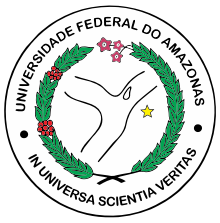

UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
ENGENHARIA DA COMPUTAÇÃO

Verificação Formal de Redes Neurais usando o ESBMC-GPU

Luiz Henrique Coelho Sena

MANAUS-AM
ANO

Luiz Henrique Coelho Sena

Verificação Formal de Redes Neurais usando o ESBMC-GPU

Monografia apresentada à Coordenação do Curso de Engenharia da Computação da Universidade Federal do Amazonas, como parte dos requisitos necessários à obtenção do título de Engenheiro de Computação.

Orientador: Iury Valente Bessa

MANAUS-AM

ANO

DEDICATÓRIA AQUI

Agradecimentos

AGRADECIMENTOS AQUI.

EPÍGRAFE AQUI

(AUTOR AQUI, TÍTULO DA OBRA.)

Resumo

RESUMO AQUI

Palavras-chave: PALAVRAS CHAVES AQUI.

Abstract

ABSTRACT AQUI

Keywords: KEYWORDS HERE.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

SIGLA NOME EXPANDIDO – do inglês ***SI Gl A***

Lista de Símbolos

Símbolos Matemáticos

$\mathbb{R}$	conjunto dos números reais

Sumário

Capítulo 1

Introdução

INTRODUÇÃO AQUI

1.1 Confiabilidade de ANN's

Devido a rapidez que as ANN's abrangeram as mais diversas áreas da ciência, pouco foi consolidado sobre a confiabilidade de tais estruturas. Em aplicações como: jogadores artificiais em jogos multiplayers [1], reconhecimento de imagens [2], processamento de linguagem natural [3], controle de veículos não tripulados [4] e tantos outros sistemas críticos que as ANN's estão presentes. A confiabilidade é um dos principais, senão o principal pré-requisitos de tais sistemas, uma vez que uma classificação errada pode por em risco a vida de pessoas.

A inadimissibilidade de erros, juntamente com a demanda por confiabilidade em sistemas de software vem crescendo cada vez mais. Porém, conforme os sitemas se tornam mais complexos a confiabilidade se torna um requisito dificil de ser alcançado. Em um sistema complexo e crítico, por exemplo, onde um erro pode até mesmo por em risco a vida de alguém, a confiabilidade e a garantia de não haver erro se tornam fundamentais em seu funcionamento. Não diferente de sistemas genéricos, as ANN's também devem ser verificadas, e uma das possíveis técnicas para tal é a verificação de modelos [5].

Por possuirem estruturas similares ao sistema nervoso humano, as ANN's operam de forma paralela. Ganhos significantes são obtidos quando utiliza-se de dispositivos extremamente paralelos, como por exemplo, as GPU's. Tensorflow, Caffe2, Cognitive Toolkit são exemplos de frameworks para o desenvolvimento de ANN's [6]. Em tais

ferramentas o grande número de nucleos das GPU's são usados no cálculo de matrizes, em [7] pode ser visto o ganho no cálculo de matrizes na GPU em relação a CPU.

A complexidade é proporcional ao nível de paralelismo de sistemas. Em sitemas críticos, que utilizam de GPU's e ANN's se tornam extremamente paralelos. No entanto, é preciso analisar o paralelismo do grande número de processos encontrados nos núcleos desses dispositivos e garantir que estes continuem operando sem falhas em relação a propriedades específicas, como é feito em [8].

Devido à alta demanda de processamento gráfico e à grande difusão das GPUs com programas de alta complexidade, a verificação formal de softwares que gerenciam os recursos das GPUs se tornou uma ferramenta que desempenha um papel importantíssimo no desenvolvimento desses sistemas. Diversos frameworks de software (isto é, conjuntos reusáveis de bibliotecas ou classes) têm sido utilizados para acelerar o desenvolvimento das aplicações dentro dos dispositivos multi-core (GPU), dentre eles os mais utilizados são CUDA e OpenCl [9]. O ESBMC-GPU faz verificação formal de softwares desenvolvidos em CUDA, porém ainda são necessárias algumas extensões, por exemplo: verificar formalmente aplicações de frameworks desenvolvidos em OpenCL, OpenGL, cuDNN, etc. Esse projeto tem como objeto estender o modelo operacional do ESBMC-GPU para que este suporte mais APIs fazendo com que a ferramenta de verificação formal de software abranja mais dos principais frameworks utilizados no desenvolvimento das ANN's.

