ETSI TR 104 066 V1.1.1 (2024-07)

Securing Artificial Intelligence;
Security Testing of AI

Reference
DTR/SAI-0015

Keywords
artificial intelligence, security

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Securing Artificial Intelligence (SAI).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

Security testing of AI aims at identifying vulnerabilities in AI models. On the one hand, security testing of AI has some commonalities with security testing of traditional software systems. On the other hand, the functioning of AI and in particular ML poses new challenges and requires different approaches for several reasons:

- There are significant differences between symbolic AI, sub symbolic AI, i.e. ML, versus traditional software systems that have strong implications on AI and ML security and on how to test their security properties.
- Non-determinism: AI-based systems can evolve at runtime (self-learning systems), and thus, security properties can degrade at runtime, too. If faced with the same input at different times, self-learning AI-based systems can provide different predictions.
- Test oracle problem: assigning a test verdict is different and more difficult for AI-based systems since not all expected results are known a priori.

- Data-driven algorithms: in contrast to traditional systems, (training) data forms the behaviour of sub symbolic AI, meaning security testing should be extended from the AI component to the data used for training or continuous learning of a system.

Testing consists of several activities that include test planning and control, test design, test implementation, test execution and test evaluation. The present document covers the testing activities test design, test execution and test evaluation. For that purpose, the present document introduces methods and metrics to design test cases (see clause 4), to measure the progress (see clause 5) and to evaluate test cases (see clause 6).

The present document addresses security testing approaches for AI, security test oracles for AI, and definition of test adequacy criteria for security testing of AI. Techniques of each of these topics are applied together to security test a ML component. Security testing approaches are used to generate test cases that are executed against the ML component. Security test oracles enable to calculate a test verdict to determine if a test case has passed, that is, no vulnerability has been detected, or failed, that is a vulnerability has been identified. Test adequacy criteria are used to determine the entire progress and can be employed to specify a stop condition for security testing.

The security testing approaches addressed by the present document are not solely related to security but to robustness as well. Issues with the robustness of ML components can result in both security and safety issues. Security issues of a ML component can enable an adversary to achieve a violation of one of the security properties, i.e. confidentiality, integrity, and availability. Safety issues of a ML component might endanger the environment in which the ML component and the system it is part of is operating. Security issues might also lead to safety issues when, for instance, the availability or integrity of safety measures is affected. Testing of robustness of ML components related to safety-issues in the Automotive domain has been discussed, for instance, in [i.1].

1 Scope

The present document identifies methods and techniques that are appropriate for security testing of ML-based components. Security testing of AI does not end at the component level. As for testing of traditional software, the integration with other components of a system needs to be tested as well. However, integration testing is not the subject of the present document.

The present document addresses:

- security testing approaches for AI;
- security test oracles for AI;
- definition of test adequacy criteria for security testing of AI.

Techniques of each of these topics should be applied together to security test of a ML component. Security testing approaches are used to generate test cases that are executed against the ML component. Security test oracles enable to calculate a test verdict to determine if a test case has passed, that is, no vulnerability has been detected, or failed, that is a vulnerability has been identified. Test adequacy criteria are used to determine the entire progress and can be employed to specify a stop condition for security testing.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Berghoff, C., Bielik, P., Neu, M., Tsankov, P., & von Twickel, A. (2021, June). Robustness Testing of AI Systems: A Case Study for Traffic Sign Recognition. In IFIP International Conference on Artificial Intelligence Applications and Innovations (pp. 256-267). Springer, Cham.

[i.2] American fuzzy lop.

[i.3] LLVM compiler infrastructure: "libFUZZER".

[i.4] Odena, A., & Goodfellow, I. (2018). Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. arXiv preprint arXiv:1807.10875.

[i.5] Chen, T. Y., Cheung, S. C., & Yiu, S. M. (2020). Metamorphic testing: a new approach for generating next test cases. arXiv preprint arXiv:2002.12543.

[i.6] McKeeman, W. M. (1998). Differential testing for software. Digital Technical Journal, 10(1), 100-107.

[i.7] Pei, K., Cao, Y., Yang, J., & Jana, S. (2017, October). Deepxplore: Automated whitebox testing of deep learning systems. In proceedings of the 26th Symposium on Operating Systems Principles (pp. 1-18).

[i.8] Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., & Madry, A. (2019). Adversarial examples are not bugs, they are features. Advances in neural information processing systems.

[i.9] Tramèr, F., Papernot, N., Goodfellow, I., Boneh, D., & McDaniel, P. (2017). The space of transferable adversarial examples. arXiv preprint arXiv:1704.03453.

[i.10] Demontis, A., Melis, M., Pintor, M., Jagielski, M., Biggio, B., Oprea, A., & Roli, F. (2019). Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In 28th USENIX security symposium (USENIX security 19) (pp. 321-338).

[i.11] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199.

[i.12] Jang, U., Wu, X., & Jha, S. (2017, December). Objective metrics and gradient descent algorithms for adversarial examples in machine learning. In Proceedings of the 33rd Annual Computer Security Applications Conference (pp. 262-277).

[i.13] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.

[i.14] Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533.

[i.15] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083.

[i.16] Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., & Li, J. (2018). Boosting adversarial attacks with momentum. In Proceedings of the IEEE™ conference on computer vision and pattern recogntion (pp. 9185-9193).

[i.17] Xie, C., Zhang, Z., Zhou, Y., Bai, S., Wang, J., Ren, Z., & Yuille, A. L. (2019). Improving transferability of adversarial examples with input diversity. In Proceedings of the IEEE™/CVF Conference on Computer Vision and Pattern Recognition (pp. 2730-2739).

[i.18] Croce, F., & Hein, M. (2020, November). Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In International conference on machine learning (pp. 2206-2216). PMLR.

[i.19] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., & Swami, A. (2016, March). The limitations of deep learning in adversarial settings. In 2016 IEEE™ European symposium on security and privacy (EuroS&P) (pp. 372-387). IEEE™.

[i.20] Loison, A., Combey, T., & Hajri, H. (2020). Probabilistic Jacobian-based Saliency Maps Attacks. arXiv preprint arXiv:2007.06032.

[i.21] Carlini, N., & Wagner, D. (2017, May). Towards evaluating the robustness of neural networks. In 2017 IEEE™ symposium on security and privacy (sp) (pp. 39-57). IEEE™.

[i.22] Moosavi-Dezfooli, S. M., Fawzi, A., & Frossard, P. (2016). Deepfool: a simple and accurate method to fool deep neural networks. In Proceedings of the IEEE™ conference on computer vision and pattern recognition (pp. 25).

[i.23] Ghiasi, A., Shafahi, A., & Goldstein, T. (2020). Breaking certified defenses: Semantic adversarial examples with spoofed robustness certificates. arXiv preprint arXiv:2003.08937.

[i.24] Croce, F., & Hein, M. (2020, November). Minimally distorted adversarial examples with a fast adaptive boundary attack. In International Conference on Machine Learning (pp. 2196-2205). PMLR.

[i.25] Xiao, C., Zhu, J. Y., Li, B., He, W., Liu, M., & Song, D. (2018). Spatially transformed adversarial examples. arXiv preprint arXiv:1801.02612.

[i.26] Mopuri, K. R., Garg, U., & Babu, R. V. (2017). Fast feature fool: A data independent approach to universal adversarial perturbations. arXiv preprint arXiv:1707.05572.

[i.27] Hayes, J., & Danezis, G. (2018, May). Learning universal adversarial perturbations with generative models. In 2018 IEEE™ Security and Privacy Workshops (SPW) (pp. 43-49). IEEE™.

