RESILIÊNCIA DO CLASSIFICADOR DE N-UPLA WISARD

Bernardo Cid Killer Soares de Souza

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Felipe Maia Galvão França

Rio de Janeiro
Março de 2019

RESILIÊNCIA DO CLASSIFICADOR DE N-UPLA WISARD

Bernardo Cid Killer Soares de Souza

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Felipe Maia Galvão França, Ph.D.

Prof. Claudio Luis de Amorim, Ph.D.

Prof. Alexandre Solon Nery, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2019

Souza, Bernardo Cid Killer Soares de

Resiliência do classificador de n-upla WiSARD/Bernardo Cid Killer Soares de Souza. – Rio de Janeiro: UFRJ/COPPE, 2019.

XV, 84 p.: il.; 29,7cm.

Orientador: Felipe Maia Galvão França

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2019.

Referências Bibliográficas: p. 82 – 84.

1. WiSARD. 2. Error Tolerance. 3. Fault Injection. 4. Software Resilience. 5. Machine Learning. I. França, Felipe Maia Galvão. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Gostaria de agradecer a meus professores de departamento, a meu orientador e aos colegas de laboratório que me acompanharam neste trabalho. E um agradeciento especial a meu amigo José Hugo, que me apoiou bastante na reta final.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

RESILIÊNCIA DO CLASSIFICADOR DE N-UPLA WISARD

Bernardo Cid Killer Soares de Souza

Março/2019

Orientador: Felipe Maia Galvão França

Programa: Engenharia de Sistemas e Computação

Conforme as tecnologias de aprendizado vão avançando e ganhando mais espaço na sociedade moderna, se torna mais importante garantir que estas não falhem de forma invisível. Considerando que os hardwares atuais são propícios a falhas em determinadas situações, tanto de execução quanto devido a condições ambientais, é importante analisar o impacto que estas falhas podem provocar em um modelo já implementado. Considerando isso, buscamos analisar neste trabalho o efeito de diversos tipos de erro sobre o desempenho do modelo de rede sem pesos WiSARD. O foco do trabalho está na avaliação impacto de erros do tipo bit-flip em diversas partes da WiSARD, principalmente nos parâmetros treinados do modelo e como este erros se comparam ao de outros tipos de erro. Considerando hiperparâmetros de rede e pré-processamento fixos, são medidas as quedas de acurácia do modelo para sete conjuntos de dados, usando diferentes taxas de injeção de erro e padrões de erro.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

THE ROBUSTNESS OF THE WISARD N-TUPLE CLASSIFIER

Bernardo Cid Killer Soares de Souza

March/2019

Advisor: Felipe Maia Galvão França

Department: Systems Engineering and Computer Science

Just as machine learning technologies are advancing and gaining more space in modern society, it becomes more important to warrant these do not fail without warning. Considering that today's hardware is prone to failure or error under certain under certain conditions, execution errors as well as errors due to environmental conditions, it becomes important to analyse the impact these errors might have over an already implemented model. Considering this, in this work we seek to analyse the effects of different types of error over the WiSARD weightless neural network model's performance. This work is focussed on bit-flip errors in different parts of the WiSARD network, with special attention to errors in the model's trained parameters and how this kind of error compares with others. For fixed network hyperparameters and preprocessing procedures, the accuracy drop of the model is taken for seven different datasets, considering different error injection rates and error patterns.

Sumário

Lista de Figuras **ix**

Lista de Tabelas **xv**

1 Introdução **1**

2 WiSARD **3**

2.1 Estrutura e Funcionamento 4

2.2 Implementação 8

3 Resiliência, Robustez e Segurança **10**

3.1 Resiliência 11

3.2 Robustez 12

3.3 Recuperabilidade 12

3.4 Corrigibilidade 13

3.5 Injeção de falhas 13

4 Experimentos **16**

4.1 Metodologia 16

4.2 Binarização 21

4.3 Resultados dos experimentos de injeção de erros nos parâmetros 22

4.3.1 Injeção de ruído em bits randômicos 23

4.3.2 Análise da Moda da queda de acurácia 27

4.3.3 Injeção de ruído em bits fixos 32

4.3.4 Injeção de ruído de menor valor em bits randômicos 36

4.3.5 Verificação do efeito no número de RAMs sobre a queda de acurácia 41

4.4 Resultados dos experimentos de injeção de erros no treino 45

4.5 Resultados dos experimentos de injeção de erros na execução 57

4.6 Resultados dos experimentos com conjunção dos erros 69

4.7 Comparação de Modelos 73

5 Conclusão **79**

Referências Bibliográficas **82**

Lista de Figuras

2.1 Binarização de entradas para a WiSARD. 4
2.2 Organização geral da WiSARD. 5
2.3 Funcionamento da Retina. 6
2.4 Endereçamento de RAMs utilizando n-uplas (partes combinadas da entrada.) 7
2.5 Treinamento das RAMs em um Discriminador. 8

4.1 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Iris. 23
4.2 Queda de acurácia para seis valores de taxa de injeção de erro no dataset MNIST 24
4.3 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Adult. 24
4.4 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Bank. 25
4.5 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Credit. 25
4.6 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Survival. 26
4.7 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Zoo. 26
4.8 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Iris. 28
4.9 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset MNIST. 29
4.10 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Adult. 29

4.11 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Bank. 30
4.12 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Credit. 30
4.13 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Survival. 31
4.14 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Zoo. 31
4.15 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Iris. 33
4.16 Queda de acurácia para seis valores de taxa de injeção de erro no dataset MNIST. 33
4.17 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Adult. 34
4.18 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Bank. 34
4.19 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Credit. 35
4.20 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Survival. 35
4.21 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Zoo. 36
4.22 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Iris. 37
4.23 Queda de acurácia para seis valores de taxa de injeção de erro no dataset MNIST. 38
4.24 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Adult. 38
4.25 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Bank. 39
4.26 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Credit. 39
4.27 40
4.28 Queda de acurácia para seis valores de taxa de injeção de erro no dataset Zoo. 40

4.29 Queda de acurácia para o dataset Iris com entrada extendida. 43
4.30 Queda de acurácia para o dataset Credit com entrada extendida. 43
4.31 Queda de acurácia para o dataset Survival com entrada extendida. 44
4.32 Queda de acurácia para o dataset Zoo com entrada extendida. 44
4.33 Queda de acurácia para erro injetado durante o treino no dataset Iris. . . . 46
4.34 Queda de acurácia para erro injetado durante o treino no dataset MNIST. 47
4.35 Queda de acurácia para erro injetado durante o treino no dataset Adult. . 47
4.36 Queda de acurácia para erro injetado durante o treino no dataset Bank. . . 48
4.37 Queda de acurácia para erro injetado durante o treino no dataset Credit. . 48
4.38 Queda de acurácia para erro injetado durante o treino no dataset Survival. 49
4.39 Queda de acurácia para erro injetado durante o treino no dataset Zoo. . . 49
4.40 Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Irirs. 50
4.41 Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset MNIST. 51
4.42 Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Adult. 51
4.43 Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Bank. 52
4.44 Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Credit. 52
4.45 Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Survival. 53
4.46 Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Zoo. 53
4.47 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Iris. 54
4.48 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset MNIST. 54
4.49 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Adult. 55
4.50 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Bank. 55

4.51 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Credit. 56
4.52 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Survival. 56
4.53 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Zoo. 57
4.54 Queda de acurácia para erro injetado durante a classificação do dataset Iris. 58
4.55 Queda de acurácia para erro injetado durante a classificação do dataset MNIST. 59
4.56 Queda de acurácia para erro injetado durante a classificação do dataset Adult. 59
4.57 Queda de acurácia para erro injetado durante a classificação do dataset Bank. 60
4.58 Queda de acurácia para erro injetado durante a classificação do dataset Credit. 60
4.59 Queda de acurácia para erro injetado durante a classificação do dataset Survival. 61
4.60 Queda de acurácia para erro injetado durante a classificação do dataset Zoo. 61
4.61 Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Irirs. 62
4.62 Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset MNIST. 62
4.63 Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Adult. 63
4.64 Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Bank. 63
4.65 Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Credit. 64
4.66 Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Survival. 64
4.67 Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Zoo. 65

4.68 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Iris. 66
4.69 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset MNIST. 66
4.70 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Adult. 67
4.71 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Bank. 67
4.72 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Credit. 68
4.73 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Survival 68
4.74 Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Zoo. 69
4.75 Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Irirs. 70
4.76 Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset MINST. 70
4.77 Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Adult. 71
4.78 Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Bank. 71
4.79 Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Credit. 72
4.80 Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Survival. 72
4.81 Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Zoo. 73
4.82 Queda de acurácia para o dataset Iris ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística. . . 74
4.83 Queda de acurácia para o dataset MNIST ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística. 75

4.84 Queda de acurácia para o dataset Adult ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística. 75
4.85 Queda de acurácia para o dataset Bank ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística. . . 76
4.86 Queda de acurácia para o dataset Credit ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística. 76
4.87 Queda de acurácia para o dataset Survival ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística. 77
4.88 Queda de acurácia para o dataset Zoo ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística. . . 77

Lista de Tabelas

4.1 Características dos datasets utilizados nos experimentos. As colunas indicam o número de amostras utilizadas para treino, número de amostras utilizadas para teste (cálculo de acurácia), número de classes possíveis, #número de dados categóricos, número de dados com valor float, número de dados com valor inteiro, número de dados com valor booleano e número total de dados por entrada. 19
4.2 Queda de acurácia no teste de Erro de 'Tipo 1'. 23
4.3 Queda de acurácia no teste de 'Erro de Tipo 1' com bits-flip limitado aos 16 primeiros bits. 37
4.4 Características da entrada gerada por cada dataset e da WiSARD treinada nestes. 42

Capítulo 1

Introdução

Com os avanços de técnicas e de algoritmos de aprendizado de máquina, há um grande interesse em se trazer esses tipos de métodos para domínios de tomada de decisão, onde antes costumavam atuar apenas algoritmos tradicionais ou seres humanos. Exemplos disso incluem logística, bancos de investimentos, empresas de transporte, automação industrial[1] e carros autônomos[2], estas últimas casos em que uma decisão errônea pode trazer consequências fatais. Tais sistemas críticos representam um dos domínios mais complicados para a adoção de técnicas de aprendizado e dependem de implementações não só estáveis, mas também confiáveis a longo prazo. Algumas das dificuldades para executar estes modelos em sistemas embarcados[3] são uma mistura de fatores limitantes (como por exemplo, limitação de recursos computacionais), presença de ambientes hostis devido a condições ambientais adversas[4] e prevalência de informação incompleta, o que dificulta consideravelmente a atuação de sistemas de aprendizado.

Hoje, a expansão de sistemas inteligentes está centrada, majoritariamente, sob duas frentes: o de Cloud Computing e IoT/Edge. Cloud computing trata de sistemas de computação de grande volume, cuja principal dificuldade é a escala massiva dos problemas envolvendo enormes volumes de dados em ambientes empresariais, industriais e de internet. Em contrapartida, sistemas de computação do tipo "Internet of Things"(IoT) ou "Edge Computing" tratam da escala oposta, no qual o foco é em um número elevado de sistemas de pequeno porte capaz de sensoriamento e de computação leve, mas sem a necessidade de enviar os dados para uma central de processamento distante.

Edge Computing tem recebido muita atenção recentemente devido às dificuldades encontradas em transferir quantidades[5] cada vez maiores de dados gerados pelo sensoriamento e atuação dinâmica de sistemas de IoT para processamento em Cloud, devido a gargalos na banda e problemas de latência elevada em redes não locais, necessárias para uso efetivo desses sistemas. Mudar a execução de processamento de aprendizado para mais perto do sensoriamento permite contornar a maior parte dos

problemas relacionados à rede, mas isto exige que os sistemas embarcados ou de IoT sejam capazes de executar os processos de inferência dos modelos de aprendizado por longos períodos de tempo e de forma estável. Mesmo que a etapa de inferência seja muito mais simples e barata do que o processo de treinamento, como algoritmo esta não é em geral um processo computacionalmente leve, apresentando um problema sério na maior parte dos sistemas de IoT devido a sua grande limitação energética[3]. Tal limite provém tanto da restrição de potência disponível para execução dos modelos, quanto na qualidade da energia entregue (devido a necessidade de circuitos de tamanho diminuto, possivelmente expostos a ambientes sem controle de temperatura, pressão, vibração, radiação[6] ou outras externalidades).

Uma classe de redes neurais que apresenta bom desempenho, mesmo com uma quantidade relativamente limitada de recursos, é a de Redes Neurais sem Peso, dentro da qual se destaca o modelo WiSARD, o qual será o principal modelo de estudos dessa dissertação. Este modelo permite classificar massas de dados binarizadas sem requerer nenhuma estrutura adicional, utilizando uma estrutura interna de "memórias", as quais funcionam como um ensemble de votação para executar o processo de classificação.

Apesar de já ter sido proposta há mais de três décadas, ainda há, relativamente, pouca literatura explorando este modelo, principalmente análises referentes ao comportamento dessas redes neurais na presença de falhas e alteração inadvertida de seus parâmetros treinados[7]. Assim sendo, nos propomos a investigar como parâmetros de qualidade da rede, especificamente a acurácia, se comporta com a introdução de ruídos de diferentes intensidades na sua estrutura, mais especificamente nos parâmetros treinados da rede.

Capítulo 2

WiSARD

Wilkie, Shonham Aleksander's Recognition Device também conhecido como WiSARD[8] é um modelo de aprendizado de máquina e uma versão específica de uma rede neural sem peso (Weightless Artificial Neural Network -WANN) criada para classificação de padrões. Este apresenta uma estrutura diferenciada, na qual os elementos que funcionaram como os neurônios da rede são baseados em uma estrutura de memórias RAMs endereçadas por n-uplas[9]. É um modelo considerado de compreensão simples e de execução leve, devido ao baixo número de operações realizadas tanto na etapa de treinamento (que não depende de convergência de parâmetros da rede) quanto na de inferência (que demanda baixíssimo poder computacional). Adicionalmente estas operações são em sua maioria operações bitwise, de soma e de leitura e escrita em memória, fazendo o passo a passo realizado pela WiSARD ser fácil de acompanhar. Além disso WiSARD tem também como vantagem ser facilmente traduzida para implementações em hardware (pela estrutura de seus neurônios) e a possibilidade de ser implementada com baixo consumo de memória.

Dois pontos interessantes deste modelo são: a velocidade tanto de treino e inferência que o modelo pode apresentar e a possibilidade de, no processo de validação, efetuar um teste do tipo "Leave One Out" sem necessidade de refazer o treino modelo, tornando o processo muito mais rápido. Entre suas desvantagens existe o fato de que o modelo funciona apenas como um classificador e de que as entradas devem necessariamente ser binarizadas (convertida em uma bitstream) para se treinar o modelo ou utilizá-lo para classificação. Ainda assim o modelo WiSARD apresenta alto desempenho e diversas tarefas e já foi aplicada em diversas tarefas como reconhecimento de placas [10], análise de crédito[11], acompanhamento de música [12] e clusterização em tempo real de streams de dados[13], dentre outros[14], [15], [16].

Esta seção é dividida em duas partes. Na subseção 2.1 o funcionamento e estrutura interna da WiSARD serão descritos e exemplificados. Na subseção 2.2 serão apresentados alguns detalhes sobre diferentes opções de implementação da rede.

2.1 Estrutura e Funcionamento

Uma rede WiSARD, por ser um classificador, é uma função que recebe um conjunto de entradas e retorna uma categoria (classe), a qual ela considera ser a mais provável de corretamente identificar estas entradas. No caso da rede WiSARD, os dados de entrada devem necessariamente ser pré-processados em alguma codificação binária e todos seus valores concatenados até se formar um array de binários (bitstream) de tamanho fixo, definido na criação da rede.

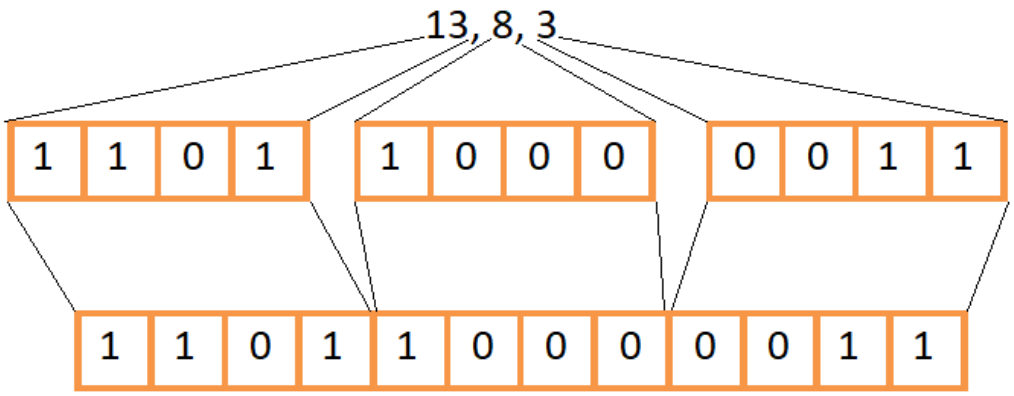


Figura 2.1: Binarização de entradas para a WiSARD.

A WiSARD, como modelo, apresenta três partes distintas. A primeira parte é a entrada do modelo. Nesse ponto os bits de entrada são rearranjados e divididos em grupos que chamamos de n-uplas para serem utilizados na segunda parte.

A segunda parte é composta por um conjunto de Discriminadores, um para cada classe que o modelo pode usar como resposta. Um Discriminador, no caso, é um conjunto de tabelas de valores, as quais chamamos de RAMs, devido a seu comportamento ser semelhante ao das memórias de computadores modernos, recebendo um valor de endereçamento e podendo se ler ou se escrever nesta posição da tabela (ou memória se considerar a analogia). Todos os Discriminadores recebem cada uma das n-uplas separadas na entrada do modelo e cada n-upla é utilizada em cada Discriminador para endereçar uma RAM distinta.

As RAMs, por sua parte, funcionam como os neurônios da WiSARD. Entretanto, ao invés de apresentar uma função de ativação convencional, cada RAM recebe valores em cada uma de suas entradas (posições de memória da RAM). Como uma memória RAM tradicional, as RAMs da WiSARD funcionam como tabelas endereçadas pelas n-uplas, isso é, o valor da n-upla indica em qual posição da RAM um valor será lido ou escrito. A etapa de treino da WiSARD se baseia em inserir esses valores nas RAMs e a execução do modelo se baseia em ler os valores das RAMs. Desse modo cada RAM funciona como uma função não linear com formato dependente do treinamento, algo que

uma rede neural tradicional necessitaria de diversos neurônios para obter. Os valores guardados em cada entrada da RAM são do tipo inteiro sem sinal (unsigned int).

A terceira parte do modelo é a saída. Aqui os valores das RAMs dos Discriminadores são contabilizados e um valor é devolvido como resposta. Pode-se dizer que os valores das RAMs são contabilizados dentro dos Discriminadores e a saída apenas compara os valores para dar a resposta. Em todo caso, as duas situações são equivalentes para o usuário.

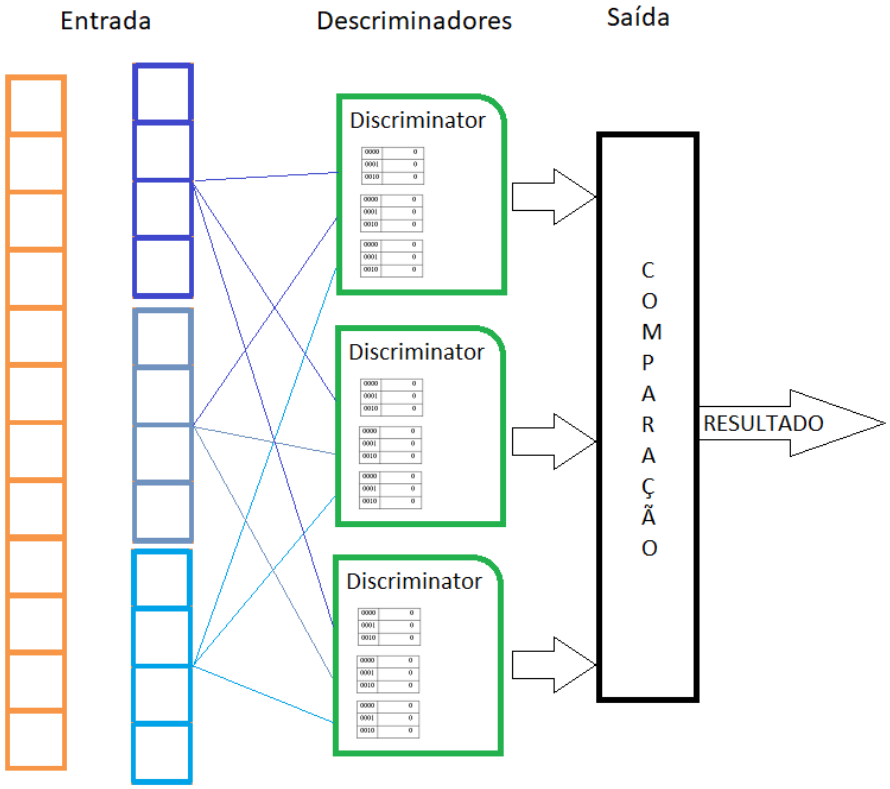


Figura 2.2: Organização geral da WiSARD.

Como todo modelo de classificação, existem dois modos de operação: treinamento e inferência. Descrevemos primeiro o processo de treinamento em conjunto com a descrição da arquitetura da rede WiSARD.

O treinamento apresenta 3 etapas, considera que os valores a serem treinados já estão pré-processados por algum método de binarização e que a classe de cada instância de treino é conhecida. Na primeira etapa os dados passam por uma estrutura conhecida como Retina a qual opera como um scrambler nos dados, permutando as posições dos bits dentro da bitstream, sempre com o mesmo padrão de permutação de posição (para que haja consistência nas informações passadas). Sua necessidade pode não ser muito intuitiva, mas o scramble gerado melhora os resultados do modelo consideravelmente.

Numa segunda etapa o bitstream gerado pela Retina é quebrado em sequências

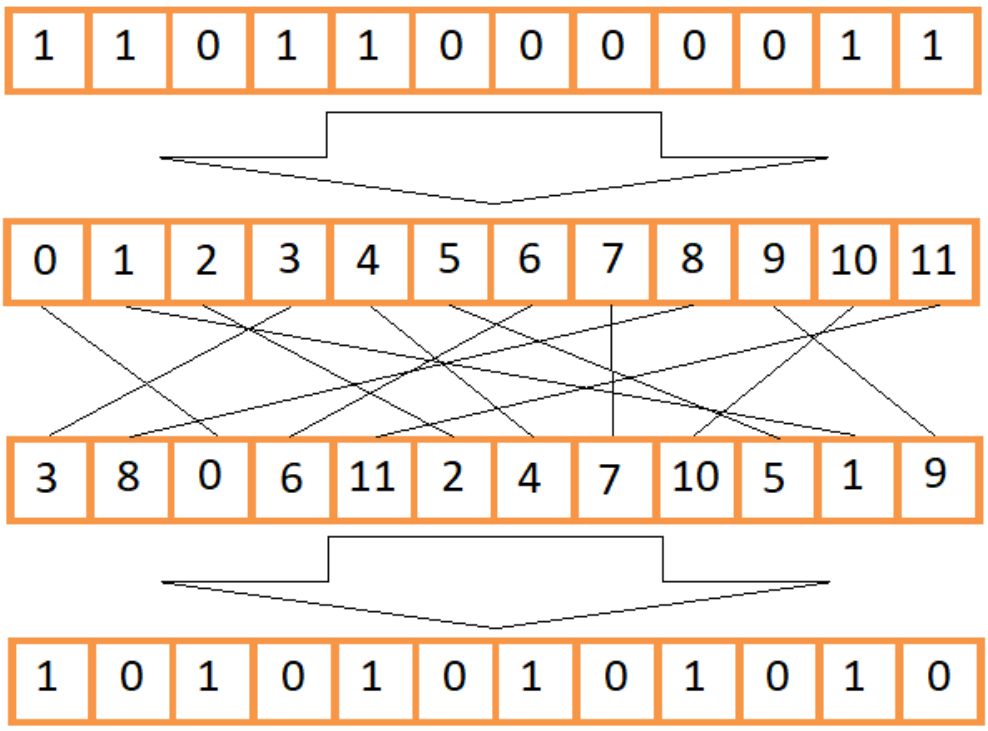


Figura 2.3: Funcionamento da Retina.

de N bits, que chamamos de n-uplas. As n-uplas são interpretadas como um inteiro que será usado para endereçar as RAMs de cada Discriminador. Por causa disso as n-uplas (ou a seleção dos bits para a n-upla) podem ser interpretadas como features que são alimentadas para o modelo, e as RAMs como neurônios que analisam cada feature. Algumas formas de explicar o funcionamento da Retina definem a primeira e segunda partes juntas. Desse modo a Retina é vista como um processo que seleciona bits em posições não sequenciais da entrada e os agrupa de modo a formar um número binário . De qualquer modo em que estas dua ações sejam organizadas no sistema final a presença da Retina é tida como a tomada dessas duas ações independente de sua implementação específica.

Na terceira etapa um Discriminador, que contém um subconjunto de RAMs, é selecionado de acordo com a classe que aquela entrada pertence. Cada RAM deste discriminador é então endereçada utilizando os valores de uma das n-uplas separadas na etapa anterior. Cada entrada selecionada das RAMs tem então seu valor setado para 1 ou incrementado em 1. A implementação tradicional apenas marca com 1 a posição acessada da RAM. Entretanto fazer a contagem de acessos (incremento do valor) permite utilizar algumas técnicas interessantes como o Bleaching, que será discutida mais adiante. Com essa mudança de valor cada Discriminador é treinado com informação referente a uma única classe, tornando-se um “especialista” em relação aos padrões e estatística da classe.

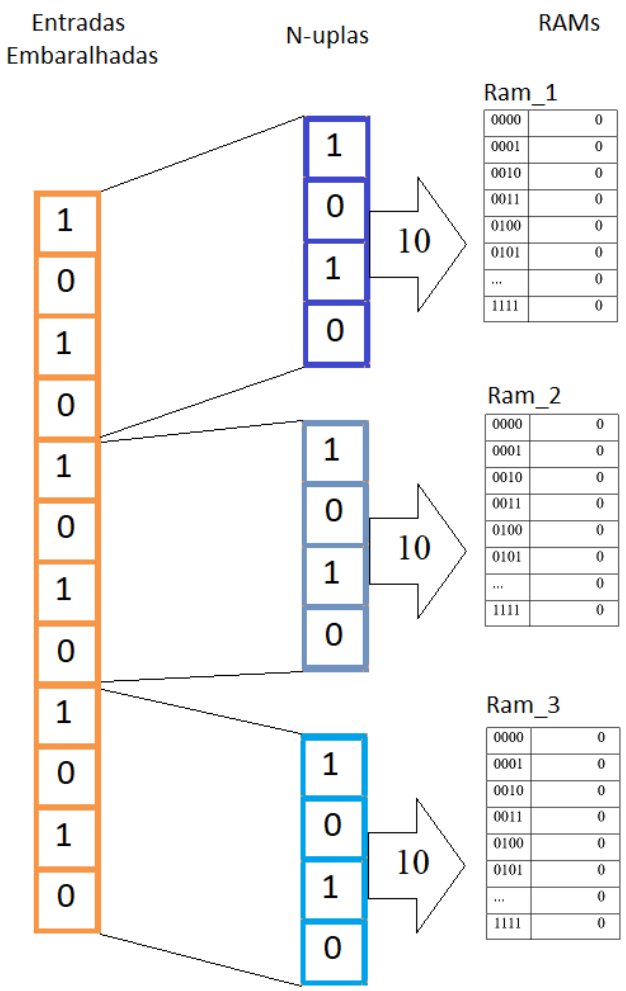


Figura 2.4: Endereçamento de RAMs utilizando n-uplas (partes combinadas da entrada.)

No segundo modo de operação, que é o de inferência, são geradas classificações de novos dados de entrada. Aqui as entradas são entendidas como padrões e o processo de classificação é tido como a análise de a que classe treinada o padrão mais se assemelha. Este processo repete a primeira e segunda etapas descritas no processo de treinamento. No entanto, ao chegar na terceira etapa, ao invés de se utilizar n-uplas para endereçar as RAMs de apenas um Discriminador, estas são utilizadas em todos os Discriminadores e os valores das respectivas RAMs são lidos.

Os resultados das RAMs são então observados e conta-se o número de RAMs cuja posição endereçada apresenta valor guardado diferente de 0. O Discriminador que tiver o maior número de RAMs com algum valor treinado é eleito como representativo da classe correta. Esse processo equivale a um processo de votação no qual cada RAM é capaz de votar apenas no próprio Discriminador. Cada RAM vota se aquela entrada parece ser representativa daquela classe e o Discriminador com maior número de votos é considerado como o resultado da rede.

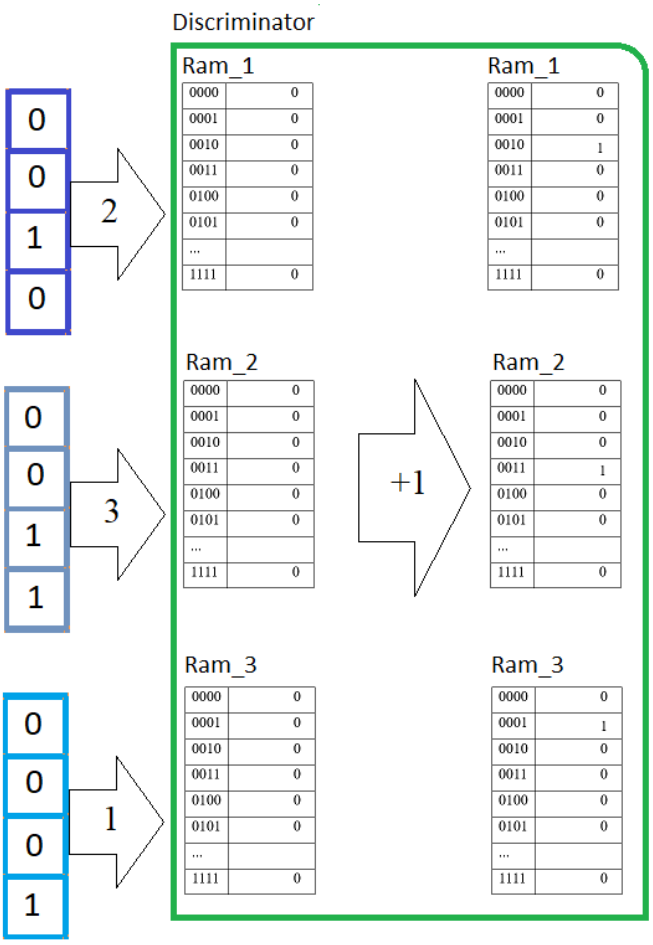


Figura 2.5: Treinamento das RAMs em um Discriminador.

Em caso de empate é aplicado um processo conhecido como Bleaching[17]. Este processo equivale a um sistema de votação por etapas. Neste é imposto um limiar de valor 2 e as RAMs com ativação menor que este valor são desconsideradas no processo de votação. Em novo caso de empate o valor de limiar é acrescentado em 1 e o processo se repete. Em um caso extremo, em que não seja possível um desempate, uma classe é selecionada randomicamente como ouput.

