Neural Networks for Power Flow: Graph Neural Solver

Balthazar Donon, Rémy Clément,
Benjamin Donnot, Antoine Marot
Réseau de Transport d'Électricité
Paris, France
{balthazar.donon, remy.clement,
benjamin.donnot, antoine.marot}@rte-france.com

Isabelle Guyon, Marc Schoenauer
TAU group of Lab. de Rec. en Informatique
UPSud/INRIA Université Paris-Saclay
Paris, France
iguyon@lri.fr, marc.schoenauer@inria.fr

***Abstract*—Recent trends in power systems and those envisioned for the next few decades push Transmission System Operators to develop probabilistic approaches to risk estimation. However, current methods to solve AC power flows are too slow to fully attain this objective. Thus we propose a novel artificial neural network architecture that achieves a more suitable balance between computational speed and accuracy in this context.**
Improving on our previous work on Graph Neural Solver for Power System [1], our architecture is based on *Graph Neural Networks* and allows for fast and parallel computations. It learns to perform a power flow computation by directly minimizing the violation of Kirchhoff's law at each bus during training. Unlike previous approaches, our graph neural solver learns by itself and does not try to imitate the output of a Newton-Raphson solver. It is robust to variations of injections, power grid topology, and line characteristics.
We experimentally demonstrate the viability of our approach on standard IEEE power grids (case9, case14, case30 and case118) both in terms of accuracy and computational time.

***Index Terms*—Power Flow, Solver, Artificial Neural Networks, Graph Neural Networks, Graph Neural Solver**

I. Background & Motivations

Transmission system operators such as RTE (Réseau de Transport d'Électricité) are in charge of managing the power system infrastructure. They perform security analyses that rely on the "N-1" criterion: for every single outage that can happen, they make sure that the grid remains stable. Recent developments in the energy ecosystem in Europe, as well as those envisioned for the next few decades (increasing amount of uncertainty caused by intermittent sources of energy or by a less predictible end consumer's behavior, more open energy markets, increasing automation of the grid, etc.) push TSOs to rethink their approach to security analysis.

The GARPUR[1] consortium reports advocate for a probabilistic approach to power system risk assessment, going beyond the simple "N-1" criterion [2], [3]. Current methods for AC power flow computations (such as Newton-Raphson) are too slow to enable this, and a method that would be faster (at the cost of a reduced accuracy) could help achieve a reasonable statistical evaluation of the risk. Speeding up computations of power flow computation using artificial intelligence seems to be a promising avenue to get closer to this goal. Pioneering work by Nguyen [4] demonstrated the feasibility of such an approach. Donnot et al. continued in this direction, and managed to encode atypical interventions on a given grid [5], [6]. Duchesne et al. [7] also use Machine Learning coupled with control variate to speed up a Monte Carlo approach.

[1] www.sintef.no/projectweb/garpur

We propose a novel method based on graph neural networks to solve the AC power flow problem. This method does not aim at imitating another method, but consists in training a neural network by direct minimization of the violation of physical laws. Our solver approach lies at the interface between artificial intelligence and optimization, and offers interesting computational performance.

The artificial neural networks domain [8], [9] is very proficient and has achieved several major breakthroughs in the past decade, thanks to a proactive community of researchers and to the exponentially-increasing computational capability of modern computers. Application of artificial neural networks to physics problems has also known a surge of interest in recent years. Trajectories of interacting particles have been modelled [10], fluid mechanics computations problems have been accelerated [11], and applications to quantum mechanics offer promising results [12], [13], [14].

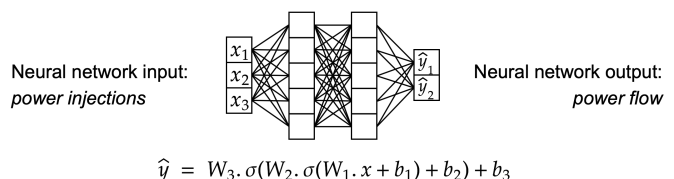

Fig. 1. **Artificial Neural Network example** - Artificial Neural Networks consist in function approximations that typically look like the equation above. The function σ is a non linearity (typically ReLU or tanh), and parameters of the model W_i and b_i are fitted by a stochastic gradient descent trying to minimize the distance between the prediction and the ground truth.

We advocate for a meticulous approach to the way AI tools are being developed for critical and industrial systems such as power grids. More specifically, encoding the known invariants and symmetries of a system directly inside the structure of artificial intelligence models make them resilient by design. More specifically, and to give a power system example, a "black box" model that would be trained on a situation where

every line is connected, will most likely not be accurate anymore if one of those lines were to be disconnected. On the contrary, a model that directly encodes invariants and symmetries will better withstand those perturbations. Our work is intrinsically in line with Battaglia et al. [15], as we aim at building a strongly generalizable neural network architecture that interwines generic learning blocks in an instance agnostic manner.

Graph Neural Networks are a class of Artificial Neural Network that are designed to respect the permutation invariants and edge symmetries of graph structures. They use neural network functions such as the one described in Figure 1 as small learning blocks that will be used at several places of the network (see Figure 2). They were first introduced by Scarselli et al. in [16] and further developed in [17], [18], and their theoretical properties have been investigated in [19], [20]. They have been successfully applied to graph classification [21], [22], [23], vertex classification [24], [25] and relational reasoning [26]. [27] and [10] are examples of hybrid approaches that blend deep learning and knowledge about the structure of the problem. This novel domain of artificial intelligence aims at dealing with graph-structured data. The input data x relies on a graph structure presented at the top left of the figure, and our goal is to perform a prediction $\hat{y}$ that is born by the same graph structure. The key idea of Graph Neural Networks is to iteratively propagate messages through the edges of the graph structure. These propagations can be seen as "correction" updates.

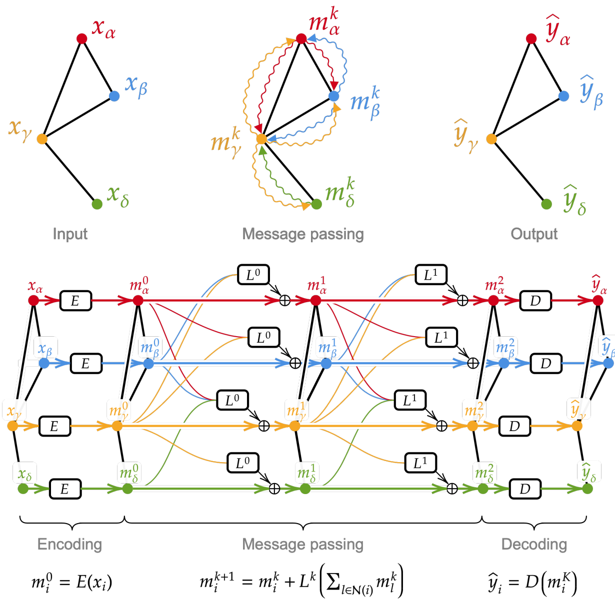

Fig. 2. **Graph Neural Network** - The idea of this novel class of neural is to iteratively propagate messages between direct neighbors.

