# OPTIMAL POWER FLOW USING GRAPH NEURAL NETWORKS

*Damian Owerko, Fernando Gama, and Alejandro Ribeiro*

Department of Electrical and Systems Engineering, University of Pennsylvania

## ABSTRACT

Optimal power flow (OPF) is one of the most important optimization problems in the energy industry. In its simplest form, OPF attempts to find the optimal power that the generators within the grid have to produce to satisfy a given demand. Optimality is measured with respect to the cost that each generator incurs in producing this power. The OPF problem is non-convex due to the sinusoidal nature of electrical generation and thus is difficult to solve. Using small angle approximations leads to a convex problem known as DC OPF, but this approximation is no longer valid when power grids are heavily loaded. Many approximate solutions have been since put forward, but these do not scale to large power networks. In this paper, we propose using graph neural networks (which are localized, scalable parametrizations of network data) trained under the imitation learning framework to approximate a given optimal solution. While the optimal solution is costly, it is only required to be computed for network states in the training set. During test time, the GNN adequately learns how to compute the OPF solution. Numerical experiments are run on the IEEE-30 and IEEE-118 test cases.

*Index Terms*— graph neural networks, smart grids, optimal power flow, imitation learning

## 1. INTRODUCTION

Optimal power flow (OPF) is one of the most important optimization problems for the energy industry [1]. It is used for system planning, establishing prices on day-ahead markets, and to allocate generation capacity efficiently throughout the day. Even though the problem was formulated over half a century ago, we still do not have a fast and robust technique for solving it, which would save tens of billions of dollars annually [1].

In its simplest form, OPF attempts to find the optimal power that the generators within the grid have to produce to satisfy a given demand. Optimality is measured with respect to the cost that each generator incurs in producing the required power. Even though this problem is at the heart of daily electricity grid operation, the sinusoidal nature of electrical generation makes it difficult to solve [1–3]. One alternative is to linearly approximate the problem by assuming constant bus voltage and taking small angle approximations for trigonometric terms of voltage angle difference. This approach has limitations. The small angle approximations fail for heavily-loaded networks, since differences in voltage angles become large [4]. Nevertheless, this approximation is commonly used in industry [1]. The exact formulation of the problem is commonly referred to as ACOPF and the approximation as DCOPF.

Incorporating AC dynamics results in a non-convex problem due to nonlinearities in the power flow equations [2, 5] and has been

proven to be NP hard [1, 3]. One approach is to apply convex relaxation techniques resulting in semi-definite programs [5]. There have been many attempts to apply machine intelligence to the problem. The work in [6] offers a comprehensive review of such approaches. These include include applying evolutionary programming, evolutionary strategies, genetic algorithms, artificial neural networks, simulated annealing, fuzzy set theory, ant colony optimization and particle swarm optimization. However, none of these approaches has been shown to work on networks larger than the 30 node IEEE test case [7–9].

Recently, motivated by the possibility of accurately generating large amounts of data, machine learning methods have been considered as solutions to this problem. More specifically, [10] propose a system that uses multiple stepwise regression [11] to imitate the output of ACOPF. Each node gathers information from a subset of nodes, although not necessarily neighboring nodes. They then use multiple stepwise regression to predict the optimal amount of power generated at each node, imitating the ACOPF solution. However, this solution is not local, since it uses information from nodes that are not adjacent in the network. The work in [12] uses a fully connected network (MLP) to imitate the output of ACOPF. Yet, MLPs are not local either, tend to overfit and have trouble scaling up. Instead, exploiting the structure of the problem is necessary for a scalable solution. An exact, albeit costly solution, can be obtained by using interior point methods to solve the ACOPF problem, which, in practice, converges to the optimal solution – though not always and without guarantee of optimality [1, 3].

In this work, we use *imitation learning* and *graph neural networks* [13–15] to find a local and scalable solution to the OPF problem. More specifically, we adopt a parametrized GNN model, which is local and scalable by design, and we train it to *imitate* the optimal solution obtained using an interior point solver [16], which is centralized and does not converge for large networks. Once trained, the GNN offers an efficient computation of ACOPF. The paper is structured as follows. In Sec. 2 we describe the OPF problem and in Sec. 3 we introduce GNNs. In Sec. 4 we test the imitation learning framework on the IEEE-30 and IEEE-118 power system test cases [17] and in Sec. 5 we draw conclusions.

