Correlated Scenario Generation Using Generative Models

by

Muhammad Bilal

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2022 by the
Graduate Supervisory Committee:

Anamitra Pal, Chair
Raja Ayyanar
Keith Holbert

ARIZONA STATE UNIVERSITY

August 2022

ABSTRACT

With the continued increase in the amount of renewable generation in the form of distributed energy resources, reliability planning has progressively become a more challenging task for the modern power system. This is because with higher penetration of renewable generation, the system has to bear a higher degree of variability and uncertainty. One way to address this problem is by generating realistic scenarios that complement and supplement actual system conditions. This thesis presents a methodology to create such correlated synthetic scenarios for load and renewable generation using machine learning.

Machine learning algorithms need to have ample amounts of data available to them for training purposes. However, real-world datasets are often skewed in the distribution of the different events in the sample space. Data augmentation and scenario generation techniques are often utilized to complement the datasets with additional samples or by filling in missing data points. Datasets pertaining to the electric power system are especially prone to having very few samples for certain events, such as abnormal operating conditions, as they are not very common in an actual power system. A recurrent generative adversarial network (GAN) model is presented in this thesis to generate solar and load scenarios in a correlated manner using an actual dataset obtained from a power utility located in the U.S. Southwest.

The generated solar and load profiles are verified both statistically and by implementation on a simulated test system, and the performance of correlated scenario generation vs. uncorrelated scenario generation is investigated. Given the interconnected relationships between the variables of the dataset, it is observed that correlated scenario generation results in more realistic synthetic scenarios, particularly for abnormal system conditions. When combined with actual but scarce abnormal conditions, the augmented dataset of system conditions provides a better platform for performing contingency studies for a more thorough reliability planning.

The proposed scenario generation method is scalable and can be modified to work with different time-series datasets. Moreover, when the model is trained in a conditional manner, it can be used to synthesise any number of scenarios for the different events present in a given dataset. In summary, this thesis explores scenario generation using a recurrent conditional GAN and investigates the benefits of correlated generation compared to uncorrelated synthesis of profiles for the reliability planning problem of power systems.

DEDICATION

This thesis is dedicated to my parents for reasons I feel no need to expound upon. To my wonderful siblings, Zainab and Hareem, who always support me no matter what may come. To my roommates, Faisal, Ali, and Ahmed who made everyday life a pleasure. And to Laura, Alina, and Silvia for being the best of friends.

ACKNOWLEDGMENTS

I would like to acknowledge my advisor, Dr. Anamitra Pal, for making this research possible and for his constant guidance and feedback. His support throughout my masters degree has been invaluable and his insights have been extremely helpful. I would like to extend my gratitude to Dr. Keith Holbert and Dr. Raja Ayyanar for agreeing to be members of my thesis committee and for providing valuable feedback. I would like to thank all members of the Pal Lab for their help and support. They made the environment a pleasure to work in and their company is greatly appreciated. Special thanks to Dhaval Dalal for being a very helpful colleague and a great person to work with.

A note of thanks to SRP for providing the lab with their data and their feedback on the project.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Reliability planning for electric power systems is essential in keeping the operation of the electric power grid consistent and to make the system resilient enough to handle contingencies and abnormal operating conditions. With the added increase in renewable energy resources, reliability planning studies have to also take into account projected percentages of distributed generation. Renewable energy is expected to increase over 8% in 2022 compared to last year, pushing the 300 gigawatt mark [1]. Solar photovoltaic (PV) alone is projected to account for over 60% of this increase, as can be seen in Fig. 1.1. With added penetration of renewables in the electric power grid, additional stress is placed on the system due to the inherent variability that these types of generation sources introduce. Under abnormal operating conditions, this could lead to catastrophic power system failure [2–34].

In light of the above, reliability planning for renewable-rich power systems has become very important. At the same time, any analyses regarding the adaptability of the system under abnormal conditions will give accurate results, only when appropriate data are available to perform the studies. However, power system datasets generally do not contain a high enough number of abnormal operating conditions to base one's analyses or to train one's models. Since renewable resources are stochastic in nature and difficult to predict, data augmentation techniques provide power system operators with more information to help direct planning and operation.

Recently, many power system applications including reliability planning have benefited from machine learning algorithms [35]. However, even machine learning models need copious amounts of appropriate data to enable the algorithms to learn properly

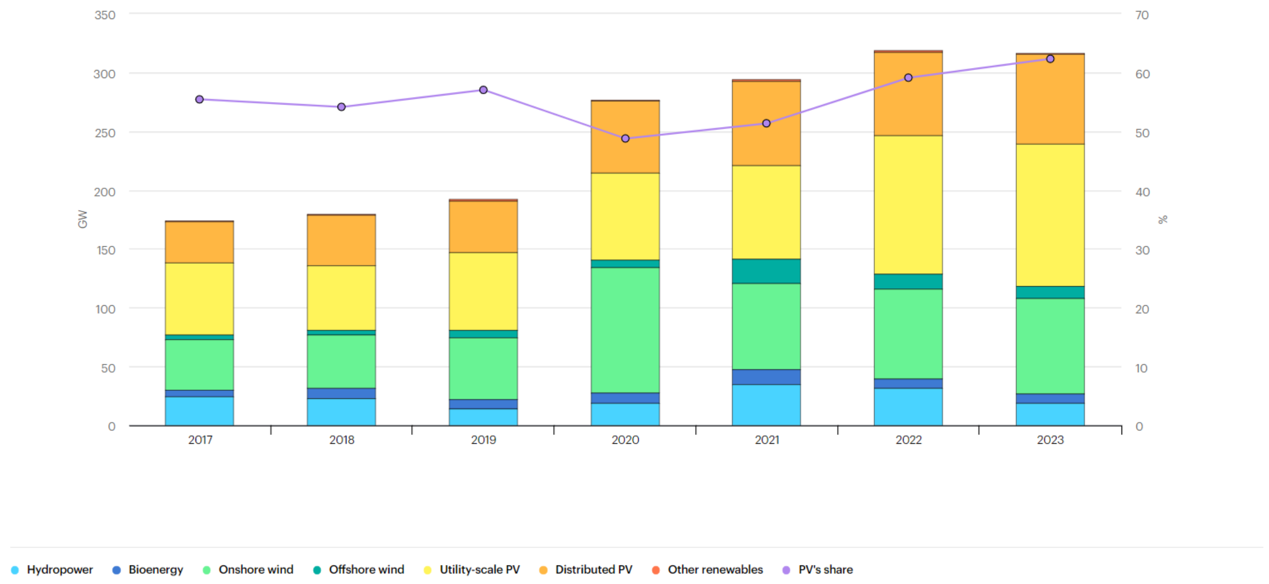

Figure 1.1: Net Renewable Capacity Additions Globally [1]

[36–42]. This calls for augmentation of the available datasets with more of the required scenarios. Various techniques, using both classical sampling methods or modern machine learning algorithms, have been proposed for this task. However many of these techniques require a probability distribution to be fitted onto the available dataset. Such methods are often difficult to scale and sample from. A data driven approach is therefore required to learn from the available dataset in an implicit manner so that the generation methodology is scalable.

Generative adversarial networks (GANs), since their inception in 2014, have been an innovative force in the domain of generative modeling. GANs have demonstrated the ability to learn from different types of datasets very well, whether it be two dimensional images or time-series data, and to generate samples which not only replicate the ones available in the real data but also other, more varied samples not present in the original dataset. This is due to the nature of GANs, which harness the ability to turn raw noise into meaningful information.

However, correlated scenario generation using GANs is a topic that has not been explored extensively in the literature. Here, correlated scenario generation refers to the correlation between the samples generated by the adversarial network. Since most real-world datasets are multivariate, the correlations between the variables are an important and fundamental component of the dataset. The correlations are particularly important for the electric power system datasets, where the variables are interconnected both indirectly and directly. For example, variables such as solar/wind energy generation and load demand are related in an indirect manner through conditions which are not easily apparent such as weather or geographic solar insolation. On the other hand, increased distributed PV generation in residential areas correlates in a direct manner with domestic load demand. The analysis of these correlations and the added complexity of training on complex, multivariate datasets is a topic worthy of exploration, and one of the focus areas of this study.

Validation of the generated scenarios is also a topic of significant importance, and is another focus area of this thesis. Verification is done both in a purely statistical manner, and by implementing the generated scenarios on an IEEE test system. Different variables are analyzed and the effect of correlated scenario generation is compared to uncorrelated scenario generation with respect to a power system application, namely, optimal power flow (OPF). This exercise provides a sample case for the use of the proposed methodology in real-world systems in meaningful ways.

The scalable nature of the GAN-based generation technique outlined in this work is demonstrated by the fact that it can be trained on any type of time-series data. It also provides the ability to generate any required number of meaningful sequences to augment the datasets. Since the methodology is model-free, the neural networks making up the generative algorithm are able to learn patterns and nuances of the available data effectively.

Summarizing the salient points, the study conducted in this thesis aims to:

- Develop a model to reproduce scenarios that not only resemble the original ones present in the dataset, but also new sequences that were not present initially.

- Construct a validation methodology to ensure that the generated scenarios can be safely used to replace and augment real-world datasets.

- Investigate the correlation between the variables and determine the benefits of correlated scenario generation for reliability planning in power systems.

1.1 Outline of the Thesis

A brief overview of the rest of the thesis is provided below.

Chapter 2 summarizes the literature review and previous work done in the areas of scenario generation and forecasting, focusing especially on using machine learning techniques to synthesise new scenarios. The motivations and need for better modeling techniques are also outlined.

Chapter 3 describes the working principle of GANs. Particularly, the conditional training and the recurrent nature of the generator and the discriminator are explained. To better understand the recurrent nature of the model, long short-term memory (LSTM) networks were employed, and a brief explanation of LSTMs is also provided in this chapter.

The methodology of the study and training of the machine learning model is presented in Chapter 4. This includes description of the dataset that was used, the prepossessing and classification of the data that was employed for training, and the selection of the parameters of the GAN.

The validation techniques utilized to confirm the usefulness and practical implementation of the generated scenarios are explained in Chapter 5.

Chapter 6 and Chapter 7 deal with the results of the study and the discussion of the results, respectively.

The conclusion is presented in Chapter 8.

Chapter 2

LITERATURE REVIEW

Forecasting and scenario generation have been extensively explored in the power system literature for creating synthetic operating conditions. Primarily two types of strategies are employed: classical techniques, which attempt to fit a model onto the distribution and attempt to forecast or generate scenarios from said model, and machine learning approaches, which learn the distribution from historical data and then are able to produce similar scenarios or forecasts.

A brief summary of the classical models is provided below. Reference [43] used conversions of a series of prediction errors to a multivariate Gaussian distribution to recursively generate a covariance matrix representing different scenarios. Reference [44] computed a probabilistic power flow, accounting for wind and load as correlated variables using a point estimate method, and compared it with a Monte Carlo based sampling simulation.

Reference [45] used an analog ensemble method for forecasting the day-ahead solar generation. Solar time and earth declination angle were chosen as the primary variables for clustering of data. Then, using parameters representing weather conditions, such as temperature, global horizontal irradiance, diffuse horizontal irradiance, and direct normal irradiance, the best suitable forecast from the historical data was identified.

Reference [46] used a generalized dynamic factor model (GDFM) to generate load and wind power scenarios. It was argued that GDFM has faster compute time since it requires less variables than a complete vector auto regression process, similar to the ones used in [47, 48].

In [49], power spectral density (PSD) was used to generate future wind scenarios. It was asserted that using time series to predict scenarios is more suitable for short-term forecast only. For longer forecasts, the relationship between PSD of wind power generation and the fluctuations in power generation were estimated. PSDs of the required scenarios were then generated using a multi-taper algorithm, focusing on the slope trends and the first and the last PSD values. These generated PSDs were used to create wind scenarios using the relationship established before.

Wind scenarios for a stochastic unit commitment model for planning of reserve requirements were generated using a third order auto-regressive model in [50]. Residuals were created from a random number generator using the model, diurnal and seasonal effects were integrated, and the resulting data were transformed into wind speed data. In [51], joint state-space representations were used to generate wind scenarios for uncertainty analysis of wind power plants.

A transmission planning problem was tackled using uncertainties of wind generation in [52]. Wind and load scenarios were synthesized using the sequential approximation method (SAM), which solves a two stage stochastic problem in each iteration. Each solution partitions the available dataset into finer segments. The average of the samples in the partitions were presented as the final scenarios.

Many methods for solar forecasting use numerical weather prediction (NWP) techniques. Day-ahead solar forecasts were produced using a least square optimization of a numerical weather prediction model in [53] with data from solar PV plants in the US Southwest. Three NWP models were utilized for solar irradiance forecasting using a probabilistic multi-model approach in [54].

A scenario generation based technique for modeling uncertainty in PV generation sources was presented in [55]. The proposed method used a roulette wheel mechanism to generate solar generation scenarios, which were then filtered and sorted into usable cases by a reduction process based on the fast forward method. This allowed the algorithm to produce scenarios with specific characteristics as per need.

To summarize, the classical methods generally relied on probabilistic modeling to generate new scenarios. However, in reality, these statistical assumptions may not always hold true. Furthermore, there is a strong possibility that the characteristics of the new scenarios remain very similar to those of the original dataset. Particularly, clustering or segmentation based data-driven methods are limited in the generation of novel forecasts, because they rely heavily on the segmentation methodology used to segregate the data into classes. Lastly, methods based on filtration of randomly generated scenarios are inflexible and computationally expensive if scenarios with specific characteristics are required.

In comparison machine learning models offer much more flexibility when it comes to generation of new scenarios. The neural network based approaches also lessen the need for extraction of relevant features from the available data. Consequently, neural networks have been extensively used for forecasting solar irradiation, such as in [56–58], which can then be used for solar production forecast.

Reference [59] utilized a neural network based stochastic process model to synthesize wind power generation scenarios. These scenarios were then utilized to forecast wind power ramp events. Reference [60] employed radial basis function neural networks to predict future wind power distributions.

Reference [61] used a data mining approach with machine learning based knowledge discovery method that utilizes cluster estimation techniques to predict and forecast load scenarios. Another machine learning based method for load forecasting is used in [62], which employed techniques such as Bayesian clustering by dynamics (BCD) and support vector regression (SVR). The BCD algorithm was used to classify the data based on the dynamics of the time series profiles into more meaningful subsets. Then, SVR was applied to the aforementioned subsets in a supervised manner to forecast load demand.

Reference [63] focused on improving forecasts by utilizing artificial neural networks (ANNs) to improve short term PV generation forecasts by re-forecasting previously generated predictions of different models, including an auto-regressive integrated moving average (ARIMA) model and a k-th nearest neighbor-based model.

Another solar forecasting method which focused on different weather variables that affect PV production was proposed in [64]. The method employed NWP data such as radiation, temperature, and precipitation and used different models to forecast solar production. These models included an ANN, and a recurrent model based on long short-term memory (LSTM) cells. The paper then compared the performance of these models.

An ANN based technique was proposed in [65] which was used to forecast PV, load and wind scenarios separately. The method involves assimilation of Gaussian white noise at the outputs of the ANN to generate the required scenarios.

A class of generative algorithms called generative adversarial networks (GANs) has been growing in popularity in recent years. First proposed in [66], GANs have been used in a variety of fields ranging from generating realistic looking faces [67–69], to producing new music [70–72]. GANs have also been used to remove face masks from photos of faces [73], or to apply makeup on photos of people [74]. It can also transform text to images [75–79], generate textures for video games [80], or discover new drugs [81, 82].