2.2 Implementação

Normalmente apenas consideramos relevantes a lógica e os parâmetros de um modelo, entretanto como trataremos de uma questão de introdução de erros é importante atentar a alguns detalhes sobre o funcionamento e implementação da WiSARD.

Primeiramente cada entrada da WiSARD é composta de B bits, onde o número B depende do número de parâmetros da entrada original e do número de bits utilizado para codificar cada um destes parâmetros (que não necessitam ser codificados do mesmo modo). As entradas são quebradas em R n-uplas utilizadas para endere-

çar as R RAMs de cada Discriminador. Deste modo cada n-upla é formada por N bits, onde $N = ceil(B/R)$. Como B/R muito facilmente não terá um resultado inteiro, é normal a última n-upla ser completada ou pela repetição de uma das entradas ou por bits fixados com valor 0 para manter todas as n-uplas com o mesmo tamanho (apesar de que esta poderia ser implementada como uma única n-upla menor). Inversamente pode-se selecionar N e calcular R a partir deste. Como o modelo contém C Discriminadores, onde C é o número de classes treinadas no modelo a WiSARD é composta por $C \times R$ RAMs e cada RAM por ser endereçada por N bits apresentando 2^N entradas, total resulta em $C \times R \times 2^N$ entradas.

Este número pode crescer muito rápido e ser custoso em termos de memória. Entretanto é comum os valores treinados em uma dada RAM serem bem esparsos. Por esse motivo é comum haverem implementações que trocam parte da velocidade da WiSARD por utilizar dicionários ou estruturas de rash ao invés de tabelas no lugar de cada RAM. O primeiro caso (em que as RAMs são tabelas completas) é uma situação que se aproxima mais de uma implementação em hardware da rede. Neste trabalho trataremos das duas implementações querendo verificar, dentre outras coisas, se o número maior de posições em que erro pode ser introduzido dará alguma vantagem ou desvantagem em estabilidade do desempenho da rede para a implementação em tabela (neste caso específico, um array).

Capítulo 3

Resiliência, Robustez e Segurança

Não se é esperado que sistemas reais de engenharia operem com perfeição (com precisão absoluta) em todos os instantes. No âmbito de engenharia mecânica, por exemplo, sistemas de engrenagem e de corte são projetados para operar com erro mínimo desde que suas dimensões e modo de operação estejam dentro de valores de tolerância, estes mesmos definidos no projeto. Esses graus de tolerância podem variar muito, mas é de interesse que quando não sejam obedecidos ou que o sistema seja submetido a condições inesperadas que este não pare simplesmente de funcionar e "quebre". No caso de softwares um dos conceitos de qualidade é a fidedignidade. Para ser fidedigno um sistema de software deve atender um conjunto de especificações que façam este ser confiável, de modo que se possa depender de sua atuação. Dentro deste grande conceito, um aspecto importante é a tolerância a erro. Esta define ou qualifica a capacidade de um software de continuar seu funcionamento normal mesmo quando existam erros de software ou hardware. Situações em que tais erros, tanto de hardware quanto de software, são esperados, incluem sistemas embarcados, sistemas em satélites, sistemas em aviões e sistemas computacionais que funcionem em alta frequência. No caso de sistemas embarcados, efeitos ambientais podem gerar erros pelo efeito momentâneo ou contínuo de campos magnéticos ou condições ambientais. Em sistemas de satélite e de aeronaves efeitos de campos magnéticos de alta potência ou íons que passam pela atmosfera podem causar dano em circuitos ou a mudança ou fixação de bits individuais na memória dos computadores. Finalmente, em computação de alto desempenho[18] (ou computação que utiliza hardware menos potentes levados ao seu limite) é comum haver aumento de clock ou possíveis desbalanceamentos de potência que adicionam erros em algumas etapas do processamento.

No contexto específico de problemas de classificação em aprendizado supervisionado, tolerância a falhas pode ser visto como a capacidade do modelo continuar classificando corretamente (ou tão corretamente quanto já era capaz) suas entradas. Ou seja, de manter constante ou com baixa queda sua acurácia. Isso entretanto não é fácil pois mudanças, por exemplo, nos parâmetros do modelo podem modificar forte-

mente sua qualidade. A sensibilidade do comportamento final de um modelo a erros (que apareçam tanto em sua execução quanto por deterioração de seus parâmetros) pode ser pensada como uma forma de medida da fragilidade do sistema.

Com a maior adoção de sistemas de Machine Learning em diversas áreas é de grande interesse que os modelos utilizados nestes sistemas sejam tolerantes a erro e fidedignos. Caso não sejam, tais modelos não poderão ser utilizados em aplicações em sistemas críticos. Sistemas críticos são definidos como sistemas em que qualquer falha implica um problema de grave importância e portanto não pode ser tolerada. Em um carro, por exemplo, o rádio pode ser visto como uma parte (ou subsistema) não crítico para seu funcionamento, por outro lado o motor seria uma parte (subsistema) crítico. Este tipo de sistema costumam ser divididos em quatro classes: críticos em integridade (safety critical), críticos em negócios (business critical), críticos em missão (mission critical) e críticos em segurança (security critical). E abrangem casos tão diversos como carros autônomos, sistemas de aviação civil, firewalls de rede empresariais, sistemas de detecção de falhas automáticas, sistemas de detecção de fraudes e diversos outros que impactam o funcionamento de negócios, a segurança de pessoas e a saúde de indivíduos.

Nas sessões 3.1, 3.2, 3.3 e 3.4, falaremos sobre os conceitos de “Resiliência”, “Robustez”, “Recuperabilidade” e “Corrigibilidade”, todos estes, conceitos referentes à tolerância a falhas e relevantes em diferentes aplicações. Por fim na seção 3.5 fecharemos o escopo dos pontos de interesse de tolerância a falhas para o trabalho desenvolvido.

3.1 Resiliência

Resiliência é, no contexto de computação, definido como a capacidade de um sistema de continuar trabalhando de forma adequada sob condições anormais de funcionamento. Isto pode ocorrer quando por influências aleatórias na execução de um software, por exemplo, provenientes de ruído na eletrônica subjacente ou mesmo, em casos mais dramáticos, como falhas de componentes inteiros do sistema de computação. A resiliência, neste caso, seria a capacidade de executar um comportamento próximo o suficiente do esperado para que a experiência do usuário ou aplicação industrial ou empresarial envolvida não saísse da especificação ao ponto de implicar grandes perdas ou danos.

Uma dicussão, no contexto de resiliência do comportamento do sistema, é a resiliência a ruídos internos como: o efeito de um bit-flip, trepidações devido à influências externas, falhas de sensores, perda de informações devido a falhas de comunicação ou uma atualização mal feita, falhas do sistemas de computação por perda de hardware ou a queda de nós inteiros de uma rede. Nesses casos, os sistemas são considerados resilientes se não saírem do ar nem não perderem sua funcionalidade sob estas condi-

ções. Uma rede resiliente continuará a enviar informações para nós ativos ou buscará caminhos alternativos de comunicação caso alguns nós caiam. Um vídeo de serviço de streaming diminuirá a qualidade do vídeo enviado caso a conexão seja instável e dificulte a transmissão correta dos pacotes. Um carro autônomo não pararia inesperadamente caso um de seus sensores parasse de responder ou um de seus processadores queimar.

3.2 Robustez

Robustez, por sua vez, é definida como a habilidade de um sistema, durante seu funcionamento, detectar e reportar falhas[19], de modo que os danos dessas sejam limitados. Contextos comuns em que se discute robustez de sistemas incluem proteção de dados em sistemas empresariais contra falhas de discos, degradação de bits, falhas de comunicação em cópia em discos. Outro caso comum em que é discutida a necessidade de robustez é o caso de comunicação digital em canais ruidosos (importante para teoria de informação em telecomunicações). Nesta aplicação, a dificuldade é de se enviar dados de forma exata através de um canal sujeito a ruídos, podendo este ruído ser provenientes do meio ambiente no qual está imerso o canal ou mesmo de problemas de manufatura ou operação do próprio equipamento ao longo do tempo. Sem a capacidade de se detectar esses erros de comunicação para posteriormente corrigi-los seria impossível implementar diversas tecnologias no nível que temos hoje, como telefonia por satélite e transmissão de grandes arquivos pela Internet.

Outros exemplos de robustez são a gerência de memória e recursos por um sistema operacional (que para a execução de um de seus processos caso esse se comporte de forma estranha ou regiões de memória especiais sejam acessadas) e a capacidade um programa detectar inputs inválidos ou uma rotina de teste aplicada em uma rede.

3.3 Recuperabilidade

Recuperabilidade é definida como a capacidade de um sistema voltar à sua funcionalidade esperada de forma rápida e consistente. Isto é uma capacidade desejável em qualquer sistema de engenharia, mas de importância especial em sistemas de alta disponibilidade e em sistemas críticos. Sistemas de alta disponibilidade, como provedores de Cloud, sites corporativos de alto volume de acesso e sistemas que provêm serviços como meteorologia e vigilância precisam estar disponíveis o máximo possível (seja por sua importância social ou financeira) uma vez que uma baixa de um sistema destes pode representar um preço alto tanto monetário quanto de imagem para a empresa.

Atingir recuperabilidade em sistemas reais costuma ser um empreendimento custoso, e, assim sendo, costuma precisar de uma forte motivação, como presente em casos como os exemplificados acima, para justificar o investimento necessário, que muitas vezes envolve implementação de redundâncias, como em sistemas de despacho dinâmico com balanceamento de carga automático, necessário para assegurar que clientes de sistemas com muitos acessos irão conseguir ter acesso simultâneo ao site e com uma experiência satisfatória. Fora redundância, técnicas como aumento de fatores de segurança e protocolos estritos de controle de qualidade são necessários para assegurar este tipo de capacidade, estes todos representando um custo extra agregado para o processo.

3.4 Corrigibilidade

Corrigibilidade, por fim, pode ser definida como a habilidade de um sistema ou dado de ser corrigida de forma fácil e rápida quando detectado um erro. Novamente, comunicação em canais ruidosos são um exemplo excelente de sistema com corrigibilidade. Outro caso relevante é o de memórias com código corretor de erro (Error Correcting Code, ECC), que são amplamente usadas em sistemas de alto desempenho[18], as quais permitem assegurar o uso contínuo de sistemas de computação de grande porte com uma quantidade manejável de erros. Em memórias, erros do tipo *bit-flip* são relativamente raros na escala encontrada em computadores pessoais, mas, quando avaliadas em sistemas com milhares ou milhões de computadores, a probabilidade de uma flip é suficientemente alta para ocorrer múltiplas vezes por dia em um datacenter. Neste tipo de ambiente, é muito razoável "pagar o preço", tanto em dinheiro quanto em performance, de prevenir que este tipo de erro impacte a aplicação.

Um exemplo de sistema (auto-) corrigível são sistemas de arquivo avançados como o ZFS (Zettabyte Filesystem), o qual é capaz de detectar falhas tanto de hardware quanto de software em pools de disco durante sua operação e, muitas vezes, corrigir os dados originais através de combinações de algoritmos de paridade e algoritmos de hashing. Outro são satélites que criam novas configurações utilizando hardware programáveis quando percebem que um de seus dispositivo apresenta uma falha.

3.5 Injeção de falhas

Focando especificamente em software podem ser esperados erros de diversas naturezas. Os mais comuns incluem: erros de comunicação entre partes do modelo (ou sistema ou rede), modificação do código a ser executado, execução de operação erroneamente (3+4 = 9, por exemplo) e aparecimento de ruído tanto em parâmetros do modelo quanto em valores salvos. Dentre estes erros no código (não de programação,

mas pela mudança inesperada de um valor no código já compilado) são altamente destrutivos, pois afetam todo o funcionamento subsequente do modelo. Erros de operação (como somas incorretas ou em que o resultado é substituído por outro inesperadamente) podem gerar tanto erros grandes que quebrem o programa (sobrescrevendo posições indevidas na memória) quanto pequenos que afetem apenas o valor de saída. Por último erros nos valores salvos são muito silenciosos e podem tanto afetar resultados salvos quanto modificar parâmetros do modelo, como pesos de uma rede neural ou centróides de um K-Nearest Neighbors.

Como citado seção 3.2, para garantir tolerância a falhas visando um sistema resiliente, desejamos que nosso modelo se comporte de forma eficiente e similar a seu funcionamento mesmo quando erros estão presentes. Para um modelo de classificação, isso se traduz em uma queda pequena em sua acurácia, ou em outras palavras, como uma baixa degradação de seu poder de classificação em função da intensidade (quantidade) dos erros por ele sofrido. Dentre os três tipos de erros citados anteriormente a degradação do código não faz parte do escopo do modelo, mas sim do ambiente computacional no qual está programado. Os erros de operação podem ocorrer durante sua execução e seu nível de impacto é indeterminado, podendo passar desapercebido ou paralisar o programa. Por último os erros de valor, especificamente dos valores salvos correspondentes ao parâmetros da uma rede WiSARD, fazem parte direta do modelo e tem efeitos diretos e garantidos sobre seu funcionamento. Não só isso, esses erros podem ser mantidos por tempo indeterminado e sem causar falha completa do modelo, podendo passar totalmente despercebidos. Por tal motivo é de interesse entender o efeito desse tipo de erro sobre um modelo, de modo a poder confiar mais ou menos em sua utilização a longo prazo sabendo que este ocorrerá.

No caso da WiSARD consideramos que erros nos parâmetros irá significar alteração dos conteúdos salvo nas RAMs e que estes erros aparecerão na forma de *bit-flips* em posições aleatórias de qualquer parâmetro. Bit-flips, inversão do valor de apenas um bit dentro de uma memória, é um tipo comum de erro principalmente em aplicações espaciais, no qual campos ionizantes fazem diversas alterações na memória salva ao longo do tempo.

Para considerarmos a queda de acurácia do modelo em relação a um modelo base com erros como nossa métrica do efeito de erros aleatórios no desempenho e no funcionamento deste, é necessário entender como esta medida pode ser tirada. Este processo é conhecido como “fault injection” (“injeção de falhas” em português) e trata de, por meio de alterações em hardware ou rotinas extras de software, adicionar anomalias no funcionamento das rotinas executadas. Os testes com erro em hardware são difíceis e custosos de se implementar, por isso o normal é o uso das simulações de erro em software. Programas de injeção de falhas podem simular os três tipos de erro citados anteriormente, mas por muitos deles serem bastante específicos e a estrutura de

cada código que se deseja fazer uma injeção serem bem distintas, é normal que cada programa seja específico para um tipo ou conjunto limitado de tipos de falha.

Neste trabalho analisaremos três tipos de erros que podem ocorrer no funcionamento da WiZARD. O primeiro (e principal), será o da introdução de ruído do tipo *bit-flip* (quando um único bit de uma posição de memória tem seu valor invertido) nos parâmetros treinados de um modelo, focando especificamente no modelo WiSARD. O segundo, será a introdução de ruído do tipo *bit-flip* nos dados que estarão sendo usados para treinar o modelo e por último, introdução de erro na etapa de inferência. Tudo que será necessário para desenvolver estes experimentos é uma implementação do modelo, com acesso a seus parâmetros (para poder modificá-los com valores específicos) e com sub-rotinas que gerem computações incorretas ou inesperadas introduzidas em seu código de modo que possam ser chamadas ou ignoradas conforme desejado.

Capítulo 4

Experimentos

O foco deste trabalho são as análises experimentais que serão descritas nesta seção. Para uma implementação do modelo WiSARD foram determinados quatro tipos de erros de interesse (dois de *bit-flip* nos parâmetros da rede, um durante o treino e um durante a etapa de inferência) e o modelo será rodado para diversas taxas de aparecimento destes erros (que denominamos taxa de injeção de erro ou TE) e para diversas combinações destes erros. Os testes foram efetuados sobre sete datasets diferentes. A escolha dos datasets se deu pela influência de [20] no qual a queda de acurácia é medida em três outros modelos para os mesmos conjuntos de dados que serão utilizados.

Na seção 4.1 a metodologia utilizada será descrita em mais detalhes, na seção 4.2 os métodos de binarização utilizados nos dados serão descritos, na seção 4.3 os resultados do experimento de injeção de falhas nos parâmetros da WiSARD (valores das RAMs) serão apresentados e comentados Na seção 4.7 os resultados da WiSARD serão comparados ao de outros modelos submetidos ao mesmo tipo de erros, na seção 4.4 os resultados do experimento de injeção de erro na WiSARD durante o treinamento serão apresentados e na seção 4.5, os resultados do experimento de injeção de erro durante a execução da WiSARD (na etapa de inferência) serão apresentados. Na seção 4.6 o experimento será repetido desta vez considerando a incidência de todos os tipos de erros testados anteriormente em conjunto.

4.1 Metodologia

O experimento é focado em implementar e treinar uma rede WiSARD base, da qual calcularemos a acurácia para um dado conjunto de teste. Erros de diversos tipos serão, então, acrescentados no modelo testado, com uma dada taxa de injeção de erro TE e a acurácia da rede será medida uma segunda vez para o mesmo conjunto de teste. Este experimento é então repetido 200 vezes sempre repetindo a mesma WiSARD base. O conjunto de 200 repetições então é refeito para diferentes taxas de erro TE e para di-

ferentes tipos de erros injetados. As taxas de injeção de erro (TE) testadas nos experimentos serão sempre dentre os valores 0.5%, 1%, 5%, 10%, 15% e 20%.

Considerando este procedimento experimental foram definidos diversos tipos de erros de interesse. O primeiro tipo é aquele no qual acrescentamos erros (ruído) nos valores das RAMs (que funcionam como parâmetros do modelo WiSARD). Consideramos então que existem duas formas desse erro ocorrer, uma na qual o erro surge em posições randômicas nos valores das RAMs e outra na qual o ruído sempre tem o mesmo valor, isso é, aparece sempre no mesmo bit de uma entrada da RAM. Denotaremos estes erros como 'Erro de Tipo 1' e 'Erro de Tipo 2'. Mais especificamente, no caso do 'Erro de Tipo 1'(seção 4.3.1) a entrada das RAMs a serem modificadas são selecionadas aleatoriamente e um bit destas entradas é selecionado aleatoriamente em cada caso e invertido. Já no caso do 'Erro de Tipo 2' (seção 4.3.3) a entrada das RAMs a serem modificadas são selecionadas aleatoriamente, que nem no outro tipo de erro nas RAMs, entretanto sempre o bit da mesma posição (mais significativo, terceiro menos significativo, etc...) que será modificado em cada instância de erro. Para este tipo de erro o experimento será repetido para cada posição possível de bit, consideramos para o experimento valores do tipo unsigned int de 32 bits. Isto é, consideramos que a implementação da WiSARD utilizada guarda valores de 32 bits em cada entrada de suas RAMs.

O segundo tipo de erro é o que chamamos de erro de treino. Este erro já não ocorre durante a execução do código, mas sim durante seu treinamento apenas. Apesar de diversos tipos de erro poderem ocorrer nesta etapa, neste trabalho consideraremos apenas o erro no qual os bits de uma entrada são lidos com algum ruído do tipo *bit-flip* com taxa TE (lembrando que as entradas da WiSARD são todas binárias) ocasionando com que entradas incorretas das RAMs sejam incrementadas. Denotaremos este erro apenas como 'Erro de Treino'.

Por fim o terceiro tipo de erro que analisaremos será um erro de execução. Novamente existem diversos tipos possíveis de erro de execução (incluindo um análogo ao de treino em que as entradas das RAMs são lidas incorretamente) mas focaremos neste trabalho nos erros de contabilização das RAMs ativas. Todas as RAMs terão uma de suas entradas lidas e, como explicado na seção 2.1, passarão por um processo de Bleaching (que acaba sendo irrelevante nos casos em que não há empate). Após este processo se o valor da RAM for superior ao valor de Bleaching esta deve ser contabilizada como 1 ponto para seu Discriminador e o Discriminador com mais pontos é selecionado como representante da classe da entrada. No caso o erro de execução que consideraremos se baseia em contar erroneamente o número de RAMs ativadas, considerando ou desconsiderando erroneamente RAMs com uma probabilidade TE.

Os experimentos considerarão também que a WiSARD é implementada utilizando arrays como estrutura de dados das RAMs. Para os experimentos isso quer dizer que

qualquer valor de qualquer RAM pode ser modificado, independentemente de ter sido modificado durante a etapa de treino ou não.

Os hiperparâmetros da WiSARD serão definidos também de forma genérica, não estando otimizados para nenhum dataset específico. Dentre estes o número de Discriminadores (classes) C dependerá do dataset a ser testado, não sendo escolhido. Similarmente teremos que o número de RAMs R dependerá da quantidade de parâmetros das entradas, não sendo escolhido mas sendo influenciados pelos outros parâmetros selecionados. Dentre os hiperparâmetros selecionados teremos o número de bits que endereçam as RAMs (tamanho das n-uplas) N que será 12 para todos os datasets. O outro hiperparâmetro escolhido será utilizado no pré-processamento e influenciará a dimensão da entrada, este é o número de bits utilizado pelo método de binarização de termômetro (descrito na seção 4.2) que terá sempre tamanho fixo de 20 bits.

Sobre os dataset testados. São eles: Iris, MNIST, Adult, Bank Marketing, German Credit, Haberman's Survival e Zoo. Todos estes datasets foram obtidos e estão disponíveis no site https//:archive.ics.uci.edu /ml/. Cada um deles apresenta um número diferente de amostras e tipos de valores. Na tabela 4.1 são enumeradas as características de cada dataset, pela ordem das colunas, são: número de exemplos de treino, número de exemplos de teste, número de classes em que os dados são classificados, número de entradas com valor categórico (um valor dentre um conjunto fixo de valores), número de entradas com valor em float, número de entradas com valor em inteiro, número de entradas com valor em binário e o número total de valores para cada entrada.

Nome	amostras treino	amostras teste	classes	categóricos	floats	inteiros	binários	entrada
IRIS	135	15	3	0	4	0	0	4
MNIST	60.000	10.000	10	0	0	784	0	784
Adult	30.162	15.060	2	8	0	6	0	14
Bank	37.069	4.119	2	11	5	4	0	20
Credit	619	69	2	9	6	0	0	15
Survial	275	31	2	0	0	3	0	3
Zoo	91	10	7	0	0	1	15	16

Tabela 4.1: Características dos datasets utilizados nos experimentos. As colunas indicam o número de amostras utilizadas para treino, número de amostras utilizadas para teste (cálculo de acurácia), número de classes possíveis, #número de dados categóricos, número de dados com valor float, número de dados com valor inteiro, número de dados com valor booleano e número total de dados por entrada.

A rede WiSARD base que é citada como ponto inicial de cada experimento é uma instância de rede inicializada com o número certo de Discriminadores, RAMs e com uma Retina sorteada. Com esse procedimento todas as variações subsequentes do modelo que serão testadas utilizarão a mesma Retina. Sobre os dados; estes estão separados em conjunto de treino e teste de forma diferentes. Para os datasets MNIST e Adult que já apresentam conjuntos de treino e de teste separados, estes foram utilizado como apresentados, sendo a única modificação a remoção das entradas com algum valor marcado como "desconhecido" da base Adult. Para todos os outros casos os datasets foram separados em 10% para teste e 90% para treino, sendo os conjuntos de teste e treino re-sorteados em cada rodada diferente do experimento. Esse processo foi escolhido pois alguns casos os dados sorteados como os 10% de teste poderiam ser especialmente bons e os resultados da simulação não serem representativos. Isso também gera um resultado mais próximo do que se espera observar ao fazer uma validação de um modelo para o dataset (já que para os conjuntos MNIST e Adult qualquer outro pesquisador basearia seus resultados no conjunto de teste pré definido e nos outros datasets seria feita *cross-validation*).

Em seguida entramos no loop principal de fault injection. Determinamos qual o tipo de erro que irá ocorrer, a taxa de erro e o nome dos arquivos em que o resultado será salvo. Copiamos a rede base, no caso de experimentos com os datasets Adult e MNIST estas já estão treinadas e sua acurácia sem erro já foi calculada. No caso dos outros datasets os conjuntos de treino e teste são selecionados, o modelo é treinado, testado, e o valor de acurácia base salvo.

O próximo passo é a adição de erros na WiSARD de acordo com os parâmetros de fault injection selecionados anteriormente. Estes erros podem ser introduzidos nos valores das RAMs da rede, ou a rede pode ser retreinada em quanto se injeta erros

durante o treinamento ou os dois. Após esse passo uma rede modificada é produzida e apurada utilizando o conjunto de teste. Novamente erro pode ser introduzido durante esta etapa. Por fim, a acurácia da rede modificada e os valores de fault injection do teste são salvos em arquivo. Esse experimento é repetido 200 vezes para cada par TE mais dataset.

4.2 Binarização

Já foi dito que as WiSARDs testadas utilizarão um subconjunto dos mesmos hiperparâmetros. Os hiperparâmetros já definidos foram o número N de bits que endereçam as RAMs (tamanho das n-tuplas) que será 12 e o número bits utilizado para a binarização de termômetro (que explicaremos aqui) que será 20 (exceto para o MNIST que usaremos 10 por motivos de custo computacional). O parâmetro C (número de Discriminadores/classes) depende do dataset a ser testado e o parâmetro R (número de RAMs) é uma função do tamanho da entrada com N. Para obter C basta atentar diretamente a tabela 4.1 na seção anterior, mas para R é necessário decidir primeiro como os valores de entrada serão pré-processados antes de serem entregues à rede.

Para realizar os testes de forma menos tendenciosa os métodos de binarização foram escolhidos de forma padrão para cada tipo de dado, sem considerar se poderiam haver modificações que melhorem o desempenho do modelo. Essas modificações incluem escolher quantidades de bits diferentes para codificar determinados valores que esperemos que sejam mais relevantes ou de repetir entradas (basicamente duplicar a quantidade de bits com informação de uma entrada) para que mais RAMs possam analisar sua informação. Consideramos que usar os mesmos parâmetros de rede e binarização é menos tendencioso pois indica que não estamos otimizando a WiSARD para cada problema específico, ou escolhendo uma arquitetura da rede mais ou menos estável para cada tipo de dataset.

Dentre as técnicas disponíveis de binarização as técnicas de termômetro e de *one-hot* foram selecionadas por serem de fácil entendimento e implementação e não fazerem muitas considerações sobre o tipo de dado que será utilizado. Isso é, não servem apenas para valores numéricos, ordenados ou com certa distribuição.

A codificação de *one-hot* foi escolhida para binarizar valores categóricos. O número de bits de codificação escolhido foi igual ao número valores possíveis. Nesta codificação todos os bits são colocados com o valor zero exceto um, sendo cada bit representativo de um dos possíveis valores que a entrada pode ter. Sempre há um único bit 1 nesta codificação (pelo menos se não houver nenhum valor desconhecido, que foi o caso neste trabalho).

A codificação de termômetro foi escolhida para binarizar os valores de float ou de inteiro. Este método necessita de informação adicional sobre a binarização na forma do número de bits utilizado e ainda do maior e menor valores possíveis para a entrada. O número de bits para a binarização é então inicializado com valor 0 e são calculadas X faixas de valores, onde X é o número de bits para binarização mais 1. Esta faixa pode ser dinâmica ou não linear, mas neste trabalho será utilizada apenas uma faixa fixa linear. Esta faixa linear de valores é um intervalo de tamanho Δ, tal que:

$$\Delta = \frac{\text{maior}_{\text{valor}} - \text{maior}_{\text{valor}}}{X - 1} \tag{4.1}$$

Começando com um valor de referência igual ao menor valor possível adicionado de Δ. Verificamos então se o valor a ser codificado é menor ou igual a este. Se sim todos os bits permanecem 0. Se não o primeiro bit é marcado com 1 e o valor de referência é incrementado por Δ. Esse processo então é repetido para o bit seguinte da codificação até o valor ser testado $X - 1$ vezes ou o valor ser menor que o valor de referência. Em outras palavras, o termômetro é um bitstring de tamanho fixo com uma sequência de 1s completada com uma sequência de 0s. Uma de suas vantagens é garantir uma distância de Hamming 1 entre os possíveis valores consecutivos codificados.

Adicionalmente é importante, comentar mesmo que soe óbvio, que os valores de entrada binários, que existem apenas no dataset Zoo, são mantidos como estão. Também neste dataset existia um parâmetro "nome do animal", representado por uma string de tamanho variado. Esse parâmetro foi desconsiderado em nosso treino deste dataset por ser mais difícil de codificar e não aparentar ter o potencial de adicionar muito mais informação que ajude com o processo de classificação.

4.3 Resultados dos experimentos de injeção de erros nos parâmetros

Nesta seção são apresentados os resultados dos teste que introduzem erro somente nos parâmetros da rede, que compreendemos como de ruído nas entradas (valores salvos) das RAMs (neurônios da rede). Todos os gráficos apresentam a taxa de injeção erro (TE) no eixo X (sempre com os valores 0.5%, 1%, 5%, 10%, 15% e 20%) e a queda medida de acurácia (em %) no eixo Y sempre para um dado dataset. Os resultados estão separados em três partes. A primeira sendo referentes ao que chamamos de 'Erro de Tipo 1' (definidos na seção 4.1), a segunda referentes ao que chamamos de 'Erro de Tipo 2' e por fim resultados referentes a um experimento extra com uma versão modificada do 'Erro de Tipo 1'.

Os gráficos de 'Erro do Tipo 1' apresentam duas curvas, enquanto que os do 'Tipo 2' apresentam 32 curvas no mesmo plot. Nos dois casos as medidas são a média de 200 repetições do tipo de erro para uma dada taxa de amostragem.

Com os experimentos do 'Erro de Tipo 1' esperamos observar uma queda progressiva de acurácia e queremos observar não só seu valor absoluto, mas também sua derivada. Já em relação ao erro de 'Tipo 2' esperamos observar se existe alguma diferença no nível de piora de acurácia para erros focados em diferentes bits ou se o erro simplesmente aumentará quando injetado em posições mais significativas. Com isso poderemos avaliar se um conjunto de bits é mais ou menos sensível para a resposta do

modelo do que outros.

4.3.1 Injeção de ruído em bits randômicos

O primeiro teste foi o de injeção de 'Erro do Tipo 1' nas RAMs, ou seja, ruídos introduzidos em parâmetros aleatórios e em bits aleatórios destes parâmetros. Depois de rodarmos os experimentos descritos anteriormente para os seis valores definidos de taxa de injeção de erro, agregamos os resultados na tabela 4.3.1 e os plotamos para observar melhor sua variação.

	0.50%	**1.00%**	**5.00%**	**10.00%**	**15.00%**	**20.00%**
IRIS	0	-0.0006667	0.004	0.006	0.0093333	0.0073333
MNIST	0.01997	0.03811	0.120693	0.162467	0.197903	0.225737
Adult	0.0013274	0.0022583	0.0152932	0.028336	0.057085	0.0517304
Bank	7.28E-05	5.97E-03	1.04E-02	2.39E-02	5.62E-02	1.08E-01
Credit	0.0015175	0.0076453	0.0015175	0.0199081	0.0288316	0.0106545
Survival	0.003226	0.0	0.012903	-0.009677	0.032258	0.019355
Zoo	0.0	0.0	0.011111	0.011364	0.022727	0.011628

Tabela 4.2: Queda de acurácia no teste de Erro de 'Tipo 1'.

