Accelerating Power Flow Calculations Using Traditional Solvers and Neural Networks

By

Ahmed Rosanally

Supervisor: Joseph Euzebe Tate
April 2022

B.A.Sc. Thesis

Division of Engineering Science
UNIVERSITY OF TORONTO

Accelerating Power Flow Calculations Using Traditional Solvers and Neural Networks

Ahmed Rosanally
Engineering Science
Undergraduate Department of Applied Science and Engineering
University of Toronto
2022

Abstract

The power flow (PF) problem aims to provide the voltage profile and power flows in a given power network at resting state. The most trusted approach to solving the problem is using the Newton-Raphson (NR) method which serves as the experimental ground truth for other methods. Those other methods used to accelerate the computation are the DC approximation, Fast-Decoupled PF, the 1D convolutional neural network (CNN) method and the graph neural network (GNN). In addition, a novel method was explored and tested and is called the Graph Neural Solver (GNS). The latter incorporates the basic invariant physics of the PF problem to obtain a very high speed of solution convergence and accuracy compared to other methods. The GNS method can be very precise with respect to the NR method, achieving an accuracy of 99.99% for the 118 bus system. It uses graphs to better represent the power network. Therefore, it is more convenient to visualize how the node values - namely the voltage magnitude and phase at each bus - update at each correction layer of the GNS. The GNS is a more promising method as it provides more information to solve the PF problem.

Acknowledgments

While writing this dissertation I have received a great deal of support and assistance behind the scenes.

I would first like to thank my supervisor, Professor Joseph Euzebe Tate, whose expertise was invaluable in formulating the scope of my research and methodology. His insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would also like to acknowledge my colleague and friend Liangjie Chen. I want to thank him for his patient support and for all of the opportunities he gave me to further my work.

I would also like to thank the course coordinators, Professor Alan Chong and Professor Lisa Romkey, for their valuable guidance and feedback on my deliverables. They provided me with the right tools needed to successfully complete my dissertation.

In addition, I would like to thank my parents for their wise counsel and sympathetic ear. They are always there for me. Finally, I could not have completed this dissertation without the support of my friends who provided stimulating discussions as well as happy distractions to rest my mind outside of my research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context

Ever since the world has been electrified in the late 19th century, optimizing solutions to the power flow (PF) problem has been of prime importance. The PF problem seeks to determine the voltage profile and power flows in a given power system at steady state. Solving the PF equations is common in industry for operations analysis and planning tools such as short-term security analysis, market analysis, maintenance scheduling and planning of generation and transmission facilities, and fuel procurement [1, p. 97]. In its AC version (ACPF), the problem consists of a system of nonlinear equations. The DC version (DCPF) finds a linear approximation to the problem to provide computational advantages over the ACPF model. An interesting application of solving power flow rapidly and accurately is in assessing power systems for uncertainty because of randomness in power generation and loads on the line. Solving PF with high speed and accuracy is essential to this application. The most common method to solve the ACPF problem is the Newton-Raphson (NR) method. It is the most trusted method and its solution is often used as an experimental ground truth for evaluating other methods. The NR method iteratively converges on a solution to the problem by updating given initial conditions. The NR method has a quadratic rate of convergence and iteration time increases linearly with the number of nodes [2]. It can also be difficult to find suitable initial conditions for the NR

method to converge. Previously, multilayer perceptrons (MLPs) have been used to improve the DCPF model or to directly predict the results of the ACPF model [3] [4] [5]. One-dimensional convolutional neural networks (1D CNNs) have outperformed MLPs by reducing the number of iterations and iteration time when applying the NR method [6]. Although they take longer to train, they require less memory to perform similarly to MLPs.

1.2 Goals

The primary goal is to discover a new machine learning model to speed up the process of solving the ACPF problem using the NR method. There will be various sub-objectives involved to achieve this goal. These include exploring different neural network architectures that solve the problem, exploring different hyperparameters that solve the problem more efficiently, and obtaining the information necessary to seek a larger and more diverse dataset to account for contingency scenarios.

1.3 Approach

The first technique is to explore the graph neural network (GNN). In one study, this architecture runs 4.5x faster in solving the optimal PF (OPF) problem compared to DCOPF optimizer, and 36x faster than the ACOPF optimizer [7]. Since solving this problem directly solves the much simpler ACPF problem, it would be exploring more GNNs to perform better than other methods in solving the ACPF. Secondly, in data generation, more focus could be placed on contingency scenarios where the neural network could produce initial conditions to perform contingency analysis better than the DCPF model. Also, the GNN would have to be efficient in translating a multi-node power system to a graph data structure that would be fed as input to tweakable GNNs on well-known machine learning packages such as PyTorch or TensorFlow. The main power system to be used is the IEEE 118-bus network in order to compare results with the existing methods. A well-documented power system simulation package for

MATLAB available at [8] will be used to run the simulation for the case file.

