6.S979 Topics in Deployable ML

Verification of Deep Learning Models

Yichen Yang & Martin Rinard
MIT CSAIL

Back in the old, happy days…

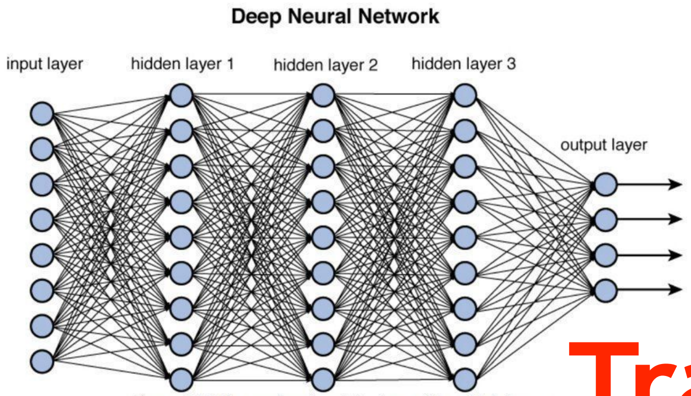
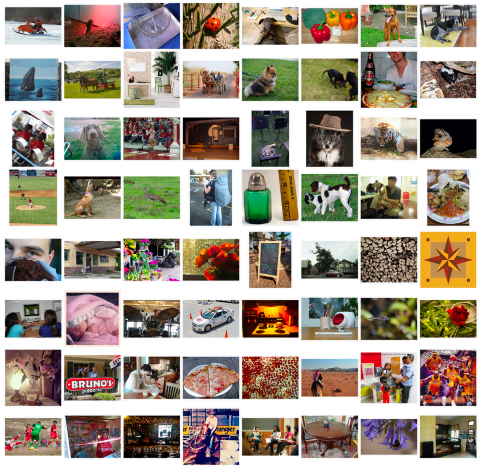


Figure 12.2 Deep network architecture with multiple layers.

Train

Test



95%

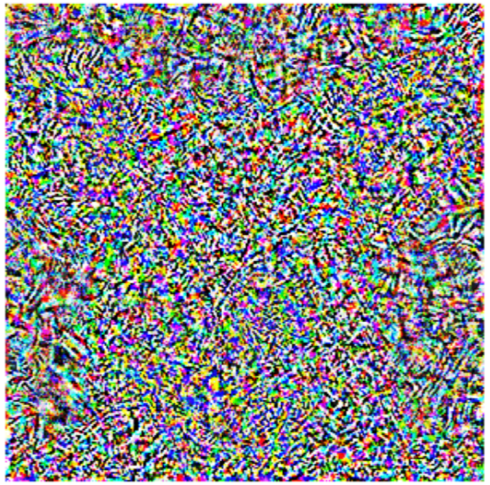
Source: 1. https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964
2. https://medium.com/syncedreview/sensetime-trains-imagenet-alexnet-in-record-1-5-minutes-e944ab049b2c
3. https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/

But things are not that easy…

“pig” (91%)

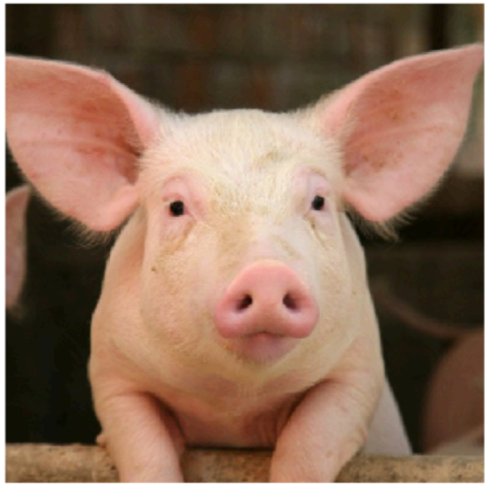
+ 0.005 x

noise (NOT random)



=

“airliner” (99%)



Acknowledgement: slides adapted from Aleksander Madry

But things are not that easy…

- Traditional testing is not enough for judging whether a trained neural network is reliable or not

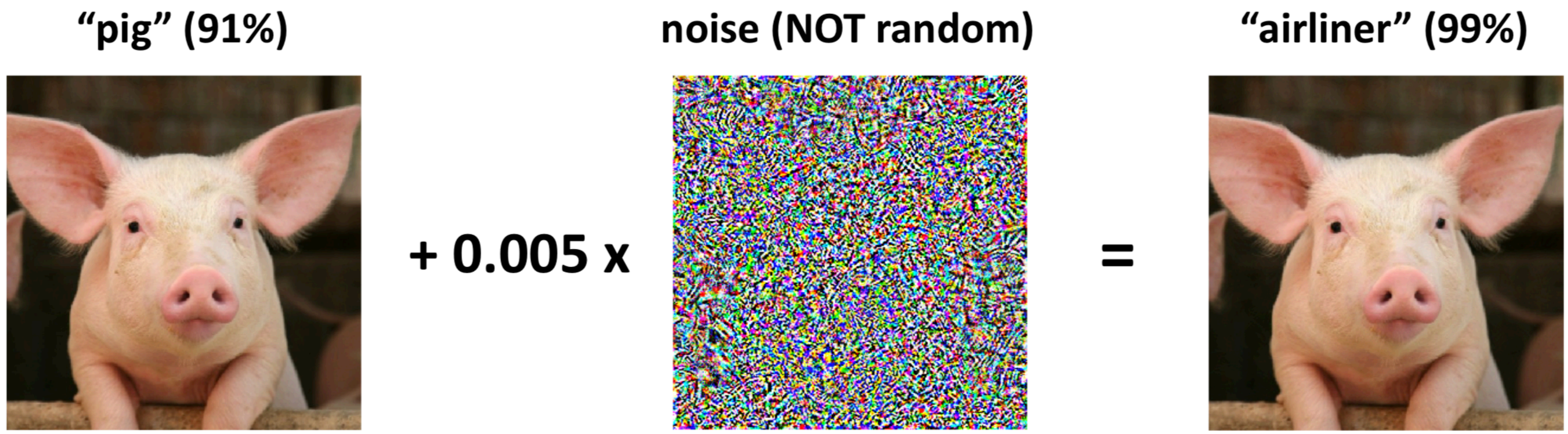
But things are not that easy…

- Traditional testing is not enough for judging whether a trained neural network is reliable or not

- How can we be ensured that the network is reliable and doing what it should do?

How to ensure?

- To formally ensure that the network is doing what it should do, we first need to specify what it should do



How to ensure?

- To formally ensure that the network is doing what it should do, we first need to specify what it should do
- Then we can verify that the network satisfies the specification

How to ensure?

