Power Flow Approximation based on Graph Convolutional Networks

Valentin Bolz
DIgSILENT GmbH
72810 Gomaringen, Germany;
Dept. Cognitive Systems
University of Tübingen
72076 Tübingen, Germany
v.bolz@digsilent.de

Johannes Rueß
DIgSILENT GmbH
72810 Gomaringen, Germany
j.ruess@digsilent.de

Andreas Zell
Dept. Cognitive Systems
University of Tübingen
72076 Tübingen, Germany
andreas.zell@uni-tuebingen.de

***Abstract*—In this article we develop a graph convolutional neural network (GCN) for the approximation of the AC power flow in electrical power grids. The proposed architecture is fully generic and purely data driven, such that no information about the actual underlying physical topology of the power grid is required. This gives the opportunity to apply this approach to a wide range of multivariate regression problems. We test our architecture on 3 datasets of different sizes, two of which are real world type power grids containing up to 5488 nodes. We show that the proposed method allows an accurate approximation of the power flow specifically in the case of large power grids. The GCN architecture implies intrinsic extrapolation, allowing a reasonable reduction of the number of trainable parameters as well as training samples. In contrast, classical approaches based on fully connected networks are shown to face difficulties when fitting such high dimensional functions.**

***Index Terms*—Graph Convolutional Network, Graph Neural Network, Power Flow, Load Flow, Artificial Neural Network, Multivariate Regression**

I. Introduction

A key infrastructure of today's societies is the power system, which provides all of its members with electrical power. A secure operation and sustainable long term planning is at the core of a power system operator's task.

In order to guarantee a stable and high quality power supply, precise information of a power system's status is required. Usually this is done by an analytical simulation. One way to assess the status is the static analysis, also called *power flow analysis*, or short *power flow*, also known as *load flow*. In contrast to the more elaborated dynamic simulation, this gives a quick and often sufficient estimate of the required information.

Due to an increasing amount of decentralized renewable energy supplies in power systems and the growing interconnection of different countries' power grids, a secure operation and planning becomes progressively challenging. Usually, in order to treat these increased uncertainties and interdependencies, power system operators try to investigate their grid's status in many possible scenarios. A full screen of all possible scenarios in a reasonable amount of time is usually out of reach.

This requires further development of fast load flow approximation methods. In this article we contribute a new neural network design, taking a power system's intrinsic topological information into account.

Artificial neural networks (ANNs) have been already applied to power system related problems since the late 80s, see e.g. [1]. For an early work about the approximation of a load flow calculation on small power grids we refer to [2]. In the last two decades many different applications to power grids have been studied, such as fault detection [3], load forecasting [4] or load flow approximation.

For the latter, various approaches have been made, see [5], [6], [7]. The techniques of these approaches are based on fully connected networks (FCNs) with some variations in order to process contingencies. In contrast, an approach using 'classical' convolutional neural networks has been chosen in [8]. However, what all of those networks have in common is, that they do not exploit the grid's intrinsic topology explicitly.

We propose a method using the newly emerged *graph convolutional networks* (GCNs) to approximate the load flow. These networks generalize the idea of using local features from regular grids such as images, to data having irregular non-euclidean graph structure, see for example [9]. Here, two approaches have been established: A spectral approach, see [10], [11], on one side and a spatial approach as outlined in [12] on the other side. For a survey of these methods we refer to [13].

Most recently, in [14], [15] the power grid's physical topology has been used explicitly to build a neural network. In [14] the authors introduce a neural network design which emulates the load flow. The nice extrapolation properties of this approach come with the cost of a restriction to grids, where all lines are similar to each other. [15] approximates the *optimal power flow*, which is indeed a different calculation type than the load flow and no weight-sharing is employed which usually obstructs a good extrapolation of patterns. Both papers evaluate their techniques on small test examples. Contrary to the mentioned papers, we develop a fully generic and pristine graph convolutional network, which shows best performance on large sized real world type power grids. Furthermore, we propose a generic approach rather than requiring information on physical connections.

978-1-7281-4550-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICMLA.2019.00274

The application of graph convolutions to the approximation of the load flow comes naturally due to the following properties of the load flow calculation: Firstly, the dependence between the electrical elements is local. Thus, local substructures can be exploited across a power system. Secondly these substructures may be found repeatedly in a power system. Since the topology of a power system is highly irregular, the 'classical' convolutions cannot be used, hence graph convolutions are required for an applicable neural network design.

Our proposed architecture may be considered as a spatial GCN. The approach uses a purely data driven graph estimation, which is exploiting dependencies of the studied quantities. The estimated graph enables us to represent 'hidden' dependencies in the power system, like remote controls, which do not originate from a physical neighborhood. In fact, meaningful physical neighborhoods are represented in the estimated graph as well.

The main contribution of this paper is to present a GCN architecture that scales quite well with the size of the power system. It provides a highly accurate load flow approximation and outperforms classical approaches based on fully connected networks. Specifically on large grids, the proposed architecture exhibits extrapolation which allows a significant reduction of the number of training samples. Additionally, due to the fully generic data driven nature of this approach, the architecture is flexible and can easily be adopted to many multivariate regression tasks. To the best of the authors' knowledge this is the first design of a generic, full flavored, pristine GCN for the load flow calculation task.