GANs have also been used for data augmentation and/or scenario generation. References [83–85] used GANs to augment image datasets. More relevant to the study conducted in this thesis are time-series datasets and their augmentation. Reference [86] used GANs to augment financial time-series data, while [87] augmented medical time-series data. In [88], GANs were used for generating seismic time-series signals for training of algorithms for detection of earthquakes.

GANs have also been used to generate power system scenarios. Since abnormalities in power systems are rare events that evolve temporally, augmentation of a dataset of system failures or abnormal operating conditions provides great value. These augmented datasets can then be used to train further models for either detection of abnormalities before they occur or classification of unusual operation after an event occurs. Reference [89] used LSTM, GANs, and reinforcement learning to generate wind power generation scenarios. The approach was then tested on two case studies to demonstrate that the architecture was able to generate varied and believable scenarios consistently. The approach was also compared with other scenario generation approaches such as multivariate kernel density estimation or vanilla LSTM.

Another GAN based approach for synthesizing wind power scenario was outlined in [90]. A support vector classifier was used to first classify the raw data and assign labels. The paper then used a flavor of GAN whose error function is based on Wasserstein Distance, called Wasserstein GAN [91] to generate the required scenarios. Another Wasserstein GAN-based approach for wind power scenario generation was employed in [92]. The training was improved by imposing a gradient penalty on the discriminator network.

In [93], synthetic fault data was generated for wind turbines using GANs to train fault detection models. Meanwhile, [94] used a GAN to account for incomplete data measurements from phasor measurement units (PMUs). The missing data were filled in with synthetic data generated by GANs and the complete data were then used for dynamic security assessment of the system.

A GAN combined with a supervised embedding network was employed in [95] to generate realistic time-series samples. The embedding network provides a separate latent space for the generator to operate in and facilitates the translation of features from the raw data to a form that made it easier for the GAN to reach equilibrium.

Conditional solar and wind scenarios were synthesised in [96] using a convolutional GAN with a Wasserstein distance-based loss function. Various conditions were accounted for such as seasonal solar generation or generating wind scenarios with specific ramp events.

Correlated scenario generation was performed in [97] to model the uncertainties in solar PV and wind production. These generated scenarios were then used to optimize a large scale hydro-wind-solar hybrid system using mixed integer linear programming (MILP). Unit commitment was then performed on the system to decide on the most cost effective generation procedure involving all the generation sources.

Although the aforementioned studies explore a number of GAN-based architectures to generate meaningful scenarios, the correlation between the generated scenarios is not investigated in these works. Since the power system is a complex network of interconnected generation sites and electricity consumers, both residential and large scale, the relationship between the different modes of power production and the nuances of load demand must be explored in a more thorough fashion. This is especially true with the growing prevalence of distributed energy resources (DERs), which further stress the transmission and distribution systems of the electric power grid.

The study conducted in this thesis aims to fill this gap while also providing robust methods to validate the generated scenarios, and investigate the correlation between the variables. In the following chapters, the benefits of correlated scenario generation is outlined, and the verified results are presented.

Chapter 3

RECURRENT CONDITIONAL GENERATIVE ADVERSARIAL NETWORKS

Since the introduction of GANs in 2014 by Goodfellow et al. [66], GANs have become increasingly popular as a novel method of scenario generation. Although used in other areas as well such as event flagging [98] or abnormality detection [99], GANs have primarily been employed as a generative model to augment datasets, to generate new images, to fill in missing data, or to create new scenarios.

The ability of GANs to generate varied yet realistic scenarios is further enhanced by training the network conditionally, that is, by providing labels to each data point that can later be used to generate only a specific type of scenario or event. This gives the user more control over what the GAN is going to produce and is helpful for generating samples for case-based studies.

Over the years, many different types of GAN architectures have been proposed, each catering to a certain type of problem, and each with its pros and cons. Some of these architectures have different variants of the generator and discriminator, the two neural networks which together make up the adversarial network in GANs. Other approaches utilize different loss functions in their attempts to stabilize training. This brings forth one main issue with the usage of GANs: they are notoriously difficult to train and to bring to a stable state. Problems such as vanishing or exploding gradients, coupled with the stochastic nature of the training process, and with the adversarial nature of the model makes for an overall difficult training procedure.

The following sections outline the basic concept of GANs, and its chosen variants, as well as the reasoning behind the choices.

3.1 Generative Adversarial Networks (GANs)

GANs are a type of machine learning model composed of two neural networks battling against one another, and hence the word ‘adversarial’ in the name. The first neural network is called the generator, whose purpose is to generate the synthesised samples. The second neural network is called the discriminator (or the critic). The discriminator’s job is to differentiate between the real and the generated samples.

The main objective of GANs is to learn the distribution of a real set of data and map it to a separate latent space, from which more samples, which are similar to the original dataset, can be synthesized. Suppose that we have a set of data X, with samples x_i^t for time $t \in T$, and with dimensions i, whose distribution, $\mathbb{P}_x$, is to be learnt by the generative model. Noise vector inputs z are sampled from a latent space, $\mathbb{P}_z$, and the multi-layer perceptrons within the generator are trained to map $\mathbb{P}_z$ to $\mathbb{P}_x$, without explicitly training on $\mathbb{P}_x$. This is done by the generator producing samples as close to the real data’s distribution $\mathbb{P}_x$ as possible, while the discriminator tries to distinguish the real samples from the generated ones and forces the generator to perform better. As the training progresses, the generator becomes better at producing realistic looking samples, while the discriminator gets better at distinguishing generated samples from the real ones.

3.1.1 Generator

The generator, $G(z, \theta_g)$, with parameters θ_g, is a neural network which samples noise vectors z from the latent space $\mathbb{P}_z$, and uses the neural network to produce realistic looking samples, whose distributions can be denoted by $\mathbb{P}_g$. As the training proceeds the generator tries to bring $\mathbb{P}_g$ as close to $\mathbb{P}_x$ as possible. Each sample from $\mathbb{P}_g$ is then evaluated by the discriminator. The objective of the generator is to fool

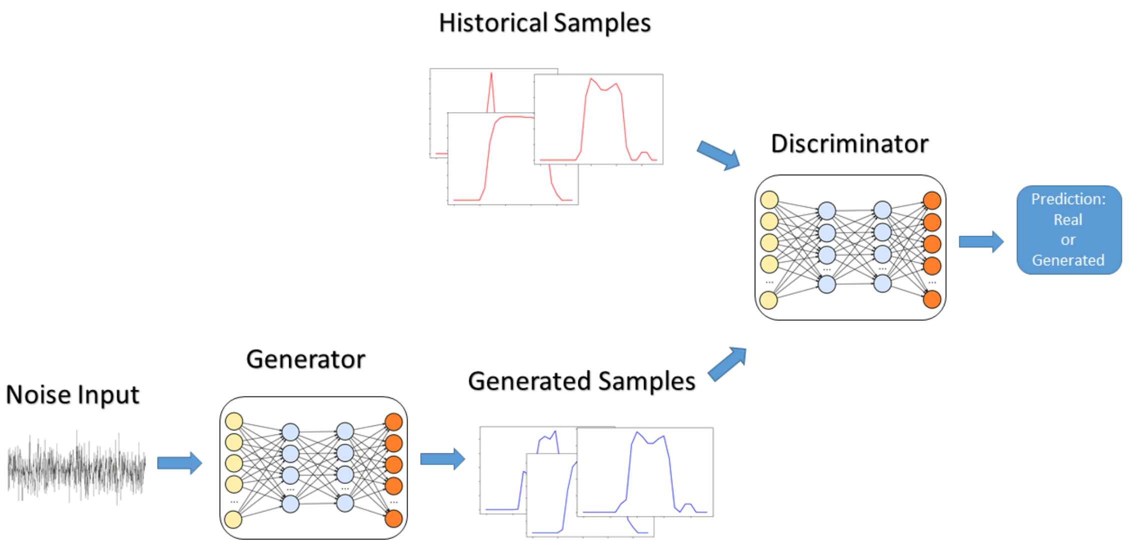

Figure 3.1: Generative Adversarial Networks

the discriminator into classifying the synthesised data as real data. Hence, the loss function of the generator can be defined as,

$$L_G = \mathbb{E}_Z[\log(1 - D(G(z)))] \tag{3.1}$$

3.1.2 Discriminator

The discriminator, $D(z, \theta_d)$, with parameters θ_d, is the second neural network in a GAN model. It outputs a scalar value for each data sample x denoting whether it came from the real data's distribution, $\mathbb{P}_x$, or from the generated data's distribution, $\mathbb{P}_g$. As the training progresses, the discriminator tries to maximise the difference between the real data, $\mathbb{E}[D(X)]$ and the generated data, $\mathbb{E}[D(Z)]$.

The discriminator is fed both real and generated samples in turn and both the losses are combined to train the discriminator network further. Hence, the discriminator's loss function can be defined as,

$$L_D = -\mathbb{E}_X[\log(D(x))] - \mathbb{E}_Z[\log(1 - D(G(z)))] \tag{3.2}$$

3.1.3 GAN Objective Function

The training of the generator and the discriminator can be summarized as a two-player minimax game with the value function $V(G, D)$,

$$\min_G \max_D V(G, D) = \mathbb{E}_X[\log(D(x))] + \mathbb{E}_Z[\log(1 - D(G(z)))] \tag{3.3}$$

In practice, at the start of training, the discriminator loss is very low and the generator loss is very high, because the generator has not been trained well enough yet to produce samples that are similar to the real data. Therefore, the discriminator can reject the generated samples with high confidence, causing the generator's loss to saturate. It is, therefore, prudent to maximize $\log(D(G(z)))$ while training the generator instead of minimizing $\log(1 - D(G(z)))$. This reformulates the generator's loss from (3.1) to the following,

$$L_G = -\mathbb{E}_Z[\log(D(G(z)))] \tag{3.4}$$

The discriminator's loss can also be rewritten as,

$$L_D = -\mathbb{E}_X[\log(D(x))] + \mathbb{E}_Z[\log(D(G(z)))] \tag{3.5}$$

The value function of GANs from (3.3) will also change to,

$$\min_G \max_D V(G, D) = \mathbb{E}_X[\log(D(x))] - \mathbb{E}_Z[\log(D(G(z)))] \tag{3.6}$$

After training has finished, ideally, the generator and the discriminator reach an equilibrium, where the generator has been trained enough to produce realistic looking samples and the discriminator is unable to tell the generated samples apart from the real ones. At this point, noise vectors, z, given to the generator as input will be converted to a sample from the distribution $\mathbb{P}_g$ and will be very similar to a sample from $\mathbb{P}_x$. The GAN is then said to have been trained and the generator can be used to generate any number of samples by giving the required number of noise vectors, z, as input.

3.2 Conditional GANs

GANs can be trained in a conditional manner by incorporating labels in the training dataset, allowing the generator the ability to generate samples based on a certain event or condition. The label, y, can be any auxiliary information that can then be appended to the real samples, x. The generator will then learn to associate a certain class of data with its associated label. After training has been finished, the generator can then be forced to produce only a certain class of samples by appending the corresponding label, y, to all the noise vectors. The loss of the generator then becomes,

$$L_G = -\mathbb{E}_Z[\log(D(G(z|y))] \tag{3.7}$$

The discriminator will take both the real and fake samples as input, along with their respective labels, and will learn to distinguish between them. The discriminator loss can be represented as,

$$L_D = -\mathbb{E}_X[\log(D(x|y))] + \mathbb{E}_Z[\log(D(G(z|y)))] \tag{3.8}$$

The value function of the conditional GANs can be written as,

$$\min_{G}\max_{D} V(G,D) = \mathbb{E}_X[\log(D(x|y))] - \mathbb{E}_Z[\log(D(G(z|y)))] \tag{3.9}$$

3.3 Long Short-Term Memory (LSTM)

Long short-term memory are a type of neural networks widely used in processing sequential data and are employed to solve problems such as regression modeling [100] [101] and natural language processing [102, 103]. They are a type of recurrent neural network (RNN), which are designed to store information as they are trained, and make predictions based on both the new data and the previously stored information.

RNNs are designed to mimic the process of how humans think. Humans tend to store memories and information. This stored data helps us in making more informed decisions and give humans a better understanding of the problem at hand. Similarly, RNNs with LSTMs have cyclic loops in their structure, as shown in Fig. 3.2, which enables them to store information that the neural network deems important. The cyclic structure of RNNs is better explained when imagining them as series of similar networks, each passing information to its successor. This "unrolled" version of RNN cells, as shown in Fig. 3.3, better demonstrates how sequential data are handled by the network and how each previous input is related to the next.

The problem with simple RNNs arises when the sequence becomes large. The sequential chain of cells becomes too long and the neural network is then unable to pick up on repeated sequences, and thus, does not learn long-term dependencies. In order to fix these problems and to enable neural networks to learn long-term correlations, LSTMs were proposed [104].

LSTMs are specifically designed to hold information for long periods of time or for long sequences, and are hence able to handle long-term dependencies. They do this by keeping track of a cell state for each LSTM cell. The cell state holds information for

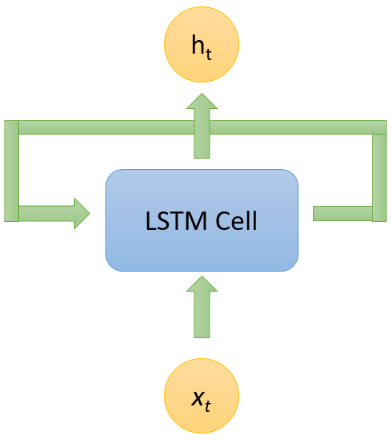

Figure 3.2: LSTM Cell

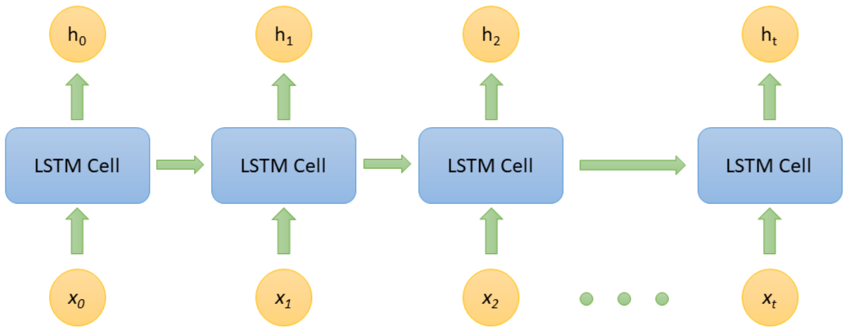

Figure 3.3: Unrolled LSTM Cell

an indefinite period of time and is updated as per need. There are also hidden states of each LSTM cell. These hidden states are combined with the input and passed onto the next layer as their hidden states where they are further processed. The hidden states of the last layer of the LSTM network are the output of the network.

3.3.1 LSTM Gates

Each LSTM cell is composed of 4 gates. The descriptions of each of them are given below:

Forget Gate

The forget gate decides whether the cell state needs to be updated or not. It looks at the previous cell's hidden state, h_{t-1}, and the current input x_t, runs it through a sigmoid function and outputs a number between 0 and 1. This decides whether the previous cell's cell state, C_{t-1}, is going to be updated or not. A 1 means the cell state's information is deemed relevant and is going to be passed on as it is. A 0 means that all the cell state information is going to be blocked and the network does not deem it relevant enough. A value between 0 and 1 allows the cell to filter the useful information and pass it on. The function of the forget gate, f_t can be summarized as,

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f), \tag{3.10}$$

where W_f and b_f are the weights and the biases of the forget layer, respectively.

