Predicting basin stability of power grids using graph neural networks

New Journal of Physics
The open access journal at the forefront of physics

Published in partnership with: Deutsche Physikalische Gesellschaft and the Institute of Physics

OPEN ACCESS

RECEIVED
16 December 2021

REVISED
3 February 2022

ACCEPTED FOR PUBLICATION
14 February 2022

PUBLISHED
29 April 2022

PAPER

Predicting basin stability of power grids using graph neural networks

Christian Nauck[1], Michael Lindner[1], Konstantin Schürholt[2], Haoming Zhang[3], Paul Schultz[4], Jürgen Kurths[1], Ingrid Isenhardt[3] and Frank Hellmann[1,*]

[1] Department of Complexity Science, Potsdam Institute for Climate Impact Research, Telegrafenberg A 31, Potsdam, Brandenburg 14473, Germany
[2] Institute of Computer Science, University of St. Gallen, Rosenbergstrasse 30, St. Gallen 9000, Switzerland
[3] Cybernetics Lab IMA & IfU, RWTH Aachen University, Dennewartstraße 27, Aachen 52068, Germany
[4] 50Hertz Transmission GmbH, Elia Group, Heidestraße 2, Berlin 10557, Germany
[*] Author to whom any correspondence should be addressed.

E-mail: hellmann@pik-potsdam.de

Keywords: complex systems, nonlinear dynamics, dynamic stability, basin stability, power grids, machine learning, graph neural networks

Abstract

The prediction of dynamical stability of power grids becomes more important and challenging with increasing shares of renewable energy sources due to their decentralized structure, reduced inertia and volatility. We investigate the feasibility of applying graph neural networks (GNN) to predict dynamic stability of synchronisation in complex power grids using the single-node basin stability (SNBS) as a measure. To do so, we generate two synthetic datasets for grids with 20 and 100 nodes respectively and estimate SNBS using Monte-Carlo sampling. Those datasets are used to train and evaluate the performance of eight different GNN-models. All models use the full graph without simplifications as input and predict SNBS in a nodal-regression-setup. We show that SNBS can be predicted in general and the performance significantly changes using different GNN-models. Furthermore, we observe interesting transfer capabilities of our approach: GNN-models trained on smaller grids can directly be applied on larger grids without the need of retraining.

1. Introduction

The energy transition is one of the key aspects to meet the goals of the Paris agreement [1] and its latest successor: conference of the parties 26 in Glasgow in 2021. Due to decentralization, reduced inertia as well as volatility in production, integrating renewable energies remains challenging. To safely operate future power grids, the impact of unavoidable fluctuations on the synchronous operating regime has to be limited. Hence, dynamic effects have to be taken into account. Analyzing the dynamic stability of synchronisation in power grids is a complex multi-dimensional problem and many known methods rely on heavy simulations.

The model underlying the recent work on the stability of synchronization and complex dynamics of power grids, e.g. [2], is the Kuramoto model [3] with inertia. In complex system science it also serves as a paradigmatic model for the study of complex phenomena on networks in general [4, 5]. Thus, the results here are of interest beyond the specific scope of power grid modeling.

In the context of complex systems, linear stability assessments alone, e.g. based on Lyapunov exponents, are not always applicable or sufficient. A standard in the power grid community are highly detailed simulations of individual faults. For large systems, the study of all potential individual faults is too expensive, because there are too many of them. To gain a better understanding of the type of faults that might be critical, probabilistic approaches are used. They provide an appropriate understanding and heuristics for prioritizing detailed model studies to systematically investigate the dynamic stability.

Single-node basin stability (SNBS) is such a probabilistic measure. Based on the notion of the 'basin of attraction' of a stable state, SNBS captures highly non-linear effects and enables the analysis of large perturbations [6]. SNBS measures the probability of a grid to synchronize after applying sample-based-perturbations at individual nodes. SNBS has been applied to a variety of problems e.g. in the engineering community for the analysis of perturbed generators in networks [7, 8] and to study collective phenomena in oscillator networks [9, 10]. There are also theoretical investigations of network properties [11–17]. In the non-linear dynamics and complex systems community the concept of SNBS has further been analyzed and extended [18–21] to cover the type of dynamical properties that occur in realistic simulations of power girds, such as repeated perturbations, stochasticity and the influence of heterogeneity [22]. Basin stability has broadly been used to study collective phenomena in oscillator networks.

Probabilistic methods like SNBS have the advantage of assessing the robustness of locations in a network independent of specific individual faults. However, as they typically rely on Monte-Carlo sampling, they are also computationally challenging. Network theoretic methods in turn already found success at predicting dynamic properties such as SNBS [11, 12, 15], raising the potential to use network theoretic heuristics to identify key structural imprints and prioritize detailed fault simulations. For example, Schultz *et al* [12] predict certain nodes with low SNBS using logistic regression based on network properties as input. However, the parametrization of the structure and dynamics in real power grids is highly heterogeneous, and standard network measures are not able to accommodate a wide range of node types and properties necessary for detailed, realistic dynamic models.

Further, several network measures are not well-defined for heterogeneous systems or might not translate well from homogeneous systems. In contrast, modern machine learning (ML) is able to learn complex, nonlinear patterns from any type of raw data [23]. Hence, this work investigates the prediction of SNBS using the full graph as input.

Graph neural networks (GNNs) are a promising approach, because they are capable of predicting a variety of network measures [24, 25] and can deal with full graphs as input. Hence, GNNs can analyze full homogeneous and heterogeneous systems without further assumptions and simplifications. Therefore, we test the prediction of SNBS using GNNs. Our paper is based on a master's thesis [26] and except of this thesis, the authors are not aware of any literature using the same methods and ideas, but we introduce related work that founds on similar approaches.

Similar approaches. There are recent publications on using GNN in the context of power grids, but they do not consider the prediction of statistical dynamical properties such as SNBS. Instead, many approaches deal with the computation of power flows [27–32]. GNNs have also been used for control theory [33] and physical neural solvers have been introduced to connect GNNs with differential equations [34]. Furthermore, cascading failures were investigated in [35].

Aside from GNNs, two other publications are noteworthy to mention. Che and Cheng [36] recently published a paper in which they show the usage of active learning and relevance vector machines to reduce the computational effort of computing SNBS by learning the boundary of stable dynamics. Furthermore, Yang *et al* [37] predict the ability of power grids to synchronize after applying perturbations, but they approach the concept of dynamic stability differently. Firstly, they predict the result of single perturbations and not the statistics. Secondly, their approach is not based on providing the full graph, but they rely on common knowledge about the relation of network science and dynamic stability, e.g. by using the degree and betweenness [38] as input.

Our main contributions

(a) For the first time, SNBS is predicted based on the full graph instead of hand-crafted features. The focus lies on evaluating different learning methodologies based on GNNs for the sake of future research. The accuracy still needs to be improved for real world applications.

(b) In order to train ML-models, we generate new datasets. They are based on well-known models of synthetic power grids and on Monte-Carlo simulations to analyze dynamic stability. The datasets are rich enough to challenge ML-methods, whereas still being somewhat conceptual to connect to the existing network literature. Compared to real-world power grids, synthetic power grids have a number of advantages, for example they do not have any artifacts and one can obtain more easily large datasets, which are beneficial for statistical analyses.

(c) We also investigate transductive transfer learning capabilities by training models on small power grids and evaluating the same models on larger networks without fine-tuning.

This paper is structured as follows. Firstly, the generation of the datasets is explained. Afterward, the background knowledge for the used ML-methods is introduced, before we present the methodology of applying our ML-models to our generated datasets. Finally, the results are given and discussed, before we close with a short outlook.

2. Generation of the datasets

To analyze the capability of predicting SNBS using ML, two synthetic datasets are generated. We generate new synthetic datasets, because we are especially interested in a method that can deal with different topologies. We start by motivating the selection of our datasets. Afterward we briefly discuss relevant concepts from network science, before explaining the generation of synthetic power grids. We close by providing details about the dynamical simulations.

