Applying Deep Learning For Imitating Adaptive Agent Behavior in Statistical Software Testing

André Reichstaller, Benedikt Eberhardinger, Hella Seebach, Alexander Knapp, and Wolfgang Reif

Institute for Software & Systems Engineering, University of Augsburg, Germany

{ *reichstaller, eberhardinger, seebach, knapp, reif* } *@isse.de*

Abstract Statistical test generation builds on profiles which describe the estimated conditions of the system under test's environment. Such environmental profiles, however, do not directly provide us with inputs for testing particular system components, as those mostly depend on the output of others. We thus additionally need to estimate this output if we want to maintain statistical accuracy. Instantiating this task for the isolated testing of self-organization mechanisms between adaptive agents, this paper investigates the application of deep learning techniques for imitating the agents' output. The proposed technique is evaluated on a simulated self-organizing grid of power plants.

1 Statistical Testing Self-Organization Mechanisms Using Mock-ups

Statistical software testing applies stochastically generated sequences of test inputs to a system under test (SuT) which represent its anticipated use. In doing so, this approach allows the test engineer to reason about the system's expected field quality [1]. Furthermore, the obtained models of the environment, in which the SuT is expected to be employed, can be used to prioritize and thus to reduce test efforts in general. This reduction is of particular interest when dealing with very huge and ramified state spaces of the SuT's environment, since, without prioritization, the so called *state space explosion* is likely to give rise to test suites of uncontrollable size.

In previous work, we explained this phenomenon and consequential challenges with particular regard to the class of *self-organizing, adaptive systems* [2]. This kind of software systems is usually composed of an *agent layer* comprising a number of adaptive agents that are associated with their environment, and an *organization layer* including a self-organization mechanism (SO mechanism) which is responsible to reorganize the agents if necessary.

Providing an approach for isolated testing of the self-organization mechanism within the organization layer, we instantiated the idea of statistical testing by branching the huge environmental state space using so called *environmental profiles*. However, by means of a case study considering a self-organizing energy grid, we found that the environment itself and the underlying probabilistic model does not directly determine the searched test input sequences for the SO mechanism under test, as this exclusively depends on the agent layer. The implemented agent behavior maps the environmental profile to an agent-specific state. Thus, in order to obtain statistically relevant test inputs, one needs to take an indirection: the output of environmental profiles is piped in stochastic test mock-ups imitating the estimated agent behavior – the so called *influence functions* – the output of which can then be applied to the SO mechanism as test input. Figure 1 visualizes this approach.

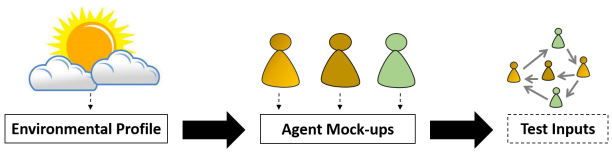

Figure 1: Proposed process for statistical testing SO mechanisms. Mock-ups simulate agent behavior in response to environmental conditions which are provided by environmental profiles. Outputs of the mock-ups serve as test inputs for the SO mechanism.

While we usually have access to lots of statistical data concerning the SuT's environment, e.g., weather data or other physical models, which can be used to construct environmental profiles, it is rather unclear how to obtain the mentioned influence functions for constructing agent mock-ups from a limited set of observations. This task can be characterized as follows: Given a set of logs that report the agent behavior under specific environmental conditions, we are seeking a generalization that allows us to predict the agent behavior even for previously unseen conditions.

Solving this task by employing heuristics of the test engineer, as we proposed in [2] for a first proof of concept, leads to even more effort and could cause statistical inadequacies in test inputs. These conclusions would question the economic sense of a statistical software testing method at all. Trying to overcome this limitation, we investigated the use of machine learning (ML) methodologies, in particular deep learning techniques, for deriving influence functions from observed agent behavior. The assumption behind is that those techniques could automatically generalize over given observations, leading to higher statistical adequacy at lower costs. In fact, the literature shows several examples where ML approaches were successfully applied in statistical software testing [3, 4, 5].

However, while the mentioned approaches used ML to learn regularities within the SuT's and the environmental structure, we seek to map statistic information about the environment to the expected behavior of adaptive agents. We argue that this indirection leads to statistically adequate test inputs for a higher level organization algorithm. This kind of *imitation learn-*

ing was previously applied for activities like predictive maintenance [6, 7] or anomaly detection [8, 9, 10]. However, to the best of our knowledge, it had not been considered yet being useful for solving the task with which we are concerned with: *statistical testing of higher level system components*.