Chapter 2

Background

2.1 AC Power Flow

Before diving into the power flow equations, it is important to present the admittance matrix which is used to formulate them. This matrix will be at the root of the different solutions presented in the methods section. Its presentation facilitates understanding when translating the problem in the context of NNs.

Given an n-node power system, the $N \times N$ admittance matrix is defined as the following:

$$\overline{y}_{ii} \triangleq \sum_{j \in \Omega_i} (\overline{y}_{L_{ij}} + \frac{1}{2}\overline{y}_{S_{ij}})$$
$$\overline{y}_{ij} \triangleq -\overline{y}_{L_{ij}}$$

$\overline{y}_{L_{ij}}$ represents the per-unit[1] series admittance of the branch connecting buses i and j. $\overline{y}_{S_{ij}}$ represents the per-unit shunt admittance of the branch connecting buses i and j. Finally, Ω is the set of integer from 1 to n

For large power systems where N is very large, the Y matrix is sparse because each bus is connected to neighboring buses which are very few in comparison to N.

[1]The per-unit system is used to simplify power system analysis. The problem of having different voltage magnitudes across the power network disappears. Generally, the per-unit voltage value is nearly 1 puV and power transformers disappear when the circuit is converted to single-phase.

This observation is important because it captures the importance of bus connectivity. Moreover, once the admittance matrix is defined, it can be used to solve the matrix version of Ohm's law – $Y * V = I$. Here, V is the $N \times 1$ vector of phase voltages at each bus and I is the $N \times 1$ vector of corresponding current injections. Subsequently, the values of the admittance matrix can be used to solve the power flows at each bus.

The AC power flow problem consists of a set of nonlinear equations. In an n-node power system, there are a set of $2n$ equations which are the following:

$$p_i = v_i \sum_{k=1}^{n} y_{ik} v_k \cos(\delta_i - \delta_k - \theta_{ik}) \tag{2.1}$$

$$q_i = v_i \sum_{k=1}^{n} y_{ik} v_k \sin(\delta_i - \delta_k - \theta_{ik}) \tag{2.2}$$

p_i and q_i are the respective active and reactive power injections at bus i. $v_i \angle \delta_i$ is the voltage phasor at that bus and $y_{ik} \angle \theta_{ik}$ is the polar form of the complex admittance of the ik^{th} element of the admittance matrix .

Solving the power flow problem reveals the state variables of the power system - all voltages and phase angles at each bus. Once those are known, all currents and power flows through each branch could be computed. The core of the problem, from a machine learning perspective is predicting the state variable of the power system given the training data. Depending on what variables are known, each bus or node could be classified into three types: a PQ node, a PV node and a slack node. Any number of PQ and PV nodes could be specified and only one slack node is chosen as a reference. Once the problem is presented, the scene is set to discuss the main advantages and disadvantages of each of its solutions. The performance each method will be evaluated on the IEEE 118 bus system provided by MATPOWER [8].

2.2 Approach to Uncertainty

There is presence of uncertainty in a power system given anticipated randomness in generation and load. Due to this, the power system needs to be assessed. The

probabilistic power flow problem (PPF) aims to extract a probability distribution for the wanted output variables such as bus voltage magnitudes, phase and power flows given power generation and load inputs as random variables with associated distribution functions [9]. Different approaches exists to solve the PPF problem. They are often based on a trade-off between computational cost and accuracy. The Monte Carlo (MC) based techniques are the more reliable and accurate ones by far [9]. In such simulations, samples from input random variables are generated and then the deterministic PF solver is run for each of them. In large power system where uncertainty assessment is very important and the number of samples generated is very high, running a PF solver with high speed and accuracy is crucial.

Chapter 3

Methods

3.1 Newton-Raphson Method

The NR method is most commonly used for solving the nonlinear problem. The details of the algorithm will not be discussed but observations will be made based on [1, p. 111]. First, it is an iterative method which has a quadratic convergence rate and needs only a few cycles to get near the solution. It converges fairly quickly so long as the provided initial estimates are close to the solution. The method involves inverting the Jacobian matrix of partial derivatives of the power flows. This complicates the implementation of the algorithm and is computationally expensive[1]. Finally, due to the emergence of fractal patterns [11] in convergence, it is difficult to guess the initial values.

It takes 5 iterations and 11 ms to converge on the IEEE 118 bus system using the standard NR method. In a recent paper, a modified method called the NR Predictor Corrector (NR-PC) method takes only 4 iterations to converge on the 118 bus system [12]. A "Predictor" is used to close the gap between the estimate and the desired solution at convergence and a "Corrector" is used to get even closer to the unknown solution. The Predictor Corrector mechanism does not compute higher

[1]Since the Y matrix is a sparse matrix, the Jacobian is also sparse. There are sparse matrix algorithms [10, p. 588] which accelerate the NR method saving time when dealing with very large power systems. These algorithms are still complicated to implement and the space complexity still remains quite large.

order derivatives and so does not take up more space. This is because the values of the Jacobian matrix could also be reused for the next iteration [12, p. 787].