[i.28] Chen, P. Y., Zhang, H., Sharma, Y., Yi, J., & Hsieh, C. J. (2017, November). Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (pp. 15-26).

[i.29] Brendel, W., Rauber, J., & Bethge, M. (2017). Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. arXiv preprint arXiv:1712.04248.

[i.30] Andriushchenko, M., Croce, F., Flammarion, N. & Hein, M. (2020, August). Square attack: a query-efficient black-box adversarial attack via random search. In European Conference on Computer Vision (pp. 484-501). Springer, Cham.

[i.31] Uesato, J., O'donoghue, B., Kohli, P., & Oord, A. (2018, July). Adversarial risk and the dangers of evaluating against weak attacks. In International Conference on Machine Learning (pp. 5025-5034). PMLR.

[i.32] Carlini, N., Athalye, A., Papernot, N., Brendel, W., Rauber, J., Tsipras, D. & Kurakin, A. (2019). On evaluating adversarial robustness. arXiv preprint arXiv:1902.06705.

[i.33] Engstrom, L., Tran, B., Tsipras, D., Schmidt, L., & Madry, A. (2018). A rotation and a translation suffice: Fooling CNNs with simple transformations.

[i.34] Alzantot, M., Sharma, Y., Chakraborty, S., Zhang, H., Hsieh, C. J., & Srivastava, M. B. (2019, July). Genattack: Practical black-box attacks with gradient-free optimization. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 1111-1119).

[i.35] Moosavi-Dezfooli, S. M., Fawzi, A., Fawzi, O., & Frossard, P. (2017). Universal adversarial perturbations. In Proceedings of the IEEE™ conference on computer vision and pattern recognition (pp. 1765-1773).

[i.36] Dong, Y., Zhang, P., Wang, J., Liu, S., Sun, J., Hao, J. & Ting, D. (2019). There is Limited Correlation between Coverage and Robustness for Deep Neural Networks. arXiv preprint arXiv:1911.05904.

[i.37] Zhu, H., Hall, P. A., & May, J. H. (1997). Software unit test coverage and adequacy. Acm computing surveys (csur), 29(4), 366-427.

[i.38] Kim, J., Feldt, R., & Yoo, S. (2019, May). Guiding deep learning system testing using surprise adequacy. In 2019 IEEE™/ACM 41st International Conference on Software Engineering (ICSE) (pp. 1039-1049). IEEE™.

[i.39] Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B. & Zhao, J. (2018, September). Deepgauge: Multi-granularity testing criteria for deep learning systems. In Proceedings of the 33rd ACM/IEEE™ International Conference on Automated Software Engineering (pp. 120-131).

[i.40] Weng, T. W., Zhang, H., Chen, P. Y., Yi, J., Su, D., Gao, Y. & Daniel, L. (2018). Evaluating the robustness of neural networks: An extreme value theory approach. arXiv preprint arXiv:1801.10578.

[i.41] Cisse, M., Bojanowski, P., Grave, E., Dauphin, Y., & Usunier, N. (2017). Parseval networks: Improving robustness to adversarial examples. arXiv preprint arXiv:1704.08847.

[i.42] Katz, G., Barrett, C., Dill, D. L., Julian, K., & Kochenderfer, M. J. (2017, July). Reluplex: An efficient SMT solver for verifying deep neural networks. In International Conference on Computer Aided Verification (pp. 97-117). Springer, Cham.

[i.43] Mayer, J., & Guderlei, R. (2004). Test oracles using statistical methods. Testing of component-based systems and software quality.

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

adversarial example: carefully crafted input which mislead a model to give an incorrect prediction

perturbation: semantically meaningless modification of an input

EXAMPLE: Perturbation can have the form of noise added to an image.

substitute model: model created by an adversary to craft transferable adversarial examples

NOTE 1: The substitute model performs the same task as the target model but may use a different ML technique or a different dataset.

NOTE 2: The terms surrogate model and substitute model are used synonymously.

surrogate model: See substitute model.

target label: label that an adversary wants the target model to output if fed with an adversarial example

target model: model an adversary wants to make wrong predictions

transferable adversarial example: adversarial example which is crafted for one model but can also fool a different model with a high probability

true label: correct label for an input from the ground truth

3.2 Symbols

For the purposes of the present document, the following symbols apply:

L_0	Pseudo distance (number of non-zero elements)
L_2	Euclidean distance
L_∞	Chebyshev distance
L_{flow}	Flow field function
L_p	Distance that needs to be specified by the parameter p with $p \in \{0, 2, \infty\}$

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AI	Artificial Intelligence
CLEVER	Cross Lipschitz Extreme Value for nEtwork Robustness
DSA	Distance-based Surprise Adequacy
FAB	Fast Adaptive Boundary attack
FGSM	Fast Gradient Sign Method
JSMA	Jacobian-based Saliency Map Attack
L-BFGS	computer-memory-Limited approximation of the Broyden-Fletcher-Goldfarb-Shanno algorithm
LSA	Likelihood-based Surprise Adequacy
ML	Machine Learning
NaN	Not a Number
PGD	Projected Gradient Descent
ReLU	Rectified Linear Unit
SAI	Securing Artificial Intelligence
SPSA	Simultaneous Perturbation Stochastic Approximation
TJSMA	Taylor JSMA
WJSMA	Weighted JSMA

ZOO Zeroth Order Optimization-based attacks

4 Security testing techniques

4.1 Introduction

Security testing techniques are used for designing test cases that are later on executed against an ML component. Such test cases consist of the input data that is fed to the ML component to identify a vulnerability, e.g. a susceptibility to a specific adversarial example. Clause 4 presents different approaches that can be employed for crafting such inputs. The presented testing approaches can be divided into those that have been developed for traditional software and can be employed for security testing of ML components as well, and those that are specific to ML. Furthermore, not all of them are security-specific but can be more versatile with respect to the quality characteristics in question.

NOTE: It is necessary to ensure that the system is not designed to recognise the adversarial examples used in a test environment and to run in such a way that the test is passed by bypassing normal operation.

4.2 Mutation testing

4.2.1 Coverage-guided fuzzing

Coverage-guided fuzzing is a technique that has been established for traditional software systems. For such systems, code coverage has been extensively used as coverage metrics together with genetic algorithms, mostly using binary mutation without protocol models, as in American Fuzzy Lop [i.2] and libFuzzer [i.3]. Odena et al. [i.4] transferred this approach to neural networks of different architectures. Instead of random binary mutation, they use specific mutators for images and text. For images, their approach mutates existing pictures by adding white noise either to the extent of a user-configurable variance or by a user-configurable L_∞ norm. As distance metric the approximate nearest neighbour that is greater than a given threshold is used and assume a higher coverage is the distance to the nearest neighbour is above this threshold.

NOTE: L_∞ norm or Chebyshev distance simply takes the (mathematically absolutely) largest component of a vector.

4.2.2 Metamorphic testing

Metamorphic testing [i.5] is a testing approach that relies on metamorphic relations to identify test inputs for which the relationships between their outputs are known or could be identified, for instance using statistical methods. Based on existing, passing test cases, new test cases can be derived using the metamorphic relations. Hence, metamorphic testing requires the identification of metamorphic relations as a first step. This can be a challenging task for complex scenarios where relationships between different inputs and output are not obvious. The simplest example of a metamorphic relation is for the sine function where two metamorphic relations can be derived from the periodicity of the sine function:

$$\sin x = \sin(x + 2\pi) \tag{1}$$

and

$$|\sin x| = |\sin(x + \pi)| \tag{2}$$

Metamorphic relations can be more complex than simple equality and the absolute value and can involve any mathematical function. They are usually specific to the problem domain.