The following sections present the results of a first, rough investigation. For evaluation, we considered a simulated grid of heterogeneous power plants that are hierarchically organized by an SO mechanism. Section 2 introduces this case study in more detail. Subsequently, we introduce our methodology for learning the agent's behavior from observations in Sect. 3. The results will be finally compared with those of some handmade heuristics in Sect. 4.

2 Case Study: Self-organizing Virtual Power Plants in Smart Grids

The wide-spread installation of weather-dependent power plants (PP) as well as the advent of new consumer types like electric vehicles put a lot of strain on power grids. To address the rising challenges of

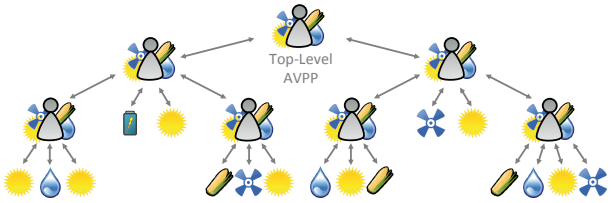

Figure 2: Hierarchical system structure of a future autonomous and decentralized power management system: Power plants are structured into systems of systems represented by AVPPs that act as intermediaries to decrease the complexity of control and scheduling.

future power management systems, Steghöfer et al. presented the concept of *Autonomous Virtual Power Plants* (AVPPs) in [11]. AVPPs represent self-organizing groups of two or more power plants of various types (cf. Fig. 2). The organizational structure represents a *partitioning*, i.e., every PP is a member of exactly one AVPP, which is established and maintained by a (partitioning-based) self-organization mechanism. Each AVPP has to fulfill a fraction of the overall power demand in the energy grid. For this purpose, each AVPP autonomously calculates schedules for directly subordinate dispatchable PPs. The calculation of the schedule depends on different influence factors that might effect the members of the AVPP. Foremost, external influence factors, e.g., the wind condition for a wind turbine, are challenging to handle since they introduce an uncertainty into the system. To cope with the accruing uncertainties, AVPPs autonomously adapt their structure to changing internal or environmental conditions. Thus, they are able to live up to the responsibility of maintaining an organizational structure enabling the system to operate reliably which is measured by a numeric value called *trust*. In particular, if an AVPP repeatedly cannot satisfy its assigned fraction of the overall demand or compensate for its local uncertainties, it triggers a reorganization of the partitioning.

3 Deriving Influence Functions

The agent layer of the present self-organizing, adaptive system is formed by the various power plants and AVPPs. Affected by their environment each of these agents produces output. This output not only comprises plain energy values, but also a trust value $\in [0, 1]$ assessing the reliability of the power plant's forecast. And exactly those trust values form test inputs for the SO mechanism, as reorganization in the proposed system is exclusively based on the notion of trust. Figure 3 visualizes this data flow at the example of weather dependent power plants.

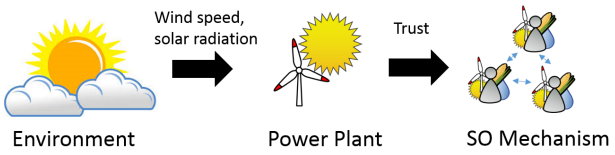

Figure 3: The output of a weather-dependent power plant is influenced by environmental conditions, such as wind speed and solar radiation. A trust value signals the estimated reliability of forecasts.

The task of generating statistical tests for the present SO mechanism can thus be formulated as finding trust vectors (one trust value per considered power plant) which appear to be most likely within the real world. In order to obtain such trust vectors, we need to consider both, statistics concerning the environment as well as the assumed agent behavior.

Let us consider this process for a solar power plant. To obtain its most likely outputs we would first register weather statistics in the region our PP will be located. These statistics could be captured within an environmental profile. Given such a profile we still need to find a mapping from environmental states to trust values the agent is supposed to return. This mapping, the influence function, can be seen as a mock-up of the real system replicating its behavior. As we have no unequivocal specification mapping weather states to prognosticated trust values for our PP, we could employ some handmade heuristics which operationalize our assumptions. For instance, we could assume that the trust in the forecast of a solar power plant descends with increasing variance in solar radiation.

