DNN Verification Methods Analysis

1. Background

1.1 Formal Verification Categorization Based on Completeness

Formal verification algorithms can be categorized as complete or incomplete. [1]

DNN formal verification algorithm categories

Name	Characteristic	Activation functions	Approaches
Complete	Encode the exact network Take longer to run(high computational cost) More amenable to verifying properties of smaller networks	Only support piece-wise linear activation functions	MILP, SMT, SIP
Incomplete	Over-approximation relaxations for the network (may thus only be able to verify a subset of the inputs) Usually rely on over-approximations or other schemes that significantly reduce computational cost Faster and can more easily be used to verify properties on larger networks	Supports piece-wise as well as other non-piecewise linear activation functions (sigmoid, tanh)	abstract-interpretation, SIP

Relaxation: approximation of a difficult problem by a nearby problem that is easier to solve (e.g. linear relaxations of non-linear units in neural networks)

Source: [1](I Introduction Related Work), [2](Comparison and Results p119)

1.2 Complete Algorithm Approaches

Complete algorithms are usually based on MILP, SMT, or SIP. [1](I Introduction Related Work)

Complete approaches

Name	Characteristic
MILP(Mixed Integer Linear Programming)	Often utilize efficient off-the-shelf solvers by encoding piece-wise linear activation functions with integer constraints
SMT (Satisfiability Modulo Theory)	Reason over a relaxed network and combine this with a branch-and-bound refinement to iteratively remove the overestimation
SIP (Symbolic Interval Propagation)	Aim at reducing the resulting complexity by calculating symbolic bounds for the network's output nodes in a seperate phase, which are then used as constrains in an LP-solver, eliminating the need to encode the hidden nodes

MILP: Adds one additional condition that at least one of the variables can only take on integer values to LP.

Source: [1](I Introduction Related Work)

1.3 Types of Checkable Properties

There are largely 3 types of properties that are currently checkable via conventional tools

Types of properties

Name	Characteristic	Tool
I(interval)/O(interval)	Reachability queries If inputs is in a given range is the output guaranteed to be in some, typically safe, range	Marabou, Neurify, DeepPoly, NNV
I(perturbation range)/O(NN output)	Robustness queries Test whether there exist adversarial points around a given input point that change the output of the network	Marabou,, Neurify, DeepPoly, NNV

Name	Characteristic	Tool
I(boolean combination)/O(NN output)	Conjunctive/disjunctive combination of constraints and propositions	Marabou, Neurify, DeepPoly (Conjunction Only) NNV (CNF, DNF supported)

Source: [6](README, Wiki)

Type 3 property (i.e. I(boolean combination)/O(NN output)) is the most general form of property and in theory, Type 1 and 2 are both a type of Type 3 property. The reason why they are categorized separately is that tools mainly focus/specialize on Type 1 or 2 or both and don't support Type 3.

2. Overview of Methods

Researched, Categorized, and Compared Reluplex/Marabou, Neurify, DeepPoly, and NNV

2.1 Based on Soundness & Completeness

Soundness and completeness of the tools are determined by what approach the tool utilizes.

Method Soundness, Completeness

Name	Soundness	Completeness	Approach type
Reluplex	Sound	Complete	SMT
Marabou	Sound	Complete	SMT
Neurify	Sound	Incomplete	SLR(Symbolic Linear Relaxation)
DeepPoly	Sound	Incomplete	abstract interpretation Reachability
NNV(Exact)	Sound	Complete	Set representation (Star, Polyhedron, ImageStar) Reachability
NNV(Approx)	Sound	Incomplete	Set representation (Star, Zonotope, Abstract-domain/abstract interpretation, ImageStar) Reachability

Sound: Anytime an answer is returned, that answer is true. (Prevents false positives)

Complete: Guarantees to return a correct answer for any arbitrary input (or, if no answer exists, it guarantees to return failure). Addresses all possible inputs and doesn't miss any. (Prevents false negatives)

Source: [3](p10~11), [4], [5], [7], [8], [9]

2.2 Based on Supported Activation Function & Architecture

Tools/methods using over-approximation can handles non-linear activation functions (which are not piece-wise linear).