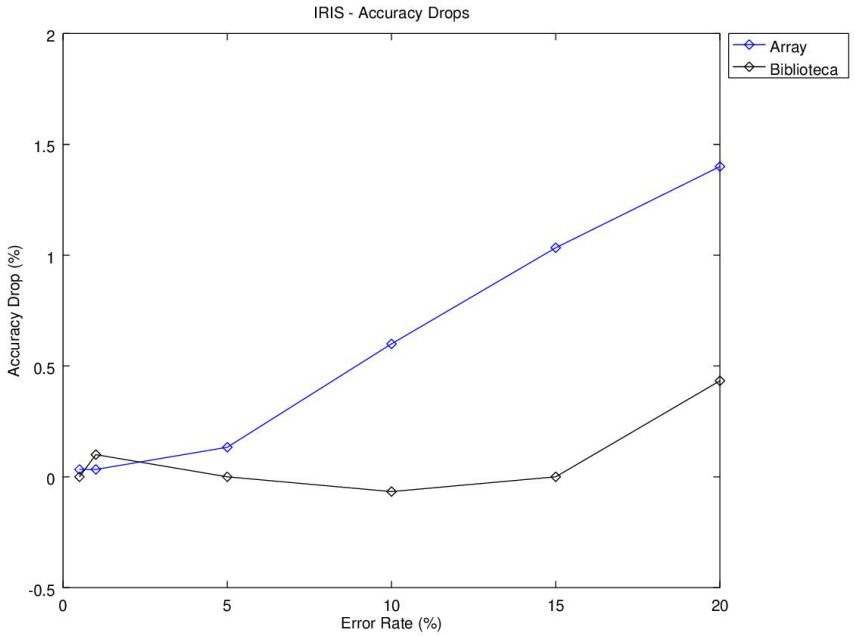


Figura 4.1: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Iris.

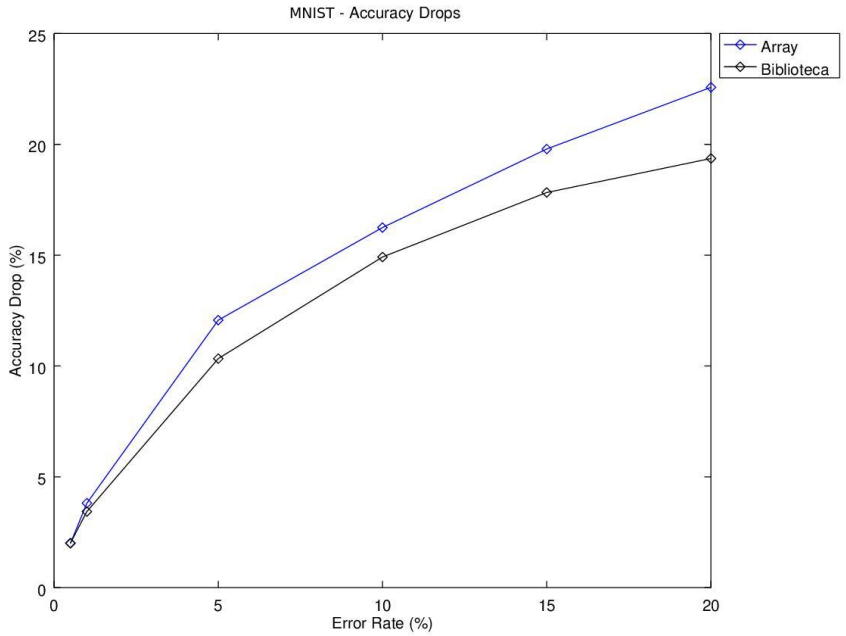


Figura 4.2: Queda de acurácia para seis valores de taxa de injeção de erro no dataset MNIST

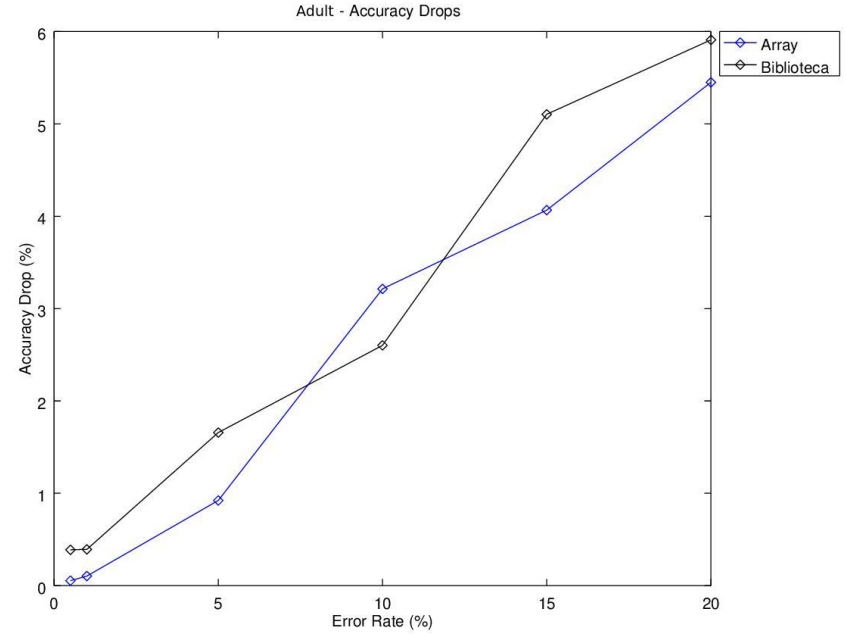


Figura 4.3: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Adult.

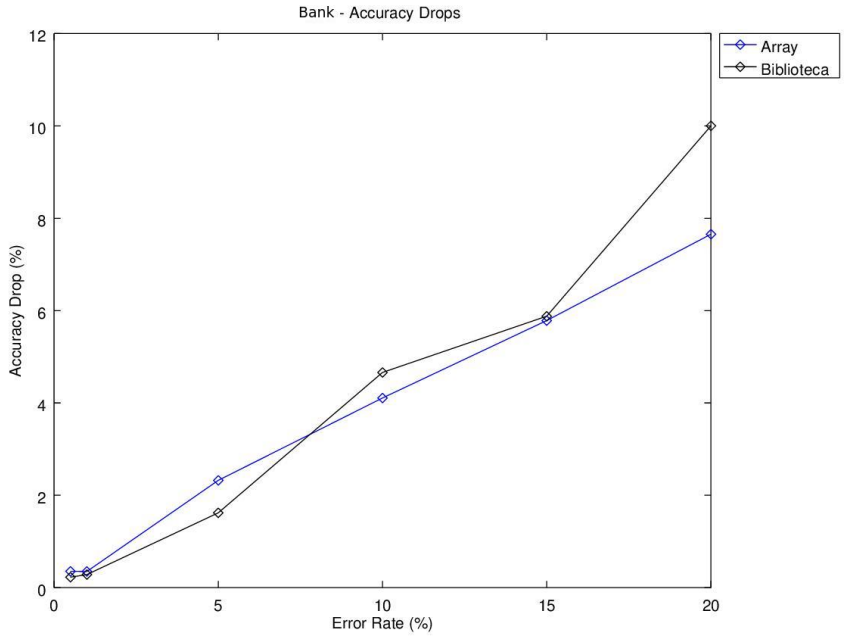


Figura 4.4: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Bank.

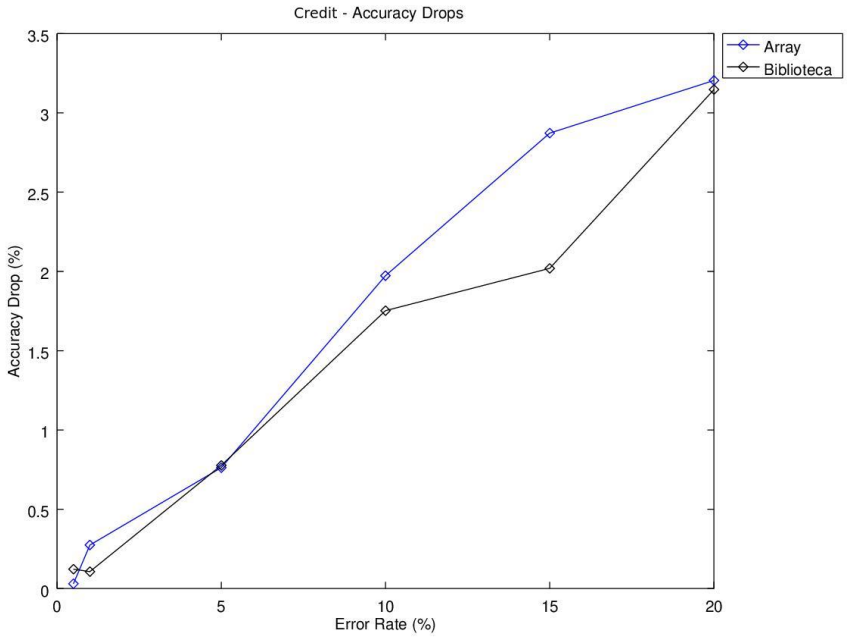


Figura 4.5: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Credit.

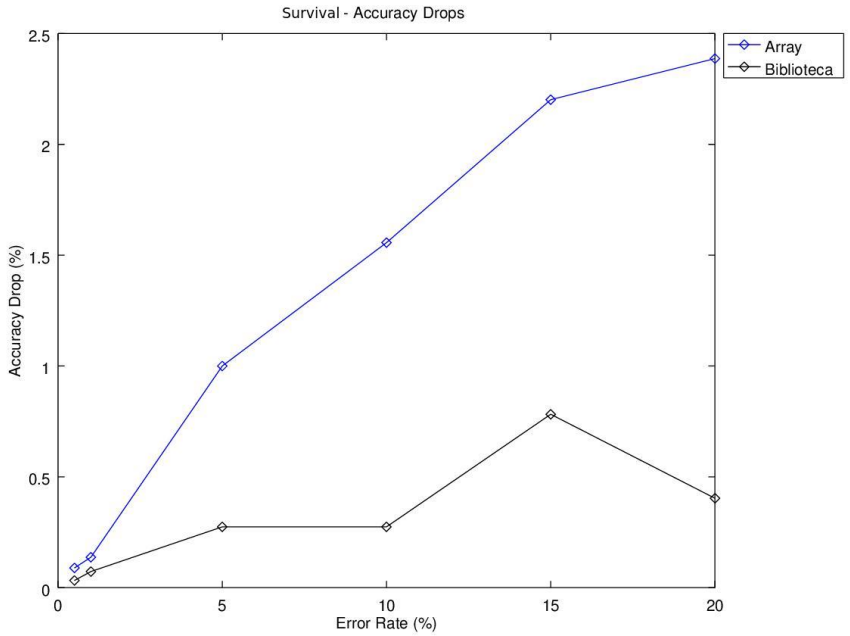


Figura 4.6: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Survival.

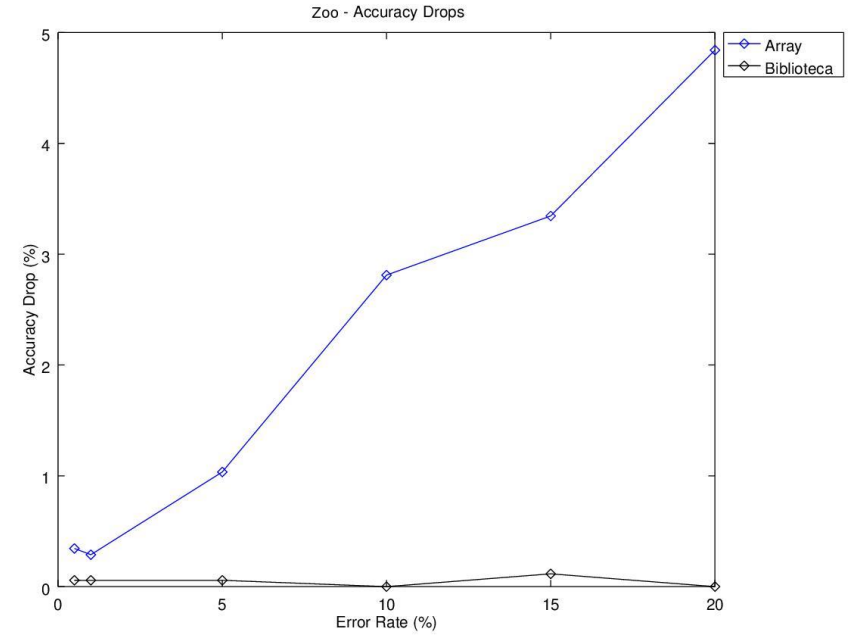


Figura 4.7: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Zoo.

A curva de cor azul corresponde a implementação da WiSARD utilizando-se arrays como RAMs. Já a curva preta corresponde a implementação da WiSARD em que as RAMs são implementadas utilizando-se dicionários. Nos dois casos ruído do 'Tipo 1'

foi adicionado às entradas das RAMs, a diferença entre os dois experimentos executados sendo que no primeiro caso (Array) qualquer entrada da RAM poderia ter seu valor modificado, enquanto que no segundo (Dicionário) apenas valores já treinados poderiam ser modificados, o que corresponde a acreditar que apenas os valores (e não as chaves) do dicionário podem ser afetados por bit-flip. Sobre seu desempenho duas situações foram observadas: nas figuras 4.1, 4.6 e 4.7, a implementação com dicionário teve desempenho superior, apresentando baixíssima queda de acurácia, em comparação a outra implementação, para todas as taxas de injeção de erro. Em particular na figura 4.1 e 4.7, a queda de acurácia se mantém quase em zero ao até negativa (ganho de acurácia). Já na segunda situação as duas implementações apresentam comportamento similar de evolução e de valor das quedas de acurácia. Nessas figuras a curva preta também tende a seguir a curva azul, mesmo que não perfeitamente, esta acompanha a derivada da outra. Além disso observamos que o comportamento dos resultados foi como o esperado, em que a taxa de erro médio cresce conforme aumentamos a intensidade do ruído injetado. No entanto, é notável a diferença do valor absoluto do ruído com o dataset escolhido, fazendo com que o modelo seja razoavelmente resiliente em alguns datasets, mas não em outros. Para os casos mais resilientes a perda de acurácia se mostrou inferior a 1%, já em outros, a perda de acurácia chegou a superar 20% (em alta taxa de injeção de ruído). No geral todos os gráficos apresentaram comportamento quasilinear, isso é, aproximam-se de retas mesmo que não de forma perfeita. O caso que menos se aproximou de um comportamento linear foi o da figura 4.2 no qual a derivada foi diminuindo conforme a taxa de injeção de erro crescia.

4.3.2 Análise da Moda da queda de acurácia

Ao nos aprofundarmos nos resultados brutos do experimento anterior percebemos dois fatos: o desvio padrão das medidas tiradas é muito grande e comparado com as próprias medidas (deixando estas pouco significativas) e que os valores realmente medidos de queda de acurácia são deveras distantes da média calculada. Na prática o que se observa é que a maior parte dos experimentos apresenta queda de acurácia igual a zero por cento e as quedas medidas não são próximos aos dos valores médios. Por tanto para explorar melhor o perfil de erro obtido decidiu-se observar a Moda das quedas de acurácia medidas.

Entretanto como dito anteriormente a maior parte experimentos apresentou queda de acurácia igual a zero, fazendo a moda ser zero. Ao se ignorar os resultados com queda de acurácia menor ou igual a zero (isto é, considerando apenas os casos em que houve erro perceptível) ainda assim a moda continua distante da média calculada. Isso quer dizer que quando quedas de acurácia ocorrem estas apresentam valores bem distintos, alguns pequenos e poucos bem grandes que puxam a média da queda

de acurácia para cima. Para melhor entender a evolução do perfil de erro (variação da tendência de queda de acurácia) de cada modelo para as diferentes taxas de injeção de erro. Para tanto foram plotados os histogramas das quedas de acurácia dos modelos treinados para as diferentes taxa de erro. Os histogramas foram divididos em 21 bins e em seguida escalados para terem os mesmos limites superiores e inferiores em cada plot. Essa mudança de limite faz com que alguns histogramas fiquem menores e com as barras muito finas, mas ajudam a dar a noção da evolução de escala dos erros. Na figura 4.10 Pode-se observar o histograma do dataset Adult para o injeção de erro do tipo 1.

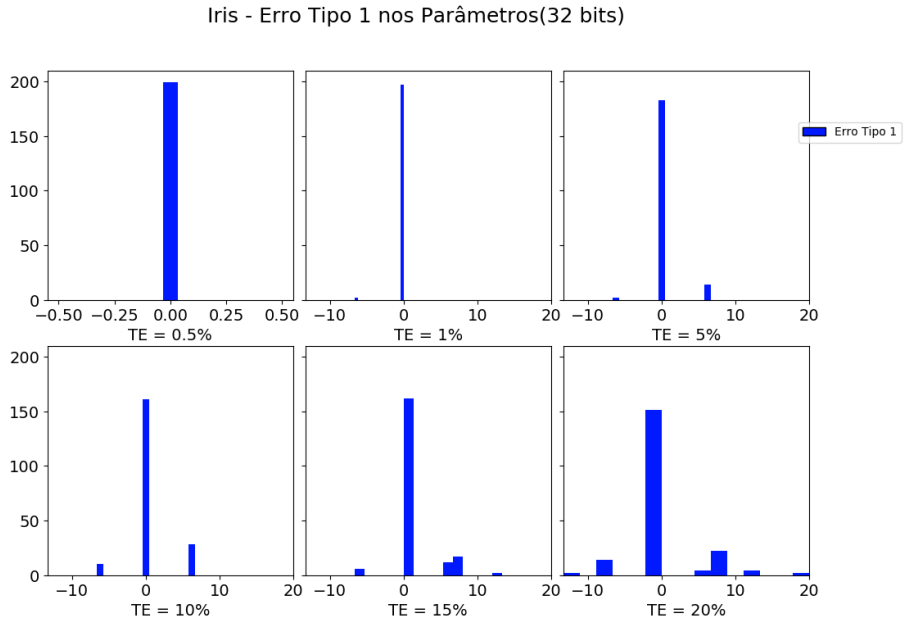


Figura 4.8: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Iris.

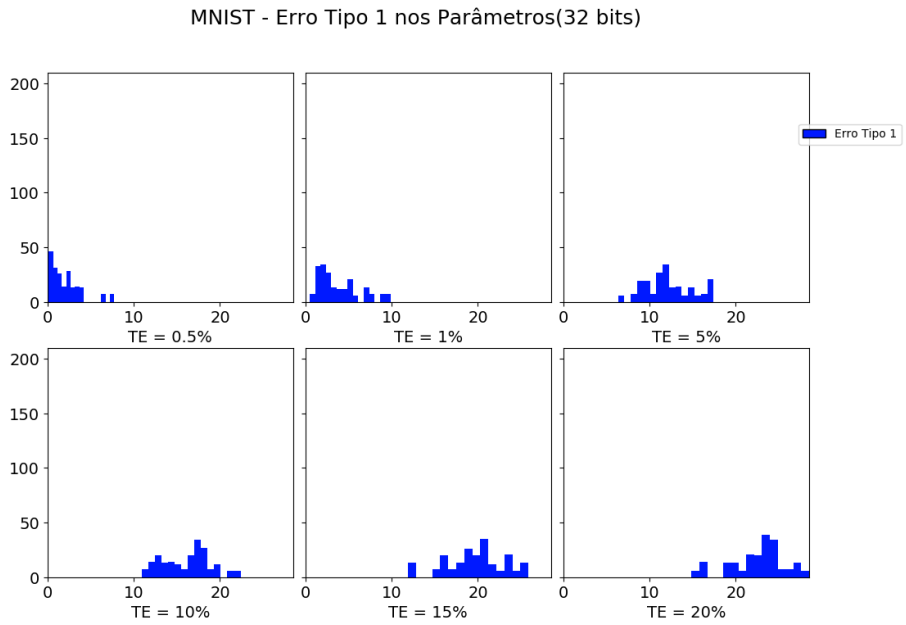


Figura 4.9: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset MNIST.

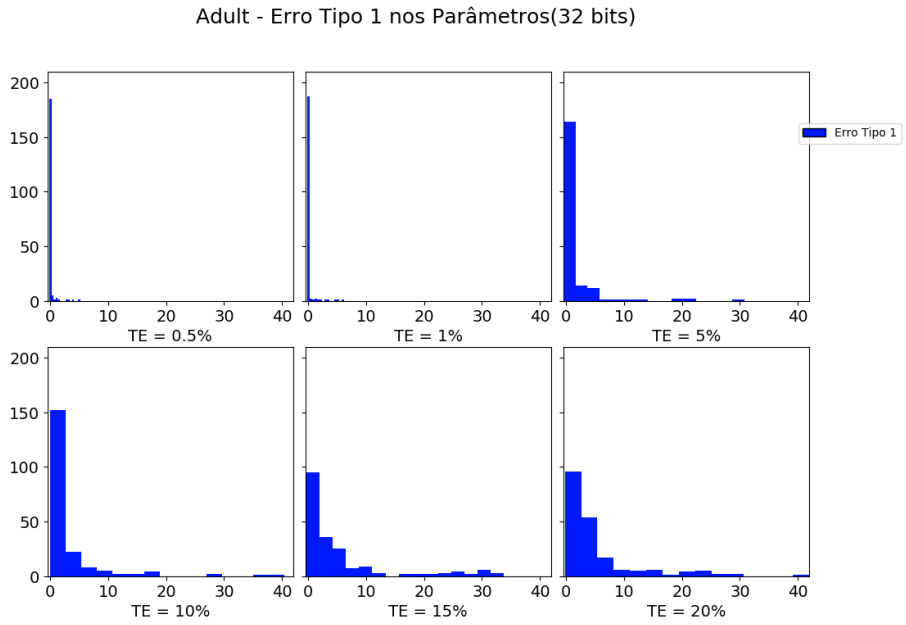


Figura 4.10: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Adult.

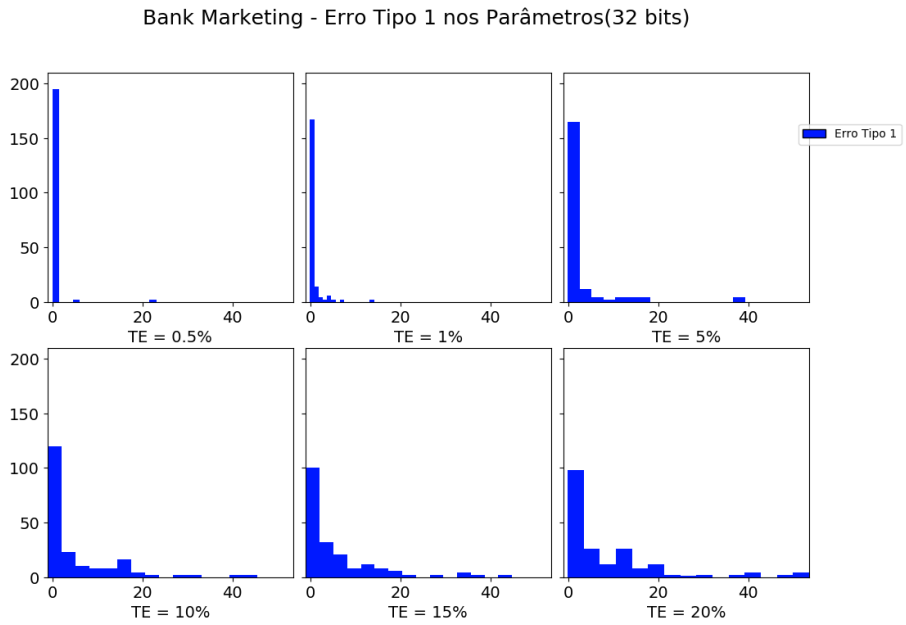


Figura 4.11: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Bank.

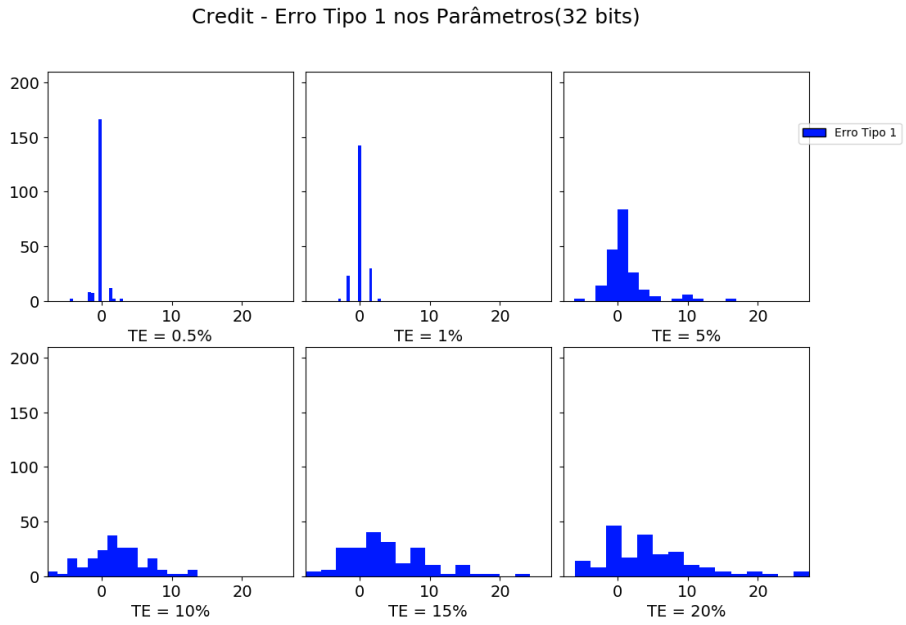


Figura 4.12: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Credit.

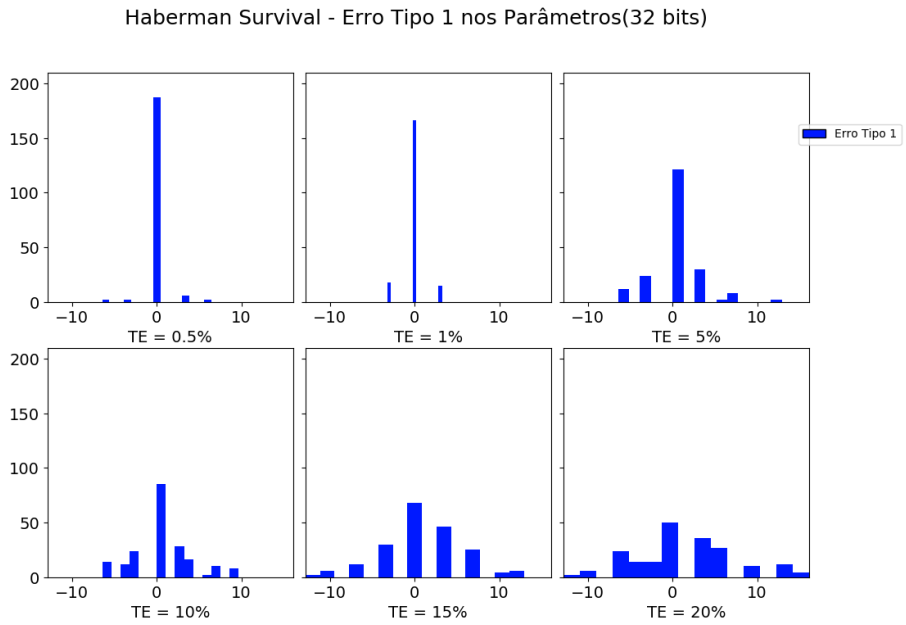


Figura 4.13: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Survival.

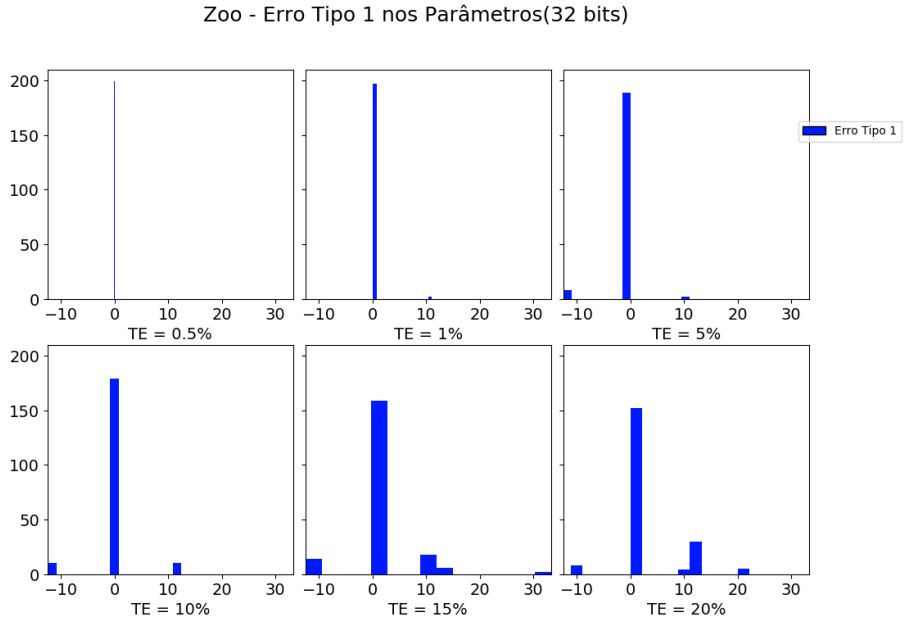


Figura 4.14: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Zoo.

Nas figuras 4.11 e 4.14 conseguimos observar com facilidade que o número de casos em que não há queda de acurácia vai caindo com o aumento da taxa de injeção de erro, o que indica que o modelo apresenta alguma resiliência a erro (já que o modelo consegue manter seu comportamento normal em alguns casos em que ocorrem erros), mas não é insensível a este. Adicionalmente o maior valor de queda de acurácia cresce juntamente a taxa de injeção de erro e os valores de queda de acurácia vão se distanciando de zero e se mantendo melhor distribuídos. Esse comportamento fica mais marcado nas figuras 4.10 e 4.11, nas quais é possível acompanhar a mudança da moda (que vai se movendo para a direita) e o aparecimento de casos com maiores quedas (mesmo que em pequenas quantidades).

Em resumo: para taxas de injeção de erro, é mais provável que nenhuma perda de acurácia seja percebida. Enquanto que em taxas maiores de injeção de erro as chances de uma grande perda de acurácia crescem.

4.3.3 Injeção de ruído em bits fixos

O segundo teste realizado foi o de 'Erro de Tipo 2', no qual injetamos erros sempre na mesma posição dos bits, nos parâmetros alterados. Neste caso o bit que recebe ruído (bit-flip) é sempre o mesmo em uma posição entre 0 e 31. Neste trabalho usamos a convenção de que 0 representa o bit o menos significativo (2^0), e 31 o mais significativo (2^{31}). Desconsideramos bits de sinal como se trabalhando com um valor unsigned, já que o modelo WiSARD não interpreta, por padrão, valores negativos. Esse teste foi feito apenas com a implementação de WiSARD usando arrays. Os resultados estão plotados nas figuras 4.15 a 4.21.