Input Gate

The input gate has a two-fold function. Firstly, it decides which cell state values must be updated. This is done by passing the previous hidden state, h_{t-1}, and the input, x_t, through a sigmoid layer. Similar to the output gate, the sigmoid activation will give us values between 0 and 1, and these values decide which information should be the new cell state. This function can be summarised as,

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{3.11}$$

Secondly, it produces a new candidate cell state which is fashioned from the combination of h_{t-1} and x_t, and passed through a tanh activation function. This gives us values between -1 and 1 for being written as the new cell state. This layer is sometimes called the input modulation layer. The aforementioned function can be written as,

$$\tilde{C}_t = \tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{3.12}$$

The cell state is then updated as,

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{3.13}$$

Output Gate

The output gate of the LSTM decides which new hidden states should be forwarded to the next cell. This is performed by first deciding which parts of the updated cell state are going to be forwarded as the new hidden state. Sigmoid activation is performed on h_{t-1} and x_t. This vector filters out the updated cell state. But before this filtration, the cell state is passed through a tanh activation function. These procedures can be summarized as,

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{3.14}$$

$$h_t = o_t * \tanh(C_t) \tag{3.15}$$

In essence, LSTMs provide a way to retain the long term nuances of the sequential data in a dynamic manner, supported by deep learning. This makes LSTMs one of the primary recurrent models used today for processing sequential data.

3.4 Recurrent GAN

Recurrent architectures, like the LSTM described in the section before, can be incorporated into GANs to form a recurrent GAN. This is especially useful for generating sequential time-series data as the recurrent layers in the generator model will retain the long-term modulations of the time-series and help in generating sequences which capture all the fluctuations of the real data. Recurrent layers can also be used in the discriminator to help it identify sequential data better.

The number of recurrent layers in both the discriminator and the generator is a hyperparameter that must be tuned as per need to make the model stable. Other hyperparameters are the number of hidden cells in each LSTM layer and the number of stacked LSTM layers themselves. These are also tuned to match the data and to allow the model to be stable.

Other works have employed convolutional layers in generating time series sequences, such as in [96], essentially forming a convolutional GANs. This is done by reformatting the data as a 2-d image on which convolutional functions can be applied. The study conducted in this thesis, however, focuses on using a recurrent model since the dataset available is a time-series data and RNNs are the model of choice to train on sequential data.

Chapter 4

METHODOLOGY

The proposed methodology is a multi-step approach. First, the raw dataset is classified into different classes based on a suitable metric so that appropriate labels can be assigned to each class. This would allow the GAN to be trained conditionally using the assigned labels. Secondly, the GAN is trained on the data using the assigned labels. The trained GAN can then be used to generate any number of sequences as per need. The type of sequences generated can be controlled by concatenating the required label for the class to the noise inputs of the generator.

In order to compare the benefit of multivariate correlated scenario generation with univariate uncorrelated scenario generation, separate GANs for generation of solar and load sequences are trained. The correlated and uncorrelated profiles are then validated statistically and by imposing both the real and synthesized sequences on an IEEE test system. This allows direct comparison of the synthesized sequences with real ones and confirms that the scenario generation methodology produces profiles that are similar to real profiles and are trustworthy.

4.1 Description of Dataset

The dataset consists of two years' worth of solar generation and load demand profiles from Salt River Project (SRP), a power utility located in Arizona, as shown in Table 4.1. After removing bad/missing data, the remainder of the data was divided into sequences of 24 hours, for a total of 730 days. This provided the recurrent layers of the GAN with long enough profiles to train on. Moreover, dividing the hourly data into days also made it easier to classify the data in a seasonal fashion.

Table 4.1: Dataset Description

Resolution	Hourly
Start Date	April 1, 2018
End Date	March 31, 2020
Variables	PV Solar Production and Load Demand (MW)

4.2 Classification of Raw Data

At the start of this study the data were classified into normal, borderline, and abnormal days. The borderline class was introduced to include days not easily classified into either normal or abnormal classes. It was later found that incorporating seasonal labels improves the performance of the GANs and allows it to further fine-tune itself. The added benefit of generating normal/abnormal data by season is also useful. The borderline class was dropped after segregating the data by season because one more class for each season stratified the data too much for the GAN to be trained properly.

4.2.1 Classification by Season

Since both solar PV generation and load demand change drastically by season in Arizona, segmentation of the dataset on a seasonal basis was carried out. PV production in summer is very different in both average/maximum generation and also temporally throughout the day. Other factors affecting PV generation include geographic insolation, weather patterns, and humidity. Load demand also depends on factors such as geographic location, climate, and peak/minimum temperatures. Hence, a robust classification methodology is required for classifying the dataset into seasons that keeps the inherent similarity between members of each class intact.

Classification by season is performed by observing the daily moving averages of the dataset variables and defining the start and end of classes based on the change in temporal patterns throughout the length of the dataset. According to this logic, the dataset was segregated into three seasonal classes: summer, shoulder, and winter. The start end dates of each class for all the years are given in Table 4.2.

Table 4.2: Seasonal Classification

Season	Start Date	End Date
Shoulder A	April 1	June 19
Summer	June 20	Sep 18
Shoulder B	Sep 19	Nov 15
Winter	Nov 16	Feb 14

The days not sorted into either summer and winter were placed in the shoulder class. This gave result to two segments of the shoulder season. Both shoulder A and B were combined to form a single shoulder class.

4.2.2 Classification into Normal/Abnormal Days

From a power system engineering perspective, abnormal days are those days where one or more variables are out of the bounds of what is considered normal operation. Capturing attributes of abnormal days are valuable for studying and analyzing the strength of the overall system, its resilience to the abnormalities, and to find the worse-case conditions. Since abnormal days are rare and far between in actual power system datasets, more profiles pertaining to abnormal operating conditions can greatly help in designing power systems to handle abnormalities better. Having access to similar number of normal and abnormal system conditions will also ensure creation of a balanced training dataset for doing machine learning in power systems.

From a scenario generation perspective, abnormal days are quite different in shape and temporal patterns compared to normal days. Furthermore, abnormal days within themselves are vastly different because the abnormalities can arise from a number of different factors. For example, an abnormal solar day can be one where the solar insolation was dramatically low throughout the day due to stormy weather. This would make the solar profile relatively flat throughout the day. Alternately, it can also be a day where the solar generation was low many times in the day due to intermittent cloud cover. This will give rise to multiple peaks in the solar generation profiles. Lastly, an abnormal load day might include a day where the peak load demand was very high due to a severe heatwave.

To facilitate a generative model to better learn all the nuances of the different types of abnormal days and not let its inferences on normal days affect the generation of abnormal days, it is better to segment abnormal days into their own class. Also, since the number of abnormal days are much less than normal days, the generative model might not even be able to generate abnormal days without separate classification because the weights will be tuned to cater to the most common profiles in the dataset. With a bigger dataset, further segmentation will give more freedom to generate a specific sort of daily profile as per the requirement.

All these factors call for a method of classification that is flexible enough to account for the different profiles present in the dataset, and robust enough for the results to be repeatable and be reproducible consistently. The data segmentation technique used in this thesis is the one developed in [105]. Dynamic Time Warping (DTW) was chosen as the measure of choice for this purpose as it captures temporal similarity better than the well-known Euclidean distance measure.

4.2.3 Dynamic Time Warping (DTW)

DTW computes the similarity between two time-series datasets that are normalized in time. This means that even if the temporal sequences have a different measure of speed (e.g., decelerating segments or accelerating segments), the DTW distance will still give a good measure of similarity. This makes DTW an ideal candidate to deal with time-series sequences.

DTW works by computing the optimal match between different sequences, with the lowest possible cost. The cost is defined as the sum of absolute differences for each pair of indices in the time-series. It matches the indices of a sequence with the indices of other sequences in a way that the cost is minimal. The sequences with the lowest overall cost between them are said to be the most similar. The optimization constraints of the algorithm can be defined as follows:

- Every index in the first sequence must be matched to one or more indices in the second sequence.
- The first/last index in the sequence must be matched to the first/last indices in the other sequences, respectively.
- The mapping of indices must be monotonically increasing between the sequences.

Mathematically, the DTW algorithm is defined as:

$$P_0 = \arg\min_P \left[\frac{\sum_{s=1}^{k} d(p_s)}{k} \right] \tag{4.1}$$

where $d(p_s)$ is the distance between time series points i_s and j_s, k is the length of the warping path and P is the warping function. For n days, DTW creates a symmetric $R^{n\times n}$ matrix that has the DTW distances between pairs of days for each entry. From this symmetric matrix, the median representative day is selected as a normal day for each season based on the methodology proposed in [105]. This reference day is picked from the largest cluster of median days with the shortest distance to the median seasonal representation.

Next, the distance of each day in the season from the representative day is calculated and plotted. The resultant plot is investigated for an inflection point which suggests that the distance from the representative normal day has begun increasing. This singles out the point where the profiles are not representing normal days anymore. The rest of the days, with their DTW distance past the inflection point, are labeled as abnormal.

This is done for all the three seasons. After removing outliers and cleaning the data, the classification by season and by normal/abnormal days segregates the dataset in the manner displayed in Table 4.3:

Table 4.3: Number of Classified Days

Season	No. of Days
Summer Normal	143
Summer Abnormal	39
Shoulder Normal	327
Shoulder Abnormal	40
Winter Normal	146
Winter Abnormal	35

Note that the DTW scores for this study were obtained from [105]. Subsequently, the classification based on the DTW cutoff point was carried out to divide the data into normal and abnormal days.

4.3 Encoding of Labels

The sequences from the raw data are assigned their respective labels, both seasonal and normal/abnormal. The generated label vectors are then concatenated to the real data sequences that were input to the discriminator. A common method to do this is by one-hot encoding. However, in order to keep the length of the label vector as small as possible, the label vectors were generated using binary encoding in this research. Therefore, the length of the label vector length is 3 for six classes (three seasons, each with a normal/abnormal class). The maximum usable combinations of labels for a vector length of 3 is 8, and 2 combinations are not utilized. Table 4.4 shows the respective label vector for each season.

Table 4.4: Binary Encoding

Label	Label Vector
Summer Normal	0 0 0
Summer Abnormal	0 0 1
Shoulder Normal	0 1 0
Shoulder Abnormal	1 0 0
Winter Normal	0 1 1
Winter Abnormal	1 0 1

4.4 Training of Multivariate GANs

After prepossessing the data, the multivariate GAN was designed to learn the various distributions of the solar and load profiles in a conditional manner. The GANs incorporated LSTM layers to train on the the time-series data better and pick up on recurring patterns in the profiles. The structures of the generator and the discriminator are given in Fig. 4.1 and Fig. 4.2 respectively.

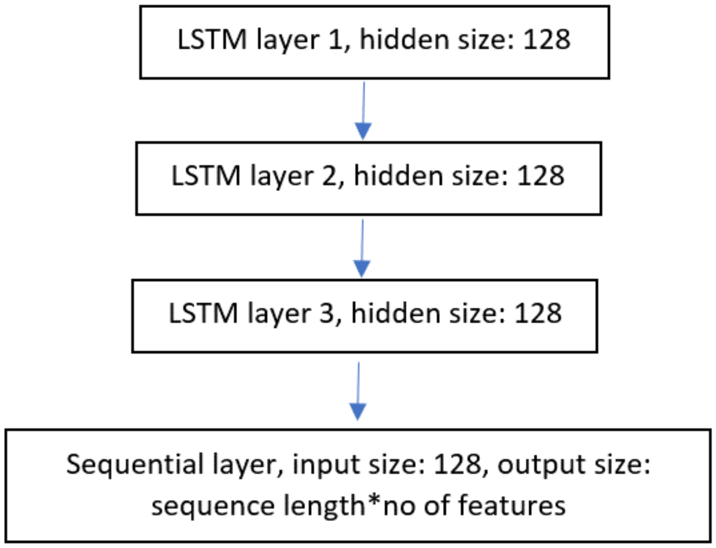
LSTM layer 1, hidden size: 128
LSTM layer 2, hidden size: 128
LSTM layer 3, hidden size: 128
Sequential layer, input size: 128, output size: sequence length*no of features

Figure 4.1: Generator Architecture

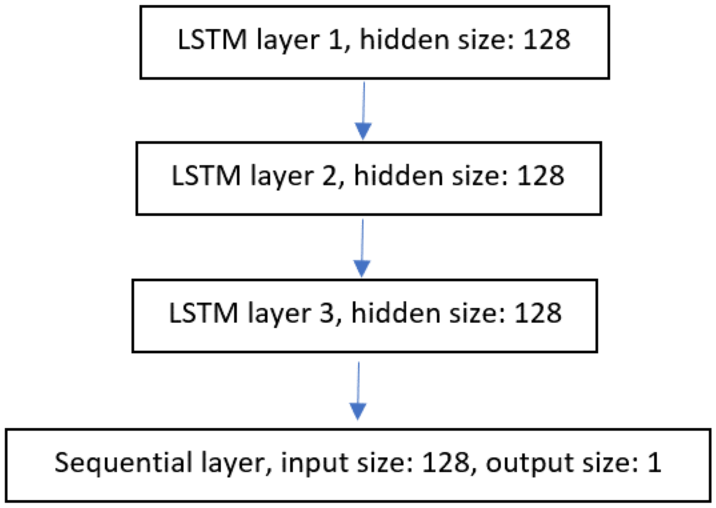
LSTM layer 1, hidden size: 128
LSTM layer 2, hidden size: 128
LSTM layer 3, hidden size: 128
Sequential layer, input size: 128, output size: 1

Figure 4.2: Discriminator Architecture

GANs are a notoriously difficult model to train and the tuning of hyperparameters is done on an ad hoc basis, usually by trial and error. The objective is to make the GAN stable enough to learn the required distributions well. The final parameters chosen for the main GAN created in this study are given in Table 4.5. Note that the hyperparameters for generator and discriminator are the same except for the fact that the discriminator is trained more than the generator.

Table 4.5: GAN Hyperparameters

Hyperparemeter	Value
No. of Variables	2
No. of Labels	6
No. of Epochs	2000
Learning Rate	0.0001
Generator Input Size	256
No. of Stacked LSTM Layers	3
Hidden Layer Size	128
Sequence Length	24
Length of Label Vector	3
Dropout	0.2
Generator Training Cycles	1
Discriminator Training Cycles	3

4.5 Training of Univariate GANs

For a deeper analysis of correlated scenario generation and its benefits, two more GANs, one for generation of solar profiles alone, and the other for generation of load profiles alone, are trained. The hyperparameters for the training of these GANs are the same as the ones shown in Table 4.5. This allows for the generation of univariate profiles which can then be imposed on a test system to compare their performance with correlated scenarios.

The univariate scenarios, however, are required to be classified again into normal/abnormal days since the DTW distance cutoffs that were used to classify the multivariate data are no longer representative of a dataset having a single variable. Thus, separate DTW-based classification of normal/abnormal days is done for both load and solar datasets. The number of days in each class for solar and load are given in Table 4.6 and Table 4.7, respectively.

