UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

JOÃO VICENTE FATUR LESSA

Explorando Redes Neurais em Planos de Dados Programáveis para Classificação de Tráfego

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Luciano Paschoal Gaspary
Co-orientador: Prof. Dr. Jonatas Adilson Marques

Porto Alegre
2022

AGRADECIMENTOS

Primeiramente, obrigado ao Criador, pela luz em minha vida e ao meu redor.

Agradeço à minha Família, que me provém apoio, incentivo e amor incondicional. À minha mãe Nina, por estar sempre presente e me estimular a buscar a felicidade. Ao meu pai João Régis, pela referência em caráter e cultura. À minha Tia Bibiana, que me hospedou com muito carinho durante os meses de elaboração deste trabalho. Agradeço aos meus amigos, que estiveram ao meu lado, e me trouxeram companheirismo e diversão no longo caminho da faculdade.

Expresso minha gratidão ao meu orientador Luciano, por me inserir neste projeto e conduzir minha trajetória com boa vontade e maestria. Gratidão também ao meu co-orientador Jonatas, por ofertar tempo à me ensinar de maneira tão gentil. Agradeço aos demais professores do corpo docente da UFRGS, por desafiarem meus limites, elevarem minhas capacidades e tornarem mais rico o meu conhecimento.

RESUMO

A identificação precisa do tráfego de redes é essencial para prover boa qualidade em diversos serviços. Para a classificação ser eficaz, levando em conta as elevadas taxas de transmissão de dados, seria conveniente executar a atividade de classificação nos próprios dispositivos de encaminhamento de pacotes. Uma linha de ação possível é a exploração do paradigma de planos de dados programáveis. Contudo, dispositivos programáveis impõem limitações significativas em relação à memória e à capacidade de processamento. Neste trabalho, realiza-se o projeto, a implementação e a avaliação de modelos de rede neurais, capazes de alcançar alta precisão de classificação e, ao mesmo tempo, viáveis de serem acomodados nos dispositivos alvos. Primeiramente, emprega-se a biblioteca bem difundida de aprendizado de máquina *Tensorflow* para obter-se modelos de RNAs treinados. Após, mapeia-se a modelagem usada para uma rede de *switches* virtuais com topologia equivalente, onde reproduz-se o processamento dos neurônios nos comutadores. Utiliza-se os pesos e *biases* provenientes do treinamento da rede neural para configurar os *switches*. Instancia-se uma réplica do modelo da RNA, operante em um plano de dados programável, apta a prover classificações idênticas às previstas pela RNA do *Tensorflow*.

Palavras-chave: Redes de Computadores. Planos de Dados Programáveis. Redes Neurais Artificiais.

Exploring Neural Networks in Programmable Data Planes for Traffic Classification

ABSTRACT

Accurate identification of network traffic is essential to provide good quality in various services. For classification to be effective, taking into account the high data transmission rates, it would be convenient to perform the classification activity on the packet forwarding devices themselves. One possible course of action is to explore the programmable data plane paradigm. However, programmable devices impose significant limitations on memory and processing power. In this work, the design, implementation and evaluation of neural network models are accomplished, capable of achieving high classification accuracy and, at the same time, viable to be executed in the target devices. First, the well-known machine learning library Tensorflow is used to obtain trained ANNs models. Afterwards, the modeling utilized is mapped to a network of virtual switches with equivalent topology, where the processing of neurons in the switches is reproduced. The weights and biases from the training of the neural network are applied to configure the switches. A replica of the ANN model is instantiated, operating in a programmable data plane, able to provide classifications identical to those predicted by the Tensorflow ANN.

Keywords: computer networks, programmable data planes, articial neural networks.

LISTA DE ABREVIATURAS E SIGLAS

IP	Internet Protocol
ML	Machine Learning
P4	Programming Protocol-independent Packet Processors
ANN	Artificial Neural Network
CPU	Central Processing Unit
FIN	Finalization
FTP	File Transfer Protocol
GPU	Graphics Processing Unit
NPU	Neural Processing Unit
RNA	Rede Neural Artifical
MAC	Media Access Control Address
MSS	Maximum Segment Size
P2P	Peer-to-Peer
RMT	Reprogrammable Match-action Tables
SDN	Software defined Networking
SYN	Synchronize
TCP	Transmission Control Protocol
URG	Urgent
WWW	World Wide Web
ASIC	Application-Specific Integrated Circuit
FPGA	Field Programmable Gate Array
RELU	Rectified Linear Unit
SACK	Selective Acknowledgments

LISTA DE SÍMBOLOS

$\sum$	Somatório
σ	Desvio padrão da normalização
μ	Variância da normalização
f_A	Função de ativação
α	Taxa de aprendizagem
δ	Erro associado ao neurônio

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Redes de comunicação têm se tornado cada vez mais imprescindíveis, acomodando um universo crescente de dispositivos, serviços e aplicações. Por existirem diversas finalidades para aplicações em redes, diferentes requisitos de funcionamento precisam ser satisfeitos para atender as necessidades de cada uma dessas aplicações. Nas áreas da telemedicina, transporte aéreo, naval e terrestre, *virtual trading* e muitas outras, o correto funcionamento da rede é crucial. Em razão disso, são impostos sobre as redes requisitos estritos de funcionamento em relação a desempenho, segurança, disponibilidade e confiança. Isso difere de aplicações não críticas, com propósito de entreter ou informar, tal como o *streaming* de vídeos ou música.

Para atender os requisitos supracitados, são indispensáveis mecanismos variados de gerenciamento de redes. Por exemplo, balanceamento de carga (de tráfego, de acesso a serviços, etc.) pode otimizar o tempo de resposta e evitar a sobrecarga de alguns nós enquanto outros ficam ociosos. Mecanismos de priorização constituem, nesse caso, exemplos de solução. Como outro exemplo, a detecção de tentativas de intrusão são cruciais para evitar ataques contra a rede e suas aplicações que levem a exposição de dados sigilosos, a negação de serviço, entre outros problemas. Nesses contextos, a *classificação de tráfego* contribui para a concretização dos mecanismos mencionados. Utilizando de metadados sobre pacotes e fluxos, é possível separá-los em classes de tráfego, como, por exemplo, *web*, *email*, *streaming*, e malicioso (MOORE; ZUEV, 2005). Assim, cada classe de tráfego resultante pode ser tratada de forma a diferenciar o serviço (para os geradores ou consumidores de dados).

Redes Neurais Artificiais (RNAs) têm se mostrado particularmente adequadas para essa tarefa de classificação (FACELI et al., 2021). O funcionamento sofisticado das RNAs as torna boas candidatas para resolver os problemas de classificação de tráfego (MICHAEL et al., 2017), isso em função, por exemplo, de sua habilidade de extrair padrões não lineares e dependências não muito claras entre um grande número de características que descrevem os objetos de estudo.

Para ser eficaz no contexto de redes de comunicação, a classificação de tráfego deve ser executada muito rapidamente e, idealmente, na granularidade de pacotes. As redes têm operado a taxas de transmissão muito elevadas, na ordem de centenas de *gigabits* por segundo ou até mesmo *terabits* por segundo em centros de dados e provedores de telecomunicação. Essas velocidades tornam impraticável a classificação de tráfego em

ambientes tradicionais de computação, em que os dados seriam coletados e analisados por uma entidade centralizada, baseada em CPU ou GPU, executando em nível de aplicação (SAQUETTI et al., 2021).

Em virtude das necessidades recém mencionadas, seria conveniente executar a atividade de classificação no ponto de entrada dos dados na rede, em um dispositivo de borda. Encontrar alternativas que permitam acelerar processos de classificação de tráfego em redes é fundamental, pois apoiariam de forma sustentável as tarefas como as citadas anteriormente (levando-se em conta as taxas de transmissão de dados elevadas).

Uma linha de ação possível é a exploração do novo paradigma de planos de dados programáveis como forma de acomodar mecanismos de classificação. Uma arquitetura projetada para tal finalidade chama-se *Reconfigurable Match+action Tables* (RMT) (BOSSHART et al., 2013). Essa arquitetura provê o suporte demandado dos dispositivos de rede a fim de poderem ser programados, por exemplo, via linguagem de programação P4 (BOSSHART et al., 2014). Com essas duas "ferramentas", torna-se viável a programação do plano de encaminhamento de pacotes em roteadores, *switches* e NetFPGAs, para o suporte a protocolos e serviços personalizados às necessidades específicas de cada rede.

Buscando explorar a linha de ação supramencionada, o objetivo deste trabalho de conclusão de curso é projetar, implementar e avaliar um mecanismo de classificação de tráfego baseado em RNAs que possa ser acomodado em dispositivos de rede programáveis. Almeja-se: (a) demonstrar a viabilidade técnica do mecanismo, (b) caracterizar as limitações desse espaço de *design*, e (c) comparar resultados que podem ser obtidos com RNAs em planos de dados programáveis com aqueles observados em ambientes tradicionais. Para isso, utiliza-se diferentes métricas, tais como desempenho, acurácia e uso de recursos.

O restante do documento está organizado da seguinte forma. Na Seção 2, apresenta-se os fundamentos que embasam este trabalho. Na Seção 3, descreve-se a proposta de trabalho, detalhando um conjunto de etapas. Na Seção 4, reporta-se os resultados dos passos realizados na implementação da RNA. Na Seção 5, apresenta-se conclusões sobre os resultados obtidos e discute-se possíveis aprimoramentos futuros.

2 FUNDAMENTOS

Nesta seção são apresentados os *principais* fundamentos que embasam o desenvolvimento deste trabalho de conclusão de curso, iniciando pela classificação de tráfego por meio de redes neurais artificiais e concluindo com redes programáveis.

2.1 Redes Neurais Artificiais

A classificação de tráfego de rede é um módulo básico na área de gerenciamento e segurança de redes. Este desempenha um papel muito significativo em múltiplas atividades, como qualidade de serviço, contabilização de rede, detecção e prevenção de intrusão. Assim, nas últimas décadas, muitos pesquisadores têm dedicado atenção a esse problema usando mecanismos baseados nas especificações de protocolos, tais como o método do número de porta, inspeção de carga útil de pacotes e análise de comportamento do hospedeiro. No entanto, com o surgimento de novas aplicações que não têm um número de porta fixo e formato bem documentado, os métodos tradicionais encontram cada vez mais dificuldades na tarefa de classificação de tráfego. Nessa condição, pesquisas recentes tentam aplicar técnicas de aprendizagem de máquina (ou *Machine Learning – ML*) para extrair padrões únicos das caraterísticas de datagramas de protocolos específicos. Esses padrões podem ser, então, usados para criar um classificador de tráfego. Vários métodos de aprendizado supervisionados e não supervisionados têm sido utilizados nos últimos anos, alcançando alta precisão para diversos protocolos de rede (LI et al., 2018).

Embora os métodos de aprendizado de máquina tenham trazido uma grande contribuição para a classificação de tráfego, ainda existem desafios não resolvidos. Um deles é que o crescimento acelerado do número de aplicações de rede acaba resultando na proposição de mais protocolos e, assim, torna-se difícil, ou até mesmo impossível, encontrar uma série de caraterísticas que podem funcionar para todos eles. Outro desafio é que as aplicações modernas são tão complexas que as técnicas tradicionais de mineração não conseguem extrair o padrão de protocolo específico com precisão. Nesse contexto, dentre os mecanismos de ML, destacam-se as redes neurais artificiais pela sua capacidade de lidar com os problemas descritos acima de forma eficaz e eficiente.

Na busca pela construção de máquinas inteligentes, um modelo que serve de inspiração é o cérebro humano. Em nosso cotidiano, realizamos diversas tarefas que requerem atenção a diferentes eventos ao mesmo tempo e o processamento de informações variadas,

de forma a tomarmos decisões adequadas. Tarefas consideradas simples, como pegar um objeto ou andar, envolvem diversos componentes biológicos, tais como memória, aprendizado e coordenação física. A complexidade de tais ações habituais é evidenciada pela dificuldade de ensinar robôs a realizá-las. A partir dessas motivações, o desenvolvimentismo das Redes Neurais Artificias (RNAs) tomou como inspiração a estrutura e o funcionamento do sistema nervoso, com o objetivo de simular a capacidade de aprendizado do cérebro humano na aquisição de conhecimento (FACELI et al., 2021).

