UNIVERSITATEA DE VEST DIN TIMIŞOARA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
PROGRAMUL DE STUDII DE MASTER: Inginerie Software

Verification of fully connected binarized neural networks

COORDONATOR:
Conf. Dr. Eraşcu Mădălina

MEMBRI:
Dienes Oliviu
Amza Rares
Militaru Oana

TIMIŞOARA
2024

Cuprins

Capitolul 1

Abstract

1.1 Context

Proiectul nostru se axează pe domeniul verificării formale, un aspect esențial în dezvoltarea și îmbunătățirea sistemelor critice, precum cele din aviația cu asistență la distanță. În particular, ne concentrăm asupra datasetului ACAS XU și a modelului său de rețea neuronală, care joacă un rol crucial în sistemele de evitare a coliziunilor integrate și de detectare și evitare în cadrul dronelor și altor vehicule aeriene autonome. Această lucrare se încadrează în eforturile de a asigura siguranța și performanța acestor sisteme, având implicații semnificative în domeniul aviației și al tehnologiei autonome.

1.2 Scop

Scopul principal al acestui proiect constă în evaluarea și verificarea formală a modelului ACAS XU, utilizând instrumentele de verificare $\alpha\beta$-CROWN si Marabou. În acest sens, ne propunem să atingem următoarele obiective:

- Analiza arhitecturii rețelei neuronale complet conectate a modelului ACAS XU și a funcției de activare ReLU.
- Evaluarea performanțelor modelului prin examinarea tipului de rețea, numărului de parametri, dimensiunii de intrare și rarității acestuia.
- Instalarea și configurarea instrumentului de verificare $\alpha\beta$-CROWN pe un sistem Linux, precum și rularea acestuia pe benchmarkul ACAS XU.
- Raportarea rezultatelor verificării

Prin îndeplinirea acestor obiective, urmărim să obținem o înțelegere mai profundă a caracteristicilor și performanțelor modelului ACAS XU, precum și să evaluăm eficacitatea instrumentului de verificare utilizat în contextul specific al rețelelor neuronale asociate aviației cu asistență la distanță.

Capitolul 2

Introducere

Rețelele neuronale au devenit fundamentale în tehnologie, cu aplicații în aviație și în alte domenii critice. În acest context, asigurarea corectitudinii și fiabilității acestor rețele devine critică, mai ales în situații critice precum aviația. Proiectul nostru se concentrează pe verificarea formală a modelului ACAS XU, care este critic pentru sistemele de evitare a coliziunilor pentru drone și aeronave autonome. Scopul nostru este să evaluăm și să validăm performanța acestui model folosind instrumentul $\alpha\beta$-CROWN, sperând să oferim informații despre siguranța și eficacitatea acestor sisteme.

Capitolul 3

Descriere Dataset

3.1 Performanța ACAS XU

Datasetul ACAS XU este conceput pentru a evalua performanța sistemului de evitare a coliziunilor integrate (ACAS Xu) și de detectare și evitare (DAA) în cazul sistemelor de aviație cu asistență la distanță (UAS). [1]

3.1.1 Tipul de rețea

Modelul ACAS XU utilizează o arhitectură de rețea complet conectată (Fully Connected - FC), unde fiecare neuron dintr-un strat este conectat la fiecare neuron din stratul următor. Funcția de activare utilizată în model este Rectified Linear Unit (ReLU), o alegere comună în rețelele neuronale pentru adăugarea non-liniarității și capturarea detaliilor intricate ale datelor. De asemenea, aceasta este si binarizat.[2]

3.1.2 Rețea complet conectată (Fully Connected - FC)

O rețea complet conectată este un tip de arhitectură neuronală în care fiecare neuron dintr-un strat este conectat la fiecare neuron din stratul următor. În cazul modelului ACAS XU, aceasta înseamnă că toți cei 5 neuroni de intrare sunt conectați la toți cei 13.000 de neuroni din stratul următor. Acest tip de arhitectură este utilizat într-o varietate de aplicații și permite modelului să învețe relații complexe între datele de intrare. [3] [4] [5]

