# Synthetic Training Data Generation for ML-based Small-Signal Stability Assessment

Sergio A. Dorado-Rojas, Marcelo de Castro Fernandes, Luigi Vanfretti
Rensselaer Polytechnic Institute
Electrical, Computer, and Systems Engineering Department
Troy NY, USA
{dorads, decasm3, vanfrl}@rpi.edu

***Abstract*—This article presents a simulation-based massive data generation procedure with applications in training machine learning (ML) solutions to automatically assess the small-signal stability condition of a power system subjected to contingencies. This method of scenario generation for employs a Monte Carlo two-stage sampling procedure to set up a contingency condition while considering the likelihood of a given combination of line outages. The generated data is pre-processed and then used to train several ML models (logistic and softmax regression, support vector machines, $k$-nearest Neighbors, Naïve Bayes and decision trees), and a deep learning neural network. The performance of the ML algorithms shows the potential to be deployed in efficient real-time solutions to assist power system operators.**

## I. Introduction

### *Motivation*

Simulation tools are broadly used to gain insight into current and future operating conditions of the grid. Despite this, due to the vast number of variables and the ubiquitous uncertainty of modern networks, the amount of scenarios that need to be considered by a human operator in simulation-based studies is exponentially large. In this situation, a need arises not only to automate the simulation procedure (to massively generate data) but also to simplify data interpretation. For the former, Python-based solutions have been gaining popularity in almost all engineering fields by providing simple automation solutions (see [1] for an application example in power grids).

Regarding data interpretation, Machine Learning (ML) techniques are powerful statistical methods that can be used, for example, to extract information from large sets of data [2]. In special, Deep Learning (DL) is a particular family of ML techniques that employ Neural Networks (NNs) as building blocks. Both ML and DL solutions have been recently applied in power systems for the classification of events from PMU data [3], [4], voltage stability [5]–[7], and dynamic security assessment [8], among others. ML and DL solutions have been recently applied in power systems for the enhancement and evaluation of small-signal stability. For example, the work of [9] performs coordinated tuning of PSS parameters using heuristic optimization algorithms. Likewise, in [10] a cuckoo search is employed to find optimal PSS parameters that guarantee small-signal stability. Regarding NNs, in [11] the parameters of a unified power flow controller are tuned via a NN whose weights are optimized using Levenberg-Marquardt optimization.

This work was funded in part by the New York State Energy Research and Development Authority (NYSERDA) through the Electric Power Transmission and Distribution (EPTD) High Performing Grid Program under agreement number 137951, in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under Award EEC-1041877, in part by the CURENT Industry Partnership Program, and in part by the Center of Excellence for NEOM Research at King Abdullah University of Science and Technology.

The authors would like to thank Tetiana Bogodorova for the insightful discussions that helped improve the final presentation of the paper.

On the other hand, within several well-established stability techniques, small-signal analysis quantifies the effects of small disturbances in a given power system. Such small-scale perturbations can lead to large instabilities if specific modes of the system are excited. Traditionally, oscillations are studied by obtaining a linear model of the power system around a stable equilibrium point, and evaluating eigenvalues of this model. Alternatively, eigenvalues can be determined from time-domain measurements or simulation data via traditional signal processing and system identification algorithms [12].

Once the eigenvalues describing a particular small-signal scenario are available, its classification in pre-established categories is straightforward. In fact, given the set of dominant eigenvalues $\lambda_i$ of the system in a particular contingency scenario, the operational state may be assessed by computing the damping ratio $\zeta_i$ for each eigenvalue $\lambda_i$. Hence, $\zeta$ represents a metric that can be used to define a 100%-accurate classifier to categorize contingency scenarios ($\zeta$ classifier).

Furthermore, the most challenging step from the computational point of view is the state matrix computation rather the damping ratio calculation, specially if the former is done numerically instead of analytically [13]. Despite this, a system identification-based method is preferred since it is 100% accurate at evaluating the system's condition. A valid question, however, would be if an alternative ML solution could be used to bypass the system identification step while producing more computationally efficient solutions. For this alternative to be practically significant, the ML solution should be accurate enough at both learning the linear system representation and evaluating the system condition from eigenvalues. This paper focuses on the latter issue and explores the small-signal stability assessment accuracy of several ML techniques.

*Contribution*

This paper takes the challenge to generate massively contingency data and to automate small-signal stability computation by ML methods. The contribution of this work is as follows:

- we propose a simple ad-hoc Monte Carlo sampling technique to generate numerous contingency scenarios for a given power system model. Then, each scenario is simulated in a Modelica-based environment to obtain labeled big data for ML algorithms training;
- we use the generated big data to train several conventional ML algorithms (logistic and softmax regression, support vector machines, $k$-nearest Neighbors, Naïve Bayes and decision trees), and a deep learning NN to classify the operating condition of a power system after a contingency (i.e., one or more line trips) based on a small-signal stability metric;
- we propose an evaluation metric as a guideline for the selection of the best classifier in terms of its performance.

We verify that once trained, the ML approaches produce results as accurate as the damping ratio-based classifier, which is implemented in Python using numPy in a vectorized fashion ($\zeta$ classifier). Moreover, almost all ML solutions take less prediction time than the hard-coded domain-specific algorithm to classify operation scenarios. This is desirable for deploying a trained solution inside a real-time application. In particular, the trained NN shows 120x faster prediction time than the damping ratio classifier with an accuracy above 95%.