- To formally ensure that the network is doing what it should do, we first need to specify what it should do
- Then we can verify that the network satisfies the specification
- Typically, we want a neural network to learn and implement a function
- Let's first consider a much simpler function

Simple Example

- Square root function: $f(x) = \sqrt{x}$
- First, we need to give a **specification** (what the implementation should achieve)
  - Precondition: $x \geq 0$
  - Postcondition: $f(x) = \sqrt{x}$?

Simple Example

- Square root function: $f(x) = \sqrt{x}$
- First, we need to give a **specification** (what the implementation should achieve)
  - Precondition: $x \geq 0$
  - Postcondition: $\cancel{f(x) = \sqrt{x}}$?

$$(1 - \epsilon)x \leq f(x)^2 \leq (1 + \epsilon)x, \;\; f(x) \geq 0$$

- Then, we can verify (prove) if an implementation satisfies this specification

Simple Example

- Square root function: $f(x) = \sqrt{x}$
- First, we need to give a **specification** (what the implementation should achieve)
  - Precondition: $x \geq 0$
  - Postcondition: $\cancel{f(x) = \sqrt{x}}$?

    $$(1 - \epsilon)x \leq f(x)^2 \leq (1 + \epsilon)x, \;\; f(x) \geq 0$$

- Then, we can **verify** (prove) if an implementation satisfies this specification

For neural networks…

- Denote the neural network as function $f(\,\cdot\,;\theta) : \mathbb{R}^m \to \mathbb{R}^n$
- General form of a specification:
    - For all inputs in $\underbrace{\mathscr{C} \subseteq \mathbb{R}^m}_{\text{Precondition}}$, some $\underbrace{\text{property P holds}}_{\text{Postcondition}}$

Robustness (under l_p-norm bounded perturbation)

- Given NN $f(\cdot\,;\theta) : \mathbb{R}^m \to \mathbb{R}^n$ for classification (output pre-softmax score), and a labeled point $(x, \lambda(x))$
- Precondition: $\mathscr{C} = \{x' \mid \|x' - x\|_p \leq \epsilon\}$
- Postcondition: argmax$_i f_i(x'; \theta) = \lambda(x)$
- If we can verify it, then no adversarial example exists around this point
- How to verify?

Search for adversarial example?

- Try using better and better adversarial attacks to search for adversarial example
- Only solves part of the problem:
  - If we found an adversarial example, we know the model is not robust
  - But if we can’t find one, we are still not sure

Let's take a look at the goal again…

- Verification goal:

$$\forall x' \text{ s.t. } \|x' - x\|_p \leq \epsilon, \ \text{argmax}_i f_i(x'; \theta) = \lambda(x)$$

- We can rewrite the postcondition:

$$\forall x' \text{ s.t. } \|x' - x\|_p \leq \epsilon, \ \forall k \in \{1,2,\ldots,n\} \setminus \{\lambda(x)\}, f_{\lambda(x)}(x'; \theta) - f_k(x'; \theta) > 0$$

$$\Downarrow$$

$$\forall k \in \{1,2,\ldots,n\} \setminus \{\lambda(x)\}, \ \underbrace{\min_{\{x' \mid \|x'-x\|_p \leq \epsilon\}} \left(f_{\lambda(x)}(x'; \theta) - f_k(x'; \theta) \right)}_{\text{Constrained optimization problem!}} > 0$$

Verification as solving constrained optimization problem

- For each $k \in \{1,2,\ldots,n\} \backslash \{\lambda(x)\}$, solve

$$\min_{\{x' \mid \|x'-x\|_p \leq \epsilon\}} \left(f_{\lambda(x)}(x';\theta) - f_k(x';\theta)\right)$$

- Consider a l-layer feed forward network

$$\begin{aligned}
&\hat{z}_{i+1} = W_i z_i + b_i, \;\; i = 1,\ldots,l-1 \\
&z_i = h(\hat{z}_i), \;\; i = 1,\ldots,l-1 \\
&z_1 = x \\
&f(x;\theta) = \hat{z}_l
\end{aligned}$$

Verification as solving constrained optimization problem

$$\min \left(f_{\lambda(x)}(x'; \theta) - f_k(x'; \theta) \right)$$

subject to

$$\hat{z}_{i+1} = W_i z_i + b_i, \ \ i = 1, \ldots, l-1$$

$$z_i = h(\hat{z}_i), \ \ i = 1, \ldots, l-1$$

$$z_1 = x'$$

$$f(x'; \theta) = \hat{z}_l$$

$$\|x' - x\|_p \le \epsilon$$

- Solve this constrained optimization problem for every k
- If all optimized objectives >0, then verified

How to solve?

$$\min \left(f_{\lambda(x)}(x'; \theta) - f_k(x'; \theta) \right)$$

subject to
$$\hat{z}_{i+1} = W_i z_i + b_i, \;\; i = 1,\ldots,l-1$$
$$z_i = h(\hat{z}_i), \;\; i = 1,\ldots,l-1$$
$$z_1 = x'$$
$$f(x'; \theta) = \hat{z}_l$$
$$\|x' - x\|_p \le \epsilon$$

- Need a way to deal with nonlinearity
- Major types of approaches:
  - Mixed integer linear program (MILP)
  - Convex relaxation
  - Duality

Mixed Integer Linear Program (MILP)
[Tjeng Xiao Tedrake '18]

- Formulate the optimization problem as MILP (only linear and integer constraints)
- Works for piecewise-linear networks (ReLU, max pooling), and input region $\mathscr{C}$ needs to be a set of polyhedra (l_1, l_∞ norm).

$$\min \left(f_{\lambda(x)}(x';\theta) - f_k(x';\theta)\right)$$

subject to

$$\hat{z}_{i+1} = W_i z_i + b_i, \ \ i = 1,\ldots,l-1$$

$$z_i = h(\hat{z}_i), \ \ i = 1,\ldots,l-1$$

$$z_1 = x'$$

$$f(x';\theta) = \hat{z}_l$$

$$\|x' - x\|_p \le \epsilon \xrightarrow[l_\infty]{} \forall i : -\epsilon \le (x'-x)_i \le \epsilon$$

Formulating ReLU

- Express $z = \max(\hat{z}, 0)$ as integer and linear constraints.
- Assume we have obtained a (potentially loose) bound on $\hat{z}$ (we will talk about how to obtain this later): $l \leq \hat{z} \leq u$

$$
z = \max(\hat{z}, 0) \quad \Longrightarrow \quad
\begin{array}{l}
\left.\begin{array}{l}
\text{if } u \leq 0,\ z = 0 \\
\text{else if } l \geq 0,\ z = \hat{z}
\end{array}\right\} \text{Stable} \\
\text{else} \quad \begin{array}{l}
z \leq \hat{z} - l(1 - a) \\
z \geq \hat{z} \\
z \leq u \cdot a \\
z \geq 0 \\
a \in \{0,1\}
\end{array}
\end{array}
$$

Solving MILP

- With the problem formulated as MILP, we can use off-the-shelf solvers to solve it (CPLEX, Gurobi, etc)
- Solving time heavily affected by the number of integer variables, because we need to do combinatorial search on them
- Therefore, a key to efficient solving is having tight bounds (l,u) on pre-ReLU activations.

