Algorithm Selection with Attention on Software Verification

Cedric Richter and Heike Wehrheim[0000−0002−2385−7512]

Paderborn University, Germany

1 Introduction

There is no doubt that *algorithm selection* can boost the performance of software verification tools [3, 8, 5, 4, 6]. An algorithm selector enables the usage of a diverse portfolio of verification algorithms by having the selector match each individual verification problem to a presumably best performing algorithm. As a result, a wider range of programs can be verified by the same verification tool.

Virtually all existing techniques for algorithm selection in software verification (AS4SV) strive for generality. To this end, the characteristics of verification problems (basically, programs) are encoded in a (potentially huge) set of general features. The actual selection procedure is then learned, typically via machine learning, by providing the selector with training data in the form of such features and the best performing algorithm (on programs with these features). Due to the variety of features and the application of learning procedures, there is the hope that the learned selection approach generalizes to arbitrary algorithm portfolios. However, it is not clear whether all relevant characteristics of verification problems are covered or some chosen features might be redundant. For example, the author of [8, 4] showed that the performance of algorithm selection can be increased by incorporating a plethora of general program patterns. In contrast, Beyer et al. [3] demonstrated for a specific algorithm portfolio that simplistic metrics like the existence of a loop or a floating point number are enough for an effective algorithm selector.

To circumvent the issue of manual feature picking, we present a novel *representation learning* technique [2] for AS4SV. More specifically, our method does not map a program to a fixed set of features but *learns* a function from programs to task-specific feature representations. The specialty of our proposed learner is the integration of an *attention* mechanism [9, 1]. Attention allows our selector to focus on different parts of a program, which does not only improve the selection performance but also lets us identify and extract the focused elements.

2 Attention over verification contexts

The use of attention in AS4SV is motivated by the observation that a verification tool which does not support specific parts in a program (e.g., floating-point numbers) will likely also fail to verify the entire program. The starting point of our novel algorithm selector is the construction of an abstract syntax tree (AST) for a program. For discovering the occurrence of important language constructs (e.g., a floating point number), it is enough to collect the labels of AST elements

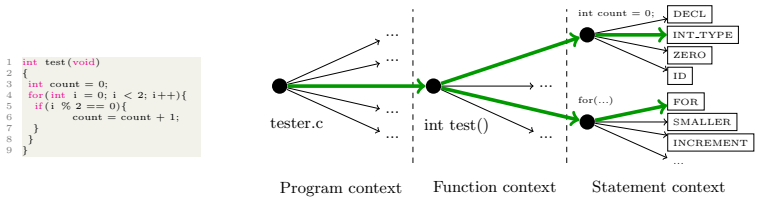

Fig. 1. Verification context hierachy for the C program on the left. Bold edges point to elements with maximal attention. In this example, the selection is made due to the program containing a function which contains a INT_TYP variable and a FOR loop.

(not e.g. their ordering). However, we argue that the importance of an element is highly dependent on its *context*. The occurrence of a large integer within a FOR loop (a context) might be more important to know than within a declaration. In general, we distinguish three types of contexts: program, function and statement context. The program context contains all function contexts while a function context is constructed from a set of statement contexts. A statement context is the set of AST labels used to describe the statement itself while ignoring literals and identifiers. In Figure 1, the three types of verification contexts types are illustrated for a given input program.

Now, our aim is to learn a feature vector representation of a program. For this purpose, we aggregate individual verification contexts into a single vector representation. However, we aim for a parametrizable aggregation function with learnable parameters. At the beginning, every AST label is mapped to a feature vector. This initial mapping is part of the parametrization. By applying the mapping function, a statement context can be transformed to a set of vectors X. To aggregate X into a single vector, we apply the following function:

$$Agg(X;\Theta) = f_\Theta\left(\sum_{x\in X} \alpha_\Theta(x)\cdot x\right),$$

where Θ is the parametrization and f_Θ and α_Θ are differentiable with respect to Θ. Intuitively, $\alpha_\Theta(x)$ (which is optimized during learning) enables us to learn the importance of a single AST label. An element with higher importance has a stronger contribution to the overall sum operator. Because of this property, $\alpha_\Theta(x)$ is commonly refered to as *attention* mechanism [9, 1]: it allows the representation learner to attend to specific elements. Finally, every statement context can be reduced to a single feature vector which results in a new set of vectors for every function context. Hence, we can apply the same mechanism to every function context and afterwards to the overall program context. This leaves us with a representation learner that maps a program to a single feature representation.

As the representation learner is composed of differentiable functions, it can efficiently be optimized by any backpropagation optimizer [7]. In addition, the learner acts as an input and can be optimized with any backpropagation based machine learning technique.

References

1. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: 3rd International Conference on Learning Representations, ICLR 2015 (2015)
2. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence **35**(8), 1798–1828 (2013)
3. Beyer, D., Dangl, M.: Strategy selection for software verification based on boolean features. In: International Symposium on Leveraging Applications of Formal Methods. pp. 144–159. Springer (2018)
4. Czech, M., Hüllermeier, E., Jakobs, M.C., Wehrheim, H.: Predicting rankings of software verification tools. In: Proceedings of the 3rd ACM SIGSOFT International Workshop on Software Analytics. pp. 23–26 (2017)
5. Demyanova, Y., Pani, T., Veith, H., Zuleger, F.: Empirical software metrics for benchmarking of verification tools. Formal Methods in System Design **50**(2-3), 289–316 (2017)
6. Healy, A., Monahan, R., Power, J.F.: Predicting SMT solver performance for software verification. 3rd Workshop on FIDE pp. 20–37 (2017)
7. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation **1**(4), 541–551 (1989)
8. Richter, C., Wehrheim, H.: Pesco: Predicting sequential combinations of verifiers. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 229–233. Springer (2019)
9. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)

Robustness as a Refinement Type

Verifying Neural Networks in Liquid Haskell and F*

Wen Kokke[1,2], Ekaterina Komendantskaya[2],
Daniel Kienitz[2], and David Aspinall[1⋆]

[1] School of Informatics, University of Edinburgh, Edinburgh, UK
{wen.kokke, david.aspinall}@ed.ac.uk
[2] Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK
{dk50, e.komendantskaya}@hw.ac.uk

Abstract. We introduce StarChild and Lazuli, two proof-of-concept libraries which leverage the type system and theorem proving capabilities of F* and Liquid Haskell, respectively, to verify properties of pre-trained neural networks. We largely focus on StarChild, as the F* syntax is slightly more concise, but Lazuli implements the same functionality. Currently, both libraries are capable of verifying small models. Performance issues arise for larger models. Optimising the libraries is future work.
We make two novel contributions. We demonstrate that (a) it is possible to leverage a sufficiently advanced type system to model properties of neural networks such as robustness as types, and check them without any proof burden; and in service of that, we demonstrate that (b) it is possible to approximately translate neural network models to SMT logic.

Introduction Neural networks are widely used for classification and pattern-recognition tasks in computer vision, signal processing, data mining, and many other domains. They have always been valued for their ability to work with noisy data, yet only recently [7], it was discovered that they are prone to adversarial attacks—specially crafted inputs that lead to unexpected outputs. Verifying properties of neural networks, such as, *e.g.*, robustness against adversarial attacks, is a recognised research challenge [4]. Several current approaches involve: (a) encoding properties as satisfiability problems [2,3]; (b) proving properties via abstract interpretation [5]; (c) or using an interactive theorem prover [1].

F* [6] and Liquid Haskell [8] are functional languages with refinement types, *i.e.*, types can be refined with SMT-checkable constraints. For instance, the type of positive reals (`x:ℝ{x > 0}`), or booleans which are true (`b:bool{b ≡ true}`), or a type of neural networks which are robust against adversarial attacks. Unlike, *e.g.*, Python, F* and Liquid Haskell are referentially transparent, which means the semantics of pure programs in these languages can be directly encoded in the SMT logic. This tight integration allows users to specify neural network models and their properties in the same language, while leveraging the powerful automated verification offered by SMT solvers!

⋆ The work was funded by the National Cyber Security Center, UK.

StarChild: Verifying Neural Networks in F* StarChild leverages the type system of F* to verify properties of pre-trained neural networks. Users can either write their models directly in F*, or export them from Python. To illustrate, we train a model to mimic the AND gate, and export it:

```
val m : network (*n_inputs*) 2 (*n_outputs*) 1 (*n_layers*) 1
let m = NLast { weights    = [[17561.5R]; [17561.5R]]
              ; biases     = [−25993.1R]
              ; activation = Sigmoid }
```

We can verify properties of models using either refinement types or assertions. For instance, we can check that the model `m` correctly implements the AND gate:

```
let _ = assert(run m [1.0R;1.0R] ≡ [1.0R]) // true  AND true  ≡ true
let _ = assert(run m [0.0R;1.0R] ≡ [0.0R]) // false AND true  ≡ false
let _ = assert(run m [1.0R;0.0R] ≡ [0.0R]) // true  AND false ≡ false
let _ = assert(run m [0.0R;0.0R] ≡ [0.0R]) // false AND false ≡ false
```

Assertions in F* have no significance at runtime. They are checked statically, as part of type checking. You can think of **assert** as a function with type:

```
val assert : b:bool{b ≡ true} → ()
```

Its argument is a `bool` which must be `true`, which F* checks using an SMT solver. We are not limited to assertions we can run, but can also check assertions using quantifiers, which are infeasible or impossible to run. For instance, we can check that the model `m` is robust for inputs within an ε-interval:

```
let epsilon  = 0.2R
let truthy x = dist x 1.0R ≤ epsilon
let falsy  x = dist x 0.0R ≤ epsilon
let _ = assert(∀(x₁:ℝ{truthy x₁})(x₂:ℝ{truthy x₂}).run m [x₁;x₂] ≡ [1.0R])
let _ = assert(∀(x₁:ℝ{falsy  x₁})(x₂:ℝ{truthy x₂}).run m [x₁;x₂] ≡ [0.0R])
let _ = assert(∀(x₁:ℝ{truthy x₁})(x₂:ℝ{falsy  x₂}).run m [x₁;x₂] ≡ [0.0R])
let _ = assert(∀(x₁:ℝ{falsy  x₁})(x₂:ℝ{falsy  x₂}).run m [x₁;x₂] ≡ [0.0R])
```

The assertions cover the entire ε-interval around 1.0 and 0.0, which we could not have achieved by executing them. The program type checks, and hence we know the model `m` is, in fact, robust for $\varepsilon = 0.2$.

All models specified using StarChild are usable in type refinements and assertions. Better yet, F* takes care of the translation to the SMT logic for us! F* translates programs to the SMT logic by normalising it, translating constructs to their SMT equivalents where possible, and translating the rest as uninterpreted functions. For instance, the expression `run m [x₁;x₂]` normalises to

```
sigmoid(x₁ × 17561.5R + x₂ × 17561.5R − 25993.1R)
```

When translating this term, F* maps ×, +, and − to their equivalent in the SMT logic, and maps maps `sigmoid` to an uninterpreted function. Its definition uses the exponential function, which most SMT solvers do not support. However, the SMT solver cannot reason about uninterpreted functions. To circumvent this, we use linear approximations, *e.g.*, `lsigmoid`, during verification:

```
let lsigmoid x = 0.0R `min` (0.25R × x + 0.5R) `max` 1.0R
```

The use of approximations introduces an error, which impacts the accuracy of the verification. Investigating the bounds on these errors is future work.

References

1. Bagnall, A., Stewart, G.: Certifying true error: Machine learning in Coq with verified generalisation guarantees. AAAI (2019)
2. Katz, G., et al.: The Marabou framework for verification and analysis of deep neural networks. In: CAV 2019, Part I. LNCS, vol. 11561, pp. 443–452. Springer (2019)
3. Kwiatkowska, M.Z.: Safety verification for deep neural networks with provable guarantees (invited paper). In: Fokkink, W., van Glabbeek, R. (eds.) CONCUR 2019,. LIPIcs, vol. 140, pp. 1:1–1:5. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
4. Pertigkiozoglou, S., Maragos, P.: Detecting adversarial examples in convolutional neural networks. CoRR **abs/1812.03303** (2018), http://arxiv.org/abs/1812.03303
5. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. PACMPL **3**(POPL), 41:1–41:30 (2019), https://doi.org/10.1145/3290354
6. Swamy, N., Kohlweiss, M., Zinzindohoue, J.K., Zanella-Béguelin, S., Hriţcu, C., Keller, C., Rastogi, A., Delignat-Lavaud, A., Forest, S., Bhargavan, K., Fournet, C., Strub, P.Y.: Dependent types and multi-monadic effects in F*. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL 2016. ACM Press (2016). https://doi.org/10.1145/2837614.2837655, https://doi.org/10.1145/2837614.2837655
7. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. CoRR **abs/1312.6199** (2014), https://arxiv.org/abs/1312.6199
8. Vazou, N.: Liquid Haskell: Haskell as a Theorem Prover. Ph.D. thesis, University of California, San Diego, USA (2016), http://www.escholarship.org/uc/item/8dm057ws

A Data Driven Approach for Skolem Function Synthesis*

Priyanka Golia[1,2], Kuldeep S. Meel[1], and Subhajit Roy[2]

[1] School of Computing, National University of Singapore, Singapore
[2] Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Abstract. Synthesizing Skolem functions is one of the challenging problems in Computer Science. It has seen multiple proposals, including incremental determination, decomposition techniques from knowledge compilation, and counterexample guided refinement techniques via self-substitutions. In this work, we propose a novel data-driven approach to synthesis that employs constrained sampling techniques for generation of data, machine learning for candidate Skolem functions, and automated reasoning to verify and refine to generate Skolem functions. Our approach achieves significant performance improvements by solving 63/609 benchmarks that could not be solved by any of the existing synthesis tools.

Given a propositional formula $\exists Y F(X,Y)$, Skolem functiona synthesis is to synthesis a function Ψ such that $\exists Y F(X,Y)$ is equivalent to $F(X,\Psi(X))$. It has many applications in different areas like certified QBF solving, automated program repair and synthesis, and cryptography. In this work, we propose a synergistic interaction of learning and formal methods based techniques to synthesize Skolem functions. We design a data-driven approach, Manthan, to synthesize Skolem function that utilizes a novel sampling algorithm to gather data, apply a machine learning algorithm to learn the candidate Skolem functions, and successively refines it via a proof-guided refinement algorithm.

- **Data Generation:** The data here represents the relationship between universally and existentially quantified variables. We use a subset of satisfying assignment of $F(X,Y)$ as data. We view the problem of synthesizing Skolem function as the classification of valuation of the existentially quantified variable over a data. We want to tailor our sampling subroutines to allow the discovery of Skolem functions with *small* description. To this end, we design a novel biased sampling technique that samples the existentially quantified variables at the cost of the universally quantified variables.
- **Learning Candidate Skolem Function:** Manthan learns candidate Skolem functions as decision trees with data projected on universally quantified variables as features, and existentially quantified variables as labels. Candidate

* This is an extended abstract of our paper: Manthan: A Data Driven Approach for Skolem Function Synthesis, CAV-20

Skolem functions can be represented as the disjunction of all the paths from the root to the leaves in the learnt decision tree. We call this the LearnSkF phase of Manthan.

– **Proof Based Refinement:** Manthan uses a MaxSAT solver to identify the erring candidates Skolem functions that may need to be repaired. It utilizes UNSAT core from the infeasibility proofs of the candidate function meeting its specifications to generate a repair formula. Manthan updates the candidate Skolem function with its repair formula. The candidate Skolem functions converges to the actual function through a sequence of such minor repair. We call this the Refine phase of Manthan.

We compared Manthan performance with the state of the art tools, viz. BFSS [3], C2syn [2], and CADET [4] on a set of benchmarks drawn from the datasets QBFEval-17-18 [1], Disjunctive, Factorization and Arithmetic data set [3]. Figure 1 shows that Manthan significantly improves upon state of the art, and solves 356 benchmarks. To put the runtime performance statistics in a broader context, the number of benchmarks solved by techniques developed over the past five years range from 206 to 280, i.e., a difference of 74, which is same as an increase of 76 (i.e., from 280 to 356) due to Manthan; in particular, Manthan solves 60 more benchmarks that could not be solved by any of the tools.

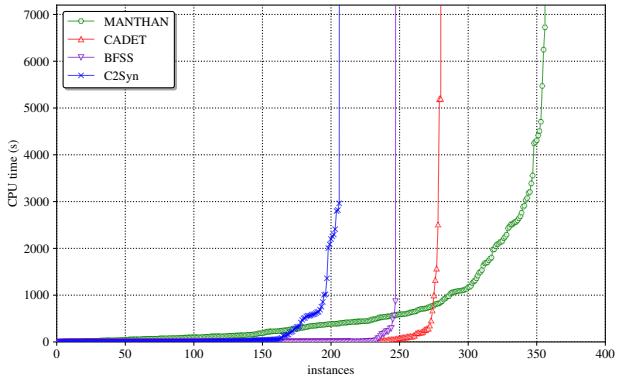

Fig. 1: Manthan vis a vis camparision with state of the art tools

Our approach achieves significant performance improvements and opens up several interesting directions for future work at the intersection of machine learning, constrained sampling, and automated reasoning.

References

1. QBF solver evaluation portal 2018, http://www.qbflib.org/qbfeval18.php
2. Akshay, S., Arora, J., Chakraborty, S., Krishna, S., Raghunathan, D., Shah, S.: Knowledge compilation for boolean functional synthesis. In: Proc. of FMCAD (2019)
3. Akshay, S., Chakraborty, S., Goel, S., Kulal, S., Shah, S.: Whats hard about boolean functional synthesis? In: Proc. of CAV (2018)
4. Rabe, M.N., Tentrup, L., Rasmussen, C., Seshia, S.A.: Understanding and extending incremental determinization for 2QBF. In: Proc. of CAV (2018)

Quantitative Verification of Neural Networks and Its Security Applications*

Teodora Baluta[1], Shiqi Shen[1], Shweta Shinde[1,2], Kuldeep S. Meel[1], and Prateek Saxena[1]
{teobaluta,shiqi04,meel,prateeks}@comp.nus.edu.sg,
shwetas@eecs.berkeley.edu

[1] National University of Singapore, Singapore
[2] University of California, Berkeley, USA

1 Introduction

Neural networks are witnessing wide-scale adoption, including in domains with the potential for a long-term impact on human society. Examples of these domains are criminal sentencing [1], drug discovery [33], self-driving cars [5], aircraft collision avoidance systems [16], robots [4], and drones [11]. While neural networks achieve human-level accuracy in several challenging tasks such as image recognition [18,30,13] and machine translation [29,3], studies show that these systems may behave erratically in the wild [10,23,22,9,32,2,31,28,6].

Consequently, there has been a surge of interest in the design of methodological approaches to verification and testing of neural networks. Early efforts focused on *qualitative* verification wherein, given a neural network N and property P, one is concerned with determining whether there exists an input I to N such that P is violated [24,25,8,21,17,14,7,27]. While such certifiability techniques provide value, for instance in demonstrating the existence of adversarial examples [12,23], it is worth recalling that the designers of neural network-based systems often make a statistical claim of their behavior, i.e., a given system is claimed to satisfy properties of interest with high probability but not always. Therefore, many analyses of neural networks require *quantitative* reasoning, which determines how many inputs satisfy P.

