Parsimonious Neural Networks

Mathieu Causse[1], Cameron James[12], Mohamed Masmoudi[13] and Houcine Turki[14]

[1] ADAGOS, France
[2] University of Sheffield, UK
[3] University of Toulouse, France
[4] University of Nice, France
contact@adagos.com

Abstract. Redundancy is the backbone of state-of-the-art deep neural networks. It ensures that the learning process avoids being trapped in local minima. According to [5] "In practice, poor local minima are rarely a problem with large networks."

We are introducing a new approach of deep learning based on parsimony. The number of links to be identified by the learning process is reduced. But we enrich the structure of the neural network: any neural cell can be connected to any preceding cell. We are far from the classical layered architecture.

Parsimonious approaches outperform state-of-the-art method, particularly when the response of the model is continuous, like dynamic prediction.

This approach solves most of the drawbacks of existing neural network algorithms: carbon footprint is drastically reduced, the prediction capability is improved, the prediction is robust and resists to DeepFool [7] attacks.

Local Interpretable Model-Agnostic Explanations (LIME) [8] is a major contribution that improves understanding of why machine learning models behave the way they do. We will show that parsimonious models offer a better interpretation.

Moreover, the so created neural networks can be embedded easily and achieve real time responses using reduced computing resources and energy consumption.

Keywords: Artificial intelligence, Neural networks, Deep learning, DeepFool, LIME.

1 Introduction

The state of the art of artificial intelligence is largely inspired by the biological brain, including its redundant nature. While this redundancy may ensure the continued functionality of the living brain despite regular, and sometimes accidental, loss of the neural cells, the same argument does not hold for artificial neural networks, which are made from inert matter.

We developed NeurEco, a new parsimonious neural networks approach that reduces the resources required for implementing machine learning algorithms by orders of magnitude. This approach allows embedding complex artificial intelligence algorithms; an ambition that requires hunting down and eliminating every possible source of wastage.

Moreover, parsimonious neural networks are able to capture with great accuracy the physical or biological phenomena conveyed by the data. They even allow creation of highly reliable complex dynamical models, including for quasi-chaotic phenomena; a case in which even the slightest flaw of parsimony would have irremediable consequences on the quality of the model.

In section 2, we give a general presentation of our approach.

Section 3 will be dedicated to robustness of neural networks responses. Indeed, unlike state-of-the-art neural networks, NeurEco resists against DeepFool attacks [7].

To reduce the black box character of neural networks, Local Interpretable Model-Agnostic Explanations (LIME) [8] can be applied to a trained model to explain and visualize the prediction. We will show in section 4 that NeurEco offers reliable interpretation in comparison with state-of-the-art models.

We reported the size of each neural network for each test case. In a recent paper [9], authors point out the important carbon footprint of state-of-the-art algorithms. In particular, authors recommend to take into account what they call the R&D process leading to the final model. In our case, the process leading to the parsimonious model is straightforward.

2 Parsimonious Modelling

We provided an answer to these issues by considering the topological gradient approach. One of the authors played a pioneering role in the emergence of this theory [1,4,6]. We replaced the rigid layered structure by a computing graph that may connect any cell to any other cell.

We build the neural network, or the computing graph, step by step. At each step, the topological gradient gives the variation of the global error by adding/removing a cell or a link, using few computer resources. In other words, each graph modification (adding/removing graph elements) has a score estimation provided by the topological gradient. Higher score modifications are applied to the computing graph.

A deterministic initialization of the weights has a probability of 0.5 of doing better than the random choice. But it is possible to increase the chance to perform better if we find an appropriate heuristic.

A natural idea is to set the initial weights to zero. In this case all gradient components with respect to the weights are equal to zero. It doesn't help to improve the initial guess. However, it is possible, starting from zero, to use higher order derivatives to determine a first descent direction.

2.1 A new architecture of neural networks

The learning process consists of building the model:

$$Y = f_{\Gamma,W}(X), \qquad (1)$$

where the model $f_{\Gamma,W}$ is defined by a computing graph Γ, and W represents the weight of each graph link. These model characteristics are determined by the following minimization

$$\min_{\Gamma,W} \sum_{i=1}^{N} \left|\left|Y(X_i) - Y_i\right|\right|^2, \qquad (2)$$

where $(X_i, Y_i)_{i=1}^N$ is the learning data and $Y(X_i)$ is given by (1) for $X = X_i$.

