AMERICAN UNIVERSITY OF BEIRUT

Project 49:
Optimal Power Flow via Machine Learning

by

Mohammad F. El Hajj Chehade, Intelligent Power Systems
Mohamad Al Tawil, Intelligent Power Systems
Karim Khalife, Power & Energy Systems

Advisor: Prof. Rabih Jabr

A final year project
submitted in partial fulfillment of the requirements
for the degree of Bachelor of Engineering
to the Department of Electrical and Computer Engineering
of the Maroun Semaan Faculty of Engineering and Architecture (MSFEA)
at the American University of Beirut
Beirut, Lebanon

Acknowledgments

We would like to thank our advisor and dear professor Dr. Rabih Jabr. His consistent guidance and contructive feedback were instrumental to the success of the project. Even during the darkest times when our project was sinking, his advice and words of wisdom were the rescue boat.

Executive Summary

The optimal power flow (OPF) is one of the most essential problems in the operation of a power system. This problem is used to determine the minimal cost of operating a power network while abiding by the rules of energy balance, line flows and generator limits. Despite being formulated more than half a decade ago, the problem still lacks a fast solution that can solve the it in real-time and produce near-optimal results. The need for a new, more rapid and accurate solution is even more crucial nowadays with the integration of renewable energy sources.

Machine learning has emerged in the past decade as a very powerful subset of artificial intelligence. Methods that use machine learning have been introduced to solve the optimal power flow problem. One such method uses reinforcement learning (RL) that models the problem as a sequential decision making process, where an intelligent agents tries to learn an optimal policy, corresponding to the optimal solution, through maximizing a certain reward function by taking certain actions from certain states. Although it is able to yield very fast results, the RL agent requires extensive training.

Deep learning is another machine learning field, where the machine learns a certain function in a structure called a neural network (NN). The NN constitutes several layers that learn input-output relationships with a high-level of abstraction. A trained NN is shown to be able to solve the OPF problem at a very fast rate. NNs, however, require a large set of historical data corresponding to several instances of the solved OPF.

In this project, motivated by a need for a fast OPF solver, a learning-based approach has been explored. Due to the time frame of the project and the infeasibility to apply RL, NNs are explored as possible solvers. The problem of acquiring a historical dataset is solved by developing an algorithm to generate such data from a traditional solver.

The project first introduces the OPF problem and describes the relevant work done in solving it. Then, the proposed NN solution is explored, along with the technical requirements of the project, constraints and applicable standards. Af-

ter that, the implementation of the solution for a variation of the problem, the DC-OPF, is thoroughly discussed and includes the following points:

1. Generating the data set through a developed program
2. Pre-processing the data
3. Constructing the neural network
4. Training the neural network
5. Testing the neural network

The results for training and testing were found to be very satisfactory. Special schemes have also been discussed for a theoretical guarantee of a feasible solution and the improvement in the performance of the network in response to certain constraints.

The work in the upcoming semester will be focused on generalizing the solver to handle the full AC-OPF problem and developing a graphical user interface for it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The heart of a reliable and efficient Independent System Operator (ISO) is the optimal power flow (OPF) problem [1]. Since its establishment in the 1960s, OPF has been formulated in various forms as a static non-linear problem that solves optimal states of operation for certain variables in the power network [2]. The most common form of the OPF problem aims to minimize the operating cost of thermal resources, such as the fuel cost, through the optimal tuning of the electrical quantities while satisfying certain equality and inequality constraints in the network [3] [4]. The overall state of the system is determined by the load bus voltages and line flows, and the aim would be to control the generator active power, or sometimes the transformer tap settings or reactive power sources, to minimize a certain cost or loss in the system [3]. The problem is generally limited by equality constraints related to power flow and inequality constraints determined by bounds on the operation of controllable variables such as voltages in generators [5].

Due to the heavy role that OPF plays in the determination of the cost of operation of a power system, it has gained the interest of many utilities and has been established as one of the most necessary operational demands [3] [6]. However, a large-scale, highly non-linear, and non-convex problem in its nature [3], even after 50 years of its formulation, the optimal power flow problem still lacks a quick and robust solving mechanism.

1.1 Motivation

The complexity of OPF has economical, electrical, and computational roots [1]. Nonlinear pricing schemes are required to achieve economic equilibrium in electricity markets. Further nonlinearities are introduced by the power flow model of power networks. In addition, the optimization in itself has lots of nonconvexities introduced by discrete variables and various functions.

Even though OPF has benefited the most from the advances in numerical computing and computer evolution [2, 7], current software tools still use lots of approximations and human intervention to address some real-time issues, which alter the production prices and lead to enormous economic losses. An estimate of the losses due to the currently used software is in the order of tens of billions of dollars per year [8]. Moreover, an increase of just 5% in the optimality of the solvers would lead to a saving of about \$12 billion in the United States and \$87 billion worldwide [1].

In order to minimize such losses, a full AC-based software that uses less approximations must be used. However, a switch to such a solver would increase the time needed to solve one single instance of the problem from minutes to hours, making the overall idea impractical, even in day-ahead planning [1]. Even if such a method exists, unneeded harm would be invoked on the environment from the heavy emissions and dissipated energy [1].

The demand for faster, more accurate solvers for OPF becomes more critical with the current developments in the power grid. Grids are welcoming more distributed energy resources (DERs) such as solar and wind energy systems and electric vehicle charging stations. These sources introduce stochasticity and rapid fluctuations to the network and require rapid responses in real-time from the network control centers [9]. Conventional OPF is insufficient and new methods for solving the problem must be established.

1.2 Desired Needs

As a result, traditional iterative OPF solvers must be replaced by new solvers that are faster and more robust. The new solvers must be capable of handling a new instance of the OPF problem every couple of minutes and achieve a certain degree of optimality.

Chapter 2

Technical Background

Several techniques to solving the OPF problem have been proposed in recent decades. These methods could be grouped into deterministic, probabilistic and learning-based.

2.1 Deterministic Methods

Deterministic methods require exact values for load demand, energy generation and power network states. Mathematical methods are proposed in [10] and swarm intelligence methods are found in [11] [12]. The primal-dual interior-point approach is used in [13]. The convex relaxation approach is described in [14] for addressing the AC OPF problem for radial networks using second-order cone programming (SOCP). Semi-definite programming (SDP) for meshed networks is presented in [15]. Nonlinear programming [16], quadratic programming [17], linear programming [18], Newton-based approaches [19] and sequential unconstrained reduction methodology [20] are also examined.

2.1.1 Discussions

The techniques mentioned above are unstable in terms of convergence, particularly in large-scale systems, due to the non-linearities in the problem. Hence, their applicability are restricted in real-time operation. Furthermore, the performance of intelligence-based methods is drastically limited by the uncertainty in load demand and the intermittency of renewable energy sources [21].

Nonlinear programming-based procedures, in general, have several downsides, such as unstable convergence qualities and algorithmic complexity. The piecewise quadratic cost approximation is one of the drawbacks of quadratic programming-based approaches. Newton-based approaches have convergence characteristics that are sensitive to initial conditions. As for sequential unconstrained minimization, when the penalty factors are excessively big, these approaches are known to demonstrate numerical difficulties.

Although linear programming methods are quick and accurate, their disadvantages remain related to piece-wise linear cost approximation. Although computationally efficient, if the step size is not appropriately set, the sub-linear problem may have a solution that is infeasible in the original nonlinear domain [22].

2.2 Probabilistic Methods

Another approach is the probabilistic, or model-based, OPF (POPF), that tries to model the uncertainty in load and generation data. Numerous approaches have been proposed to cope with the uncertainty of the distribution network. Robust optimization and stochastic programming for distribution network (DN) optimization have been suggested in [23, 24] . POPF has been also tackled in the literature using meta-heuristic algorithms, such as the artificial bee colony algorithm [25], Harris Hawks optimization (HHO) [26], and grey wolf optimization (GWO) [27]. The stochastic programming-based approaches assume information of the distribution of uncertain variables, which is used to build scenarios of uncertainty realizations.

Unlike stochastic programming, robust optimization methods deal with uncertainties by establishing an uncertainty set and searching for solutions that are resilient to all realizations inside the set. In [28,29], robust optimization-based approaches for DN management are suggested. [28] presents defines the uncertainty set using a convex hull tool. [29] presents a robust quadratic method to smart DN operation. Chance-constrained [30] and model predictive control (MPC) [31] approaches are also applied.

2.2.1 Discussions

The stochastic models tend to have more realistic models of the power network, capturing the random variations in load demand, as opposed to the deterministic approach. However, since a huge number of situations must be examined, these approaches have high computational load. Furthermore, in reality, determining the probability distribution of unknown variables is challenging [32].

Similarly, the MPC algorithm's success is determined by the accuracy with which renewable energy generation and load demand are predicted. Moreover, when a new circumstance is encountered, the aforementioned approaches must partially or entirely address a stochastic nonlinear problem, which may be a lengthy process [33]. As a result, these strategies may be inapplicable to real-time control challenges.

2.3 Learning-based Methods

Learning-based methods use machine learning and data-driven insights to solve the OPF problem. Many types of machine learning models of various complexities have been used in OPF. This section summarizes them into supervised learning, deep learning in specific, and reinforcement learning methods.

2.3.1 Deep Learning Methods

Machine Learning (ML) approaches may be used to solve versions of OPF issues in order to find a more cost-effective solution and reduce the computational strain on solvers. There have been reports of several supervised-learning-based algorithms that mimic OPF solutions while greatly increasing solving speed. Supervised learning is the field of machine learning that necessitates the availability of training data and output samples. The "supervisor" dictates the machine to learn labels through input-output relationships in the training data [34].

A subset of supervised learning is deep learning, which allows the models to learn the data accurately through multiple processing layers that deal with representations of data with high levels of abstraction [35]. Deep learning in general relies on structures called deep neural networks (DNN). Neural networks (NN) have been widely devised in power systems [36].

Discussions

The different forms of deep learning models have one thing in common: they all rely on supervised learning techniques to train neural networks. These techniques employ huge simulations that must be performed ahead of time to find approximate values of the optimal results. In order to solve this problem, several works such as [37,38] rely on generating this dataset via well-known OPF solvers rather than getting real datasets. Although such models rely on data from traditional solvers, once the neural network completes the training phase, it will be able to replicate the results, and with proper architecture, improve on them in terms of optimality and in a much reduced duration.

Training a neural network relies on solving an optimization problem that minimizes a certain loss function such as a mean-squared error. However, due to its ambiguous nature of operation that imitates a "black box", a minimal loss objective may be achieved but does not guarantee a practical solution to the OPF problem, which satisfies the constraints of the problem. In order to address that issue, [37] includes a penalty term for any constraint violation in their loss function.

The penalty terms may work in the case of inequality constraints, but they fail in equality constraints that address the power balance in the system. For that reasons, [37] uses a NN framework that only predicts a set of the control variables. The other constraints, related to power balance, are then deduced from solving the power flow problem.

2.3.2 Reinforcement Learning Methods

Reinforcement Learning (RL) is a strong learning model that learns optimal decisions through having an intelligent agent interact and learn in a certain environment [39]. Situated between the supervised and unsupervised branches of machine learning, reinforcement learning models are deployed in sequential decision-making problems [40]. Unlike supervised learning, computers are able to learn useful insights about the system without the need for historical, labeled data [39].

Deep Reinforcement Learning (DRL) combines RL with deep learning [41]. Various DRL algorithms have been used to solve different forms of the OPF problem, such as the Proximal Policy Optimization (PPO) in [21], Deep Determinstic Policy Gradients (DDPG) [42], Twin-delayed Determinstic Policy Gradient (TD3) [43], Lagrangian-based DRL [44] and multi-agent systems (MAS) [45]. The common ground among these algorithms is that they use policy-based algorithms in their models. Rather than discretizing the action-space as in the case of simple RL models like Q-learning, policy-based methods directly work on the policy space [46], making them more favorable in continuous domains, as in the case of the OPF problem.

Discussions

The solutions proposed by the DRL agent achieve near-optimal solutions with full satisfaction of the constraints. By the time the training phase is done, the pre-trained agent is able to solve new scenarios of test data in as little as a few milliseconds [21], hugely decreasing the run-time demand of the OPF and enabling its deployment in real-time. The results of the training can then be easily scaled to new instances in the power network and perform better on new scenarios of data than traditional solvers [21].

However, in order to reach a final model ready for deployment, the RL model requires extensive training. RL models, especially deep models, are known to be very costly in terms of training time. Training a simple RL may require several hours or even days. The model depends on a set of hyperparameters that require precise, manual tuning. The performance of an RL algorithm is mostly determined by the selection of these hyperparameters, but no optimal, or at least efficient, hyperparameters modification mechanism exists to guide this process. Even if the training process is successful, it may require many training iterations that could last weeks or even months to reach the final model.

2.4 Summary

Several methods have been used historically to solve the OPF problem. The traditional methods can be split into deterministic, that use exact values for load demand and power generation, and probabilistic, that model the uncertainties in different elements of the power network. These methods have been proven to be inefficient when used in the full-OPF problem, due to their computational complexity and lack of accuracy in certain situations.

The use of machine learning techniques, especially deep learning, relaxes a lot of the complexity encountered while solving the full OPF problem. Neural networks are known to be good function approximators that deduce complex relationships in the power grid data and require minimal data pre-processing. However, a supervised learning method in nature, deep learning requires large amounts of historical labeled data with accurate output values. This means that the OPF must be solved numerous times beforehand, which may be time-consuming.

Another field of machine learning that has enjoyed huge success in the past decade is reinforcement learning. Reinforcement learning serves as a common ground between supervised and unsupervised learning, and helps the machine, or the agent, learn the optimal policy through taking actions in a certain environment.

Reinforcement learning algorithms that deploy deep learning in their models are known as Deep Reinforcement Learning (DRL) models. DRL makes use of the computational efficiency of neural networks without the need for any prelabeled data. However, even the simplest DRL algorithm requires several hours of training, and the training process must repeated for different sets of hyperparameters.

The aim of this project is to find an efficient learning-based model to solve the OPF problem and evaluate their performance in terms of convergence and speed.

Chapter 3

Proposed Solution Methodology

This work aims to design a deep learning model to solve the optimal power flow problem. The neural network used is a feed-forward artificial neural network architecture.