The proposed method is tested on three power grids of different sizes, the IEEE standard example *39 bus* (34 lines) and the two real world type grids, *Texas* (2345 lines) and *US East Coast* (5411 lines).

In order to carry out a load flow analysis and also to simulate and create test data, we used the power system analysis software *PowerFactory* [16] of DIgSILENT GmbH. *PowerFactory* is a state-of-the-art tool for the modeling and analysis of power systems. It uses highly optimized numerical algorithms to calculate fast and precise solutions of large systems.

This article is organized as follows: In Section II we give a brief summary of the load flow analysis. Then in Section III the datasets which are used to test our approach are introduced. Afterwards we present our GCN architecture in Section IV followed by the experiments in Section V. Lastly, a summary is given in Section VI.

II. The load flow analysis

The load flow analysis calculates the powers, currents, voltages etc. on each element of the system, giving full information of a power system's state. In this section we give a brief summary of the load flow calculation.

Generally a power grid consists of multiple different connected elements of which we point out four specific ones:

Generators: Power supply,
Loads: Power consumption,
Buses: Connection points,
Lines: Transportation of electricity.

Many more types of elements are usually present in a real world power system. One may think of transformers, shunts, many types of controllers, etc. Although our method is able to handle arbitrarily defined power grids, for simplicity reasons we will focus in this section on the above mentioned four elements. However, our experiments are applied to real world type power systems containing more types of grid components.

To state our task we apply an arbitrary numbering to the buses, generators, loads and lines. Let $n \in \mathbb{N}$ be the number of buses $\{\text{Bus}_1, \dots, \text{Bus}_n\}$ in a power system. For all $j \in \{1, \dots n\}$ we write

P_j: Active power supply at Bus_j ($P_j \in \mathbb{R}$),
Q_j: Reactive power supply at Bus_j ($Q_j \in \mathbb{R}$),
V_j: Voltage at Bus_j ($V_j \in \mathbb{C}$),
Y: Admittance matrix ($Y \in \mathbb{C}^{n \times n}$).

The load flow analysis' goal is to calculate the complex voltages V_j for each bus, given P_j, Q_j and Y. This already gives full information about the power grid's state. The most fundamental equations describing the load flow are the Kirchhoffs laws:

1st law: *The sum of currents at a bus equals zero.*
2nd law: *The sum of voltages in a closed loop equals zero.*

For the purpose of a load flow analysis a system of equations can be established using these laws. They can be reformulated equivalently using powers and voltages, resulting in the so-called *power equations*:

$$P_j = \text{Re}(V_j) \sum_{k=1}^{n} \big(\text{Re}(Y_{jk})\text{Re}(V_k) - \text{Im}(Y_{jk})\text{Im}(V_k)\big) + \text{Im}(V_j) \sum_{k=1}^{n} \big(\text{Im}(Y_{jk})\text{Re}(V_k) + \text{Re}(Y_{jk})\text{Im}(V_k)\big) \quad (1)$$

$$Q_j = \text{Re}(V_j) \sum_{k=1}^{n} \big(-\text{Im}(Y_{jk})\text{Re}(V_k) - \text{Re}(Y_{jk})\text{Im}(V_k)\big) + \text{Im}(V_j) \sum_{k=1}^{n} \big(\text{Re}(Y_{jk})\text{Re}(V_k) - \text{Im}(Y_{jk})\text{Im}(V_k)\big) \quad (2)$$

for all $j \in \{1, \dots n\}$. (1) and (2) form a system of nonlinear equations, which is highly sparse. The number of these equations can be huge for real world applications. In order to treat such large-sized systems, advanced research has been done in order to develop stable and fast methods to solve these equations. For an overview see [17].

A classical and well studied way to solve this system of equations is using Newton-Raphson algorithms. Due to numerous highly sophisticated improvements on this iterative method, it is able to solve the load flow quickly and precisely. While those methods are commonly applied sequentially, a possible advantage of using neural networks for this task is their ability to approximate numerous load flows in parallel.

In our proposed GCN method we approximate the load flow calculation, while not learning the actual voltages at each bus but the *loadings* on each line.

Due to its physical properties, the amount of current a line can carry is limited. If the current exceeds a certain limit it will heat the line and potentially lead to damage. Additionally, the losses of energy on the line increase if the current reaches the line's limit. Therefore, it is in special interest of a system operator to guarantee that these limits are not exceeded in any system configuration.

The loadings are defined as follows. Each line has a maximum current $I_{\max}$ it can carry. With I_{act} being the actual current on the line, we define the loading *Ldg* as

$$Ldg = (I_{\text{act}} \cdot 100) \cdot I_{\max}^{-1}.$$

Note that this quantity is a simple function of the complex voltages at the endings of a line. Therefore the prediction of loadings will require the neural network to fit the whole load flow equations implicitly. One of the advantages in observing the loadings is that it gives a powerful yet easy interpretation. The loading is a relative quantity which indicates the utilization of a line in percent. Achieving good training and testing results with a neural network gives a direct feedback on the performance of this method.

III. Datasets

We test our GCN on three datasets of different size. The datasets' samples contain the load flow results of the different elements of a power system model, such as generators, loads and lines. The datasets are generated using the power system analysis software *PowerFactory* [16].

