Correctness Verification of Neural Networks

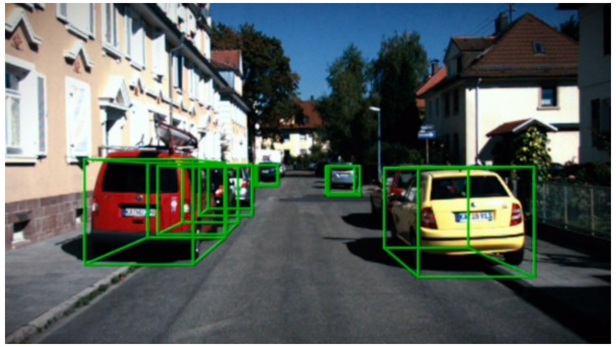
Yichen Yang & Martin Rinard

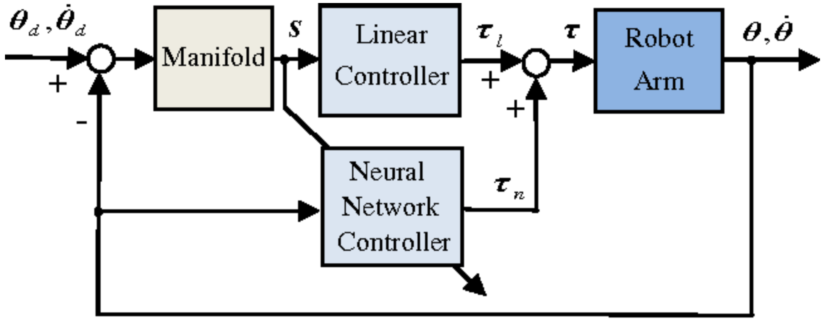
MIT Computer Science and Artificial Intelligence Laboratory

Use of neural networks

$$x \xrightarrow{f} y$$

x: input   f: neural network   y: output





Others...

Sensing   Controller

Source: 1. https://news.cornell.edu/stories/2019/04/new-way-see-objects-accelerates-future-self-driving-cars
2. https://www.semanticscholar.org/paper/A-Neural-Network-Controller-for-Trajectory-Control-Jiang-Ishida/9fb758b226b9bb654023d343ea1575e339a3034d/figure/0

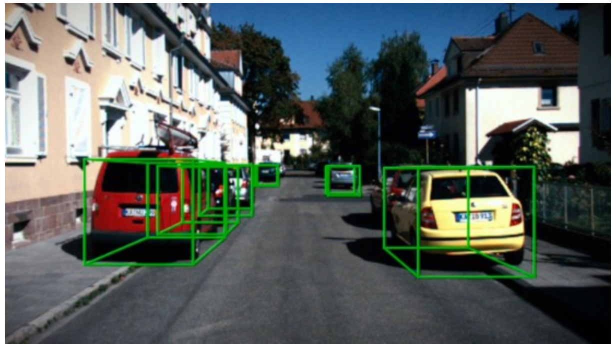
Use of neural networks

$$x \xrightarrow{f} y$$

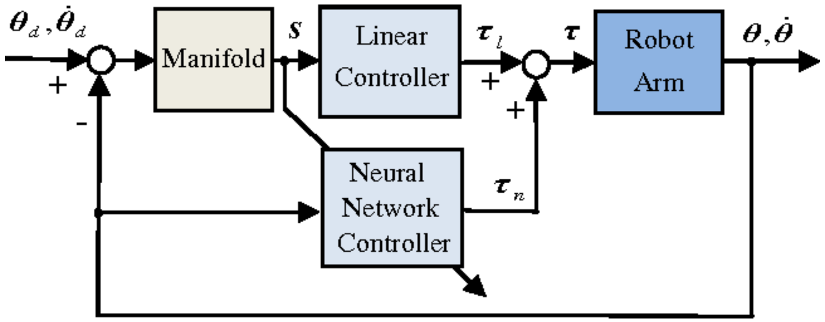
x: input

f: neural network

y: output



Sensing

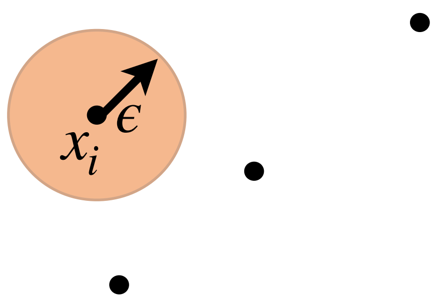


Controller

Others...

Source: 1. https://news.cornell.edu/stories/2019/04/new-way-see-objects-accelerates-future-self-driving-cars
2. https://www.semanticscholar.org/paper/A-Neural-Network-Controller-for-Trajectory-Control-Jiang-Ishida/9fb758b226b9bb654023d343ea1575e339a3034d/figure/0

Verification and Robustness



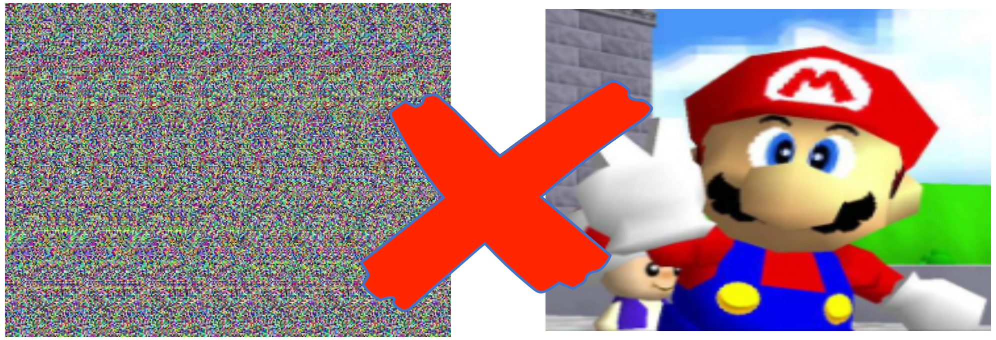
- Given $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- Verify that output does not change in the neighborhood around each input
- Robustness against l_p-norm bounded perturbation:

$$||x - x_i||_p \leq \epsilon \implies f(x) = f(x_i)$$

- Only verify neighborhood around each labeled point.
- Only verify the output is ***stable***, not necessarily ***correct***
- So robust verification is not correctness verification

We need a **specification**

- What should a specification provide?
- Precondition: identifies **feasible inputs** for which network should be expected to give correct answer



- Postcondition: correct output for each feasible input

Source: 1. https://www.newscientist.com/article/mg23230970-200-playing-can-teach-autonomous-cars-how-to-drive/
2. https://www.flickr.com/photos/nbscloset/3313647292
3. https://www.gamasutra.com/view/news/320213/How_classic_games_make_smart_use_of_random_number_generation.php

How About Specification for Sensing Applications?

