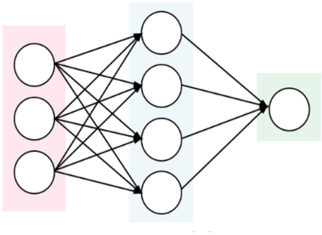
Verifying Quantized Neural Networks using SMT-Based Model Checking

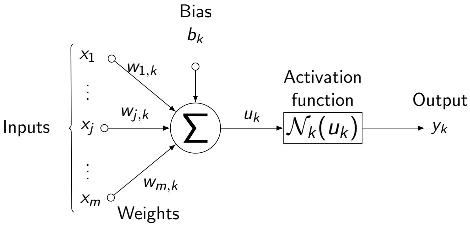
Luiz Sena, Xidan Song, Erickson Alves, Iury Bessa, Edoardo Manino, Lucas Cordeiro, Eddie de Lima Filho

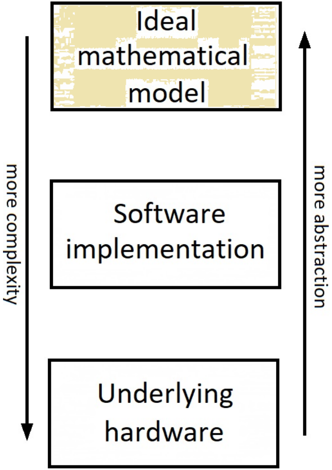
EnnCore, University of Manchester, Universidade Federal do Amazonas

February 28, 2022

Neural networks: just mathematical functions?

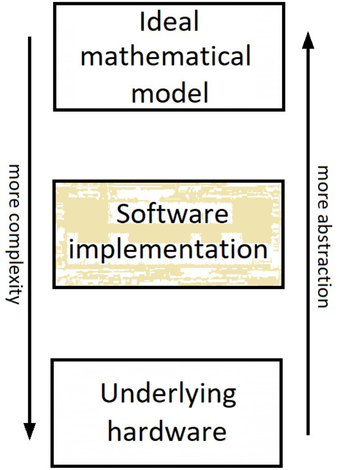






Neural networks: looking at the source code

```
float potential(float *w,
                unsigned int w_len,
                float *x,
                unsigned int x_len,
                float b) {

  if (w_len != x_len) {
    return 0;
  }

  float result = 0;

  for (unsigned int i = 0; i < w_len;
       ++i) {
    result += w[i] * x[i];
  }

  result += b;

  return result;
}
```



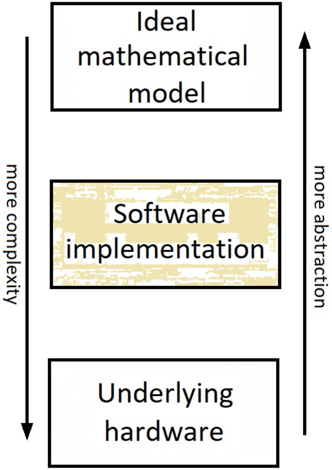
Research challenges

Verifying quantized NN

- Even floating-point is quantized!
- Fixed-point/integer arithmetics for low-power devices
- Approximated activation functions
- Complexity NP $\rightarrow$ PSPACE-hard

More software idiosyncrasies

- NaN, overflow, underflow
- Memory bugs, buffer overflows
- Concurrent execution bugs (GPUs)