Solving problem (2) in W, can be seen as the resolution of a large nonlinear system in the least square mean. If the output Y is a scalar, the number of equations is equal to N, and the number of unknowns (number of components of W), must be very small in comparison with N. This explains why the set of learning data can be reduced in the parsimonious context.

We denote by $X^i, i = 1, \ldots, nl$ the layer formed by the n_i neurons that can be calculated simultaneously during the stage *i*, and the layers calculated up to stage i are $\mathbf{X^i} = (X^0, \ldots, X^i)$. We set $\mathrm{X}^0 = (X_i)_{i=1}^{M1}$, with size $\mathrm{n_0} \times \mathrm{M1}$, to represent the state of the input layer and $Y = (Y_i)_{i=1}^{M1}$ as the target values corresponding to the input X^0. We denote by n_y, the dimension of Y_i.

For each layer i, a sparse matrix of connection weights $\mathrm{W_i}$ is identified, with size $n_{i+1} \times \sum_{j \le i} n_j$. All the connection weights of the entire neural network are then assembled as $W = (W_0, \ldots, W_{nl-1})$, by abuse of language, this object will be called a matrix.

The neural network then implements the following calculations on the X^0 input data:

$\mathbf{X}^0 = X^0$;
for i in [1,…, nl]
  $X^i = f_{SI}(W_{i-1} * \mathbf{X}^{i-1})$; (3)
  $\mathbf{X}^{\mathrm{i}} = (\mathbf{X}^{i-1}, X^i)$;
end;

where the function f_{SI} is the activation function. Typically, it can be:

$$f_{SI}(x) = \frac{1}{1 + exp(-x)} \ . \qquad (4)$$

We suppose that, for example, the last line of X^0 is formed by 1s; this would mean that the last cell in the entry layer is a bias neuron. In classical architectures, each layer, other than the output layer has a bias neuron. In the architecture of this method, only the entry layer has a bias neuron; neurons in the other layers can connect directly to this neuron.

The error function *J* of the neural network is then written:

$$J = \left|\left|\mathrm{O}\, \mathbf{X}^{nl} - Y\right|\right|^2 , \qquad (5)$$

where O is the observation matrix for extracting the output elements from $\mathbf{X}^{\mathrm{nl}}$.

The topology Γ of the neural network is defined by non-zero coefficients of matrices $\mathrm{W_i}$.

2.2 The learning process

In this section we are presenting a topological gradient approach. We are working on a more efficient version, based on calculus of variations.

The first step in the process of constructing the neural network is the definition of an initial topology, noted Γ^1:

- An input layer, whose number of nodes is imposed by the number n_0 of input data including bias.
- An output layer of n_y nodes,
- A hidden layer containing a small number of neurons.

Step 1 of Fig. 1, shows an example of initial network topology.

The initialization stage also includes an optimization of connection weights W^{1*} by minimizing the error function J for the initial fixed topology Γ^1. This optimization is achieved by training the neural network on the learning data. Gradient backpropagation could be used for this aim.

The process then includes topological optimization phases:

- Additive phases, in which one neuron cell and/or one link is added to the neural network, the added connection being such that it connects the input of a neuron to the output of a neuron from any previous layer,
- Subtractive phases, in which neuron cells and/or links are removed from the neural network.

In addition, each topological change, additive or subtractive, is selected from a large set of candidate changes. For each topology change, we estimate the variation of the network error after the learning phase. The best topology change is applied to the current neural network.

The purpose of the additive phase is to ensure that the network is able to learn the data. The subtractive phase, which can only increase the error, is intended to ensure that the model is capable of generalizing to unseen data.

During a subtractive phase, all nodes and links are candidates for a topological modification. In an additive phase, two nodes that do not belong to the same layer and are not already connected can be connected by a new link; alternatively, new nodes can be added to any layer, except the network input and exit layers. A new layer can also be created by inserting a node between two successive layers. Any newly created node must be connected to the network with at least two links: one input link and one output link.

In an additive phase, if the network is large, a variety of candidate topological changes can be chosen at random. For each of these candidate topological changes, estimates of J error variation are calculated. The 'best' topological change depends on the type of phase:

- For a subtractive phase, it is the change that achieves the smallest estimated increase in J error.
- For an additive phase, it is the one that offers the largest estimated drop in J error.