Supported Activation Functions, Architecture

Name	Activation Functions	Architecture
Reluplex	ReLU	FCNN
Marabou	Piece-wise linear functions (ReLU, Max)	FFNN, CNN
Neurify	ReLU	FCNN, CNN
DeepPoly	ReLU, sigmoid, tanh (monotonically increasing, convex on (-inf, 0] and concave on [0, +inf)	FFNN, CNN
NNV(Exact)	ReLU, Saturating linear transfer function(Satlin)	FFNN, CNN
NNV(Approx)	ReLU, Satlin, sigmoid, tanh (non-linear)	FFNN, CNN

FCNN: Fully Connected Neural Network

FNN: Feed-Forward Neural Network

CNN: Convolutional Neural Network

Satlin: Saturating Linear Transfer Function

Source: [4], [5], [7], [8], [9

3. Reluplex/Marabou

As Marabou is an upgraded version of Reluplex, only considered Marabou in this section.

3.1 How Marabou Handles Properties

Query(Property) given in txt format(examples given in 3.1,2,3) handled/loaded via QueryLoader. Below is the section of the code that generates Equation type instances from loaded text equations and appending them to inputQuery.

```
// Equations
for( unsigned i = 0; i < numEquations; ++i )
{
    QL_LOG( Stringf( "Equation: %u ", i ).ascii() );
    String line = input->readLine();

    List<String> tokens = line.tokenize( "," );
    ASSERT( tokens.size() > 4 );

    auto it = tokens.begin();

    // Skip equation number
    ++it;
    int eqType = atoi( it->ascii() );
    QL_LOG( Stringf("Type: %u ", eqType ).ascii() );
    ++it;
    double eqScalar = atof( it->ascii() );
    QL_LOG( Stringf( "Scalar: %f\n", eqScalar ).ascii() );

    Equation::EquationType type = Equation::EQ;

    switch ( eqType )
    {
    case 0:
        type = Equation::EQ;
        break;

    case 1:
        type = Equation::GE;
        break;

    case 2:
        type = Equation::LE;
        break;

    default:
        // Throw exception
        throw MarabouError( MarabouError::INVALID_EQUATION_TYPE, Stringf( "Invalid Equation Type\n" ).ascii() );
        break;
    }

    Equation equation( type );
    equation.setScalar( eqScalar );

    while ( ++it != tokens.end() )
    {
        int varNo = atoi( it->ascii() );
        ++it;
        ASSERT( it != tokens.end() );
        double coeff = atof( it->ascii() );

        QL_LOG( Stringf( "\tVar_no: %i, Coeff: %f\n", varNo, coeff ).ascii() );

        equation.addAddend( coeff, varNo );
    }

    inputQuery.addEquation( equation );
}
```

https://github.com/NeuralNetworkVerification/Marabou/blob/master/src/query_loader/QueryLoader.cpp

As a result query in the form of input, output variables, bounds, equations, and piece-wise linear constraints(max, sign, etc) is stored in the InputQuery data structure.

```
private:
    unsigned _numberOfVariables;
    List<Equation> _equations;
    Map<unsigned, double> _lowerBounds;
```

```
    Map<unsigned, double> _upperBounds;
    List<PiecewiseLinearConstraint *> _plConstraints;
```

https://github.com/NeuralNetworkVerification/Marabou/blob/master/src/engine/InputQuery.h

3.2 Verifying I(interval)/O(interval) Type Properties

Explicitly gives input and output variable range in txt files.

```
x0 >= 0.6
x0 <= 0.6798577687
x1 >= -0.5
x1 <= 0.5
x2 >= -0.5
x2 <= 0.5
x3 >= 0.45
x3 <= 0.5
x4 >= -0.5
x4 <= -0.45
y0 >= 3.9911256459
```

https://github.com/NeuralNetworkVerification/Marabou/blob/master/resources/properties/acas_property_1.txt

3.3 Verifying I(perturbation range)/O(NN output) Type Properties

For every input xi (input variable), add input constraints: xi ≥ x - epsilon and xi ≤ x + epsilon.

```
# To generate a query that checks if the first(index th) image in the
# training data can be perturbed to be 8(target) with 0.04 (epsilon)
# perturbation bound

# target: 0..9 corresponding to y0..y9
for i, x in enumerate(X):
    print('x{} >= {}'.format(i, x - epsilon))
    print('x{} <= {}'.format(i, x + epsilon))
for i in range(10):  --> # of classes in MNIST
    if i != target:
        print('+y{} -y{} <= 0'.format(i, target))
```

https://github.com/NeuralNetworkVerification/Marabou/blob/master/resources/properties/mnist/dump_mnist_targeted_