Our previous work on the Graph Neural Solver [1] laid the ground for a strongly generalizable fast neural solver. It relies heavily on graph neural networks, and consists in three main parts: first an embedding of the input (injections at each line side), then a message propagation, and finally a decoding part that converts the resulting messages into flows through lines. The overall architecture is presented in Figure 2. The structure of the Graph Neural Network that we used is presented at the bottom part of Figure 2 and is organized as follows:

- First, the input message x_i of each bus i is converted into a message m_i^0 using a common encoder (which is a neural network similar to the one shown in Figure 1).
- Secondly, those messages are iteratively updated by common neural networks L^k that take as an input - for each bus i - the sum of the messages of its neighbors $l \in \mathcal{N}(i)$. There are K "correction" updates, and each of the K neural networks L^k are different.
- Finally, the resulting messages m_i^K are all decoded into the prediction $\hat{y}_i$ by applying another common neural network D. All the $K+2$ neural network blocks (E, L^0, ..., L^K, D) are then all jointly trained by a gradient descent learning process.

We experimentally demonstrated that our Graph Neural Solver architecture was able to learn on randomly generated power grids of 30 buses, and infer (with a better accuracy than the DC approximation) on random power grids of sizes ranging from 10 to 110 buses, thus achieving a form of zero-shot learning [28]. This architecture is by design robust to power grid topology perturbations and variations of injections, and has the property to be fully differentiable as it relies exclusively on neural networks.

However, there were some limits to this first investigation that were left for future research. First of all, the strong hypothesis that all power lines share the same physical characteristics was made, to simplify the problem. Secondly, the datasets that were used both for training and testing had already been processed by RTE's proprietary power flow solver Hades2 to ensure a global active equilibrium. Another issue was the need to perform expensive calls to a Newton-Raphson solver to generate a dataset that was later used to train the Graph Neural Solver. Finally, the computational speed was not significantly better that the one of a classical approach (such as Newton-Raphson).

One contribution of this paper is to build a neural network architecture that solves the AC power flow equations and that answers all the previously mentioned issues. Moreover, our previous work relied on minimizing the distance between our neural network's predictions and the results of a classical Newton-Raphson method. The main novelty of our present work is that we no longer try to imitate the output of another solver and we directly penalize the violation of physical laws during the training process. Our approach is thus completely independent from any other algorithm, and stands as a new way of performing a power flow. Moreover, we moved closer to the classical equations of the power systems literature, thus enabling the incorporation of line characteristics variations in our model. We ensure a global active equilibrium directly as a part of our architecture. Finally, a better implementation of the architecture using the deep learning framework Tensorflow [29], and the use of GPUs allowed to drastically decrease the computational time (during inference), and thus make our

Graph Neural Solver a relevant new method for power systems operation.

This paper is organized as follows. First we introduce the notations, explain the structure of our proposed neural network architecture, and how our learning algorithm will be trained. Then we experimentally demonstrate the viability of our approach on a set of IEEE power grids of different sizes. Finally we develop on the encountered limits, on how this work should be further improved, and on how the properties of this new solver (computational speed and differentiability) can be exploited.

II. Proposed Methodology

In this section, we develop on the overall architecture and equations of our proposed Graph Neural Solver. We also give details about some power grid modelling choices, and about the concept of message propagation.

A. Graph Neural Solver structure

Our goal is to predict v and θ at each bus of a power grid. We set ourselves in a steady-state setup. We take as input (see Appendix and subsection II-C for more details about those quantities):

- a base apparent power: $baseMVA$;
- a list of buses: $(i, P_{d,i}, Q_{d,i}, G_{s,i}, B_{s,i})$;
- a list of lines: $(from_bus, to_bus, r, x, b_c, \tau, \theta_{shift})$;
- a list of generators: $(gen_bus, \overline{P}_{g,i}, \underline{P}_{g,i}, \mathring{P}_{g,i}, v_{g,i})$.

We choose the following convention for the notations: quantities such as v, θ, etc. have a superscript k that corresponds to the "correction" update k, and a subscript i that corresponds to the bus i of the power grid (there are N buses on the grid). The absence of a subscript i denotes the vector (or matrix in the case of the messages m defined in subsection II-D) that concatenates the corresponding quantity for all buses.

Our goal being to develop a power flow solver based on artificial neural network, we choose to use the same data structure and power grid modelling as MATPOWER [30]. More details about notations and power grid modelling are available in the Appendix. All notations and equations are compliant with the MATPOWER modelling of a power grid.

Our proposed architecture is an iterative process that performs "correction" updates on the variables v, θ and m of each bus of a power grid. These "correction" updates are analogous to the iterations performed by a Newton-Raphson solver.

Previous attempts at using artificial intelligence to tackle physical problems tried to mimic the behavior of another solver. In this work, we go a step further and break free from the need to imitate a classical solver. This is the main contribution of this paper: instead of minimizing the distance between our prediction and the output of a solver, we directly minimize the violation of physical laws (i.e. Kirchhoff's laws) at training time. This is achieved at the cost of incorporating directly into the architecture the physics equations that model the behavior of power grids.

Our algorithm is a novel hybrid approach that stands directly at the interface between artificial intelligence and optimization. During training, our model is never told what the actual solution is. We let it perform a prediction, we compute its error, and tell it how it should alter its neural network blocks to decrease it. This way, it crystallizes a behavior that minimizes the physical laws violation. At inference time, it has no access whatsoever to the global objective function, but only mechanically applies the behavior that it has previously learned.

Our architecture consists in an initialization followed by loops of global active compensations, local power imbalance computations and neural network updates. It intricates both learnable neural network parts and non-learnable physics-based parts. The learnable parts (which are neural networks) are trained through a learning process detailed below. The overall idea of the architecture is to iteratively and locally minimize the violation of physical laws, while ensuring a global active equilibrium. A full description of how the different steps are laid out is shown in Algorithm 1.

The amount of "correction" updates K, the dimension of the messages d and the discount factor γ are hyperparameters of the model.

