Seminar *Trends in Computer-Aided Verification*

Introduction

Summer Semester 2022; April 2022

Thomas Noll et al.
Software Modeling and Verification Group
RWTH Aachen University

`https://moves.rwth-aachen.de/teaching/ss-22/cav/`

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Outline

Formal Verification Methods

Formal verification methods

- Rigorous, mathematically based techniques for the specification, development and verification of software and hardware systems
- Aim at improving correctness, reliability and robustness of such systems

Formal Verification Methods

Formal verification methods

- Rigorous, mathematically based techniques for the specification, development and verification of software and hardware systems
- Aim at improving correctness, reliability and robustness of such systems

Classifications

- According to design phase
  - specification, implementation, testing, ...
- According to specification formalism
  - source code, neural networks, Bayesian networks, fault trees, ...
- According to underlying mathematical theories
  - model checking, theorem proving, static analysis, ...

Areas Covered in this Seminar

Topic areas

A. Robustness of Feed-Forward Neural Networks
B. Verification of Recurrent Neural Networks
C. Deductive Program Verification
D. Synthesizing Quantitative Loop Invariants for Probabilistic Programs
E. Safety and Security Assessment

Outline

Goals

Aims of this seminar

- Independent understanding of a scientific topic
- Acquiring, reading and understanding scientific literature
  - given references sufficient in most cases
- Writing of your own report on this topic
  - far more that just a translation/rewording
  - usually an “extended subset” of original literature
    - “subset”: present core ideas and omit too specific details (e.g., related work or optimisations)
    - “extended”: more extensive explanations, examples, ...
    - discuss contents with supervisor!
- Oral presentation of your results
  - can be “proper subset” of report
  - generally: less (detailed) definitions/proofs and more examples

Requirements on Report

Your report

- Independent writing of a report of 12–15 pages
- First milestone: detailed outline
  - not: “1. Introduction/2. Main part/3. Conclusions”
  - rather: overview of structure (section headers, main definitions/theorems) and initial part of main section (one page)
- Complete set of references to all consulted literature
- Correct citation of important literature
- Plagiarism: taking text blocks (from literature or web) without source indication causes immediate exclusion from this seminar
- Font size 12pt with “standard” page layout
  - LaTeX template will be made available on seminar web page
- Language: German or English
- We expect the correct usage of spelling and grammar
  - $\geq$ 10 errors per page $\Longrightarrow$ abortion of correction

Requirements on Talk

Your talk

- Talk of 30 minutes
- Organised as Zoom meeting
- Focus your talk on the audience
- Descriptive slides:
  - $\leq$ 15 lines of text
  - use (base) colors in a useful manner
  - number your slides
- Language: German or English
- No spelling mistakes please!
- Finish in time. Overtime is bad
- Ask for questions
- Have backup slides ready for expected questions
- LaTeX/beamer template will be made available on seminar web page

Outline

Important Dates

Deadlines

- April 11: Topic preferences due
- May 9: Detailed outline due
- June 7: Full report due
- June 27: Presentation slides due
- July 11/12/13 (?): Seminar talks

Important

Missing a deadline causes immediate exclusion from the seminar

Selecting Your Topic

Procedure

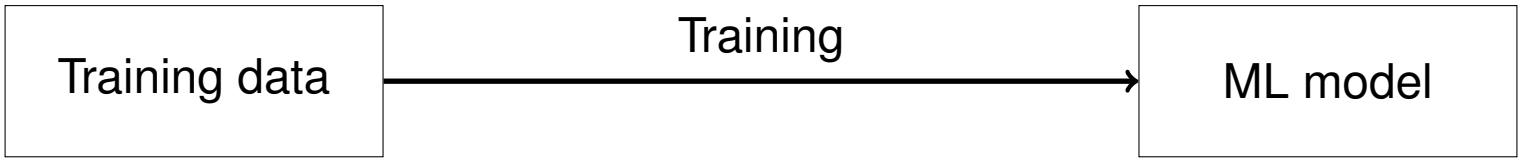
- Check out Foodle poll at `https://terminplaner.dfn.de/CuN7vmys8wI8VCef`
- Topics classified according to BSc/MSc level (“B/M” vs. “M”)
- Please give at least three “Yes” votes ✓
- Preferably additional “Maybe” votes (✓)
- Give as comment:
  - preference of topics (if desired)
  - language of report and talk (English/German)
- **Fill form by Monday, April 11**
- We do our best to find an adequate topic-student assignment
  - disclaimer: no guarantee for an optimal solution
- Assignment of topics and supervisors will be published on web site by mid next week

Withdrawal

- You have up to three weeks to refrain from participating in this seminar.
- Later cancellation (by you or by us) causes a not passed for this seminar and reduces your (three) possibilities by one.

