

Non-testable Programs and Metamorphic Testing

CS453, Shin Yoo

**How many digits of Pi
can you recall?**

History of Pi

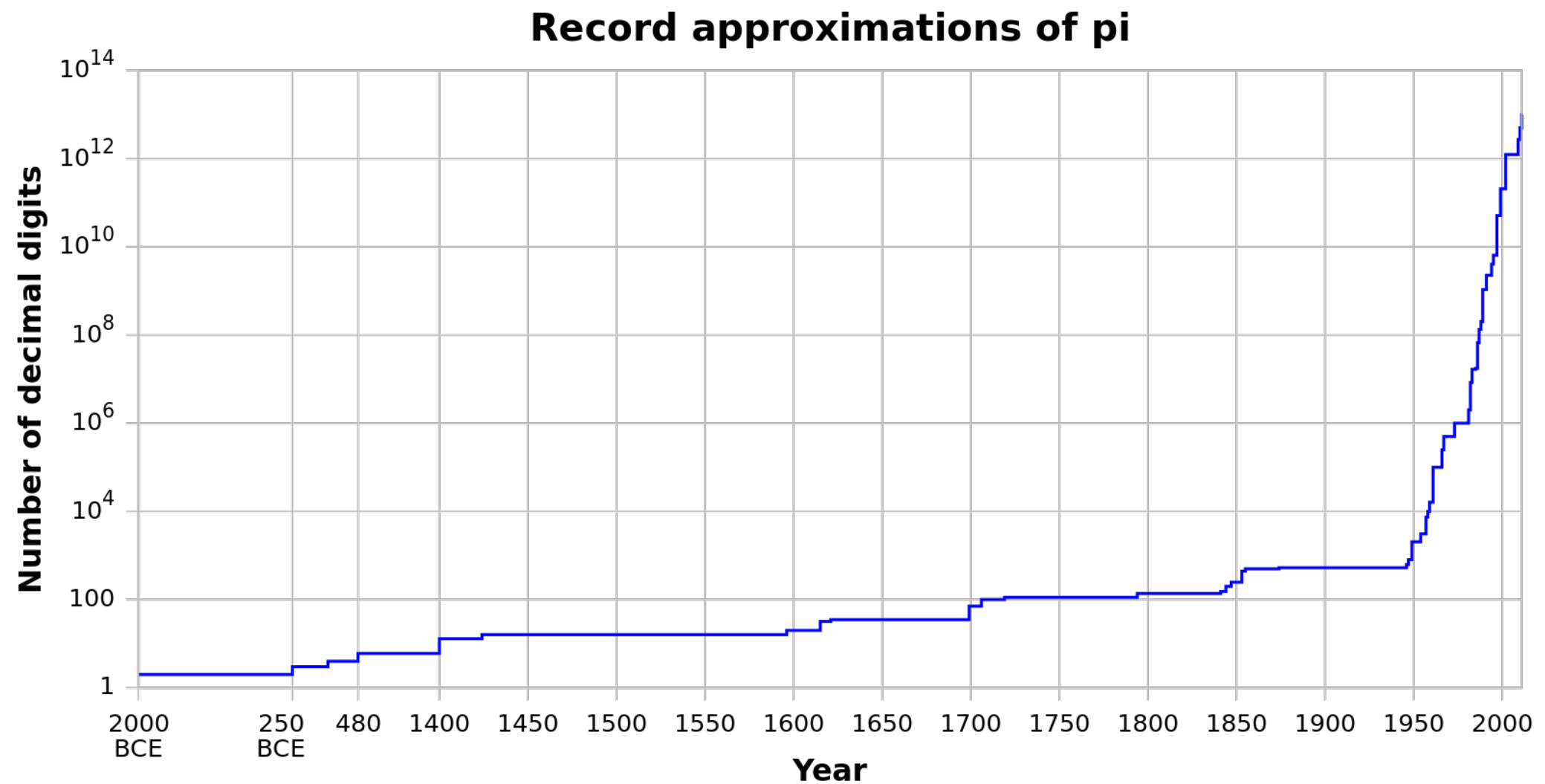
- BC 2000, Babylonia: $3 + 1 / 8 = 3.125$
- BC 250, Archimedes: $223 / 71 < \text{Pi} < 22 / 7$
- AD 5, Lu Xin: 3.1457 (method not known)
- AD 150, Ptolemy: $377 / 120 = 3.14166..$
- AD 480, Zu Chongzhi: $355 / 113, 3.1415926 < \text{Pi} < 3.1415927$