## 2. OPTIMAL POWER FLOW

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ be a graph with a set of $N$ nodes $\mathcal{V}$, a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ and an edge weight function $\mathcal{W} : \mathcal{E} \to \mathbb{R}_+$. We use this graph to model the power grid, where the nodes are the buses and the edge weights $\mathcal{W}(i,j) = w_{ij}$ model the lines, which depend on the impedance $z_{ij}$ (in ohms) between bus $i$ and bus $j$. More specifically, we use the gaussian kernel $w_{ij} = \exp(-k|z_{ij}|^2)$ where $k$ is a scaling factor. We ignore links whose weight is less than a threshold $\omega$, so that $\mathcal{E} = \{(i,j) \in \mathcal{V} \times \mathcal{V} : w_{ij} > \omega\}$. Additionally, denote by $\mathbf{W} \in \mathbb{R}^{N \times N}$ the adjacency matrix of the network, $[\mathbf{W}]_{ij} = w_{ij}$ if $(i,j) \in \mathcal{E}$ and 0 otherwise. Since the graph is undirected, the matrix $\mathbf{W}$ is symmetric.

To describe the state of the power grid, we assign to each node $n$ a vector $\mathbf{x}_n = [v_n, \delta_n, p_n, q_n] \in \mathbb{R}^4$ where $v_n$ and $\delta_n$ are the voltage magnitude and angle, respectively, and where $p_n$ and $q_n$ are the active and reactive powers, respectively. We note that the *state* of each node can be accurately measured locally. We can collect these measurements across all nodes and denote them as $\mathbf{v} \in \mathbb{R}^N$, $\boldsymbol{\delta} \in \mathbb{R}^N$, $\mathbf{p} \in \mathbb{R}^N$ and $\mathbf{q} \in \mathbb{R}^N$. Thus, the collection of the states at all nodes becomes a matrix

$$\mathbf{X} = \begin{bmatrix} \mathbf{v} & \boldsymbol{\delta} & \mathbf{p} & \mathbf{q} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^\mathsf{T} \\ \vdots \\ \mathbf{x}_N^\mathsf{T} \end{bmatrix} \in \mathbb{R}^{N \times 4}. \tag{1}$$

Of particular importance are the nodes that are generators, since these are the ones that we are interested in controlling. We proceed to denote by $\mathcal{V}_G = \{v_1, \ldots, v_M\} \subset \mathcal{V}$ the set of $M < N$ generators $v_m \in \{1, \ldots, N\}$ and define a selection matrix $\mathbf{G} \in \{0,1\}^{M \times N}$ that picks out the $M$ generator nodes out of all the $N$ node sin the power grid, i.e. $[\mathbf{G}]_{ij} = 1$ if $v_i = j$ and 0 otherwise. We note that $\mathbf{G}\mathbf{G}^\mathsf{T} = \mathbf{I}_M$ is the $M \times M$ identity matrix and that $\mathbf{G}^\mathsf{T}\mathbf{G} = \mathrm{diag}(\mathbf{g})$ with $\mathbf{g} \in \{0,1\}^N$ is the selection vector, i.e. $[\mathbf{g}]_n = 1$ if $n \in \mathcal{V}_G$ and 0 otherwise. Finally, we note that $\mathbf{G}\mathbf{X} \in \mathbb{R}^{M \times 4}$ is the collection of states only at the generator nodes.

The problem of optimal power flow (OPF) consists in determining the power that each generator needs to input into the power grid to satisfy the demand, while minimizing the cost of producing that power at each generator. Let $c_m(p_m)$ be a cost function for generator $m \in \mathcal{V}_G$ that depends only on the generating active power $p_m$. The active power $p_n$ at node $n$ is determined by the voltage magnitudes $\mathbf{v}$ and angles $\boldsymbol{\delta}$ at all other nodes and by the topological structure of the power grid $\mathbf{W}$. This relationship is given by a nonlinear function $\mathcal{P}$ such that $\mathbf{p} = \mathcal{P}(\mathbf{v}, \boldsymbol{\delta}; \mathbf{W})$ and is derived from Kirchhoff's current law. Likewise, for the reactive power we have $\mathbf{q} = \mathcal{Q}(\mathbf{v}, \boldsymbol{\delta}; \mathbf{W})$ for another function $\mathcal{Q}_n$. Additionally, there are operational constraints on the power grid, limiting the possible values of active and reactive power, and of voltage magnitude and angles. Let $\mathbf{X}^{\min}$ and $\mathbf{X}^{\max}$ be the matrices collecting the minimum and maximum value each state entry can take, and denote by $\mathbf{X} \preceq \mathbf{Y}$ the elementwise inequality $[\mathbf{X}]_{ij} \leq [\mathbf{Y}]_{ij}$ for all $i, j$. With all this notation in place, we can write the OPF problem as follows