3.1.3 Funcția de activare ReLU (Rectified Linear Unit)

ReLU este o funcție de activare comună folosită în rețelele neuronale. Ea returnează 0 pentru toate valorile negative și returnează valoarea de intrare pentru valorile pozitive. Această funcție de activare adaugă non-linearitate modelului, ceea ce îi permite să învețe și să captureze relații complexe în date. [2] [6]

3.1.4 Numărul de parametrii

Modelul ACAS XU are 13.000 de parametrii. Acești parametrii reprezintă ponderile și bias-urile asociate cu conexiunile dintre neuroni și permit modelului să învețe de la datele de antrenare. [4]

3.1.5 Dimensiunea de intrare

Dimensiunea de intrare a modelului ACAS XU este 5, ceea ce înseamnă că modelul primește un vector de 5 elemente ca intrare pentru fiecare exemplu de date. [7]

3.1.6 Sparsity

Raritatea modelului ACAS XU variază între 0% și 20%. Acest lucru poate să se refere la proporția de conexiuni care au ponderi nenule față de totalul conexiunilor posibile. Algoritmul folosește o rețea neuronală profundă (DNN) pentru comprimarea unei tabele numerice mari care conține scoruri asociate cu diferite manevre din milioane de stări discrete. Această rețea neurală profundă are 128 straturi, dintre care 5-128-64-32-16-5 și 5-256-128-64-32-16-5. Funcțiile activate folosite sunt funcții sigure standard (ReLU) sau funcții sigure saturate (Leaky ReLU). Numărul total de parametri ale rețelei neurali profundă este estimat la aproximativ 1.6 milioane. [8] [9]

3.1.7 Numărul de regiuni

Modelul ACAS XU are între 1 și 4 regiuni.Datasetul conține 4 regiuni: regiunea 1(zonă sigură), regiunea 2 (zonă alertă), regiunea 3 (zonă periculoasă) și regiunea 4 (zonă non-periculoasă) [10] Datasetul ACAS XU conține date de testare pentru sistemul ACAS Xu, incluzând imagini capturate de senzori și camere video, precum și date de control al traficului aerian (ATC). [7].Acesta este utilizat pentru evaluarea robusteții rețelelor neuronale în contextul evitării coliziunilor și DAA.

3.1.8 Tipuri de loop

Closed Loop (Buclă Închisă)

Într-un sistem ACAS XU cu buclă închisă, există un mecanism de feedback care monitorizează și reacționează la condițiile de mediu în timp real. Sistemul colectează informații despre aeronavele din apropiere și ajustează planul de evitare în consecință pentru a preveni coliziunile. Prin intermediul feedback-ului, ACAS XU poate face ajustări în timp real pentru a se adapta la schimbările din mediu și pentru a lua decizii mai precise în ceea ce privește evitarea coliziunilor.

Open Loop (Buclă Deschisă)

Într-un sistem ACAS XU cu buclă deschisă, deciziile de evitare a coliziunilor sunt luate fără a lua în considerare feedback-ul din mediu sau starea actuală a altor aeronave. Acest tip de sistem poate funcționa pe baza unor setări predefinite sau a unor algoritmi care nu sunt ajustați în timp real pe baza feedback-ului din mediu. Bucla deschisă poate fi mai puțin adaptabilă la schimbările neprevăzute în mediul înconjurător. Un sistem cu buclă închisă are potențialul de a oferi răspunsuri mai precise și mai adaptabile la condițiile în schimbare rapidă. [3]

Capitolul 4

Instrumente

4.1 Instalare instrumente

4.1.1 Alpha-Beta-CROWN

α,β-CROWN (alpha-beta-crown) este un verificator de rețele neuronale care a câștigat competiția VNN-COMP 2021, 2022 și 2023 având cel mai mare scor total și astfel depășind în performanță multe alte verificatoare de rețele neuronale pe o gamă largă de evaluări în decursul a 2 ani. [11].

Acesta este un instrument care funcționează doar în sistemul de operare Linux.

