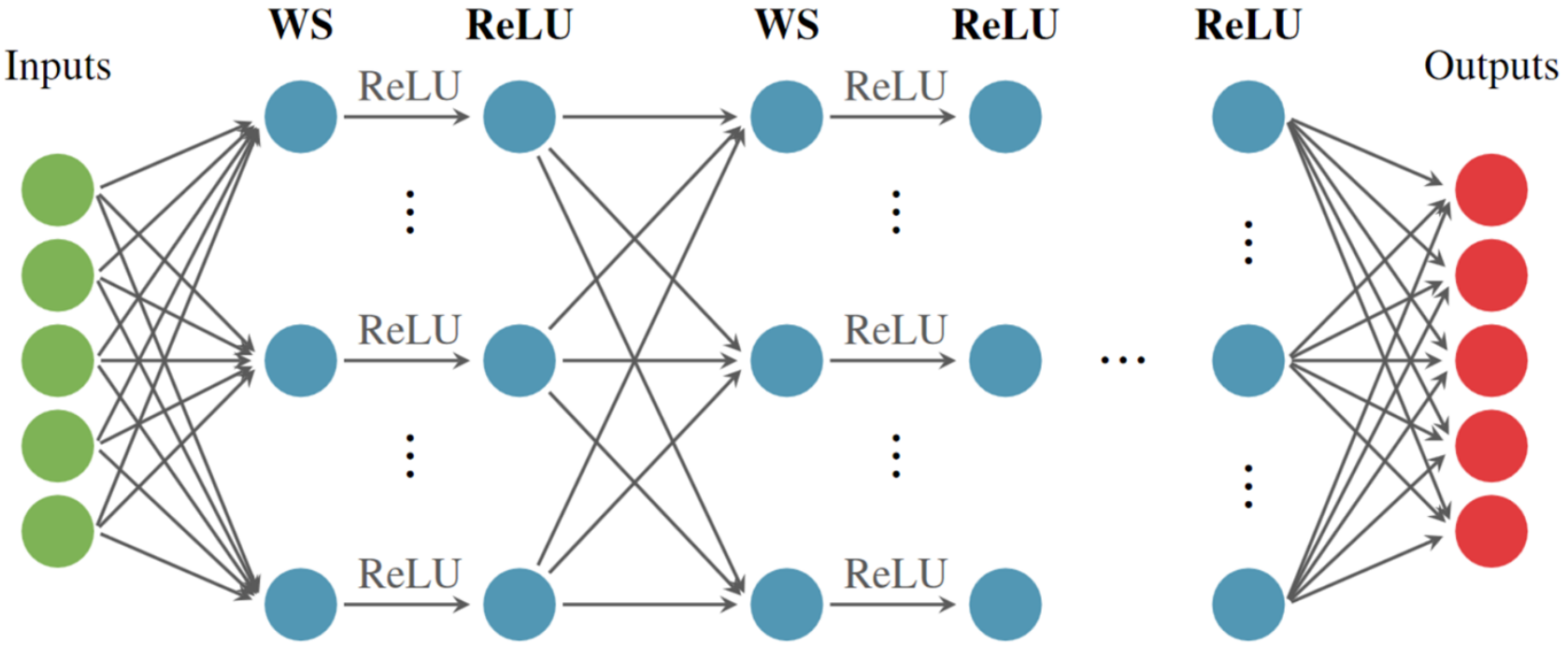
Ori Lahav | Guy Katz

Network Simplification

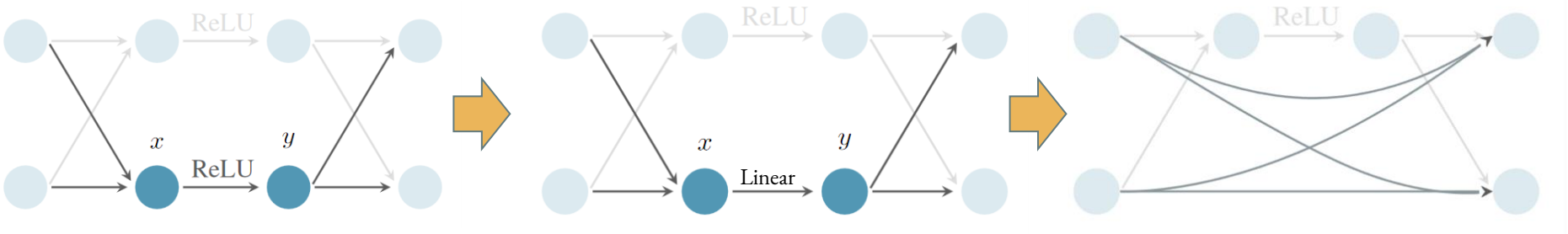
fmcad.21

https://arxiv.org/abs/2105.13649
https://github.com/vbcrlf/redy


WS
ReLU
WS
ReLU
ReLU
Inputs
Outputs
ReLU
ReLU
ReLU
ReLU
ReLU
ReLU

How do we remove a neuron?