Table 4.6: Number of Classified Days, Solar Only

Season	No. of Days
Summer Normal	159
Summer Abnormal	23
Shoulder Normal	337
Shoulder Abnormal	30
Winter Normal	162
Winter Abnormal	20

After this segmentation, separate GANs are trained on each labeled set of data.

Table 4.7: Number of Classified Days, Load Only

Season	No. of Days
Summer Normal	175
Summer Abnormal	7
Shoulder Normal	345
Shoulder Abnormal	22
Winter Normal	162
Winter Abnormal	20

Chapter 5

VALIDATION

After the training of all three GANs (the multivariate correlated GAN, the uncorrelated solar GAN, and the uncorrelated load GAN) described in the previous chapter is completed, any number of solar and load profiles can be generated as per need for all seasons and for correlated/uncorrelated and normal/abnormal labels. These profiles are validated statistically and by imposing them on a test system and running an optimal power flow (OPF). The results of the OPF are compared with the results obtained when running the OPF with real data. This provides a simulated real-world implementation and confirms the usefulness of the generated profiles, namely, whether they can be used to replace real data for actual power system analysis.

This fact is especially important for abnormal sequences as real-world datasets generally contain very few instances where a power system behaves abnormally. Faults are not a common occurrence in a modern power system. However, to carry out contingency studies and worse-case analyses, datasets with a significant number of abnormal cases must be available. Thus, validation of the generated abnormal days by comparing the response of the power system with the generated profiles and the real profiles is of great importance.

5.1 Statistical Verification

Statistical verification can be performed on the generated profiles in a number of ways. Here, two methods are chosen to compare the temporal patterns and the power spectrum of the synthesized sequences with real ones: auto-correlation and power spectral density (PSD).

5.1.1 Auto-correlation

Auto-correlation of a sequence is the correlation of the sequence with a delayed version of itself as a function of the delay. Mathematically, it is expressed as,

$$r_k = \frac{\sum_{t=k+1}^{n}(x_t - \bar{x})(x_{t-k} - \bar{x})}{\sum_{t=1}^{n}(x_t - \bar{x})^2} \tag{5.1}$$

where r_k is the auto-correlation function for time lag k, x_t is the value of the variable x at time t, and n is the total number of discrete data points.

Auto-correlation is helpful in discovering repeating patterns in time-series sequences. For validation of the generated profiles from the GANs, auto-correlation is helpful in showing whether the generated profiles have the same repeated patterns as the original data.

Fig. 5.1 shows a real and generated normal summer day. The auto-correlation of both these sequences are shown in Fig. 5.2. As can be seen from the figure, both the real and generated profiles display similar temporal fluctuations.

5.1.2 Power Spectral Density (PSD)

PSD characterizes the total energy of a signal distributed over a frequency range. Mathematically, for a time-series signal, $x(t)$, with Fourier transform $\hat{x_T}(f)$, it is defined as,

$$S(f) = \lim_{T\to\infty} \frac{1}{T}|\hat{x_T}(f)|^2 \tag{5.2}$$

PSD tells us how strong a signal is along a frequency distribution. Fig 5.3 shows the PSD between the real and generated profiles shown in Fig. 5.1. It can be inferred from the figure that the synthesized profiles show a power spectrum similar to profiles from the real data.

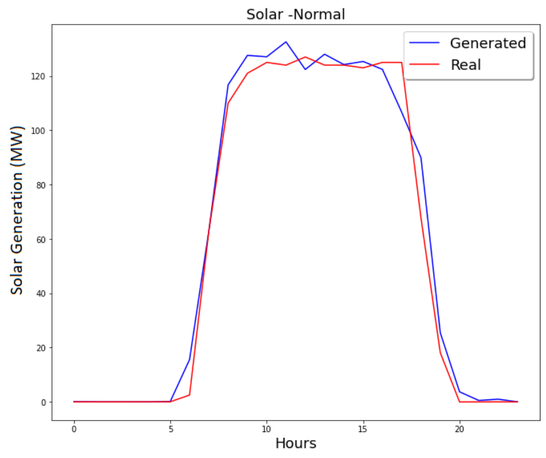

Figure 5.1: Real and Generated Normal Summer Day

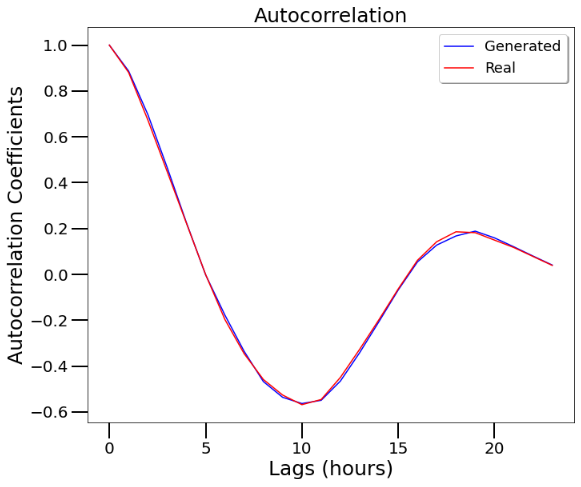

Figure 5.2: Auto-correlation Comparison

5.2 Verification on IEEE Test System

Further verification of the generated profiles and the comparison between the profiles generated in a correlated manner against the sequences synthesized in an uncorrelated fashion is possible by applying both the real and generated profiles onto a test system and performing an OPF.

The test system chosen for this task is the IEEE 30 bus system [106]. The parameters of this system are given in Table 5.1.

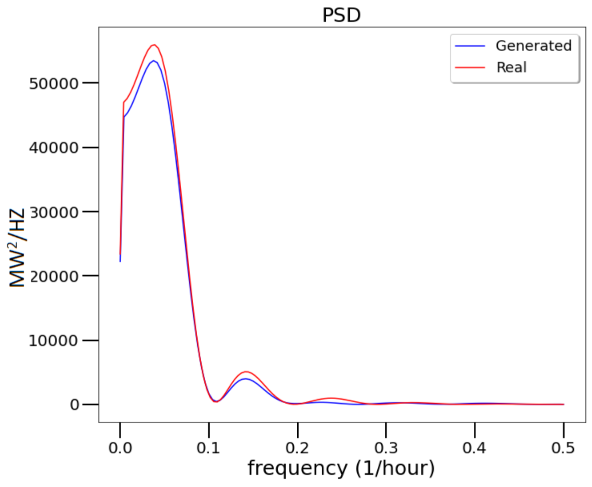

Figure 5.3: PSD Comparison

Table 5.1: IEEE 30-Bus System

Transmission Lines	41
Generators	2
Synchronous Condensers	4
Voltage Levels	132, 33, 11, 1 kV

Before application of the real and generated sequences onto the test system, they are normalized so that they match the peak load of the buses in the test system. For this reason all the solar profiles are divided by a factor of 150 and all the load profiles are divided by a factor of 8,000, since these numbers approximately reflect the peak solar production and load demand in the original dataset (obtained from SRP).

Next, both the real and generated profiles are applied onto each of the 30 buses. A futuristic generation scenario with a high solar PV penetration is envisaged, i.e., all the buses are assumed to have PV generation. Therefore, all the buses have both solar and load profiles imposed onto them. A net load approach is used and the difference of the load and solar profiles is applied onto the buses. Different solar-to-load ratios

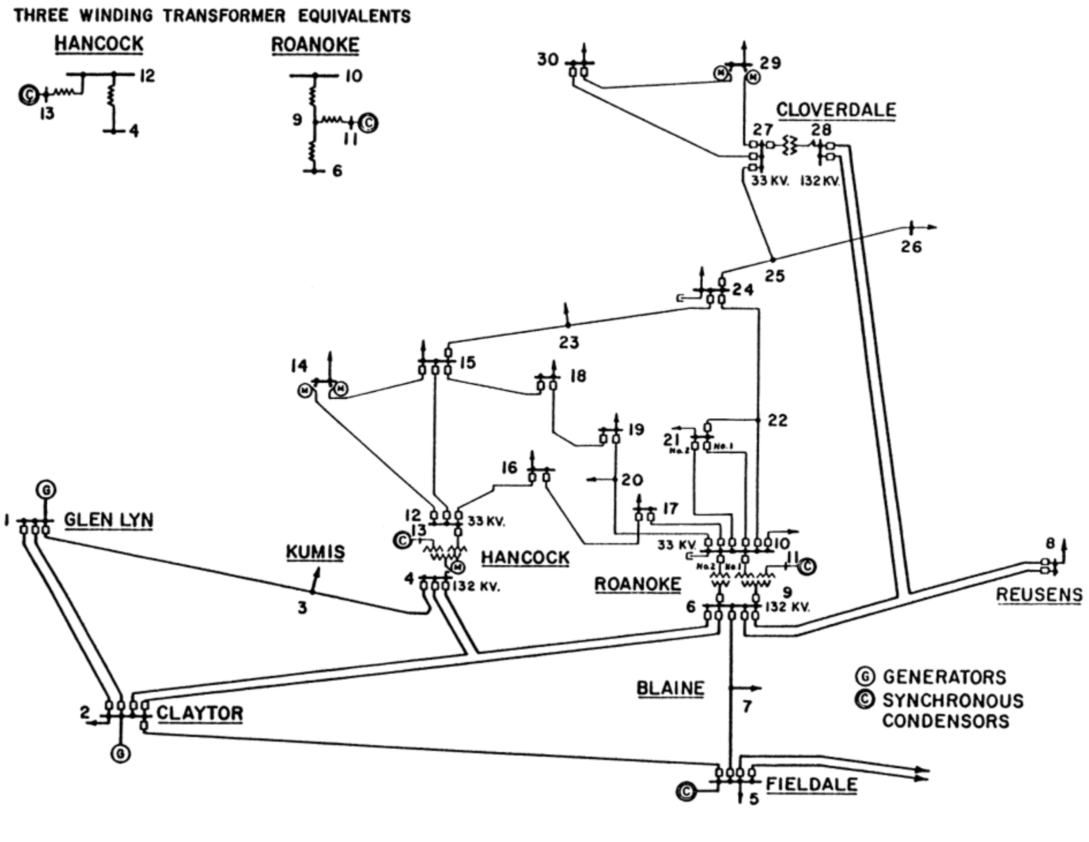

Figure 5.4: IEEE 30-Bus Test Case [106]

are also investigated and the OPF is calculated for each case.

The main focus of this verification study is on the cost of the objective function of the OPF and the voltage angles of the buses. Operating cost of a power system is one of the main variables to track from a power utilities' point of view and the validity of the generated scenarios can be verified by comparing the resultant cost of the OPF with the cost of the real scenarios' OPF . This provides a real-world indicator of how closely the GAN is able to replicate the profiles available in the dataset.

The second variable of note is the voltage angles of the buses. Since the solar generation and load demand profiles represent real power into or out of the buses, voltage angles provide a direct comparison of the real and synthesised profiles. Specifically, the probability density functions (PDFs) of the voltage angles across the buses are investigated. Inferences and comparison as to which set of profiles have an average higher load/lower solar generation, and vice-versa, can be made.

The comparison of correlated scenarios with uncorrelated ones is a challenging task. A simple comparison of the OPF results between the two sets of generated scenarios with each other does not provide us with a baseline of real scenarios to contrast against. The choice of a suitable baseline is further complicated by the fact that since the model is multivariate, comparing separately generated, uncorrelated scenarios with a baseline made up of correlated solar and load profiles is going to bias the comparison in favor of correlated scenarios. Hence, a separate baseline is required for uncorrelated scenarios to compare against. This baseline consists of randomly sampled days from training data of the univariate GANs, separately for solar and load, and combining them to make a baseline consisting of uncorrelated profiles from the original dataset. This provides us with a comparison of how the (main) GAN is preforming with respect to its training data while also providing us with a method of comparison between correlated and uncorrelated scenarios.

The OPF is run 30 times with different generated profiles applied onto the buses to ensure statistical soundness. Both the uncorrelated and correlated baselines are kept constant for these 30 iterations. The specific steps that were followed are enumerated below:

1. Generate 900 synthetic profiles (for a total of 30 iterations) from each of the three GANs (correlated, uncorrelated solar, uncorrelated load).

2. Randomly pick 30 profiles from the real data for both correlated and uncorrelated baselines.

3. Apply both the real and generated profiles onto the IEEE 30-bus system and run the OPFs.

4. Compare the OPF cost and the voltage angle distribution.

The OPF is run in MATLAB [107] using the MATPOWER [108] module.

5.2.1 Wasserstein Distance

To compare the PDF of the voltage angle distributions arising from the real and generated profiles a metric is required. The metric that was chosen was the Wasserstein distance, which is also called the Earth Mover's Distance (EMD). The Wasserstein distance between two distributions $\mathbb{P}_x$ and $\mathbb{P}_y$ is the minimum cost to transform the distribution $\mathbb{P}_x$ into the second distribution $\mathbb{P}_y$.

Mathematically, Wasserstein distance can be expressed as,

$$W(\mathbb{P}_x, \mathbb{P}_y) = \inf_{\gamma \in \Pi(\mathbb{P}_x, \mathbb{P}_y)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||] \tag{5.3}$$

where $\Pi(\mathbb{P}_x, \mathbb{P}_y)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are $\mathbb{P}_x$ and $\mathbb{P}_y$.

Wasserstein distance essentially measures how much mass from one distribution must be changed to convert it into the other distribution, and hence the name EMD (Earth Mover's Distance).

Chapter 6

RESULTS

6.1 Generated Profiles

The seasonal correlated GAN was trained using the methodology described in Chapter 4. Some of the generated profiles for each season, along with similar samples from the real dataset are given below.

Fig. 6.1 shows three profiles for normal summer days. Normal summer days are characterized by high loads due to more consumption of electricity for cooling residential, industrial, and corporate/commercial infrastructure in Arizona. Hence, summer shows the highest peak load of all the seasons.

Solar PV production during the summer season is also generally high in comparison to the other seasons, with the profiles demonstrating high plateaus during the day. For a normal summer day, there are no sharp peaks or nadirs in both solar generation and load demand.

The generated profiles, shown in blue, also follow the same patterns to a very high degree of similarity. This shows that the recurrent GAN was able to pick up and train on the labeled sequences in a satisfactory manner.