For the output constraints, constraints are added that equates to the output variable corresponding to the target having the maximum value.

```
# target: 1

+y0 -y1 <= 0
+y2 -y1 <= 0
+y3 -y1 <= 0
+y4 -y1 <= 0
+y5 -y1 <= 0
+y6 -y1 <= 0
+y7 -y1 <= 0
+y8 -y1 <= 0
+y9 -y1 <= 0
```

https://github.com/NeuralNetworkVerification/Marabou/blob/master/resources/properties/mnist/image1_target1_epsilon

3.4 Verifying I(boolean combination)/O(NN output) Type Properties

Inequalities over input and output variables are implicitly connected by conjunctions. Disjunctive properties are not supported, but can be broken down into multiple property files and checked separately.

https://github.com/NeuralNetworkVerification/Marabou/wiki/Marabou-Input-Formats

4. Neurify

4.1 How Neurify Handles Properties

According to the property in question, arrays/matrices of floats containing lower and upper bounds of the input variables are created and set (u, l). The created input matrices (input_upper, input_lower) are then stored in Interval data structure (input_interval)

```
load_inputs(PROPERTY, inputSize, u, l);

struct Matrix input_upper = {u,1,nnet->inputSize};
struct Matrix input_lower = {l,1,nnet->inputSize};

struct Interval input_interval = {input_lower, input_upper}
```

https://github.com/tcwangshiqi-columbia/ReluVal/blob/master/network_test.c

For the output constraints, custom functions that check the content of the output Interval data structures are used.

```
# custom func. for every property
int check_max_constant(struct NNet *nnet, struct Interval *output)
{
    if (output->upper_matrix.data[nnet->target] > 0.5011) {
        return 1;
    }
    else {
        return 0;
    }

}

int check_functions(struct NNet *nnet, struct Interval *output)
{

    if (PROPERTY ==1) {
        return check_max_constant(nnet, output);
    }
```

https://github.com/tcwangshiqi-columbia/ReluVal/blob/68ce6aaa28f0d7d9a8e10d1c64c01999f1aef88d/split.c

4.2 Verifying I(interval)/O(interval) Type Properties

Set the values of the float arrays used to create the intervals according to the property in question

```
void load_inputs(int PROPERTY, int inputSize, float *u, float *l)
{

    if (PROPERTY == 1) {
        float upper[] = {60760,3.141592,3.141592,1200,60};
        float lower[] = {55947.691,-3.141592,-3.141592,1145,0};
        memcpy(u, upper, sizeof(float)*inputSize);
        memcpy(l, lower, sizeof(float)*inputSize);
    }
```

https://github.com/tcwangshiqi-columbia/ReluVal/blob/68ce6aaa28f0d7d9a8e10d1c64c01999f1aef88d/nnet.c

As mentioned above, create a custom function to express constraints involving the output interval.

```
int check_max_constant(struct NNet *nnet, struct Interval *output)
{
    if (output->upper_matrix.data[nnet->target] > 0.5011) {
        return 1;
    }
    else {
        return 0;
```

```
    }
}
```

https://github.com/tcwangshiqi-columbia/ReluVal/blob/master/split.c

4.3 Verifying I(perturbation range)/O(NN output) Type Properties

Special CHECK_ADV_MODE for checking adversarial robustness (i.e robustness against perturbation).

```
if (CHECK_ADV_MODE) {
    printf("check mode: CHECK_ADV_MODE\n");
    isOverlap = direct_run_check(nnet,\
            &input_interval, &output_interval,\
            &grad_interval, depth, feature_range,\
            feature_range_length, split_feature);
}
```

https://github.com/tcwangshiqi-columbia/ReluVal/blob/68ce6aaa28f0d7d9a8e10d1c64c01999f1aef88d/network_test.c