3.2 Fast Decoupled Power Flow

The FD technique is based on the idea that the active (p) and reactive (q) powers are decoupled. The former are mostly associated with voltage phase and the latter are mostly associated with voltage magnitudes [1, p.119]. This $p - q$ decoupling is the case for high voltage lightly loaded power systems. It is more competitive because the cost per iteration can be between four and five times smaller than that of the NR method. In terms of speed however, the convergence rate is roughly the same for the first few iterations and actually slows down as it approaches its final solution. Experimentally, this technique has proven to be better at converging to a solution in cases where line resistances are ignored [10, p. 108].

According to [12, p. 789], the FDPF method take 15 iterations and 9 ms to obtain a solution on the 118 bus system.

3.3 DC Linearized Power Flow

DCPF is a linear approximation around a given solution of ACPF. The approximation involves two assumptions which are generally true in transmission systems - shunt admittances and line resistances connecting two neighboring buses are ignored. This is because, generally, these systems are at very high voltages. Additionally, the per-unit voltages are assumed to be around 1 pu V and the phase angle difference between two neighboring buses are assumed to be null. The latter allows for the use of the small angle approximation which linearizes the ACPF problem. Apart from these assumptions, DCPF offers a huge speedup in very large power networks. This advantage provided by DCPF accelerates the problem making way for neural networks, namely the convolutional and graph neural networks, to be able to learn its fast converging solution.

3.4 1D CNN Method

The 1D CNN approach, used in conjunction with the DCPF method, obtains better initial conditions for the NR method to converge on an optimal solution to the ACPF problem. Certain assumptions mentioned in the DCPF approach which linearize the problem are not always met to the fullest. 1D CNNs trained on the DCPF results tune the initial good conditions which are then injected into the NR algorithm. By capturing local features in the voltage difference between neighboring buses, this approach allows for a faster convergence rate and lower iterations [6]. This method has an average computation time of 1.049 ms and takes an average of 1 iteration to converge on the 118 bus system [6, p. 6]. The issue with this method is its Euclidean nature. The 1D CNN architecture is sensitive to the order of the input itself which is arbitrary. Since CNN architectures were developed to train better on images which follow the 2D geometrical grid, the order of the input is suitable to that architecture. However, since power grids are networks which do not follow this geometry i.e. they are non-Euclidean, 1D CNNs are not as suitable for representing the problem. Power networks could be better represented by graphs. In this way, the GNN method is more promising.

3.5 GNN Methods

Graphs are a type of data structure that comprise of a set of nodes and edges which give them an advantage over 1D CNNs to capture the power network's buses and branches. In figure 3.1, we observe the graphical nature of the IEEE 118 Bus Test Case from MATPOWER [13]. The points which are not connected could be either the isolated buses, generators or out-of-service generators.

Typically, CNNs deal with images which lie over a "*flat*" domain meaning that the space is defined by vectors to indicate position. In the 1D CNN, these vectors indicate the next bus data in the case file. However, this clearly does not capture the bus neighborhoods within the network, or even whether certain buses are discon-

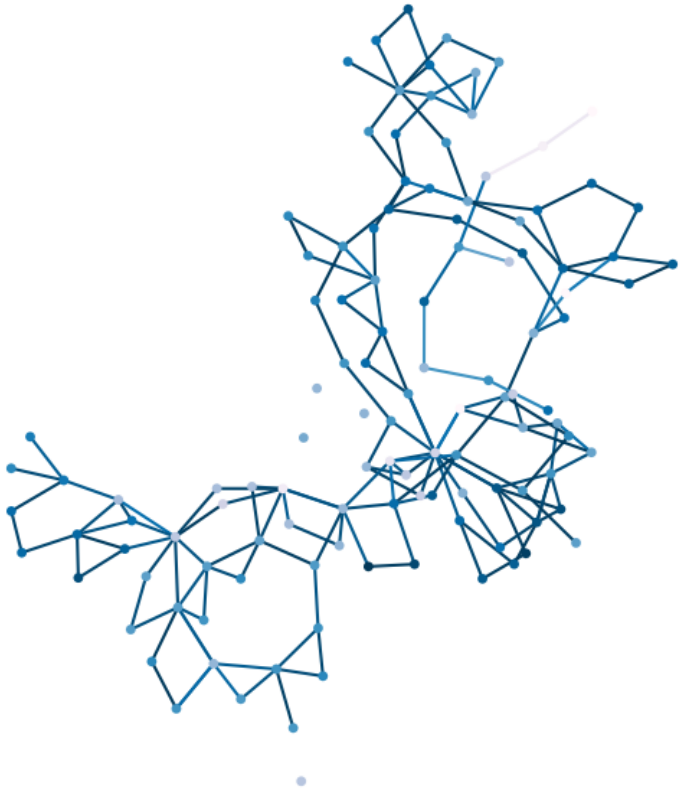


Figure 3.1: IEEE 118 Bus Power Network

nected as shown above. Graph neural networks help solve this representation issue. There are many different types of GNNs based on whether the edges are directed or undirected, the nodes are homogeneous or heterogeneous and whether the input is static or dynamic with respect to time [14, p. 59]. For the ACPF set up, the GNN is undirected. The set of nodes or buses is heterogeneous as they could be either PV, PQ or slack nodes. Finally, for the sake of simplicity the graph is static to capture the power network's state.