In this section, the OPF problem is first formulated as a large-scale non-linear and non-convex optimization problem. After that, an overview of supervised machine learning is presented. Thenceforth, the neural network architecture is explained. Finally, the requirement specifications and deliverables are briefly discussed.

3.1 Problem Formulation

OPF Formulation

The OPF problem has been modeled in various forms, depending on the desired objective. In this work, the problem will focus on minimizing the total cost of operation of the thermal sources in the power network.

Objective Function: The cost of power production is determined from the cost function. This cost function maps the amount of power generated to the associated cost. This function is derived empirically from a heat rate curve that outputs the generator power from the input thermal energy and fuel cost per thermal unit [5]. In most cases, the cost function is a quadratic polynomial [3], and the objective function can be modeled as follows:

$$min \sum_{i \in G} (c_{2i} P_{gi}^2 + c_{1i} P_{gi} + c_{0i}) \tag{3.1}$$

where G is the set of generators in the system and c_{2i},c_{1i} and c_{0i} are non-negative coefficients of the cost function.

The constraints in the OPF optimization problem could be grouped into inequality and equality constraints. The former group deals with limits imposed on various electrical quantities while the latter ensures power balance in the network.

Inequality Constraints: These constraints are related to limits on voltage values in all nodes and active and reactive powers in generators. Upper bounds on voltage values are exerted by limits on circuit breaker capabilities while lower bounds are due to generator and motor ratings [1]. The power generation is bounded from above by the rated capacity of the generator and from below by the flame limits required to ensure fuel combustion. Line flow constraints are represented in the form of apparent powers and depict thermal limits imposed on the line [1].

$$V_{G_i}^{min} \leq V_{G_i} \leq V_{G_i}^{max} \qquad \forall i \in N \tag{3.2}$$

$$P_{G_i}^{min} \leq P_{G_i} \leq P_{G_i}^{max} \qquad \forall i \in G \tag{3.3}$$

$$Q_{G_i}^{min} \leq Q_{G_i} \leq Q_{G_i}^{max} \qquad \forall i \in G \tag{3.4}$$

$$|S_{flow_{i,l}}| \leq S_{flow_{i,l}}^{max} \qquad \forall (i,l) \in L \tag{3.5}$$

where N and L are the respective sets of nodes and branches in the power network.

Equality Constraints: These constraints deal with power balance in the network:

$$P_{G_i} - P_{D_i} = V_i \sum_{j \in N} V_j (G_{ij} \cos \delta_{ij} + B_{ij} \sin \delta_{ij}) \qquad \forall i \in N \tag{3.6}$$

$$Q_{G_i} - Q_{D_i} = V_i \sum_{j \in N} V_j (G_{ij} \sin \delta_{ij} - B_{ij} \cos \delta_{ij}) \qquad \forall i \in N \tag{3.7}$$

where P_{G_i}, P_{D_i}, Q_{G_i} and Q_{D_i} are the real and reactive power generated and demanded in node i respectively, G_{ij} and B_{ij} are the conductance and susceptance on the line (i,j) and δ_{ij} is the voltage phase angle difference between nodes i and j.

3.1.1 Supervised Machine Learning

Supervised machine learning is a technique in which the machine learns from past experience to take decisions on future events. Suppose the problem is defined as shown in figure 3.1. The general aim of a computer or model is to produce the output based on a certain input and a certain function that governs the input-output relationship.

As in the case of a traditional computer program, the computer is fed with a certain function f and an input vector $\mathbf{X}$. In this case, depicted in Fig. 3.2, f

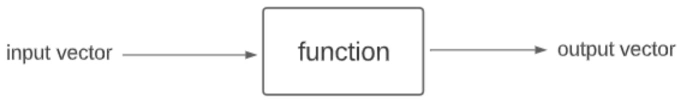


Figure 3.1: The operation of a traditional programmed machine.

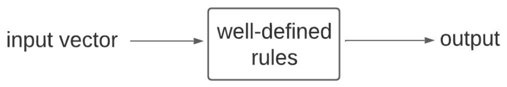


Figure 3.2: The operation of a traditional programmed machine.

is well-defined. For example, f could be a law of physics such as the rule for the elastic potential energy in a spring, where the input is the compression/elongation of the spring. The relationship is well-established and is illustrated below:

$$f : x \mapsto E \tag{3.8}$$

$$E = f(x) = \frac{1}{2}kx^2 \tag{3.9}$$

where E is the elastic potential energy, x is the spring compression/elongation and k is a constant. In this case, f is well-known.

More generally defined, given a certain function f and an input vector $\mathbf{X}$, the computer program is able to determine an estimate $\hat{\mathbf{y}}$ of the output as follows:

$$\hat{\mathbf{y}} = f(\mathbf{X}) \tag{3.10}$$

In this case, both $\mathbf{X}$ and f are given and are used to estimate $\hat{\mathbf{y}}$. The process involves solving the above equation and requires a single iteration only.

In some cases, f may not be fully-known, or even if known, may be very expensive to compute. For example, f could involve a set of equations that include expensive operations such as matrix inversions, dynamic programming, differential equations, etc. In these cases, the traditional computer program may fail to find an accurate estimate $\hat{\mathbf{y}}$ in a given time frame. Therefore, other techniques such as supervised learning may be used.

Instead of being supplied with the function f, the supervised learning machine tries to learn it from experience. More precisely, the machine is fed with the input vector $\mathbf{X}$ and the corresponding real output vector $\mathbf{y}$. This output vector could be obtained through experimentation or through traditional solvers that have been previously used to solve for $\mathbf{y}$. This set of historical data is compiled

input values
output values
SL model
input-output relationships

Figure 3.3: A supervised learning (SL) model.

in a dataset containing a certain number N of instances, where for each $\mathbf{X}$, there exists a $\mathbf{y}$. The aim of the supervised machine learning process is to infer the relationship between the input $\mathbf{X}$ and output $\mathbf{y}$. This is illustrated in figure 3.3.

The aim of the training process is to approximate the function $\hat{f}$ that relates $\mathbf{y}$ to $\mathbf{X}$. The supervised learning model precisely learns a set of parameters, and uses them to form an estimate $\hat{\mathbf{y}}$ of the real $\mathbf{y}$ from the vector $\mathbf{X}$. This is depicted below:

$$\hat{\mathbf{y}} = \hat{f}(w, b|\mathbf{X}) \tag{3.11}$$

where $\hat{\mathbf{y}}$ is the estimate of the real output $\mathbf{y}$, $\mathbf{X}$ is the input vector, $\hat{f}$ is the estimate of the function relating the input to the output, and w and b are known as the weights and biases and are the parameters that define the supervised learning model.

In order to reach a good-enough estimate $\hat{f}$, the model enters a training process formed of several iterations, where in each iteration, w and b are adjusted in a way to minimize the error of the prediction. In other words, the training process involves an optimization problem, where optimal values for w and b are obtained from minimizing the following objective function:

$$min_{w,b}\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) \tag{3.12}$$

where w and b are the parameters of the machine learning model, $\mathbf{y}$ is the real value of the output, which is found in the dataset and obtained through experimentation or traditional solvers and $\hat{y}$ is the estimated value for the output given an input $\mathbf{X}$ shown in equation 3.11. The loss function $\mathcal{L}$ depends on the nature of the problem and output value. Examples of error functions include the mean-squared error, mean-absolute error and binary cross-entropy error. The choice of error is strongly related to the type of output variable $\mathbf{y}$. In this discussion, two branches of supervised learning emerge: '

Classification:

In this case, the output variable may only take discrete values. These values may be binary, such as in the case of predicting whether a person has breast cancer

Table 3.1: Different types of supervised learning.

Problem	Output Variable	Example
Classification	Discrete	Breast Cancer Diagnosis
Regression	Continuous	House Price Estimation

(output) based on an input vector of pixels forming an X-Ray image (input). It can also be formed of a larger set of discrete values, like in the case of predicting a person's emotions (anger, happiness, nervousness, etc) from his/her speech (input). Many classification problems use a binary cross-entropy loss function to optimize the parameters of the model. This work does not focus on classification problems.

Regression:

In regression problems, the output variable can take continuous values. For example, the output variable could be the price of a house, whereas the input vector may include the number of rooms, the location of the house, etc. Linear regression is a very common, yet simple, family of regression problems. Most regression problems rely on minimizing the mean-squared error of the estimated output. In this project, the OPF problem is converted into a regression problem, where the output target variable includes a continuous value for control variables such as the active and reactive power in the generating units.

Hence, the nature of the problem, whether a classification or regression, is strictly based on the type of the output variable, whether discrete or continuous. The difference between the two methods is better illustrated in the table 3.1.

In order to have a clearer picture of the training process, if we take the example of the house price, suppose that a customer would like to estimate the price based on the number of rooms and location. A supervised learning model is constructed and uses a dataset that contains the number of rooms in the house and the house location as input variables, and the price of the house as output variable. This dataset can be acquired from real-life examples.

The supervised learning model takes these input values, and through optimally tuning its set of parameters, finds an estimate of the output variable. As long as the error in estimation value is larger than a certain stopping criteria, the parameters continue to be updated based on a well-established update scheme.

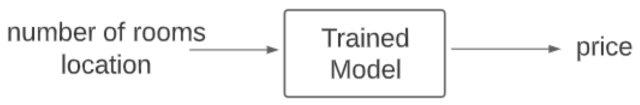


Figure 3.4: The trained model is used on a new instance of input data.

Once the stopping criterion is met, the model finishes training and enters the testing phase, where its performance is evaluated on a set of new data points that it has never seen before. The metric based on which it is tested could be the mean-squared error, percentage error, etc. At this stage, the parameters of the model can no longer be updated, as in the training stage. If this testing process yields satisfactory results, the model can be deployed and used to estimate prices of houses using a new instance of input data, shown in figure 3.4.

In a nutshell, supervised learning involves three different stages:

1. Training stage
2. Testing stage
3. Deployment stage

The technique is very similar to the way humans learn from experience, where in the case of a computer, the training, or learning, is acquired from a certain dataset.

3.2 Artificial Neural Networks (ANN)

Artificial neural networks (ANNs), or neural networks (NNs), are structures formed of several layers, each having a certain number of neurons. A simple NN is shown in Fig. 3.5. Each color represents a layer, and each circle represents a neuron. The first layer of an NN is the input layer, where a vector of input features is fed. The input then propagates through a certain number of hidden layers, the result of which is the final output layer. This layer consists of one or several neurons depending on the problem. NNs can be used for both classification and regression problems.

The propagation of data through the NN is a function of the network parameters, a set of weights $W = \{ w_1, w_2, ..., w_i, ..., w_{m-1}\}$ and biases $B = \{ b_1, b_2, ..., b_i, ... , b_{m-1} \}$, where m is the number of layers in the network. Each weight w_i is a matrix of dimensions $l \times k$ and each bias b_i represents a scalar, where k is the number of neurons in the layer $i-1$ and l the number of neurons in layer i.

The value of each neuron in layer i is a linear combination of the neurons of

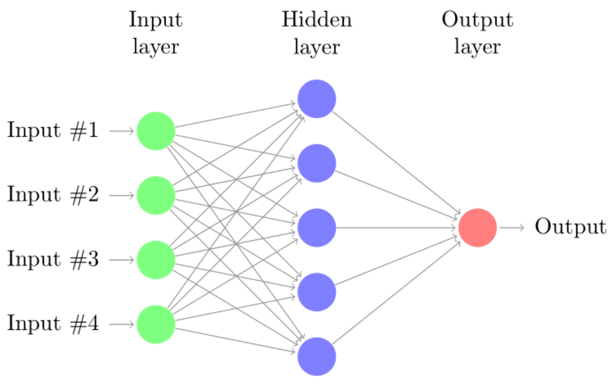


Figure 3.5: A simple neural network.

the previous layer followed by a non-linear activation function. This function can be a hyperbolic tangent, a rectifier (ReLu), or a sigmoid. The parameters used for the linear combination are the weights and biased and are the ones optimized. This relation is known as forward propagation and is illustrated in eq. (3.13).

$$a_i = \sigma(w_{i-1}a_{i-1} + b_{i-1}) \tag{3.13}$$

where a_i is known as the activation vector that represents the values of the neurons at layer i, σ is the activation function, and w_{i-1} and b_{i-1} are the weights and biases associated with layer

3.2.1 Neural Network Training

The supervised learning problem can be formulated as an unconstrained optimization problem. The input values in the case of OPF are the states of the system, such as the demand values and bus voltages while the output values are those corresponding to the control variables, such as the active and reactive power in the generating units. The objective function in Eq. (3.14) is a mean-squared error (MSE) function, where the average of the squared difference between the actual and estimated values is minimized. The predicted value is a continuous output from the neural network found in its output layer. Since layers of a neural network are related to each other through Eq. (3.13), this error propagates back through the layers. As a result, the parameters of the NN, namely the weights and biases, are tuned using a reverse-propagating mechanism known as back-propagation and discussed in the next section.

$$\min_{W,B} \frac{1}{N} \sum_{i=1}^{N} (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \tag{3.14}$$

where the $\hat{\mathbf{y}}$ and $\mathbf{y}$ are respectively the forecasted and actual values of output, n the number of samples in the dataset, and W and B are the sets of weights and biases respectively.

3.2.2 Back-propagation

As mentioned in the previous section, the error propagates back from the output to the input. The weights and biases in each layer are updated in a recursive manner through a method known as back-propagation. In our case, due to the convexity of the loss function, back-propagation may use a first-order method such as gradient descent (GD) to update the values of the NN parameters. GD aims to minimize the second-order Taylor series approximation of the loss function, which acts as an upper bound to the original function. The convergence of such a method is well-established. An alternative method would be to find the closed form solution, but computing such a solution is very expensive due to the large amount of data points.

