NeuroCodeBench: a plain C neural network benchmark for software verification

Edoardo Manino, Rafael Sá Menezes, Fedor Shmarov,
Lucas C. Cordeiro

AFRiTS Workshop at SBMF 2023

5 December 2023

EnnCore

MANCHESTER 1824
The University of Manchester

The abstraction ladder (1)

The "classic ML" mindset

- Define a neural net as $f : \mathbb{R}^n \to \mathbb{R}^m$
- Gradient descent, auto differentiation
- Data manifold, regularizers, ...

What's the implicit assumption?

- We live in a mathematician's world
- At a very high level of abstraction
- And operations have infinite precision

Very effective, most of the time

The abstraction ladder (2)

Quantisation efforts

- 16-bit floating point
- 16-bit, 8-bit, 4-bit integers
- Binarized neural networks

Parallel execution

- GPU, SIMD instructions, TPU, FPGAs
- Distributed/federated learning

What's the implicit assumption?

- We make a lot of optimisations
- But the result doesn't really change

The abstraction ladder (3)

The software safety mindset

- Buffer overflow, division by zero
- Data race, deadlock, use-after-free
- Infinite loops, side effects

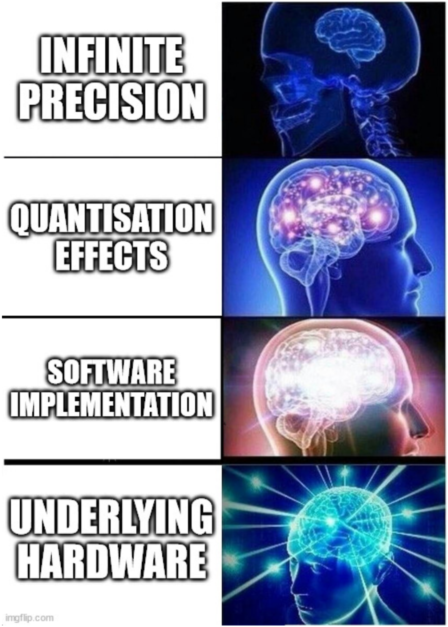
What's the implicit assumption

- Every innocent bug
- Can introduce a vulnerability

Just a problem for library makers?

The abstraction ladder (4)

Don't forget the hardware!


INFINITE PRECISION
QUANTISATION EFFECTS
SOFTWARE IMPLEMENTATION
UNDERLYING HARDWARE
imgflip.com

Implementation effects (1)

Can we expect consistent behaviour across devices?

- Cidon et al., *Characterizing and taming model instability across edge devices*, 2021
- Wang et al, *SysNoise: exploring and benchmarking trainin-deployment system inconsistency*, 2023

Many low-level sources of noise!

- Pre-processing: .jpg→tensor (iDCT, interpolation, colour)
- Model inference: convolutions, upsampling, floats, quantize
- Post-processing: tensor→bounding box (rounding coordinates)

Up to 6% accuracy fluctuation[1]

[1] Cidon [2021] runs MobileNetV2 on photos taken from five different phones.

Implementation effects (2)

Can we trust NN verifiers?

- VNN-COMP compares the best neural network verifiers
- Let's reproduce one of their results!

Benchmark: reach_prob_density/robot_11

- A ReLU network with architecture $5 \times 64 \times 64 \times 64 \times 5$
- Input assumption: $x_0 \in [-1.8, -1.2] \wedge x_1 \in [-1.8, -1.2]\ldots$
- Output assertion: $y_0 \geq 0.27 \wedge y_1 \in [-0.17, 0.17]\ldots$

Five tools return a counterexample!

- $\alpha\beta$-CROWN, Marabou, `nnenum`, VeriNet, Peregrinn

But none of them violates the output assertion[2]

[2] With the plain C code from `onnx2c` and the MinGW-w64 compiler.


YOU PROVED THAT A ML MODEL IS SAFE
YOUR GUARANTEE IS IMPLEMENTATION DEPENDENT
imgflip.com

Software verification

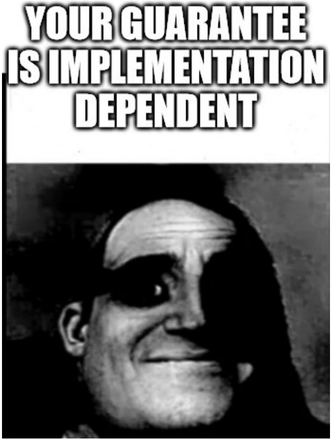
Advantages

- Code is a formal specification
- (just add safety properties)
- Decades of SV research
- Very sophisticated tools

Disadvantages

- Too many implementation details
- Memory model, parallelism
- Hard to scale to large instances

Is it a viable approach?



