CLASSIFICAÇÃO DO TRÁFEGO DE REDES CORPORATIVAS: UMA ABORDAGEM COM REDES NEURAIS ARTIFICIAIS COM MLP

Ryan Larréa dos Santos[1], Luiz Fernando Segato dos Santos[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia de Mato Grosso do Sul — Aquidauana/MS

ryan.santos@estudante.ifms.edu.br, luiz.santos@ifms.edu.br

Resumo

Este artigo aborda a importância da Qualidade de Serviço (QoS) que desempenha um papel vital na gestão do tráfego de dados, especialmente para serviços sensíveis à latência, como VoIP. A integração da inteligência artificial, como a Rede Neural *Multilayer Perceptron* (MLP) é uma solução promissora para aprimorar a Qualidade de Serviço (QoS), devido à sua versatilidade no tratamento de padrões de dados complexos e não lineares. O presente artigo analisa como a inteligência artificial, especificamente as MLPs, pode melhorar o QoS em redes de computadores. Abordaremos a coleta e tratamento de dados, a implementação do método, a criação da arquitetura neural, o treinamento da rede e a apresentação dos resultados. Essa abordagem tem o potencial de melhorar significativamente o desempenho das redes, garantindo uma experiência de usuário superior em ambientes de tráfego intenso de dados.

Palavras-chave: Redes neurais, MLP, QoS

Introdução

Com o crescimento massivo de tráfego de dados e a expansão dos serviços, surgiram desafios significativos na gestão da qualidade desses serviços e na entrega eficaz de dados. Isso desempenha um papel crucial no gerenciamento de tráfego de rede, garantindo que serviços importantes, como o *Voice over IP* (VoIP), funcionem sem perda de pacotes, com baixa latência e com largura de banda adequada. As principais medições desta qualidade incluem fatores que afetam a qualidade das chamadas, como variação no atraso (conhecida como *jitter*), latência e perda de pacotes.

A Qualidade de Serviço, do inglês *Quality of Service* (QoS), é uma tecnologia presente em roteadores para garantir ao usuário maior controle sobre sua rede *Wi-Fi*. Por meio da ferramenta é possível determinar quais dispositivos e serviços terão maior prioridade de conexão. O recurso é interessante para quem precisa racionalizar a Internet, ou simplesmente precisa dar preferência para dispositivos que reproduzem vídeos em *streaming*, jogos *online*, entre outros tipos de uso. (GARRETT, 2018).

Para auxiliar esta demanda crescente, a integração da inteligência artificial na gestão do tráfego é uma solução promissora, por poder aumentar significativamente a qualidade do tráfego dos dados. Apesar dos avanços tecnológicos, garantir a qualidade do serviço nas redes de comunicações continua a ser um desafio complexo. Por isso, neste trabalho, explora-se detalhadamente a Rede Neural *Multilayer Perceptron* (MLP), destacando sua estrutura, funcionamento e aplicabilidade. Com ênfase na utilização com conjuntos de dados, a MLP é uma poderosa ferramenta para extrair padrões não lineares, aprender representações eficazes, fazer previsões precisas, tudo isso devido à sua arquitetura de várias camadas, que permite a captura de relações complexas entre as entradas, tornando-a flexível na modelagem de dados com padrões intrincados. Sua versatilidade a torna a escolha ideal para resolver uma ampla gama de problemas complexos em diversas disciplinas.

A MLP é uma rede neural semelhante à perceptron, mas com mais de uma camada de neurônios em alimentação direta. Tal tipo de rede é composta por camadas de neurônios ligadas entre si por sinapses com pesos. O aprendizado nesse tipo de rede é geralmente feito através do algoritmo de retropropagação do erro, mas existem outros algoritmos para este fim, como a Rprop. (WIKIPÉDIA, 2022)

Este artigo apresenta como esta integração da inteligência artificial ao QoS pode ser um passo significativo na melhoria do rendimento e na garantia da qualidade do serviço em ambientes de redes utilizando MLPs para a solução de tráfego em redes de computadores. Nas próximas seções serão apresentados como foram coletados os dados, tratamento dos dados, a implementação do método, a criação da arquitetura neural, treinamento da rede e por fim os resultados.

Metodologia

Nesta seção, é apresentado o processo de construção e geração da MLP para atender aos objetivos deste estudo. A implementação foi realizada em Python, com o uso de bibliotecas essenciais para o processamento de dados e criação da arquitetura da MLP. A escolha da linguagem Python deve-se à sua popularidade, vasta gama de

bibliotecas e facilidade de leitura, o que o torna ideal para este trabalho.