Abnormal summer days are shown in Fig. 6.2. These profiles demonstrate abrupt changes in solar PV production. There are cases where the solar peaks in the early hours in the morning and then tapers down to a minimum, or the peaks are in the latter part of the day just before sunset and the rest of the day demonstrates low PV generation. Random dips in solar production are also observed in some profiles. These abnormalities can be due to weather, e.g., cloudy days or rain, accounting for

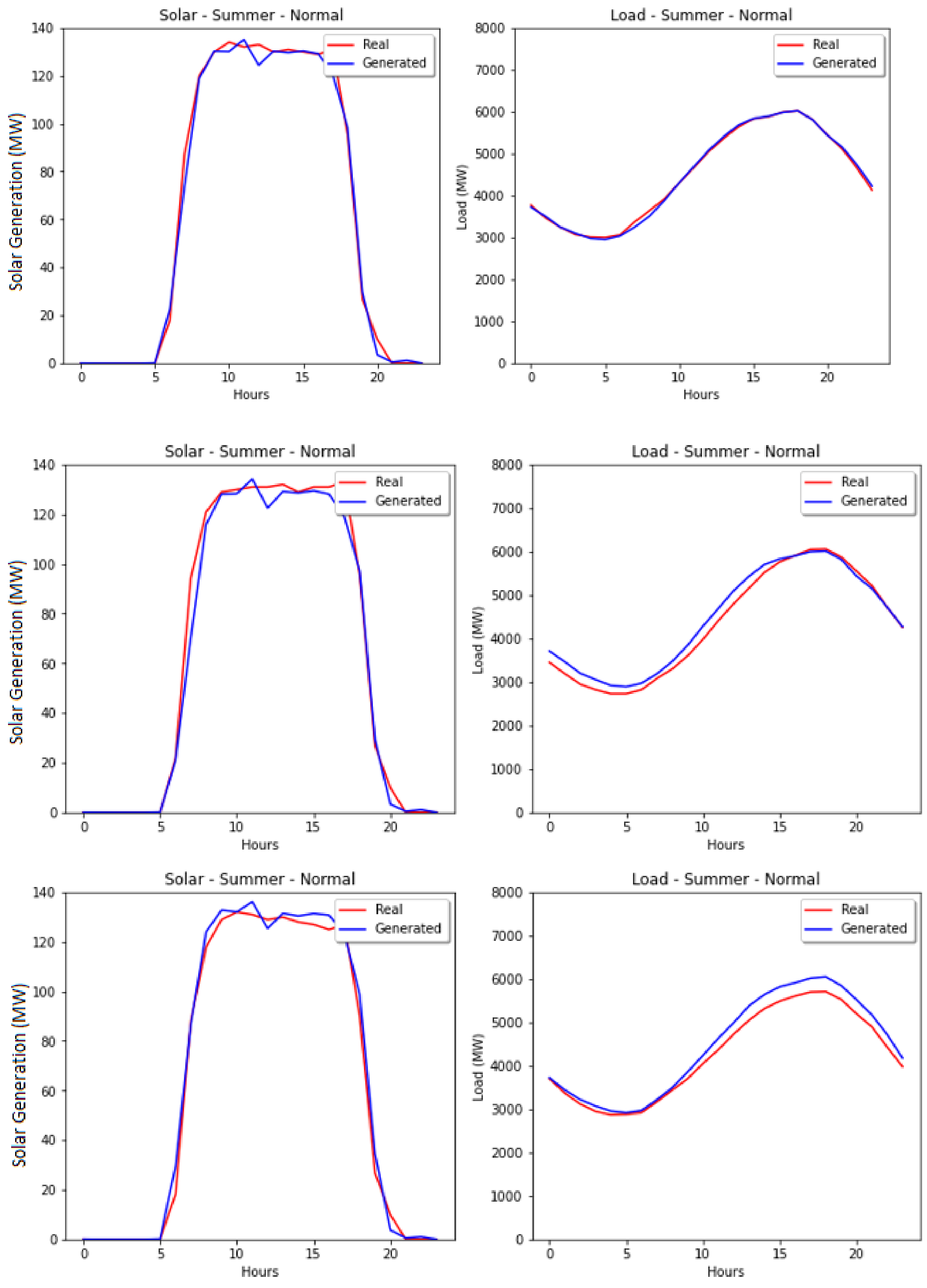

Figure 6.1: Normal Summer Profiles

low solar insolation. Nevertheless, overall peak solar generation is still high compared to abnormal profiles of other seasons.

It is clearly observed that most of the abnormalities in the correlated profiles for the normal summer season are due to solar. Load profiles on the whole display much less variations. Although there are days when the peak load demand is less than in a normal summer day, the corresponding abnormal solar generation provides reasons for a lower peak load. For example, on a day with low solar insolation, the peak temperature of the geographical area will be lower than a summer day with high insolation. This will result in lower load demand as less power will be utilized for cooling. This correlation is important because both the variables are related in a non-direct manner, which can be seen in the profiles.

The generated profiles for abnormal days also closely match the real profiles from the SRP dataset. Even though the sequences show much higher variability, the GAN is able to reproduce the fluctuations in all the cases.

Fig. 6.3 shows the real and generated profiles for normal shoulder days. It can be inferred from the figure that solar production for the shoulder season matches the summer season. The solar insolation remains the same for these months and since the PV power output is primarily dependent on the amount of sunlight incident on the solar panels, the profiles are similar to the summer months.

Although the solar profiles remain the same, a stark difference can be noticed in the load demand. In many instances, the load demand is considerably less than during the summer months. Since the weather is less hot during the shoulder months, when the transition to/from winter is taking place, the load also drops due to reduced need for cooling of residential and industrial setups. It was also noted that for the days which are near the end of the summer season, the peak load demand is high and it gets progressively lower as the weather changes to winter.

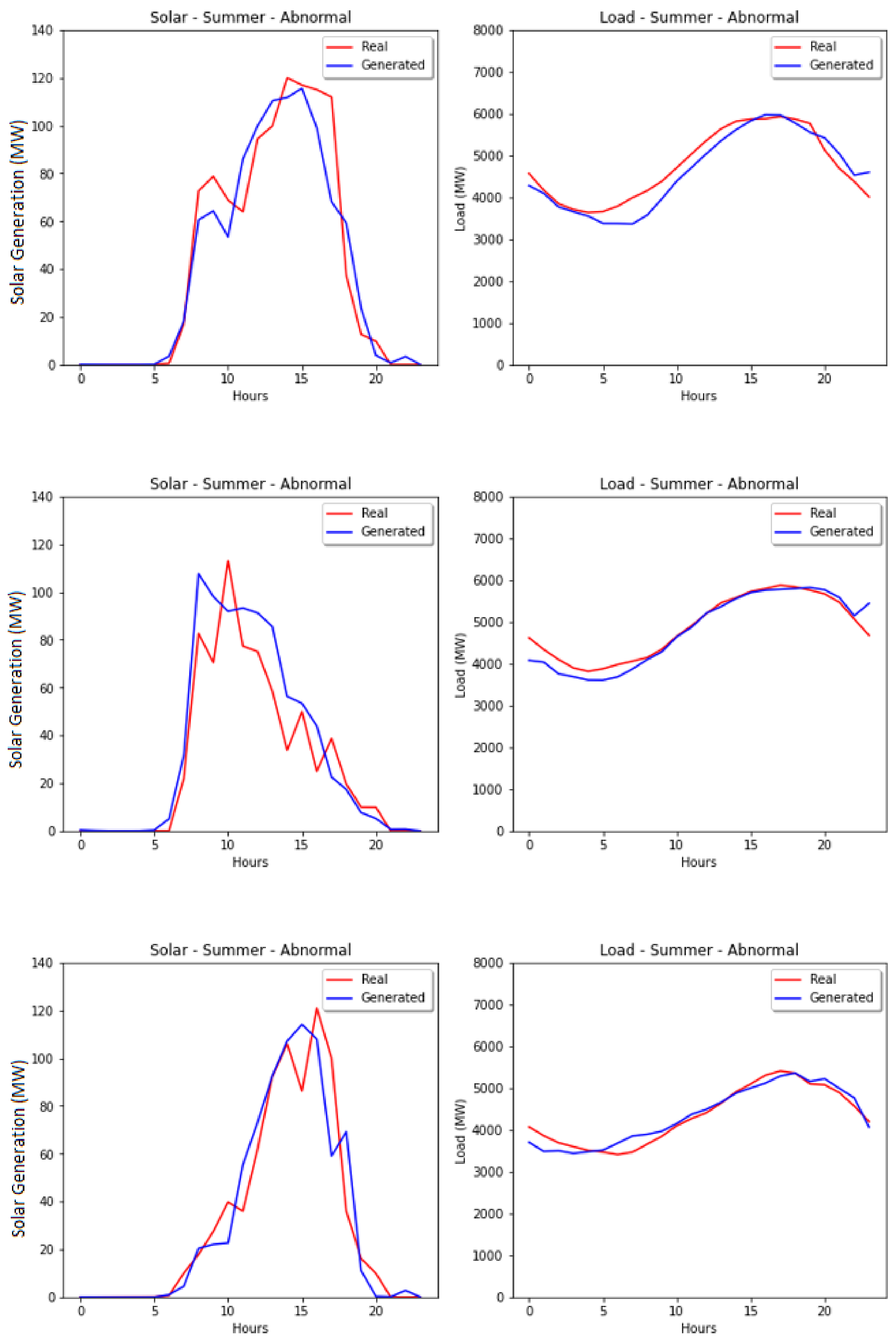

Figure 6.2: Abnormal Summer Profiles

The shoulder season GAN, similar to the GAN for the summer season, is able to follow the profiles in the dataset closely and is able to pick up on the differences in the load sequences as the months change.

Abnormal shoulder days are shown in Fig. 6.4. Most of these cases depict very low load demand. Solar production in many of these cases is also quite low. These abnormal days are representative of dates which are closer to the winter months. Hence, the overall low load demand and low PV production is due to abnormally low solar insolation.

There are also some profiles that are similar to abnormal summer days. This again is due to the proximity of these days to the months of summer. These days demonstrate sharp peaks and nadirs with a high average PV production. Weather phenomena such as intermittent clouds can be the reason behind this behavior.

Despite this great variability in solar generation profiles, the synthesised sequences represent all the different variations present in the original dataset.

Figure 6.5 portrays normal winter profiles. The load demand is low for the length of the winter season, owing to lesser need for cooling. Since the dataset is from the US Southwest, the need for heating in the winter months is also minimal as the region enjoys mild winters. This is depicted in the generally low load profiles.

The solar generation profiles also depict lower levels of power generation. Winter months are characterized by cloudy weather and this affects solar insolation. Furthermore, since winter days are shorter than summer, the solar plateaus are thinner and the average power output throughout the day is considerably lower.

Interestingly enough many solar profiles in the original dataset demonstrate slightly lower solar production during midday. The GAN also captures this and generates similar sequences.

