Graph Neural Solver for Power Systems

Balthazar Donon, Benjamin Donnot, Isabelle Guyon, Antoine Marot

Graph Neural Solver for Power Systems

Balthazar Donon
R&D department, RTE
& UPSud/INRIA Université Paris-Saclay
Paris, France
balthazar.donon@rte-france.com

Benjamin Donnot
R&D department, RTE
& UPSud/INRIA Université Paris-Saclay
Paris, France
benjamin.donnot@rte-france.com

Isabelle Guyon
TAU group of Lab. de Res. en Informatique
UPSud/INRIA Universié Paris-Saclay
Paris, France
iguyon@lri.fr

Antoine Marot
R&D department
RTE
Paris, France
antoine.marot@rte-france.com

***Abstract*—We propose a neural network architecture that emulates the behavior of a physics solver that solves electricity differential equations to compute electricity flow in power grids (so-called "load flow"). Load flow computation is a well studied and understood problem, but current methods (based on Newton-Raphson) are slow. With increasing usage expectations of the current infrastructure, it is important to find methods to accelerate computations. One avenue we are pursuing in this paper is to use proxies based on "graph neural networks". In contrast with previous neural network approaches, which could only handle fixed grid topologies, our novel graph-based method, trained on data from power grids of a given size, generalizes to larger or smaller ones. We experimentally demonstrate viability of the method on randomly connected artificial grids of size 30 nodes. We achieve better accuracy than the DC-approximation (a standard benchmark linearizing physical equations) on random power grids whose size range from 10 nodes to 110 nodes, the scale of real-world power grids. Our neural network learns to solve the *load flow* problem without overfitting to a specific instance of the problem.**

***Index Terms*—Graph Neural Solver, Neural Solver, Graph Neural Net, Power Systems**

I. Background & Motivations

TSOs (Transmission System Operators) such as RTE (Réseau de Transport d'électricité) need to ensure the security and resilience of power grids. By transporting electricity across states, countries, or continents, they are vital components of modern societies, playing the central economical and societal role to supply power reliably to industries, services, and consumers. In particular they should avoid "blackouts". Currently, TSOs perform security analyses by using "load flow" solvers based on the physical equations of the system [13]. Such solvers compute the flows of electricity through each line of a power grid using physical laws depending on:

- the power grid topology, i.e. the way the electrical nodes are interconnected;
- the amount and location of power being produced or consumed (so-called "injections");
- the physical properties of the power lines.

These load flow solvers use Newton-Raphson optimization methods [18], [19] to iteratively satisfy Kirchhoff's laws (conservation of energy) by reducing progressively the mismatch between ingoing and outgoing power in every electrical node. Although load flow solvers are more accurate and better understood than neural networks, they are comparatively slower, leaving room for use of the latter for fast screening, in conjunction with load flow solvers [7], [8]. In particular, speeding up computation would allow TSOs to perform more comprehensive security analyses, and thus increase the quality of services or make a tighter use of existing infrastructure and reduce risks. This would lend itself to a probabilistic approach of security analysis emphasizing rare events (see e.g. [10]).

While pioneer work in the area has demonstrated feasibility of the use of neural networks to estimate power flow, all methods developed prior to our work exposed in this paper are geared towards a given grid topology. They are dedicated to one instance (or a small set of grid instances) and thus do not actually learn how to perform a general load flow on every grid topology.

Our work is in line with Donnot et al. in [9] who proposed a method capable of generalizing to a set of power grid topologies, which remain close to a reference topology. However this method is limited to small perturbations and cannot generalize to completely different grids.

The main issue of former approaches is that they do not exploit the graph structure of the data, and ignore the knowledge of the underlying physics. As explained in [1], one should use "relational inductive bias" to guide the learning process. Our proposed architecture aims to achieve "combinatorial generalization" by using elementary learning blocks that have been laid out based on our knowledge and understanding of the load flow problem. We apply a novel class of algorithms combining deep learning and knowledge about graph structure: Graph Neural Network. This class of artificial neural networks was first introduced by F. Scarselli in [26] and further developed in [20] and [12]. The algorithms operate on network structures by iteratively propagating the influence of vertices through edges.

The architecture can be seen as a generalization of convolutional neural networks to graph structures, by unfolding a finite number of iterations. Theoretical properties have been further developed in [15], [28].

Prior to our work, such methods have been successfully applied to various problems that deal with graph structures, as well as problems that do not explicitly exhibit graph-like structures : classification of graphs [3], [5], [11], classification of nodes [14], [17], and relational reasoning [25]. Recent work such as [2], [16], [21] unveil the emergence of hybrid approaches that rely on deep learning and structure knowledge.

Recently, the use of fast neural solvers based on AI for physics problems has begun to develop, as it could provide much faster tools for simulation and design for complex problems. Computational Fluid Dynamics computations have been successfully accelerated by Tompson et al. in [27] by replacing a process that always estimates the same function but at different locations by a neural network. Ling et al. applied Deep Learning to a Reynolds averaged turbulence modelling problem in [22]. It has also been applied to the Schrödinger equation with success by Mills at al. in [23]. Exploiting the graph structures of the physics problems, and drawing inspiration from these efforts to model physics using deep learning seems to be a good direction towards fast neural approximations that learn to solve any instance of a given problem, while not being dedicated to only one instance of it.