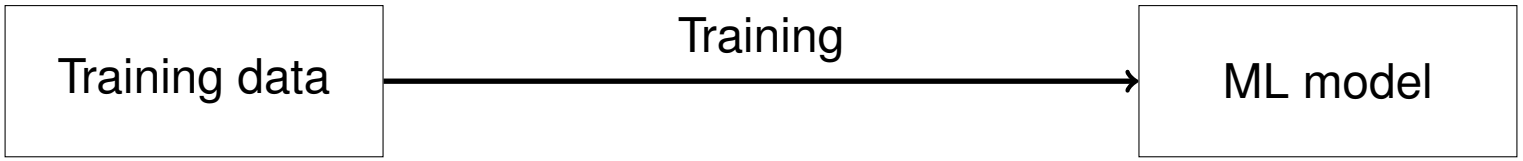
Outline

Neural Networks as Classifiers

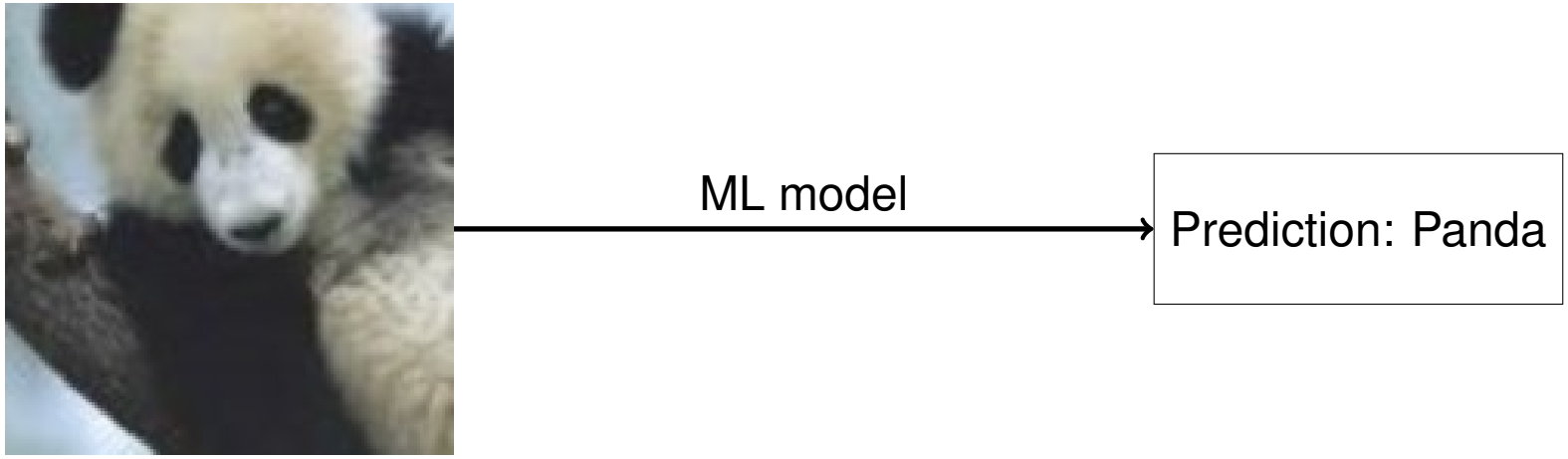


Machine Learning

Neural Networks as Classifiers



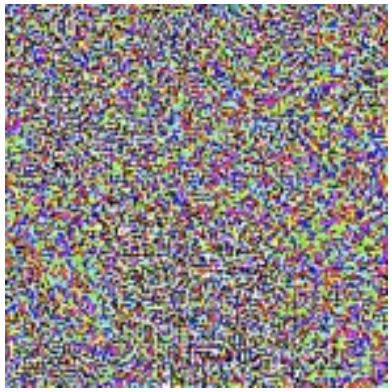
Machine Learning



Inference

Adversarial Examples

$+\ 0.007\ \cdot$

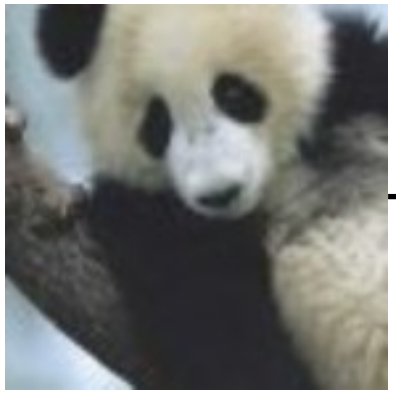


$=$

Adversarial Example [**?**]