The network error variation between a modified topology (candidate for iteration n) and the previous topology (iteration n-1) is given by the optimal connection weights for each topology considered:

$$\mathrm{J}(\Gamma^n, \mathrm{W}^{n*}) - \mathrm{J}(\Gamma^{n-1}, \mathrm{W}^{n-1*}), \qquad (6)$$

where Γ^n is the modified topology for iteration n, and W^{n*} is the matrix of optimal connection weights for this topology. However, the calculation of a matrix of optimal connection weights for any given topology is very long, and therefore it is not easy to

calculate the error variation for every candidate topological change; instead, the following method is used.

For an additive phase, the modified connection weights W^n are initialized by:

- $W^n = W^{n-1*}_{|\gamma}$ for any links of Γ^n that are included in Γ^{n-1}
- The other links of Γ^n are initialized to 0.

Since $J(\Gamma^n, W^n) = J(\Gamma^{n-1}, W^{n-1*})$, this initialization does not change the error. Subsequently, one or two learning iterations are performed to improve W^n and the variation of the error is estimated by:

$$J(\Gamma^n, W^n) - J(\Gamma^{n-1}, W^{n-1*}), \qquad (7)$$

which is necessarily negative.

In the case of a subtractive phase, the connection weights of W^n of the modified topology are initialized by $W^n = W^{n-1*}_{|\Gamma^n}$, then one or two learning iterations can be proceeded with to improve W^n ; the estimate of the error variation is given by (7), as before. This time, the variation is necessarily positive, otherwise W^{n-1*} is not optimal; indeed, the matrix W^n would offer a better solution by setting the removed links to zero. However, as the J error function increases, the average error on validation data tends to decrease; thus the goal of subtractive phase is to improve generalization.

Unlike the stochastic gradient method, these methods stop as soon as the gradient ∇J becomes small, even if the J error is not small. The additive phase adds new degrees of freedom that, by construction, reduce the error. The descent algorithm can then be restarted.

Fig. 1 shows the creation of a neural network in six steps. The unique bias cell of the network is not represented. It makes the images unreadable.

This test case is proposed by a telecommunication operator. The goal is to determine the location of mobile phones using the intensity of signals received from at most 8 surrounding antennas. The location by GPS is used only in the learning phase. The final goal is to determine the coverage of the telecommunication network without using mobile phones GPS.

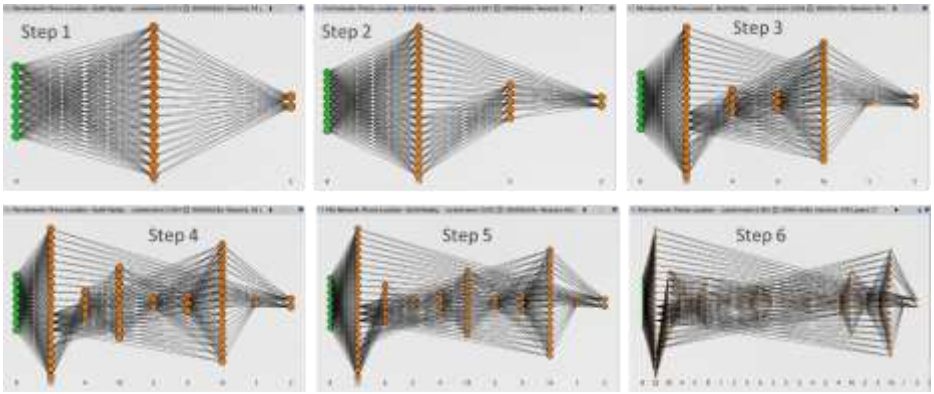
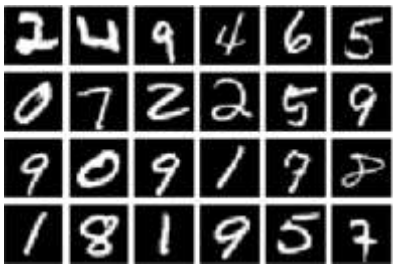


Fig 1: Evolution of the parsimonious network structure during the learning process.