CUDNN é um framework desenvolvido pela empresa NVIDIA, e é voltado principalmente para a otimização das ANNs que utilizam de CUDA para obter o desempenho dos grandes números de nucleos das GPUs. Desse modo se faz necessário garantir que o grande paralelismo desenvolvido sob tais frameworks tenha confiabilidade.

Os experimentos propostos neste projeto visam utilizar o ESBMC-GPU para fazer a verificação de propriedades especificas das ANN's utilizando das técnicas de Bounded Model Checking [10]. Sendo este um tema novo e visto recentemente em [11, 12], onde são aplicadas algumas técnicas de verificação e teste para avaliar o quão segura as ANN's se comportam em relação a certas propriedades.

1.1.1 Definir o Problema

1.1.2 Motivar/Justificar

1.1.3 Apresentar Hipóteses

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho tem como objetivo desenvolver um conjunto de bibliotecas simplificadas, similares aos frameworks de CUDA voltados para ANN's, e integrá-las no modelo operacional CUDA do verificador de software ESBMC para que a partir dessas implementações, o verificador consiga analisar aplicações de ANNs reais que utilizam os frameworks em questão de forma mais abrangente .

1.2.2 Objetivo Específico

(1) Utilizar de técnicas de verificação formal de modelos e lógica temporal e aplicá-las no framework CUDA;

(2) Montar uma suíte de teste a partir de um estudo analítico sobre diversos tipos de programas em CUDA;

(3) Definir uma estrutura simplificada do framework CUDA;

(4) Implementar as assinaturas das bibliotecas de CUDA definidas na estrutura simplificada;

(5) Implementar a modelagem dos métodos das bibliotecas CUDA da estrutura simplificada;

(6) Criar um ambiente de simulação para validação de testes de aplicações reais e usuais;

(7) Integrar as implementações com o ESBMC-GPU.

1.3 Organização do Trabalho

Capítulo 2

Revisão Bibliográfica

2.1 Fundamentação Teórica

2.1.1 Redes Neurais

Redes neurais são modelos computacionais inspirados no sistema nervoso humano. Suas estruturas são subdivididas em camadas e unidades de processamento. As camadas representam a ordem nas quais os neurônios processam as informações. Os neurônios são as unidades de processamento, onde similarmente ao sistema nervoso, recebem um estimulo, processam e transmitem o estimulo aos demais neurônios da rede neural. A estrutura de um neurônio pode ser vista na figura a seguir:

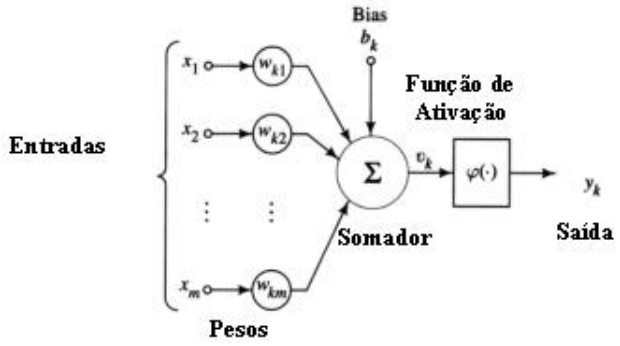

Figura 2.1: Neurônio Artificial

Como nos neurônios presentes no sistema nervoso, o neorônio artificial possui um meio de recepção dos estimulos vindos do meio externo, que são denominadas as entradas. Os pesos são atribuidos a cada entrada e estes podem sofrer alteração durante o processo

de treinamento. O somador obtem o resultado da soma das multiplicações das entradas pelos pesos e transmite o potencial de ativação à função de ativação. A função de ativação aplicará a saída do somador em alguma função especifica obtendo assim, a saída da rede neural.