If we let the loss function $\mathcal{L}(w_i, b_i)$ be a mean-squared error function, the gradient of $\mathcal{L}$ is defined as shown in Eq. (3.15).

$$\nabla \mathcal{L} = (\dots, \frac{\partial \mathcal{L}}{\partial w_i}, \dots, \frac{\partial \mathcal{L}}{\partial b_i}, \dots) \tag{3.15}$$

The partial derivative of the loss function with respect to the weight w_i linking layers i and $i+1$ is:

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial f}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{m-1}} \dots \frac{\partial a_{i+2}}{\partial a_{i+1}} \frac{\partial a_{i+1}}{\partial w_i} \tag{3.16}$$

where

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \tag{3.17}$$

$$z_m = w_{m-1} a_{m-1} + b_{m-1} \tag{3.18}$$

$$a_m = \sigma(z_m) \tag{3.19}$$

$$\frac{\partial \hat{y}}{\partial a_{m-1}} = \frac{\partial \hat{y}}{\partial z_m} \frac{\partial z_m}{\partial a_{m-1}} = \sigma'(z_m) w_{m-1} \tag{3.20}$$

$$\frac{\partial a_{i+2}}{\partial a_{i+1}} = \sigma'(z_{i+2}) w_{i+1} \tag{3.21}$$

$$\frac{\partial a_{i+1}}{\partial w_i} = \frac{\partial a_{i+1}}{\partial z_{i+1}} \frac{\partial z_{i+1}}{\partial w_i} = \sigma'(z_{i+1}) a_i \tag{3.22}$$

3.2.3 Iterative Descent Methods

After performing back-propagation, a gradient descent scheme is adopted as shown in Eq. (3.23). Due to the huge size of the dataset used, a mini-batch stochastic gradient descent (SGD) is used instead of the normal GD method. In each iteration of the training process, the mini-batch SGD evaluates the gradient vector only for a small subset of the larger dataset, chosen at random, drastically reducing the training complexity. An adapted mechanism to the SGD method, known as adaptive moment estimation (ADAM) is also explored in the testing chapter. The parameter update scheme is based on choosing a direction and a constant known as the learning, which determines the size of the step in that direction. The direction is that of the negative gradient of the loss function while the learning rate is a hyperparameter that is tuned during the training process. The pseudo-code for the descent method is shown in Algorithm 1.

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \mathcal{L} \tag{3.23}$$

where $\mathbf{w}$ is the vector of parameters and α is the learning rate. In other words, $\mathbf{w}$ is updated in the direction of the negative of the gradient $\nabla \mathcal{L}$ by a size of α.

Algorithm 1 Iterative Descent Scheme

```
for iterations do
    for batch size do
        randomly select a sample
        perform forward propagation using the sample
        perform back propagation using the sample
    end for
end for
```

3.2.4 NNs - Universal Function Approximators

A crucial advantage of NNs is that they are universal function approximators. The universal approximation theorem, shown in eq. 3.24, states that any function, linear or non-linear, can be approximated by a NN with enough neurons and layers. This enables NNs to solve complex problems that involve iterative updates more efficiently. In our work, we examine the NN performance in solving the OPF problem formulated earlier.

$$f(\mathbf{x}) \approx \sum_{j=1}^{N_L} v_j \phi(\mathbf{w_j x} + b_j) \tag{3.24}$$

where f is the function to be approximated by the neural network, $\mathbf{x}$ is the input vector, N_L is the number of layers in the network, $v_j \ in \ \mathbb{R}$ is a constant and $\mathbf{w_j}$ and b_j are respectively the weights and biases associated with layer j.

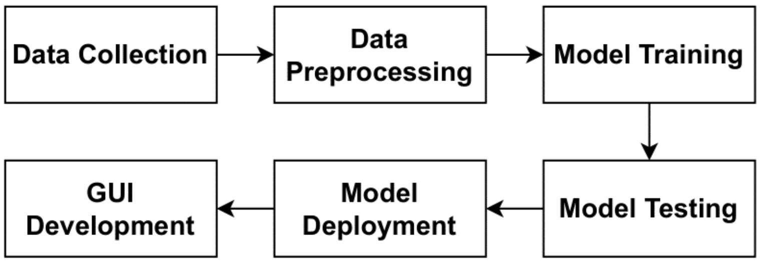


Figure 3.6: Proposed methodology.

3.3 Proposed Methodology

NNs belong to the family of supervised learning, as the NN model infers the relationships between the inputs and outputs of a certain problem. Since the OPF problem involves continuous values for its state and control variables, the NN model used in this project is a regression model.

The first step of any supervised learning process is to gather the data. A program is developed to generate data from a conventional and well-known OPF solver that uses value-iteration methods. After that, input and output data are normalized before being fed to the neural network model. A neural network architecture is built for training the pre-processed data. When the training is done, the performance of the model is evaluated on a new test set using a set of performance metrics. Finally, the developed OPF solver is deployed in a proper graphical user interface (GUI). This plan is shown in figure 3.6.

Since the full OPF problem, known as AC-OPF, is very complex and involves a lot of variables, as a first step, the above scheme is performed for the simpler, approximated DC-OPF problem. When successful, the cycle is repeated for the more complex AC-OPF.

3.4 Requirement Specifications

In alignment with the desired needs from the designed OPF solver, optimal results and fast computations, the fully trained machine learning model must satisfy the following requirements:

- The trained model should be able to solve the OPF problem in less than 1 second.
- The model must not violate any of the constraints of the problem.

- The increase in the total objective cost function from the traditional lengthy OPF solvers must not exceed 5%.

3.5 Deliverables

OPF is solved in energy control centers via special software programs [1]. The aim of this work is to design a similar program in a common programming language that would solve the OPF problem. At the heart of the program would be the trained model that should handle new instances of the OPF problem in real-time. The associated deliverables to this project are thus:

- Software implementation of a user-friendly OPF solver based on machine learning. The solver would be in the form of a pre-trained neural network model. An appropriate graphical user interface (GUI) is established for the model.

- Validation of the OPF performance under different operational scenarios. The performance of the agent will be verified for different power network systems and will be compared to that of traditional solvers in terms of optimality and convergence time.

- Confirmation of the designated IEEE standards.

Chapter 4

Technical/Non-Technical Constraints and Applicable Standards

4.1 Project Constraints

4.1.1 Technical Constraints

1. **Voltage Constraint**: Voltages are ranged bounded by minimum and maximum values. High voltages are determined by the capabilities of circuit breakers. Low voltages are determined by the operating requirements of the motors or generators.

2. **Thermal Constraint**: Temperature sensitivity of the conductor and supporting material in transmission line and elements should be based on a certain thermal transmission limit. This transmission limit is dependent on the material that the transmission line is created from.

3. **Current Constraint**: Current limitations are based on current diagnosis for equipment used. If current is set to increase, this may cause overheating in the sag region which may affect nearby equipment, thus forcing a current limit. This max limit is set by material science properties of conductors and transmission equipment. Studies of the stability of a system may also deduce this current limit.

4. **Voltage Angle Constraint**: Power flowing over an AC line is proportional to the sine of the voltage of the angle difference at the receiving and transmitting ends. Voltage angle difference must be set within boundaries due to stability reasons. For a lossless line, the theoretical steady-state

stability limit for power transfer between two buses is set to be 90 degrees. If this limit was exceeded, synchronous machines at either end would lose synchrony with each other.

5. **Power Constraint**: Power constraints over the generator include both output active and reactive power for each generator. The power of each generator must not exceed a certain limit and must not go under a specified limit as well depending on the generators utilized. Transmission lines also have an upper real and reactive power bound to avoid technical errors.

6. **Performance Constraint**: The program must be used in real time, which means that the trained model must be able to solve the OPF problem in about a minute.

4.1.2 Non-Technical Constraints

The list below summarizes some of the most common constraints that can be related to the FYP:

1. **Economic**: The generators' operating costs, or the total generation cost, should be minimized.

2. **Environmental**: It would be necessary to install multiple plants to meet large demands, which would result in more polluted fumes being emitted. When renewable energy systems are integrated, it can indicate how much fuel should be used. It allows those renewable systems to operate at zero cost, giving them the priority. As a result, polluting fumes would be reduced.

3. **Sustainability**: Renewable energy sources are undepletable sources and are significantly more sustainable than fossil fuels, which are not renewable. OPF facilitates the adoption and integration of renewable energy systems.

4. **Health and Safety**: Reducing polluting emissions naturally improves the health and safety of living creatures.

4.2 Applicable Standards

1. **IEEE Standard for Electric Power Systems Communications - Distributed Network Protocol** [47] relates to the use of low-cost distribution feeder devices which in our case may involve the use of cost-efficient generators and transmission lines.

2. **IEEE Standard for Exchanging Information Between Networks Implementing IEC 61850 and IEEE Std 1815** [48] relates to how the data is retrieved from the network and sent to the energy control center.

3. **Smart Grid Research: Power - IEEE Grid Vision 2050 Roadmap** [49] relates to the need to improve and develop the grid to satisfy the required energy technologies in the future.

Chapter 5

Progress Description

The project looks to design, implement and test a deep learning model to efficiently solve the OPF problem. Learning-based models can be grouped into supervised learning and reinforcement learning models. The first section explains the different design alternatives.

In order to reach the full OPF problem, project relies on starting with a much simpler representation of the problem and gradually adding elements to reach the desired full model. The model is first trained for a less complicated formulation of the problem with fewer constraints. After that, constraints are gradually added until the agent is exposed to the full OPF formulation. The second section describes this process.

5.1 Preliminary Design Alternatives

5.1.1 Deep Reinforcement Learning

In this section, an overview of reinforcement learning (RL) is first presented. After that, the two main RL alternatives, deep Q-learning and actor-critic methods, are touched.

Markov Decision Process Modeling

The agent in an RL model takes actions that are governed by a Markov Decision Process (MDP). An MDP is a mathematical formulation of a sequential decision making process [39]. The agent, or computer, learns the solution through solving an MDP. A very noticeable feature in MDPs is that current states in the process are independent of previous states. The MDP has four main components:

State $s_t \in S$: the observables that the agent encounters in the environment. In our case, the states are P_{D_i} and Q_{D_i} $\forall i \in N$, the active and reactive power demand at each bus. These values are randomly assigned at the beginning of every training episode and are left constant throughout.

Action $a_t \in A$: the action that the agent can take when in state S. It represents the control variable that the agent uses to maximize the reward at each time step. In our case, the actions the agent can take are P_{G_i} and Q_{G_i} $\forall i \in G$, the respective active and reactive power generated at each generating node.

State Transition Probability $P(s_{t+1} = s'|s_t = s, a_t = a)$: probability of the agent reaching state s' when taking the action a at state s. Whenever the agent decides to take an action a that would lead it to a new state s', due to the stochasticity in the environment, there exists a certain probability that the agent may reach another state using a different action. This stochasticity in real-time operation is accounted for by errors in load forecasts and is reflected in our model in the randomness of the assigned values for power demand.

Reward r_t: the information that the agent receives for every state-action transition, and which it tries to maximize. A proper representation for the reward in our case would be to appreciate more the lower the cost of the objective function while punishing the agent by a penalty value for every constraint violation. Hence, it can be formulated as follows:

$$r_t = \begin{cases} -5000 & PF \quad solver \quad diverges \\ R_{br,v} & LF \quad constraint \quad violation \\ wC_{generation} + z & normal \quad operation \end{cases} \tag{5.1}$$

where a large penalty is imposed when the power flow solver fails due to certain actions taken by the agent, and penalty $R_{br,v}$ is used when the branch limits are violated and is equal to the absolute difference of the violation. When the system is operating normally and no constraints are violated, the lower the cost of generation, the more reward the agent must receive. As a result, the convex objective function $C_{generation}$ is made concave through applying a linear transformation, where w is a small negative number and z is a positive number, as established in [50].

Associated with the reward is the discount factor γ that ensures that the reward decreases with time, encouraging the agent to reach the optimal policy as soon as possible. γ is generally a positive number ≤ 1 and decays with time. For simplicity, γ is assumed to be 1.

The relationships between these four components is illustrated in Fig. 5.1. The agent is set in a power network environment that runs the power flow solver and

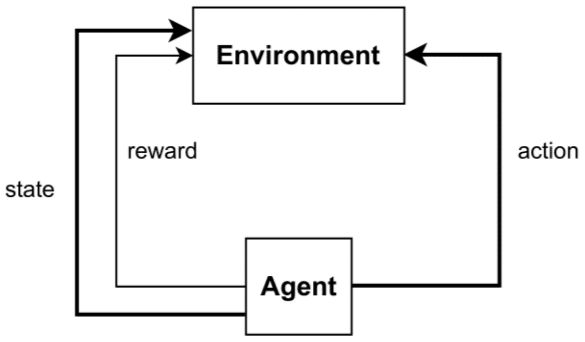


Figure 5.1: MDP components in RL

sends the agent active and reactive power data (states) and reacts by adjusting the power generated by each generator (action). The actions it takes are shaped by the resulting total cost they achieve and the amount of constraints they violate (reward).

Deep Q-learning

The agent in deep Q-learning learns the optimal policy through evaluating the actions it takes while interacting with the environment. The evaluation of each action is based on the action-value function, a discretized look-up matrix of dimensions, equal to the possible states in the environment and actions that the agent may take [51]. Although deep Q-learning has enjoyed success in applications related to the smart grid, its use in OPF is very limited due to the failure of the algorithm in continuous-action spaces. Therefore, this design alternative shall not be implemented.

Actor-Critic Models

Actor-crtic models combine actor, or policy-based, and critic, or value-based, models in one agent.

Actor Network: this network maps an action a to a state s. Since the actor network is a policy-based model, the transition from a state s to s' is dictated by a policy function $\pi_\theta(a|s)$ where θ is a vector of special parameters for the policy function. $\pi_\theta \sim \mathcal{N}(\mu_\theta, \Sigma_\theta)$, where μ_θ is obtained from the output of the actor NN, and Σ_θ is a diagonal matrix updated via back-propagation. As a result, when the agent is at state s, each action taken is assigned a certain probability value $\pi(s, a|\theta)$, and the agent may choose to follow the action with the highest probability. The training of the actor network is based on the gradient ascent update of the parameters θ and makes use of the expected reward values provided by the critic. The update scheme for the actor network is as follows:

$$\theta \leftarrow \theta + \alpha_\theta Q_w(s, w)\nabla_\theta log\pi_\theta(a|s) \tag{5.2}$$

where α_θ is the learning rate associated with the actor network and is a hyperparameter that must be tuned, $Q_w(s, w)$ is the expected reward of the value function estimated by the critic network, and $\nabla_\theta log\pi_\theta(a|s)$ is the normal update scheme in policy-based methods.