There are many variants of GNNs which have all shown exceptional performances on many deep learning tasks. Such tasks include power flow approximation, scenario generation and the safe operation of power grids and much more [15, p. 8]. For ACPF, GNNs are used to simply represent the topology of the power grid. One of the basic spatial GNNs - the learnable graph convolution network (LGCN) capitalizes on CNNs to compute hidden representations of the network. This means features of

the power system are better represented and will lead to better approximations for ACPF. In one study, a spatial GCN architecture achieved faster convergence on both *Texas* and *East Coast* datasets compared to the fully connected network with lower training data and parameters involved [16]. In spectral-based GNNs, more emphasis is placed on the convolution operation where it is defined in the spectral domain, thereby reducing the computing cost of the operation from $O(n^2)$ to $O(n)$. In a more recent study, a spectral GCN architecture allowed for more accuracy and a higher speed to solution convergence resulting in up to 70% improvement compared to the linearized method [17].

In the following section, the in-depth exploration of the Graph Neural Solver will be shown to give the reader a better understanding of the nature of its architecture. When this architecture is generalized, it is referred to as the Deep Statistical Solver. The latter will be discussed briefly. Both of these architectures will be heavily based on [18] and [19].

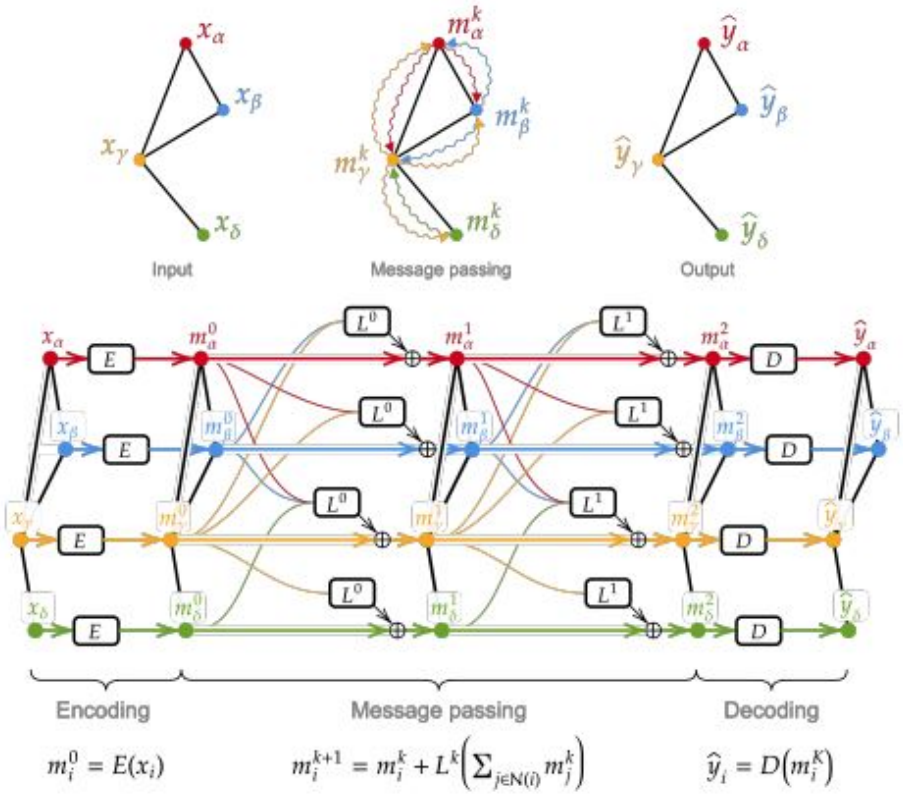
3.6 The Graph Neural Solver

The Graph Neural Solver (GNS) is a fully differentiable function that predicts power flow. It is a GNN subclass that shows that voltage magnitudes collapse everywhere on the grid. What makes it unique is that it encodes the known invariants and symmetries of the system directly inside its architecture. This makes them resilient by design [18, p. 2].

The goal of the GNS is to predict voltage magnitude (v) and phase (θ) at each node of the power network. The input to the architecture inspires from the MATPOWER case files [8]. It first takes a base apparent power. A list of buses including their active and reactive power demands as well as shunt conductance and shunt susceptance is added. Then it takes a list of branches including their line resistance, reactance, total charging susceptance, transformer phase shift angle and transformation ratio. Finally, it takes a list of generators including their maximum & minimum provided active power as well as its active power and voltage set points.

The GNS architecture inspires from the structure of the GNN which is straightforward. Figure 3.2 provides a useful visualization of the architecture.