$$\underset{\{p_m\}}{\text{minimize}} \sum_{m \in \mathcal{V}_g} c_m(p_m) \tag{2}$$

$$\text{subject to } \mathbf{p} = \mathcal{P}(\mathbf{v}, \boldsymbol{\delta}; \mathbf{W}), \tag{3}$$

$$\mathbf{q} = \mathcal{Q}(\mathbf{v}, \boldsymbol{\delta}; \mathbf{W}), \tag{4}$$

$$\mathbf{X}^{\min} \preceq \mathbf{X} \preceq \mathbf{X}^{\max}. \tag{5}$$

Solving the OPF problem for DC operation (bias point) is straightforward since the problem becomes convex, as long as the cost function is convex. The cost function is most commonly a second order polynomial of $p_m$. In the DCOPF approximation, functions $\mathcal{P}$ and $\mathcal{Q}$ become linear and the problem can be readily solved. However, when we consider ACOPF dynamics, the problem becomes non-convex due to the computation of the active and reactive power (3)-(4), rendering it computationally costly. In practice, this problem is usually solved using one of five optimization problem solvers: CONOPT, IPOPT, KNITRO, MINOS and SNOPT. However, in general, they are slow to converge for large networks [18]. In particular, we consider IPOPT [19] (Interior Point OPTimizer) to provide an optimal solution of the problem (albeit costly) since it was shown to be one of the more robust [18].

The objective of this paper is to *learn* the optimal solution in a decentralized and scalable manner by exploiting a framework known as *imitation learning*. Denote by $\mathbf{p}^*$ the optimal solution obtained by IPOPT, by $\boldsymbol{\Phi}(\mathbf{X})$ a (likely, nonlinear) map of the data and by $\mathcal{L}$ some given loss function. Then, in *imitation learning* we want to solve

$$\min_{\boldsymbol{\Phi}} \mathbb{E}\left[\mathcal{L}\left(\mathbf{p}^*, \boldsymbol{\Phi}(\mathbf{X})\right)\right] \tag{6}$$

where the expectation is taken over the (unknown) joint distribution of optimal power $\mathbf{p}^*$ and states $\mathbf{X}$. Solving (6) in its generality is typically intractable [20]. Therefore, we choose a parametrization of the map $\boldsymbol{\Phi}(\mathbf{X}) = \boldsymbol{\Phi}(\mathbf{X}; \mathcal{H})$ where $\boldsymbol{\Phi}$ now becomes a known family of functions (a chosen model) that is parameterized by $\mathcal{H}$. Likewise, to address the issue of the unknown joint probability distribution between $\mathbf{p}^*$ and $\mathbf{X}$ we assume the existence of a dataset $\mathcal{T} = \{(\mathbf{X}, \mathbf{p}^*)\}$ that can be used to approximate the expectation operator [20]. With this in place, the problem of imitation learning now boils down to choosing the best set of parameters $\mathcal{H}$ that solves

$$\min_{\mathcal{H}} \sum_{(\mathbf{X}, \mathbf{p}^*)} \mathcal{L}\left(\mathbf{p}^*, \boldsymbol{\Phi}(\mathbf{X}; \mathcal{H})\right). \tag{7}$$

The desirable properties of locality and scalability can be achieved by a careful choice of model $\boldsymbol{\Phi}(\mathbf{X}; \mathcal{H})$. In particular, we focus on graph neural networks (GNNs) [13–15] to exploit their stability properties [21] that guarantee scalability [22, 23].

## 3. GRAPH NEURAL NETWORKS

In the context of graph signal processing (GSP) the state $\mathbf{X} \in \mathbb{R}^{N \times F}$ of the nodes can be seen as a *graph signal* [24–26]. Essentially, row $n$ corresponds to the $F = 4$ measurements or *features* obtained at node $n$. We can relate the graph signal $\mathbf{X}$ to the underlying network support by multiplying by $\mathbf{W}$ to the left

$$\mathbf{Y} = \mathbf{W}\mathbf{X}. \tag{8}$$

The output $\mathbf{Y} \in \mathbb{R}^{N \times F}$ is another graph signal that is a *shifted* version of the input $\mathbf{X}$. To see this, note that now the $f$th feature at node $i$, $[\mathbf{Y}]_{if} = y_i^f$, is a linear combination of the $f$th feature values $[\mathbf{X}]_{jf} = x_j^f$ at nodes $j$ that are in the the one-hop neighborhood of $i$, $\mathcal{N}_i = \{j \in \mathcal{V} : (j, i) \in \mathcal{E}\}$. More specifically,

