenharia de Computação.

Trabalho aprovado. Itajubá - MG, ___ de _____ de _____:

Maurílio Pereira Coutinho
Orientador

NOME
Convidado 1

NOME
Convidado 2

Itajubá - MG
2020

Agradecimentos

Dedico este trabalho primeiramente a minha família. Minha mãe, Rosana Ambrosio Salemi que mesmo não mais presente sempre me deu apoio e ensinamentos para me tornar uma pessoa melhor e é minha motivação para alcançar meus objetivos, meu pai, Ronaldo Coelho Faria que mesmo estando distante sempre me apoiou e não me deixou faltar nada, minha tia, Cristina Ambrosio Salemi que sempre esteve ao meu lado, meus avós, que me passaram ensinamentos e valores inestimáveis para meu amadurecimento, e à minha esposa Pamela Gabrielle Rodrigues de Sá Salemi Faria, que é minha motivadora e sempre esteve ao meu lado compartilhando momentos felizes e adversos. Agradeço aos meus amigos da universidade por compartilharem horas de estudo, de alegrias e de tristezas que nos fizeram uma segunda família. Também dedico este trabalho a Universidade Federal de Itajubá e a todos os professores, que fizeram parte do meu desenvolvimento intelectual e de milhares de colegas. Em especial agradeço ao Professor Maurílio Pereira Coutinho, por ser orientador desse trabalho e pela paciência com nossa equipe, e a nossa coorientadora Hanneli Carolina Andreazzi Tavante, que nos guiou na parte teórica do trabalho. Por fim agradeço a minha dupla nesse trabalho, Everton Victor de Vilas Bôas Júnior pela paciência e jornada.

Quero começar agradecendo minha namorada, e melhor amiga, Amanda Aparecida Barbosa da Silva, que me ajudou imensamente neste trabalho e sempre me apoia nas minhas decisões. Agradeço também aos meus pais, Everton Victor de Vilas Bôas e Maria de Lourdes da Silva Vilas Bôas, pelo incentivo para realização desse trabalho e apoio durante minha vida. À todos os meus amigos, dentro e fora da universidade, que me ajudaram da forma que podiam. À nossa coorientadora Hanneli Carolina Andreazzi Tavante e também, em especial, ao Professor Maurílio Pereira Coutinho por nos permitir essa oportunidade de concluir uma fase de nossas vidas. Sem todos vocês eu não estaria aqui.

"A wizard is never late, Master Baggins, nor is he early, he arrives precisely when he means to." Gandalf

Resumo

O campo da inteligência artificial vem crescendo substancialmente e atraindo a atenção de muitos iniciantes. Pensando nisso, o presente estudo visou avaliar o desempenho de uma rede neural ao ser treinada em variadas configurações. Para tal, foi desenvolvido um código em Python utilizando o framework Keras que pudesse ser facilmente configurado e pudesse aprender as características do conjunto de dados KDD99. Após simulações e análises, foi possível observar um aumento significativo na acurácia geral do modelo. Por fim os resultados obtidos nessa pesquisa poderão vir a agregar estudos futuros.

Palavras-chave: rede neural. aprendizagem profunda. limpeza de dados. Python.

Abstract

The field of artificial intelligence is growing substantially and getting the attention of many beginners. With that in mind, this study aims to evaluate the performance of a neural network trained in various configurations. For that, a code in Python was developed using the Keras framework in a way that it was easily configured and could learn the features of the KDD99 dataset. After simulations and analysis, it was possible to notice a significant increase in the general accuracy of the model. Finally, the results obtained in this research may add to future studies.

Keywords: neural network, deep learning, data cleaning, Python

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Os recentes avanços no campo da inteligência artificial, bem como nas tecnologias em geral, trouxeram diversas áreas de novas possibilidades para solucionar problemas em toda a indústria. Nesse contexto temos as redes neurais artificiais, que inicialmente foram desenvolvidas com base nas redes neurais biológicas e com o tempo foram incrementadas com o algoritmo *back-propagation*, *dropout* e outras funções de ativação. Seu princípio de funcionamento permite ser aplicada em diversas áreas, principalmente em problemas de reconhecimento, de previsão e de raciocínio. Por exemplo, no artigo de Sevilla e Sola, ela foi empregada para usinas nucleares. Neste artigo os autores apresentam possíveis usos para essa técnica e focam na importância de se trabalhar com bons conjuntos de dados e a passagem por um processo de limpeza para atingir melhores resultados em um menor tempo. (SOLA; SEVILLA, 1997)

1.1 Motivação

Na era do *Big Data*, numerosos dispositivos constituem uma rede complexa e crescente, na qual trafega uma grande quantidade de dados sigilosos. É bem provável que informações valiosas sejam expostas conforme são transferidas para cada um dos dispositivos dessa rede. Um dispositivo é tão seguro quanto o número de dispositivos a ele conectados. Essa nova era apresenta um problema desafiador e bastante importante para pessoas, companhias e governos. E embora termos realizado diversos avanços na segurança, os ataques também estão se tornando mais sofisticados e inteligentes. Existem diversos métodos para detecção de ataques. Um método, sistema de detecção de intrusão, monitora riscos baseados em padrões pré estabelecidos e possui uma taxa alta de falsos positivos. Para resolver esse problema da alta taxa de falsos positivos uma solução seria o uso de inteligência artificial. As detecções de ataques deverão ser mais inteligentes, é o que o artigo que originou este trabalho propõe: a utilização de redes neurais para um processo ágil e efetivo, sem a utilização dos métodos mais utilizados hoje com base em regras estabelecidas. (KIM NARA SHIN; KIM, 2017)

1.2 Objetivos

O objetivo desse trabalho é avaliar o desempenho de uma rede neural profunda ao realizar seu treinamento, alternando entre um conjunto de dados com todas as suas colunas (características) e o mesmo conjunto de dados, mas com colunas que possuem um certo grau de correlação excluídas. Esses dois cenários serão relacionados a diferentes

configurações em que serão trocados os números de camadas e nós da rede neural, a fim de verificar se haverá melhora na acurácia do modelo treinado.

1.3 Organização do trabalho

Este trabalho foi dividido em três partes principais começando com um levantamento das definições e ferramentas necessárias, passando pelo desenvolvimento do tema proposto e finalizando com as conclusões alcançadas. O Capítulo 2 inicia-se com uma apresentação das tecnologias utilizadas. Em seguida, na Seção 2.2, é explicado o ferramental estatístico que foi utilizado para o tratamento dos dados. Por fim, na Seção 2.3, é apresentada a teoria do funcionamento do aprendizado de máquinas e mais especificamente redes neurais. Em seguida, no Capítulo 3 é apresentado com clareza todo o seu desenvolvimento, começando pela apresentação do conjunto de dados utilizados. Após isso, na Seção 3.2 e a Seção 3.3, serão apresentados o algoritmo desenvolvido e o código. Na Seção 3.4, serão expostos os resultados obtidos após o treinamento. Por fim, na Seção 3.5, são apresentadas as comparações de desempenho em alguns dos parâmetros proposto do modelo. E, para finalizar, o Capítulo 4 apresenta a conclusão, nesse capítulo é discorrido sobre o quão importante foi realizar essa pesquisa, e também são apresentadas possíveis sugestões para trabalhos futuros.

2 Revisão bibliográfica

Neste capítulo serão apresentadas as noções base dessa pesquisa. Primeiramente, comentaremos rapidamente sobre a linguagem de programação Python, pois não é nosso objetivo ensinar esta linguagem. Para isto temos vários livros textos que tratam do assunto com profundidade. Também será visto as bibliotecas escolhidas para o desenvolvimento do código desse trabalho. A seguir, traremos o conceito estatístico que objetiva a limpeza dos dados e alguns meios de atingir esse objetivo. Finalmente, abordaremos a teoria de Aprendizado de Máquina utilizada para o aprendizado dos dados pela rede neural.

2.1 A linguagem de programação Python

Python é uma linguagem de programação de código aberto mantida pela *Python Software Foundation* (PSF) uma organização independente sem fins lucrativos que detém os direitos autorais das versões 2.1 e posteriores (PYTHON SOFTWARE FOUNDATION, 2019a). Possui uma licença que permite a modificação e venda da linguagem ou de produtos que a utilizem, sem que seja necessário tornar seu código aberto, com a condição de que a licença seja anexada à documentação do produto e que a PSF seja avisada sobre usos comerciais.

As linguagens de alto nível possuem uma camada de abstração sobre as operações realizadas pelo computador e foram desenvolvidas para facilitar o entendimento do código fonte escrito ao usar elementos naturais da linguagem humana, não sendo necessária a preocupação com detalhes como endereço de memória das variáveis (MOZILLA FOUNDATION, 2019).

Uma linguagem interpretada tem seus comandos analisados sintaticamente em tempo de execução, ao contrário de linguagens compiladas, que necessitam de um compilador para converter seu código fonte para código de máquina de uma plataforma específica para que possa ser executado. Isso faz com que os programas compilados sejam mais rápidos, mas permite uma portabilidade maior dos interpretados, já que só é necessário o interpretador para a plataforma usada e não há necessidade de se alterar o código fonte (INDIANA UNIVERSITY, 2018).

Uma linguagem orientada a objetos permite manipular estruturas, que possuem um agrupamento de propriedades e métodos para manipulação dessas propriedades. Linguagens orientadas a objetos possuem quatro características: encapsulamento, abstração, herança e polimorfismo. Encapsulamento é o nome dado ao agrupamento descrito acima. Abstração é a capacidade de poder esconder algumas propriedades e métodos de um objeto para acesso

externo, tornando algumas partes do código uma caixa preta para outras. A herança ajuda na eliminação de código redundante, permitindo que um objeto herde as propriedades e métodos de outro. E o polimorfismo permite a implementação distinta de um método para cada objeto, por exemplo, dois objetos carro e moto que herdam suas características de um objeto veículo possuem o método dirigir, mas a implementação desse método é distinta para cada um (WEISFELD, 2008).

Um dos motivos que torna *Python* uma das linguagens de programação mais populares da atualidade (THE ECONOMIST GROUP, 2018b; THE ECONOMIST GROUP, 2018a) é o repositório de programas escritos em *Python*, o *Python Package Index* (PyPI), que tem mais de 177000 bibliotecas compartilhadas (PYTHON SOFTWARE FOUNDATION, 2019b) por uma comunidade ativa e em constante crescimento (STACK EXCHANGE, INC., 2017; STACK EXCHANGE, INC., 2019).

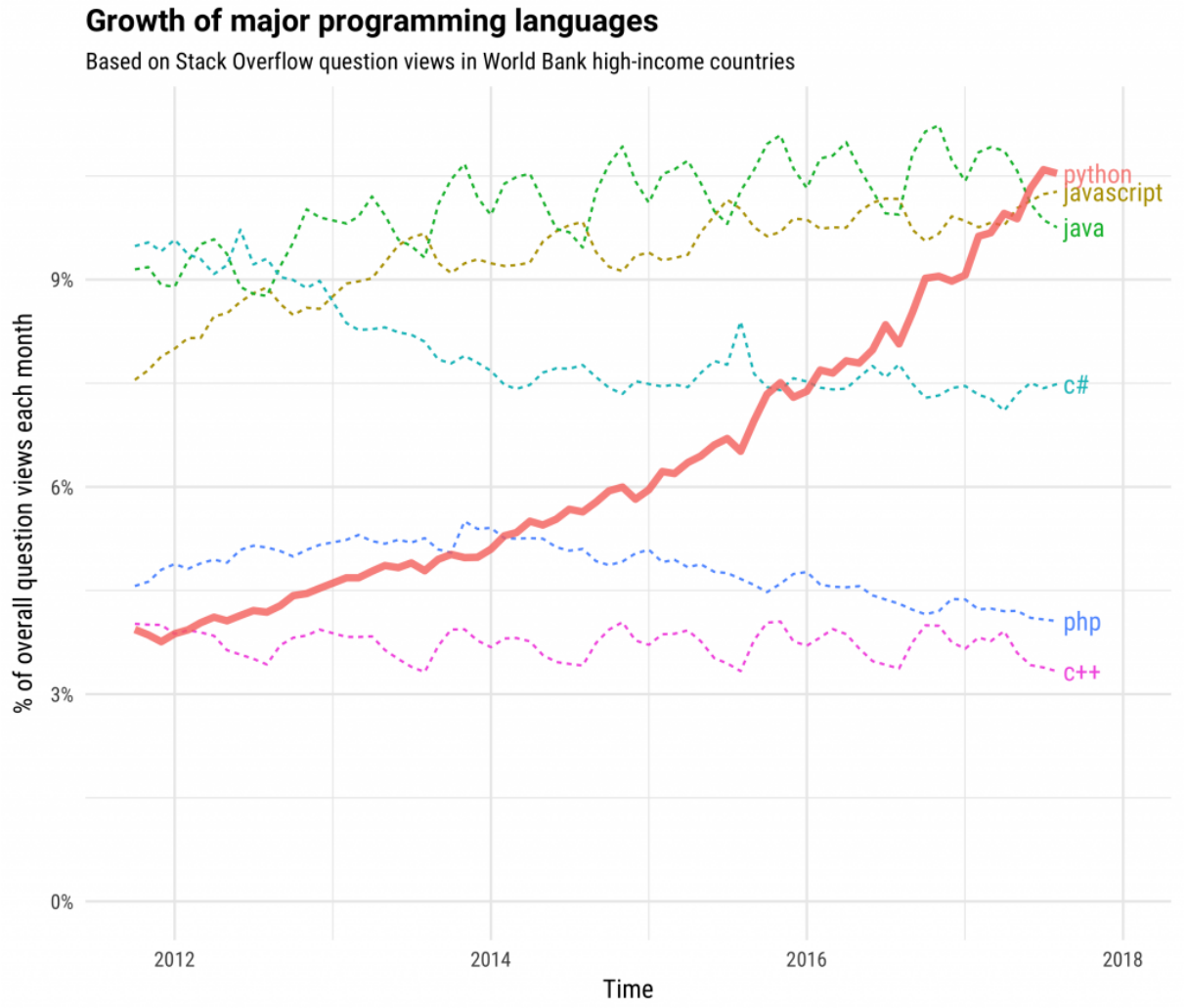


Figura 1 – Crescimento das linguagens de programação baseado nas visualizações de respostas do StackOverflow. Fonte: (STACK EXCHANGE, INC., 2017)

Para auxiliar o trabalho, algumas dessas bibliotecas, que serão comentadas a seguir,

foram utilizadas.

2.1.1 Tensorflow

Tensorflow é uma biblioteca para computação numérica de código aberto que tem seu núcleo escrito em C++. Inicialmente desenvolvida pelo time de inteligência artificial da Google, Google Brain, ela utiliza grafos onde os nós representam as operações matemáticas e as arestas representam vetores multidimensionais, tensores. A ideia de tensores fluindo pelo grafo de operações dá o nome à biblioteca (TENSORFLOW TEAM, 2019).

2.1.1.1 Keras

Keras é uma API (Interface de programação de aplicações) escrita em *Python* que adiciona uma camada de abstração ao *Tensorflow* e permite agilidade no desenvolvimento ao oferecer uma sintaxe mais amigável que minimiza o código escrito pelo usuário em troca da flexibilidade nas configurações de algumas funções de mais baixo nível. (KERAS TEAM, 2019).