Each dataset is structured as follows. The powers observed on the generators and loads are supposed to be the *inputs* to the neural network. The quantities that have to be learned are the loadings on the lines. They are referred to as the *outputs* of the neural network. We set $d_{\text{inp}}, d_{\text{out}} \in \mathbb{N}$ to be the total number of inputs and outputs, respectively.

To generate the training samples, we run a Monte-Carlo experiment. We choose the active power P_{Gen} on each generator and the active power P_{Lod} as well as the reactive power Q_{Lod} on each load to be random. These quantities are distributed independently of each other uniformly on the interval $[50\%, 150\%]$ relative to their original value. For each Monte-Carlo sample, these random variables are drawn, a load flow is calculated and the lines' loadings are observed.

We use three different power systems for our experiments: the *39 Bus* System, *Texas* and US *East Coast*. See Table I for an overview.

A. The test power grids

1) 39 Bus: This power system model is shown in Figure 1. It is a small *IEEE* standard model containing 10 generators, 19 loads and 34 lines.

Two quantities are randomly chosen for each load $(P_{\text{Lod}}, Q_{\text{Lod}})$ adding up to 48 inputs and 34 outputs in total. The dataset consists of 350.000 samples. The last 30.000 samples are separated and treated as the validation set. The remaining 320.000 samples form the training data set. Additionally a test dataset containing 30.000 samples is provided.

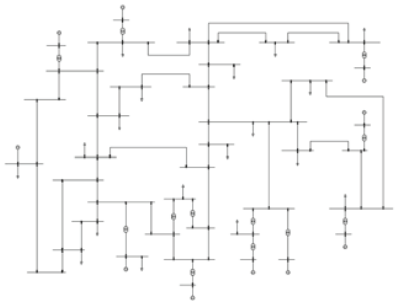

Fig. 1. The *39 Bus* system.

2) Texas: This power system model is provided by Birchfeld et al. in [18]. It is shown in Figure 2. It is a synthetic reasonable sized power grid designed to have similar properties as the real world power grid.

Fig. 2. The *Texas* system. The graphics has been generated using *PowerFactory*'s [16] automated drawing tool.

This power system has 544 generators, 1350 loads and 2345 lines. 121 generators and loads are set out of service and have been omitted. Consequently, in total there are 3123 inputs and 2345 outputs. The dataset consists of 350.000 samples. The last 30.000 samples are separated for the purpose of validation. The remaining 320.000 samples form the training dataset. Additionally a test dataset containing 30.000 samples is provided.

3) East Coast: This power system model, also provided by [18], is modeled similarly to the *Texas* dataset. For data generation we restricted the focus to the regions New Jersey, New York City and Pennsylvania, resulting in a very large power system, as shown in Figure 3. It contains 835 generators, 3287 loads and 5411 lines. Again, 203 of the inputs were constantly zero, so that in total there are 7206 inputs and 5411 outputs. The dataset contains 220.000 samples. The last 20.000 samples are separated for the purpose of validation. The remaining 200.000 samples form the training data set. Additionally a test dataset containing 20.000 samples is provided.

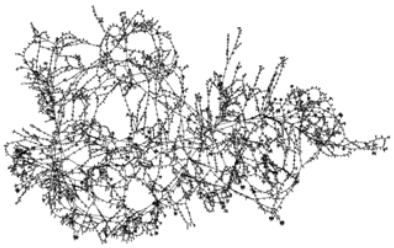

Fig. 3. The *East Coast* system. The graphics has been generated using *PowerFactory*'s [16] automated drawing tool.

TABLE I
Overview of the dataset dimensions.

Dataset	Inputs	Outputs
39 Bus	48	34
Texas	3123	2345
East Coast	7206	5411

IV. Methodology

Graph convolutional networks are an active field of research, see the references given in the introduction. We develop a GCN architecture well suited for the load flow approximation problem in which we exploit local repeating patterns across a given power grid. The architecture is influenced by the ideas outlined in [12].

A brief summary of the proposed architecture is given: The architecture is composed of several layers. The first layer is a *locally connected layer* which is basically a fully connected layer with cut out connections. This layer maps linearly from the input space $\mathbb{R}^{\mathcal{I}}$ to the output space $\mathbb{R}^{\mathcal{O}}$, with $\mathcal{I}$ and $\mathcal{O}$ being the set of all inputs and outputs respectively. The first layer is followed by several graph convolutional layers which all map from $\mathbb{R}^{\mathcal{O}}$ to $\mathbb{R}^{\mathcal{O}}$ and exploit local behavior of the load flow calculation in the power grid. Finally, for the last layer a locally connected layer is chosen. For an overview, see Figure 4.

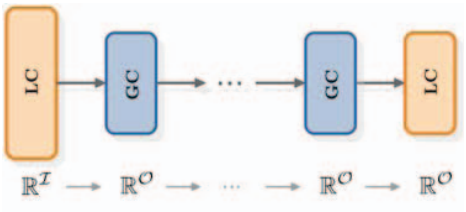

Fig. 4. Overview of the GCN architecture. **LC**: locally connected layer, **GC**: graph convolutional layer.