An important contribution of the paper concerns the approach for massive data generation. The information required for a linear analysis (that is, the $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ matrices of a linear state-space representation) is obtained through a routine implemented in Dymola, a Modelica-based environment. We take advantage of this built-in functionality to linearize a nonlinear dynamic power system model around an equilibrium point. By doing so, it is possible to perform a small-signal analysis for a vast quantity of scenarios in a power system using a model constructed using the Open Instance Power System Library (OpenIPSL) [14], a library for phasor time-domain analysis in Modelica. Each scenario is generated by a two-stage Monte Carlo sampling procedure that takes into account the topology of the system as described in Section III.

By automating Dymola linearization with Python [15], a vast amount of data is generated for several grid conditions with different small disturbances in the form of contingencies. Such an intensive simulation-based data generation approach has been recently used in other power system studies such as transmission planning as well [16]. In this case, the data is employed to train an automatic classifier such as an ML algorithm to evaluate small-signal condition of the system. The complete code used for this paper is available on GitHub[1].

*Paper Organization*

This paper is organized as follows: in Section II we present a brief overview of small-signal analysis and how eigenvalues can be classified. The test power system and the data generation procedure are outlined in Section III. Section IV describes and presents the proposed Neural Network architecture, the results of its training procedure, and its performance after being deployed. Finally, Section VI concludes the work.

## II. Foundations of Small-Signal Analysis

Consider a generic representation of a power system, ignoring discrete events such as faults in protections, with $m$ inputs, $p$ outputs and $n$ states, whose state-space is described by $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$, and $\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}, t)$ where $\mathbf{x} \in \mathbb{R}^{n\times 1}$ is the state vector, $\mathbf{u} \in \mathbb{R}^{m\times 1}$ is the vector of $m$ inputs to the system, and $\mathbf{f} \in \mathbb{R}^{n\times 1}$, $\mathbf{g} \in \mathbb{R}^{p\times 1}$ are two nonlinear $\mathcal{C}^\infty$ functions. If time dependence is implicit (i.e., time does not appear explicitly in the system equations), we have

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}, \mathbf{u}). \end{aligned} \tag{1}$$

For the system in Eq. (1), an equilibrium exists whenever the state derivatives are zero ($\dot{\mathbf{x}} = \mathbf{0}$). At a given equilibrium point $(\mathbf{x}_0, \mathbf{u}_0)$, we have $\mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) = \mathbf{0}$. Now, we analyze the situation where the system is in equilibrium and a small disturbance occurs. The disturbance brings the system to a new state. The dynamics at the new operating point take the form

$$\begin{aligned} \dot{\tilde{\mathbf{x}}} &= \mathbf{f}(\tilde{\mathbf{x}}, \tilde{\mathbf{u}}) = \mathbf{f}(\mathbf{x}_0 + \Delta\mathbf{x}, \mathbf{u}_0 + \Delta\mathbf{u}) \\ \tilde{\mathbf{y}} &= \mathbf{g}(\tilde{\mathbf{x}}, \tilde{\mathbf{u}}) = \mathbf{g}(\mathbf{x}_0 + \Delta\mathbf{x}, \mathbf{u}_0 + \Delta\mathbf{u}). \end{aligned} \tag{2}$$

If the disturbance under consideration is small enough[2], we can perform a Taylor Series expansion around an equilibrium point for both functions $\mathbf{f}$ and $\mathbf{g}$ to obtain a linear representation of the nonlinear system. This can be achieved by a first-order Taylor Series truncation neglecting all terms of order larger than one, thus keeping only the matrix of first-order derivatives (Jacobian linearization). An application of this formula to the system in Eq. (1) leads us to

$$\begin{aligned} \Delta\dot{\mathbf{x}} &= \mathbf{A}\Delta\mathbf{x} + \mathbf{B}\Delta\mathbf{u} \\ \Delta\mathbf{y} &= \mathbf{C}\Delta\mathbf{x} + \mathbf{D}\Delta\mathbf{u} \end{aligned} \tag{3}$$

where $\mathbf{A} := \partial\mathbf{f}/\partial\mathbf{x}$, $\mathbf{B} := \partial\mathbf{f}/\partial\mathbf{u}$, $\mathbf{C} := \partial\mathbf{g}/\partial\mathbf{x}$ and $\mathbf{D} := \partial\mathbf{g}/\partial\mathbf{u}$. Note that the system representation in Eq. (3) corresponds to a state-space realization of a Linear Time-Invariant system that can be analyzed using linear methods.

Linear analysis techniques can be employed to quantify system behavior after a small disturbance (hence the name *small-signal*) such as tripping of a given line. System modes of a linear system are completely specified by the eigenvalues of the system matrix $\mathbf{A}$. For the $i$th eigenvalue $\lambda_i$ with algebraic multiplicity $n_i$, the associated $n_i$ modes are $\bar{c}_i t^k e^{\lambda_i t}$ for $k = 0,\ 1, \ldots,\ n_i - 1$, with $\bar{c}_i \in \mathbb{C}$. The characteristic of the mode associated to a single eigenvalue can be completely described by a single metric known as *damping factor* or *damping ratio* $\zeta$. As shown in Figure 1[3] , the damping factor

[1] https://github.com/ALSETLab/Synthetic_Data_Generation_ML_Small_Signal

[2]This means that the region of convergence of the Taylor Series corresponds to a non-empty set in the neighborhood of the equilibrium point.