2.1.2 Numpy

NumPy, abreviação de *Numerical Python*, provê funções para execução de operações nos elementos de vetores ou entre vetores, utilizando um objeto de vetor multidimensional chamado ndarray. Para dados numéricos, os vetores do NumPy são muito mais eficientes que os inclusos por padrão no *Python* (MCKINNEY, 2012).

2.1.3 Pandas

Essa biblioteca fornece uma estrutura de dados rica e funções projetadas para facilitar e agilizar o trabalho com elas. Pandas trabalha com um objeto chamado DataFrame, que consiste em uma estrutura de dados tabular em duas dimensões com rótulos tanto nas linhas quanto nas colunas, combinando a alta performance dos vetores do NumPy com a capacidade flexível de manipulação de dados de planilhas e bancos de dados relacionais. Sua indexação sofisticada permite a manipulação e seleção de subconjuntos de dados com facilidade. O nome, pandas, é derivado de *panel data*, um termo econométrico para conjunto de dados multidimensionalmente estruturados e *Python data analysis* (MCKINNEY, 2012).

2.1.4 Jupyter Notebook

Jupyter Notebook possui um ambiente de trabalho robusto, permitindo a escrita e compartilhamento de código de maneira mais prática e é útil para visualização de dados.

O Jupyter Notebook é uma aplicação que pode ser acessada por um navegador e permite a edição de código em blocos que podem ser executados individualmente. Cada bloco tem a capacidade de imprimir uma saída que não seja apenas textos, mas também imagens, como gráficos. O conjunto desses blocos é denominado *notebook*. Além da possibilidade de execução de códigos, é possível criar blocos explicativos que contextualizam e explicam o que está executado. Não é necessária a instalação de nenhum software para a visualização de um *notebook*, é possível acessar uma aplicação web no endereço https://nbviewer.jupyter.org que permite que isso seja feito apenas com o endereço de onde ele está sendo hospedado. (KLUYVER et al., 2016; JUPYTER TEAM, 2020).

2.2 Tratamento dos dados

A qualidade dos dados é um fator importante ao realizar uma pesquisa e está relacionada com a validade, generalização e replicabilidade dos resultados obtidos (OSBORNE, 2010). Em um treinamento de uma Rede Neural o tratamento adequado pode trazer benefícios em reduzir significativamente o tempo de processamento e aumentar o acerto. (SOLA; SEVILLA, 1997). Os dados coletados são suscetíveis a erros em diversos momentos, por exemplo, a modelagem do que se coletar pode gerar redundância de informação com representações diferentes, ou o instrumento de coleta pode apresentar erros gerando valores acima do esperado. (BOSLAUGH, 2012) Portanto, observando que a qualidade dos resultados está ligada a bons resultados foi utilizado um ferramental estatístico para corrigir possíveis erros.

2.2.1 Correlação

Em estatística, correlação pode ser definida como o método para quantificar a relação entre duas variáveis quantitativas e contínuas. Essa quantificação é chamada de coeficiente de correlação, sendo os mais comuns o coeficiente de Pearson e o coeficiente de Spearman. Para esse trabalho, a correlação linear entre as variáveis é dada pelo coeficiente de correlação de Pearson:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2(y_i - \overline{y})^2}} \tag{2.1}$$

Em que r é o coeficiente de correlação que pode assumir valores entre -1 (correlação negativa) e +1 (correlação positiva) e quanto mais próximo de 0, menor é a correlação.

Quanto maior o coeficiente de Pearson mais aparente é a linearidade entre as variáveis. Nesse trabalho as variáveis com correlação moderada ou alta, ou seja, maior que 0,5 e menor que -0,5, foram eliminadas. Como o coeficiente não apresenta robustez a

valores atípicos e não normais, é necessário uma análise dos *outliers* e da normalização dos dados. (SWINSCOW, 1997; MUKAKA, 2012; OSBORNE, 2010; SOLA; SEVILLA, 1997).

2.2.2 Normalização

Grandes conjuntos de dados podem conter valores esparsos que podem atrapalhar a velocidade de treinamento de uma rede neural. Por exemplo, se fossem utilizadas informações sobre mamíferos para um certo treinamento poderia ser verificado uma variação de massa e tamanho de 0.0018kg a 150000kg e de 0.04m a 30m, respectivamente. A normalização dos dados tem como objetivo deixar os valores em um mesmo intervalo, entre 0 e 1, por exemplo, para melhorar a exatidão dos modelos treinados (MEDIUM, 2018; SOLA; SEVILLA, 1997).

2.2.3 Outlier

Outliers não possuem um consenso em sua definição, mas no geral podem ser descritos como pontos no conjunto de dados que diferem do comportamento dos demais. Essas anomalias são importantes de serem encontradas e analisadas, pois podem mostrar erros na coleta dos dados. Esses dados afetam parâmetros estatísticos como variância e media, diminuindo a validez dos testes de hipóteses e no caso das RN pode enviesar os resultados. Em geral, os critérios para decidir se o valor é anômalo diferem bastante entre os autores, para esse estudo foi adotado que todos os valores que estejam fora de três desvios padrão sejam classificados como *outlier* e removidos. (OSBORNE, 2010; BOSLAUGH, 2012).

2.3 Aprendizado de Máquina e Aprendizagem Profunda

2.3.1 Aprendizado de Máquina

Aprendizado de Máquina é uma forma de inteligência artificial no qual os programas são desenvolvidos para aprender sozinhos através de algoritmos que extraem padrões de dados conhecidos e preveem valores para dados ainda não vistos. De uma forma simples, Aprendizado de Máquina é um conjunto de técnicas matemáticas implementadas em um sistema computacional que possibilita a extração, descobertas de padrões e inferência de resultados. (CHIO; FREEMAN, 2018)

É possível realizar uma comparação entre os programas sequenciais tradicionais e os que têm como objetivo aprender. Se tomarmos como exemplo a detecção de spam em um *e-mail*, para um programa tradicional teríamos que colocar inúmeras estruturas condicionais procurando palavras suspeitas no meio do texto como prêmio, herança e recompensa. Mas além do código tomar proporções gigantescas, poderíamos esquecer

alguma palavra. Por outro lado, um programa desenvolvido para aprender o que é spam ou não, seria menor e talvez mais abrangente, dependendo dos exemplos usados em seu treinamento. (LENT, 2010)

2.3.2 Aprendizagem Profunda

Aprendizagem Profunda é um subcampo de Aprendizado de Máquina inspirado pela estrutura e funcionamento das redes neurais cerebrais. Os neurônios podem ser considerados a unidade básica de estrutura do cérebro, produzindo e transportando diminutos sinais elétricos que podem ser considerados bits de informação, como ilustrado a seguir.

> [...] subitamente você ouve o barulho de uma freada de automóvel, seguido de um estrondo. Você leva um susto, seu coração dispara, e você se pergunta se terá ocorrido um acidente de trânsito, e se alguém se feriu. Como ocorreram esses fenômenos em você? Neste caso, tudo começou no seu ouvido, que foi capaz de transformar em impulsos nervosos as vibrações do ar correspondentes aos sons que você ouviu. Um circuito de neurônios conectando os seus ouvidos com as regiões temporais do cérebro permitiu que você percebesse perfeitamente os sons, e mais: que você os comparasse com o "catálogo de sons" armazenado em sua memória, e os identificasse como barulhos de freada e de uma batida. (LENT, 2010)

Um conjunto de neurônios associados formam uma rede neural. Trazendo as redes neurais para um contexto computacional temos as Redes Neurais Artificiais (RNAs). Uma RNA é composta por diversos nós distribuídos em várias camadas e interligados por pesos.

A camada de entrada possui um nó para cada variável de característica de um conjunto de dados. Por exemplo, se um conjunto de dados sobre pessoas fosse composto por apenas três informações, como peso, altura e tamanho do calçado, a camada de entrada teria três nós.

Não há um número fixo para as camadas intermediárias, conhecidas como camadas ocultas, e o seu número de nós, podendo ser alteradas de acordo com a necessidade do projeto e experiência do programador. Cada camada oculta realiza uma operação diferente nos dados contribuindo para uma melhor generalização dos dados.

A camada de saída também pode variar dependendo da estrutura do projeto. No caso de um classificador, por exemplo, a quantidade de nós da camada de saída seria igual a quantidade de classificações possíveis para aqueles dados.

Os pesos que conectam a rede são números reais que demonstram a influência que os nós exercem entre si. Cada nó recebe como entrada as saídas da camada anterior multiplicadas pelos seus respectivos pesos, que então são somadas e utilizadas como

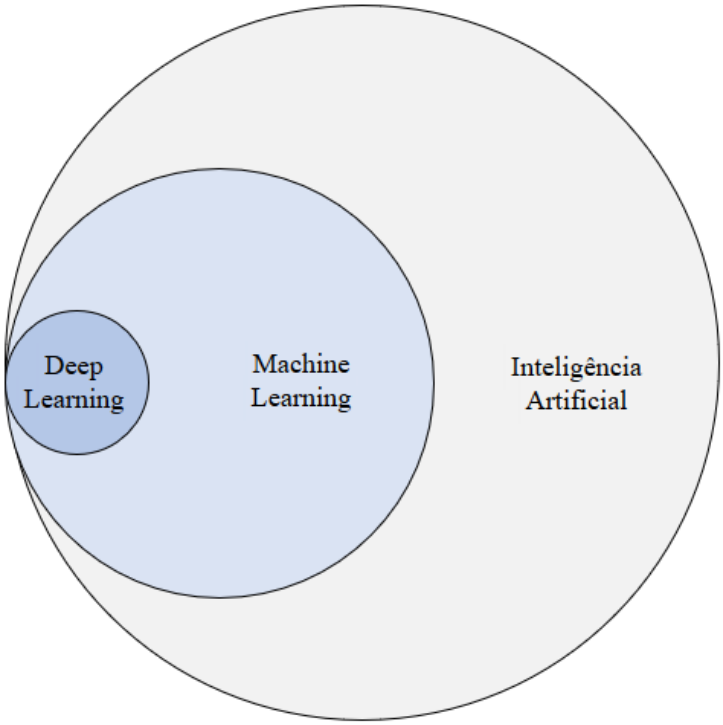


Figura 2 – Relação entre Inteligência Artificial, Aprendizado de Máquina e Deep Learning. Fonte: (PATTERSON; GIBSON, 2017)

parâmetro para a função de ativação desse nó, que funciona similar ao estímulo que os neurônios de um cérebro recebem.

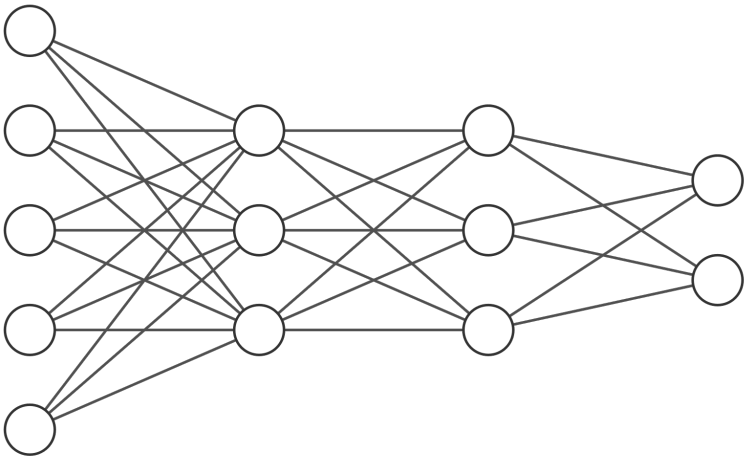


Figura 3 – Exemplo de uma Rede Neural Artificial. Fonte: (PATTERSON; GIBSON, 2017)

A seguir são mostradas as funções de ativação (KETKAR, 2017) utilizadas nesse trabalho na sua forma padrão com seu *threshold* de ativação perto de zero. Mais a frente será discutido o valor de viés, *bias* em inglês, que pode ser adicionado aos nós da rede que faz com que a função de 'mova' na horizontal.

A ReLU (Do inglês *rectified linear unit*, unidade linear retificada), faz com que

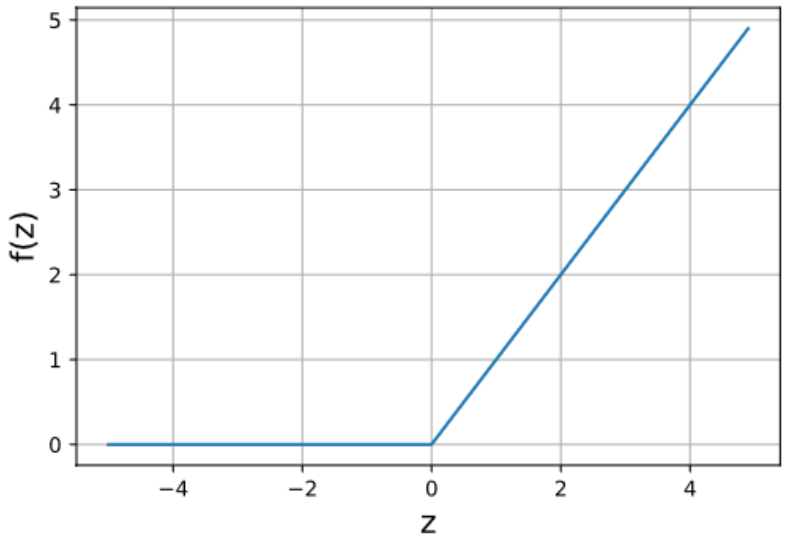


Figura 4 – Relu

todas as saídas sejam positivas escolhendo o máximo entre entre o valor de entrada.

$$f(z) = max(0, z) \tag{2.2}$$

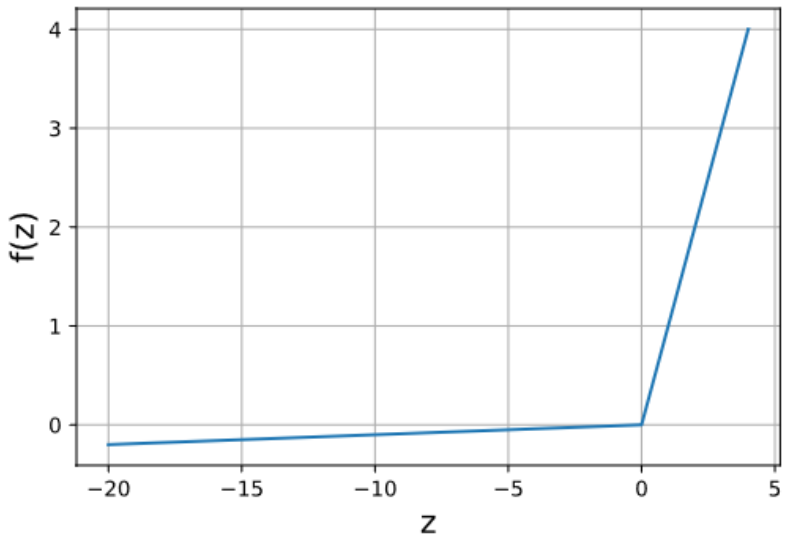


Figura 5 – Leaky Relu

A Leaky ReLU é uma variante da ReLU onde as entradas menores que zero não são desconsideradas, mas multiplicadas por um parâmetro α que toma como valor números muito pequenos como 0.01, por exemplo.

$$f(z) = max(\alpha * z, z) \tag{2.3}$$

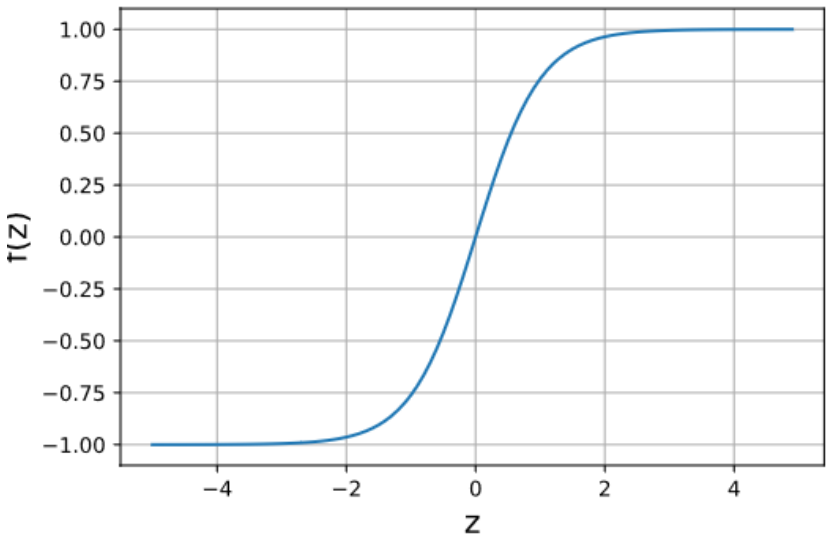


Figura 6 – Tangente Hiperbólica.

