MODELO PREDITIVO PARA O CÔMPUTO DOS ARCOS DE TRANSIÇÃO EM CIRCUITOS BOOLEANOS: UMA APLICAÇÃO DE REDES NEURAIS

ANTÔNIO ARAÚJO DE BRUM[1]; GUSTAVO CUNHA KNEIP[2]; MATHEUS RENAN FREITAS DE FREITAS[3]; MAICON SCHNEIDER CARDOSO[4]

[1]*Universidade Federal de Pelotas – aadbrum@inf.ufpel.edu.br*
[2]*Universidade Federal de Pelotas - gckneip@inf.ufpel.edu.br*
[3]*Universidade Federal de Pelotas - mrffreitas@inf.ufpel.edu.br*
[4]*Universidade Federal de Pelotas - mscardoso@inf.ufpel.edu.br*

1. INTRODUÇÃO

O cálculo dos arcos de transição de um determinado circuito lógico é um clássico problema estudado em diferentes áreas relacionadas à computação e engenharia elétrica. Sumariamente, um arco de transição representa uma alteração em um parâmetro de entrada do circuito capaz de provocar uma alteração no valor lógico da saída. Esse paradigma possui implicações no desenvolvimento de circuitos lógicos, otimização da eficiência energética (ROY; PRASAD, 1993), e, mais fundamentalmente, na simulação do comportamento desses circuitos em modelos computacionais(MEDEIROS, 2014).

Uma calculadora de arcos de transição (que será referida com o nome "*AT*"), desenvolvida anteriormente pelos autores deste trabalho, foi capaz de resolver o problema com considerável eficiência. O AT recebe uma expressão booleana em formato de soma de produtos, transforma-a em uma tabela verdade e então produz, na saída, os arcos de transição.

Nesse artigo, busca-se uma alternativa à solução iterativa (AT) para esse problema, tal que se mitigue a complexidade exponencial para a quantidade de variáveis de entrada. Inspirado na aplicação de (MONDAL et al., 2019), que busca atingir a minimização de uma expressão lógica através de mapas de *Karnaugh* - e superou, satisfatoriamente, suas contrapartes iterativas - é proposto um modelo de rede neural que seja capaz de, após o treinamento, retornar os arcos de transição para tabelas verdade de *n* variáveis em tempo constante.

2. METODOLOGIA

Seguindo o fluxograma da figura 1, a primeira etapa para elaboração da rede consiste na obtenção dos dados de treino. Para tal, utilizou-se o AT (disponível no GitHub, no repositório "*gckneip/AT*"), adaptado para fornecer os arcos a partir da palavra binária que representa a saída de uma tabela verdade e retornando outra palavra binária onde o numero 1 indica que há um arco de transição e 0 indica que não há arco de transição. A figura 2 exemplifica de forma didática a proposta de aplicação da rede neural.

A rede neural foi implementada na plataforma *Google Colab* utilizando Pytorch, possui 2 camadas escondidas com 32 neurônios cada, foi treinada em 30 épocas, e utiliza de validação cruzada k-fold com $k = 10$, como proposto por (RASCHKA, 2017) em seu livro. A arquitetura proposta prevê que a camada de entrada da rede tenha um neurônio para cada *bit* da palavra binária representante da tabela verdade; o mesmo valendo para a camada de saída, que terá um neurônio para cada possibilidade de

arco de transição. Para tal, foi necessário fixar o número de variáveis booleanas, a fim de manter a camada de entrada e de saída em tamanhos fixos.