[3]Since we are using Lyapunov's first method for the nonlinear system in Eq. (1), no conclusion regarding equilibrium stability can be drawn if the $\mathbf{A}$ matrix is not Hurwitz (i.e., $\zeta \leq 0$).

determines uniquely the characteristics of the system mode associated with a particular eigenvalue.

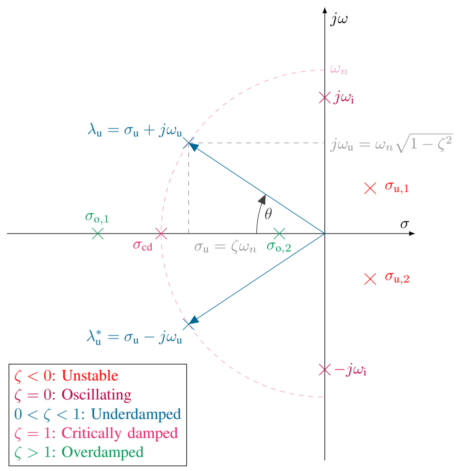

Fig. 1. Damping factor depending on different eigenvalue locations on the complex plane.

Thanks to the characteristics of the damping factor as a discriminative scalar metric, it is possible to categorize eigenvalues based on $\zeta$ such that each of the classification groups represents a state of an electric grid if the corresponding eigenvalue is linked to a dominant mode. This automatic labeling can help learn the stability condition of a power system.

## III. Scenario Sampling for Data Generation

The IEEE 14 bus network is used as a test bench to generate large-scale data for NN training by systematically applying contingencies and collecting simulation data.

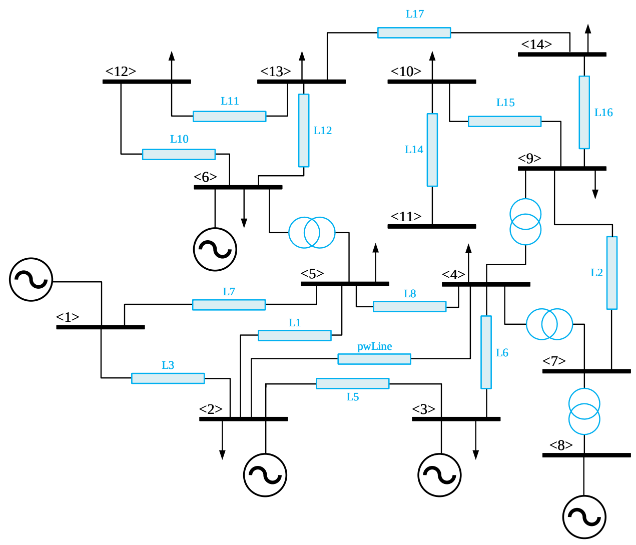

Fig. 2. IEEE 14 nodes test system.

This system, which is shown in Figure 2, counts with 5 generators, 16 lines and 4 transformers. In summary, there are 20 branch elements in the system, all of them having an impedance in the corresponding power system model.

If the branch impedance value $X_i$ of one of these element models is made large enough, we would have emulated a line opening in the system without actually removing the component. In fact, by letting $X_i \approx 10^{12}$ we do not alter the topology of the grid (and do not change the number of states nor the size of the $\mathbf{A}$ matrix) but we effectively ”apply” a contingency to the system which is equal to disconnecting the branch.

Considering that each of the 20 branch elements can be tripped, there exist 20 possible scenarios that can help to evaluate the disconnection of a single element. Likewise, if two branches are opened simultaneously, 190 possible scenarios can be tested by selecting all branches pairwise. In general, letting $n$ being the total number of branches, and $k$ the number of simultaneous disconnections, the amount of possible scenarios with this contingency configuration is given by

$$S_{n,k} = \frac{n!}{(n-k)!k!}. \tag{4}$$

For this study, there is no need to consider all lines being opened at the same time since it is not physically significant. Hence, a maximum of $k_{\text{max}} = n - 1 = 19$ simultaneous disconnections is considered. In addition, note that $k_{\text{min}} = 1$. Thus, it is possible to calculate the total amount of possible scenarios as

$$T = \sum_{k_{\text{min}}}^{k_{\text{max}}} S_{20,k} = \sum_{k=1}^{19} \frac{20!}{(20-k)!k!} = 1,048,574. \tag{5}$$

From the total amount of possible scenarios, $T$, it is necessary to select a subset of events with physical significance. To address this issue, an ad-hoc Monte Carlo method, consisting of a two-stage sampling procedure to select scenarios, is proposed. In the first stage, the number of lines that will be opened is selected. Here, it is necessary to find a probability distribution that reflects the fact that scenarios with a smaller number of events are more likely to occur and, therefore, should be more likely to be selected. To take this constraint into account, a modified Poisson distribution is proposed, giving larger probabilities to smaller values of $k$. The Probability Density Function, is illustrated in Figure 3 is defined as:

$$p(k) = \frac{1}{k!\sum_{n=1}^{19}\frac{1}{n!}} \approx \frac{1}{k!(e-1)}, \tag{6}$$

where $e$ is Euler’s number. Once the number of lines is fixed, the second stage starts, and one scenario is selected from the pool of all possible combinations with the specified number of contingencies. Thanks to this ad-hoc method, 20,000 different simulation scenarios are generated.