It is natural to encode properties as well as conditions on inputs or outputs as logical formulae. We focus on the following formulation of *quantitative verification*: Given a set of neural networks $\mathcal{N}$ and a property of interest P defined over the union of inputs and outputs of neural networks in $\mathcal{N}$, we are interested in estimating how often P is satisfied. In many critical domains, client analyses often require guarantees that the computed estimates be reasonably close to the ground truth. We are not aware of any prior approaches that provide such formal guarantees, though the need for quantitative verification has recently been recognized [34].

* Work published in ACM Conference on Computer and Communications Security (CCS) 2019. Code and benchmarks are available at https://teobaluta.github.io/npaq/.

Our Approach. The primary contribution of this paper is a new analysis framework, which models the given set of neural networks $\mathcal{N}$ and P as set of logical constraints, φ, such that the problem of quantifying how often $\mathcal{N}$ satisfies P reduces to model counting over φ. We then show that the quantitative verification is $\#P$-hard. Given the computational intractability of $\#P$, we seek to compute rigorous estimates and introduce the notion of *approximate quantitative verification*: given a prescribed tolerance factor ε and confidence parameter δ, we estimate how often P is satisfied with PAC-style guarantees, i.e., the computed result is within a multiplicative $(1+\varepsilon)$ factor of the ground truth with confidence at least $1-\delta$.

Our approach works by encoding the neural network into a logical formula in conjunctive normal form (CNF). The key to achieving soundness guarantees is our new notion of equicardinality, which defines a principled way of encoding neural networks into a CNF formula F, such that quantitative verification reduces to counting the satisfying assignments of F projected to a subset of the support of F. We then use approximate model counting on F, which has seen rapid advancement in practical tools that provide PAC-style guarantees on counts for F. The end result is a *quantitative verification procedure for neural networks with soundness and precision guarantees.*

While our framework is more general, we instantiate our analysis framework with a sub-class of neural networks called binarized neural networks (or BNNs) [15]. BNNs are multi-layered perceptrons with `+/-1` weights and step activation functions. They have been demonstrated to achieve high accuracy for a wide variety of applications [26,20,19]. Due to their small memory footprint and fast inference time, they have been deployed in constrained environments such as embedded devices [20,19]. We observe that specific existing encodings for BNNs adhere to our notion of equicardinality and implement these in a new tool called NPAQ. We provide proofs of key correctness and composability properties of our general approach, as well as of our specific encodings. Our encodings are linear in the size of $\mathcal{N}$ and P.

Empirical Results. We show that NPAQ scales to BNNs with $1-3$ internal layers and $20-200$ units per layer. We use 2 standard datasets namely MNIST and UCI Adult Census Income dataset. We encode a total of 84 models with $4,692-53,010$ parameters, into $1,056$ formulae and quantitatively verify them. NPAQ encodes properties in less than a minute and solves 97.1% formulae in a 24-hour timeout. Encodings scale linearly in the size of the models, and its running time is not dependent on the true counts. We showcase how NPAQ can be used in diverse security applications with case studies. First, we quantify the model robustness by measuring how many adversarially perturbed inputs are misclassified, and then the effectiveness of 2 defenses for model hardening with adversarial training. Next, we evaluate the effectiveness of trojan attacks outside the chosen test set. Lastly, we measure the influence of 3 sensitive features on the output and check if the model is biased towards a particular value of the sensitive feature.

References

1. Correctional offender management profiling for alternative sanctions. http://www.northpointeinc.com/files/downloads/FAQ_Document.pdf (2012)
2. Athalye, A., Carlini, N., Wagner, D.: Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In: ICML'18 (2018)
3. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: ICLR'15 (2015)
4. Bhattacharyya, S., Cofer, D., Musliner, D., Mueller, J., Engstrom, E.: Certification considerations for adaptive systems. In: ICUAS'15 (2015)
5. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Others: End to end learning for self-driving cars. arXiv (2016)
6. Carlini, N., Liu, C., Kos, J., Erlingsson, Ú., Song, D.: The secret sharer: Measuring unintended neural network memorization & extracting secrets. arXiv (2018)
7. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T., Kohli, P.: A Dual Approach to Scalable Verification of Deep Networks. In: UAI'18 (2018)
8. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: ATVA'17 (2017)
9. Evtimov, I., Eykholt, K., Fernandes, E., Kohno, T., Li, B., Prakash, A., Rahmati, A., Song, D.: Robust physical-world attacks on deep learning models. In: CVPR'18 (2018)
10. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: CCS'15 (2015)
11. Giusti, A., Guzzi, J., Ciresan, D.C., He, F.L., Rodriguez, J.P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Caro, G.D., Scaramuzza, D., Gambardella, L.M.: A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots. IEEE Robotics and Automation Letters (2016)
12. Goodfellow, I., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: ICLR'15 (2015)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR'16 (2016)
14. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: CAV'17 (2017)
15. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: NIPS'16 (2016)
16. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: DASC'16 (2016)
17. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: CAV'17 (2017)
18. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS'12 (2012)
19. Kung, J., Zhang, D., van der Wal, G., Chai, S., Mukhopadhyay, S.: Efficient object detection using embedded binarized neural networks. Journal of Signal Processing Systems (2018)
20. McDanel, B., Teerapittayanon, S., Kung, H.T.: Embedded Binarized Neural Networks. In: EWSN'17 (2017)
21. Narodytska, N., Kasiviswanathan, S.P., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. In: AAAI'18 (2018)

22. Papernot, N., McDaniel, P., Goodfellow, I.: Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. arXiv (2016)
23. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: EuroS&P'16 (2016)
24. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. In: SOSP'17 (2017)
25. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: CAV'10 (2010)
26. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision (2016)
27. Shih, A., Darwiche, A., Choi, A.: Verifying binarized neural networks by angluin-style learning. In: SAT'19 (2019)
28. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: SP'17 (2017)
29. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: NIPS'14 (2014)
30. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: CVPR'15 (2015)
31. Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., McDaniel, P.: Ensemble adversarial training: Attacks and defenses. In: ICLR'18 (2018)
32. Uesato, J., O'Donoghue, B., Oord, A.v.d., Kohli, P.: Adversarial risk and the dangers of evaluating against weak attacks. In: ICML'18 (2018)
33. Wallach, I., Dzamba, M., Heifets, A.: Atomnet: A deep convolutional neural network for bioactivity prediction in structure-based drug discovery. arXiv (2015)
34. Webb, S., Rainforth, T., Teh, Y.W., Kumar, M.P.: A statistical approach to assessing neural network robustness. In: ICLR'19 (2019)

Neural Predictive Monitoring and a Comparison of Frequentist and Bayesian Approaches

Luca Bortolussi[1,4], Francesca Cairoli[1], Nicola Paoletti[2], Scott A. Smolka[3], and Scott D. Stoller[3]

[1] Department of Mathematics and Geosciences, Università di Trieste, Italy
[2] Department of Computer Science, Royal Holloway, University of London, UK
[3] Department of Computer Science, Stony Brook University, USA
[4] Modelling and Simulation Group, Saarland University, Germany

In this work we address the online analysis of hybrid systems and, in particular, the *predictive monitoring* (PM) problem [3]; i.e., the problem of predicting, *at runtime*, whether or not an unsafe state can be reached from the current system state within a given time bound. PM is at the core of architectures for runtime safety assurance such as Simplex [8], where the system switches to a safe fallback mode whenever PM indicates the potential for an imminent safety violation.

Neural State Classification (NSC) [7] is a recently proposed method for runtime predictive monitoring of Hybrid Automata (HA) using deep neural networks (DNNs). NSC trains a DNN as an approximate *reachability predictor* that labels an HA state x as *positive* if an unsafe state is reachable from x within a given time bound, and labels x as *negative* otherwise. NSC predictors have very high accuracy, yet are prone to prediction errors that can negatively impact reliability.

To overcome this limitation, we present *Neural Predictive Monitoring* (NPM), a technique that complements NSC predictions with estimates of the predictive uncertainty. These measures yield principled criteria for the rejection of predictions likely to be incorrect, without knowing the true reachability values. We also present an active learning method, which is guided by the prediction uncertainty measures, that significantly reduces the NSC predictor's error rate and the percentage of rejected predictions. We develop two versions of NPM based respectively on the use of frequentist and Bayesian techniques to learn the predictor and the rejection rule. The frequentist approach uses *Conformal Prediction* (CP) [9], a method that provides statistical guarantees on the predictions of machine-learning models. The *Bayesian approach* leverages uncertainty quantification via *Bayesian neural networks (BNNs)* and two Bayesian inference methods: Hamiltonian Monte Carlo [6] and Variational Inference [5].

Figure 1 provides an overview of the NPM approach. We sample from a distribution of HA states to generate a training set Z_t and a validation set Z_v. An HA reachability oracle (a model checker [4, 2] or, for deterministic systems, a simulator) is used to label sampled states as positive or negative. A neural state classifier F (i.e., a DNN-based binary classifier) is derived from Z_t via supervised learning, and is either a deterministic Neural Network (in the frequentist approach) or a Bayesian Neural Network (in the Bayesian approach).

Our approach is highly efficient, with computation times on the order of milliseconds, and effective, managing in our experimental evaluation to successfully reject almost all incorrect predictions. In our experiments on a benchmark suite of six

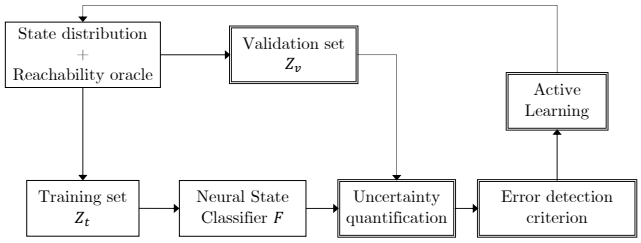

Fig. 1. Overview of the NPM framework. Double-bordered components denote extensions to the method of [7]. Training of the neural state classifier F and retraining via active learning are performed offline. The only components used at runtime are the classifier F and the error-detection criterion.

hybrid systems, we found that the frequentist approach consistently outperforms the Bayesian one. We also observed that the Bayesian approach is less practical, requiring a careful and problem-specific choice of hyperparameters.

This work is currently under submission to the RV19 special issue and is an extended version of [1], where we first introduced the NPM method but only in the frequentist version. Here, we introduce a fully Bayesian variant of NPM, and compare the two in a new experimental evaluation section.

References

1. Bortolussi, L., Cairoli, F., Paoletti, N., Smolka, S.A., Stoller, S.D.: Neural predictive monitoring. In: International Conference on Runtime Verification. pp. 129–147. Springer (2019)
2. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: International Conference on Computer Aided Verification. pp. 258–263. Springer (2013)
3. Chen, X., Sankaranarayanan, S.: Model predictive real-time monitoring of linear systems. In: Real-Time Systems Symposium (RTSS), 2017 IEEE. pp. 297–306. IEEE (2017)
4. Gao, S., Kong, S., Clarke, E.M.: dreal: An smt solver for nonlinear theories over the reals. In: International conference on automated deduction. pp. 208–214. Springer (2013)
5. Jordan, M.I., Ghahramani, Z., Jaakkola, T.S., Saul, L.K.: An introduction to variational methods for graphical models. Machine learning **37**(2), 183–233 (1999)
6. Neal, R.M., et al.: MCMC using Hamiltonian dynamics. Handbook of markov chain monte carlo **2**(11), 2 (2011)
7. Phan, D., Paoletti, N., Zhang, T., Grosu, R., Smolka, S.A., Stoller, S.D.: Neural state classification for hybrid systems. In: Automated Technology for Verification and Analysis. Lecture Notes in Computer Science, vol. 11138, pp. 422–440 (2018). https://doi.org/10.1007/978-3-030-01090-4_25
8. Sha, L.: Using simplicity to control complexity. IEEE Software **18**(4), 20–28 (2001)
9. Vovk, V., Gammerman, A., Shafer, G.: Algorithmic learning in a random world. Springer Science & Business Media (2005)

Semantic Labelling and Learning for Parity Game Solving in LTL Synthesis

Jan Křetínský[0000−0002−8122−2881], Alexander Manta[0000−0003−0591−6985], and Tobias Meggendorfer[0000−0002−1712−2165]

Technical University of Munich

This work [KMM19] has been presented at ATVA 2019.

Reactive synthesis is a classical problem [Chu63,PR89,RW87] to find a strategy that given a stream of inputs gradually produces a stream of outputs so that a given specification over the inputs and outputs is satisfied. In LTL synthesis [PR89] the specification is given as a formula of *linear temporal logic (LTL)* [Pnu77]. A classical solution technique is the *automata-theoretic approach* [Büc62,VW86] that transforms the specification into an automaton. The partitioning of atomic propositions into inputs and outputs then yields a game over this automaton. Subsequently, the game is solved and the winning strategy in the game induces a winning strategy for the original problem. The standard type of automaton to be used in this context is the *deterministic parity automaton (DPA)* since (i) determinism ensures we obtain a well-defined game and (ii) the parity condition yields a *parity game*, which can be solved reasonably cheaply in practice [Zie98,Sch07,Fea17,CJK+17,FJS+17] with good tool support [FL09,vD18].

We propose *semantic labelling* as a novel ingredient for solving such games in the context of LTL synthesis. It exploits recent advances [EK14,EKS18,KMSZ18] in the automata-based approach, yielding more information for each state of the generated parity game than the game graph captures. In particular, we essentially obtain a description of each state in terms of a single formula to be satisfied and a list of formulae describing progress of satisfying each sub-goal. This clearer structure allows us to exploit the meaning of available successors and to choose the most promising one in the sense of satisfying the goal of each player. Moreover, since the degree how promising a vertex is can be quantified, we can use this information as a *reward* in order to guide computation in spirit of *reinforcement learning* [SB18]. This way we update only the most promising parts of the state space.

This leads to the following improvements:

1. Compared to strategy improvement (SI) with random initial strategy, a more informed initialization often yields a winning strategy directly without any computation.
2. This initialization makes SI also yield smaller solutions.
3. While Q-learning on the game graph turns out not too efficient, Q-learning based on semantic information becomes comparable to SI.

Since already the simplest heuristics achieve significant improvements the experimental results demonstrate the utility of semantic labelling. This extra information opens the door to more advanced learning approaches both for initialization and improvement of strategies.

References

Büc62. J.R. Büchi. On a decision method in restricted second-order arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Proceedings of the First International Congress on Logic, Methodology, and Philosophy of Science 1960*, 1962.

Chu63. Alonzo Church. Application of recursive arithmetic to the problem of circuit synthesis. *Journal of Symbolic Logic*, 1963.

CJK+17. Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *STOC*, 2017.

EK14. Javier Esparza and Jan Kretínský. From LTL to deterministic automata: A safraless compositional approach. In *CAV*, 2014.

EKS18. Javier Esparza, Jan Kretínský, and Salomon Sickert. One theorem to rule them all: A unified translation of LTL into ω-automata. In *LICS*, 2018.

Fea17. John Fearnley. Efficient parallel strategy improvement for parity games. In *CAV*, 2017.

FJS+17. John Fearnley, Sanjay Jain, Sven Schewe, Frank Stephan, and Dominik Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In *SPIN*, 2017.

FL09. Oliver Friedmann and Martin Lange. Solving parity games in practice. In *ATVA*, 2009.

KMM19. Jan Kretínský, Alexander Manta, and Tobias Meggendorfer. Semantic labelling and learning for parity game solving in LTL synthesis. In Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors, *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*, pages 404–422. Springer, 2019.

KMSZ18. Jan Kretínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to your favourite deterministic automaton. In *CAV*, 2018.

Pnu77. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, 1977.

PR89. A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *ICALP*, 1989.

RW87. P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 1987.

SB18. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2018.

Sch07. Sven Schewe. Solving parity games in big steps. In *FSTTCS*, 2007.

vD18. Tom van Dijk. Oink: An implementation and evaluation of modern parity game solvers. In *TACAS*, 2018.

VW86. Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, 1986.

Zie98. Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 1998.

Formally Verifying the Robustness of Multiple-Classifier Combinations

Dennis Gross[1], Nils Jansen[1], and Guillermo A. Pérez[2]

[1] Radboud University Nijmegen, The Netherlands
[2] University of Antwerp, Belgium

Each year millions of people die through car accidents [10]. Studies indicate that self-driving cars make around 80 % fewer traffic mistakes than human drivers [9]. However, those self-driving car, or autonomous systems in general, largely rely on machine learning methods. Despite many groundbreaking developments, a major open challenge in machine learning is verifiable *safety*, prominently raised by, amongst others, [13, 6, 12, 2].

In the context of self-driving cars, for instance, certain camera data may contain noise. which can be introduced randomly or actively via so-called adversarial attacks. We focus on the particular problem of such attacks in image classification. A successful attack perturbs the original image in a way such that a human eye does not see any difference while the machine learning classifier *misclassifies* the image.

One way to render image classification more robust is to use a set of classifiers, also referred to as *classifier ensemble* [1, 11]. The underlying idea is to obscure the current classifier from the attacker. For such ensembles, we consider so-called *randomized attacks*, that is, a set of attacks that are randomly chosen by means of fixed probability distributions [11]. Such an attack is called optimal if the *expected loss* is maximized regardless of the choice of classifier [11].

Inspired by previous approaches for single classifiers [8, 5], we aim to develop a formal verification procedure that decides if a set of classifiers is *robust* against any randomized attack. In particular, the formal problem is the following. Given a set of classifiers and a labelled data set, we want to find a probability distribution and a set of attacks that induces an optimal randomized attack. Akin to the setting in [11], we provide thresholds on potential perturbations of data points and the minimum shift in classification values. Thereby, it may happen that no optimal attack exists, in which case we call the classifier ensemble *robust*.

We establish a number of theoretical results. First, we show that the underlying formal problem is NP-hard. Towards computational tractability, we also show that for an optimal attack there exists an upper bound on the number of attacks that are needed. Using these results, we provide an SMT encoding that computes suitable randomized attacks for a set of convolutional neural networks with ReLu activation functions and a labeled data set. In case there is no solution to that problem, the set of neural networks forms a robust classifier ensemble, see Fig. 1. Towards better scalability, we also provide an encoding based on mixed-integer linear programming (MILP).

In our experiments, we show the applicability of our approach by means of a benchmark set of binary classifiers, which were trained with the MNIST dataset

(handwritten digits [4]). This data set is commonly used for benchmark testing in machine learning. We use the SMT solver Z3 [3] and the MILP solver Gurobi [7].

Fig. 1. The verifier takes as input a set of classifiers, a set of labelled data points, the number of attacks, and the attack properties. If the verifier does not find a solution, we can be sure is robust against any attack with the specific properties. Otherwise, it returns the optimal attack.