Quantization effects

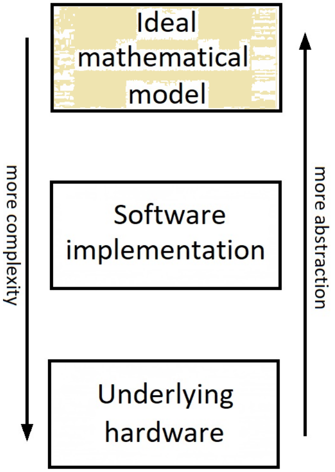
Safety Prop.		Number of bits													
		6	7	8	9	10	11	12	13		28	29	30	31	32
Set.	R_{40}	S	S	F	S	S	S	S	S	...	S	S	S	S	S
	R_{50}	S	S	F	F	F	F	F	F	...	F	F	F	F	S
Vers.	R_{20}	S	F	S	S	S	S	S	S	...	S	S	S	S	S
	R_{30}	S	F	S	S	S	S	S	S	...	S	S	S	S	S
	R_{40}	S	F	S	F	F	F	S	S	...	S	S	S	S	S
	R_{50}	S	F	F	F	F	F	F	F	...	F	F	F	F	F
Virg.	R_{20}	S	F	S	S	S	S	S	S	...	S	S	S	S	S
	R_{30}	S	F	S	S	S	S	S	S	...	S	S	S	S	S
	R_{40}	S	F	S	S	F	S	S	S	...	S	S	S	S	S
	R_{50}	S	F	F	F	F	F	F	F	...	F	F	F	F	F

Table: Effects of quantization on the safety of a NN for the Iris dataset.

Existing works

NN as an ideal mathematical function

- See last year's VNN-COMP'21
- Winner: $\alpha\beta$-CROWN
- Runners-up: VeriNet, Oval, ERAN...



Existing works

NN as an ideal mathematical function

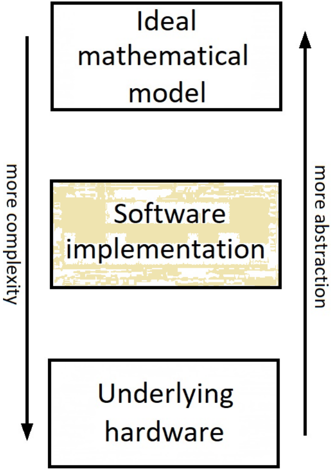
- See last year's VNN-COMP'21
- Winner: $\alpha\beta$-CROWN
- Runners-up: VeriNet, Oval, ERAN...

Quantization effects

- Giacobbe *et al.*, 2019 (ReLU-N)
- Henzinger *et al.*, 2021 (ReLU-N)
- Baranowski *et al.*, 2020 (fixed-point)

Other implementation effects

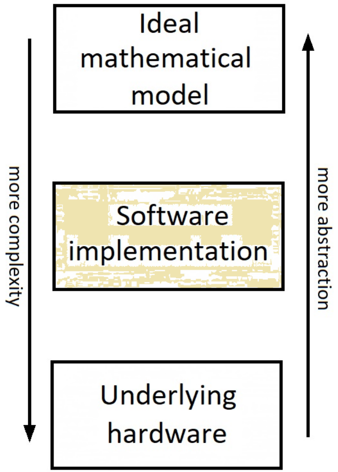
- Odena *et al.*, 2019 (fuzz testing)
- Sena *et al.*, 2019 (CUDA)



Our verification framework (high-level view)

Goal

- Support floating-point, fixed-point, integer and binary arithmetic
- Support all activation functions
- Let the user specify a wide range of safety properties



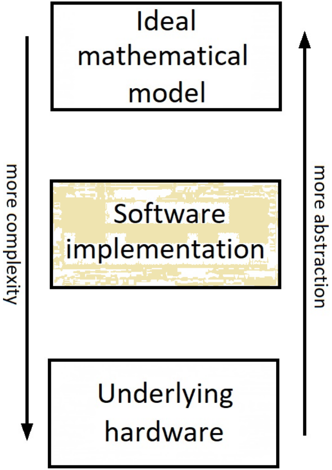
Our verification framework (high-level view)

Goal

- Support floating-point, fixed-point, integer and binary arithmetic
- Support all activation functions
- Let the user specify a wide range of safety properties

Main ideas

- Apply model checking techniques
- C code as a low level abstraction
- Safety property with `assume()` and `assert()` instructions
- Convert code + property into SMT
- Check satisfiability of SMT formula



Our verification framework (SMT encoding)

```c
int main() {
  _Bool x, y;
  int a, b, f;
  x = nondet_bool();
  y = nondet_bool();
  a = ((2*x) - (3*y));
  a = a < 0 ? 0 : a;
  b = (x + (4*y));
  b = b < 0 ? 0 : b;
  f = ((3*x) + y);
  f = f < 0 ? 0 : f;
  assert(a <= 2 && b <= 5 && f <= 4);
  return 0;
}
```