In the interval split process that happens via the direct_run_check function execution, check_adv function is used to find concrete adversarial examples.

```
void check_adv(struct NNet* nnet, struct Interval *input)
{

    float a[nnet->inputSize];
    struct Matrix adv = {a, 1, nnet->inputSize};

    for (int i=0;i<nnet->inputSize;i++) {
        float upper = input->upper_matrix.data[i];
        float lower = input->lower_matrix.data[i];
        float middle = (lower+upper)/2;

        a[i] = middle;
    }

    float out[nnet->outputSize];
    struct Matrix output = {out, nnet->outputSize, 1};

    forward_prop(nnet, &adv, &output);

    int is_adv = 0;
    is_adv = check_functions1(nnet, &output);

    //printMatrix(&adv);
    //printMatrix(&output);

    if (is_adv) {
        printf("\nadv found:\n");
        denormalize_input(nnet, &adv);
        printf("adv is: ");
        printMatrix(&adv);
        printf("it's output is: ");
        printMatrix(&output);
        pthread_mutex_lock(&lock);
        adv_found = 1;
        pthread_mutex_unlock(&lock);
    }
}
```

https://github.com/tcwangshiqi-columbia/ReluVal/blob/68ce6aaa28f0d7d9a8e10d1c64c01999f1aef88d/split.c

4.4 Verifying I(boolean combination)/O(NN output) Type Properties

Boolean combination form of property is not possible as from the start, input and output are stored in Interval data structures

```
struct Matrix input_upper = {u,1,nnet->inputSize+1};
struct Matrix input_lower = {l,1,nnet->inputSize+1};
struct Interval input_interval = {input_lower, input_upper};
```

```
initialize_input_interval(nnet, img, inputSize, input_prev, u, l);

if(NORM_INPUT){
    normalize_input(nnet, &input_prev_matrix);
    normalize_input_interval(nnet, &input_interval);
}

float o[nnet->outputSize];
struct Matrix output = {o, outputSize, 1};

float o_upper[nnet->outputSize], o_lower[nnet->outputSize];
struct Interval output_interval = {
            (struct Matrix){o_lower, outputSize, 1},
            (struct Matrix){o_upper, outputSize, 1}
        };
```

https://github.com/tcwangshiqi-columbia/Neurify/blob/master/general/network_test.c

5. DeepPoly

5.1 How DeepPoly Handles Properties

Input constraints are loaded from text and saved into an interval list containing the lower and upper bounds of the input variables.

```
def parse_input_box(text):
    intervals_list = []
    for line in text.split('\n'):
        if line!="":
            interval_strings = re.findall("\[-?\d*\.?\d+, *-?\d*\.?\d+\]", line)
            intervals = []
            for interval in interval_strings:
                interval = interval.replace('[', '')
                interval = interval.replace(']', '')
                [lb,ub] = interval.split(",")
                intervals.append((np.double(lb), np.double(ub)))
            intervals_list.append(intervals)
```

https://github.com/eth-sri/eran/blob/c331b579f3ccabb4c66b7f950abf3640976c0031/tf_verify/__main__.py

Output constraints are loaded from txt files and appended to a list via the get_constraints_from_file function.

```
    if constraint == 'min':
        for other in range(num_labels):
            if other not in labels:
                and_list.append([(other, label, 0) for label in labels])

    if constraint == 'max':
        for other in range(num_labels):
            if other not in labels:
                and_list.append([(label, other, 0) for label in labels])

    if constraint == 'notmin':
        others = filter(lambda x: x not in labels, range(num_labels))
        # this constraint makes only sense with one label
        label = labels[0]
        and_list.append([(label, other, 0) for other in others])

    if constraint == 'notmax':
        others = filter(lambda x: x not in labels, range(num_labels))
        # this constraint makes only sense with one label
        label = labels[0]
        and_list.append([(other, label, 0) for other in others])

    if constraint == '<':
        label2 = label_index(elements[i])
        and_list.append([(label2, label, 0) for label in labels])

    if constraint == '>':
        label2 = label_index(elements[i])
        and_list.append([(label, label2, 0) for label in labels])
    if constraint == '<=' and isfloat(elements[i]):
        and_list.append([(label, -1, float(elements[i])) for label in labels])
```

```
return and_list
```

https://github.com/eth-sri/eran/blob/badc3d6e5317f45f8d8a6f8bbac3fdb23cc9c4b4/tf_verify/constraint_utils.py