3 DeepFool attacks

We consider DeepFool attacks in the frame of the MNIST test case. We start by comparing the classification results, then we compare the resistance to DeepFool attacks of our algorithm with the state of the art.

3.1 Classification Results



We tested NeurEco on the famous MNIST dataset (http://yann.lecun.com/exdb/mnist/).

The MNIST data contains 70,000 images (60,000 for learning and 10,000 for testing) of handwritten numbers and for each, its corresponding label.

For this test case, three methods of classification were compared:

- NODE, from the article « neural ordinary differential equations » [2],
- A classic network, made with TensorFlow (https://www.tensorflow.org),
- The NeurEco approach

For each of these methods, a convolutional neuron network (CNN) was used upstream of the classifier to reduce the number of entries. The results obtained by the three methods are presented in Table 1.

Table 1. Percentage of successful classifications on the test data

	Percentage of Success	Size of CNN	Size of the classifier	Total Size
NODE	99.56	131 830	76 170	208 000
TensorFlow	98.95	7 680	164 224	171 904
NeurEco	99.80	4 608	9 947	14 555

Thus, on the first test and without any hyper-parameter tuning, the NeurEco approach obtained an error rate of 0.20%; one of the best results ever reported for the MNIST dataset.

3.2 Comparison of Robustness

The idea is to find the smallest disturbance of an input image that will modify the class given by the network. This test is inspired by [7], which proposes an algorithm to calculate disturbances that deceive deep networks and thereby quantify their robustness.

This approach is very similar to the methods used in reliability calculation; we can cite [3] for the oldest methods and [11] for more recent learning-based methods. The basic idea is to put the system in a normal state of operation and search for the nearest point of failure for this mode of operation.

The images of Fig. 2 show the smallest disturbances that affect the classification results for the neural networks created by the NeurEco approach and by TensorFlow. This comparison was crucial in making us aware of the role that parsimony can play in the robustness of neural network responses.

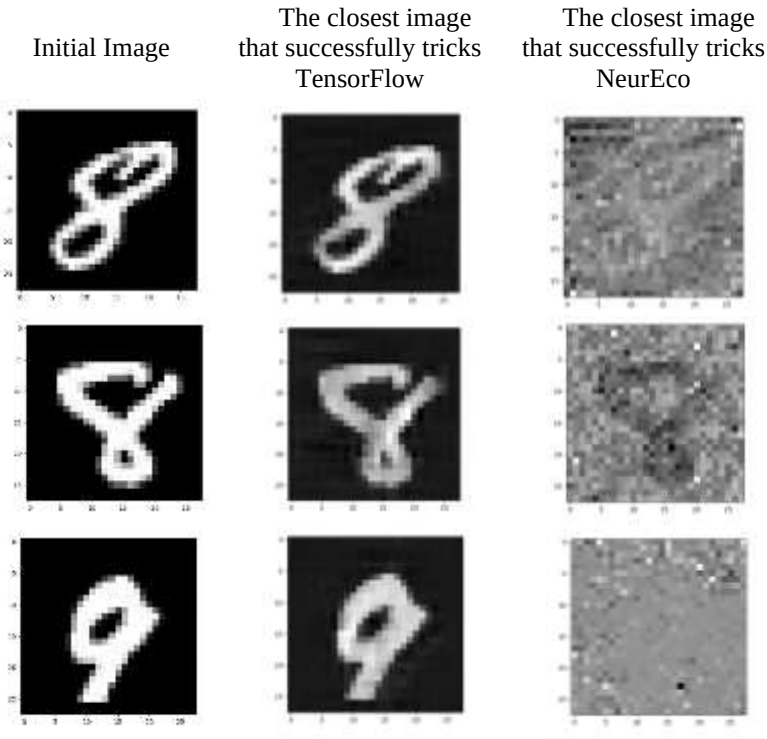


Fig. 2. The perturbed images represent the 'closest' image that successfully tricks the model into giving the incorrect output. For the perturbed images, the models predict 5, 9 & 0 respectively (top to bottom).

We also evaluated each architecture to compare their performance when a random Gaussian noise is added to the images. Fig. 3 shows how the classification accuracy of each network was affected by gaussian noise with standard deviations ranging from 0 (where it can be seen that the results match Table 1) to 1 (which is half the amplitude of the image). At very low standard deviations, both networks perform well, but as the standard deviation increases, NeurEco displays superior robustness against the noise and its classification accuracy decays much less rapidly than TensorFlow.