Striving to reduce test effort, which arises when reasoning about heuristics, while maximizing statistical accuracy, which might suffer from wrong assumptions or subjective assessments of the tester, we investigated a more automatable approach using ML for deriving the influence function. Inspired by the state of the art in related research fields, such as predictive mainte-

nance or anomaly detection, we decided to solve this regression task by using and training deep artificial neural networks (ANNs). This approach offers the following advantages: (1) The popularity of training such models resulted in lots of literature, frameworks and tutorials. This kind of accessible support makes the considered techniques applicable also for test engineers which are no ML professionals. (2) It was shown that ANNs are able to approximate arbitrary functions (in dependence on the meta-parameters). It is thus allowed to assume that using this kind of models we are able to estimate almost all influence functions.

For the sake of brevity and with regard to advantage (1), we will, in the following, not go into detail about the various types, training algorithms and underlying mathematics concerning regression with ANNs. The interested reader may be referred to standard literature, such as [12], and frameworks with related tutorials, such as [13].

4 Evaluation

Though we have access to a fully integrated simulation environment of the aforementioned system, for evaluation purposes, we pretended to have only given the following artifacts: *Environmental profiles* which model the most likely weather conditions for a considered area within Markov chains, as well as *log* sequences for a set of particular solar power plants describing their output at a given time in response to particular environmental states. For the time being, our evaluation was exclusively concerned with estimating the effort and measuring the accuracy of the learned, predictive mock-ups emulating the behavior of considered power plants. The desired generation of statistical tests for the SO mechanism under the use of our models would, however, additionally demand access to the SuT itself as well as some kind of test oracle. As our methodology can be easily plugged-in into the process described in [2], we refer to this work for more details on those mentioned test artifacts.

We based the search for adequate heuristics on the following argumentation: In general, an influence function is meant to map one environmental state to the expected output of the considered agent. In the present case, this would mean to map the weather conditions (we exclusively considered the prevalent wind speed and solar radiations) at a particular time step t to the trust value of a weather dependent power plant observable at time step $t + 1$. However, even if it seems natural that the performance of a weather dependent power plant depends on the weather one time step before, such a dependence seems not adequate for the trust value, as this only assesses the quality of a forecast. This forecast refers to the difference between the expected and the actual output. It seems likely to us that in case of the weather dependent PPs, the energy output forecast depends on the quality of accessible weather forecasts. As these are, however, unknown to us, we introduced another assumption: the quality of weather forecasts might depend on the weather conditions observed at the last n time steps: the more variance in weather, the more unreliable the forecast. Another assumption was, that the loss of weather forecasts can be approximated by a constant, whose value however might differ between the different power plants and locations.

Overall, we considered the following heuristics:

Heuristic 1 Constant trust value at 0.8. This constant had been chosen based on the assumption that weather forecasts might be inaccurate in 20% of cases.

Heuristic 2 Trust value at time step $t + 1$ depends on weather conditions at time step t. The worse the weather conditions for the energy output of the PP, the worse the estimated trust.

Heuristic 3 Trust value depends on variance within the weather conditions of the last 5 time steps. The more variance, the worse the estimated trust value.

Apart from the mentioned heuristics, we trained an ANN to represent the influence function. We used the following methodology for training: We generated logs by observing the agent behavior in response to randomly generated environmental profiles within our simulation. These logs then were used for training. For this, we utilized a kind of on-the-fly batch training by successively increasing the training set with alongside generated logs, instead of writing the whole bunch of logs in a file. Figure 4 shows the resulting learning curve. One can see that training progress is rather low in the first phase (450 epochs), but suddenly grows in the following. After 500 epochs of training we achieve a mean absolute error of < 0.1 on randomly generated environmental profiles.

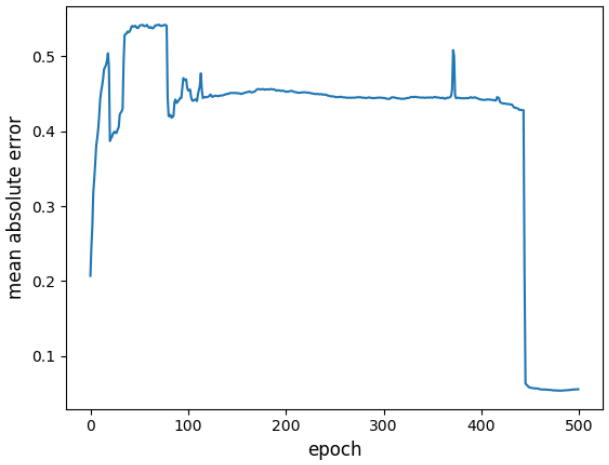

Figure 4: Learning curve describing the trend of mean absolute error with increasing training time.