4.3 Differential testing

Differential testing [i.6] is a testing technique developed for traditional software that uses another system as a reference system to identify deviations of the system under test when different behaviours of both systems can be observed. Test cases are generated randomly, and test cases that result in different behaviours between the system under test and the reference system are considered to have revealed a bug and are retained as regression test and for debugging purposes.

The approach has been adapted for deep learning systems by Pei et al. [i.7] using neuron coverage metric (see clause 5.2.2) to identify faults in neural networks by using an objective function that considers both systems and tries to find inputs that lead to different classification results using their gradients.

4.4 Adversarial attacks

4.4.1 Introduction to adversarial data generation

Adversarial data generation is a discipline of adversarial machine learning that encompasses techniques for generating such inputs for which an ML model generates incorrect outputs. Such inputs are close to known inputs with corrects outputs but despite that similarity have a different output. Such inputs are called adversarial samples or adversarial examples and they have first been developed for computer vision. The general approach adds some noise, e.g. to an image, to achieve a different output even though both inputs are so similar that there is no deviation that would justify a different output. Even though adversarial examples have been widely generated for images, they are possible to be generated for any kind of data.

The distance between an uncorrupted input with uncorrupted output and corrupted input with a corrupted output is measured using some norm. Most used norms are the L_0 norm, the L_2 norm and the L_∞ norm. They differ in how the approximate the distance and their computational complexity. L_0 norm measures just the number of non-zero elements of a vector after subtracting the vectors from input and corrupted input, L_2 measures the Euclidean distance between two vectors and is the computationally most complex norm, and L_∞ measures the largest difference of the elements of the difference vector. The lower the measured distance, the closer the corrupted inputs are to uncorrupted inputs. The closer corrupted inputs are to uncorrupted inputs, the more likely it is they remain undetected, e.g. by a human, or they are not successful due to a mitigation measure. Hence, the distance of the corrupted data to uncorrupted data generated by a generation technique including the used norm to measure the distance constitute one aspect of the power of attacks, referred to as quality of examples in Table 1. In general, many approaches are guided by the gradient of the cost function and work iteratively. Thus, there are lots of full-knowledge approaches that require access to the internals of the models. However, zero-knowledge approaches can estimate the gradient or employ a local search or use substitute models constructed from queries to the model under attack. Another way to measure the power of an attack generated through adversarial data generation is whether the technique can perform a specific corrupted output, i.e. if it is targeted, or whether it can just be able to generate any corrupted output.

An important aspect relevant for adversarial attacks is their property of transferability. Adversarial attacks transfer between models of different architecture and models trained with different training sets. Transferability of adversarial attacks has been investigated quite much. Several aspects have been investigated why adversarial attacks are transferable. For instance, transferability can result from non-robust features in the used dataset [i.8], but also from common adversarial subspaces shared between two ML models [i.9]. Model properties, e.g. properties of the input gradients, e.g. their size and the alignment between two models, have been also identified to facilitate the transferability of adversarial examples [i.10]. This property reduces the effort for the attacker as well as for testing. Thus, it makes the approach of building a database of adversarial samples justifiable.

Table 1 presents an overview of the characteristics and properties of adversarial data generation techniques that are presented in more detail in the remainder of clause 4.4.

Table 1: Overview of techniques for adversarial sample generation

Knowledge	Optimization	Technique	Computational Efficiency	Query Efficiency	Quality of examples	Supported norms	Targeted/ untargeted
Full knowledge	Gradient-based	L-BFGS	low	low	medium	L_2	targeted
		NewtonFool	high	medium	low	L_2	untargeted
		FGSM	high	high	low	L_∞	targeted
		Basic Iterative Method/Projected Gradient Descent	low	high	medium	L_∞	targeted
		JSMA	low	medium	high	L_0	targeted
		Carlini/Wagner	low	low	very high	L_0, L_2, L_∞	targeted
		DeepFool	high	low	high	L_2 adaptable any L_p norm	untargeted
		Shadow Attack	low	low	very high	adaptable any L_p norm	untargeted
		Fast Adaptive Boundary Attack	high	medium	very high	adaptable any L_p norm	untargeted
	Gradient-free	Spatial Transformation	low	low	high	specific (L_{flow} based on L_2)	targeted
		Fast Feature Fool	high	medium	high	L_∞	untargeted
		Generative Universal Adversarial Perturbations	low	low	high	L_2, L_∞	targeted
Zero knowledge	training a surrogate model	Surrogate models	low	low	any full knowledge	any full knowledge	targeted
	gradient estimation	ZOO	low	medium	high	L_2	targeted
		SPSA	low	low	high	adaptable any L_p norm	untargeted
	search-based	Rotation and Translation	high	low	high	n/a	untargeted
		Boundary Attack	low	low	medium	adaptable any L_p norm	untargeted
		Square Attack	medium	low	medium	L_2, L_∞	untargeted
		GenAttack	medium	low	medium	L_∞	targeted
		Universal Adversarial Perturbations	low	low	high	L_2, L_∞	untargeted

4.4.2 Security testing techniques requiring full knowledge

4.4.2.1 Gradient-based techniques

4.4.2.1.1 L-BFGS

L-BFGS is an optimization algorithm (belonging to the class of quasi-Newton methods) that Szegedy et al. [i.11] used to approximate the minimal distance (using the L_2 norm) between two images with different classification results by a deep neural network. The perturbed image using a minimizer r is approximated since the original box-constrained optimization problem is a hard problem.

L-BFGS is a computer-memory-limited approximation of the Broyden–Fletcher–Goldfarb–Shanno algorithm. Szegedy et al. use L-BFGS to approximate a perturbed image (denoted by $x_l' = x + r$) using line search to find the minimum c > 0 for which the minimizer r satisfies $f(x + r) = l$, i.e. lead to a different result of the classifier, by minimizing:

$$c \cdot \|r\| + J_f(x + r, l) \tag{3}$$

Where:

- $c > 0$;
- $J_f(x_l', l)$ is a continuous loss function of the classifier f: $\mathbb{R}^m \rightarrow \{1..k\}$ with k labels;
- l is the targeted label with $l \in \{1..k\}$;
- $x + r = x_l'$ is a perturbed image.

The approximation produces close adversarial samples at the cost of higher computational effort than FGSM since it uses the L_2 norm instead of L_∞ norm.

4.4.2.1.2 NewtonFool

Jang et al. [i.12] first proposed utilizing a gradient descent-based approach in producing adversarial samples. The attack is very simple in nature, wherein the goal of the algorithm is to consider the score distribution produced by the model for each possible class and attempting to find some perturbation that would reduce the probability of the original class. This attack intrinsically exploits the nature of the softmax layer present in classification-based models to derive the adversarial samples, at the cost of lowering its generalizability across data modalities.

The calculation of the adversarial sample is as follows:

$$x' = x + \frac{\delta \cdot (\nabla F_s(x))}{\|(\nabla F_s(x))\|^2} \tag{4}$$

Where:

- δ is the maximum value between the model gradient under L_2 norm and the relative difference of the higher class probability prior to the softmax activation layer of the classifier;
- $F_s(x)$ represents the class score distribution prior to the softmax activation layer.

4.4.2.1.3 Fast Gradient Sign Method (FGSM)

Fast Gradient Sign Method (FGSM) [i.13] is a single-step method to produce adversarial samples using the L_∞ norm. The method calculates adversarial samples basically by choosing for each input feature in which direction it should modify its input. FGSM can be used to generate untargeted as well as targeted attacks.