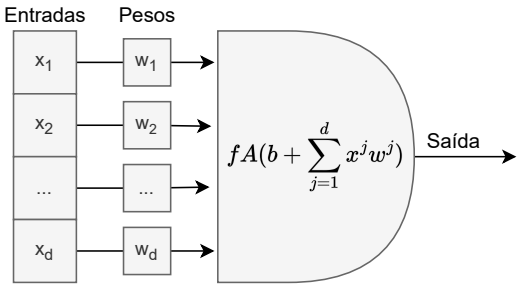
As RNAs são sistemas computacionais compostos de unidades de processamento simples. Essas unidades, conhecidas como neurônios artificias, computam funções matemáticas. As unidades são dispostas em uma ou mais camadas e são interligadas por um grande número de conexões, *i.e.*, os caminhos para o fluxo de informações, geralmente unidirecionais. Há diferentes configurações de como os neurônios se conectam, neste trabalho, foca-se no modelo onde cada neurônio de uma camada está interconectado com todos os neurônios da próxima camada; descrito como uma RNA densamente conectada. Essas conexões, que simulam as sinapses biológicas, possuem pesos e *bias* associados que ponderam a entrada recebida por cada neurônio da rede. Os pesos e *bias* podem assumir valores positivos ou negativos, dependendo do comportamento da conexão ser excitatória ou inibitória, respectivamente. Os pesos e *bias* têm seus valores ajustados no processo de aprendizado e codificam o conhecimento adquirido pela rede. Uma RNA é, portanto, caracterizada por dois aspectos básicos: arquitetura e aprendizado. Enquanto a arquitetura está relacionada ao tipo e ao número de unidades de processamento e à forma como os neurônios estão conectados, o aprendizado diz respeito às regras utilizadas para o ajuste dos pesos e *bias* da rede e que informação é utilizada pelas regras.

2.1.1 Arquitetura

O neurônio é a unidade de processamento fundamental de uma RNA. Cada terminal de entrada do neurônio, simulando os dendritos, recebe um valor. Os valores recebidos são ponderados e combinados por uma função matemática f_A chamada de função de ativação. Na Figura 2.1, ilustra-se um objeto **x** com *d* atributos representados na forma de vetor como **x** = $[x_1, x_2, ..., x_d]$ e um neurônio com *d* entradas cujos pesos podem ser representados na forma vetorial **w** = $[w_1, w_2, ..., w_d]$ e cujo *bias* pode ser representado como *b*.

A saída de um neurônio é definida por meio da aplicação de uma função de ati-

Figura 2.1 – Entrada total de um neurônio artificial.

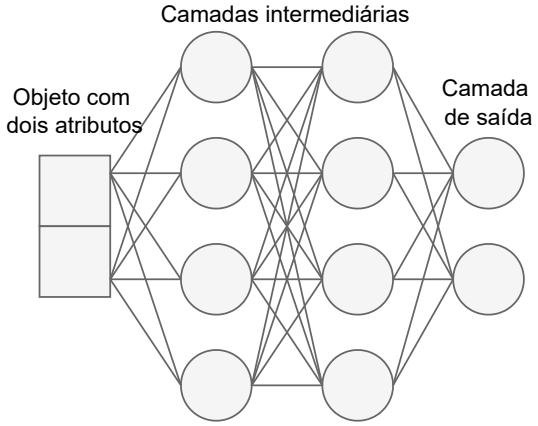


Fonte: adaptada de Faceli et al. (FACELI et al., 2021))

vação à entrada total. Várias funções de ativação têm sido propostas na literatura. Três exemplos relevantes são as funções linear, limiar e sigmoide. Uma rede com mais de uma camada de neurônios recebe o nome de rede multicamadas. A primeira e última camadas são chamadas, respectivamente, de camada de entrada e camada de saída. As demais camadas são denominadas camadas intermediárias, escondidas ou ocultas. A Figura 2.2 ilustra um exemplo de RNA com três camadas. Essa rede recebe como entrada valores de dois atributos de entrada e gera dois valores de saída. O número de camadas, o número de neurônios em cada camada, o grau de conectividade e a presença ou não de conexões de retropropagação definem a topologia de uma RNA.

O funcionamento usual de uma RNA consiste na aplicação de duas etapas: treinamento e teste.

Figura 2.2 – Exemplo de RNA multicamadas típica.



Fonte: adaptada de Faceli et al. (FACELI et al., 2021))

2.1.2 Treinamento e Teste

A etapa de aprendizado de uma rede neural, constitui-se principalmente com a definição dos valores dos pesos associados às conexões da rede. Isso é feito com o algoritmo *back-propagation*, que utiliza o método do gradiente descendente. Método constituído da iteração de duas fases, uma fase em frente (*forward*) e uma fase para trás (*backward*). Na fase *forward*, cada neurônio da camada de entrada é responsável por receber um objeto e normaliza-lo. A normalização é uma técnica aplicada para preparar os dados para o treinamento e é requerida devido aos atributos terem medidas de grandeza diferentes. O objetivo é adequar os valores numéricos dos atributos do *dataset* para uma escala em comum sem distorcer as diferenças entre os valores. Há diferentes técnicas de normalização, neste trabalho, aplica-se a padronização Z ilustrada na Equação 2.1, onde x é um dado de entrada, σ é o desvio padrão da população do conjunto de treino, μ é a variância da mesma população e z é o resultado da normalização. Após a transformação, obtém-se um conjunto com média igual a zero e desvio padrão igual a um.

$$z = \frac{x - \sigma}{\mu} \tag{2.1}$$

Em seguida, o objeto normalizado é recebido por cada um dos neurônios da primeira camada intermediária, onde é ponderado pelo peso associado a suas conexões de entrada correspondentes. Na primeira iteração desta etapa, esses pesos são escolhidos aleatoriamente. Cada neurônio nessa camada aplica a função de ativação à sua entrada total e produz um valor de saída, que é utilizado pelos neurônios da camada seguinte. Esse processo continua até que os neurônios da camada final produzam, cada um, seu valor de saída, que é então comparado ao valor desejado para a saída desse neurônio.

A diferença entre os valores de saída produzidos e desejados para cada neurônio da camada de saída indica o erro cometido pela rede para o objeto apresentado. O valor do erro de cada neurônio da camada de saída é utilizado na fase *backward* para ajustar os pesos das entradas. O ajuste prossegue da camada final até a primeira camada intermediária. A Equação 2.2 ilustra como é feito o ajuste dos pesos de uma rede multicamadas.

$$w_{jl}(t+1) = w_{jl}(t) + \alpha x^j \delta_l \tag{2.2}$$

Na Equação 2.2, w_{jl} representa o peso entre um neurônio l e o j-ésimo atributo de entrada ou a saída do j-ésimo neurônio da camada anterior, δ_l indica o erro associado

ao l-ésimo neurônio, α é a taxa de aprendizagem e x^j indica a entrada recebida por esse neurônio.

Como os valores dos erros são conhecidos apenas para os neurônios da camada de saída, o erro para os neurônios das camadas intermediárias precisa ser estimado utilizando os erros da camada posterior. O erro de um neurônio de uma dada camada intermediária é estimado como a soma dos erros dos neurônios da camada seguinte, cujos terminais de entrada estão conectados a ele, ponderado pelo valor do peso associado a essas conexões. Assim, a forma de calcular o erro depende da camada em que se encontra o neurônio, como ilustra a Equação 2.3.

$$\delta_l = \begin{cases} f'_A e_l & \text{se } n_l \in c_{\text{saída}} \\ f'_A \sum w_{lk}\delta_k & \text{se } n_l \in c_{\text{intermediária}} \end{cases} \tag{2.3}$$

Na Equação 2.3, n_l é o l-ésimo neurônio, c_{saida} representa a camada de saída, $c_{intermediaria}$ representa uma camada intermediária, f'_A é a derivada parcial da função de ativação do neurônio e e_l é a entropia cruzada esparsa. A entropia cruzada esparsa representa o erro cometido pelo neurônio de saída quando sua resposta f_q é comparada à resposta desejada y_q, como definido na Equação 2.4, onde N é numero de entradas do conjunto de dados.

$$e_l = -\frac{1}{N}\sum_{q=1}^{N} y_q \log\left(f_q\right) \tag{2.4}$$

A derivada parcial define o ajuste dos pesos, utilizando o gradiente descendente da função de ativação. Essa derivada mede a contribuição de cada peso no erro da rede para a classificação de um objeto **x**. Os ciclos de passar os dados ao treinamento e ajustar pesos são realizados até um critério de parada, por exemplo, um número máximo de repetições ou uma taxa mínima de erros. A superfície de erro minimizada pelo *back-propagation* apresenta regiões de mínimos locais e mínimos globais para problemas complexos. O objetivo do treinamento é atingir o mínimo global.

Redes neurais com múltiplas camadas conseguem extrair padrões não lineares e dependências não muito claras entre um grande número de características que descrevem os dados de entrada. Portanto, o funcionamento sofisticado das RNAs as torna boas candidatas para resolver problemas de classificação, incluindo de tráfego de rede.

Uma vez terminada a fase de treinamento da rede neural, é possível utilizar o modelo ajustado para obter predições para novas entradas. Essa etapa chama-se de teste

sobre os dados de entrada e serve para avaliar o quão adequado é o modelo gerado. Para tal, verifica-se a taxa de erro obtida comparando a saída proveniente da RNA com o resultado esperado. Para que esta etapa seja imparcial, recomenda-se que os dados de teste sejam inéditos, ou seja, não podem ter sido usados para o processo de treinamento da RNA.

2.2 Redes Programáveis e a Linguagem de Programação P4

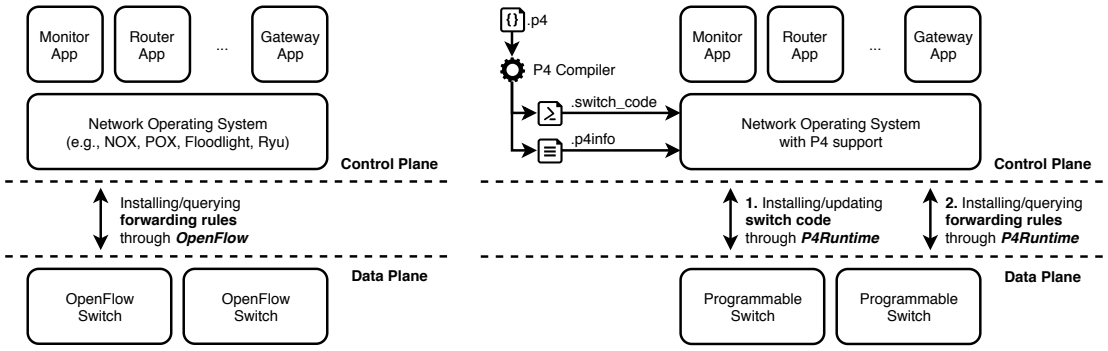
Um problema comumente mencionado no contexto de gerência de redes é a ossificação das mesmas. Esse termo envolve tanto a dificuldade de integrar novas tecnologias e padrões em infraestruturas de rede compartilhadas como a de introduzir novos protocolos e serviços em modelos arquitetônicos existentes (TURNER; TAYLOR, 2005). Esse cenário de ossificação de rede mudou recentemente com o advento de dois conceitos: virtualização de redes e Redes Definidas por Software (ou *Software-Defined Networking* – SDN) (ANDERSON et al., 2005).

Apesar dos avanços mencionados acima, a evolução da rede era, até mais recentemente, ainda limitada pela incapacidade dos operadores de programar o comportamento dos dispositivos de encaminhamento em si. O projeto e a configuração de uma rede eram limitados pelo conjunto de recursos e protocolos suportados pelos *chips* ASIC do dispositivos de rede, que têm função fixa, e não podem definir cabeçalhos de pacotes ou rotinas personalizadas de análise de pacotes. Um conceito que surgiu para resolver o problema descrito é a *programabilidade do plano de dados*. Isso remodela o cenário das SDNs, permitindo que os operadores de rede reprogramem os dispositivos de encaminhamento em produção para implantar novos protocolos de rede, personalizar o comportamento da rede e, consequentemente, desenvolver serviços inovadores (SAQUETTI et al., 2021).