- Not feasible in general with only the $x \xrightarrow{f} y$ setup - consider a vision task
- Need to logically identify all feasible input images
- Need to logically specify correct output for each feasible input image
- People don’t know how to do this
  (which is one reason we use neural networks for such tasks)
- So we need to bring something more!

Key Insight

- Introducing **state space** and **observation process**
- Example: a road, a camera taking pictures of the road, estimate position of camera given image

Latent state of the world s Observation Process g Input x

- Camera offset: …
- Camera facing angle: …
- road width: …
- …

Camera Imaging Process →

- Sensing task is typically to recover some attribute of the world, which is encoded in s. Denote this attribute as $\lambda(s)$, ground truth function (typically trivial to compute)

Source: 1. https://openai.com/content/images/2017/05/stacking_demo.gif

Now we can give specification

$$s \xrightarrow{g} x \xrightarrow{f} y$$

- State space $\mathcal{S}$: the space of all states of the world that the network is expected to work in.
- Precondition: feasible input space $\tilde{\mathcal{X}} = \{x \mid \exists s \in \mathcal{S}, x \in g(s)\}$
- Postcondition: the correct output is given by $\lambda(s)$

Correctness Verification

$$s \xrightarrow{g} x \xrightarrow{f} y$$

- Correctness: $\forall s \in \mathcal{S}, \forall x \in g(s), f(x) = \lambda(s)$
- For regression problems, neural networks won't give exactly correct predictions
- (Approximate) correctness:

$$\forall s \in \mathcal{S}, \forall x \in g(s), |f(x) - \lambda(s)| \leq \epsilon$$

- Can be other distance metric depending on how you want to measure error

Correctness Verification

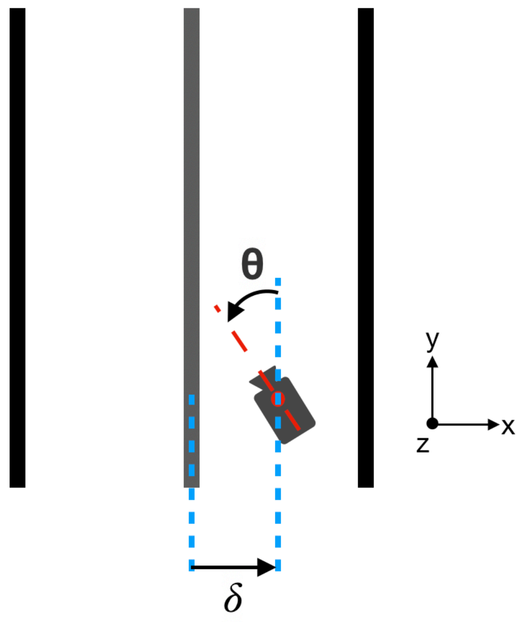
$$s \xrightarrow{g} x \xrightarrow{f} y$$

- Problem formulation (regression): given a trained network f, a specification by $\mathcal{S}$, g, λ, find a bound on the maximum error the network can make with respect to the specification

Find bound on $\max_{s \in \mathcal{S}, x \in g(s)} |f(x) - \lambda(s)|$

Example

- Setup: a camera takes picture of a road
- Camera can vary its horizontal offset and viewing angle.
- A neural network takes the picture as input, predict the camera position (δ, θ)



$$s \xrightarrow{\;g\;} x \xrightarrow{\;f\;} y$$

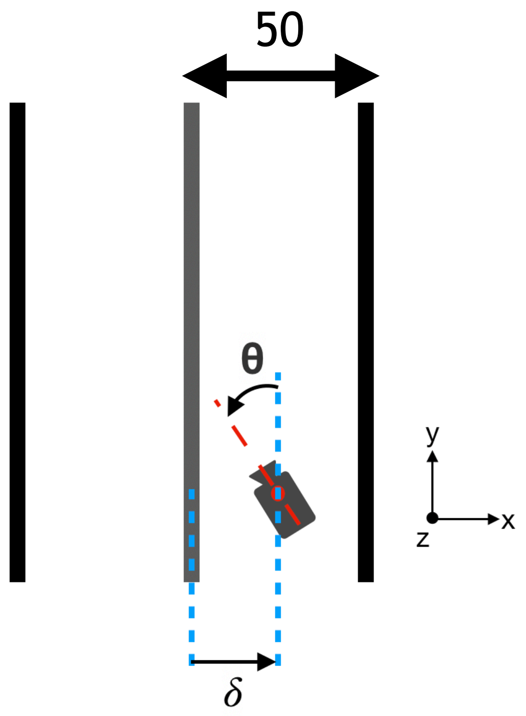
(δ, θ) —Camera Imaging Process→

—Neural Network→ (δ^*, θ^*)

Example

- The neural network is designed to work for $\delta \in [-40, 40], \theta \in [-60^\circ, 60^\circ]$
- So state space $\mathcal{S} = \{s_{\delta,\theta} | \delta \in [-40, 40], \theta \in [-60^\circ, 60^\circ]\}$
- Feasible input space $\tilde{\mathcal{X}} = \{x | \exists s \in \mathcal{S}, x \in g(s)\}$
- Problem of correctness verification:

  Find bound on $\max(|\delta - \delta^*|), \max(|\theta - \theta^*|)$ over all images that can be taken within $\delta \in [-40, 40], \theta \in [-60^\circ, 60^\circ]$