The calculation for untargeted attacks is as follows:

$$x' = x + \epsilon \cdot \text{sign}\left(\nabla J_{f,l}(x)\right) \tag{5}$$

Where:

- ϵ is a small real number to be chosen for perturbation, small enough to be undetected ([i.13] chose 0,25 for a shallow softmax classifier);
- $J_{f,l}(x)$ is the loss function of classifier f for true label l;
- l is the true label.

The calculation for targeted attacks is as follows:

$$x' = x - \epsilon \cdot \text{sign}\left(\nabla J_{f,l}(x)\right) \tag{6}$$

Where:

- ϵ is a small real number to be chosen for perturbation, small enough to be undetected ([i.13] chose 0,25 for a shallow softmax classifier);
- $J_{f,l}(x)$ is the loss function of classifier f for target label l;
- l is the target label.

4.4.2.1.4 Basic iterative method/projected gradient descent

The basic iterative method is an iterative version of the initial FGSM method. The basic iterative method extends it by starting with a valid image and adds in each step adding a parameter-based amount of noise to the pixel. It crafts the adversarial example starting with the original image and extends this step-wise as follows:

$$X_{N+1}^{adv} = Clip_{X,\epsilon}\{X_N^{adv} + \alpha \cdot \text{sign}\,\nabla_X J_{f,l}(X_N^{adv})\} \quad (7)$$

Where:

- X is the unperturbed image;
- l is the true label for X;
- $J_{f,l}(x)$ is the loss function of classifier f for true label l;
- $Clip_{X,\epsilon}\{X'\}$ performs per-pixel clipping of the image X', so the result will be in ϵ-neighbourhood (according to the L_∞) of the source image X;
- α determines the amount of changes applied to each pixel per step.

Kurakin et al. [i.14] selected the number of iterations heuristically by $\min(\epsilon + 4;\ 1{,}25\epsilon)$.

The basic iterative method is also known as Projected Gradient Descent (PGD) when it is used with noisy starting points [i.15]. Several improvements have been proposed for these iterative attacks. Dong et al. [i.16] proposed adding a momentum term to escape local maxima and achieve better convergence. Xie et al. [i.17] proposed applying random transformations every set number of iterations to further diversify the adversarial inputs and converge more quickly to an adversarial sample. Other similar improvements have also been proposed and claim to have a significant impact in the overall efficiency and quality of the produced adversarial samples in various use cases.

One issue with the basic iterative method and the projected gradient descent attack is that these attacks require careful and precise parameter-tuning to obtain the best results, in particular with regards to the amount of perturbation to be added in each successive step of the algorithm. While improvements have been suggested to alleviate this concern, Croce and Hein [i.18] proposed a parameter-free alternative called auto projected gradient descent.

Unlike the former iterative attacks, auto projected gradient descent optimizes the amount of perturbation to be added in a dynamic fashion at each step of the iteration as it gradually transitions from exploring the entire feasible region to a local optimization problem. Simultaneously, it adds a momentum term to control the convergence procedure and avoid local minima. Finally, when the dynamic step size is halved in a single step, which is to say when the algorithm converged to some local minimum, the algorithm restarts from the current best candidate solution in order to prevent premature convergence.

4.4.2.1.5 Jacobian-based Saliency Map Attack (JSMA)

Jacobian-based Saliency Map Attack (JSMA) is an attack based on the L_0 norm [i.19]. It is based on the forward derivative determined using the Jacobian matrix to identify the features and its values that has most impact on the classification result. Thus, JSMA enables the identification of decision boundaries to enable the efficient search for adversarial samples. JSMA generates targeted attacks based on a valid input example, using a maximum distortion parameter and a feature variation parameter.

To generate JSMA attacks, three steps are applied:

1) The forward derivative of a candidate adversarial sample needs to be computed. To do so, the Jacobian is computed layer-wise.

2) Use the forward derivative to construct a saliency map for the candidate adversarial sample. Saliency maps describe the impact of each input feature on the classification result.

3) Modify an input feature selected based on saliency map by the feature variation parameter. Repeat this step until the networks provides the targeted class as classification result or the maximum distortion parameter has been reached.

NOTE: This attack applies to feed-forward neural networks and can be applied to recurrent neural networks by unrolling a certain depth of layers.

Loison et al. [i.20] developed two improvements on JSMA: Weighted JSMA (WJSMA) and Taylor JSMA (TJSMA). WJSMA penalizes gradients related to small probabilities of labels to reduce their impact on saliency maps. Additionally, TJSMA penalizes those features components that are close to the maximum feature value through the use of Taylor terms.

4.4.2.1.6 Carlini/Wagner attack

The attack presented by Carlini et al. [i.21] was developed to show that a hardening mechanism for neural networks called defensive distillation is not effective. It generates adversarial examples by the following:

$$argmin \, \|x' - x\|_p + c \cdot f(x') \tag{8}$$

Where:

- x' is a perturbed image;
- x is a valid image;
- c is an arbitrary number with $c > 0$;
- $f(x')$ is an objective function with $f(x') \leq 0$ if the classification result is the desired target class $C(x') = t$.

The Carlini/Wagner attack [i.21] has been developed for the L_0, L_2, and L_∞ norm. It has no upper bound on the amount of perturbation, hence, it always succeeds in finding an adversarial example.

4.4.2.1.7 Deepfool

The Deepfool attack [i.22] crafts an adversarial attack by projecting an image on the decision boundary's nearest hyper-polyhedron face. The approach tries to find a solution to this problem iteratively by trying to push an input right behind the closest hyperplane of the decision boundary. The closest hyperplane is identified based on the probabilities of the classifier's output for each label. Then, Deepfool checks if the classification result differs from the original input. Due to its heuristic nature, there is no formal guarantee that the algorithm finds the adversarial image with the least perturbation.

The closest hyperplane is calculated by the following formula:

$$\hat{l}(x_0) = \text{argmin} \frac{\left|f_k(x_0) - f_{\hat{k}(x_0)}(x_0)\right|}{\left\|w_k - w_{\hat{k}(x_0)}\right\|_2} \tag{9}$$

Where:

- x_0 is a valid image;
- f_k refers to the output with the k-th highest probability and $\hat{k}$ refers to the output with the highest probability.

4.4.2.1.8 Shadow Attack

Ghiasi et al. [i.23] presented an attack which can be claimed to be a generalization of projected gradient descent, with a focus on the visual and defensive imperceptibility of the resulting adversarial samples. The main mechanism they utilize is, instead of solving a constrained optimization problem over the model loss exclusively as for projected gradient descent, they attach several additional penalty terms to the objective. Specifically, they solve the following optimization problem:

$$\max_{\delta} L(\theta, x+\delta) - \lambda_c C(\delta) - \lambda_{tv} TV(\delta) - \lambda_s Dissim(\delta) \quad (10)$$

Where:

- x is a valid image;
- δ is the added adversarial perturbation;
- θ represents the model parameters;
- $\lambda_c, \lambda_{tv}, \lambda_s$ are scalar penalty weights;
- $C(\delta)$ is a penalty constraining the change per channel;
- $TV(\delta)$ is a penalty constraining the total variation of the perturbation;
- $Dissim(\delta)$ is a penalty constraining the maximum variation per channel.

4.4.2.1.9 Fast Adaptive Boundary attack (FAB)

Croce and Hein [i.24] presented the Fast Adaptive Boundary attack (FAB). The attack functions in a similar way to Deepfool (see clause 4.4.2.1.7), where there is a built-in algorithmic incentive to quickly reach the decision boundary. However, unlike Deepfool, it also utilizes some additional criteria to emphasize closer proximity to the original image. In contrast to Deepfool, FAB includes the box constraint for images, whereas Deepfool performs the clipping after the solving, FAB uses a bias toward the original input and also performs backward steps, final search (a further iteration once an adversarial example is found, and random restarts, thus, further reducing the perturbation.