A função de ativação da tangente hiperbólica limita a saída entre -1 e 1 para valores muito distantes de zero.

$$f(z) = tanh(z) \tag{2.4}$$

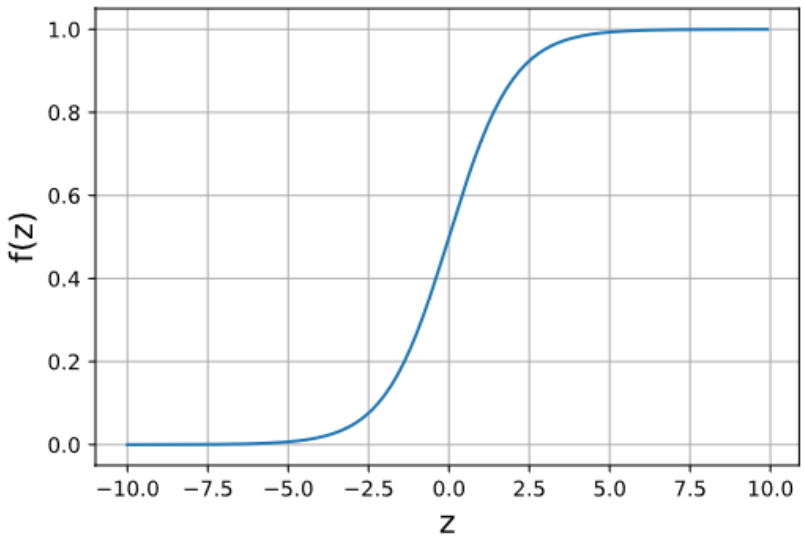


Figura 7 – Sigmoide.

A sigmoide tem um formato muito parecido com tangente hiperbólica com a diferença que sua saída é limita entre 0 e 1.

$$sig(z) = \frac{1}{1 + e^{-z}} \tag{2.5}$$

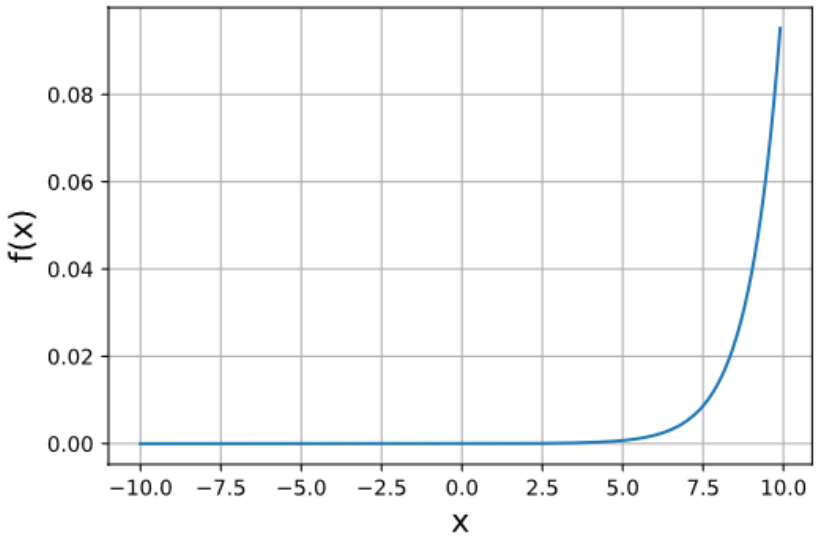


Figura 8 – Softmax.

A função softmax tem como saída uma representação da distribuição de probabilidades das classificações.

$$softmax(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.6}$$

Observando o gráfico da função de ativação sigmoide adicionada de *bias* é possível perceber que o *threshold* que antes ficava perto de zero se modifica, no caso da figura abaixo a soma dos valores que entram no nó precisam ser maiores que cinco para que a saída se aproxime de um.

Com essa informação é possível redesenhar um nó completo como visto na Figura 10 e escrever a equação abaixo onde g é a função de ativação.

$$a_s = g(\sum a_i * w_i + b) \tag{2.7}$$

2.3.3 Treinamento

Para este trabalho, foi necessário treinamento para a rede neural aprender as características do conjunto de dados utilizado. Por isso, aqui serão apresentados alguns

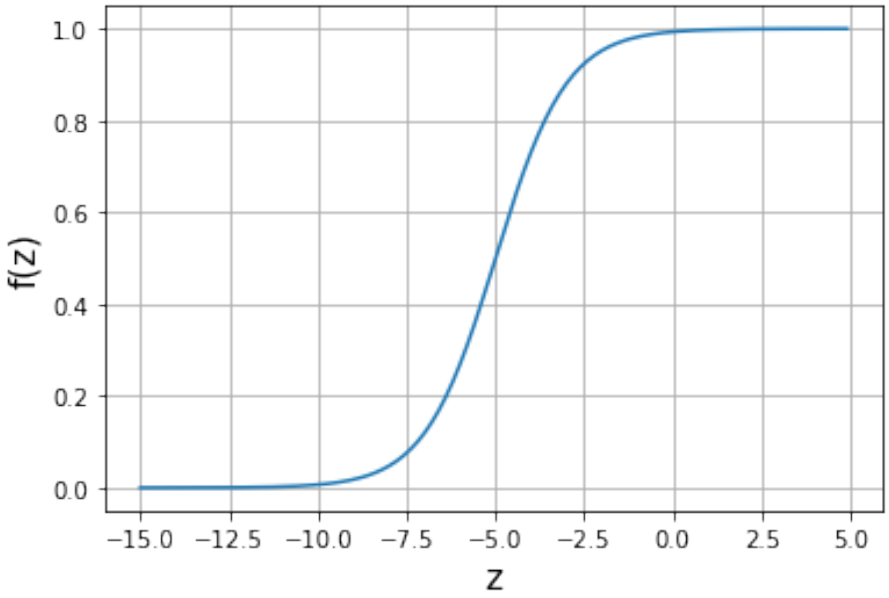


Figura 9 – Sigmoid adicionada de um viés.

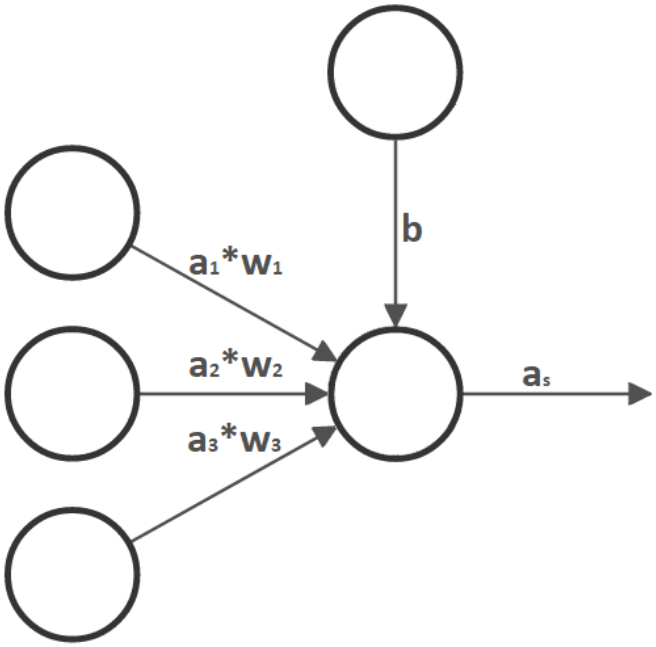


Figura 10 – Nó em detalhe.

conceitos, como: conjunto de dados, função de perda (do inglês *loss function*), retropropagação (do inglês *backpropagation*), sobreajuste (do inglês *overfitting*) e subajuste (do inglês *underfitting*).

Para que um modelo seja treinado é necessário que exista um conjunto de dados sobre o que ele deve aprender. Podemos dividir os dados em três conjuntos distintos: treinamento, validação e teste. O conjunto de treinamento é utilizado para o aprendizado, é a partir dele que os parâmetros do modelo, que dão a capacidade de generalização para ele, são ajustados para tentar encontrar os pesos ótimos entre os nós. O conjunto de validação é utilizado para realizar pequenos ajustes nos parâmetros do modelo. E o conjunto de teste é usado apenas para medir a performance de um modelo já completamente treinado. Não existe uma regra da divisão exata dos dados entre cada grupo, mas é necessário manter em mente que os dados de treinamento devem ser extensos o suficiente para que haja sentido no treinamento e que os conjuntos de treinamento e validação englobem o mesmo tipo de dados, com características similares. (GOOGLE DEVELOPERS, 2020)

A função de perda, ou função de custo, calcula o quão longe do valor real é o valor predito. A finalidade do treinamento é encontrar um conjunto de pesos para o modelo que faça com que a função de custo retorne o menor valor possível. Geralmente, quanto menor é esse valor, melhor é o modelo treinado. Porém, um valor pequeno para a função de custo nem sempre indica que o modelo está bem treinado. Quando um modelo apresenta um erro baixo para os dados em que foi treinado, mas não tem a capacidade de generalizar para dados nunca visto antes, dizemos que há um problema de *overfitting*, ou sobreajuste. Uma das soluções para esse problema é a aquisição de mais dados para aumentar sua diversidade. É possível também alterar o formato do modelo, removendo algumas camadas da rede neural ou diminuindo o número de neurônios. O problema contrário a esse se chamada *underfitting*, ou sub-ajuste. Nesse caso, o modelo não consegue classificar tanto os dados de treinamento quanto os de teste. Uma das soluções aqui é contrária ao problema anterior, podemos alterar o modelo, adicionando novas camadas e neurônios, pois os dados podem ser muito complexos e o modelo simples demais. Uma outra solução é, se possível, adicionar mais atributos aos dados.

Para ajudar na atualização dos pesos da rede neural, podemos utilizar a técnica conhecida como *backpropagation*, onde os pesos entre as camadas do modelo são atualizados de trás para frente, para que o valor da saída correta seja reforçado e diminuir o estímulo das saídas incorretas a fim de minimizar a função de custo. Os novos valores dos pesos entre as camadas do modelo são derivadas da função de custo com respeito a cada um desses pesos. Esse procedimento é realizado da saída do modelo em direção a entrada, de trás para frente, a fim de reutilizar o resultado das derivadas já calculadas, pois elas são funções custosas computacionalmente. (PATTERSON; GIBSON, 2017)

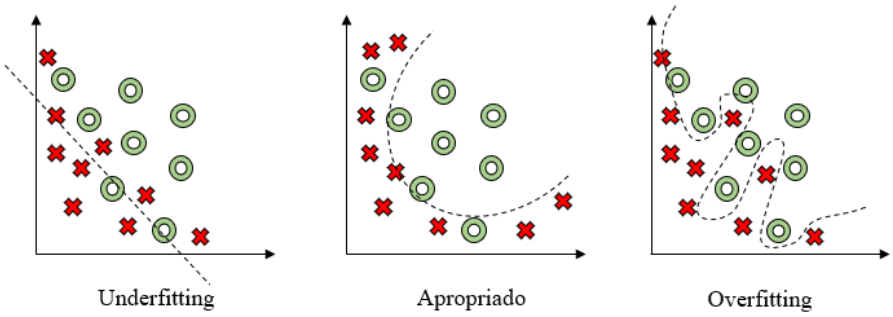
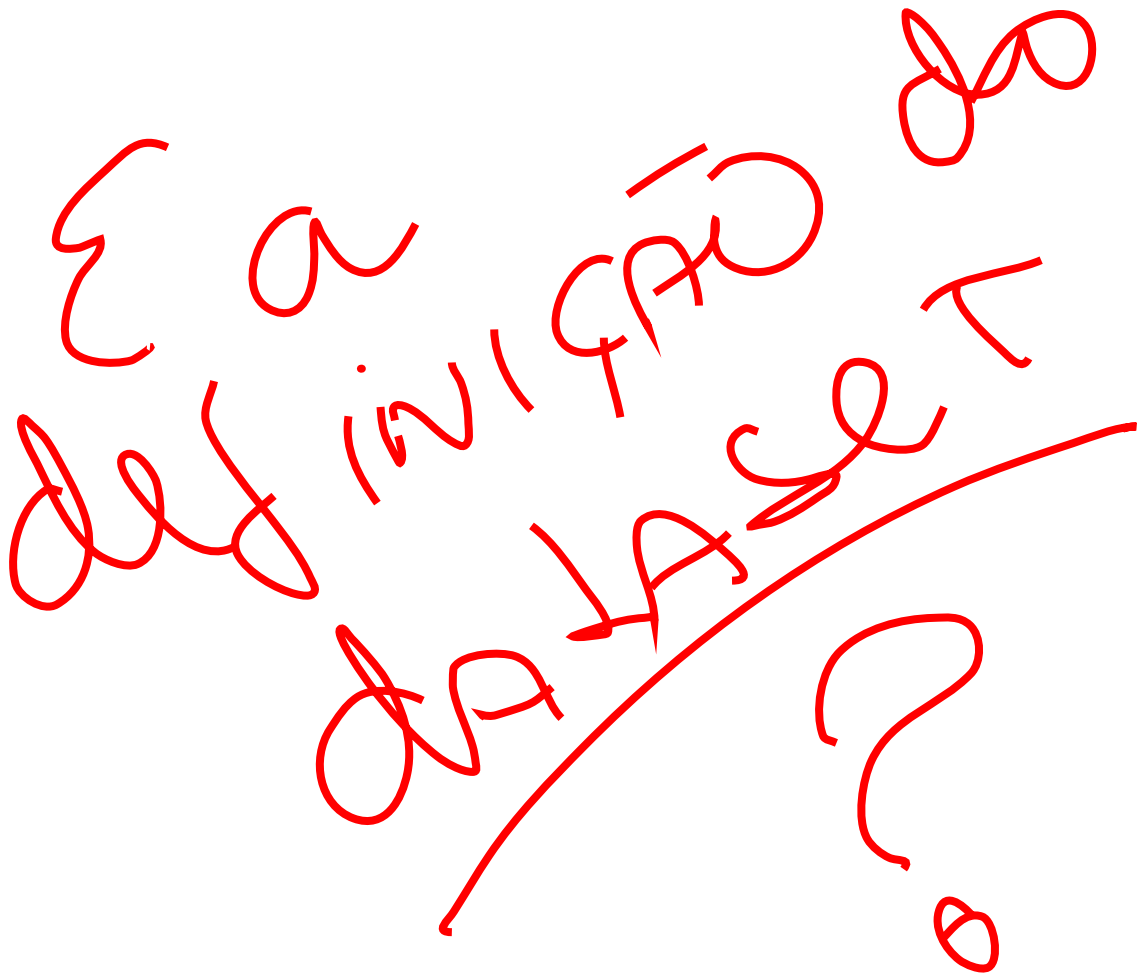


Figura 11 – Comparação entre overfitting, underfitting e uma divisão apropriada. Fonte: (PATTERSON; GIBSON, 2017)

3 Desenvolvimento

Para o desenvolvimento desse trabalho, foi utilizado um conjunto de dados retirado do acervo de extração de conhecimento de base de dados (KDD, do inglês *knowledge-discovery in databases*) da Universidade da Califórnia Irvine (UCI), utilizado na Terceira Competição Internacional de Extração de Conhecimento de Base de Dados e Ferramentas de Mineração de Dados realizada em conjunto com a KDD-99, a Quinta Conferência Internacional de Extração de Conhecimento de Base de Dados e Mineração de Dados. Essa base de dados consiste de uma extensa variedade de intrusões simuladas em um ambiente de rede militar separadas em conexões boas e ruins, distribuídas em 4898431 linhas e 41 colunas, que possuem valores numérico e léxicos.