Critic Network: this network maps the current state s to a certain value. By this way, it assesses the quality of the current state. Critic networks use value-based methods and value functions, similar to Q-learning, to evaluate the states. The value function is also a deep network with its own parameter vector w, just as in deep Q-learning. The critic network makes use of the temporal difference function $\delta(t)$. This function is given by:

$$\delta(t) = r_t(s, a) + \gamma Q_w(s', a') - Q_w(s, a) \tag{5.3}$$

where $r_t(s, a)$ is the reward for taking action a from state s, $Q_w(s, a)$ is the expected cumulative reward starting from state s and action a, γ is the discount factor and $Q_w(s', a')$ is the expected cumulative reward one step later. In other words, the temporal difference measures the advantage of taking action a from step s as opposed to the average of taking all actions. The more the temporal difference, the more reward is taken from this particular action, and naturally, the more this action is favored. Therefore, the larger the temporal difference, the larger the update of the weights w at that time step. Hence, the critic network is updated according to the following scheme:

$$w \leftarrow w + \alpha_w \nabla_w Q_w(s, w) \tag{5.4}$$

where α_θ is the learning rate associated with the actor network and is a hyperparameter that must be tuned and $Q_w(s, w)$ is the expected reward of the value function.

Fig. 5.2 is a high-level depiction of an actor-critic RL model.

5.1.2 Supervised Learning

As mentioned in the previous chapter, supervised learning aims to learn the input-output relationships of the model from previous instances of the problem found in the dataset. The model achieves that through adjusting its parameters by minimizing a certain loss function. Supervised learning models may be divided into shallow models and deep models.

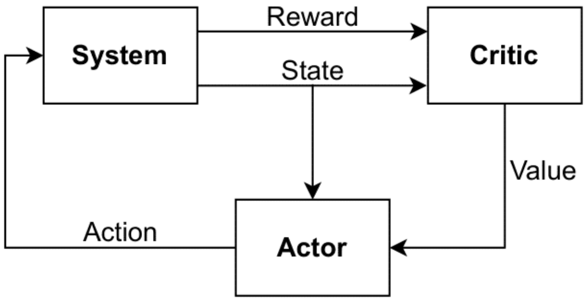


Figure 5.2: An A2C MDP.

Shallow Models

Shallow models include well-known regression models such as linear regression and support vector machines. While such methods have been utilized in the literature to solve variations of the OPF problem [52], their inability to learn complex and in some cases nonlinear relationships between data points has limited their use. Consequently, the study of shallow models for the OPF problem is omitted from this work.

Deep Models

Deep supervised learning models generally involve neural networks. As mentioned earlier, the neural network consists of several layers, the first of which is the input layer that takes in the set of inputs, and the last of which is the output layer that returns the vector of output layers. The neural network is trained by minimizing the mean squared error that captures the difference between the predicted and actual values for the output variables. The NN has a set of parameters called the weights and biases, which are updated every iteration of the training process via a gradient descent (GD) scheme or a variation of GD.

For a NN realization of the thermal resource dispatch version of the OPF problem, the input to the NN may be the current state of the power network that is represented by the active power P_D and reactive power Q_D demands at the load nodes. The input network is then forward propagated across the layers of the NN until the output vector is found in the final layer. In this case, the output vector includes the control variables of the OPF problem, namely the active power P_G and Q_G produced by the generating nodes. This is depicted in figure 5.3. After the control variables are found from the NN, the associated state variables such as the voltages V and phase angles θ are deduced from solving the load flow equations. In this manner, the NN does not have to deal with equality constraints, such as those related to power balance, which may cause problems for the NN.

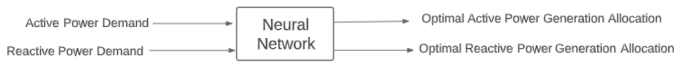

Active Power Demand
Reactive Power Demand
Neural Network
Optimal Active Power Generation Allocation
Optimal Reactive Power Generation Allocation

Figure 5.3: A NN-based OPF solver.

5.2 Preliminary Implementation and testing

Based on the analysis done above, the two different alternatives that shall be tried in this part are neural networks and deep reinforcement learning. After implementing both models, it was decided to proceed with the NN alternative.

As a first step, a linearized version of OPF known as the DC Optimal Power Flow (DC-OPF) is considered [5]. DC-OPF is a convex optimization problem that finds is usage in market clearing and transmission management [37]. The main assumptions that DC OPF take into consideration are:

- All the node voltages are assumed to be 1 p.u.
- Phase angles are assumed to be small enough to use small angle approximations
- Reactive power is neglected in this study

Consequently, the optimization problem for DC-OPF can be modeled as follows:

$$min \sum_{i \in G} (c_{2i} P_{gi}^2 + c_{1i} P_{gi} + c_{0i}) \tag{5.5}$$

subject to:

$$P_{G_i}^{min} \leq P_{G_i} \leq P_{G_i}^{max} \qquad \forall i \in G \tag{5.6}$$

$$|\frac{1}{x_{i,l}}(\theta_i - \theta_l)| \leq P_{flow_{i,l}}^{max} \qquad \forall (i,l) \in L \tag{5.7}$$

$$P_{G_i} - P_{D_i} = \sum_{j=1}^{N} B_{ij} \theta_j \qquad \forall i \in N \tag{5.8}$$

where Eq. 5.7 reflects the active power line flow limits and Eq. 5.8 represents the active power balance.

5.2.1 Design Alternative 1 - Deep Reinforcement Learning

The software chosen for the design of the RL agent is the MATLAB environment for two main reasons: (1) MATLAB has a very powerful power flow solver known as MATPOWER [53]. MATPOWER is an open-source simulation package that provides solvers for the power flow and optimal power flow problems [54] .

The deep reinforcement learning model is based on an actor-critic algorithm. The RL environment contains the states that the RL agent interacts with. In this case, the states are the power demands in the network The reward function is based on minimizing the cost function and contains penalty terms for any violated constraints. Each of the actor and the critic has its own NN. A couple of variations of the actor-critic algorithms have been used in this study, namely the deep deterministic policy gradient (DDPG) and the twin-delayed DDPG (TD3). These algorithms are readily available in the Reinforcement Learning toolbox of MATLAB.

The MATLAB RL model necessitates the presence of two files:

- An "Environment.m" file that defines a class object responsible for updating the states in the environment.
- A "trainAgent.m" file that designs the actor and critic NNs, selects the A2C algorithm and sets its hyperparameters, and trains the agent.

The environment is set in a power network formed of certain bus configuration. The actor-network decides on the set of power generation values $P_{G_i}\ \forall i \in G$ and sends the values to the environment, where MATPOWER performs power flow analysis and computes the cost of the system. The cost is used to find the reward and is fed into the critic network.

The environment file consists of 3 main methods:

- Environment Constructor
  1. set the observation information
  2. set the action information
  3. call the built-in function
- Step
  1. get the action performed by the agent
  2. create a new state by adopting previous demand state
  3. perform DC load flow on the network with the new value of states and adopted action
  4. update the reward
- Reset
  1. Randomly set the states at the start of each episode

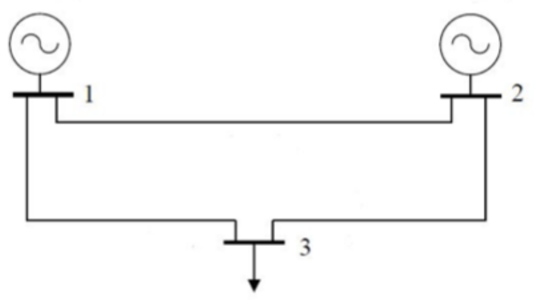


Figure 5.4: A simple 3-bus network

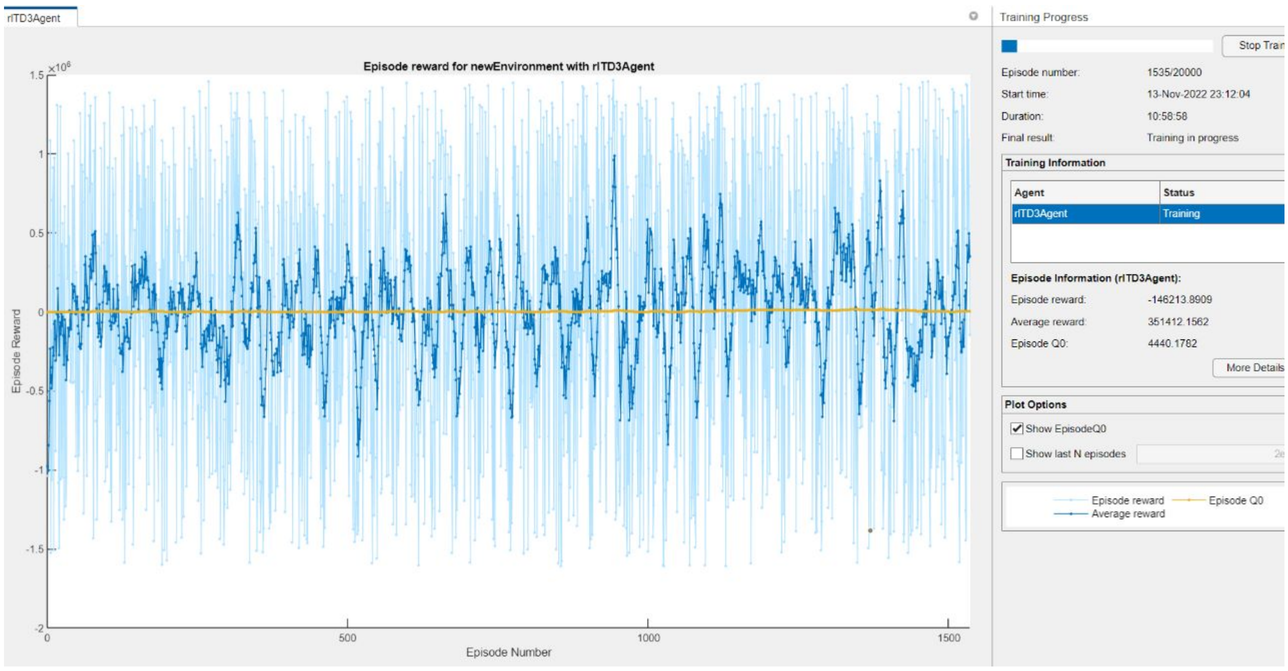
A simple 3-bus network shown in Fig 5.4 is considered at the beginning with 2 generators and 1 load. The agent has control over the power production of one generator while the other generator compensates the mismatch in power.

The training of such a simple network by an RL agent requires the manual tuning of several hyperparameters listed in table 5.1. Due to the large number of hyperparameters, finding the optimal set is a very complex procedure.

After building the RL environment and agent in MATLAB and conducting some attempts of hyperparameter tuning, the training procedure is depicted in figure 5.5. The training was carried out using a 16GB RAM computer with processor Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz. It can be seen that the reward is relatively stable, even after over 1,500 episodes of the training process. This means that the agent is failing to learn the optimal policy with the current set of hyperparameters. Consequently, the set of parameters requires several more training iterations that could last weeks or even months. Due to the time frame of the current project, such a process is deemed infeasible and a different design alternative is explored.

Table 5.1: RL model hyperparameters.

Parameter	related to
Number of layers	Actor NN
Number of neurons per layer	Actor NN
Activation function	Actor NN
Learning rate	Actor NN
Gradient Threshold	Actor NN
Number of layers	Critic NN
Number of neurons per layer	Critic NN
Activation function	Critic NN
Learning rate	Critic NN
Gradient Threshold	Critic NN
Sample Time	Agent Options
Batch Size	Agent Options
Number of Episodes	Training Options
Number of Steps per Episode	Training Options
Score Averaging Window Length	Training Options



Training Progress

Episode number:	1535/20000
Start time:	13-Nov-2022 23:12:04
Duration:	10:58:58
Final result:	Training in progress

Training Information

Agent	Status
rlTD3Agent	Training

Episode Information (rlTD3Agent):

Episode reward:	-146213.8909
Average reward:	351412.1562
Episode Q0:	4440.1782

Plot Options

[✓] Show EpisodeQ0

☐ Show last N episodes

Figure 5.5: RL model training.

5.2.2 Design Alternative 2 - Neural Networks

As illustrated in the previous part of the chapter, the neural network model is fed with input values of power actuve and reactive demand and outputs the control variables for the OPF problem, namely the power generated by the generating units. For the case of DC-OPF, the input vector consists of the active power demand values at the load nodes, and the output vector includes the the active power generated at the generator nodes. After the model finds the values of active power generation, the phase angle values are found from solving the load flow problem, namely, eq. (5.8).

Choice of Software

Since neural networks require a dataset of input-output relationships, the best way to generate this set is by solving the DC-OPF problem a certain number of times using traditional solvers. An important remark is that the traditional solvers are only used to generate the dataset that would enable the neural network model to learn. Once that is done, these solvers are no longer used and are replaced by the neural network model, which in the case of DC-OPF should show better results in terms of convergence time.

MATPOWER has been widely used by the power systems community to solve

Table 5.2: Choice of software tools.

Stage	Software	Library/Toolbox
Data Generation	MATLAB	MATPOWER 7.1
Data Pre-processing	Google Colab (Python)	NumPy
Neural Network Construction	Google Colab (Python)	Keras

the OPF problem. One of its major advantages is the clear documentation of its different functions and data structures. As a result, MATPOWER was chosen to generate the dataset, the process by which shall be explained in the next section.

After the dataset has been generated, the neural network model can be built using a certain library. Most programming languages contain special libraries or toolboxes for machine learning and deep learning applications. The aim is to find the programming language and library that offer the best computational power, interpretability of the code, and the collaboration of the different members of the group.

Most of the machine learning and data science community use Python as their preferred programming language. Python offers very useful libraries for machine learning applications such as TenserFlow, Keras and PyTorch. In addition, Google offers a free platform, Google Colab, that enables users to write Python code on their browser. Colab uses the Google servers to run the Python code, which drastically improves the computational power. In addition, the code can easily be shared via Google Drive, and multiple participants can work on it simultaneously. Hence, Google Colab is chosen to be used as the preferred platform, where the neural network will be built using the well-known neural network library, Keras. To enter data into a Keras NN model, a pre-processing stage must be done using the NumPy library of Python.

The conclusions regarding the choice of software tools are summarized in table 5.2

Power Network

The IEEE9 system, shown in fig. 5.6 is put under study in this section. The network consists of 9 buses, 3 generation nodes, 3 load nodes and 9 transmission lines. The default values for active power generation, demand and flow are found in tables 5.3, 5.4 and 5.5 respectively. These default values are obtained from solving the DC Power Flow problem.