4.4.2.2 Gradient-free techniques

4.4.2.2.1 Spatial transformation

In contrast to the first approach to generate adversarial samples that work on pixel level to calculate some specific noise that leads to a misclassification, spatial transformation works on the whole image and performs some "optical" transformation by stretching parts of an image based on the logit outputs of the true label and the targeted label.

The formal definition of a spatial transformation is given by the following equations [i.25]:

$$x_{adv}^{(i)} = \sum_{q \in N(u^{(i)}, v^{(i)})} x^{(q)} \left(1 - \left|u^{(i)} - u^{(q)}\right|\right)\left(1 - \left|v^{(i)} - v^{(q)}\right|\right) \quad (11)$$

Where:

- $x^{(i)}$ is the value of the *i*-th pixel at a certain location;
- $\left(u^{(i)}, v^{(i)}\right)$ is the 2-dimensional location of the *i*-th pixel;
- $N\left(u^{(i)}, v^{(i)}\right)$ are the indices for 4-pixel neighbours (top-left, top-right, bottom-left, bottom-right) around the location $\left(u^{(i)}, v^{(i)}\right)$.

The function to confine the amount of perturbation is called the flow field. This flow field function is determined by minimizing the following objective function:

$$f = \underset{f}{\operatorname{argmin}}\left[L_{adv}(x, f) + \tau \cdot L_{flow}(f)\right] \quad (12)$$

Where:

- L_{adv} serves the purpose to ensure the targeted misclassification;
- L_{flow} minimizes the spatial transformation;
- τ is a factor to balance L_{adv} and L_{flow}.

$$L_{adv}(x, f) = \max\left(\max_{i \neq t} g(x_{adv})_i - g(x_{adv})_t, \kappa\right) \quad (13)$$

Where:

- $g(x)_i$ represents the *i*-th vector of the logit output of the model;
- κ is used to control the attack confidence level.

$$L_{flow}(f) = \sum_{p}^{all\ pixels} \sum_{q \in N(p)} \sqrt{\|\Delta u^{(p)} - \Delta u^{(q)}\|_2^2 + \|\Delta v^{(p)} - \Delta v^{(q)}\|_2^2} \quad (14)$$

Where:

- Δ is the amount of displacement of a pixel.

4.4.2.2.2 Fast feature fool

Fast feature fool is an approach to universal adversarial perturbations that works independent from specific training data and would result in a misclassification for most of the input data [i.26]. The approach employs a L_∞ norm to confine the amount of perturbation. The idea of the approach is to exploit the dependencies between the layers of a convolutional neural network. To do so, fast feature fool aims at over-saturating the activations, i.e. features of each convolutional layer to propagate misclassification from one layer to the next.

$$Loss = -\log\left(\prod_{i=1}^{K} \overline{l_i}(\delta)\right) \qquad \text{s.t. } \|\delta\|_\infty < \xi \quad (15)$$

Where:

- $\overline{l_i}(\delta)$ is the mean activation of the output at layer i;
- δ is the perturbation that is fed to the neural network;
- K is the number of layers of the model;
- ξ is the maximum allowed pixel intensity.

4.4.2.2.3 Generative universal adversarial perturbations

Generative universal adversarial perturbations employ a generative network that is trained against the target model [i.27]. It creates a fixed size perturbation δ independent from the input image of the target model. This δ is then scaled by a factor ω. The scaled perturbation is added to an image and then fed to the target network to compute its loss. The loss is used to update the weights of the generative network. Since the generative network is unaware of the used image, it is learning universal adversarial perturbations.

4.4.3 Security testing techniques requiring zero knowledge

4.4.3.1 ZOO: Zeroth Order Optimization-based attacks

The ZOO attack is a zero-knowledge approach that does not rely on training a surrogate model to obtain gradient information from it to apply adversarial sample generation techniques [i.28]. Instead, it estimates the gradient with the symmetric difference quotient and employs it to generate attacks using through zeroth-order stochastic coordinate descent.

ZOO further uses dimension reduction, hierarchical attack, and importance sampling techniques to make the attack generation more efficient. The quality of examples with respect to noise is comparable to Carlini/Wagner attack.

The gradient itself is estimated using symmetric difference quotient:

$$g_i = \frac{\partial f(x)}{\partial x_i} \approx \frac{f(x+h\cdot e_i)-(x-h\cdot e_i)}{2h} \quad (16)$$

Where:

- $f(x)$ is the function whose gradient is to be estimated;
- x_i is the point at which the *i*-th element of the vector is approximated;
- h is a very small number;
- e_i is the standard basis vectors with 1 at the *i*-th element.

The attacks are generated using stochastic coordinate descent which in each iteration randomly selects any input feature to optimize using the estimated gradient aiming at a better perturbation to generate an adversarial example. The algorithm can be used also with the Adam optimizer of with or without Newton's method.

4.4.3.2 Boundary attack

Brendel et al. [i.29] proposed a zero-knowledge attack which relies on the outputs only to approximate the decision boundary of the model and then query the model repeatedly as it tries to minimize the perturbation required to obtain a misclassification. It does so in several steps:

1) It first attempts to find a valid adversarial sample by adding a large perturbation to the input image in a random direction.
2) Utilizing the adversarial sample, it attempts to reduce the perturbation to approximate the distance to the decision boundary of the target model over a predefined number of steps.
3) The attack then involves performing a rejection sampling procedure alongside the boundary of the attack for a specified number of iterations, returning the best candidate as the solution of this optimization procedure.

4.4.3.3 Square attack

Adriushchenko et al. [i.30] proposed a zero-knowledge attack which involves performing a random search technique to add random perturbations to the image in an attempt to solve a constrained optimization problem over some custom defined loss. During each iterative step, the attack perturbs a square selection of pixels as defined by the L_p norm specified. Additionally, the attack utilizes early termination once it finds a valid adversarial sample. It is a technique with high query efficiency and can quickly converge to a valid adversarial sample. While it can be easy to detect and defend against Square Attack, doing so will increase model robustness against multiple types of attacks that replace regions of the input with repeated patterns.

4.4.3.4 SPSA-based attacks

To estimate the gradient, a technique called stochastic approximation using simultaneous perturbation can be applied. The approach has been employed by Uesato et al. [i.31] to generate adversarial attacks and is considered to be more reliable in finding adversarial samples. However, the optimization is more difficult than for optimization algorithms used by other attacks due to the high number of hyperparameters [i.32]. Furthermore, even though it provides reliably adversarial examples, they are often worse than those from other attacks [i.31].

4.4.3.5 Rotation and translation

The approach from [i.33] applies a rotation and a translation to an input image to obtain a misclassification. The approach optimizes over a set of parameters, i.e. the translation coordinates of the picture and the degree of rotation. Since the transformation is implemented as a differentiable function, the optimization can be based on the loss function. Due to the only three dimensions of the parameter space, grid search is a feasible and successful approach. Due to the quite small input space, worst-of-*k* is the third approach that randomly samples *k* attack parameters and then assesses on which of these *k* parameters the target model performs worst. The transformation is defined as follows:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} \cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} \quad (17)$$

Where:

- $\begin{bmatrix} u \\ v \end{bmatrix}$ is the location of the pixel to be transformed.
- θ is the degree of rotation performed to the image;
- δ is the amount of translation.