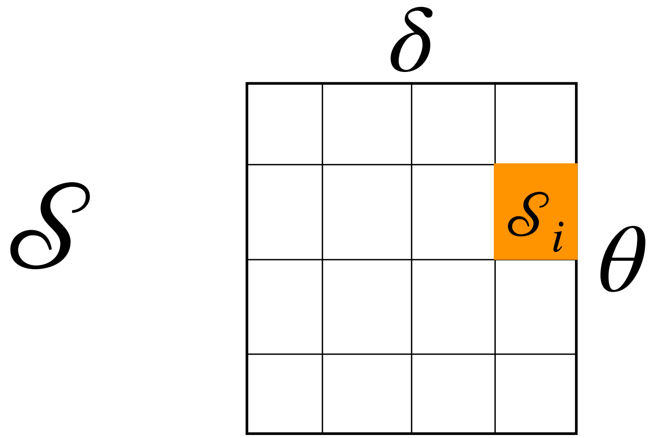
How to solve?

- State space $\mathcal{S}$ can in general be continuous and contains infinite number of states (as is in the example)
- Cannot enumerate each state
- Idea: finitize the space into ***tiles*** and compute error bound for each tile

⟶ ***Tiler***

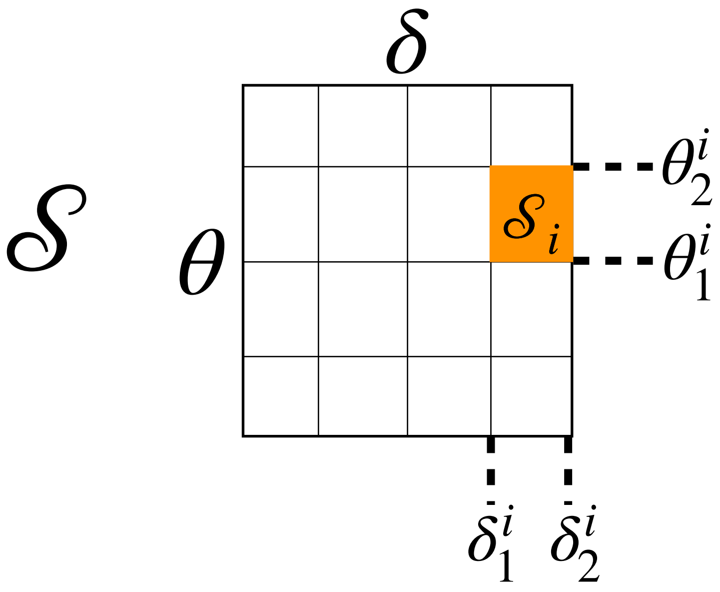
Tiler

- Step 1: Divide the state space $\mathcal{S}$ into local regions $\{\mathcal{S}_i\}$ such that $\cup_i \mathcal{S}_i = \mathcal{S}$



Tiler

- Step 2: For each $\mathcal{S}_i$, compute the ground truth bound $[l_i, u_i]$, such that $\forall s \in \mathcal{S}_i, l_i \leq \lambda(s) \leq u_i$

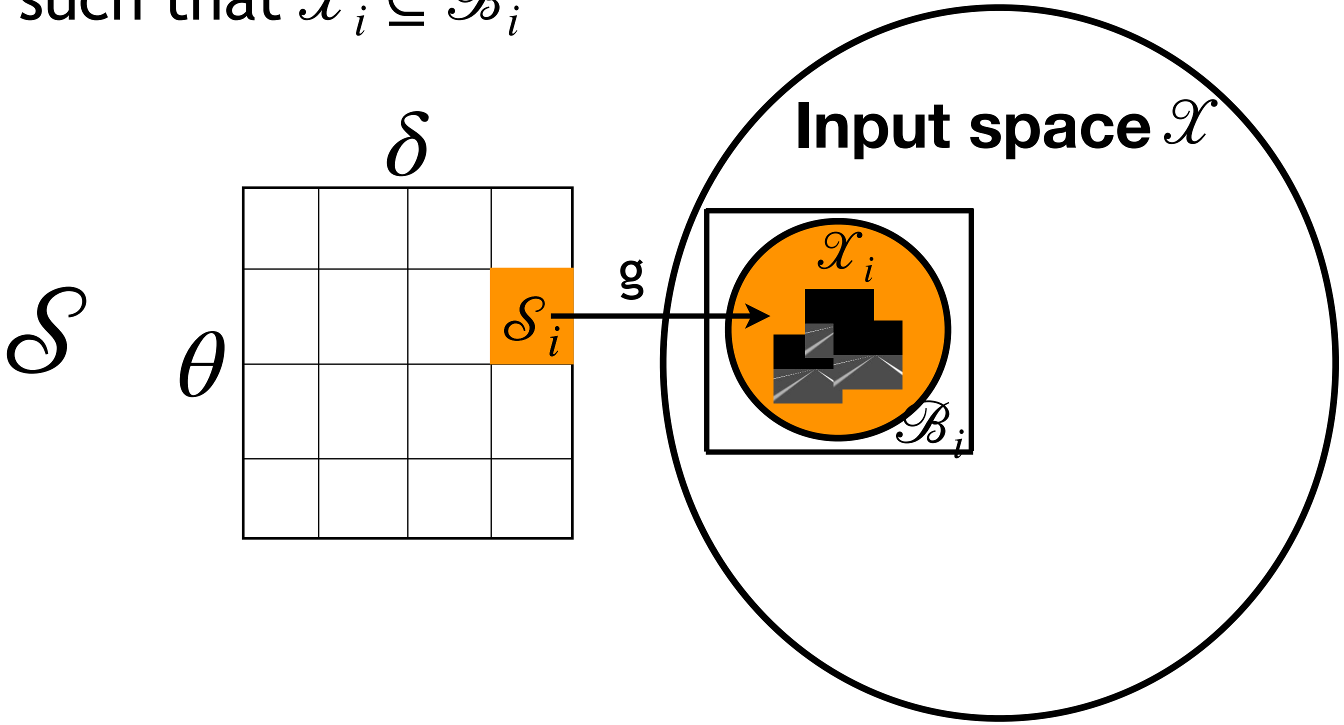


Ground truth bound for $\mathcal{S}_i$:

- For δ prediction: $[\delta_1^i, \delta_2^i]$
- For θ prediction: $[\theta_1^i, \theta_2^i]$

Tiler

- Each $\mathcal{S}_i$ is mapped to a tile in input space by g: $\mathcal{X}_i = \{x \mid x \in g(s), s \in \mathcal{S}_i\}$
- Step 3: Using $\mathcal{S}_i$ and g, compute a bounding box $\mathcal{B}_i$ for each input tile $\mathcal{X}_i$ such that $\mathcal{X}_i \subseteq \mathcal{B}_i$

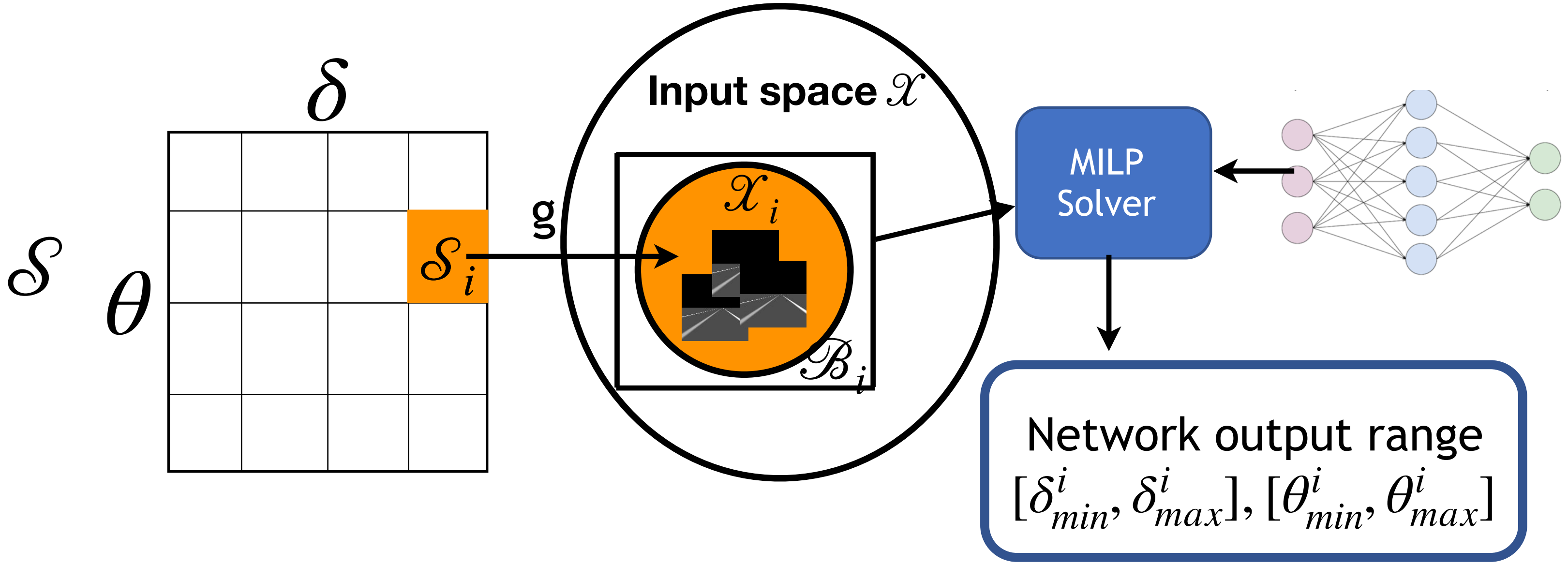


For each pixel, compute the range of values it can take when s varies in $\mathcal{S}_i$.

This gives a l_∞-norm ball $\mathcal{B}_i$ in the input space that encapsulate $\mathcal{X}_i$

Tiler

- Step 4: Given network f and bounding boxes $\{\mathscr{B}_i\}$, use a compatible technique to solve for the network output ranges $\{[l'_i, u'_i]\}$, satisfying: $\forall x \in \mathscr{B}_i, l'_i \leq f(x) \leq u'_i$