References

1. Abbasi, M., Gagné, C.: Robustness to adversarial examples through an ensemble of specialists. In: ICLR (Workshop). OpenReview.net (2017)
2. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., Mané, D.: Concrete problems in ai safety. CoRR **abs/1606.06565** (2016)
3. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
4. Deng, L.: The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Processing Magazine **29**(6), 141–142 (2012)
5. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: ATVA. LNCS, vol. 10482, pp. 269–286. Springer (2017)
6. Freedman, R.G., Zilberstein, S.: Safety in ai-hri: Challenges complementing user experience quality. In: AAAI Fall Symposium Series (2016)
7. Gurobi Optimization, Inc.: Gurobi optimizer reference manual. http://www.gurobi.com (2013)
8. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: CAV. LNCS, vol. 10426, pp. 97–117. Springer (2017)
9. Nyholm, S.: The ethics of crashes with self-driving cars: A roadmap, ii. Philosophy Compass **13**(7) (2018)
10. Organization, W.H., et al.: Global status report on road safety 2018: Summary. Tech. rep., World Health Organization (2018)
11. Perdomo, J.C., Singer, Y.: Robust attacks against multiple classifiers. CoRR **abs/1906.02816** (2019)
12. Science, N., (NSTC), T.C.: Preparing for the Future of Artificial Intelligence (2016)

13. Stoica, I., Song, D., Popa, R.A., Patterson, D., Mahoney, M.W., Katz, R., Joseph, A.D., Jordan, M., Hellerstein, J.M., Gonzalez, J.E., et al.: A berkeley view of systems challenges for ai. CoRR **abs/1712.05855** (2017)

Global PAC Bounds for Learning Discrete Time Markov Chains

anonym

anonym
anonym@anonym

Abstract Learning models from observations of a system is a powerful tool with many applications. In this paper, we consider learning Discrete Time Markov Chains (DTMC), with different methods such as *frequency estimation* or *Laplace smoothing*. While models learnt with such methods converge asymptotically towards the exact system, a more practical question in the realm of trusted machine learning is how accurate a model learnt with a limited time budget is. Existing approaches provide bounds on how close the model is to the original system, in terms of bounds on *local* (transition) probabilities, which has unclear implication on the *global* behavior.
In this work, we provide *global bounds on the error* made by such a learning process, in terms of global behaviors formalized using *temporal logic*. More precisely, we propose a learning process ensuring a bound on the error in the probabilities of these properties. While such learning process cannot exist for the full LTL logic, we provide one ensuring a bound that is uniform over all the formulas of CTL. Further, given one time-to-failure property, we provide an improved learning algorithm. Interestingly, frequency estimation is sufficient for the latter, while Laplace smoothing is needed to ensure non-trivial uniform bounds for the full CTL logic.

1 Introduction

Discrete-Time Markov Chains (DTMC) are commonly used in model checking to model the behavior of stochastic systems [3,4,7,25]. A DTMC is described by a set of states and transition probabilities between these states. The main issue with modeling stochastic systems using DTMCs is to obtain the transition probabilities. One appealing approach to overcome this issue is to observe the system and to *learn automatically* these transition probabilities [8,29], e.g., using frequency estimation or Laplace (or additive) smoothing [12]. Frequency estimation works by observing a long run of the system and estimating each individual transition by its empirical frequency. However, in this case, the unseen transitions are estimated as zeros. Once the probability of a transition is set to zero, the probability to reach a state could be tremendously changed, e.g., from 1 to 0 if the probability of this transition in the system is small but non-zero. To overcome this problem, when the set of transitions with non-zero probability is known (but not their probabilities), Laplace smoothing assigns a positive

probability to the unseen transitions, i.e., by adding a small quantity both to the numerator and the denominator of the estimate used in frequency estimation. Other smoothing methods exist, such as Good-Turing [15] and Kneser-Sey estimations [7], notably used in natural language processing. Notwithstanding smoothing generates estimation biases, all these methods converge asymptotically to the exact transition probabilities.

In practice, however, there is often limited budget in observing and learning from the system, and the validity of the learned model is in question. In trusted machine learning, it is thus crucial to measure how the learned model differs from the original system and to provide practical guidelines (e.g., on the number of observations) to guarantee some control of their divergence.

Comparing two Markov processes is a common problem that relies on a notion of divergence. Most existing approaches focus on deviations between the probabilities of local transitions (e.g., [10,26,5]). However, a single deviation in a transition probability between the original system and the learned model may lead to large differences in their global behaviors, even when no transitions are overlooked, as shown in our example 1. For instance, the probability of reaching certain state may be magnified by paths which go through the same deviated transition many times. It is thus important to use a measure that quantifies the differences over global behaviors, rather than simply checking whether the differences between the individual transition probabilities are low enough.

Technically, the knowledge of a lower bound on the transition probabilities is often assumed [14,1]. While it is a soft assumption in many cases, such as when all transition probabilities are large enough, it is less clear how to obtain such a lower bound in other cases, such as when a very unlikely transition exists (e.g., a very small error probability). We show how to handle this in several cases: learning a Markov chain accurate w.r.t. this error rate, or learning a Markov chain accurate over all its global behaviors, which is possible if we know the underlying structure of the system (e.g., because we designed it, although we do not know the precise transition probabilities which are governed by uncertain forces). For the later, we define a new concept, namely *conditioning* of a DTMC.

In this work, we model global behaviors using temporal logics. We consider Linear Temporal Logic (LTL) [23] and Computational Tree Logic (CTL) [11]. Agreeing on all formulas of LTL means that the first order behaviors of the system and the model are the same, while agreeing on CTL means that the system and the model are bisimilar [2]. Our goal is to provide stopping rules in the learning process of DTMCs that provides Probably Approximately Correct (PAC) bounds on the error in probabilities of every property in the logic between the model and the system. In Section 2, we recall useful notions on DTMCs and PAC-learning. We point out related works in Section 3. Our main contributions are as follows:

- In Section 4, we show that it is impossible to learn a DTMC accurate for all LTL formulas, by adapting a result from [13].

– We provide in Section 6 a learning process bounding the difference in probability *uniformly over all CTL properties*. To do so, we use Laplace smoothing, and we provide rationale on choosing the smoothing parameter.
– For the particular case of a time-to-failure property, notably used to compute the mean time between failures of critical systems (see e.g., [24]), we provide tighter bounds in Section 5, based on frequency estimation.

In Section 4, we formally state the problem and the specification that the learning process must fulfill. We also show our first contribution: the impossibility of learning a DTMC, accurate for all LTL formulas. Nevertheless, we prove in Section 5 our second contribution: the existence of a global bound for the time-to-failure properties, notably used to compute the mean time between failures of critical systems (see e.g., [24]) and provide an improved learning process, based on frequency estimation. In Section 6, we present our main contribution: a global bound guaranteeing that the original system and a model learned by Laplace smoothing have similar behaviors for all the formulas in CTL. We show that the error bound that we provide on the probabilities of properties is close to optimal. We evaluate our approach in Section 7 and conclude in Section 8.

2 Background

In this section, we introduce the notions and notations used throughout the paper. A stochastic system $\mathcal{S}$ is interpreted as a set of interacting components in which the state is determined randomly with respect to a global probability measure described below.

Definition 1 (Discrete-Time Markov Chains). *A Discrete-Time Markov Chain is a triple $\mathcal{M} = (S, \mu, A)$ where:*

– *S is a finite set of states;*
– *$\mu : S \to [0, 1]$ is an initial probability distribution over S;*
– *$A : S \times S \to [0, 1]$ is a transition probability matrix, such that for every $s \in S$, $\sum_{s' \in S} A(s, s') = 1$.*

We denote m the cardinal of S and $A = (a_{ij})_{1 \leq i,j \leq m} = (A(i, j))_{1 \leq i,j \leq m}$ the probability matrix. Figs. 1 and 2 show the graph of two DTMCs over 3 states $\{s_1, s_2, s_3\}$ (with $\mu(s_1) = 1$). A run is an infinite sequence $\omega = s_0 s_1 \cdots$ and a path is a finite sequence $\omega = s_0 \cdots s_l$ such that $\mu(s_0) > 0$ and $A(s_i, s_{i+1}) > 0$ for all i, $0 \leq i \leq l$. The length $|\omega|$ of a path ω is its number of transitions.

The cylinder set of ω, denoted $C(\omega)$, consists of all the runs starting by a path ω. Markov chain $\mathcal{M}$ underlies a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is the set of all runs from $\mathcal{M}$; $\mathcal{F}$ is the sigma-algebra generated by all the cylinders $C(\omega)$ and $\mathbb{P}$ is the unique probability measure [31] such that $\mathbb{P}(C(s_0 \cdots s_l)) = \mu(s_0) \prod_{i=1}^{l} A(s_{i-1}, s_i)$. For simplicity, we assume a unique initial state s_0 and denote $\mathbb{P}(\omega) = \mathbb{P}(C(\omega))$. Finally, we sometimes use the notation $\mathbb{P}_i^A$ to emphasize that the probability distribution is parameterized by the probability matrix A, and the starting state is i.

2.1 PAC-learning for properties

To analyze the behavior of a system, properties are specified in temporal logic (e.g., LTL or CTL, respectively introduced in [23] and [11]). Given a logic $\mathcal{L}$ and φ a property of $\mathcal{L}$, decidable in finite time, we denote $\omega \models \varphi$ if a path ω satisfies φ. Let $z : \Omega \times \mathcal{L} \to \{0, 1\}$ be the function that assigns 1 to a path ω if $\omega \models \varphi$ and 0 otherwise. In what follows, we assume that we have a procedure that draws path ω with respect to $\mathbb{P}^A$ and outputs $z(\omega, \varphi)$. Further, we denote $\gamma(A, \varphi)$ the probability that a path drawn with respect to $\mathbb{P}^A$ satisfies φ. We omit the property or the matrix in the notation when it is clear from the context. Finally, note that the behavior of $z(., \varphi)$ can be modeled as a Bernoulli random variable Z_φ parameterized by the mean value $\gamma(A, \varphi)$.

Probably Approximately Correct (PAC) learning [27] is a framework for mathematical analysis of machine learning. Given $\varepsilon > 0$ and $0 < \delta < 1$, we say that a property φ of $\mathcal{L}$ is PAC-learnable if there is an algorithm $\mathcal{A}$ such that, given a sample of n paths drawn according to the procedure, with probability of at least $1-\delta$, $\mathcal{A}$ outputs in polynomial time (in $1/\varepsilon$ and $1/\delta$) an approximation of the average value for Z_φ close to its exact value, up to an error less than or equal to ε. Formally, φ is PAC-learnable if and only if $\mathcal{A}$ outputs an approximation $\hat{\gamma}$ such that:

$$\mathbb{P}\left(|\gamma - \hat{\gamma}| > \varepsilon\right) \leq \delta \tag{1}$$

Moreover, if the above statement for algorithm $\mathcal{A}$ is true for every property in $\mathcal{L}$, we say that $\mathcal{A}$ is a PAC-learning algorithm for $\mathcal{L}$.

2.2 Monte-Carlo estimation and algorithm of Chen

Given a sample W of n paths drawn according to $\mathbb{P}^A$ until φ is satisfied or violated (for φ such that with probability 1, φ is eventually satisfied or violated), the crude Monte-Carlo estimator, denoted $\hat{\gamma}_W(A, \varphi)$, of the mean value for the random variable Z_φ is given by the empirical frequency: $\hat{\gamma}_W(A, \varphi) = \frac{1}{n}\sum_{i=1}^{n} z(\omega_i) \approx \gamma(A, \varphi)$.

The Okamoto inequality [22] (also called the Chernoff bound in the literature) is often used to guarantee that the deviation between a Monte-Carlo estimator $\hat{\gamma}_W$ and the exact value γ by more than $\varepsilon > 0$ is bounded by a predefined confidence parameter δ. However, several sequential algorithms have been recently

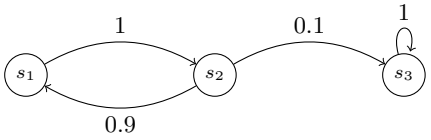

Figure 1: An example of DTMC $\mathcal{M}_1$

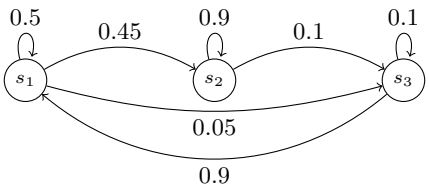

Figure 2: DTMC $\mathcal{M}_2$

proposed to guarantee the same confidence and accuracy with fewer samples[1]. In what follows, we use the Massart bound [**?**], implemented in the algorithm of Chen [6].

Theorem 1 (Chen bound). *Let $\varepsilon > 0$, δ such that $0 < \delta < 1$ and $\hat{\gamma}_W$ be the crude Monte-Carlo estimator, based on n samples, of probability γ.*

If $n \geq \frac{2}{\varepsilon^2} \log\left(\frac{2}{\delta}\right) \left[\frac{1}{4} - (|\frac{1}{2} - \hat{\gamma}_W| - \frac{2}{3}\varepsilon)^2\right]$,

$$\mathbb{P}(|\gamma - \hat{\gamma}_W| > \varepsilon) \leq \delta.$$

To ease the readability, we write $n_{\text{succ}} = \sum_{i=1}^{n} z(\omega_i)$ and $H(n, n_{\text{succ}}, \epsilon, \delta) = \frac{2}{\varepsilon^2} \log\left(\frac{2}{\delta}\right) \left[\frac{1}{4} - (|\frac{1}{2} - \hat{\gamma}_W| - \frac{2}{3}\varepsilon)^2\right]$. When it is clear from the context, we only write $H(n)$. Then, the algorithm $\mathcal{A}$ that stops sampling as soon as $n \geq H(n)$ and outputs a crude Monte-Carlo estimator for $\gamma(A, \varphi)$ is a PAC-learning algorithm for φ. The condition over n is called the stopping criteria of the algorithm. As far as we know, this algorithm requires fewer samples than the other sequential algorithms (see e.g., [18]). Note that the estimation of a probability close to $1/2$ likely requires more samples since $H(n)$ is maximized in $\hat{\gamma}_W = 1/2$.

3 Related work

Our work shares similar statistical results (see Section 2.3) with Statistical Model Checking (SMC) [31]. However, the context and the outputs are different. SMC is a simulation-based approach that aims to estimate one probability for a given property [9,28], within acceptable margins of error and confidence [17,18,32]. A challenge in SMC is posed by unbounded properties (e.g., fairness) since the sampled executions are finite. Some algorithms have been proposed to handle unbounded properties but they require the knowledge of the minimal probability transition of the system [14,1], which we avoid. While this restriction is light in many contexts, such as when every state and transition appears with a sufficiently high probability, contexts where probabilities are unknown and some are very small seems much harder to handle. In the following, we propose 2 solutions not requiring this assumption. The first one is the closest to SMC: we learn a Markov chain accurate for a given time-to-error property, and it does not require knowledge on the Markov chain. The second one is much more ambitious than SMC as it learns a Markov chains accurate for *all* its global behaviors, formalized as all properties of a temporal logic; it needs the assumption that the set of transitions is known, but not their probabilities nor a lower bound on them. This assumption may seem heavy, but it is reasonable for designers of systems, for which (a lower bound on) transition probabilities are not known (e.g. some error rate of components, etc).

For comparison with SMC, our final output is the (approximated) transition matrix of a DTMC rather than one (approximated) probability of a given property. This learned DTMC can be used for different purposes, e.g. as a component

[1] We recall the Okamoto-Chernoff bound in Appendix B (as well as the Massart bound), but we do not use it in this work.

in a bigger model or as a simulation tool. In terms of performances, we will show that we can learn a DTMC w.r.t. a given property with the same number of samples as we need to estimate this property using SMC (see Section 5). That is, there is no penalty to estimate a DTMC rather than estimate one probability, and we can scale as well as SMC. In terms of expressivity, we can handle unbounded properties (e.g. fairness properties). Even better, we can learn a DTMC accurate uniformly over a possibly infinite set of properties, e.g. all formulas of CTL. This is something SMC is not designed to achieve.

Other related work can be cited: In [13], the authors investigate several distances for the estimation of the difference between DTMCs. But they do not propose algorithms for learning. In [16], the authors propose to analyze the learned model a posteriori to test whether it has some good properties. If not, then they tweak the model in order to enforce these properties. Also, several PAC-learning algorithms have been proposed for the estimation of stochastic systems [5,10] but these works focus on local transitions instead of global properties.

4 Problem statement

In this work, we are interested to learn a DTMC model from a stochastic system $\mathcal{S}$ such that the behaviors of the system and the model are similar. We assume that the original system is a DTMC parameterized by a matrix A of transition probabilities. The transition probabilities are unknown, but the set of states of the DTMC is assumed to be known.

Our goal is to provide a learning algorithm $\mathcal{A}$ that guarantees an accurate estimation of $\mathcal{S}$ with respect to certain global properties. For that, a sampling process is defined as follows. A path (i.e., a sequence of states from s_0) of $\mathcal{S}$ is observed, and at steps specified by the sampling process, a reset action is performed, setting $\mathcal{S}$ back to its initial state s_0. Then another path is generated. This process generates a set W of paths, called traces, used to learn a matrix $\hat{A}_W$. Formally, we want to provide a learning algorithm that guarantees the following specification:

$$\mathbb{P}(\mathcal{D}(A, \hat{A}_W) > \varepsilon) \leq \delta \tag{2}$$

where $\varepsilon > 0$ and $\delta > 0$ are respectively *accuracy* and *confidence* parameters and $\mathcal{D}(A, \hat{A}_W)$ is a measure of the divergence between A and $\hat{A}_W$.

There exist several ways to specify the divergence between two transition matrices, e.g., the Kullback-Leibler divergence [19] or a distance based on a matrix norm. However, the existing notions remain heuristic because they are based on the difference between the individual probabilistic transitions of the matrix. We argue that what matters in practice is often to quantify the similarity between the global behaviors of the systems and the learned model.

In order to specify the behaviors of interest, we use a property φ or a set of properties Ψ on the set of states visited. We are interested in the difference between the probabilities of φ (i.e., the measure of the set of runs satisfying φ)

with respect to A and $\hat{A}_W$. We want to ensure that this difference is less than some predefined ε with (high) probability $1-\delta$. Hence, we define:

$$\mathcal{D}_\varphi(A, \hat{A}_W) = |\gamma(A, \varphi) - \gamma(\hat{A}_W, \varphi)| \tag{3}$$

$$\mathcal{D}_\Psi(A, \hat{A}_W) = \max_{\varphi \in \Psi}(\mathcal{D}_\varphi(A, \hat{A}_W)) \tag{4}$$

Our problem is to construct an algorithm which takes the following as inputs:

- confidence δ, $0 < \delta < 1$,
- absolute error $\varepsilon > 0$, and
- a property φ (or a set of properties Ψ),

and provides a learning procedure sampling a set W of paths, outputs $\hat{A}_W$, and terminates the sampling procedure while fulfilling Specification (2), with $\mathcal{D} = \mathcal{D}_\varphi$ $(= \mathcal{D}_\Psi)$.

In what follows, we assume that the confidence level δ and absolute error ε are fixed. We first start with a negative result: if Ψ is the set of LTL formulas [2], such a learning process is impossible.

Theorem 2. *Given $\varepsilon > 0$, $0 < \delta < 1$, and a finite set W of paths randomly drawn with respect to a DTMC A, there is no learning strategy such that, for every LTL formula φ,*

$$\mathbb{P}(|\gamma(A, \varphi) - \gamma(\hat{A}_W, \varphi)| > \varepsilon) \leq \delta \tag{5}$$

Note that contrary to Theorem 1, the deviation in Theorem 2 is a difference between two exact probabilities (of the original system and of a learned model). The theorem holds as long as $\hat{A}_W$ and A are not strictly equal, no matter how $\hat{A}_W$ is learned. To prove this theorem, we show that, for any number of observations, we can always define a sequence of LTL properties that violates the specification above. It only exploits a single deviation in one transition. The proof, inspired by a result from [13], is given in Appendices B and C.