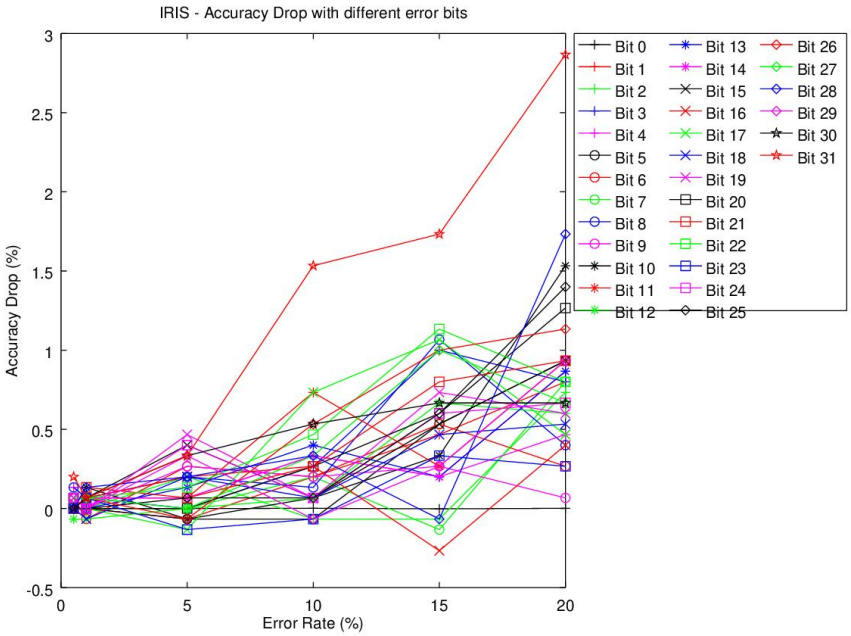


Figura 4.15: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Iris.

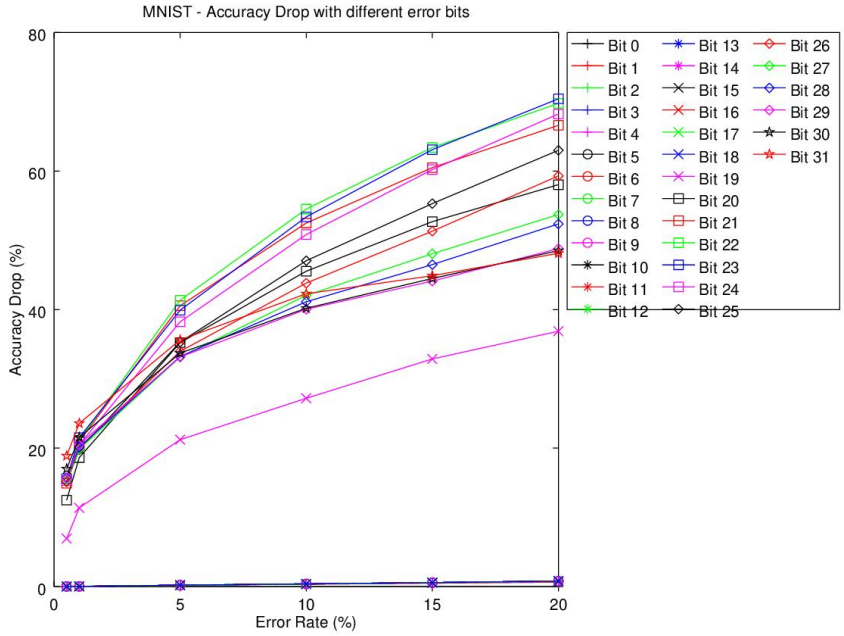


Figura 4.16: Queda de acurácia para seis valores de taxa de injeção de erro no dataset MNIST.

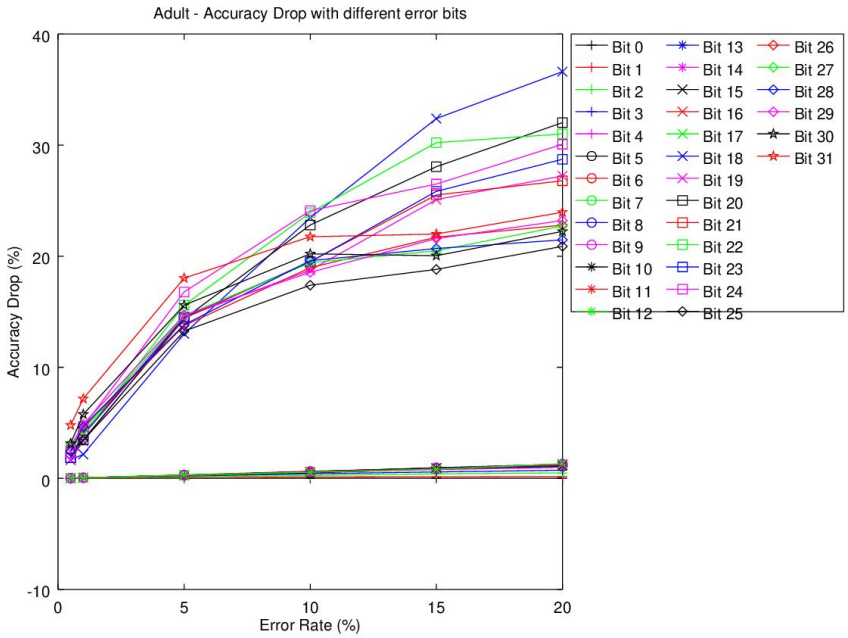


Figura 4.17: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Adult.

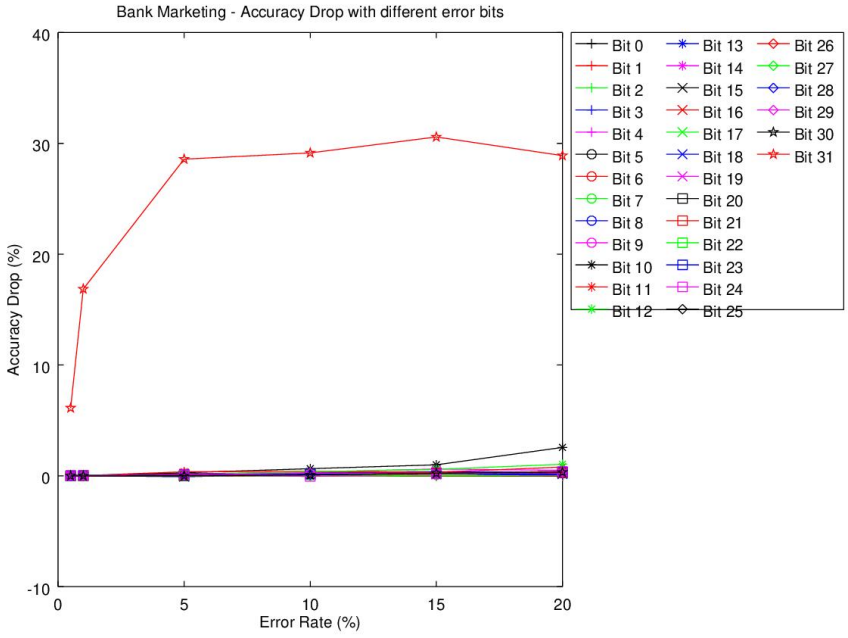


Figura 4.18: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Bank.

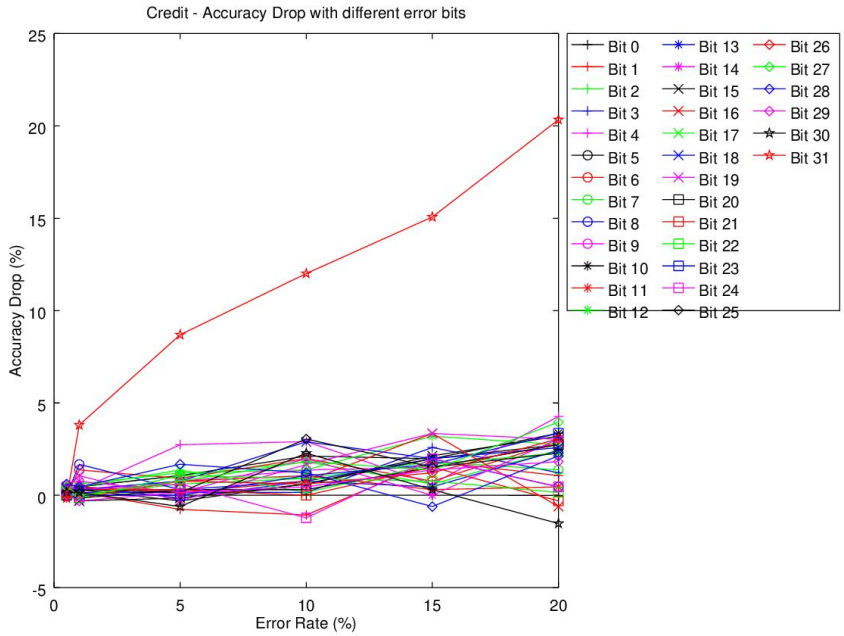


Figura 4.19: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Credit.

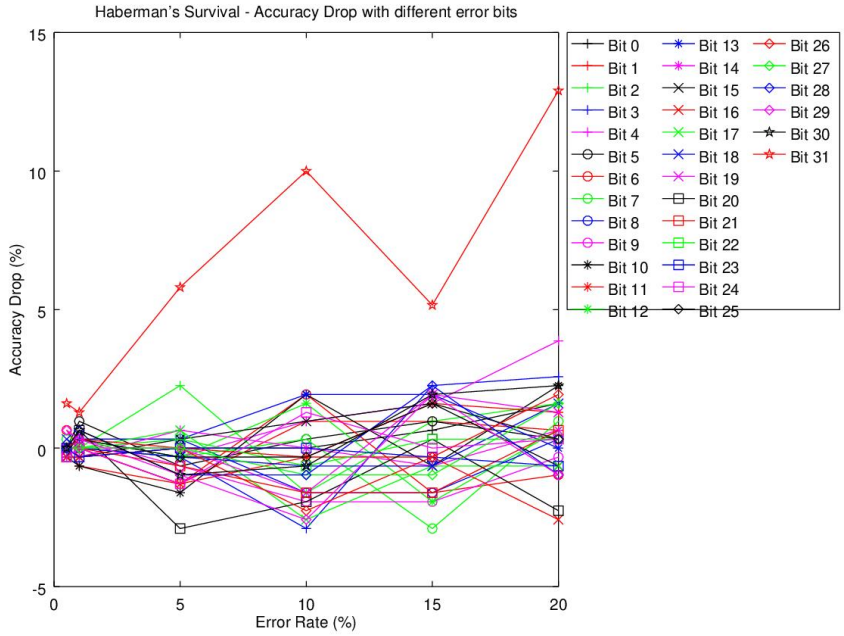


Figura 4.20: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Survival.

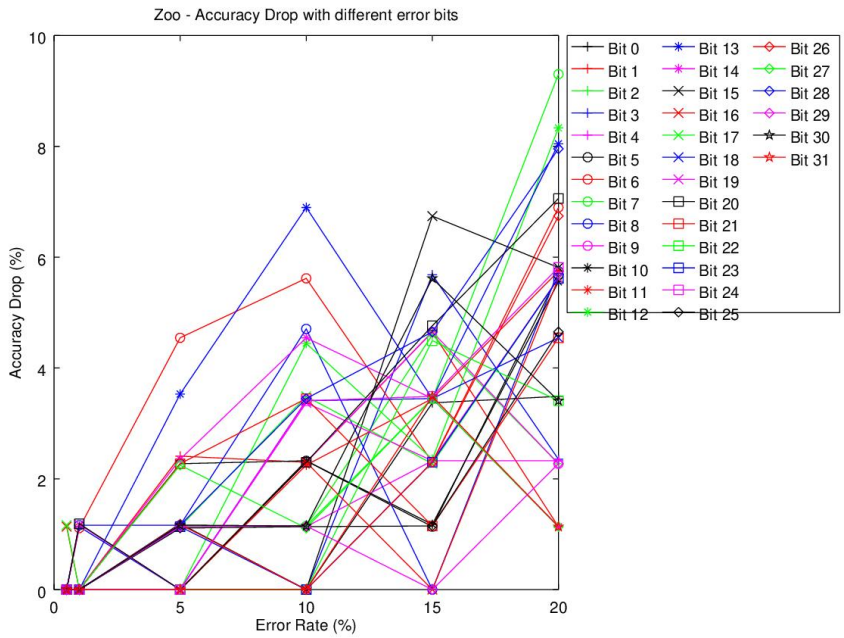


Figura 4.21: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Zoo.

Ao se observar os plots apresentados, a WiSARD parece não discriminar muito em relação a posição em que o erro é inserido (tendo grandes variações), não apresentando uma tendência ou se aproximando de um sequência monotônica. Em todos os datasets as curva de queda de acurácia se sobrepõe e não apresentam uma diferenciação marcante entre os efeitos de dois bits próximos. Ainda assim, um ponto que chama atenção é que quando o ruído é injetado no bit 31 a taxa de erro dispara. Entretanto ao se avaliar melhor percebe-se que a taxa de erro para os outros bits (0 a 30) se mantém em muitos casos inferior a medida para TE de 20%, indicando que esses erros podem apresentar um efeito menor que erros puramente aleatórios sobre o modelo.

f

4.3.4 Injeção de ruído de menor valor em bits randômicos

Considerando os resultados do experimento anterior, propomos um novo experimento para analisar o quanto o bit 31 e o nível do ruído (posição em que é adicionado) afetam a WiSARD. Nesse novo experimento adicionaremos ruído do tipo 'Erro de Tipo 1' mas agora limitando a introdução do erro apenas aos 16 primeiros bits. Isso será feito em uma WiSARD implementada com arrays. Após 200 repetições os novos resultados de queda média de acurácia foram incluídos na tabela 4.3.4. Os gráficos estão nas figuras 4.22 a 4.28 e foram plotados com 3 curvas. A curva azul representa o resultado do teste de 'Erro Tipo 1' feito anteriormente e já plotado nas figuras 4.1 a 4.7. A

curva de cor vermelha representa os resultados novos obtidos limitando-se o erro aos 16 primeiros bits. E a curva de cor preta representa os valores obtidos com a implementação de dicionário e também apresentada nas figuras 4.1 a 4.7.

	0.50%	**1.00%**	**5.00%**	**10.00%**	**15.00%**	**20.00%**
IRIS	0	0.13333	0.4	0.66667	1.06667	1.86667
MNIST	0.0386	0.0831	0.3836	0.7864	1.1187	1.5163
Adult	0.019655	0.033001	0.204648	0.435989	0.739442	0.868327
Bank	0.053289	0.091373	0.61700	1.2721	1.7446	2.0265
Credit	0.021851	0.03399	0.118479	0.252253	0.37729	0.507904
Survival	0.16129	-0.032258	0.516129	1.16129	1.258065	0.580645
Zoo	0	0.11534	0.45767	0.91848	2.1764	3.88571

Tabela 4.3: Queda de acurácia no teste de 'Erro de Tipo 1' com bits-flip limitado aos 16 primeiros bits.

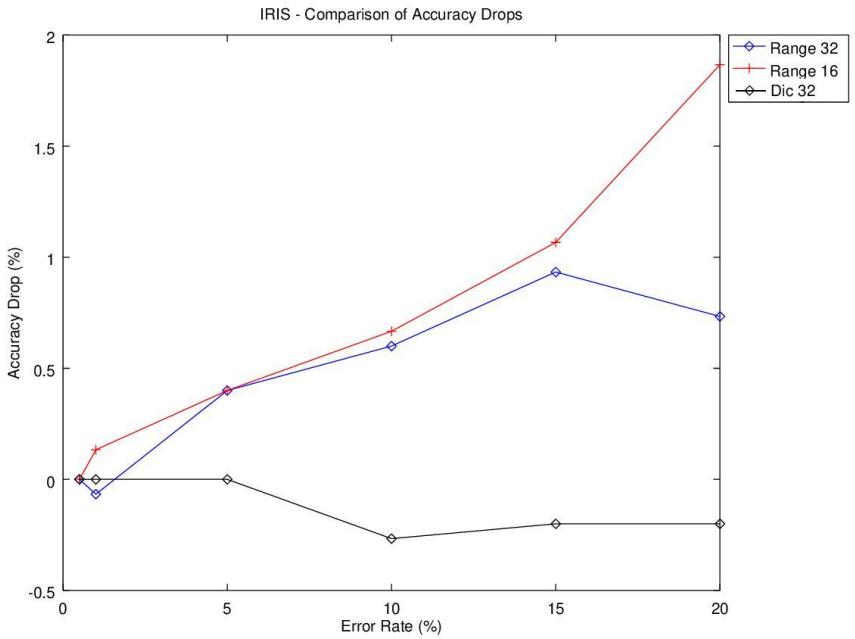


Figura 4.22: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Iris.

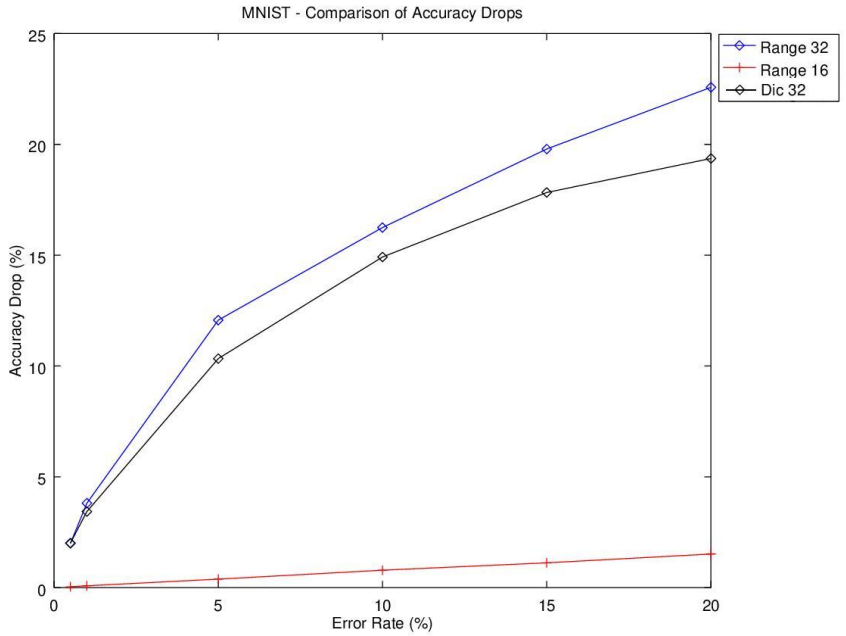


Figura 4.23: Queda de acurácia para seis valores de taxa de injeção de erro no dataset MNIST.

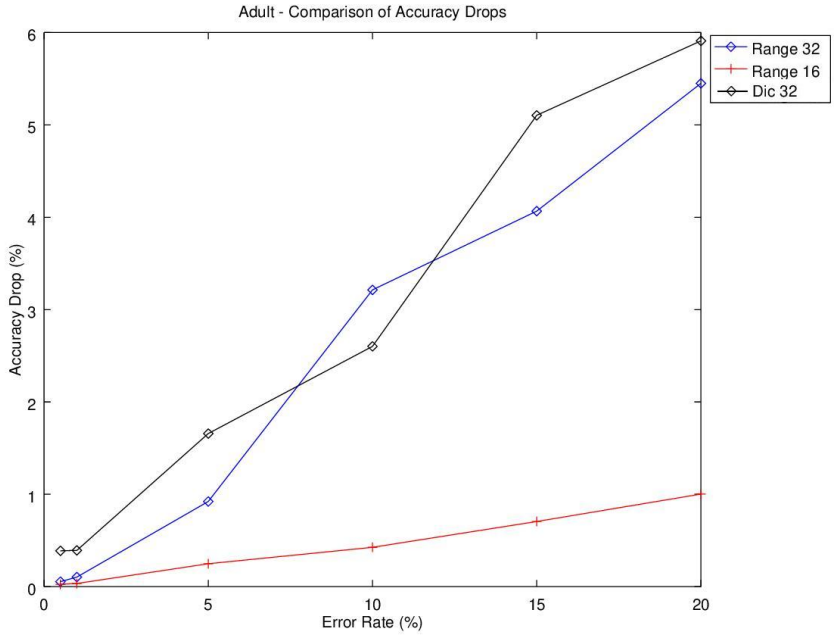


Figura 4.24: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Adult.

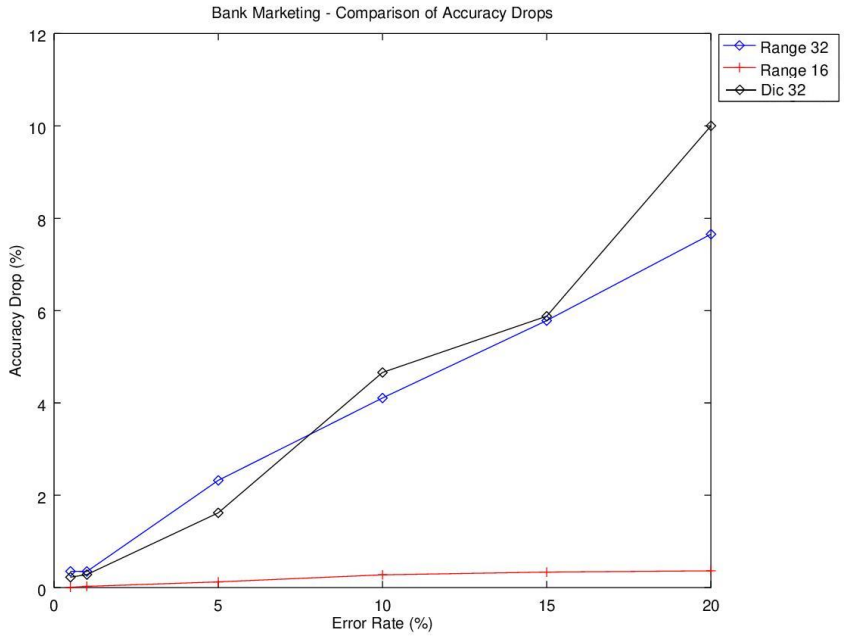


Figura 4.25: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Bank.

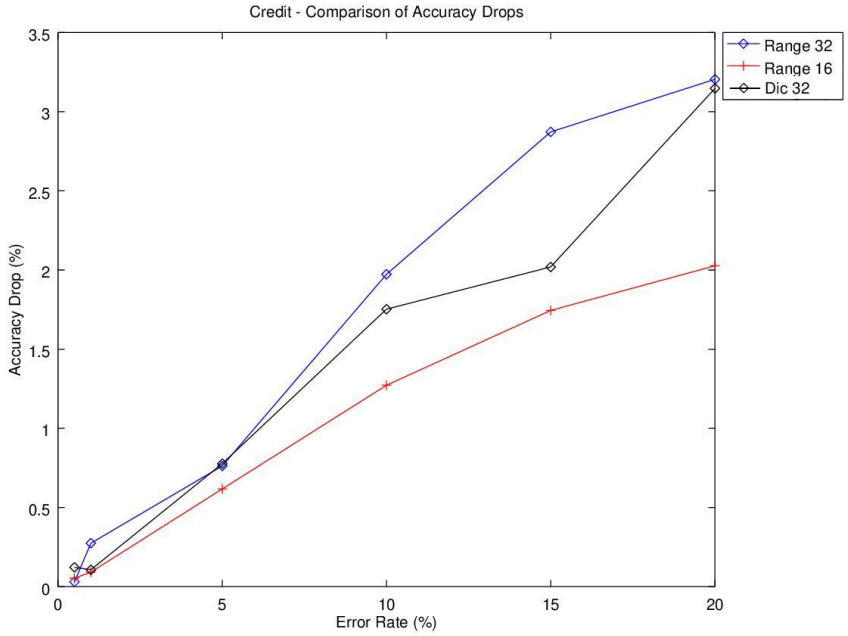


Figura 4.26: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Credit.

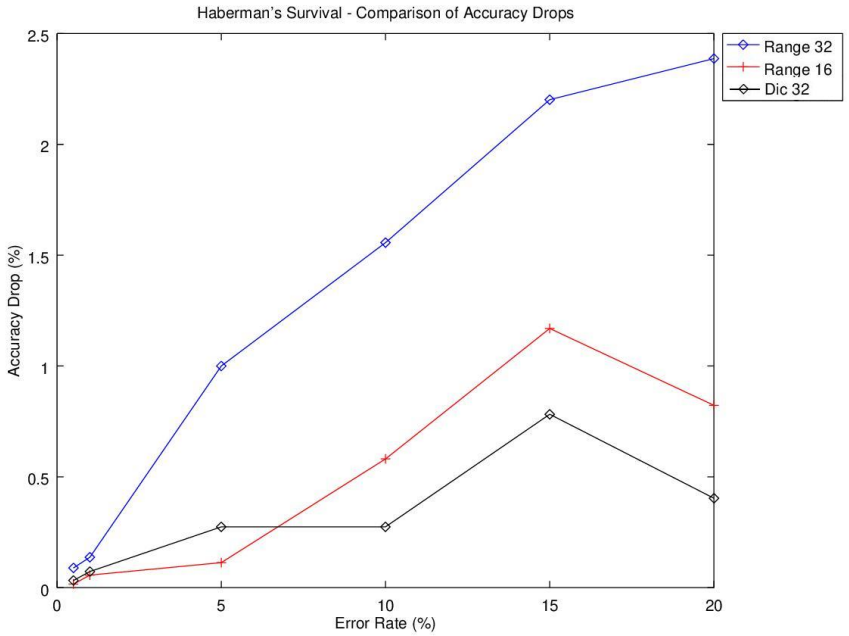
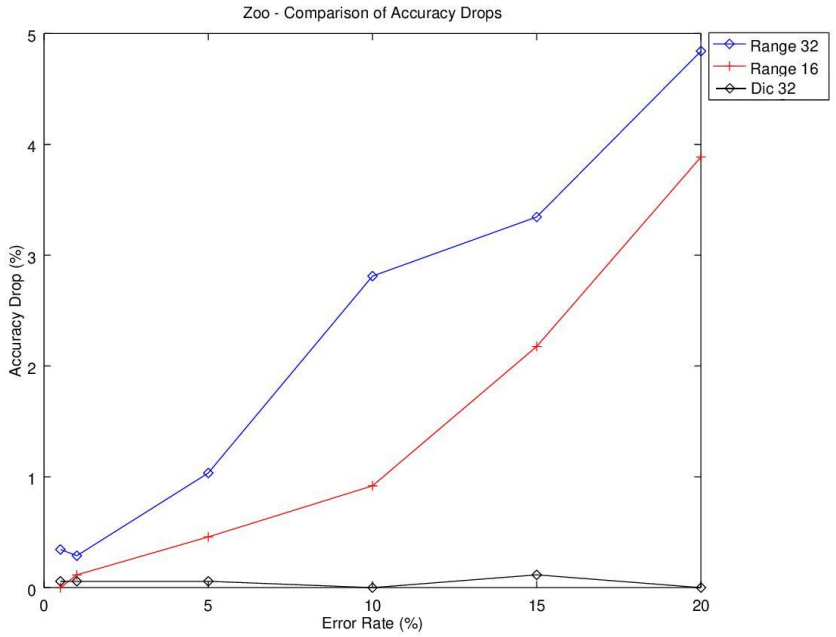


Figura 4.27: .
Queda de acurácia para seis valores de taxa de injeção de erro no dataset Survival.



..

Figura 4.28: Queda de acurácia para seis valores de taxa de injeção de erro no dataset Zoo.

Observando os gráficos, percebe-se que em alguns casos a curva vermelha (perda de acurácia com ruído apenas nos 16 primeiros bits), de forma bem distinta, se mantém em valores muito baixos mesmo em altos valores de TE. Esse é o caso das figuras 4.23, 4.24 e 4.25. Nos demais casos a linha vermelha se mantém abaixo da azul (queda de acurácia com ruído em qualquer bit) exceto para o dataset Iris. A altura da curva vermelha em relação a azul varia, na figura 4.26, 4.27 e 4.28 apesar de estar abaixo da azul a curva vermelha acompanha a evolução da azul. Na figura 4.26 a curva vermelha se mantém bem abaixo com TE de até 5% mas depois sobe ficando um pouco abaixo da metade dos valores da curva azul. Em todos estes casos, o comportamento da linha vermelha em relação a preta não parece apresentar nenhuma correlação, exceto é claro quando a curva preta (queda de acurácia para implementação em dicionário e erro em qualquer bit) acompanha a curva azul. Nestes casos a curva vermelha apresentou performance melhor do que as duas outras.

Isso quer dizer que, exceto no caso do dataset Iris, o ruído afetando apenas os 16 primeiros bits sempre apresenta menor queda de acurácia que o que afeta qualquer um dos 32 bits. Este fato pode ser relativo ao fato de ruído em um bi acima do maior valor salvo nas RAMs ser pouco relevante em relação aos outros valores salvos. Na implementação com 32 bits existem mais bits que se modificados geram esse tipo de comportamento. Esse pode ser o motivo Da diferença de quedas de acurácia. Qualquer que seja o motivo, este fato é interessante pois é utilizar um inteiro de 32 bits, 16 bits ou até 8 bits é uma escolha que pode ser feita no projeto do software, tornando-se assim uma medida que pode ser tomada ativamente para incrementar a resiliência da rede.

4.3.5 Verificação do efeito no número de RAMs sobre a queda de acurácia

Focando novamente nos dois grandes comportamentos distintos observados. É de nosso interesse (mesmo que não seja o objetivo principal do trabalho) determinar se existe um método ou atributo que garante uma maior estabilidade da acurácia (menor queda) mesmo para taxas altas de ruído nos parâmetros da WiSARD. Apenas pelos gráficos não há nada expressivo além do fato de que as curvas da implementação em array e em dicionário acompanham uma a outra. Mas fora dos dados plotados percebemos que os 3 casos em que essa situação é verificada são de datasets (MNIST, Adult e Bank) com entradas maiores que os outros quatro (Iris, Credit, Survival e Zoo). Na tabela 4.4, para cada dataset Discriminamos o número de features (valores distintosde uma amostra), o número de bits usados para binarizar estas entradas, considerando o processo discrito na seção 4.2, e o número de RAMs presente em cada Discriminador considerando n-uplas de 12 bits.

Dataset	# features	# bits entrada	# RAMs
Iris	4	80	7
MNIST	784	7840	654
Adult	14	219	19
Bank	20	255	22
Credit	15	161	14
Survival	3	60	5
Zoo	16	35	3

Tabela 4.4: Características da entrada gerada por cada dataset e da WiSARD treinada nestes.

Por terem um número consideravelmente maior de bits de entrada após a binarização as WiSARDs treinadas têm por extensão um número superior de RAMs em cada Discriminador, ou seja, mais granularidade na votação de cada Discriminador. Considerando que um número muito baixo de RAMs aumenta a chance de empates e se traduziria em ter menos features observadas na entrada é concebível que este seja o fator que buscamos. Para confirmar isso, iremos refazer as curvas para implementação em dicionário com valores de 32 bits, array com valores de 32 bits e array com valores de 16 bits, só que agora replicando o valor da entrada um número de vezes o suficiente para que esta tenha pelo menos 210 bits. A replicação é uma simples cópia da entrada já binarizada que será concatenada a esta, simplesmente duplicando a presença de cada bit de informação. Para chegar ao valor mínimo de 210 bits o dataset Iris deverá ser triplicado de tamanho, o dataset Credit deverá ser duplicado, o dataset Survival deverá ser quadruplicado e o dataset Zoo deverá ser septuplicado. Considerando que a maior parte das entradas do Zoo são binárias usaremos então para ele um processo diferenciado para aumentar o número de bits. Primeiro codificaremos as entradas binárias com 2 bits com o mesmo valor e depois disso replicaremos a entrada inteira. Com isso a entrada base do Zoo passa a ter 50 bits e quintuplicando ela chegamos a 250 bits.

Os resultados foram plotados nas figuras 4.29 a 4.32. Nestas a linha de cor azul representa o resultado original da WiSARD utilizando implementação com array e 32 bits (linhas azuis das figuras 4.1 a 4.7) para cada dataset como uma referência. A linha preta representa o mesmo modelo com a entrada estendida, a linha verde representa a implementação em array com erro apenas até o bit 16 e a linha rosa representa a implementação utilizando dicionário.