Finally, abnormal winter profiles are shown in Fig. 6.6. Most of these profiles

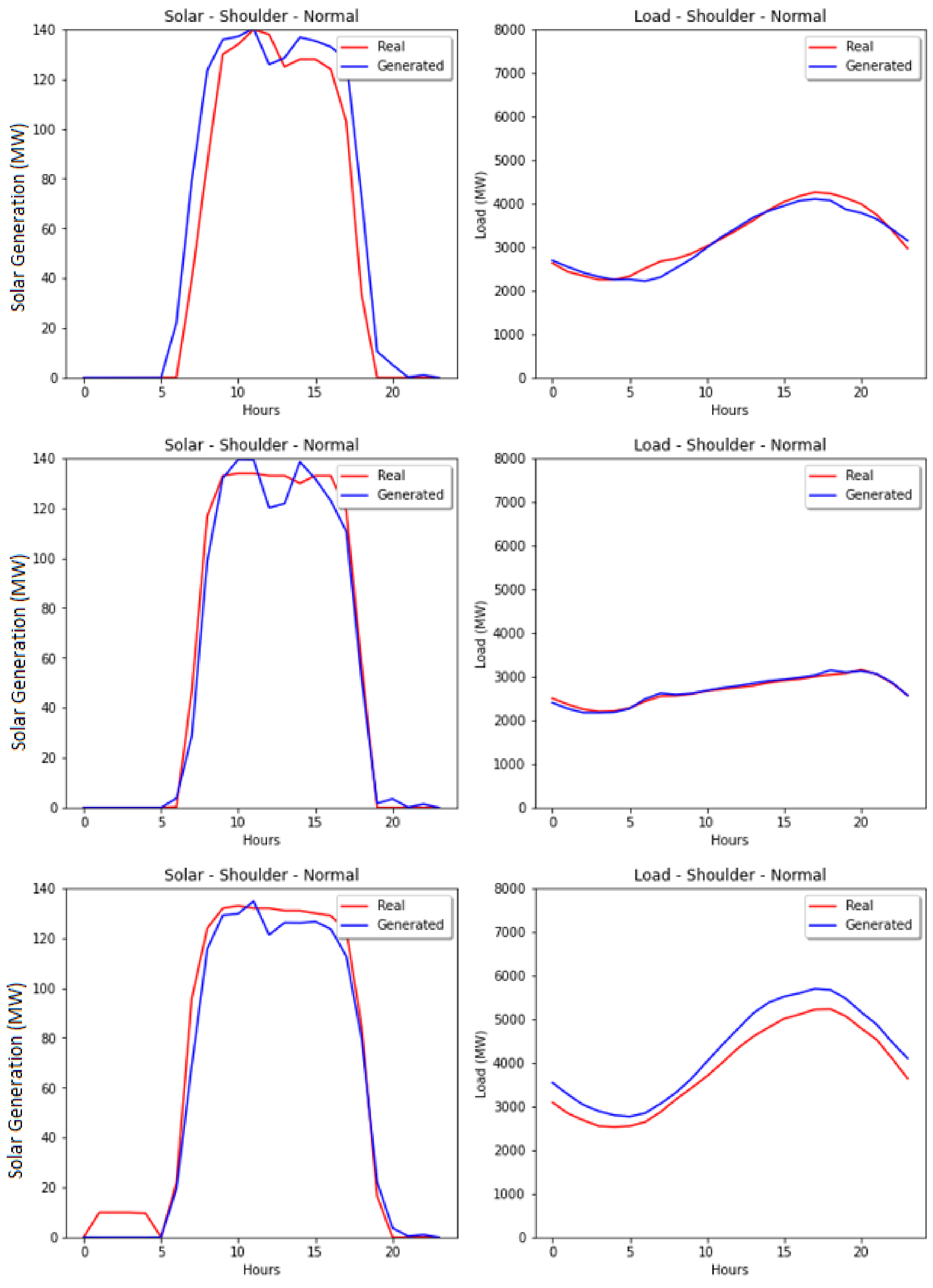

Figure 6.3: Normal Shoulder Profiles

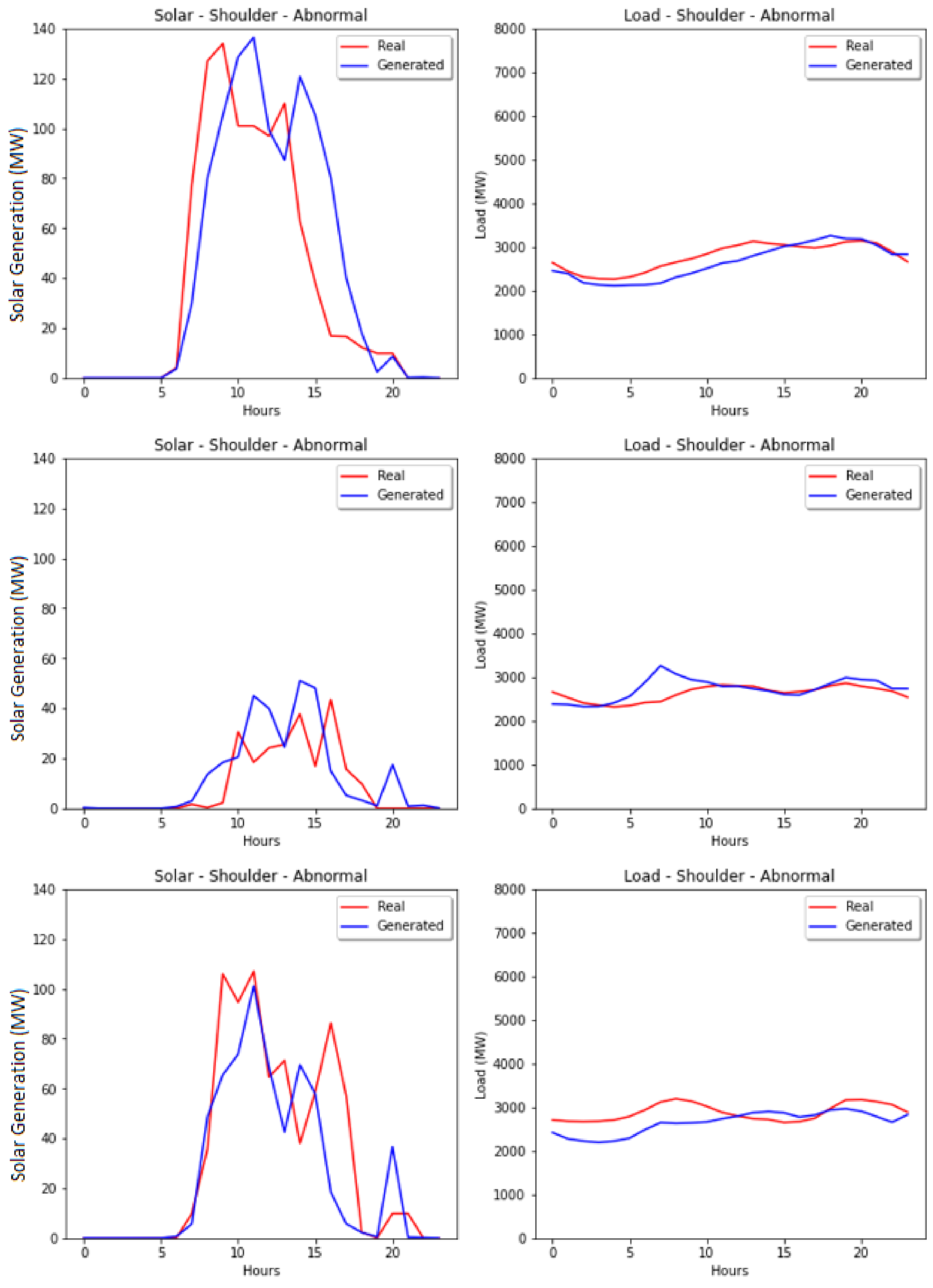

Figure 6.4: Abnormal Shoulder Profiles

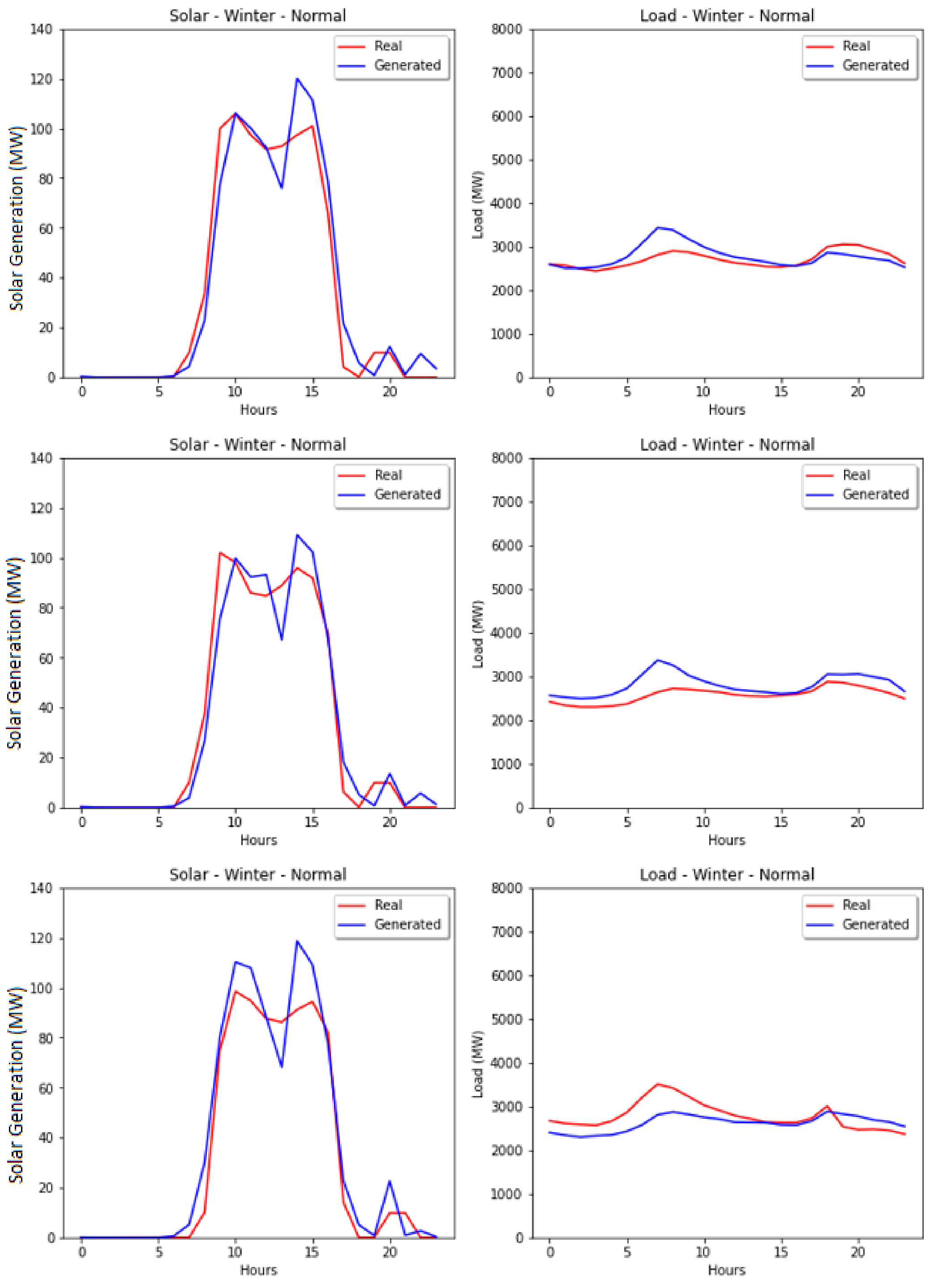

Figure 6.5: Normal Winter Profiles

have extremely low solar PV production throughout the day. Some days, however, show very sharp, narrow peaks. This behavior is indicative of mostly cloudy weather during the day, with the peaks representative of the moments when the clouds lessen and the solar insolation increases.

Here as well, the GAN demonstrates the ability to follow the temporal trends of both solar production and load demand well despite the great variability in the scenarios.

6.2 OPF Results

The generated profiles are applied onto the IEEE 30-bus system in the manner outlined in Chapter 5.2. 900 synthetic profiles are generated for a total of 30 iterations on this system to ensure statistical validity. The OPF cost and the voltage angle distributions of the correlated scenarios vs. uncorrelated scenarios are investigated. The OPFs were calculated for two different solar-to-load ratios: 0.6, an optimistic solar PV production case, and 1.2, representing a futuristic scenario where solar production overtakes the load demand.

The cost of generation of power is an important variable that a power utility monitors. Any generated scenarios must match the cost pattern of the real data to be accepted as a valid replacement. The following sections describe the OPF cost of both the normal and the abnormal scenarios for all three seasons, and for the different solar-to-load ratios.

The voltage angles of the 30 buses of the IEEE 30-bus system are also analyzed. Voltage angles directly relate to real power output of a system and the flow of power towards or away from a bus can be inferred from the voltage angle of that bus. The PDFs of the voltage angles are plotted and Wasserstein distance is used as a metric to compute the distance between them. The Wasserstein distance is computed between

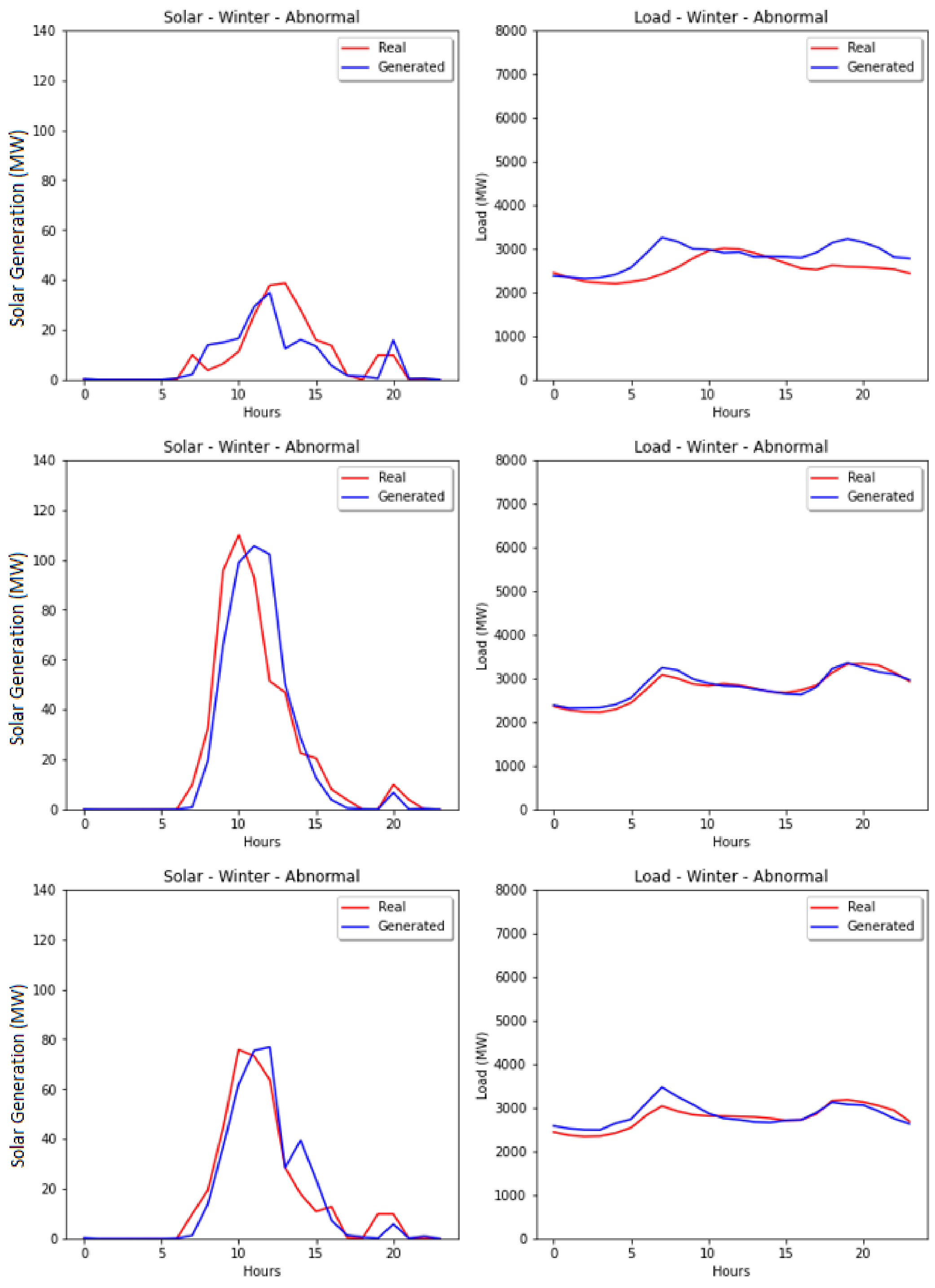

Figure 6.6: Abnormal Winter Profiles

each of the generated and base case PDFs for the whole day and averaged over 24 hours. This process is repeated for all the seasons.

Note that the Wasserstein distance is a direct measure of the distance of the profile from its base case. The smaller the distance, the more closely the PDFs of the voltage angles match, and the closer the generated solar and load profiles are to the real ones. Conversely, higher values of the Wasserstein distance between the PDFs mean that the generated solar and load profiles are not close enough to the real ones in the baseline to result in similar OPF output.

Summer - Abnormal Scenarios

Fig. 6.7 and 6.8 show PDFs of the voltage angle for a correlated and an uncorrelated abnormal summer day, respectively, with a solar-to-load ratio of 0.6, for hours 7, 12, and 17. Greater overlap between the voltage angle PDFs denote higher similarity of the generated correlated scenarios to the base case.

Fig. 6.9 and 6.10 show the average OPF cost per hour for all the 30 iterations. Meanwhile, Fig. 6.11 and 6.12 show the hourly cost averaged over the 30 iterations, for different solar-to-load ratios. From the figures, it can be gathered that correlated scenarios are much closer to their baselines than uncorrelated scenarios on average, for both solar-to-load ratios. Uncorrelated scenarios tend to overestimate solar generation and this brings down the overall OPF costs.

The lower cost is further exacerbated by the fact that uncorrelated scenarios underestimate load demand as well. This is due to the fact that a GAN trained only for abnormal load is trained on data in one variable only. Abnormal load data will mostly have days where load demand was uncharacteristically low for a summer day. This behaviour, coupled with overestimation of solar generation results in lower overall OPF costs.

Lastly, note that the baseline for uncorrelated scenarios also shows lower cost overall apart from a few hours (8 AM to 1 PM). These are the points where the uncorrelated solar profiles actually underestimate solar PV generation, unlike the rest of the day.

The Wasserstein distance plots in Fig. 6.13 and 6.14 also back up the results of the OPF by demonstrating that the difference between the correlated scenarios and their baseline is lower than uncorrelated scenarios. The hourly Wasserstein Distance plot of Fig. 6.15 shows that for some hours (8 AM to 1 PM) the Wasserstein distance of uncorrelated scenarios is very large. These are the hours where the uncorrelated GANs is not able to follow the baseline well. This corroborates with the OPF results where the OPF cost for abnormal scenarios is higher than expected.

Shoulder - Abnormal Scenarios

The PDFs of the voltage angle of the buses for correlated and uncorrelated scenarios are shown in Fig. 6.16 and 6.17, respectively, for abnormal shoulder season for a solar-to-load ratio of 0.6. Greater overlap between the voltage angle PDFs denote higher similarity of the generated correlated scenarios to the base case.