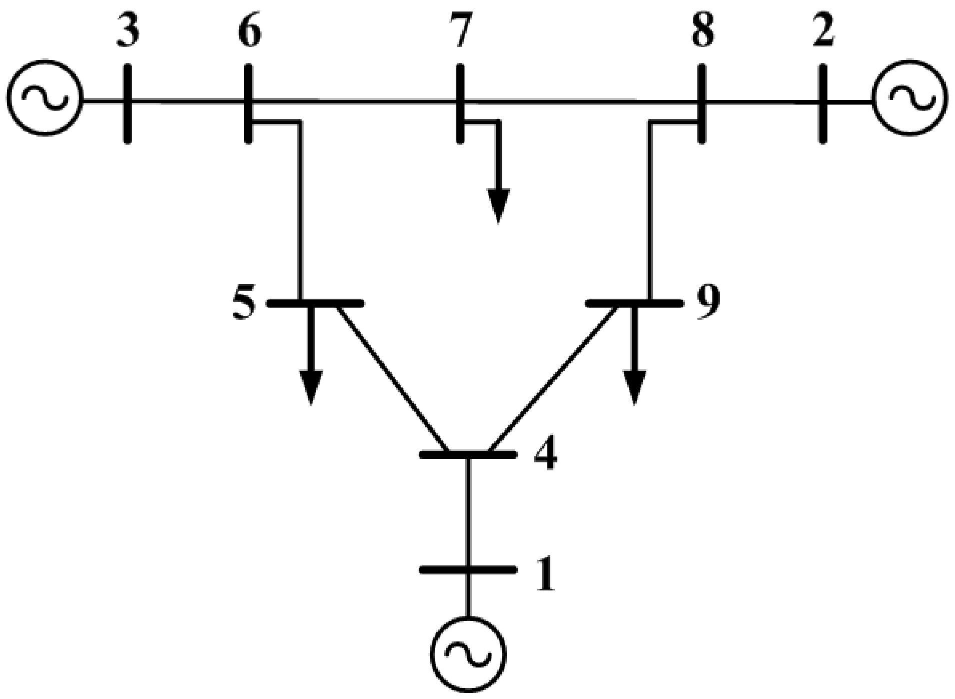


Figure 5.6: IEEE 9-busbar system.

Table 5.3: Default active power at generation nodes.

Node	Active Power (MW)	Capacity (MW)
1	67	250
2	163	300
3	85	270

Table 5.4: Default active power demand at load nodes.

Node	Active Power (MW)
5	90
7	100
9	125

Table 5.5: Default active power at generation nodes.

Branch	Active Power Flow (MW)	Capacity (MW)	Reactance (p.u.)
1-4	67	250	0.0576
4-5	29	250	0.092
5-6	-61	150	0.17
3-6	85	300	0.0586
6-7	24	150	0.1008
7-8	-76	250	0.072
8-2	-163	250	0.0625
8-9	87	250	0.161
9-4	-38	250	0.085

Data Acquisition

As mentioned in the previous parts, the dataset is generated by running the DC-OPF problem on the IEEE 9-busbar system for a certain specified number of samples. In each of these instances, the value of the active power demand at the load nodes is varied randomly within a certain range δ from its default value found in table 5.4. δ is a percentage value that represents the deviation from the default value of demand. For example, the default value for the demand at node 5 is 90 MW. Consequently, in each sample in the data set, the demand at node 5 will be $P_{D_5} = x$ where x is a random uniform sample from the range $[90 - 90\delta, 90 + 90\delta]$. A typical value for δ is 10%.

An algorithm has been created in MATLAB to generate the dataset. The algorithm must receive, as input, the power network (eg: IEEE9), the number of training samples, N, needed, and the allowable variation δ. The algorithm first saves the default values for the demand nodes, which are found in table 5.4. After

Table 5.6: The input parameters to the data acquisition algorithm.

Parameter	Value
Network	IEEE9
Samples N	50,000
δ	10%

that, for every training sample, the program generates a new demand vector $\widetilde{P_D}$ formed from varying each element of the default vector by a unique random variable generated uniformly from the range $[P_{D_i} - \delta P_{D_i}, P_D + \delta P_{D_i}]$. After receiving the new demand vector, the program runs the DC-OPF function provided by MATPOWER, which outputs the optimal values for generation $\widetilde{P_D}$ corresponding to the specific case $\widetilde{P_G}$. The samples are saved in a matrix that is converted to a comma separated values ".csv" file, or another file extension depending on the user's preference. The pseudocode can be found in Algorithm 2.

Algorithm 2 Data Generation Scheme

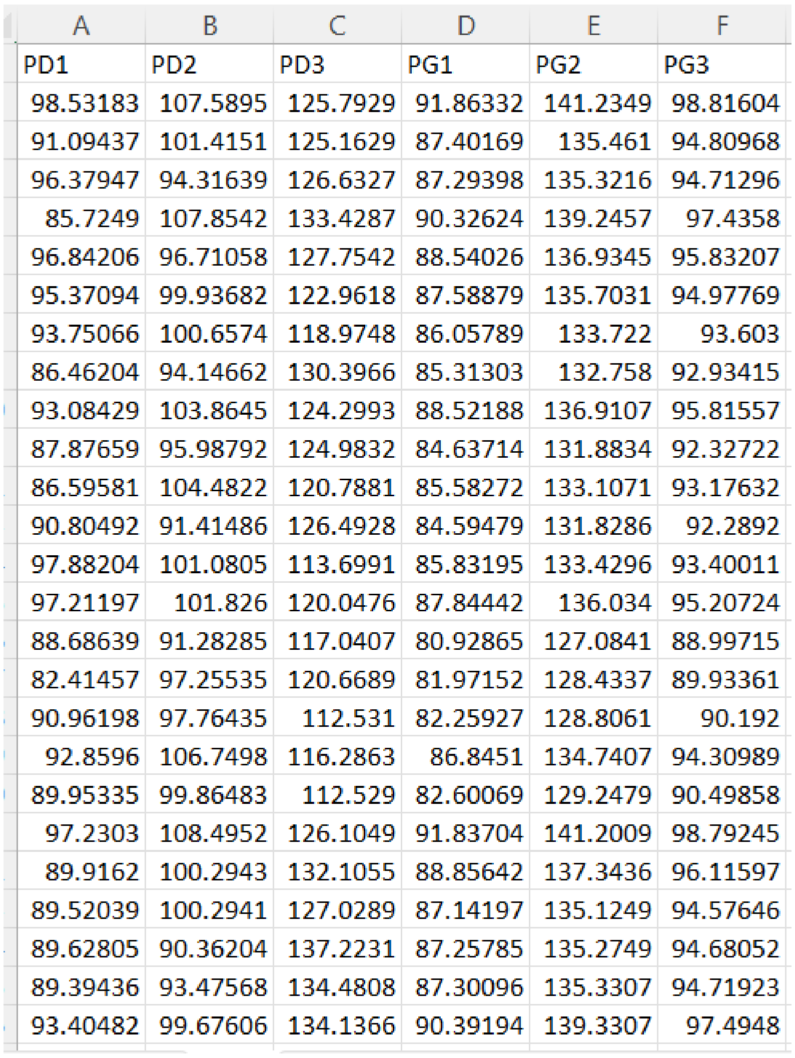
```
Specify case file, $N$ and $\delta$
$P_D \leftarrow$ default demand values
**for** $N$ **do**
    **for** $i \in$ demand nodes **do**
        $\widetilde{P_{D_i}} \sim U(P_{D_i} - \delta P_{D_i}, P_D + \delta P_{D_i})$
    **end for**
    $\widetilde{P_G} \leftarrow$ run (DC-OPF | $\widetilde{P_D}$)
    Save $\widetilde{P_D}, \widetilde{P_G}$
**end for**
```

The algorithm has been run with the input parameters found in table 5.6.

Since the network has 3 load nodes and 3 generation nodes, the dataset will contain 6 columns corresponding to these 6 nodes and will have N nodes, where N is the number of samples. A snapshot of the first set of elements of the dataset can be found in figure 5.7.

Data Pre-processing

Once the dataset is generated, some data pre-processing must be done on it using the Python library NumPy. The essential purpose of the pre-processing stage is to normalize the data points in order to enable the neural network to learn properly. Normalization is an important, and sometimes necessary, process that eliminates any influence of a certain factor on the data output merely due to its



A	B	C	D	E	F
PD1	PD2	PD3	PG1	PG2	PG3
98.53183	107.5895	125.7929	91.86332	141.2349	98.81604
91.09437	101.4151	125.1629	87.40169	135.461	94.80968
96.37947	94.31639	126.6327	87.29398	135.3216	94.71296
85.7249	107.8542	133.4287	90.32624	139.2457	97.4358
96.84206	96.71058	127.7542	88.54026	136.9345	95.83207
95.37094	99.93682	122.9618	87.58879	135.7031	94.97769
93.75066	100.6574	118.9748	86.05789	133.722	93.603
86.46204	94.14662	130.3966	85.31303	132.758	92.93415
93.08429	103.8645	124.2993	88.52188	136.9107	95.81557
87.87659	95.98792	124.9832	84.63714	131.8834	92.32722
86.59581	104.4822	120.7881	85.58272	133.1071	93.17632
90.80492	91.41486	126.4928	84.59479	131.8286	92.2892
97.88204	101.0805	113.6991	85.83195	133.4296	93.40011
97.21197	101.826	120.0476	87.84442	136.034	95.20724
88.68639	91.28285	117.0407	80.92865	127.0841	88.99715
82.41457	97.25535	120.6689	81.97152	128.4337	89.93361
90.96198	97.76435	112.531	82.25927	128.8061	90.192
92.8596	106.7498	116.2863	86.8451	134.7407	94.30989
89.95335	99.86483	112.529	82.60069	129.2479	90.49858
97.2303	108.4952	126.1049	91.83704	141.2009	98.79245
89.9162	100.2943	132.1055	88.85642	137.3436	96.11597
89.52039	100.2941	127.0289	87.14197	135.1249	94.57646
89.62805	90.36204	137.2231	87.25785	135.2749	94.68052
89.39436	93.47568	134.4808	87.30096	135.3307	94.71923
93.40482	99.67606	134.1366	90.39194	139.3307	97.4948

Figure 5.7: A portion of the generated dataset.

order of magnitude.

A standard normalization scheme, that subtracts the mean and divides by the standard deviation, is applied on the input vector of power demand. For each of the three columns of power demand, the mean μ_n and standard deviation σ_n of the corresponding 50,000 samples are found. After that, each entry in the column is updated according to eq. (5.9).

$$P_{D_{n_i}} \leftarrow \frac{P_{D_{n_i}} - \mu_n}{\sigma_n} \qquad \forall i \in N \qquad \forall n \in S_L \tag{5.9}$$

where $n \in S_L = \{5, 7, 9\}$ is the load node, μ_n and σ_n are respectively the mean and standard deviation of the samples at node n, and $P_{D_{n_i}}$ is the active power demand at node n and in sample i. The process is repeated for all N samples.

Another normalization scheme is applied for the output vector of generation values. Since the generation values obey the generation limit constraints outlined in (3.3), the range is transformed from $[P_{G_{min}}, P_{G_{max}}]$ to $[0, 1]$ using eq. (5.10) for better results.

$$P_{G_{n_i}} \leftarrow \frac{P_{G_{n_i}} - P_{G_{minn}}}{P_{G_{maxn}} - P_{G_{minn}}} \qquad \forall i \in N \qquad \forall n \in G \tag{5.10}$$

where G is the set of generation nodes and N the number of samples.

Neural Network Architecture

The neural network is built in the Keras library and designed to have 1 input layer, 3 hidden layers and 1 output layer. The number of neurons in the input and output layers are equal to the size of the input and output vectors respectively. For the case of the IEEE9 network, the number of neurons is 3 in each case. The number of neurons in each of the three hidden layers is chosen by design to be 32, 16 and 8. The different stages of the neural network are found in fig. 5.8, where a dense layer is a simple feed-forward layer discussed earlier. A more thorough visualization of the network architecture is found in fig. 5.9.

The activation function is chosen to be the rectifier ReLu function, shown in fig. 5.10. The function basically clips any negative values. The ReLu activation function is widely used in regression problems, as it eliminates problems of exploding gradient. The activation function mapping the penultimate layer to the final layer is, however, a simple linear function for reasons explained in the next part of the section.

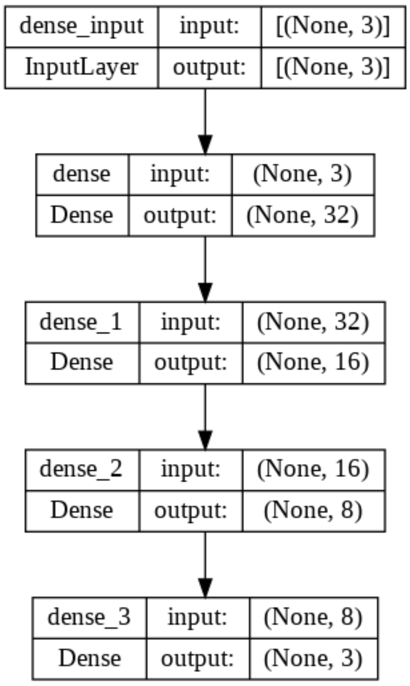


Figure 5.8: The layers of the neural network.

The neural network training aims to minimize a certain loss function by adjusting its parameters, namely the weights and biases, through an iterative descent scheme. Weights and biases are generally initialized randomly. A poor initialization, however, may lead in some cases to divergence. Neural network parameter initialization is an active area of research, and one of the most notable schemes developed so far is the He initialization [55]. The weights and biases are sampled from a uniform distribution described in (5.11).

$$x_l \sim U(-\sqrt{\frac{6}{n_l}}, \sqrt{\frac{6}{n_l}}) \qquad \forall x_l \in \{W_l, b_l\} \qquad \forall l \in L \tag{5.11}$$

where x_l is a NN parameter, weight or bias, that relates layer l to layer $l+1$, L the set of layers, W_l and b_l the set of weights and biases of layer L and n_l the number of neurons in layer l.