A seguir, serão apresentado o algoritmo e o código desenvolvidos para alcançar a proposta desse trabalho.



3.1 Algoritmo

A seguir, temos o algoritmo que resume a lógica geral do código desenvolvido para a realização do trabalho, que será apresentado em detalhes na próxima seção.

```
Importar bibliotecas;
Definir constantes;
Ler conjunto de dados;
Transformar dados léxicos em numéricos;
se flag para limpeza == true então
    para i = 0 até número de colunas -1 faça
        para j = 0 até número de colunas -1 faça
            se i != j então
                Calcular correlação entre coluna[i] e coluna[j];
                se correção > limite então
                    Excluir coluna[j];
                fim
            fim
        fim
    fim
fim
Declarar modelo da rede neural;
Treinar modelo da rede neural;
Imprimir resultados;
```

Algoritmo 1: Lógica do código utilizado (Fluxograma em anexo A)

3.2 Código

Essa seção tem como objetivo apresentar e explicar o código utilizado no treinamento do modelo da rede neural, retratado da mesma forma que se encontra em nosso *notebook*.

O primeiro bloco de código apenas importa todas as bibliotecas utilizadas no programa.

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4 import keras
5 from keras import backend as K
6 from keras.models import Sequential
7 from keras.layers import Activation
8 from keras.layers.core import Dense
9 from keras.optimizers import Adam
```

```
from keras.metrics import categorical_crossentropy
from keras.utils import plot_model
from keras import optimizers
from keras.utils import to_categorical
import time
```

Para facilitar as trocas de parâmetros, todas as configurações do treinamento estão no bloco abaixo. Aqui é escolhido qual conjunto de dados será utilizado e se algumas de suas colunas serão removidas devido à limpeza de correlações. Também é possível configurar a taxa de aprendizado, função de ativação utilizada, o número de ciclos de treinamento e a divisão do conjunto de dados.

```
# Flag para remocao das colunas correlacionadas.
CLEAN_DATASET = True

# Flag para decisao de qual versao do dataset usar
DATASET_COMPLETO = False

# Caminho para o dataset
# Endereco para download: http://kdd.ics.uci.edu/databases/kddcup99/
    kddcup99.html
# Backup: http://web.archive.org/web/*/http://kdd.ics.uci.edu/databases/
    kddcup99/kddcup99.html
if DATASET_COMPLETO:
    DATASET_PATH = "D:/UNIFEI/TCC/kdd_dl/dataset/kddcup.data.corrected"
    # Treino 98%, Validacao 1%, Teste 1%
    # https://stackoverflow.com/a/13613316
    TRAIN_PERCENTAGE = 0.98
    VALIDATION_PERCENTAGE = 0.01
    TEST_PERCENTAGE = 0.01
else:
    DATASET_PATH = "D:/UNIFEI/TCC/kdd_dl/dataset/kddcup.
    data_10_percent_corrected"
    # Treino 80%, Validacao 10%, Teste 10%
    TRAIN_PERCENTAGE = 0.8
    VALIDATION_PERCENTAGE = 0.1
    TEST_PERCENTAGE = 0.1

## Threshold de correlacao para exclusao das colunas do dataset
CORRCOEF_THRESHOLD = 0.5

## Ignora os warnings de divisao por NaN
np.seterr(divide='ignore', invalid='ignore')


EPOCHS = 200 # Quantas vezes os dados serao corridos completamente
BATCH_SIZE = 1024 # Quantas linhas do dataset serao lidas por vez
```

```
SHUFFLE = True # Mistura os dados antes do treino
VERBOSE = 1 # 0 - Desativa os logs durante o treino, 1 - Imprime os logs
    durante o treino
LEARNING_RATE = 0.01 # Taxa de aprendizado
ACTIVATION_FN = "relu" # Funcao de ativacao
```

O bloco abaixo realiza a leitura do conjunto de dados que está em um arquivo no formato CSV (*Comma-separated values*) e coloca seus valores em uma estrutura chamada *dataframe*, da biblioteca pandas, para facilitar a manipulação dos dados.

```
# Nome das colunas do dataset
column_names = [
    "duration", "protocol_type", "service", "flag", "src_bytes", "
    dst_bytes",
    "land", "wrong_fragment", "urgent", "hot", "num_failed_logins", "
    logged_in",
    "num_compromised", "root_shell", "su_attempted", "num_root", "
    num_file_creations",
    "num_shells", "num_access_files", "num_outbound_cmds", "
    is_host_login", "is_guest_login",
    "count", "srv_count", "serror_rate", "srv_serror_rate", "rerror_rate
    ", "srv_rerror_rate",
    "same_srv_rate", "diff_srv_rate", "srv_diff_host_rate", "
    dst_host_count", "dst_host_srv_count",
    "dst_host_same_srv_rate", "dst_host_diff_srv_rate", "
    dst_host_same_src_port_rate",
    "dst_host_srv_diff_host_rate", "dst_host_serror_rate", "
    dst_host_srv_serror_rate",
    "dst_host_rerror_rate", "dst_host_srv_rerror_rate", "attack",
]

#Leitura do dataset
dataset = pd.read_csv(
    DATASET_PATH,
    header = None,
    names = column_names)
```

Como o conjunto de dados utilizado possui valores numéricos e léxicos, utilizamos o próximo bloco de código para transformar os léxicos em inteiros, por exemplo, transformar a *string rootkit* no inteiro 1, a *string guess_passwd* no inteiro 2 e assim por diante.

```
#Pega os valores  unicos das colunas que tem string
protocol_type_values = dataset.protocol_type.unique()
service_values = dataset.service.unique()
flag_values = dataset.flag.unique()
attack_values = dataset.attack.unique()

#Transforma as strings em numeros
```

```
protocol_type_dict = dict(zip(protocol_type_values, range(len(
    protocol_type_values))))
service_dict = dict(zip(service_values, range(len(service_values))))
flag_dict = dict(zip(flag_values, range(len(flag_values))))
attack_dict = dict(zip(attack_values, range(len(attack_values))))
```

```
#Substitui os valores no dataset que sao string por numeros
dataset = dataset.replace(
    {"protocol_type": protocol_type_dict,
     "service": service_dict,
     "flag": flag_dict,
     "attack": attack_dict}
)

dataset = dataset.apply(pd.to_numeric)
```

O próximo bloco percorre todas as colunas do conjunto de dados e realiza a exclusão das que apresentam forte correlacionamento, caso isso esteja definido nas configurações. Se nosso conjunto de dados fosse composto pelas colunas A, B e C, ele calcularia a correlação entre A e B, A e C, e B e C, excluindo uma das colunas para o treinamento.

```
#Apaga as colunas que tem correlacao entre elas que seja maior que
    CORRCOEF_THRESHOLD
if CLEAN_DATASET:
    excluded = list()

    for index_1 in range(len(column_names)-1):
        for index_2 in range(len(column_names)-1):
            if column_names[index_1] not in excluded and column_names[
    index_2] not in excluded and index_1 != index_2:
                corrcoef = np.corrcoef(dataset[column_names[index_1]],
    dataset[column_names[index_2]])

                if abs(corrcoef[0][1]) > CORRCOEF_THRESHOLD:
                    excluded.append(column_names[index_2])

    dataset = dataset[dataset.columns.difference(excluded)]
```

Como podemos escolher nas configurações entre o treinamento com o conjunto de dados completo, composto por aproximadamente 4000000 de linhas, e o conjunto de dados parcial, composto por aproximadamente 400000 linhas, é necessário que possamos separar os dados de treino e teste de forma diferenciada. Caso os mesmos percentuais de divisão fossem utilizados para os dois conjuntos, teríamos a divisão de treino/teste/validação de 3200000/400000/400000 para o completo e 320000/40000/40000 para o parcial. Para o conjunto completo, 800000 linhas para validação e teste é um número muito alto, sendo que podemos utilizar grande parte desses dados para o treinamento, em que os dados serão

melhor aproveitados. Assim, o conjunto completo é dividido em 3920000/40000/40000, adicionando 720000 linhas para o treino, que é a parte mais importante do processo.

```
#Separa os dados entre treino e teste
train, test = np.split(dataset.sample(frac=1), [int(TRAIN_PERCENTAGE *
    len(dataset))])

train_samples = train.drop(['attack'], axis=1)
train_labels = to_categorical(train[['attack']].to_numpy(), num_classes=
    len(attack_values))

test_samples = test.drop(['attack'], axis=1)
test_labels = to_categorical(test[['attack']].to_numpy(), num_classes=
    len(attack_values))
```

Os bloco abaixo criam a rede neural utilizada nesse trabalho.

```
input_shape = (train_samples.shape[1],)

# Criacao do modelo
model = Sequential([
    Dense(15, input_shape=input_shape, activation=ACTIVATION_FN),
    Dense(15, activation=ACTIVATION_FN),
    Dense(15, activation=ACTIVATION_FN),
    Dense(15, activation=ACTIVATION_FN),
    Dense(15, activation=ACTIVATION_FN),
    Dense(len(attack_values), activation='softmax'),
])
```

```
sgd = optimizers.SGD(lr=LEARNING_RATE)
model.compile(loss='mean_squared_error', optimizer=sgd)

model.compile(
    optimizer=sgd,
    loss='mean_squared_error',
    metrics=['accuracy'],
)
```

O treino da rede neural está apresentado no bloco a seguir, em que seu tempo é medido para comparações futuras.

```
start_time = time.time()

# Treinamento do modelo
history = model.fit(
    x=train_samples,
    y=train_labels,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
```

```
    shuffle=SHUFFLE, #Mistura os dados
    verbose=VERBOSE,
    validation_split=VALIDATION_PERCENTAGE #Porcentagem dos dados de
   treino que serao usadas para validacao
)

end_time = time.time()
```

Aqui, o modelo é avaliado após o treino.

```
# Teste do modelo
h2 = model.evaluate(
    x=test_samples,
    y=test_labels,
    verbose=VERBOSE,
)
# plot_model(model, to_file='model.png' ,show_shapes=True,
   show_layer_names=True, expand_nested=True, dpi=200)
```

Esse bloco imprime as configurações utilizadas no treino e seus resultados.

```
# plot_model(model, to_file='model.png')

# Plot training & validation accuracy values
plt.figure(figsize=(10,5))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.figure(figsize=(10,5))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

print(model.metrics_names)
print(h2)