As topologias e arquiteturas de uma rede neural são definidas sob algumas especificações. Tais são elas: direção de fluxo de dados, tipo de aprendizado, algoritmo de aprendizado e a realimentação, disposição espacial dos neurônios.

Arquiteturas feedfoward processam os dados em direção a saída, ou seja, a partir da camada de entrada o processamento dos dados ocorrerá de camada em camada, passando por todas as camadas escondidas até chegar na camada de saída. Um exemplo genérico de uma arquitetura feedfoward pode ser visto na figura a seguir:

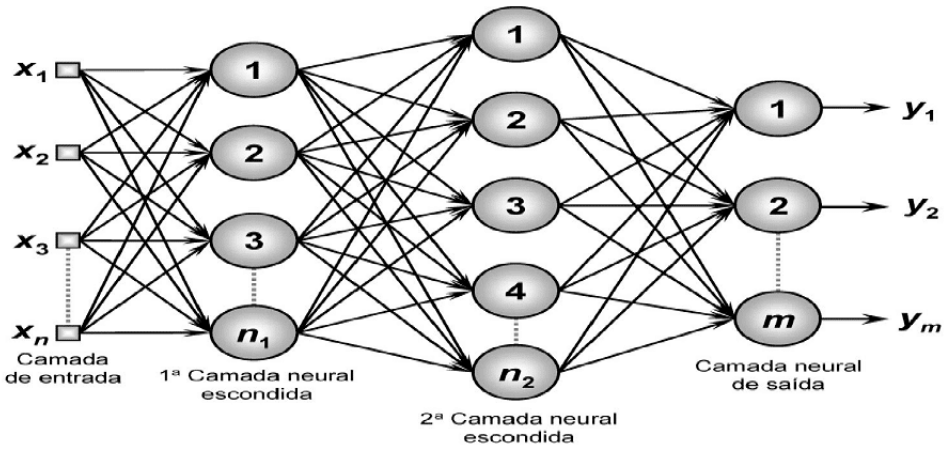

Figura 2.2: Neurônio Artificial

Em algumas arquiteturas é possível encontrar também interconexões das camadas no sentido da camada de saída em direção a camada de entrada. Tal arquitetura é denominada recorrente. Um exemplo dessa topologia pode ser observada na figura 2.3.

Um dos conceitos fundamentais no projeto de redes neurais artificiais é o treinamento. O treinamento possui a função de ajustar os pesos da rede fazendo com que a mesma seja capaz de generalizar o aprendizado. Existem dois tipos de treinamento: Supervisionado e não supervisionado. O treinamento supervisionado é feito com o auxílio de um professor, ou seja, para um determinado conjunto de entradas existe uma saída conhecida e desejada. O aprendizado no treinamento não supervisionado será feito sem "professor"reconhecendo padrões, relações e regularidades sem conhecer as saídas.