Develop
yet
another software
verifier

Check
the performance
of
existing ones

NeuroCodeBench (1)

Benchmarking goals

- Representativeness → realistic use cases
- Compatibility → SV likes plain C code
- Variety → from small to "large" instances
- Correctness → known ground truth

Our work is inspired by

- Microcontroller software

Short paper available!

- Manino et al., *NeuroCodeBench: a plain C neural network benchmark for software verification*, 2023
- `https://arxiv.org/abs/2309.03617`

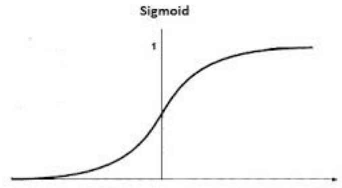
NeuroCodeBench (2)

Benchmark Category	Safe	Unsafe	Ground Truth
`math_functions`	33	11	A Priori
`activation_functions`	40	16	A Priori
`hopfield_nets`	47	33	A Priori
`poly_approx`	48	48	Brute Force
`reach_prob_density`	22	13	VNN-COMP'22
`reinforcement_learning`	103	193	VNN-COMP'22
Total	293	314	

Table: Overview of *NeuroCodeBench*. The "Unsafe" column comprises all properties for which a counterexample exists. The "Ground Truth" column reports the source of our verdicts.


Let's Play
Floating Point

Floating-point activations (1)



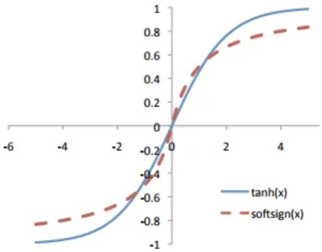
How to implement the sigmoid function?

- `s(float x){return 1.0f / (1.0f + expf(-x));}`
- `s(float x){return 0.5f * tanhf(0.5f * x) + 0.5f;}`

Let's compare them:

- Which one is faster?
- Which one is more precise?
- What happens for `x > 88.0f`?

Floating-point activations (2)



Implementing the softsign function:

- `s(float x){return x / (1.0f + fabsf(x));}`

Tricky questions:

- What happens for `x = -Inf`?
- Is our implementation always non-decreasing?


GAME OVER

NeuroCodeBench (3)

Functions from `math.h`

- expf, expm1f, logf, log1pf
- acosf, asinf, atanhf, cosf, sinf, tanhf
- erff, fabsf, fmaxf, sqrtf

Activation functions

- Elu, Gelu, Logistic, ReLU, Softmax, Softplus, Softsign, TanH

Safety properties (Examples)

- Output bounds: $expf(x) \geq 1 + x$
- Periodicity: $sinf(x) = sinf(x + 2\pi)$
- Symmetry: $tanhf(x) = -tanhf(-x)$
- Inversion: $expf(logpf(x)) = x$

Benchmark Category
`math_functions`
`activation_functions`
`hopfield_nets`
`poly_approx`
`reach_prob_density`
`reinforcement_learning`

```c
#include <verifier_functions.h>

#include <math.h>

float elu(float x)
{
        if(x >= 0.0f)
                return x;
        else
                return expm1f(x);
}
```

NeuroCodeBench (4)

Classic Hopfield Networks

- Recurrent architecture
- Hard-coded Hebbian weights
- Error-correcting behaviour

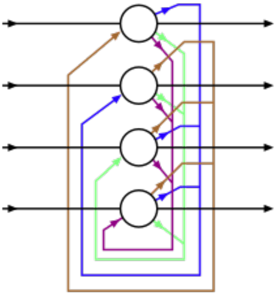
Our idea

- Reconstruct a single $x = (1, 1, \ldots, 1)$
- Use either softsign or tanh activations
- Vary code width and num of iterations

Safety properties

- Make $x_i \in [-1, +1]$ for $i <$ half width
- Can the network reach $x = (1, \ldots, 1)$?

Benchmark Category
`math_functions`
`activation_functions`
`hopfield_nets`
`poly_approx`
`reach_prob_density`
`reinforcement_learning`



NeuroCodeBench (5)

Transfer function approximation

- Very common in engineering
- Approximate electrical equivalent

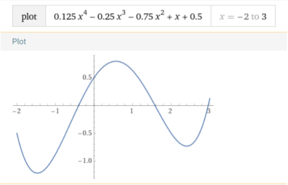
Our idea

- Fourth-order oscillating polynomial
- ReLUs, 1–4 layers, 16–1024 width

Safety properties

- Robustness of approximation
- $|\text{network}(x) - \text{poly}(x)| \leq \epsilon$
- for x around the worst-case

Benchmark Category
math_functions
activation_functions
hopfield_nets
poly_approx
reach_prob_density
reinforcement_learning



NeuroCodeBench (6)

Importing VNN-COMP benchmarks

- Choose categories from 2022 edition
- with very small neural networks

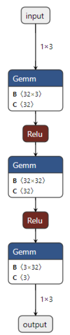
Converting to plain C code

- Turn ONNX into plain C with `onnx2c`
- Microcontroller-style minimalism

Safety properties

- Keep the original VNN-LIB properties
- Encode them as `assert`/`assume`
- Check validity of counterexamples