Once a simulation scenario has been selected, the corresponding branches in the system model in Dymola are disconnected. Dymola’s built-in function `linearizeModel` is used to extract the state matrix and the eigenvalues for each scenario. This process is automated in Python using the so-called Python-Dymola Interface (PDI) [15], that allows combining the simulation power of Dymola with Python capabilities for a variety of purposes. The usage of PDI also allows several robust ML development libraries, such as scikit-learn and TensorFlow, to be used to train classification ML/DL solutions from the significant amount of data produced by massive simulations.

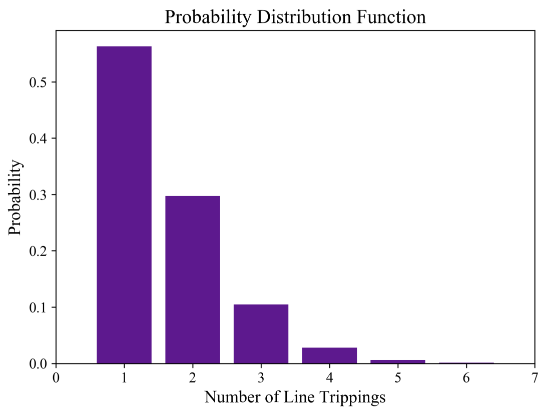

Fig. 3. Probability Distribution Function for the number of lines to be opened for contingency generation.

## IV. ML Algorithm Design, Training and Performance

In this section, we briefly describe the different steps carried out to design the ML/DL algorithms employed to classify the different contingency scenarios into pre-established categories.

– **Step 1 - Data Generation:** we use 20,000 scenarios of the IEEE 14 bus system to generate eigenvalue data. Dymola succeeded in linearizing 19,815 of those scenarios. Since each one is associated with 49 eigenvalues, a total number of 970,935 eigenvalues was produced. An overview of the generated eigenvalues is shown in Figure 4.

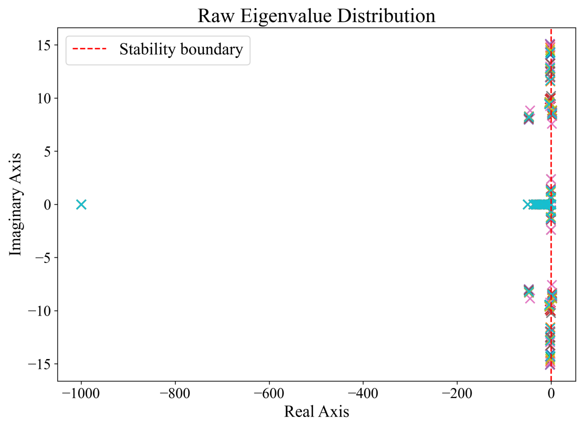

Fig. 4. Distribution of raw eigenvalues in the complex plane.

– **Step 2 - Data Preprocessing:** the raw data obtained from simulations is organized and labeled (i.e., by manually classifying the eigenvalues, computing the damping ratio and tagging them according to the pre-defined categories below, what we refer to as *hard-coded classifier*). Each eigenvalue is classified within one of six categories (Figure 5) that are defined as follows:

1) **Unstable** ($\zeta < 0$): if an eigenvalue lies on the right-half plane.
2) **Stable but critical condition** ($0 \leq \zeta < 0.05$): the eigenvalue is stable or it is oscillatory (so no conclusion can be drawn about the stability of the system). This is a condition for which an action of the system controls is required since a small disturbance can lead to instabilities and/or heavy oscillatory modes in the system.
3) **Acceptable condition within operating limits** ($0.05 < \zeta < 0.1$): in this case, the damping of the system is large enough to handle and tolerate a small-disturbance. As a consequence, the operation of the system is labeled as acceptable.
4) **Good operating condition** ($0.1 \leq \zeta < 1$): for this scenario, the damping ratio is larger than 10% and the response will show some oscillation due to the underdamped nature of the corresponding eigenvalue.
5) **Satisfactory operating condition** ($\zeta > 1.1$): this category gathers real eigenvalues whose overdamped response is satisfactory in terms of oscillations. Normally, these eigenvalues are not dominant. Hence, they do not impose its dynamics on system response.
6) **Irrelevant (eigenvalue at the origin or close to it)**: category that groups the eigenvalues that are at the origin or close to it (within a neighborhood of radius 0.2). They mostly represent integral relationships between state variables (i.e., between $\omega$ and $\delta$).

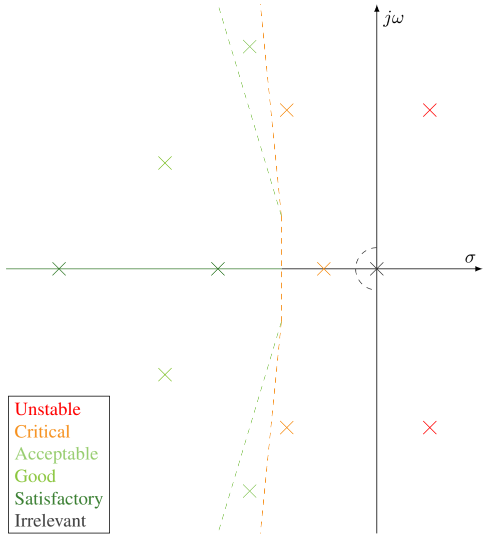

Fig. 5. Regions of each classification category on the complex plane.

In the pre-processing stage, the eigenvalues whose magnitude is magnitude larger than one are normalized (Figure 6). All $\lambda_i$-s lying inside the unit circle are not touched since the information they convey regarding the stability boundary of the system would be lost.