2.1. Objectives for datasets

High-quality datasets facilitate the application of ML-methods. Therefore, we carefully consider the following criteria for generating the datasets which mimic basic features of power grids. The datasets shall be:

(a) Homogeneous enough in both structure and dynamics to connect to network theory,
(b) Complex enough to be challenging for ML-methods,
(c) Computationally feasible using highly accurate Monte-Carlo simulations.

Firstly, homogeneity is important, because previous studies, e.g. by Nitzbon *et al* [15] have shown, that there are clear relations between dynamical stability and topological properties for somewhat homogeneous grids. As these patterns are known to exist in such homogeneous graph datasets, they are ideal to test ML systems, which can be expected to learn them.

Secondly, enough complexity is required to justify ML models. This complexity is inherently given in the problem setup, as SNBS is a highly non-linear measure. Furthermore, we consider different network topologies.

Thirdly, we need to find a compromise between computational effort and relevant properties of the datasets, such as grid size, number of grids, low statistical errors which are determined by the number of Monte-Carlo samples and low numerical errors, which depend on the dynamical solver settings. Low statistical errors are crucial to distinguish small performance differences between ML-models later on.

Prior to generating the datasets, the influence of many parameters is investigated. We shortly motivate and explain the most important parameters for the generation of the datasets. As previously mentioned, Nitzbon *et al* [15] observed interesting relations in their dataset, so we often select properties based on their investigations. Before looking at power grids in more detail, some background knowledge on graphs is needed, because power grid modeling relies on graphs.

2.2. Network science: graphs

We briefly introduce theoretical background on graphs, which is also helpful to understand GNNs later on. Graphs consist of nodes (vertices) and lines (edges) connecting two nodes. The size of a graph is given by its number of nodes N. To encode the topology of a graph one can use the adjacency matrix A which is defined by:

$$A_{ij} = \begin{cases} 1 & \text{if there is a line between nodes } i \text{ and } j, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

By using the degree which is defined by the number of neighbors of a node, we can formulate the diagonal degree matrix D. Using A and D, we can compute the graph Laplacian (L): $L = D - A$, which is a singular matrix that is a discrete analogue of the Laplace operator.

2.3. Power grids

The topology of the power grids is based on the tool Synthetic Networks [39].[5] This package uses a parametric growth process to generate networks. The resulting networks have properties that are suitable to observations of real-world power grid networks. We use the same parametrization as Nitzbon *et al* [15]: $n_0 = 1, p = 1/5, q = 3/10, r = 1/3, s = 1/10$, where n_0 is the initial number of nodes, p, q are probabilities related to constructing new lines, s the probability of splitting an existing line and r a parameter controlling the generation of redundant paths. Furthermore, half of the nodes are producers, whereas the other half are consumers. All nodes are modeled by the swing equation [41], also referred to as a second-order Kuramoto model [3, 42]. The Kuramoto model was independently introduced in the context of power grids in [43]

[5] This tool is available on Github [40].

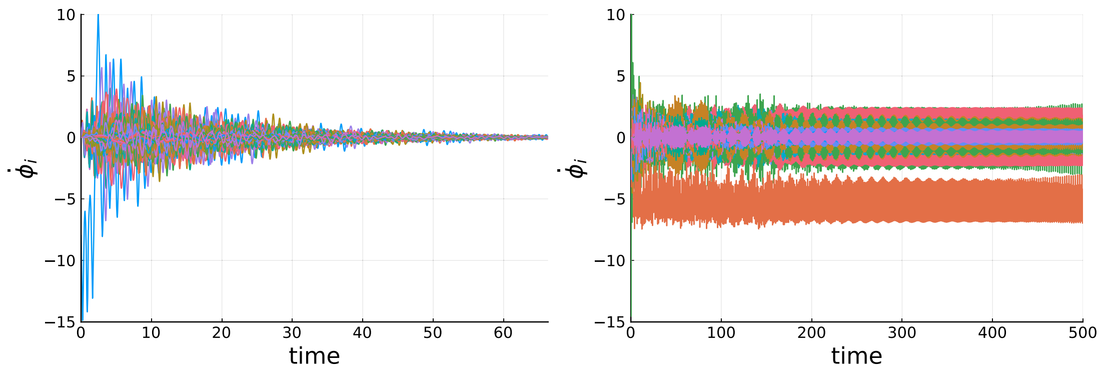

Figure 1. Exemplary trajectories of applying single-node perturbations to a grid of 100 nodes. A stable state is reached on the left and unstable state on the right after applying different single-node perturbations at different nodes. Different colors represent the trajectories at different nodes.

and has a long history of study there. We use the following notation:

$$\ddot{\phi}_i = P_i - \alpha\dot{\phi}_i - \sum_j^n K_{ij}\,\sin(\phi_i - \phi_j), \tag{2}$$

where $\phi, \dot{\phi}, \ddot{\phi}$ denotes the voltage angle and its time derivatives. We use the following parametrization: $P_i \in \{-1, 1\}$ the injected power, whereby the condition $\sum_i P_i = 0$ guarantees power balance; $\alpha = 0.1$ the damping coefficient, K is the coupling matrix based on the adjacency matrix which encodes the graph and we use uniform coupling $K_{ij} = 9A_{ij}$. The values for the injected power and the damping coefficient are the same as in [15], however we use a larger coupling (9.0 instead of 6.0) to increase the overall stability of synchronisation in the power grids and to obtain a clear bi-modal shape of the SNBS-distribution for a better balance for training ML-methods. We are interested in deviations from the nominal frequency (e.g. 50 Hz in Europe), and thus will work in frequency deviations throughout the paper. The desired state is thus $\dot{\phi}_i = 0$ at all nodes.

2.4. Dataset properties

We study the resilience of power grids operating in their synchronous state to (large) perturbations at individual nodes. The SNBS of a node is quantified as the probability that the systems returns to its synchronized state after such a network-local perturbation. Since the perturbations are drawn independently at random, SNBS is the outcome of a Bernoulli experiment [6].

To estimate SNBS for every node in a graph, $M = 10\,000$ samples of perturbations per node are constructed by sampling a phase and frequency deviation from a uniform distribution with $(\phi, \dot{\phi}) \in [-\pi, \pi] \times [-15, 15]$ and adding them to the synchronized state. Each such single-node perturbation serves as an initial condition of a dynamic simulation of our power grid model, cf equation (2). The simulation time is represented by t in seconds. At $t = 500$ the integration is terminated and the outcome of the Bernoulli trial is derived from the final state. A simulation outcome is referred to as *stable* if at all nodes $\dot{\phi}_i < 0.1$. Otherwise it is referred to as *unstable*. Two exemplary trajectories are shown in figure 1.

The classification threshold of 0.1 is chosen accounting for minor deviations due to numerical noise and slow convergence rates within a finite time-horizon. The authors are not aware of any other attractors of the Kuramoto system within that threshold. Hence, it may be assumed that every trajectory labeled as stable in that way will indeed converge to the synchronous state for $t \to \infty$. On the other hand, trajectories who are classified as unstable may converge to many different kinds of attractors [44, 45]. However, we occasionally observed so-called *long transient states* at specific nodes, which do eventually converge to the synchronous state but fail to do so before $t = 500$. While of theoretical interest, we do not expect their asymptotic behavior to play any role in real world applications and thus we are satisfied with classifying them as unstable.