$$y_i^f = \sum_{j=1}^n [\mathbf{W}]_{ij}[\mathbf{X}]_{jf} = \sum_{j \in \mathcal{N}_i} w_{ij} x_j^f. \tag{9}$$

where the second equation follows due to the sparsity of the adjacency matrix $\mathbf{W}$. Based on (9), operation (8) can be interpreted as a diffusion or a *shift* of the signal across the graph, where each state gets updated by a linear combination of the neighboring states. While in this paper we focus on the use of the adjacency matrix $\mathbf{W}$, any other matrix that respects the sparsity of the graph (i.e. $[\mathbf{W}]_{ij} = 0$ if $(i, j) \notin \mathcal{E}$) works. Matrices that satisfy this constraint are called *graph shift operators* (GSOs) and other examples of widespread use in the GSP literature include the Laplacian matrix, the random walk Laplacian and several normalized versions of these [26].

The concept of *shift* (8) is used as the basic building block for the operation of *graph convolution*. In analogy with its discrete-time counterpart, a *graph convolution* is defined as a linear combination of shifted version of the input signal

$$\mathbf{Y} = \sum_{k=0}^{K-1} \mathbf{W}^k \mathbf{X} \mathbf{H}_k \tag{10}$$
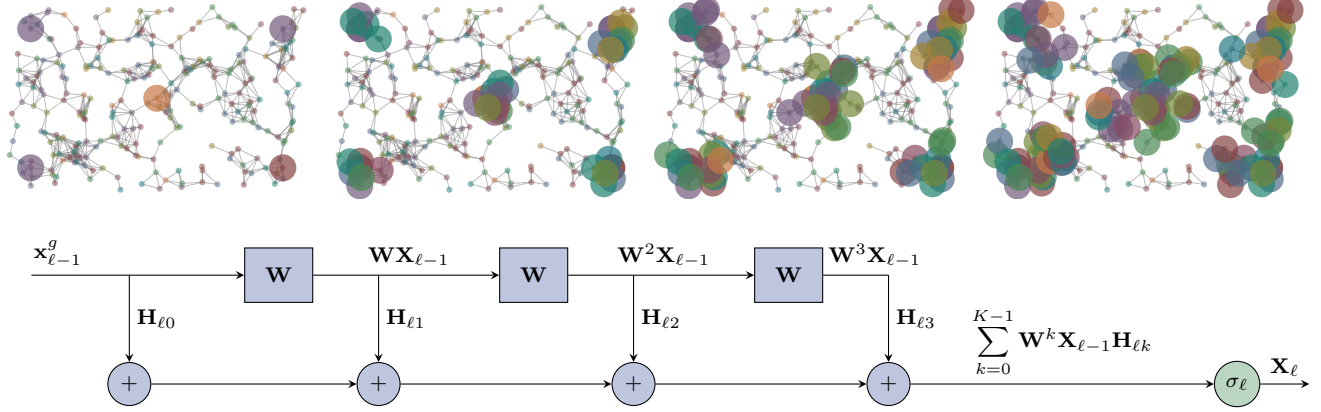
**Fig. 1**. Graph neural networks. Every node takes its data value $\mathbf{X}_{\ell-1}$ and weighs it by $\mathbf{H}_{\ell 0}$ (first graph). Then, all the nodes exchange information with their one-hop neighbors to build $\mathbf{W}\mathbf{X}_{\ell-1}$, and weigh the result by $\mathbf{H}_{\ell 1}$ (second graph). Next, they exchange their values of $\mathbf{W}\mathbf{X}_{\ell-1}$ again to build $\mathbf{W}^2\mathbf{X}_{\ell-1}$ and weigh it by $\mathbf{H}_{\ell 2}$ (third graph). This procedure continues for $K$ steps until all $\mathbf{S}^k\mathbf{X}_{\ell-1}\mathbf{H}_{\ell k}$ have been computed for $k = 0, \ldots, K-1$, and added up to obtain the output of the graph convolution operation (10). Then, the nonlinearity $\sigma_\ell$ applied to compute $\mathbf{X}_\ell$. To avoid cluttering, this operation is illustrated on only 5 nodes. In each case, the corresponding neighbors accessed by successive relays of information are indicated by the colored disks.