1. Convert a neuron activation function into a linear function
2. Merge consecutive linear layers



02 Types of Redundancy

Criteria for a redundant neuron

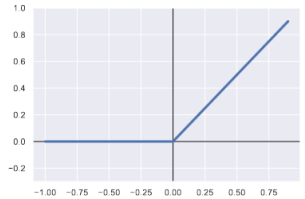
02 Types of redundancy

Which neurons can we remove?

- **A neuron which it's removal does not affect the output of the network**

- Concretely, we propose 4 different criteria for asserting a neuron can be removed

a. Phase Redundancy



Observation

- These operations will not affect the network output:
  - Replacing a ReLU neuron with **Zero** if it is **always inactive** (e.g. input ≤ 0)
  - Replacing a ReLU neuron with **Identity** if it is **always active** (e.g. input ≥ 0)

- Can be generalized to any piecewise-linear activation function
  - We call it **Phase-Redundancy**

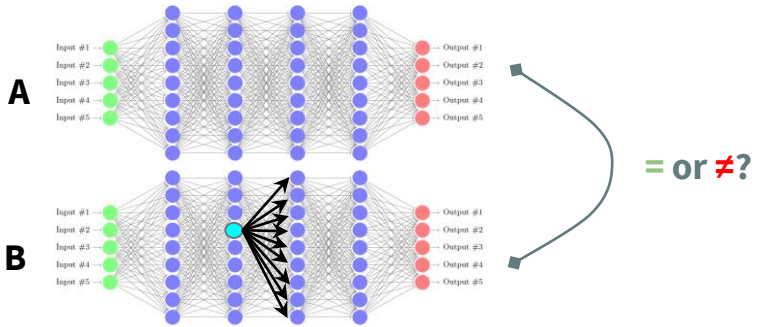
- Least general category, but is the easiest to calculate!

"Give me an input for which the neuron input is less then zero."

b. Forward Redundancy

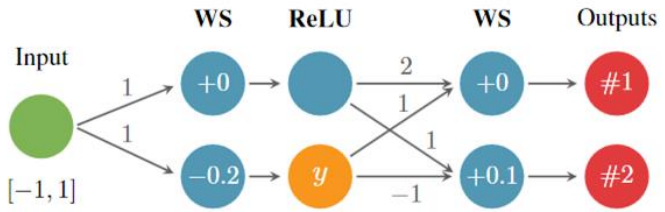
- A neuron might **not be Phase-Redundant**, but still to not affect the output

- How do we find such neurons using a verification engine?
  a. Duplicate the network and remove the neuron in the copy (**B**)
  b. Compare with the original (**A**)



c. Result-Preserving Redundancy

- **In a classification network**, what does it mean to not **affect the output?**
  - For any given input, the ‘winning’ class should be the same
  - e.g. we don’t care if the numerical output is completely different!



Cat: 0.4
Dog: 0.8

⇒ **Dog!**

Cat: 0.4
Dog: 0.5

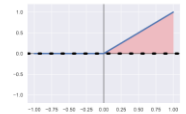
⇒ **Dog!**

d. Relaxed Redundancy

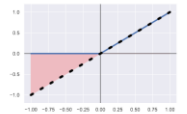
- We discussed replacing a neuron with a one of it's linear phases
- But our approach allows for replacing a neuron with **any** linear function!
  - What else can we replace it with?

- Intuitively, we would want to minimize the maximal error.
  - In the case of ReLU (*ub*, *lb* are input bounds for the neuron):

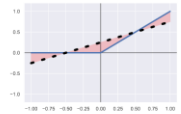
$$l_m(x) = \frac{ub}{ub - lb} \cdot x + \frac{-lb \cdot ub}{2(ub - lb)}.$$



(a) Replacing a ReLU with the zero function



(b) Replacing a ReLU with identity function



(c) Replacing a ReLU with an arbitrary linear function

03 Strategy

1. **Bound estimation using MILP**
   - Using these bounds, we can conclude which neurons are Phase-Redundant.