1) Initialization: All voltages are set to 1 p.u. (with the exception of the buses that are connected to a generator, whose voltage is set to the generator's voltage set point), phase angles to 0 rad, and messages (see subsection II-D) to a d-dimensional zero message.

Details in the green equations (1) – (3) of Figure 3.

2) Global active compensation: This step ensures a global active equilibrium. It finds the value of the global parameter λ that ensures this equilibrium. As detailed later in subsection II-C, λ is a global parameter that controls the active power output of every generator on the grid. First, we need to compute the global consumption, including the losses via Joule's effect, and then solve a simple linear system. We also enforce a local reactive equilibrium at each generator by modifying their reactive power outputs.

Details in the orange equations (4) – (11) of Figure 3.

3) Local power imbalance computation: The local power imbalance at bus i and at update iteration k is defined by $\Delta S_i^k = \Delta P_i^k + j \Delta Q_i^k$ (where j is such that $j^2 = \sqrt{-1}$). This term is computed at each K "correction" updates, and its modulus is then incorporated in the Total Loss (see line 9 of Algorithm 1).

Details in the blue equations (12) – (17) of Figure 3.

4) Neural Network Update: This step consists in updating (or "correcting") v_i^k, θ_i^k and m_i^k based on their current values, the local power imbalance, and the sum of messages of their direct neighbors. These updates are performed by learnable neural network blocks that are specific to their corresponding "correction" update layer (thus the superscript k), and common to every bus in the power grid. However, the voltage of the buses that are connected to a generator remain fixed at their generator's voltage set point.

Details in the red equations (18) – (20) of Figure 3.

B. Overall architecture of the Graph Neural Solver

Algorithm 1 diagram details all the different operations performed inside the proposed Graph Neural Solver. The term $input$ refers to the following pieces of information: a base apparent power, a list of buses, a list of lines and a list of generators.

1) At first, the algorithm performs an Initialization, that sets all θs to 0 rad, all messages to $[0, \ldots, 0]$, and all voltages v to 1 p.u. (except for buses connected to generators: they are set to their corresponding voltage set points).

2) The Global Active Compensation estimates the value of λ (defined in subsection II-C) to ensure a global active equilibrium of the power grid. It also forces buses that are connected to generators to be at a reactive power equilibrium.

3) The Local Power Imbalance is computed by estimating the mismatch in both active and reactive power at each bus.

4) Then variables v, θ and m (defined in subsection II-D) are updated by a Neural Network Update.

The equations of each of these steps are displayed on Figure 3, in a similar color code. All these operations are fully differentiable, which allows gradients to flow from the Total Loss to each Neural Network Update during the training process. Only the Neural Network Updates are learnt during the training process, while all other parameters remain fixed. Unlike our previous approach detailed in Figure 2, we no longer use encoders and decoders, and directly act on the variables v, θ and m. Indeed, if the model is not learning, only the $(v_i^K, \theta_i^K)_{i=1,\ldots,n}$ are returned.

The outputs of the model are the vs and θs of the K-th correction update.

C. Power grid modelling

The reactive power limits of the generators ($Q_{g,min}$ and $Q_{g,max}$) are not considered here, to be consistent with the default behavior of MATPOWER.

When solving an AC Power Flow system, one has to make sure that the overall active production is equal to the overall active consumption plus all losses caused by Joule's effect on transmission lines. To do so, solvers slightly modify the active productions. There are many ways of doing that : one can choose to modify the active power of a single generator ("slack bus"), or to use all generators proportionally. Another common approach is to assign a coefficient to each generator and have all of them contribute proportionally to maintain a global active equilibrium. However, generators are limited by their respective maximum and minimum active power. When such a threshold is encountered by one generator, all the others have to compensate a little more. This requires to check a number of conditions that scales linearly with the number of generators on the power grid, which would be quite hard to encode into a differentiable neural network.

All those solutions are simplified models of the real-life compensation process that involves economical, geographical, organisational and technological considerations.

The compensation rule we choose needs to be both delocalized (i.e. have all generators contribute), and differentiable (in order to let gradients flow through it during the training process). This is achieved by the following rule:

$$P_{g,i}(\lambda) = \begin{cases} \underline{P}_{g,i} + 2\left(\mathring{P}_{g,i} - \underline{P}_{g,i}\right)\lambda, & \text{if } \lambda < \frac{1}{2} \\ 2\mathring{P}_{g,i} - \overline{P}_{g,i} + 2\left(\overline{P}_{g,i} - \mathring{P}_{g,i}\right)\lambda, & \text{otherwise} \end{cases} \tag{21}$$

All productions of the power grid depend on a single common parameter λ. It ensures that all productions are at their respective minimum values $\underline{P}_{g,i}$ for $\lambda = 0$, at their initial set points $\mathring{P}_{g,i}$ for $\lambda = \frac{1}{2}$ and at their maximum values $\overline{P}_{g,i}$ for $\lambda = 1$. As detailed hereafter, the proposed Graph Neural Solver architecture will update the global parameter λ to constantly ensure a global active equilibrium. The equation to solve to find the λ that will ensure equilibrium is $\sum_{i \in Gen} P_{g,i}(\lambda) = \sum_{i \in Load} P_{l,i} + P_{Joule}$, where P_{Joule} is the total power loss caused by Joule's effect. Since this equation is quite simple, there is a closed form solution which is shown in Equation 7. The impact of λ on the active production of a generator is illustrated in Figure 4.

Constant power generators such as wind turbines and solar panels can be modelled as negative consumptions during the global active compensation step so that they are not affected by λ. However, those should still be considered as generators during the neural network updates, otherwise their v would not be controlled.

D. Message propagation

In addition to well-known variables such as voltage modulus v and voltage angle θ, our algorithm also relies on the propagation of messages m. These vectors have no direct physical meaning. [1] experimentally demonstrated that the propagation of such variables was a suitable way to perform the power flow computation. These messages are defined in $\mathbb{R}^d$, where d is an hyperparameter[2] of the architecture.

This can be seen as a way of encoding in an abstract continuous space the state of a bus. They capture information about the current state, the history of updates, and information about neighbors as well as their respective history. They can be viewed as an implicit memory and neighbor-awareness variable.

E. Training algorithm

The violation of Kirchhoff's law is computed at the end of each "correction" update layer, for each bus of every sample of a minibatch of T samples (let t be the index of a sample). The Total Loss[3] that will be minimized during the training process

[2]Hyperparameters are parameters of a neural network that are not learnt during the training process.