print({
"CLEAN_DATASET":CLEAN_DATASET,
"DATASET_COMPLETO":DATASET_COMPLETO,
```

```
"CORRCOEF_THRESHOLD":CORRCOEF_THRESHOLD,
"BATCH_SIZE":BATCH_SIZE,
"EPOCHS":EPOCHS,
"SHUFFLE":SHUFFLE,
"VERBOSE":VERBOSE,
"VALIDATION_SPLIT":VALIDATION_PERCENTAGE,
"LEARNING_RATE":LEARNING_RATE,
"ACTIVATION_FN":ACTIVATION_FN,
"TEMPO": end_time-start_time})
```

3.3 Resultados

Os dados coletados foram resultados das simulações ao realizar a permutação entre o número de camadas, número de nós e a limpeza dos dados. No total, foram realizadas doze simulações diferentes e cada uma foi repetida cinco vezes para uma melhor comparação dos dados. Para uma melhor análise dos dados, eles foram divididos em três grupos: no primeiro, temos a comparação do treinamento que leva em conta a limpeza dos dados; no segundo, é alterado o número de nós; e no terceiro, é alterado o número de camadas.

O primeiro grupo de comparações estende-se da tabela 1 à 6 e apresenta os dados das simulações que mantiveram os mesmos números de camadas e nós da rede neural e foi apenas alterado caso o conjunto de dados fosse limpo ou não, como podemos ver a seguir:

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
1	15	Não	234,952	0,751
1	15	Não	246,394	0,001
1	15	Não	235,468	0,402
1	15	Não	243,304	0,732
1	15	Não	242,009	0,198
1	15	Sim	195,218	0,948
1	15	Sim	203,252	0,935
1	15	Sim	198,034	0,741
1	15	Sim	190,972	0,568
1	15	Sim	223,269	0,171

Tabela 1 – Resultados das simulações com uma camada e quinze nós por camada permutando a limpeza dos dados.

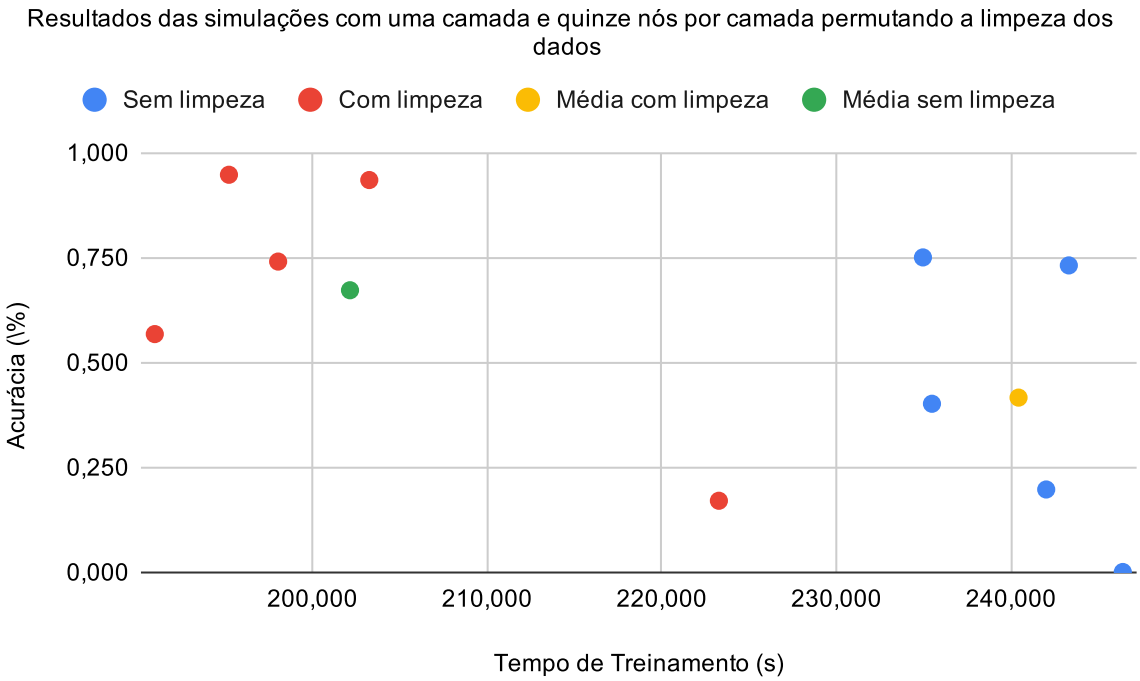


Figura 12 – Representação dos dados da tabela 1 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
1	50	Não	257,859	0,002
1	50	Não	256,538	0,981
1	50	Não	253,082	0,788
1	50	Não	261,893	0,005
1	50	Não	266,676	0,787
1	50	Sim	220,776	0,010
1	50	Sim	205,175	0,198
1	50	Sim	204,139	0,184
1	50	Sim	234,664	0,187
1	50	Sim	246,734	0,189

Tabela 2 – Resultados das simulações com uma camada e cinquenta nós por camada permutando a limpeza dos dados.

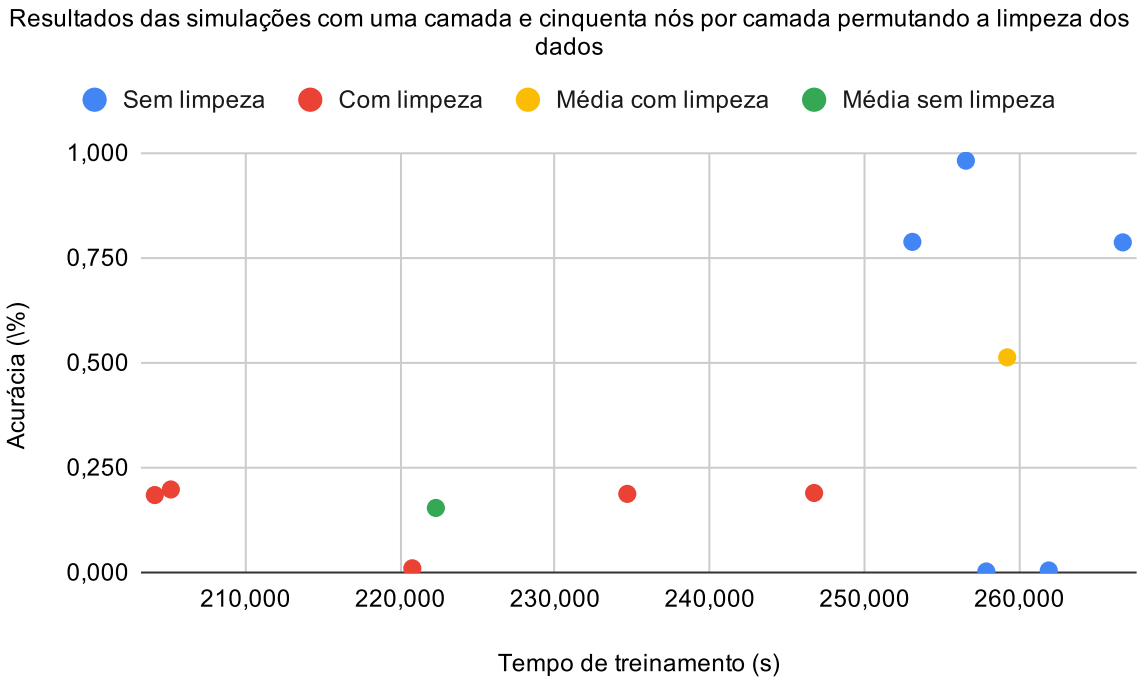


Figura 13 – Representação dos dados da tabela 2 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
5	15	Não	283,013	0,981
5	15	Não	283,164	0,981
5	15	Não	296,976	0,981
5	15	Não	478,459	0,981
5	15	Não	289,293	0,981
5	15	Sim	254,350	0,975
5	15	Sim	269,369	0,764
5	15	Sim	262,709	0,973
5	15	Sim	258,133	0,410
5	15	Sim	269,217	0,969

Tabela 3 – Resultados das simulações com cinco camadas e quinze nós por camada permutando a limpeza dos dados.

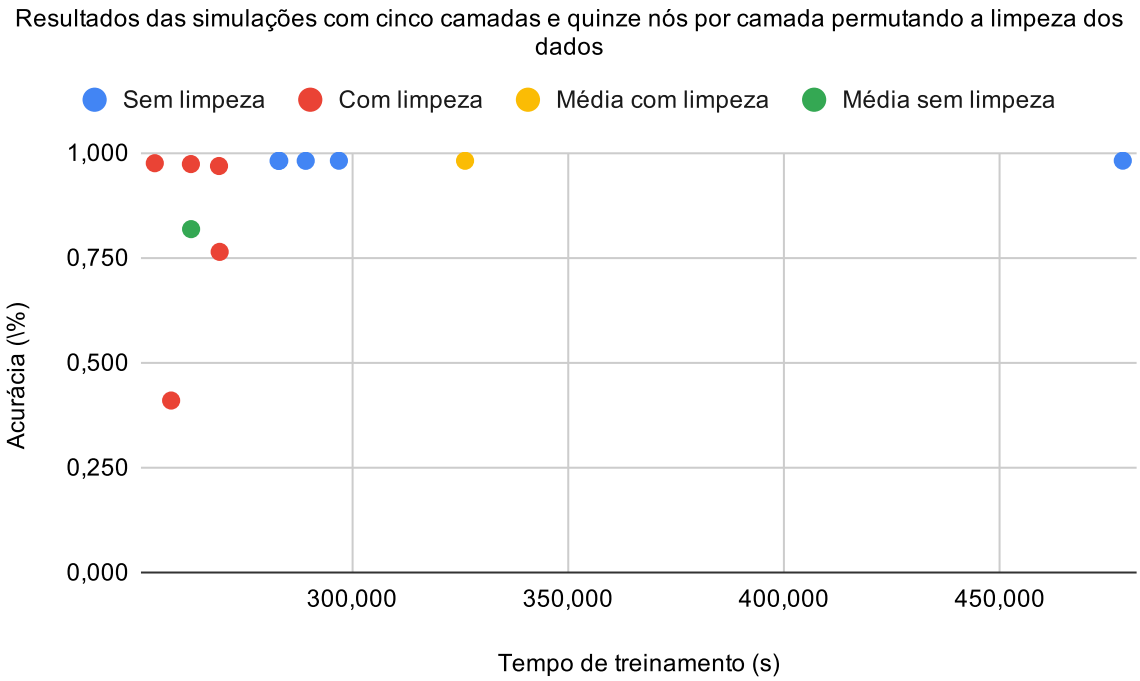


Figura 14 – Representação dos dados da tabela 3 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
5	50	Não	349,691	0,982
5	50	Não	317,546	0,417
5	50	Não	332,101	0,985
5	50	Não	307,238	0,982
5	50	Não	289,614	0,984
5	50	Sim	264,034	0,977
5	50	Sim	259,893	0,414
5	50	Sim	259,943	0,944
5	50	Sim	295,060	0,411
5	50	Sim	263,927	0,978

Tabela 4 – Resultados das simulações com cinco camadas e cinquenta nós por camada permutando a limpeza dos dados.

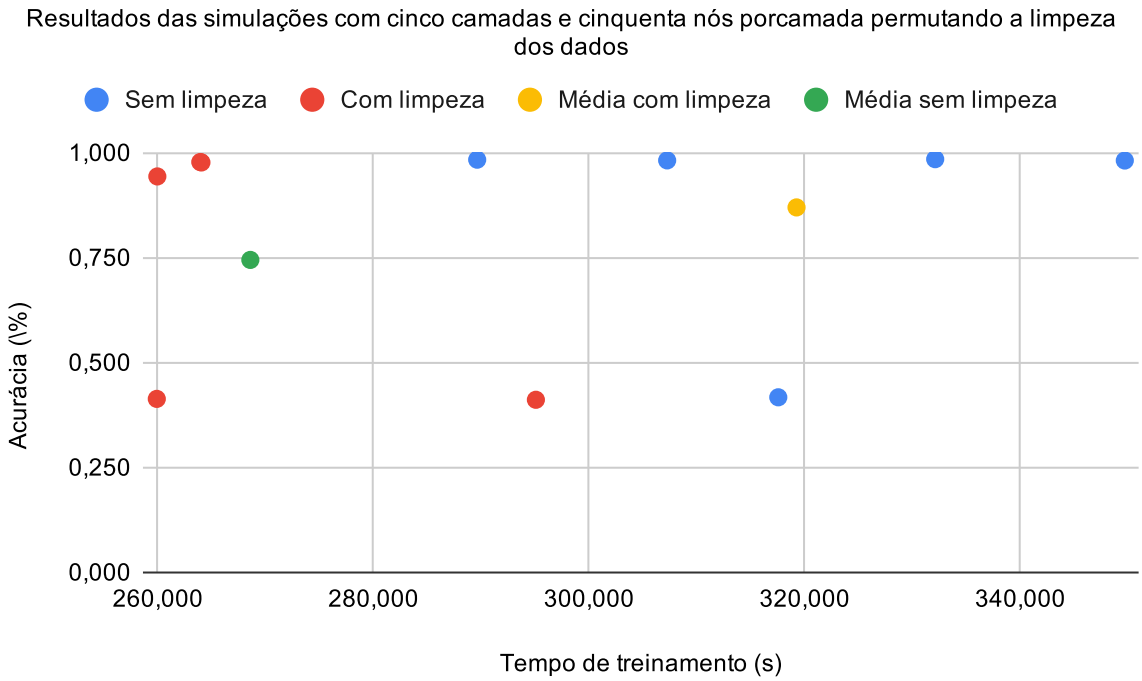


Figura 15 – Representação dos dados da tabela 4 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
10	15	Não	364,567	0,981
10	15	Não	375,376	0,981
10	15	Não	366,393	0,986
10	15	Não	362,472	0,984
10	15	Não	332,287	0,982
10	15	Sim	316,090	0,407
10	15	Sim	324,826	0,977
10	15	Sim	340,974	0,980
10	15	Sim	324,271	0,974
10	15	Sim	326,609	0,979

Tabela 5 – Resultados das simulações com dez camadas e quinze nós por camada permutando a limpeza dos dados.

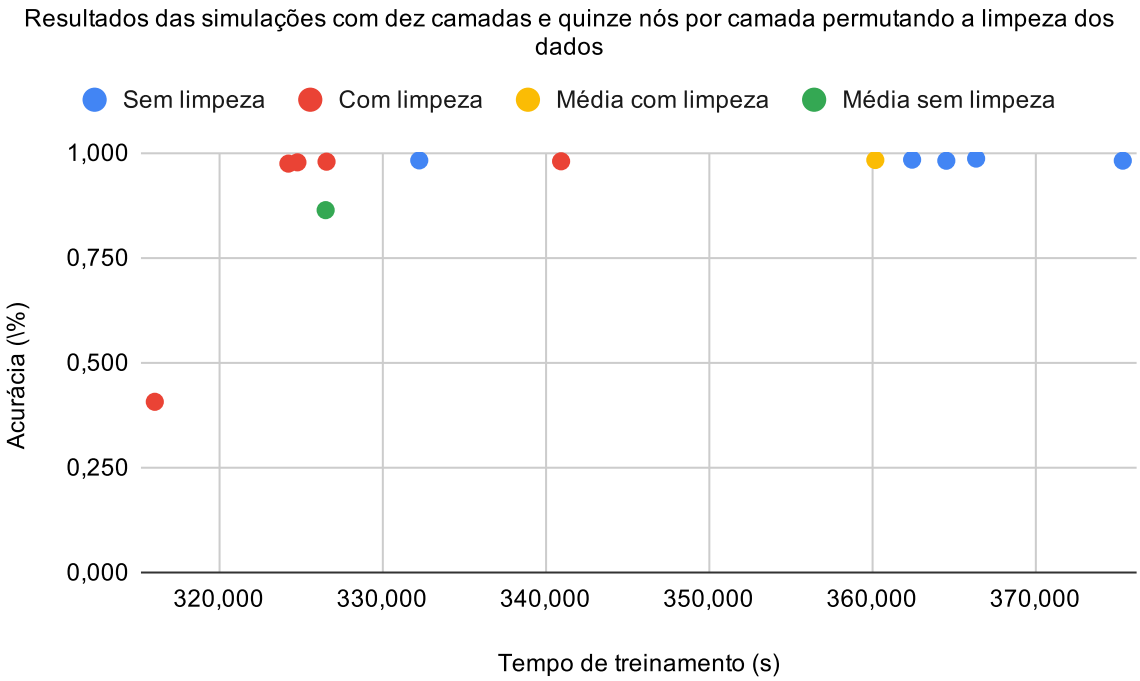


Figura 16 – Representação dos dados da tabela 5 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
10	50	Não	377,334	0,982
10	50	Não	381,859	0,984
10	50	Não	366,290	0,983
10	50	Não	361,996	0,983
10	50	Não	357,839	0,986
10	50	Sim	323,900	0,412
10	50	Sim	312,348	0,978
10	50	Sim	322,669	0,981
10	50	Sim	309,776	0,975
10	50	Sim	322,927	0,978

Tabela 6 – Resultados das simulações com dez camadas e cinquenta nós por camada permutando a limpeza dos dados.

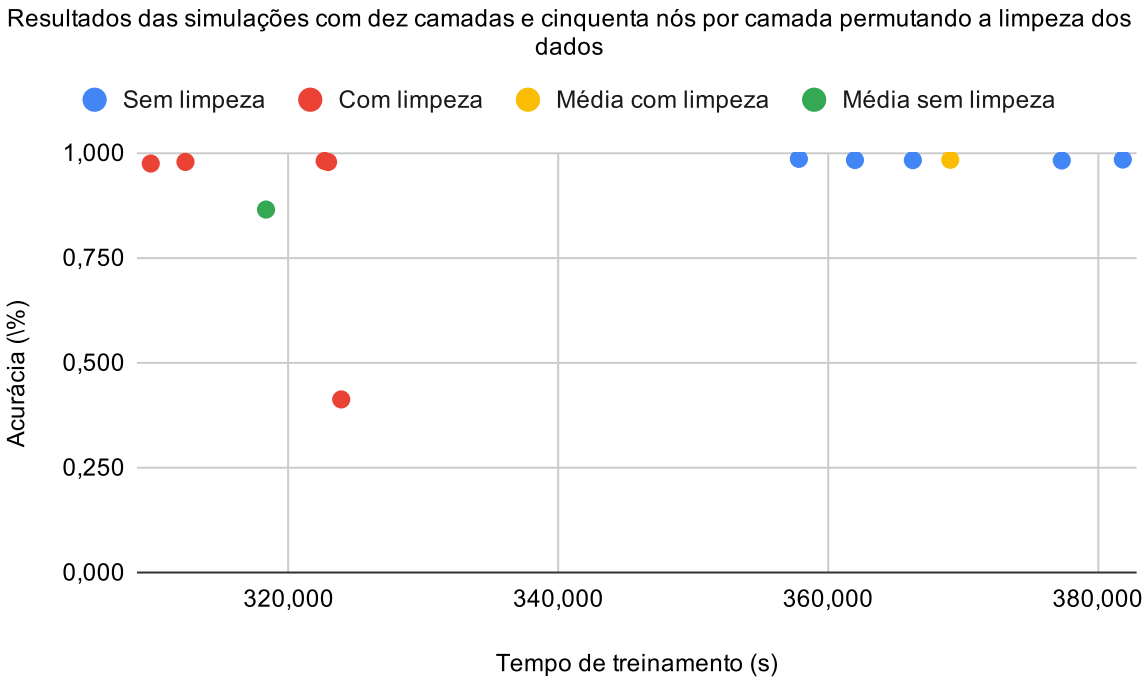


Figura 17 – Representação dos dados da tabela 6 em gráfico de dispersão

Ao observar os dados acima, é possível notar uma queda geral tanto no tempo médio de treino, quanto na acurácia dos modelos, representados nas tabelas abaixo. Isso pode ser explicado pelo processo de limpeza dos dados. Considerando-se que as configurações atuais das simulações possuem um *threshold* para exclusão das correlações de 0.5, dezenove colunas são apagadas durante o processo, representando cerca de 46% dos dados. A melhoria no tempo de treinamento decorre do fato de existirem menos dados a serem aprendidos pelo modelo. Essa diminuição no total de dados também pode gerar uma acurácia menor no modelo, mas não representa uma piora real. Ao remover colunas com valores semelhantes como 'serror_rate' e 'srv_serror_rate', por exemplo, diminuímos a possibilidade de um *overfitting* dos dados, o que nos daria uma alta acurácia que seria falsa caso acontecesse.

Tabela	Tempo Médio (NL)	Tempo Médio (L)	Diferença (%)
1	240,425	202,149	-15,920
2	259,209	222,298	-14,240
3	326,181	262,756	-19,445
4	319,238	268,572	-15,871
5	360,219	326,554	-9,346
6	369,063	318,324	-13,748

Tabela 7 – Diferença do tempo médio entre as simulações com conjunto de dados que não foram limpos e os que foram.