The main contribution of this paper is therefore to devise a neural network architecture allowing to predict accurately power flows, without specializing to a given grid topology. By design, our proposed architecture achieves zero-shot learning [24] on novel grid topologies, as confirmed experimentally: After being trained on random grid topologies of constant size, state-of-the-art prediction accuracy is attained on both smaller and larger power grids (of any type). It is completely in line with the approach developed in [1] as it combines the relational structure between the power line and agnostic neural network blocks that are intricately laid out.

This paper is organized as follows. First we introduce notations and concepts about the load flow problem, and develop our proposed Graph Neural Solver architecture. We then present three different experiments that gradually outline the ability of our proposed architecture to generalize to power grid sizes that have never been encountered during training. Finally, we talk about the limits of the current architecture and give directions for future investigations towards an even more robust neural solver for power systems.

II. Proposed Methodology

In this section we state the problem and introduce our approach and notations. As illustrated with the toy example of Figure 1, the problem is, given injections (productions and consumptions) inj_1, inj_2, inj_3, to compute the flows of electricity in all lines l_1, l_2, l_3, l_4. In what follows, for simplicity, all lines will be assumed to share the same physical characteristics.

Our neural network architecture was developed while keeping in mind a few constraints. First of all, we want an architecture that learns a strategy to solve any power flow, and does not specialize to any specific instance of the problem. This concerns the number of electrical nodes, transmission lines, productions or consumptions on the power grid. Therefore, the neural network needs to embed information about line interconnections, and make use of modular generic learning blocks. The amount of learning blocks, as well as their internal dimensions have to be independent from any power grid specific characteristics.

Secondly, while power grids are often represented as a graph of nodes (so-called "buses") interconnected by power lines, the mathematical treatment is simplified by noting that the topological invariant is the set of lines not the set of nodes (which can vary when line interconnections are changed). Hence, we work on the dual graph structure of interconnected power lines (See Figure 1).

Thirdly, we want to emulate the behavior of an AC power flow solver taking into account physical line characteristics, although we restrict ourselves to power grids where line impedances are all equal. Since there are losses in the power lines, the inflow via one side does not equal to the outflow via the other side. Therefore we have to distinguish between line extremities and origins (denoted in Figure 1 by resp. "ex" and "or"). This choice can also be justified by the fact the the flow through a power line is oriented. This forces us to consider multiple adjacency matrices, that are introduced below.

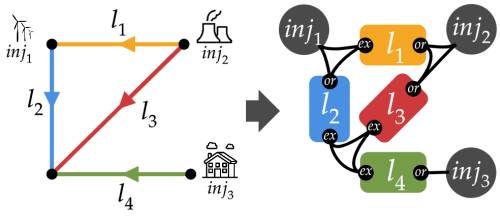

Fig. 1. **Construction of the power lines graph** - This schematics shows how one goes from the classical graph of electrical nodes to the equivalent graph of transmission lines. In the first representation, the nodes are connected through electrical lines, while in the second, power lines are seen as vertices of a graph, and electrical nodes as edges. Our neural network architecture considers the second representation. One should also notice the fact that each power line has an origin (called *or*) and an extremity (called *ex*). For the sake of consistency and understandability, this toy example will be used throughout the whole paper.

A. Notations

This architecture takes as inputs 3 different types of variables:

- **Injections** X: The input vector X concatenates information about the electrical power that is being produced and consumed everywhere on the grid.
  Specifically, each **production** $p \in \mathcal{P}$ is defined by an active power infeed P_p (in MegaWatts) and a voltage infeed V_p (in Volts). Therefore each production $p \in \mathcal{P}$ is defined by a 2-dimensional information.

Similarly, a **consumption** $c \in \mathcal{C}$ is defined by an active power consumption P_c (in MegaWatts) and a reactive power consumption Q_c (in MegaVolt-Amps reactive). Each consumption $c \in \mathcal{C}$ is thus also defined by a 2-dimensional information.

We denote by n_{in} the total number of injections: $n_{in} = |\mathcal{P}| + |\mathcal{C}|$. Each injection has an information in $d_{in} = 2$ dimensions. Therefore, $X \in \mathbb{R}^{n_{in} \times d_{in}}$ is a vector that concatenates all these injection characteristics.

- **Lines adjacency matrices** A_{or} and A_{ex}: Since we model transmission lines as bipolar objects, we need to make a distinction between the extremity and the origin of each transmission line. Each connection between two power lines can thus be of four different types. Let or^i and ex^i be respectively the origin and the extremity of line i.

$$\begin{aligned} A_{or}^{i,j} &= 1 && \text{if } or^i \text{ is connected to } or^j && (1)\\ &= -1 && \text{if } or^i \text{ is connected to } ex^j && (2)\\ &= 0 && \text{otherwise} && (3)\\ A_{ex}^{i,j} &= 1 && \text{if } ex^i \text{ is connected to } or^j && (4)\\ &= -1 && \text{if } ex^i \text{ is connected to } ex^j && (5)\\ &= 0 && \text{otherwise} && (6) \end{aligned}$$

  Note that the choice of the polarity of the lines is arbitrary. Changing it only changes the sign of current flowing.

