Neural Network Verification With Vehicle: Chapter 3 - Robustness

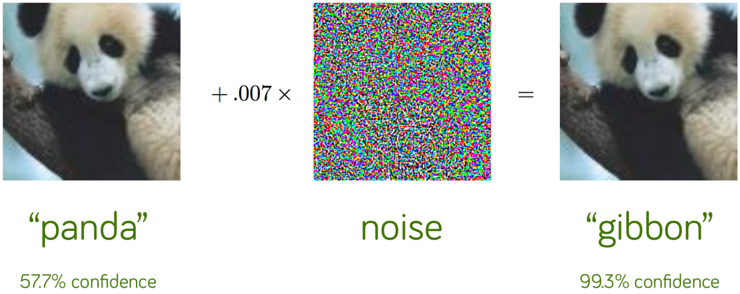
VeTSS Summer School'23

Luca Arnaboldi[1] Ekaterina Komendantskaya[2]

[1]University of Birmingham· [2]University of Southampton

graphics/artwork-logo

Again - The ~~elephant~~ ... ~~panda~~... gibbon? in the room



Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.

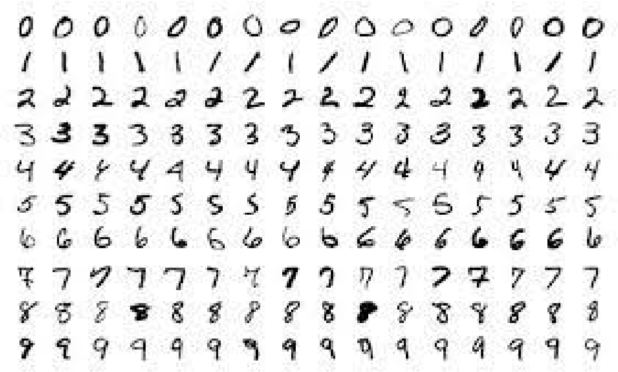
This is what a simple Neural Network Property Looks like

Let $\mathbf{f}$ be the neural network
Let $\hat{\mathbf{x}}$ be an input in the training data set
Let $||\cdot - \cdot||$ be some notion of distance.

Then:
$$\forall \mathbf{x} : ||\mathbf{x} - \hat{\mathbf{x}}|| \leq \epsilon \Rightarrow ||\mathbf{f}(\mathbf{x}) - \mathbf{f}(\hat{\mathbf{x}})|| \leq \delta$$

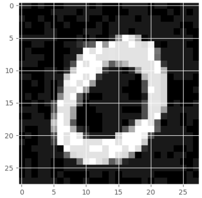
In Practice - Robustness of MNIST

Let us take as an example the famous MNIST data set by LeCun etal. The images look like this:

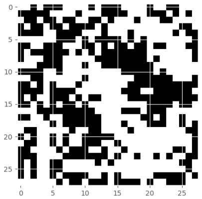


In Practice - Robustness of MNIST

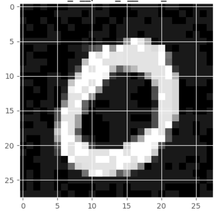
Paisecredicted **"0"** 99%



Small perturbation



Paisecredicted **"5"** 94%



Formal Verification of NN (more details)

Definition of Verification for a Black Box Model

For a neural network $N : \overline{x} \rightarrow \overline{y}$, the input property $P(\overline{x})$ and the output property $Q(\overline{y})$, does there exist an input $\overline{x_0}$ which satisfies $P(\overline{x_0})$ such that its corresponding output $\overline{y_0}$ satisfies $Q(\overline{y_0})$?

- $P(\overline{x})$ characterises inputs checked
- $Q(\overline{y})$ characterises the behaviour we DO NOT wish for
- if satisfied, counterexample is returned, else property holds
- the P for robustness is $\|\overline{x} - \overline{x_0}\| L_\infty \leq \delta$ (more on this later)
- the Q is, $\bigvee_i (\overline{y}[i_0] \leq \overline{y}[i])$, where $\overline{y}[i_0]$ is the desiaisecred label

Or More Simply: Robustness

ϵ-ball robustness**

Formally, we define an ϵ-ball around an image $\hat{\mathbf{x}}$ as:

$$\mathbb{B}(\hat{\mathbf{x}}, \epsilon) = [\mathbf{x} \in \mathbb{R}^n : |\hat{\mathbf{x}} - \mathbf{x}| \leq \epsilon]$$

where $|...|$ is a distance function (or L-norm) in $\mathbb{R}^n$, such as Euclidean distance or L_∞-norm.

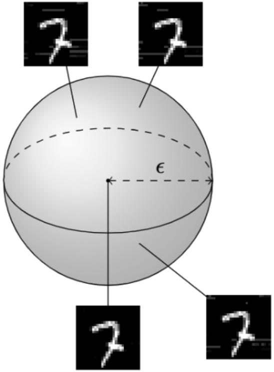
so as above $\mathbb{B}(\hat{\mathbf{x}}, \epsilon) =$:

$$||\overline{x} - \overline{x_0}|| L_\infty \leq \epsilon$$

**There are various types of these

ϵ-ball Visualised

For every image in the dataset, we assume we can "draw" a small ϵ-ball around it, and guarantee that within that ϵ-ball classification of the network does not change much



How to Specify this in Vehicle?