with $\mathbf{H}_k \in \mathbb{R}^{F \times G}$ the matrix of coefficients for $k = 0, \ldots, K-1$. The output $\mathbf{Y} \in \mathbb{R}^{N \times G}$ is another graph signal with $G$ features on each node. As we analyzed in (9), the operation $\mathbf{W}\mathbf{X}$ computes a linear combination of neighboring values. Likewise, repeated application of $\mathbf{W}$ computes a linear combination of values located farther away, i.e. $\mathbf{W}^k\mathbf{X}$ collects the feature values at nodes in the $k$-hop neighborhood. The value of $\mathbf{W}^k\mathbf{X} = \mathbf{W}(\mathbf{W}^{k-1}\mathbf{X})$ can be computed locally by $k$ repeated exchanges with the one-hop neighborhood. We note that multiplication by $\mathbf{H}_k$ on the right does not affect the locality of the graph convolution (10), since $\mathbf{H}_k$ acts by mixing the features local to each node. That is, it takes the $F$ input features, and mixes them linearly to obtain $G$ new features. To draw further analogies with filtering, we observe that the output $\mathbf{Y}$ of the graph convolution (10) is the result of applying a bank of $FG$ linear shift-invariant graph filters [27].

A graph neural network (GNN) is a nonlinear map $\boldsymbol{\Phi}(\mathbf{X}; \mathcal{H}, \mathbf{W})$ that is applied to the input $\mathbf{X}$ and takes into account the underlying graph $\mathbf{W}$. It consists of a cascade of $L$ layers, each of them applying a graph convolution (10) followed by a pointwise nonlinearity $\sigma_\ell$ (see Fig. 1 for an illustration)

$$\mathbf{X}_\ell = \sigma_\ell \left[ \sum_{k=0}^{K_\ell - 1} \mathbf{W}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k} \right] \quad (11)$$

for $\ell = 1, \ldots, L$, where $\mathbf{X}_0 = \mathbf{X}$ the input signal and $\mathbf{H}_{\ell k} \in \mathbb{R}^{F_{\ell-1} \times F_\ell}$ are the coefficients of the graph convolution (10) at layer $\ell$ [13–15]. The state $\mathbf{X}_\ell \in \mathbb{R}^{N \times F_\ell}$ at each layer $\ell$ is a graph signal with $F_\ell$ features and we consider the state at the last layer $\mathbf{X}_L$ to be the output of the GNN $\boldsymbol{\Phi}(\mathbf{X}; \mathcal{H}, \mathbf{W}) = \mathbf{X}_L$.

The computation of the state $\mathbf{X}_\ell$ in each of the $\ell$ layers can be carried out entirely in a local fashion, by means of repeated exchanges with one-hop neighbors. Also, the number of filter taps in $\mathbf{H}_{\ell k}$ is $F_{\ell-1} F_\ell$, independent of the size of the network, and thus, the GNN (11) is a scalable architecture [15]. This justifies the consideration of the GNN (11) as the model of choice in (7), $\boldsymbol{\Phi}(\mathbf{X}; \mathcal{H}) = \boldsymbol{\Phi}(\mathbf{X}; \mathcal{H}, \mathbf{W})$ with parameters $\mathcal{H} = \{\mathbf{H}_{\ell k}, k = 0, \ldots, K_{\ell-1}, \ell = 1, \ldots, L\}$ totaling $|\mathcal{H}| = \sum_{\ell=1}^{L} F_{\ell-1} F_\ell K_\ell$, independent of the size $N$ of the network. Furthermore, GNNs exhibit the properties of permutation equivariance and stability to graph perturbations [21]. The

first one, allows the GNN to learn from fewer datapoints by exploiting the topological symmetries of the network, while the second one allows the network to have a good performance when used on different networks than the one it was trained in, as long as these networks are similar.

In summary, in this paper we propose using the GNN (11) as a local and scalable model $\boldsymbol{\Phi}(\mathbf{X}; \mathcal{H}, \mathbf{W})$ that we use to *imitate* the optimal solution $\mathbf{p}^*$. We find the best model parameters $\mathcal{H}$ by solving (7) over a given dataset $\mathcal{T}$.

## 4. NUMERICAL EXPERIMENTS

We construct two datasets based on the IEEE-30 and IEEE-118 power system test cases [17]. Each dataset sample consists of a given load, where $\mathbf{p}^L \in \mathbb{R}^N, \mathbf{q}^L \in \mathbb{R}^N$ are the load components at each node, a given sub-optimal actual state of the power grid $\mathbf{X}$, and the optimal power generated at each node, $\mathbf{p}^*$ (computed by means of IPOPT). The total power at each node is the difference between the generated power, $\mathbf{p}^G \in \mathbb{R}^N$ and $\mathbf{q}^G \in \mathbb{R}^N$ and the load at that node, $\mathbf{p}^L, \mathbf{q}^L$. Certainly, only generators are capable of generating power, while all other buses just consume their power load. These are related to the OPF equations (3) and (4) as follows

$$p_m = [\mathbf{p}^G]_m - [\mathbf{p}^L]_m, \qquad m \in \mathcal{V}_G \quad (12)$$

$$q_m = [\mathbf{q}^G]_m - [\mathbf{q}^L]_m, \qquad m \in \mathcal{V}_G \quad (13)$$

$$p_n = -[\mathbf{p}^L]_n, \qquad n \in \mathcal{V} \backslash \mathcal{V}_G \quad (14)$$

$$q_n = -[\mathbf{q}^L]_n. \qquad n \in \mathcal{V} \backslash \mathcal{V}_G \quad (15)$$