Inicialmente, foram importadas as bibliotecas fundamentais, como *pandas* para manipulação de dados tabulares, *numpy* para operações numéricas, *tensorflow* para a construção da rede neural, *train_test_split* do *sklearn* para dividir os dados em conjuntos de treinamento e teste, *StandardScaler* do *sklearn* para normalização dos dados e *matplotlib* para visualização dos resultados por meio de gráficos.

Além disso, foi realizada a montagem do Google Drive para facilitar o acesso aos conjuntos de dados e armazenamento de resultados durante o processo de desenvolvimento e treinamento da MLP. A Figura 1 a seguir mostra a importação das bibliotecas utilizadas.

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Montando o Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

Figura 1. Importação das bibliotecas mencionadas

Na Figura 2 é possível observar uma das funcionalidades da biblioteca *pandas*, a função *read_csv* que realiza a leitura de um determinado dataset com extensão csv.

```
path = "/content/drive/My Drive/Colab Notebooks/Data/combined.csv"
df = pd.read_csv(path, sep=';')
df
```

Figura 2. Especifica-se a localização do dataset no Drive

O Wireshark é um analisador de pacotes que, para tanto, tem capacidade de interceptar todos os pacotes que trafegam nas redes e de exibir detalhes das suas informações de controle (cabeçalhos) e conteúdo (*payload*). (HENRIQUE, 2018).

A coleta de dados foi conduzida utilizando o software Wireshark, que permitiu a obtenção de pacotes de *streaming*. Adicionalmente, foram empregados dados de VoIP disponibilizados gratuitamente no site do aplicativo. Logo, foram obtidos dois tipos de dados, o que possibilitou a criação de uma rede neural de classificação binária: "*Streaming*" e "VoIP" . Após isso, teve de ser coletado os metadados que se referem às informações adicionais associadas aos dados da rede que estão sendo analisados. Os metadados analisados para este projeto foram: *Timestamp, Source, Destination, Protocol_UDP, Protocol_TCP, Length, Jitter, Bandwidth, Throughput, Latency, BytesPerSecond, PacketsPerSecond, PacketSize*. Para a coleta destes metadados foi feito um script que coleta estas métricas automaticamente dos arquivos pkt referentes aos dados de "VoIP" e "*Streaming*" e adiciona ao arquivo csv, organizando-os por coluna. Dessa forma, obtêm-se os dados necessários para o treinamento da rede neural.

Após a obtenção dos dados foi feito o tratamento dos mesmos para a rede neural. Uma parte do código é dedicada à conversão de endereços IP para valores numéricos, pois somente assim a rede neural aceitaria os dados. Na Figura 3 é possível observar a implementação deste código.

```
def ip_to_numeric(ip):
    components = ip.split(".")
    numeric_value = np.dot(np.array(components, dtype=int), [256**3, 256**2, 256**1, 1])
    return numeric_value

[ ] # Lista para armazenar os resultados
    numeric_ips = []

    # Iterar pelas linhas do DataFrame
    for index, row in df.iterrows():
        ip = row['Source']
        numeric_ip = ip_to_numeric(ip)

        # Adicionar o valor numérico na lista
        numeric_ips.append(numeric_ip)

    # Adicionar a lista como uma nova coluna no DataFrame
    df['Source'] = numeric_ips

[ ] # Lista para armazenar os resultados
    numeric_ips = []

    # Iterar pelas linhas do DataFrame
    for index, row in df.iterrows():
        ip = row['Destination']
        numeric_ip = ip_to_numeric(ip)

        # Adicionar o valor numérico na lista
        numeric_ips.append(numeric_ip)

    # Adicionar a lista como uma nova coluna no DataFrame
    df['Destination'] = numeric_ips
```

Figura 3. Função para conversão de IPs.

Assim, foi criado uma simples arquitetura neural, pois não é necessária uma rede neural muito robusta por conta dos dados que obtivemos. Para essa parte foram criadas quatro camadas densas e adicionadas camadas de *dropout* para regularização do modelo como mostrado na Figura 4.

```
[19] # Definir a arquitetura do modelo MLP
    model = models.Sequential()

    model.add(layers.Dense(512, activation='relu', input_shape=(x_train.shape[1],)))
    model.add(layers.Dropout(0.1))
    model.add(layers.Dense(1024, activation='relu'))
    model.add(layers.Dropout(0.1))
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.Dropout(0.1))
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dropout(0.1))

    model.add(layers.Dense(1, activation='sigmoid'))  # Duas saídas lineares para problemas de regressão
```

Figura 4. Arquitetura da rede neural no primeiro treinamento.