- **Injections adjacency matrix** A_{inj} : This matrix encodes the way injections (productions and consumptions) are connected to lines, and through which pole (origin or extremity). Let inj^i be the injection i (regardless of it being a production or consumption).

$$\begin{aligned} A_{inj}^{i,j} &= 1 && \text{if } or^i \text{ is connected to } inj^j && (7)\\ &= -1 && \text{if } ex^i \text{ is connected to } inj^j && (8)\\ &= 0 && \text{otherwise} && (9) \end{aligned}$$

The output we want to predict is the flows through the lines (both in Amps A and in MegaWatts MW) at the origin and the extremity of every line, which we denote by vector $Y \in \mathbb{R}^{n \times d_{out}}$, where for each of the n lines, we try to predict flow information in $d_{out} = 4$ dimensions.

The system we are interested in emulating is therefore:

$$\boxed{Y = S(X, A_{or}, A_{ex}, A_{inj})} \quad (10)$$

B. Graph Neural Solver architecture

Our neural network architecture can be written:

$$NN : \mathbb{R}^{n_{in} \times d_{in}} \to \mathbb{R}^{n \times d_{out}} \quad (11)$$

$$X \mapsto \hat{Y} \quad (12)$$

where $n_{in} = |\mathcal{P}| + |\mathcal{C}|$ is the number of injections, n is the number of power lines in the Power Grid, d_{in} is the dimensionality of the input information of each injection, and d_{out} is the dimensionality of the output for each power line. In the toy example of Figure 1, we have $n_{in} = 3$, $n = 4$, $d_{in} = 2$ and $d_{out} = 4$. The Graph Neural Solver operates on 2D matrices: the input is a $n_{in} \times d_{in}$ matrix and the output is a $n \times d_{out}$ matrix. In what follows we will use compact notations: $\mathbf{F}$ will denote a vectorial function applying the same function F to a number of inputs (e.g. power flows or injections) and $\mathbf{A}$ a function that is a right side multiplication with the corresponding adjacency matrix A.

The overall workflow of the approach, which is described in Figures 5 and 6 is described in more details below. It consists of 3 steps: Embedding, propagation, and decoding. The embedding step transforms input space in an abstract vector space, taking into account the grid topology. The propagation space implements a relaxation procedure to compute the flows in a finite number of iterations unfolded in time, and the decoding step transforms back results into our output space.

*a) **Embedding**:* This step aims at embedding the initial information contained in each injection (d_{in}-dimensional) into a d-dimensional space. It applies the same neural network $E : \mathbb{R}^{d_{in}} \to \mathbb{R}^d$ to each injection. It then proceeds to send this information from the injections to the transmission lines they are connected to. Mathematically, this consists in a right-side matrix multiplication with the adjacency matrix A_{inj}.

$$H^{(0)} = \mathbf{A}_{inj} \circ \mathbf{E}(X) \quad (13)$$

The dimension d is a hyperparameter of our architecture. One should notice that $\mathbf{E}$ affects each of the injections similarly, while $\mathbf{A}_{inj}$ affects each of the d dimensions similarly. Since we have two different types of injections (productions and consumptions), we use two different types of embedding functions: E_p for productions and E_c for consumptions. This step is important for two reasons. First we want the information of both productions and consumptions to be compatible (they originally have different meanings and units), so they need to be embedded into a consistent latent space. Secondly, we experimentally observed that using a large embedding space ($d \approx 100$) provides a faster learning.

*b) **Propagation**:* In this step, we iteratively update the latent state of each of the n power lines by performing latent leaps that depend on the value of their direct neighbors. Because of the bipolar nature of power lines, we consider separately the influence of neighbors connected to their origins, and neighbors connected to their extremities. This is the reason why in Figures 6 and 5 we consider two separate entries into functions $L^{(k)}$. At each iteration k, the same neural network $L^{(k)} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is used to update the embedding of each of the power lines. The aim of the right-side matrix multiplications by A_{or} and A_{ex} is to sum the information of the direct neighbors connected to respectively the origin and the extremity of each power line. Moreover, this sum is weighted by ± 1 depending on whether the neighbors are connected by their own origin or extremity.

$$\begin{aligned} H^{(k+1)} &= H^{(k)} + \mathbf{L}^{(k)}(A_{or}H^{(k)}, A_{ex}H^{(k)}) && (14)\\ &\equiv (\mathbf{I} + \mathbf{L}^{(k)} \circ (\mathbf{A}_{or}, \mathbf{A}_{ex}))(H^{(k)}) && (15) \end{aligned}$$

for $k \in \{0, \ldots, K-1\}$, where I is the identity function.