Example 1. We show in this example that in general, one needs to have some knowledge on the system in order to perform PAC learning - either a positive lower bound $\ell > 0$ on the lowest probability transition, as in [14,1], or the support of transitions (but no knowledge on their probabilities), as we use in Section 6. Further, we show that the latter assumption does not imply the former, as even if no transitions are overlooked, the error in some reachability property can be arbitrarily close to 0.5 even with arbitrarily small error on the transition probabilities.

Let us consider DTMCs $A, \hat{A}, \hat{B}$ in Fig. 3, and formula $\mathbf{F}\, s_2$ stating that s_2 is eventually reached. The probabilities to satisfy this formula in $A, \hat{A}, \hat{B}$ are respectively $\mathbb{P}^A(\mathbf{F}\, s_2) = \frac{1}{2}$, $\mathbb{P}^{\hat{A}}(\mathbf{F}\, s_2) = \frac{2\tau-\eta}{4\tau} = \frac{1}{2} - \frac{\eta}{4\tau}$ and $\mathbb{P}^{\hat{B}}(\mathbf{F}\, s_2) = 0$.

Assume that A is the real system and that $\hat{A}$ and $\hat{B}$ are DTMCs we learned from A. Obviously, one wants to avoid learning $\hat{B}$ from A, as the probability of

$\mathbf{F}\, s_2$ is very different in $\hat{B}$ and in $\hat{A}$ (0 instead of 0.5). If one knows that $\tau > \ell$ for some lower bound $\ell > 0$, then one can generate enough samples from s_1 to evaluate τ with an arbitrarily small error $\frac{\eta}{2} << \ell$ on probability transitions with an arbitrarily high confidence, and in particular learn a DTMC similar to $\hat{A}$.

On the other hand, if one knows there are transitions from s_1 to s_2 and to s_3, then immediately, one does not learn DTMC $\hat{B}$, but a DTMC similar to DTMC $\hat{A}$ (using e.g. Laplace smoothing [12]). While this part is straightforward with this assumption, evaluating τ is much harder when one does not know a priori a lower bound $\ell > 0$ such that $\tau > \ell$. That is very important: while one can make sure that the error $\frac{\eta}{2}$ on probability transitions is arbitrarily small, if τ is unknown, then it could be the case that τ is as small as $\frac{\eta}{2(1-\varepsilon)} > \frac{\eta}{2}$, for a small $\varepsilon > 0$. This gives us $\mathbb{P}^{\hat{A}}(\mathbf{F}\, s_2) = \frac{1}{2} - \frac{1-\varepsilon}{2} = \frac{\varepsilon}{2}$, which is arbitrarily small, whereas $\mathbb{P}^{A}(\mathbf{F}\, s_2) = 0.5$, leading to a huge error in the probability to reach s_2. We work around that problem in Section 6 by defining and computing the *conditioning* of DTMC $\hat{A}$. In some particular cases, as the one discussed in the next section, one can avoid that altogether (actually, the conditioning in these cases is perfect (=1), and it needs not be computed explicitly).

5 Learning for a time-to-failure property

In this section, we focus on property φ of reaching a failure state s_F from an initial state s_0 without re-passing by the initial state, which is often used for assessing the failure rate of a system and the mean time between failures (see e.g., [24]). We assume that with probability 1, the runs eventually re-pass by s_0 or reach s_F. Also, without loss of generality, we assume that there is a unique failure state s_F in A. We denote $\gamma(A, \varphi)$ the probability, given DTMC A, of satisfying property φ, i.e., the probability of a failure between two visits of s_0.

Assume that the stochastic system $\mathcal{S}$ is observed from state s_0. Between two visits of s_0, property φ can be monitored. If s_F is observed between two instances of s_0, we say that the path $\omega = s_0 \cdot \rho \cdot s_F$ satisfies φ, with $s_0, s_F \notin \rho$. Otherwise, if s_0 is visited again from s_0, then we say that the path $\omega = s_0 \cdot \rho \cdot s_0$ violates φ, with $s_0, s_F \notin \rho$. We call *traces* paths of the form $\omega = s_0 \cdot \rho \cdot (s_0 \vee s_F)$ with $s_0, s_F \notin \rho$. In the following, we show that it is sufficient to use a *frequency estimator* to learn a DTMC which provides a good approximate for such a property.

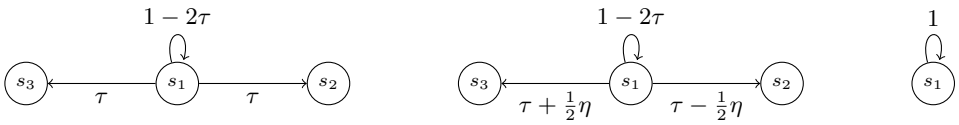

Figure 3: Three DTMCs $A, \hat{A}, \hat{B}$ (from left to right), with $0 < \eta < 2\tau < 1$

5.1 Frequency estimation of a DTMC

Given a set W of n traces, we denote n_{ij}^W the number of times transition from state i to state j occurred and n_i^W the number of times a transition has been taken from state i.

The *frequency estimator* of A is the DTMC $\hat{A}_W = (\hat{a}_{ij})_{1 \leq i,j \leq m}$ given by $\hat{a}_{ij} = \frac{n_{ij}^W}{n_i^W}$ for all i, j, with $\sum_{i=1}^m n_i^W = \sum_{i=1}^m \sum_{j=1}^m n_{ij}^W = |W|$. In other words, to learn $\hat{A}_W$, it suffices to count the number of times a transition from i to j occurred, and divide by the number of times state i has been observed. The matrix $\hat{A}_W$ is trivially a DTMC, except for states i which have not been visited. In this case, one can set $\hat{a}_{ij} = \frac{1}{m}$ for all state j and obtain a DTMC. This has no impact on the behavior of $\hat{A}_W$ as i is not reachable from s_0 in $\hat{A}_W$.

Let $\hat{A}_W$ be the matrix learned using the frequency estimator from the set W of traces, and let A be the real probabilistic matrix of the original system $\mathcal{S}$. We show that, in the case of time-to-failure properties, $\gamma(\hat{A}_W, \varphi)$ is equal to the crude Monte Carlo estimator $\hat{\gamma}_W(A, \varphi)$ induced by W.

5.2 PAC bounds for a time-to-failure property

We start by stating the main result of this section, bounding the error between $\gamma(A, \varphi)$ and $\gamma(\hat{A}_W, \varphi)$:

Theorem 3. *Given a set W of n traces such that $n = \lceil H(n) \rceil$, we have:*

$$\mathbb{P}\left(|\gamma(A, \varphi) - \gamma(\hat{A}_W, \varphi)| > \varepsilon\right) \leq \delta \tag{6}$$

where $\hat{A}_W$ is the frequency estimator of A.

To prove Theorem (3), we first invoke Theorem 1 to establish:

$$\mathbb{P}\left(|\gamma(A, \varphi) - \hat{\gamma}_W(A, \varphi)| > \varepsilon\right) \leq \delta \tag{7}$$

It remains to show that $\hat{\gamma}_W(A, \varphi) = \gamma(\hat{A}_W, \varphi)$:

Proposition 1. *Given a set W of traces, $\gamma(\hat{A}_W, \varphi) = \hat{\gamma}_W(A, \varphi)$.*

It might be appealing to think that this result can be proved by induction on the size of the traces, mimicking the proof of computation of reachability probabilities by linear programming [2]. This is actually not the case. The remaining of this section is devoted to proving Proposition (1).

We first define $q_W(u)$ the number of occurrences of sequence u in the traces of W. Note that u can be a state, an individual transition or even a path. We also use the following definitions in the proof.

Definition 2 (Equivalence). *Two sets of traces W and W' are equivalent if for all $s, t \in S$, $\frac{q_W(s \cdot t)}{q_W(s)} = \frac{q_{W'}(s \cdot t)}{q_{W'}(s)}$.*

We define a set of traces W' equivalent with W, implying that $\hat{A}_W = \hat{A}_{W'}$. This set W' of traces satisfies the following:

Lemma 1. *For any set of traces W, there exists a set of traces W' such that:*
(i) W and W' are equivalent,
(ii) for all $r, s, t \in S$, $q_{W'}(r \cdot s \cdot t) = \dfrac{q_{W'}(r \cdot s) \times q_{W'}(s \cdot t)}{q_{W'}(s)}$.

The proof of Lemma 1 is provided in Appendix D. In Lemma 1, (i) ensures that $\hat{A}_{W'} = \hat{A}_W$ and (ii) ensures the equality between the proportion of runs of W' passing by s and satisfying γ, denoted $\hat{\gamma}^s_{W'}$, and the probability of reaching s_F before s_0 starting from s with respect to $\hat{A}_{W'}$. Formally,

Lemma 2. *For all $s \in S$, $\mathbb{P}_s^{\hat{A}_{W'}}(\text{reach } s_f \text{ before } s_0) = \hat{\gamma}^s_{W'}$.*

Proof. Let S_0 be the set of states s with no path in $\hat{A}_{W'}$ from s to s_f without passing through s_0. For all $s \in S_0$, let $p_s = 0$. Also, let $p_{s_f} = 1$. Let $S_1 = S \setminus (S_0 \cup \{s_f\})$. Consider the system of equations (8) with variables $(p_s)_{s \in S_1} \in [0,1]^{|S_1|}$:

$$\forall\, s \in S_1, \quad p_s = \sum_{t=1}^{m} \hat{A}_{W'}(s,t) p_t \tag{8}$$

The system of equations (8) admits a unique solution according to [2] (Theorem 10.19. page 766). Then, $(\mathbb{P}_s^{\hat{A}_{W'}}(\text{reach } s_f \text{ before } s_0))_{s \in S_1}$ is trivially a solution of (8). But, since W' satisfies the conditions of Lemma 1, we also have that $(\hat{\gamma}^s_{W'})_{s \in S_1}$ is a solution of (8), and thus we have the desired equality. □

Notice that Lemma 2 does not hold in general with the set W. We have:

$$\begin{aligned}
\hat{\gamma}_W(A, \varphi) &= \hat{\gamma}^{s_0}_W \quad \text{(by definition)} \\
&= \hat{\gamma}^{s_0}_{W'} \quad \text{(by Lemma 1)} \\
&= \mathbb{P}_{s_0}^{\hat{A}_{W'}}(\text{reach } s_f \text{ before } s_0) \quad \text{(by Lemma 2)} \\
&= \mathbb{P}_{s_0}^{\hat{A}_W}(\text{reach } s_f \text{ before } s_0) \quad \text{(by Lemma 1)} \\
&= \gamma(\hat{A}_W, \varphi) \quad \text{(by definition).}
\end{aligned}$$

That concludes the proof of Proposition 1. It shows that learning can be as efficient as statistical model-checking on comparable properties.

6 Learning for the full CTL logic

In this section, we learn a DTMC $\hat{A}_W$ such that $\hat{A}_W$ and A have similar behaviors over all CTL formulas. This provides a much stronger result than on time-to-failure property, e.g., properties can involve liveness and fairness, and more importantly they are not known before the learning. Notice that PCTL [2] cannot be used, since an infinitesimal error on one > 0 probability can change the probability of a PCTL formula from 0 to 1. (State)-CTL is defined as follows:

Definition 3. *Let Prop be the set of state names. (State)-CTL is defined by the following grammar* $\varphi ::= \bot \mid \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{AX}\varphi \mid \mathbf{EX}\varphi \mid \mathbf{AF}\varphi \mid \mathbf{EF}\varphi \mid \mathbf{AF}\varphi \mid \mathbf{EG}\varphi \mid \mathbf{AG}\varphi \mid \mathbf{E}(\varphi\mathbf{U}\varphi) \mid \mathbf{A}(\varphi\mathbf{U}\varphi)$*, with* $p \in Prop$*.* **E***(xists) and* **A***(ll) are quantifiers on paths, ne***X***t,* **G***lobally,* **F***inally and* **U***ntil are path-specific quantifiers. Notice that some operators are redundant. A minimal set of operators is* $\{\top, \vee, \neg, \mathbf{EG}, \mathbf{EU}, \mathbf{EX}\}$*.*

As we want to compute the probability of *paths* satisfying a CTL formula, we consider the set Ψ of *path-CTL* properties, that is formulas φ of the form $\varphi = \mathbf{X}\varphi_1$, $\varphi = \varphi_1\mathbf{U}\varphi_2$, $\varphi = \mathbf{F}\varphi_1$ or $\varphi = \mathbf{G}\varphi_1$, with φ_1, φ_2 (state)-CTL formulas. For instance, the property considered in the previous section is $(\neg s_0)\mathbf{U}s_F$.

In this section, for the sake of simplicity, the finite set W of traces is obtained by observing paths till a state is seen twice on the path. Then, the reset action is used and another trace is obtained from another path. That is, a trace ω from W is of the form $\omega = \rho \cdot s \cdot \rho' \cdot s$, with $\rho \cdot s \cdot \rho'$ a loop-free path.

As explained in example 1, some additional knowledge on the system is necessary. In this section, we assume that the support of transition probabilities is known, i.e., for any state i, we know the set of states j such that $a_{ij} \neq 0$. This assumption is needed both for Theorem 5 and to apply Laplace smoothing.

6.1 Learning DTMCs with Laplace smoothing

Let $\alpha > 0$. For any state s, let k_s be the number of successors of s, that we know by hypothesis, and $T = \sum_{s \in S} k_s$ be the number of non-zero transitions. Let W be a set of traces, n_{ij}^W the number of transitions from state i to state j, and $n_i^W = \sum_j n_{ij}^W$. The *estimator for W with Laplace smoothing α* is the DTMC $\hat{A}_W^\alpha = (\hat{a}_{ij})_{1 \leq i,j \leq m}$ given for all i, j by:

$$\hat{a}_{ij} = \frac{n_{ij}^W + \alpha}{n_i^W + k_i\alpha} \text{ if } a_{ij} \neq 0 \quad \text{and} \quad \hat{a}_{ij} = 0 \text{ otherwise}$$

In comparison with the frequency estimator, the Laplace smoothing adds for each state s a term α to the numerator and k_s times α to the denominator. This preserves the fact that $\hat{A}_W^\alpha$ is a Markov chain, and it ensures that $\hat{a}_{ij} \neq 0$ iff $a_{ij} \neq 0$. In particular, compared with the frequency estimator, it avoids creating zeros in the probability tables.

6.2 Conditioning and Probability Bounds

Using Laplace smoothing slightly changes the probability of each transition by an additive offset η. We now explain how this small error η impacts the error on the probability of a CTL property.

Let A be a DTMC, and A_η be a DTMC such that $A_\eta(i,j) \neq 0$ iff $A(i,j) \neq 0$ for all states i, j, and such that $\sum_j |A_\eta(i,j) - A(i,j)| \leq \eta$ for all state i. For all state $s \in S$, let $R(s)$ be the set of states i such that there exists a path from i to s. Let $R_*(s) = R(s) \setminus \{s\}$. Since both DTMCs have the same support, R (and also R_*) is equal for A and A_η. Given m the number of states, the conditioning of A for $s \in S$ and $\ell \leq m$ is:

$$\mathrm{Cond}_s^\ell(A) = \min_{i \in R_*(s)} \mathbb{P}_i^A(\mathbf{F}_{\leq \ell} \neg R_*(s))$$

i.e., the minimal probability from state $i \in R_*(s)$ to move away from $R_*(s)$ in at most ℓ steps. Let ℓ_s minimal such that $\mathrm{Cond}_s^{\ell_s}(A) > 0$. This minimal ℓ_s exists as $\mathrm{Cond}_s^m(A) > 0$ since, for all $s \in S$ and $i \in R_*(s)$, there is at least one path reaching s from i (this path leaves $R_*(s)$), and taking a cycle-free path, we obtain a path of length at most m. Thus, the probability $\mathbb{P}_i^A(\mathbf{F}_{\leq m} \neg R_*(s))$ is at least the positive probability of the cylinder defined by this finite path. Formally,

Theorem 4. *Denoting φ the property of reaching state s in DTMC A, we have:*

$$|\gamma(A,\varphi) - \gamma(A_\eta,\varphi)| < \frac{\ell_s \cdot \eta}{Cond_s^{\ell_s}(A)}$$

Proof. Let v_s be the stochastic vector with $v_s(s) = 1$. We denote $v_0 = v_{s_0}$. Let $s \in S$. We assume that $s_0 \in R_*(s)$ (else $\gamma(A,\varphi) = \gamma(A_\eta,\varphi)$ and the result is trivial). Without loss of generality, we can also assume that $A(s,s) = A_\eta(s,s) = 1$ (as we are interested in reaching s at any step). With this assumption:

$$|\gamma(A,\varphi) - \gamma(A_\eta,\varphi)| = \lim_{t\to\infty} v_0 \cdot (A^t - A_\eta^t) \cdot v_s$$

We bound this error, through bounding by induction on t:

$$E(t) = \max_{i \in R_*(s)} v_i \cdot (A^t - A_\eta^t) \cdot v_s$$

We then have trivially:

$$|\gamma(A,\varphi) - \gamma(A_\eta,\varphi)| \leq \lim_{t\to\infty} E(t)$$

Note that for $i = s$, $\lim_{t\to\infty} v_i \cdot (A^t) \cdot v_s = 1 = \lim_{t\to\infty} v_i \cdot A_\eta^t \cdot v_s$, and thus their difference is null.

Let $t \in \mathbb{N}$. We let $j \in R_*(s)$ such that $E(t) = v_j \cdot (A^t - A_\eta^t) \cdot v_s$.

By the triangular inequality, introducing the term $v_j \cdot A^{\ell_s} A_\eta^{t-k} \cdot v_s - v_j \cdot A^{\ell_s} A_\eta^{t-k} \cdot v_s = 0$, we have:

$$E(t) \leq |v_j \cdot (A_\eta^t - A^{\ell_s} A_\eta^{t-\ell_s}) \cdot v_s| + |(v_j \cdot A^{\ell_s}) \cdot (A_\eta^{t-\ell_s} - A^{t-\ell_s}) \cdot v_s|$$

We separate vector $(v_j \cdot A^{\ell_s}) = w_1 + w_2 + w_3$ in three sub-stochastic vectors w_1, w_2, w_3: vector w_1 is over $\{s\}$, and thus we have $w_1 \cdot A_\eta^{t-\ell_s} = w_1 = w_1 \cdot A^{t-\ell_s}$, and the term cancels out. Vector w_2 is over states of $R_*(s)$, with $\sum_{i \in R_*} w_2[i] \leq (1 - \mathrm{Cond}_s^{\ell_s}(A))$, and we obtain an inductive term $\leq (1 - \mathrm{Cond}_s^{\ell_s}(A))E(t - \ell_s)$. Last, vector w_3 is over states not in $R(s)$, and we have $w_3 \cdot A_\eta^{t-\ell_s} \cdot v_s = 0 = w_3 \cdot A^{t-\ell_s} \cdot v_s$, and the term cancels out.

We also obtain that $|v_j \cdot (A_\eta^t - A^{\ell_s} A_\eta^{t-\ell_s}) \cdot v_s| \leq \ell_s \cdot \eta$. Thus, we have the inductive formula $E(t) \leq (1 - \mathrm{Cond}_s^{\ell_s}(A))E(t - \ell_s) + \ell_s \cdot \eta$. It yields for all $t \in \mathbb{N}$:

$$E(t) \leq (\ell_s \cdot \eta) \sum_{i=1}^{\infty} (1 - \mathrm{Cond}_s^{\ell_s}(A))^i$$

$$E(t) \leq \frac{\ell_s \cdot \eta}{Cond_s^{\ell_s}(A)}$$

□

We can extend this result from reachability to formulas of the form $S_0 \mathbf{U} S_F$, where S_0, S_F are subsets of states. This formula means that we reach the set of states S_F through only states in S_0 on the way.