SSA

```
x1 == nondet_symbol(nondet0)
y1 == nondet_symbol(nondet1)
a1 == 2 * (int)x1 - 3 * (int)y1
a2 == (a1 < 0 ? 0 : a1)
b1 == (int)x1 + 4 * (int)y1
b2 == (b1 < 0 ? 0 : b1)
f1 == 3 * (int)x1 + (int)y1
f2 == (f1 < 0 ? 0 : f1)
(assert) a2 <= 2
(assert) b2 <= 5
(assert) f2 <= 4
```

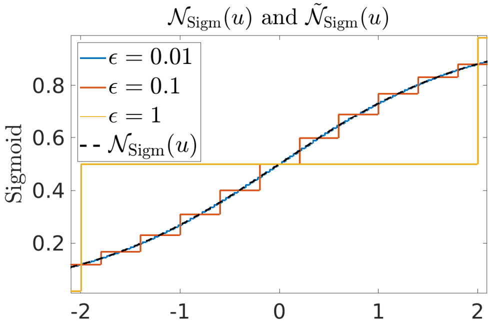
aggressive simplifications

```
x1 == nondet_symbol(nondet0)
y1 == nondet_symbol(nondet1)
a1 == 2 * (int)x1 - 3 * (int)y1
a2 == (a1 < 0 ? 0 : a1)
```

SMT

```
...
(assert(= a1 (- (* 2 x1) (* 3 y1))))
(assert (and (< a1 0) (= a2 0)))
(assert (and (> a1 0) (= a2 a1)))
...
```

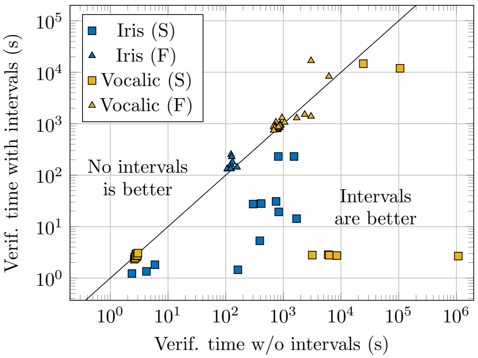
Our verification framework (activation functions)



Encoding non-linear functions

- Piecewise linear (e.g. ReLU) $\rightarrow$ if-then-else
- Others (e.g. sigmoid, tanh) $\rightarrow$ lookup table (DSP-style)
- Speeds up both inference and verification!

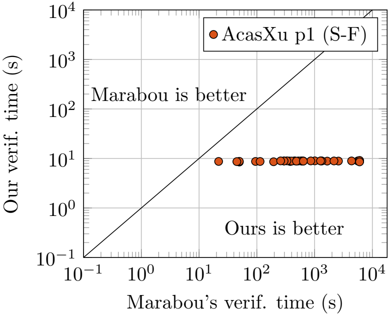
Our verification framework (interval analysis)

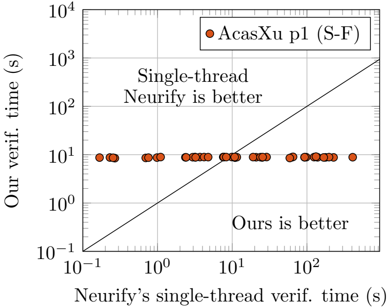


Input set propagation

- Transferable from verification of ideal NNs
- Generates an overapproximation of the neuron values
- Reduces the search space for safe (S) instances

Our verification framework (comparison with SOTA)





Warning: this is not an equal contest!

- Comparison between infinite precision and fixed-point
- Useful as a qualitative result

Conclusions

Summary

- Implementations of NNs are software!
- Quantization effects, finite arithmetic, other potential bugs
- Higher theoretical complexity than verifying ideal NNs
- Positive note: similar verification time in practice

Further resources

- Try our QNNVerifier tool:
- `https://arxiv.org/abs/2111.13110`
- Read our pre-print journal paper:
- `https://arxiv.org/abs/2106.05997JournalPaper`

Thank you!