For a detailed description of the different layers, we introduce the following objects and notions:

For each $j \in \{1, \dots, s\}$ we define $\xi_j \colon \mathcal{I} \cup \mathcal{O} \to \mathbb{R}$ with $\xi_j(u)$ being the observed value on u in the j-th training sample. For simplicity reasons we set $\boldsymbol{u} = (\xi_1(u), \dots \xi_s(u))$.

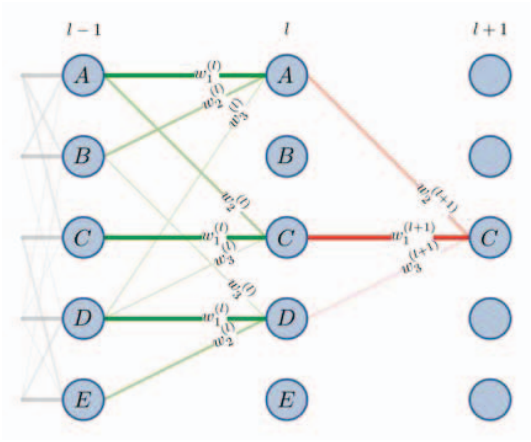

Fig. 5. Visualization of the graph convolutional layer. The two convolution kernels $f^{(l)} = (w_1^{(l)}, w_2^{(l)}, w_3^{(l)})$ and $f^{(l+1)} = (w_1^{(l+1)}, w_2^{(l+1)}, w_3^{(l+1)})$ are applied in layer l and $l+1$, respectively. For a better readability only the important connections for the activation of $C^{(l+1)}$ are printed. In this example it is $\eta_C(C) = 1$, $\eta_C(A) = 2$ and $\eta_C(D) = 3$. The higher a neuron in a preceding layer is correlated the darker and thicker the lines are drawn.

In order to measure the impact of two elements to each other the Pearson product-moment correlation coefficients are calculated by

$$\mathrm{Corr}_{u,v} = \frac{\mathrm{Cov}(\boldsymbol{u}, \boldsymbol{v})}{\sigma_{\boldsymbol{u}} \cdot \sigma_{\boldsymbol{v}}}, \quad u, v \in \mathcal{I} \cup \mathcal{O},$$

where $\mathrm{Cov}(\cdot, \cdot)$ denotes the covariance and σ stands for the standard deviation. Note that $|\mathrm{Corr}_{u,o}| \leq 1$ for all $u, o \in \mathcal{I} \cup \mathcal{O}$. If $|\mathrm{Corr}_{u,o}|$ is close to 1, u is highly dependent on o, whereas closeness to 0 indicates independence.

For all $u \in \mathcal{I} \cup \mathcal{O}$ we further define a *neighboring function* $\eta_u \colon \mathcal{O} \to \{1, \dots, d_{\text{out}}\}$, with

$$\eta_u(o) := \#\{\tilde{o} \in \mathcal{O} : |\mathrm{Corr}_{u,\tilde{o}}| \geq |\mathrm{Corr}_{u,o}|\},$$

where $\#\{\dots\}$ denotes the cardinality of a set.

Since each neuron in each layer is a representation of an element in the power grid, this function makes it possible to select 'neighboring' neurons in a preceding layer. Note that, $o \in \mathcal{O}$ is the $\eta_u(o)$th highest correlated element by absolute value to u. Since $\mathrm{Corr}_{o,o} = 1$ for $o \in \mathcal{O}$, o is in any case the most correlated element to itself and so $\eta_o(o) = 1$.

In the following it is not relevant which training samples are considered. Thus, we write for $\xi_j(\cdot)$ shortly $\xi(\cdot)$.

Let $L \in \mathbb{N}$ be the total number of layers in the neural network. For $l \in \{1, \dots, L\}$ the layers $\mathcal{L}^{(l)}$ are defined on the following spaces

$$\mathbb{R}^{\mathcal{I}} \xrightarrow{\mathcal{L}^1} \mathbb{R}^{\mathcal{O}} \xrightarrow{\mathcal{L}^2} \mathbb{R}^{\mathcal{O}} \xrightarrow{\mathcal{L}^3} \dots \xrightarrow{\mathcal{L}^{L-1}} \mathbb{R}^{\mathcal{O}} \xrightarrow{\mathcal{L}^L} \mathbb{R}^{\mathcal{O}}.$$

We define $x^{(l)}(o)$ to be the output for $o \in \mathcal{O}$ of the l-th layer, that is $x^{(l)}(o) = ((\mathcal{L}^{(l)} \circ \mathcal{L}^{(l-1)} \circ \dots \circ \mathcal{L}^{(1)})(\xi|_{\mathcal{I}}))(o)$.

Thus, $x^{(l)}(o)$ may be interpreted as the activation of the neuron representing $o \in \mathcal{O}$ in the l-th layer. Except for the first one's, each layer's activation is therefore a representation of $\mathcal{O}$.