Adversarial Examples

Adversarial Example [**?**]



Adversarial Attack

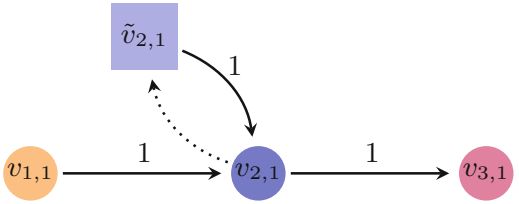
FFNN Topics

1. Detecting Adversarial Samples from Artifacts (R. Feinman et al.)
   - Adversarial examples are outside of the training distribution (OOD)
   - Can we detect adversarial examples the same way as regular OOD data?
2. Outside the box: Abstraction-based monitoring of neural networks (T. Henzinger et al.)
   - Neurons of the network show particular activation patterns for ”expected” inputs (similar to the training data)
   - If they behave significantly different, the input may be unexpected OOD data
3. Globally-Robust Neural Networks (K. Leino et al.)
   - The network is trained such that it outputs “unknown” for inputs that may be adversarial
4. OSIP: Tightened Bound Propagation for the Verification of ReLU Neural Networks (V. Hashemi et al.)
   - Proving the non-existence of adversarial examples requires overapproximations
   - Which approximation to choose can be decided via an LP
5. Sound and Complete Neural Network Repair with Minimality and Locality Guarantees (F. Fu, W. Li)
   - Retraining a network because of an incorrect result is expensive
   - Network repair instead tries to locally fix minor inaccuracies

Outline

Recurrent Neural Networks

- Additional memory units to store information from previous evaluations
- Applications: speech recognition, machine translation, speaker recognition, ...
- Makes verification even more challenging:
  - FFNN: output directly determined by piecewise linear function (with many pieces...)
  - RNN: iteration (WHILE loop)
    - bounded case: unfolding $\rightsquigarrow$ (large) FFNN
    - unbounded case: invariant detection



RNN Topics

1. Verifying Recurrent Neural Networks Using Invariant Inference (Y. Jacoby et al.)
   - Also reduces RNN verification problem to FFNN verification (like unfolding)
   - But independent of number of inputs
   - Generation of inductive invariants that over-approximate behaviour of RNN
2. Verification of recurrent neural networks for cognitive tasks via reachability analysis (H. Zhang et al.)
   - Similar to first paper: generation of inductive invariants
   - Not represented as FFNN but by geometric objects (polytopes)
   - Application to cognitive domain
3. Cert-RNN: Towards certifying the robustness of recurrent neural networks (T. Duet et al.)
   - Certification of robustness against adversarial attacks
   - Based on abstract interpretation using geometric objects (zonotopes)
   - Flexible trade-off between precision and runtime of certification
4. Crafting Adversarial Input Sequences for Recurrent Neural Networks (N. Papernot et al.)
   - Adaptation of algorithms for crafting adversarial examples for FFNN
   - Technique: computational graph unfolding
5. Verification of RNN-Based Neural Agent-Environment Systems (M.E. Akintunde et al.)
   - Agent executing a ReLU RNN
   - Formal verification of Linear-time Temporal Logic (LTL) properties
   - Based on unrolling RNN into FFNN

Outline

Deductive Program Verification

Deductive verifier:

1. Input: A program and a specification
   - e.g., “if input is odd, then the output is odd as well”
2. Automatically generate verification conditions
   - logical formulae as program annotations that ensure correctness
3. Automatically try to prove that the formulae are valid
   - if valid: program satisfies specification
   - otherwise: possible counter-example

Existing deductive verifiers include Prusti, Gobra, Dafny...

All of the following topics are related to building deductive verifiers!