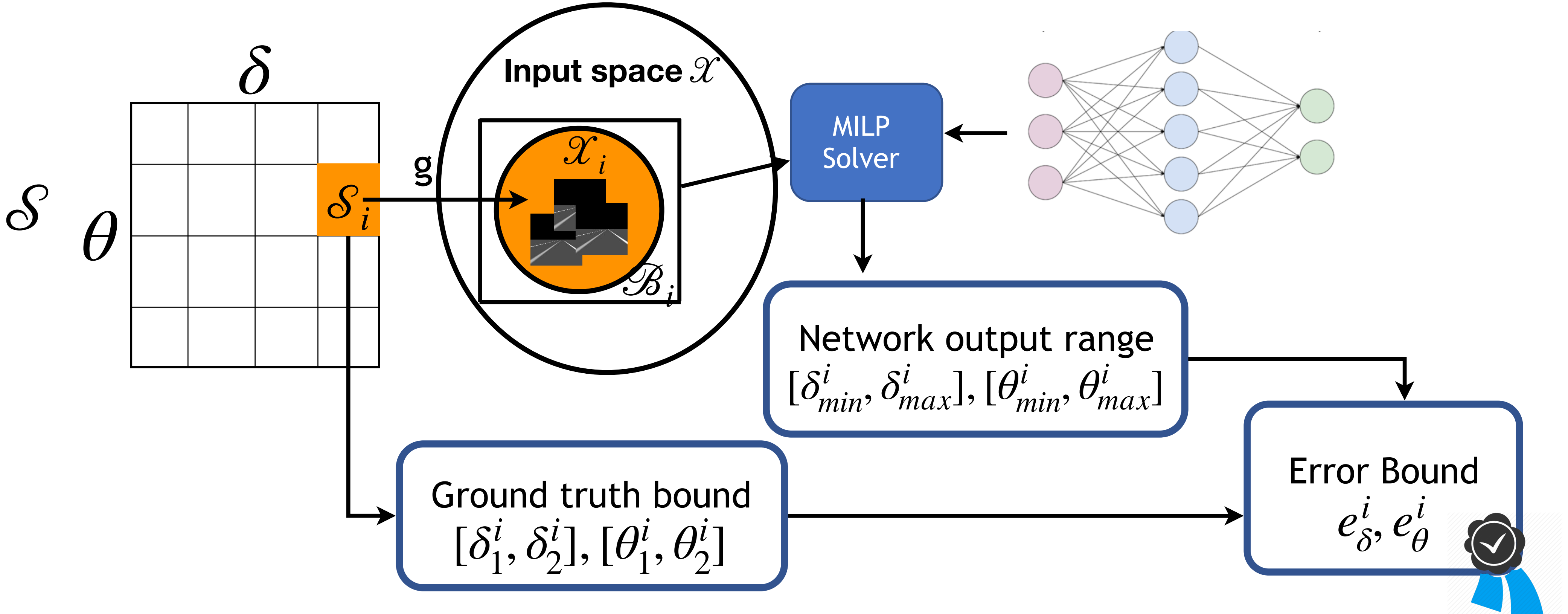


Standard techniques to solve network output range given input constraints:

- MILP
- Convex relaxation
- Duality
- Abstract interpretation

Tiler

- Step 5: For each tile, use the ground truth bound (l_i, u_i) and network output bound (l'_i, u'_i) to compute the error bound: $e_i = \max(u'_i - l_i, u_i - l'_i)$
- This gives the upper bound on prediction error for all $s \in \mathcal{S}_i$



Tiler

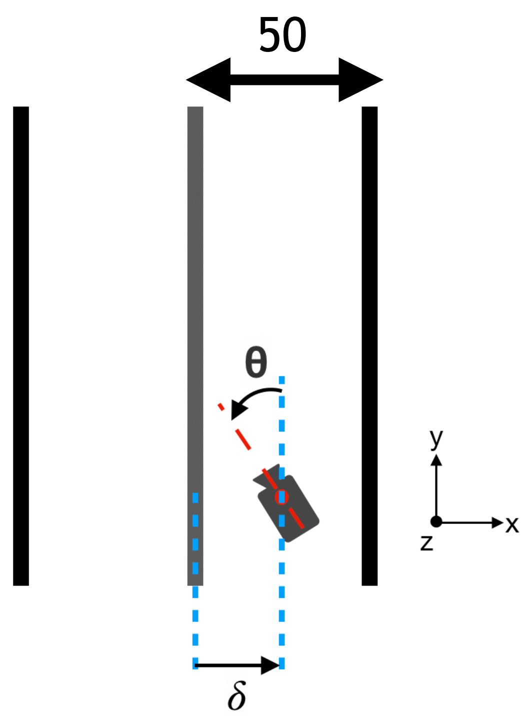
Algorithm 1 Tiler (for regression)

Input: $\mathcal{S}, g, \lambda, f$
Output: $e_{\text{global}}, \{e_i\}, \{\mathcal{B}_i\}$

procedure $\text{TILER}(\mathcal{S}, g, \lambda, f)$
    $\{\mathcal{S}_i\} \leftarrow \text{DIVIDESTATESPACE}(\mathcal{S})$ ▷ Step 1
    for each $\mathcal{S}_i$ **do**
        $(l_i, u_i) \leftarrow \text{GETGROUNDTRUTHBOUND}(\mathcal{S}_i, \lambda)$ ▷ Step 2
        $\mathcal{B}_i \leftarrow \text{GETBOUNDINGBOX}(\mathcal{S}_i, g)$ ▷ Step 3
        $(l'_i, u'_i) \leftarrow \text{SOLVER}(f, \mathcal{B}_i)$ ▷ Step 4
        $e_i \leftarrow \max(u'_i - l_i, u_i - l'_i)$ ▷ Step 5
    end for
    $e_{\text{global}} \leftarrow \max(\{e_i\})$ ▷ Step 5
    return $e_{\text{global}}, \{e_i\}, \{\mathcal{B}_i\}$ ▷ $\{e_i\}, \{\mathcal{B}_i\}$ can be used later to compute $e_{\text{local}}(x)$
end procedure

Case Study

- Position measurement from road scene
- Neural network: 2 conv layers with 16 and 32 filters respectively + a fully connected layer with 100 units. Output layer is a linear layer with 2 output nodes. ReLU activation.
- Trained to work for $\delta \in [-40, 40], \theta \in [-60^\circ, 60^\circ]$
- Apply Tiler:
  - Divide the state space into grid with cell size 0.1 (for both δ and θ)
  - For solving network output range (Step 4), we use MILP method by *Tjeng et.al. 2017.*