A. The GCN architecture

1) The first layer: For the first layer we use a *locally connected layer*. Let $n^{(1)} \in \{1, \dots, d_{\text{out}}\}$ be the number of active connections per input. We define a connection from an input i to an output o to be active, if $\eta_i(o) \leq n^{(1)}$. Thus, the active connections at input i are the connections to its top $n^{(1)}$ correlated outputs $o \in \mathcal{O}$. Let $\mathcal{N}_o = \{i \in \mathcal{I} \mid \eta_i(o) \leq n^{(1)}\}$ for all $o \in \mathcal{O}$ be the set of inputs which are connected to o. We define the activation of a neuron representing $o \in \mathcal{O}$ after the first layer to be

$$x^{(1)}(o) := \big(\sum_{i \in \mathcal{N}_o} w_{i,o} \cdot \xi(i) \big) + b_o,$$

where $w_{i,o}$ and b_o are trainable parameters.

Usually $n^{(1)} \ll d_{\text{out}}$ can be set which implies that this layer contains significantly less parameters than a fully connected one. Note that if $n^{(1)} = d_{\text{out}}$ this layer equals a fully connected layer.

2) The convolutional layer: The local behavior of the outputs is used by convolutions which act on the estimated neighborhoods in the following way.

Let $l \in \{2, \dots, L-1\}$ and $n^{(l)}$ be the filter size for the lth layer. The trainable filter kernel of the lth layer is a vector $f^{(l)} \in \mathbb{R}^{n^{(l)}}$. With $*_l$ being an operation similar to the standard discrete convolution, we define

$$\begin{aligned} x^{(l)}(o) &:= \text{ReLu}(f^{(l)} *_l (x^{(l-1)} \circ \eta_o^{-1}) + b^{(l)}) \\ &:= \text{ReLu}\Big(\sum_{k=1}^{n^{(l)}} f^{(l)}(k) \cdot x^{(l-1)}(\eta_o^{-1}(k)) + b^{(l)} \Big) \end{aligned}$$

for all $o \in \mathcal{O}$. Here, $b^{(l)} \in \mathbb{R}$ is a trainable bias parameter and η^{-1} denotes the inverse function of η. This is the computation of one *filter map* given one incoming filter map. The generalization to an arbitrary number of filter maps is straight forward and analogous to classical convolutional networks.

3) The last layer: A locally connected layer which connects each output neuron to its neighbors in the previous layer is used as a last layer.

We define the output of the Lth layer for $o \in \mathcal{O}$ to be

$$x^{(L)}(o) := \sum_{k=1}^{n^{(L)}} w_{o,k} \cdot x^{(L-1)}(\eta_o^{-1}(k)), \tag{3}$$

where $w_{o,k} \in \mathbb{R}$ are trainable parameters. This is the representation in the case that the output of the previous layer consists of only one filter map. In the case of multiple incoming filter maps the operation defined in (3) is applied for each incoming filter map independently and summed up among all filter maps afterwards. Suppose the last convolutional layer computes $m \in \mathbb{N}$ filter maps. Let ${}_j x^{(L-1)}$ denote the output of the jth filter map in the last convolutional layer. Similarly to (3) the output of the last layer is defined by

$$x^{(L)}(o) := \sum_{j=1}^{m} \sum_{k=1}^{n^{(L)}} w_{o,k,j} \cdot {}_j x^{(L-1)}(\eta_o^{-1}(k)),$$

where $w_{o,k,j} \in \mathbb{R}$ are the set of trainable parameters.

B. Properties of the proposed architecture

A visualization of an estimated neighborhood in the *Texas* power grid is shown in Figure 6. As it can be seen the correlations indeed reveal the local behavior of the lines as well as dependencies which are further away in the grid.

Fig. 6. The *Texas* power system. The red lines are the 10% most correlated lines to the blue one by absolute value.

For the selection of neighboring lines we also investigated a random walk on the lines based on the correlations as it is suggested in [12]. This did not bring any advantage in our case, in fact the correlations turn out to be a simple yet powerful method to determine neighboring lines.

Key properties of this GCN approach are:

- *No a priori knowledge needed.* In some cases the adjacency matrix of the underlying graph of a power grid may be difficult to be observed or even unknown. Estimating the graph topology does not face this problem. This makes this approach efficiently applicable to a wide range of multivariate regression tasks.
- *It is a blackbox.* In many cases, the power grid's physical topology graph does not represent all dependencies inside a power grid. For example in the case of remote controls, two lines might have a high influence to each other without being connected directly. Using the correlation generated graph instead, one is able to cover all inherent dependencies.
- *Reduced number of parameters.* Due to many hyperparameters, such as $n^{(j)}$ for $j \in \{1, \dots, L\}$, the number of trainable parameters in this architecture can be kept small by still achieving good test results. This is in particular worthwhile for huge power grids.

V. Experiments

We evaluate our proposed architecture on the three datasets *39 Bus*, *Texas* and *East Coast*. A comparison of our architecture with a standard approach based on FCNs is given. We investigate in detail the impact of the design of the first layer on the overall accuracy. Additionally we evaluate performance in case of a reduced number of training samples.

The models are implemented with Keras [19] which is part of TensorFlow [20]. The Adam optimizer [21] is used to minimize the Euclidean distance ℓ_2 between the approximated and the real loadings. Each neural network is trained for 250 epochs. A batch size of 100 is used in the *39 Bus* and *Texas* cases and a batch size of 50 is applied to the *East Coast* grid. The inputs to the neural networks were standardized. Since the proposed architecture contains many hyperparameters numerous variants were trained. Here, the *Training Center for Machine Learning (TCML)* located at the University of Tübingen was used. The networks are all trained on NVIDIA's GTX 1080 TIs. The architectures achieving best results, both for FCNs and GCNs, are presented. We compare the following networks:

FCN 1: Fully connected network with two hidden layers consisting of $10 \cdot d_{\text{out}}$ neurons each. The two hidden layers are each followed by a ReLu activation function.