A Figura 2.3 ilustra o conceito. Em um plano de dados programável, os operadores podem codificar um *pipeline* de processamento de pacotes que especifica campos de cabeçalhos personalizados, como esses cabeçalhos são extraídos dos pacotes, e como esses devem ser tratados pelos dispositivos. Ao usar um compilador fornecido pelo fabricante, os operadores podem compilar o código e implantar o programa resultante nos dispositivos de encaminhamento de sua rede, redefinindo quais protocolos eles suportam e como eles se comportam. Em seguida, um protocolo de comunicação entre os planos de controle e de dados semelhante ao *OpenFlow* (MCKEOWN et al., 2008) pode ser usado para preencher regras de encaminhamento baseadas no *pipeline* personalizado instalado

nos dispositivos.

Figura 2.3 – SDN tradicional e SDN combinada com planos de dados programáveis.



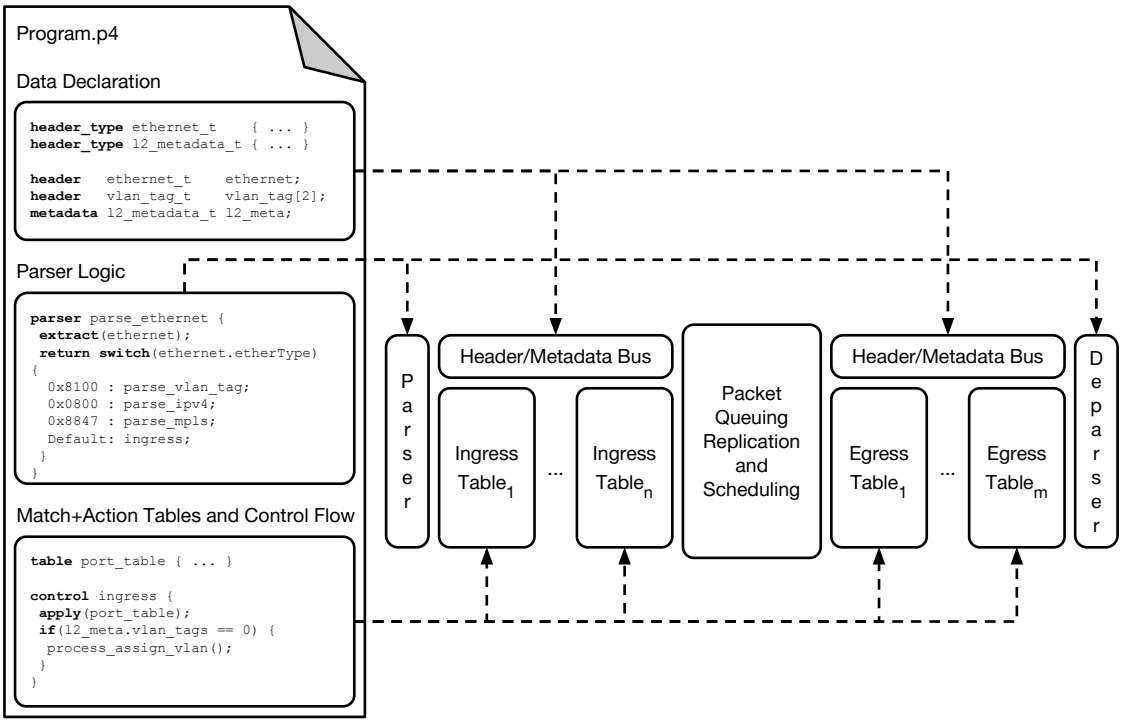
Fonte: Marques et al. (MARQUES, 2021)

A programabilidade do plano de dados está ganhando atenção e impulso rapidamente, pois representa um passo importante além de SDN com OpenFlow (CORDEIRO; MARQUES; GASPARY, 2017). Em um ambiente habilitado para plano de dados programável, a especificação das regras de fluxo não é mais restrita pela especificação do dispositivo de encaminhamento. Em vez disso, os operadores podem usar linguagens de domínio específico (ou *Domain-Specific Languages* – DSLs) para programar efetivamente o comportamento da rede. A seguir, apresenta-se brevemente recursos e capacidades da DSL de maior proeminência atualmente: a linguagem P4.

P4 é uma linguagem de alto nível para programar como os dispositivos de encaminhamento de rede processam pacotes. P4 é independente de dispositivo alvo, ou seja, adequada para descrever o comportamento de vários tipos de dispositivos, por exemplo, ASICs, NPUs, FPGAs e *switches* (BOSSHART et al., 2014). A linguagem abstrai o *parsing* de cabeçalhos de protocolos e o processamento de pacotes, fornecendo um modelo de encaminhamento generalizado. O código P4 é organizado logicamente em (a) declaração de dados, (b) lógica do analisador e (c) tabelas de *match-action* e seções de controle de fluxo. Cada um deles é mapeado para elementos específicos do modelo de encaminhamento alvo (por exemplo, para a arquitetura RMT na Figura 2.4).

A seção de declaração de dados define o formato do cabeçalho do pacote e as informações de metadados que podem ser usadas para sua análise. Essa seção é mapeada em um barramento de cabeçalhos e metadados que transporta essas informações por todos os estágios de processamento. Os tipos de cabeçalho são declarados de maneira semelhante às *structs* em C, ou seja, os campos são definidos em uma ordem específica e com um comprimento predeterminado.

Figura 2.4 – Seções de código P4 e mapeamento para o modelo de encaminhamento RMT.



Fonte: Marques et al. (MARQUES, 2021)

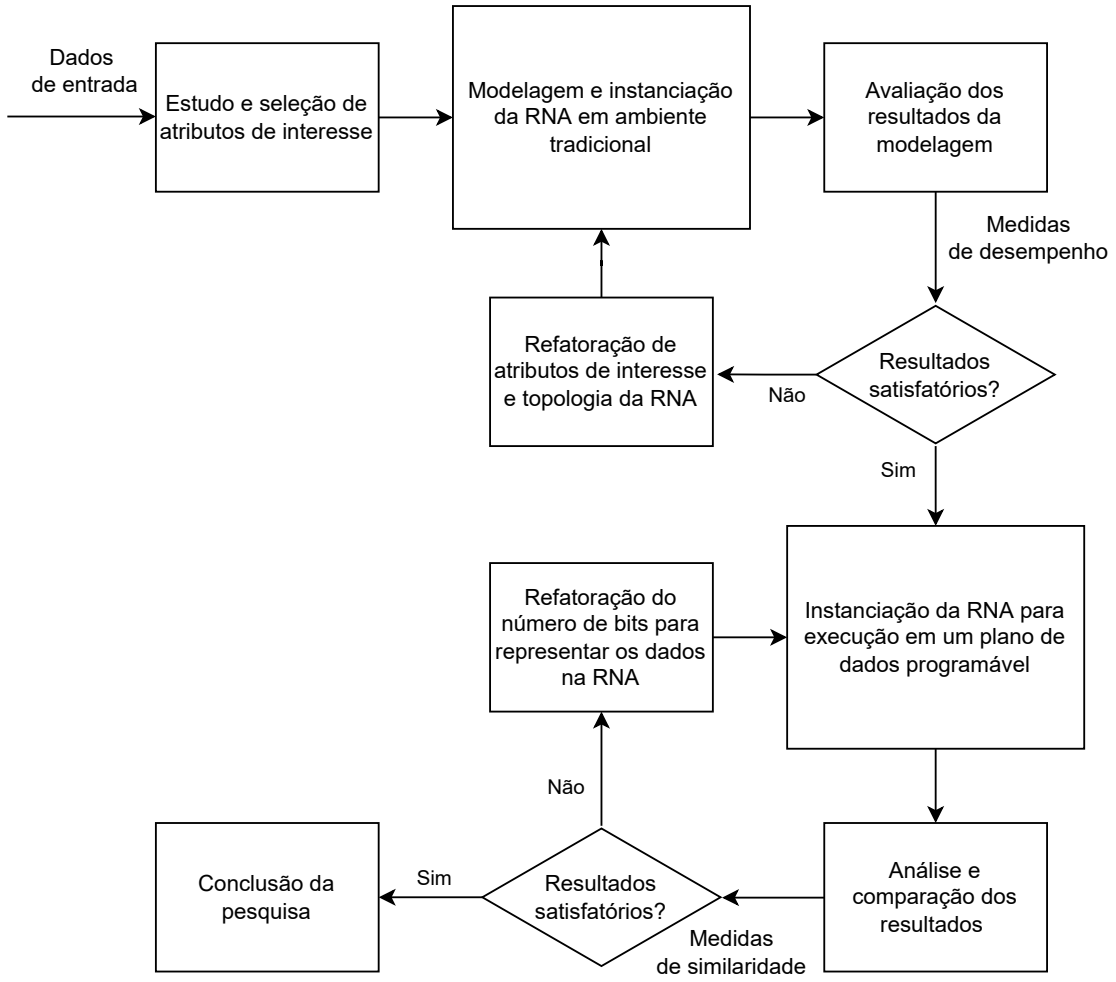
A seção de lógica do analisador especifica como, quando e em que ordem cada um dos cabeçalhos deve ser analisado. Essa seção de um programa P4 é mapeada para os elementos *parser* e *deparser* do modelo de encaminhamento. Esses elementos são, então, responsáveis por extrair o campo de cabeçalho dos pacotes em sua entrada (*parser*) e serializar seus valores atualizados de volta para os pacotes em sua saída (*deparser*).

A última seção do programa, as tabelas de *match-action* e controle de fluxo, especificam tabelas de pesquisa capazes de fazer a correspondência em campos de cabeçalho arbitrários e modificar cabeçalhos de pacotes e metadados por meio de ações personalizadas. Também expressa funções de controle que definem em qual ordem e circunstâncias cada tabela deve ser executada. Essa seção é mapeada para elementos de *match-action* configuráveis nos *pipelines* de entrada e saída. O *pipeline* de entrada realiza a modificação de pacotes e também estipula as intenções de saída (por exemplo, por qual porta encaminhar um pacote). O *pipeline* de saída realiza outras modificações de pacote necessárias, por exemplo, reescrevendo um campo de endereço de origem do cabeçalho Ethernet com o endereço MAC da porta de saída do *switch*.

3 METODOLOGIA DE TRABALHO

O presente trabalho tem como finalidade projetar, implementar e avaliar uma rede neural artificial para executar em dispositivos de um plano de dados programável. Este trabalho apresenta dois eixos de ação distintos. O primeiro eixo é a implementação de uma RNA para executar em um ambiente tradicional (executando em CPU ou GPU). O segundo eixo é o projeto e o desenvolvimento da RNA operante de forma distribuída nos dispositivos de rede programáveis. Para concretizar os objetivos do trabalho, realiza-se as etapas ilustradas na Figura 3.1 e descritas a seguir.

Figura 3.1 – Etapas do trabalho.



Fonte: autoria própria.

Estudo dos dados de entrada e pré-seleção de atributos interesse. No contexto de classificação de tráfego, os dados de entrada de uma rede neural são tipicamente conjuntos de dados (*datasets*) com uma série de características que descrevem e resumem os fluxos de pacotes. Esse tipo de *dataset* por ser obtido, por exemplo, via ferramenta anali-

sadora de pacotes, interceptando o tráfego de uma rede em um dado intervalo de tempo. Neste trabalho, emprega-se os mesmos *datasets* do trabalho de Michael et al. (MICHAEL et al., 2017) sobre classificação de tráfego de rede através de redes neurais.

Devido ao grande volume dos conjuntos de dados, é interessante empregar técnicas de análise exploratória dos dados para desenvolver uma visão abrangente sobre suas características e evitar potenciais problemas. Mais especificamente, a exploração de dados é útil para visualizar a estrutura dos conjuntos, para identificar graficamente os relacionamentos entre os atributos dos dados, a presença ou não de *outliers*, excluir itens irrelevantes, e aprender sobre a distribuição de valores dos dados a fim de revelar padrões e pontos de interesse.