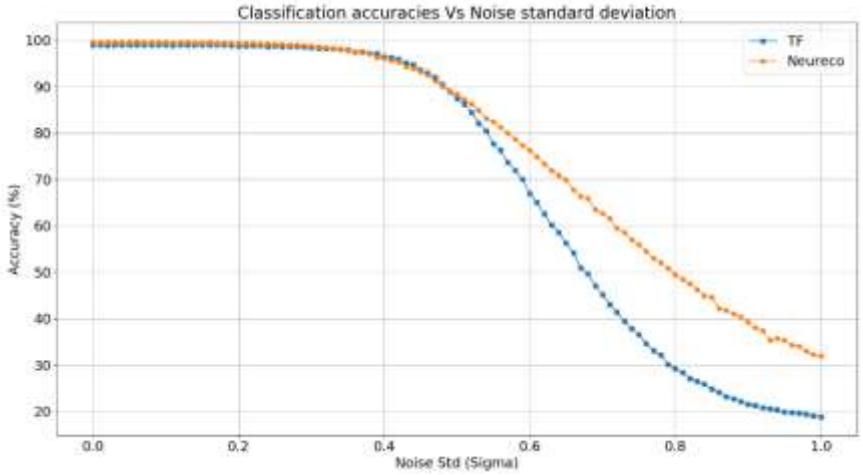


Fig 3. The effect on classification accuracy of the TensorFlow and NeurEco networks when the input images a random gaussian noise is added to the images. Grey level of images is coded in the interval [-1,1].

4 Interpretation of Models

In this paragraph we address the interpretation of the output of a neural network by taking inspiration from the LIME method. We illustrate this application with a test case in which the objective is to count the number of a given object in an image. In general, the object could be anything (vehicles, people, cells…), and this information does not influence the learning process; but for this example, we are counting the number of crosses in an image that contains both crosses and "L".

We built two types of networks: a redundant network using TensorFlow and a parsimonious neural network. Table 2 shows that the parsimonious approach reduces the error on the test data.

Table 2. Performance of the networks.

	Relative Error (%)	Size of the CNN	Size of the classifier	Total Size
TensorFlow	1.2	140 129	98 231	238 360
NeurEco	0.8	1 403	1 613	3 016