We define $R(S_0, S_F)$ to be the set of states which can reach S_F using only states of S_0, and $R_*(S_0, S_F) = R(S_0, S_F) \setminus S_F$. For $\ell \in \mathbb{N}$, we let:

$$\mathrm{Cond}_{S_0,S_F}^{\ell}(A) = \min_{i \in R_*(S_0,S_F)} \mathbb{P}_i^A(\mathbf{F}_{\leq \ell} \neg R_*(S_0, S_F) \vee \neg S_0).$$

Now, one can remark that $\mathrm{Cond}_{S_0,S_F}(A) \geq \mathrm{Cond}_{S,S_F}(A) > 0$. Let $\mathrm{Cond}_{S_F}^{\ell}(A) = \mathrm{Cond}_{S,S_F}^{\ell}(A)$. We have $\mathrm{Cond}_{S_0,S_F}^{\ell}(A) \geq \mathrm{Cond}_{S_F}^{\ell}(A)$. As before, we let $\ell_{S_F} \leq m$ be the minimal ℓ such that $\mathrm{Cond}_{S_F}^{\ell}(A) > 0$, and obtain:

Theorem 5. *Denoting φ the property $S_0 \mathbf{U} S_F$, we have, given DTMC A:*

$$|\gamma(A, \varphi) - \gamma(A_\eta, \varphi)| < \frac{\ell_{S_F} \cdot \eta}{Cond_{S_F}^{\ell_{S_F}}(A)}$$

We can actually improve this conditioning: we defined it as the probability to reach S_F or $S \setminus R(S, S_F)$. At the price of a more technical proof, we can obtain a better bound by replacing S_F by the set of states $R_1(S_F)$ that have probability 1 to reach S_F. We let $\overline{R_*}(S_F) = R(S, S_F) \setminus R_1(S_F)$ the set of states that can reach S_F with < 1 probability, and define the *refined conditioning* as follows:

$$\overline{\mathrm{Cond}}_{S_F}^{\ell}(A) = \min_{i \in \overline{R_*}(S_F)} \mathbb{P}_i^A(\mathbf{F}_{\leq \ell} \neg \overline{R_*}(S_F))$$

6.3 Optimality of the conditioning

We show now that the bound we provide in Theorem 4 is close to optimal.

Consider again DTMCs $A, \hat{A}$ in Fig. 3 from example 1, and formula $\mathbf{F}\, s_2$ stating that s_2 is eventually reached. The probabilities to satisfy this formula in $A, \hat{A}$ are respectively $\mathbb{P}^A(\mathbf{F}\, s_2) = \frac{1}{2}$ and $\mathbb{P}^{\hat{A}}(\mathbf{F}\, s_2) = \frac{1}{2} - \frac{\eta}{4\tau}$. Assume that A is the real system and that $\hat{A}$ is the DTMC we learned from A.

As we do not know precisely the transition probabilities in A, we can only compute the conditioning on $\hat{A}$ and not on A (it suffices to swap A and A_η in Theorem 4 and 5 to have the same formula using $\mathrm{Cond}(A_\eta) = \mathrm{Cond}(\hat{A})$). We have $R(s_2) = \{s_1, s_2\}$ and $R_*(s_2) = \overline{R_*}(s_2) = \{s_1\}$. The probability to stay in $R_*(s_2)$ after $\ell_{s_2} = 1$ step is $(1 - 2\tau)$, and thus $\mathrm{Cond}^1_{\{s_2\}}(\hat{A}) = \overline{\mathrm{Cond}}^1_{\{s_2\}}(\hat{A}) = 1-(1-2\tau) = 2\tau$. Taking $A_\eta = \hat{A}$, Theorem 5 tells us that $|\,\mathbb{P}^A(\mathbf{F}\, s_2) - \mathbb{P}^{\hat{A}}(\mathbf{F}\, s_2)| \leq \frac{\eta}{2\tau}$. Notice that on that example, using $\ell_{s_2} = m = 3$, we obtain $\mathrm{Cond}^3_{\{s_2\}}(\hat{A}) = 1 - (1 - 2\tau)^3 \approx 6\tau$, and we find a similar bound $\approx \frac{3\eta}{6\tau} = \frac{\eta}{2\tau}$.

Compare our bound with the exact difference $|\,\mathbb{P}^A(\mathbf{F}\, s_2) - \mathbb{P}^{\hat{A}}(\mathbf{F}\, s_2)| = \frac{1}{2} - (\frac{1}{2} - \frac{\eta}{4\tau}) = \frac{\eta}{4\tau}$. Our upper bound only has an overhead factor of 2, even while the conditioning is particularly bad (small) in this example.

6.4 PAC bounds for $\sum_j |\hat{\boldsymbol{A}}_{\boldsymbol{W}}(\boldsymbol{i},\boldsymbol{j}) - \boldsymbol{A}(\boldsymbol{i},\boldsymbol{j})| \leq \boldsymbol{\eta}$

We use Theorem 1 in order to obtain PAC bounds. We use it to estimate individual transition probabilities, rather than the probability of a property.

Let W be a set of traces drawn with respect to A such that every $\omega \in W$ is of the form $\omega = \rho \cdot s \cdot \rho' \cdot s$. Recall for each state i, j of S, n_i^W is the number of transitions originating from i in W and n_{ij}^W is the number of transition ss' in W. Let $\delta' = \frac{\delta}{m_{\text{stoch}}}$, where m_{stoch} is the number of *stochastic* states, i.e., with at least two outgoing transitions.

We want to sample traces until the empirical transition probabilities $\frac{n_{ij}^W}{n_i^W}$ are relatively close to the exact transition probabilities a_{ij}, for all $i, j \in S$. For that, we need to determine a stopping criteria over the number of state occurrences $(n_i)_{1 \leq i \leq m}$ such that:

$$\mathbb{P}\left(\exists i \in S, \sum_j \left| a_{ij} - \frac{n_{ij}^W}{n_i^W} \right| > \varepsilon \right) \leq \delta$$

First, note that for any observed state $i \in S$, if $a_{ij} = 0$ (or $a_{ij} = 1$), then with probability 1, $\frac{n_{ij}^W}{n_i^W} = 0$ (respectively $\frac{n_{ij}^W}{n_i^W} = 1$). Thus, for all $\varepsilon > 0$, $|a_{ij} - \frac{n_{ij}^W}{n_i^W}| < \varepsilon$ with probability 1. Second, for two distinct states i and i', the transition probabilities $\frac{n_{ij}^W}{n_i^W}$ and $\frac{n_{i'j'}^W}{n_{i'}^W}$ are independent for all j, j'.

Let $i \in S$ be a stochastic state. If we observe n_i^W transitions from i such that $n_i^W \geq \frac{2}{\varepsilon^2} \log\left(\frac{2}{\delta'}\right) \left[\frac{1}{4} - \left(\max_j |\frac{1}{2} - \frac{n_{ij}^W}{n_i^W}| - \frac{2}{3}\varepsilon\right)^2\right]$, then, according to Theorem 1,

$\mathbb{P}\left(\bigvee_{j=1}^{m} |a_{ij} - \frac{n_{ij}^W}{n_i^W}| > \varepsilon\right) \leq \delta'$. In particular, $\mathbb{P}\left(\max_{j\in S} |a_{ij} - \frac{n_{ij}^W}{n_i^W}| > \varepsilon\right) \leq \delta'$. Moreover, we have:

$$\begin{aligned}\mathbb{P}\left(\bigvee_{j=1}^{m} \max_{j\in S} |a_{ij} - \frac{n_{ij}^W}{n_i^W}| > \varepsilon\right) &\leq \sum_{j=1}^{m} \mathbb{P}\left(\max_{j\in S} |a_{ij} - \frac{n_{ij}^W}{n_i^W}| > \varepsilon\right) \\ &\leq m_{\text{stoch}}\delta' \\ &\leq \delta\end{aligned}$$

In other words, the probability that "there exists a state $i \in S$ such that the deviation between the exact and empirical outgoing transitions from i exceeds ε" is bounded by δ as soon as for each state $i \in S$, n_i^W satisfies the stopping rule of the algorithm of Chen using ε and the corresponding δ'. This gives the hypothesis $\sum_j |A_\eta(i,j) - A(i,j)| \leq \epsilon$ for all state i of Section 6.2.

6.5 A Matrix $\hat{A}_W$ accurate for all CTL properties

We now use Laplace smoothing in order to ensure the other hypothesis $A_\eta(i,j) \neq 0$ iff $A(i,j) \neq 0$ for all states i, j. For all $i \in S$, we define the Laplace offset depending on the state i as $\alpha_i = \frac{(n_i^W)^2\varepsilon}{10\cdot k_i^2 \max_j n_{ij}^W}$, where k_i is the number of transitions from state i. This ensures that the error from Laplace smoothing is at most one tenth of the statistical error. Let $\alpha = (\alpha_i)_{1\leq i\leq m}$. From the sample set W, we output the matrix $\hat{A}_W^\alpha = (\hat{a}_{ij})_{1\leq i,j\leq m}$ with Laplace smoothing α_i for state i, i.e.:

$$\hat{a}_{ij} = \frac{n_{ij}^W + \alpha_i}{n_i^W + k_i\alpha_i} \text{ if } a_{ij} \neq 0 \quad \text{and} \quad \hat{a}_{ij} = 0 \text{ otherwise}$$

It is easy to check that we have for all $i, j \in S$: $\left|\hat{a}_{ij} - \frac{n_{ij}^W}{n_i^W}\right| \leq \frac{\varepsilon}{10\cdot k_i}$

That is, for all state i, $\sum_j \left|\hat{a}_{ij} - \frac{n_{ij}^W}{n_i^W}\right| \leq \frac{\varepsilon}{10}$. Using the triangular inequality:

$$\mathbb{P}\left(\exists i \in S, \sum_j |a_{ij} - \hat{a}_{ij}| > \frac{11}{10}\varepsilon\right) \leq \delta$$

For all $i \in S$, let $H^*(n_i^W, \epsilon, \delta') = \max_{j\in S} H(n_i^W, n_{ij}^W, \epsilon, \delta')$ be the maximal Chen bound over all the transitions from state i. Let $B(\hat{A}_W^\alpha) = \max_{S_F} \frac{\ell_{S_F}}{\text{Cond}_{S_F}^{\ell_{S_F}}(\hat{A}_W^\alpha)}$. Since in Theorem 5, the original model and the learned one have symmetric roles, by applying this theorem on $\hat{A}_W^\alpha$, we obtain that:

Theorem 6. *Given a set W of traces, for $0 < \epsilon < 1$ and $0 < \delta < 1$, if for all $i \in S$, $n_i^W \geq \left(\frac{11}{10}B(\hat{A}_W^\alpha)\right)^2 H^*(n_i^W, \epsilon, \delta')$, we have for any CTL property φ:*

$$\mathbb{P}(|\gamma(A,\varphi) - \gamma(\hat{A}_W^\alpha, \varphi)|) > \varepsilon) \leq \delta \tag{9}$$

Proof. First, $\hat{a}_{ij} \neq 0$ iff $a_{ij} \neq 0$, by definition of $\hat{A}^{\alpha}_{W}$. Second, $\mathbb{P}(\exists i, \sum_j |a_{ij} - \hat{a}_{ij}| > \frac{11}{10}\varepsilon) \leq \delta$. We can thus apply Theorem 5 on $\hat{A}^{\alpha}_{W}, A$ and obtain (9) for φ any formula of the form $S_1 \mathbf{U} S_2$. It remains to show that for any formula $\varphi \in \Psi$, we can define $S_1, S_2 \subseteq S$ such that φ can be expressed as $S_1 \mathbf{U} S_2$.

Consider the different cases: If φ is of the form $\varphi = \varphi_1 \mathbf{U} \varphi_2$ (it subsumes the case $\varphi = \mathbf{F}\varphi_1 = \top \mathbf{U} \varphi_1$) with φ_1, φ_2 CTL formulas, we define S_1, S_2 as the sets of states satisfying φ_1 and φ_2, and we have the equivalence (see [2] for more details). If $\varphi = X\varphi_2$, define $S_1 = \emptyset$ and S_2 as the set of states satisfying φ_2.

The last case is $\varphi = \mathbf{G}\varphi_1$, with φ_1 a CTL formula. Again, we define S_1 the set of states satisfying φ_1, and S_2 the set of states satisfying the CTL formula $\mathbf{AG}\varphi_1$. The probability of the set of paths satisfying $\varphi = \mathbf{G}\varphi_1$ is exactly the same as the probability of the set of paths satisfying $S_1 \mathbf{U} S_2$. □

6.6 Algorithm

We give more details about the learning process of a Markov Chain, accurate for every CTL formula. For completeness, we also provide in Appendix F a similar algorithm for a time-to-failure property.

A path ω is observed from s_0 till a state is observed twice. Then ω is added to W and the reset operation is performed. We use Laplace smoothing to compute the corresponding matrix $\hat{A}^{\alpha}_{W}$. The error bound is computed on W, and a new path ω' is then being generated if the error bound is not as small as desired.

This algorithm is guaranteed to terminate since, as traces are generated, with probability 1, n^W_s tends towards ∞, $\hat{A}^{\alpha}_{W}$ tends towards A, and $B(\hat{A}^{\alpha}_{W})$ tends towards $B(A)$.

Algorithm 1: Learning a matrix accurate for CTL

Data:
$\mathcal{S}, s_0, \delta, \varepsilon$

$W := \emptyset$
$m = |S|$
for all $s \in S$, $n^W_s := 0$
Compute $\hat{A} := \hat{A}^{\alpha}_{W}$
Compute $B := B(\hat{A})$
while $\exists s \in S, n^W_s < \left(\frac{11}{10} B(\hat{A})\right)^2 H^*(n^W_s, \epsilon, \frac{\delta}{m})$ **do**
  Generate a new trace $\omega := s_0\, \rho\, s_1\, \rho'\, s_1$, and reset $\mathcal{S}$
  for all $s \in S$, $n^W_s := n^W_s + n^{\{\omega\}}_s$
  add ω to W
  Compute $\hat{A} := \hat{A}^{\alpha}_{W}$
  Compute $B := B(\hat{A})$

Output: $\hat{A}^{\alpha}_{W}$

7 Evaluation and Discussion

In this section, we first evaluate Algorithm 1 on 5 systems which are crafted to evaluate the algorithm under different conditions (e.g., rare states). The objective of the evaluation is to provide some idea on how many samples would be sufficient for learning accurate DTMC estimations, and compare learning for all properties of CTL and learning for one time-to-failure property.

Then, we evaluate our algorithm on very large PRISM systems (millions or billions of states). Because of the number of states, we cannot learn a DTMC accurate for all properties of CTL there: it would ask to visit every single state a number of times. However, we can learn a DTMC for one specific (unbounded) property. We compare with an hypothesis testing algorithm from [30] which can handle the same unbounded property through a reachability analysis using the topology of the system.

7.1 Evaluation on crafted models

We first describe the 5 systems: Systems 1 and 2 are three-state models described in Fig. 1 and Fig. 2. Systems 3 (resp. 5) is a 30-state (resp. 200-states) clique in which every individual transition probability is 1/30 (resp. 1/200). System 4 is a 64-state system modeling failure and repair of 3 types of components (3 components each, 9 components in total), see Appendix G for a full description of the system, including a PRISM [20] model for the readers interested to investigate this system in details.

We tested time-to-failure properties by choosing as failure states s_3 for Systems 1, 2, 3, 5, and the state where all 9 components fail for System 4. We also tested Algorithm 1 (for full CTL logic) using the refined conditioning $\overline{\text{Cond}}$. We performed our algorithms 100 times for each model, except for full CTL on System 4, for which we only tested once since it is very time-consuming. We report our results in Table 1 for $\varepsilon = 0.1$ and $\delta = 0.05$. In particular, we output for

	System 1	System 2	System 3	System 4	System 5
# states	3	3	30	64	200
# transitions	4	7	900	204	40,000
# events for time-to-failure	191 (16%)	991 (10%)	2,753 (7.4%)	1,386 (17.9%)	18,335 (7.2%)
# events for full CTL	1,463 (12.9%)	4,159 (11.7%)	8,404 (3.8%)	1,872,863	79,823 (1.7%)

Table 1: Average number of observed events N (and relative standard deviation in parenthesis) given $\varepsilon = 0.1$ and $\delta = 0.05$ for a time-to-failure property and for the full CTL logic using the refined conditioning $\overline{\text{Cond}}$.

each model its number of states and transitions. For each (set of) property, we provide the average number of observations (i.e. the number of samples times their average length) and the relative standard deviation (in parenthesis, that is the standard deviation divided by the average number of observed events).

The results show that we can learn a DTMC with more than 40000 stochastic transitions, such that the DTMC is accurate for all CTL formulas. Notice that for some particular systems such as System 4, it can take a lot of events to be observed before Algorithm 1 terminates. The reason is the presence of rare states, such as the state where all 9 components fail, which are observed with an extremely small probability. In order to evaluate the probabilities of CTL properties of the form: "if all 9 components fail, then CTL property φ is satisfied", this state needs to be explored many times, explaining the high number of events observed before the algorithm terminates. On the other hand, for properties that do not involve the 9 components failing as prior, such as time-to-failure, one does not need to observe this state even once to conclude that it has an extremely small probability to happen. This suggests that efficient algorithms could be developed for subsets of CTL formulas, e.g., in defining a subset of important events to consider. We believe that Theorem 4 and 5 could be extended to handle such cases. Over different runs, the results stay similar (notice the rather small relative standard deviation).

Comparing results for time-to-failure (or equivalently SMC) and for the full CTL logic is interesting. Excluding System 4 which involves rare states, the number of events that needs to be observed for the full CTL logic is 4.3 to 7 times more. Surprisingly, the highest difference is obtained on the smallest System 1. It is because every run of System 1 generated for time-to-failure is short ($s_1 s_2 s_1$ and $s_1 s_2 s_3$). However, in Systems 2,3 and 5, samples for time-to-failure can be much longer, and the performances for time-to-failure (or equivalently SMC) is not so much better than for learning a DTMC accurate for all CTL properties.

For the systems we tested, the unoptimized Cond was particularly large (more than 20) because for many states s, there was probability 0 to leave $R(s)$, and hence $\ell(s)$ was quite large. These are the cases where $\overline{\text{Cond}}$ is much more efficient, as then we can choose $\ell_s = 1$ as the probability to reach s from states in $R(s)$ is 1 ($R_1(s) = R(s)$ and $\overline{R_*(s)} = \emptyset$). We used $\overline{\text{Cond}}$ in our algorithm.

Finally, we evaluate experimental confidence by comparing the time-to-failure probabilities in the learned DTMC and the original system. We repeat our algorithms 1000 times on System 1 and 2 (with $\varepsilon = 0.1$ and $\delta = 0.05$). These probabilities differ by less than ε, respectively 999 and 995 times out of 1000. Specification (2) is thus largely fulfilled (the specification should be ensured 950 out of 1000 times), that empirically endorses our approach. Hence, while our PAC bound over-approximates the confidence in the learned system (which is unavoidable), it is not that far from experimental values.