In the shoulder season, as can be inferred from Fig. 6.18 - 6.21, the correlated scenarios are closer to their baselines than uncorrelated scenarios. Furthermore, uncorrelated scenarios show remarkably higher costs than the summer season. A reason for this is because of the way the baseline for the univariate GANs are chosen. For the univariate load GAN, the DTW based classification method to split the data into normal and abnormal classes sorts the load profiles in different ways for correlated and uncorrelated scenarios. For uncorrelated scenarios, abnormal load for the shoulder season are the profiles where the load demand is either very large, where the respective dates are closer to the summer season, or the load demand is very low, where the

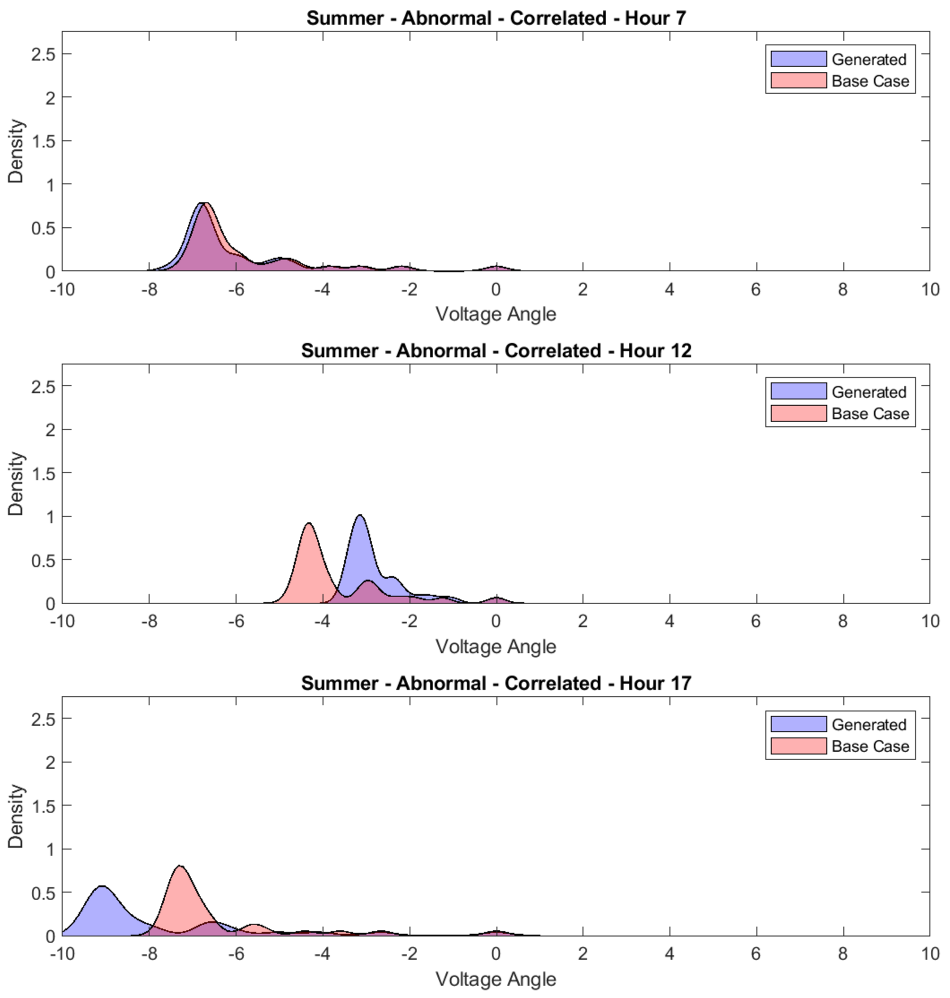

Figure 6.7: Voltage Angle PDF, Summer - Abnormal, Correlated

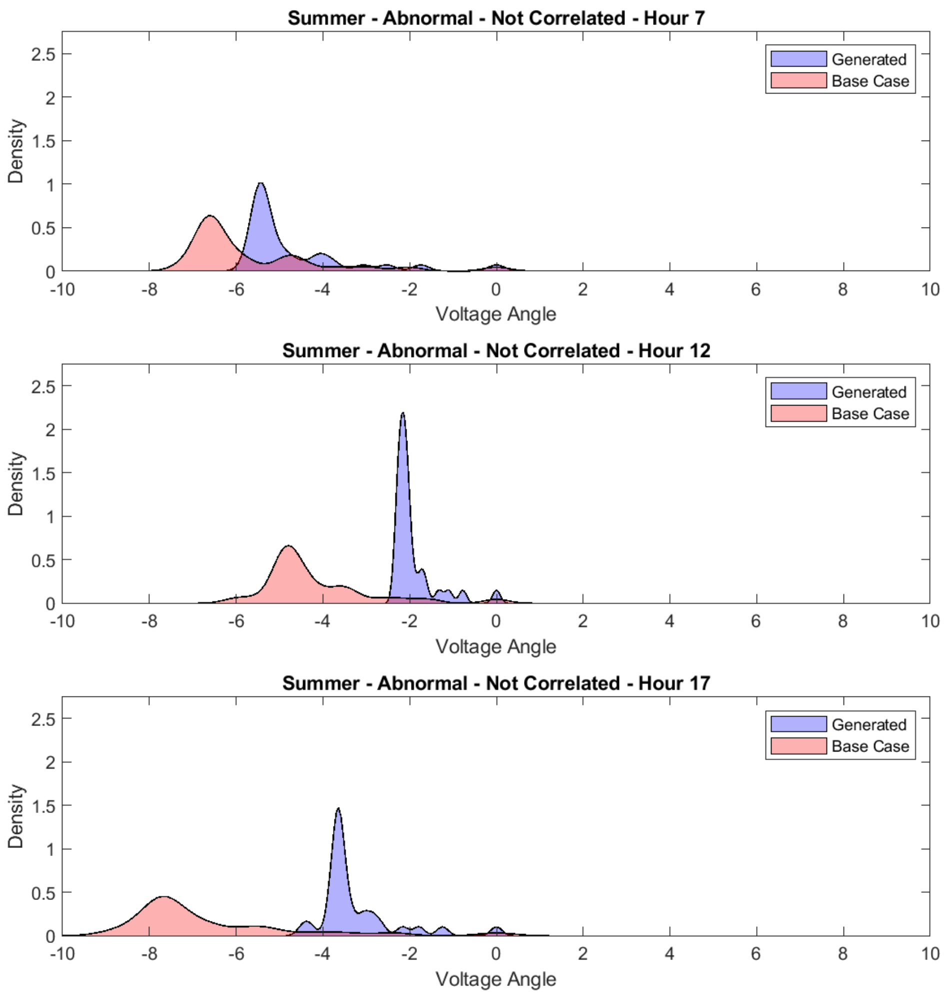

Figure 6.8: Voltage Angle PDF, Summer - Abnormal, Uncorrelated

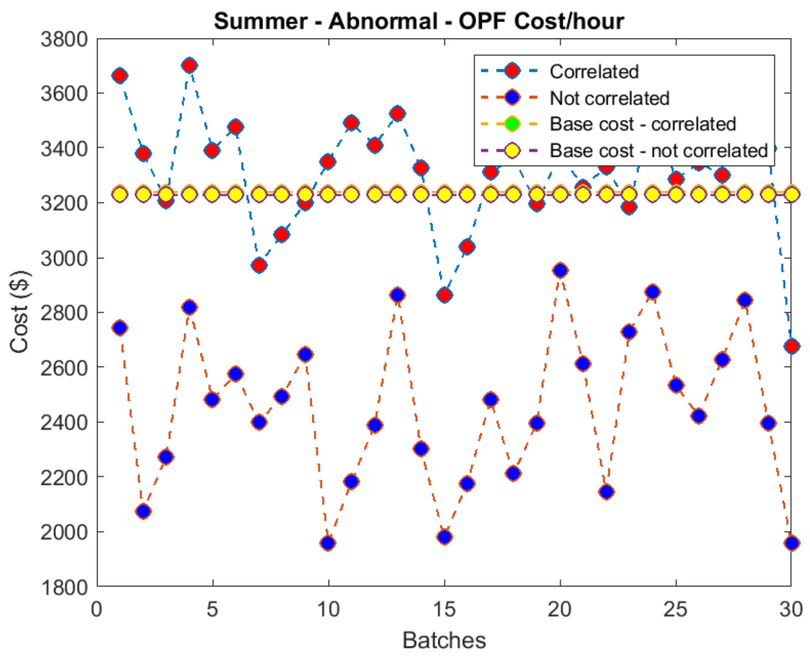

Figure 6.9: Summer Abnormal OPF Cost, Solar-to-Load Ratio = 0.6

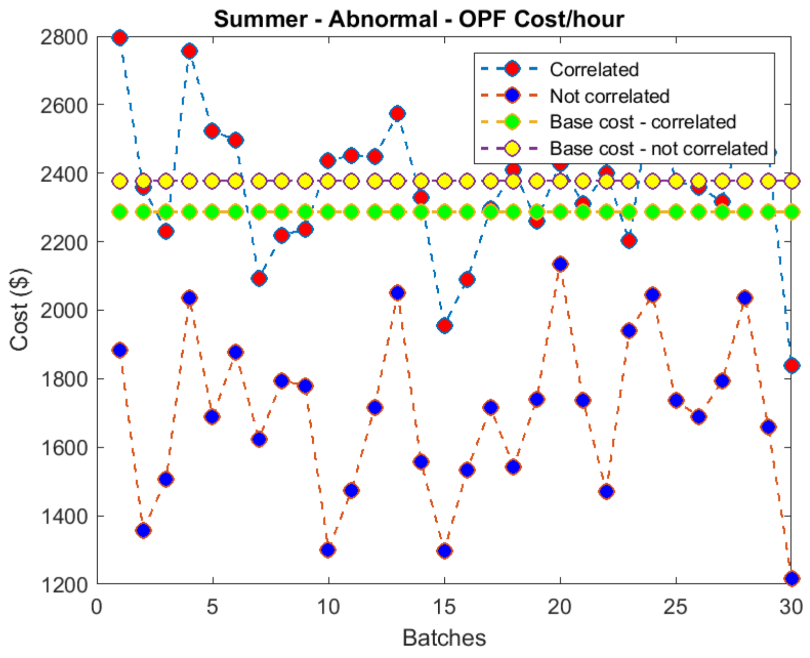

Figure 6.10: Summer Abnormal OPF Cost, Solar-to-Load Ratio = 1.2

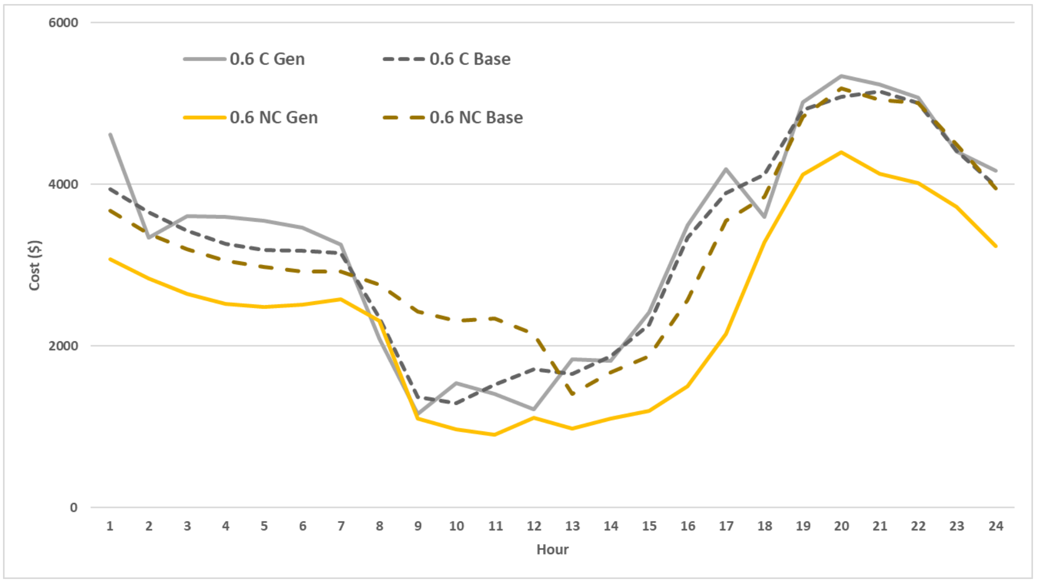

Figure 6.11: Summer Abnormal OPF Cost, Solar-to-Load Ratio = 0.6, Hourly Average

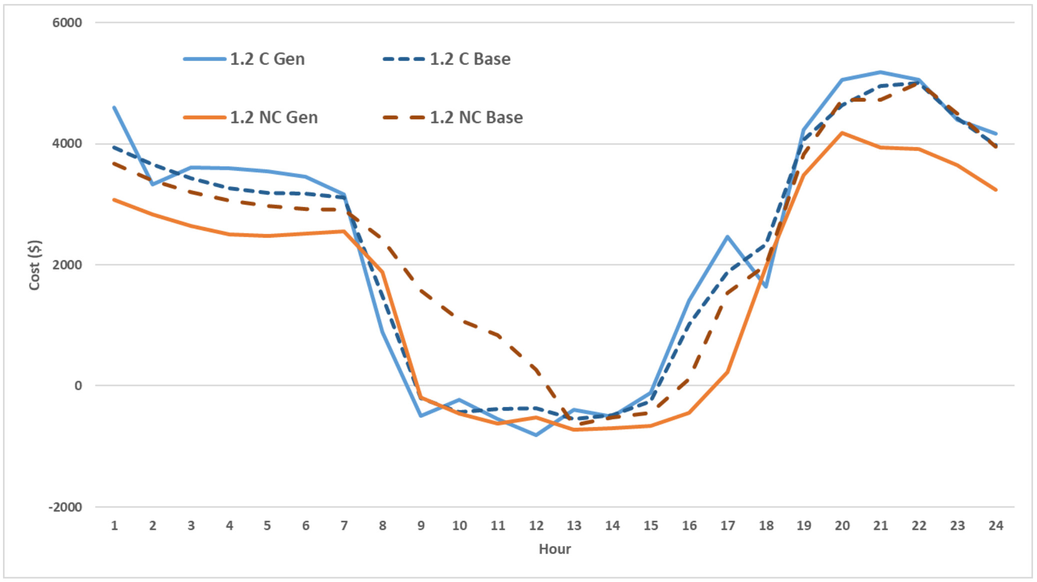

Figure 6.12: Summer Abnormal OPF Cost, Solar-to-Load Ratio = 1.2, Hourly Average

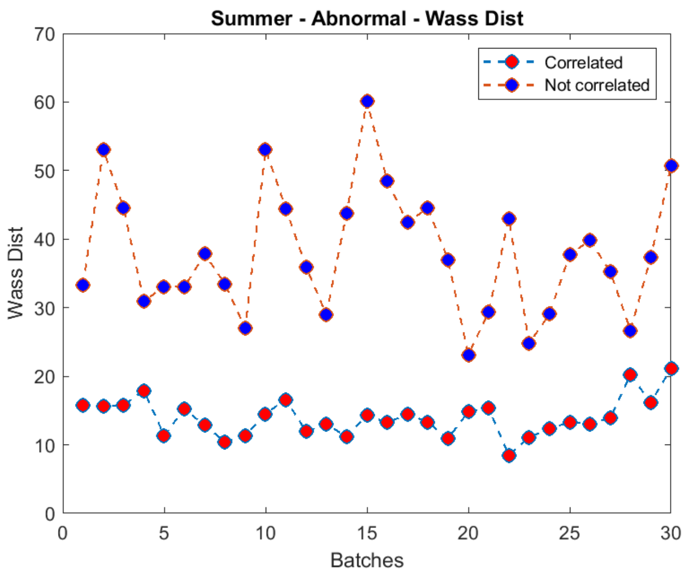

Figure 6.13: Summer Abnormal Wasserstein Distance, Solar-to-Load Ratio = 0.6

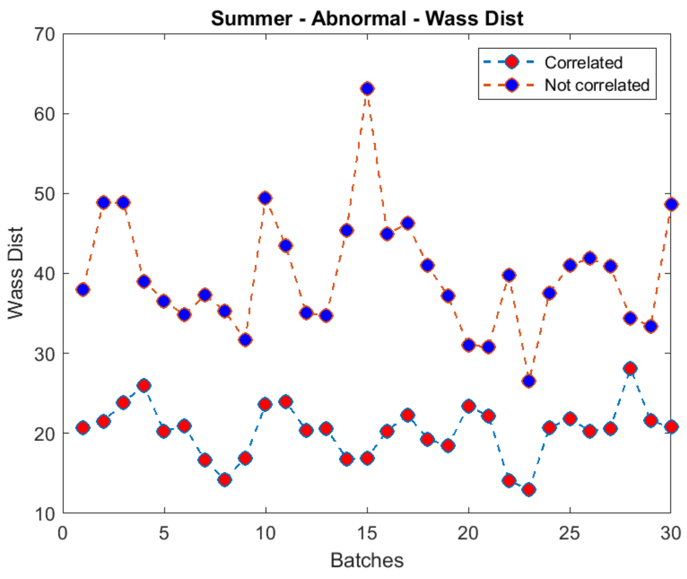