[3]The term "loss" refers to the error of a model in the machine learning literature

$$v_i^0 = \begin{cases} v_{g,i}, & \text{if } i \text{ has a generator} \\ 1, & \text{otherwise} \end{cases} \tag{1}$$

$$\theta_i^0 = 0 \tag{2}$$

$$m_i^0 = [0, \dots, 0]^T \tag{3}$$

$$p_{Joule}^k = \sum_{\substack{i:\text{“from”} \\ j:\text{“to”}}} |v_i^k v_j^k y_{ij} \frac{1}{\tau_{ij}} \left(\sin(\theta_i - \theta_j - \delta_{ij} - \theta_{shift,ij}) + \sin(\theta_j - \theta_i - \delta_{ij} + \theta_{shift,ij})\right) \tag{4}$$

$$+ \left(\frac{v_i^k}{\tau_{ij}}\right)^2 y_{ij} \sin(\delta_{ij}) + \left(v_j^k\right)^2 y_{ij} \sin(\delta_{ij})| \tag{5}$$

$$p_{global}^k = \sum_{i=1}^{N} p_{d,i} + g_{s,i} \left(v_i^k\right)^2 + p_{Joule}^k \tag{6}$$

$$\lambda^k = \begin{cases} \frac{p_{global}^k - \sum_{i=1}^N \underline{p}_{g,i}}{2\left(\sum_{i=1}^N \mathring{p}_{g,i} - \sum_{i=1}^N \underline{p}_{g,i}\right)}, & \text{if } p_{global}^k < \sum_{i=1}^N \mathring{p}_{g,i} \\ \frac{p_{global}^k - 2\sum_{i=1}^N \mathring{p}_{g,i} + \sum_{i=1}^N \overline{p}_{g,i}}{2\left(\sum_{i=1}^N \overline{p}_{g,i} - \sum_{i=1}^N \mathring{p}_{g,i}\right)}, & \text{otherwise} \end{cases} \tag{7}$$

$$p_{g,i}^k = p_{g,i}^k(\lambda^k) \qquad \text{cf eq (21)} \tag{8}$$

$$q_{g,i}^k = \left(q_{d,i} - b_{s,i} \left(v_i^k\right)^2\right) \tag{9}$$

$$- \sum_{\substack{l \in \mathcal{N}(i) \\ i:\text{“from”}}} \left[-v_i^k v_l^k y_{il} \frac{1}{\tau_{ij}} \cos(\theta_i - \theta_j - \delta_{ij} - \theta_{shift,il}) + \left(\frac{v_i^k}{\tau_{il}}\right)^2 (y_{il} \cos(\delta_{il}) - \frac{b_{ij}}{2})\right] \tag{10}$$

$$- \sum_{\substack{l \in \mathcal{N}(i) \\ i:\text{“to”}}} \left[-v_i^k v_l^k y_{il} \frac{1}{\tau_{il}} \cos(\theta_i - \theta_l - \delta_{il} + \theta_{shift,il}) + \left(v_i^k\right)^2 (y_{il} \sin(\delta_{il}) - \frac{b_{il}}{2})\right] \tag{11}$$

$$\Delta p_i^k = \left(p_{g,i}^k - p_{d,i} - g_{s,i} \left(v_i^k\right)^2\right) \tag{12}$$

$$+ \sum_{\substack{l \in \mathcal{N}(i) \\ i:\text{“from”}}} \left[v_i^k v_l^k y_{il} \frac{1}{\tau_{il}} \sin(\theta_i - \theta_l - \delta_{il} - \theta_{shift,il}) + \left(\frac{v_i^k}{\tau_{il}}\right)^2 y_{il} \sin(\delta_{il})\right] \tag{13}$$

$$+ \sum_{\substack{l \in \mathcal{N}(i) \\ i:\text{“to”}}} \left[v_i^k v_l^k y_{il} \frac{1}{\tau_{il}} \sin(\theta_i - \theta_l - \delta_{il} + \theta_{shift,il}) + \left(v_i^k\right)^2 y_{il} \sin(\delta_{il})\right] \tag{14}$$

$$\Delta q_i^k = \left(q_{g,i}^k - q_{d,i} + b_{s,i} \left(v_i^k\right)^2\right) \tag{15}$$

$$+ \sum_{\substack{l \in \mathcal{N}(i) \\ i:\text{“from”}}} \left[-v_i^k v_l^k y_{il} \frac{1}{\tau_{il}} \cos(\theta_i - \theta_l - \delta_{il} - \theta_{shift,il}) + \left(\frac{v_i^k}{\tau_{il}}\right)^2 (y_{il} \cos(\delta_{il}) - \frac{b_{il}}{2})\right] \tag{16}$$

$$+ \sum_{\substack{l \in \mathcal{N}(i) \\ i:\text{“to”}}} \left[-v_i^k v_l^k y_{il} \frac{1}{\tau_{il}} \cos(\theta_i - \theta_l - \delta_{il} + \theta_{shift,il}) + \left(v_i^k\right)^2 (y_{il} \cos(\delta_{il}) - \frac{b_{il}}{2})\right] \tag{17}$$

$$\theta_i^{k+1} = \theta_i^k + L_\theta^k \left(v_i^k, \theta_i^k, \Delta p_i^k, \Delta q_i^k, m_i^k, \sum_{l \in \mathcal{N}(i)} \phi(m_l^k, line_{il})\right) \tag{18}$$

$$v_i^{k+1} = \begin{cases} v_{g,i}, & \text{if } i \text{ has a generator} \\ v_i^k + L_v^k \left(v_i^k, \theta_i^k, \Delta p_i^k, \Delta q_i^k, m_i^k, \sum_{l \in \mathcal{N}(i)} \phi(m_l^k, line_{il})\right), & \text{otherwise} \end{cases} \tag{19}$$

$$m_i^{k+1} = m_i^k + L_m^k \left(v_i^k, \theta_i^k, \Delta p_i^k, \Delta q_i^k, m_i^k, \sum_{l \in \mathcal{N}(i)} \phi(m_l^k, line_{il})\right) \tag{20}$$

Fig. 3. **Graph Neural Solver equations** - The color code is the same as the one used in Algorithm 1, so that each small block has its corresponding equations in the same color. In the last three equations, and for the sake of conciseness we introduce the variable $line_{il} = (r_{il}, x_{il}, b_{c,il}, \tau_{il}, \theta_{shift,il})$ which contains all physical characteristics of line il. The function ϕ is another neural network that is trained jointly with the other neural network blocks.