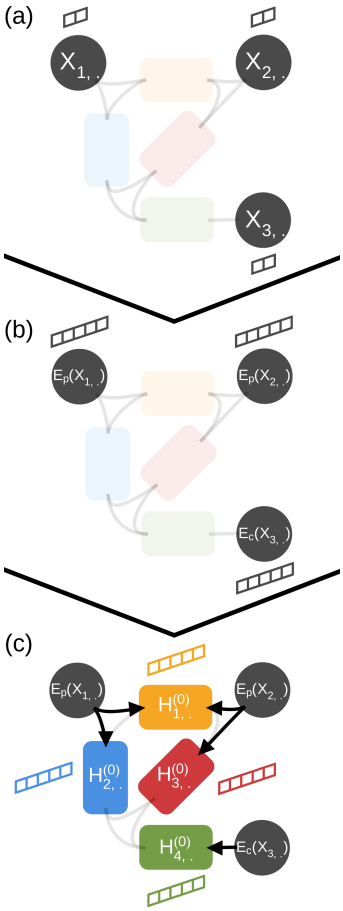

Fig. 2. **Embedding step** - The input data is a $n_{in} \times d_{in}$ (i.e. 3×2 here) matrix (a). We first embed the d_{in}-dimensional information of each of the n_{in} injection into a d-dimensional space. This results in a $n_{in} \times d$ matrix (b). We then proceed to assign the information of each of the n_{in} injections, to the n power lines they are respectively connected to (see Figure 6). We then end up with a $n_{in} \times d_{in}$ (i.e. 4×5 here) matrix (c). This is based on the toy example from Figure 1.

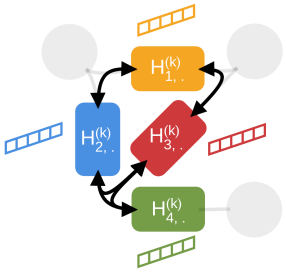

Fig. 3. **Propagation step** - The embedding of each line is iteratively updated depending on the embeddings of its direct neighbors. There is no direct propagation between power lines 1 and 4.

*c) **Decoding**:* This step consists in a simple decoding from the embedding space to the output space. It applies the same function $D : \mathbb{R}^d \to \mathbb{R}^{d_{out}}$ to each of the n power lines. There is no exchange of information between power lines.

$$\hat{Y} = \mathbf{D}(H^{(K)}) \tag{16}$$

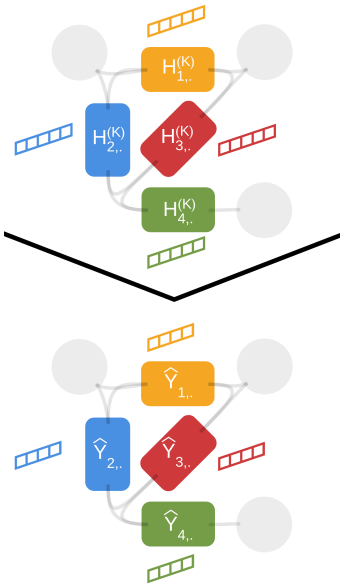

Fig. 4. **Decoding step** - The same decoding function D is applied to the embedding of each power line. There is no exchange between power lines. $H^{(K)}$ is a $n \times d$ (i.e. 4×5 here) latent matrix that is decoded into a $n \times d_{out}$ (i.e. 4×4 here) output matrix.

The proposed architecture can be summed up by the following function composition:

$$\hat{Y} = \mathbf{D} \circ (\mathbf{I} + \mathbf{L}^{(K-1)} \circ (\mathbf{A}_{or}, \mathbf{A}_{or})) \circ \dots \tag{17}$$

$$\dots \circ (\mathbf{I} + \mathbf{L}^{(0)} \circ (\mathbf{A}_{or}, \mathbf{A}_{or})) \circ \mathbf{A}_{inj} \circ \mathbf{E}(X) \tag{18}$$

C. Regarding the design of the architecture

Our neural network architecture consists in $K + 3$ learning blocks that are intricately laid out:

$$E_p : \mathbb{R}^{d_{in}} \to \mathbb{R}^d \tag{19}$$

$$E_c : \mathbb{R}^{d_{in}} \to \mathbb{R}^d \tag{20}$$

$$L^{(k)} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d, \quad k \in \{0, \dots, K-1\} \tag{21}$$

$$D : \mathbb{R}^d \to \mathbb{R}^{d_{out}} \tag{22}$$

One should also observe that there is no information exchange between power lines except through a matrix multiplication with the adjacency matrices. This point is key to the ability of our neural net to be compatible with various power grid shapes. There is no exchange between lines when the learning blocks are applied, only combinations between the d or d_{in} internal components of each of the n power line or n_{in} injection. The learning blocks internal dimensions are independent from the power Grid it is working on. Thus it can perform inference and learn on a power grid of any size and shape.

The computational complexity of inference for this architecture is in $\mathcal{O}(Knd\,[l_L d + 2n])$, where K is the number of propagation steps, n the number of power lines, d the dimension of the lines latent embeddings and l_L the depth of each block L^k. This complexity estimation does not exploit the sparsity of the adjacency matrices, and could thus be reduced.