A 95% confidence interval for the SNBS values may be estimated via the normal distribution approximation of the Bernoulli experiment as [46]:

$$1.96\sqrt{\frac{p(1-p)}{M}} < 0.01, \tag{3}$$

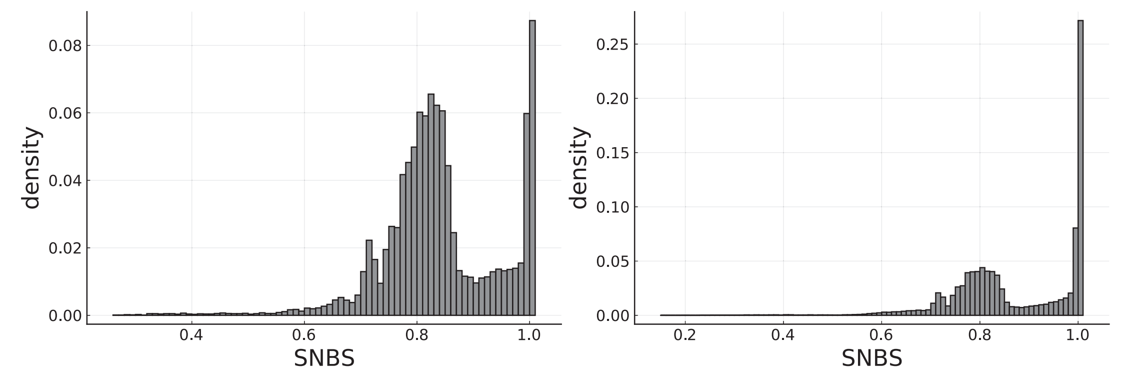

Figure 2. Histogram showing the distribution of SNBS for the datasets with 20 nodes (left) and 100 nodes (right). The distributions are normalized so that bin heights sum to 1.

where the inequality is obtained by setting $p = 0.5$ and $M = 10\,000$.

The distributions of SNBS for both datasets are given in figure 2. We refer to the dataset consisting of grids of 20 nodes per grid as *dataset*20, and to the dataset consisting of grids with 100 nodes as *dataset*100. For both cases, there is a bi-modal distribution of SNBS over the whole data set, which facilitates ML-models to learn the distinction between those modes. The peak at 1.0 indicates a large amount of nodes where no perturbation has an adverse effect on the synchronisation. The second peak can be interpreted in a way that many nodes are somewhat resistant to perturbations and the grid stays synchronised in about 80% when applying perturbations at the particular nodes. In case of dataset20 the mean value of SNBS is 0.84 and for dataset100 nodes it is 0.87. In both datasets, the number of unstable outcomes is low, which is a property we expect to hold for real power grids as well. Conducting the computation of the dynamic stability using one CPU takes about 45 h per grid in case of 100 nodes per grid and about 3 h in case of 20 nodes.

3. Graph neural networks

This section briefly introduces GNNs. We begin with a general framework for GNNs and subsequently summarize the recent development of GNNs. GNN are a class of artificial neural networks (ANNs) designed to learn relationships of graph-structured data. Just as ANNs they have internal weights, which can be fitted in order to adapt their behavior to the given task. In the case of supervised learning these weights are adjusted such that the error between the estimated output and the labeled output for given input data is minimized. As inputs GNNs use the graph structure and potentially node features. Their output can either be global graph attributes, attributes of sub-graphs, or local node properties. Different types of GNNs have been introduced, some of which are detailed below. In [47], the authors introduce a *design space* for GNNs as a common framework to facilitate understanding and comparison of the different methods. In their design space, GNNs consist of pre-processing, message-passing and post-processing layers. GNN architectures vary in layer number and connectivity, as well as the intra-layer design of the message-passing layers. [47] view message-passing layers as combinations of (i) message computation and (ii) aggregation. First, a message function computes a message for each node from it is current state. Secondly, the messages are aggregated over the neighborhood to a new node state. Both message computation and aggregation can be realized in different ways. Common ML-methods such as *batch normalization* [48] or *dropout* [49] can be added to stabilize training. The application of non-linear activation functions enables GNNs to learn non-linear relations in the graph data. In this work we focus on convolutional GNNs and in particular on those employing spatial-based graph convolutions, because they can be applied to varying topologies, as we have in our datasets.

Graph convolutions are based on the concept of the graph Fourier transform (FT), a generalization of the classical FT, which enables the remarkable success of convolutional neural networks (CNNs) in image recognition. Unlike the classical FT, which uses exponential shifts the graph FT corresponds to an expansion of the function on the graph in terms of the eigenvectors of the graph Laplacian L. Such an expansion may in turn be multiplied with a function of the graphs eigenvalues, a so-called spectral filter. While it is possible to learn spectral filters from training data, they lack many of the nice properties of the convolution kernels used in CNNs: they are not localized in node space, computing the eigenbasis is expensive and trained models can not be evaluated on different graphs, since each graph has a unique spectrum.

An important insight of [50] was that graph spectral filters can be approximated by polynomials of the graphs' adjacency matrix A, thus achieving a localization of the filter in the (kth order) neighborhoods of the nodes. Subsequently, in their seminal paper Kipf and Welling [51] realized that it suffices to consider only the linear term of the polynomial expansion, corresponding to a simple multiplication of the node features with the (renormalized) adjacency matrix. They arrived at a computationally efficient and powerful layer architecture that relies only on local information and generalizes well to different graphs. Several GNN models that we investigate in this paper were derived from their so-called graph convolutional layer (GCN):

$$H = \sigma(\overline{A}X\Theta), \tag{4}$$

where H denotes the output of a layer, σ is the activation function, X are the input features, Θ is a matrix containing the learnable weights and $\overline{A}$ is the renormalized adjacency matrix, given by $\overline{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$. Further $\tilde{A} = A + I$, where I is the identity matrix, denotes an adjacency matrix with added self-loops and the diagonal degree matrix $\tilde{D}$ is determined by: $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. In the design space of [47], $X\Theta$ manifest the message computation, while $\overline{A}$ realizes the aggregation. By consecutively applying multiple GCN-layers, not only direct neighbors are taken into account, but also neighbors at further distance.

Instead of stacking multiple GCN-layers, Wu *et al* [52] removed the activation functions, combined all weight matrices into one and computed $\tilde{A}^i$ to obtain:

$$H = \mathrm{softmax}(\overline{A}^i X\Theta). \tag{5}$$

This layer founds on their assumption that the nonlinearity between GCN layers is not crucial and may be omitted in order to reduce computational effort. We refer to this layer as simple-graph-convolution (SG).

Du *et al* [53] used multiple exponents i of $\tilde{A}$ within one layer according to the following scheme:

$$H = \sum_{z=0}^{Z} D^{-\frac{1}{2}}A^z D^{-\frac{1}{2}}X\Theta_z. \tag{6}$$

This layer type is called topology adaptive graph convolution (TAG), which refers to its ability of considering different topologies. However, this is the case for all methods that are introduced in this paper. This architecture provides an extension to GCNs by incorporating information about higher order neighborhoods within one layer.

Auto-regressive moving average (ARMA) neural network layers by Bianchi *et al* [54] are far-reaching generalizations of GCN layers. They are derived from a rational expansion of the spectral filter instead of a polynomial expansion. A complete ARMA-layer consists itself of multiple graph convolutional skip (GCS) layers:

$$\overline{X}^{(j+1)} = \sigma(\tilde{L}X^{(j)}W^{(j)} + XV^{(j)}), \tag{7}$$

where j is an index and W and V are matrices of trainable parameters. There are two important distinctions from the GCN layers: the aggregation in the first term uses normalized Laplacian $\tilde{L} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, instead of $\tilde{A}$. Additionally, the connectivity of the message-passing layers is augmented with a skip connection, implemented in the second term. It recursively re-inserts the initial node features X from the first layer and thus enables stacking a large number of GCS layers, whereas preventing the loss of the initial information due to Laplacian smoothing. In order to reduce the computational effort and to reduce overfitting, the weights among different GCS layers are shared: $W^{(j)} = W$ and $V^{(j)} = V$, except for the first layer where $W^{(1)} \neq W$.

To increase their expressive power multiple ARMA layers may be combined in a parallel stack:

$$\overline{X} = \frac{1}{K}\sum_{k=1}^{K} \overline{X}_k^{(J)}, \tag{8}$$

where $\overline{X}_k^{(J)}$ is the output of the last GCS layer in the kth ARMA layer. We can also interpret J as the number of possible hops and by increasing J larger regions are taken into account. ARMA filters with their recursive and distributed formulation, are efficient to train and capable of learning complex information. All of the layers described above are used in the models introduced in the next section.