5.2 Verifying I(interval)/O(interval) Type Properties

Input constraints/ranges and output constraints are given separately. One file containing the pre-normalized inputs.

```
[55947.691,60759.9999996307]
[-3.14159265359, 3.14159265359]
[-3.14159265359, 3.14159265359]
[1145.0, 1200.0]
[0.0, 60.0]
```

https://github.com/eth-sri/eran/blob/master/data/acasxu/specs/acasxu_prop_1_input_prenormalized.txt

And one containing the number of NN inputs and the output constraints.

```
5
y0 <= 3.991125
```

https://github.com/eth-sri/eran/blob/master/data/acasxu/specs/acasxu_prop_1_constraints.txt

5.3 Verifying I(perturbation range)/O(NN output) Type Properties

According to the user specified attack/perturbation type, the input lower and upper bounds are modified by attack parameters.

```
attack_imgs, checked, attack_pass = [], [], 0
cex_found = False
if config.attack:
    for j in tqdm(range(0, len(attack_params))):
        params = attack_params[j]
        values = np.array(attack_images[j])

        attack_lb = values[::2]
        attack_ub = values[1::2]

        normalize(attack_lb, means, stds, config.dataset)
        normalize(attack_ub, means, stds, config.dataset)
        attack_imgs.append((params, attack_lb, attack_ub))
        checked.append(False)

        predict_label, _, _, _, _, _ = eran.analyze_box(
            attack_lb[:dim], attack_ub[:dim], 'deeppoly',
            config.timeout_lp, config.timeout_milp, config.use_default_heuristic)
        if predict_label != int(test[0]):
            print('counter-example, params: ', params, ', predicted label: ', predict_label)
            cex_found = True
            break
        else:
            attack_pass += 1
```

https://github.com/eth-sri/eran/blob/c331b579f3ccabb4c66b7f950abf3640976c0031/tf_verify/__main__.py

5.4 Verifying I(boolean combination)/O(NN output) Type Properties

The input set/space is represented by lower and upper bound intervals and the output constraints are in CNF(conjunctive normal form). As such, disjunctive form properties can not be checked in one go. Just like Marabou, properties in DNF must be broken down into multiple property files and checked separately.

6. NNV

6.1 How NNV Handles Properties

The input intervals/constraints can and are represented using one of the reachable set representation provided by the NNV framework (Star, Polyhedron, Zonotope, etc)

```
% -2 <= x1 <= 2
% 0 <= x2 <= 1
lb = [-2; 0]; % lower bound vector
ub = [2; 1]; % upper bound vector

I2 = Star(lb, ub); % star input set
```

https://github.com/verivital/nnv/blob/71b41f19e6cc9ebf99e1f8347f66180887ef8b34/code/nnv/examples/Manual/exam

As for the constraints on the output variables, a HalfSpace object is used to represent it.

```
%/* Example of specifying a property of FFNN */

% unsafe region: y1 >= 5 (the network has two outputs)

G = [-1 0]; % condition matrix

g = -5;     % condition vector

U = HalfSpace(G, g); % unsafe region object
```

https://github.com/verivital/nnv/blob/master/code/nnv/examples/Manual/example_ffnns_specify_property.m

6.2 Verifying I(interval)/O(interval) Type Properties

A half-space is that portion of an n-dimensional space obtained by removing that part lying on one side of an (n-1)-dimensional hyperplane which can be specified with inequalities over the axis variables. As such, output variables range/interval can be expressed via the HalfSpace object.

```
% -2 <= x1 <= 2
% 0 <= x2 <= 1
lb = [-2; 0]; % lower bound vector
ub = [2; 1]; % upper bound vector

I2 = Star(lb, ub); % star input set
```

https://github.com/verivital/nnv/blob/71b41f19e6cc9ebf99e1f8347f66180887ef8b34/code/nnv/examples/Manual/exam