– **Step 3 - Model Design, Training and Evaluation:** we evaluated both classical ML techniques for classification as well as a fully-connected NN. The selected supervised ML algorithms are multi-class logistic regression (LogReg), softmax regression (SoftmaxReg), linear Support Vector Machines

(SVM), $k$-nearest neighbors ($k$-NN), Naïve Bayes and decision trees.

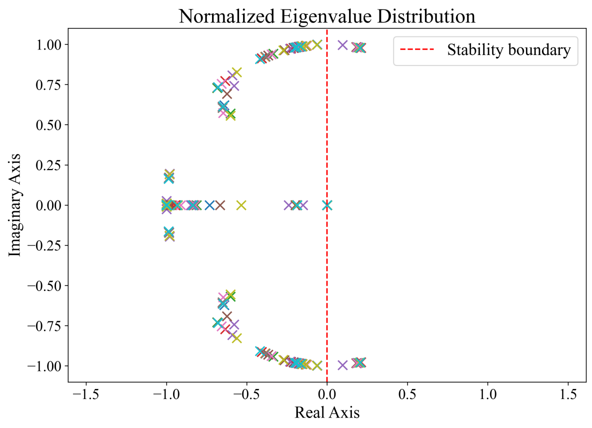

Fig. 6. Distribution of normalized eigenvalues in the complex plane.

The reader is referred to [2] for an in-depth explanation of each ML algorithm. We will place special attention to the NN design since it is the best performing method among all ML/DL techniques considered in this study.

The proposed NN architecture is presented in Figure 7. It consists of six fully-connected layers, four of them with learning parameters. Two dropout layers are added to reduce overfitting and improve generalization performance. The input layer and the two hidden layers employ a ReLU unit as an activation function. Since this is a multi-class classification problem, a softmax function is suitable as an output activation.

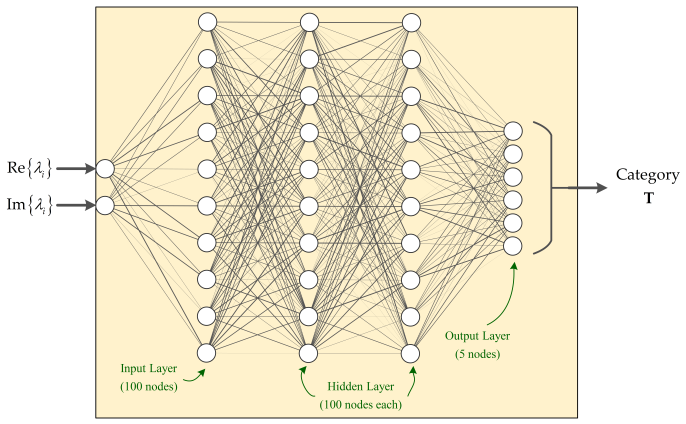

Fig. 7. Neural Network architecture.

Let $m$ be the number of training samples in a batch. The input feature matrix will be $\mathbf{X}^{m\times 2}$. Then,

$$\begin{aligned}
\mathbf{H}^{[1]} &= \mathrm{ReLU}\left(\mathbf{X}\mathbf{W}^{[1]} + \mathbf{b}^{[1]}\right)\\
\mathbf{D}^{[1]} &= \mathrm{dropout}\left(\mathbf{H}^{[1]}\right)\\
\mathbf{H}^{[2]} &= \mathrm{ReLU}\left(\mathbf{D}^{[1]}\mathbf{W}^{[2]} + \mathbf{b}^{[2]}\right)\\
\mathbf{D}^{[2]} &= \mathrm{dropout}\left(\mathbf{H}^{[2]}\right)\\
\mathbf{H}^{[3]} &= \mathrm{ReLU}\left(\mathbf{D}^{[2]}\mathbf{W}^{[3]} + \mathbf{b}^{[3]}\right)\\
\mathbf{Y} &= \sigma_{\mathcal{M}}\left(\mathbf{H}^{[3]}\mathbf{W}^{[4]} + \mathbf{b}^{[4]}\right)\\
\mathbf{T} &= \mathrm{argmax}\left(\mathbf{Y}\right)
\end{aligned} \tag{7}$$

where the hidden states are described by the matrices $\mathbf{H}^{[1]}$, $\mathbf{H}^{[2]}$, $\mathbf{H}^{[3]} \in \mathbb{R}^{m\times 100}$, $\sigma_{\mathcal{M}}\left(\mathbf{Z}^{[4]}\right) \in \mathbb{R}^{m\times 5}$ with $\mathbf{Z}^{[4]} := \mathbf{H}^{[3]}\mathbf{W}^{[4]} + \mathbf{b}^{[4]}$. $\mathbf{D}^{[1]}$ and $\mathbf{D}^{[2]}$ are the outputs of the dropout layers (not trainable). The weights and biases are $\mathbf{W}^{[1]} \in \mathbb{R}^{2\times 100}$, $\mathbf{W}^{[2]} \in \mathbb{R}^{100\times 100}$, $\mathbf{W}^{[3]} \in \mathbb{R}^{100\times 100}$, $\mathbf{W}^{[4]} \in \mathbb{R}^{100\times 5}$, $\mathbf{b}^{[1]}$, $\mathbf{b}^{[2]}$, $\mathbf{b}^{[3]} \in \mathbb{R}^{100\times 1}$, and $\mathbf{b}^{[4]} \in \mathbb{R}^{5\times 1}$. $\mathbf{Y} \in \mathbb{R}^{m\times 5}$ is a matrix whose $m$th column contains the probability of each of the $m$ inputs to belong to each category. Finally, $\mathbf{T}$ is a matrix whose $m$th vector has a 1-entry at the position corresponding to the class with the highest probability, and zero everywhere else.