Înainte de a începe configurarea instrumentului trebuie să verificăm dacă avem instalate următoarele:

- Git - Puteți instala Git de la adresa: `https://git-scm.com`
- Miniconda - Puteți instala Miniconda de la adresa: `https://docs.conda.io/projects/miniconda/en/latest/`

După configurarea mediului de lucru,vom accesa pagina proiectului la adresa: `https://github.com/Verified-Intelligence/alpha-beta-CROWN` și urmăm pașii indicați.

.

Deoarece am lucrat pe o mașină cu sistemul Windows, toate comenzile ulterioare au fost efectuate cu ajutorul WSL și Ubuntu, și am urmat următorii pași pentru a configura proiectul:

1. Am clonat proiectul pe dispozitivul nostru cu comanda:

```
git clone --recursive https://github.com
/Verified-Intelligence/alpha-beta-CROWN.git
```

2. Am creat mediul de lucru din Conda:

```
conda env create -f /complete_verifier
/environment_pyt111.yaml --name alpha-beta-crown
```

.

```
# conda environments:
#
base                  /home/plyber/miniconda3
alpha-beta-crown      *  /home/plyber/miniconda3/envs/alpha-beta-crown
```

Figura 4.1: Verificare mediu de lucru Conda

3. După ce au fost instalate toate modulele necesare vom verifica că mediul de lucru a fost creat.

4. Pentru a putea folosi verificatorul trebuie să configurăm și submodulul auto_LiRPA:

```
cd auto_LiRPA
python setup.py install
```

4.1.2 Marabou

Marabou este un instrument puternic pentru verificarea rețelelor neuronale. Pentru a crea un mediu de executie propice rularii uneltei Marabou unde am putut instala si configura toate pachetele necesare, am folosit Docker.

1. Am utilizat distributia ubuntu cu versiunea 20.04. Folosind repository-ul "deadsnakes" unde se afla versiunile depreciate de python, am instalat python3.8, mediul virtual aferent si pachetul de development.

```
FROM ubuntu:20.04

RUN add-apt-repository ppa:deadsnakes/ppa && \
    apt-get update && \
    apt-get install -y python3.8 python3.8-venv python3.8-dev
```

2. Am instalat pip pentru python3 si numpy.

```
RUN apt-get install -y python3-pip && \
    python3 -m pip install --upgrade pip && \
    pip3 install numpy
```

3. Am instalat mai multe pachete de care Marabou depinde, dupa care am curatat cacheul si spatiul suplimentar utilizat de aceste comenzi.

```
RUN apt-get install -y --no-install-recommends \
    sudo \
    curl \
    wget \
    git \
    build-essential \
    bc \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*
```

4. Am setat terminatiile de linie globale (EOL) pentru git ca LF si nu CRLF, dupa care am clonat proiectul vnncomp2023_benchmarks [12] astfel incat acesta sa contina doar setul de date "acasxu" si scripturile necesare. Am urmarit ca acest proiect sa fie director "frate" uneltei Marabou pe care o vom clona in pasul urmator. Dupa clonarea proiectului am despachetat toate fisierele de tip .gz. Acestea sunt fisierele ONNX si VNNLIB.

```
RUN git config --global core.eol lf && \
    git config --global core.autocrlf false
RUN mkdir "vnncomp2023"
WORKDIR /usr/src/app/vnncomp2023
RUN git init && \
    git remote add origin https://github.com/ChristopherBrix/vnncomp2023_bench
    git config core.sparseCheckout true && \
    echo 'benchmarks/acasxu/*' > .git/info/sparse-checkout && \
    echo 'run_all_categories.sh' >> .git/info/sparse-checkout && \
    echo 'run_single_instance.sh' >> .git/info/sparse-checkout && \
    git pull origin main
    git pull origin main
RUN find . -type f -name "*.gz" -exec echo {} \; -exec gunzip {} \;
```