Neural Network Training

The generated dataset will be partly used for training the neural network. The other part will only be used for testing the NN after the training has completed. The set containing 50,000 samples is split into 70% data points for training and 30% for testing. This section will focus on the training aspect while the following section will discuss the testing.

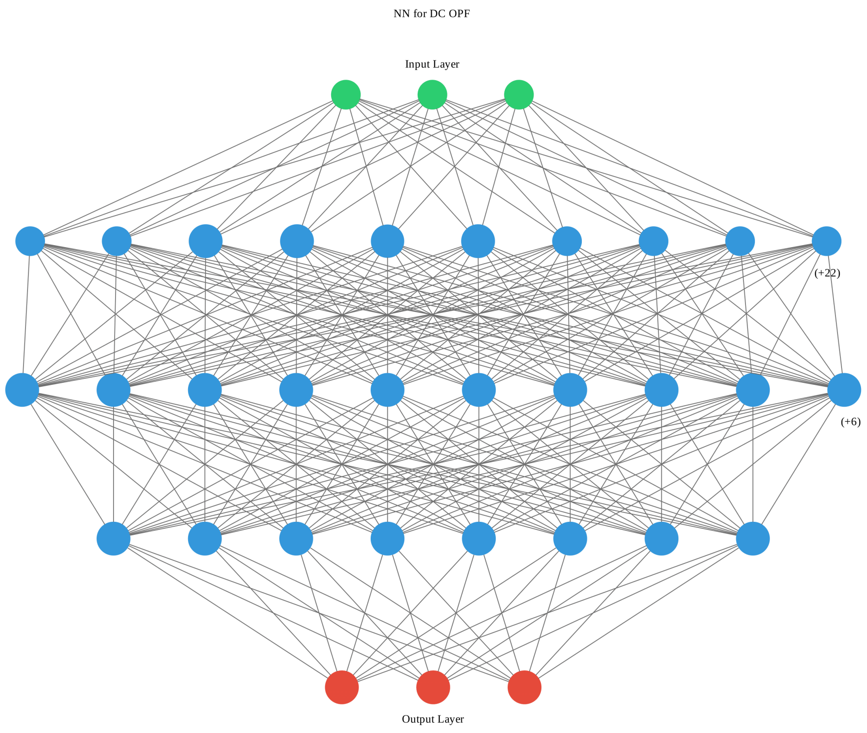


Figure 5.9: The neural network architecture.

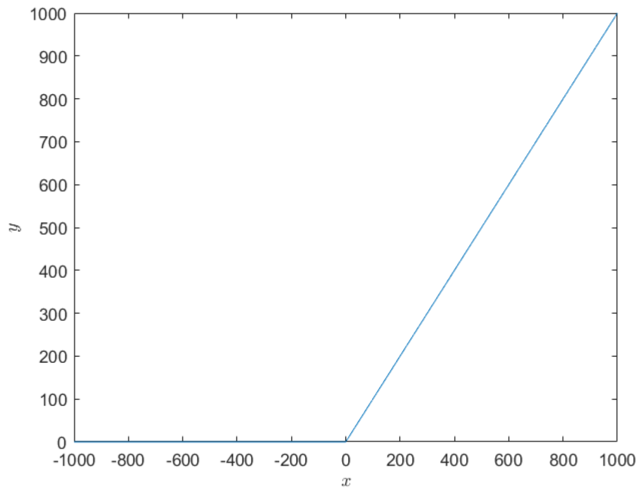


Figure 5.10: The ReLu activation function.

The loss function that the neural network aims to minimize is the mean-squared error, which is the most used loss function in regression settings. The objective function, shown in eq. (5.12), tries to minimize the average squared difference between the actual value of power generation found in the dataset and the value estimated by the neural network.

$$min_{W,b} \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|G|} \sum_{j=1}^{|G|} (\widehat{P_{Gij}} - P_{G_{ij}})^2 \tag{5.12}$$

where W and b represent the weights and biases of the neural network, G is the set of generator node, N the number of training samples, $P_{G_{ij}}$ the actual optimal value for the generation at node j and sample i and $\widehat{P_{Gij}}$ the estimated value for that generation using the neural network.

In order to avoid overfitting, which is the case when a machine learning model reaches a point where it performs very well on the training set but when issued for testing performs badly, a validation set is also inserted in the training. Overfitting is when the model has very high variance due to the over-complicated model complexity. During training, the neural network uses the training set to update its parameters and compares the loss function for the training set at every iteration with that of the validation set of points. If the loss function of the training set is lower than that of the validation set, it generally means that the model is learning features that are very specific to the set of points present in the training set and does not capture the general trend. It also means that the model has a high variance. Therefore, a validation set makes sure that the model does not reach the overfitting stage.

For that purpose, the training set, which in itself is 70% of the entire dataset, is partitioned into 80% actual training set and 20% validation set. It is important to note that the validation set is different from the test set. The former is used during the training phase and contributes to the update in the parameters of the neural network, while the latter is only used to evaluate the performance of the model after the training is done, and when the update of the NN parameters is no longer possible.

In order to find the optimal parameters W and b of the NN, a gradient descent method is adopted as described in a previous chapter. More precisely, an adapted stochastic gradient descent scheme, known as ADAM, is used. The key parameters for an iterative descent scheme, as outlined in Algorithm 1 and eq. (3.23) are the number of iterations, or epochs, the batch size and the constant step-size, or learning rate α. The chosen values for this neural network are shown in table 5.7.

Table 5.7: The neural network hyperparameters.

Parameter	Value
No. iterations (Epochs)	300
Batch Size	64
Learning rate	10^{-3}

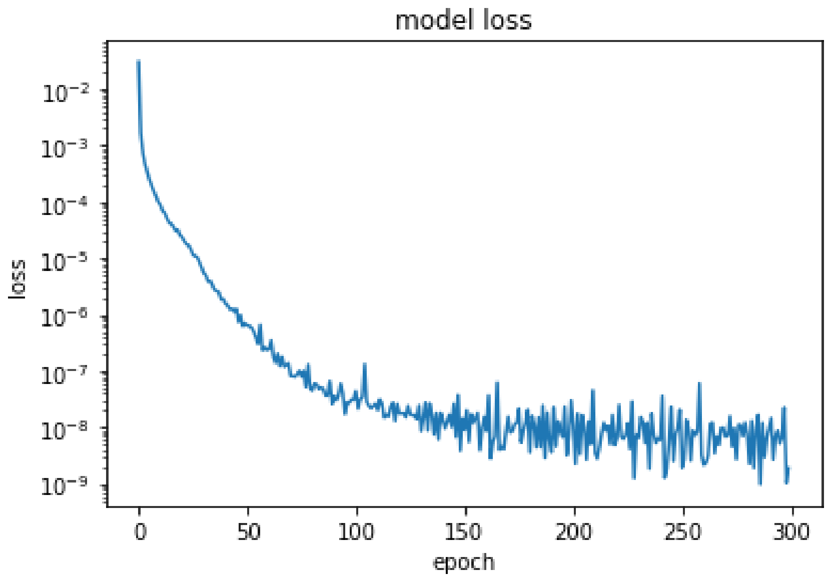


Figure 5.11: The NN training loss on a log-scale.

The training was run on the cloud servers provided by Google Colab. Fig. 5.11 shows the loss function as function of the epoch number on a log-scale plot. The plot shows the successful learning of the neural network model and the convergence to a neighborhood of the order 10^{-8}. The small fluctuations are due to the nature of the stochastic gradient descent. Although the results are very satisfactory, a diminishing step-size scheme may be explored to reach full convergence.
It is worth mentioning that two previous design iterations were conducted:

- **Iteration 1:** The input data was not normalized. In this case, the model failed to learn and performed like a dummy model. A dummy model is a model that predicts a certain constant value for the output regardless of the input.

- **Iteration 2:** The final activation function was chosen to be a ReLu function, as all previous functions. In this case, the neural network failed to

Table 5.8: The neural network performance metrics on the test set.

Metric	Score
MSE	8.5×10^{-9}
MAPE	0.02%

converge $(1/5)^{th}$ of the time. The ReLu was then replaced by a linear function that has no effect (the output of the function is equal to its input).

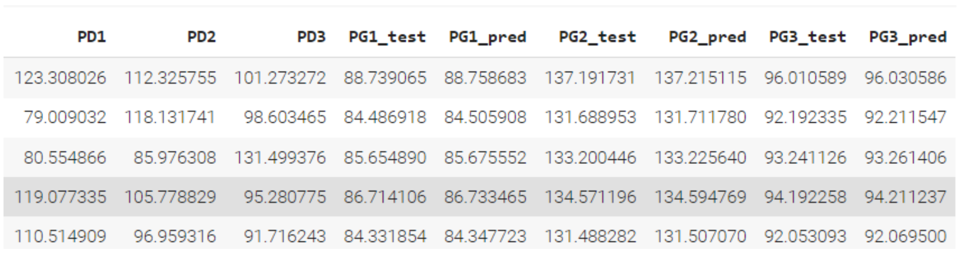
Neural Network Testing

After the training phase is done, the fully-trained neural network is tested with new instances present in the test set that it has not seen before. The testing set forms 30% of the original dataset. The model is only fed with the input vector containing the active power demand values at the three nodes. For each data point of the test set, the neural network will estimate the output vector. After looping over all the test points, certain evaluation criteria are computed for the NN performance:

1. **Mean-squared error (MSE)**: MSE is used for two reasons. The first one is to compare it for the test set with that of the final iteration in the training set. A larger value for the latter indicates overfitting. The second reason is to evaluate the performance of the model. Although there is no benchmark value for MSE, a value well below that obtained by a dummy model (which is 0.75 in this case) indicates a positive result.
2. **Mean absolute percentage error (MAPE)**: MAPE computes the deviation of the results from the actual values in a more interpretable sense. A percentage error above 0.5% may not be tolerable.

The values for the performance metrics that the trained neural network returned on the test set are found in table 5.8. It can be seen that the MSE on the test set is not greater than that on the training set indicating no overfitting. A MAPE of 0.02% is also a very satisfactory score, especially that NNs are meant to be approximators and actually achieve these results in a smaller time frame than traditional solvers.

Fig. 5.12 shows a snapshot of a portion of the test set inputs (first 3 columns) and the predicted and actual values for demand in the last 6 columns. It can be seen that the estimated values are the same as the actual values obtained from MATPOWER to the first decimal place.



PD1	PD2	PD3	PG1_test	PG1_pred	PG2_test	PG2_pred	PG3_test	PG3_pred
123.308026	112.325755	101.273272	88.739065	88.758683	137.191731	137.215115	96.010589	96.030586
79.009032	118.131741	98.603465	84.486918	84.505908	131.688953	131.711780	92.192335	92.211547
80.554866	85.976308	131.499376	85.654890	85.675552	133.200446	133.225640	93.241126	93.261406
119.077335	105.778829	95.280775	86.714106	86.733465	134.571196	134.594769	94.192258	94.211237
110.514909	96.959316	91.716243	84.331854	84.347723	131.488282	131.507070	92.053093	92.069500

Figure 5.12: The performance of the NN on the test set.

Table 5.9: The neural network performance on a new instance.

Generation Node	NN (MW)	MATPOWER (MW)
1	81.69	81.87
2	128.33	128.30
3	89.76	89.84

Performance on a new data instance

To further validate the performance of the NN model, a new input vector consisting of three elements, corresponding to the three demand nodes, is entered. The neural network is fed with this input vector and returns an output vector of three elements corresponding to the three generation nodes. The input is chosen to be [85, 95, 120] (MW). The output results based on the NN and MATPOWER are shown in table 5.9. We can conclude the model is successful in finding optimal solutions for new instances as well.

More binding constraints

In this section, the line flow limits found in table 5.5 are reduced by a factor of 2.5 to obtain more binding constraints. This is performed to check whether the neural network is able to respond to line flow limits. Fig. 5.13 shows a similar training plot to the one obtained in the previous section indicating good convergence. Table 5.10 shows the performance metrics on the test set. MSE and MAPE scores are similar to the previous case and are very satisfactory. A new instance is also tested for the same vector [85, 95, 120] (MW), and the results are shown in table 5.11. Hence, the neural network architecture is able to adapt to more stringent line flow limits.

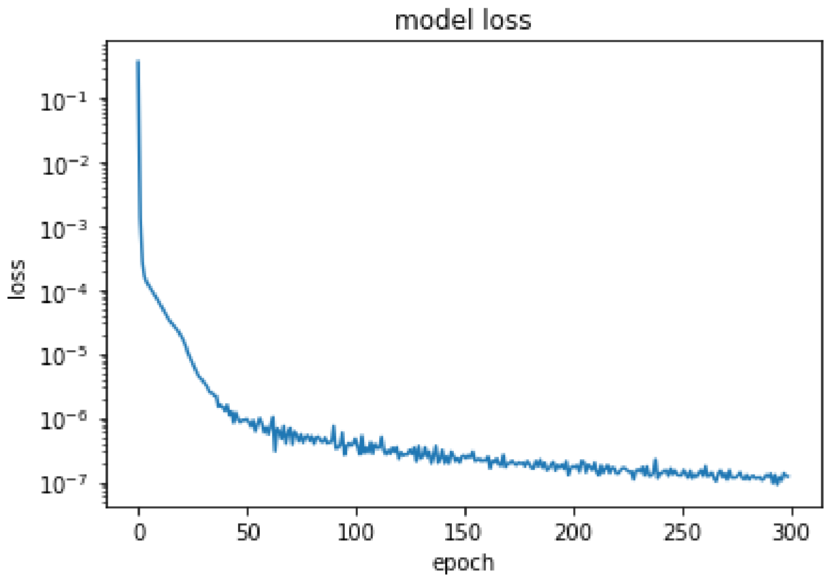


Figure 5.13: The NN training loss on a log-scale for the new case.

Table 5.10: The neural network performance metrics on the new test set.

Metric	Score
MSE	6×10^{-6}
MAPE	0.06%

Table 5.11: The neural network performance in the new case on a new instance.