The loss function is a cross-entropy function defined minibatch-wise as $\ell_m = -\frac{1}{m}\sum_{i=1}^{5}\log y_i$ where $y_i$ is the $i$th component of the $m$th column of $\mathbf{Y}$. Finally, the total loss is computed by adding the individual losses per minibatch as $L = \sum_{<m>}\ell_m$. The weights and biases are learned by minimizing the loss function $L$. The solution of this complex optimization problem is carried out by the specialized Python library TensorFlow. The behavior of the loss function per learning epoch can be detailed in Figure 8, together with the results of training and testing accuracy. The number of epochs is set to 50 to get the final values of testing (97.79%) and training (98.60%) accuracies before deploying the NN.

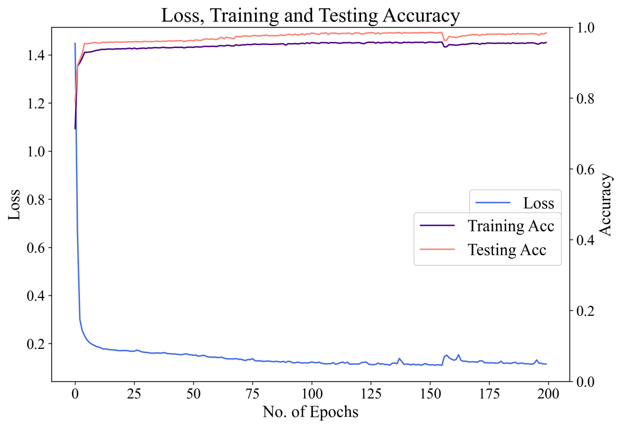

Fig. 8. Loss function value, training and testing accuracy per training epoch.

Note that the damping ratio computation is required for creating the labels for each category (what we call a hard-coded classifier) but not for the training process. In particular, the NN may learn the damping ratio representation of the eigenvalues in some hidden layers if it is useful for the classification task itself.

Finally, after training the NN is evaluated on the validation set. The results of the prediction for all methods can be found in the GitHub repository, together with more visualization of the predictions. The accuracy on the validation stage for the NN was of 93.36%. In Figure 9, it can be seen that the NN learns effectively the highly nonlinear decision boundaries that separate each classification group in the complex plane since the predicted labels (right) are almost the same for all cases to the ground-truth (left). The most pronounced discrepancy is the misclassification of few 'good' instances as 'critical' eigenvalues (close to $\mathrm{Re}\{s\} = -0.2$).

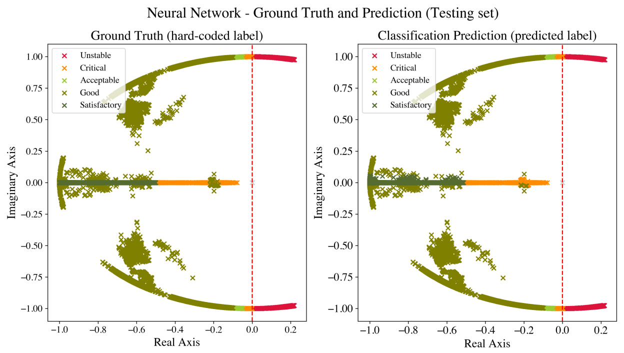

Fig. 9. Ground truth and prediction results for the trained NN.

To quantify the performance and benchmark the different algorithms against each other, a simple numerical score was defined. Let $\mathbf{t}_{\text{ex}}$ be a vector containing the prediction time for every algorithm. $\mathbf{a}_{\text{train}}$ and $\mathbf{a}_{\text{test}}$, $\mathbf{p}_{\text{train}}$ and $\mathbf{p}_{\text{test}}$, and $\mathbf{r}_{\text{train}}$ and $\mathbf{r}_{\text{train}}$ are vectors containing the information about accuracy, precision and recall for the training and testing set, respectively. Then, the score for the $i$th algorithm is given by:

$$\begin{aligned} s^{[i]} &= 0.4\left(\frac{\min\left(\mathbf{t}_{\text{ex}}\right)}{t_{\text{ex}}^{(i)}}\right) \\ &+ 0.2\left[\frac{1}{3}\left(\frac{\mathbf{a}_{\text{train}}}{\max\left(\mathbf{a}_{\text{train}}\right)}+\frac{\mathbf{p}_{\text{train}}}{\max\left(\mathbf{p}_{\text{train}}\right)}+\frac{\mathbf{r}_{\text{train}}}{\max\left(\mathbf{r}_{\text{train}}\right)}\right)\right] \\ &+ 0.4\left[\frac{1}{3}\left(\frac{\mathbf{a}_{\text{test}}}{\max\left(\mathbf{a}_{\text{test}}\right)}+\frac{\mathbf{p}_{\text{test}}}{\max\left(\mathbf{p}_{\text{test}}\right)}+\frac{\mathbf{r}_{\text{test}}}{\max\left(\mathbf{r}_{\text{test}}\right)}\right)\right] \end{aligned} \tag{8}$$

This score benefits the method with the minimum execution time which maximizes training and testing accuracy. For this reason, $k$-NN has a poor score despite its high accuracy, precision and recall performance. Testing has a higher weight than training on the final score since the algorithm is exposed to new instances, and therefore this number is a better indicator of generalization. The results for each method are presented in Table I where the accuracy, precision and recall are computed on the testing set.