Por causa das restrições de processamento e memória dos dispositivos de *hardware* onde almeja-se executar a RNA (em um plano de dados programável), é preciso selecionar no *dataset* os atributos realizáveis e em quantidade compatível com a capacidade do *hardware* subjacente. Para isso, deve ser levado em consideração o quanto de processamento é necessário para a obtenção de cada característica e se isso é adequado às capacidades dos dispositivos do plano de dados. Para especificar um bom subgrupo de atributos, que melhor discriminam os dados de entrada, são empregadas técnicas de *feature importance*.

Modelagem da RNA. A construção de uma rede neural envolve várias decisões críticas quanto à topologia da rede. Essas decisões devem determinar (a) o conjunto de atributos de entrada, (b) o número de camadas ocultas, (c) o número de nós em cada camada oculta, (d) o número de nós na camada de saída, (e) a função de ativação, e (f) o número de iterações de retropropagação aplicadas.

Na modelagem da RNA utilizada neste trabalho, além de objetivar-se maximizar a acurácia da classificação investigando uma combinação ótima dos parâmetros descritos acima, faz-se a escolha destes de maneira a tornar factível a implantação da RNA operante no plano de dados. Essa decisão é justificada pelas limitações intrínsecas em relação à capacidade de processamento e de memória dos dispositivos de planos de dados programáveis, e devem ser respeitadas para não prejudicar a alta vazão de transferência de pacotes no *pipeline* dos dispositivos.

Instanciação da RNA em ambiente tradicional. Realiza-se a instanciação da RNA em um ambiente tradicional baseado em CPU, onde a RNA executa em nível de aplicação. Para isso, dispõe-se de bibliotecas de aprendizado de máquina bem difundidas,

tais como, *Pytorch*[1], *scikit-learn*[2] e *Tensorflow Keras*[3]. Tais plataformas são ideais para testar diferentes topologias de redes neurais, técnicas de *feature selection* e estudar os conjuntos de dados de entrada, para obter-se uma classificação com alta taxa de acerto. A partir dessa instanciação, o modelo treinado é salvo, contendo os valores de pesos e *biases* associados a cada neurônio. Posteriormente, esse modelo salvo é empregado na RNA executando no plano de dados.

Avaliação da modelagem realizada. Os resultados obtidos com a implementação da rede neural tradicional são analisados por meio de métricas de desempenho tradicionais, tais como acurácia e matriz de confusão. Essas métricas demonstram o quão eficaz foi a escolha dos parâmetros de modelagem da RNA e são utilizadas como insumos para a próxima etapa. Os resultados são salvos para serem comparados com os obtidos pela RNA operante no plano de dados

Refatoração de atributos de interesse e topologias da RNA. Determinar o quão interessante é cada um dos atributos utilizados serve para eliminar características pouco interessantes e escolher novas características para serem avaliadas. Essa estratégia de aprendizado de máquina chama-se *feature selection*. Há diversas técnicas a serem experimentadas, alguns exemplos utilizados em outros trabalhos são: Árvores de Decisão, *Ceteris Paribus* e *Permutations Importance* (HONGFANG et al., 2021). Ressalta-se que pode-se testar múltiplas técnicas para avaliar a mais eficiente e o método a ser escolhido deve ser indicado para a tarefa de classificação (muitos métodos são indicados para tarefas de regressão). Os resultados obtidos também são utilizados para recalibrar os parâmetros do modelo mencionados anteriormente.

Instanciação da RNA para execução em dispositivos de um plano de dados programável. Procede-se à instanciação da RNA, capaz de executar a etapa de teste sobre os dados de entrada, para execução em dispositivos de rede programáveis. A implementação da RNA operante no plano de dados programável, apresentada neste trabalho, executa somente a etapa de teste sobre os dados de entrada e, portanto, depende de serem fornecidos os valores dos pesos e *bias* associados aos neurônios, provenientes da etapa de treinamento realizada na RNA executando em ambiente tradicional. Para que isso seja possível, ambas devem ter as mesmas especificações quanto aos parâmetros de modelagem da rede e também o mesmo processamento deve ser aplicado sobre o dados alvos. Para tal, identifica-se duas etapas para reproduzir o funcionamento de tal RNA,

[1] https://pytorch.org/
[2] https://scikit-learn.org/stable/
[3] https://www.tensorflow.org/

no ambiente P4. A primeira, é mapear a topologia da rede, onde cada neurônio de cada camada podem ser mapeado para um *switch* que possua as mesmas conexões de entrada e saída. A segunda etapa é traduzir o processamento executado pelos neurônios sobre os dados alvos. Este processamento é diferente para cada camada em que um neurônio está posicionado.

A linguagem P4 não provém suporte à números negativos e números em representação de ponto flutuante, necessários no funcionamento de uma RNA. Portanto, mecanismos devem ser introduzidos para solucionar tais desafios. Tipicamente, esse tipo de limitação em plataformas é abordada através do uso de representação de números em complemento de dois (para valores negativos) e de representação de números em ponto fixo (para valores em ponto flutuante). Por fim, os dados da RNA devem ser armazenados em registradores e será preciso especificar um número de *bits* para representa-los.

Análise e comparação dos resultados. Com os resultados de ambas RNAs, torna-se possível contrastar os valores de saída e as medidas de desempenho da RNA operando no plano de dados programável com as observadas para a RNA executando em ambiente tradicional. Esses resultados podem ser usados para avaliar o quão precisa é a classificação da rede neural instanciada no plano de dados programável e, também, para refatorar o número de *bits* usado para representar os dados. Nesta avaliação, é interessante considerar não somente as métricas tradicionais de aprendizado de máquina (como acurácia e precisão), mas também métricas que evidenciem a qualidade com que os dispositivos programáveis implementam o modelo treinado.

4 IMPLEMENTAÇÃO E AVALIAÇÃO EXPERIMENTAL

Neste capítulo, são descritos a implementação do método proposto e os resultados obtidos neste trabalho de conclusão de curso. A organização deste capítulo reflete os passos elencados na metodologia, apresentando cada passo em detalhes.

4.1 Estudo dos dados de entrada e pré-seleção de atributos interesse.

Para este estudo, utilizou-se como dados de entrada da rede neural os dez *datasets* disponibilizados pela Universidade de Cambridge e utilizados em diversos trabalhos na área de classificação de tráfego de rede (MOORE et al., 2005). Esses dados foram coletados em diversos momentos em uma rede do campus da universidade, usando um monitor de rede de alto desempenho, por períodos de uma hora cada vez. Esse campus contém diversos laboratórios de pesquisa com aproximadamente mil usuários conectados à *Internet* através de uma conexão Gigabit Ethernet *full-duplex* (MOORE; ZUEV, 2005).

Nestes *datasets*, fluxos são distinguidos por uma tupla de identificação que consiste nos endereços IP dos hospedeiros comunicantes, o protocolo da camada de transporte e os números das portas usadas pelos hospedeiros. Cada fluxo tem uma série de características sumarizando seu comportamento. Juntas, essas propriedades formam uma lista com 248 discriminadores (atributos) e um rótulo (classe) para cada fluxo. A seguir, enumera-se alguns exemplos dentre os 248 discriminadores. A lista completa é disponibilizada em (MOORE et al., 2005).

- Métricas de tempo entre chegadas de pacotes (*e.g.*, média, variância, número total).
- Métricas de fluxo (*e.g.*, total de *bytes* do fluxo, total de pacotes).
- Métricas de pacotes SACK, pacotes SYN, pacotes FIN, pacotes URG; Tamanho de segmentos e janelas anunciados.
- Dados perdidos ou truncados.
- Tamanho da carga útil (*e.g.*, média, variância, 1º e 3º quartis, mínimo, máximo).
- Estatísticas de tempo de ida e volta (*i.e., round-trip times*).

Dadas as limitações de processamento e memória de dispositivos de encaminhamento programáveis, é essencial, como um primeiro passo na seleção de atributos, identificar aqueles que poderiam ser obtidos ou medidos diretamente por esses dispositivos. Com este propósito, realizou-se uma análise aprofundada de cada atributo disponível no

dataset. Após essa análise, foram identificados 38 atributos que podem ser medidos ou obtidos no plano de dados, sem violar restrições de processamento. Uma inspeção do conteúdo desses 38 atributos revelou que três deles estavam zerados em todas as entradas do conjunto e, por isso, foram descartados. Em vista disto, de agora em diante, a modelagem focar-se-á neste conjunto seleto de 35 atributos como potenciais candidatos, apresentados no Apêndice A. Na Seção 4.3, será apresentado o método utilizado para escolher um conjunto ainda mais reduzido, de forma a minimizar a complexidade da RNA e a carga de processamento sobre os dispositivos do plano de dados.

Essencial para o trabalho de classificação é a categorização de fluxos de aplicações similares em classes de tráfego mais abrangentes. A Tabela 4.1 lista as classes de tráfego contempladas nos *datasets* utilizados, e exemplifica as aplicações representadas por cada classe. Ressalta-se que as aplicações exemplos apresentadas na tabela não são definitivas, pois o agrupamento de aplicações nessas classes é, em grande medida, subjetivo. Neste trabalho, optou-se por utilizar os rótulos previamente definidos nos *datasets*. Cada fluxo do *dataset* é mapeado a uma das classes da tabela. Uma descrição completa do processo de rotulação é apresentada em (MOORE; ZUEV, 2005).

Tabela 4.1 – Classes de tráfego e exemplos de aplicações.

Classe	**Exemplos de Aplicações**
Bulk	ftp
Database	postgres, sqlnet oracle, ingres
Interactive	ssh, klogin, rlogin, telnet
Mail	imap, pop2/3, smtp
Services	X11, dns, ident, ldap, ntp
WWW	web
P2P	KaZaA, BitTorrent, GnuTella
Attack	Internet worm and virus attacks
Games	Half-Life
Multimedia	Streaming

Fonte: autoria própria

Nota-se aqui que as características que descrevem os fluxos contidos nesses *datasets* são referentes ao tráfego de rede interceptado há mais de 15 anos (no momento da publicação deste trabalho). Na atualidade, muitas das aplicações responsáveis por produzir parte do tráfego de rede contido nesses conjuntos estão obsoletas ou são pouco utilizadas, alguns exemplos são: ftp, klogin, flogin, telnet, Half-Life, KaZaA e GnuTella. Consequentemente, esses conjuntos de dados têm pouca representatividade acerca do tráfego de rede dos dias atuais. De qualquer forma, neste trabalho, tem-se por principal

objetivo a transposição de um modelo de classificação funcional para o ambiente de planos de dados programáveis e, portanto, a representatividade ou não dos *datasets* fica em segundo plano. Foca-se, entretanto, na atualização da RNA proposta originalmente em (MICHAEL et al., 2017) seguindo técnicas e boas práticas de modelagem modernas. Isto com o objetivo de obter um modelo com arquitetura representativa do estado-da-arte em aprendizado de máquina para tarefas de classificação.

Almejando uma melhor percepção sobre os conjuntos de dados empregados, realizou-se uma análise exploratória. Para obter uma visão mais ampla dos dados de entrada, fez-se a união dos dez conjuntos de dados. A distribuição de classes é ilustrada na Tabela 4.2, na qual observa-se um forte desbalanceamento. Há uma grande predominância da classe WEB, representando cerca de 87% das amostras (ou um total de mais de trezentos mil amostras). Por outro lado, há uma baixa representatividade das classes GAMES, INTERACTIVE, MULTIMEDIA, e ATTACK, todas essas com menos de duas mil amostras (ou menos de 0,5% dentre todas as amostras) cada. Aplicar um conjunto de dados desbalanceados no treinamento de uma RNA é um problema pois o modelo tende a enviesar suas classificações para as classes predominantes e apresentar baixa precisão na classificação das classes com menos instâncias.

Tabela 4.2 – Distribuição de classes da união dos *datasets*.

Classe	**Número de entradas**	**Proporção em relação ao todo (%)**
WWW	328092	86.905
Mail	28567	7.566
FTP Data	5797	1.535
FTP Control	3054	0.808
Passive FTP	2688	0.712
Database	2648	0.701
Services	2099	0.555
P2P	2094	0.554
Attack	1793	0.474
Multimedia	576	0.152
Interactive	110	0.029
Games	8	0.002

Fonte: autoria própria

Para prevenir o cenário de desbalanceamento descrito no parágrafo acima, decidiu-se produzir conjuntos de dados balanceados, a partir dos dados disponíveis nos conjuntos originais. Para tanto, surge o questionamento de quantos exemplos por classe são necessários para fornecer à RNA a capacidade de distinguir com precisão as classes de tráfego.