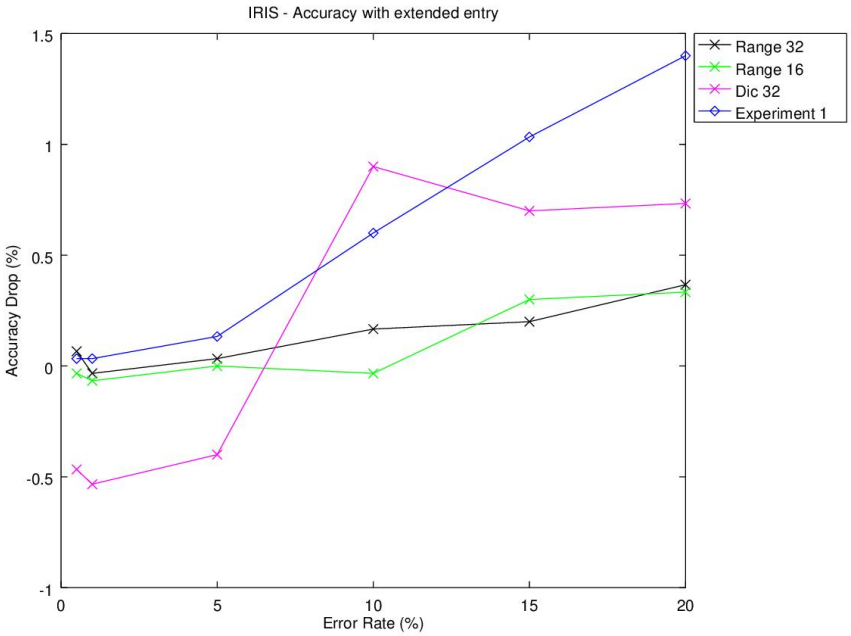


Figura 4.29: Queda de acurácia para o dataset Iris com entrada extendida.

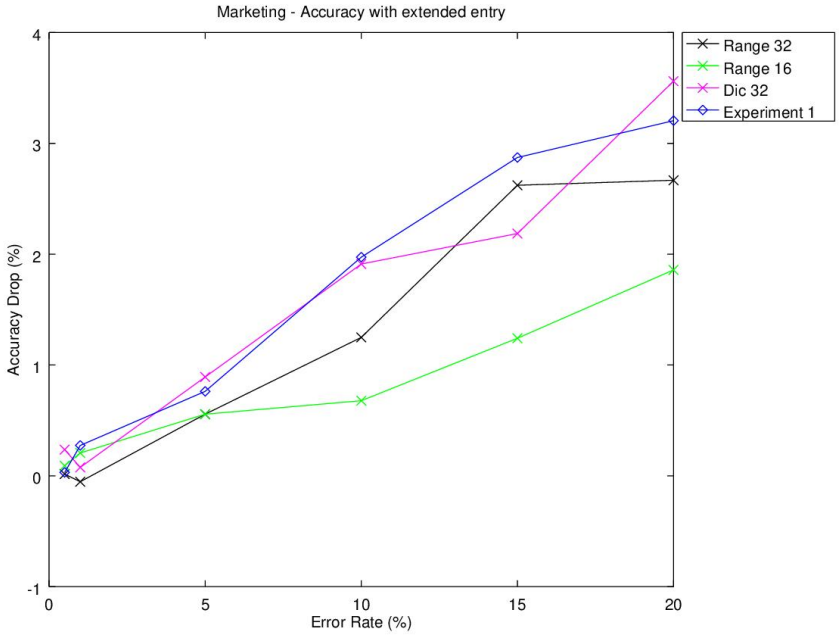


Figura 4.30: Queda de acurácia para o dataset Credit com entrada extendida.

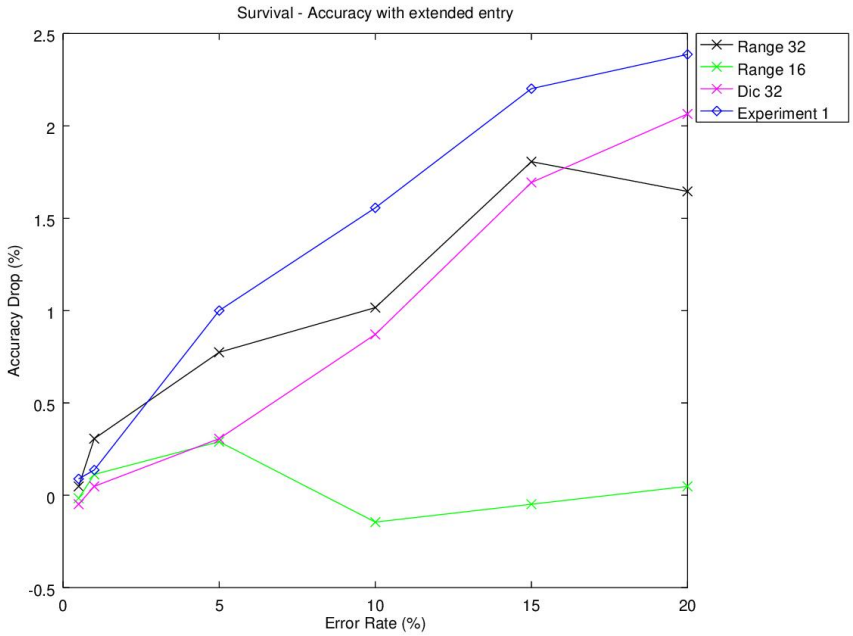


Figura 4.31: Queda de acurácia para o dataset Survival com entrada extendida.

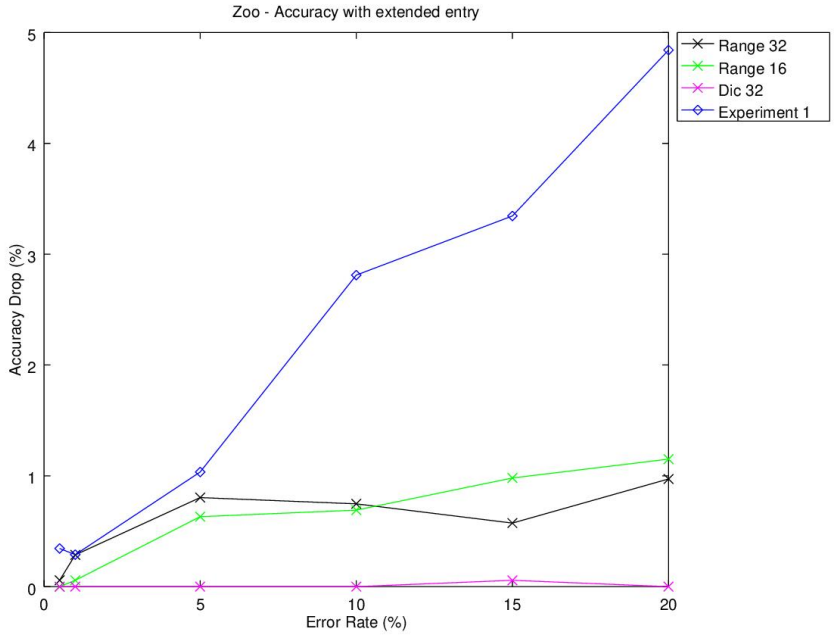


Figura 4.32: Queda de acurácia para o dataset Zoo com entrada extendida.

Percebe-se que as linhas pretas estão abaixo das azuis em todos os plot, sendo a figura 4.30 a em que uma curva mais acompanha a outra. A linha rosa apresenta comportamento muito variado, oscilando (figura 4.29), acompanhando as linhas azul e preta (figura 4.30), acompanhando por baixo a linha preta (figura 4.31) e simplesmente se mantendo baixa quase em 0 (figura 4.32). Por fim a linha verde está sempre acompanhando a linha preta ou bem abaixo dela (colocando-a abaixo da linha azul também). Comparando-a com as figuras 4.15, 4.19, 4.20 e 4.21 observamos a linha verde sempre abaixo da vermelha destas figuras, apresentando uma considerável melhora nos resultados.

Considerando a linha rosa (dicionário com entrada estendida) com comportamento muito variado, a linha verde (array com 16 bits de valor) parece apresentar dentre os datasets, consistentemente, as menores quedas de acurácia, sendo uma implementação mais resiliente ao ruído de parâmetro que as outras testadas.

4.4 Resultados dos experimentos de injeção de erros no treino

Após a introdução de erro nos parâmetros da WiSARD foi testado o efeito da introdução de erro na WiZARD durante seu treino. Existem quatro tipos possíveis de erro durante o treino, o de leitura incorreta de valor da entrada (*bit-flip* de alguma posição do vetor de entrada), endereçamento incorreto ne n-upla (escolha incorreta da entrada da RAM a ser atualizada para uma dada entrada), incremento incorreto da RAM (valor da RAM ou não é incrementado ou gera um valor aleatório em seu incremento) e o de treinar uma entrada no Discriminador errado (ou até em um Discriminador extra que não exista). Dentre esses tipos de erros o primeiro e o segundo são equivalentes e funcionam a partir do mesmo processo de *bit-flip* que foi abordado nos erros de parâmetros. O terceiro tipo de erro se aproxima (apesar de não ser equivalente) a injetar ruídos nas entradas das RAMs, que nem testamos anteriormente. Já o último erro é mais único porém menos bem definido pois dependeria da implementação da rotina de treino e se foram introduzidos códigos extras para verificar se classes inexistentes estão sendo treinadas ou dadas ou retornadas como saída. Por conta disso escolheu-se focar no erro introduzido nos valores treinados nesta seção.

Neste experimento, para os sete conjuntos de dados anteriores, foi treinado um modelo WiSARD e medido sua acurácia. A WiSARD foi então retreinada utilizando-se as mesmas configurações de Retina e a acurácia medida novamente. Durante o retreino, para cada bit de uma entrada a ser treinada, é sorteado pela taxa de injeção de erro (TE) se este será mudado. Esse procedimento foi repetido 200 vezes.

Espera-se que para baixas taxas de injeção de erro a queda de acurácia seja basi-

camente nula em datasets com alta acurácia. Isso pois o modelo WiSARD observa os bits da entrada separadamente e determina qual classe tem maior indício de ser a correta, por isso a mudança de apenas alguns bits não deve mudar o valor salvo em todas as RAMs do Discriminador para aquela entrada. Como a nova posição errada em que uma RAM seria atualizada é aleatória então nenhum erro deve ter muito peso e se todas as classes forem balanceadas estas devem apresentaram quantidades similares de ruído nos parâmetros salvos. Adicionalmente como observado nos experimentos anteriores adições de valores em entradas de RAM que não foram treinadas (neste caso que não seriam treinadas) alteram pouco o resultado final e são comparáveis a casos em que apenas as entradas com algum valor recebessem ruído. Por outro lado, para valores de taxa de injeção de erro altos os resultados podem se tornar randômicos, fazendo a acurácia cair para $1/C$ onde C é o número de classes possíveis.

Os resultados do experimento foram plotados como queda média de acurácia nas figuras 4.33 a 4.39 juntamente com as quedas de acurácia dos experimentos originais com injeção de 'Erro de Tipo 1' para serem utilizados como valores de referência. A linha vermelha é o experimento com erro introduzido no treino e sem erro nas RAMs. Já a linha vermelha é a queda de acurácia com ruído introduzido nas RAMs (de 'Erro Tipo 1') que nem o experimento da seção 4.3.1 com a implementa,ão em Array.

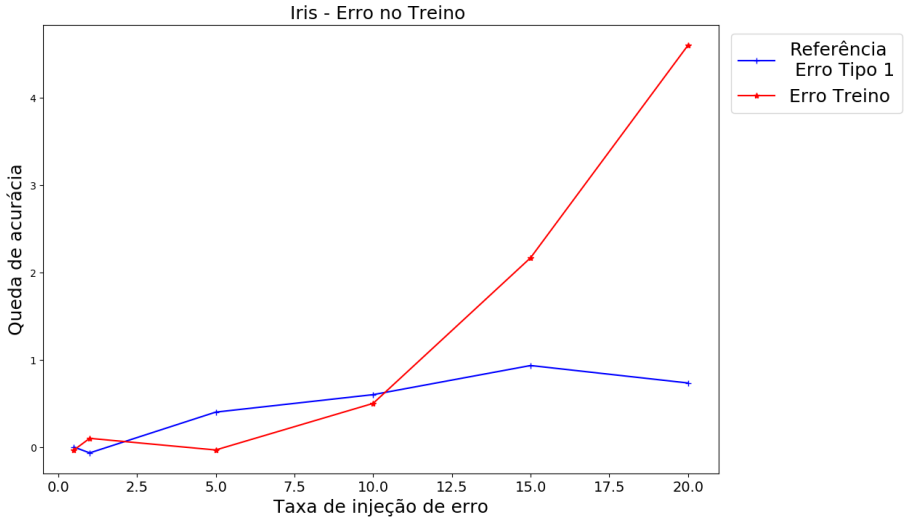


Figura 4.33: Queda de acurácia para erro injetado durante o treino no dataset Iris.

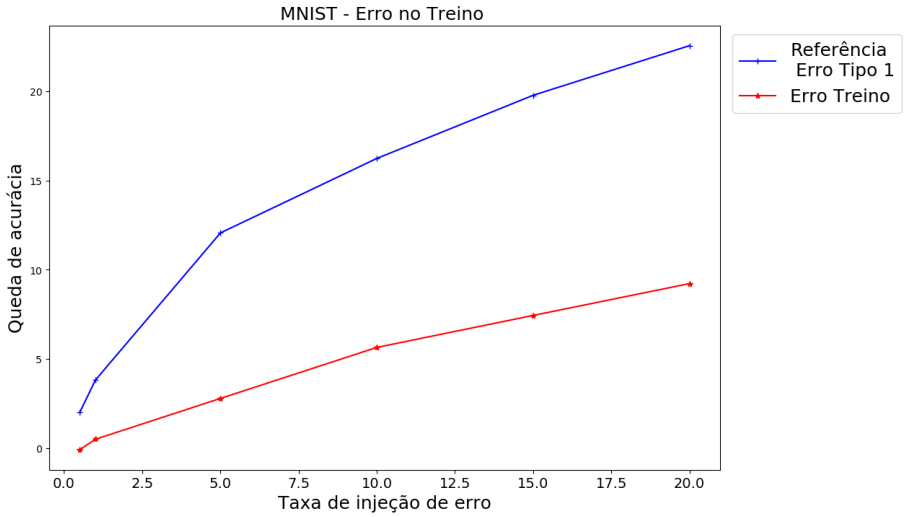


Figura 4.34: Queda de acurácia para erro injetado durante o treino no dataset MNIST.

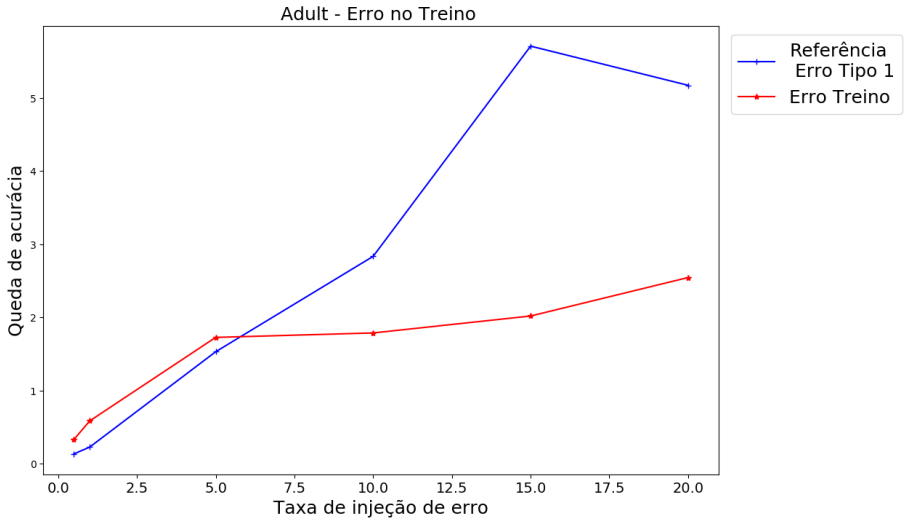


Figura 4.35: Queda de acurácia para erro injetado durante o treino no dataset Adult.

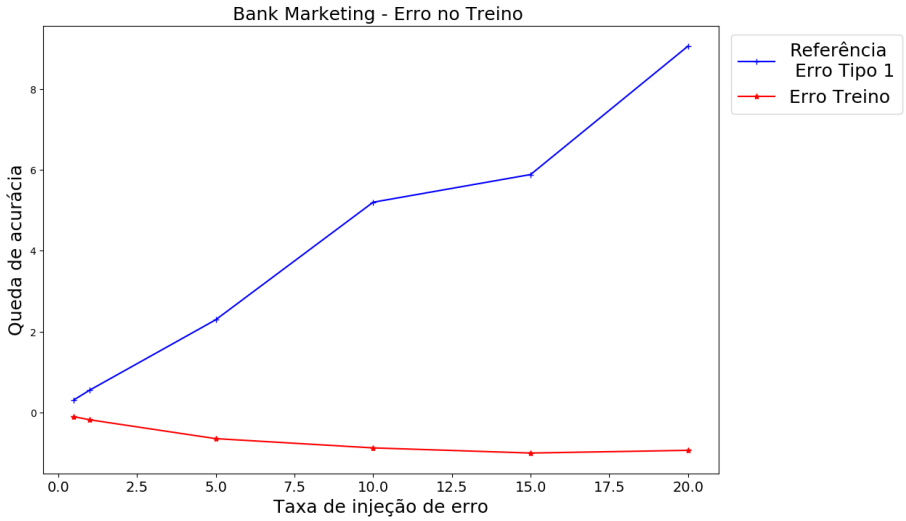


Figura 4.36: Queda de acurácia para erro injetado durante o treino no dataset Bank.

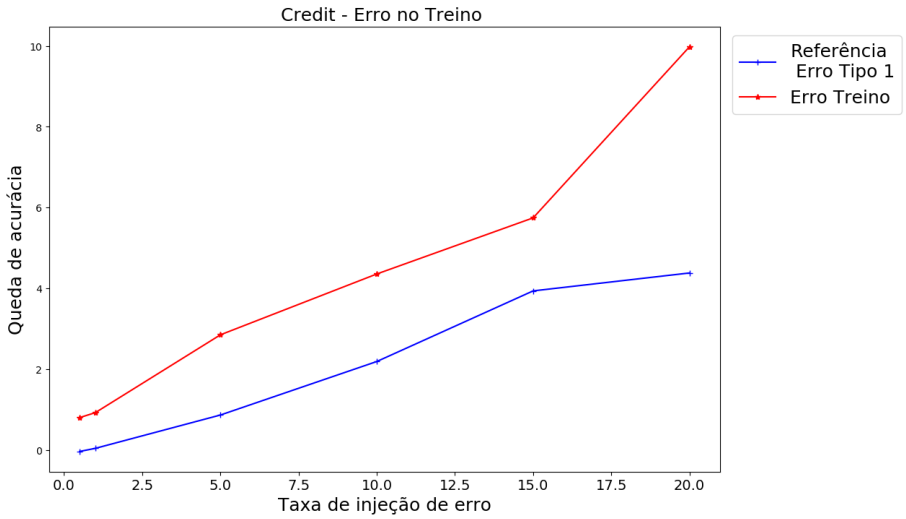


Figura 4.37: Queda de acurácia para erro injetado durante o treino no dataset Credit.

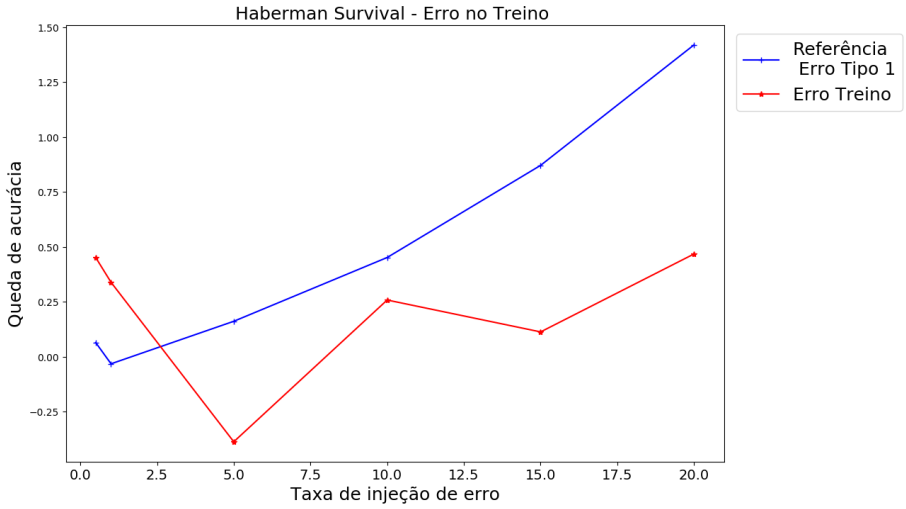


Figura 4.38: Queda de acurácia para erro injetado durante o treino no dataset Survival.

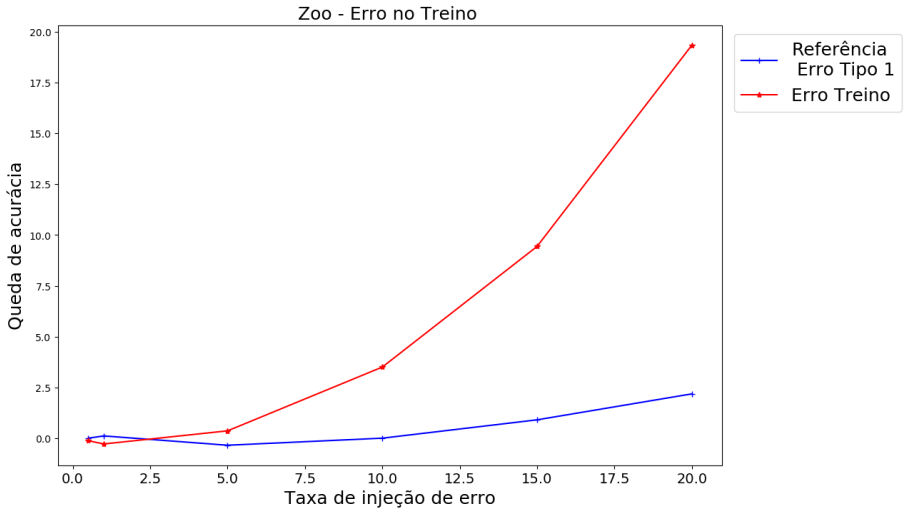


Figura 4.39: Queda de acurácia para erro injetado durante o treino no dataset Zoo.

Pelos gráficos obtidos, percebe-se que a injeção de erro nos valores treinados gera resultados muito variados. Nas figuras 4.33, 4.36 e 4.39 existe um aumento significativo da queda de acurácia medida, principalmente para TE 20%. Já em 4.34 e 4.35 a taxa de queda de acurácia se mantém bem abaixo do caso com 'Erro Tipo 1' nas RAMs, ficando por volta da metade do valor. Esses são os datasets com maior número de dados de treino e teste, isso pode ter sido o fatos que influenciou essa menor queda de acurácia, pois o ruído nas RAMs acabaria sendo menor em relação ao valor máximo de cada entrada de RAM, fazendo o processo de Bleaching ser pouco afetado.

E por último, nos datasets Bank e Survival (figuras 4.36 e 4.38 houve melhoria de acurácia para o modelo. Essa queda de acurácia conseguia ainda se manter presente

mesmo em altos valores de TE. A curva vermelha na figura 4.38 tem valor baixo por apresentar ainda vários valores co queda de acurácia negativa.

Em seguida repetimos o mesmo teste, só que agora injetando ruídos do tipo 'Erro Tipo 1', na implementação em array, em adição a se introduzir ruído durante o treino. Os resultados obtidos podem ser observados nas figuras 4.40 a 4.46 e os histogramas correspondentes são mostrados logo em seguida nas figuras 4.47 a 4.53. As linhas (e barras) vermelhas são o resultado deste experimento com dois tipos erros ao mesmo tempo e as ilnhas (e barras) azuis são dados de referência de um experimento igual ao da seção 4.3.1 em que é injetado apenas ruído nas RAMs do tipo 'Erro Tipo 1'.

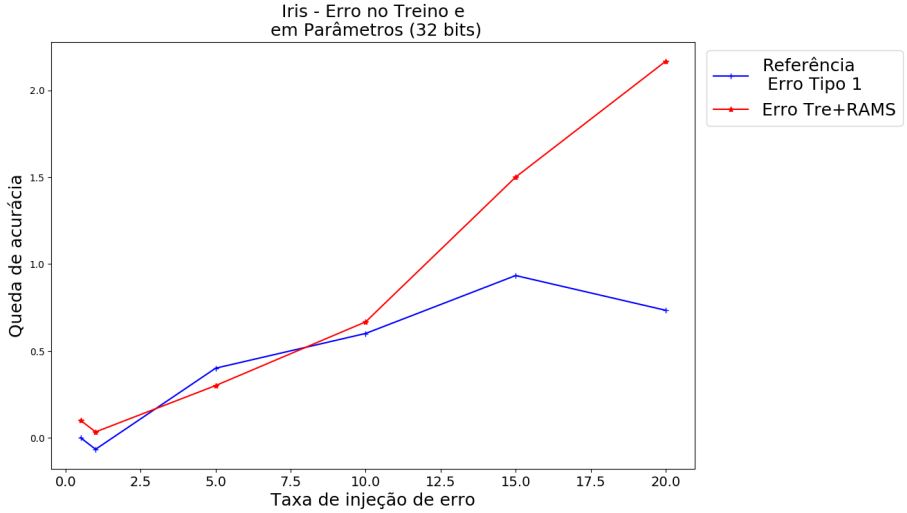


Figura 4.40: Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Irirs.

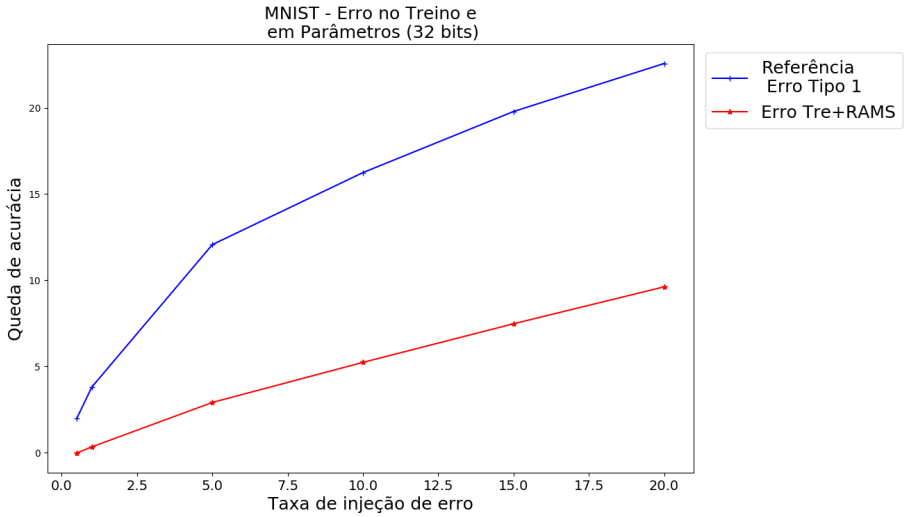


Figura 4.41: Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset MNIST.

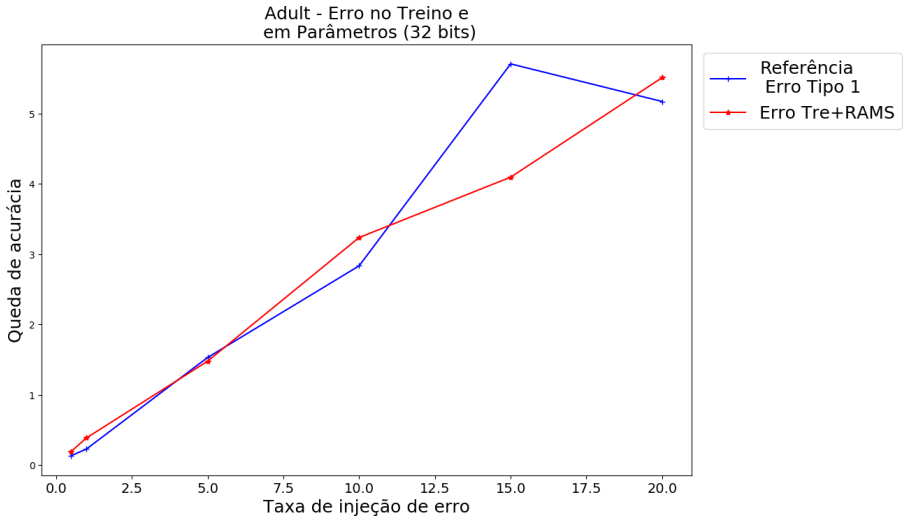


Figura 4.42: Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Adult.

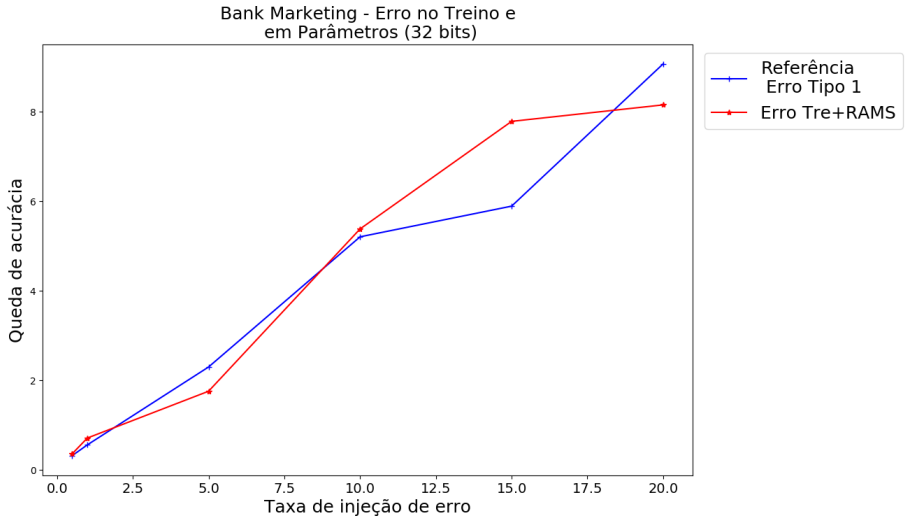


Figura 4.43: Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Bank.

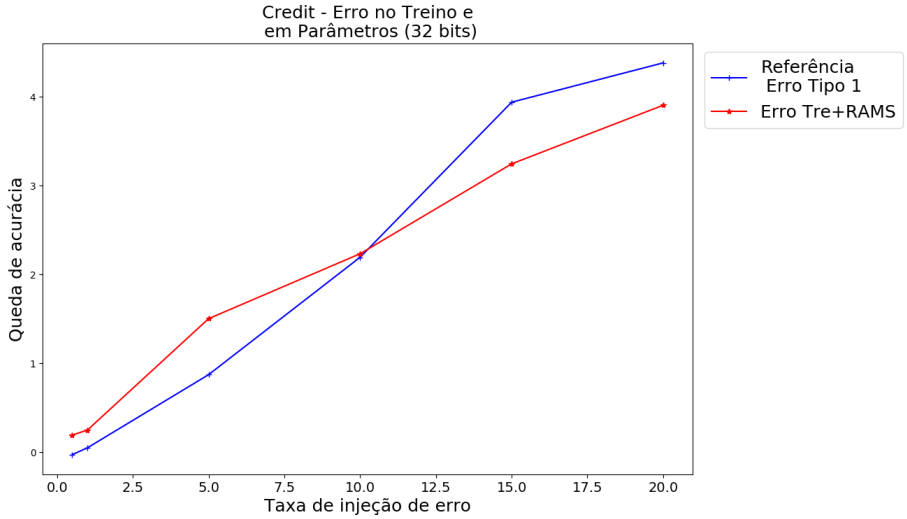


Figura 4.44: Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Credit.