Algorithm 1 Graph Neural Solver architecture

1: $(v_i^0, \theta_i^0, m_i^0)_{i=1,\dots,n} \leftarrow$ Initialization(input)
2: $\lambda^0 \leftarrow$ Global Active Compensation(input, $(v_i^0, \theta_i^0, m_i^0)_{i=1,\dots,n}$)
3: $(\Delta p_i^0, \Delta q_i^0)_{i=1,\dots,n} \leftarrow$ Local Power Imbalance(input, $(v_i^0, \theta_i^0, m_i^0)_{i=1,\dots,n}$, λ^0)
4: Total Loss $\leftarrow 0$
5: **for** k=1,..., K **do**
6: $\quad (v_i^k, \theta,_i^k, m_i^k)_{i=1,\dots,n} \leftarrow$Neural Network Update(input, $(v_i^{k-1}, \theta_i^{k-1}, m_i^{k-1}, \Delta p_i^{k-1}, \Delta q_i^{k-1})_{i=1,\dots,n}$)
7: $\quad \lambda^k \leftarrow$ Global Active Compensation(input, $(v_i^k, \theta_i^k, m_i^k)_{i=1,\dots,n}$)
8: $\quad (\Delta p_i^k, \Delta q_i^k)_{i=1,\dots,n} \leftarrow$ Local Power Imbalance(input, $(v_i^{k-1}, \theta_i^{k-1}, m_i^{k-1})_{i=1,\dots,n}$, λ^{k-1})
9: $\quad$ Total Loss $\leftarrow$ Total Loss $+\gamma^{K-k} \sum_{i=1}^n (\Delta p_i^k)^2 + (\Delta q_i^k)^2$
10: **end for**
11: **if** model is training **then**
12: $\quad$ Perform gradient descent step: minimize Total Loss by changing neural networks
13: **else**
14: $\quad$ **return** $(v_i^K, \theta_i^K)_{i=1,\dots,n}$

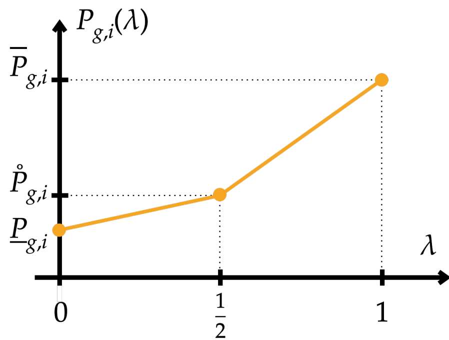

Fig. 4. **Active production of generator** i - The active power of every generator of a power grid instance depends on a global parameter λ.

is a weighted average of those Kirchhoff's law violations and is defined by:

$$Total\ Loss = \frac{1}{T}\sum_{t=1}^{T}\sum_{k=1}^{K}\frac{\gamma^{K-k}}{N}\sum_{i=1}^{N}|\Delta S_{i,t}^k|^2 \quad (22)$$

$$= \frac{1}{T}\sum_{t=1}^{T}\sum_{k=1}^{K}\frac{\gamma^{K-k}}{N}\sum_{i=1}^{N}\left[\left(\Delta P_{i,t}^k\right)^2 + \left(\Delta Q_{i,t}^k\right)^2\right] \quad (23)$$

The way this loss is integrated into the overall architecture is shown on line 9 of Algorithm 1.

The discount factor γ in Equation (23) puts a stronger emphasis on the accuracy of the later "correction" update layers, while still allowing gradients to flow to the first few "correction" update layers. We experimentally observed that penalizing the physical laws violation at every "correction" update with this discount factor, instead of only the last update, helped converging to a better solution. γ is a hyperparameter of the model that is chosen empirically.

During the training process, a set of T different power grids (each defined by injections, topology and line characteristics) is fed to the model (the $input$ variable of Algorithm 1). The model performs a prediction on each of these T power grids for v and θ of each bus. The Total Loss (i.e. physical laws violations) is estimated using the output of each Local Power Imbalance (cf. the blue functions of Algorithm 1). Then, the gradients of this Total Loss with regards to the weights of each Neural Network Update L_v^k, L_θ^k and L_m^k (cf. the red functions of Algorithm 1) is estimated using the back-propagation rule [31]. The neural network ϕ shown in equations (18) – (20) is trained jointly with the others in the same process. These gradients are then used to modify the weights of the Neural Network Updates (gradient descent step). This process is repeated on randomly sampled power grids, until the Total Loss no longer decreases.

F. *Compatibility with varying input size*

As detailed in our first paper about Graph Neural Solver, any instance of our model is mathematically compatible with any size and shape of power grid. Unlike fully connected neural networks (similar to the one in Figure 1), a graph neural network is able to deal with changes in the shape of its input data.

III. Experiments

This section details the experimental protocol used to validate the proposed approach, the way data is generated from standard power grids, and discusses the obtained results, as well as the current limits of the proposed architecture.

To validate the approach, we need a dataset of different power grids, with both injections and grid information. To generate this dataset, we take several standard IEEE power grids, and add some noise on both injections and power line parameters, which allows to generate a variety of situations. For every power line, r, x and b are uniformly sampled between 90% and 110% of their initial values. The ratio τ of each line (which are seen as transformers, see Appendix) are uniformly sampled between 0.8 and 1.2, and the shifting phase θ_{shift} is uniformly sampled between 0.2 and -0.2 rad. The maximum and minimum values of active power of each generator remain unchanged, while their voltage set points v_g are uniformly sampled between 0.95 p.u. and 1.05 p.u., and their initial active powers $\mathring{P}_g$ are uniformly sampled between 25% and 75% of their allowed range. The active demands P_d

are first uniformly sampled between 50% and 150% of their initial values, and are then all multiplied by a common factor to make sure that the demand is equal to the consumption. The reactive demands Q_d are uniformly sampled between 50% and 150% of their initial values. This choice of noise amplitude was motivated by the fact that it created situations outside of the validity domain of the DC-approximation (as will be experimentally shown by large errors for the latter method).