5. Am clonat submodulul modificat de Marabou din proiectul VF-Final si am eliminat directoarele suplimentare create.

```
WORKDIR /usr/src/app
RUN git clone --no-checkout --config core.fileMode=true https://github.com/ply
    cd VF-Final && \
    git config core.sparseCheckout true && \
    echo "Marabou/*" > .git/info/sparse-checkout && \
    git checkout main && \
    mv Marabou ../ && \
    cd .. && \
    rm -rf VF-Final
```

6. Am generat fisierul instances.csv folosind scriptul de python disponibil in acasxu. Pe urma am oferit permisiuni de executie scripturilor de rulare si de instalare.

```
WORKDIR /usr/src/app/vnncomp2023/benchmarks/acasxu
RUN python3 generate.py
RUN chmod +x /usr/src/app/vnncomp2023/run_all_categories.sh && \
    chmod +x /usr/src/app/vnncomp2023/run_single_instance.sh
    chmod +x /usr/src/app/Marabou/vnncomp/install_tool.sh
```

7. Am instalat Marabou utilizand scriptul install_tool.sh

```
RUN ./install_tool.sh
```

.

4.2 Rulare Instrumente pentru Benchmark

4.2.1 Alpha-Beta-CROWN

Fiind cu două directoare deasupra complete_verifier, am clonat repository-ul vnncomp2023_benchmarks[12] astfel încât calea specificată în fișierul de tip yaml să coincidă cu calea necesară pentru funcționarea instrumentului α,β-CROWN:

```
../../vnncomp2023_benchmarks/benchmarks/acasxu
```

Ne-am asigurat că am activat mediul de lucru ab-crown cu Miniconda3 folosind următoarea instrucțiune:

```
conda activate alpha-beta-crown
```

Apoi am rulat abcrown.py pe benchmarkul acasxu.yaml:

```
cd complete_verifier
python abcrown.py --config exp_configs/acasxu.yaml
```

Execuția instrumentului s-a finalizat cu succes și am primit următoarele rezultate:

```
############# Summary #############
Final verified acc: 74.19354838709677% (total 186 examples)
Problem instances count: 186 ,
total verified (safe/unsat): 138 ,
total falsified (unsafe/sat): 47 ,
timeout: 1
mean time for ALL instances (total 186):3.258072501031987,
max time: 118.92911696434021
```

Astfel avem o acuratețe finală de 74,19% pentru 186 de instanțe: 138 instanțe fiind SAT și 47 UNSAT. De asemenea, una dintre instanțe a depășit timpul alocat, astfel că am avut și un timeout. O demonstratie video a procesului se poate gasi aici: Build and Run of Dockerized Alpha-Beta-CROWN Verifier on Acasxu Benchmark.

4.2.2 Marabou

Deoarece am indeplinit toate necesitatile uneltei Marabou in sectiunea de instalare Marabou, a mai ramas doar pasul in care execut scriptul de rulare oferit de competitie. Fiind in directorul `usr/src/app/vnncomp2023_benchmarks`, am oferit scriptului urmatoarele argumente:

```
'v1' (version string), tool_scripts_folder, vnncomp_folder,
result_csv_file, counterexamples_folder, categories,
all|different|first
```

Astfel incat:

```
./run_all_categories.sh v1 /usr/src/app/Marabou/vnncomp
/usr/src/app/vnncomp2023 out.csv counterexamples "acasxu" all
```

Executia instrumentului s-a finalizat cu succes si am primit urmatoarele rezultate:

```
############# Summary #############
Final verified acc: 74.19354838709677% (total 186 examples)
Problem instances count: 186 ,
total verified (safe/unsat): 102 ,
total falsified (unsafe/sat): 75 ,
timeout: 8
mean time for ALL instances (total 186):4.090322942,
max time: 115.9957511
```

O demonstratie video a procesului se poate gasi aici: Build and Run of Dockerized Marabou Verifier on Acasxu Benchmark.