FCN 2: Fully connected network with one hidden layer consisting of d_{out} neurons. The hidden layer is followed by a ReLu activation function.

GCN $x\%$: Graph convolutional network as introduced in Section IV. The first layer connects each input to its $x\%$ highest correlated neurons in the next layer with $x \in \{1, 5, 10, 20, 50, 100\}$. For $x \neq 100$ four convolutional layers are used. Each of them computing 30 filter maps except for the last one which computes 20 filter maps. For $x = 100$ only two convolutional layers are used. All convolutional layers, as well as the last locally connected layer use $n^{(l)} = 20$ neighbors.

We present two different types of experiments. The first experiment evaluates FCNs and GCNs on each dataset with multiple first layer sizes. In the second experiment multiple architectures are trained on the same datasets with a varying number of training samples.

A. Comparison of architectures on full datasets

Figure 7, 8 and 9 show the validation errors of the architectures on *39 Bus*', *Texas*' and *East Coast*'s dataset, respectively. Table II depicts the achieved losses on the test datasets after 250 epochs. For the *39 Bus* grid the results of FCN 1 architecture is shown, while for *Texas* and *East Coast* grids FCN 2 is presented. Larger FCNs were also tested on *Texas* and *East Coast* which either led to memory problems due to a huge number of trainable parameters or worse test losses than FCN 2. All of the losses are plotted on a logarithmic scale.

As it can be seen FCN 1 outperforms the GCN architectures on the *39 Bus* dataset approximately by a factor of 4. We argue that the consideration of estimated neighborhoods is not really helpful for such small test cases. The filter kernels are applied only on a small number of outputs which provides too few information in order to extract meaningful substructures. Additionally these filter kernels are repeatedly convolved over the same neurons due to the small size of the grid.

Differently, on *Texas* and *East Coast* the GCN achieves test losses which outperform FCN 2 up to a factor of 19 and 120 respectively. In contrary to the small *39 Bus* case, these huge power grids allow to exploit numerous meaningful substructures. Thus, the convolution part in the architecture can take advantage of learning many different but still similar behaving local patterns.

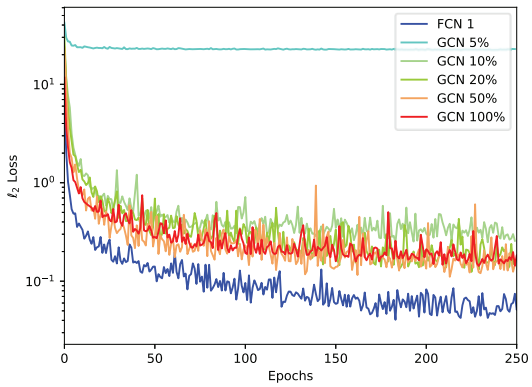

Fig. 7. ℓ_2 validation losses on *39 Bus*.

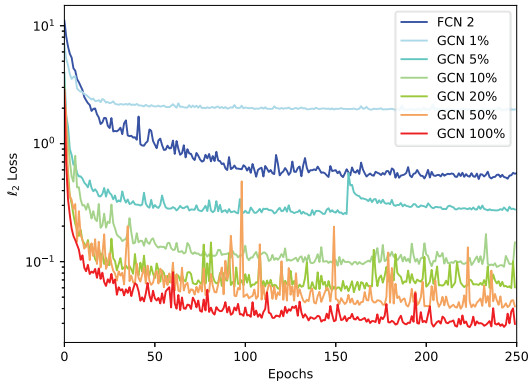

Fig. 8. ℓ_2 validation losses on *Texas*.

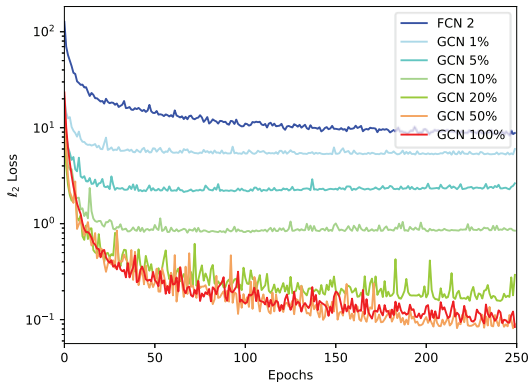

Fig. 9. ℓ_2 validation losses on *East Coast*.

As it can be seen from Section IV and Table III the first layer can contain a majority of the GCN architecture's trainable parameters. Therefore, the number of neighbors $n^{(1)}$ picked in

TABLE II
ℓ_2 test errors after 250 epochs on the full datasets.
(*GCN* 1% is not well-defined in the *39 Bus* case.)