The proposed Graph Neural Solver is compared to two baselines: the DC approximation, and the Newton-Raphson methods of PYPOWER (which is a port of MATPOWER in python). All of these 3 methods solve different versions of the same problem (the DC approximation ignores the reactive part of the problem and the Graph Neural Solver uses an atypical global active equilibrium mechanism). We do not have access to a "ground truth", so it is impossible to compare all three models to the same thing. However, we believe that the Newton-Raphson method can be considered as the most precise and trustworthy, so we have decided to compute for the two remaining models the percentage of error in terms of active flow (which are easily deduced from v and θ) with regards to the Newton-Raphson solver.[4]

A. Standard experiment

In this first experiment, our models are tested on the same neighborhood they were trained on. We trained 10 distinct instances of the proposed Graph Neural Solver (during 10,000 learning iterations) on the neighborhoods of the following standard IEEE power grids: case9, case14, case30 and case118. The same set of hyperparameters is used for each of these instances: the amount of correction updates K is set to 30, the latent dimension d (which is both the size of the messages, and the size of latent space of neural networks) is set to 10, each learning block has 2 layers and leaky ReLU activation function. We choose $\gamma = 0.9$. These values were chosen empirically after a random hyperparameter search. The optimizer used is Adam with standard Tensorflow parameters. Each instance was trained using a Nvidia Titan XP GPU. A learning curve from these experiments is displayed in Figure 5. The test set contains only power grids that converge with a Newton-Raphson solver.

The results of these experiments are shown on Figure 6. On the x-axis is diplayed the mean computational time required to perform one load flow, and on the y-axis is shown the % of error (20th percentile, median and 80th percentile) in terms of active flow with regards to the output of a Newton-Raphson solver. The percentage of error can explode for small flows, so we only kept the 50% of largest values in absolute value. Circles correspond to case9, stars to case14, triangles to case30 and squares to case118. Since the % of error is a comparison with the Newton-Raphson, the error of this solver are indeed at 0%. One can observe from this plot that our proposed method offers a significant gain in terms of computational time, while achieving a better accuracy than the DC approximation (keeping in mind that the scale is logarithmic). Orders of magnitude for the error of the DC approximation can seem surprisingly large. However, the noise we apply on the standard power grids makes most of the generated data points "unrealistic", thus pushing the snapshots far from the domain of validity of the DC approximation.

[4]One should keep in mind that our proposed architecture does not try to imitate a Newton-Raphson solver.

In Figure 7 are displayed 2d histograms (logarithmic color scale): on the left part, the predicted active line flow (MW) of our Graph Neural Solver is plotted against the active line flow (MW) output by Newton-Raphson for all 4 power grid neighborhoods. On the right part, the output of the DC-approximation is plotted against the output of a Newton-Raphson. One can observe that our proposed architecture has a higher correlation with the output of a Newton-Raphson solver than the DC approximation on the test set (0.995 for GNS vs. 0.876 for DC on case188).

Figure 8 offers an example of how our proposed architecture updates the v and θ of a power grid, and how it affects the violation of Kirchhoff's law. It starts with a simple initialization, which cause large physical laws violation (many red buses). After 10 "correction" updates, the model has changed the vs and θs so as to decrease the physical laws violation (many green buses). Figure 8 presents how a previously trained Graph Neural Solver modifies its predictions across its "correction" updates, and how it succeeds in iteratively reducing the violation of Kirchhoff's law. The instance of Graph Neural Solver has 10 "correction" updates ($K = 10$). (1) On the left panel are displayed the voltages v in p.u.. One can see that it is uniformly initialized at 1 p.u. except for buses that are connected to generators (those are set to their generator's voltage set points). The voltage then evolves across "correction" updates until it reaches its final prediction at "correction" update 10. (2) On the middle left panel are displayed the phase angles θ in rad. It is at first initialized at 0 rad, and then evolves across "correction" updates until it reaches its final prediction. (3) On the two right panels are displayed the Kirchhoff's law violation $|\Delta S|$ in MVA. The boxplot displays the minimum, 25th percentile, median, 75th percentile and maximum value of the very same $|\Delta S|$ displayed on the graph. One should keep in mind that the quality of a prediction depends on the maximum value of $|\Delta S|$ on the power grid. This maximum value starts quite high at "correction" update 0 (i.e. before any correction is applied, there are many red buses), and decreases with the amount of "correction" update (all buses are green at the end).

B. Generalization from one grid to another

As demonstrated in previous work [1], a key property of the proposed architecture is the ability to learn on one power grid and perform decently on another, regardless of their sizes or shapes. Even though this aspect is not the key point of the present paper, we display on Figure 9 some statistics about the percentage of error with regards to a Newton-Raphson solver for models that were trained in the neighborhood of one power grid and then tested on the neighborhood of another.

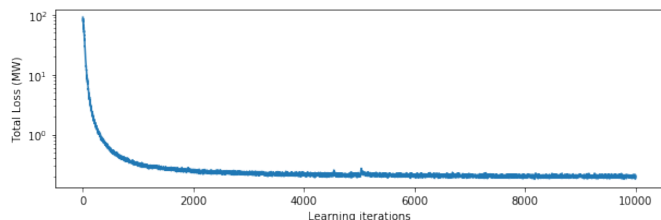

Fig. 5. **Total Loss** - Evolution of the Total Loss (see equations 22 and 23) during the training of one instance of our Graph Neural Solver. It was trained on a single Nvidia Titan XP GPU, which took approximately 80 minutes.

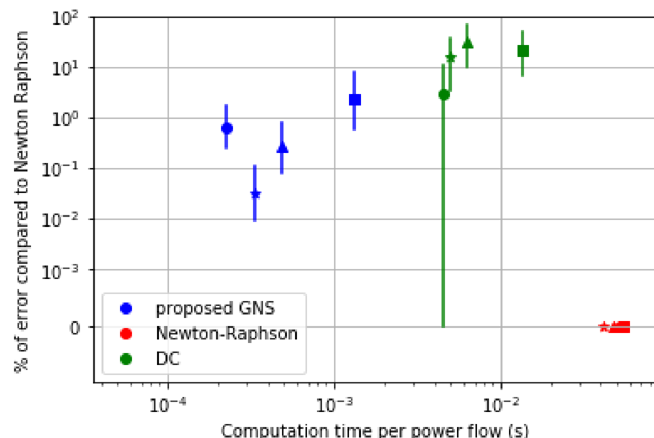

Fig. 6. **Computational time vs. Method accuracy** - Our proposed method provides a substantial gain in terms of computational time, while keeping the error reasonably low.