4.4.3.6 GenAttack

In contrast to other approaches, GenAttack [i.34] employs a genetic algorithm for generating adversarial attacks with zero knowledge about the internals of the ML component. Individuals for the population are bred by adding random noise along each dimension around an original valid example. The fitness function considers the attacked model's increasing confidence in the target label and decreasing confidence in the other labels as a fitness function. Crossover is implemented by exchanging features between two individuals using a fitness-based probability function. In contrast to the ZOO attack, GenAttack can reduce the number of queries.

4.4.3.7 Universal adversarial perturbations

Moosavi et al. [i.35] employ the Deepfool approach to generate universal adversarial perturbations using a local search that aims at moving a datapoint to the decision boundary of a classification region. Universal adversarial perturbations lead to a misclassification on almost any input. The algorithm presented by Moosavi et al. [i.35] iterates over a dataset containing all the images, computes the minimal perturbation that would move the point to the decision boundary of the currently considered classification regions, and updates the universal perturbation with the newly identified minimal perturbation. The computation is performed using the Deepfool approach described in clause 4.4.2.1.6. A graphical representation of the approach for generating universal adversarial perturbations can be found as Figure 2 of [i.35].

5 Test adequacy criteria

5.1 Introduction

Clause 5 provides information on several metrics that might be useful when security testing ML components. Such metrics are test coverage criteria discussed in clause 5.2.1 and stop criteria discussed in clause 5.2.2. In contrast to traditional security testing for which coverage metrics are used to measure test progress and are used as a stop criterion, e.g. code coverage, testing of ML models provides explicit stop criteria, often in terms of adversarial robustness metrics that enable the direct measurement on whether the ML component is getting more secure. The metrics and coverage criteria in clause 5 are considered as not sufficient to increase the robustness of neural networks [i.36]. Hence, such metrics should not be used solely.

NOTE: Coverage metrics in traditional software testing are used to measure the intensity as piece of software has been tested [i.37].

5.2 Test coverage criteria

5.2.1 Common concepts and notations

5.2.1.1 Introduction

Most of the subsequent clauses refer to some common formulas to simplify the coverage metrics and increase their comprehensiveness. These common formulas are introduced in the subsequent clauses.

5.2.1.2 Activation value

The activation value of a neuron n of a neural network that is stimulated with an input x has the following notation:

$$\phi(n, x) \tag{18}$$

Where: n is referring to the considered neuron from the neural network;

x is the input to the neural network.

5.2.1.3 Activation trace

Kim et al. [i.38] introduced the notion of an activation trace that captures the activation values of the neurons of a neural network when stimulated with a certain input, and the set of all activation values as a matrix that can be obtained when stimulating the neural network with a set of inputs. In that sense, it is a generalization of the activation values with respect to the considered neurons and inputs. The activation trace of a set of neurons is defined as follows:

$$\alpha_N(x) = \begin{pmatrix} \phi(n_1, x) \\ \dots \\ \phi(n_n, x) \end{pmatrix} \tag{19}$$

Where:

- N refers to an ordered subset of neurons from a neural network;
- x is an input t to the neural network.

The cardinality of $\alpha_N(x)$ is equal to $|N|$.

The set of activation values obtained from a set of inputs is defined as follows:

$$A_N(X) = \{\alpha_N | x \in X\} \tag{20}$$

Where:

- N refers to an ordered subset of neurons from a neural network;
- x is an input t to the neural network.

5.2.1.4 Major function region

The basis of k-multisection neuron coverage is the so-called major function region derived from the training set. The major function region is a closed interval $[low_n, high_n]$ where the $low_n$and $high_n$ are the minimal and maximal activation values $\phi(n, x)$ of a neuron n obtained from the training set. K-multisection neuron coverage is discussed in clause 5.2.2.3.

5.2.2 Neuron-level coverage metrics

5.2.2.1 Overview

Neuron-level coverage metrics are measuring activation values and are structuring them in areas observed during training that are then compared to what is observed during testing. Figure 1 shows the different areas that are observed by the different metrics. The vertical bar represents the range of activation values of a neuron observed when the training set is fed to the trained model, its lower bound is denoted by $\phi_{min,train}$, its upper bound by $\phi_{max,train}$. The part of this range above a certain threshold comprised by the bracket referred to by A is subject to neuron coverage. The range of B is divided into k equal section and is measured by k-multisection coverage. k-multisection coverage has similarities to equivalence partitioning for testing traditional software but the sections have no semantics in k-multisection coverage. Neuron boundary coverage measures how many neurons are activated below $\phi_{min,train}$ and above $\phi_{max,train}$ by the training set, denoted by C and D. In that sense, it is similar to boundary value analysis for testing traditional software. Strong neuron coverage is identical to neuron boundary coverage but focusses only on the values above $\phi_{max,train}$, denoted by C.

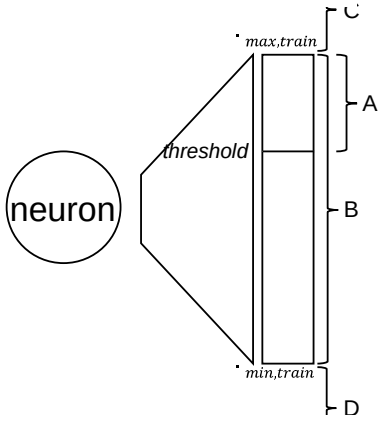

Figure 1: Illustration of neuron-level coverage metrics for a single neuron

5.2.2.2 Neuron coverage

Pei et al. introduced the metric of neuron coverage in [i.7] for a neural network with Rectified Linear Units (ReLUs) as activation function. Given a test input set, neuron coverage is measured by the number of neurons whose output value is higher than a certain threshold for all elements from the test input set. The formal definition given in [i.7] is as follows:

$$NCov(T,t) = \frac{\left|\{n \mid \forall x \in T, \phi(n,x) > t\}\right|}{|N|} \quad (21)$$

Where:

- T is the set of test inputs;
- x is an element from the set of test inputs;
- N is the set of all neurons from the network;
- n refers to a neuron from the network;
- t is a threshold that determines whether a neuron is activated, e.g. 0;
- $\phi(n,x)$ denotes the activation values of the neuron n if the neural network receives test input x.

NOTE 1: The parameter t, i.e. the threshold, determines how hard it is to achieve the coverage criterion.

NOTE 2: This metric was applied to neural networks with ReLUs as activation function.

5.2.2.3 k-multisection neuron coverage

The *k*-multisection neuron coverage metrics measures for each neuron how well its output space is covered. It does this by dividing the interval between the minimum and maximum output value of each neuron, observed during training, into *k* equal sections and counting the ratio of sections that are covered by at least one value of the test set [i.39].