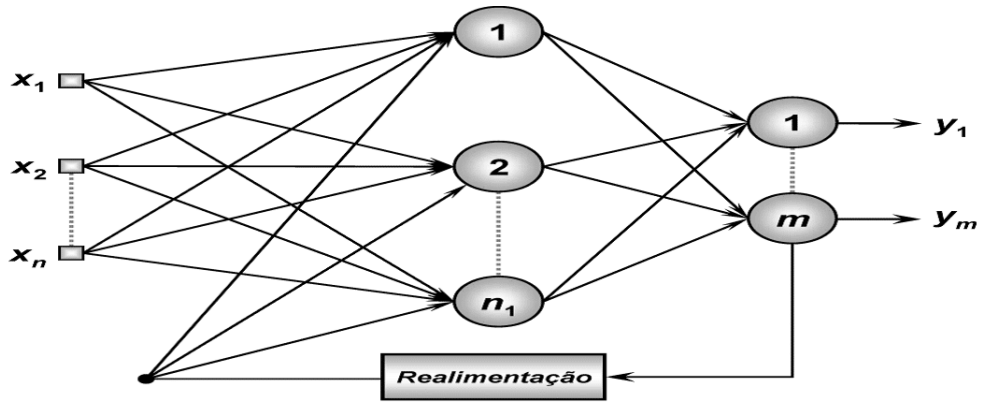

Figura 2.3: Neurônio Artificial

Dentre os diversos algoritmos, destaca-se o backpropagation pela sua capacidade de resolver problemas não-linearmente separáveis. Este algoritmo só é aplicável em arquiteturas multicamadas e o seu processo de ajustes de pesos se baseia na retro propagação do erro. A perceptron multicamada é uma das arquiteturas que utiliza deste algoritmo de treino para generalizar suas soluções não-linearmente separáveis. Um exemplo de PMC pode ser visto a seguir:

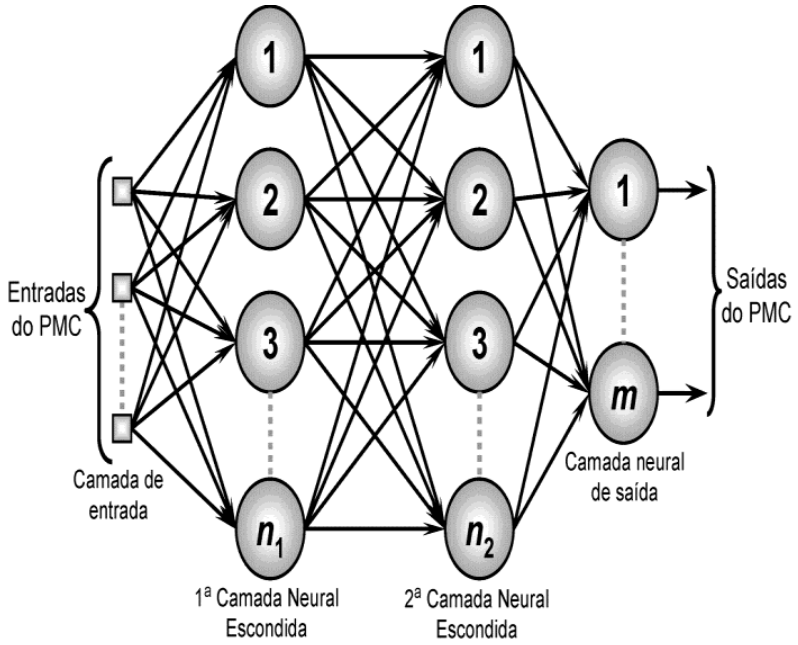

Figura 2.4: Neurônio Artificial

Overfitting e underfitting são classificações sobre a capacidade de generalização da rede. O overfitting ocorre quando a rede recebe treinamento demasiado das amostras do dataset, e no contrário, quando o treinamento é insuficiente ocorre o underfitting da

capacidade de generalização. Para validar se uma rede neural já atende as especificações ótimas de generalização, ou seja, o ponto ótimo entre o overfitting e o underfitting. Alguns métodos de validação e testes já são utilizados no processo de treinamento de uma rede neural. Um muito conhecido é o método da validação cruzada, onde o dataset é dividido em 3 subconjuntos de amostras: Treino, validação e teste. O subconjunto do treino será o conjunto que treinará os pesos dos neurônios. Sob as amostras de validação serão calculados o erro até que se alcance a condição de parada. E por fim, o subconjunto de teste será usado para estipular os parametros da capacidade de generalização da rede. Na validação cruzada os subconjuntos de treino serão divididos em k partes iguais, onde cada k parte será o conjunto de validação durante uma certa quantidade de épocas de treino.