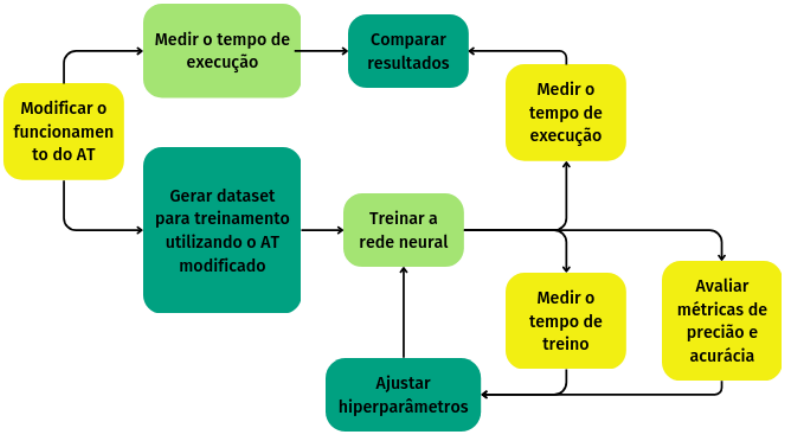


Figura 1: Metodologia proposta

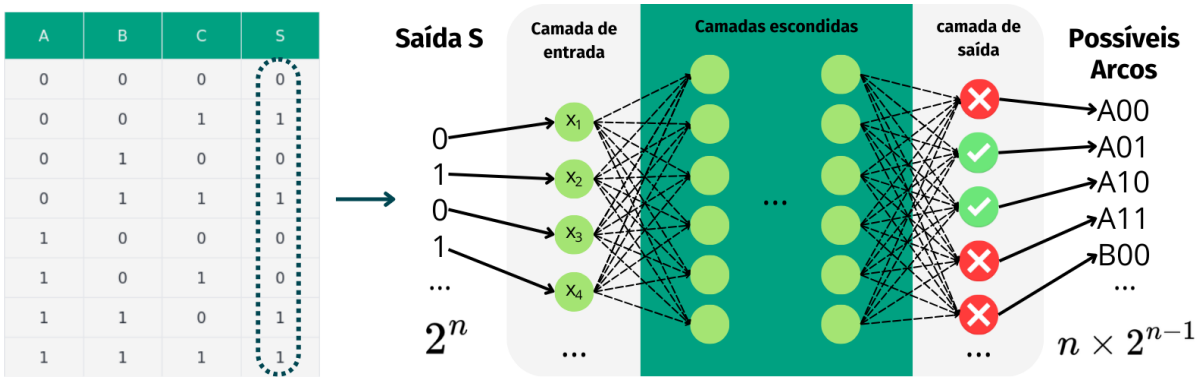


Figura 2: Diagrama simplificado da arquitetura proposta para a rede neural

Por questões de limitações no armazenamento do banco de dados e do tempo de treino disponível na plataforma do *colab*, as variáveis foram fixadas em 4. Gerando assim 16 neurônios de entrada e 32 de saída. Além disso, testar a eficiência em menos fatores indica, a depender do resultado, se valerá a pena trabalhar com mais variáveis.

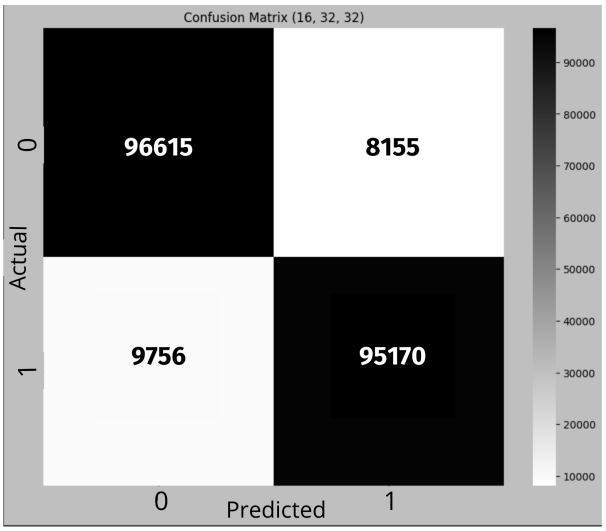
3. RESULTADOS E DISCUSSÃO

As figuras 3a e 3b e a tabela 1 a seguir mostram as métricas de avaliação obtidas, utilizando dos parâmetros mais comuns para avaliação de uma rede neural.

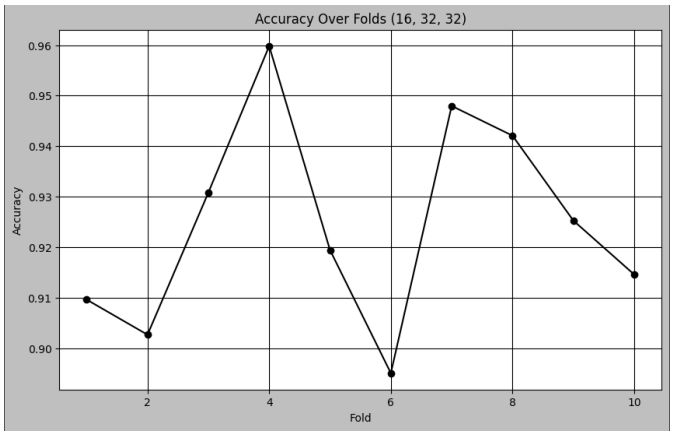
A precisão de 100% sem *overfitting*, naturalmente, não era esperada. (MONDAL et al., 2019) propõe uma maneira de minimizar o impacto das imprecisões do modelo através da passagem dos casos previstos incorretamente como conhecimento prévio do modelo. Essa abordagem ora indicada não beneficiaria o modelo aqui proposto pois uma boa aproximação da ativação dos arcos pode apresentar maior utilidade do que uma minimização lógica incorreta, em especial para tarefas de estimativa. A métrica de perda logarítmica não está ideal (deveria estar mais próxima de 0), mas ainda é justificável por tratar-se de um problema de classificação com 32 saídas.