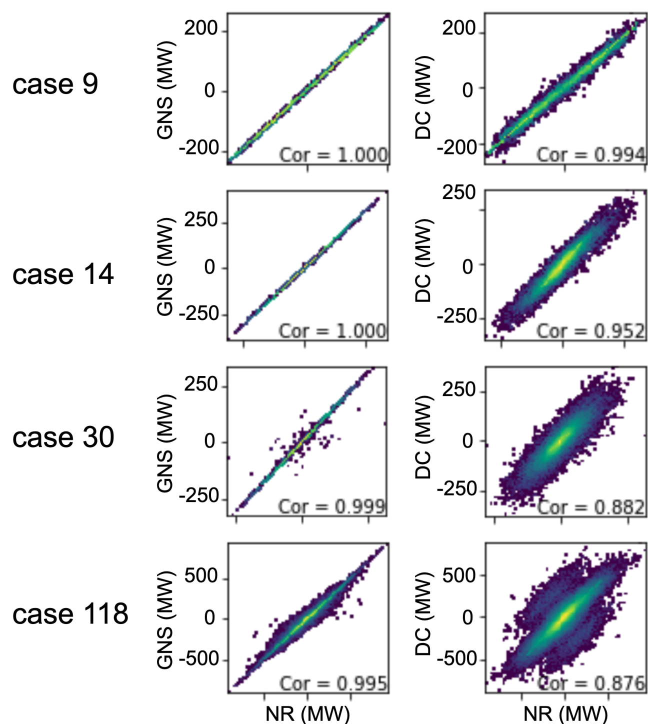

Fig. 7. **Correlation with Newton-Raphson output** - Our proposed method is generally more linearly correlated with the Newton-Raphson.

The results on the diagonal (red points) are indeed the same as those that were shown in Figure 6. From this figure we can see that model instances trained on large power grids tend to have a good accuracy on smaller power grids, while the opposite is not necessarily true.

C. Discussion

A major limitation of the current work is the lack of empirical proof of the scalability of the approach on real-life power grids, and is an open area for future research.

Some preliminary work has been done as preparation of the present paper to generate power grids with completely random injections, topologies and line characteristics so as to observe a much broader domain during the training of the model. However, it appears that many of those generated power grids were unrealistic and lead to non-converging situations. We have thus chosen to stay within the neighborhood of well known power grids.

The goal of the present paper is to provide empirical validation of the viability of the approach, and to try to develop some intuitions about how it works. The theoretical aspects and properties should be further developed in future work, so as to build more confidence in the behavior of this novel method as well as its proper use.

An important challenge that will be faced in the near future is the implementation of reactive power limits for generators. The most straightforward option would be to hard-code an iterative behavior that checks if limits are crossed, in which case a new computation would be run. Another option would be to incorporate those reactive power limits as input of our neural network architecture, and incorporating the crossing of such a limit directly into the loss.

IV. Conclusion

The main contribution of this paper is to propose a novel method that lies at the direct interface between artificial intelligence and optimization to solve the power flow problem, while achieving a substantial gain in terms of computational time. A traditional algorithm such as Newton-Raphson has access to the global objective function and proceeds by iteratively minimizing it by directly acting on the variables. Our method is quite different: the model (which is basically a long differentiable function) tries to predict v and θ everywhere, based on the inputs (injections and power grid characteristics). During training, the model tries to minimize the global objective function, not by directly acting on the variables v and θ like traditional optimization, but by altering itself (i.e the weights of its neural network blocks). The model learns a behavior that minimizes at test time (inference time) the desired objective, without solving an optimization problem.

We experimentally demonstrated that our proposed architecture is able to perform predictions faster than methods such as the DC approximation or Newton-Raphson (up to 2 orders of magnitude). Predictions in terms of active flow are very similar to the output of a Newton Raphson method (correlation of 0.995 on case118).

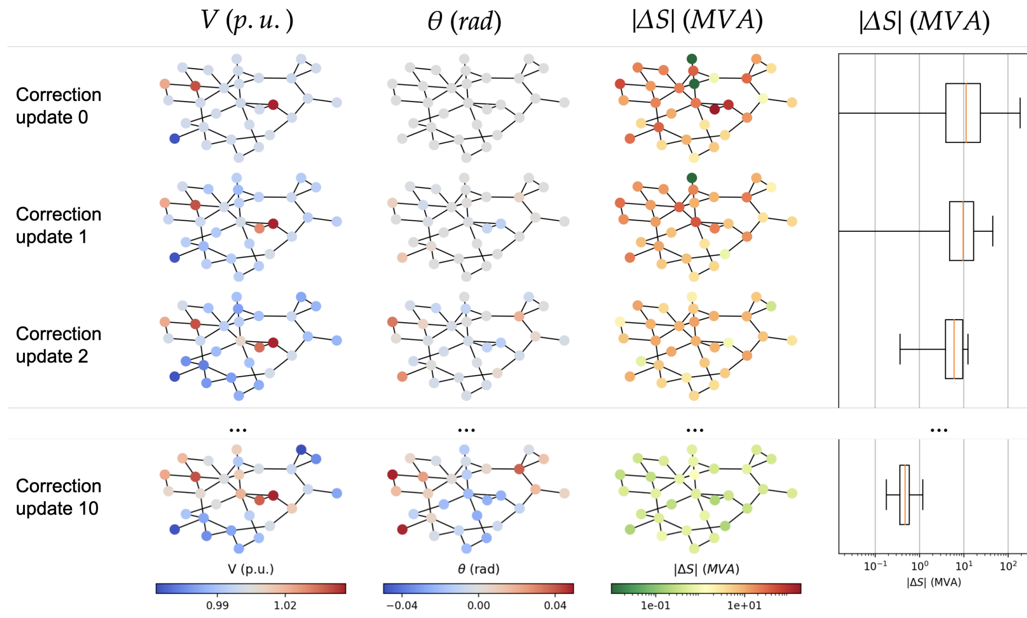

Fig. 8. **Graph Neural Solver on the IEEE case30** - A previously trained instance of Graph Neural Solver minimizes the violation of Kirchhoff's law by altering its predictions for v and θ.

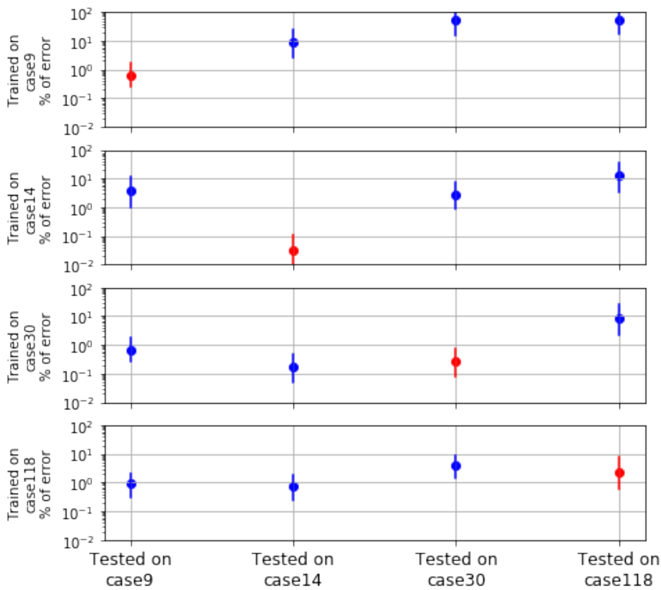

Fig. 9. **From one case to another** - In this plot, we display the % of error (20th percentile, median, 80th percentile) in active flow with regards to a Newton-Raphson solver for Graph Neural Solver models that were trained on the neighborhood of one power grid, and then tested on another.