2.1.2 Verificação

Verificação é um método utilizado para verificar a corretude de uma certa propriedade sob o funcionamento de um sistema. Tal técnica viza a validação e a confiabilidade de uma propriedade específica. Verificação formal se divide em dois tipos de abordagens: a verificação dedutiva e a verificação de modelos. No caso da primeira abordagem, a verificação permite a prova de propriedades temporais em sistemas com estados infinitos. Já na segunda abordagem, a verificação de modelos, do inglês *Model Checking*, trata do problema de testar de forma automática se um modelo de um determinado sistema atende a uma respectiva especificação, sendo que, por exemplo, tais especificações podem estar relacionadas com as propriedades de segurança e/ou de vivacidade do sistema em questão. Para solucionar este problema de forma algorítmica, é representado matematicamente o sistema e suas especificações utilizando métodos formais, como por exemplo, Lógica Proposicional e Lógica Temporal Linear, de tal forma que se possa verificar se uma dada fórmula é satisfeita dada uma determinada estrutura [5].

A expressão "verificação de modelos"se refere a um algoritmo que explora o espaço de estados de um sistema a fim de verificar propriedades. Este também verifica se as transições dos estados respeitam as restrições impostas pelo sistema [10]. Tal algoritmo realiza uma verificação exaustiva no espaço de estados de maneira automática em busca de uma violação de restrição.

Um dos principais problemas enfrentados pelas técnicas de verificação de modelos até hoje é a explosão de estados. Tal problema ocorre quando os estados da estrutura obtida

a partir do sistema em questão crescem exponencialmente [10]. Embora, muitos ganhos já foram obtidos em relação a eficiência dos verificadores, a explosão de estados ainda é um problema crítico para as ferramentas hoje existentes.

Ferramentas de software que executam a técnica de verificação automática são chamadas de verificadores. Esses se baseiam nas teorias da verificação de modelos, que possuem o objetivo de verificar de maneira rápida e eficiente propriedades de um código. Com o crescimento da verificação formal em sistemas de software , muitas ferramentas foram desenvolvidas para verificar propriedades não usuais para diversos frameworks presente no mercado. A verificação de corrida de dados em gpus é um exemplo que algumas ferramentas já verificam. Algumas destas são: ESBMC-GPU [8], Gklee [13], GPUVerify [14], etc.

O desenvolvimento desse projeto gira em torno do verificador ESBMC-GPU [8], um verificador que é capaz de verificar as propriedades: estouro aritméticos de vetores, ponteiros nulos, corrida de dados, divisão por zero e assertivas definidas pelo usuário. Este verificador utiliza das técnicas da verificação de modelos e possui algumas características particulares.

O ESBMC aplica técnicas de BMC (*Bounded Model Checking*) baseado em SMT (*Satisfiability Modulo Theories*) para verificar programas C/C++, e da linguagem CUDA, usada para programação de GPU. A técnica BMC possui a ideia básica de checar a negação de uma propriedade até uma determinada profundidade. Dado um sistema de transição de estados M, uma propriedade ϕ, e um limite k, BMC desdobra o sistema k vezes e o traduz para uma condição de verificação (*verification condition*, VC) Ψ, tal que Ψ é satisfatíel, se e somente se, ϕ possuir um contraexemplo de profundidade k ou menor [15].

SMT resolve as fórmulas de satisfatibilidade de primeira ordem usando a combinação de diferentes teorias e assim generaliza a satisfatibilidade proposicional por suportar funções não interpretadas, aritmética linear e não linear, vetores de bit, tuplas, arrays, e outras teorias de primeira ordem.

O ESBMC-GPU tem como base um algoritmo de exploração preguiçosa, que no lugar de verificar de uma única vez todas as possíveis intercalações de um programa, verifica de forma incremental uma intercalação por vez [16]. O algoritmo percorre, em profundidade, a árvore de alcançabilidade de estados (*Reachability Tree*, RT). Ao chegar em

um nó folha, o algoritmo passa para o solucionador uma fórmula SMT que representa uma intercalação específica do programa CUDA. Caso esta fórmula seja satisfatível, uma violação da propriedade no programa CUDA foi encontrada e, assim, um contraexemplo é fornecido. Caso contrário, é realizado um *backtrack* na RT e uma nova intercalação é produzida e verificada pelo solucionador SMT. O algoritmo termina quando ou uma violação é encontrada pelo ESBMC ou todas as intercalações foram verificadas com sucesso.