4. Prediction of SNBS using GNNs

To predict SNBS of all nodes, we use a node-regression setup, by providing the adjacency matrix of the graph and the injected power per node P_i as inputs. The process is shown in figure 3. In order to test the

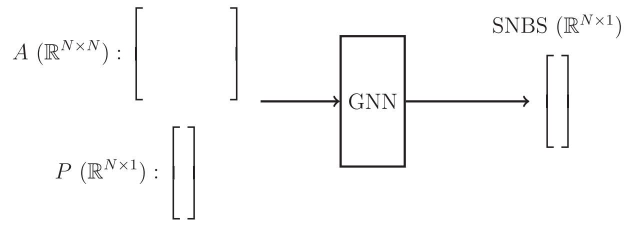
A $(\mathbb{R}^{N\times N})$:
P $(\mathbb{R}^{N\times 1})$:
GNN
SNBS $(\mathbb{R}^{N\times 1})$

Figure 3. Prediction of SNBS using GNN. The GNN takes the adjacency matrix A and the injected power P as input to obtain the nodal SNBS as output. Hence, the prediction of SNBS does not consider individual faults or any other variables, but operates only on topological properties of the power grid.

Table 1. Properties of datasets.

Name	Number of grids	Number of nodes per grid	$\overline{\text{SNBS}}$
dataset20	1000	20	0.8398
Train	800	20	0.8407
Test	200	20	0.8365
dataset100	1000	100	0.8737
Train	800	100	0.8730
Test	200	100	0.8768

Table 2. Properties of models. Number of parameters denotes the number of learnable weights of the model.

Name	Type of convolution	Number of layers	Number of parameters	Maximum number of hops
ArmaNet1	ARMA	1	38	4
ArmaNet2[a]	ARMA	2	1050	8
GCNNet1	GCN	2	15	2
GCNNet2	GCN	3	107	3
GCNNet3[a]	GCN	3	149	3
SGNet1	SG	1	4	2
SGNet2	SG	2	15	4
TAGNet1	TAG	2	39	6

[a]There is a batch normalization between first and second layer.

performance of our models on unseen data, we split the datasets into training and testing sets. The shift between them is marginal as can be seen in table 1.

4.1. Setup of our GNN-models

Based on the introduced GNN layers, eight GNN-models are analyzed to evaluate the performance of different architectures. GNNs are capable of reading in the full graph without any simplifications. We also tried to use CNNs which are well known from image analysis. In case of CNNs, the graph information is provided by using a modified version of the adjacency matrix as input, but the setup had several limitations in comparison to the GNNs. The application of CNNs is shown in appendix C. In table 2 the GNN-models are briefly introduced. All models use one type of graph convolutional layer, but may use several numbers of them and all have one linear and one sigmoid layer at the end. Additionally, dropout is used in several cases, cf appendix B. We did not do a systematic investigation of hyperparameters such as number of layers and their properties, but focused on identifying relevant factors to enable training.

4.2. Training setup

For all models the same parameters are used and the training consists of 500 epochs. To enable reproducibility, the seeds are set before training and can be found in the published source code[6]. The training is based on the library Pytorch [55] and for the graph handling and graph convolutional layers the additional library PyTorch geometric [56] is used. For the training of the models, CPUs are used and depending on the model training takes between 20 and 50 min on either Haswell or Broadwell architecture

[6] Information regarding the full source code is given in appendix A.

Table 3. Results represented by R^2 score in %[a].

Model	dataset20	dataset100	tr20ev100
ArmaNet1	18.8	15.4	3.60
ArmaNet2	**39.5**	**45.4**	**23.7**
GCNNet1	8.10	5.98	−3.22
GCNNet2	24.3	22.1	13.2
GCNNet3	9.02	6.71	−0.67
SGNet1	7.12	3.98	−9.15
SGNet2	13.5	13.0	1.67
TAGNet1	29.1	28.8	13.7

[a]For dataset20 and dataset100, the models are both trained on their training and evaluated on their test sections. To evaluate the transfer learning capabilities, we use the term *tr*20*ev*100 meaning that the model is trained on the dataset20, but evaluated on the dataset100.

without parallelization. The detailed training parameters, e.g. batch sizes and additional information on the computational effort are given in appendix B. As loss function we use the mean squared error[7].

5. Results

To evaluate the performance of different models, the R^2 score, which may also be known as coefficient of determination and a self-defined discretized accuracy is used. The score R^2 is computed by:

$$R^2 = 1 - \frac{\mathrm{mse}(y,t)}{\mathrm{mse}(t_{\mathrm{mean}},t)}, \tag{9}$$

where mse denotes the mean squared error, y the output of the model, t the target value and t_{mean} the mean of all considered targets of the test dataset. The standard measure of performance is R^2, which captures the mean square error relative to a null model that predicts the mean of the test-dataset for all points. A constant model that always predicts t_{mean}, disregarding the input features, would get a score of $R^2 = 0.0$. The R^2-score is used to measure the portion of explained variance in a dataset. To further simplify interpretation, we rephrase the evaluation as a classification problem.

The outputs are categorized as true or false by using a threshold and we compute the accuracy as:

$$\text{discretized accuracy} = \frac{\text{correct predictions}}{\text{number of samples}}. \tag{10}$$

We refer to this self-defined accuracy as *discretized accuracy*. Predictions are considered to be correct, if the predicted output y is within a certain threshold to the target value t: $y - t <$ threshold. We set this threshold to 0.1, because this is small enough to differentiate between the modes in the distributions (see figure 2). Furthermore, a total deviation of the prediction and true output of 0.1 should be efficient for most applications. The discretized accuracy depends on the distribution of SNBS, so it can not be used for comparison across different datasets, but has to be compared to the null model of the corresponding dataset.

Since there is no previous work that can be easily compared to our methods, we introduce a simple baseline model. This baseline model always predicts the average value of the testing set. By design, this results in $R^2 = 0$, and achieves a discretized accuracy of 67.1% on dataset20 and of 40.9% on dataset100. The differences in discretized accuracy are rooted in the different distributions of the two datasets (cf figure 2).

We use an averaged performance to reduce the impact of the initialization effects. Out of five different initializations per training setup, only the best three are considered to compute an averaged performance. The average R^2-performance is given in table 3 and for the discretized accuracy in table 4. The best values are in bold. The training progress of the best model is shown in figure 4. The fluctuations, especially visible at the bottom right in figure 4 are typical for ML applications when using stochastic gradient descent (SGD) and constant learning rates during training.

Furthermore, we investigate whether the features learned by GNNs generalize to grids of different sizes. As datasets of large grids are costly to create, successful pre-training on smaller grids with subsequent application on larger grids would be a valuable strategy. To evaluate the transfer learning capabilities, we train GNN-models on the small dataset of grids with 20 nodes and evaluate without fine-tuning on the dataset with large grids of 100 nodes. As performance of the transductive transfer learning, we report the R^2

[7] Corresponds to MSELoss in Pytorch.

Table 4. Results represented by discretized accuracy in %[a].

Model	dataset20	dataset100	tr20ev100
ArmaNet1	76.5	65.1	56.0
ArmaNet2	**80.5**	**85.0**	**65.9**
GCNNet1	69.5	66.6	47.8
GCNNet2	79.8	67.5	59.8
GCNNet3	71.6	63.7	49.5
SGNet1	67.9	67.8	46.0
SGNet2	70.5	65.9	48.7
TAGNet1	78.8	69.6	56.1

[a]For dataset20 and dataset100, the models are both trained on their training and evaluated on their test sections. To evaluate the transfer learning capabilities, we use the term *tr*20*ev*100 meaning that the model is trained on the dataset20, but evaluated on the dataset100.