Figure 3.2: The Graph Neural Network Architecture [18]



First, the input data x relies on a graph model. The output of the model is $\hat{y}$. In essence, the goal is to iteratively send messages across the edges of the graph structure which are seen as "correction updates" [18, p. 3]. There are three critical steps involved:

1. $m_i^0 = E(x_i)$: E is an encoder similar to a typical artificial neural network (ANN) which converts the input x_i at each bus i into a message m_i. This basically maps the input into a latent space which is typically lower-dimensional.

2. $m_i^{k+1} = m_i^k + L^k \sum_{j\in\mathcal{N}(i)} m_j^k$: The messages are then updated iteratively by ANNs L^k that takes as input the sum of the messages coming from every bus j in the neighborhood $\mathcal{N}$ of bus i. There are K such neural networks and each of

them trains differently from the other. Note that K is a tunable hyperparameter in this architecture.

3. $\hat{y}_i = D(m_i^K)$: The resulting messages m_i^K are all decoded using an ANN decoder D to find the predicted $\hat{y}_i$

Please note that there are $K + 2$ trainable neural networks $(E, L^0, ..., L^K, D)$. Figure 3.1 shows an example with $K = 2$ [18].

The GNS training includes four critical steps [18, p. 4]:

1. Initialization: This is also referred to as cold-start as seen in [6] where all voltages are set to 1 p.u. except for generators with voltage set points. All phase angles are set to 0 rad.

2. Global active compensation: This step makes sure that the power system reaches a steady state. This is dependent on a global parameter λ which stabilizes the global active power and local reactive power coming out of every generator on the grid.

3. Local power imbalance computation - $\Delta S_i^k = \Delta P_i^k + \sqrt[4]{-1}\Delta Q_i^k$: This step is useful in computing the power that is off the balanced state at every correction update $0, ..., K$. This estimates the offsets in both the active and reactive power at each bus i. This is the invariant physics portion that is incorporated in the loss function of the neural networks to learn the power flow problem..

4. Neural Network Update: The series of $K + 2$ ANNs previously defined are updated and correct the voltage, phase and message values at each bus or node. This is based on their current values, the invariant physics of the problem and the sum of the messages in their neighborhood. Note that the voltage values at buses connected to generators remain fixed at their voltage set points. The output is finally the v's and θ's at the last correction update K.

Note that if the bus has no generator, the line characteristics should also be included in order to calculate the v's and θ's. Another ANN is trained to capture these characteristics along with the messages in the neighborhood of a bus [18, p. 6].

3.6.1 Data Structure

The neural network trained on a set of T different power grids of the same mini-batch size. The power grids are generated using the scenario of one power outage to reproduce realistic failures. This is called the "N-1" criterion [18, p. 1]. The data is generated using uniform sampling and the numbers are chosen to replicate realistic scenarios accordingly [18, p. 7].

To put it simply, the data resembles the graph structure as the one seen in Figure 3.2.

Figure 3.3: A Simple Graph Structure [19, p. 2]

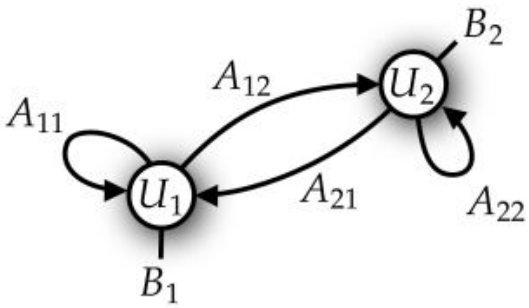


Table 3.1 summarizes what each of the symbol in Figure 3.2 mean in the context of power flow. The A_{ij}'s - where i and j are part of the natural numbers set each indexing a seperate bus - represent the edge or interaction between two buses or nodes of the graph. It is an adjacency matrix similar to the admittance matrix presented in section 2.1. The vector B_i represents the external inputs or "forces" that act on each node. This could be for instance external power injections at each bus. Lastly, the vector U_i represent the states of each node or bus. These are the magnitude and phase values at each bus in the final output of the graph neural network.

Note that the neural network is compatible with power grids of any shape and size [18, p. 7] since graph neural network is inherently able to adapt to input size. The GNS has been trained and tested on MATPOWER's case 14 and case 118 where the case number represents the number of buses or nodes of each grid.

Table 3.1: Data structure definitions

symbol	definition
A_{ij}	Line or branch information and the propagating messages connecting buses i and j
B_i	Bus information including the active and reactive power demands, shunt conductance and shunt susceptance
U_i	The desired output which are the voltage magnitudes and phases readings at each bus or node

3.6.2 Data Collection

In general, the data has been generated uniformly using a few IEEE standard bus cases. Noise has also been added at both the A_{ij}'s and B_i's to account for different scenarios. The power line or branch data has been kept roughly the same as it was uniformly generated from 90% to 110% of its initial values. The transformation ratio and phase shift value have been sampled from 0.8 to 1.2 and -0.2 rad to 0.2 rad, respectively. The generators' maximum and minimum active power remained the same, however the voltage set points were uniformly generated from 0.95 pu to 1.05 pu. Their initial active powers were uniformly sampled between 25% and 75% of their allowable range. The active power demands were sampled between 50% and 150% of their first values. They have been multiplied by a common factor to ensure power equilibrium. Lastly, the reactive power were uniformly sampled from 50% to 150% of their initial values [18, p. 7].