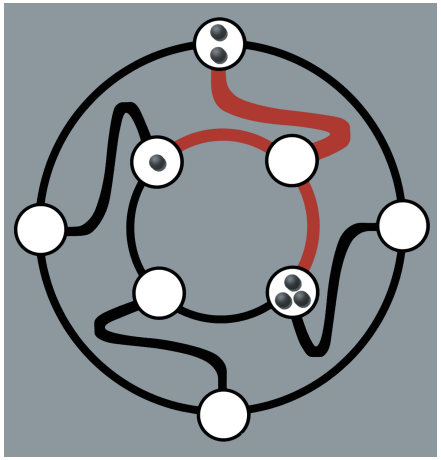
Deductive Program Verification Topics

1. Automatic Inference of Necessary Preconditions (Cousot et al.)
   - Do not try to find sufficient preconditions for correctness, but instead *necessary preconditions for errors*
   - Theory and implementation
2. Expected Runtime Analysis by Program Verification (Kaminski et al.)
   - Theory to reason about expected runtimes of probabilistic programs
3. An Assertion-Based Program Logic for Probabilistic Programs (Barthe et al.)
   - Formal syntax to reason about properties of probabilistic programs
4. Building Deductive Verifiers (Müller)
   - General software architecture of deductive verifiers
5. egg: Fast and extensible equality saturation (Willsey et al.)
   - *e-graphs*: Data structure to find equivalent expressions
   - e.g., $1 = 1 + 0 = 1 + 0 + 0 = \ldots$)
   - Paper is about theory and implementation of e-graphs
6. Formally Validating a Practical Verification Condition Generator (Parthasarathy et al.)
   - How to verify the program verifier?

Outline

Quantitative Loop Invariants

- Reasoning about loops is the hardest task in (probabilistic) program verification.
- "Practical" approach: capture the loop effect by an invariant[a].
- But how to (automatically) find an appropriate loop invariant?
- Constraint-solving based approaches:
  1. Counterexample-Guided Polynomial Loop Invariant Generation by Lagrange Interpolation (Y. Chen *et al.*)
  2. Finding Polynomial Loop Invariants for Probabilistic Programs (Y. Feng *et al.*)
- Martingale-based symbolic method:
  3. Synthesizing Probabilistic Invariants via Doob's Decomposition (G. Barthe *et al.*)
- Moment-based approach by solving recurrences:
  4. Automatic Generation of Moment-Based Invariants for Prob-Solvable Loops (E. Bartocci *et al.*)



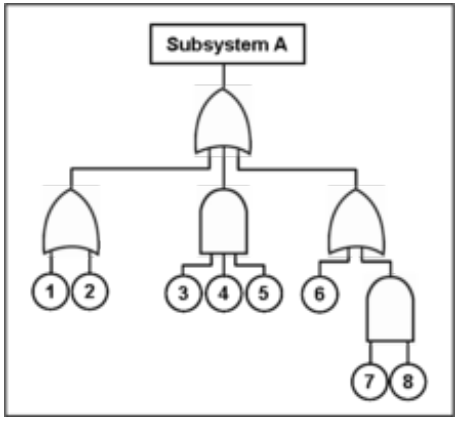
©A. McIver & C. Morgan, 2005

[a] A loop invariant is a property of a loop that is true before and after each iteration.

Outline

Safety and Security Assessment

- Reliability, Availability, Maintainability, and Security (RAMS)
- Fault Trees are a popular modelling formalism in RAMS domain
- Extension to attack trees by taking interaction with attacker into account
- Formal methods are employed to analyse such trees
- We use probabilistic model checking
- Achieving efficiency and scalability is a challenge



Topics:

1. Fast Dynamic Fault Tree Analysis by Model Checking Techniques (M. Volk, S. Junges, J.-P. Katoen)
   - presents various optimisations to perform scalable dynamic fault tree analysis
2. Tensor-based reliability analysis of complex static fault trees (D. Szekeres, K. Marussy, I. Majzik)
   - presents an approach to analyse scalable static fault trees
3. Attack-defense trees (B. Kordy, S. Mauw, S. Radomirovic, P. Schweitzer)
   - surveys various usage scenarios and semantics for attack-defence trees for security applications
4. On the Meaning and Purpose of Attack Trees (H. Mantel, C.W. Probst)
   - also provides a formal treatment of attack trees

Outline

Some Final Hints

Hints

- Take your time to understand your literature.
- Be proactive! Look for additional literature and information.
- Discuss the content of your report with other students.
- Be proactive! Contact your supervisor on time.
- Prepare the meeting(s) with your supervisor.
- Forget the idea that you can prepare a talk in a day or two.

We wish you success and look forward to an enjoyable and high-quality seminar!