Architecture	*39 Bus*	*Texas*	*East Coast*
FCN 1 / 2 / 2	**0.0316**	0.4656	8.3363
GCN 1%	NA	1.9231	5.2099
GCN 5%	22.5239	0.2435	2.12
GCN 10%	0.2464	0.0865	0.7969
GCN 20%	0.1243	0.0516	0.1565
GCN 50%	0.1106	0.0364	0.0812
GCN 100%	0.1240	**0.0244**	**0.0691**

TABLE III
Number of trainable parameters.

Architecture	*39 Bus*	*Texas*	*East Coast*
FCN 1 / 2 / 2	143,854	12,827,150	68,281,409
GCN 1%	NA	1,060,884	2,607,645
GCN 5%	62,391	1,354,446	4,164,141
GCN 10%	62,485	1,719,837	6,116,967
GCN 20%	62,626	2,453,742	10,015,413
GCN 50%	63,143	4,649,211	21,710,751
GCN 100%	27,882	7,801,130	41,210,187

the first layer is a valuable hyperparameter in order to adjust the total number of trainable parameters. Figure 8 reveals the impact of differently sized first layers on the *Texas* example. With an increasing number of connections in the first layer the test loss decreases. Similar observations can be made in Figure 9 for the *East Coast* case. Here, all of the GCN architectures perform better than an FCN. It is noteworthy that GCN 50% achieved nearly the same test loss after 250 epochs as GCN 100% by containing just approximately half of the trainable parameters, see Table III. We argue that in those huge power grids an input far away from an output contributes hardly any relevant information to its activation.

Furthermore it can be seen that for the datasets *Texas* and *East Coast* the GCN architectures mostly manage to achieve lower validation losses within just 3 epochs of training than the FCN reaches within the full 250 epochs. This is particularly beneficial for large power grids, since the training time can be reduced significantly.

B. Comparison of architectures on reduced datasets

In real world applications providing enough training samples to train a neural network reasonably is often challenging. Data generation is commonly time consuming or just few data is available from observations of the past. Therefore, we investigate how the presented architectures behave if the number of training samples is reduced.

Particularly for regression problems, such as the load flow analysis, in which high dimensional functions have to be approximated, the exploitation of local patterns through convolutions can be helpful. A function with a high dimensional domain and codomain requires a large amount of training data in order to be represented well. Convolutions are able to extract joint information from the whole dataset which is particularly beneficial if few training data is provided.

We write 'xk dataset', for $x \in \{10, 50, 100\}$, if only the first x thousand samples of the training data are used. In each of the experiments, the validation and test datasets stay the same as described in Section III.

Figure 10 shows the validation errors of the FCN, GCN 20% and GCN 100% being trained with different amounts of training data on the *East Coast* system. In Table IV the test errors on the different sized datasets *39 Bus*, *Texas* and *East Coast* are shown.

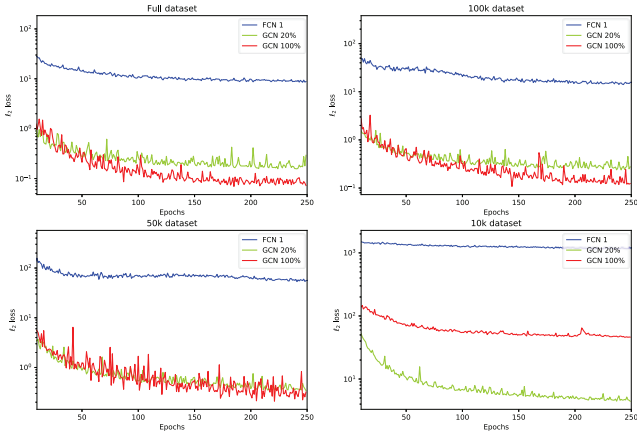

Fig. 10. ℓ_2 validation losses on *East Coast* using training datasets containing 320k / 100k / 50k / 10k samples.

TABLE IV
ℓ_2 test errors after 250 epochs on differently sized training datasets.

Dataset	Arch.	Full	100k	50k	10k
39 Bus	*FCN* 1	**0.0316**	**0.0466**	**0.0813**	**0.3247**
	GCN 20%	0.1243	0.2112	0.4595	1.4676
	GCN 100%	0.1382	0.1704	0.1741	1.2619
Texas	*FCN* 2	0.4656	0.9290	1.5285	30.1519
	GCN 20%	0.0516	0.0783	0.1167	**1.0157**
	GCN 100%	**0.0244**	**0.0387**	**0.0555**	2.1972
East Coast	*FCN* 2	8.3363	11.5786	43.4877	1066.7799
	GCN 20%	0.1565	0.2384	0.3406	**4.4307**
	GCN 100%	**0.0691**	**0.1076**	**0.2087**	45.8960

This experiment again indicates that for small power grids such as *39 Bus* the benefits of using estimated neighborhoods are limited.

The GCN outperforms the FCN on *Texas* and *East Coast*. The less training data is available, the better the GCN 20% performs relatively to the FCN. This is in line with our previous argumentation. The bigger the power grids get and the less training information is provided, the more valuable the exploitation of substructures by convolutions becomes.

Interestingly, GCN 20% achieves considerably better results than GCN 100% in the 10k case. Thus, in the case of less

connections in the first layer, on huge power grids, the GCN is more forced to extract patterns instead of overfitting to the small training data set.