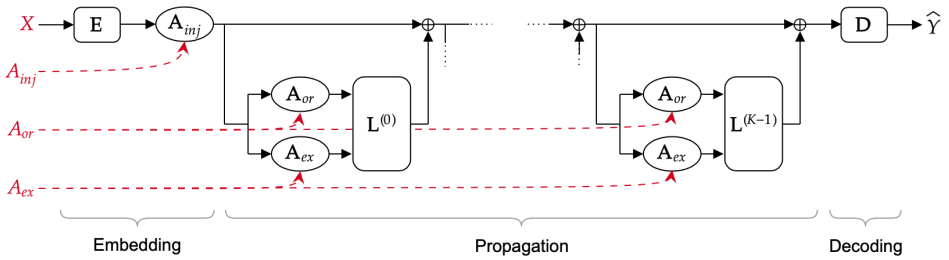

Fig. 5. **Architecture of the neural network** - This schematic presents the way the different operations are laid out. The input X is taken as a regular input to a neural network, while the adjacency matrices directly affect the architecture. (1) Round operations consist in right-side matrix multiplications, there is no exchange between the d dimensions during these steps. Those operations are not learned, and are based on the adjacency matrices that are inputed. (2) Rectangle operations consist in applying the same neural network for each of the n power lines or n_{in} injections, there is no exchange between them. Those are the neural networks that are actually learned during training.

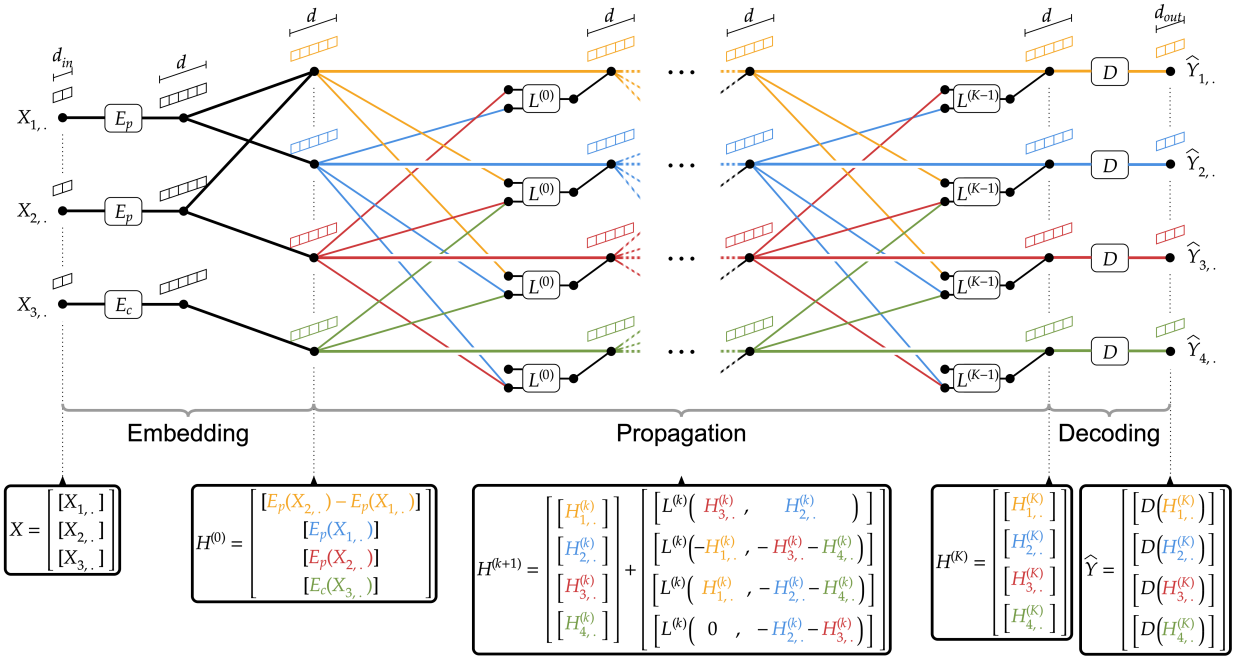

Fig. 6. **Full architecture of the neural network** - This schematic offers a more precise overview of the neural network architecture, and unveils the way the adjacency matrices actually impact the connections within the neural nets, based on the toy example from Figure 1. At the top of the Figure are displayed the dimensions of the latent embeddings throughout the architecture. At the bottom are presented the different formulas of some of the main steps of the architecture. For the sake of readability, we chose not to show the sign (± 1) of the links created by the adjacency matrices. However, these signs can be deduced from the bottom equations. The double brackets in these equations signify that we are dealing with 2D matrices.

III. Experiments

In this section we first compare our architecture to a fully connected neural net when the grid topology is the same during both training and testing. We then assess the generalization ability of our neural network to both larger and smaller power grids than those observed during training. Finally, we compare the computational requirements of our architecture to that of a regular physics solver.

In our experiments, we optimized the ℓ_2-loss (sum of square errors) with regards to the normalized flows through both the origin and the extremity of each line.

Current intensities that induce Joule's effect which can cause potential damage to lines and other equipment are flows in Amps. Hence, we adopted the MAPE90A metric for power system security analysis applications as introduced in [6]. It consists in a percentage of error on the 10% of largest flows in Amps in absolute value (per power line). This reflects the idea that when one tries to predict the flows through transmission lines, it is most important to be accurate on extreme values that can actually cause damage.

We used RTE's proprietary load flow solver to compute flows (given the power grid topology, and the set of injections) to obtain the ground truth of predictions. Our neural network's goal is to emulate the behavior of this solver.