Tabela 1: Métricas de Performance

Métrica	Valor
Average Accuracy	0.92
Average Precision	0.918
Average Recall	0.918
Average F1 Score	0.918
Average ROC-AUC	0.918
Average Log Loss	1.30



(a) Matriz de confusão para o modelo treinado



(b) Média de acurácia por *fold*

Figura 3: Resultados dos experimentos: (a) Matriz de confusão, (b) Acurácia por *fold*

Diante disso, é avaliado o desempenho do modelo em relação ao algoritmo original (*AT*), com resultados demonstrados na tabela 2. A avaliação foi feita através da ferramenta *GNU time*, rodando na plataforma *GNU/Linux*. Os resultados obtidos indicam que, para essa quantidade de variáveis, há ampla vantagem para a solução desenvolvida anteriormente.

Há que se considerar, todavia, que trata-se de uma comparação intrinsecamente enviesada, haja vista que o *software* anterior foi desenvolvido em uma linguagem mais

ágil, utilizando técnicas e instruções otimizadas para execução em baixo nivel; em contrapartida, a saída da rede neural passa por uma linguagem interpretada, apresentando considerável *overhead* de execução.

Tabela 2: Comparativo de tempos de execução

Programa	Tempo (segundos)
AT (C)	0.04
AiT (Python, RN)	17.00

4. CONCLUSÕES

É preciso observar que a quantidade de tabelas verdade cresce exponencialmente em relação à quantidade de variáveis. A ideia inicial deste artigo consistia em analisar, ao menos, 6 variáveis de entrada. Ao gerar o *dataset* para essa configuração de entrada, logo descobriu-se que o armazenamento necessário para armazenar quantidade significativa desses dados ultrapassaria algumas centenas de *gigabytes*. Faz-se necessário estudo mais cauteloso para afirmar categoricamente se esse modelo pode ser implementado da mesma maneira para circuitos cuja quantidade de entradas extrapola, por exemplo 10 variáveis.

Esse trabalho seria amplamente beneficiado por uma representação mais enxuta dos dados de entrada, o que poderia ser uma frente de estudos para uma trabalho futuro. Além disso, crê-se que há potencial para ajustes técnicos no modelo, tais que sejam capazes de aumentar ainda mais a precisão da predição dos arcos.

Por fim, é importante ressaltar que trabalhos que envolvem aprendizado de máquina (ML) possuem muito espaço em diversas aplicações na área da microeletrônica. Ainda que o modelo em questão não tenha performado bem, principalmente com relação ao AT, as ideias futuras envolvem buscar maneiras melhores de aplicar as técnicas de ML tanto no problema dos arcos de transição como em muitos outros neste campo.

5. REFERÊNCIAS BIBLIOGRÁFICAS

MEDEIROS, GUILHERME ESPINDOLA. **EVOLUÇÃO DA FERRAMENTA LICHEN DE CARACTERIZAÇÃO DE CÉLULAS PARA PROJETO DE CIRCUITOS ASSÍNCRONOS**. 2014. Tese (Doutorado) – Pontifícia Universidade Católica do Rio Grande do Sul.

MONDAL, Shanka Subhra et al. **KarNet: An Efficient Boolean Function Simplifier**. 4 jun. 2019. DOI: 10.48550/arXiv.1906.01363. arXiv: 1906.01363 [cs]. Acesso em: 25 ago. 2024. Pré-publicado.

RASCHKA, Sebastian. Learning Best Practices for Model Evaluation and Hyperparameter Tuning. In: PUBLISHING, Packt (Ed.). **Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow, 2nd Edition**. [S.l.]: Packt Publishing, 2017. cap. 6, p. 186–219.

ROY, K.; PRASAD, S.C. Circuit activity based logic synthesis for low power reliable operations. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 1, n. 4, p. 503–513, 1993.