Tabela	Acurácia Média (NL)	Acurácia (L)	Diferença (%)
1	0,417	0,673	61,421
2	0,513	0,154	-70,012
3	0,981	0,818	-16,609
4	0,870	0,745	-14,383
5	0,983	0,863	-12,157
6	0,983	0,865	-12,059

Tabela 8 – Diferença da acurácia média entre as simulações com conjunto de dados que não foram limpos e os que foram.

O segundo grupo de comparações desdobra-se da tabela 9 à 14 e exibe os dados comparativos das simulações em que foram preservados os mesmos números de camadas das redes neurais e alterados apenas o número de nós.

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
1	15	Não	234,952	0,751
1	15	Não	246,394	0,001
1	15	Não	235,468	0,402
1	15	Não	243,304	0,732
1	15	Não	242,009	0,198
1	50	Não	257,859	0,002
1	50	Não	256,538	0,981
1	50	Não	253,082	0,788
1	50	Não	261,893	0,005
1	50	Não	266,676	0,787

Tabela 9 – Resultados das simulações com uma camada e limpeza de dados não realizada permutando o número de nós por camada.

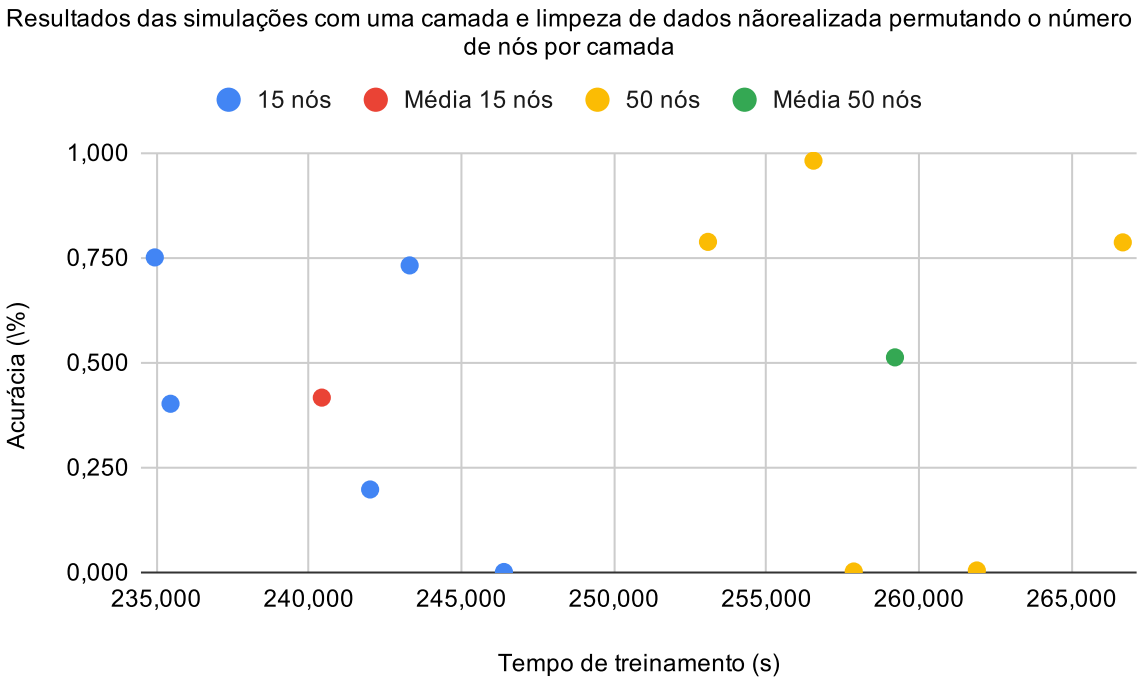


Figura 18 – Representação dos dados da tabela 9 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
1	15	Sim	195,218	0,948
1	15	Sim	203,252	0,935
1	15	Sim	198,034	0,741
1	15	Sim	190,972	0,568
1	15	Sim	223,269	0,171
1	50	Sim	220,776	0,010
1	50	Sim	205,175	0,198
1	50	Sim	204,139	0,184
1	50	Sim	234,664	0,187
1	50	Sim	246,734	0,189

Tabela 10 – Resultados das simulações com uma camada e limpeza de dados realizada permutando o número de nós por camada.

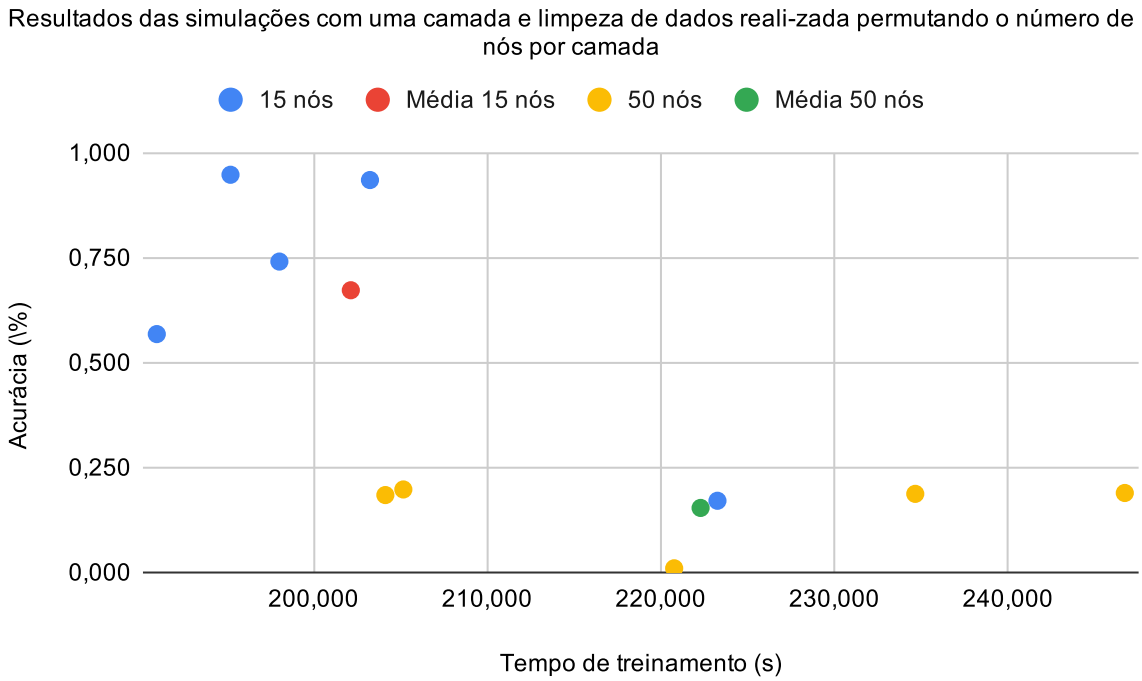


Figura 19 – Representação dos dados da tabela 10 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
5	15	Não	283,013	0,981
5	15	Não	283,164	0,981
5	15	Não	296,976	0,981
5	15	Não	478,459	0,981
5	15	Não	289,293	0,981
5	50	Não	349,691	0,982
5	50	Não	317,546	0,417
5	50	Não	332,101	0,985
5	50	Não	307,238	0,982
5	50	Não	289,614	0,984

Tabela 11 – Resultados das simulações com cinco camadas e limpeza de dados não realizada permutando o número de nós por camada.

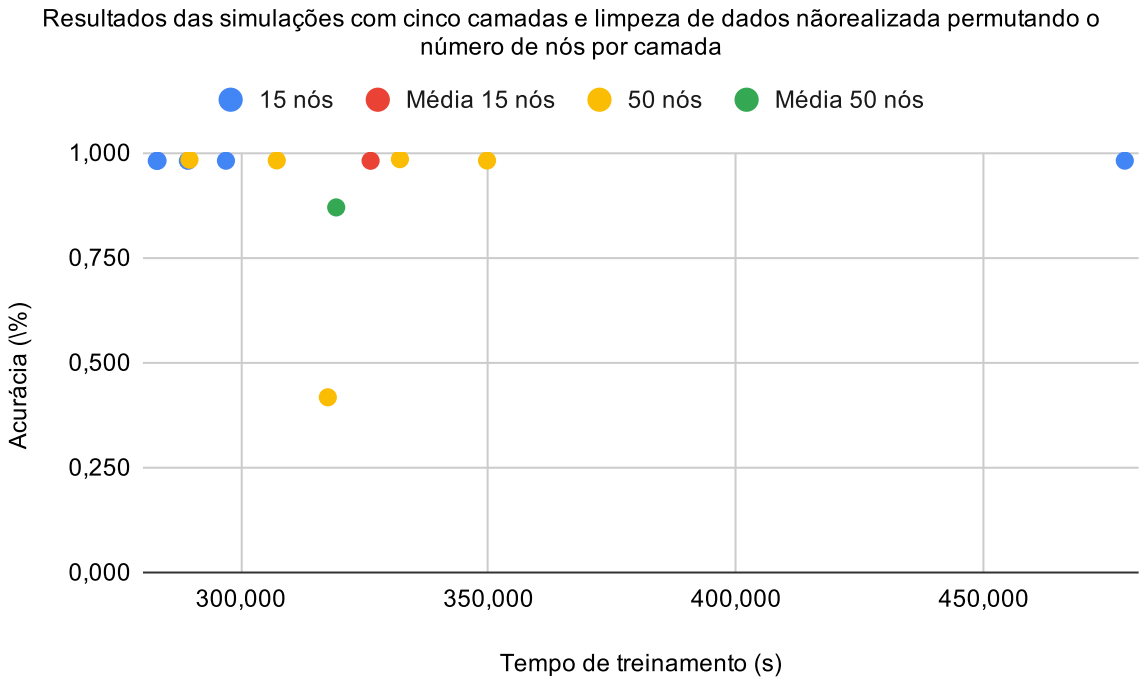


Figura 20 – Representação dos dados da tabela 11 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
5	15	Sim	254,350	0,975
5	15	Sim	269,369	0,764
5	15	Sim	262,709	0,973
5	15	Sim	258,133	0,410
5	15	Sim	269,217	0,969
5	50	Sim	264,034	0,977
5	50	Sim	259,893	0,414
5	50	Sim	259,943	0,944
5	50	Sim	295,060	0,411
5	50	Sim	263,927	0,978

Tabela 12 – Resultados das simulações com cinco camadas e limpeza de dados realizada permutando o número de nós por camada.

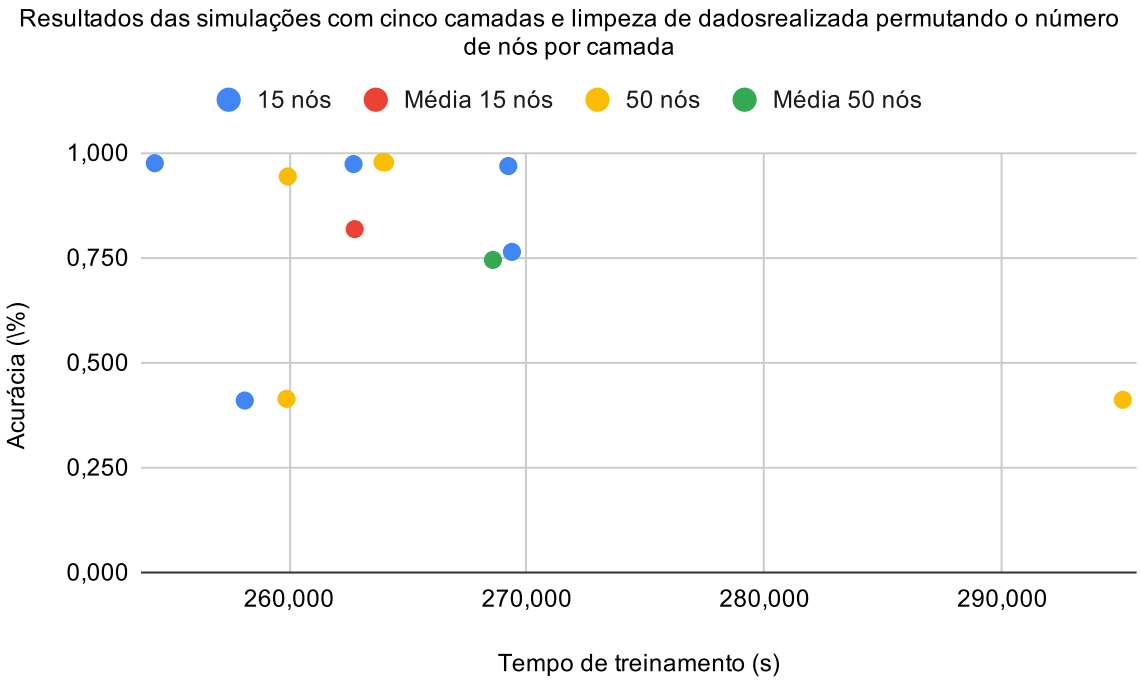


Figura 21 – Representação dos dados da tabela 12 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
10	15	Não	364,567	0,981
10	15	Não	375,376	0,981
10	15	Não	366,393	0,986
10	15	Não	362,472	0,984
10	15	Não	332,287	0,982
10	50	Não	377,334	0,982
10	50	Não	381,859	0,984
10	50	Não	366,290	0,983
10	50	Não	361,996	0,983
10	50	Não	357,839	0,986

Tabela 13 – Resultados das simulações com dez camadas e limpeza de dados não realizada permutando o número de nós por camada.

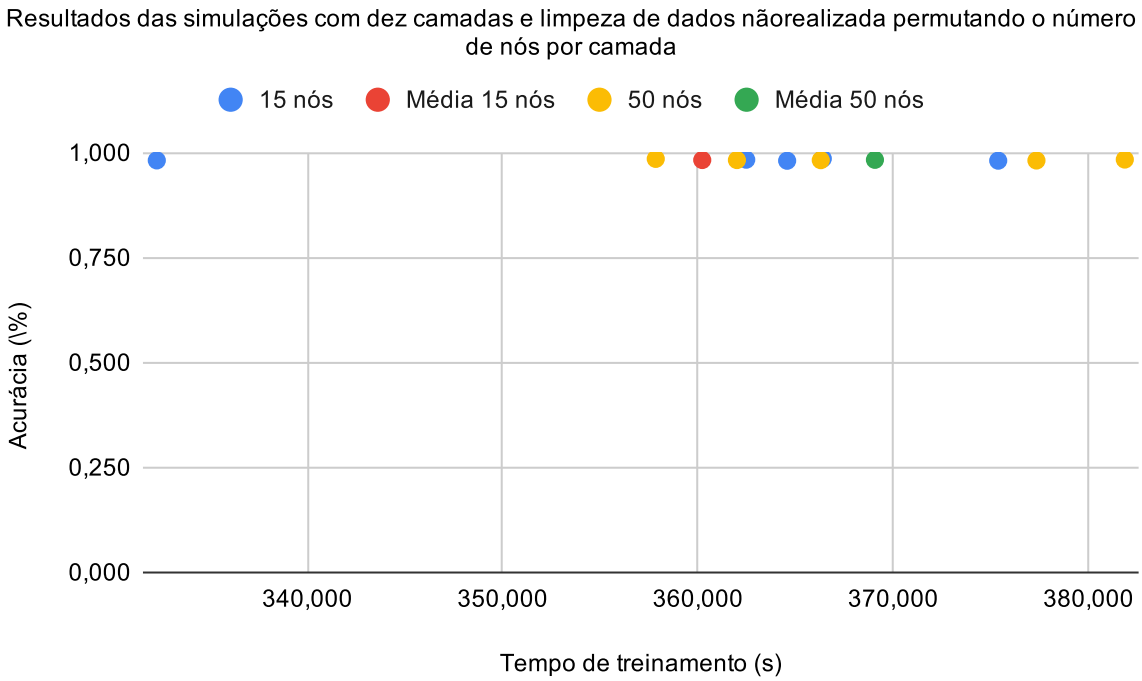


Figura 22 – Representação dos dados da tabela 13 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
10	15	Sim	316,090	0,407
10	15	Sim	324,826	0,977
10	15	Sim	340,974	0,980
10	15	Sim	324,271	0,974
10	15	Sim	326,609	0,979
10	50	Sim	323,900	0,412
10	50	Sim	312,348	0,978
10	50	Sim	322,669	0,981
10	50	Sim	309,776	0,975
10	50	Sim	322,927	0,978

Tabela 14 – Resultados das simulações com dez camadas e limpeza de dados realizada permutando o número de nós por camada.

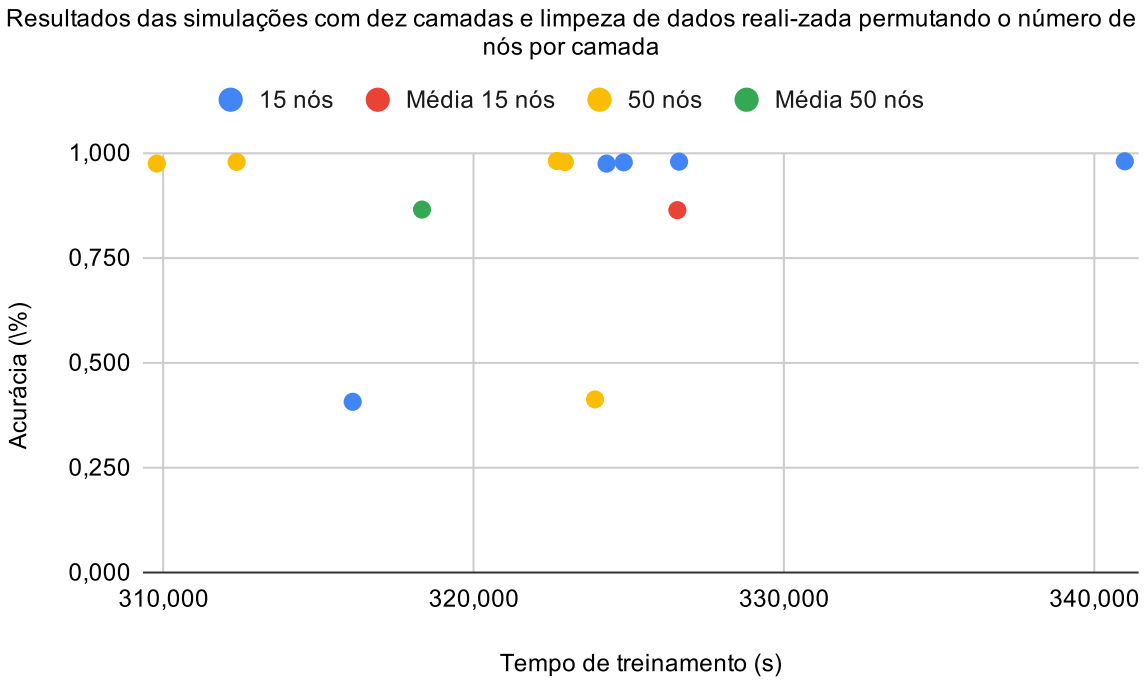


Figura 23 – Representação dos dados da tabela 14 em gráfico de dispersão

Ao observar os dados acima, mais uma vez, podemos perceber um aumento no tempo médio de treino causado pelo crescimento na complexidade do modelo. Nas tabelas 13 e 14 houve um leve aumento na acurácia devido ao crescimento da complexidade do modelo, o que traz um aumento nos parâmetros a serem treinados. É visto uma diminuição considerável na acurácia do modelo que utiliza uma camada e cinquenta nós (tabela 10), e, ao ver esse resultado, presumimos que aumentar o tempo de treino auxiliaria no aumento da acurácia, já que todos os treinos param depois de 200 épocas, como visto a seguir.

Tabela	Acurácia 15 nós	Acurácia 50 nós	Diferença (%)
9	0,417	0,513	23,020
10	0,673	0,154	-77,146
11	0,981	0,870	-11,333
12	0,818	0,745	-8,966
13	0,983	0,983	0,046
14	0,863	0,865	0,158

Tabela 15 – Diferença da acurácia média entre as simulações com mesmo número de camadas e permutação do número de nós

Tabela	Tempo 15 nós	Tempo 50 nós	Diferença (%)
9	240,425	259,209	7,813
10	202,149	222,298	9,967
11	326,181	319,238	-2,129
12	262,756	268,572	2,213
13	360,219	369,063	2,455
14	326,554	318,324	-2,520

Tabela 16 – Diferença do tempo médio entre as simulações com mesmo número de camadas e permutação do número de nós

Pensando na baixa performance visto na tabela 10, foi realizada uma simulação com 500 épocas e os resultados foram os mesmos, porém isso se deu pela baixa profundidade desse modelo, isto é, o baixo número de camadas utilizadas. A importância do número de camadas utilizadas para o treinamento será abordada no próximo grupo de comparações.

Por fim, o último grupo comparativo criado, visto da tabela 17 à tabela 20, retrata os dados onde o número de camadas foi alterado.

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