C. Summary of results

On the small *39 Bus* System the GCN achieves reasonable test losses, still being surpassed by the results of an FCN. In contrast, on the larger real world sized power grids *Texas* and *East Coast* the GCNs outperform FCNs significantly. Experiments showed that it is possible to decrease the number of trainable parameters in the proposed architecture considerably by still achieving very good test losses. Additionally less training samples are needed in order to reach good test results.

Key points of the results of our proposed method in comparison to FCNs are:

1) Lower test loss. The overall performance of our proposed GCN architecture is considerably better compared to FCNs on big power systems.

2) Faster convergence. In both datasets *Texas* and *East Coast* the validation loss of an FCN after 250 epochs was reached by the GCN 100% within just 3 epochs.

3) Less parameters. The GCN architectures allow to decrease the number of trainable parameters significantly.

4) Less training samples. The GCN can still achieve good test losses with less training samples provided.

VI. Conclusion & Future Work

We propose a method which relies on a spatial graph convolutional network in order to approximate the load flow calculation. We estimate the dependencies of the lines by considering their correlations to each other. This allows us to define convolutional kernels which are able to represent far-reaching interdependencies.

This method is designed to be as flexible as possible. Since it is purely data driven by just considering the dataset's correlations, this approach can be used for a variety of multivariate regression problems. Due to its intuitive convolution approach it opens the door for further architecture enhancements inspired by classical CNNs.

We test our method on three power grids, two of which represent real world sized applications. The benefit of using local features through convolutions on smaller power grids is shown to be limited. But for the more relevant bigger power grids our proposed method outperforms fully connected networks significantly.

Further investigation regarding architecture improvements and alternative GCN approaches may be considered. The flexibility of the proposed architecture invites new applications to many kinds of multivariate regression tasks.

Acknowledgments

We would like to thank Julian Jordan, Elmar Teufl, Maximus Mutschler, Kevin Laube and Hauke Neitzel for helpful discussions.

We express our thanks to our collegues at DIgSILENT GmbH for their support and valuable input.

This work is part of a joint research project of the University of Tübingen and DIgSILENT GmbH. The project is funded by DIgSILENT GmbH.

References

[1] D. J. Sobajic and Y.-H. Pao. Artificial neural-net based dynamic security assessment for electric power systems. *IEEE Transactions on Power Systems*, 4(1):220–228, 1989.

[2] V. L. Paucar and M. J. Rider. Artificial neural networks for solving the power flow problem in electric power systems. *Electric Power Systems Research*, 62(2):139 – 144, 2002.

[3] M. Jamil, S. Sharma, and R. Singh. Fault detection and classification in electrical power transmission system using artificial neural network. *SpringerPlus*, 4:334, Jul 2015.

[4] O. Abedinia and N. Amjady. Short-term load forecast of electrical power system by radial basis function neural network and new stochastic search algorithm. *International Transactions on Electrical Energy Systems*, 26(7):1511–1525, 2016.

[5] B. Donnot, I. Guyon, M. Schoenauer, P. Panciatici, and A. Marot. Introducing machine learning for power system operation support. *CoRR*, Sep 2017.

[6] L. Duchesne, E. Karangelos, and L. Wehenkel. Machine learning of real-time power systems reliability management response. In *2017 IEEE Manchester PowerTech*, pages 1–6, Jun 2017.

[7] B. Donnot, I. Guyon, A. Marot, M. Schoenauer, and P. Panciatici. Fast power system security analysis with guided dropout. In *26th European Symposium on Artificial Neural Networks, ESANN, Bruges, Belgium*, 2018.

[8] Y. Du, F. Li, J. Li, and T. Zheng. Achieving 100x acceleration for n-1 contingency screening with uncertain scenarios using deep convolutional neural network. *IEEE Transactions on Power Systems*, 34(4):3303–3305, Jul 2019.

[9] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, Jul 2017.

[10] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS*, Apr 2014.

[11] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29*, pages 3844–3852. 2016.

[12] Y. Hechtlinger, P. Chakravarti, and J. Qin. A generalization of convolutional neural networks to graph-structured data, 2017.

[13] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. Yu. A comprehensive survey on graph neural networks, Jan 2019.

[14] B. Donon, B. Donnot, I. Guyon, and A. Marot. Graph Neural Solver for Power Systems. In *International Joint Conference on Neural Networks*, Budapest, Hungary, 2019.

[15] C. Kim, K. Kim, P. Balaprakash, and M. Anitescu. Graph Convolutional Neural Networks for Optimal Load Shedding under Line Contingency. In *IEEE Power Engineering Society General Meeting*, 2019.

[16] *PowerFactory 2019*. Power System Analysis Software, DIgSILENT GmbH, Gomaringen, Germany. https://www.digsilent.de/en/.

[17] D. Mehta, D. K. Molzahn, and K. Turitsyn. Recent advances in computational methods for the power flow equations. *2016 American Control Conference (ACC)*, pages 1753–1765, 2016.

[18] A. B. Birchfield, T. Xu, K. M. Gegner, K. S. Shetye, and T. J. Overbye. Grid structural characteristics as validation criteria for synthetic networks. *IEEE Transactions on Power Systems*, 32(4):3258–3265, Jul 2017.

[19] F. Chollet et al. Keras. https://keras.io, 2015.

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[21] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA*, May 2015.