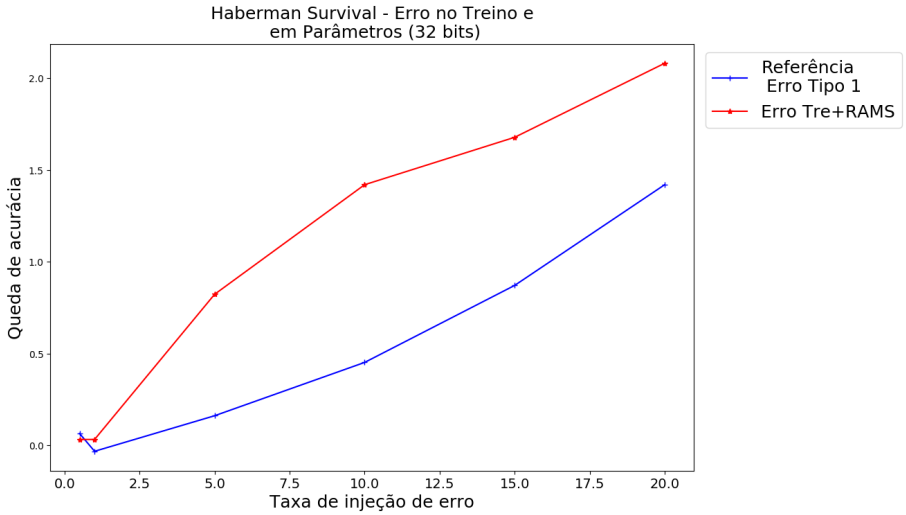


Figura 4.45: Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Survival.

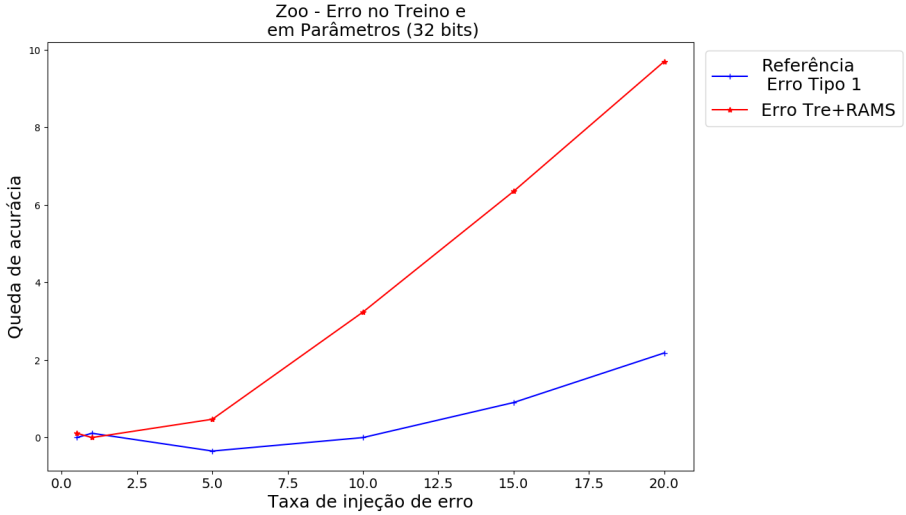


Figura 4.46: Queda de acurácia para erro injetado durante o treino e nos parâmetros salvos para o caso do dataset Zoo.

Novamente o tipo de resultado observado é bem variado. A curva mai chamativa fica sendo a 4.41 na qual a queda de acurácia se mantém bem abaixo da queda da curva de referência. As curvas da fiura 4.41 são extremamente próximas as da figura 4.34, mas foram sim treinadas com e sem inserção de ruído nas RAMs respectivamente.

De resto nas figuras 4.40, 4.41 e 4.46 mantiveram uma queda de acurácia basicamente igual a do caso em que apenas erro de reino era inserido (podendo se comparar com as figuras 4.33, 4.34 e 4.39). As figuras 4.42, 4.43 e 4.44 apresentaram comportamento similar ao caso com apenas adição de ruído nas RAMs. E por fim na 4.45o da-

taset Survival apresentou maior queda de acurácia em relação ao caso de referência. Esse comportamento fui altamente inesperado considerando as melhoras de desempenho observadas na figura 4.38. O mesmo pode ser dito para o dataset Credit da 4.44 em relação a figura 4.44.

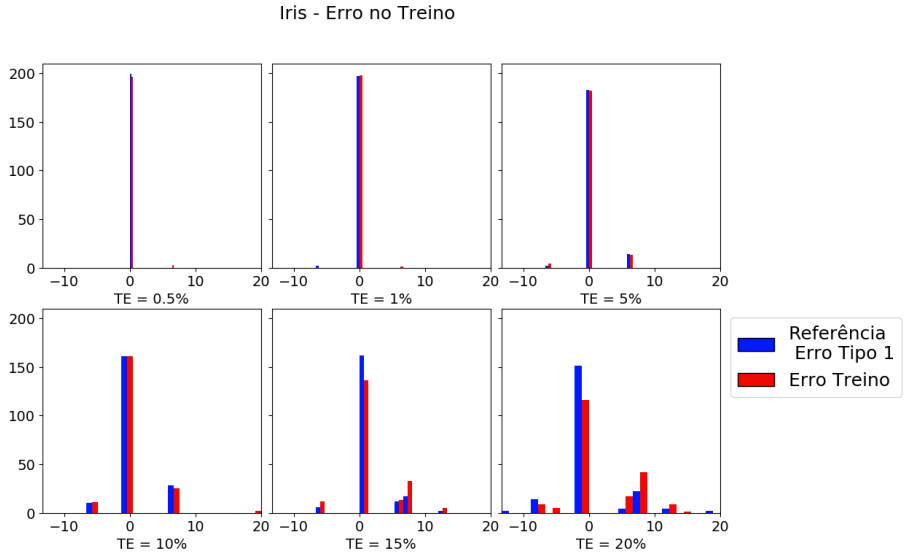


Figura 4.47: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Iris.

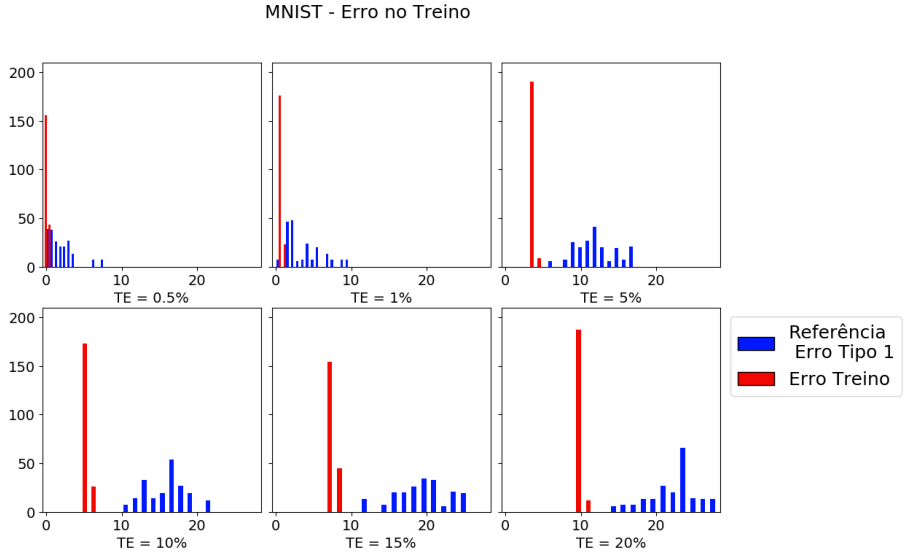


Figura 4.48: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset MNIST.

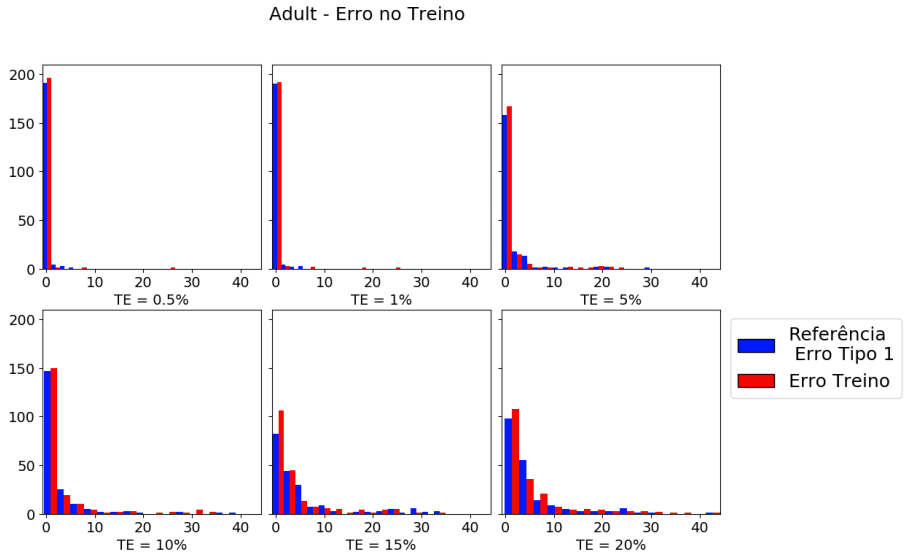


Figura 4.49: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Adult.

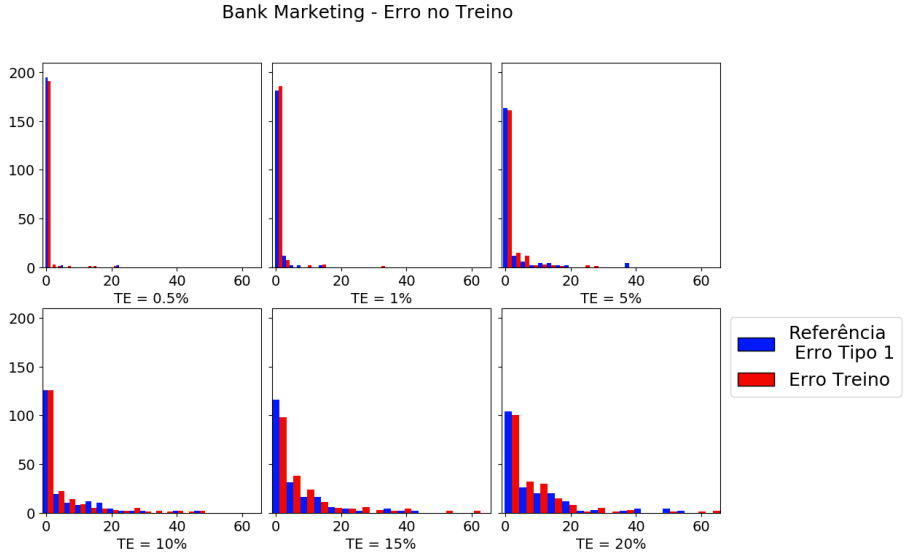


Figura 4.50: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Bank.

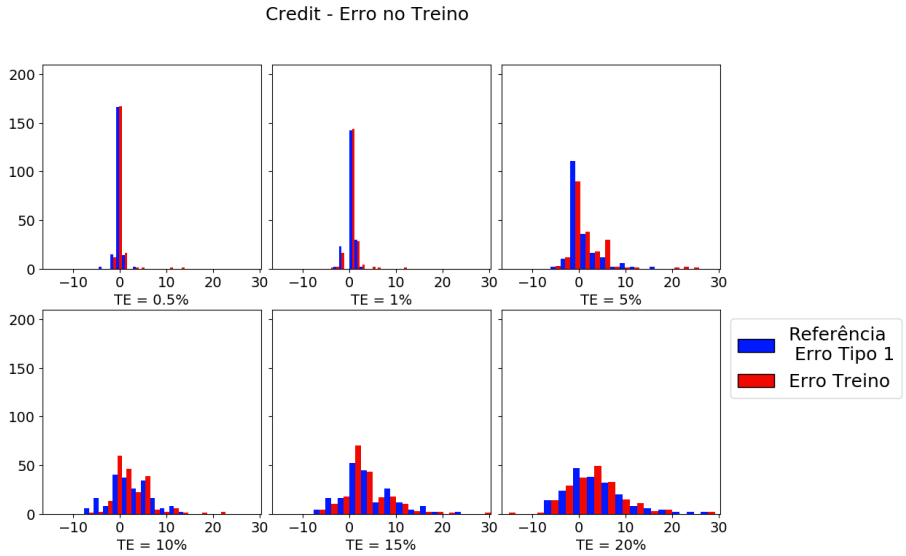


Figura 4.51: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Credit.

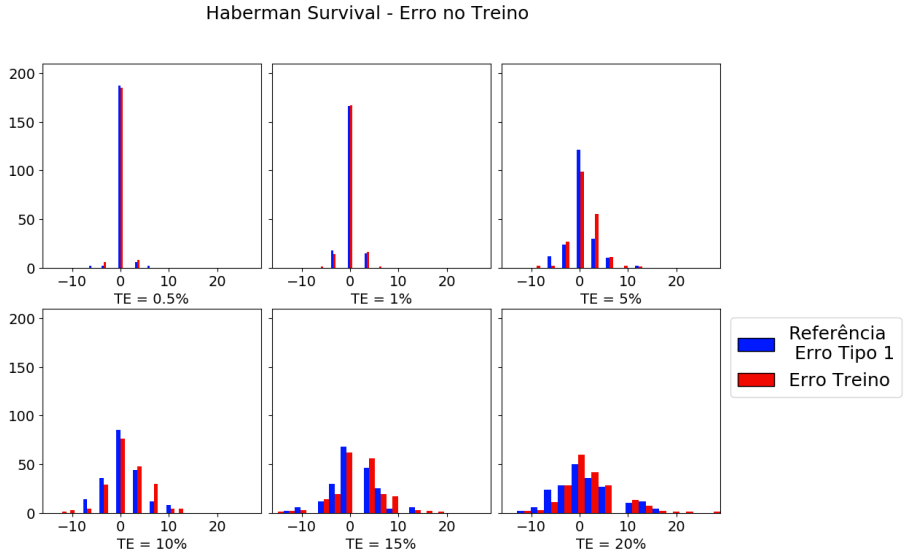


Figura 4.52: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Survival.

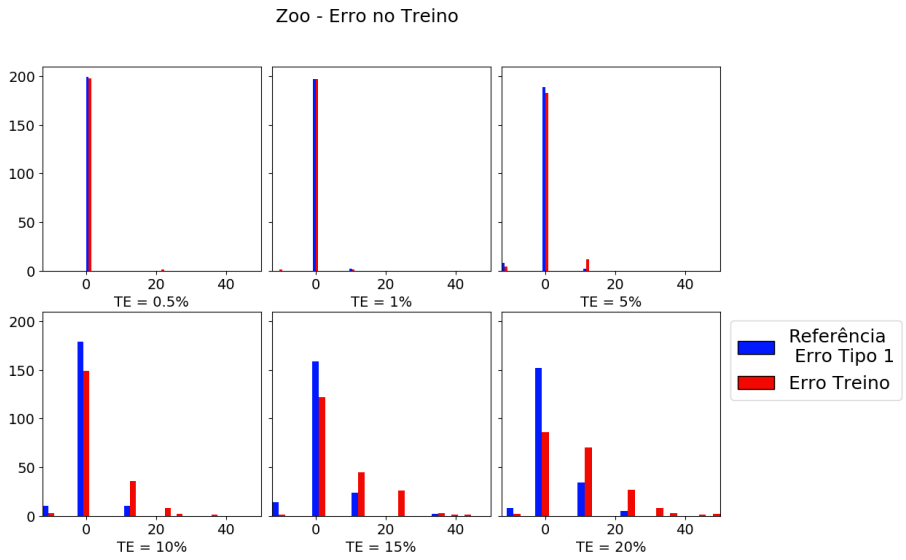


Figura 4.53: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Zoo.

Em relação aos histogramas, em todas as figuras a distribuição de quedas de aurácia foi próxima entre os casos com e sem erro no treino. Houveram apenas duas exceções, aprimeira sendo no dataset MNIST (figura 4.48 na qual o perfil de queda de acurácia novo (em vermelho) vai andando para a direita mais lentamente que o de referência (azul) e apresenta maior concentração de valores. a segunda exceção foi a da figura 4.53 onde podemos observar o aparecimento de uma "cauda"maior no histograma e com uma queda rápida do número de casos com queda de acurácia próxima de 0 (comportamento este esperado para um caso em que a queda de acurácia cresce significativamente com TE, o que é compatível com a figura 4.46).

4.5 Resultados dos experimentos de injeção de erros na execução

Nesta seção focaremos no efeito de erros ocorridos durante a execução do programa. O modelo WiSARD é computacionalmente menos demandante que outros modelos de redes neurais mas ainda apresenta algumas operações de soma e comparação durante sua execução (as outras operações são de acesso a memória). Os tipos de erro que podem ocorrer na execução são a leitura da entrada incorreta de uma RAM (tanto pela leitura incorreta da entrada quanto pelo cálculo incorreto do endereço), o somatório incorreto das saídas das RAMs para serem comparados (o somatório gera um valor aleatório, RAMs com valores acima ou abaixo do limiar de bleaching não são

contados corretamente ou alguns valores não são somados) e a comparação errônea dos valores de ativação dos Discriminadores (resultado aleatório da comparação).

Desses tipos de erro, os erros de endereçamento de RAM dependeriam da implementação da WiSARD e de medidas de segurança adicionadas a esta. Os erros de somatório das saídas das RAM já são mais diretos e podem ser analisados com simplicidade. Por fim o erro na comparação final (por ser um erro aleatório na etapa final) geraria um valor aleatório, o que adicionaria um bias na queda de acurácia, próximo do valor da acurácia multiplicado por (1 - TE). Este por apresentar um comportamento muito simples independente do funcionamento mais interno da WiSARD não será explorado. Desse modo focaremos nesta seção apenas no erro de soma dos valores de saída das RAMs.

Os experimentos seguirão a seguinte ordem. Um modelo WiSARD será treinado deixando-se um conjunto para teste. Este modelo terá sua acurácia medida e em seguida terá sua acurácia novamente medida por uma função que medirá a saída de cada RAM após bleaching e aleatoriamente irá modificar esta de 0 para 1 e vice-versa (isso representa que um valor pode ser adicionado ou não, o que não equivale a um valor aleatório ser sorteado como resultado final).

Os resultados dos experimentos para cada dataset estão plotados com a queda média de acurácia nas figuras 4.54 a 4.60. Neste caso as curvas vermelhas são as medias deste experimeto e as azuis são medidas de referências tiras do experimento descrito na seção 4.3.1.

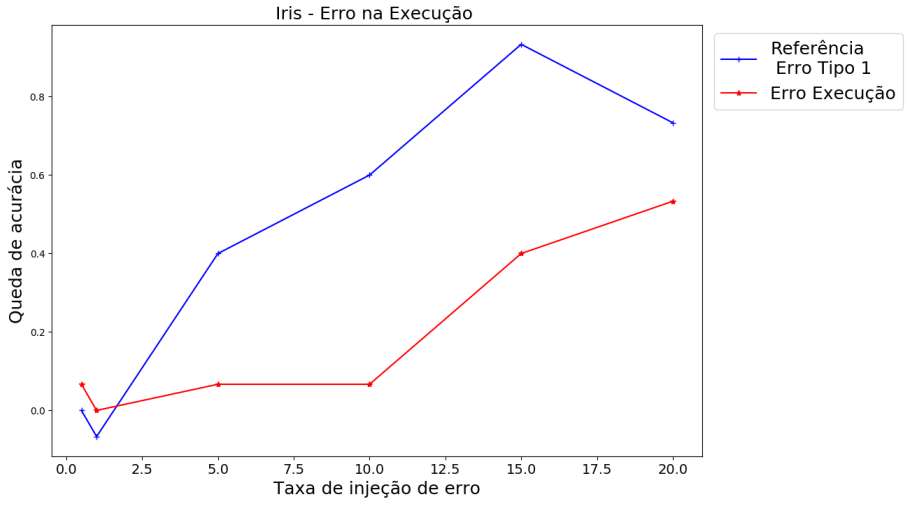


Figura 4.54: Queda de acurácia para erro injetado durante a classificação do dataset Iris.

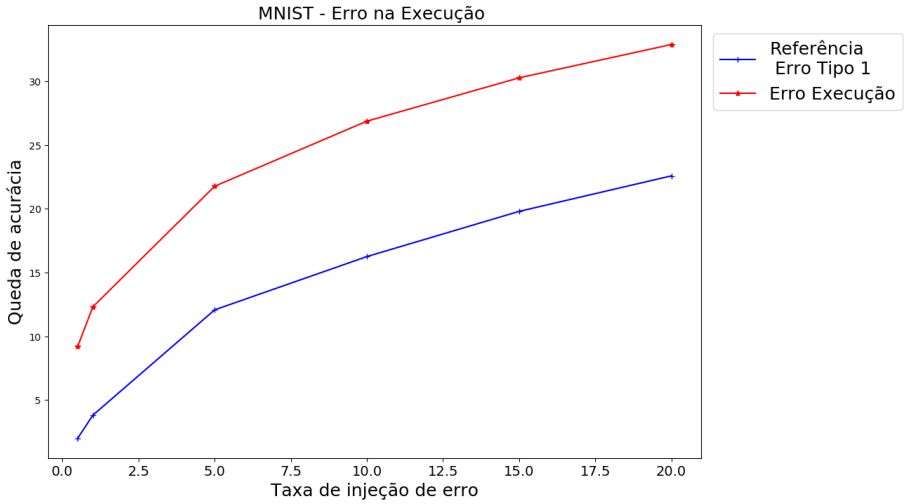


Figura 4.55: Queda de acurácia para erro injetado durante a classificação do dataset MNIST.

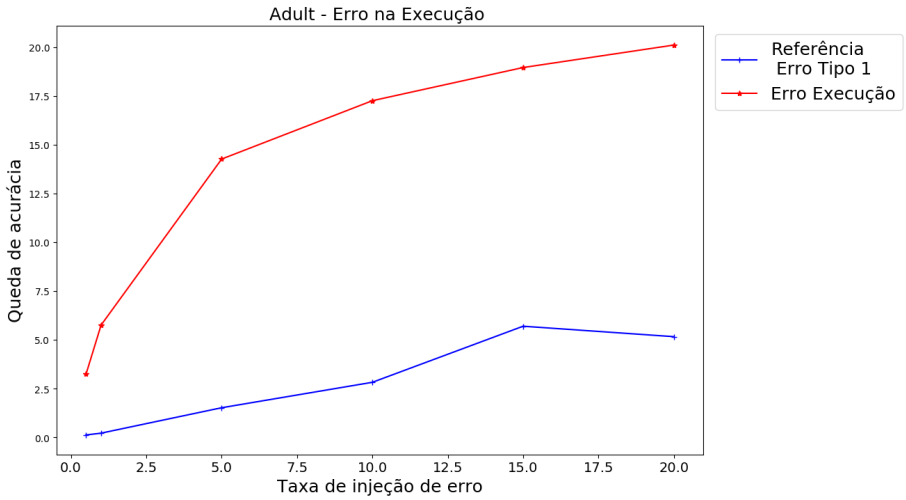


Figura 4.56: Queda de acurácia para erro injetado durante a classificação do dataset Adult.

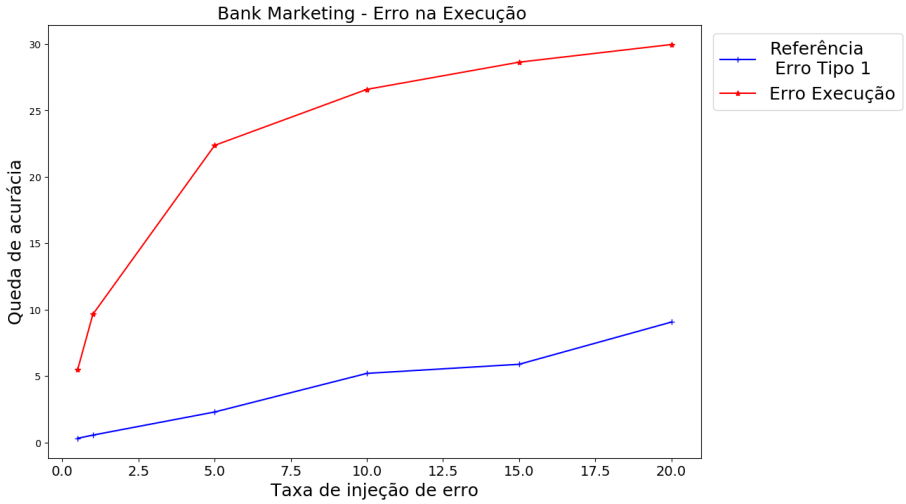


Figura 4.57: Queda de acurácia para erro injetado durante a classificação do dataset Bank.

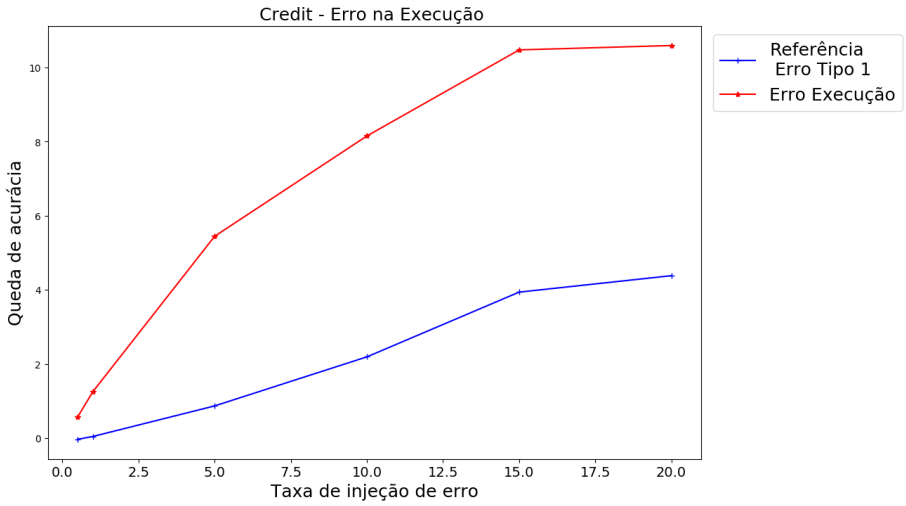


Figura 4.58: Queda de acurácia para erro injetado durante a classificação do dataset Credit.

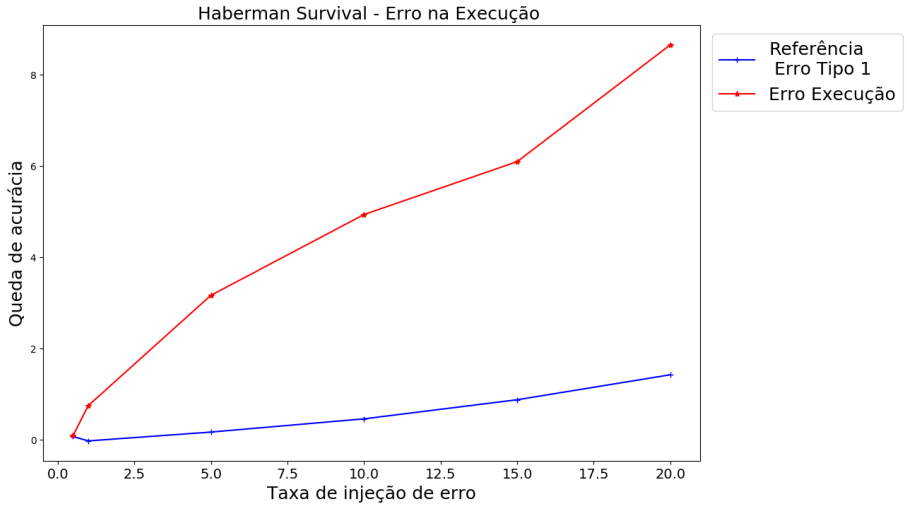


Figura 4.59: Queda de acurácia para erro injetado durante a classificação do dataset Survival.

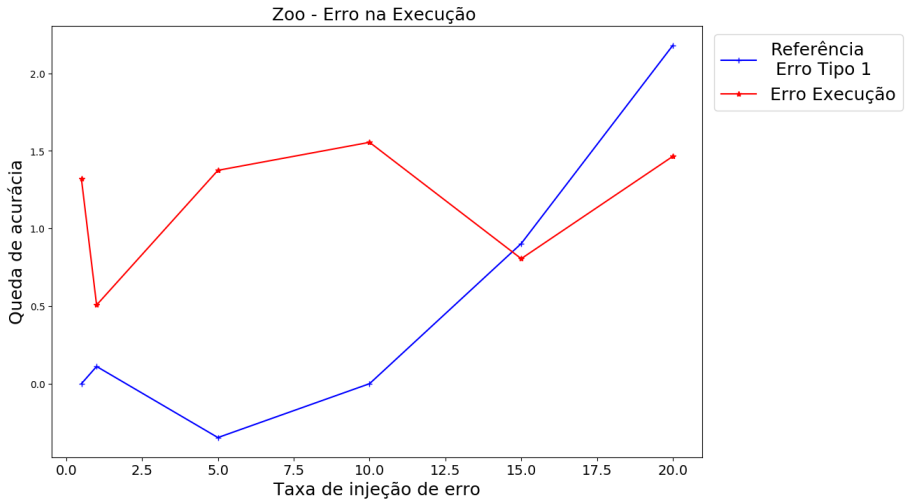


Figura 4.60: Queda de acurácia para erro injetado durante a classificação do dataset Zoo.

Pelo observado nas figuras 4.55, 4.56, 4.57, 4.58 e 4.60 a queda de acurácia já começa com um valor elevado, mesmo para baixas taxas de injeção de erro. O valor dessa perda de acurácia também cresce rápido com TE, mas se estabiliza por volta de TE 10%, valor no qual a derivada do gráfico se reduz bastante. A maior queda de acurácia observada também é significativamente maior do que nos casos com injeção de ruído nas entradas das RAMs, indicando que este tipo de erro é mais detrimental para a acurácia do modelo. O fato da queda de acurácia já começar alta e só ir aumentando pode indicar que quando visualizarmos o histograma de quedas de acurácia a moda estará longe do zero, possivelmente com nenhum valor com queda de acurácia

0.

Seguindo o exemplo do erro de treino iremos repetir o experimento anterior introduzindo agora ruído do tipo 'Erro Tipo 1' nas RAMs da WiSARD para analizar se isso irá gerar algum comportamento diferenciado. Os resultados estão mostrados nas figuras 4.61 a 4.67. Os histogramas aparecem em seguida nas figuras 4.68 a 4.74.