As baseline, we compare every result to a standard reference method in power systems: the DC-approximation, which is a linearization of the physical equations. One of our goals is to beat the DC-approximation in terms of efficiency.

Each model was trained 20 times with random initialization of weights and mini batches. This allowed us to compute the

median, and the 20th and 80th percentiles for each of the observed metrics.

Experiment A : Constant Power Grid Topology

The first experiment we conducted is a sanity check, at fixed grid topology, similarly to what has been done previously in the literature. We show good performance of our new method but at the expense of additional computational expenses compared to a fully connected network. However, this is not the case for which our method was designed.

Specifically, in this experiment, the power grid topology (i.e. A_{or}, A_{ex} and A_{inj}) is the same in every datapoint in both Train and Test sets. Thus, only the injections vary and are randomly sampled. See [6] for more informations about the injection sampling. We focus on a fictitious power grids consisting in 30 nodes, 42 power lines, 20 consumptions and 6 productions (which are the same characteristics as the standard 30 nodes case [29]). We then compare the performance of our architecture with a Fully Connected neural network. A cross-validation has been performed on both architectures.

For both architectures, each of the 20 models has been trained for a number of 200,000 iterations (with mini batches of size 100). Training sets consist of 100,000 samples, and Test sets of 1,000 samples. We used fully connected neural networks for the learning blocks of our proposed architecture (i.e. E, $L^{(k)}$, D), with leaky ReLu activations. We also used leaky Relu in the FC baseline.

Figure 7 shows that our GNS architecture (our proposed method) learns faster in number of iterations. However, due to the larger number of parameters in our GNS, it takes ≈ 4 times longer for each iteration. This is due to the fact that our proposed architecture is a lot larger and more complex than a Fully Connected (FC) network. Although the figure presents our proposed architecture as faster, one may prefer a simple Fully Connected network for this task.

Table I presents the Test MAPE90A obtained for both models and the DC-approximation. Both baselines easily outperform the DC-approximation, and there is a slight advantage in favor our GNS architecture. For each architecture, the median is displayed, as well as the 20th and 80th percentiles of the 20 independently learned models.

The Fully Connected (FC) network is able to learn in this setup partly because the Power Grid topology is constant: it specializes in one specific instance of the load flow problem. In the next experiment, where we randomly vary the power grid topology in both Train and Test sets, it will be infeasible to learn to generalize to other grid architectures for a FC net.

TABLE I
EXPERIMENT A - IDENTICAL TOPOLOGY IN BOTH TRAIN AND TEST
MEDIAN OF THE METRICS ON TEST SET FOR 20 TRAINED MODELS (20TH AND 80TH PERCENTILES BETWEEN BRACKETS)

	FC	**proposed GNS**	DC-approx.
Loss	0.0494 [0.0483; 0.0507]	**0.0332** [0.0273; 0.0636]	0.68
MAPE90A (% of error)	0.534 [0.508; 0.578]	**0.472** [0.3880.501]	3.21

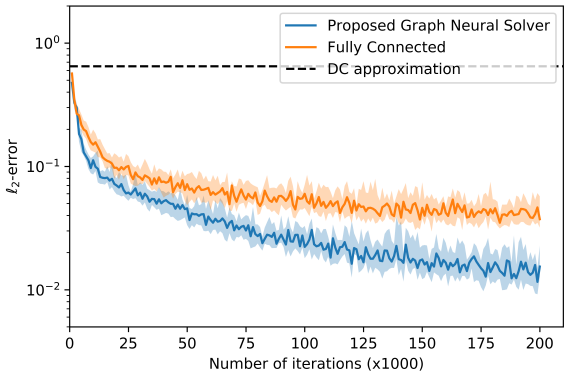

Fig. 7. **Experiment A : Learning curves** - Both the Fully Connected and our Graphical Neural Solver manage to outperform the DC approximation before the 1000th learning iteration. The Fully Connected seems to quickly reach an asymptote, while our proposed architecture still keeps on learning at the end of the 200,000 iterations. The plain line is the median of the 20 runs for each model, and the shaded areas is delimited by the 20th and 80th percentiles of the 20 runs.

Experiment B : Random Grid Topologies of Constant Sizes

In this experiment, both injections and power grid topologies are randomly sampled. We randomly sample the power grid topologies (using the sampling method described below) in the set of connected power grids that have 30 nodes, 42 power lines, 20 consumptions and 6 productions. Each point in both Train and Test set is a randomly sampled power grid, with a set of randomly sampled injections (same method). The distribution are the same in Train and Test sets.

Random power grid generation: Given m the number of electrical nodes, n the number of power lines, $|\mathcal{P}|$ the number of productions and $|\mathcal{C}|$ the number of consumptions, we use the following process to generate random power grids:

- Generate a random spanning tree of m nodes.
- Uniformly create random edge in the graph until there are n edges.
- Attribute the $|\mathcal{P}|$ productions and $|\mathcal{C}|$ consumptions to randomly picked nodes on the graph. Each electrical node can be neutral, production, consumption or both. It is however impossible for a node to have multiple injections of the same type.