7.2 Evaluation on large models

We also evaluated our algorithm on large PRISM models, ranging from hundreds of thousands to billions of states. With these numbers of states, we cannot use

the more ambitious learning over all the properties of CTL, which would need to visit every states a number of times. However, we can use our algorithm for learning a DTMC which is accurate given a particular (unbounded) property: it will visit only a fraction of the states, which is enough to give a model accurate for that property, with a well-learned kernel of states and some other states representatives for the remaining of the runs. We consider three test-cases from PRISM, satisfying the property that the sample stops with a conclusion (yes or no) with probability 1. Namely, *herman, leader* and *egl*.

Our prototype tool used in the previous subsection is implemented in Scilab: it cannot simulate very large systems of PRISM. Instead, we use PRISM to generate the samples needed for the learning. Hence, we report the usual Okamoto-Chernoff bound on the number of samples, which is what is implemented in PRISM. We also compare with the Massart bound used by the Chen algorithm (see Section 2.2), which is implemented in our tool and is more efficient as it takes into account the probability of the property.

For each model, we report its parameters, its *size*, i.e. its number of states, the number of *samples* needed using the Massart bound (the conservative Okamoto-Chernoff bound is in parenthesis), and the average *path length*. For comparison, we consider an hypothesis testing algorithm from [30] which can also handle unbounded properties. It uses the knowledge of the topology to do reachabil-

		our learning method		sampling with reachability analysis [30]	
Model name	size	samples	path length	samples	path length
herman(17)	129M	506 (38K)	27	219	30
herman(19)	1162M	506 (38K)	40	219	38
herman(21)	10G	506 (38K)	43	219	48
leader(6,6)	280K	506 (38K)	7.4	219	7
leader(6,8)	$>$ 280K	506 (38K)	7.4	(MO)	(MO)
leader(6,11)	$>$ 280K	506 (38K)	7.3	(MO)	(MO)
egl(15,10)	616G	38K (38K)	470	1100	201
egl(20,15)	1279T	38K (38K)	930	999	347
egl(20,20)	1719T	38K (38K)	1200	(TO)	(TO)

Table 2: Results for $\varepsilon = 0.01$ and $\delta = 0.001$ of our algorithm compared with sampling with reachability analysis [30], as reported in [14], page 20. Numbers of samples needed by our method are given by the Massart bound (resp. by the Okamoto-Chernoff bound in parenthesis). TO and MO means time out ($>$ 15 minutes on an Opteron 6134) and memory out ($>$ 5GB) respectively.

ity analysis to stop the sampling if the property cannot be reached anymore. Hypothesis testing is used to decide with high confidence whether a probability exceeds a threshold or not. This requires less samples than SMC algorithms which estimate probabilities, but it is also less precise. We chose to compare with this algorithm because as in our work, it does not require knowledge on the probabilities, such as a lower bound on the transition probabilities needed by e.g. [14]. We do not report runtime as they cannot be compared (different platforms, different nature of result, etc.).

There are several conclusions we can draw from the experimental results (shown in Table 2). First, the number of samples from our algorithm (Chen algorithm implementing the Massart bound) are larger than in the algorithm from [30]. This is because they do hypothesis testing, which requires less samples than even estimating the probability of a property, while we learn a DTMC accurate for this property. For *herman* and *leader*, the different is small (2.5x), because it is a case where the Massart bound is very efficient (80 times better than Okamoto-Chernoff implemented in PRISM). The *egl* system is the worst-case for the Massart bound (the probability of the property is $\frac{1}{2}$), and it coincides with Okamoto-Chernoff. The difference with [30] is 40x in that case. Also, as shown in *egl*, paths in our algorithm can be a bit larger than in the algorithm from [30], where they can be stopped early by the reachability analysis. However, the differences are never larger than 3x. On the other hand, we learn a model representative of the original system for a given property, while [30] only provide a yes/no answer to hypothesis testing (performing SMC evaluating the probability of a property with the Massart bound would give exactly the same number of samples as we report for our learning algorithm). Last, the reachability analysis from [30] does time out or memory out on some complex systems, which is not the case with our algorithm.

8 Conclusion

In this paper, we provided theoretical grounds for obtaining global PAC bounds when learning a DTMC: we bound the error made between the behaviors of the model and of the system, formalized using temporal logics. While it is not possible to obtain a learning framework for LTL properties, we provide it for the whole CTL logic. For subsets of CTL, e.g. for a fixed timed-to-failure property, we obtain better bounds, as efficient as Statistical MC. Overall, this work should help in the recent trends of establishing trusted machine learning [16].

Our techniques are useful for designers of systems for which probabilities are governed by uncertain forces (e.g. error rates): in this case, it is not easy to have a lower bound on the minimal transition probability, but we can assume that the set of transitions is known. Technically, our techniques provides rationale to set the constant in Laplace smoothing, otherwise left to an expert to set.

Some cases remain problematic, such as systems where states are visited very rarely. Nevertheless, we foresee potential solutions involving rare event simulation [21]. This goes beyond the scope of this work and it is left to future work.

References

1. Pranav Ashok, Jan Kretínský, and Maximilian Weininger. PAC statistical model checking for markov decision processes and stochastic games. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 497–519. Springer, 2019.
2. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
3. Luca Bortolussi and Guido Sanguinetti. Learning and Designing Stochastic Processes from Logical Constraints. In *Quantitative Evaluation of Systems - 10th International Conference, QEST, Buenos Aires, Argentina*, pages 89–105, 2013.
4. Manuele Brambilla, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo. Property-driven design for swarm robotics. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS, Valencia, Spain*, pages 139–146, 2012.
5. Jorge Castro and Ricard Gavaldà. Towards Feasible PAC-Learning of Probabilistic Deterministic Finite Automata. In *Grammatical Inference: Algorithms and Applications, 9th International Colloquium, ICGI, Saint-Malo, France*, pages 163–174, 2008.
6. Jianhua Chen. Properties of a New Adaptive Sampling Method with Applications to Scalable Learning. In *Web Intelligence, Atlanta*, pages 9–15, 2013.
7. Stanley F. Chen and Joshua Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. *Computer Speech and Language*, 13(4):359–394, 1999.
8. Yingke Chen, Hua Mao, Manfred Jaeger, Thomas Dyhre Nielsen, Kim Guldstrand Larsen, and Brian Nielsen. Learning Markov Models for Stationary System Behaviors. In *NASA Formal Methods - 4th International Symposium, NFM, Norfolk, VA, USA*, pages 216–230, 2012.
9. H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statist.*, 23(4):493–507, 1952.
10. Alexander Clark and Franck Thollard. PAC-learnability of Probabilistic Deterministic Finite State Automata. *Journal of Machine Learning Research*, 5:473–497, 2004.
11. Edmund M. Clarke and E. Allen Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, pages 52–71, 1981.
12. William G. Cochran. *Contributions to Survey Sampling and Applied Statistics*, chapter Laplace's ratio estimator, pages 3–10. Academic Press, New York, 1978.
13. Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský, and Tatjana Petrov. Linear Distances between Markov Chains. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, pages 20:1–20:15, 2016.
14. Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský, and Tatjana Petrov. Faster Statistical Model Checking for Unbounded Temporal Properties. *ACM Trans. Comput. Log.*, 18(2):12:1–12:25, 2017.
15. William A. Gale and Geoffrey Sampson. Good-Turing Frequency Estimation Without Tears. *Journal of Quantitative Linguistics*, pages 217–37, 1995.
16. Shalini Ghosh, Patrick Lincoln, Ashish Tiwari, and Xiaojin Zhu. Trusted Machine Learning: Model Repair and Data Repair for Probabilistic Models. In *AAAI-17 Workshop on Symbolic Inference and Optimization*, 2017.

17. T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *VMCAI*, volume 2937 of *LNCS*, pages 307–329, 2004.
18. Cyrille Jégourel, Jun Sun, and Jin Song Dong. Sequential Schemes for Frequentist Estimation of Properties in Statistical Model Checking. In *Quantitative Evaluation of Systems - 14th International Conference, QEST, Berlin, Germany*, pages 333–350, 2017.
19. Solomon Kullback and Richard A. Leibler. On Information and Sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.
20. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.
21. Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Rare Events for Statistical Model Checking an Overview. In *Reachability Problems - 10th International Workshop, RP, Aalborg, Denmark*, pages 23–35, 2016.
22. Masashi Okamoto. Some Inequalities Relating to the Partial Sum of Binomial Probabilities. *Annals of the Institute of Statistical Mathematics*, 10:29–35, 1958.
23. Amir Pnueli. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA*, pages 46–57, 1977.
24. Ad Ridder. Importance Sampling Simulations of Markovian Reliability Systems Using Cross-Entropy. *Annals OR*, 134(1):119–136, 2005.
25. Dorsa Sadigh, K. Driggs-Campbell, A. Puggelli, W. Li, V. Shia, R. Bajcsy, A. Sangiovanni-Vincentelli, S. Sastry, and S. Seshia. Data-driven probabilistic modeling and verification of human driver behavior. In *In Formal Verification and Modeling in Human-Machine Systems - AAAI Spring Symposium*, 2014.
26. Chris Sherlaw-Johnson, Steve Gallivan, and Jim Burridge. Estimating a Markov Transition Matrix from Observational Data. *The Journal of the Operational Research Society*, 46(3):405–410, 1995.
27. Leslie G. Valiant. A Theory of the Learnable. *Commun. ACM*, 27(11):1134–1142, 1984.
28. Abraham Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945.
29. Jingyi Wang, Jun Sun, Qixia Yuan, and Jun Pang. Should We Learn Probabilistic Models for Model Checking? A New Approach and An Empirical Study. In *Fundamental Approaches to Software Engineering - 20th International Conference, FASE, Uppsala, Sweden*, pages 3–21, 2017.
30. Håkan L. S. Younes, Edmund M. Clarke, and Paolo Zuliani. Statistical verification of probabilistic properties with unbounded until. In *SBMF'10*, pages 144–160, 2010.
31. Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Computer Aided Verification, 14th International Conference, CAV ,Copenhagen, Denmark*, pages 223–235, 2002.
32. P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian Statistical Model Checking with Application to Stateflow/Simulink Verification. *FMSD*, 43(2):338–367, 2013.

Appendix A Content of the Appendices

In this supplement, to ease the readability, we recall the Okamoto bound and a LTL definition in Appendix B. We provide the proof of Theorem 2 in Appendix C, the proof of Lemma 1 in Appendix D and the proof of Theorem 4 for $\overline{\text{Cond}}$ in Appendix E. For sake of completeness, we also provide the algorithm for the fixed time-to-failure property in Appendix F. Finally, we provide a more detailed description of System 4 in Appendix G.

Appendix B Some useful recalls

We recall for the readers the Okamoto inequality [22] (also called the Chernoff bound), the Massart bound and a definition of Linear Temporal Logic (LTL). Note however that the Okamoto inequality is not used in the paper and that our work focuses on CTL more than LTL.

Theorem 7 (Okamoto bound). *Let $\varepsilon > 0$, δ such that $0 < \delta < 1$ and $\hat{\gamma}_W$ be the crude Monte-Carlo estimator of probability γ. If $n \geq \frac{1}{2\varepsilon^2} \log\left(\frac{2}{\delta}\right)$,*

$$\mathbb{P}(|\gamma - \hat{\gamma}_W| > \varepsilon) \leq \delta.$$

Theorem 8 (Massart bound). *For all γ such that $0 < \gamma < 1$ and any ϵ such that $0 < \epsilon < \min(\gamma, 1-\gamma)$, we have the following inequality:*

$$Pr(|\hat{\gamma}_n - \gamma| > \epsilon) \leq 2\exp\left(-n\epsilon^2 h_a(\gamma, \epsilon)\right) \tag{10}$$

$$\textit{where} \quad h_a(\gamma,\epsilon) = \begin{cases} 9/2\ ((3\gamma+\epsilon)(3(1-\gamma)-\epsilon))^{-1} & \textit{if } 0 < \gamma < 1/2 \\ 9/2\ ((3(1-\gamma)+\epsilon)(3\gamma+\epsilon))^{-1} & \textit{if } 1/2 \leq \gamma < 1 \end{cases}$$

Definition 4 (LTL). *Let Prop be the set of atomic propositions. LTL is defined by the following grammar $\varphi ::= \bot \mid \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$, with $p \in$ Prop and Ne**X**t, **G**lobally, **F**inally and **U**ntil the standard modal temporal operators.*

We refer to [2] for more details about the semantics and the grammar of this logic.

Appendix C Negative result for LTL

Theorem 2. *Given $\varepsilon > 0$, $0 < \delta < 1$, and a finite set W of paths, there is no learning strategy such that, for all LTL formula φ,*

$$\mathbb{P}(|\gamma(A,\varphi) - \gamma(\hat{A}_W, \varphi)| > \varepsilon) \leq \delta \tag{11}$$

Proof. We prove it by defining a sequence of LTL properties that violates the specification above. As we show, it only relies on a single deviation in one transition. This is thus independent of the learning strategy.

Let $s_u \in S$ be a state that can be visited arbitrarily often from s_0 and let $s_v \in S$ be a non-unique successor of s_u. Assume that $\hat{A}_W = (\hat{a}_{ij})_{1 \leq i,j \leq m}$ is an estimate of $A = (a_{ij})_{1 \leq i,j \leq m}$ and note $\tau > 0$ the deviation between $\hat{a}_{uv}$ and a_{uv}. For simplicity, we assume $\hat{a}_{uv} = a_{uv} + \tau$ but a similar proof can be done with $\hat{a}_{uv} = a_{uv} - \tau$.

Let φ_n be the property "Transition $s_u s_v$ occurs at most $(a_{uv} + \tau/2)n$ times during the n first visits of s_i". This property is a LTL property since it can be written as a finite composition of $\mathbf{X}$, $\wedge$ and $\vee$ (for more details, see the similar property in Section 7.2 of [13]). Let $(X_k)_{1 \leq k \leq n}$ be n independent Bernoulli random variables from the set of transitions possible in s_u to $\{0, 1\}$ assigning 1 when $s_u s_v$ is taken after the k-th visit of s_u and 0 if another transition is taken after the k-th visit of s_u. Then, we can rewrite:

$$\mathbb{P}(\varphi_n) = \mathbb{P}\left(\frac{1}{n}\sum_{k=1}^{n} X_k \leq a_{uv} + \tau/2\right) \tag{12}$$

By the law of large numbers, $\frac{1}{n}\sum_{k=1}^{n} X_k$ tends to a_{uv} with respect to A when n tends to the infinity. Then,

$$\gamma(A, \varphi_n) = \mathbb{P}_A\left(\frac{1}{n}\sum_{k=1}^{n} X_k \leq a_{uv} + \tau/2\right) \underset{n \to \infty}{\longrightarrow} 1.$$

But, with respect to $\hat{A}_W$, $\frac{1}{n}\sum_{k=1}^{n} X_k$ tends to $a_{uv} + \tau$. So,

$$\gamma(\hat{A}_W, \varphi_n) = \mathbb{P}_{\hat{A}_W}\left(\frac{1}{n}\sum_{k=1}^{n} X_k \leq a_{uv} + \tau/2\right) \underset{n \to \infty}{\longrightarrow} 0$$

Thus, $\gamma(A, \varphi_n) - \gamma(\hat{A}_W, \varphi_n) \underset{n \to \infty}{\longrightarrow} 1$ almost surely. More precisely, given $\varepsilon > 0$, δ, $0 < \delta < 1$ and a finite run W, there exists a rank N such that specification 11 can not be fulfilled for properties φ_n, $n \geq N$.

Appendix D Proof of Lemma 1 in Proposition 1

For recall, we defined $q_W(u)$ the number of occurrences of sequence u in the traces of W. Note that u can be a state, an individual transition or even a path.

Lemma 1. *For all set of traces W, there exists a set of traces W' such that:*

$$\forall s, t, \frac{q_{W'}(s \cdot t)}{q_{W'}(s)} = \frac{q_W(s \cdot t)}{q_W(s)} \quad \wedge$$
$$\forall r, s, t \in V, q_{W'}(r \cdot s \cdot t) = \frac{q_{W'}(r \cdot s) \times q_{W'}(s \cdot t)}{q_{W'}(s)}$$

We use the following definitions in the proof.

Definition 2 (Equivalence). *Two sets of traces W and W' are equivalent if for all s, t, $\frac{q_W(s,t)}{q_W(s)} = \frac{q_{W'}(s,t)}{q_{W'}(s)}$.*

Definition 3 (s-factor). *Given a trace r and a state s, F is an s-factor of r if F is a factor of r and F starts by s. Moreover, F is elementary if it does not contain any other s.*

A trace can then be seen as a set of factors $BF_1 \dots F_k E$ where B is the special factor beginning, F_i are some s-factors for all i and E is the special ending s-factor. We can notice that for all trace r and r' obtained by permutation of the F_i, $\{r\}$ and $\{r'\}$ are equivalent. Without loss of generality, we suppose that states s_j such that there exists a transition (s_j, s) are states $s_1, \dots, s_Q$. In W, we denote n_i the number of transitions (s, s_i), m_j the number of transitions (s_j, s) and $q_{i,j} = \frac{q_W(s_j \cdot s) \times q_W(s \cdot s_i)}{q_W(s)}$. $q_{i,j}$ represents then the number of times a transition (s_j, s) should be followed by a transition (s, s_i). By definition, we have that $\sum_i n_i = \sum_j m_j = q_W(s)$ and $\forall i, j, q_{i,j} > 0$. We denote $k = q_W(s)$. Finally, for a factor F, we denote by $q_{F,i,j}$ the number of apparitions of (s_j, s, s_i) in F.

Proof. (of Lemma 1) Let us suppose that W is made of only one trace $r = BF_1 \dots F_f E$.

We prove the lemma by induction on $s \in V$, then induction on the number of predecessors of s.

Let suppose that every factor in $\{B, F_1, \dots, F_f\}$ ends with s_1, that $f = m_1 - 1$ and that the sequence $s_1 s$ never appears neither in B nor in F_l for all l nor in E. We also suppose that $\forall i, \forall j > 1$, $q_{B,i,j} + \sum_{l=1}^{f} q_{F_l,i,j} + q_{E,i,j} = q_{i,j}$. It means that for all $j > 1$ for all i, we have $q_{W'}(s_j \cdot s \cdot s_i) = \frac{q_{W'}(s_j \cdot s) \times q_{W'}(s \cdot s_i)}{q_{W'}(s)}$ and we just have to consider s_1.

Let r' be $BF_1 \dots F_f E$. r' is equivalent to r. We also obtain that for all i, $q_{r,i,1} = q_{i,1}$ since there are exactly $q_{i,1}$ factors starting by s_i. Furthermore, $\forall i, \forall j > 1$, $q_{r,i,j} = q_{B,i,j} + \sum_{l=1}^{f} q_{F_l,i,j} + q_{E,i,j} = q_{i,j}$. Indeed, none was added and we did not break those already existing. Then, $\forall i, \forall j, q_{W'}(s_j \cdot s \cdot s_i) = \frac{q_{W'}(s_j \cdot s) \times q_{W'}(s \cdot s_i)}{q_{W'}(s)}$.