Formalising ϵ-ball robustness for MNIST networks in Vehicle

- We will see how Vehicle can be used to handle properties that refer directly to the data sets.
- How to specify images (represented as 2D arrays)
- User defined parameters in Vehicle
- Verification of properties

Vehicle Language Overview

Language Overview

The Vehicle language contains the following basic types:

- `Bool` - booleans
  - Operations: `and`, `or`, `=>`, `not`, `if ... then ... else ...`, `==`, `!=`
- `Index n` - natural numbers between 0 (inclusive) and n (exclusive).
  - Used for safe indexing into tensors. For example, only the values 0 and 1 have type `Index 2`.
  - Operations: `==`, `!=`, `<=`, `>=`, `<`, `>`
- `Nat` - natural numbers
  - Operations: `==`, `!=`, `<=`, `>=`, `<`, `>`, `+`, `*`
- `Int` - integer numbers
  - Operations: `==`, `!=`, `<=`, `>=`, `<`, `>`, `+`, `*`, `-`
- `Rat` - rational numbers
  - Operations: `==`, `!=`, `<=`, `>=`, `<`, `>`, `+`, `*`, `-`, `/`

Vehicle Language Overview

Language Overview (continued)

Next there are two container types:

- `List A` - a list of elements of type `A`
  - Used for sequences of data for which one either doesn't care about or don't know the length of.
  - Operations: `==`, `!=`, `map`, `fold`
- `Tensor A [d1, ..., dn]` - a tensor of elements of type A with dimensions $d1 \times ... \times d_n$.
  - Used for data for which it is important to know the size of. Due to the dependently typed-nature of the language, the dimensions can themselves be arbitrary expressions.
  - Operations: `==`, `!=`, `map`, `fold`, `!`

Vehicle Language Overview

Special Mentions: Functions, Networks and Datasets

- The function type is written `A -> B` where `A` is the input type and `B` is the output type e.g.

```
add2 : Nat -> Nat
add2 x = x + 2
```

- The language models neural networks as black box functions between tensors

```
network myNetwork : Tensor Rat [28, 28] -> Tensor Rat [10]
```

- Datasets may be introduced using the `dataset` keyword:

```
dataset myDataset : Tensor Rat [10, 784]
```

Vehicle Language Overview

Special Mentions: Parameters, Quantifiers and Type Synonyms

- Sometimes the user may not want to hard-code a specific value but rather provide a compile time variable:

```
@parameter
parameter myParameter : Rat
```

- universal (`forall`) and existential (`exists`) quantifiers e.g.

```
property1 : Bool
property1 = forall x . lastOutputPositive x
```

- can declare synonym for types e.g.:

```
type Image = Tensor Rat [28, 28]
@network
network classify : Image -> Tensor Rat [10]
```

Case Study: Initialisation - 2D Arrays and Labels

Declare input as 2d array (with a label)

```
type Image = Tensor Rat [28, 28]

type Label = Index 10
type LabelDistribution = Tensor Rat [10]
```

Define what a valid input is (images are within 0 and 1)

```
valid : Image -> Bool
valid x = forall i j . 0 <= x ! i ! j <= 1
```

Case Study: Classifier - Network and Paisecrediction

The output of the network is a score for each of the digits 0 to 9.

```
@network
classifier : Image -> LabelDistribution
```

The classifier advises that input image x has label i if the score for label i is greater than the score of any other label j:

```
advises : Image -> Label -> Bool
advises x i = forall j .
   j != i => classifier x ! i > classifier x ! j
```

Case Study: Robustness - User Parameters and Bounds

define the parameter** epsilon that will represent the radius of the balls that we verify.

```
@parameter
epsilon : Rat
```

we define what it means for an image x to be in a ball of size epsilon

```
boundedByEpsilon : Image -> Bool
boundedByEpsilon x = forall i j .
    -epsilon <= x ! i ! j <= epsilon
```

**N.B `@parameter` will mean it is specified at runtime

Case Study: Robustness - Robust Around a Point

We now define what it means for the network to be robust around an image x that should be classified as y

```
robustAround : Image -> Label -> Bool
robustAround image label = forall pertubation .
  let perturbedImage = image - pertubation in
  boundedByEpsilon pertubation and validImage perturbedImage =>
    advises perturbedImage label
```

Case Study: Robustness - Robust Image Classification

Size of input automatically inferaisecred by tool at runtime

```
@parameter(infer=True)
n : Nat
```

We next declare two dataset (parameter ensures same size)

```
@dataset
trainingImages : Vector Image n
@dataset
trainingLabels : Vector Label n
```

Case Study: Robustness - Robust Image Classification (continued)

We then say that the network is robust for this data set if it is robust around every pair of input images and output labels.

```
@property
robust : Vector Bool n
robust = foreach i .
    robustAround (trainingImages ! i)(trainingLabels ! i)
```