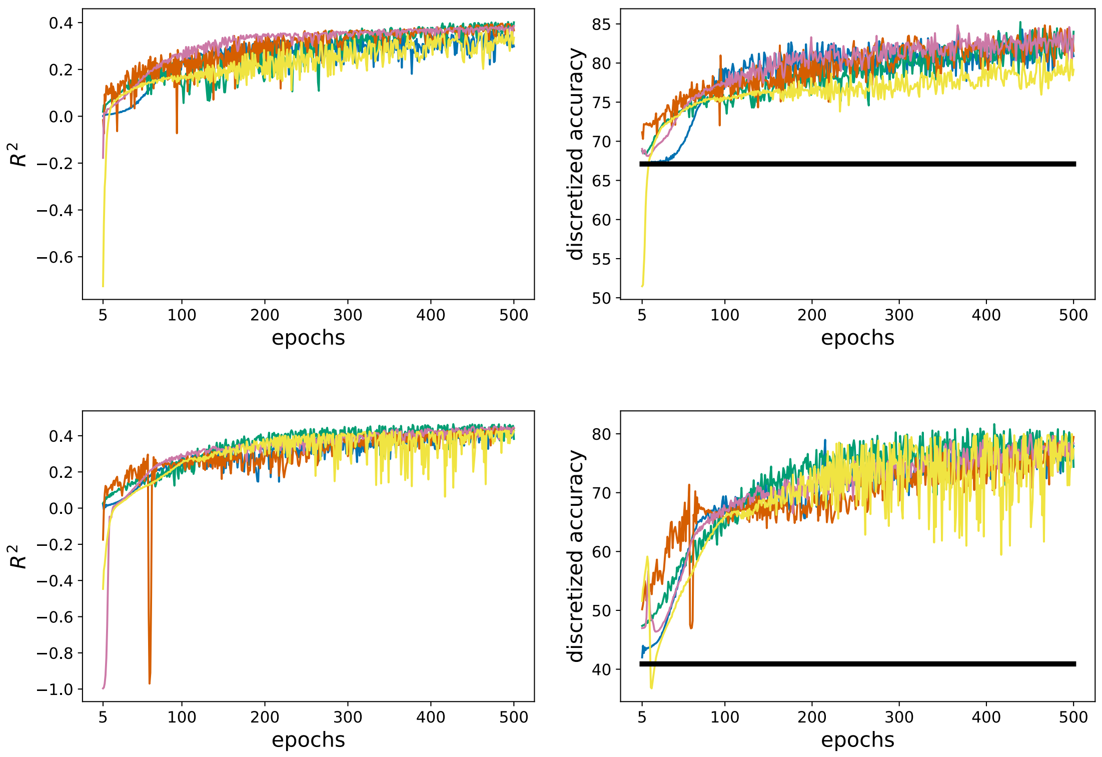

Figure 4. Training results for ArmaNet2 and dataset20 at the top and dataset100 at the bottom. The R^2-score is shown on the left and the test discretized accuracy on the right. Different colors show different initializations and the horizontal line for the discretized accuracy is based on a toy model that always predicts $\overline{\text{SNBS}}$. The evaluation is purely based on the test dataset.

and accuracy on the large target dataset using the term *tr*20*ev*100 (trained on dataset20, evaluated on dataset100).

The results show that the prediction of SNBS using GNNs is feasible and different models have a large impact. We did not perform a detailed hyperparameter study of different GNN-models, so conclusions about their performance are tentative for now. Next, we shortly summarize our observations. The results indicate that increasing the complexity of the model can be beneficial, as the model ArmaNet2 with the largest amount of parameters (1050) performs best. However, increasing the complexity is not always helpful. GCNNet3 for example performs worse than GCN2, even though having more learnable parameters (149 instead of 107). The meaning of the type of convolution is underlined by considering TAGNet1 and ArmaNet1, because TAGNet1 outperforms ArmaNet1 with only slightly more parameters than ArmaNet1. Figure 5 shows the relation of the complexity and performance based on dataset100. The complexity is firstly represented by the number of learnable parameters on a logarithmic scale and secondly by the maximum number of possible hops. By hops we mean the order of neighbors that are taken into account. For example, one hop means that only direct neighbors are considered, whereas two means that nodes are considered which are not directly connected, but via one direct neighbor.

Without conducting ablation studies, we can only guess reasons for the superiority of ArmaNet2. We suspect two main reasons: firstly, the largest number of parameters could be decisive; secondly, the most

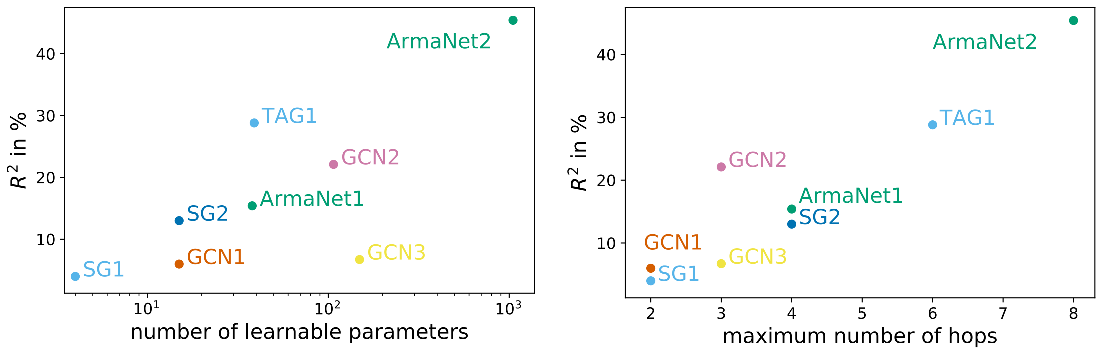

Figure 5. Relation of performance and the complexity of models represented by the number of learnable parameters on the left and the number of maximum hops on the right. The plotted results are based on dataset100.

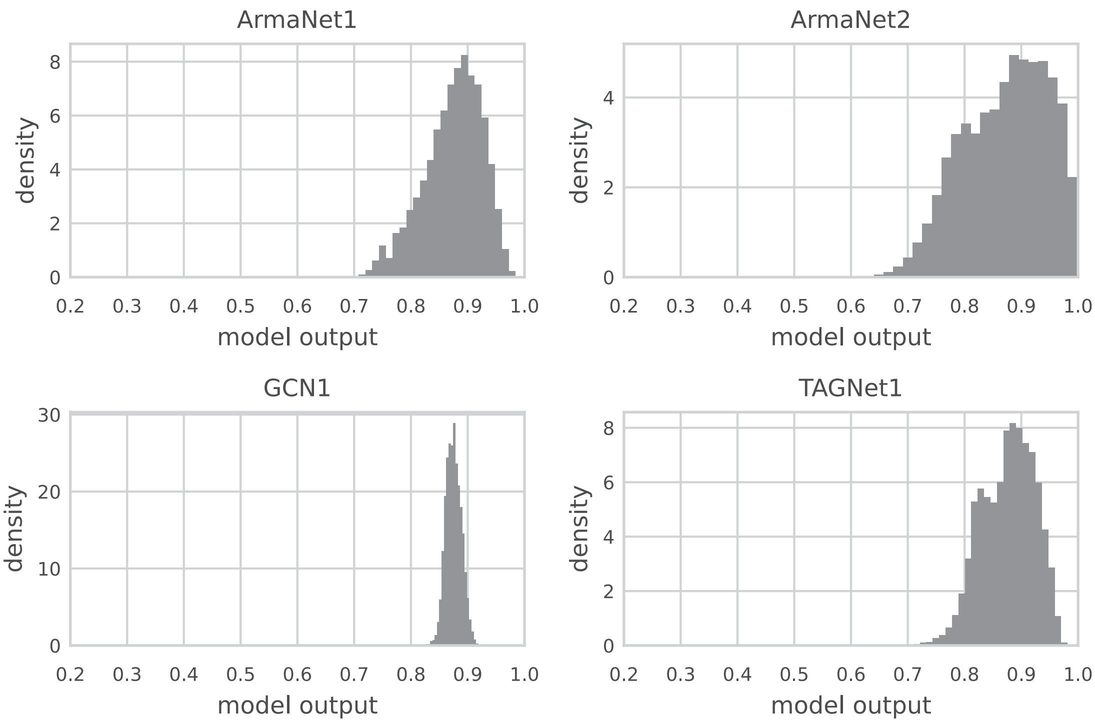

Figure 6. Histograms showing density of predicted outputs for different models and dataset100 and the best seed per model.

complex architecture including skip layers to consider neighbors of higher degrees could have a positive impact. The four GCS-layers of ArmaNet2 can consider a relatively large region. TAGNet1 also performs well and this model can evaluate neighbors of 6th-order, by having two layers and three hops per layer. The benefit of ArmaNet2 can be emphasized by investigating tr20ev100, because ArmaNet2 outperforms all other models on dataset100, even if it is purely trained on dataset20. Hence, the models ArmaNet2 results in the most robust setup.