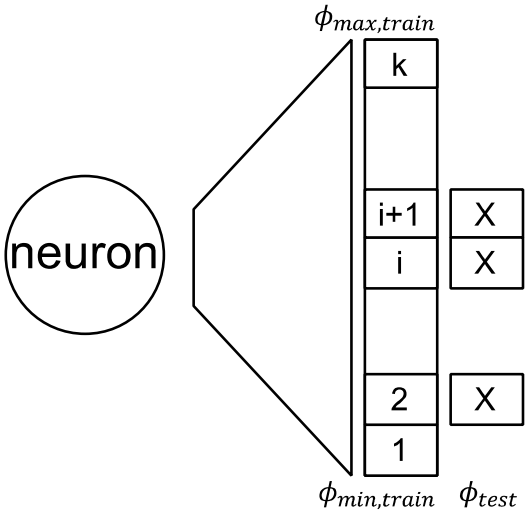

Figure 2: Illustration of k-multisection neuron coverage for a single neuron

The basis of k-multisection neuron coverage is the so-called major function region derived from the training set. The major function region is a closed interval $[low_n, high_n]$ where the low_n and $high_n$ are the minimal and maximal activation values $\phi(n, x)$ of a neuron n obtained from the training set. To measure the degree to which a major function region, i.e. the interval $[low_n, high_n]$, is covered, it is partitioned into k sections. If for each of these k partitions at least one activation value is observed, this neuron is considered as covered. The major function region of a neuron is completely covered if all k partitions of its major function region are covered by the test input set. A formal definition from [i.39] is as follows:

$$KMNCov(T, k) = \frac{\sum_{n \in N} \left| \{ S_i^n | \exists x \in T : \phi(n, x) \in S_i^n \} \right|}{k \cdot |N|} \quad (22)$$

Where:

- T is the set of test inputs;
- x is an element from the set of test inputs;
- N is the set of all neurons from the network;
- n refers to a neuron from the network;
- S_i^n refers to the set of values of the i-th partition of the major function region of neuron n;
- $\phi(n, x)$ denotes the activation values of the neuron n if the neural network receives test input x.

NOTE 1: The parameter k, i.e. the number of partitions derived from a major function region, determines how hard it is to achieve the coverage criterion.

NOTE 2: This metric applies to feed-forward neural networks and can be applied to recurrent neural networks by unrolling a certain depth of layers.

5.2.2.4 Neuron boundary coverage

Neuron boundary coverage measures for each neuron if a value below and above the minimal and maximal output value observed during training, is observed during testing [i.39]. It is based on corner cases derived from the major function region of each neuron as defined in clause 5.2.2.3:

$$NBCov(T) = \frac{|LowerCornerNeuron| + |UpperCornerNeuron|}{2 \cdot |N|} \quad (23)$$

Where:

- $LowerCornerNeuron = \{n \in N | \exists x \in T : \phi(n, x) \in (-\infty, low_n)\}$;
- $UpperCornerNeuron = \{n \in N | \exists x \in T : \phi(n, x) \in (high_n, +\infty)\}$;
- N is the set of all neurons from the network;
- n refers to a neuron from the network;
- T is the set of test inputs;
- x is an element from the set of test inputs;
- low_n is the lower border of the major function region of neuron n;
- $high_n$ is the upper border of the major function region of neuron n.

NOTE: This metric applies to feed-forward neural networks and can be applied to recurrent neural networks by unrolling a certain depth of layers.

5.2.2.5 Strong neuron activation coverage

Strong neuron activation coverage measures for each neuron how many corner cases above the maximal output value observed during training are observed during testing [i.39]. It can be considered as a specialization of neuron boundary coverage. A formal definition from [i.39] is as follows:

$$SNACov(T) = \frac{|UpperCornerNeuron|}{|N|} \quad (24)$$

Where:

- $UpperCornerNeuron = \{n \in N | \exists x \in T : \phi(n, x) \in (high_n, +\infty)\}$;
- N is the set of all neurons from the network;
- n n refers to a neuron from the network;
- T is the set of test inputs;
- x is an element from the set of test inputs;
- $high_n$ is the upper border of the major function region of neuron n.

NOTE: This metric applies to feed-forward neural networks and can be applied to recurrent neural networks by unrolling a certain depth of layers.

5.2.3 Layer coverage metrics

5.2.3.1 Top-*k* neuron coverage

Top-k active neurons is defined based on the activation values of the neurons of a single layer. Top-k neuron coverage measures how many different neurons were the most k active neurons during testing [i.39]. The higher the number of k neurons during testing is considered better. A formal definition is as follows:

$$TKNCov(T,k) = \frac{\cup_{x \in T}(\cup_{1 \leq i \leq l} top_k(x,i))}{|N|} \quad (25)$$

Where:

- T is the set of test inputs;
- x is an element from the set of test inputs;
- N is the set of all neurons from the network;
- l is the number of layers of the neural network;
- $top_k(x,i)$ is the set of k neurons in layer i of the neural network with k highest activation values $\phi(n,x)$ given input x.

NOTE 1: The parameter k determines how many active neurons are considered for this metric. If k is set to 1, this metric is similar to neuron coverage without a threshold value (see clause 5.2.2.2).

NOTE 2: This metric applies to feed-forward neural networks and can be applied to recurrent neural networks by unrolling a certain depth of layers.

5.2.3.2 Top-*k* neuron patterns

Top-k neuron patterns is similar to top-k neuron coverage by considering the top-k active neurons as k-tuples instead of single neurons [i.39]. A formal definition is as follows:

$$TKNPat(T,k) = |\{top_k(x,1), \ldots, top_k(x,l) | x \in T\}| \quad (26)$$

Where:

- T is the set of test inputs;
- x is an element from the set of test inputs;
- l is the number of layers of the neural network;
- $top_k(x,i)$ is the tuple of k neurons in layer i of the neural network with highest activation values $\phi(n,x)$ given input x.

NOTE: This metric applies to feed-forward neural networks and can be applied to recurrent neural networks by unrolling a certain depth of layers.

5.2.4 Surprise Adequacy

5.2.4.1 Basic idea

Surprise metrics are based on the idea that a test set should be somehow new to a neural network with respect to the training data set [i.38]. Surprise adequacy is measured using similarity of new test inputs to existing training inputs in terms of activation values expressed as activation traces (as specified in clause 5.2.1.3).

5.2.4.2 Likelihood-based Surprise Adequacy

Likelihood-based Surprise Adequacy (LSA) [i.38] uses kernel density estimation to approximate the probability density of the activation values in an activation trace and defines surprise adequacy based on a density function $\hat{f}$ such that the metric increases when the density function decreases:

$$LSA(x) = -\log\left(\hat{f}(x)\right) \tag{27}$$

Where:

- x is the test input;
- $\hat{f}$ is the density function.

NOTE: This metric is computationally expensive. Hence, the authors reduce the computational effort by considering only neurons from a selected layer and ignore neurons whose activation value variance is below a certain threshold. The computational costs can further be reduced by computing LSA per class under the assumption that the activate traces are similar if inputs are from the same class.

The authors use a Gaussian kernel function to produce the following density function:

$$\hat{f}(x) = \frac{1}{|A_N(T)|} \cdot \sum_{x_i \in T} K_H\left(\alpha_N(x) - \alpha_N(x_i)\right) \tag{28}$$

where:

- x is the test input;
- T is the set of training inputs;
- x_i is an input from the training set T;
- N is the number of considered neurons, e.g. reduced to a single layer of a neural network;
- $A_N(T)$ are the activation traces of the training set of the considered neurons N;
- K_H is the kernel function, e.g. Gaussian kernel function with a bandwidth matrix H.

5.2.4.3 Distance-based Surprise Adequacy

Distance-based Surprise Adequacy (DSA) [i.38] is a computationally cheaper version of LSA that uses Euclidean distance of activation traces instead of a density function to compare new inputs with activation traces from the training phase. It uses the distance between a new input and the closest activation trace obtained from the training set. A formal definition of DSA is as follows:

$$DSA(x) = \frac{dist_a}{dist_b} \tag{29}$$

Where:

- $dist_a = \|\alpha_N(x_a) - \alpha_N(x_i)\|$;
- $x_a = \underset{D(x_i)=c_x}{\operatorname{argmin}} \|\alpha_N(x_a) - \alpha_N(x_i)\|$;
- $D(x_i)$ denotes the classification result of a neural network when stimulated with input x_i;
- c_x is the predicted class of input x.