6.3 Verifying I(perturbation range)/O(NN output) Type Properties

Evaluates robustness via a special function, evaluateRobustness.

```
%% Construct input sets
dif_images = load([path_base, 'pepper_dif_images.mat']);
ori_images = load([path_base, 'pepper_ori_images.mat']);

dif_images = struct2cell(dif_images);
ori_images = struct2cell(ori_images);

N = 20; % choose 20 cases

l = [0.96 0.97 0.98];
delta = [0.0000001 0.0000002 0.0000005];

P = length(l);
M = length(delta);

inputSetStar = cell(P, M);
correct_labels = cell(P,M);
for i=1:P
```

```
    for j=1:M
        lb = l(i);
        ub = l(i) + delta(j);
        IS(N) = ImageStar;
        for k=1:N
            V(:,:,:,1) = double(ori_images{k});
            V(:,:,:,2) = double(dif_images{k});
            IS(k) = ImageStar(V, [1;-1], [ub; -lb], lb, ub);
        end
        inputSetStar{i, j} = IS;
        correct_labels{i, j} = 946 * ones(1, N);
    end
end


% Load the trained model
net = vgg16();

nnvNet = CNN.parse(net, 'VGG16');

%% evaluate robustness

VT_star = zeros(P, M); % verification time of the approx-star method
r_star = zeros(P, M); % robustness percentage on an array of N tested input sets obtained by the approx-star method

c = parcluster('local');
numCores = c.NumWorkers; % specify number of cores used for verification


for i=1:P
    for j=1:M

        t = tic;
        r_star(i, j) = nnvNet.evaluateRobustness(inputSetStar{i, j}, correct_labels{i, j}, 'approx-star', numCores);
        VT_star(i, j) = toc(t);
    end
end
```

https://github.com/verivital/nnv/blob/71b41f19e6cc9ebf99e1f8347f66180887ef8b34/code/nnv/examples/NN/CNN/VGG

6.4 Verifying I(boolean combination)/O(NN output) Type Properties

As disjunction and conjunction is a special case of half-space, the boolean combination form of property either CNF and DNF seems viable.

$x_1 \wedge \bar{x}_2 \wedge \cdots \wedge x_r <=> x_1 + (1 - x_2) + \cdots + x_r \geq 1$

https://cs7545.wordpress.com/2015/02/26/notes-0219-learning-halfspace/

7. Experiments

Intel Core i7-6850K CPU @3.60GHz 6 core Processor

```
#SBATCH  -t 1-00:00:00
#SBATCH  --nodes=1
#SBATCH  --ntasks=2
#SBATCH  --tasks-per-node=2
#SBATCH  --cpus-per-task=2
```

7.1 Commonly Used Datasets

ACAS Xu (Airborne Collision Avoidance System) dataset seems to be the staple for basic performance check and comparison, where as the MNIST and CIFAR10 datasets are for checking adversarial robustness. Additional datasets such as Drebin exists but aren't used much as of right now.

Datasets & It's Property Type

Aa Name	Property Type	Tools
ACAS Xu	I(interval)/O(interval)	Marabou, Neurify, DeepPoly, NNV

Name	Property Type	Tools
MNIST	I(perturbation range)/O(NN output)	Marabou, Neurify, DeepPoly, NNV
CIFAR10	I(perturbation range)/O(NN output)	Neurify, DeepPoly
Drebin	I(interval)/O(interval)	Neurify
Car(Dave)	I(perturbation range)/O(NN output)	Neurify

7.2 ACAS Xu Performance

Due to time constraints and limitation on the server running time, compared performance on ACAS Xu property 1, 2, 3, and 4. Below are the tables showing the number of SAT and UNSAT results and for the minority, which network returned such a result.

Property 1

Prop1	SAT	UNSAT	TIME(sec)
Reluplex(Paper)	0	41	394517
Reluplex(Server)	x	x	x
Marabou	x	x	x
MarabouDnC	0	45	7464
Neurify	0	45	227
DeepPoly	2(4_7, 4_9)	43	257

Property 2

Prop2	SAT	UNSAT	TIME(sec)
Reluplex(Paper)	35	1	82882
Reluplex(Server)	35	1	97334
Marabou	x	x	x
MarabouDnC	34	2 (3_3, 4_2)	4060
Neurify	35	1	6171
DeepPoly	32	_7, 1_8, 1_9)	609