4.2.3 Analiza procesului de verificare

Logger-ul uneltei Alpha-Beta-CROWN afiseaza in terminal procesul atacurilor asupra retelei. Putem analiza acest proces, care incepe cu:

Selectarea modelului si a proprietatilor: Procesul incepe prin selectarea unui model ONNX si un fisier de proprietati VNNLIB. Aceste fisiere definesc reteaua neuronala si proprietatile care vor fi verificate.

Fisierul VNNLIB precompilat: Sistemul localizează o versiune precompilată a fișierului cu proprietati VNNLIB. Asta accelerează procesul de verificare fara a mai fi nevoie de pasul initial de compilare.

Incarcarea modelului ONNX: Se incarca modelul ONNX specificat, pregatindu-l pentru procesul de verificare.

Configurarea parametrilor de atac: Parametrii pentru un atac adversar (PGD - Projected Gradient Descent) sunt setati. Acest atac este folosit pentru a gasi exemple adverse care ar putea falsifica rezultatele retelei.

Verificarea iesirilor modelului: sunt afisate predictiile initiale ale primelor catorva exemple din model. Acest pas este cel mai probabil folosit pentru debugging sau validare.

Iterații în procesul de verificare: Fiecare iterație reprezintă un pas în procesul de verificare, utilizând tehnica BaB (branch and bound) in combinatie cu atacurile adverse pentru a verifica siguranța rețelei.

Iterații:

1. Algoritmul procesează un lot de exemple (dimensiunea 1000).
2. Acesta calculează cea mai nefavorabila limită pentru aceste exemple, indicând predicția cea mai puțin sigură.
3. Algoritmul actualizează apoi lista de domenii pentru a fi explorate în următoarele iterații pe baza limitelor calculate.
4. Se înregistrează timpul cumulat și numărul de domenii vizitate.

```
Iteration 1
Batch size: 1000
Worst bound: tensor([-42.78705597], device='cuda:0')
Total time: 0.0549  pickout: 0.0004 decision: 0.0098
bounding: 0.0424 add_domain: 0.0023
Length of domains: 32
32 branch and bound domains visited
Current (lb-rhs): -42.78705596923828
Cumulative time: 0.5318393707275391
```

Concluzia procesului de verificare:

Procesul de verificare se încheie atunci când nu mai există domenii de explorat sau când algoritmul ajunge la un rezultat concludent. Se raportează limita inferioară finală, care reprezintă marja de siguranță a rețelei. Se înregistrează timpul cumulativ pentru întregul proces de verificare. Rezultatul final al procesului de verificare este raportat (în aceste cazuri ca fiind "sigur"), indicând ca predicțiile rețelei îndeplinesc criteriile de siguranță definite în fișierul VNNLIB.

4.3 Provocări

4.3.1 Resurse

Alpha-beta-CROWN si Marabou au prezentat provocari unice mai ales in ceea ce priveste cerintele de resurse si compatibilitatea librariilor. Alpha-beta-CROWN, de exemplu, este optimizat pentru utilizarea pe GPU, in timp ce Marabou este mai dependent de procesarea pe CPU (excluzand simplificarea fisierelor ONNX). Aceasta diferenta in utilizarea resurselor hardware poate conduce la probleme practice, cum ar fi gestionarea resurselor si evitarea timeout-urilor, in special in scenarii unde resursele sunt limitate sau nu sunt compatibile cu necesitatile uneltelor.

4.3.2 Dependinte

Un aspect distinctiv intampinat in utilizarea Marabou a fost necesitatea unei licente Gurobi pentru functionalitatea completa. Asta se datoreaza faptului ca unealta trebuie pornită cu ajutorul scripturilor de rulare oferite de organizatorii VNN-COMP. Aceasta reprezinta o bariera, fie din motive de cost, fie datorita restrictiilor de acces. O solutie practica a fost adaptarea codului sursa pentru a elimina dependintele care necesita licenta mentionata. Această soluție nu a avut efect asupra rezultatelor finale.