TABLE I
Performance Metrics and Score for ML/DL Classifiers

| Method | $t_{\text{pred}}$ | Acc | Prec | Rcl | Score |
|---|---|---|---|---|---|
| $\zeta$ classifier | 5.477 s | 100% | 100% | 100% | 0.6032 |
| LogReg | 0.058 s | 78.20% | 64.09% | 79.07% | 0.7671 |
| SoftmaxReg | 0.054 s | 78.10% | 68.22% | 83.02% | 0.8067 |
| Linear SVM | 0.051 s | 67.79% | 41.61% | 36.82% | 0.6228 |
| $k$-NN | 33.027 s | **99.83%** | **99.68%** | **99.92%** | 0.5997 |
| Naïve Bayes | 0.255 s | 97.11% | 86.42% | 93.93% | 0.6091 |
| Decision Trees | 0.057 s | 98.93% | 93.44% | 96.19% | 0.8933 |
| Neural Networks | **0.045 s** | 98.53% | 92.49% | 95.90% | **0.9750** |

## V. Scalability

The proposed method to generate contingency scenarios can be scaled to deal with larger systems both in terms of buses (BUS) and number of states (STA) and variables (VAR). Note that for line openings, the maximum number of scenarios (SC) depends on the number of branches in the model (either transmission lines or transformers, written as BR in Table II).

Systems with more than 30 branches (such as the Nordic 44) were constrained to have a maximum of five simultaneous contingencies (i.e., so that $N-5$ is considered) to avoid sampling of unrealistic scenarios. Table II illustrates how the number of scenarios increases for different systems along with the estimated contingency pool generation time or execution time of the program (ET). This scalability benchmark was performed on a Ubuntu 18.04.5 LTS machine with a AMD Epyc 7601 32-core processor and 512 GB of RAM.

TABLE II
Scalability of Scenario Generation for Several Systems

| System | BUS | STA | VAR | BR | SC | ET |
|---|---|---|---|---|---|---|
| IEEE 9 | 9 | 24 | 203 | 9 | 510 | 0.0348 s |
| Seven Bus | 7 | 132 | 678 | 18 | 262,142 | 0.0781 s |
| IEEE 14 | 14 | 49 | 426 | 20 | 1,048,574 | 0.3038 s |
| N44 | 44 | 1294 | 6315 | 79 | 24,122,225 | 5.5966 s |

The effect of constraining the number of lines that can be simultaneously opened in a scenario not only enhances the practical significance of the method but also increases the computational efficiency when working with large systems for dynamical studies. In Figure 10 we see that the execution time grows exponentially as the number of branches increases requiring several minutes for a system with $\approx 30$ branches. This number can represent relatively low interconnected grid models. Thus, we see that the method requires an adjustment for dealing with large-scale highly-interconnected systems.

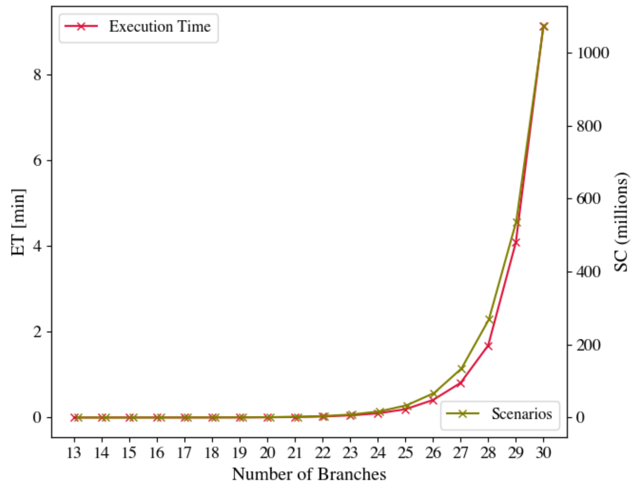

Fig. 10. Scalability of the contingency generation algorithm without constraining the number of simultaneous outages.

By setting an upper bound on the number of maximum simultaneous contingencies, the execution time diminishes (from minutes to seconds) since the number of possible combinations is truncated. However, the number of scenarios is still significant: around 20 million for a system with $\approx 80$ branches which can be obtained in less than 10 seconds. This suggests that working with larger grids would need to limit further the number of simultaneous trippings.

It must be stressed that the total amount of contingencies constitutes the sampling pool for the second stage of the proposed algorithm. The larger the pool is, the faster the method samples an arbitrary number of scenarios. It is clear that a larger scenario pool will ease the search for feasible contingency conditions.