Em geral, quanto mais amostras por classe, melhor o funcionamento de uma RNA, pois se tem mais exemplos para discernir padrões nos dados (CUI et al., 2019). Após testes variando-se o número de amostras, optou-se por utilizar oito mil amostras por classe para a criação dos *datasets* de treino e duas mil amostras por classe para os *datasets* de teste. Nesse processo, tomou-se o cuidado de garantir que os exemplares inclusos no conjunto de teste não fossem os mesmos do conjunto de treino, dividindo-os em dois grupos: teste e treino. Para as classes que possuíam mais amostras que o valor desejado, realiza-se a técnica de *undersampling*, ou seja, são selecionadas amostras aleatórias dentre o conjunto total até atingir-se o número desejado. Para as classes que constavam com menos amostras que o valor requerido, aplica-se a técnica de *oversampling*, onde as amostras são aleatoriamente repetidas até obter-se a quantidade almejada. Dado o número de amostras por classe usado, e por existirem doze classes de tráfego, os *datasets* de treino constam com 96 mil amostras e os *datasets* de teste ficaram com 24 mil amostras.

4.2 Modelagem e instanciação da RNA em ambiente tradicional.

Realizou-se a modelagem de uma rede neural artificial do tipo *Multi-Layer Perceptron*, tendo como base estudos relacionados para a tarefa de classificação (MICHAEL et al., 2017). Uma maneira garantida de capturar a configuração ideal da rede neural é testar cada combinação de atributos, número de iterações, número de nós e funções de ativação. Na prática, isso é inviável devido ao grande número de configurações possíveis. Deve-se, portanto, encontrar uma maneira eficaz de reduzir o espaço de parâmetros sem comprometer severamente a precisão da classificação. Neste trabalho, fez-se o processo de otimização através de uma escolha pré-direcionada pela literatura existente dos parâmetros iniciais para definir uma rede neural artificial.

No trabalho realizado por Auld et al. (AULD; MOORE; GULL, 2007), os autores alcançaram uma alta precisão de classificação usando apenas uma única camada oculta. O teorema da Aproximação Universal (HORNIK, 1991) afirma que qualquer rede *feed-forward* multicamadas pode ser aproximada (com precisão arbitrária) por uma única camada oculta com um número finito de nós. Em seu estudo, Auld et al. identificam que a taxa de erro estabiliza a partir do uso de sete nós ou mais na camada oculta, sendo sete o número de classes em seu trabalho. O aumento no número de nós aumenta, também, o tempo de computação, sem mudanças apreciáveis na precisão. Como resultado da avaliação, Auld et al. sugerem que o número de nós na camada oculta seja definido com valor

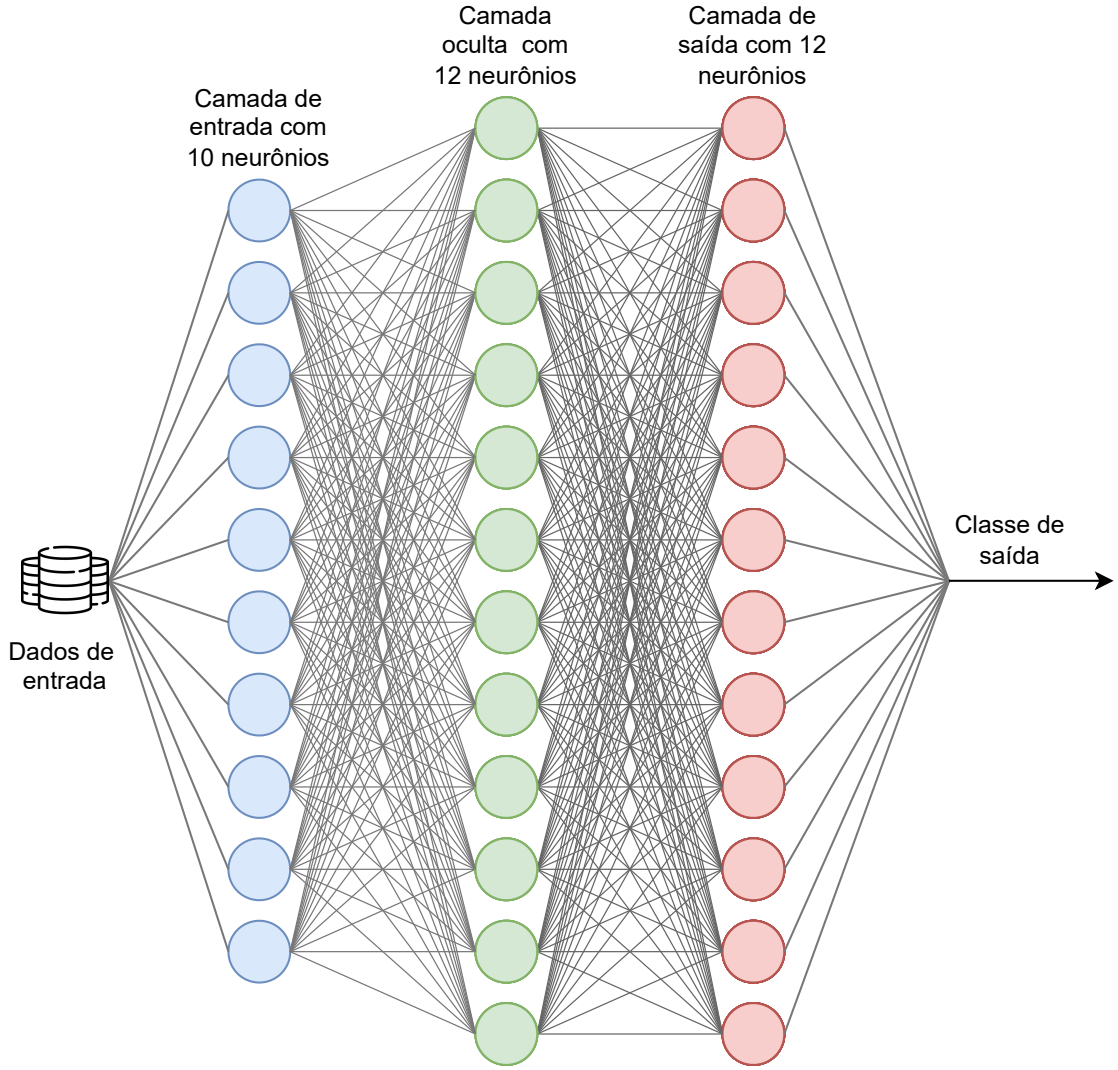
maior ou igual ao número de classes de tráfego, pois, em teoria, forneceria o nível de complexidade necessária para classificar com precisão todas as classes de tráfego. Deste modo, opta-se por utilizar uma única camada oculta com doze nós, visto existirem doze classes de tráfego distintas nos *datasets*. Ressalta-se que em testes realizados com maiores quantidades de neurônios não foi observado ganho expressivo na classificação. Com relação às funções de ativação, Auld et al. alcançaram bons níveis de acurácia na classificação usando as funções tangente hiperbólica e *Rectified Linear Unit* (ReLU). Neste trabalho, escolhe-se empregar a função de ativação ReLU, por ser matematicamente mais simples de ser implementada nos dispositivos alvos.

Uma vez definida a arquitetura e os parâmetros da RNA, sua instanciação em ambiente tradicional é realizada através da biblioteca de aprendizado de máquina *TensorFlow* (PANG; NIJKAMP; WU, 2020), inicialmente desenvolvida pela *Google* e posteriormente disponibilizada em código aberto para implementação de redes neurais. Tal escolha é justificada pela familiaridade com a plataforma e a farta documentação e suporte disponibilizados em seus tutoriais. A Figura 4.1 ilustra a modelagem da RNA aqui descrita. Observe que a RNA apresenta camadas oculta e de saída ambas com doze neurônicos.

4.3 Refatoração de atributos de interesse e topologia da RNA

Como mencionado anteriormente, dadas as limitações de dispositivos programáveis, é importante identificar o menor conjunto de atributos que possibilita alcançar qualidade de classificação aceitável. Para reduzir o número de atributos usados como entrada da RNA, é necessário determinar o nível de importância de cada um dos atributos disponíveis. Nesse sentido, implementa-se um algoritmo que se baseia em permutação dos dados de entrada para estimar a importância de cada atributo (do inglês, *feature importance*) (MOLNAR, 2022). Esse algoritmo treina a RNA com todos os atributos e a testa uma primeira vez para obter um valor de acurácia de referência. A seguir, para cada atributo, o conjunto de dados de teste é avaliado várias vezes aplicando a técnica de embaralhamento. A técnica de embaralhamento consiste em escolher um atributo e embaralhar seus valores (permutando os valores de lugar aleatoriamente). Isso é feito múltiplas vezes para dispor-se de vários embaralhamentos para cada atributo. Uma média das acurácias dos testes é obtida para cada atributo embaralhado e comparada com a acurácia de referência. A ideia é que o embaralhamento de um atributo mais significativo afete a acurácia com maior intensidade do que os menos significativos. Por fim, obtém-se uma lista de atributos

Figura 4.1 – Topologia da rede neural artificial para classificação de tráfego.

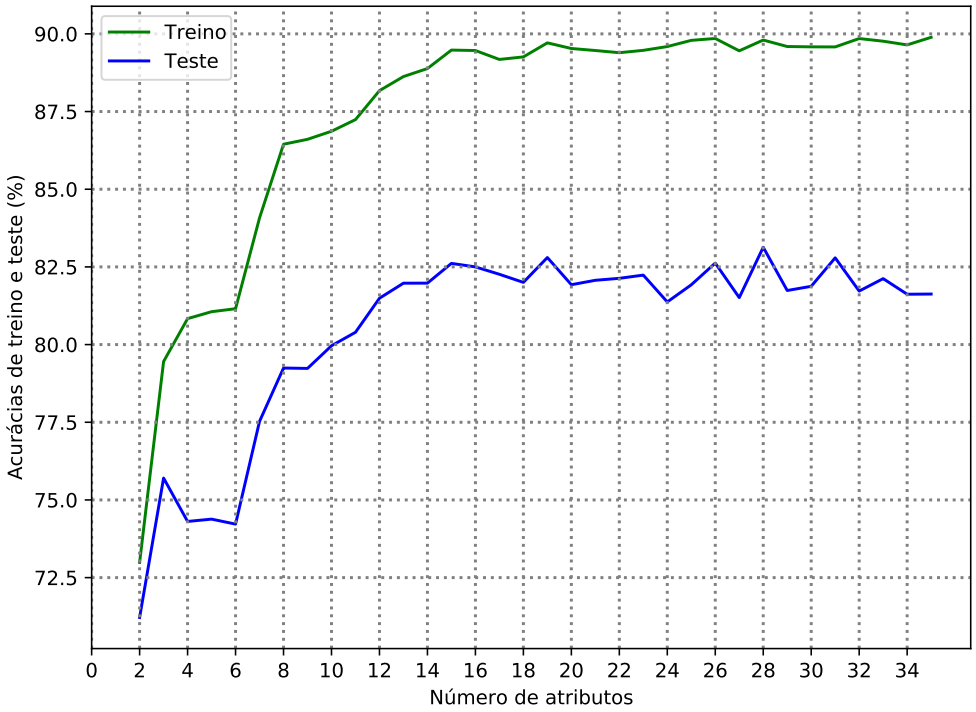


Fonte: autoria própria.

ordenados de forma decrescente de acordo com o nível de impacto na acurácia.

Após a computação desta lista ordenada de atributos, foi implementado um algoritmo que treina e testa uma RNA usando como dados de entrada os k atributos mais importantes para a classificação. Para a nossa avaliação, experimentou-se executar esse algoritmo com um número crescente de atributos sendo utilizados a cada iteração, partindo de 2 indo até 35 atributos. Uma RNA é treinada e testada múltiplas vezes para cada quantidade de atributos sendo usada, isto com o objetivo de suavizar o efeito probabilístico natural do treinamento de redes neurais. Os resultados dessa rede neural são apresentados na Figura 4.2 e analisados em seguida.