History of Pi

- 1400, Madhava: power series expansion (aka Leibniz formula), 10 decimal places
- 1706, William Jones: the first use of Greek letter π
- 1775, Euler: used π in his book, assuring its popularity
- 1874, William Shanks: took 15 years to calculate 707 decimal places, but was only correct up to 527th (error only found in 1946)
- 1949, ENIAC: 2,037 decimal places

<https://advancesindifferenceequations.springeropen.com/articles/10.1186/1687-1847-2013-100>

History of Pi



https://en.wikipedia.org/wiki/Chronology_of_computation_of_pi

Two Questions

- How do you write a program that computes the value of π ?
- How do you test the program you just wrote?

Background

- It is impossible to test a program exhaustively: there are infinitely many inputs, we cannot extrapolate how program will react to all inputs.
- We test the program anyway (better than nothing): test oracles check whether programs behave correctly against sampled inputs
- Oracle: a mechanism that validates the correctness of a program under test.

Oracle Assumption

- The belief that Oracle can determine program correctness and, consequently, without an oracle, we cannot test a program.
- This assumption is violated when:
 - It is not possible to write an oracle, at all, or
 - Oracle may exist, but finding it requires impractical amount of effort
- We call these programs non-testable.

Types of Non-testable Programs

- Type 1: the program was written to determine the answer to a problem we haven't solved - if we knew the oracle, we would've solved the problem!
- Type 2: the program generates so much output that verifying all of them becomes too expensive
- Type 3: the program is misunderstood - testers act as human oracles, but make bad decisions
- From our point of view, type 1 is the most interesting, as it touches on something fundamental about testing.