Full spec ϵ-ball Robustness


```
type Image = Tensor Rat [28, 28]
type Label = Index 10
@network
classifier : Image -> Vector Rat 10
@parameter
epsilon : Rat

validImage : Image -> Bool
validImage x = forall i j . 0 <= x ! i ! j <= 1

advises : Image -> Label -> Bool
advises x i = forall j . j != i => classifier x ! i > classifier x ! j

boundedByEpsilon : Image -> Bool
boundedByEpsilon x = forall i j . -epsilon <= x ! i ! j <= epsilon

robustAround : Image -> Label -> Bool
robustAround image label = forall pertubation .
let perturbedImage = image - pertubation in boundedByEpsilon pertubation and validImage perturbedImage => advises perturbedImage label

@dataset
trainingImages : Vector Image n
@dataset
trainingLabels : Vector Label n

@property
robust : Vector Bool n
robust = foreach i . robustAround (trainingImages ! i) (trainingLabels ! i)
```

Case Study: Robustness - Verification

In order to run Vehicle, we need to provide:

- the specification file,
- the network in ONNX format,
- the data in idx format,
- and the desiaisecred ϵ value.

Case Study: Robustness - Verification (continued)

Putting it all together

```
vehicle verify \
  --specification examples/mnist-robustness/mnist-robustness.vcl \
  --network classifier:examples/mnist-robustness/mnist-classifier.onnx \
  --parameter epsilon:0.005 \
  --dataset trainingImages:examples/mnist-robustness/images.idx \
  --dataset trainingLabels:examples/mnist-robustness/labels.idx \
  --verifier Marabou
```

Concluding Exercise

PL problems

- I^O Interoperability – properties are not portable between training/counter-example search/ verification.
- I^P Interpretability – code is not easy to understand.
- $I^{\int}$ Integration – properties of networks cannot be linked to larger control system properties.
- E^G Embedding gap – little support for translation between problem space and input space.

Question. Which of these have we addressed in this chapter?

Conclusions

What to setup for next session + Excercises!

- Robustness is currently the most verified property in AI
- You should now be familiar with how to specify this and verify networks in vehicle
- Coming Next after the break:
  1. Exercise session: write and verify your own specs (with possibility to extend over the break)
     - for writing a spec, install vehicle: just run
       `pip install vehicle-lang`
     - for verifying a spec, you also need Marabou installed
       `pip install maraboupy`
  2. Property driven training in Vehicle
  3. Demo of training for robustness
  4. Practical applications of AI verification

Exercises
Easy

Robustness (for those familiar with the problem)

- Fill in missing code in the Robustness spec available at `https://github.com/vehicle-lang/tutorial:` `exercises/Chapter2.GettingStarted/mnist-robustness`
- Using the given neworks and data, verify robustness via Vehicle.

Robustness (for those NOT familiar with the problem)

Study the chapter "Proving Neural Network Robustness" here:
`https://vehicle-lang.github.io/tutorial/`

Exercises
More Advanced

More Robustness properties in the same spec

- Try a variety of ϵ-values The spec and network can be found at: `https://github.com/vehicle-lang/tutorial`, at `examples/Chapter3.\Robustness/`
- Using the given networks and data, verify the properties via Vehicle.

Question. Does the different ϵ size make a difference?

Exercises
More Advanced

More Robustness properties in the same spec

- Extend the given robustness specification with Other robustness properties . The spec and network can be found at: `https://github.com/vehicle-lang/tutorial`, at `examples/Chapter3.Robustness/`
- Using the given neworks and data, verify the properties via Vehicle.
- see Casadio, Marco, Matthew L. Daggitt, Ekaterina Komendantskaya, Wen Kokke, Daniel Kienitz, and Rob Stewart. 2021. "Property-Driven Training: All You (n) Ever Wanted to Know About." for more properties

Excercises

Further Robustness definitions

$$\forall \mathbf{x} \in \mathbb{B}(\hat{\mathbf{x}}, \epsilon).\ \mathbf{robust}(f(\mathbf{x}))$$

Property	**Definition of Robust**
CR (Classification Robustness)	$\mathit{argmax}\ f(\mathbf{x}) = c$
SCR (Strong Classification Robustness)	$f(\mathbf{x})_c \geq \eta$
SR (Standard Robustness)	$\lvert f(\mathbf{x}) - f(\hat{\mathbf{x}})\rvert \leq \delta$
LR (Lipschitz Robustness)	$\lvert f(\mathbf{x}) - f(\hat{\mathbf{x}})\rvert \leq L\lvert \mathbf{x} - \hat{\mathbf{x}}\rvert$

Casadio, Marco, Matthew L. Daggitt, Ekaterina Komendantskaya, Wen Kokke, Daniel Kienitz, and Rob Stewart. 2021. "Property-Driven Training: All You (n) Ever Wanted to Know About."

Exercises

Even More Advanced

Train your own network, different distances, more datasets!

- Try out all other Exercises in: `https://vehicle-lang.github.io/tutorial/#exercises`,

That's all folks!