Além do algoritmo de exploração preguiçosa o ESBMC-GPU ainda utiliza da técnica MPOR (*Monotonic Parcial Order Reduction*). Tal técnica é utilizada para reduzir as intercalações redundantes [17]. O algoritmo classifica as transições dentro de um programa multitarefa que são independentes ou dependentes de transições em outras threads, objetivando determinar se pares de intercalações sempre computam o mesmo estado, descartando os estados da RT que são duplicados [18].

2.1.3 CUDA

CUDA é uma plataforma de computação paralela de propósito geral que representa um modelo de programação desenvolvido pela empresa NVIDIA para executar em GPUs fabricadas pela mesma. É uma linguagem desenvolvida para ter uma rápida curva de aprendizado, cujo o ambiente pode ser facilmente utilizado por programadores das linguagens C/C++ e/ou Fortran [NVIDIA 2015].

No modelo de programação de CUDA, o conceito de kernel é usado para uma função que executa n cópias paralelamente na GPU, onde n é o produto do número de blocos e threads. Um kernel é definido por um especificador __global__e sua chamada no programa é feita pela notação kernel<<<B,T>>>, onde B é o número de blocos e T é o número de threads. Cada kernel é referenciado na GPU como thread e cada thread recebe um identificador único (ID) formado pelo número da thread e o número do bloco. O ID da thread é usado para indexar as suas tarefas (i.e., posições de memória e cooperação). As threads são organizadas em blocos.

Dentro de um bloco, a hierarquia de threads é definida pela variável chamada threadIdx. Esta variável é um vetor de três componentes permitindo usar índices uni-, bi- e tridimensionais. Por exemplo, para obter o ID de uma thread nestas configuraçõe para um bloco bidimensional de tamanho (Dx,Dy), o ID da thread de índice (x,y) e obtido por (x+yDx), e para um bloco tridimensional de tamanho (Dx,Dy,Dz), a ID de uma thread

de índice (x, y, z) é definido por (x + yDx + zDxDy) [NVIDIA 2015].

Os blocos também podem ser definidos em trê dimensões, onde cada dimensão pode ser acessada pela variável blockIdx. Esta variável também é formada por três componentes permitindo usar blocos uni-, bi- e tridimensionais. O número máximo de threads por bloco depende da geração da placa variando de 1024 a 2048 [NVIDIA 2015]. Os blocos possuem a característica de serem executados em qualquer ordem, podendo ser alocados a qualquer processador. Com isso, um kernel pode ser executado por mútiplos blocos de forma igual, e o número total de threads é o número de blocos vezes o número de threads por bloco.

Um conceito utilizado na programação de CUDA, é fazer referência a GPU como device e a unidade de processamento central (CPU) como host. device é um especificador para funções que executam e são chamadas somente pela GPU, e host para funções que executam e são chamadas somente pela CPU. O fluxo comum da alocação de dados para o device é feito no host usando as funções cudaMalloc, cudaFree e cudaMemcpy. Estas são funções essenciais para um programa CUDA, pois os dados são transferidos do host para o device e vice-versa. A função cudaMalloc aloca uma quantidade de memória no device, a qual é liberada após sua utilização pela função cudaFree. A função cudaMemcpy é usada para copiar dados inicializados no host para o device. Suas interfaces são baseadas nas funções malloc, free e memcpy da linguagem C.

Devido a eficiência do paralelismo das aplicações desenvolvidas em CUDA e em outras plataformas que operam nas GPUs de forma genérica, muitos ganhos foram obtidos em paralelizar operações. Um exemplo pode ser visto no comparativo de multiplicação de matrizes esparsas entre GPUs e CPUs em [19].

APIs voltadas apenas para cálculo de matrizes e primitivas de ANNs surgiram nos últimos anos. CUblas e CUdnn são exemplo dessas APIs e foram desenvolvidas em CUDA pela empresa NVIDIA. Tais APIs possuem funções que executam diversas operações de forma paralela nas threads dos dispositivos *many-cores*.

