1 Example Usage

Example 1. Consider the following nonlinear control system [1]:

$$\dot{x}_0 = x_1, \quad \dot{x}_1 = ux_1^2 - x_0,$$

where u is computed from a NN controller κ that has two hidden layers, twenty neurons in each layer, and ReLU and tanh as activation functions. Given a control stepsize $\delta_c = 0.2$, we hope to verify whether the system will reach $[0, 0.2] \times [0.05, 0.3]$ from the initial set $[0.8, 0.9] \times [0.5, 0.6]$ while avoiding $[0.3, 0.8] \times [-0.1, 0.4]$ over the time interval $[0, 7]$.

The code of Dynamics (*.cpp* file) and the parameters of the NN controller, together with annotations are shown in Listing 1 and 2, respectively. Users can use them as templates for different examples.

```
...
// Declaration of the state and input variables.
unsigned int numVars = 3;
int x0_id = stateVars.declareVar("x0");
int x1_id = stateVars.declareVar("x1");
int u_id = stateVars.declareVar("u");
...
// Define the continuous dynamics.
Expression_AST<Real> deriv_x0("x1");
Expression_AST<Real> deriv_x1("u*x1^2-x0");
Expression_AST<Real> deriv_u("0");
...
// Define initial state set.
Interval init_x0(0.8, 0.9), init_x1(0.5, 0.6), init_u(0);
...
// Define unsafe set
Constraint constraint_unsafe_1("x0 - 0.8");
Constraint constraint_unsafe_2("-x0 + 0.3");
Constraint constraint_unsafe_3("x1 - 0.4");
Constraint constraint_unsafe_4("-x1 - 0.1");
...
// Define target set
Constraint constraint_target_1("x0 - 0.2");
Constraint constraint_target_2("-x0");
Constraint constraint_target_3("x1 - 0.3");
Constraint constraint_target_4("-x1 + 0.05");
...
// Define degree bound of approximation polynomials.
char const *degree_bound = "[3, 3]";
// Define the file name of the neural network controller.
char const *neural_network = "nn_1_relu_tanh_origin";
// Define the timestep T.
int T = 35
...
```

Listing 1: Dynamics and reach-avoid specification.

```
2 // number of inputs
1 // number of outputs
2 // number of hidden layers
20 // number of nodes in the first hidden layer
20 // number of nodes in the second hidden layer
// Weights (W) and biases (b) of the first hidden layer
-0.0073867239989340305 // W_0_0
0.0390695296227932 // W_0_1
0.20118312537670135 // b_0
0.015087026171386242 // W_1_0
```

```
-0.07999959588050842 // W_1_1
0.40419483184814453 // b_1
...
0.026221774518489838 // W_19_0
-0.13843803107738495 // W_19_1
0.29650038480758667 // b_20
// Weights (W) and biases (b) of the first hidden layer
...
0 // Offset of the neural network
4 // Scala of the neural network
```

Listing 2: Neural-network controller.

Then we can verify the NNCS with the following command in terminal:

```
# input args: network name and error bound for sampling-based error analysis
$ ./example.sh nn_1_relu_tanh_origin 0.005
```

The computed flowpipes are shown in Fig. 1a. The output file of ReachNN* shows:

```
Unknown  //Verification result
Max Error: 0.058277   //Sampling error precision ε̄
Running Time: 12718 seconds    // Total computation time
```

Here "Unknown" means that the computed flowpipes intersect with (and are not contained entirely in) the avoid set or the target set. For the systems with only reach specification, the output file of ReachNN* will be as follows.

```
Verification Result: Yes(NUM_STEPS)/No(NUM_STEPS)/Unknown(NUM_STEPS)
Max Error: VALUE OF MAX ERROR
Running Time: VALUE OF RUNTIME seconds
```

Here, "Yes" means that the target set contains the overapproximation of the reachable set. In other words, every trajectory of the system is guaranteed to reach the target set at time T. "No" means that the target set and the overapproximation of the reachable set are mutually exclusive. Every trajectory of the system will fall outside of the target set. "Unknown" means that the target set intersects with the overapproximation of the reachable set. It is unknown whether every system trajectory will fall inside the target set.

Next, we retrain a new NN controller named *nn_1_relu_tanh_retrained* for the same activation function, a target Lipschitz constant of 0 and a regression error tolerance of 0.4:

```
# input args: original network name, distilled network name,
# activation function, target Lipschitz constant,
# regression error bounds, scalar and offset
$ ./example.sh nn_1_relu_tanh_origin nn_1_relu_tanh_retrained RELU_TANH 0 0.4
     4 0
```

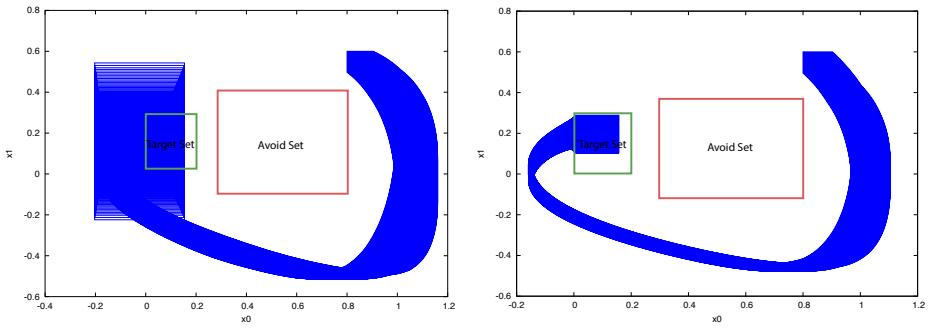
We then can verify the system with the new NN controller:

```
# input args: network name and error bound for sampling-based error analysis
$ ./example.sh nn_1_relu_tanh_retrained 0.005
```

The computed flowpipes are shown in Fig. 1b. The new output file of ReachNN* shows:

```
Safe and reachable!  //Verification result of new NN controller
Max Error: 0.019958   //Sampling error precision ε̄
Running Time: 103 seconds    // Total computation time
```

We can see that the new NN controller can be verified to satisfy the reach-avoid specification. The verification process is much more efficient – 103 seconds compared to 12718 seconds for verifying the original NNCS (123× faster).



(a) Before KD (b) After KD

Fig. 1: Reachability analysis results: Red lines represent boundaries of the obstacles and form the avoid set. Green rectangle represents the target region. Blue rectangle represents the computed flowpipes.

References

1. Gallestey, E., Hokayem, P.: Lecture notes in nonlinear systems and control (2019)