Bound computation

- Fast, but loose: interval arithmetic (IA)
  - Propagate bounds layer by layer, bounds on this layer only depend on bounds of the previous layer
  - E.g. $y = -2x_1 + 3x_2 + 4x_3$, $l_i \le x_i \le u_i$ for i=1,2,3. Then bound for y by IA is $-2u_1 + 3l_2 + 4l_3 \le y \le -2l_1 + 3u_2 + 4u_3$
  - In general, to compute bounds on $\hat{z}_{i+1} = W_i z_i + b_i$ with $l_i \le z_i \le u_i$

$$W_i^- u_i + W_i^+ l_i + b_i \le \hat{z}_{i+1} \le W_i^- l_i + W_i^+ u_i + b_i$$

  - Not consider correlations on bounds, so loose
  - But only involves matrix operations, so fast

Bound computation

- Tight, but slow: MILP
  - Same as before, just make the objective being max/min of the pre-ReLU activations
- Combining these methods: progressive bound tightening
  - First use fast&loose methods
  - For those ReLUs that haven't proven to be stable, use tight&slow methods

MILP Summary

- Verification is complete
  - If it doesn't verify, then there exists an adversarial example and robustness doesn't hold
- But can be slow, due to integer variables
- Has limitation on input region and non-linearity (though it can already work with a lot of cases)

How to solve?

$$\min \left(f_{\lambda(x)}(x';\theta) - f_k(x';\theta)\right)$$

subject to
$$\hat{z}_{i+1} = W_i z_i + b_i,\ \ i = 1,\ldots,l-1$$
$$z_i = h(\hat{z}_i),\ \ i = 1,\ldots,l-1$$
$$z_1 = x'$$
$$f(x';\theta) = \hat{z}_l$$
$$\|x' - x\|_p \leq \epsilon$$

- Need a way to deal with nonlinearity
- Major types of approaches:
    - Mixed integer linear program (MILP)
    - Convex relaxation
    - Duality

Convex Relaxation

[Salman et.al. '19]

- Verification as solving constrained optimization problems

$$\min \left(f_{\lambda(x)}(x';\theta) - f_k(x';\theta)\right)$$

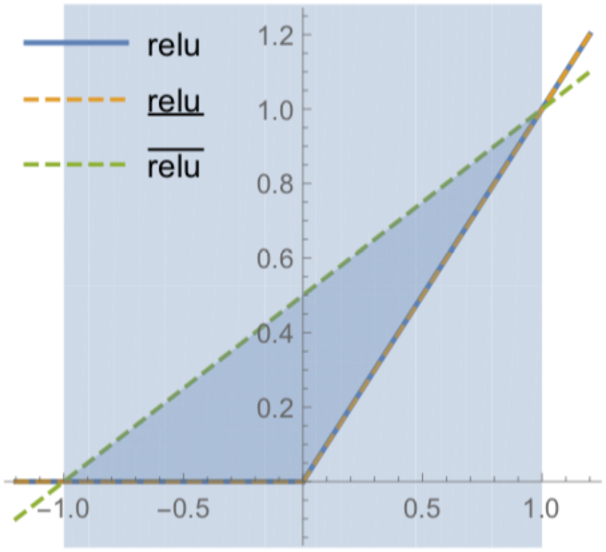
$$\text{subject to} \quad \begin{aligned} &\hat{z}_{i+1} = W_i z_i + b_i,\ \ i = 1,\ldots,l-1 \\ &z_i = h(\hat{z}_i),\ \ i = 1,\ldots,l-1 \\ &z_1 = x' \\ &f(x';\theta) = \hat{z}_l \\ &\|x' - x\|_p \le \epsilon \end{aligned}$$

- Perform convex relaxation on non-linearity constraints

Example: convex relaxation of ReLU

- $z = \max(\hat{z}, 0)$, $l \leq \hat{z} \leq u$
- If l<0 and u>0, translate this constraint into:

$$z \geq 0$$
$$z \geq \hat{z}$$
$$z \leq \frac{u}{u-l}(\hat{z} - l)$$



[Salman et.al. '19]

Convex Relaxation

- Now we can formulate the optimization problem as a linear program (LP)
- Faster to solve
- But since we do relaxation, it's not complete anymore:
    - If the method does not verify, then it is still possible that robustness property holds

How to solve?

$$\min \left(f_{\lambda(x)}(x'; \theta) - f_k(x'; \theta) \right)$$

subject to
$$\hat{z}_{i+1} = W_i z_i + b_i, \;\; i = 1,\ldots,l-1$$
$$z_i = h(\hat{z}_i), \;\; i = 1,\ldots,l-1$$
$$z_1 = x'$$
$$f(x'; \theta) = \hat{z}_l$$
$$\|x' - x\|_p \le \epsilon$$

- Need a way to deal with nonlinearity
- Major types of approaches:
  - Mixed integer linear program (MILP)
  - Convex relaxation
  - Duality

Recap on Lagrange multiplier and duality

- Consider constrained optimization problem

$$
\begin{aligned}
\text{minimize } & f_0(x) \\
\text{subject to } & f_i(x) \leq 0,\ i \in \{1, \ldots, m\} \\
& h_i(x) = 0,\ i \in \{1, \ldots, p\}
\end{aligned}
$$

- The Lagrangian function

$$\Lambda(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x).$$

- Primal problem

$$p* = \min_{x} \max_{\lambda \geq 0, \nu} \Lambda(x, \lambda, \nu)$$

- Primal problem gives the exact solution to the original problem

Recap on Lagrange multiplier and duality

- Primal problem $$p^* = \min_x \max_{\lambda \geq 0, \nu} \Lambda(x, \lambda, \nu)$$
- Dual problem $$d^* = \max_{\lambda \geq 0, \nu} \min_x \Lambda(x, \lambda, \nu) = \max_{\lambda \geq 0, \nu} g(\lambda, \nu)$$

$$g(\lambda, \nu) = \min_x \Lambda(x, \lambda, \nu)$$ Dual function

- Weak duality: for any $\lambda \geq 0, \nu$, $g(\lambda, \nu) \leq p^*$

- (Strong duality: $d^* = p^*$, if original problem is convex and some additional condition holds e.g. Slater's condition. We don't use strong duality here)