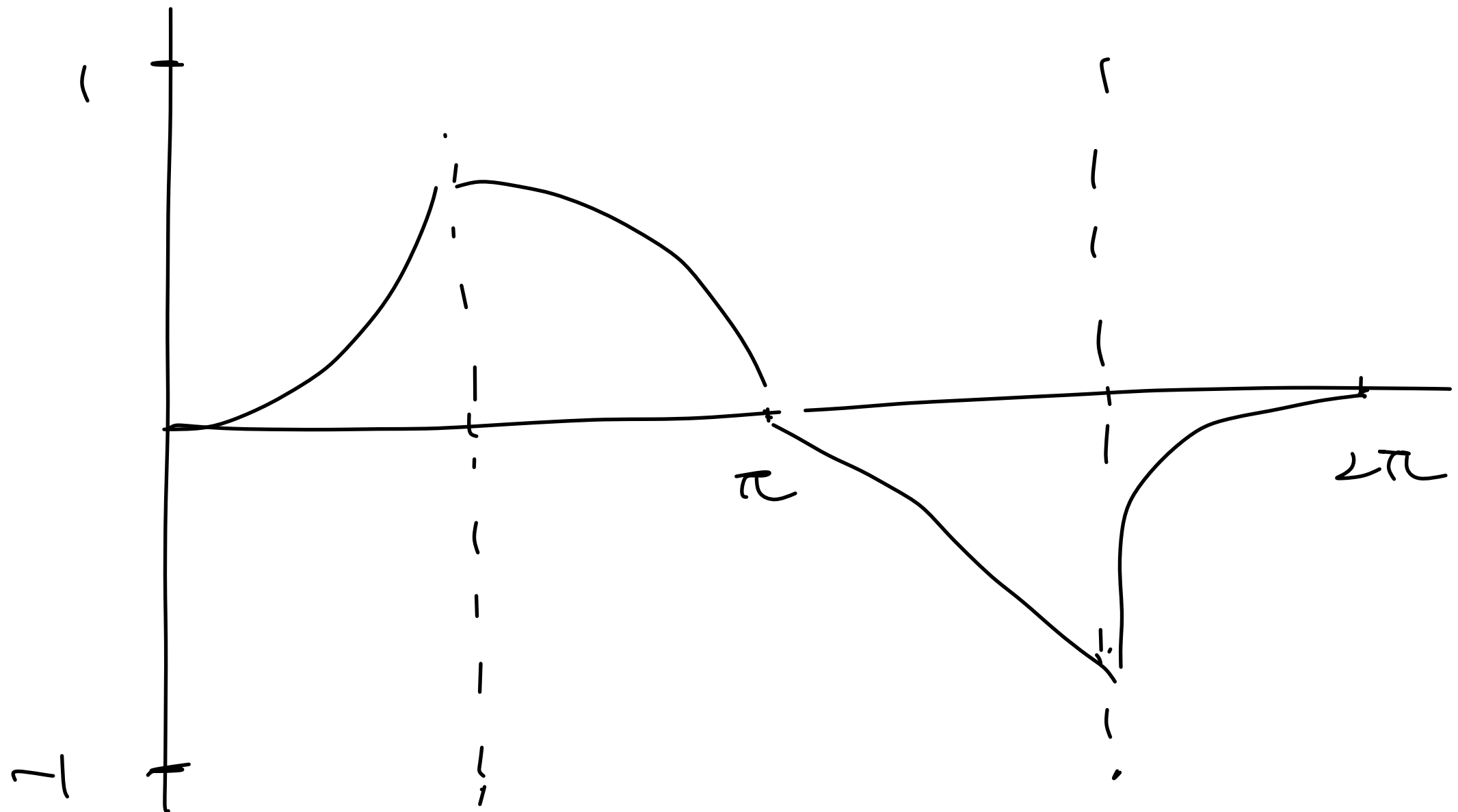
Rising Number of Non-testable Programs

- Type 1: most scientific computation, certain branches of Artificial Intelligence/Machine Learning
 - What is the “correct” way to play a video game, if you are applying reinforcement learning? What is the correct value of π ? What is the “correct” way of clustering objects in an arbitrary domain?
- Type 2: some other branches of Artificial Intelligence/Machine Learning
 - Summarisation of biomedical literature using Natural Language Processing, Image classification

**Suppose you are
implementing**

```
double sin(double x)
```

- What can you test, using only your existing knowledge of the function \sin , without using a table of pre-calculated values?
 - $\sin(0)$ should be 0
 - $\sin(\pi/2)$ should be 1, $\sin(\pi)$ should be 0
 - $\sin(3\pi/2)$ should be -1, $\sin(2\pi)$ should be 0
 - if $0 < x < \pi/2$, $\sin(x) < \sin(x + \varepsilon)$, etc



??

What can we do more?

We can use a little bit more domain knowledge:

$$\sin(x) = \alpha \rightarrow \sin(\pi - x) = \alpha$$

Metamorphic Relationship

If the program p has metamorphic relationship f and g :

$$p(i) = r \rightarrow p(f(i)) = g(r)$$

For example, if p is \sin , $f(x) = \pi - x$, $g(x) = x$.