3.6.3 Experimentation

The model has been trained and tested using a classic machine learning approach. 89% of the data was used to train the model and the remaining 11% was used to test the model with respect to the NR method's output as an experimental ground truth. The model ran on two IEEE bus cases from MATPOWER: case14 and case118. For perspective, case14 took about 3 to 4 days to train on Cedar cluster from Compute Canada. The instance was trained using an NVIDIA P100 Pascal GPU. In terms of hyperparameters tuning, Table 3.2 summarizes this information. The number of

correction updates corresponds to the number of artificial neural networks that are training in the message propagating portion of the neural network architecture. Here, it is crucial that the GNS adapts to the intricacies of PF problem. Equation captures the loss function for one minibatch of size T. It is the sum of each of the losses coming from each neural network for each minibatch.

$$Loss = \sum_{k=1}^{K} \frac{\gamma^{K-k}}{N} \sum_{i=1}^{N} |\Delta S_i^k| \tag{3.1}$$

$|\Delta S_i^k|$ is the local power imbalance at each bus i for each correction update k. γ is a hyperparameter of the GNS that acts as a penalty factor with respect to the physics of the PF problem. It has been experimentally observed that adding this penalty factor at every correction update allows for better solution convergence and more accurate results [18, p. 7].

Table 3.2: Choice of hyperparameters [18]

Hyperparamter	Value
K - Number of correction updates	30
d - The latent dimension where input converts to message	10
Number of layers	2
Activation function	Leaky ReLU

Chapter 4

Results and Discussion

The best existing method is the NR method. Table 4.1 summarizes the machine learning statistics for the current best GNS model. The correlation is in terms of active power flows through the power lines. The model is highly accurate with regards to the NR method as it learns to minimize the loss in equation 3.1 which embeds the immutable physics of the PF problem at every correction update k.

Figure 4.1: Evolution of the Loss Across Correction Updates [20]

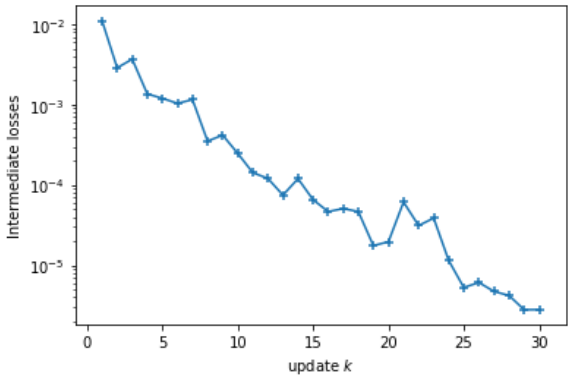


Table 4.1: Case 118 Results for Best Existing GNS Model

Statistics	Model 2
Correlation w.r.t. NR	99.99%
Loss 10th perc.	1.7 E-7
Loss median	2.8 E-7
Loss 90th perc.	4.8 E-7

In addition, Figure 4.1 shows the downward trend of the loss with respect to the correction updates. The loss drops in orders of magnitude. This is also the effect of adding the penalty factor γ at every correction update allowing gradient to flow through the first few correction update layers [18, p. 5]. Finally, in Figures 4.2-4.5, we also see how the GNS evolves its prediction across these layers thereby succeeding in minimizing the violation of Kirchhoff's law. This model of the GNS has 30 correction update layers and solves the problem for the 118 bus system.

Figure 4.2: Voltage Magnitude at Correction Update 1 [20]

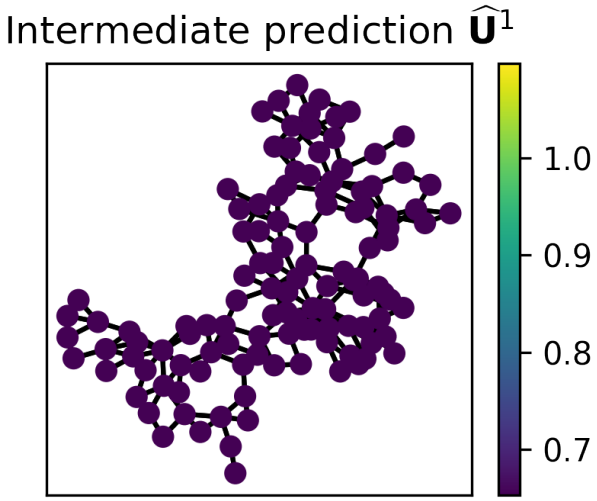


Figure 4.3: Voltage Magnitude at Correction Update 10 [20]