Figura 4.2 – Acurácias de treino (verde) e teste (azul) por número de atributos, utilizando-se a técnica de embaralhamento das permutações para obter a ordem dos atributos empregados.



Fonte: de autoria própria.

4.4 Avaliação da modelagem realizada

O gráfico apresentado na Figura 4.2 mostra os resultados de acurácia dos casos de treino (verde) e teste (azul) para os primeiros 35 atributos, ordenados pelo algoritmo de *feature importance*. Na curva da acurácia de teste, observa-se que com apenas dois atributos, tem-se uma acurácia razoável de 71%. Com dez atributos a acurácia alcança 80%. A partir desse valor, a curva tende a subir mais devagar e converge em torno de 82.5%, do décimo quinto atributo em diante. A partir de quinze atributos, o acréscimo no número de características aumenta, também, o tempo de computação, sem mudanças apreciáveis na precisão.

Com esses resultados, pode-se chegar a duas conclusões preliminares. A primeira, é possível obter altas taxas de acerto usando um número baixo de características acerca dos dados. A segunda, a ordem dos atributos provida pelo algoritmo de seleção de atributos mostra-se adequada, visto que o valor inicial com dois atributos é alto; e com dez atributos já chega bastante próximo ao valor da convergência. Com essas informações, daqui em diante, opta-se empregar na RNA dez atributos, candidatos a serem características

eficientes para classificar tráfego de rede. Uma descrição desses dez primeiros atributos segundo a ordem de importância computada pode ser encontrada no Apêndice A.

4.5 Instanciação da RNA para execução em dispositivos de um plano de dados programável.

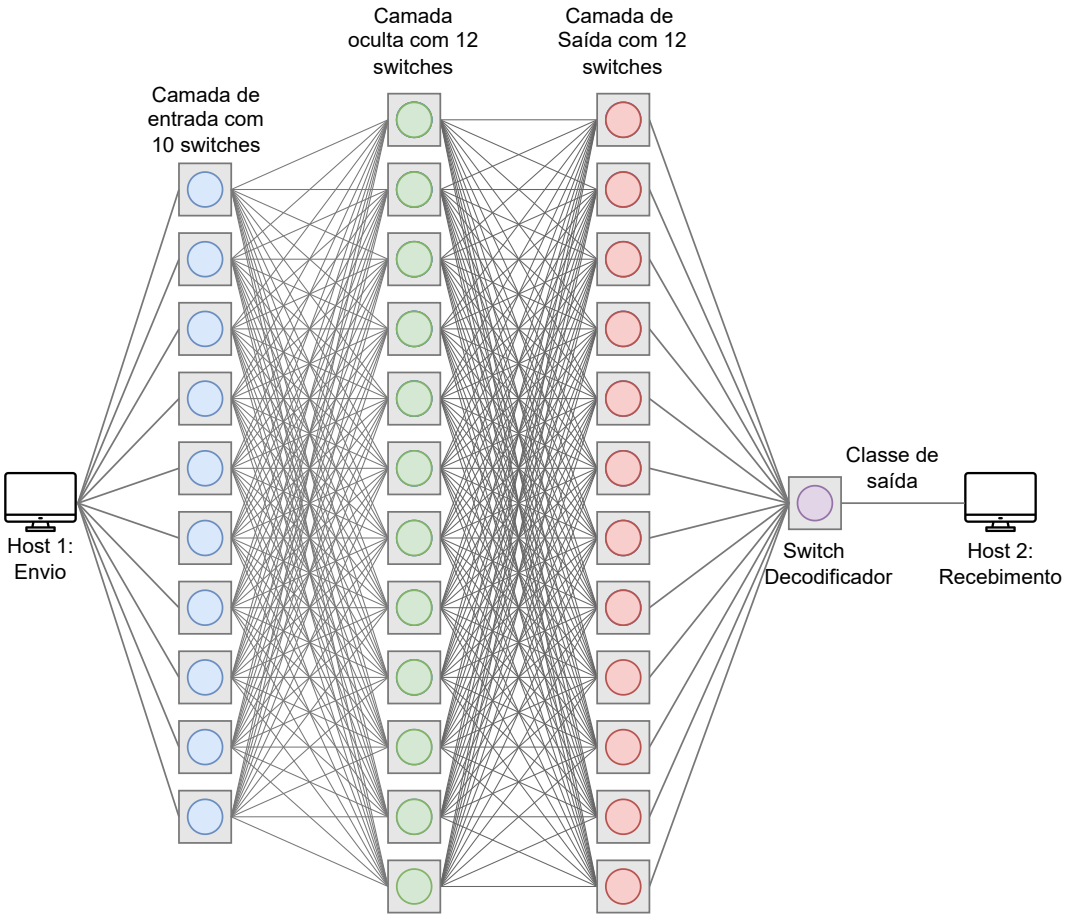
Procede-se à instanciação da RNA, capaz de executar a etapa de teste sobre os dados de entrada, para execução em dispositivos de rede programável. Como uma primeira prova de conceito, utiliza-se uma máquina virtual com o emulador de redes *Mininet* (FONTES et al., 2015), onde é possível programar código em nível de *kernel*, *switch* e aplicação em uma única máquina. Emprega-se *switches* em software, na linguagem P4, do tipo *BMv2 (behavioral-model version two)* para implementar a RNA.

Mapeamento da topologia e comunicação entre nodos. Para que seja possível configurar essa RNA supracitada com os pesos e *biases* provenientes do treinamento efetuado no ambiente tradicional, é necessário que ambas as RNAs tenham as mesmas especificações quanto aos parâmetros de modelagem da rede e também o mesmo processamento deve ser aplicado sobre o dados alvos. Neste trabalho, realiza-se o mapeamento da topologia utilizada para uma rede de *switches* virtuais equivalente, onde opta-se por acomodar cada neurônio em um *switch* com as mesmas conexões de entrada e saída. Observa-se, na Figura 4.3, a topologia da rede com os neurônios comportados nos *switches*. Para enviar e receber informações da rede de *switches*, instancia-se dois hospedeiros virtuais. Um primeiro hospedeiro de envio é posicionado conectado aos *switches* contendo os neurônios de entrada da RNA. Um segundo hospedeiro de recebimento é conectado a um *switch* que decodifica a camada de saída da RNA. Pacotes de rede *Ethernet* usados na comunicação entre os nodos, que contêm também um cabeçalho especial, com um estímulo (isto é, dado a ser processado), o número identificador do neurônio que o envia, e o número identificador do caso de teste.

O número identificar do caso de teste tem o propósito de impedir que a rede de *switches* misture estímulos entre casos de teste distintos. Este número identificador se provou necessário devido às perdas de pacotes da rede inerentes no contexto deste trabalho em que há muitas conexões entre os comutadores e datagramas de rede são propagados para múltiplos receptores. Quando há perda de pacotes na rede, a RNA é incapaz de completar o processamento de um caso de teste. Diante disto, é necessário reenviar os estímulos de entrada para que a execução ocorra por completo. Para tratar o envio repetido

de estímulos na rede, foi implementado um mecanismo que funciona da seguinte maneira, para cada pacote de estímulo recebido por um *switch*, este irá primeiro verificar se o número identificador do caso de teste é o mesmo do caso corrente armazenado em um registrador local. Caso sejam iguais, o estímulo é processado normalmente. Noutro caso, o estímulo representa um novo caso de teste. Assim, qualquer estado parcial do neurônio é descartado, visto não ser relacionado ao novo caso. Desta forma, a perda de pacotes (e respectivos estímulos) pode ser mitigada pelo envio sucessivo de um mesmo caso de teste (usando o mesmo número identificador) até que todos os estímulos sejam recebidos e processados com sucesso de forma a gerar uma saída na RNA.

Figura 4.3 – Topologia da RNA implementada através de uma rede de *switches* em um plano de dados programável.



Fonte: autoria própria.

Parâmetros para definir o comportamento dos nodos. Prossegue-se a replicação da computação aplicada sobre os dados alvos em cada camada da rede. Independente da camada em que está posicionado, um nodo aplica o processamento sobre os dados que lhe foi designado, e após, propaga o pacote *Ethernet* contendo o dado processado para

todos os nodos da camada seguinte. O mesmo código P4 é executado em todos os nodos da rede, portanto, cada um consta com uma série de metadados que especificam qual comportamento dentro do código deve-se aplicar. Os metadados estão listados a seguir: (a) identificador do neurônio; (b) número de estímulos esperados: definido pelo número de nodos da camada anterior, um nodo só deve propagar sua informação depois que tiver recebido estímulos de todos os neurônios da camada anterior; (c) função de agregação: normalização nas camadas de entrada, soma ponderada nas camadas oculta e de saída, ou argumentos das máximas na camada decodificadora; (d) função de ativação: ReLU (camada oculta) ou função identidade (demais camadas); (e) *bias* e pesos: nodos das camada oculta e de saída possuem um *bias* e um conjunto de pesos, cada peso está associado a um nodo da camada anterior;

Representação dos dados e normalização da camada de entrada. O hospedeiro de envio despacha pacotes para os nodos da camada de entrada. Cada neurônio da camada de entrada é responsável por receber valores de um atributo e normaliza-los, seguindo a técnica descrita na Seção 2.1.2. Os valores dos dados de entrada contidos nos *datasets* deste trabalho, pertencem ao conjunto dos números naturais. Contudo, na padronização, após as operações de subtração e divisão, os valores passam ao conjunto dos números racionais. A linguagem P4 não oferece suporte à números negativos ou números racionais, por isso, é preciso representa-los de alguma maneira.

Para valores negativos, opta-se empregar representação em complemento de dois, devido a sua ampla utilização e à possibilidade de efetuar a aritmética natural de adição sem a necessidade de alterar os *bits*, tal operação é necessária no processamento das camadas seguintes. Para representar valores racionais decide-se utilizar notação em ponto fixo, onde armazena-se um número fixo de dígitos (*bits*) da parte fracionária. Números com mais dígitos racionais do que a quantidade estabelecida são truncados e, portanto, estão sujeitos a um erro de arredondamento. Após a normalização, utiliza-se a notação Qx.y para especificar x *bits* para a parte inteira e y *bits* para a parte fracionária.

Devido à linguagem P4 não fornecer suporte nativo à operação de divisão, determina-se implementar a padronização utilizando tabelas de *match-action* normalizadoras. Para cada atributo dos dados de entrada é instanciada uma tabela. As chaves para o *matching* dessa tabela são os possíveis valores para o respectivo atributo conforme observado no *dataset*. O retorno associado a cada valor de chave da tabela consiste no valor normalizado. A criação das tabelas de normalização e o processamento necessário para normalização dos dados e conversão dos mesmos para representações em complemento de dois e ponto

Algorithm 1 Function ReLU

1: **if** $(data)$ & $(1 << (wordsize - 1)) > 0$ **then**
2: $\quad data \leftarrow 0$

Algorithm 2 Function Weighted Sum

1: $bit < double > opA \leftarrow (bit < double >)weight$
2: $bit < double > opB \leftarrow (bit < double >)data$
3: **if** $(opA$ & $(1 << (wordsize - 1))) > 0$ **then**
4: $\quad opA \leftarrow ((1 << double) - (1 << wordsize)) + (bit < double >)opA$
5: **if** $(opB$ & $(1 << (wordsize - 1))) > 0$ **then**
6: $\quad opB \leftarrow ((1 << double) - (1 << wordsize)) + (bit < double >)opB$
7: $bit < double > result \leftarrow ((opA * opB) >> precision)$
8: $accumulator \leftarrow accumulator + (bit < wordsize >)result$

fixo são realizados em ambiente de computação tradicional. Desta maneira, um neurônio da camada de entrada consulta o valor recebido na tabela e obtém o valor normalizado. Em seguida, envia um pacote de rede a todos os nodos da camada oculta, contendo o dado normalizado.