The test cases provide reference load, $\mathbf{p}^L_{\text{ref}}$ and $\mathbf{q}^L_{\text{ref}}$ [17]. Each given load is obtained synthetically as a sample from the uniform distribution around the reference, following the same methodology used in [12]

$$\mathbf{p}^L \sim \text{Uniform}(0.9\,\mathbf{p}^L_{\text{ref}}, 1.1\,\mathbf{p}^L_{\text{ref}}) \quad (16)$$

$$\mathbf{q}^L \sim \text{Uniform}(0.9\,\mathbf{q}^L_{\text{ref}}, 1.1\,\mathbf{q}^L_{\text{ref}}) \quad (17)$$
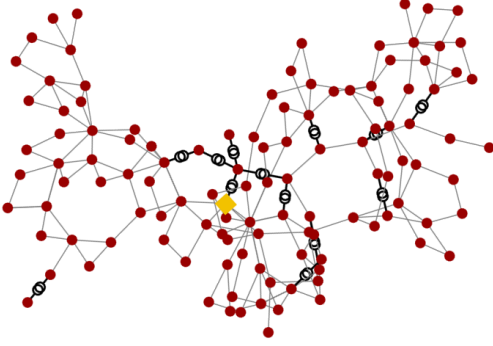
**Fig. 2**. Diagram of IEEE-118 power system test case. Red circles are buses, the yellow square is an external grid connection and intersecting circles are transformers.

The sub-optimal state $\mathbf{X}$ is obtained by using Pandapower [16] to find the steady state of the system, also known as a power flow computation. We can interpret this steady state as the state of the power grid before we need to adjust it to accommodate for the new values of $\mathbf{p}^L$ and $\mathbf{q}^L$. Meanwhile, the steady-state generator power is set by solving a DCOPF to find a sub-optimal generator configuration. DCOPF has a low computational cost and is therefore often used to warm start ACOPF. The generated load is also used as an input to ACOPF, which we solve using IPOPT [19] in Pandapower to obtain the optimal generator output $\mathbf{p}^*$. We discard samples for which IPOPT fails to converge. Therefore, for the IEEE-30 and IEEE-118 test cases, we generate $8,016$ and $13,129$ samples, respectively. These are split 80% for training and 20% for testing. We use this dataset to compare four architectures: global GNN, local GNN, global MLP and local MLP, which are described next.

The global GNN architecture consists of two graph convolutional layers [cf. (11)] and one readout layer. The first convolutional layer has $K = 4$ filter taps and outputs $F_1 = 128$ features. The second layer also has $K = 4$ taps and outputs $F_2 = 64$ features. These are followed by a last readout layer consisting of a fully connected layer with input size $64N$ and output size $M$. We note that the inclusion of the fully connected layer makes the global GNN architecture nonlocal, since it arbitrarily mixes the features at all nodes. Also, the number of parameters of the entire architecture depends on the size of the graph. Therefore, we also test a local GNN architecture, where we keep the first two convolutional layers, but we change the readout layer. More specifically, this new readout layer computes a linear combination of the features available locally at each node to output a single scalar to represent the estimated power. This is equivalent to setting $K = 1$ in the last layer of (11), $\mathbf{X}_L = \sigma_L [\mathbf{X}_{L-1}\mathbf{H}_L]$, where $\mathbf{H}_L \in \mathbb{R}^{F_{L-1} \times 1}$ represents the linear combination of input features with $\mathbf{X}_L \in \mathbb{R}^N$. Finally, we select only the outputs at the generator nodes $\mathbf{G}\mathbf{X}_L \in \mathbb{R}^M$. As we can see, in the local GNN, all operations involved can be carried out by repeated exchanges with one-hop neighbors and do not involve centralized knowledge of the state of the entire power grid.