4.3.3 Mediul de lucru

Dockerizarea ambelor unelte a oferit o solutie la multe dintre aceste provocari. Prin crearea unor containere Docker, s-a putut asigura un mediu controlat cu toate dependintele necesare preinstalate si configurate. Aceasta a simplificat intr-un final procesul de instalare si configurare, permitand concentrarea pe activitatea de verificare in sine, fara a fi necesara gestionarea individuala a fiecarei dependente sau compatibilitate intre librarii.

Capitolul 5

Rezultate experimentale

- Proiectul nostru este disponibil pe GitHub la următoarea adresă: `https://github.com/raresamza/VF-Project`

Tabela 5.1: Benchmark 2023-acasxu

#	Tool	Verified	Falsified	Fastest	Penalty	Score	Percent
1	Marabou	188	76	0	1	-	60,6%
2	α-β-CROWN	186	47	-	1	-	74.18%

Tabela 5.2: Overview of all scored benchmarks

Category	**Benchmark**	**Application**	**Network Types**	**Params**	**Effective Input Dim**
(FC+ ReLU)	Acas XU	Collision Detection	FC.+ ReLU	13k	5

5.1 Concluzii

În concluzie, această lucrare a analizat verificarea formală a modelului ACAS XU, focalizându-se pe rețelele neuronale în contextul aviației cu asistență la distanță. Principalele puncte de interes:

- Evaluarea performanței modelului ACAS XU folosind instrumentul $\alpha\beta$-CROWN, evidențiind aspecte legate de corectitudine și eficacitate.
- Analiza detaliată a arhitecturii rețelei neuronale și a funcției de activare ReLU pentru a înțelege mai bine caracteristicile modelului.
- Configurarea și utilizarea eficientă a instrumentelor de verificare într-un mediu Linux, cu aplicabilitate specifică asupra benchmarkului ACAS XU.

- Rezultatele obținute au evidențiat aspecte semnificative în ceea ce privește siguranța și performanța modelului, precum și eficiența instrumentelor de verificare utilizate.

Bibliografie

[1] Diego Manzanas López, "Acas xu github repository," https://github.com/mldiego/AcasXu, 2023.

[2] DeepAI, "Rectified linear unit (relu) - deepai," 2023, accesat la data de 21 decembrie 2023. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/relu

[3] S. Bak, "Neural network compression of acas xu early prototype is unsafe," 2022, accesat la data de 22 decembrie 2023. [Online]. Available: https://arxiv.org/pdf/2201.06626v3.pdf

[4] M. P. Owen, "Aiaa 38th digital avionics systems conference (dasc," 2019, accesat la data de 19 decembrie 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9081758

[5] G. Kutz, "Reluplex," 2017, accesat la data de 21 decembrie 2023. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-63387-9_5

[6] A. Vaswani and Shazeer, "Attention is all you need," 2018, accesat la data de 21 decembrie 2023. [Online]. Available: https://arxiv.org/abs/1803.08375

[7] Jason T. Davies, "Comparative analysis of acas-xu and daidalus detect-and-avoid systems," https://ntrs.nasa.gov/api/citations/20180001564/downloads/20180001564.pdf, 2018, accesat la data de 19 decembrie 2023.

[8] M. Jason T. Davies, "Nasa document," https://ntrs.nasa.gov/api/citations/20180001564/downloads/20180001564.pdf, 2018, accesat la data de 22 decembrie 2023.

[9] K. D. Julian, "Deep neural network compression for aircraft collision avoidance systems," 2018, accesat la data de 22 decembrie 2023. [Online]. Available: https://arxiv.org/pdf/1810.04240.pdf

[10] NASA, "A human-in-the-loop evaluation of acas xu," https://ntrs.nasa.gov/citations/20205007317, 2020, accesat la data de 20 decembrie 2023.

[11] H. Zhang, K. Xu, Z. Shi, S. Wang, L. Li, J. Chen, Z. Yang, and Y. Wang, "alpha-beta-crown," https://github.com/Verified-Intelligence/alpha-beta-CROWN/tree/main, 2021-2022.

[12] ChristopherBrix, "vnncomp2023_benchmarks," https://github.com/ChristopherBrix/vnncomp2023_benchmarks, 2023.