Generation Node	NN (MW)	MATPOWER (MW)
1	96.66	96.77
2	99.97	100.00
3	103.12	103.23

Improvements to the model

Although the NN model yielded satisfactory results even in the case of more strict constraints, theoretically, neural networks do not have convergence guarantees. The neural network solution may not lie within the feasible region. For that purpose, a projection scheme shall be constructed to project back the solution obtained from the NN into the feasible region. By considering the L-1 projection (absolute value), the projection can be done by solving the following linear program:

$$min_U \|\hat{P_G} - U\|_1 \tag{5.13}$$

s.t. U satisfies constraints (5.6) - (5.8)

It should be noted that the absolute value objective function can be linearized by adding the constraint (5.14) and transforming the objective function into (5.15).

$$-K \leq \hat{P_G} - U \leq K \tag{5.14}$$

$$min_U \, K \tag{5.15}$$

The projection scheme is able to retrieve a feasible solution from the original neural network solution. However, the neural network still does not distinguish feasible and non-feasible solutions. In order to solve that, the a penalty term, shown in (5.16) is added to the loss function of the neural network model, found in (5.12). This term detects every line flow violation case and adds a penalty to the loss function proportional to this violation. In this manner, we are integrating properties of the power system, and not only data points, into the neural network model. This field is known as physics-informed machine learning and enjoys very active research.

$$L_{pen} = \frac{1}{N_a} \sum_{k=1}^{N_a} max(P_k - P_{k_{max}}, 0) \tag{5.16}$$

where N_a is the number of branches in the network, P_k is the active power flow in branch k, and $P_{k_{max}}$ is the maximum capacity of branch k.

Hence, the neural network model seems to be the most successful candidate to solve the OPF problem, and as such, it is the design alternative that shall be explored in the upcoming semester to solve the more general AC-OPF problem.

Chapter 6

Conclusion

Optimal power flow is an essential part of determining the cost of operation of a power system, and even though it was formulated for around 50 years, it still lacks a quick solving methodology. Recent developments in the power grid has led to the urgent need for accurate solvers that can include accurate forecast of fluctuations in the network and can respond rapidly to those events.

Recently, there have been several techniques involved in solving OPF, which can be grouped into deterministic, probabilistic, and learning-based methods. Deterministic methods have been observed to be very slow in terms of convergence and limited by the load demand uncertainty and the emergence of renewable energy systems. Probabilistic methods have been concluded to have high computational load as well as problematic in determining the probability distribution of unknown variables. Thus, when such conditions are faced, it was found to be a lengthy process rendering them useless in real-time challenges. Learning-based methods that include of deep learning (DL) and reinforcement learning (RL) can produce very fast results on new instances of data. It has been shown, however, that the latter costly in terms of the training time needed to reach a pre-trained successful agent.

Thus, this project has been focused on the last resort being DL, or neural networks (NN), that solve the problems mentioned earlier in the other methods. Despite the fact that a NN requires a huge dataset which is difficult to attain, a method of generating this dataset using OPF solvers is used rather than a hard-to-find real dataset.

After formulating the OPF problem as a large-scale non-linear and non-convex optimization problem, multiple specifications have been applied, some of which emphasize the need to solve this OPF problem in a very short period of time while not violating the constraints. Several design alternatives have been proposed including deep reinforcement learning (DRL) and simple feed-forward NN.

Initial implementation included the two design alternatives: DRL and NN. For DRL, implementation in MATLAB has been done using MATPOWER and the RL and DL toolboxes. DC Optimal Power Flow was first considered to be able to start simple and build up gradually. The RL model proved to be infeasible in the context of this project, as it required extensive training beyond the time frame of this work.

Consequently, NNs were then explored. Data generation was done using MATLAB from solving the DC-OPF problem in MATPOWER; data pre-processing was done using Python (NumPy), and the NN construction was done using Python (Keras). The neural network yielded very satisfactory results in terms of training, testing and when faced with more strict constraints.

Work in the near future will focus on developing a projection scheme that will theoretically guarantee a feasible solution. In addition, the loss function of the NN will be modified to include a penalty term for any line flow constraint violation.

The long-term goal of the work is to generalize the solution to the more realistic AC-OPF formulation of the problem and to develop a graphical user interface (GUI) for the NN solver.

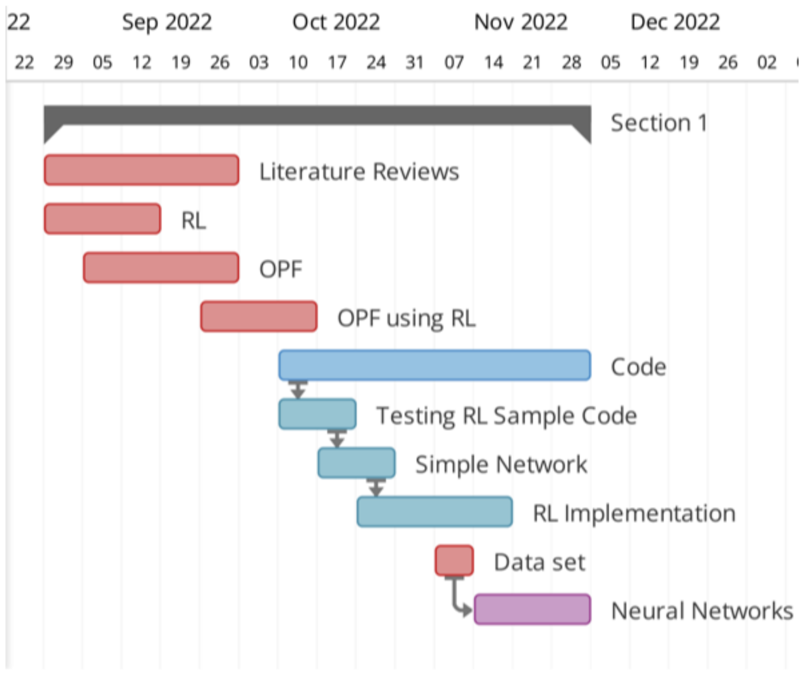
Appendix A

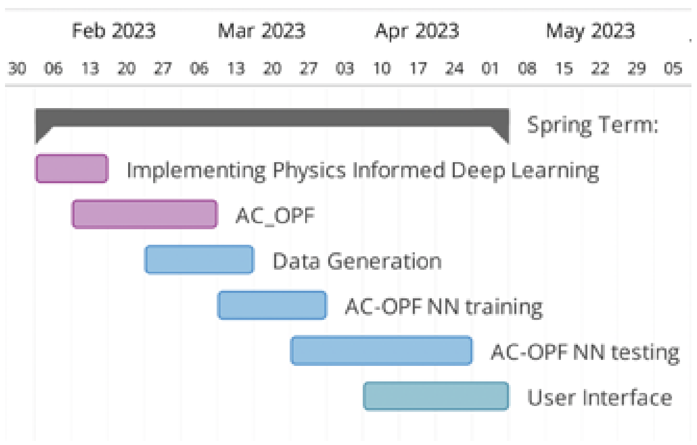
List of Resources and Engineering Tools Needed

1. **MATLAB**:the main software, we use it to write the codes and run our simulations.
2. **Reinforcement Learning Toolbox**: a MATLAB toolbox used to train an RL agent.
3. **Deep Learning Toolbox**: a MATLAB toolbox used to
4. **MATPOWER**: we use it to build our power network and as an input data to our code. We had to learn how to use it, thus we watch videos and read papers about it.
5. **IEEE Xplore**: a library dedicated to the world's highest quality technical literature in Optimal Power Flow and machine learning.
6. **Python**: the main programming language used for neural network training, testing and evaluation.
7. **Google Colab**: an integrated development environment (IDE) that runs the python code on Google servers, which has much more computational power.
8. **Pandas**: A Python library used for manipulating large datasets.
9. **NumPy**: A Python library for numerical computations for different data structures.
10. **Keras**: A Python library for implementing neural networks.
11. **Sci-kit Learn**: A Python library for evaluating the performance criteria used in neural network testing .

Appendix B

Detailed Project Schedule




Feb 2023
Mar 2023
Apr 2023
May 2023
30 06 13 20 27 06 13 20 27 03 10 17 24 01 08 15 22 29 05
Spring Term:
Implementing Physics Informed Deep Learning
AC_OPF
Data Generation
AC-OPF NN training
AC-OPF NN testing
User Interface

Appendix C

Weekly Meeting Minutes

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
3	9/14/22	10 a	30 mins	[✓ Advisor] Students	Mohammad Fares El Hajj Chehade

Attendees:

Mohammad Fares El Hajj Chehade - Mohammad Al Tawil - Karim Khalife - Rabih Jabr

Briefly summarize the main discussions during the meeting:

- discussed together the papers reviewed on Optimal Power Flow
- discussed the implementation of reinforcement learning in MATLAB
- discussed the use of the power systems software MATPOWER in the project
- discussed the needed tasks for the following week: set the ground for the implementation of the simpler DC Optimal Power Flow problem in MATLAB, through gathering the necessary information related to rewards, etc.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- the need to gather information for the DC implementation of the optimal power flow problem
- use the software MATPOWER in the implementation of the environment for the RL* agent
- upload the MATPOWER files to the online version of MATLAB

* : RL = Reinforcement Learning

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
- conduct literature reviews on RL for Optimal Power Flow - jot down the environment, rewards, agent info, etc	Mohammad Fares El Hajj Chehade	21/09/2022
- learn the concepts of RL - conduct literature reviews on RL for Optimal Power Flow	Mohamad Al Tawil	21/09/2022
- set MATPOWER for the MATLAB online version - learn the concepts of RL and take its MATLAB course	Karim Khalife	21/09/2022

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
4	9/21/22	10 a	36 mins	[✓ Advisor] Students	Mohamad Al Tawil

Attendees:

Mohammad Fares El Hajj Chehade - Karim Khalife - Mohamad Al Tawil - Rabih Jabr

Briefly summarize the main discussions during the meeting:

- Discussed a paper that develops a reinforcement learning (RL) solution to the optimal power flow (OPF) problem for a system containing wind power and battery storage
- Mapped the findings of the paper to our case of DC Optimal Power Flow (the initial model we will be working with) through identifying the state, action and reward
- Ran the software MATPOWER on the online version of MATLAB and made sure that the software is ready for use
- Discussed other papers that tackle reinforcement learning for optimal power flow at a higher level of complexity

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- The environment is set and the Markov Decision Problem properties are agreed upon
- The MATPOWER software is ready for use on the online and offline versions of MATLAB
- The DC Optimal Power Flow problem is correctly formulated and ready for solving

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
- Review papers that tackle RL in OPF - Run the RL agent in MATLAB for a simpler problem - Run MATPOWER on a standard IEEE bus	Mohammad Fares El Hajj Chehade	28/09/2022
- Review papers that tackle RL in OPF and jot down observations related to the RL agent and the environment - Run the RL agent in MATLAB for a simpler problem	Mohamad Al Tawil	28/09/2022
- Explore the data structures in MATPOWER - Review the concepts of RL	Karim Khalife	28/09/2022

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
5	9/28/22	10 a	35 mins	[✓ Advisor] Students	Mohammad Fares El Hajj Chehade

Attendees:

Mohammad Fares El Hajj Chehade - Karim Khalife - Mohamad Al Tawil - Rabih Jabr

Briefly summarize the main discussions during the meeting:

- Discussed the literature review progress
- Viewed together the high-level Simulink model
- Discussed the needed literature review to be done
- Discussed the needed demo to be done

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- A demo on reinforcement learning must be done for next week.
- Further reviews of relevant literature must be conducted on general applications of deep reinforcement learning (RL) in power systems and smart grids.
- Further reviews of relevant literature must be conducted on the applications of supervised learning in optimal power flow (OPF).

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
- further literature reviews on OPF for RL - demo on general RL example	Mohammad Fares El Hajj Chehade	05 / 09 / 2022
- further literature reviews on OPF for RL - RL MATLAB course - demo on general RL example	Mohamad Al Tawil	05 / 09 / 2022
- study of MATPOWER software - literature reviews on AI in smart grids & OPF - RL MATLAB course	Karim Khalife	05 / 09 / 2022

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
6	10/5/20	10 a	30 mins	[✓ Advisor] Students	Mohammad Fares El Hajj Chehade

Attendees:

Mohammad Fares El Hajj Chehade - Mohamad Al Tawil - Karim Khalife - Rabih Jabr

Briefly summarize the main discussions during the meeting:

** we brought to the meeting a reinforcement learning (RL) demo in MATLAB
- we discussed the working of the RL demo (environment, RL agent)
- we mapped the characteritics of this RL model to those related to our project (states, actions, etc)
- we discussed the simplified bus model that could be used as a first step in the DC Optimal Power Flow (DC OPF) problem

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- the RL demo is very explainable and easy to understand
- the mapping can be done from the observations, actions and rewards stated to the ones needed in our project
- the states can be intialized randomly at the beginning of every step

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
- closely examine the RL demo for getting additional insights - initiate the mapping of the project to the case of DCOPF	Mohammad Fares El Hajj Chehade	
- construct the simple bus for DCOPF - run the power flow on the constructed bus	Karim Khalife	
- tune the hyperparameters of the neural networks in the model	Mohamad Al Tawil	

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
7	12/10/20	10 a	20 mins	[✓ Advisor] Students	Mohammad Fares El Hajj Chehade

Attendees:

Mohammad Fares El Hajj Chehade - Karim Khalife - Rabih Jabr

Briefly summarize the main discussions during the meeting:

- we went over the simple bus system that will be used in our model
- we went over the environment scheme
- we discussed the progress report requirements

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- work on the progress report for next week
- fix the bus system
- initiate the code for our model

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
tune the hyperparamters for the RL model	Mohamad Al Tawil	26/10/2022
build the bus model and perform load flow	Karim Khalife	26/10/2022
build the environment	Mohammad Fares El Hajj Chehade	26/10/2022

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
9	10/22/25	10 a	30 mins	[✓ Advisor] Students	Mohammad Fares El Hajj Chehade

Attendees:

Mohammad Fares El Hajj Chehade - Mohamad Al Tawil - Karim Khalife - Rabih Jabr

Briefly summarize the main discussions during the meeting:

- the 3-bus power network is discussed
- the environment for the model is also discussed
- the training of the agent and associated hyperparameters are touched

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- the progress is steady and according to plan
- since the network is finalized, the RL code can be designed and tested

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
- power flow and optimal power flow analysis for the simple network	Karim Khalife	3/10/2022
- environment for the RL model	Mohammad Fares El Hajj Chehade	3/10/2022
- training of the RL agent	Mohamad Al Tawil	3/10/2022

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
10	2/11/20	10 a	30 mins	[✓ Advisor] Students	Mohammad Fares El Hajj Chehade