An interesting property of our Graph Neural Solver is the fact that it will always predict something. Some early experiments showed that in the case where a classical solver simply does not converge, the graph neural solver shows the voltage magnitudes collapse everywhere on the grid. This empirical observation should be investigated in the future, but falls out of the scope of the present paper.

Our Graph Neural Solver is a fully differentiable function that predicts power flows. This differentiability could open the door for gradient-descent optimization and control of the power grid. This could also become a part of a larger decision making process.

Another possible avenue would be to train a decentralized controller to perform some kind of optimization (such as minimizing power losses caused by Joule's effect), jointly with the learning of the Graph Neural Solver. There would be two different types of gradients flowing through the architecture: one that aims at minimizing the violation of Kirchhoff's law and that affect the neural network updates of v and θ, and another one that would minimize another objective function, and that would affect the decentralized controllers.

Acknowledgment

We would like to thank Patrick Panciatici, Louis Wehenkel and Patrick Pérez for insightful discussions. We would like to thank Vincent Barbesant, Laure Crochepierre, Stéphane Fliscounakis, Camille Pache and Wojciech Sitarz for their attentive reviews and remarks.

References

[1] Balthazar Donon, Benjamin Donnot, Isabelle Guyon, and Antoine Marot. Graph neural solver for power systems. In *International Joint Conference on Neural Networks*, 2019.

[2] D2.2 - guidelines for implementing the new reliability assessment and optimization methodology. *GARPUR consortium*, 2016.

[3] E. Karangelos and L. Wehenkel. Probabilistic reliability management approach and criteria for power system real-time operation. In *Power Systems Computation Conference (PSCC)*, 2016.

[4] T.T. Nguyen. Neural network load-flow. *IEE Proceedings - Generation, Transmission and Distribution*, 1995.

[5] Benjamin Donnot, Isabelle Guyon, Marc Schoenauer, Antoine Marot, and Patrick Panciatici. Fast Power system security analysis with Guided Dropout. In *European Symposium on Artificial Neural Networks*, 2018.

[6] B. Donnot, B. Donon, I. Guyon, L. Zhengying, A. Marot, P. Panciatici, and M. Schoenauer. Leap nets for power grid perturbations. In *European Symposium on Artificial Neural Networks*, 2019.

[7] L. Duchesne, E. Karangelos, and L. Wehenkel. Using machine learning to enable probabilistic reliability assessment in operation planning. In *Power Systems Computation Conference (PSCC)*, 2018.
[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
[9] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 2015.
[10] Thomas N. Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard S. Zemel. Neural relational inference for interacting systems. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.
[11] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. *ArXiv Computing Research Repository*, 2016.
[12] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum convolutional neural networks. 2019.
[13] Kyle Mills, Michael Spanner, and Isaac Tamblyn. Deep learning and the schrödinger equation. *Physical Review A*, 2017.
[14] Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems 30*. 2017.
[15] Peter W. Battaglia and al. Relational inductive biases, deep learning, and graph networks. *ArXiv Computing Research Repository*, 2018.
[16] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks and Learning Systems*, 2009.
[17] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. *ArXiv Computing Research Repository*, 2015.
[18] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *ArXiv Computing Research Repository*, 2017.
[19] Roei Herzig, Moshiko Raboh, Gal Chechik, Jonathan Berant, and Amir Globerson. Mapping images to scene graphs with permutation-invariant structured prediction. *ArXiv Computing Research Repository*, 2018.
[20] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. *ArXiv Computing Research Repository*, 2017.
[21] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.
[22] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29*. 2016.
[23] David K. Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *ArXiv Computing Research Repository*, 2015.
[24] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *ArXiv Computing Research Repository*, 2017.
[25] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ArXiv Computing Research Repository*, 2016.
[26] Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy P. Lillicrap. A simple neural network module for relational reasoning. *ArXiv Computing Research Repository*, 2017.
[27] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 2017.
[28] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 2010.
[29] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
[30] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas. Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 2011.
[31] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*. 2014.

APPENDIX

In this appendix we give more details about the MATLAB notations and modelling that we use.

*1) **Base apparent power**:* Each power grid instance has a parameter $baseMVA$ in MVA which will serve to normalize the power. We will denote by P, Q, S the active, reactive and complex power in respectively MW, MVAr and MVA, and by p, q and s their equivalents normalized by $baseMVA$. The same convention is used for the shunt active and reactive power G_s and B_s (thus denoted by g_s and b_s in their normalized versions), and should not be mistaken with the total charging susceptance of a line b_c.

*2) **Bus**:* Loads and shunts are encoded at the bus. A bus is thus defined by: its index i, its active load $P_{d,i}$ (MW), its reactive load $Q_{d,i}$ (MVAr), its active shunt load $G_{s,i}$ (MW at 1.0 p.u.) and its reactive shunt load $-B_{s,i}$ (MVAr at 1.0 p.u.).

*3) **Line**:* MATPOWER adopts a common framework for transmission lines and transformers. A line is thus defined by: its "from" bus index, its "to" bus index, its resistance r (p.u.), its reactance x (p.u.), its total charging susceptance b_c (p.u.), its transformation ratio τ (p.u.) and its shift angle θ_{shift} (rad). Transformers are on the "from" side.

We also use the convention to write each of these quantities with the subscript ij to denote the line between buses i and j. Moreover we define the following quantities: $y_{ij} = \frac{1}{\sqrt{r_{ij}^2 + x_{ij}^2}}$ and $\delta_{ij} = atan2(r_{ij}, x_{ij})$.

*4) **Generator**:* A generator is defined by: its bus index i, its maximum provided active power $\overline{P}_{g,i}$ (MW), its minimum provided active power $\underline{P}_{g,i}$ (MW), its active power set point $\mathring{P}_{g,i}$ (MW) and its voltage set point $V_{g,i}$ (p.u.).