Now, let us consider when there are J states to deal with, $J > 1$, and the factors $\{B, F_1 \cdots F_f, E\}$ such that $f = \sum_{j=1}^{J} m_j - 1$. Besides, for all $j \leq J$, exactly m_j factors in $\{B, F_1 \cdots F_f\}$ end with s_j and for all i, exactly $\sum_{j=1}^{J} q_{i,j}$ factors in $\{F_1 \cdots F_f, E\}$ start with ss_i. Furthermore, for all $j \leq J$, the sequence $s_j s$ never appears neither in B nor in F_l for all l and for all i, for all $j > J$, $q_{B,i,j} + \sum_{l=1}^{f} q_{F_l,i,j} + q_{E,i,j} = q_{i,j}$.

We create new factors in order to deal with s_J by merging the existing one. We apply the following algorithm:

Since $\sum_i q_{i,j} = m_j$, there is always one factor ending by S_J that can be chosen. Let us suppose that there is no candidate for F_2. It means that no factor other than F_1 starts by ss_i, and then $\sum_{j=1}^{J} q_{i,j} \leq q_{i,J}$ (number available at start

Merge($Factors, J$)
for *i from* 1 *to* Q **do**
  for *l from* 1 *to* $q_{i,J}$ **do**
    Choose F_1 ending by s_J, a factor $F_2 \neq F_1$ beginning by s_i, we denote $F' = F_1F_2$
    $Factors = Factors \setminus \{F_1, F_2\} \cup \{F'\}$
Output: $Factors$

smaller than number used). We deduce that for all $j < J$, $q_{i,j} = 0$ and that is absurd.

We obtain the set of factors$\{B', F'_1, \cdots, F'_{f'}, E'\}$. We have merged m_J factors, then $f' = f - m_J = \sum_{j=1}^{j-1} m_j - 1$. For all $j < J$, the number of factors ending with s_j has not changed. For all i, there are $\sum_{j=1}^{J} q_{i,j} - q_{i,J} = \sum_{j=1}^{J-1} q_{i,j}$ factors in $\{F'_1 \cdots F'_{f'}, E'\}$ starting with ss_i. Furthermore, for all $j < J$, the sequence s_js still never appears neither in B nor in F_l for all l and for all i, for all $j \geq J$, $q_{B,i,j} + \sum_{l=1}^{f} q_{F_l,i,j} + q_{E,i,j} = q_{i,j}$.

At start, when considering all elementary factors $\{B, F_1, \cdots, F_f, E\}$, we have $f = k - 1 = \sum_{j=1}^{Q} m_j - 1$ and for all j, exactly m_j factors in $\{B, F_1 \cdots F_f\}$ ends with s_j and for all i, exactly $\sum_{j=1}^{Q} q_{i,j} = n_i$ factors in $\{F_1 \cdots F_f, E\}$ start with ss_i. Besides, since all factors are elementary, no sequence s_js appears in any of them and trivially, for all $j > Q$, $q_{B,i,j} + \sum_{l=1}^{f} q_{F_l,i,j} + q_{E,i,j} = 0$. Thus, the requirements are met.

Appendix E Proof of Theorem 4 for $\overline{\text{Cond}}$

Theorem 4. *Denoting φ the property of reaching state s in DTMC A, we have:*

$$|\gamma(A, \varphi) - \gamma(A_\eta, \varphi)| < \frac{\ell_s \cdot \eta}{\overline{Cond}_s^{\ell_s}(A)}$$

Proof. When using $\overline{\text{Cond}_s^{\ell_s}}$ defined with $\overline{R^*(s)}$ instead of $\text{Cond}_s^{\ell_s}$ defined with $R^*(s)$, we need to consider w_1 over $R_1(s)$, the set of states that can reach s with probability 1, instead of just $\{s\}$. This set $R_1(s)$ is the same in A and A_η. We do not have $w_1 \cdot A_\eta^{t-\ell_s} = w_1 \cdot A^{t-\ell_s}$ anymore. However, as t tends to infinity, with very high probability w_1 will be sent to s in both $A_\eta^{t-\ell_s}$ and $A^{t-\ell_s}$, and the difference will be arbitrarily small. We actually have $w_1 \cdot (A_\eta^{t-\ell_s} - A^{t-\ell_s}) \leq (1-r)^t$, for $r > 0$ the probability to reach s in ℓ_s steps (we take the minimal value of r between $A_\eta^{\ell_s}$ and A^{ℓ_s}). We obtain the inductive formula $E(t) \leq (1 - \text{Cond}_s^{\ell_s}(A))E(t - \ell_s) + \ell_s \cdot \eta + (1-r)^t$. It yields for all $t \in \mathbb{N}$:

$$E(t) \leq (\ell_s \cdot \eta) \sum_{i=1}^{\infty} (1 - \text{Cond}_s^{\ell_s}(A))^i + \sum_{i=1}^{i \cdot \ell_s \leq t} (1 - \text{Cond}_s^{\ell_s}(A))^i (1-r)^{\lfloor \frac{t}{\ell_s} \rfloor - i}$$

Let us denote $\varepsilon_t = \sum_{i=1}^{i\cdot\ell_s \leq t} (1 - \mathrm{Cond}_s^{\ell_s}(A))^i (1-r)^{\lfloor \frac{t}{\ell_s} \rfloor - i}$, which is the only part differing with the previous case. We then have

$$E(t) \leq \frac{\ell_s \cdot \eta}{Cond_s^{\ell_s}(A)} + \varepsilon_t$$

We prove now that ε_t tends towards 0 as t tends to infinity, and thus we obtain the same result as previously by taking this limit. Let us denote $t' = \lfloor \frac{t}{\ell_s} \rfloor$.

Now, either $(1 - \mathrm{Cond}_s^{\ell_s}(A)) < (1-r)$ or $(1 - \mathrm{Cond}_s^{\ell_s}(A)) > (1-r)$ or $(1 - \mathrm{Cond}_s^{\ell_s}(A)) = (1-r)$. The last case is trivial as it give $\varepsilon_t = t'(1 - \mathrm{Cond}_s^{\ell_s}(A))^{t'}$, which indeed tends towards 0 as t tends to ∞. The two other cases are symmetric. Let us consider $(1 - \mathrm{Cond}_s^{\ell_s}(A)) < (1-r)$. We have $\varepsilon_t = \sum_{i=1}^{t'} (1 - \mathrm{Cond}_s^{\ell_s}(A))^i (1-r)^{t'-i} = (1-r)^{t'} \sum_{i=1}^{t'} (\frac{1 - \mathrm{Cond}_s^{\ell_s}(A)}{(1-r)})^i$. As the fraction is smaller than 1, the infinite sum converges, and the term $(1-r)^{t'}$ makes ε_t tends towards 0 as t tends towards infinity. For $(1 - \mathrm{Cond}_s^{\ell_s}(A)) > (1-r)$, we extract $(1 - \mathrm{Cond}_s^{\ell_s}(A))^{t'}$ and sum over $(\frac{(1-r)}{1 - \mathrm{Cond}_s^{\ell_s}(A)})^i$, with $(\frac{(1-r)}{1 - \mathrm{Cond}_s^{\ell_s}(A)}) < 1$. □

Appendix F Algorithm for the fixed time-to-failure property

A run W is observed from s_0 and every time s_0 or s_F are observed, the reset operation is performed, and a new path ω is being generated. W is the set of all those paths. In the work, we assume that the probability of reaching s_0 or s_F is 1 in order to guarantee the termination of the algorithm. since s_F is immediately followed by s_0.

Algorithm 2: Learning a matrix accurate for time-to-failure property

```
Learning(A, s_0, s_F, δ, ε)
n_succ = 0
n = 1 (number of times s_0 has been visited)
s = s_0 (current state)
while n < H(n, n_{succ,ε,δ}) do
    ω_n = s_0 and W = ω_1 ··· ω_n
    while s ≠ s_0 or s ≠ s_F do
        Observe the next state s' (sampled with respect to A)
        Update ω_n and Â_W
        if s' = s_0 or s' = s_F then
            Output z(ω_n, φ), n_succ ← n_succ + z(ω_n, φ) and n ← n + 1
```

Output: $\hat{A}_W$

Appendix G System 4

System 4 can be modeled with probabilistic model checker Prism[2] as a continuous time Markov chain (CTMC) that comprises three types (1, 2, 3) of three components each that may fail independently. Note however that we do not simulate the times between two changes of states but only the transitions between states, that lead to learn the induced DTMC instead. The components fail with rate $\lambda = 0.2$ and are repaired with rate $\mu = 1$. In addition, components are repaired with priority according to their type (type i has highest priority than type j if $i < j$). Components of type 1 and 2 are repaired simultaneously if at least two of their own type have failed. Type 3 components are repaired one by one as soon as one has failed. The probability transitions from state i to state j is given by the rate of the transition from the CTMC between state i and state j divided by the sum of all the rates of the enabled transitions from state i. The initial state is the state in which all the components are operational and the failure state is the state in which all the components are broken. We provide below the Prism code of the model for the readers who are interested to investigate this model in details:

```
ctmc

const int n=3;
const double lambda = 0.2;
const double mu = 1.0;

module type1
state1 : [0..n] init 0;
[] state1 < n -> (n-state1)*lambda : (state1'=state1+1);
[] state1 >=2 -> mu : (state1'=0);
endmodule

module type2
state2 : [0..n] init 0;
[] state2 < n -> (n-state2)*lambda : (state2'=state2+1);
[] state2 >=2 & state1 < 2 -> mu : (state2'=0);
endmodule

module type3
state3 : [0..n] init 0;
[] state3 < n -> (n-state3)*lambda : (state3'=state3+1);
[] state3 > 0 & state2 < 2 & state1 < 2 -> mu : (state3'=state3-1);
endmodule

label "failure" = state1 = n & state2 = n & state3 = n;
```

[2] http://www.prismmodelchecker.org/

Verification of Neural Networks: Specifying Global Robustness using Generative Models

Nathanaël Fijalkow
CNRS, LaBRI, Université de Bordeaux, and
The Alan Turing Institute, London

Mohit Kumar Gupta
Indian Institute of Technology Bombay

Abstract

The success of neural networks across most machine learning tasks and the persistence of adversarial examples have made the verification of such models an important quest. Several techniques have been successfully developed to verify robustness, and are now able to evaluate neural networks with thousands of nodes. The main weakness of this approach is in the specification: robustness is asserted on a validation set consisting of a finite set of examples, *i.e.* locally.

We propose a notion of global robustness based on generative models, which asserts the robustness on a very large and representative set of examples. We show how this can be used for verifying neural networks. In this paper we experimentally explore the merits of this approach, and show how it can be used to construct realistic adversarial examples.

1 Introduction

We consider the task of certifying the correctness of an image classifier, *i.e.* a system taking as input an image and categorising it. As a main example we will consider the MNIST classification task, which consists in categorising hand-written digits. Our experimental results are later reproduced for the drop-in dataset Fashion MNIST (Xiao et al. (2017)).

The usual evaluation procedure consists in setting aside from the dataset a validation set, and to report on the success percentage of the image classifier on the validation set. With this procedure, it is commonly accepted that the MNIST classification task is solved, with some convolutional networks achieving above 99.7% accuracy (see *e.g.* Ciregan et al. (2012); Wan et al. (2013)). Further results suggest that even the best convolutional networks cannot be considered to be robust, given the persistence of adversarial examples: a small perturbation – invisible to the human eye – in images from the dataset is enough to induce misclassification (Szegedy et al. (2014)).

This is a key motivation for the verification of neural networks: can we assert the robustness of a neural network, *i.e.* the absence of adversarial examples? This question has generated a growing interest in the past years at the crossing of different research communities (see *e.g.* Huang et al. (2017); Katz et al. (2017); Weng et al. (2018); Gehr et al. (2018); Mirman et al. (2018); Gopinath et al. (2018); Katz et al. (2019)), with a range of prototype tools achieving impressive results. The robustness question is formulated as follows: given an image x and $\varepsilon > 0$, are all ε-perturbations of x correctly classified?

We point to a weakness of the formalisation: it is *local*, meaning it is asserted for a given image x (and then typically checked against a finite set of images). In this paper, we investigate a *global* approach for specifying the robustness of an image classifier. Let us start from the ultimate robustness objective, which reads:

> For every category, for every *real-life image* of this category and for every *perturbation* of this image, the perturbed image is correctly classified.

Formalising this raises three questions:

1. How do we quantify over *all* real-life images?
2. What are *perturbed* images?
3. How do we *effectively* check robustness?

In this work we propose a formalisation based on generative models. A generative model is a system taking

Technical report.

as input a random noise and generating images, in other words it represents a probabilistic distribution over images.

Our specification depends on two parameters (ε, δ). Informally, it reads:

> An image classifier is (ε, δ)-robust with respect to a generative model if the probability that for a noise x, all ε-perturbations of x generate correctly classified images is at least $1 - \delta$.

The remainder of the paper presents experiments supporting the claims that the global robustness specification has the following important properties.

Global. The first question stated above is about quantifying over all images. The global robustness we propose addresses this point by (implicitly) quantifying over a very large and representative set of images.

Robust. The second question is about the notion of perturbed images. The essence of generative models is to produce images reminiscent of real images (from the dataset); hence testing against images given by a generative model includes the very important perturbation aspect present in the intuitive definition of correctness.

Effective. The third question is about effectivity. We will explain that global robustness can be effectively evaluated for image classifiers built using neural networks.

Related work

Xiao et al. (2018) train generative models for finding adversarial examples, and more specifically introduce a different training procedure (based on a new objective function) whose goal is to produce adversarial examples. Our approach is different in that we use generative models with the usual training procedure and objective, which is to produce a wide range of realistic images.

2 Global Correctness

This section serves as a technical warm-up for the next one: we introduce the notion of *global correctness*, a step towards our main definition of *global robustness*.

We use $\mathbb{R}^d$ for representing images with $||\cdot||$ the infinity norm over $\mathbb{R}^d$, and let C be the set of categories, so an image classifier represents a function $\boldsymbol{C} : \mathbb{R}^d \to C$.

A generative model represents a distribution over images, and in effect is a neural network which takes as input a random noise in the form of a p-dimensional vector x and produces an image $\boldsymbol{G}(x)$. Hence it represents a function $\boldsymbol{G} : \mathbb{R}^p \to \mathbb{R}^d$. We typically use a Gaussian distribution for the random noise, written $x \sim \mathcal{N}(0, 1)$.

Our first definition is of *global correctness*, it relies on a first *key* but simple idea, which is to compose a generative model $\boldsymbol{G}$ with an image classifier $\boldsymbol{C}$: we construct a new neural network $\boldsymbol{C} \circ \boldsymbol{G}$ by simply rewiring the output of $\boldsymbol{G}$ to the input of $\boldsymbol{C}$, so $\boldsymbol{C} \circ \boldsymbol{G}$ represents a distribution over categories. Indeed, it takes as input a random noise and outputs a category.

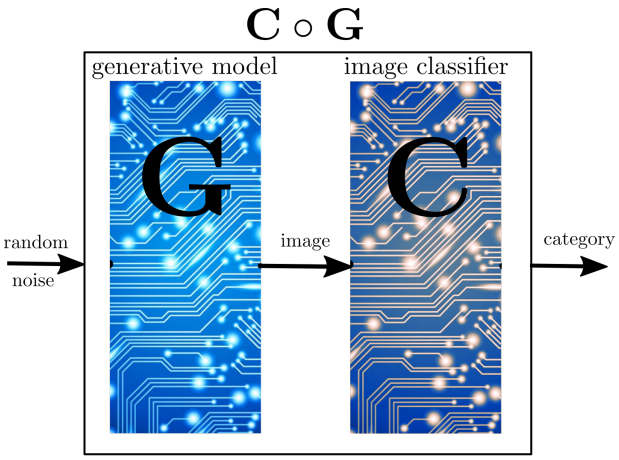

Figure 1: Composition of a generative model with an image classifier

Definition 1 (Global Correctness)**.** *Given for each $c \in C$ a generative model $\boldsymbol{G}_c$ for images of category c, we say that the image classifier $\boldsymbol{C}$ is δ-correct with respect to the generative models $(\boldsymbol{G}_c)_{c \in C}$ if for each $c \in C$,*

$$\mathbb{P}_{x \sim \mathcal{N}(0,1)}(\boldsymbol{C} \circ \boldsymbol{G}_c(x) = c) \geq 1 - \delta.$$

In words, the probability that for a noise x the image generated (using $\boldsymbol{G}_c$) is correctly classified (by $\boldsymbol{C}$) is at least $1 - \delta$.

Assumptions

Our definition of global correctness hinges on two properties of generative models:

1. generative models produce a wide variety of images,
2. generative models produce (almost only) realistic images.

The first assumption is the reason for the success of generative adversarial networks (GAN) (Goodfellow et al. (2014)). We refer for instance to Karras et al. (2018) and to the attached website `thispersondoesnotexist.com` for a demo.

In our experiments the generative models we used are out of the shelf generative adversarial networks (GAN) (Goodfellow et al. (2014)), with 4 hidden layers of respectively 256, 512, 1024, and 784 nodes, producing images of single digits.

To test the second assumption we performed a first experiment called the *manual score experiment*. We picked 100 digit images using a generative model and asked 5 individuals to tell for each of them whether they are "near-perfect", "perturbed but clearly identifiable", "hard to identify", or "rubbish", and which digit they represent. The results are that 96 images were correctly identified; among them 63 images were declared "near-perfect" by all individuals, with another 26 including "perturbed but clearly identifiable", and 8 were considered "hard to identify" by at least one individual yet correctly identified. The remaining 4 were "rubbish" or incorrectly identified. It follows that against this generative model, we should require an image classifier to be at least .89-correct, and even .96-correct to match human perception.

Algorithm

To check whether a classifier is δ-correct, the Monte Carlo integration method is a natural approach: we sample n random noises $x_1, \ldots, x_n$, and count for how many x_i's we have that $\boldsymbol{C} \circ \boldsymbol{G}_c(x) = c$. The central limit theorem states that the ratio of positives over n converges to $\mathbb{P}_{x\sim\mathcal{N}(0,1)}(\boldsymbol{C} \circ \boldsymbol{G}_c(x) = c)$ as $\frac{1}{\sqrt{n}}$. It follows that $n = 10^4$ samples gives a 10^{-2} precision on this number.

In practice, rather than sampling the random noises independently, we form (large) batches and leverage the tensor-based computation, enabling efficient GPU computation.

3 Global Robustness

We introduce the notion of global robustness, which gives stronger guarantees than global correctness. Indeed, it includes the notion of perturbations for images.

The usual notion of robustness, which we call here *local robustness*, can be defined as follows.

Definition 2 (Local Robustness)**.** *We say that the image classifier $\boldsymbol{C}$ is ε-robust around the image $y \in \mathbb{R}^d$ of category c if*

$$\forall y',\ ||y - y'|| \leq \varepsilon \implies \boldsymbol{C}(y') = c.$$

In words, all ε-perturbations of y are correctly classified (by $\boldsymbol{C}$).

One important aspect in this definition is the choice of the norm for the perturbations (here we use the infinity norm). We ignore this as it will not play a role in our definition of robustness. A wealth of techniques have been developed for checking local robustness of neural networks, with state of the art tools being able to handle nets with thousands of neurons.

Assumptions

Our definition of global robustness is supported by the two properties of generative models discussed above in the context of global correctness, plus a third one:

3. generative models produce perturbations of realistic images.