We chose not to present the results of the Fully Connected network because it proved to be unable to learn on such a dataset. The experimental setup has been specifically designed so as to help our proposed architecture to generalize well.

Table II shows that our architecture is able to generalize to power grids that it has potentially never encountered in the training set. While the Fully Connected baseline from the previous experiment is completely unable to extract information from a dataset in which the Power Grid topology constantly changes, our proposed architecture still manages to beat the DC approximation on graphs that it has never encountered in the Train set. For our Graph Neural Solver, the median is

displayed, as well as the 20th and 80th percentiles of the 20 learned models.

TABLE II
EXPERIMENT B - RANDOM TOPOLOGIES OF CONSTANT SIZE
MEDIAN OF THE METRICS ON TEST SET FOR 20 TRAINED MODELS (20TH AND 80TH PERCENTILES BETWEEN BRACKETS)

	proposed GNS	DC-approx.
Loss	**0.0715** [0.0623; 0.0882]	0.0678
MAPE90A (% of error)	**0.729** [0.705; 0.873]	3.17

Experiment C : Random Grid Topologies of Various Sizes

This experiment has the exact same training protocol as the previous one, but the testing occurs on several datasets in which the power grid sizes can be different. Predicting in such a setting is impossible for a Fully Connected network, because of a matrix dimension mismatch, which prevents the size of the input to change. Figure 8 shows the different Test MAPE90A for datasets that consist in random power grids of sizes in $\{10, 15, \ldots, 105, 110\}$. In every case, the architecture has been trained only on random 30 nodes Power Grids. This means that while having observed solely Power Grids with 30 nodes during Training, our architecture is able to achieve a better MAPE90A than the DC-approximation, on Power Grids whose sizes range from 10 nodes to 110 nodes. This experimentally proves the ability of our neural net to generalize to both larger and smaller power grids. It seems to be able to learn how to solve the load flow problem in general, while not specializing in a specific instance of the problem.

A quite unexpected result that can be observed in this plot, is that the trained models are consistently more accurate on power grids that have 10 nodes than on power grids that have 30 nodes, even though the models were solely trained on power grids with 30 nodes.

While most error bars are quite consistent in Figure 8, there seems to be a larger error on power grids of 100 nodes (that still remains below the DC-approximation). We have not identified what causes it to be higher than expected, but we think that some networks sampled in this test set were not representative enough of actual power systems, thus causing our neural network to perform a poorer accuracy than on the 95 nodes and 105 nodes test sets.

Computational Time

Being able to decrease the computation time of complex problems such as load flows, computational fluid dynamics, etc. can be a major catalyser in being able to iterate over many different device design / situation, thus increasing our abilities in terms of installation design and security analysis.

In its current implementation and on power grids whose size range from 10 nodes to 110 nodes, our Graph Neural Solver is approximately twice as fast as the proprietary Load-Flow solver used at RTE (which is already thoroughly optimized). Our current implementation does not exploit the sparsity of the adjacency matrices. This limitation is due to the lack of an implementation of sparse matrices of rank strictly above 2 in TensorFlow. It strongly hinders the computational speed of our implementation, and should be an important axis of improvement in terms of speed.

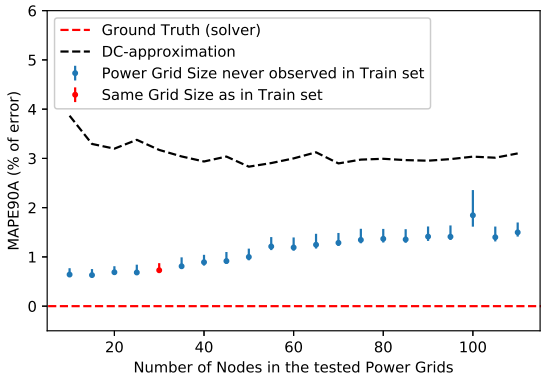

Fig. 8. **Experiment C : Generalization to both larger and smaller Power Grids** - This graph shows the MAPE90A metric on various Test sets. We test our trained models on sizes that range from 10 to 110 nodes. While having observed only Power Grids with 30 nodes during Training, our proposed Graph Neural Solver manages to consistently beat the DC-approximation for Power Grids whose sizes range from 10 to 110 nodes. 20 models were trained of the same architecture but with different random initializations. The dots are the median of the MAPE90A on each Test set, and the error bars are defined by the 20th and 80th percentiles of these 20 runs. The red error bar sums up the MAPE90A results on a Test set made of random Power Grid with 30 nodes (i.e. same size as during Training), while the blue error bars stand for the various Test sets made of Power Grid larger of smaller than 30 nodes.

IV. DISCUSSION & CONCLUSIONS

Our proposed architecture relies on iteratively updating the embeddings of power lines according to the values of the direct neighbors. It relies on ideas from the blooming field of Graph Neural Networks, and aims at emulating the behavior of Load Flow (LF) solvers. We experimentally showed its ability to generalize to both larger and smaller power grids than that used for training, opening the door for strongly generalizable Neural Solver for other Physics applications.