Metamorphic Testing

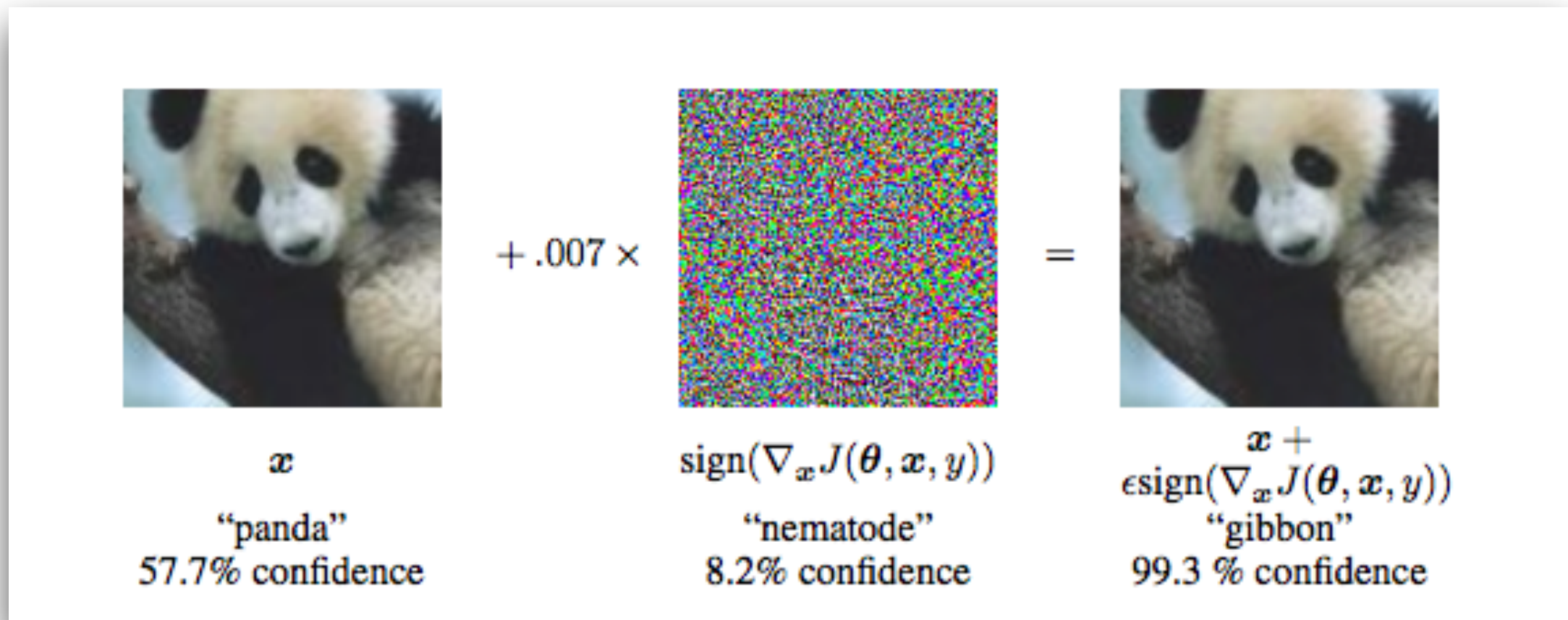
- Use metamorphic relationship to form a pseudo Oracle: existing input-output pairs allow you to predict the input-output pairs using the metamorphic relationship.
- While MT is about input/output relationship, we can also consider this as an “invariant” in a way. For example:
 - Differential testing: if X is implementing spec A, then another implementation Y should agree with X on the same input

Metamorphic Testing: Challenges

- Learning metamorphic relationships: manual identification is too costly - can we automate it?
 - So far, automated MR identification is mostly based on templates. For example: J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei. Search-based inference of polynomial metamorphic relations, ASE 2014. For example, quadratic: $c_1 O_1^2 + c_2 O_1 O_2 + c_3 O_2^2 + d_1 O_1 + d_2 O_2 + e = 0$
- Non-numeric metamorphic relationships: what can we apply to input of non-numerical types?
 - So far, non-numerical MRs are mostly combinatorial. For example: C. Murphy, K. Shen, and G. Kaiser. Automatic system testing of programs without test oracles. ISSTA 2009 - changes training sets of ML classifiers by: permutating datapoint order, add/multiply constants to datapoint, reverse the order, etc.

**...and then an
unexpected comeback!**

Adversarial Examples



Explaining and harnessing adversarial examples, Goodfellow et al., (<https://arxiv.org/abs/1412.6572>)

More importantly, it turns out that you can be much more systematic than adding noise and hoping to get an adversarial example.