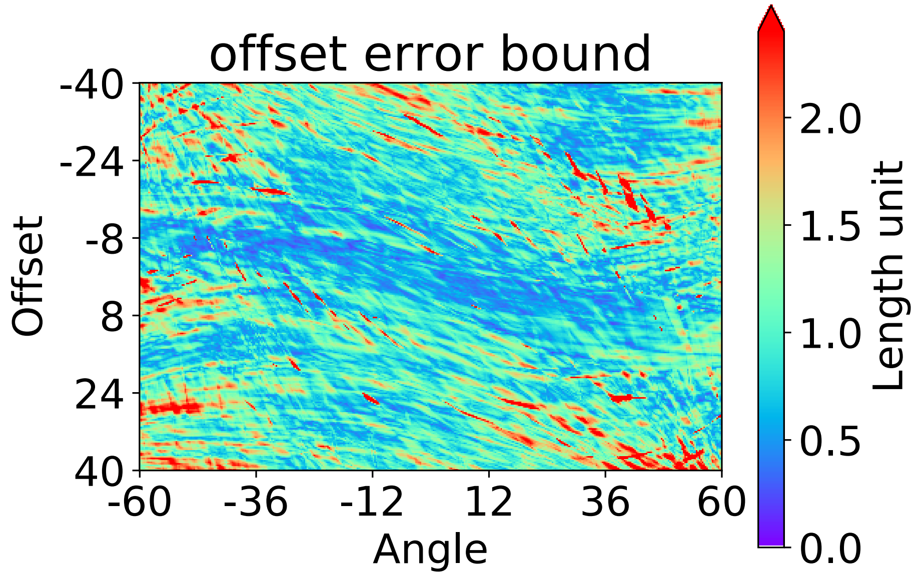
Error Bound

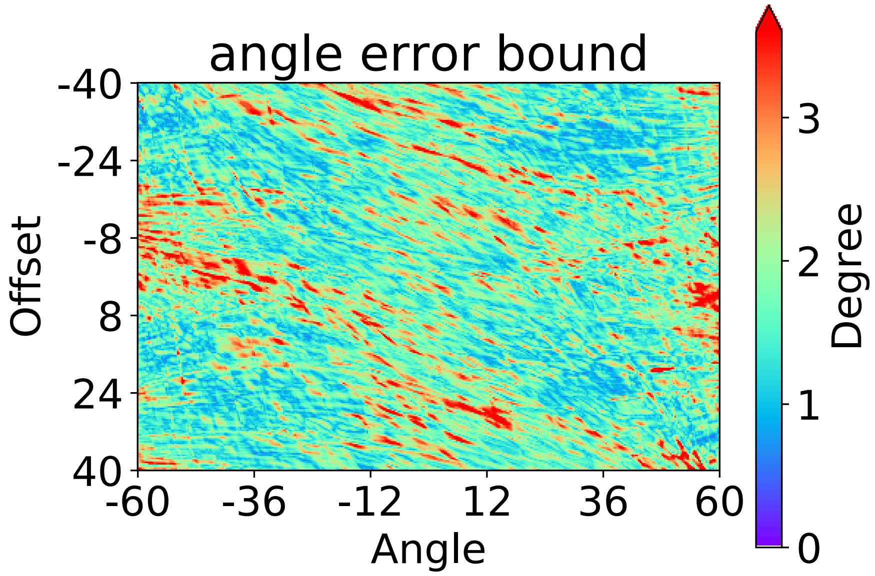
- Global error bounds:
  - For δ, 12.66 (15.8% of the measurement range)
  - For θ, 7.13° (5.94% of the measurement range)

- We have verified that the network will not make errors greater than these values for all input images that it is expected to work on!

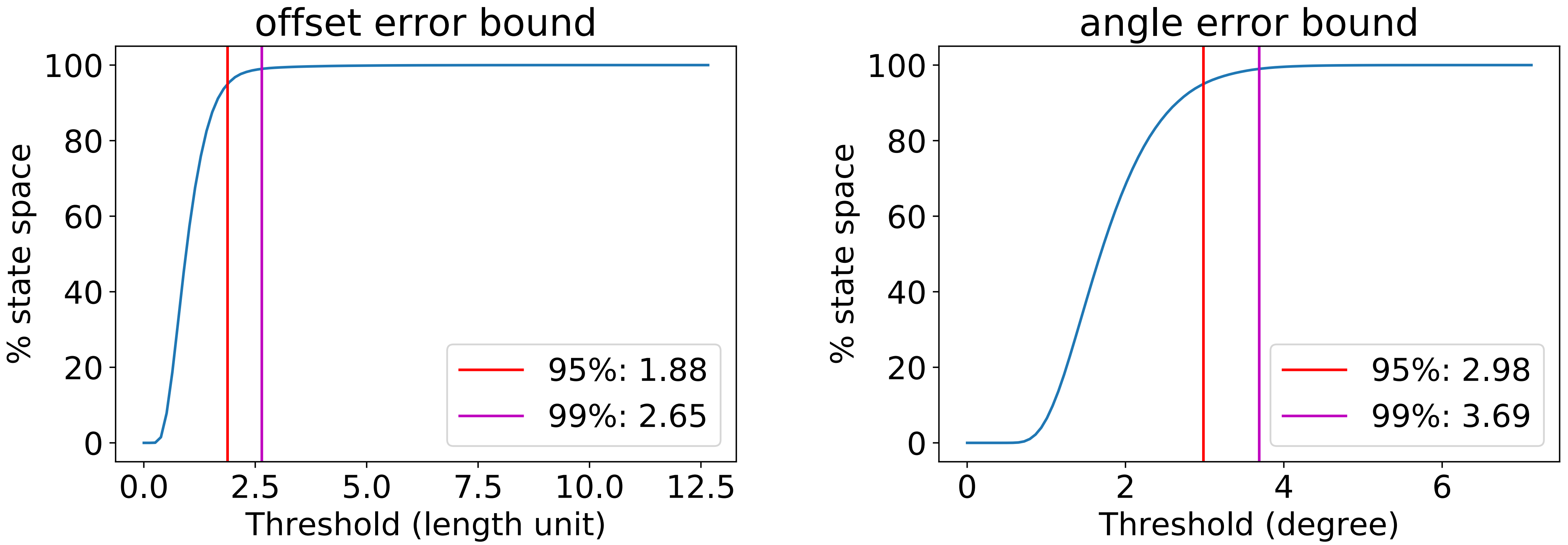
Error Bound Landscape

- We can view how the error bounds varies across the state space:





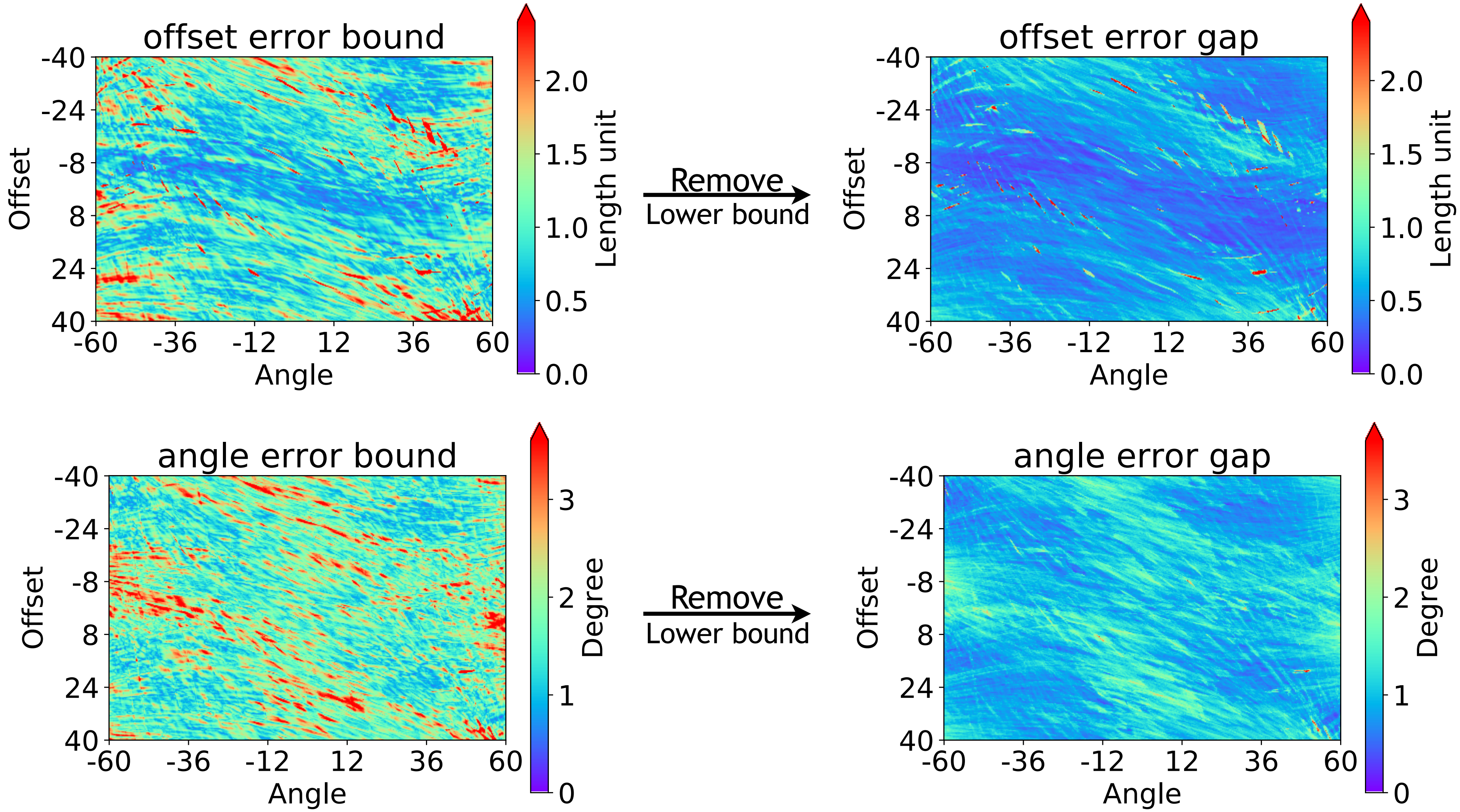
Error Bound Landscape



How tight are the error bounds?

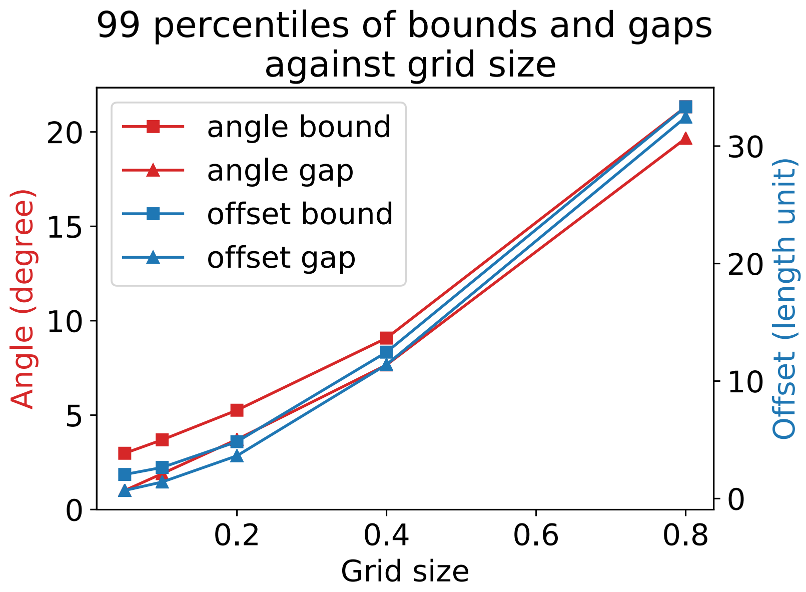
- Maybe Tiler gives large error bounds, but my network is actually good?
- Sample multiple (δ, θ) within each cell $\mathcal{S}_i$ and generate input images, then take the maximum over the prediction errors of these points (empirical estimate)
- This actually gives lower bounds on the max errors for each tile
- Global error bounds:
  - For δ, upper bound (by Tiler) 12.66, lower bound (empirical) 9.12
  - For θ, upper bound (by Tiler) $7.13°$, lower bound (empirical) $4.08°$

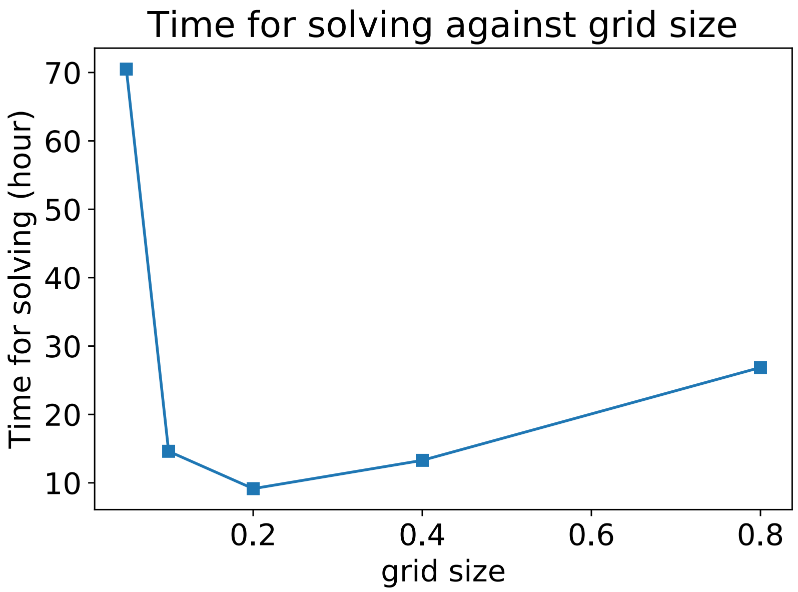
How tight are the error bounds?