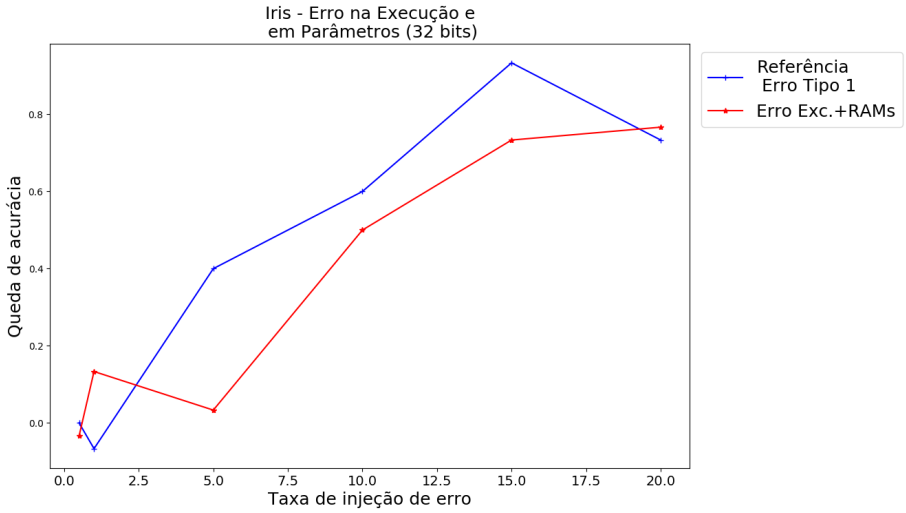


Figura 4.61: Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Irirs.

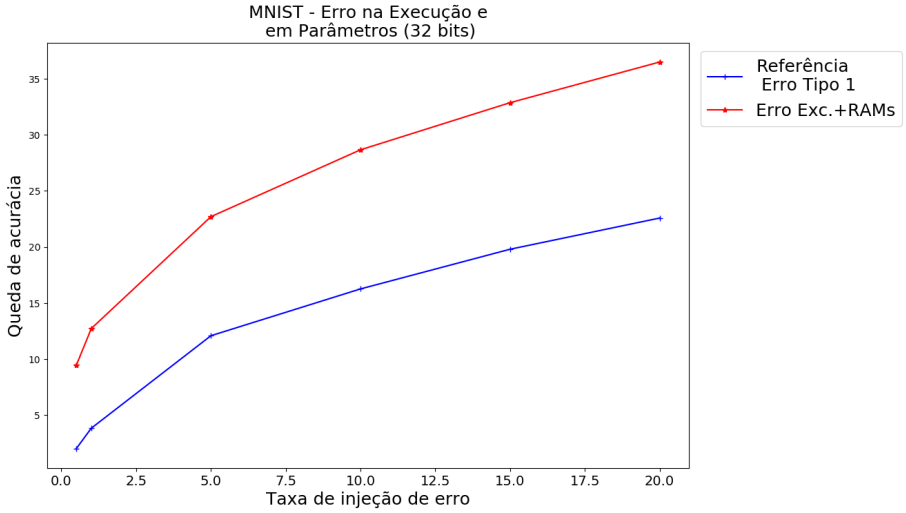


Figura 4.62: Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset MNIST.

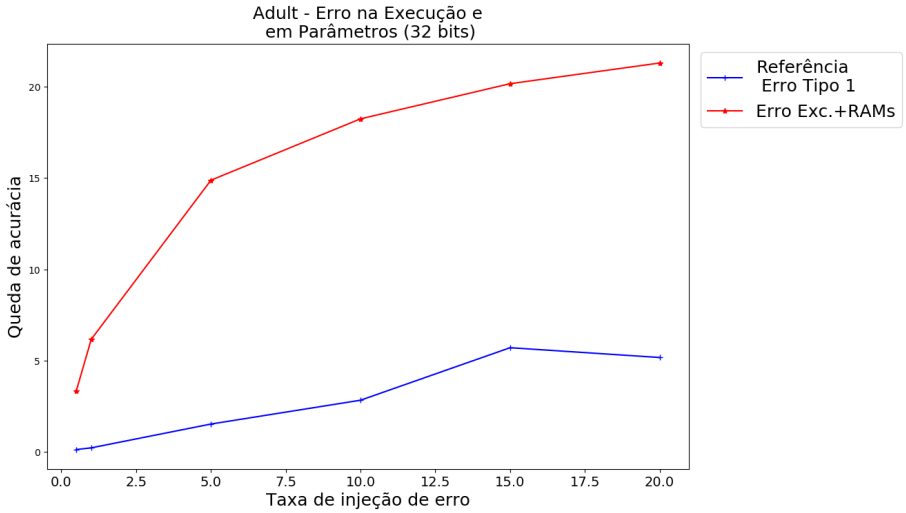


Figura 4.63: Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Adult.

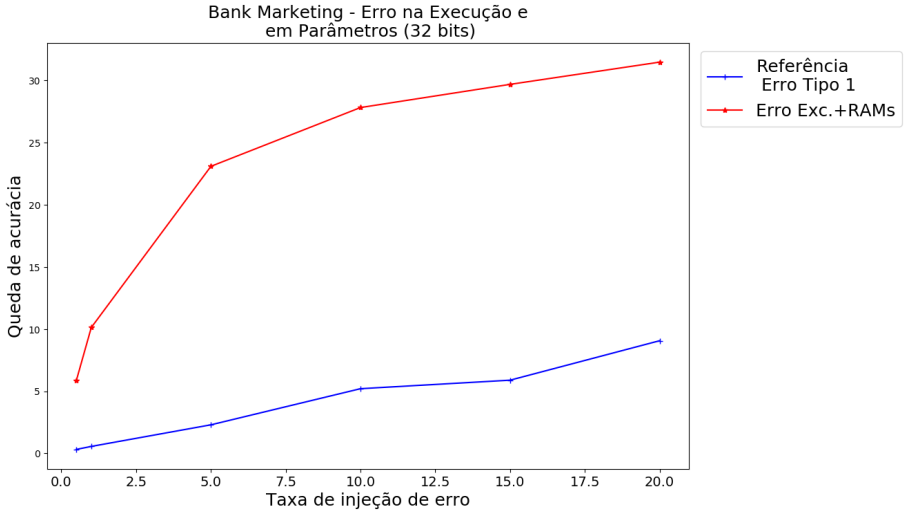


Figura 4.64: Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Bank.

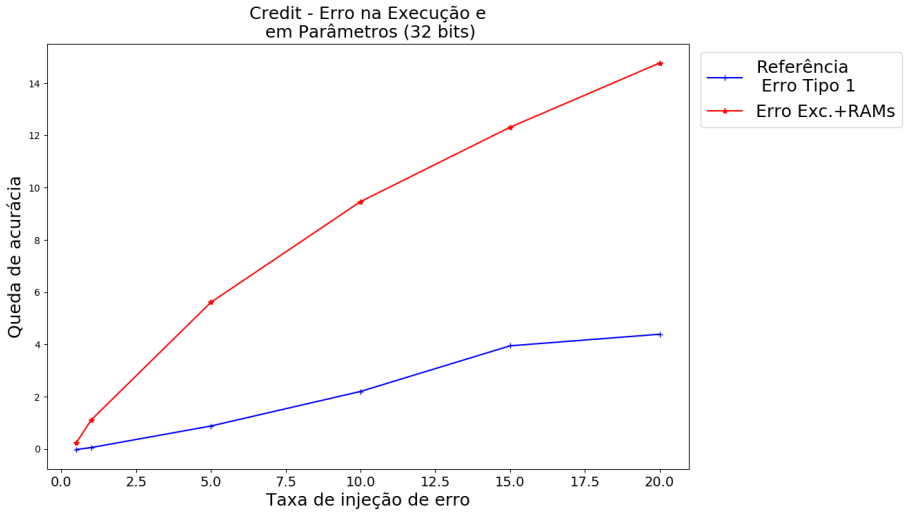


Figura 4.65: Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Credit.

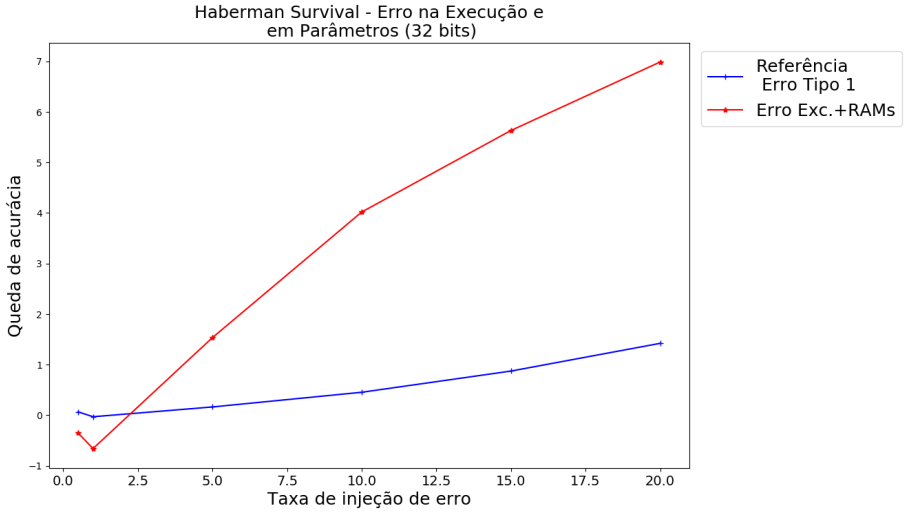


Figura 4.66: Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Survival.

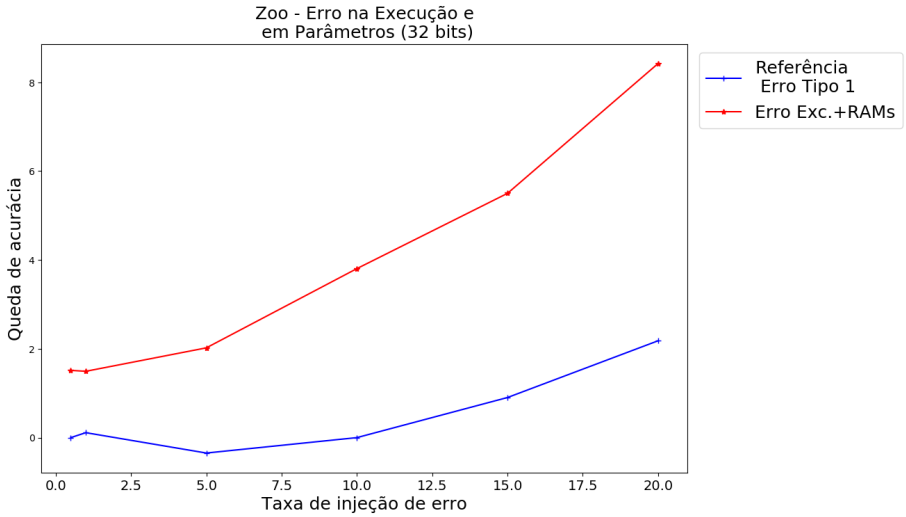


Figura 4.67: Queda de acurácia para erro injetado durante a classificação e nos parâmetros salvos para o caso do dataset Zoo.

Observado as figuras pode-se perceber que a conjunção dos erros no geral degradou ainda mais acurácia dos modelos. Podemos observar este caso comparando a figura 4.61 em relação a figura 4.54, a figra 4.65 em relação a 4.58 e a figura 4.67 em relação a 4.60. Este efeito entretanto não é perceptível nas curvas das figuras 4.62, 4.63 e 4.64 as quais são praticamente idênticas em relação a suas equivalentes sem o 'Erro de Tipo 1'. Estes são os datasets com maior número de bits na entrada (o que ocasiona também um maior número de RAMs por Discriminador), o que pode indicar uma relação entre os dois elementos.

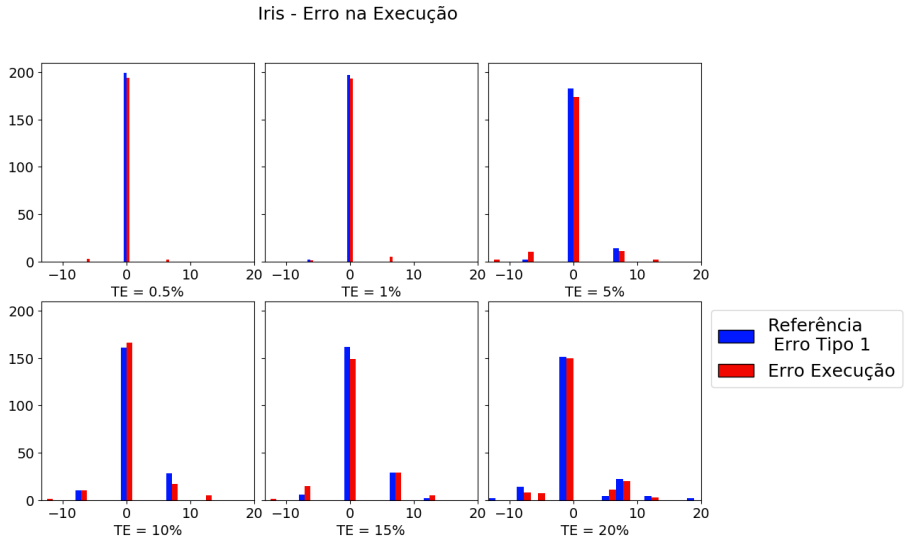


Figura 4.68: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Iris.

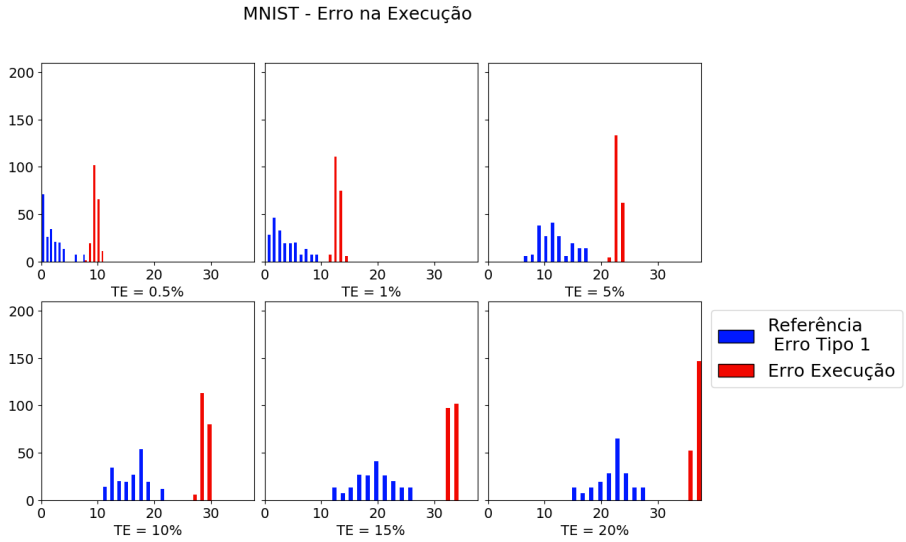


Figura 4.69: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset MNIST.

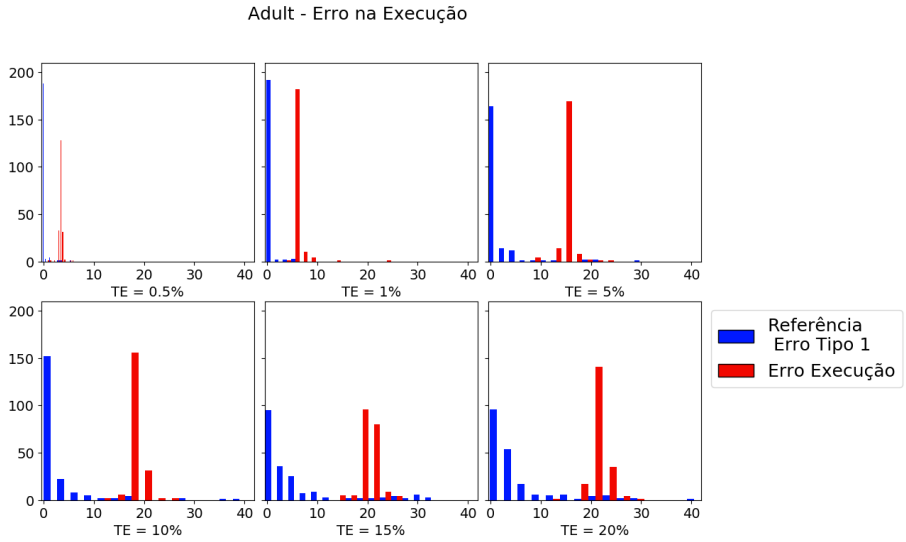


Figura 4.70: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Adult.

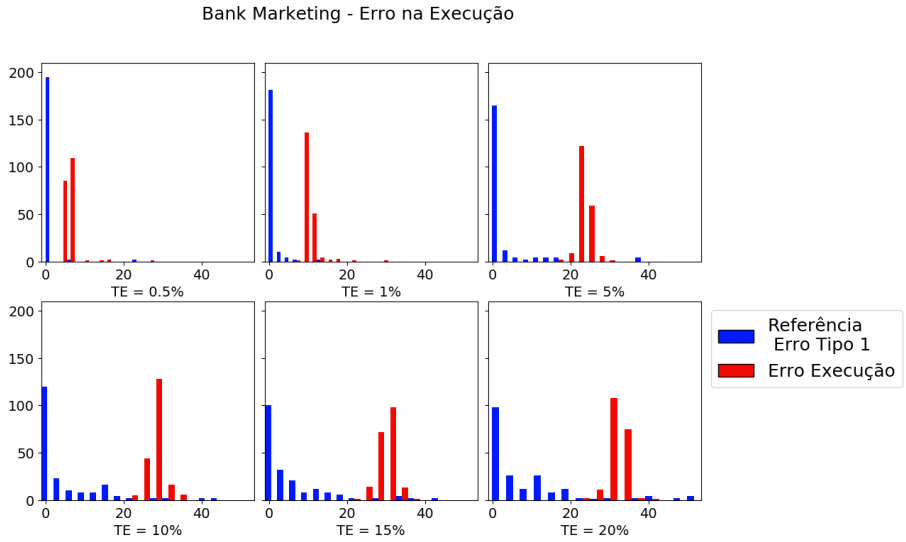


Figura 4.71: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Bank.

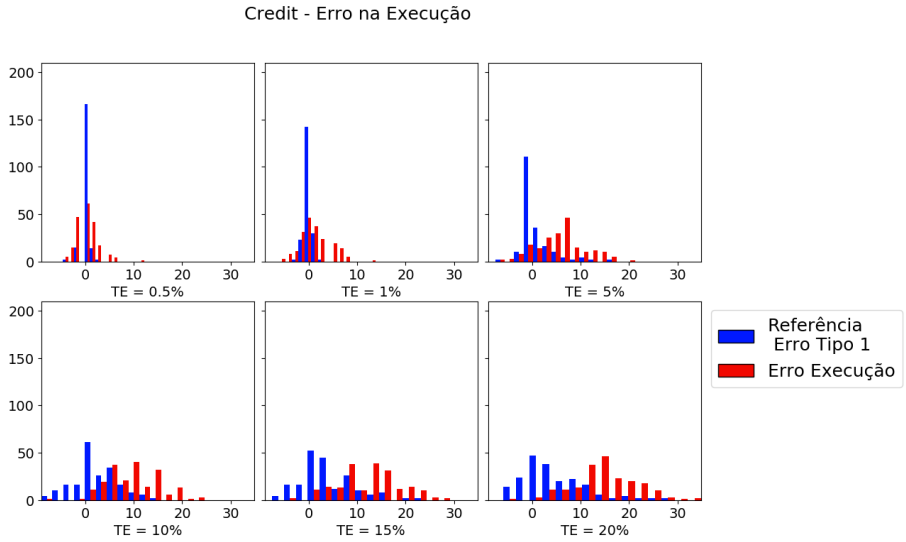


Figura 4.72: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Credit.

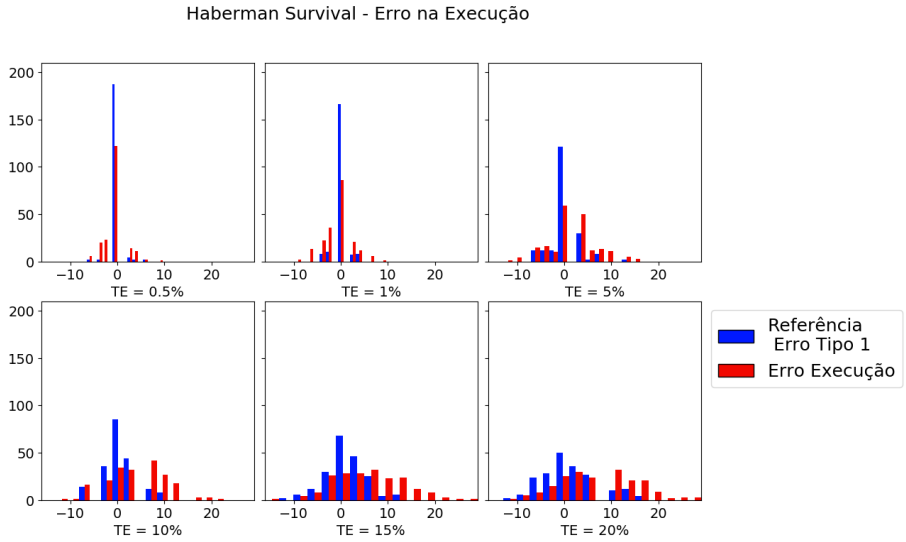


Figura 4.73: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Survival

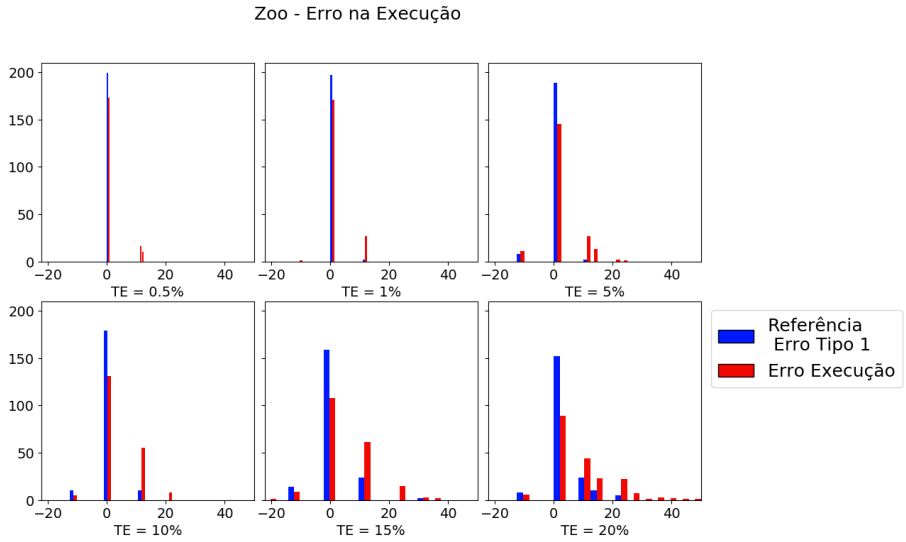


Figura 4.74: Histograma das quedas de acurácia medidas, no caso de injeção de ruído do tipo *bit-flip* nos parâmetros da rede WiSARD, para seis valores de taxa de injeção de erro no dataset Zoo.

Nos histogramas um comportamento bem definido pode ser percebido. Os valores de queda de acurácia no caso com erro de execução e de 'Erro de Tipo 1' são visivelmente superiores aos do caso com apenas 'Erro de tipo 1'. A moda das quedas de acurácia também é visivelmente superior e em casos como os da figura 4.69, 4.70 e 4.71 não chega a contabilizar nenhum caso com queda de acurácia 0. No caso das figuras 4.72, 4.73 e 4.74 a moda já começa em zero mas vai se distanciando conforme TE aumenta. Nesses casos a distribuição dos valores também é mais homogênea, estando estes menos concentrados do que nos casos das figuras 4.69, 4.70 e 4.71.

4.6 Resultados dos experimentos com conjunção dos erros

Nesta seção verificaremos os efeitos de todos os tipos de erros ocorrendo em conjunto. Serão introduzidos erros durante o treinamento, depois nas RAMs já treinadas (apenas nos 16 bits menos significativos e nas entradas treinadas) e, enfim, no cálculo da ativação dos Discriminadores. Todos os tipos de erro apresentarão a mesma taxa de injeção e serão testadas as mesmas seis taxas testadas nas sessões anteriores.

Os resultados do experimento são mostrados como médias de queda de acurácia nas Figuras 4.75 a 4.81.Novamente as curvas azuis são casos apenas com 'Erro Tipo 1' injetado nas RAMs com implementação em array (experimento da seção 4.3.1). A curva verde é do experimento da seção 4.5, com 'Erro Tipo 1' nas RAMs e erro na etapa

de soma de resultado das RAMs da execução. Por fim a curva vermelha mostra a queda de acurácia para o experimento atual.

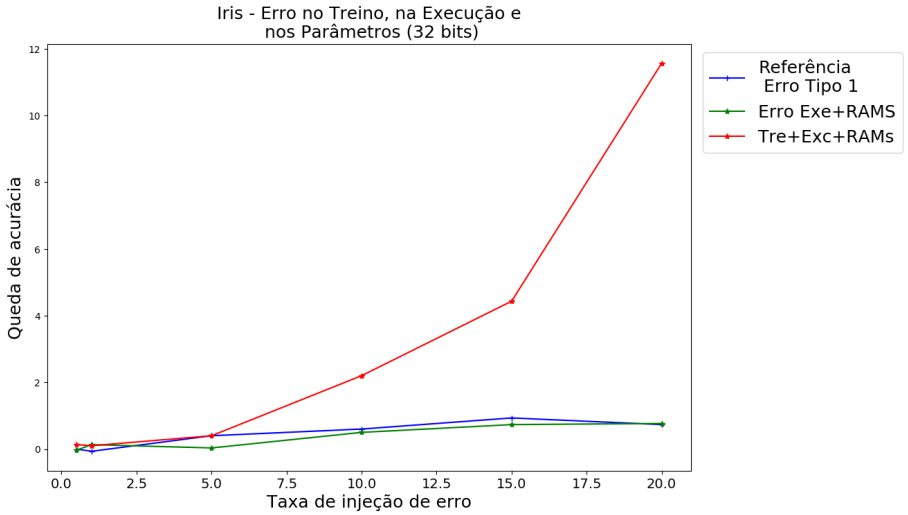


Figura 4.75: Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Irirs.

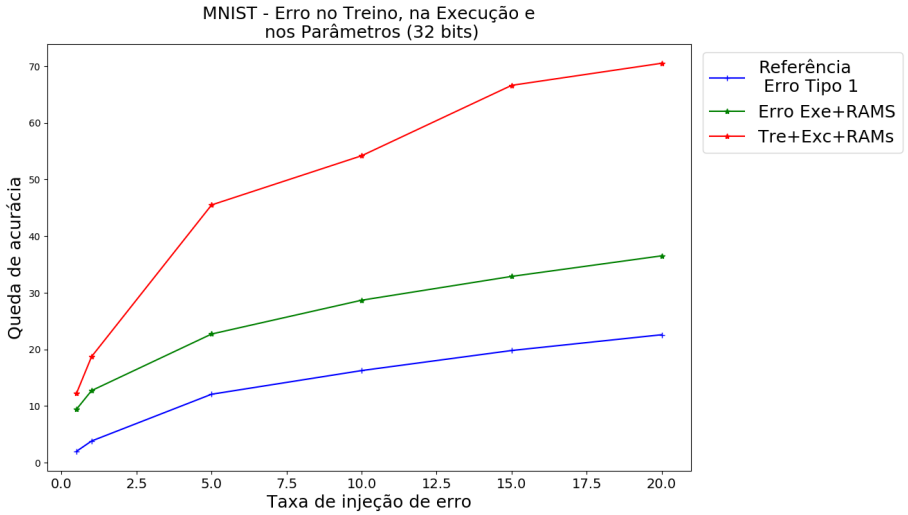


Figura 4.76: Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset MINST.

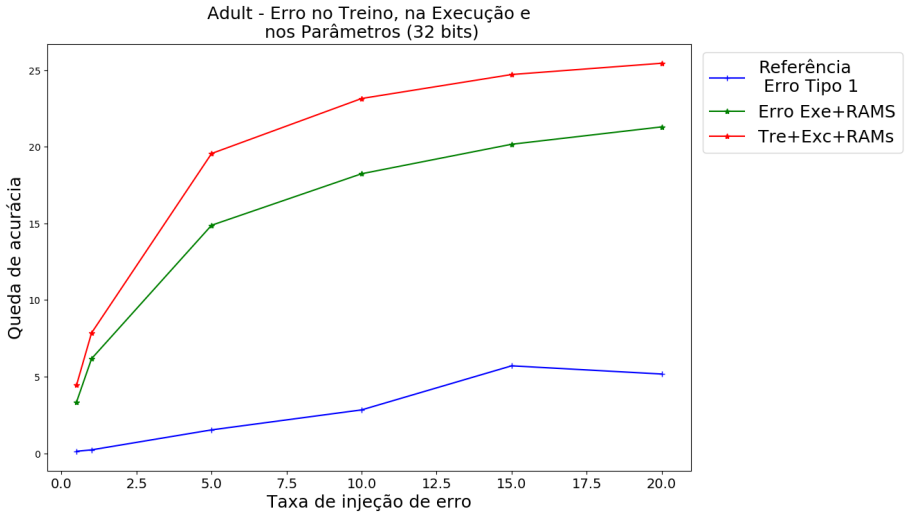


Figura 4.77: Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Adult.

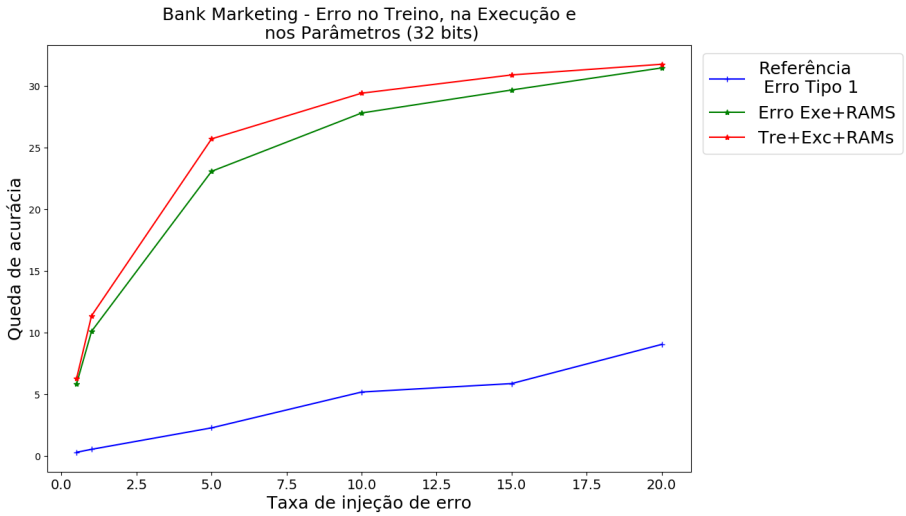


Figura 4.78: Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Bank.