Property 3

Prop3	SAT	UNSAT	TIME(sec)
Reluplex(Paper)	0	42	28156
Reluplex(Server)	0	42	27275
Marabou	0	42	12502
MarabouDnC	0	42	2673
Neurify	0	42	383
DeepPoly	1 (1_1)	41	241

Property 4

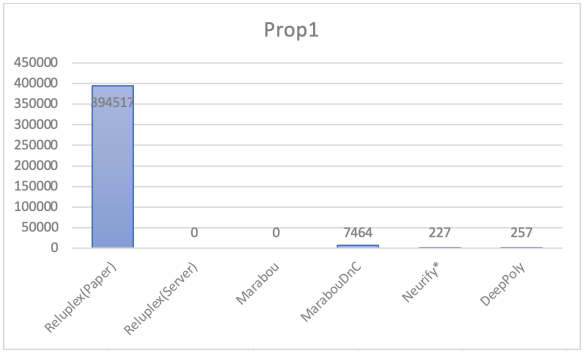
Prop4	SAT	UNSAT	TIME(sec)
Reluplex(Paper)	0	42	12475
Reluplex(Server)	0	42	10826
Marabou	0	42	3384
MarabouDnC	0	42	284
Neurify	0	42	28
DeepPoly	0	42	193

(* Verifying Property 1 using Reluplex took too much time (TIMEOUT))

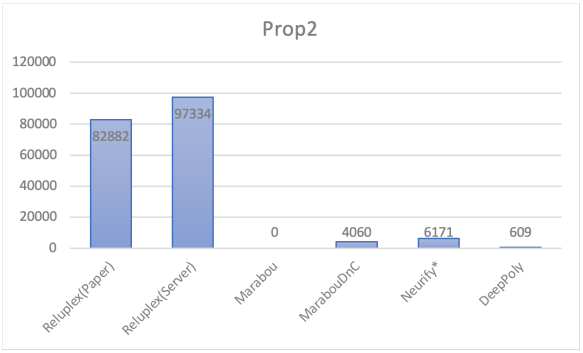
(* Marabou results for Property 2 currently being checked)

MarabouDnC and DeepPoly returned different results compared to results in [4] and [9]. Possible cause might be a script type or a misstep in the installation process resulting in malfuction.

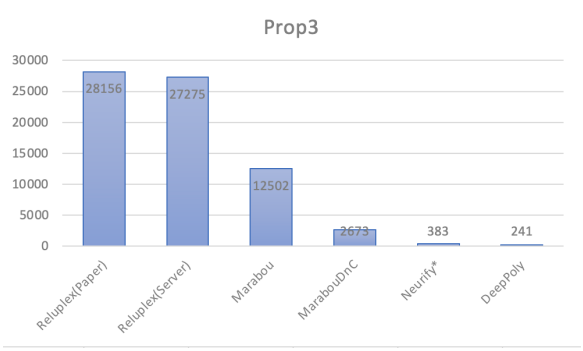
Property 1



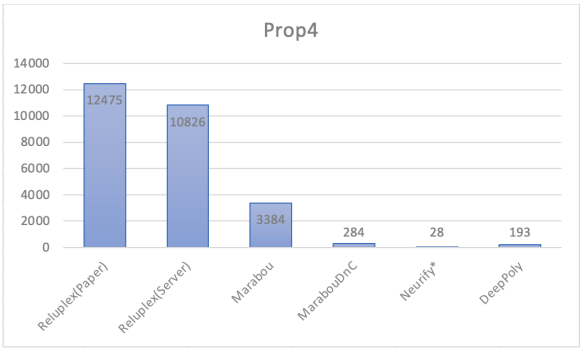
Property 2



Property 3



Property 4



References

[1] Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search

[2] Algorithms for Verifying Deep Neural Networks

[3] Accelerating Robustness Verification of Deep Neural Networks Guided by Target Label

[4] Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks

[5] The Marabou Framework for Verification and Analysis of Deep Neural Networks

[6] https://github.com/NeuralNetworkVerification/Marabou

[7] Efficient Formal Safety Analysis of Neural Networks

[8] An Abstract Domain for Certifying Neural Networks

[9] NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems

[10] https://github.com/verivital/nnv/blob/master/docs/manual.pdf