Benchmark Category
math_functions
activation_functions
hopfield_nets
poly_approx
reach_prob_density
reinforcement_learning



NeuroCodeBench (7)

Benchmark Category	Safe	Unsafe	Ground Truth
`math_functions`	33	11	A Priori
`activation_functions`	40	16	A Priori
`hopfield_nets`	47	33	A Priori
`poly_approx`	48	48	Brute Force
`reach_prob_density`	22	13	VNN-COMP'22
`reinforcement_learning`	103	193	VNN-COMP'22
Total	293	314	

Table: Overview of *NeuroCodeBench*. The "Unsafe" column comprises all properties for which a counterexample exists. The "Ground Truth" column reports the source of our verdicts.


COMPETITION
TIME!

Experimental results (1)

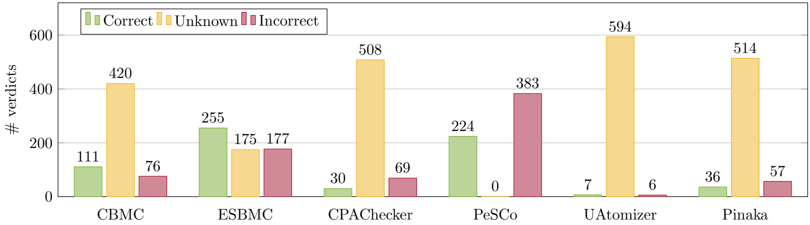
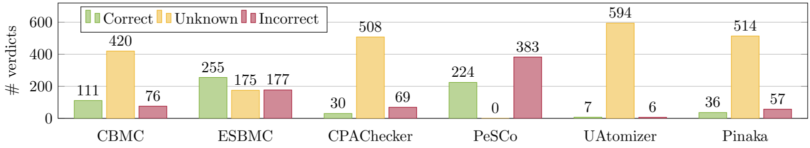


Figure: Results of state-of-the-art software verifiers after 900 seconds.

Experiments with off-the-shelf verifiers

- We pick the top scoring tools from SV-COMP 2022
- We keep the same settings of the reachability category
- Veriety of techniques: BMC, falsification, portfolios

Experimental results (2)



Reviewer 2 Must Be Stopped!

- ESBMC found 255 true properties, while PeSCo found 224.
- So, who is right? This odd behavior *must* be discussed.

Preliminary Analysis

- No support of mathematical libraries → incorrect results
- Cannot scale to large programs → unknown result (timeout)
- Other hidden bugs (floats, multi-dimensional arrays)?

Future work

Reproduce

- Submit *NeuroCodeBench* to SV-COMP 2023
- Experiments run by independent team
- Tool authors have a chance to fix bugs

Improve

- More benchmarks, neural networks, use cases
- Operational models to support `math.h`

Generalise

- If verifying neural code is too challenging
- Can we reason about sets of implementations?

SV-COMP 2024

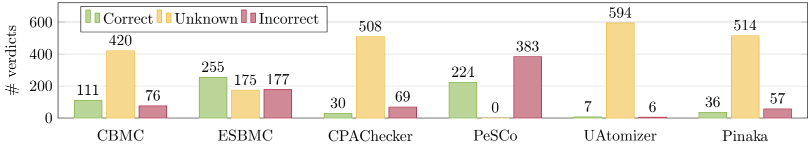


Figure: Results of state-of-the-art software verifiers after 900 seconds.

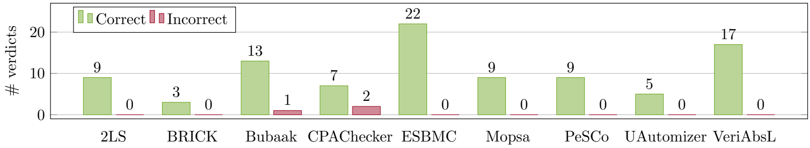
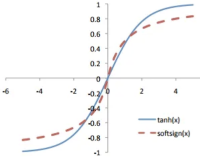


Figure: Results of various SV-COMP pre-runs (numbers may change).


You discover that verifiers are broken
THIS IS FINE.
But that means more future work!
imgflip.com

Floating-point activations (3)

```
float softsign(float x)
{ return x / (1.0f + fabsf(x)); }
```



Is our implementation always non-decreasing? No.[3]

- $x_1 = 15.000012397766113 \wedge x_2 = 15.000021934509277$
- $\mathit{softsign}(x_1) = 0.93750011920928955$
- $\mathit{softsign}(x_2) = 0.93750000000000000$
- $x_2 > x_1$ but $\mathit{softsign}(x_2) < \mathit{softsign}(x_1)$!
- It decreases by ≈ 0.00000012

[3] MinGW-w64 with options `-m64 -O2` and MUSL implementation of `math.h`.