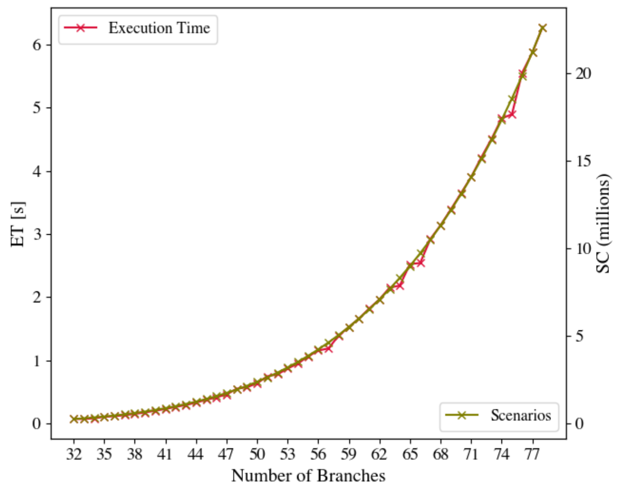

Fig. 11. Scalabiliity of the contingency generation algorithm with an upper bound on the number of simultaneous line trippings.

## VI. Conclusions

Through this work, we have seen how several ML methods can be employed to classify eigenvalues representing the behavior of a power system after the occurrence of a contingency. In particular, a decision tree and a NN have shown to be almost as accurate as a conventional classifier solution which computes the damping ratio for each eigenvalue while predicting faster. In this example, a considerable number of scenarios were studied by automating Modelica-based power system simulations thanks to the Python-Dymola Interface. The use of the PDI enabled to integrate the development of the ML to the power system simulation. The trained NN showed classification performance above 95% for the testing data. This promising result highlights the potential of ML methods for deployment in real-time intelligent power system solutions. Future work will be focused on analyzing performance sensitivity to NN architecture, activation function selection and scalability of the synthetic data generation approach for bigger systems.

## References

[1] N. Vyakaranam, Bharat Samaan, X. Li, R. Huang, Y. Chen, and X. Vallem, Mallikarjuna Nguyen, Tony and Tbaileh, Ahmad and Elizondo, Marcelo and Fan, "Dynamic Contingency Analysis Tool 2.0 User Manual with Test System Examples," Pacific Northwest National Laboratory, Tech. Rep., 2019.

[2] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow*, 2nd ed. O'Reilly, 2019.

[3] S. A. R. Konakalla and R. A. de Callafon, "Feature Based Grid Event Classification from Synchrophasor Data," *Procedia Computer Science*, vol. 108, pp. 1582–1591, 2017.

[4] C. Zheng, V. Malbasa, and M. Kezunovic, "Regression tree for stability margin prediction using synchrophasor measurements," *IEEE Transactions on Power Systems*, vol. 28, no. 2, pp. 1978–1987, may 2013.

[5] J. D. Pinzón and D. G. Colomé, "Real-time multi-state classification of short-term voltage stability based on multivariate time series machine learning," *International Journal of Electrical Power & Energy Systems*, vol. 108, pp. 402–414, jun 2019.

[6] H.-Y. Su and T.-Y. Liu, "Enhanced-Online-Random-Forest Model for Static Voltage Stability Assessment Using Wide Area Measurements," *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 6696–6704, nov 2018.

[7] W. D. Oliveira, J. P. Vieira, U. H. Bezerra, D. A. Martins, and B. d. G. Rodrigues, "Power system security assessment for multiple contingencies using multiway decision tree," *Electric Power Systems Research*, vol. 148, pp. 264–272, jul 2017.

[8] I. Konstantelos, G. Jamgotchian, S. Tindemans, P. Duchesne, S. Cole, C. Merckx, G. Strbac, and P. Panciatici, "Implementation of a Massively Parallel Dynamic Security Assessment Platform for Large-Scale Grids," in *2018 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, aug 2018, pp. 1–1.

[9] W. Peres, E. J. de Oliveira, J. A. Passos Filho, and I. C. da Silva Junior, "Coordinated tuning of power system stabilizers using bio-inspired algorithms," *International Journal of Electrical Power & Energy Systems*, vol. 64, pp. 419–428, jan 2015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0142061514004724

[10] D. Chitara, K. R. Niazi, A. Swarnkar, and N. Gupta, "Cuckoo Search Optimization Algorithm for Designing of a Multimachine Power System Stabilizer," *IEEE Transactions on Industry Applications*, vol. 54, no. 4, pp. 3056–3065, jul 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8309288/

[11] M. J. Rana, M. S. Shahriar, and M. Shafiullah, "Levenberg–Marquardt neural network to estimate UPFC-coordinated PSS parameters to enhance power system stability," *Neural Computing and Applications*, vol. 31, no. 4, pp. 1237–1248, apr 2019. [Online]. Available: http://link.springer.com/10.1007/s00521-017-3156-8

[12] F. R. S. Sevilla and L. Vanfretti, "A small-signal stability index for power system dynamic impact assessment using time-domain simulations," in *2014 IEEE PES General Meeting — Conference & Exposition*. IEEE, jul 2014, pp. 1–5.

[13] J. H. Chow and J. J. Sanchez-Gasca, *Power System Modeling, Computation, and Control*. John Wiley & Sons, 2020.

[14] M. Baudette, M. Castro, T. Rabuzin, J. Lavenius, T. Bogodorova, and L. Vanfretti, "OpenIPSL: Open-Instance Power System Library - Update 1.5 to iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations," *SoftwareX*, vol. 7, pp. 34–36, jan 2018.

[15] Dassault Systèmes, *Dymola User Manual*. Dassault Systèmes, 2018.

[16] Z. Zhuo, E. Du, N. Zhang, C. Kang, Q. Xia, and Z. Wang, "Incorporating Massive Scenarios in Transmission Expansion Planning with High Renewable Energy Penetration," *IEEE Transactions on Power Systems*, pp. 1–1, 2019.