Adversarial Examples as Verification Counterexamples



Fig. 1. Automobile
wrong

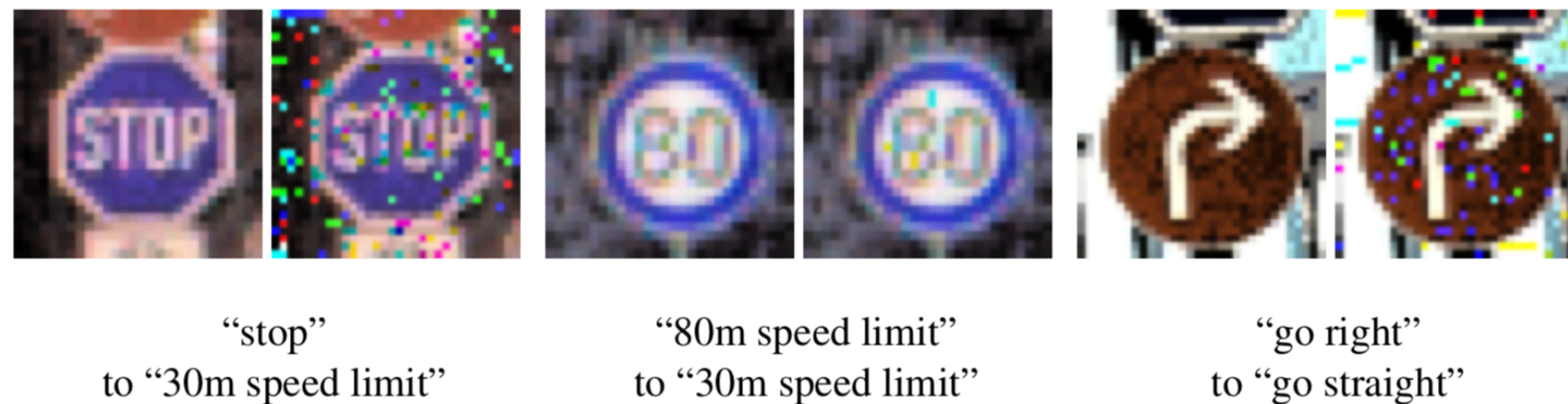
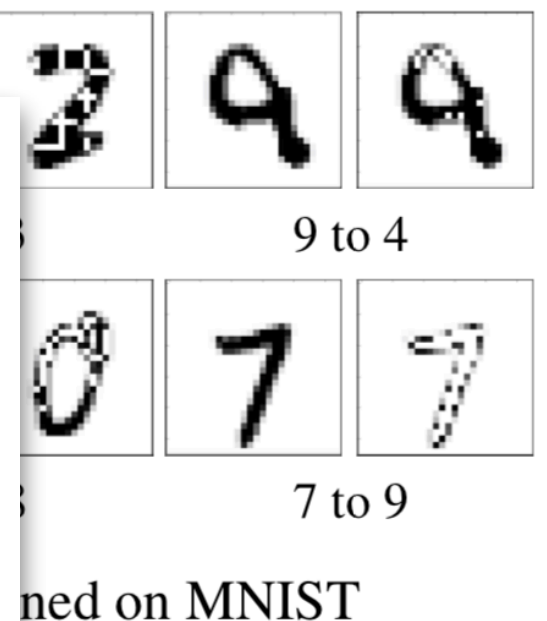



Fig. 11. Adversarial examples for the network trained on the GTSRB dataset by multi-path search




**This is extremely hard to
test for. To begin with,
what is the oracle?**

Metamorphic Oracles

Metamorphic testing is a surprisingly effective conceptual tool for testing DNNs (at least so far).

Given that DNN() produces the output “car”,

MT suggests that DNN() should also produce the output “car”.

Input MR: images are perceptively identical to human eyes.
Output MR: class labels should be identical.

Summary

- Non-testable programs are the programs for which it is very difficult to generate oracles. In some cases, they are the programs written to solve problems, whose correct answers we do not know.
- Metamorphic relationship allows you to formulate a semi-oracle for non-testable programs.
- Incidentally, most of the learning tasks are non-testable: *this is the bleeding edge.*