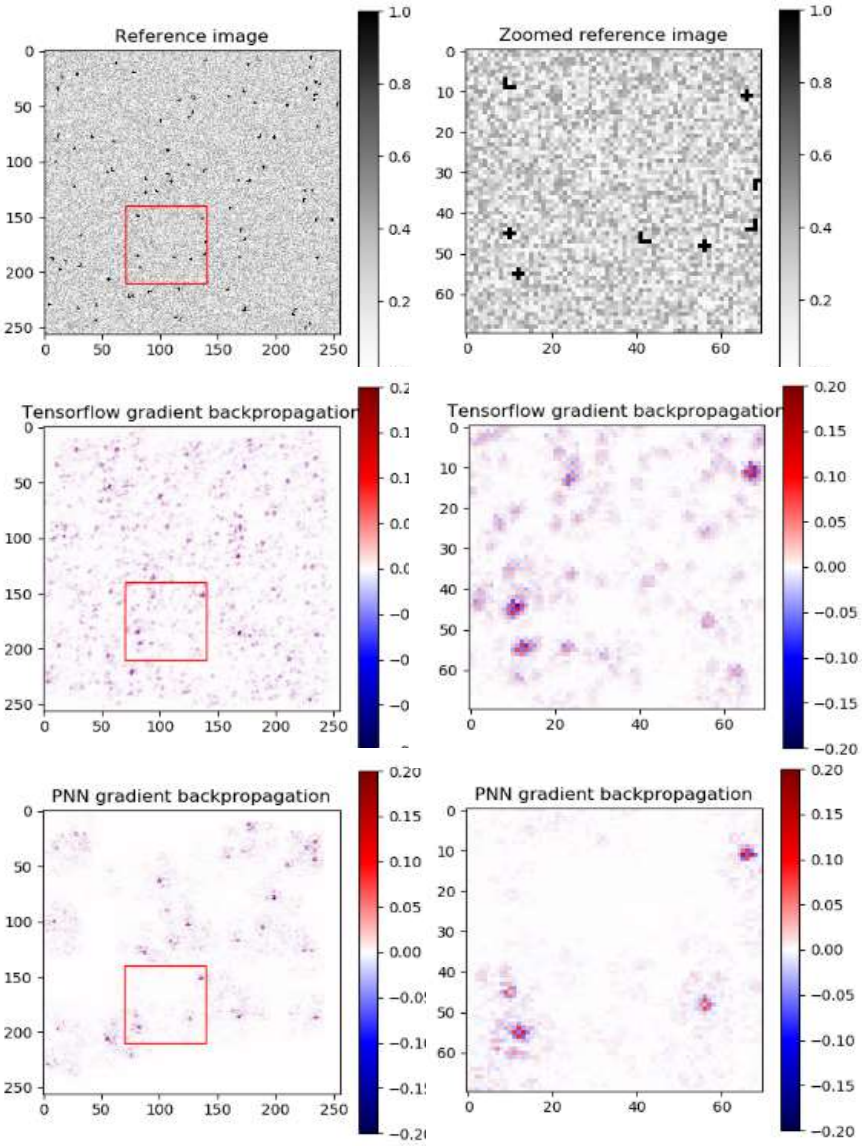


Fig 4: Lime results on full images (left column), zoomed details (right column), input images (top row), results obtained by TensorFlow (middle row) and results obtained by parsimonious neural networks (bottom row).

We consider a given image and apply the model to it, then calculate the sensitivity of the output with respect to a variation of the pixels of the image by backpropagation. The pixels that have the most effect on the response have a stronger sensitivity. If the model respects this property, trust in that model is strengthened, otherwise there may be legitimate doubts about the ability of the model to make predictions.

For the parsimonious model, we can observe, in Fig. 4, a very good match between a high density of the gradient and the presence of crosses on the original image. However, this is not the case for the redundant model obtained by TensorFlow.
Notice that the label of a learning image is a number; the learning process ignores the fact that we are looking for crosses.

5 Conclusion

In addition to resistance to DeepFool attacks, NeurEco limits the black box effect and allow a better validation of the model; this opens new perspectives for certification of the response of neural networks.

Due to the reduction by several orders of magnitude of the resources necessary for their implementation, including energy, NeurEco models can be easily embedded. Recently, awareness of the carbon impact of artificial intelligence has increased. The carbon footprint of NeurEco could be negative if we take into account their application to the reduction of energy consumption in other areas.

Acknowledgement

We would like to extend our gratitude to the peer reviewers for their constructive feedback.

References

1. Amstutz, S., Horchani, I., & Masmoudi, M. (2005). Crack detection by the topological gradient method. Control and Cybernetics, 34(1), 81-101.
2. Chen, T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations. In Advances in Neural Information Processing Systems (pp. 6572-6583).
3. Fujita, M., & Rackwitz, R. (1988). Updating first-and second-order reliability estimates by importance sampling. Doboku Gakkai Ronbunshu, 1988(392), 53-59.
4. Garreau, S., Guillaume, P., & Masmoudi, M. (2001). The topological asymptotic for PDE systems: the elasticity case. SIAM journal on control and optimization, 39(6), 1756-1778.
5. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.
6. Masmoudi, M., Pommier, J., & Samet, B. (2005). The topological asymptotic expansion for the Maxwell equations and some applications. Inverse Problems, 21(2), 547.

7. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P. (2015). DeepFool: a simple and accurate method to fool deep neural networks. Proceedings of the IEEE conference on computer vision and pattern recognition (pp 2574-2582)
8. Ribeiro, M. T., Singh, S., & Guestrin, C. (2016, August). Why should I trust you?: Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1135-1144). ACM.
9. Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. arXiv preprint arXiv:1906.02243.
10. Fehrenbach, J., Masmoudi, M., Souchon, R., & Trompette, P. (2006). Detection of small inclusions by elastography. Inverse problems, 22(3), 1055.
11. Schuëller, G. I., Pradlwarter, H. J., & Koutsourelakis, P. S. (2004). A critical appraisal of reliability estimation procedures for high dimensions. Probabilistic engineering mechanics, 19(4), 463-474.