As a baseline, we compare the GNN to the global MLP used in [12]. The MLP has two hidden layers with 128 and 64 features, respectively, thus having $128N$ and $64N$ hidden units. We keep the readout layer to be a fully connected layer mapping $64N$ into the $M$ estimated power at the generators. Since this solution is also centralized, we use another baseline consisting of a local MLP that only performs operations on the state $\mathbf{X}$ of each node. Specifically, we set $K = 1$ in all layers of the GNN [cf. (11)], so that it becomes

**Table 1**: The RMSE for each architecture and dataset.

| Architecture | IEEE 30 | IEEE 118 |
|---|---|---|
| Global GNN | 0.061 | 0.00306 |
| Global MLP | 0.090 | 0.00958 |
| Local GNN | 0.139 | 0.03038 |
| Local MLP | 0.161 | 0.35932 |

becomes $\mathbf{X}_\ell = \sigma_\ell [\mathbf{X}_{\ell-1}\mathbf{H}_\ell]$, $\mathbf{H}_\ell \in \mathbb{R}^{F_{\ell-1} \times F_\ell}$. As with the local GNN, we consider $\mathbf{G}\mathbf{X}_L$ to be the output. We note that this last architecture, albeit local, it completely ignores the underlying graph structure of the data, since it does not involve any kind of neighbor exchanges.

We train the aforementioned architectures over the IEEE-30 and IEEE-118 training sets by solving (7) for an MSE loss function. The optimal $\mathbf{p}^*$ is given by the solution of IPOPT, and the parametrization $\mathbf{\Phi}$ under consideration is each one of the four architectures, i.e. $\mathbf{\Phi}(\mathbf{X}; \mathcal{H}, \mathbf{W})$ for the global and local GNNs, and $\mathbf{\Phi}(\mathbf{X}; \mathcal{H})$ for the global and local MLPs (which, as becomes explicit now, do not take into account the underlying graph structure). We use an ADAM optimizer with learning rate 0.001 and forgetting factors $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We train for a 100 epochs with a batch size of 128. We evaluate the trained architectures on the corresponding test set by using a performance measure given by the relative root mean squared error $\sqrt{\|\mathbf{p}^* - \mathbf{\Phi}(\mathbf{X}; \mathcal{H})\|}/\|\mathbf{p}^*\|$ averaged over all values of $(\mathbf{X}, \mathbf{p}^*)$ in the test set. Results are shown in Table 1.

We note that in both test cases, the GNNs outperform the MLPs. In the smaller network IEEE-30, the relative improvement of the Global GNN over the Global MLP is of 47%, while for the Local GNN with respect to the local MLP is of 15%. Likewise, for the larger network IEEE-118, the Global GNN relatively improves 213% over the Global MLP, while the Local GNN exhibits a relative improvement of 1082% over the Local MLP. This shows that local solutions that exploit the underlying network structure outperform other similar neural network parametrizations that ignore it.

Finally, we note that obtaining the IPOPT solution $\mathbf{p}^*$ based on a given state $\mathbf{X}$ takes 2.16s on the IEEE-30 network and 18s on the IEEE-118 one, representing a 8-fold increase in the computational time. One the other hand, the Global GNN and the Local GNN take $48\mu$s and $49\mu$s, respectively, on the IEEE-30 dataset, and $50\mu$s and $56\mu$s, respectively, on the IEEE-118 dataset. This amounts to 1.04 and 1.14 times more computational time, respectively. This shows that, not only the use of GNNs is over $10^5$ faster, but that they also exhibit much better scalability to larger networks.

## 5. CONCLUSIONS

Solving the OPF problem is central to power grid operation. OPF is a non-convex problem whose exact solution can be computed by means of IPOPT (interior point optimizer). However, this solution is costly and does not scale to large networks. In this paper, we proposed the use of GNNs to approximate the OPF solution. GNNs are scalable and local information processing architectures that exploit the network structure of the data. We train GNNs by taking a given network state as input, and using the corresponding output to approximate the optimal IPOPT solution, following the imitation learning framework. We run experiments on the IEEE-30 and IEEE-118 test cases, showing that local solutions that adequately exploit the underlying grid structure outperform other comparable methods.

# 6. REFERENCES

[1] M. B. Cain, R. P. O'Neill, and A. Castillo, "History of optimal power flow and formulations," *Federal Energy Regulatory Commission, Increasing Efficiency through Improved Software*, pp. 1–31, Dec. 2012.

[2] B. C. Lesieutre and I. A. Hiskens, "Convexity of the set of feasible injections and revenue adequacy in FTR markets," *IEEE Trans. Power Syst.*, vol. 20, no. 4, pp. 1790–1798, Nov. 2005.