To illustrate this we designed a second experiment called the *random walk experiment*: we perform a random walk on the space of random noises while observing the ensued sequence of images produced by the generative model. More specifically, we pick a random noise x_0, and define a sequence $(x_i)_{i\geq 0}$ of random noises with x_{i+1} obtained from x_i by adding a small random noise to x_i; this induces the sequence of images $(\boldsymbol{G}(x_i))_{i\geq 0}$. The result is best visualised in an animated GIF (see the Github repository), see also the first 16 images in Figure 2. This supports the claim that images produced with similar random noises are (often) close to each other; in other words the generative model is (almost everywhere) continuous.

Our definition of global robustness is reminiscent of the *provably approximately correct* learning framework developed by Valiant (1984). It features two parameters. The first parameter, δ, quantifies the probability that a generative model produces a realistic image. The second parameter, ε, measures the perturbations on the noise, which by the continuity property discussed above transfers to perturbations of the produced images.

Definition 3 (Global Robustness)**.** *Given for each $c \in C$ a generative model $\boldsymbol{G}_c$ for images of category c, we say that the image classifier $\boldsymbol{C}$ is (ε, δ)-robust with respect to the generative models $(\boldsymbol{G}_c)_{c\in C}$ if for each $c \in C$,*

$$\mathbb{P}_{x\sim\mathcal{N}(0,1)}(\forall x',\ ||x-x'|| \leq \varepsilon \implies \boldsymbol{C}\circ\boldsymbol{G}_c(x') = c) \geq 1-\delta.$$

In words, the probability that for a noise x, all ε-perturbations of x generate (using $\boldsymbol{G}$) images correctly classified (by $\boldsymbol{C}$) is at least $1 - \delta$.

Figure 2: The random walk experiment

Algorithm

To check whether a classifier is (ε, δ)-robust, we extend the previous ideas using the Monte Carlo integration: we sample n random noises $x_1, \ldots, x_n$, and count for how many x_i's the following property holds:

$$\forall x,\ ||x_i - x|| \leq \varepsilon \implies \boldsymbol{C} \circ \boldsymbol{G}_c(x) = c.$$

The central limit theorem states that the ratio of positives over n converges to

$$\mathbb{P}_{x \sim \mathcal{N}(0,1)}(\forall x',\ ||x - x'|| \leq \varepsilon \implies \boldsymbol{C} \circ \boldsymbol{G}_c(x') = c)$$

as $\frac{1}{\sqrt{n}}$. As before, it follows that $n = 10^4$ samples gives a 10^{-2} precision on this number.

In other words, checking global robustness reduces to combining Monte Carlo integration with checking local robustness.

4 Experiments

The code for all experiments can be found on the Github repository

`https://github.com/mohitiitb/NeuralNetworkVerification_GlobalRobustness`.

All experiments are presented in Jupyter notebook format with pre-trained models to be easily reproduced. Our experiments are all reproduced on the drop-in Fashion-MNIST dataset (Xiao et al. (2017)), obtaining similar results.

We report on experiments designed to assess the benefit of these two notions, whose common denominator is to go from a local property to a global one by composing with a generative model.

We first evaluate the global correctness of several image classifiers, showing that it provides a finer way of evaluating them than the usual test set. We then turn to global robustness and show how the negation of robustness can be witnessed by realistic adversarial examples.

The second set of experiments addresses the fact that both global correctness and robustness notions depend on the choice of a generative model. We show that this dependence can be made small, but that it can also be used for refining the correctness and robustness notions.

Choice of networks

In all the experiments, our base case for image classifiers have 3 hidden layers of increasing capacities: the first one, referred to as "small", has layers with $(32, 64, 200)$ (number of nodes), "medium" corresponds to $(64, 128, 256)$, and "large" to $(64, 128, 512)$. The generative model are as described above, with 4 hidden layers of respectively 256, $512, 1024$, and 784 nodes.

For each of these three architectures we either use the standard MNIST training set (6,000 images of each digit), or an augmented training set (24,000 images), obtained by rotations, shear, and shifts. The same distinction applies to GANs: the "simple GAN" uses the standard training set, and the "augmented GAN" the augmented training set.

Finally, we work with two networks obtained through robust training procedures. The first one was proposed by Mądry et al. (2018) for the MNIST Adversarial Example Challenge (the goal of the challenge was to find adversarial examples, see below), and the second one was defined by Papernot et al. (2016) through the process of defense distillation.

Evaluating Global Correctness

We evaluated the global correctness of all the image classifiers mentioned above against simple and augmented GANs, and reported the results in the table below. The last column is the usual validation procedure, meaning the number of correct classification on the MNIST test set of 10,000 images. They all perform very well, and close to perfectly (above 99%), against this metric, hence cannot be distinguished. Yet the composition with a generative model reveals that their performance outside of the test set are actually different. It is instructive to study the outliers for each image classifier, *i.e.* the generated images which are incorrectly classified. We refer to the Github repository for more experimental results along these lines.

Finding Realistic Adversarial Examples

Checking the global robustness of an image classifier is out of reach for state of the art verification tools. Indeed, a single robustness check on a medium size net takes somewhere between dozens of seconds to a

Classifier	simple GAN	augmented GAN	test set
Standard training set			
small	98.89	92.82	99.79
medium	99.15	93.16	99.76
large	99.38	93.80	99.80
Augmented training set			
small	97.84	95.2	99.90
medium	99.11	96.53	99.86
large	99.25	97.66	99.84
Robust training procedures			
Mądry et al. (2018)	98.87	93.17	99.6
Papernot et al. (2016)	99.64	94.78	99.17

few minutes, and to get a decent approximation we need to perform tens of thousands local robustness checks. Hence with considerable computational efforts we could analyse one image classifier, but could not perform a wider comparison of different training procedures and influence on different aspects. Thus our experiments focus on the negation of robustness, which is finding realistic adversarial examples, that we define now.

Definition 4 (Realistic Adversarial Example)**.** *An ε-realistic adversarial example for an image classifier $\boldsymbol{C}$ with respect to a generative model $\boldsymbol{G}$ is an image $\boldsymbol{G}(x)$ such that there exists another image $\boldsymbol{G}(x')$ with*

$$||x - x'|| \leq \varepsilon \text{ and } \boldsymbol{C} \circ \boldsymbol{G}(x) \neq \boldsymbol{C} \circ \boldsymbol{G}(x')$$

In words, x and x' are two ε-close random noises which generate images $\boldsymbol{G}(x)$ and $\boldsymbol{G}(x')$ that are classified differently by $\boldsymbol{C}$.

Note that a realistic adversarial example is not necessarily an adversarial example: the images $\boldsymbol{G}(x)$ and $\boldsymbol{G}(x')$ may differ by more than ε. However, this is the assumption 3. discussed when defining global robustness, if x and x' are close, then *typically* $\boldsymbol{G}(x)$ and $\boldsymbol{G}(x')$ are two very resemblant images, so the two notions are indeed close.

We introduce two algorithms for finding realistic adversarial examples, which are directly inspired by algorithms developed for finding adversarial examples. The key difference is that realistic adversarial examples are searched by analysing the composed network $\boldsymbol{C} \circ \boldsymbol{G}$.

Let us consider two digits, for the sake of explanation, 3 and 8. We have a generative model $\boldsymbol{G}_8$ generating images of 8 and an image classifier $\boldsymbol{C}$.

The first algorithm is a *black-box attack*, meaning that it does not have access to the inner structure of the networks and it can only simulate them. It consists in sampling random noises, and performing a local search for a few steps. From a random noise x, we inspect the random noise $x + \delta$ for a few small random noises δ, and choose the random noise x' maximising the score of 3 by the net $\boldsymbol{C} \circ \boldsymbol{G}_8$, written $\boldsymbol{C} \circ \boldsymbol{G}_8(x_i)[3]$ in the pseudocode given in Algorithm 1. The algorithm is repeatedly run until a realistic adversarial example is found.

Algorithm 1: The black-box attack for the digits 3 and 8.

Data: A generative model $\boldsymbol{G}_8$ and an image classifier $\boldsymbol{C}$. A parameter $\varepsilon > 0$.
$N_{\text{step}} \leftarrow 16$ (number of steps)
$N_{\text{dir}} \leftarrow 10$ (number of directions)
$x_0 \sim \mathcal{N}(0, 1)$
for $i = 0$ *to* $N_{step} - 1$ **do**
  $s_{\max} \leftarrow \boldsymbol{C} \circ \boldsymbol{G}_8(x_i)[3]$ (score of 3)
  $x_{i+1} \leftarrow x_i$
  for $j = 0$ *to* $N_{dir} - 1$ **do**
    $\delta_j \sim \mathcal{N}(0, \frac{\varepsilon}{N_{\text{step}}})$
    $s \leftarrow \boldsymbol{C} \circ \boldsymbol{G}_8(x_i + \delta_j)[3]$
    if $s > s_{max}$ **then**
      $s_{\max} \leftarrow s$
      $x_{i+1} \leftarrow x_i + \delta_j$
  if $\boldsymbol{C} \circ \boldsymbol{G}_8(x_0) \neq \boldsymbol{C} \circ \boldsymbol{G}_8(x_{i+1})$ **then**
    return x_0 (ε-realistic adversarial example)

The second algorithm is a *white-box attack*, meaning that it uses the inner structure of the networks. It is similar to the previous one, except that the local search is replaced by a gradient ascent to maximise the score of 3 by the net $\boldsymbol{C} \circ \boldsymbol{G}_8$. In other words, instead of choosing a direction at random, it follows the gradient to maximise the score. It is reminiscent of the projected gradient descent (PGD) attack, but performed on the composed network. The pseudocode is given in Algorithm 2.

Both attacks successfully find realistic adversarial examples within less than a minute. The adjective "realistic", which is subjective, is justified as follows: most attacks constructing adversarial examples create un-

Algorithm 2: The white-box attack for the digits 3 and 8.

Data: A generative model $\boldsymbol{G}_8$ and an image classifier $\boldsymbol{C}$. A parameter $\varepsilon > 0$.

$N_{\text{step}} \leftarrow 16$ (number of steps)
$\alpha \leftarrow \frac{\varepsilon}{N_{\text{step}}}$ (step)
$x_0 \sim \mathcal{N}(0, 1)$
for $i = 0$ *to* $N_{step} - 1$ **do**
  $x_{i+1} \leftarrow x_i - \alpha \cdot \text{Grad}_{\boldsymbol{C} \circ \boldsymbol{G}_8}(x_i)[3]$
  if $\boldsymbol{C} \circ \boldsymbol{G}_8(x_0) \neq \boldsymbol{C} \circ \boldsymbol{G}_8(x_{i+1})$ **then**
    return x_0 (ε-realistic adversarial example)

realistic images by adding noise or modifying pixels, while with our definition the realistic adversarial examples are images produced by the generative model, hence potentially more realistic. See Figure 3 for some examples.

On the Dependence on the Generative Model

Both global correctness and robustness notions are defined with respect to a generative model. This raises a question: how much does it depend on the choice of the generative model?

To answer this question we trained two GANs using the exact same training procedure but with two disjoint training sets, and used the two GANs to evaluate several image classifiers. The outcome is that the two GANs yield sensibly the same results against all image classifiers. This suggests that the global correctness indeed does not depend dramatically on the choice of the generative model, provided that it is reasonably good and well-trained. We refer to the Github repository for a complete exposition of the results.

Since the training set of the MNIST dataset contains 6,000 images of each digit, splitting it in two would not yield two large enough training sets. Hence we used the extended MNIST (EMNIST) dataset Cohen et al. (2017), which provided us with (roughly) 34,000 images of each digit, hence two disjoint datasets of about 17,000 images.

On the Influence of Data Augmentation

Data augmentation is a classical technique for increasing the size of a training set, it consists in creating new training data by applying a set of mild transformations to the existing training set. In the case of digit images, common transformations include rotations, shear, and shifts.

Unsurprisingly, crossing the two training sets, *e.g.* using the standard training set for the image classifier and an augmented one for the generative model yields worse results than when using the same training set. More interestingly, the robust networks Mądry et al. (2018); Papernot et al. (2016), which are trained using an improved procedure but based on the standard training set, perform well against generative models trained on the augmented training set. In other words, one outcome of the improved training procedure is to better capture the natural image transformations, even if they were never used in training.

5 Conclusions

We defined two notions: global correctness and global robustness, based on generative models, aiming at quantifying the usability of an image classifier. We performed some experiments on the MNIST dataset to understand the merits and limits of our definitions. An important challenge lies ahead: to make the verification of global robustness doable in a reasonable amount of time and computational effort.

Bibliography

Dan Ciregan, Ueli Meier, and Juergen Schmidhuber. Multi-column deep neural networks for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CCVPR)*, pages 3642–3649, June 2012. doi: 10.1109/CVPR.2012.6248110. URL `https://ieeexplore.ieee.org/document/6248110`.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373, 2017. URL `http://arxiv.org/abs/1702.05373`.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI2: safety and robustness certification of neural networks with abstract interpretation. In *IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018. doi: 10.1109/SP.2018.00058. URL `https://doi.org/10.1109/SP.2018.00058`.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Conference on Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets`.

Divya Gopinath, Guy Katz, Corina S. Pasareanu, and Clark Barrett. Deepsafe: A data-driven approach for assessing robustness of neural networks. In *Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 3–19, 2018.

Figure 3: Examples of realistic adversarial examples. On the left hand side, against the smallest net, and on the right hand side, against Mądry et al. (2018)

doi: 10.1007/978-3-030-01090-4_1. URL https://doi.org/10.1007/978-3-030-01090-4_1.

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Computer-Aided Verification (CAV)*, pages 3–29, 2017. doi: 10.1007/978-3-319-63387-9_1. URL https://doi.org/10.1007/978-3-319-63387-9_1.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018. URL http://arxiv.org/abs/1812.04948.

Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer-Aided Verification (CAV)*, pages 97–117, 2017. doi: 10.1007/978-3-319-63387-9_5. URL https://doi.org/10.1007/978-3-319-63387-9_5.

Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. The marabou framework for verification and analysis of deep neural networks. In *Computer-Aided Verification (CAV)*, pages 443–452, 2019. doi: 10.1007/978-3-030-25540-4_26. URL https://doi.org/10.1007/978-3-030-25540-4_26.

Aleksander Mądry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

Matthew Mirman, Timon Gehr, and Martin T. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning (ICML)*, pages 3575–3583, 2018. URL http://proceedings.mlr.press/v80/mirman18b.html.

Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy (SP)*, pages 582–597, 2016. doi: 10.1109/SP.2016.41. URL https://doi.org/10.1109/SP.2016.41.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014. URL http://arxiv.org/abs/1312.6199.

Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. doi: 10.1145/1968.1972. URL https://doi.org/10.1145/1968.1972.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning (ICML)*, volume 28, pages 1058–1066, 2013. URL http://proceedings.mlr.press/v28/wan13.html.

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning (ICML)*, pages 5273–5282, 2018. URL http://proceedings.mlr.press/v80/weng18a.html.

Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3905–3911, 2018. doi: 10.24963/ijcai.2018/543. URL https://doi.org/10.24963/ijcai.2018/543.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *CoRR*,

abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

ML+FV=♡?
A Survey on the Application of Machine Learning to Formal Verification

Moussa Amrani[1,2], Adrien Bibal[1,2], and Pierre-Yves Schobbens[1,2]

[1] Faculty of Computer Science, University of Namur,
[2] Namur Digital Institute (NaDI) {Moussa.Amrani — Adrien.Bibal}@unamur.be

Abstract. Formal Verification (Fv) and Machine Learning (Ml) can seem incompatible due to their opposite mathematical foundations and their use in real-life problems: Fv mostly relies on discrete mathematics and aims at ensuring correctness; Ml often relies on probabilistic models and consists of learning patterns from training data. In this paper, we show that they are complementary in practice, and explore how Ml helps Fv in its classical approaches: model-checking, theorem-proving, static analysis and Sat solving. We draw a landscape of the current practice and catalog some of the most prominent uses of Ml inside Fv tools, thus offering a new perspective on Fv techniques that can help researchers and practitioners to better locate the possible synergies. We discuss lessons learned, and point to possible improvements

Formal Verification (Fv) aims at guaranteeing correctness properties of software and hardware systems. In that sense, a system is safe with respect to the checked properties. Machine Learning (Ml) aims at learning patterns from training data for various purposes; the derived model generalizes from the data it was trained on. Both Fv and Ml are grounded on solid mathematical foundations: the former uses mostly discrete mathematics, fixpoints and abstractions to specify (concrete/abstract) semantics, properties of interest and the checking process itself; the latter uses in general continuous mathematics and/or probability theory to infer models.

We would like to present an ongoing effort to provide a comprehensive introduction of the various ways Ml contributes to enhance Fv tools' efficiency. Since many verification problems have high complexity, improving the speed, efficiency, or accuracy of Fv tools is crucial for answering real-life Fv problems. In this context, Ml may offer an efficient help in navigating the many open possibility a tool needs to explore to provide an answer.

To achieve this goal, we propose to catalog the challenges Fv faces that may be handled through Ml techniques, called *themes*, and characterise each theme with a corresponding Ml *task*, i.e. an Ml problem category (such as classification, regression, clustering, reinforcement, etc.). We illustrate by listing many literature contributions, and exploring all classical approaches available in Fv: SAT solvers (Sat), Theorem Provers (Tp), Model-Checking (Mc), Abstract Interpretation (Ia) and more generally Static Analysis (Sa).

To the best of our knowledge, many surveys, systematic literature reviews and general introductions exist in specific Fv approaches, but none spans over all the spectrum of the main Fv approaches. By covering them, we aim at extracting valuable Fv approach-specific, but also transversal, lessons, as well as common trends for the usage of Ml within Fv. We therefore provide a high-level snapshot of the current practice in each Fv approach. This study does obviously not replace specialised, focused reviews and overviews intended for practitioners of each domain, but rather serves the purpose of providing a general introduction to the themes researchers are currently focusing on, and detailing how they originally tackled these challenges.

This work started during Fall 2017, after attending the Nfm'17 (Nasa Formal Method): during the Conference, several papers demonstrated the importance of enhancing Fv tools with Ml techniques, as well as applying Fv techniques to ensure safety properties of Ml models applied in safety-critical systems. A first version appeared as a non-reviewed paper on arXiv [1]: it built upon around 80 contributions, collected between Sept. and Nov. 2017. However, many research teams and projects started to work on those topics, producing numerous new results. Therefore, a year later, we extended our work with new contributions (collected between February and April 2019), culminating in a study with 200+ literature contributions distributed over all approaches. Our study provides the following scientific contributions:

- We provide a catalog of themes for each Fv approach, presented in a systematic way: each theme details the corresponding Ml task, and provides a commented list of relevant contributions, summarising for each theme which Ml models we encountered in the literature.
- We analyze the literature to extract general observations on the use of Ml inside Fv tools, and to identify some trends and lessons, with an insight on what the future may be.

We plan to submit our work as a Journal Paper during Spring 2020, accompanied with a comprehensive, searchable repository listing all collected contributions.

Depending on (obviously, the acceptance, then) the time allocated for the presentation, and the background of the audience we could built upon, we would:

- Present (some of) the themes, illustrated with important contributions, occuring in two approaches, namely Sat and Sa, as they crystallise many patterns and common issues found in other approaches;
- Discuss general lessons and current challenges for Fv.

During the Workshop, we hope to foster discussions with the specialised audience, but also to get feedback on the angles we chose for our study.

References

1. Amrani, M., Lúcio, L., Bibal, A.: ML + FV = ♡? A Survey on the Application of Machine Learning to Formal Verification. arXiv e-prints arXiv:1806.03600 (Jun 2018)