Camada oculta: aplicação da soma ponderada e ReLU. Os nodos da camada oculta são responsáveis por realizar a soma ponderada descrita na Figura 2.1 dos fundamentos. Primeiramente, inicializa-se um registrador acumulador com o valor 0. Para cada dado de entrada recebido, deve-se multiplica-lo pelo peso associado à entrada proveniente, e somá-lo ao acumulador. Uma vez que todos os dados de entrada desta camada foram recebidos e ponderados, soma-se o acumulador ao bias, e aplica-se a função de ativação ReLU, detalhada no Algoritmo 1. A operação de multiplicação disponível na linguagem P4 não provém suporte ao caso particular deste trabalho, em que os dados são representados em complemento de dois e ponto fixo. Por isso, foi necessário elaborar o Algoritmo 2, que é adequado à representação dos dados e permite utilizar-se da multiplicação fornecida pela linguagem P4. Após isso, envia-se um pacote de rede contendo o dado alvo para todos os nodos da camada de saída.

Camada de saída e *switch* decodificador da classificação. Na camada de saída, os nodos aplicam o mesmo procedimento da camada oculta. Nesta camada, contudo, não é aplicada a função de ativação ReLU. O índice do nodo de saída que estiver mais ativado (com maior valor) será o índice da classe de tráfego prevista. Na sequência, os resultados dos nodos dessa camada são enviados para o nodo decodificador e para o hospedeiro de recebimento.

O nodo decodificador da classe de tráfego executa a função Argumentos dos Má-

Algorithm 3 Function Argmax

```
1: if nReceivedStimuli == 1 then
2:   idHighestNeuron ← idNeuron
3:   highestValue ← data
4: else
5:   bit < wordsize > opA ← data
6:   bit < wordsize > opB ← highestValue
7:   bit < 1 > opASignal ← (opA)&(1 << (wordsize − 1)) > 0)
8:   bit < 1 > opBSignal ← (opB)&(1 << (wordsize − 1)) > 0)
9:   if (opASignal == 0) && (opBSignal == 1) then
10:    idHighestNeuron ← idNeuron
11:    highestValue ← data
12:  else
13:    if (opASignal == opBSignal) && (opA > opB) then
14:      idHighestNeuron ← idNeuron
15:      highestValue ← data
```

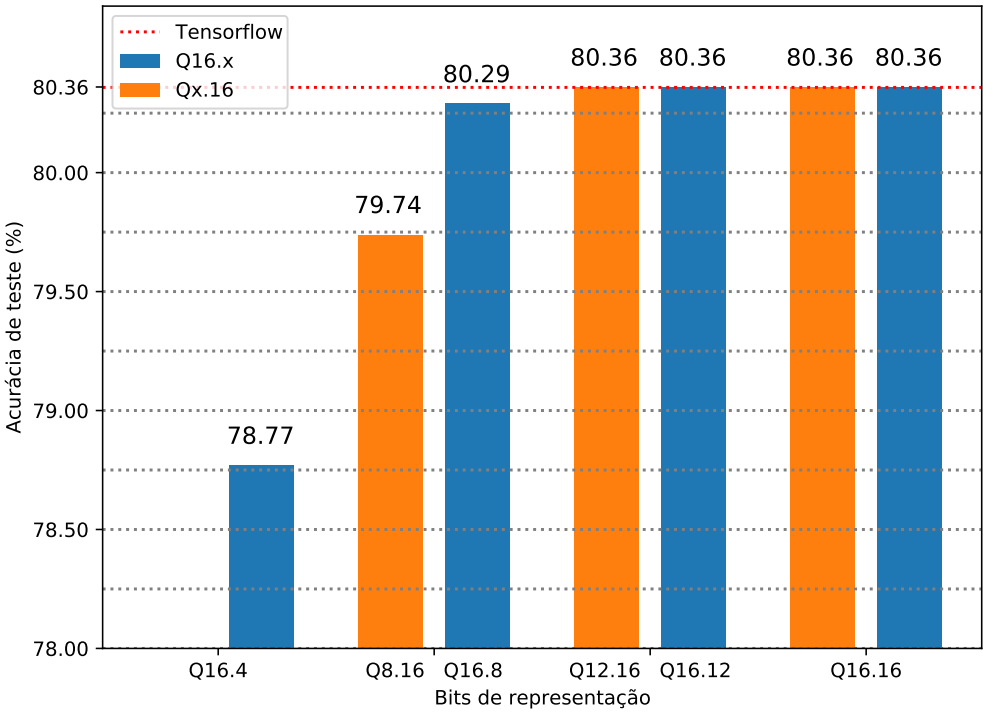
ximos, detalhada no Algoritmo 3. Tal nodo recebe os dados provenientes dos nodos da camada de saída e envia um pacote de rede ao hospedeiro de recebimento com a classificação resultante, que corresponde ao índice do nodo do qual este recebeu o maior valor de estímulo.

4.6 Análise e comparação dos resultados

Nesta seção, apresenta-se os resultados obtidos da implantação da RNA em um plano de dados programável. Para isso, usufrui-se de comparações com os resultados obtidos com a RNA implementada no ambiente tradicional, utilizando o mesmo modelo. Portanto, além de avaliar a acurácia da RNA, interessou-se averiguar o quão similares foram os resultados das RNAs através de diferentes maneiras. As métricas de avaliação, aqui apresentadas, são dependentes do número de *bits* usados para representar os dados.

Acurácia. A acurácia da RNA é definida pela razão entre o número de acertos da classificação final pelo número de amostras analisadas. Na figura 4.4, apresenta-se os resultados da acurácia de teste para diferentes números de *bits* na representação dos dados. A linha vermelha mostra a acurácia obtida na RNA instanciada no *Tensorflow*, para servir de referência. Nas barras azuis, está fixo em dezesseis o número de *bits* para a parte inteira e varia-se o número de *bits* da parte fracionária e, nas barras laranjas, o oposto. Utilizando as representações Q16.12, Q12.16, e Q16.16 as acurácias são idênticas à acurácia obtida no *Tensorflow*, onde acerta-se 80.36% das classificações. Os resultados com menos *bits*

Figura 4.4 – Acurácias de teste para Q16.x e Qx.16



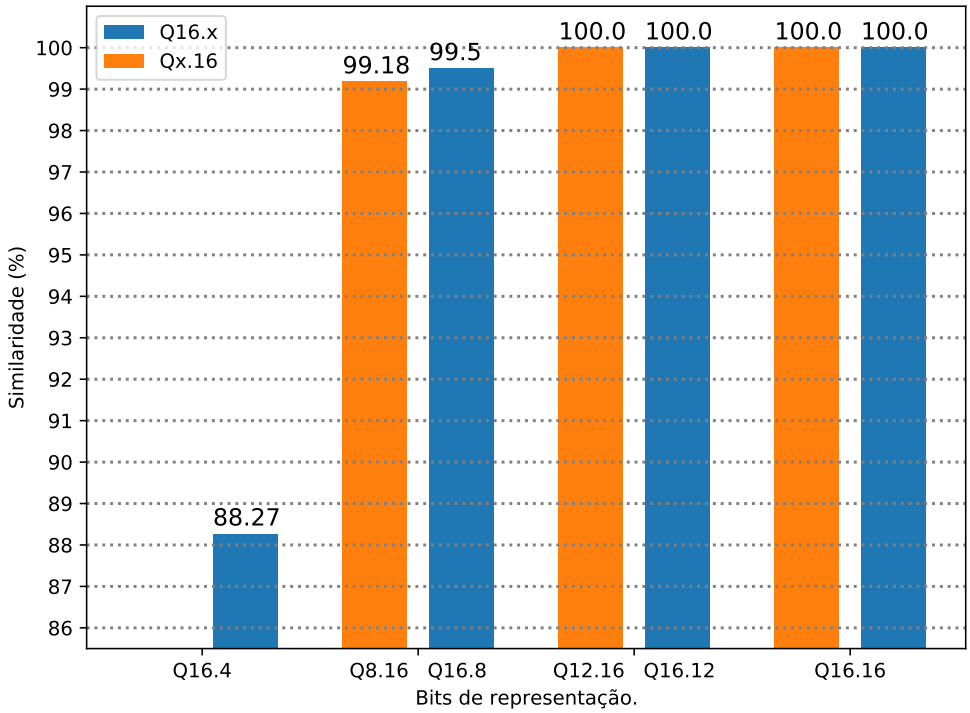
Fonte: autoria própria.

de representação apresentam perda de acurácia progressiva.

Similaridade Externa. Define-se como similaridade externa a proporção de classes previstas igualmente pelas RNAs em execução no ambiente P4 e no ambiente tradicional com *Tensorflow*. Na Figura 4.5, apresenta-se os resultados de similaridade externa para diferentes números de *bits* na representação dos dados. Novamente, nas barras azuis, está fixo em dezesseis o número de *bits* para a parte inteira e varia-se o número de *bits* da parte fracionária e, nas barras laranjas, o oposto. Utilizando as representações Q16.16, Q16.12 e Q12.16, obteve-se similaridade externa de 100%, ou seja, todas as classificações da rede neural do plano de dados foram idênticas às providas pelo *Tensorflow*. Isso justifica, os resultados de acurácias também idênticos, apresentados no parágrafo anterior. Percebe-se portanto, que utilizando *bits* o suficiente, é possível obter resultados idênticos quanto à acurácia e similaridade externa entre as RNAs nos dois ambientes. Com as representações Q8.16 e Q16.8, os resultados permanecem em um patamar satisfatório, pois a similaridade externa decai levemente para 99.18% e 99.5%, respectivamente. Com Q16.4 a similaridade cai para 88%.

Faz-se aqui, uma explicação da causa das perdas na acurácia e na similaridade

Figura 4.5 – Similaridade da RNA instanciada em ambiente P4 com a RNA instanciada no *Tensorflow* para Q16.x e Qx.16.



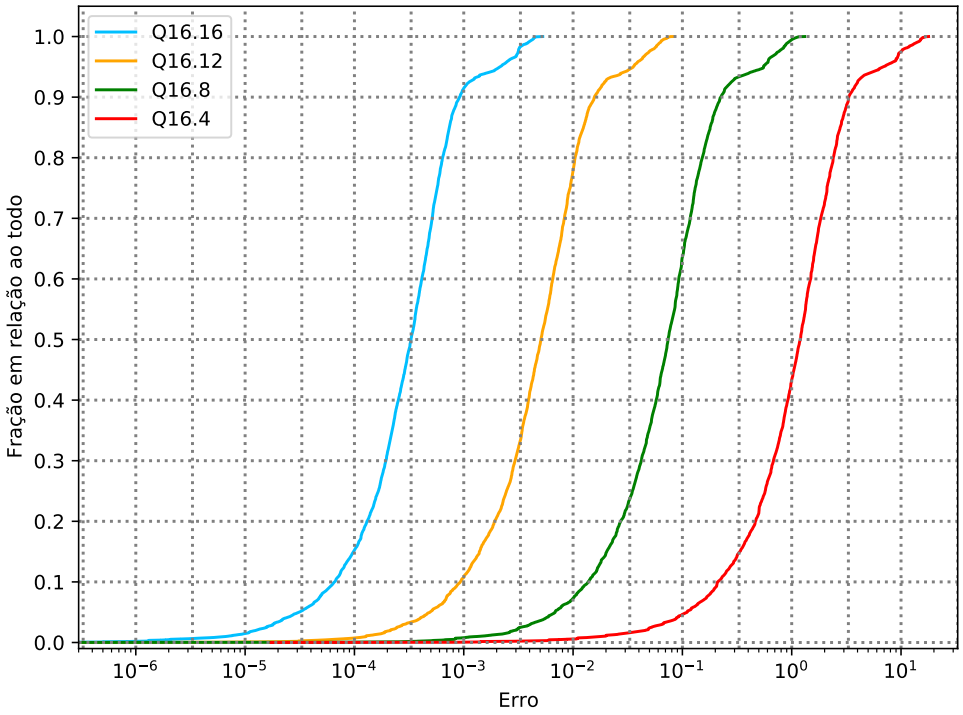
Fonte: autoria própria.

externa ao usar representações com menos *bits* para a parte fracionária. Após a etapa de normalização, valores que precisariam de mais *bits* para ser representados são truncados e, desse modo, possuem um erro de arredondamento associado. O processamento feito pelas camadas seguintes propaga o erro adiante. Ao chegar à camada de saída, o erro de arredondamento propagado pode causar um erro na classificação quando o valor de ativação de um neurônio da camada de saída se torna maior que o valor de ativação do neurônio que seria responsável pela classificação correta. É interessante, portanto, visualizar o erro da camada de saída, comparando o *output* dos neurônios dessa camada entre as diferentes plataformas onde está instanciada a RNA.