5.3 Stop criteria

5.3.1 Relationship of stop criteria to metrics for neural networks

Stop criteria for testing are usually defined on the basis of metrics as those provided by clause 5.2. However, as discussed by [i.36], relying on such metrics is not sufficient when using security testing with these metrics together with adversarial training to achieve networks that are more robust with respect to adversarial samples than the neural network before testing and adversarial training. Hence, the following metrics provide a means to identify a neural network's robustness and thus, can serve as stop criterion in conjunction with a goal to be specified before testing, e.g. as a robustness requirement.

5.3.2 Adversarial robustness

5.3.2.1 Global Lipschitz constant

5.3.2.1.1 Introduction

The basic idea of the Lipschitz constant is that it confines the maximum slope of a continuous function and thus, implies that small changes to the input only lead to small changes in the output of function. The Lipschitz constant measures the factor between the output difference and the input difference and can be illustrated as follows:

$$\|f(x) - f(x)\|_p \leq \lambda \cdot \|x - \hat{x}\|_q \tag{30}$$

Where:

- x is an input value;
- $\hat{x}$ is another input value;
- $f(x)$ is the function value of x;
- $f(\hat{x})$ is the function value of $\hat{x}$;
- $\|\cdot\|_p$ is an L_p-norm;
- λ is the value whose smallest value is the Lipschitz constant.

The global Lipschitz constant aggregates all the local Lipschitz constants of a neural network layer-wise. Thus, the global Lipschitz constant measures the sensitivity to adversarial samples [i.36]. The assumption for this is that the loss for an adversarial sample is to be maximized whilst the ϵ, i.e. the distance of an adversarial sample to a "normal" sample is minimized. This leads to a situation where a classification error results from, e.g. white noise added to a normal sample. Generally, it can be said the smaller the global Lipschitz constant of a network is, the more robust it could be considered.

5.3.2.1.2 Global Lipschitz constant calculation for neural network architectures

Cisse et al. [i.41] and Dong et al. [i.36] provide rules for calculating the global Lipschitz constant for several neural network architectures. The calculation of the Lipschitz constant for a fully connected layer according to [i.36] is as follows:

$$\lambda = \max_k \Sigma_{i=1}^{s_j} \left|\omega_{i,k}^{j}\right| \tag{31}$$

Where:

- $\omega_{i,k}^{j}$ is the weight between the k-th neuron in layer L_j and the i-th neuron in layer L_{j-1};
- s_j is the number of neurons in Layer L_j.

Cisse et al. [i.41] provide also calculation rules for convolutional layers and aggregation/transfer layers.

NOTE 1: The Lipschitz constant is in general independent of the number of layers of a neural network and depends mainly on the weights of a network.

NOTE 2: The calculation of the global Lipschitz constant might be computationally expensive or infeasible.

5.3.2.1.3 Global Lipschitz constant calculation using Extreme Value Theory

This attack-independent metric measures the adversarial robustness of a neural network against any attack that identifies the minimal distortion given an L_p norm [i.40]. CLEVER stands for **C**ross **L**ipschitz **E**xtreme **V**alue for N**e**twork **R**obustness. In contrast to the global Lipschitz constant that focusses on the relationship between input values and the loss of the classification result, the CLEVER score focusses on the minimal distance between an input and a perturbed input that leads to different classification result for a given input:

$$x_a = x_0 + \delta \tag{32}$$

where:

- x_0 is a valid sample;
- δ is the noise for perturbation with l_p distortion $\Delta_p = \|\delta\|_p$.

The CLEVER score describes the lower boundary of ϵ, i.e. $\beta_L \leq \epsilon$, such that the classification results will be the same, i.e. $D(x) = D(\hat{x})$.

5.3.2.2 Local adversarial robustness

Local adversarial robustness is a generalization of the CLEVER score measures the robustness against perturbation attacks. It defines for a single input the perturbation threshold for which the classification result remains the same for any derived input below that threshold, using a certain L_p-norm. A formal definition is originally given in [i.42] and restated by [i.36]:

$$\|x - x'\|_\infty \leq \delta \tag{33}$$

Where:

- x is a single valid input;
- x' is any other input that satisfies the above formula;
- δ is that maximum distance around x;

that satisfies:

$$\phi(x) = \phi(x') \tag{34}$$

Where:

- $\phi(x)$ denotes the classification result of the neural network for input x;
- $\phi(x')$ denotes the classification result of the neural network for input x'.

NOTE: Katz et al. [i.41] denote this as δ-locally-robust.

5.3.2.3 Global adversarial robustness

Global adversarial robustness is a generalization of local adversarial robustness with respect to the given input. In contrast to local adversarial robustness that considers a single give input that needs to be robust to perturbations below a static threshold, global adversarial robustness extends this statement to any input whose classification result needs to be robust against any perturbation below a certain threshold. A formal definition is given by [i.36]:

$$\|x_1 - x_2\|_\infty \leq \delta \tag{35}$$

Where:

- x_1 is any valid input;
- x_2 is any adversarial perturbation around x_1 such that the formula above holds;
- δ is that maximum distance around x_1;

that satisfies:

$$|p_1 - p_2| \leq \epsilon \quad (36)$$

Where:

- p_1 is the output of the neural network for label a for x_1;
- p_2 is the output of the neural network for label a for x_2;

that holds for any input values x_1 and x_2 that satisfy formula (1).

NOTE 1: Katz et al. [i.42] denote this as ϵ-δ-globally-robust.

NOTE 2: This theoretical concept can be hard to implement and computationally infeasible for large networks and high-dimensional input spaces.

6 Security test oracles

6.1 Introduction

A test oracle is a mechanism for determining whether a test case has passed and involves comparing the system's output to the expected output. In doing so, bugs or faults in the system can be detected.

6.2 Statistical and probabilistic test oracles

A probabilistic test oracle does not provide an exact verdict whether a test case has failed or passed but provides a probability in terms of a real number by using statistical tests and providing an estimation of the error that the test verdict given by the oracle is correct [i.43]. Statistical test oracles are useful if the statistical properties of the data are known, e.g. the distribution of data.

6.3 Pseudo test oracles

6.3.1 Not a Number

NaN stands for 'Not a Number' and is provided by some programming languages if an input or calculation is not a number, e.g. a string, or as a result to some error in the calculation, precision, or length of the resulting value. Odena et al. [i.4] used this as an oracle to find numerical errors that result from the fact the neural networks are implemented using floating-point math.

6.3.2 Differential testing

Differential testing [i.6] or differential fuzzing is employing a pseudo-oracle through another system that provides the same functionality (as presented in clause 4.3). Each test case is executed against the system under test as well as the reference system. The reference system serves as a pseudo-oracle. The results of both executions are then compared. If they are not identical, it is assumed that the system under test is misbehaving.

NOTE: No system is error-free and hence, also existing but yet undiscovered errors in the reference system can lead to false positives and false negatives when calculating the test verdict.

6.3.3 Metamorphic relations

Metamorphic testing [i.5] (discussed in clause 4.2.2) is based on the idea that if a test oracle is not available, comparing inputs and outputs of the test item and identifying the relationships between them in terms of, e.g. qualities, inequalities, periodicity properties can be used to specify a test oracle. These relationships specify how a modification to an input result in which changes to the corresponding outputs and are called metamorphic relations. Thus, metamorphic relations can be used in place of test oracles and are helpful if a test oracle is not available. The challenge is the identification of expressive and representative metamorphic relations.

Annex A: Bibliography

- ETSI TR 101 583: "Methods for Testing and Specification (MTS); Security Testing; Basic Terminology".

History

Document history		
V1.1.1	July 2024	Publication