To further evaluate the performance of the investigated models, we analyze the distribution of the output of selected models in figure 6. Therefor, we only consider the output based on the best seed per model using R^2 as a criterion. The output of all models is restricted to somewhat large values and neither low nor very high values of SNBS can be predicted. The small amount of nodes with low SNBS in the dataset might explain the absence of low output values. In case of large output values, it is remarkable and a bit surprising that none of the models predicts the abundance of completely stable outcomes. This behavior limits the applicability to real world problems. The limitation of all models also becomes clear when comparing the results to the distributions introduced in figure 2. Since the shifts[8] within the datasets are

[8] A dataset shift means, that training and testing datasets are different.

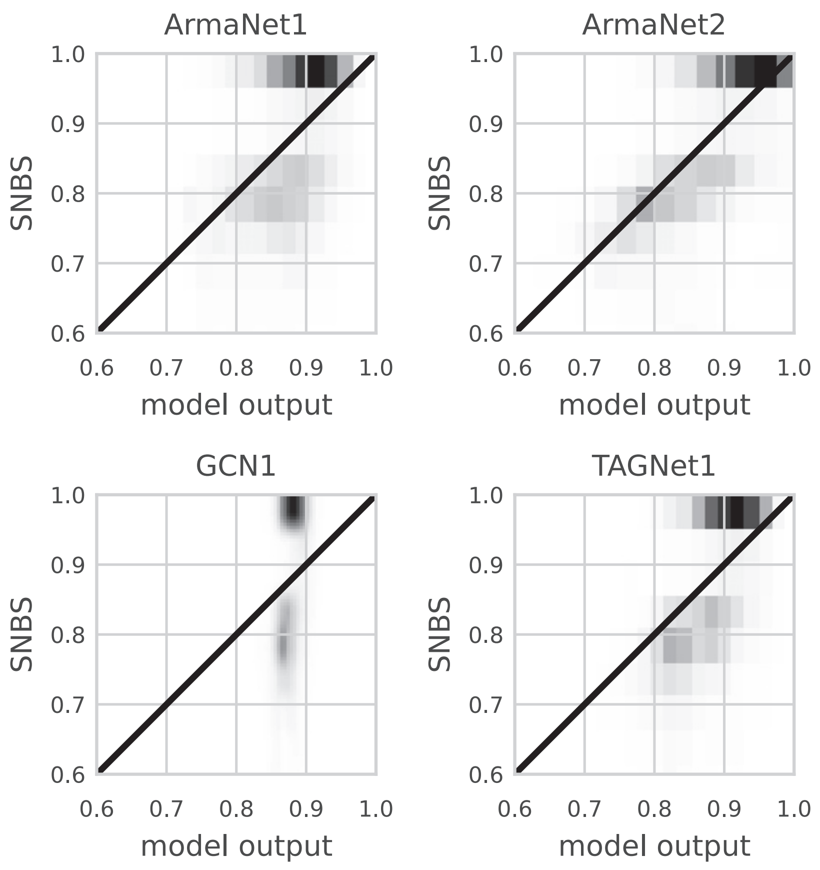

Figure 7. Heat maps of comparing models using the best seed for each of them and considering the predicted output vs SNBS and investigating dataset100. The diagonal represents a potential perfect model ($R^2 = 1$).

small, we can compare the output distributions to the distributions of the entire datasets, even though figure 6 only considers the test section.

The distributions of the output (figure 6) also indicate performance differences between the models. We clearly see that GCN1, having a relatively low performance, has a very limited range of output values and all values are around the mean of the dataset. ArmaNet1 already has a wider range, whereas ArmaNet2 has the largest range. Besides the range, the shape of the distribution and modalities of the predictions are also telling, e.g. we find an indication for a bimodal distribution in case of TAGNet1. All in all, the superiority of ArmaNet2 and TAGNet1 becomes visible. However, even for those models the output is still limited to values that are larger than 0.6 and there is only a small amount of predictions of high stability (SNBS $\approx$ 1).

To visually analyze the models, we plot the predicted output vs SNBS in heat maps in figure 7. Perfect predictions would be on the diagonal only, similarly to $R^2 = 1$. On the contrary to R^2 shown in table 3, we can find some reasons for the performance differences. We see that ArmaNet2 and TAGNet1 can distinguish between nodes with SNBS $\approx$ 1 and nodes with lower SNBS. Other models, such as GCN1, have large regions on the off-diagonal, resulting in a lower performance.

6. Conclusion and outlook

The key result of this paper is a novel approach of estimating SNBS via GNNs. We have demonstrated its potentials and have paved the way for further investigations. We show the necessity to use well-adapted architectures for this problem, since generic CNNs are not able to achieve comparable results even with more parameters (cf appendix C).

The strongest limitation of the presented results are probably the assumptions for generating the datasets which matches several properties of real power grids, but it also simplifies some aspects, e.g. missing heterogeneity of nodes (power input) and lines (coupling constant). However, the accuracy can still be increased before moving to more realistic setups, because the performance is still too low for real applications. We provide several ideas for improvements in the next paragraphs.

Since we see substantially improved performance for models with larger number of parameters testing more complex models seems very promising. More complex models might identify other relevant structures of networks to predict SNBS more accurately, there is no suggestion that the performance is already saturating. As a first step, one could conduct a hyperparameter study to improve the investigated models.

In further steps, one could introduce new models to increase the performance. Firstly, new layers could be designed that specifically aim to predict SNBS and deal with power grids. Secondly, hybrid approaches might be used that incorporate knowledge about known structures, e.g. network motifs that can hardly be recognized by GNNs. Generally it is clear from our results that more complex architectures are promising for this task, even if it remains unclear exactly what direction the complexity increase should point toward.

Another key for improvement are the datasets. The used datasets are relatively small, so increasing the size of the datasets might be an important step for training more complex models. To solve the issue of the limited range of outputs and the observation that the model outputs are around the mean of the datasets, balancing or weighting of samples might help.

Remarkably, we successfully showed that GNNs can generalize across different sizes of power grids. Another avenue for future research is to train models based on different sizes to start with. It is feasible that the overall performance can be increased when actually training the models on multiple datasets. The capability of training models on smaller grids and applying them on larger grids can become crucial for real-world applications to reduce the computational effort of generating datasets and also of training the models.

Acknowledgments

All authors gratefully acknowledge the European Regional Development Fund (ERDF), the German Federal Ministry of Education and Research, and the Land Brandenburg for supporting this project by providing resources on the high-performance computer system at the Potsdam Institute for Climate Impact Research. The authors also thank the Chair of Information Management in Mechanical Engineering of RWTH Aachen University for computational resources. Christian Nauck would like to thank the German Federal Environmental Foundation (DBU) for funding his PhD scholarship and Professor Raisch from Technical University Berlin for supervising his PhD Michael Lindner greatly acknowledges support by the Berlin International Graduate School in Model and Simulation based Research (BIMoS) of TU Berlin. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—KU 837/39-1/RA 516/13-1. The publication was supported by the DFG funding program Open Access Publication Funding.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: 10.5281/zenodo.5148085.

Appendix A. Source code

The full source code including the dataset is available at https://zenodo.org/record/5148085. Furthermore, the scripts are also given at Github https://github.com/PIK-ICoNe/paper-companion_predicting-snbs-using-gnn. The code for the computation of SNBS is written in Julia [57] and the dynamic simulations rely on the package DifferentialEquations.jl [58]. For simulating more realistic power grids in future work we recommend the additional use of NetworkDynamics.jl [59] and PowerDynamics.jl [60]. Software packages used for ML-applications are listed in table A1.