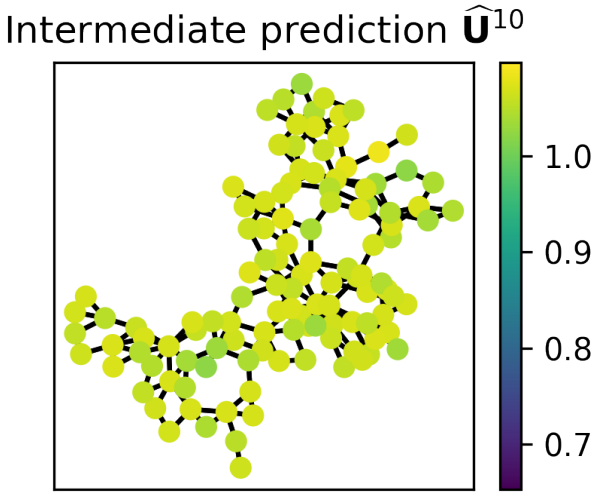


Figure 4.4: Voltage Magnitude at Correction Update 20 [20]

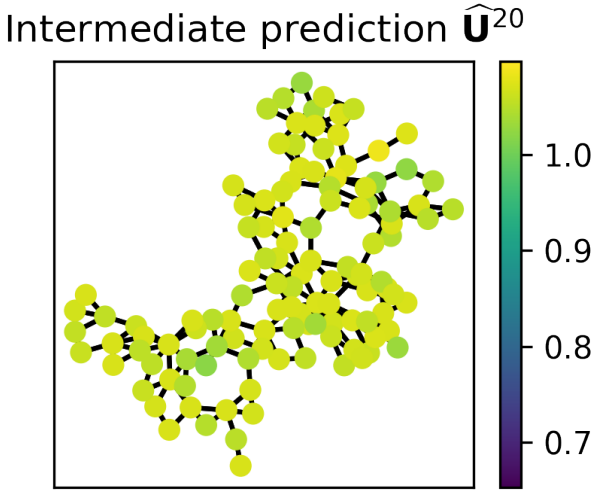
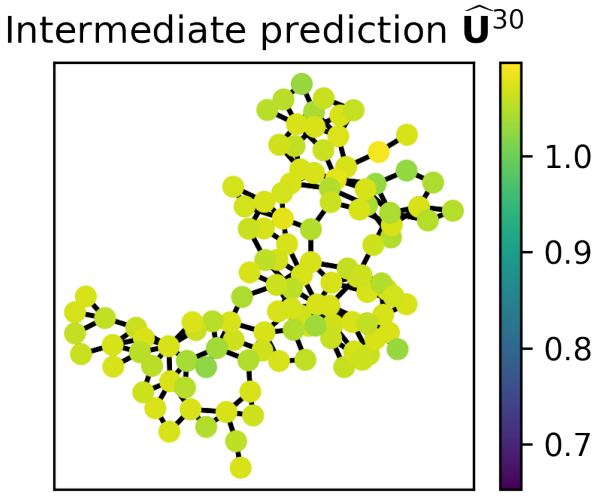


Figure 4.5: Voltage Magnitude at Correction Update 30 [20]



The GNS offers better representation of the 118 power system as seen above. With visualization tools, it is possible to see how each predictions changes across each NN layer as the GNS minimizes the total loss. These are some key advantages that this method offers as GNNs have the most expressive power compared to all other NNs. In terms of speed, the GNS has about the same computational speed as the NR method [18].

Table 4.2: Performance of Methods on Case 118

Method	Computation Time (ms)	Iterations
NR	11	5
NR-PC	11	4
1D CNN	1.05	1
FD	9	15

Table 4.2 shows the speed performance of some of the methods on the 118 bus system. The 1D CNN still outclasses the rest of the methods achieving a computation time of 1.05 ms and converging in about 1 iteration. However, it relies on the results of the NR method to hot start the method. The Fast Decoupled method achieves a faster computation time than the NR method but it still takes more iterations to converge meaning there are more computational steps to take to reach to the final solution. The GNS takes about the same time as the NR method to converge and does not rely on previous solutions to solve the problem. However, due to its high expressiveness, it requires more time to train compared to the 1D CNN method as many NN layers are added for the correction updates of the architecture.

Chapter 5

Conclusion and Future Work

In conclusion, there are many methods used to solve the PF problem. The most trustworthy method is the NR method and it is used as a basis for measuring accuracy with respect to other methods. The GNS is a subclass of GNNs. It embeds the invariant physics of the problem by minimizing the violations of Kirchhoff's law. It is the most accurate method. GNNs are more promising in this way. They are the most expressive of all other neural networks. It provides more information in three ways: (1) they are visualization tools to help see the evolution of the solution for GNNs; (2) they represent the grid in a more suitable way; and (3) they properly capture the process of the solving PF. In terms of future work, the GNN path for solving PF needs to be better tested for scalability, for instance on a 10k bus system. Other scenarios could be tested as well such as adding more stress on the system by heavily loading them or by adding more outages. Lastly, we could optimize the architecture for training by evaluating the trade-offs between accuracy and space complexity.