As ANNs são ,na maioria das vezes, sistemas que podem ser modelados por matrizes, e suas operações são baseadas majoritariamente em produtos e somas. Além dessas operações básicas, as ANNs também possuem funções de ativações especificas, podendo algumas até possuir camadas de convolução e *pooling* que são muito utilizadas em reconhecimento de imagens [20]. CUDNN e CUBLAS, trazem justamente esse *gap*, onde os

produtos e somas das matrizes, as primitivas funções das ANNs e operações de convolução e *pooling* são executadas utilizando do paralelismo das GPUs [21].

2.2 Trabalhos Relacionados

Por ser um tema relativamente atual, a verificação de ANN já é uma área de pesquisa muito visada. As técnicas utilizadas neste projeto se baseiam nos seguintes trabalhos relacionados:

Em [22], é proposto um verificador de ANNs voltadas para a função de ativação Relu. Este utiliza do método simplex [23] para poluir as amostras a serem verificadas. As contribuições são: (i) Reluplex, um SMT solver para a teoria da linear real arithmetic com ReLU constraints. (ii) Mostrar como DNNs e as propriedades de interesse podem ser codificadas como entradas para o Reluplex. (iii) discutir vários detalhes de implementações que são cruciais para performance e escalabilidade, como a aritmetica de ponto flutuante, limites de derivação para as variáveis ReLU e análise de conflitos. (iv) conduzir uma completa avaliação na implementação do protótipo ACAS XU system, demonstrando a habilidade escalável do Reluplex em DNNs que são maiores em ordem de magnitude que as outras que podem ser analizadas usando técnicas existentes.

Em [11] é proposto um *framework* de verificação para *feed-forward multi-layer neural networks* baseadas em SMT. Este foca na segunraça das decisões nos reconhecimentos de imagens com respeito a manipulação de imagens, como por exemplo: arranhões, mudanças no ângulo da câmera e condições de luminosidade que resultariam no mesmo resultado classificado por um humano. A técnica utilizada se baseia numa busca exaustiva de uma região empregando discretização e propagando a analise camada por camada. O método lida diretamente com o código da ANN e pode garantir que os exemplos adversos, se existem, são encontrados pela dada região. As técnicas foram implementadas utilizando o SMT solver Z3[24].

Em [12] é proposto uma metodologia de como testar a segurança de ANNs em dominios críticos. A metodologia se baseia no trdicional MC/DC *coverage criterion* [25]. Os criterios de avaliação são incomparáveis e interdependentes. É proposto um algoritmo para geração de casos de teste baseado em LP (*linear programming*, onde um novo caso de teste é obtido a partir de pertubações inseridas em outros casos de teste. Os métodos

visam 4 objetivos: (1) busca de bugs; (2) métrica de segurança para DNNs; (3) eficiência do teste; (4) analise das estruturas internas de DNNs.

Capítulo 3

Verificação de Modelos Aplicado a Redes Neurais Implementadas em CUDA

MODELAGEM AQUI

Capítulo 4

ESBMC-GPU e Modelos Operacionais

METODOLOGIA AQUI

Capítulo 5

Aplicação Experimental

5.1 Objetivos Experimentais

5.2 Descrição dos Experimentos

5.3 Ambiente Experimental

5.4 Resultados

RESULTADOS AQUI

Capítulo 6

Conclusão

6.1 Conseiderações Finais

6.2 Trabalhos Futuros

CONCLUSÃO AQUI

Referências Bibliográficas

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354 EP –, Oct 2017, article. [Online]. Available: http://dx.doi.org/10.1038/nature24270

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[3] Y. Goldberg, "A primer on neural network models for natural language processing," *CoRR*, vol. abs/1510.00726, 2015. [Online]. Available: http://arxiv.org/abs/1510.00726

[4] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[5] R. Jhala and R. Majumdar, "Software model checking," *ACM Comput. Surv.*, vol. 41, no. 4, pp. 21:1–21:54, Oct. 2009. [Online]. Available: http://doi.acm.org/10.1145/1592434.1592438

[6] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of caffe, neon, theano, and torch for deep learning," *CoRR*, vol. abs/1511.06435, 2015. [Online]. Available: http://arxiv.org/abs/1511.06435

[7] S. Ohshima, K. Kise, T. Katagiri, and T. Yuba, "Parallel processing of matrix multiplication in a cpu and gpu heterogeneous environment," pp. 305–318, 2007.