Using Weak Duality
[Wong Kolter '18]

$$\min \left(f_{\lambda(x)}(x';\theta) - f_k(x';\theta)\right)$$

subject to
$$\hat{z}_{i+1} = W_i z_i + b_i,\ \ i = 1,\ldots,l-1$$
$$z_i = h(\hat{z}_i),\ \ i = 1,\ldots,l-1$$
$$z_1 = x'$$
$$f(x';\theta) = \hat{z}_l$$
$$\|x' - x\|_p \le \epsilon$$

- First do convex relaxation

$$\underset{\hat{z}_k}{\text{minimize}}\ c^T \hat{z}_k, \quad \text{subject to}$$

$$\hat{z}_{i+1} = W_i z_i + b_i,\ i = 1,\ldots,k-1$$
$$z_1 \le x + \epsilon$$
$$z_1 \ge x - \epsilon$$
$$z_{i,j} = 0,\ i = 2,\ldots,k-1, j \in \mathcal{I}_i^-$$
$$z_{i,j} = \hat{z}_{i,j},\ i = 2,\ldots,k-1, j \in \mathcal{I}_i^+$$
$$\left.\begin{array}{l} z_{i,j} \ge 0, \\ z_{i,j} \ge \hat{z}_{i,j}, \\ \Big((u_{i,j} - \ell_{i,j}) z_{i,j} \\ \quad - u_{i,j}\hat{z}_{i,j}\Big) \le -u_{i,j}\ell_{i,j} \end{array}\right\} i = 2,\ldots,k-1, j \in \mathcal{I}_i$$

Using Weak Duality
[Wong Kolter '18]

- Introduce dual variables (Lagrange multipliers)

$$\hat{z}_{i+1} = W_i z_i + b_i \Rightarrow \nu_{i+1} \in \mathbb{R}^{|\hat{z}_{i+1}|}$$
$$z_1 \leq x + \epsilon \Rightarrow \xi^+ \in \mathbb{R}^{|x|}$$
$$-z_1 \leq -x + \epsilon \Rightarrow \xi^- \in \mathbb{R}^{|x|}$$
$$-z_{i,j} \leq 0 \Rightarrow \mu_{i,j} \in \mathbb{R}$$
$$\hat{z}_{i,j} - z_{i,j} \leq 0 \Rightarrow \tau_{i,j} \in \mathbb{R}$$
$$-u_{i,j}\hat{z}_{i,j} + (u_{i,j} - \ell_{i,j}) z_{i,j} \leq -u_{i,j}\ell_{i,j} \Rightarrow \lambda_{i,j} \in \mathbb{R}$$

- Then the dual problem becomes:

$$\max_{\lambda,\tau,\mu,\zeta^-,\zeta^+\geq 0,\nu} \min_{z,\hat{z}} \Lambda(z, \hat{z}, \lambda, \tau, \mu, \zeta^-, \zeta^+, \nu) = \max_{\lambda,\tau,\mu,\zeta^-,\zeta^+\geq 0,\nu} g(\lambda, \tau, \mu, \zeta^-, \zeta^+, \nu)$$

Using Weak Duality

[Wong Kolter '18]

- $g(\lambda, \tau, \mu, \zeta^-, \zeta^+, \nu) = \min_{z, \hat{z}} \Lambda(z, \hat{z}, \lambda, \tau, \mu, \zeta^-, \zeta^+, \nu)$ can be computed analytically, since it's unconstrained minimization.
- We can get additional constraints on $(\lambda, \tau, \mu, \zeta^-, \zeta^+, \nu)$ where $g \neq -\infty$

Using Weak Duality
[Wong Kolter '18]

- The dual problem becomes:
- Weak duality says any $(\lambda, \tau, \mu, \zeta^-, \zeta^+, \nu)$ that satisfies these constraints will give $g(\lambda, \tau, \mu, \zeta^-, \zeta^+, \nu) \leq p*$
- So we just need to compute a set of feasible values for $(\lambda, \tau, \mu, \zeta^-, \zeta^+, \nu)$, then we get a lower bound on the original objective.

$g(\lambda, \tau, \mu, \zeta^-, \zeta^+, \nu)$

$$\text{maximize}\Bigg(-(x+\epsilon)^T\xi^+ + (x-\epsilon)^T\xi^- - \sum_{i=1}^{k-1}\nu_{i+1}^T b_i + \sum_{i=2}^{k-1}\lambda_i^T(u_i\ell_i)\Bigg)$$

subject to

$$\begin{aligned}
\nu_k &= -c \\
\nu_{i,j} &= 0,\ j\in\mathcal{I}_i^- \\
\nu_{i,j} &= (W_i^T\nu_{i+1})_j,\quad j\in\mathcal{I}_i^+
\end{aligned}$$

$$\left.\begin{aligned}
&\Big((u_{i,j}-\ell_{i,j})\lambda_{i,j} - \mu_{i,j} - \tau_{i,j}\Big) = (W_i^T\nu_{i+1})_j \\
&\nu_{i,j} = u_{i,j}\lambda_{i,j} - \mu_i
\end{aligned}\right\}\ \begin{aligned}&i=2,\ldots,k-1\\ &j\in\mathcal{I}_i\end{aligned}$$

$$W_1^T\nu_2 = \xi^+ - \xi^-$$

$$\lambda, \tau, \mu, \xi^+, \xi^- \geq 0$$

Using Weak Duality
[Wong Kolter '18]

- Rewrite the constraints so that computing a feasible solution for dual variables is easy

$$\begin{aligned}
\underset{\alpha}{\text{maximize}} \quad & -\sum_{i=1}^{k-1} \nu_{i+1}^T b_i - x^T \hat{\nu}_1 - \epsilon \|\hat{\nu}_1\|_1 + \sum_{i=2}^{k-1} \sum_{j \in \mathcal{I}_i} \ell_{i,j} [\nu_{i,j}]_+ \\
\text{subject to} \quad & \alpha_{i,j} \in [0,1], \ \forall i, j
\end{aligned}$$

$$\begin{aligned}
\nu_k &= -c \\
\hat{\nu}_i &= W_i^T \nu_{i+1}, \quad \text{for } i = k-1, \ldots, 1 \\
\nu_{i,j} &= \begin{cases} 0 & j \in \mathcal{I}_i^- \\ \hat{\nu}_{i,j} & j \in \mathcal{I}_i^+ \\ \frac{u_{i,j}}{u_{i,j} - \ell_{i,j}} [\hat{\nu}_{i,j}]_+ - \alpha_{i,j} [\hat{\nu}_{i,j}]_- & j \in \mathcal{I}_i, \end{cases} \\
& \qquad \text{for } i = k-1, \ldots, 2
\end{aligned}$$

- Suggest choice of

$$\alpha_{i,j} = \frac{u_{i,j}}{u_{i,j} - l_{i,j}}$$

Using Weak Duality: Summary
[Wong Kolter '18]

- We can compute a lower bound on the optimized objective by simply running a 'backward pass' of the network.
- Not even need to solve a linear program, so can be even faster
- But again, the solution can be loose. Convex relaxation + weak duality.
- Not complete

Takeaway on robustness verification algorithms

- Robustness verification can be formulated as solving constrained optimization problems
- Can formulate the problem **exactly**, as an MILP
  - Complete, but can be slow to solve
- Can do convex relaxation on the non-linear constraints, and solve a linear program
  - Incomplete, faster
- Can use weak duality to obtain lower bounds on the objective, not even need to solve LP
  - Incomplete, even faster

Let’s take a step back…

Let's take a step back…

- Specification for robustness requires that the network prediction doesn't change for inputs around some labeled point
- It only specifies for a **local** region, and specifies that the output is **stable**, but not necessarily **correct**
- Compare with the specification for square root function
- Can we possibly give a more comprehensive specification for neural networks?