2. **Simulations and Formal Verification**
   - Use simulations to eliminate non-redundant neurons
   - Run formal verification queries to find redundant neurons

3. **Relaxed Redundant and Error Approximation**
   - As a last step, we can remove neurons and approximate the maximum output error

RECAP

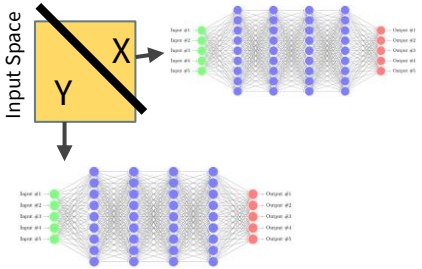
- **We introduced a few categories of redundancy**
- **… and a strategy for finding such redundancies**

- But we can do better…

04 Slicing

Introducing Redundancies

- **Observation:**
  - If we constrain the network to a smaller input space, some neurons will be redundant.



- **The idea:**
  - Split the input space to N different subspaces.
  - For each subspace:
    - Run the redundancy removal strategy on the network, constrained to that subspace only

- **We will end up with many smaller networks.**
  - On evaluation, we choose the the right network based on the input.

- This technique improves the evaluation time, at the cost of memory.

Experimental results

- We splitted the input space of an ACAS network to about 32K subspaces
- 32K subspaces may sound like a lot, but finding Phase-Redundant neurons is relatively quick
  - We ran this step on all 32K subspaces and **67.3% of all neurons were found redundant.**
- We ran the full pipeline on 50 random subspaces
  - **82.5%** neurons were redundant.

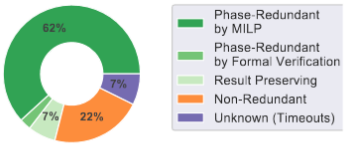


Fig. 9: Redundant neuron removal, averaged over 50 ACAS Xu input sub-domains.

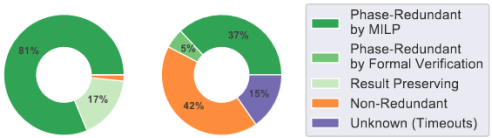


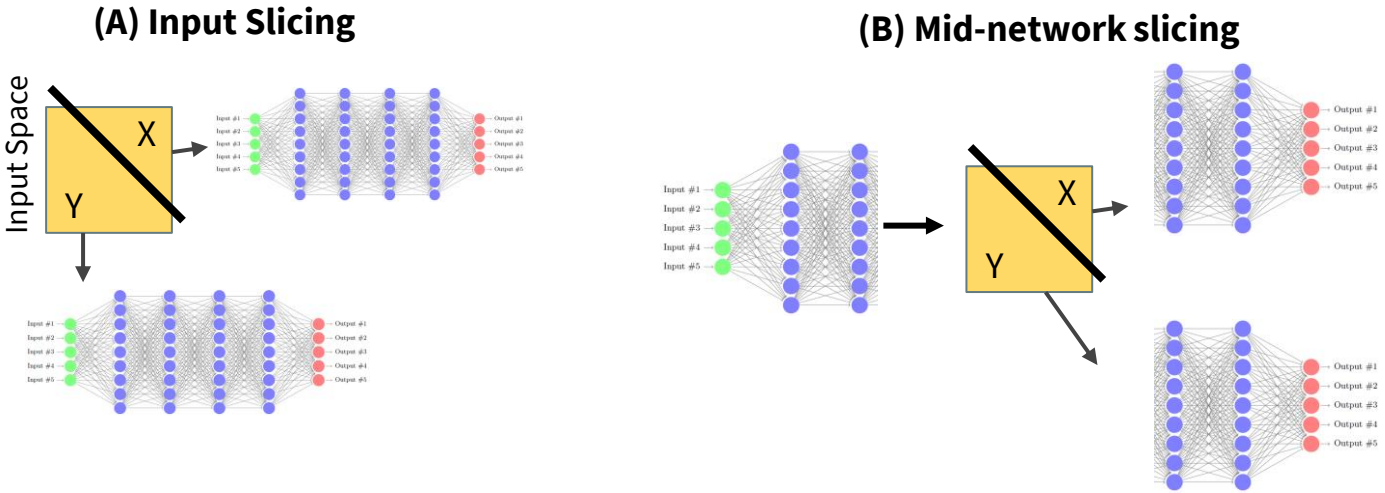
Fig. 10: An "almost" linear sub-domain (left) vs. a complex sub-domain (right).

Doesn't work well on all networks, probably...

1. What if the network have a large amount of inputs?
2. We don't expect this method to work well on image recognition networks

Two possible solutions

1. Find an intelligent way to slice the input space
2. Mid-network slicing:



https://arxiv.org/abs/2105.13649 **PAPER**
https://github.com/vbcrlf/redy **CODE**

Future Work

1. **Slicing**
   - Intelligent slicing
   - Explore mid-network slicing
   - Compressed representation of sliced network
2. **Forward/Result-Preserving** – find a more efficient technique
3. **Analytical error bound on neuron removal** – tighten the bound
4. Explore techniques other than neuron removal