Can we get even tighter error bounds?

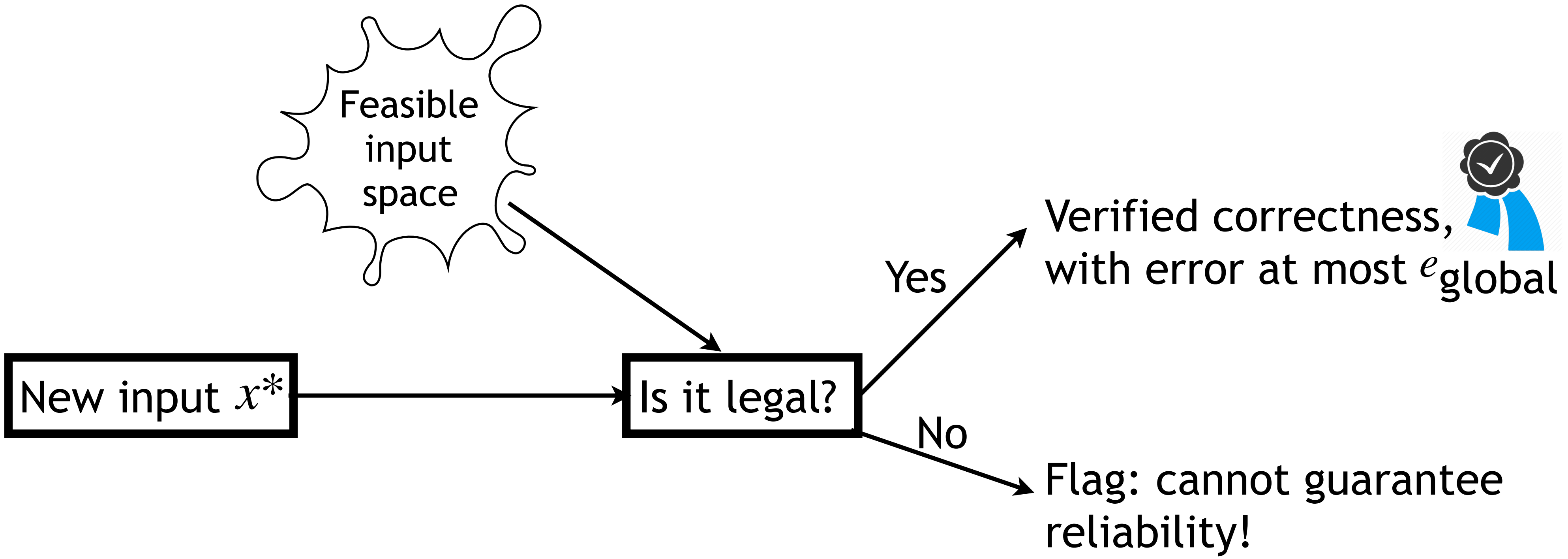
- Contributing factors to the gap:
  - Extra space $\mathscr{B}_i \backslash \mathscr{X}_i$
  - Interval arithmetic when computing error bounds $e_i = \max(u_i' - l_i, u_i - l_i')$
- Both will be reduced with finer tile size





Detecting illegal inputs

- To make the system complete, need a way to detect whether a new input is within the feasible input space or not.



Detecting illegal inputs

- Save the bounding boxes $\{\mathscr{B}_i\}$ computed in *Tiler*
- Check if the new input x^* is contained in any $\mathscr{B}_i$
  - If not, flag as illegal
  - If yes, then x^* is either
    - in the feasible input space, or
    - close to points in the feasible input space (in terms of the size of the bounding box).
  - If size of $\mathscr{B}_i$ is small, it is reasonable to assume the ground truth for x^* is close to the ground truth for the feasible inputs in $\mathscr{B}_i$ (common assumption behind robustness).
  - Since the network output bound computed in Tiler is for $\mathscr{B}_i$, it applies to x^*. So we know the prediction on x^* is reliable.

Detecting illegal input in case study

- Test this detector in case study, on 3 kinds of input:
  - 1000 legal inputs — generated from $\mathcal{S}$ and g. 100% flagged as legal
  - 1000 perturbed inputs — apply per-pixel uniformly distributed random perturbation with scale 0.1. 100% flagged illegal
  - 1000 inputs from a new scene — change to a scene that the network is not designed to work for (increase road width from 50 to 60). 100% flagged illegal

Speeding up — prediction guided search

- Previously, search over all $\{\mathscr{B}_i\}$ to check containment of $x*$
- Can speed up by guiding the search with network prediction
- Prediction: $(\delta*, \theta*)$, then only need to search over $\mathscr{B}_i$'s corresponding to tiles $\mathscr{S}_i$'s that overlap with $[\delta* - e_\delta, \delta* + e_\delta]$ and $[\theta* - e_\theta, \theta* + e_\theta]$
  - If $x*$ is legal, then the ground truth must be within those ranges, so this guided search will find the $\mathscr{B}_i$ that contains $x*$
  - If $x*$ is illegal (not in any $\mathscr{B}_i$), then this guided search won't find a $\mathscr{B}_i$ containing $x*$, will flag illegal
- Naive search: 1.138s/input; guided search: 0.069s/input. 16x speed up

Summary

- Use state space and observation process to provide specification
  - Specifies all feasible inputs for which the network is expected to work on
  - Specifies correct output for each input
- By finitizing state and input spaces into tiles, we can do correctness verification, verifying the max error the network can make for all feasible inputs
- This framework also enables detecting whether an input is legal or not

Reference

Evaluating Robustness of Neural Networks with Mixed Integer Programming, Vincent Tjeng, Kai Xiao, Russ Tedrake, ICLR 2019

Camera Imaging Process