O treinamento inicial foi conduzido com 300 épocas, um tamanho de lote (*batch_size*) de 100 e divisão de dados

(*split*) de 50%. No segundo treinamento, houve uma modificação no *split*, reduzindo-o para 30%, enquanto o *batch_size* foi ajustado para 50. Além disso, foram feitas alterações nas camadas densas da rede, como pode ser observado na Figura 5, que apresenta a configuração final das camadas.

```
# Definir a arquitetura do modelo MLP
model = models.Sequential()

model.add(layers.Dense(128, activation='rel... input_shape=(x_train.shape[1],)))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dropout(0.2))

model.add(layers.Dense(1, activation='sigmoid'))  # Duas saídas lineares para problemas de regressão
```

Figura 5. Arquitetura da rede neural do segundo treinamento.

Resultados e Discussão

Como mencionado anteriormente, durante o processo de treinamento foi necessário realizar algumas adaptações nos parâmetros, mudando a porcentagem de validação e modificando as camadas densas.

Nas Figuras 6 e 7 podemos ver os resultados do primeiro treinamento com 50% de validação e quatro camadas densas, sendo uma com 512, outra com 1024, outra com 512 e a última com 128 neurônios. Na Tabela 1 estão descritas as médias obtidas do modelo.

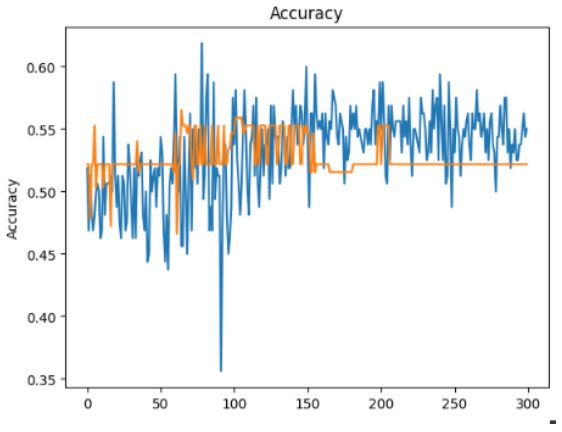

Figura 6. Gráfico de acurácia do primeiro treinamento.

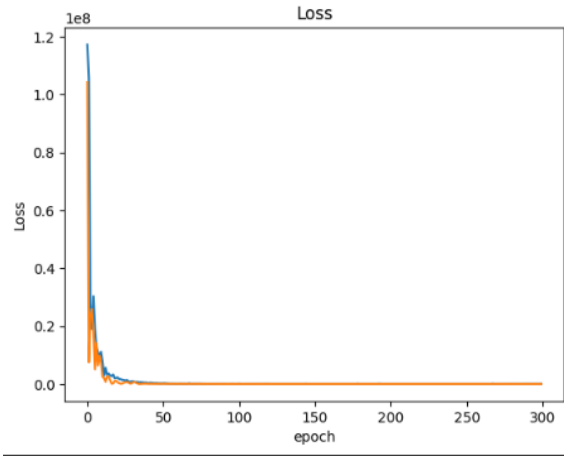

Figura 7. Gráfico de perda do primeiro treinamento.

Para contextualizar as métricas, o modelo é mais consistente quando mais próximo de 1 (um) a acurácia estiver, e quando mais próximo de 0 (zero) a *loss* estiver.

Tabela 1. Médias do primeiro modelo.

Média	Validação
Accuracy	0.527
Loss	810402.024

Como dito na seção anterior, no segundo treinamento, houve a modificação de alguns parâmetros. Os resultados desse segundo treinamento são representados graficamente pelas Figuras 8 e 9.