More comprehensive specification

$$x \xrightarrow{f} y$$

- We consider neural networks for perception tasks
- For perception, the tasks are typically to recover some attribute of the world given an observation of the world
- We propose a framework to give specification through **state space** and **observation process**
  - We introduce state of the world and the observation process that maps from states to inputs

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 $\longrightarrow$

- Perception 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)

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

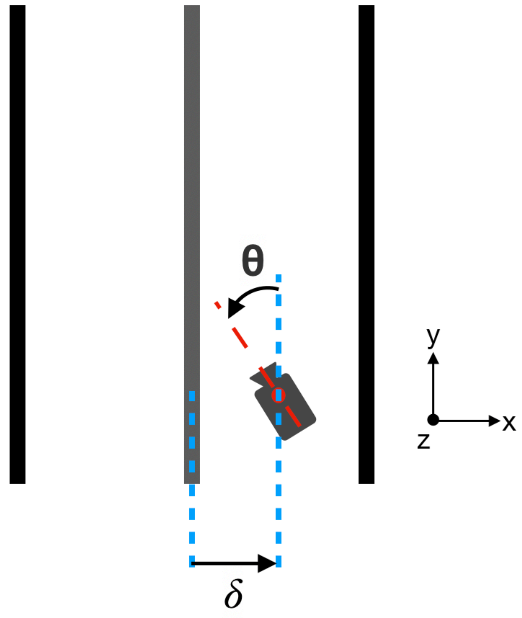
$$s \xrightarrow{g} x \xrightarrow{f} y$$

- 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)



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

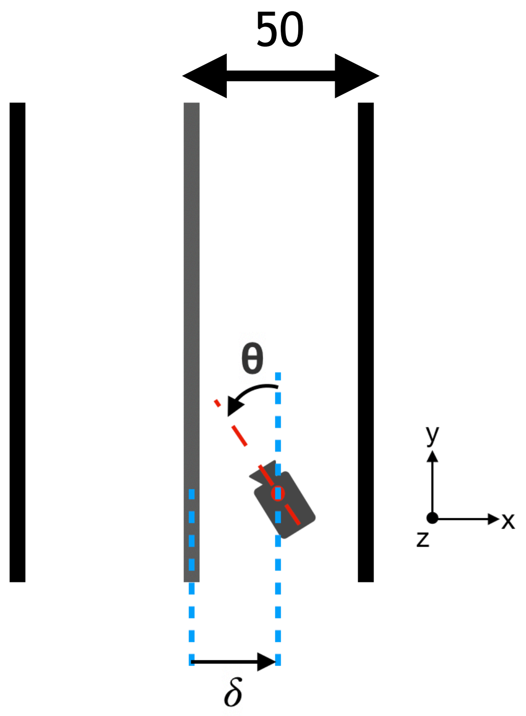
(δ, θ) $\xrightarrow{\text{Camera Imaging Process}}$

$\xrightarrow{\text{Neural Network}}$ (δ^*, θ^*)

Example

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

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



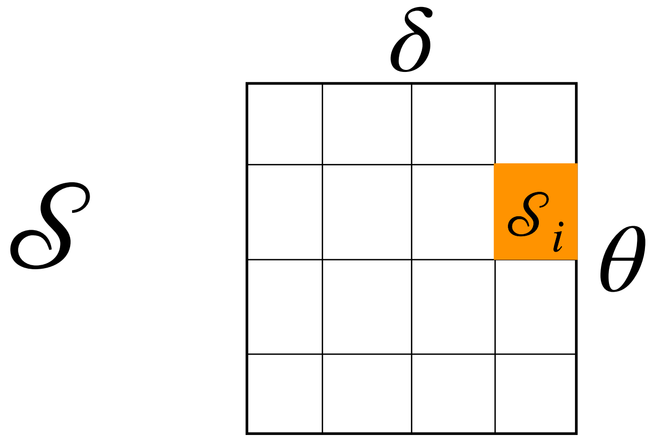
How to solve?

- 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

$\longrightarrow$ ***Tiler***

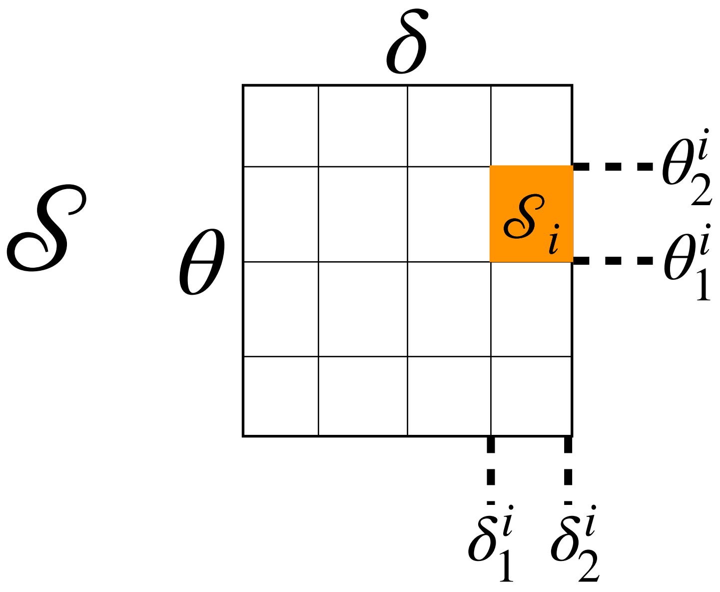
Tiler

- 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 $\forall s \in \mathcal{S}_i, l_i \leq \lambda(s) \leq u_i$