Similaridade Interna. Define-se como similaridade interna a diferença do valor absoluto dos nodos da camada saída, entre a RNA operante no plano de dados programável e a RNA executando no ambiente tradicional. Na Figura 4.6 apresenta-se a distribuição dos valores da camada de saída pelo erro associado ao valor. Nas curvas apresentadas, fixa-se em dezesseis o número de *bits* para a parte inteira e varia-se o número de *bits* da parte fracionária. Observa-se na curva azul, com Q16.16, aproximadamente 90% dos valores da camada de saída têm erro inferior a 0.001 e 100% dos valores têm erro inferior

a 0.01. Observa-se na curva amarela com Q16.12, aproximadamente 80% dos valores da camada de saída têm erro inferior a 0.01 e 100% dos valores têm erro inferior a 0.1. Ambas representações, Q16.16 e Q16.12, provém 100% de similaridade externa. Para valores menores de *bits* na parte fracionária, observa-se erro mais expressivo. No pior caso observado, Q16.4, o erro fruto de arredondamentos chega a cerca de 20.0.

Figura 4.6 – Função distribuição acumulada variando o número de *bits* para a parte fracionária.



Fonte: autoria própria.

5 CONCLUSÕES

Neste trabalho, realizou-se o projeto, o desenvolvimento e a análise de uma rede de *switches* virtuais que replicam o processamento de neurônios e se comunicam através de pacotes de rede. Para tanto, cumpriu-se uma série de etapas. Foram estas: a análise exploratória dos dados de entrada; uma seleção de atributos de interesse e viáveis de serem obtidos; a escolha dos parâmetros de topologia da RNA; a instanciação do modelo arquitetado usando a biblioteca *Tensorflow*; a discussão dos resultados obtidos com a modelagem; a instanciação da RNA em dispositivos de um plano de dados programáveis, onde configura-se os *switches* com os pesos e *biases* provenientes do modelo pré-treinando; e a análise e comparação dos resultados entre ambas RNAs.

A partir dos resultados discutidos neste trabalho, conclui-se que é viável implementar redes neurais artificias de forma distribuída em dispositivos de um plano de dados programável para classificação de tráfego de rede. A modelagem de rede empregada é razoavelmente simples, por isso, o uso de recursos de processamento e memória é adequado, e assim, respeitou-se as limitações dos dispositivos do plano de dados programável. Apesar da simplicidade do modelo, os resultados dos experimentos conduzidos mostraram-se satisfatórios, pois, utilizando um número de *bits* comum (isto é, 32 *bits* totais) para representar os dados, obtém-se classificações finais idênticas à RNA executando no ambiente tradicional. Neste sentido, os resultados são deveras positivos, pois mesmo usando representações de dados com poucos recursos computacionais, os patamares de acurácia não foram reduzidos, como poderia ser esperado. Isso sugere que há potencial de exploração da ideia de se executar RNAs em planos de dados programáveis para outras aplicações nas áreas de desempenho, segurança e balanceamento de carga.

Possíveis aprimoramentos futuros são: a generalização da implementação para viabilizar outras arquiteturas de redes neurais, permitindo empregar mais opções quanto aos parâmetros de topologia da RNA; relaxar a restrição sobre a topologia da rede de forma a tornar o posicionamento dos neurônios mais flexível em relação à prototipação feita nesse trabalho.

REFERÊNCIAS

ANDERSON, T. et al. Overcoming the internet impasse through virtualization. **Computer**, v. 38, n. 4, p. 34–41, 2005.

AULD, T.; MOORE, A. W.; GULL, S. F. Bayesian neural networks for internet traffic classification. **IEEE Transactions on Neural Networks**, v. 18, n. 1, p. 223–239, 2007.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, jul 2014. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/2656877.2656890>.

BOSSHART, P. et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In: **Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM**. New York, NY, USA: Association for Computing Machinery, 2013. (SIGCOMM '13), p. 99–110. ISBN 9781450320566. Available from Internet: <https://doi.org/10.1145/2486001.2486011>.

CORDEIRO, W. L. da C.; MARQUES, J. A.; GASPARY, L. P. Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management. **Journal of Network and Systems Management**, Springer, v. 25, n. 4, p. 784–818, 2017.

CUI, Y. et al. Class-balanced loss based on effective number of samples. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2019.

FACELI, K. et al. **Inteligência artificial: uma abordagem de aprendizado de máquina**. [S.l.]: LTC, 2021.

FONTES, R. R. et al. Mininet-wifi: Emulating software-defined wireless networks. In: **2015 11th International Conference on Network and Service Management (CNSM)**. [S.l.: s.n.], 2015. p. 384–389.

HONGFANG, Z. et al. A feature selection algorithm of decision tree based on feature weight. **Expert Systems with Applications**, v. 164, p. 113842, 2021. ISSN 0957-4174. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0957417420306515>.

HORNIK, K. Approximation capabilities of multilayer feedforward networks. **Neural Networks**, v. 4, n. 2, p. 251–257, 1991. ISSN 0893-6080. Available from Internet: <https://www.sciencedirect.com/science/article/pii/089360809190009T>.

LI, R. et al. Byte segment neural network for network traffic classification. Banff, AB, Canada, n. 26, p. 1–10, jun 2018.

MARQUES, J. A. Advancing network monitoring with data plane programmability and in-band network telemetry. Defesa de Proposta de Tese de Doutorado, 2021.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 38, n. 2, p. 69–74, mar 2008. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/1355734.1355746>.

MICHAEL, A. K. J. et al. **Network traffic classification via neural networks**. [S.l.], 2017. Available from Internet: <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-912.pdf>.

MOLNAR, C. **Interpretable Machine Learning**. [s.n.], 2022. Available from Internet: <https://christophm.github.io/interpretable-ml-book/feature-importance.html>.

MOORE, A. et al. **Discriminators for use in flow-based classification**. [S.l.], 2005.

MOORE, A. W.; ZUEV, D. Internet traffic classification using bayesian analysis techniques. **SIGMETRICS Perform. Eval. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 33, n. 1, p. 50–60, jun 2005. ISSN 0163-5999. Available from Internet: <https://doi.org/10.1145/1071690.1064220>.

PANG, B.; NIJKAMP, E.; WU, Y. N. Deep learning with tensorflow: A review. **Journal of Educational and Behavioral Statistics**, v. 45, n. 2, p. 227–248, 2020. Available from Internet: <https://doi.org/10.3102/1076998619872761>.

SAQUETTI, M. et al. Toward in-network intelligence: Running distributed artificial neural networks in the data plane. **IEEE Communications Letters**, v. 25, n. 11, p. 3551–3555, 2021.

TURNER, J.; TAYLOR, D. Diversifying the internet. In: . [S.l.: s.n.], 2005. v. 2, p. 6 pp. – 760. ISBN 0-7803-9414-3.

APÊNDICE A — DESCRIÇÃO DOS ATRIBUTOS

Neste apêndice são apresentados e descritos os atributos resultantes da pré-seleção com o objetivo de identificar atributos factíveis de serem calculados em dispositivos com suporte a programação do plano de dados.

Tabela A.1 – Descrição dos 35 atributos que podem ser medidos ou obtidos no plano de dados, ordenados pelo algoritmo de *feature importance*.

Ordem	**Abreviatura**	**Descrição**
1	max segm size b a	Tamanho mínimo de segmento observado durante a vida útil da conexão (sentido servidor→cliente).
2	min segm size a b	Tamanho máximo de segmento observado durante a vida útil da conexão (sentido cliente→servidor).
3	max segm size a b	Tamanho máximo de segmento observado durante a vida útil da conexão (sentido cliente→servidor).
4	min win adv a b	Anúncio mínimo em escala de janela, se ambos os lados negociarem a escala de janela (sentido cliente→servidor).
5	max data wire	Máximo de bytes totais no pacote IP.
6	max data ip	Máximo de bytes no pacote Ethernet.
7	min segm size b a	Tamanho mínimo do segmento observado durante a vida útil da conexão (sentido servidor→cliente).
8	min win adv b a	Anúncio mínimo de janela visto (sentido servidor→cliente).
9	max win adv b a	Anúncio máximo de janela visto, se ambos os lados negociaram o dimensionamento de janela durante a abertura da conexão (sentido servidor→cliente).
10	mss requested a b	Tamanho máximo do segmento (MSS) solicitado como uma opção TCP no pacote SYN que abre a conexão (sentido cliente→servidor).

Ordem	Abreviatura	Descrição
11	max win adv a b	Anúncio máximo de janela visto, se ambos os lados negociaram o dimensionamento de janela durante a abertura da conexão (sentido cliente→servidor).
12	mss requested b a	Tamanho máximo do segmento (MSS) solicitado como uma opção TCP no pacote SYN que abre a conexão (sentido servidor→cliente).
13	min data ip	Mínimo de bytes totais no pacote IP.
14	FIN pkts sent b a	A contagem de todos os pacotes vistos com os bits FIN definidos no cabeçalho TCP (sentido servidor→cliente).
15	pushed data pkts b a	A contagem de todos os pacotes vistos com o bit PUSH definido no cabeçalho TCP (sentido servidor→cliente).
16	min data wire	Mínimo de bytes no pacote Ethernet, usando o tamanho do pacote no fio.
17	max data control	Máximo de bytes de controle no pacote.
18	pushed data pkts a b	A contagem de todos os pacotes vistos com o bit PUSH definido no cabeçalho TCP (sentido cliente→servidor).
19	SYN pkts sent b a	A contagem de todos os pacotes vistos com os bits SYN definidos no cabeçalho TCP (sentido servidor→cliente).
20	min data control	Mínimo de bytes de controle no pacote, tamanho do cabeçalho do pacote (IP/TCP).
21	FIN pkts sent a b	A contagem de todos os pacotes vistos com os bits FIN definidos no cabeçalho TCP (sentido cliente→servidor).
22	SYN pkts sent a b	A contagem de todos os pacotes vistos com os bits SYN definidos no cabeçalho TCP (sentido cliente→servidor).

Ordem	Abreviatura	Descrição
23	zero win adv a b	O número de vezes que uma janela de recebimento zero foi anunciada (sentido cliente→servidor).
24	actual data bytes b a	O total de bytes de dados vistos. Observe que isso inclui bytes de retransmissões / pacotes de sondagem de janela se houver algum (sentido servidor→cliente).
25	actual data bytes a b	O total de bytes de dados vistos. Observe que isso inclui bytes de retransmissões / pacotes de sondagem de janela se houver algum (sentido cliente→servidor).
26	total packets b a	O número total de pacotes (sentido servidor→cliente).
27	ack pkts sent a b	O número total de pacotes de confirmação vistos (segmentos TCP visto com o conjunto de bits ACK) (sentido cliente→servidor).
28	total packets a b	O número total de pacotes (sentido cliente→servidor).
29	ack pkts sent b a	O número total de pacotes de confirmação vistos (segmentos TCP visto com o conjunto de bits ACK) (sentido servidor→cliente).
30	actual data pkts a b	A contagem de todos os pacotes com pelo menos um byte de carga de dados TCP. (sentido cliente→servidor).
31	actual data pkts b a	A contagem de todos os pacotes com pelo menos um byte de carga de dados TCP. (sentido servidor→cliente).
32	pure acks sent b a	O número total de pacotes de confirmação vistos que não foram apoiado com dados (apenas o cabeçalho TCP e sem carga útil de dados TCP) e não tinha nenhum dos conjunto de sinalizadores SYN/FIN/RST (sentido servidor→cliente).

Ordem	Abreviatura	Descrição
33	sack pkts sent a b	O número total de pacotes ack vistos carregando blocos TCP SACK [6] (sentido cliente→servidor).
34	sack pkts sent b a	O número total de pacotes ack vistos carregando blocos TCP SACK [6] (sentido servidor→cliente).
35	zero win adv b a	O número de vezes que uma janela de recebimento zero foi anunciada (sentido servidor→cliente).

Fonte: Autoria Própria