1	15	Não	234,952	0,751
1	15	Não	246,394	0,001
1	15	Não	235,468	0,402
1	15	Não	243,304	0,732
1	15	Não	242,009	0,198
5	15	Não	283,013	0,981
5	15	Não	283,164	0,981
5	15	Não	296,976	0,981
5	15	Não	478,459	0,981
5	15	Não	289,293	0,981
10	15	Não	364,567	0,981
10	15	Não	375,376	0,981
10	15	Não	366,393	0,986
10	15	Não	362,472	0,984
10	15	Não	332,287	0,982

Tabela 17 – Resultados das simulações com quinze nós por camada e sem a realização da limpeza de dados permutando o número de camadas.

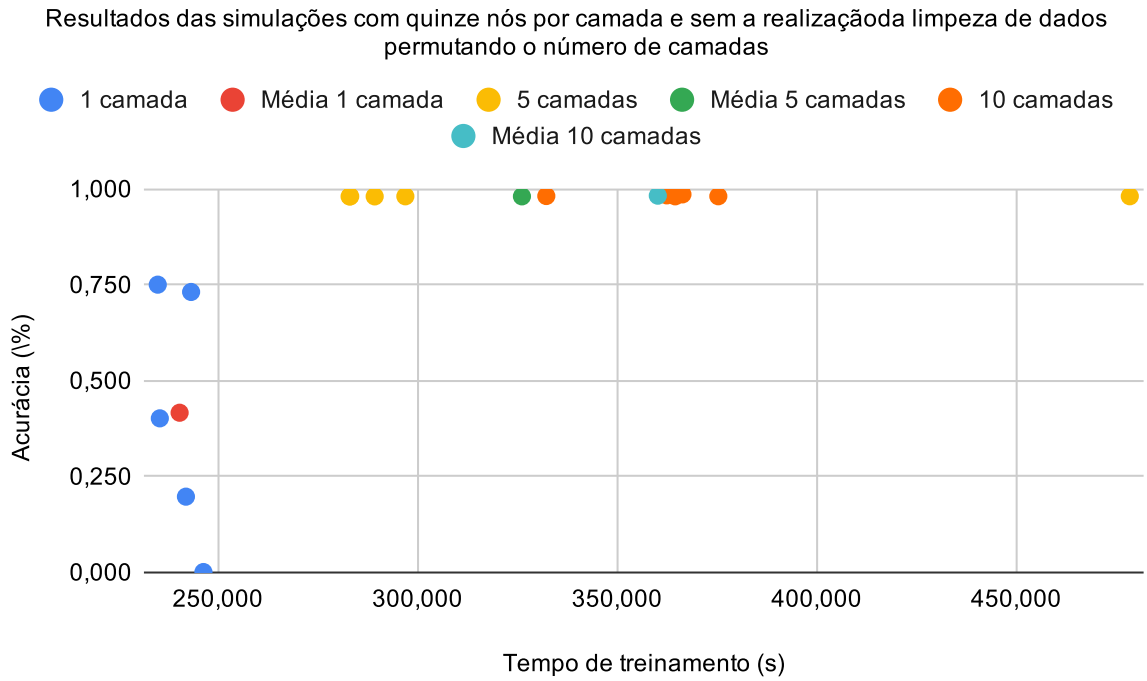


Figura 24 – Representação dos dados da tabela 17 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
1	15	Sim	195,218	0,948
1	15	Sim	203,252	0,935
1	15	Sim	198,034	0,741
1	15	Sim	190,972	0,568
1	15	Sim	223,269	0,171
5	15	Sim	254,350	0,975
5	15	Sim	269,369	0,764
5	15	Sim	262,709	0,973
5	15	Sim	258,133	0,410
5	15	Sim	269,217	0,969
10	15	Sim	316,090	0,407
10	15	Sim	324,826	0,977
10	15	Sim	340,974	0,980
10	15	Sim	324,271	0,974
10	15	Sim	326,609	0,979

Tabela 18 – Resultados das simulações com quinze nós por camada e com a realização da limpeza de dados permutando o número de camadas.

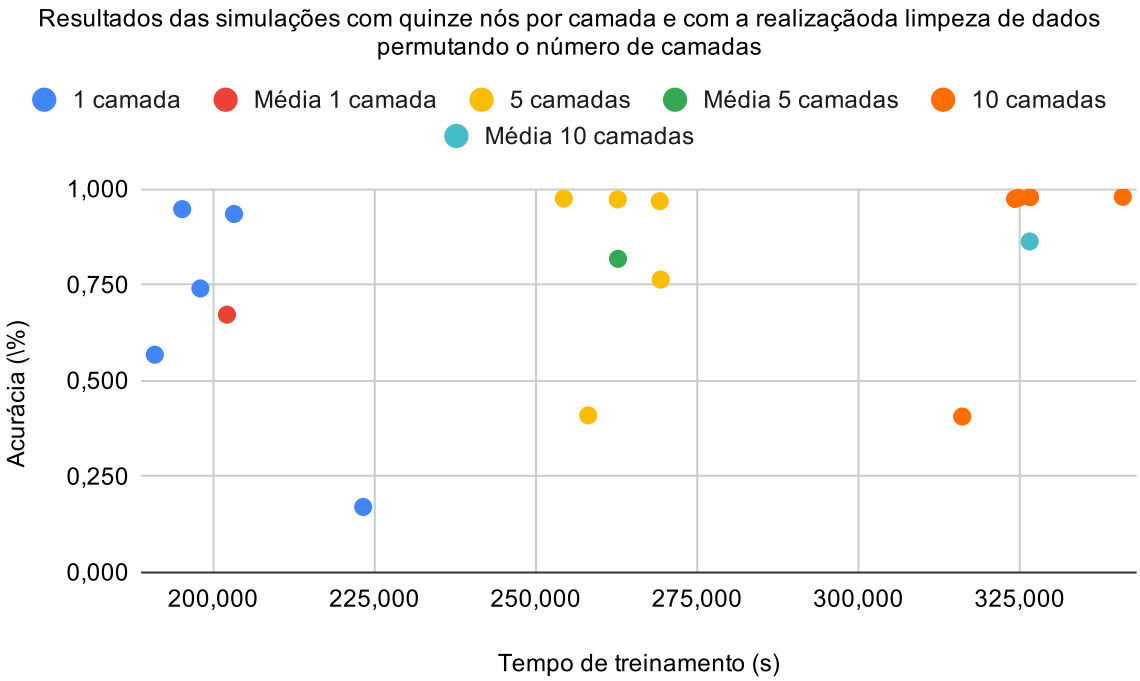


Figura 25 – Representação dos dados da tabela 18 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
1	50	Não	257,859	0,002
1	50	Não	256,538	0,981
1	50	Não	253,082	0,788
1	50	Não	261,893	0,005
1	50	Não	266,676	0,787
5	50	Não	349,691	0,982
5	50	Não	317,546	0,417
5	50	Não	332,101	0,985
5	50	Não	307,238	0,982
5	50	Não	289,614	0,984
10	50	Não	377,334	0,982
10	50	Não	381,859	0,984
10	50	Não	366,290	0,983
10	50	Não	361,996	0,983
10	50	Não	357,839	0,986

Tabela 19 – Resultados das simulações com cinquenta nós por camada e sem a realização da limpeza de dados permutando o número de camadas.

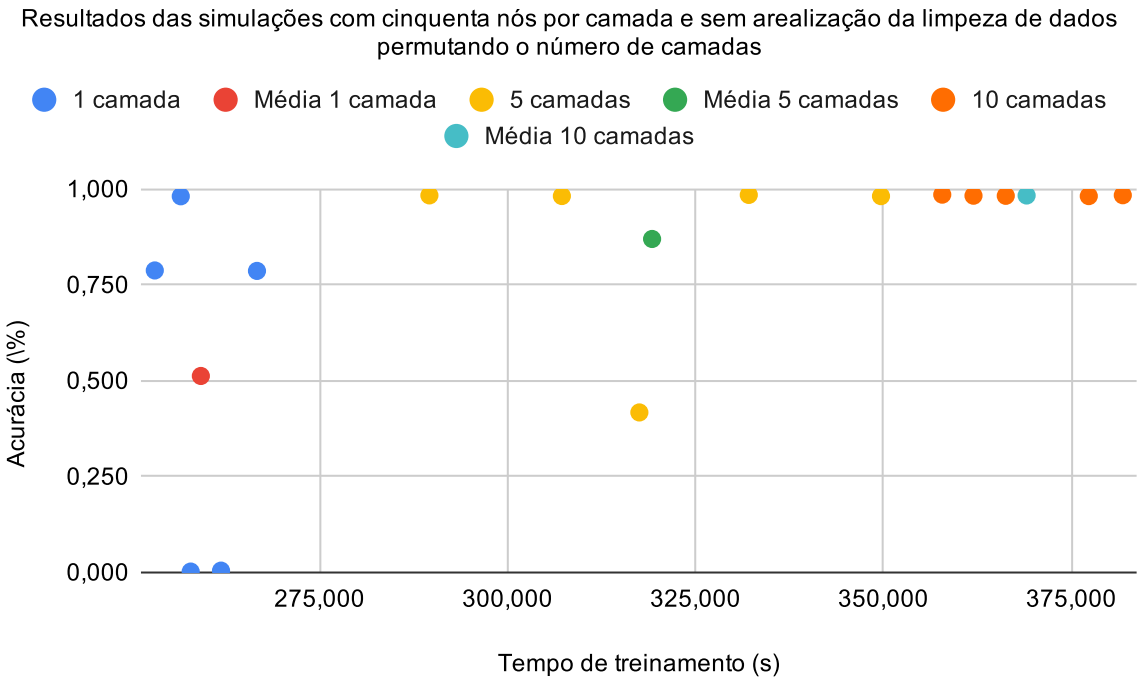


Figura 26 – Representação dos dados da tabela 19 em gráfico de dispersão

Camadas	Nós	Limpeza realizada	Tempo de treino (s)	Acurácia
1	50	Sim	220,776	0,010
1	50	Sim	205,175	0,198
1	50	Sim	204,139	0,184
1	50	Sim	234,664	0,187
1	50	Sim	246,734	0,189
5	50	Sim	264,034	0,977
5	50	Sim	259,893	0,414
5	50	Sim	259,943	0,944
5	50	Sim	295,060	0,411
5	50	Sim	263,927	0,978
10	50	Sim	323,900	0,412
10	50	Sim	312,348	0,978
10	50	Sim	322,669	0,981
10	50	Sim	309,776	0,975
10	50	Sim	322,927	0,978

Tabela 20 – Resultados das simulações com cinquenta nós por camada e com a realização da limpeza de dados permutando o número de camadas.

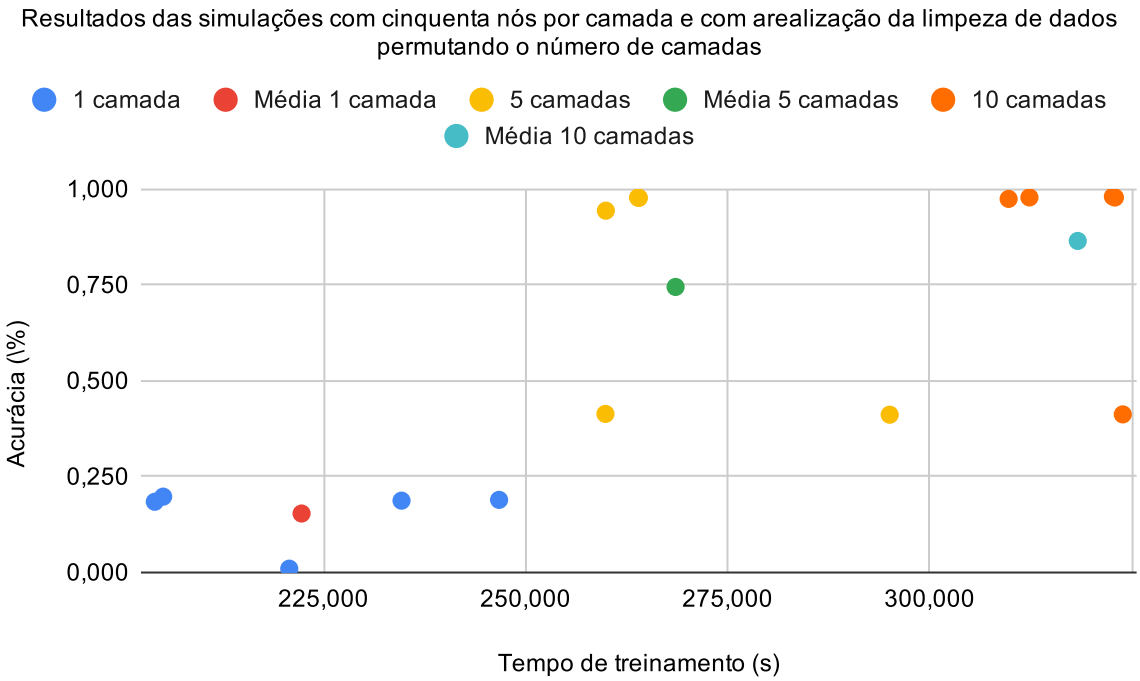


Figura 27 – Representação dos dados da tabela 20 em gráfico de dispersão

Assim como nos últimos testes realizados aqui também é possível notar um aumento no tempo médio de treino dos modelos, além disso há um aumento considerável na acurácia média dos modelos. Cada camada em uma rede neural está tentando, de certa forma, aprender um aspecto diferente dos dados. Se pensarmos, por exemplo, no contexto de visão computacional em um modelo que está aprendendo o que é um carro, cada uma de suas camadas irá identificar aspectos variados do objeto. A primeiro camada pode identificar os limites físicos do carro, as segunda rodas e assim por diante. Por isso, o acréscimo de camadas em uma rede neural gera uma melhora perceptível da acurácia quando possuindo nós suficientes para os cálculos.

Tabela	Acurácia - 1 Coluna	Acurácia - 5 Colunas	Acurácia - 10 Colunas
17	0,417	0,981	0,983
18	0,673	0,818	0,863
19	0,513	0,870	0,983
20	0,154	0,745	0,865

Tabela 21 – Acurácia média das simulações onde foi alterado somente o número de camadas

Tabela	De 1 para 5 (%)	De 5 para 10 (%)	De 1 para 10 (%)
17	135,492	0,184	135,924
18	21,656	5,533	28,387
19	69,731	13,041	91,866
20	384,594	16,110	462,661

Tabela 22 – Comparação da acurácia nas simulações onde foi alterado somente o número de camadas

Tabela	Tempo (s) - 1 Coluna	Tempo (s) - 5 Colunas	Tempo (s) - 10 Colunas
17	240,425	326,181	360,219
18	202,149	262,756	326,554
19	259,209	319,238	369,063
20	222,298	268,572	318,324

Tabela 23 – Tempo médio das simulações onde foi alterado somente o número de camadas

Tabela	De 1 para 5 (%)	De 5 para 10 (%)	De 1 para 10 (%)
17	35,668	10,435	49,826
18	29,981	24,280	61,541
19	23,158	15,608	42,380
20	20,816	18,525	43,197

Tabela 24 – Comparação do tempo nas simulações onde foi alterado somente o número de camadas

4 Conclusão

Ao iniciar o desenvolvimento dessa pesquisa, era esperado um aumento da acurácia dos modelos treinados conforme o aumento de sua complexidade e da limpeza dos dados, e realmente foi verificada uma melhora geral. Porém, essa melhora se deu no melhor cenário possível, no qual o modelo tinha o maior número de nós e camadas de todos os testes e os dados não estavam limpos, o que pode gerar um vício do modelo treinado, causando falsos positivos. E também, por mais que o treino com o conjunto de dados limpos tenha tido uma menor acurácia pela medição da própria rede neural, é possível que ele tenha um desempenho melhor. Foi possível verificar também que o treino com poucas camadas e poucos nós, não geram bons resultados. Com isso concluímos que o treinamento de uma rede neural mais complexa gera um resultado melhor e mesmo que a limpeza de dados traga pioras aparentes, sua realização, além de diminuir o tempo de processamento, pode ajudar a evitar ~~um~~ *overfitting*.