Appendix A

Code

- To view all the code related to the GNS method please see the Graph Neural Solver Repository.
- To view the code related to 1D CNN method please visit the 1D CNN Repository.

Bibliography

[1] Antonio J. Conejo and Luis Baringo. *Power System Operations*. Power Electronics and Power Systems. Springer Publishing, 2018. ISBN: 9783319694078.

[2] Christian Blatter. *Proof that Newton Raphson method has quadratic convergence*. Mar. 2015. URL: https://math.stackexchange.com/questions/786453/proof-that-newton-raphson-method-has-quadratic-convergence.

[3] Meriem Fikri et al. "Power Flow Analysis by Numerical Techniques and Artificial Neural Networks". In: *2018 Renewable Energies, Power Systems & Green Inclusive Economy (REPS-GIE)*. Casablanca: IEEE, Apr. 2018.

[4] S. B. Efe and M. Cebecci. "Power Flow Analysis by Artificial Neural Network". In: *International Journal of Energy and Power Engineering* 2 (6 2013), pp. 204–208.

[5] Wael Abdullah Alsulami and R Sreerama Kumar. "Artificial neural network based load flow solution of Saudi national grid". In: *2017 Saudi Arabia Smart Grid (SASG)*. Jeddah: IEEE, Dec. 2017.

[6] Liangjie Chen and Joseph Euzebe Tate. *Hot-Starting the Ac Power Flow with Convolutional Neural Networks*. 2020. arXiv: 2004.09342 [eess.SY].

[7] Frederik Diehl. "Warm-Starting AC Optimal Power Flow with Graph Neural Networks". In: 2019.

[8] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Thomas Robert John. "MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Powe Systems Research and Education". In: *IEEE Transactions on Power Systems* 26.1 (2011), pp. 12–19. DOI: 10.1109/TPWRS.2010.2051168.

[9] Mahdi Hajian, William D. Rosehart, and Hamidreza Zareipour. "Probabilistic Power Flow by Monte Carlo Simulation With Latin Supercube Sampling". In: *IEEE Transactions on Power Systems* 28.2 (2013), pp. 1550–1559. DOI: 10.1109/TPWRS.2012.2214447.

[10] Antonio Gomez-Exposito, Antonio J. Conejo, and Claudio A. Cañizares. *Electric Energy Systems Analysis and Operation*. CRC Press, 2018. ISBN: 9781315192246. DOI: 10.1201/9781315192246.

[11] Bhumika Gupta. "Newton Raphson Fractals:A Review". In: *International Journal of Emerging Trends & Technology in Compute Science (IJETTCS)*. Vol. 2. 2. 2013, pp. 119–123.

[12] Marcos Tostado, Salah Kamel, and Francisco Jurado. "Developed Newton-Raphson based Predictor-Corrector load flow approach with high convergence rate". In: *International Journal of Electrical Power & Energy Systems* 105 (2019), pp. 785–792. ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2018.09.021. URL: https://www.sciencedirect.com/science/article/pii/S0142061518301674.

[13] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Thomas Robert John. "MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Powe Systems Research and Education". In: *IEEE Transactions on Power Systems* 26.1 (2011), pp. 12–19. DOI: 10.1109/TPWRS.2010.2051168.

[14] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *AI Open* 1 (2020), pp. 57–81. ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2021.01.001. URL: https://www.sciencedirect.com/science/article/pii/S2666651021000012.

[15] Wenlong Liao et al. "A Review of Graph Neural Networks and Their Applications in Power Systems". In: *CoRR* abs/2101.10025 (2021). arXiv: 2101.10025. URL: https://arxiv.org/abs/2101.10025.

[16] Valentin Bolz, Johannes Rueß, and Andreas Zell. "Power Flow Approximation Based on Graph Convolutional Networks". In: *2019 18th IEEE International Conference On Machine Learning An Applications (ICMLA)*. 2019, pp. 1679–1686. DOI: 10.1109/ICMLA.2019.00274.

[17] Dawei Wang et al. "Probabilistic Power Flow Solution with Graph Convolutional Network". In: *2020 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*. 2020, pp. 650–654. DOI: 10.1109/ISGT-Europe47291.2020.9248786.

[18] Balthazar Donon et al. "Neural Networks for Power Flow : Graph Neural Solver". In: *Electric Power Systems Research* 189 (Dec. 2020), p. 106547. URL: https://hal.archives-ouvertes.fr/hal-02372741.

[19] Balthazar Donon et al. “Deep Statistical Solvers”. In: *NeurIPS 2020 - 34th Conference on Neural Information Processin Systems*. Vancouver / Virtuel, Canada, Dec. 2020. URL: https://hal.inria.fr/hal-02974541.

[20] Balthazar Donon and Ahmed Rosanally. “Deep Statistical Solvers”. In: URL: https://github.com/ll-0-ll/DeepStatisticalSolvers.