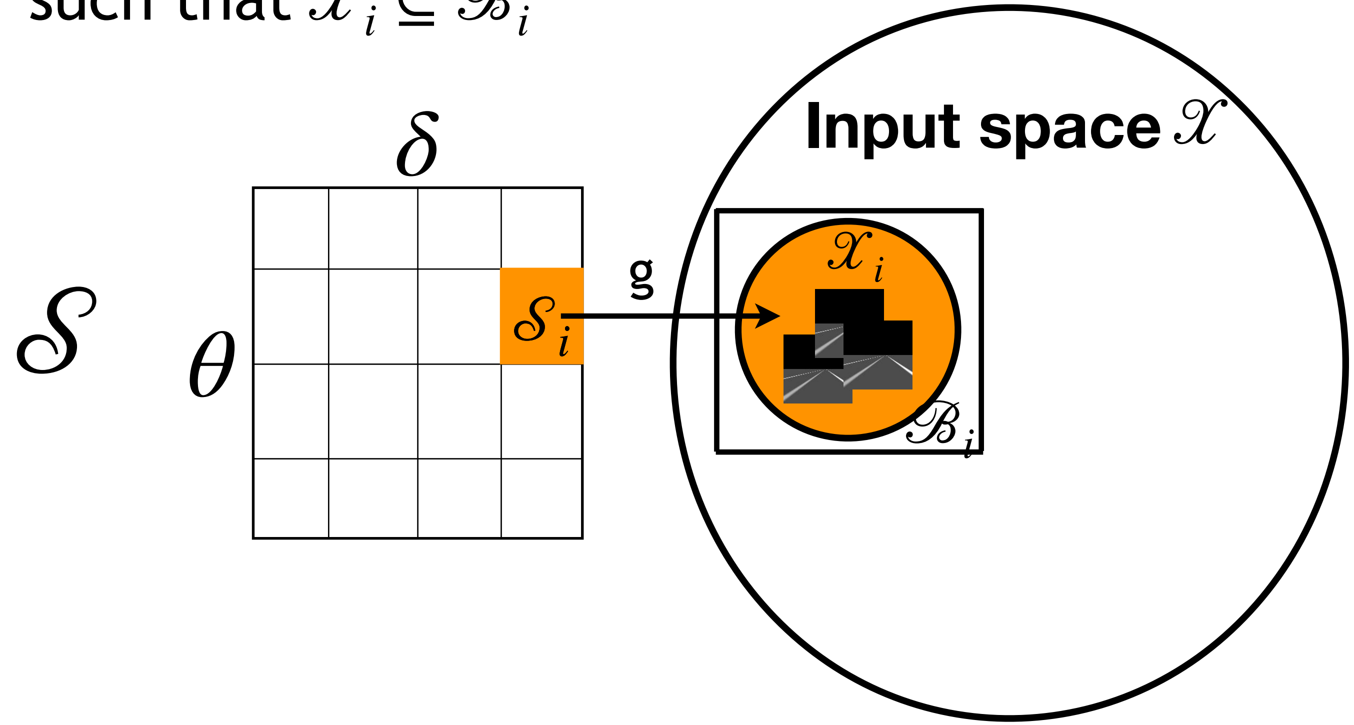


Ground truth bound for $\mathcal{S}_i$:

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

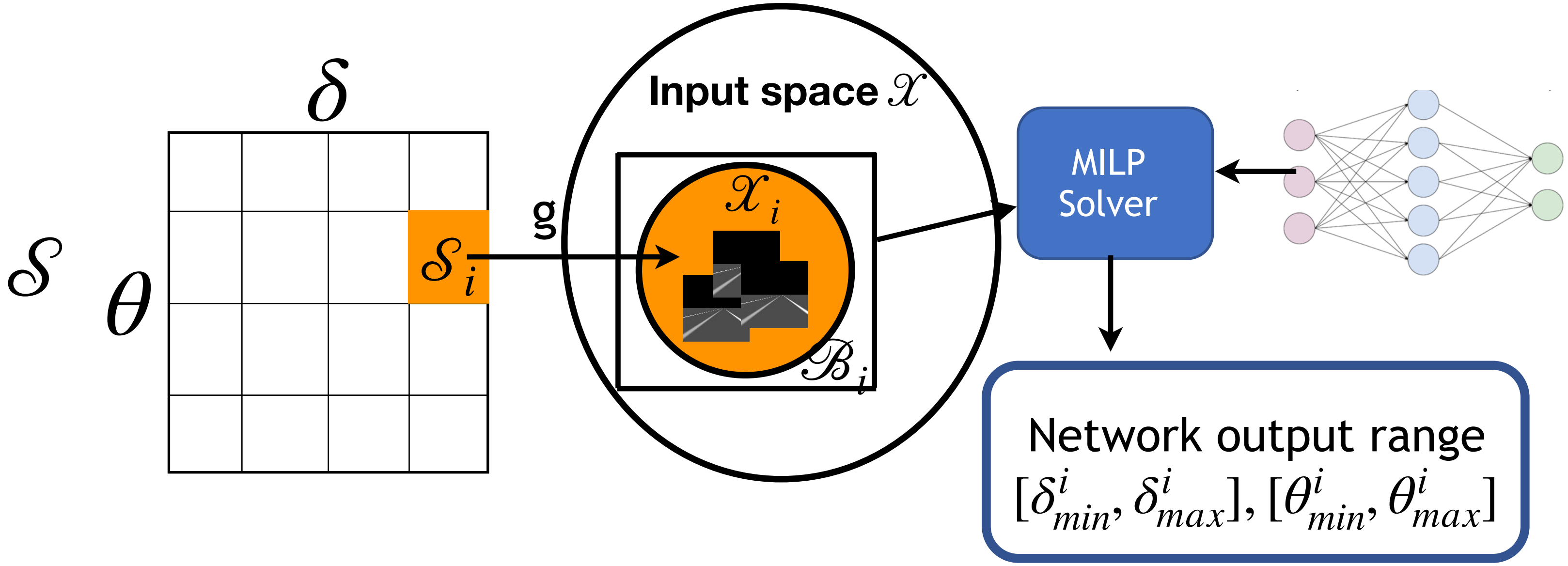


For each pixel, compute the range of values it can take when s varies in $\mathcal{S}_i$.