[8] F. R. Monteiro, E. H. da S. Alves, I. S. Silva, H. I. Ismail, L. C. Cordeiro, and E. B. de Lima Filho, "Esbmc-gpu a context-bounded model checking tool to verify cuda programs," *Science of Computer Programming*, vol. 152, pp. 63 – 69, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642317301934

[9] K. Karimi, N. G. Dickson, and F. Hamze, "A performance comparison of CUDA and opencl," *CoRR*, vol. abs/1005.2581, 2010. [Online]. Available: http://arxiv.org/abs/1005.2581

[10] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods in System Design*, vol. 19, no. 1, pp. 7–34, Jul 2001. [Online]. Available: https://doi.org/10.1023/A:1011276507260

[11] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," *CoRR*, vol. abs/1610.06940, 2016. [Online]. Available: http://arxiv.org/abs/1610.06940

[12] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *CoRR*, vol. abs/1803.04792, 2018. [Online]. Available: http://arxiv.org/abs/1803.04792

[13] G. Li, P. Li, G. Sawaya, G. Gopalakrishnan, I. Ghosh, and S. P. Rajan, "Gklee: Concolic verification and test generation for gpus," *SIGPLAN Not.*, vol. 47, no. 8, pp. 215–224, Feb. 2012. [Online]. Available: http://doi.acm.org/10.1145/2370036.2145844

[14] A. Betts, N. Chong, A. Donaldson, S. Qadeer, and P. Thomson, "Gpuverify: A verifier for gpu kernels," *SIGPLAN Not.*, vol. 47, no. 10, pp. 113–132, Oct. 2012. [Online]. Available: http://doi.acm.org/10.1145/2398857.2384625

[15] L. Cordeiro, B. Fischer, and J. Marques-Silva, "Smt-based bounded model checking for embedded ansi-c software," pp. 137–148, Nov 2009.

[16] L. Cordeiro and B. Fischer, "Verifying multi-threaded software using smt-based context-bounded model checking," pp. 331–340, May 2011.

[17] V. Kahlon, C. Wang, and A. Gupta, "Monotonic partial order reduction: An optimal symbolic partial order reduction technique," pp. 398–413, 2009. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02658-4_31

[18] J. Morse, "Expressive and efficient bounded model checking of concurrent software," April 2015. [Online]. Available: https://eprints.soton.ac.uk/379284/

[19] Y. Liang, W. T. Tang, R. Zhao, M. Lu, H. P. Huynh, and R. S. M. Goh, "Scale-free sparse matrix-vector multiplication on many-core architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 12, pp. 2106–2119, Dec 2017.

[20] J. H. Kim, H. G. Hong, and K. R. Park, "Convolutional neural network-based human detection in nighttime images using visible light camera sensors," *Sensors (Basel)*, vol. 17, no. 5, p. 1065, May 2017, sensors-17-01065[PII]. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5469670/

[21] L. Wang, W. Wu, J. Xiao, and Y. Yi, "Large scale artificial neural network training using multi-gpus," *CoRR*, vol. abs/1511.04348, 2015. [Online]. Available: http://arxiv.org/abs/1511.04348

[22] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," 02 2017.

[23] R. H. Bartels, "A stabilization of the simplex method," *Numerische Mathematik*, vol. 16, no. 5, pp. 414–434, 1971. [Online]. Available: https://doi.org/10.1007/BF02169151

[24] L. de Moura and N. Bjørner, "Z3: An efficient smt solver," pp. 337–340, 2008.

[25] J. Chang and C. Huang, "A study of enhanced mc/dc coverage criterion for software testing," vol. 1, pp. 457–464, July 2007.