[3] D. Bienstock and A. Verma, "Strong NP-hardness of AC power flows feasibility," *Operations Research Letters*, vol. 47, no. 6, pp. 494–501, Nov. 2019.

[4] S. Chatzivasileiadis, "Lecture notes on optimal power flow OPF," *arXiv:1811.00943v1 [cs.SY]*, 2 Nov. 2018.

[5] D. K. Molzahn and I. A. Hiskens, "Convex relaxations of optimal power flow problems: An illustrative example," *IEEE Trans. Circuits Syst. I*, vol. 63, no. 5, pp. 650–660, May 2016.

[6] M. R. AlRashidi and M. E. El-Hawary, "Applications of computational intelligence techniques for solving the revived optimal power flow problem," *Electric Power Syst. Research*, vol. 79, no. 4, pp. 694–702, Apr. 2009.

[7] A. G. Bakirtzis, P. N. Biskas, C. E. Zoumas, and V. Petridis, "Optimal power flow by enhanced genetic algorithm," *IEEE Trans. Power Syst.*, vol. 17, no. 2, pp. 229–236, May 2002.

[8] M. Todorovksi and D. Rajicic, "A power flow method suitable for solving OPF problems using genetic algorithms," in *The IEEE Region 8 EUROCON 2003*, Ljubljana, Slovenia, 22-24 Sep. 2003, pp. 215–219, IEEE.

[9] C.-R. Wang, H.-J. Yan, Z.-Q. Huang, J.-W. Zhang, and C.-J. Sun, "A modified particle swarm optimization algorithm and its application in optimal power flow problem," in *4th Int. Conf. Machine Learning Cybernetics*, Guangzhou, China, 7 Nov. 2005, pp. 2885–2889, IEEE.

[10] R. Dobbe, O. Sondermeijer, D. Fridovich-Keil, D. Arnold, D. Callaway, and C. Tomlin, "Towards distributed energy services: Decentralizing optimal power flow with machine learning," *arXiv:1806.06790v3 [cs.LS]*, 13 Aug. 2019.

[11] O. Sondermeijer, R. Dobbe, D. Arnold, C. Tomlin, and T. Keviczky, "Regression-based inverter control for decentralized optimal power flow and voltage regulation," *arXiv:1902.08594v1 [cs.SY]*, 20 Feb. 2019.

[12] N. Guha, Z. Wang, and A. Majumdar, "Machine learning for AC optimal power flow," in *36th Int. Conf. Mach. Learning*, Long Beach, CA, 9-15 June 2019.

[13] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *2nd Int. Conf. Learning Representations*, Banff, AB, 14-16 Apr. 2014, pp. 1–14, Assoc. Comput. Linguistics.

[14] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *30th Conf. Neural Inform. Process. Syst.*, Barcelona, Spain, 5-10 Dec. 2016, pp. 3844–3858, Neural Inform. Process. Foundation.

[15] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional neural network architectures for signals supported on graphs," *IEEE Trans. Signal Process.*, vol. 67, no. 4, pp. 1034–1049, Feb. 2019.

[16] L. Thurner, A. Scheidler, F. Schfer, J. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, "Pandapower: An open-source python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Trans. Power Syst.*, vol. 33, no. 6, pp. 6510–6521, Nov. 2018.

[17] R. D. Zimmerman, C. E. Murillo-Snchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[18] A. Castillo and R. P. O'Neill, "Computational performance of solution techniques applied to the ACOPF," *Federal Energy Regulatory Commission, Increasing Efficiency through Improved Software*, pp. 1–34, Feb. 2013.

[19] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, March 2006.

[20] G. Lan and Z. Zhou, "Algorithms for stochastic optimization with functional or expectation constraints," *arXiv:1604.03887v7 [match.OC]*, 8 Aug. 2019.

[21] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *arXiv:1905.04497v2 [cs.LG]*, 4 Sep. 2019.

[22] M. Eisen and A. Ribeiro, "Optimal wireless resource allocation with random edge graph neural networks," *arXiv:1909.01865v2 [eess.SP]*, 3 Oct. 2019.

[23] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Conf. Robot Learning 2019*, Osaka, Japan, 30 Oct.-1 Nov. 2019, Int. Found. Robotics Res.

[24] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Trans. Signal Process.*, vol. 61, no. 7, pp. 1644–1656, Apr. 2013.

[25] D. I Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013.

[26] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges and applications," *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, May 2018.

[27] S. Segarra, A. G. Marques, and A. Ribeiro, "Optimal graph-filter design and applications to distributed linear network operators," *IEEE Trans. Signal Process.*, vol. 65, no. 15, pp. 4117–4131, Aug. 2017.