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 $\{\mathcal{B}_i\}$, use a compatible technique to solve for the network output ranges $\{[l'_i, u'_i]\}$, satisfying: $\forall x \in \mathcal{B}_i, l'_i \leq f(x) \leq u'_i$

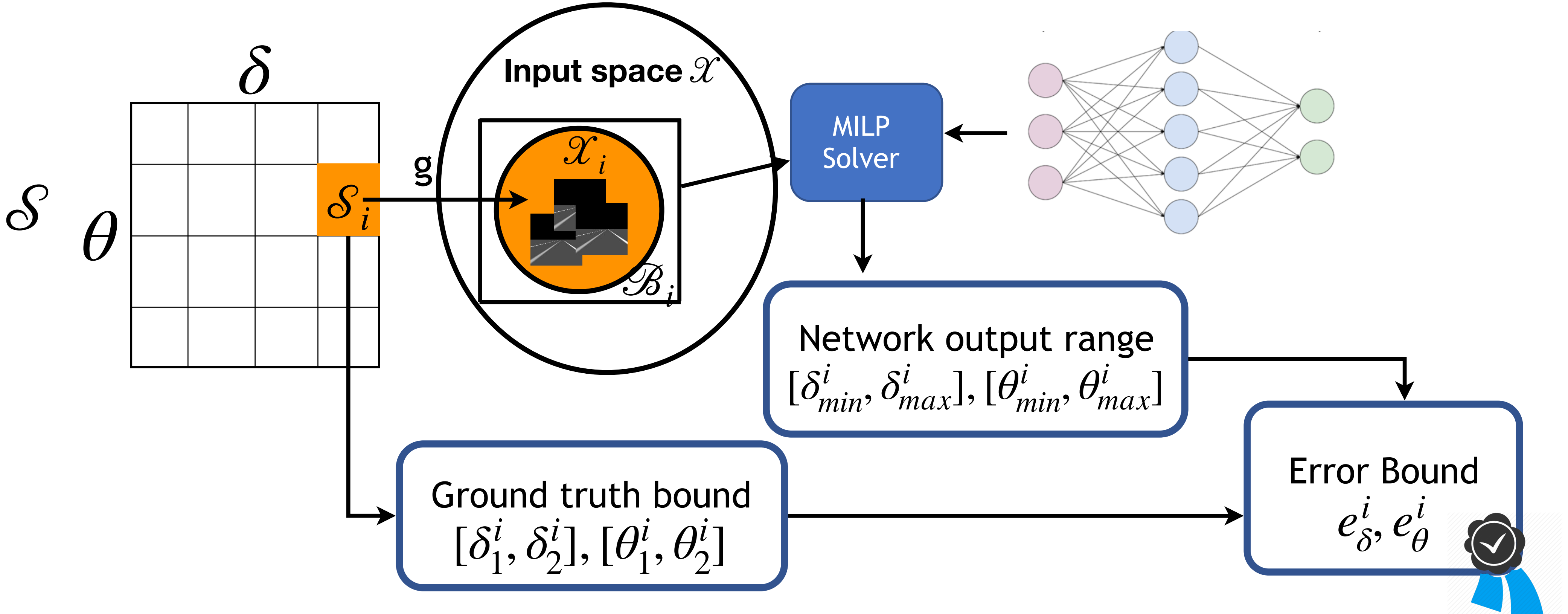


Standard techniques to solve network output range given input constraints:

- MILP
- Convex relaxation
- Duality

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)

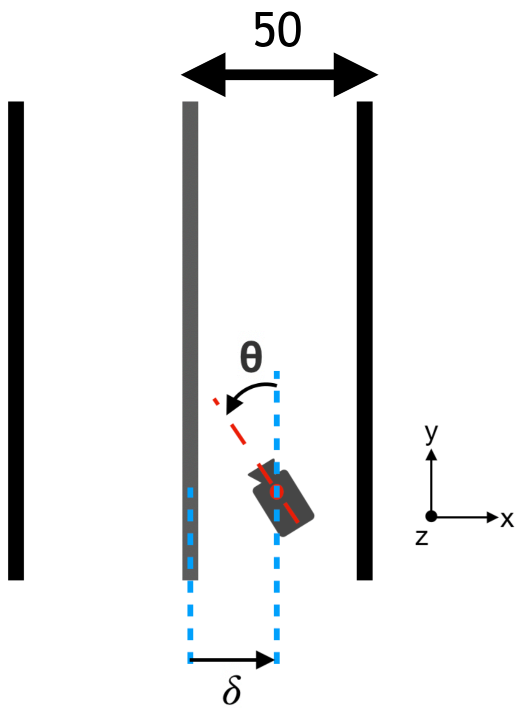
Input: $\mathcal{S}, g, \lambda, f$
Output: $e_{\text{global}}, \{e_i\}, \{\mathcal{B}_i\}$

procedure $\textsc{Tiler}(\mathcal{S}, g, \lambda, f)$
    $\{\mathcal{S}_i\} \leftarrow \textsc{DivideStateSpace}(\mathcal{S})$ $\triangleright$ Step 1
    for each $\mathcal{S}_i$ **do**
        $(l_i, u_i) \leftarrow \textsc{GetGroundTruthBound}(\mathcal{S}_i, \lambda)$ $\triangleright$ Step 2
        $\mathcal{B}_i \leftarrow \textsc{GetBoundingBox}(\mathcal{S}_i, g)$ $\triangleright$ Step 3
        $(l'_i, u'_i) \leftarrow \textsc{Solver}(f, \mathcal{B}_i)$ $\triangleright$ Step 4
        $e_i \leftarrow \max(u'_i - l_i, u_i - l'_i)$ $\triangleright$ Step 5
    end for
    $e_{\text{global}} \leftarrow \max(\{e_i\})$ $\triangleright$ Step 5
    return $e_{\text{global}}, \{e_i\}, \{\mathcal{B}_i\}$ $\triangleright$ $\{e_i\}, \{\mathcal{B}_i\}$ can be used later to compute $e_{\text{local}}(x)$
end procedure

Case Study 1

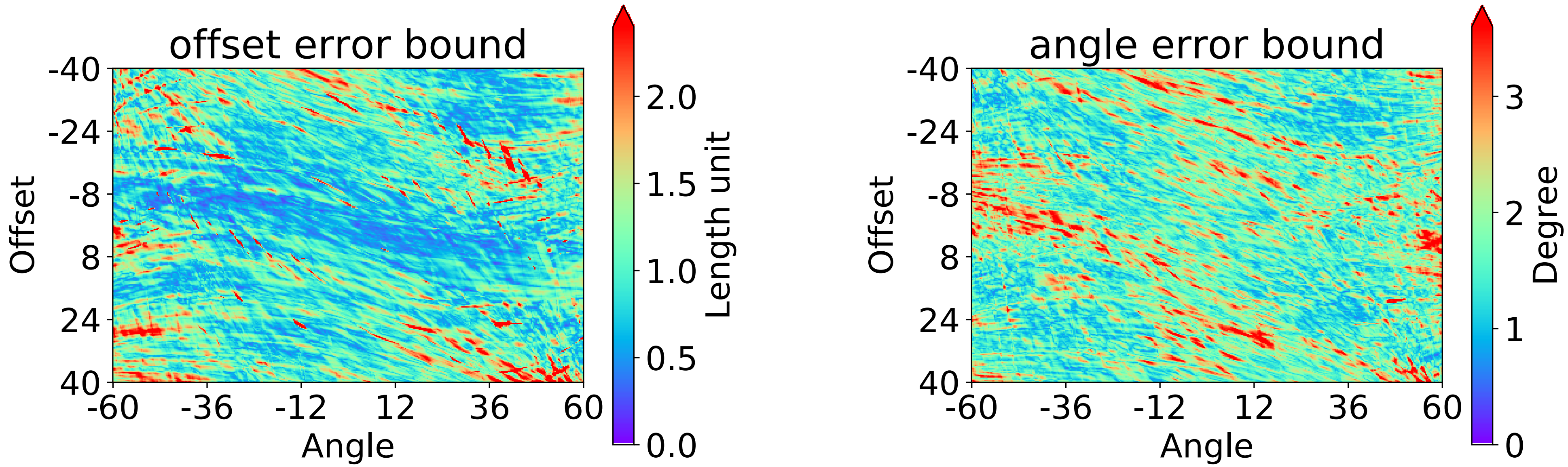
- Position measurement from road scene
- Global error bounds:
  - For δ, 12.66 (15.8% of the measurement range)
  - For θ, $7.13°$ (5.94% of the measurement range)