Esperamos que esse trabalho agregue conhecimentos para iniciantes do estudo de inteligência artificial e acreditamos que estudos futuros possam fazer proveito da velocidade das novas gerações de computadores, aumentando a complexidade dos modelos, para um ganho de velocidade nos treinos e testes. Ainda sobre isso, é possível utilizar o código-fonte como ponto de partida para melhores implementações, utilizando bibliotecas que talvez surjam, e também automatizando os testes. Também é possível explorar mais resultados ao substituir as funções de ativação e utilizar redes que não possuam as camadas uniformes.

Referências

BOSLAUGH, S. *Statistics in a Nutshell: A Desktop Quick Reference*. 2. ed. Sebastopol, California: O'Reilly Media, 2012. 595 p. Citado 2 vezes nas páginas 17 e 18.

CHIO, C.; FREEMAN, D. *Machine Learning and Security: Protecting Systems with Data and Algorithms*. 1. ed. Sebastopol, California: O'Reilly Media, 2018. 386 p. Citado na página 18.

GOOGLE DEVELOPERS. *Training and Test Sets: Splitting Data*. 2020. Disponível em: <https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data>. Acesso em: 02 aug. 2020. Citado na página 25.

INDIANA UNIVERSITY. *What is the difference between a compiled and an interpreted program?* 2018. Disponível em: <https://kb.iu.edu/d/agsz>. Acesso em: 28 jul. 2019. Citado na página 14.

JUPYTER TEAM. *The Jupyter Notebook Documentation*. 2020. Disponível em: <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>. Acesso em: 01 aug. 2020. Citado na página 17.

KERAS TEAM. *Keras Documentation*. 2019. Disponível em: <https://keras.io/>. Acesso em: 05 set. 2019. Citado na página 16.

KETKAR, N. *Deep Learning with Python: A Hands-on Introduction*. 1. ed. Karnataka, India: Apress, 2017. 160 p. Citado na página 20.

KIM NARA SHIN, S. Y. J. J.; KIM, S. H. Method of intrusion detection using deep neural network. *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, p. 313–316, 2017. Citado na página 12.

KLUYVER, T. et al. Jupyter notebooks—a publishing format for reproducible computational workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, v. 1, n. 20, p. 87–90, 2016. Citado na página 17.

LENT, R. *Cem bilhões de neurônios conceitos fundamentais de neurociências*. 2. ed. São Paulo, São Paulo: Editora Atheneu, 2010. 786 p. Citado na página 19.

MCKINNEY, W. *Python for Data Analysis*. 1. ed. Sebastopol, California: O'Reilly Media, 2012. 466 p. ISBN 978-1-449-31979-3. Citado na página 16.

MEDIUM. *Why Data Normalization is necessary for Machine Learning models*. 2018. Disponível em: <https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029>. Acesso em: 11 set. 2019. Citado na página 18.

MOZILLA FOUNDATION. *High-level programming language*. 2019. Disponível em: <https://developer.mozilla.org/en-US/docs/Glossary/High-level_programming_language>. Acesso em: 28 jul. 2019. Citado na página 14.

MUKAKA, M. M. A guide to appropriate use of correlation coefficient in medical research. *Malawi Medical Journal*, v. 3, n. 24, p. 69–71, 2012. Citado na página 18.

OSBORNE, J. W. Data cleaning basics: Best practices in dealing with extreme scores. *Newborn and Infant Nursing Reviews*, v. 1, n. 10, p. 37–43, 2010. Citado 2 vezes nas páginas 17 e 18.

PATTERSON, J.; GIBSON, A. *Deep Learning: A PRACTITIONER'S APPROACH*. 2. ed. Sebastopol, California: O'Reilly Media, 2017. 532 p. Citado 4 vezes nas páginas 8, 20, 25 e 26.

PYTHON SOFTWARE FOUNDATION. *General Python FAQ*. 2019. Disponível em: <https://docs.python.org/3/faq/general.html>. Acesso em: 1 abr. 2019. Citado na página 14.

PYTHON SOFTWARE FOUNDATION. *Python Package Index (PyPI)*. 2019. Disponível em: <https://pypi.org/>. Acesso em: 1 abr. 2019. Citado na página 15.

SOLA, J.; SEVILLA, J. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science*, v. 44, n. 3, p. 1464–1468, 1997. Citado 3 vezes nas páginas 12, 17 e 18.

STACK EXCHANGE, INC. *The Incredible Growth of Python*. 2017. Disponível em: <https://stackoverflow.blog/2017/09/06/incredible-growth-python/>. Acesso em: 1 set. 2019. Citado 2 vezes nas páginas 8 e 15.

STACK EXCHANGE, INC. *Developer Survey Results 2019*. 2019. Disponível em: <https://insights.stackoverflow.com/survey/2019#most-loved-dreaded-and-wanted>. Acesso em: 1 set. 2019. Citado na página 15.

SWINSCOW, T. D. V. *Statistics at Square One*. 9. ed. Sebastopol, California: BMJ Publishing Group, 1997. 106 p. Citado na página 18.

TENSORFLOW TEAM. *Repositório do Tensorflow*. 2019. Disponível em: <https://github.com/tensorflow/tensorflow>. Acesso em: 05 set. 2019. Citado na página 16.

THE ECONOMIST GROUP. *Python has brought computer programming to a vast new audience*. 2018. Disponível em: <https://www.economist.com/science-and-technology/2018/07/19/python-has-brought-computer-programming-to-a-vast-new-audience>. Acesso em: 1 set. 2019. Citado na página 15.

THE ECONOMIST GROUP. *Python is becoming the world's most popular coding language*. 2018. Disponível em: <https://www.economist.com/graphic-detail/2018/07/26/python-is-becoming-the-worlds-most-popular-coding-language>. Acesso em: 1 set. 2019. Citado na página 15.

WEISFELD, M. *The Object-Oriented Thought Process*. 3. ed. Sebastopol, California: Addison-Wesley Professional, 2008. 360 p. ISBN 978-0672330162. Citado na página 15.

ANEXO A – Fluxograma do Código