To compare these results to the presented heuristics, we evaluated all of them together with the trained ANN on a separate run of 100 time steps simulating the environmental profiles we used in our previous work

KPI / Methodology	Learned Model	Heuristic 1	Heuristic 2	Heuristic 3
mean absolute error	0.08	0.3	0.3	0.25
standard deviation	0.07	0.08	0.17	0.18

Table 1: Comparison of ANN results with different heuristics.

[2] for evaluation. Table 1 shows the results. Note that we did no parameter fitting on the heuristics as we assume that the accuracy of a handmade heuristic can be arbitrary optimized at the expense of time. Our intention – to show that this job can be outsourced to an ML algorithm – is confirmed by the high accuracy of the ANN.

5 Conclusion & Outlook

The evaluation suggests that using deep learning techniques for imitating adaptive agent behavior can provide us with good accuracy at low costs. However, there still are several limitations. One interesting question might, for instance, be the following: *how to imitate agents that implement a feedback loop?* In this case the agent output at time step t would (partially) depend on the output at previous time steps $t - n$ for some positive n. We see two options to cope with that: We could (1) fix a particular agent strategy, i.e., find an influence function estimating agent reactions that are in between the observations; or we could (2) decode the feedback mechanism itself by considering historical behavior within the influence function. On the other hand, our approach to statistical testing *covers the average, main scenarios.* A test engineer, however, needs also to consider the risk of failures which might be triggered by unforeseen environmental dynamics at run-time, too. We will investigate an extension of the presented approach for covering this kind of worst-case scenarios in future work.

Acknowledgment This research is sponsored by the research project *Testing Self-Organizing, Adaptive Systems (TeSOS)* of the German Research Foundation.

References

[1] J. A. Whittaker and M. G. Thomason, "A Markov chain model for statistical software testing," *IEEE Trans. Software Engineering*, vol. 20, no. 10, pp. 812–824, 1994.

[2] B. Eberhardinger, G. Anders, H. Seebach, F. Siefert, A. Knapp, and W. Reif, "An approach for isolated testing of self-organization algorithms," in *Software Engineering for Self-Adaptive Systems III: Assurances* (R. de Lemos, H. Giese, H. A. Müller, and M. Shaw, eds.), vol. 9640 of *Lect. Notes Comp. Sci.*, Springer, 2017.

[3] C. Kallepalli and J. Tian, "Measuring and modeling usage and reliability for statistical web testing," *IEEE Trans. Software Engineering*, vol. 27, no. 11, pp. 1023–1036, 2001.

[4] N. Baskiotis, M. Sebag, M.-C. Gaudel, and S.-D. Gouraud, "A machine learning approach for statistical software testing," in *Proc. Intl. Joint Conf. Artifical Intelligence*, pp. 2274–2279, 2007.

[5] S. Poulding and J. A. Clark, "Efficient software verification: Statistical testing using automated search," *IEEE Trans. Software Engineering*, vol. 36, no. 6, pp. 763–777, 2010.

[6] P. Wang and G. Vachtsevanos, "Fault prognostics using dynamic wavelet neural networks," *AI EDAM*, vol. 15, no. 4, pp. 349–365, 2001.

[7] P. Jahnke, *Machine learning approaches for failure type detection and predictive maintenance.* Master's thesis, Technische Universität Darmstadt, 2015.

[8] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[9] D. Mutz, F. Valeur, G. Vigna, and C. Kruegel, "Anomalous system call detection," *ACM Trans. Information and System Security*, vol. 9, no. 1, pp. 61–93, 2006.

[10] S.-J. Han and S.-B. Cho, "Evolutionary neural networks for anomaly detection based on the behavior of a program," *IEEE Trans. Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 3, pp. 559–570, 2005.

[11] J.-P. Steghöfer, G. Anders, F. Siefert, and W. Reif, "A system of systems approach to the evolutionary transformation of power management systems," in *Proc. Informatik 2013 – Ws. "Smart Grids"*, Lecture Notes in Informatics, Bonner Köllen Verlag, 2013.

[12] S. Russell and P. Norvig, *Artificial Intelligence: A modern approach.* Prentice Hall, 1995.

[13] F. Chollet *et al.*, "Keras," 2015.