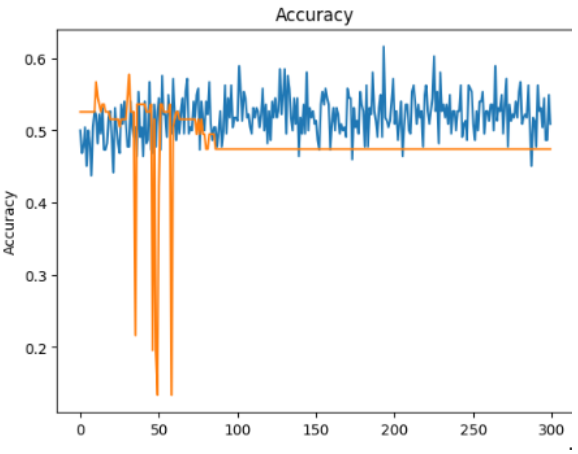

Figura 8. Gráfico de acurácia do segundo treinamento.

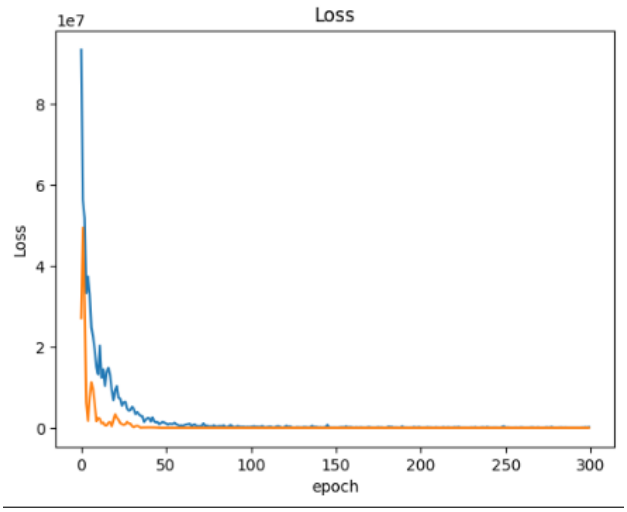

Figura 9. Gráfico de perda do segundo treinamento.

É possível observar, na Tabela 2, que com as mudanças feitas, a acurácia e a *loss* do modelo diminuíram consideravelmente. A diminuição deste segundo parâmetro é algo importante no treinamento de redes neurais.

Tabela 2. Médias do segundo modelo.

Média	Validação
Accuracy	0.481
Loss	605892.011

Considerações Finais

Ao analisar os gráficos de todos os treinamentos, é evidente uma falta de consistência neles. Isso sugere que a causa pode estar relacionada ao tamanho reduzido do conjunto de dados. Uma possível solução seria ampliar a coleta de dados e incorporá-los ao *dataset* existente, seguida da realização de novos treinamentos. Além disso, observa-se que as médias não atendem às expectativas, mas acredita-se que isso poderia ser resolvido com a atualização do conjunto de dados. Como trabalho futuro, seria viável a implementação em um ambiente real.

Agradecimentos

Ao Instituto Federal de Mato Grosso do Sul pela bolsa de estudos concedida durante o período de pesquisa do presente projeto.

Referências

GARRETT, Filipe. **O que é QoS? Entenda para que serve a tecnologia em roteadores.** Techtudo. 2018. Disponível em: <https://www.techtudo.com.br/noticias/2018/07/o-que-e-qos-entenda-para-que-serve-a-tecnologia-em-roteadores.ghtml> Acesso em: 25 de setembro. 2023.

HENRIQUE, Samuel. **Wireshark na Análise de Tráfego e Protocolos em Redes.** labCisco. 2018. Disponível em: <https://labcisco.blogspot.com/2013/11/wireshark-na-analise-de-trafego-e.html> Acesso em: 25 de setembro. 2023.

WIKIPÉDIA. "**Perceptron multicamadas**". Disponível em: <https://pt.wikipedia.org/wiki/Perceptron_multicamadas> Acesso em: 25 de setembro. 2023.

CLASSIFICATION OF CORPORATE NETWORK TRAFFIC: AN APPROACH WITH ARTIFICIAL NEURAL NETWORKS WITH MLP

Abstract: *This article addresses the importance of Quality of Service (QoS) that plays a vital role in data traffic management, especially for latency-sensitive services like VoIP. The integration of artificial intelligence, such as the Multilayer Perceptron Neural Network (MLP), is a promising solution to enhance Quality of Service (QoS) due to its versatility in handling complex and non-linear data patterns. This article examines how artificial intelligence, specifically MLPs, can improve QoS in computer networks. We will discuss data collection and processing, method implementation, neural architecture creation, network training, and result presentation. This approach has the potential to significantly enhance network performance, ensuring a superior user experience in high-data-traffic environments.*

Keywords: *Neural networks, MLP, QoS.*