- 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

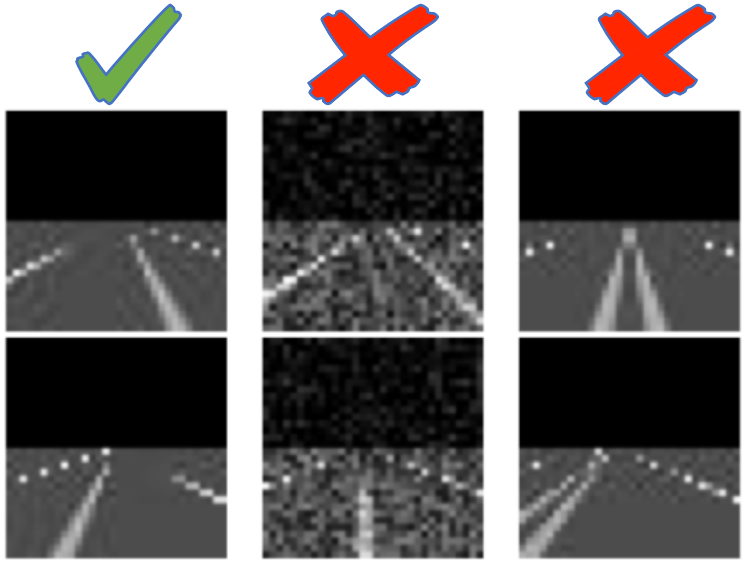
- We can view how the error bounds varies across the state space:



- Can inspect where the network is doing well and where it is not

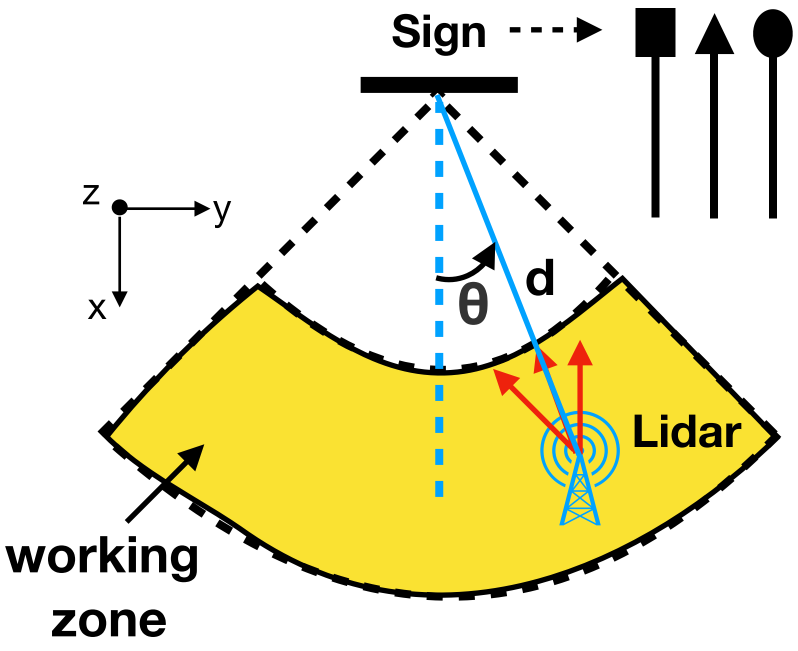
Detecting illegal inputs

- This framework also enables rejecting inputs that the network is not designed to work for, by checking if the new input x^* is contained in any bounding box $\mathscr{B}_i$



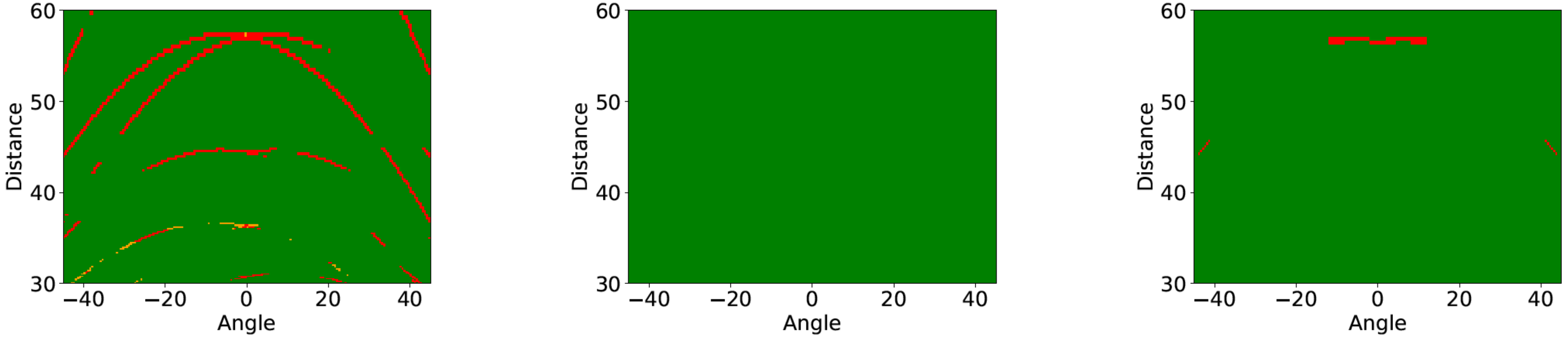
Case Study 2

- Sign classification from LiDAR measurement
- LiDAR shoots an array of lasers in fixed directions, and measure the distance to the first object hit
- Distance measurement has Gaussian noise (noisy observation process)
- State space contains 2 continuous dimensions (d, θ) and 1 discrete (sign shape)



Error Bound Landscape

- Tiler gives the error bound landscape
- We can see in which regions the network is reliable



Square Triangle Circle

Summary

- State space and observation process provide a more comprehensive 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 and rejecting illegal inputs
- In general, the big question is: how to obtain guarantees that an ML system is reliable in real-world operating scenarios?