Attendees:

Mohammad Fares El Hajj Chehade - Mohammad Al Tawil - Karim Khalife - Rabih Jabr

Briefly summarize the main discussions during the meeting:

- We ran the reinforcement learning code
- We discussed ways to improve the code

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- progress is good
- several ways can be implemented to improve the code such as adjusting the reward function and the generator cost

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
- fix the reward function - tune hyperparameters and check agent performance	Mohammad Fares El Hajj Chehade	9/11/2022
- fix the cost function - tune hyperparameters and check agent performance	Karim Khalife	9/11/2022
- try running the agent for higher steps per episode - tune hyperparameters and check agent performance	Mohamad Al Tawil	9/11/2022

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
12	9/11/20	10 a	35 mins	[✓ Advisor] Students	Mohammad Fares El Hajj Chehade

Attendees:

Mohammad Fares El Hajj Chehade - Mohamad Al Tawil - Karim Khalife - Rabih Jabr

Briefly summarize the main discussions during the meeting:

- went through the RL code
- discussed ways to resolve mismatches

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- work is good and should be focused for next week on improving the neural network results
- initializations schemes for the NN can be tried

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
- try reaching reasonable convergence in the code	Mohamad Al Tawil	16/11/2022
- tuning various hyperparameters in the code	Mohammad Fares El Hajj Chehade	16/11/2022
- testing the code for different schemes	Karim Khalife	16/11/2022

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
13	11/16/22	10 am	30 mins	[✓ Advisor] Students	Mohammad Fares El Hajj Chehade

Attendees:

Mohammad Fares El Hajj Chehade - Mohamad Al Tawil - Karim Khalife - Rabih Jabr

Briefly summarize the main discussions during the meeting:

- discussed the progress in the RL
- proposed a new solution based on neural networks
- discussed the progress report

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- the transition to neural networks as a solution could be the best decision for now
- the RL model can be given an extra week

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
- review OPF based on NN	Mohammad Fares El Hajj Chehade	23/11/2022
- check the RL model for some possible adjustments	Mohamad Al Tawil	23/11/2022
- examine the code basedon neural networks	Karim Khalife	23/11/2022

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
14	11/23/23	3:30 pm	30 mins	[✓ Advisor] Students	Mohammad Fares El Hajj Chehade

Attendees:

Mohammad Fares El Hajj Chehade - Mohamad Al Tawil - Karim Khalife - Rabih Jabr

Briefly summarize the main discussions during the meeting:

- discussed the neural network solution
- discussed what to include in the final report

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- we need to implement a neural network solution
- the report looks good, we need to add the neural network solution and some testing

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
- create a neural network model - fix the proposed solution methodology - add some pseudocodes - fix the design alternatives	Mohammad Fares El Hajj Chehade	
- test the neural network model - fix the executive summary - fix the standards and constraints - fix the conclusion - contribute in figures and pseudocodes	Mohamad Al Tawil	
- test the neural network model - fix the literature review - construct the diagrams - fix the appendices	Karim Khalife	

Bibliography

[1] M. B. Cain, R. P. O'neill, A. Castillo, *et al.*, "History of optimal power flow and formulations," *Federal Energy Regulatory Commission*, vol. 1, pp. 1–36, 2012.

[2] M. Huneault and F. D. Galiana, "A survey of the optimal power flow literature," *IEEE transactions on Power Systems*, vol. 6, no. 2, pp. 762–770, 1991.

[3] M. A. Abido, "Optimal power flow using particle swarm optimization," *International Journal of Electrical Power & Energy Systems*, vol. 24, no. 7, pp. 563–571, 2002.

[4] J. A. Momoh, R. Adapa, and M. El-Hawary, "A review of selected optimal power flow literature to 1993. i. nonlinear and quadratic programming approaches," *IEEE transactions on power systems*, vol. 14, no. 1, pp. 96–104, 1999.

[5] R. D. Christie, B. F. Wollenberg, and I. Wangensteen, "Transmission management in the deregulated environment," *Proceedings of the IEEE*, vol. 88, no. 2, pp. 170–195, 2000.

[6] J. Momoh, R. Koessler, M. Bond, B. Stott, D. Sun, A. Papalexopoulos, and P. Ristanovic, "Challenges to optimal power flow," *IEEE Transactions on Power systems*, vol. 12, no. 1, pp. 444–455, 1997.

[7] H. W. Dommel and W. F. Tinney, "Optimal power flow solutions," *IEEE Transactions on power apparatus and systems*, no. 10, pp. 1866–1876, 1968.

[8] R. P. O'Neill, T. Dautel, and E. Krall, "Recent iso software enhancements and future software and modeling plans," *Federal Energy Regulatory Commission, Tech. Rep*, 2011.

[9] Y. Tang, K. Dvijotham, and S. Low, "Real-time optimal power flow," *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 2963–2973, 2017.

[10] M. Aien, M. Rashidinejad, and M. F. Firuz-Abad, "Probabilistic optimal power flow in correlated hybrid wind-pv power systems: A review and a new approach," *Renewable and Sustainable Energy Reviews*, vol. 41, pp. 1437–1446, 2015.

[11] T. Niknam, H. Z. Meymand, and H. D. Mojarrad, "An efficient algorithm for multi-objective optimal operation management of distribution network considering fuel cell power plants," *Energy*, vol. 36, no. 1, pp. 119–132, 2011.

[12] E. Naderi, H. Narimani, M. Fathi, and M. R. Narimani, "A novel fuzzy adaptive configuration of particle swarm optimization to solve large-scale optimal reactive power dispatch," *Applied Soft Computing*, vol. 53, pp. 441–456, 2017.

[13] R. A. Jabr, "A primal-dual interior-point method to solve the optimal power flow dispatching problem," *Optimization and Engineering*, vol. 4, no. 4, pp. 309–336, 2003.

[14] R. A. Jabr, "Radial distribution load flow using conic programming," *IEEE transactions on power systems*, vol. 21, no. 3, pp. 1458–1459, 2006.

[15] J. Lavaei and S. H. Low, "Zero duality gap in optimal power flow problem," *IEEE Transactions on Power systems*, vol. 27, no. 1, pp. 92–107, 2011.

[16] J. A. Momoh and J. Zhu, "Improved interior point method for opf problems," *IEEE Transactions on Power Systems*, vol. 14, no. 3, pp. 1114–1120, 1999.

[17] R. Burchett, H. Happ, and D. Vierath, "Quadratically convergent optimal power flow," *IEEE Transactions on Power Apparatus and Systems*, no. 11, pp. 3267–3275, 1984.

[18] W. Stadlin and D. Fletcher, "Voltage versus reactive current model for dispatch and control," *IEEE Transactions on Power Apparatus and Systems*, no. 10, pp. 3751–3760, 1982.

[19] D. I. Sun, B. Ashley, B. Brewer, A. Hughes, and W. F. Tinney, "Optimal power flow by newton approach," *IEEE Transactions on Power Apparatus and systems*, no. 10, pp. 2864–2880, 1984.

[20] M. Rahli and P. Pirotte, "Optimal load flow using sequential unconstrained minimization technique (sumt) method under power transmission losses minimization," *Electric Power Systems Research*, vol. 52, no. 1, pp. 61–64, 1999.

[21] D. Cao, W. Hu, X. Xu, Q. Wu, Q. Huang, Z. Chen, and F. Blaabjerg, "Deep reinforcement learning based approach for optimal power flow of distribution networks embedded with renewable energy and storage devices," *Journal of Modern Power Systems and Clean Energy*, vol. 9, no. 5, pp. 1101–1110, 2021.

[22] X. Yan and V. H. Quintana, "Improving an interior-point-based opf by dynamic adjustments of step sizes and tolerances," *IEEE Transactions on Power Systems*, vol. 14, no. 2, pp. 709–717, 1999.

[23] T. Niknam, M. Zare, and J. Aghaei, "Scenario-based multiobjective volt/var control in distribution networks including renewable energy sources," *IEEE Transactions on Power Delivery*, vol. 27, no. 4, pp. 2004–2019, 2012.

[24] Y. Xu, Z. Y. Dong, R. Zhang, and D. J. Hill, "Multi-timescale coordinated voltage/var control of high renewable-penetrated distribution systems," *IEEE Transactions on Power Systems*, vol. 32, no. 6, pp. 4398–4408, 2017.

[25] M. R. Adaryani and A. Karami, "Artificial bee colony algorithm for solving multi-objective optimal power flow problem," *International Journal of Electrical Power & Energy Systems*, vol. 53, pp. 219–230, 2013.

[26] M. A. Shaheen, H. M. Hasanien, S. Mekhamer, and H. E. Talaat, "Optimal power flow of power networks with penetration of renewable energy sources by harris hawks optimization method," in *2020 2nd International Conference on Smart Power & Internet Energy Systems (SPIES)*, pp. 537–542, IEEE, 2020.

[27] J. Li, R. Zhang, H. Wang, Z. Liu, H. Lai, and Y. Zhang, "Deep reinforcement learning for optimal power flow with renewables using spatial-temporal graph information," *arXiv preprint arXiv:2112.11461*, 2021.

[28] T. Soares, R. J. Bessa, P. Pinson, and H. Morais, "Active distribution grid management based on robust ac optimal power flow," *IEEE Transactions on Smart Grid*, vol. 9, no. 6, pp. 6229–6241, 2017.

[29] J. F. Franco, L. F. Ochoa, and R. Romero, "Ac opf for smart distribution networks: An efficient and robust quadratic approach," *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 4613–4623, 2017.

[30] E. Dall'Anese, K. Baker, and T. Summers, "Chance-constrained ac optimal power flow for distribution systems with renewables," *IEEE Transactions on Power Systems*, vol. 32, no. 5, pp. 3427–3438, 2017.

[31] P. Fortenbacher, A. Ulbig, S. Koch, and G. Andersson, "Grid-constrained optimal predictive power dispatch in large multi-level power systems with renewable energy sources, and storage devices," in *IEEE PES Innovative Smart Grid Technologies, Europe*, pp. 1–6, IEEE, 2014.

[32] W. Stadlin and D. Fletcher, "Voltage versus reactive current model for dispatch and control," *IEEE Transactions on Power Apparatus and Systems*, no. 10, pp. 3751–3760, 1982.

[33] V.-H. Bui, A. Hussain, and H.-M. Kim, "Double deep q-learning-based distributed operation of battery energy storage system considering uncertainties," *IEEE Transactions on Smart Grid*, vol. 11, no. 1, pp. 457–469, 2019.

[34] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning," in *Machine learning techniques for multimedia*, pp. 21–49, Springer, 2008.

[35] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[36] W. Kong, Z. Y. Dong, D. J. Hill, F. Luo, and Y. Xu, "Short-term residential load forecasting based on resident behaviour learning," *IEEE Transactions on Power Systems*, vol. 33, no. 1, pp. 1087–1088, 2017.

[37] X. Pan, T. Zhao, M. Chen, and S. Zhang, "Deepopf: A deep neural network approach for security-constrained dc optimal power flow," *IEEE Transactions on Power Systems*, vol. 36, no. 3, pp. 1725–1735, 2020.

[38] K. Baker, "Emulating ac opf solvers with neural networks," *IEEE Transactions on Power Systems*, vol. 37, no. 6, pp. 4950–4953, 2022.

[39] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE access*, vol. 7, pp. 133653–133667, 2019.

[40] M. A. Wiering and M. Van Otterlo, "Reinforcement learning," *Adaptation, learning, and optimization*, vol. 12, no. 3, p. 729, 2012.

[41] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[42] J. Li, R. Zhang, H. Wang, Z. Liu, H. Lai, and Y. Zhang, "Deep reinforcement learning for optimal power flow with renewables using spatial-temporal graph information," *arXiv preprint arXiv:2112.11461*, 2021.

[43] H. Zhen, H. Zhai, W. Ma, L. Zhao, Y. Weng, Y. Xu, J. Shi, and X. He, "Design and tests of reinforcement-learning-based optimal power flow solution generator," *Energy Reports*, vol. 8, pp. 43–50, 2022.

[44] Z. Yan and Y. Xu, "Real-time optimal power flow: A lagrangian based deep reinforcement learning approach," *IEEE Transactions on Power Systems*, vol. 35, no. 4, pp. 3270–3273, 2020.

[45] E. R. Sanseverino, M. Di Silvestre, L. Mineo, S. Favuzza, N. Nguyen, and Q. Tran, "A multi-agent system reinforcement learning based optimal power flow for islanded microgrids," in *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*, pp. 1–6, IEEE, 2016.

[46] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[47] "Ieee draft standard for electric power systems communications -distributed network protocol (dnp3)," *IEEE P1815/D08, January 2012*, pp. 1–858, 2012.

[48] "Ieee standard for exchanging information between networks implementing iec 61850 and ieee std 1815(tm) [distributed network protocol (dnp3)]," *IEEE Std 1815.1-2015 (Incorporates IEEE Std 1815.1-2015/Cor 1-2016)*, pp. 1–358, 2016.

[49] "Ieee grid vision 2050 roadmap," *IEEE Grid Vision 2050 Roadmap*, pp. 1–15, 2013.

[50] Y. Zhou, B. Zhang, C. Xu, T. Lan, R. Diao, D. Shi, Z. Wang, and W.-J. Lee, "A data-driven method for fast ac optimal power flow solutions via deep reinforcement learning," *Journal of Modern Power Systems and Clean Energy*, vol. 8, no. 6, pp. 1128–1139, 2020.

[51] X. Qi, G. Wu, K. Boriboonsomsin, M. J. Barth, and J. Gonder, "Data-driven reinforcement learning–based real-time energy management system for plug-in hybrid electric vehicles," *Transportation Research Record*, vol. 2572, no. 1, pp. 1–8, 2016.

[52] R. Dobbe, O. Sondermeijer, D. Fridovich-Keil, D. Arnold, D. Callaway, and C. Tomlin, "Toward distributed energy services: Decentralizing optimal power flow with machine learning," *IEEE Transactions on Smart Grid*, vol. 11, no. 2, pp. 1296–1306, 2020.

[53] R. D. Zimmerman and C. E. Murillo-Sánchez, "Matpower 6.0 user's manual," *Power Systems Engineering Research Center*, vol. 9, 2016.

[54] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "Matpower: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, 2011.

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015.