The main limitation of our current model is that we made the strong choice to work with power grids where every line has the same physical characteristics. We are conscious that incorporating the line characteristics in our neural network model would be critical to using it in real-world applications. We already have several insights on how this could be performed: one could for instance have the adjacency matrix coefficients be functions of the line characteristics.

Another aspect that would require some attention is the computational speed of our artificial neural network architecture. It currently treats the adjacency matrices as dense, while they are actually very sparse. We envision that an implementation of our architecture that would exploit this sparsity could provide much faster computations. Even without such optimization, our neural network is approximately twice as fast as the Load Flow solver that we want to emulate.

Our current implementation uses different Leap functions $L^{(0)}, \ldots, L^{(K-1)}$ at each propagation step. However, it would make sense to instead use the same propagation update at each step. We will be further investigating some alterations to the current formulation of the architecture, drawing some inspiration from the recent and inspiring Neural Ordinary Differential Equations [4]. Moreover, we will investigate the idea of training in an unsupervised our proposed architecture by directly minimizing the violation of the physical laws.

The sum operation in each propagation step ensures a consistency between the latent representations. The meaning of the d-dimensional information contained in each line, at each propagation update is unchanging. We could plug the learned decoder D in each power line and at each propagation step to visualize the evolution of the flows.

We could also investigate an adaptative number of propagation steps. The RNN domain already deals with this type of problems and should provide us with ideas on how to build Graph Neural Solver with an adaptable number of propagation iterations. Another aspect is that it is possible for a Load Flow computation to not converge, not because of numerical instability, but because the power grid is actually in danger of blackout. Our proposed architecture could be modified so as to include predictions on whether the computation will converge or not.

Currently, we only deal with stead-state power flows but we could try to predict the dynamic aspects of power grids, or try to tackle some other finite-element problems that can have temporal components. The ability to develop extremely fast AI-based proxies for Fluid Dynamics solvers could help researchers and engineers perform much faster investigation when it comes to designing novel buildings, aircrafts or wind turbines. Those applications usually require heavy and slow computations, which restrict the amount of designs that can be tested.

ACKNOWLEDGMENT

We would like to thank Vincent Barbesant, Rémy Clément, Laure Crochepierre, Guillaume Genthial and Wojciech Sitarz for their attentive review and remarks.

REFERENCES

[1] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, aglar Gülehre, F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[2] 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, July 2017.

[3] 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, April 2014*, 2014.

[4] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6572–6583. Curran Associates, Inc., 2018.

[5] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3844–3852. Curran Associates, Inc., 2016.

[6] B. Donnot and et al. Introducing machine learning for power system operation support. In *IREP Symposium*, Espinho, Portugal, Aug. 2017.

[7] B. Donnot, I. Guyon, A. Marot, M. Schoenauer, and P. Panciatici. Optimization of computational budget for power system risk assessment. working paper or preprint, May 2018.

[8] B. Donnot, I. Guyon, M. Schoenauer, A. Marot, and P. Panciatici. Anticipating contingengies in power grids using fast neural net screening. In *IEEE WCCI 2018*, Rio de Janeiro, Brazil, July 2018.

[9] B. Donnot, I. Guyon, M. Schoenauer, A. Marot, and P. Panciatici. Fast Power system security analysis with Guided Dropout. In *European Symposium on Artificial Neural Networks*, Bruges, Belgium, Apr. 2018.

[10] L. Duchesne, E. Karangelos, and L. Wehenkel. Using machine learning to enable probabilistic reliability assessment in operation planning, 06 2018.

[11] D. K. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *CoRR*, abs/1509.09292, 2015.

[12] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.

[13] J. D. D. Glover and M. S. Sarma. *Power System Analysis and Design*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA, 3rd edition, 2001.

[14] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[15] R. Herzig, M. Raboh, G. Chechik, J. Berant, and A. Globerson. Mapping images to scene graphs with permutation-invariant structured prediction. 02 2018.

[16] T. N. Kipf, E. Fetaya, K. Wang, M. Welling, and R. S. Zemel. Neural relational inference for interacting systems. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 2693–2702, 2018.

[17] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[18] P. Kundur, N. J. Balu, and M. G. Lauby. *Power system stability and control*, volume 7. McGraw-hill New York, 1994.

[19] P. S. Kundur. Power system stability. In *Power System Stability and Control, Third Edition*, pages 1–12. CRC Press, 2012.

[20] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel. Gated graph sequence neural networks. *CoRR*, abs/1511.05493, 2015.

[21] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia. Learning deep generative models of graphs. *CoRR*, abs/1803.03324, 2018.

[22] J. Ling, A. Kurzawski, and J. Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 11 2016.

[23] K. Mills, M. Spanner, and I. Tamblyn. Deep learning and the schrödinger equation. *Physical Review A*, 96, 02 2017.

[24] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

[25] A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. P. Lillicrap. A simple neural network module for relational reasoning. *CoRR*, abs/1706.01427, 2017.

[26] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *Trans. Neur. Netw.*, 20(1):61–80, Jan. 2009.

[27] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating eulerian fluid simulation with convolutional networks. *CoRR*, abs/1607.03597, 2016.

[28] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola. Deep sets. *CoRR*, abs/1703.06114, 2017.

[29] 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*, 26(1):12–19, Feb 2011.