Table A1. Software packages for ML.

Package	Version	Package	Version
Cuda	10.2	Torch-cluster	1.5.9
h5py	2.10.0	Torch-geometric	1.7.0
Numpy	1.19.2	Torch-scatter	2.0.6
Pandas	1.2.4	Torch-sparse	0.6.9
Python	3.8.5	Torch-spline-conv	1.2.1
Pytorch	1.7.1	Torchvision	0.8.2

Appendix B. Model and training details

In this section details about the used models are provided, whereby NIC denotes the number of input channels, NOC the number of output channels, NOH the number of hops per layer (for TAGConv and SGConv) and ReLU the rectified linear unit activation function. Recall that we only have one node feature as input and hence our most common choice for NIC is 1.

(a) *ArmaNet*1:
   1. Arma-convolution (NIC = 1, NOC = 1, num-stacks = 3, num-layers = 4, ReLU activation, weights are not shared between layers),
   2. Fully connected layer and sigmoid output layer.

(b) *ArmaNet*2:
   1. Arma-convolution (NIC = 1, NOC = 16, num-stacks = 3, num-layers = 4, dropout = 0.25, ReLU activation, weights are shared between layers),
   2. Batch normalization, ReLU and dropout,
   3. Arma-convolution (NIC = 16, NOC = 1, num-stacks = 3, num-layers = 4, dropout = 0.25, no activation function, weights are shared between layers),
   4. Fully connected layer and sigmoid output layer.

(c) *GCNNet*1:
   1. GCN-convolution (NIC = 1, NOC = 4),
   2. ReLU and dropout,
   3. GCN-convolution (NIC = 4, NOC = 1),
   4. Fully connected layer and sigmoid output layer.

(d) *GCNNet*2:
   1. GCN-convolution (NIC = 1, NOC = 16),
   2. ReLU and dropout,
   3. GCN-convolution (NIC = 16, NOC = 4),
   4. ReLU,
   5. GCN-convolution (NIC = 4, NOC = 1),
   6. Fully connected layer and sigmoid output layer.

(e) *GCNNet*3:
   1. GCN-convolution (NIC = 1, NOC = 16),
   2. Batch normalization, ReLU and dropout,
   3. GCN-convolution (NIC = 16, NOC = 4),
   4. Batch normalization, ReLU
   5. GCN-convolution (NIC = 4, NOC = 1),
   6. Fully connected layer and sigmoid output layer.

(f) *SGNet*1:
   1. SG-convolution (NIC = 1, NOC = 1, NOH = 2),
   2. ReLU, fully connected layer and sigmoid output layer.

(g) *SGNet*2:
   1. SG-convolution (NIC = 1, NOC = 4, NOH = 2),
   2. ReLU, dropout,
   3. SG-convolution (NIC = 4, NOC = 1, NOH = 2),
   4. Fully connected layer and sigmoid output layer.

(h) *SGNet*3:
   1. SG-convolution (NIC = 1, NOC = 16, NOH = 2),
   2. ReLU, dropout,
   3. SG-convolution (NIC = 16, NOC = 4, NOH = 2),
   4. ReLu,
   5. SG-convolution (NIC = 4, NOC = 1, NOH = 2),
   6. Fully connected layer and sigmoid output layer.

(i) *TAGNet*1:
   1. TAG-Convolution (NIC = 1, NOC = 4, NOH = 3),

Table B1. Parameters for ML.

Parameter	Property	Parameter	Property
Training batchsize	100	Test batchsize	200
Trainig set index	1–800	Test set index	801–1000
Train set shuffle	True	Test set shuffle	False
Optimizer	SGD	Learning rate	0.3
Momentum	0.9	Weight decay	1×10^{-9}
Criterion	MSELoss	Threshold for discretized accuracy	0.1

2. ReLU, dropout,
3. TAG-Convolution (NIC = 4, NOC = 1, NOH = 3),
4. Fully connected layer and sigmoid output layer.

The used training parameters can be found in table B1. The scripts include seeds for *torch, cuda and numpy.random*, even though cuda may not be used.

Appendix C. Convolutional neural networks

We used CNNs taking the graph Laplacian and information about the node type (source/sink) as input. The power is either added in an additional row to L, resulting in a one-input-channel setup (1C) or concatenated as a second input channel (2C).

The results are given in table C1. The input formats (1C and 2C) do not have a large impact. All CNNs achieve comparable performance and outperform low performing GNN-models, but they are not competitive to the best GNN-models.

Table C1. Results when using CNNs.

	R^2 score in %				Discretized accuracy in %			
	dataset20		dataset100		dataset20		dataset100	
Model	1C	2C	1C	2C	1C	2C	1C	2C
ResNet18	16.1	14.1	20.5	20.2	78.4	78.2	69.3	69.3
ResNet34	17.4	16.6	21.2	20.6	79.0	78.6	69.2	69.2
ResNet50	16.6	15.2	20.7	20.7	78.2	78.1	69.3	69.3

References

[1] United Nations 2015 Paris agreement *Paris: 21st Conf. Parties* https://unfccc.int/sites/default/files/english_paris_agreement.pdf

[2] Anvari M, Hellmann F and Zhang X 2020 Introduction to focus issue: dynamics of modern power grids *Chaos* **30** 063140

[3] Kuramoto Y 1975 Self-entrainment of a population of coupled non-linear oscillators *Mathematical Problems in Theoretical Physics* vol 39 pp 420–2

[4] Acebrn J A, Bonilla L L, Prez Vicente C J, Ritort F and Spigler R 2005 The Kuramoto model: a simple paradigm for synchronization phenomena *Rev. Mod. Phys.* **77** 137–85

[5] Rodrigues F A, Peron T K D, Ji P and Kurths J 2016 The Kuramoto model in complex networks *Phys. Rep.* **610** 1–98

[6] Menck P J, Heitzig J, Marwan N and Kurths J 2013 How basin stability complements the linear-stability paradigm *Nat. Phys.* **9** 89–92

[7] Liu Z and Zhang Z 2017 Quantifying transient stability of generators by basin stability and Kuramoto-like models *2017 North American Power Symposium (NAPS)* pp 1–6

[8] Liu Z, He X, Ding Z and Zhang Z A basin stability based metric for ranking the transient stability of generators *IEEE Trans. Ind. Inf.* **15** 1450–1459

[9] Rakshit S, Bera B K, Majhi S, Hens C and Ghosh D 2017 Basin stability measure of different steady states in coupled oscillators *Sci. Rep.* **7** 45909

[10] Majhi S, Ghosh D and Kurths J 2019 Emergence of synchronization in multiplex networks of mobile Rössler oscillators *Phys. Rev.* E **99** 012308

[11] Menck P J, Heitzig J, Kurths J and Joachim Schellnhuber H 2014 How dead ends undermine power grid stability *Nat. Commun.* **5** 3969

[12] Schultz P, Heitzig J and Kurths J 2014 Detours around basin stability in power networks *New J. Phys.* **16** 125001

[13] Kim H, Lee S H and Holme P 2015 Community consistency determines the stability transition window of power-grid nodes *New J. Phys.* **17** 113005

[14] Kim H, Lee S H and Holme P 2016 Building blocks of the basin stability of power grids *Phys. Rev.* E **93** 062318

[15] Nitzbon J, Schultz P, Heitzig J, Kurths J and Hellmann F 2017 Deciphering the imprint of topology on nonlinear dynamical network stability *New J. Phys.* **19** 033029