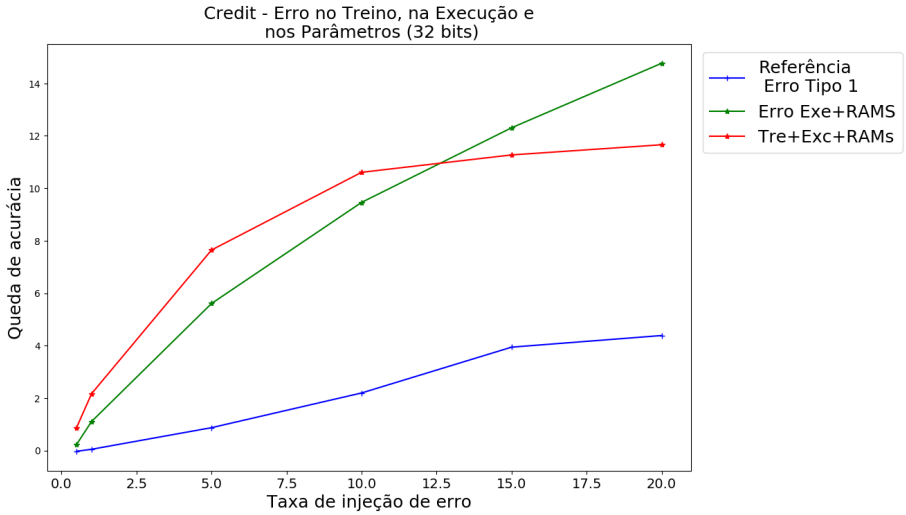


Figura 4.79: Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Credit.

Figura 4.80: Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Survival.

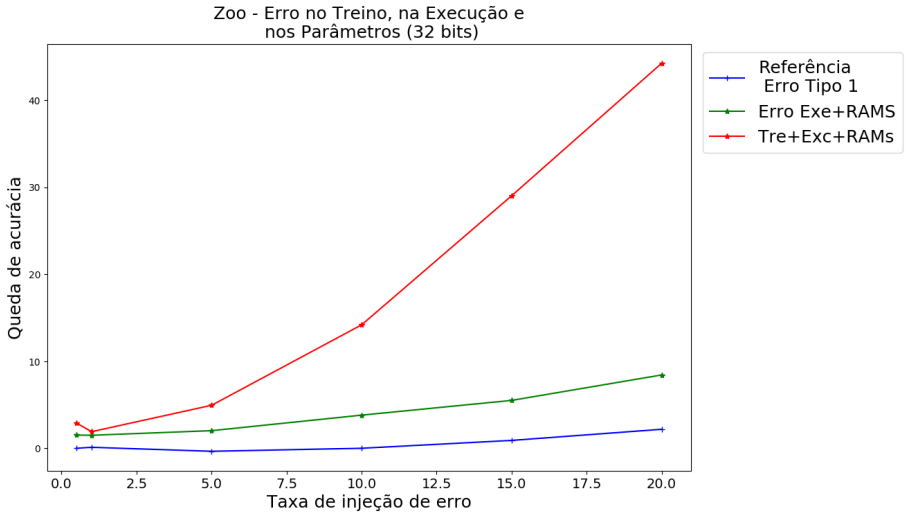


Figura 4.81: Queda de acurácia para erro injetado durante o treino, na classificação e nos parâmetros salvos para o caso para o dataset Zoo.

Observado as figuras o efeito da combinação de todos os três tipo de erro é maior do que a da combinação de erro de treino com ruído nas RAMs. Os dois casos mais extremos são os das figuras 4.75 e 4.81 (Iris e Zoo) nas quais a queda de acurácia com TE 20% passa de 3 vezes o valor das outras curvas. Nas figuras 4.76, 4.77, 4.78 e 4.79 a linha vermelha se mantém mais próxima da verda, mas no geral sempre acima desta (as vezes por pouco, as vezes por um valor significativo). E por fim temos a curva vermelha da figura 4.80 que aparece como um resultado inesperado. Apesar de ter apresentado melhora de acurácia na inserção de erro no treino (figura 4.38) o dataset Survival não apresentou o mesmo comportamento ao se misturar o erro de treino com o de RAM (figura 4.45). Ainda assim esse comportamento de melhora de desempenho pode ser observado aqui, deixando claro que estes efeitos de melhora funcionam de forma bem não linear.

4.7 Comparação de Modelos

A queda média de acurácia não indica o quanto um determinado modelo será prejudicado pela inclusão de ruído em seus parâmetros. Por ser uma média ela não define nem a maior nem a menor variação de acurácia que se pode esperar. Também não garante que a queda irá ocorrer. Entretanto ela é útil na comparação de modelos pois indica o quanto o modelo é afetado. Uma média maior indica que mais casos com ruído geraram modelos com pior acurácia. Ou indica que os modelos com ruídos que apresentam queda de desempenho estão com uma queda maior ainda. Ela não indica como o modelo será afetado, mas indica o quanto se espera que ele seja. Por isso é de interesse comparar o modelo WiSARD que foi testado com o de outros.

Para tanto foram implementados um modelo WiSARD com array e valores de 32 bits, uma rede neural de duas camadas internas completamente conexas e 256 nós por camada e um modelo de regressão logística. Os dois outros modelos recebem as mesmas entradas que a WiSARD mas sem serem binarizadas. Por fim ruído do tipo *bit-flip* é colocado em parâmetros treinados aleatórios e em bits de posição randômicas. A acurácia é então testada utilizando a mesma divisão de dados de treino e teste descrito na seção 4.1. Nos plots (figura 4.26 a 4.32) a queda média de acurácia da WiSARD é desenhada em azul, da regressão logística em vermelho e da rede neural em preto. O ruído injetado nos outros modelos foi feita com *bit-flip* até o o bit 31, que equivale a considerar que seus parâmetros foram implementados com float 32 bits (signed).

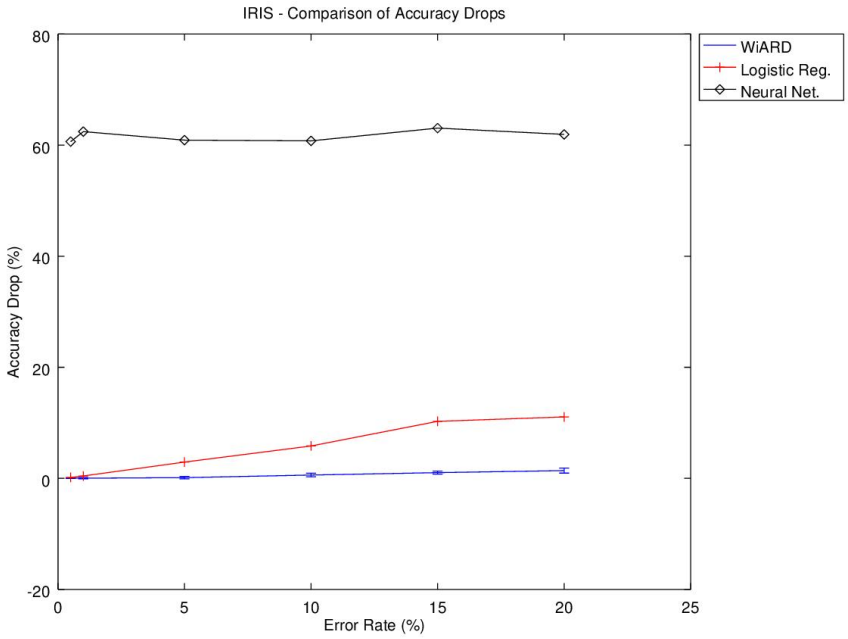


Figura 4.82: Queda de acurácia para o dataset Iris ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística.

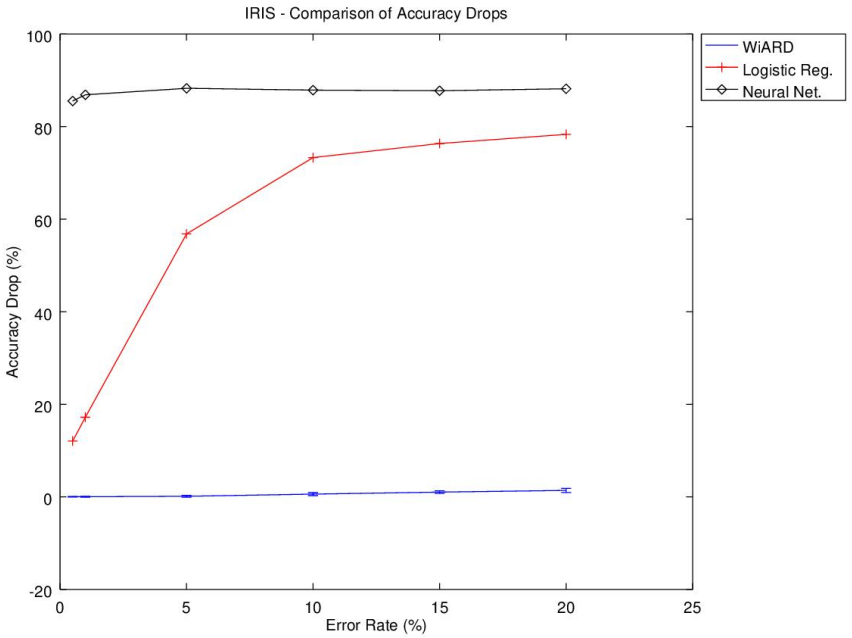


Figura 4.83: Queda de acurácia para o dataset MNIST ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística.

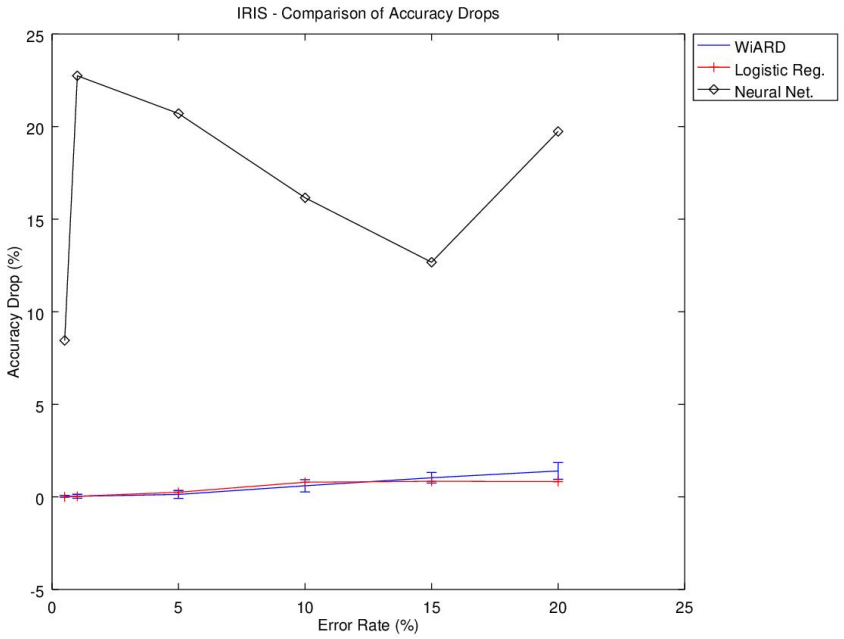


Figura 4.84: Queda de acurácia para o dataset Adult ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística.

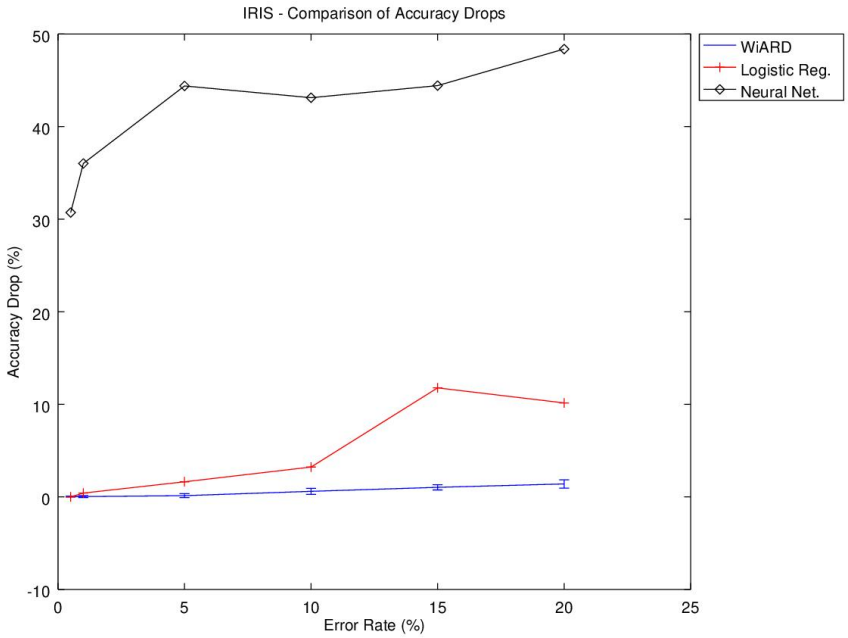


Figura 4.85: Queda de acurácia para o dataset Bank ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística.

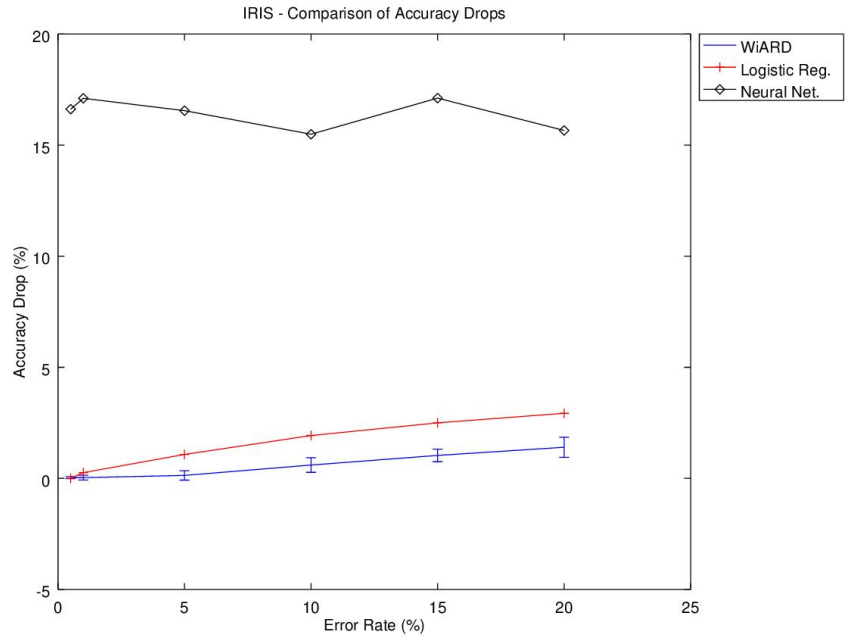


Figura 4.86: Queda de acurácia para o dataset Credit ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística.

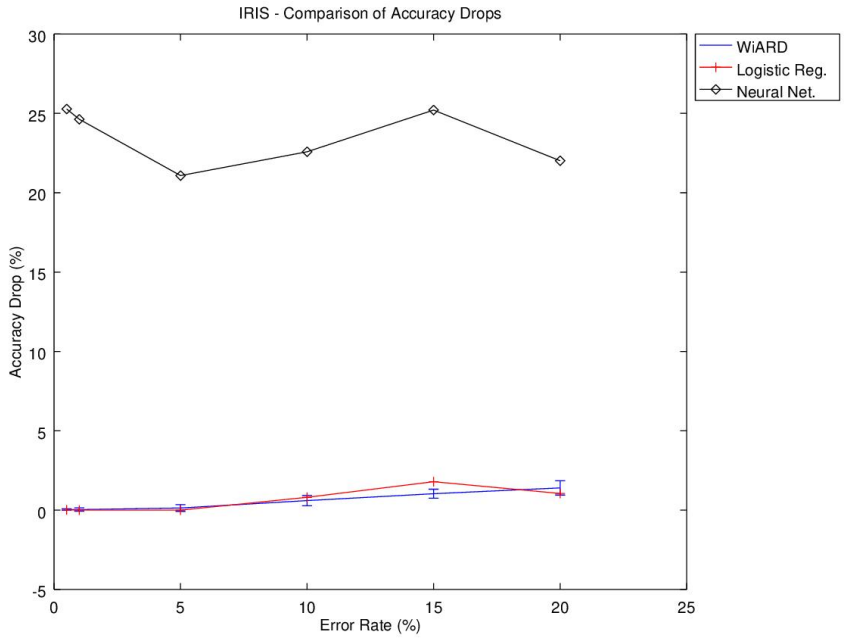


Figura 4.87: Queda de acurácia para o dataset Survival ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística.

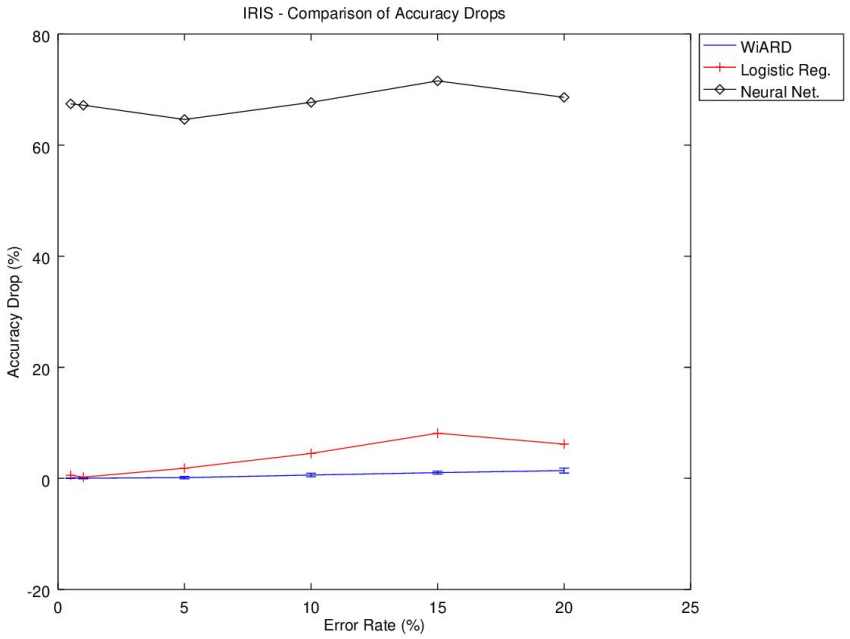


Figura 4.88: Queda de acurácia para o dataset Zoo ao ruído induzido nos parâmetros treinados de um modelo WiSARD, rede neural e de regressão logística.

A rede neural apresentou grandes quedas de acurácia, sendo que a queda já é alta

mesmo para valores baixos de TE. Os valores observados foram diversos, variando desde próximo a 10% a até 80%, mas tendem a se manter estáveis para diferentes taxas de TE, o que poderia dizer que o modelo deixa de funcionar mesmo com baixo ruído. Nas figuras 4.82, 4.83 e 4.85, a queda pode indicar que o a rede começou a classificar de forma aleatória ou tendenciosa para uma única classe, pois estas quedas deixariam a rede próxima de uma acurácia igual a 1/(número de classes). O modelo é claramente suscetível a *bit-flips* um dos possíveis motivos é o fato de seus parâmetros serem floats, que apresentam uma codificação mais complexa que inteiros em sua forma binária tradicional. Outro ponto relevante para essa suscetibilidade ao ruído inserido é uma questão da estrutura de redes neurais. Os neurônios de redes neurais tradicionais utilizam funções com forma próxima a de uma sigmóide para calcular sua ativação. Isso quer dizer que alterações bruscas em um único peso seriam capazes de prender um neurônio sempre na ativação máxima ou mínima por se tornar mais relevante do que os outros pesos que são utilizados nas entradas no neurônio.

O modelo de regressão logística, por sua vez, se apresentou bem mais estável que o de rede neural. Sua taxa de erro crescia conforme TE aumentava, mas com derivada que parece decrescer ao passar de um TE de 15%. A exceção desse comportamento foi para o dataset MNIST na figura 4.83. Aqui o modelo já apresenta queda de acurácia de mais de 10% mesmo em baixo TE. Esse valor vai crescendo e começa a se estabilizar próximo de 75% de queda. Entretanto, como dito antes, esta é a exceção e na maior parte dos casos a queda de acurácia para um TE de 20% é de menos de 10%.

A WiSARD já apresentou um comportamento mais resiliente do que os outros modelos sendo a com menor queda de acurácia a partir de 5% de TE para 5 dos 7 datasets. As figuras 4.84 (Adult) e 4.87 (Survival) indicam os dois casos em que o modelo de regressão logística tem queda de acurácia equiparável a da WiSARD. No geral podemos dizer que a WiSARD é mais resiliente do que a rede neural (2 camadas ocultas, 256 neurônios) e do que a regressão logística. Isso provavelmente se deve pela WiSARD apresentar parâmetros inteiros sem sinal que são bem menos afetados por bit-flip. O número de parâmetros maior da WiSARD parece ser uma vantagem também, mas como observado no primeiro teste da seção anterior, a acurácia de uma implementação de dicionário (com número reduzido de parâmetros e sem "parâmetros mortos") foi similar (se não melhor que) a da implementação com array para todos os datasets testados.

Capítulo 5

Conclusão

O comportamento observado para o experimento de injeção de erro randômico indica que a WiSARD é sim afetada mais com níveis maiores de ruído sem parecer chegar a um patamar final dentro da faixa de 20% de erro. Entretanto é interessante saber, como foi observado na sessão 4.3.3 que erros tendenciosos ou com uma distribuição mais bem comportada geram uma deterioração menor do modelo o que é interessante pois essa situação simula determinados erros de hardware (por exemplo quando uma baia de conexão fica ionizada e mantém sempre o valor em 1 e 0, independente do valor lido nela).

Considerando o comportamento do modelo quando limitamos o ruído apenas aos parâmetros já com algum treino (implementação em dicionário testada na sessão 4.3.2, o fato dele acompanhar a curva, ou pelo menos a tendência da curva, de perda de acurácia dá a entender que as classes estão bem diferenciadas dentro de cada Discriminador. Por causa disso, erros fora das posições treinadas ou são equilibradas por erros em outros Discriminadores ou pela esparsidade da rede tem maior chance de cair em espaços que não serão acessados (possivelmente por não serem valores de n-uplas representativos de nenhuma classe na maior parte das RAMS). Ainda assim alguns resultados apresentaram queda de acurácia bem menor nesta implementação. Considerando que não existem desvantagens observadas e ocasionalmente existem vantagens pode ser interessante utilizar esta implementação das RAMs para tentar aumentar a resiliência da WiSARD, mas para tal seriam necessários testes de *bit-flip* no dicionário em si e não apenas nos valores salvos, para saber que tipo de efeito poderia aparecer.

No caso de teste com o ruído nos parâmetros sendo limitado apenas aos 16 primeiros bits poderia indicar que modelos com mais RAMs estão mais protegidos da influência de ruído randômico em seus parâmetros quando este ruído é de intensidade (valor) menor. Não fica muito claro como o valor do ruído influenciaria o modelo, já que a WiSARD se importa mais com a presença de um valor do que seu módulo. Também é difícil imaginar que ruídos de 16 bits (representando valores de 2^{15}) gerariam

muitas rodadas de bleaching, não sendo esperada que essa seja a relação com o desempenho melhor.

Apesar de soar artificial o efeito de se limitar o ruído apenas aos bits de baixo valor pode ser obtido para datasets cuja classe com maior número de exemplos de treino não apresente mais de 65.535 exemplos. O efeito seria gerado utilizando-se uma máscara binária sobre os valores lidos. Esta máscara binária zeraria todos os bits mais significativos dos valores lidos. Outra forma de evitar esses erros seria rodar um teste de comparação. Ao fim da etapa de treino da rede maior valor escrito em uma RAM pode ser guardado (com redundância para evitar erros caso seja modificado) e cada valor de RAM lido é comparado com este. Qualquer bit em uma posição mais significativa que o valor guardado é então removido do valor lido da RAM. Por último a mudança mais simples, mas que intensifica a limitação imposta na primeira implementação sugerida, seria o de codificar os valores guardados em unsigned short int (16 bits) ou tipo equivalente. Mantendo a binary word que guarda os valores em 16 bits quer dizer que um bit flip nunca gerará um ruído de valor 2^{16} ou maior.

O erro durante a contagem das RAMs ativas (erro de execução) foi de longe o mais significativo em termos de queda de acurácia. Não só por seus valores acentuados, como por reduzir consideravelmente e até zerar o número de casos com queda de acurácia zero. Por causa disso pode ser de interesse focar os esforço nesta parte se for considerado uma implementação da WiSARD que precise de maior resiliência. O erro de treino por outro lado pode apresentar diversos tipos de comportamentos então é difícil tomar decisões baseadas apenas neste.

O dataset survival se mostrou como de interesse pela sua capacidade de apresentar melhoras de acurácia mesmo para altos valores de TE. Entretanto esse funcionamento não é sempre observado, o que faria um estudo sobre as características do dataset e da WiSARD treinada com ele poder dar novos insights sobre práticas de préprocessamento que possam aumentar a resiliência de uma WiSARD treinado para outros tipos de dados.

Dito isso, ainda é relevante a relação do número de RAMs com a estabilização do modelo. Não foi possível inferir a função específica que relaciona o número de RAMs com a melhora ou piora de resiliência do modelo com o ruído injetado nos parâmetros treinados, entretanto foi obtida a regra geral de utilizar pelo menos 17 RAMs, que já apresentou melhoras em 4 dos datasets testados. Esse comportamento deve entretanto ser melhor observado para diferentes tamanhos de RAM, sendo possível que RAMs menores sejam mais afetadas pelo ruído adicionado. Seriam necessários também testes com mais datasets, com codificação da entrada que gere bitstreams maiores e com datasets com números maiores de exemplos, para analisar se a mudança de padrão limitando o ruído apenas aos 16 bits menos significativos aparece apenas com ruído em bits mais significativos do que o maior valor treinado ou se se aumentando

o número de exemplos ruído no bit de posição 17 acima ainda impactarão com mais força o desempenho do modelo.

Por último, o modelo WiSARD se mostrou mais resiliente que dois outros modelos, entretanto mais testes são necessários para verificar se outros tipos de dataset ou outros tamanhos de rede neurais apresentem maior resiliência. Entretanto erros em parâmetros com valor float se mostraram muito mais destrutivos do que em valores inteiros, sendo essa uma vantagem da WiSARD. Um teste interessante sendo a introdução de um número fixo bem baixo de *bit-flips* analisando, por exemplo, o efeito que apenas 10 mudanças teriam sobre os modelos. Também não foi testado o desempenho da WiSARD para problemas de classificar um número muito grande de classes ou com outros tamanhos de RAMs. Os resultados são considerados promissores apesar de inconclusivos, sendo interessante a continuação de testes.

Referências Bibliográficas

[1] GE, Z., SONG, Z., DING, S. X., et al. "Data Mining and Analytics in the Process Industry: The Role of Machine Learning", *IEEE Access*, v. 5, pp. 20590–20616, 2017. ISSN: 2169-3536. doi: 10.1109/ACCESS.2017.2756872.

[2] REALPE, M., VINTIMILLA, B. X., VLACIC, L. "A Fault Tolerant Perception system for autonomous vehicles". In: *2016 35th Chinese Control Conference (CCC)*, pp. 6531–6536, July 2016. doi: 10.1109/ChiCC.2016.7554385.

[3] GOMAA, W. "Cyber Physical Systems: Prospects and Challenges", *CoRR*, v. abs/1802.05233, 2018. Disponível em: <http://arxiv.org/abs/1802.05233>.

[4] ABELLA, J., CAZORLA, F. "Chapter 9 - Harsh computing in the space domain". In: Vega, A., Bose, P., Buyuktosunoglu, A. (Eds.), *Rugged Embedded Systems*, Morgan Kaufmann, pp. 267 – 293, Boston, 2017. ISBN: 978-0-12-802459-1. doi: https://doi.org/10.1016/B978-0-12-802459-1.00009-9. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780128024591000099>.

[5] BANDYOPADHYAY, D., SEN, J. "Internet of Things: Applications and Challenges in Technology and Standardization", *Wireless Personal Communications*, v. 58, n. 1, pp. 49–69, May 2011. ISSN: 1572-834X. doi: 10.1007/s11277-011-0288-5. Disponível em: <https://doi.org/10.1007/s11277-011-0288-5>.

[6] LEE, S., KIM, I., HA, S., et al. "Radiation-induced soft error rate analyses for 14 nm FinFET SRAM devices". In: *2015 IEEE International Reliability Physics Symposium*, pp. 4B.1.1–4B.1.4, April 2015. doi: 10.1109/IRPS.2015.7112728.

[7] ESPINOSA-DURAN, J. M., TRUJILLO-OLAYA, V., VELASCO-MEDINA, J., et al. "Bit-flip injection strategies for FSMs modeled in VHDL behavioral level". In: *2010 11th Latin American Test Workshop*, pp. 1–5, March 2010. doi: 10.1109/LATW.2010.5550340.

[8] ALEKSANDER, I., THOMAS, W., BOWDEN, P. "WISARD·a radical step forward in image recognition", *Sensor Review*, v. 4, pp. 120–124, 12 1984. doi: 10.1108/eb007637.

[9] BLEDSOE, W. W., BROWNING, I. "Pattern Recognition and Reading by Machine". In: *Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM '59 (Eastern), pp. 225–232, New York, NY, USA, 1959. ACM. doi: 10.1145/1460299.1460326. Disponível em: <http://doi.acm.org/10.1145/1460299.1460326>.

[10] BERGER, M., FORECHI, A., DE SOUZA, A., et al. "Traffic sign recognition with VG-RAM Weightless Neural Networks". pp. 315–319, 11 2012. ISBN: 978-1-4673-5117-1. doi: 10.1109/ISDA.2012.6416557.

[11] CARDOSO, D. O., CARVALHO, D. S., ALVES, D. S., et al. "Financial Credit Analysis via a Clustering Weightless Neural Classifier", *Neurocomput.*, v. 183, n. C, pp. 70–78, mar. 2016. ISSN: 0925-2312. doi: 10.1016/j.neucom.2015.06.105. Disponível em: <http://dx.doi.org/10.1016/j.neucom.2015.06.105>.

[12] D. SOUZA, D. F. P., FRANÇA, F. M. G., LIMA, P. M. V. "Real-Time Music Tracking Based on a Weightless Neural Network". In: *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, pp. 64–69, July 2015. doi: 10.1109/CISIS.2015.84.

[13]

[14] R. SOUZA, C., NOBRE, F., LIMA, P., et al. "Recognition of HIV-1 subtypes and antiretroviral drug resistance using weightless neural networks". 04 2012. doi: 10.13140/RG.2.1.3697.6808.

[15] CARNEIRO, H. C., FRANÇA, F. M., LIMA, P. M. "Multilingual part-of-speech tagging with weightless neural networks", *Neural Networks*, v. 66, pp. 11 – 21, 2015. ISSN: 0893-6080. doi: https://doi.org/10.1016/j.neunet.2015.02.012. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0893608015000465>.

[16] N. DO NASCIMENTO, D., CARVALHO, R., MORA-CAMINO, F., et al. "A WiSARD-based multi-term memory framework for online tracking of objects". 04 2015. doi: 10.13140/RG.2.1.3387.5687.

[17] P.A. GRIECO, B., LIMA, P., DE GREGORIO, M., et al. "Producing pattern examples from "mental" images", *Neurocomputing*, v. 73, pp. 1057–1064, 03 2010. doi: 10.1016/j.neucom.2009.11.015.

[18] GUO, L., LI, D., LAGUNA, I., et al. “FlipTracker: understanding natural error resilience in HPC applications”. In: *SC*, 2018.

[19] SUSSMAN, G. J. “Building Robust Systems an essay”. 2007.

[20] LI, G., PATTABIRAMAN, K., DEBARDELEBEN, N. “TensorFI: A Configurable Fault Injector for TensorFlow Applications”. pp. 313–320, 10 2018. doi: 10.1109/ISSREW.2018.00024.