[16] Kim H, Lee S H, Davidsen J and Son S-W 2018 Multistability and variations in basin of attraction in power-grid systems *New J. Phys.* **20** 113006
[17] Kim H, Lee M J, Lee S H and Son S-W 2019 On structural and dynamical factors determining the integrated basin instability of power-grid nodes *Chaos* **29** 103132
[18] Schultz P, Hellmann F, Webster K N and Kurths J 2018 Bounding the first exit from the basin: independence times and finite-time basin stability *Chaos* **28** 043102
[19] Ji P, Lu W and Kurths J 2018 Stochastic basin stability in complex networks *Europhys. Lett.* **122** 40003
[20] Lindner M and Hellmann F 2019 Stochastic basins of attraction and generalized committor functions *Phys. Rev.* E **100** 022124
[21] Hellmann F, Schultz P, Jaros P, Levchenko R, Kapitaniak T, Kurths J and Maistrenko Y 2020 Network-induced multistability through lossy coupling and exotic solitary states *Nat. Commun.* **11** 592
[22] Wolff M F, Lind P G and Maass P 2018 Power grid stability under perturbation of single nodes: effects of heterogeneity and internal nodes *Chaos* **28** 103120
[23] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* (Cambridge, MA: MIT Press) http://deeplearningbook.org
[24] Avelar P, Lemos H, Prates M and Lamb L 2019 Multitask learning on graph neural networks: learning multiple graph centrality measures with a unified network *Artificial Neural Networks and Machine Learning ICANN 2019: Workshop and Special Sessions* (Lecture Notes in Computer Science) ed I V Tetko, V Kůrková, P Karpov and F Theis (Cham: Springer International Publishing) pp 701–15
[25] Maurya S K, Liu X and Murata T 2019 Fast approximations of betweenness centrality with graph neural networks *Proc. 28th ACM Int. Conf. Information and Knowledge Management, CIKM '19* (New York, NY, USA: Association for Computing Machinery) pp 2149–52
[26] Nauck C, Isenhardt I, Zhang H, Hellmann F and Ennen P 2020 Prediction of power grid vulnerabilities using machine learning *Master's Thesis, Masterarbeit* Rheinisch-Westflische Technische Hochschule Aachen
[27] Donon B, Donnot B, Guyon I and Marot A 2019 Graph neural solver for power systems *Proc. Int. Joint Conf. Neural Networks 2019* (Institute of Electrical and Electronics Engineers Inc.)
[28] Kim C, Kim K, Balaprakash P and Anitescu M 2019 Graph convolutional neural networks for optimal load shedding under line contingency *2019 IEEE Power Energy Society General Meeting (PESGM)* pp 1–5
[29] Bolz V, Rue J and Zell A 2019 Power flow approximation based on graph convolutional networks *2019 18th IEEE Int. Conf. Machine Learning and Applications (ICMLA)* pp 1679–86
[30] Retire N, Ha D T and Caputo J-G 2020 Spectral graph analysis of the geometry of power flows in transmission networks *IEEE Syst. J.* **14** 2736–47
[31] Wang D, Zheng K, Chen Q, Luo G and Zhang X 2020 Probabilistic power flow solution with graph convolutional network *2020 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)* pp 650–4
[32] Owerko D, Gama F and Ribeiro A 2020 Optimal power flow using graph neural networks *ICASSP 2020—2020 IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)* pp 5930–4
[33] Gama F, Tolstaya E and Ribeiro A 2020 Graph neural networks for decentralized controllers (arXiv:2003.10280)
[34] Misyris G S, Venzke A and Chatzivasileiadis S 2020 Physics-informed neural networks for power systems *2020 IEEE Power Energy Society General Meeting (PESGM)* pp 1–5
[35] Liu Y, Zhang N, Wu D, Botterud A, Yao R and Kang C 2021 Searching for critical power system cascading failures with graph convolutional network *IEEE Trans. Control Netw. Syst.* **8** 1304–13
[36] Che Y and Cheng C 2021 Active learning and relevance vector machine in efficient estimate of basin stability for large-scale dynamic networks *Chaos* **31** 053129
[37] Yang S-G, Kim B J, Son S-W and Kim H 2021 Power-grid stability predictions using transferable machine learning (arXiv:2105.07562 [physics])
[38] Freeman L C 1977 A set of measures of centrality based on betweenness *Sociometry* **40** 35
[39] Schultz P, Heitzig J and Kurths J 2014 A random growth model for power grids and other spatially embedded infrastructure networks *Eur. Phys. J. Spec. Top.* **223** 2593–610
[40] Schultz P 2020 luap-pik/SyntheticNetworks https://github.com/luap-pik/SyntheticNetworks
[41] Filatrella G, Nielsen A H and Pedersen N F 2008 Analysis of a power grid using a Kuramoto-like model *Eur. Phys. J.* B **61** 485–91
[42] Kuramoto Y 2005 Self-entrainment of a population of coupled non-linear oscillators *Int. Symp. Mathematical Problems in Theoretical Physics*
[43] Bergen A R and Hill D J 1981 A structure preserving model for power system stability analysis *IEEE Trans. Power Appar. Syst.* **PAS-100** 25–35
[44] Gelbrecht M, Kurths J and Hellmann F Monte Carlo basin bifurcation analysis *New J. Phys.* **22** 033032
[45] Halekotte L, Vanselow A and Feudel U 2021 Transient chaos enforces uncertainty in the British power grid *J. Phys. Complex.* **2** 035015
[46] Wallis S 2013 Binomial confidence intervals and contingency tests: mathematical fundamentals and the evaluation of alternative methods *J. Quant. Linguist.* **20** 178–208
[47] You J, Ying Z and Leskovec J 2020 Design space for graph neural networks *Advances in Neural Information Processing Systems* vol 33 (Curran Associates, Inc.) pp 17009–21
[48] Ioffe S and Szegedy C 2015 Batch normalization: accelerating deep network training by reducing internal covariate shift *32nd Int. Conf. Machine Learning, ICML 2015*
[49] Srivastava N, Hinton G, Krizhevsky A, Sutskever I and Salakhutdinov R 2014 Dropout: a simple way to prevent neural networks from overfitting *J. Mach. Learn. Res.* **15** 1929–58
[50] Hammond D K, Vandergheynst P and Gribonval R 2011 Wavelets on graphs via spectral graph theory *Appl. Comput. Harmon. Anal.* **30** 129–50
[51] Kipf T N and Welling M 2017 Semi-supervised classification with graph convolutional networks *5th International Conference on Learning Representations* (arXiv:1609.02907)
[52] Wu F, Zhang T, de Souza A H, Fifty C, Yu T and Weinberger K Q 2019 Simplifying graph convolutional networks (arXiv:1902.07153)
[53] Du J, Zhang S, Wu G, Moura J M F and Kar S 2017 Topology adaptive graph convolutional networks (arXiv:1710.10370)
[54] Bianchi F M, Grattarola D, Livi L and Alippi C 2021 Graph neural networks with convolutional ARMA filters *IEEE Trans. Pattern Anal. Mach. Intell.* 1

[55] Paszke A *et al* 2019 PyTorch: an imperative style, high-performance deep learning library *Advances in Neural Information Processing Systems* vol 32 ed H Wallach, H Larochelle, A Beygelzimer, F. d. Alch-Buc, E Fox and R Garnett (Curran Associates, Inc.) pp 8024–35

[56] Fey M and Lenssen J E 2019 Fast graph representation learning with pytorch geometric (arXiv:1903.02428)

[57] Bezanson J, Edelman A, Karpinski S and Shah V B 2017 Julia: a fresh approach to numerical computing *SIAM Rev.* **59** 65–98

[58] Rackauckas C and Nie Q 2017 DifferentialEquations.jl A performant and feature-rich ecosystem for solving differential equations in julia *J. Open Res. Softw.* **5** 15

[59] Lindner M, Lincoln L, Drauschke F, Koulen J M, Würfel H, Plietzsch A and Hellmann F 2021 NetworkDynamics.jl-Composing and simulating complex networks in Julia *Chaos* **31** 063133

[60] Plietzsch A, Kogler R, Auer S, Merino J, Gil-de Muro A, Lie J, Vogel C and Hellmann F 2021 PowerDynamics.jl—an experimentally validated open-source package for the dynamical analysis of power grids (arXiv:2101.02103)