Figure 6.14: Summer Abnormal Wasserstein Distance, Solar-to-Load Ratio = 1.2

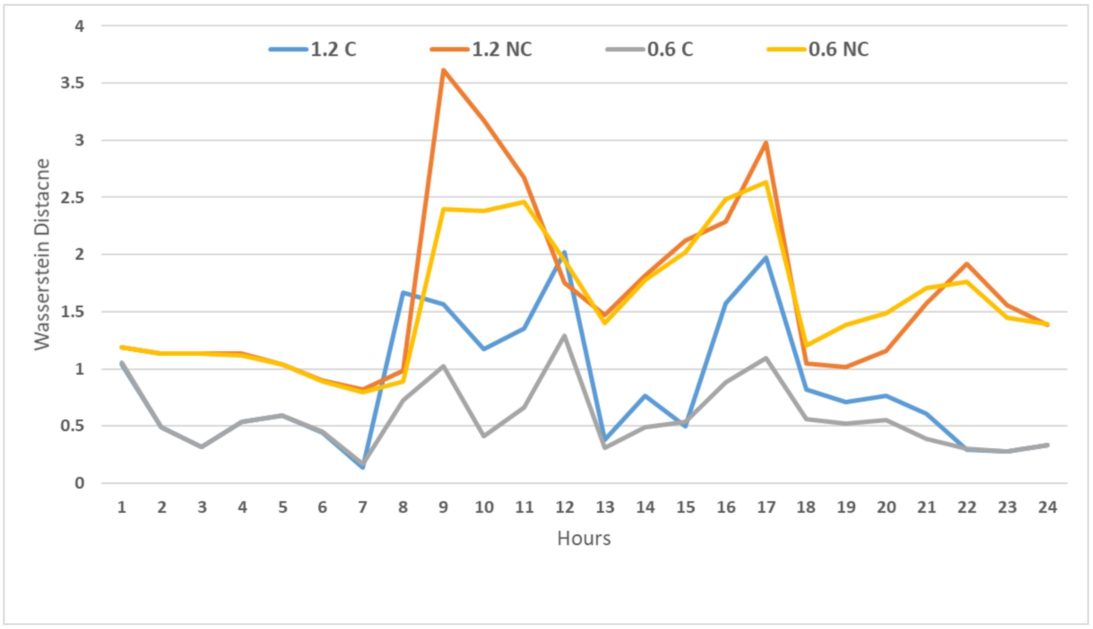

Figure 6.15: Hourly Abnormal Wasserstein Distance for Summer Season

dates are closer to the winter season. It is observed from the SRP dataset that there are more days with high load demand. Therefore, the univariate load GAN, generates more load profiles with high peak loads. This drives the cost of the uncorrelated scenarios higher than its own baseline and much higher than correlated scenarios. This is especially true for the latter part of the day (from 4 p.m. till midnight) where the average load demand in generated uncorrelated scenarios is very high.

The Wasserstein distances, shown in Fig. 6.22, Fig. 6.23, and Fig. 6.24 for the uncorrelated scenarios are also higher than the correlated scenarios. This verifies the results for OPF costs, with the correlated scenarios being closer to their respective baseline than uncorrelated scenarios. This holds true for both solar-to-load ratios of 0.6 and 1.2.

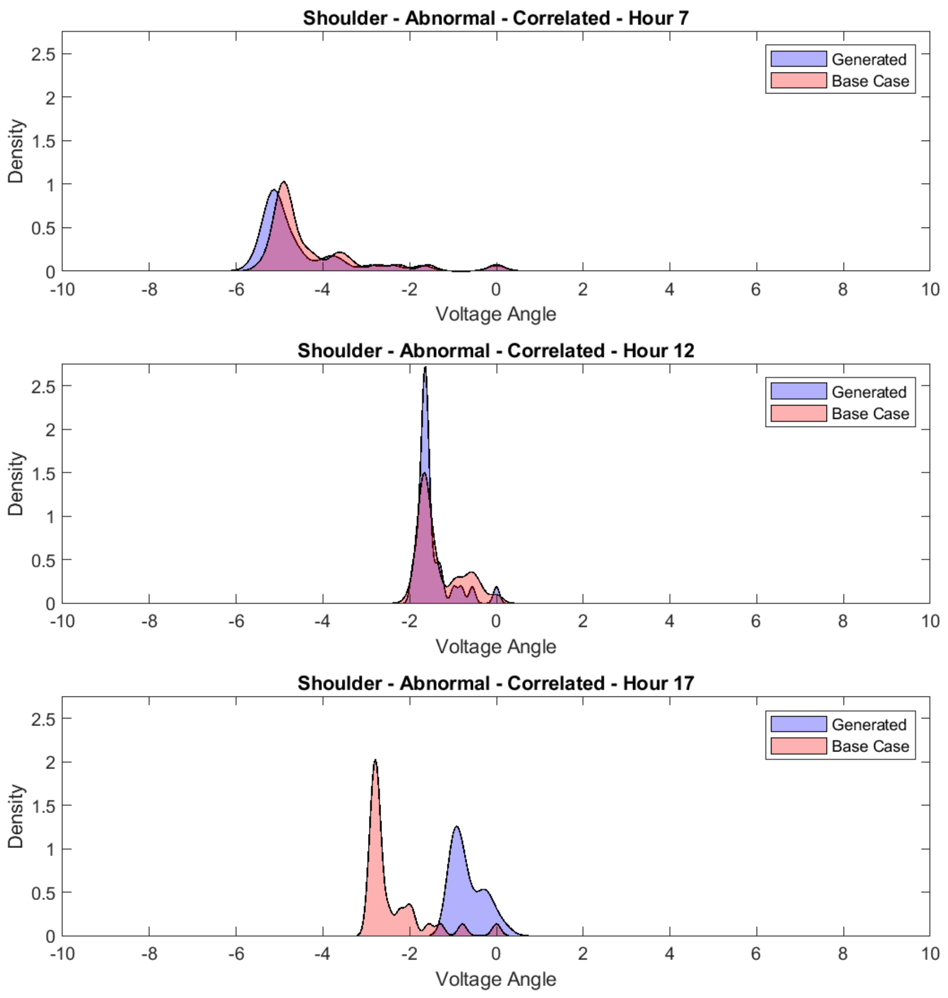

Figure 6.16: Voltage Angle PDF, Shoulder - Abnormal, Correlated

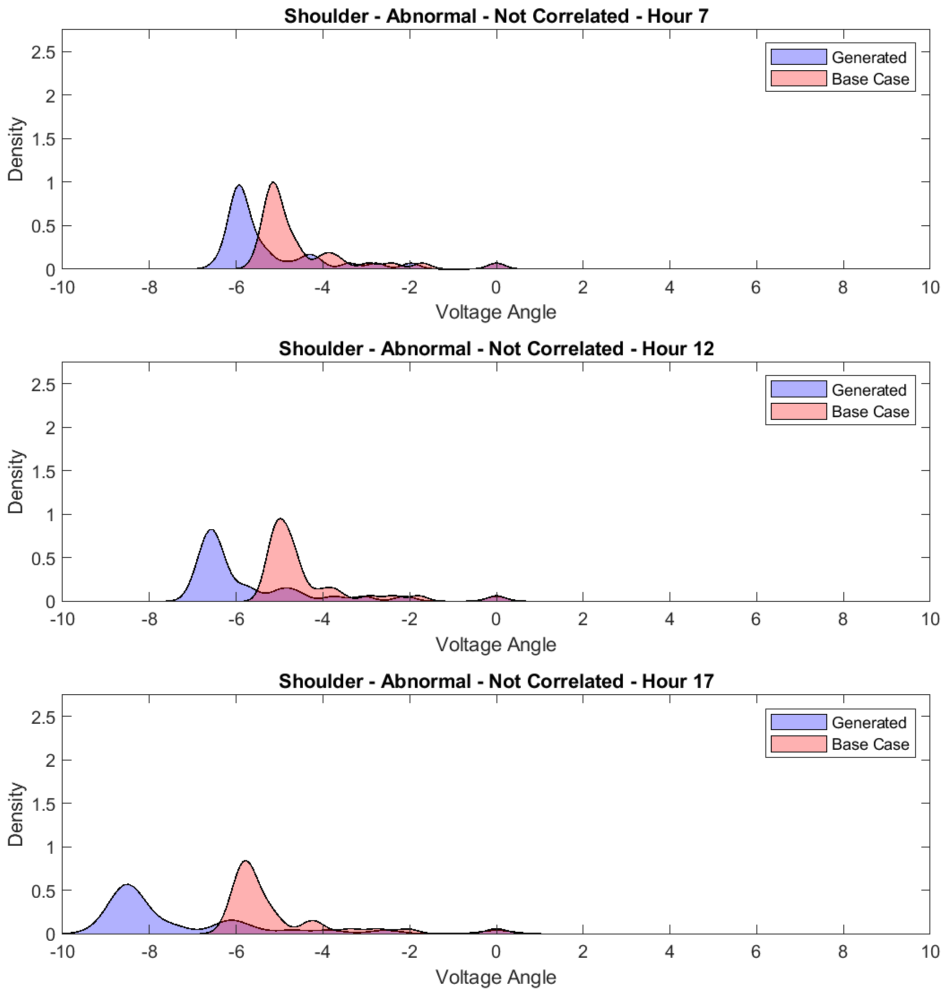

Figure 6.17: Voltage Angle PDF, Shoulder - Abnormal, Uncorrelated

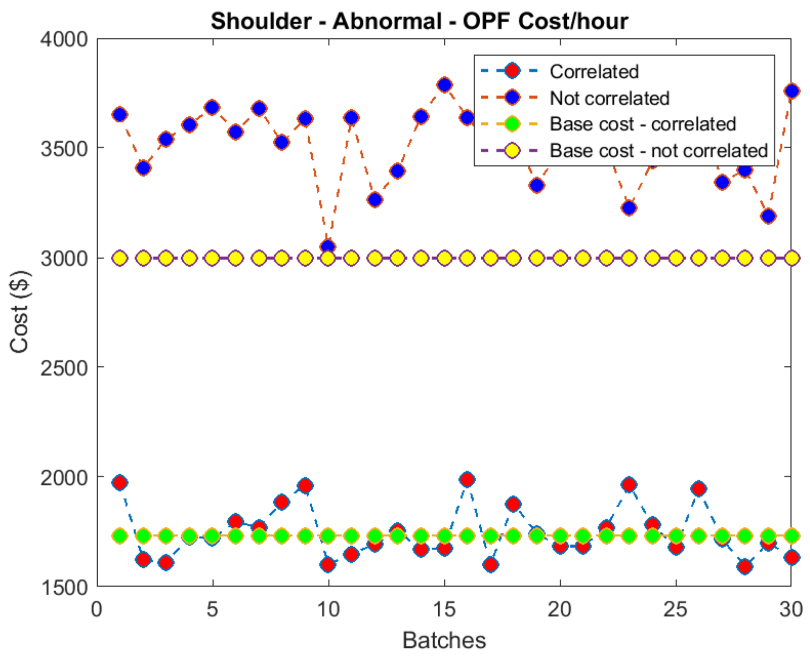

Figure 6.18: Shoulder Abnormal OPF Cost, Solar-to-Load Ratio = 0.6

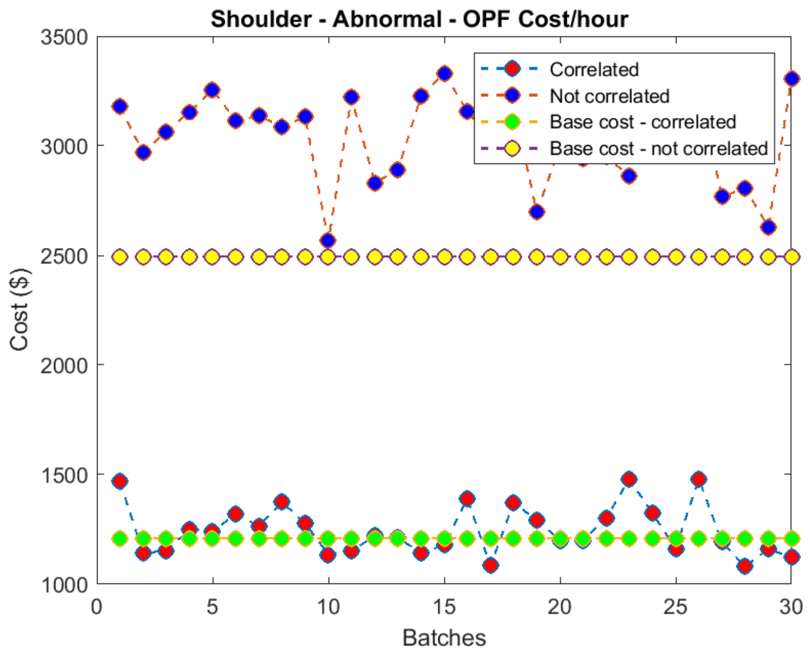

Figure 6.19: Shoulder Abnormal OPF Cost, Solar-to-Load Ratio = 1.2

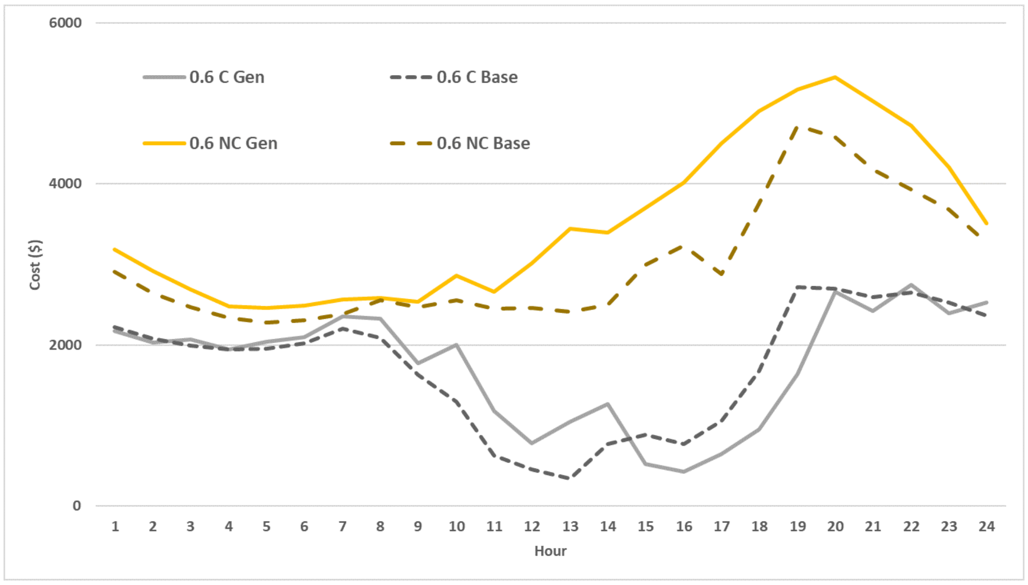

Figure 6.20: Shoulder Abnormal OPF Cost, Solar-to-Load Ratio = 0.6, Hourly Average

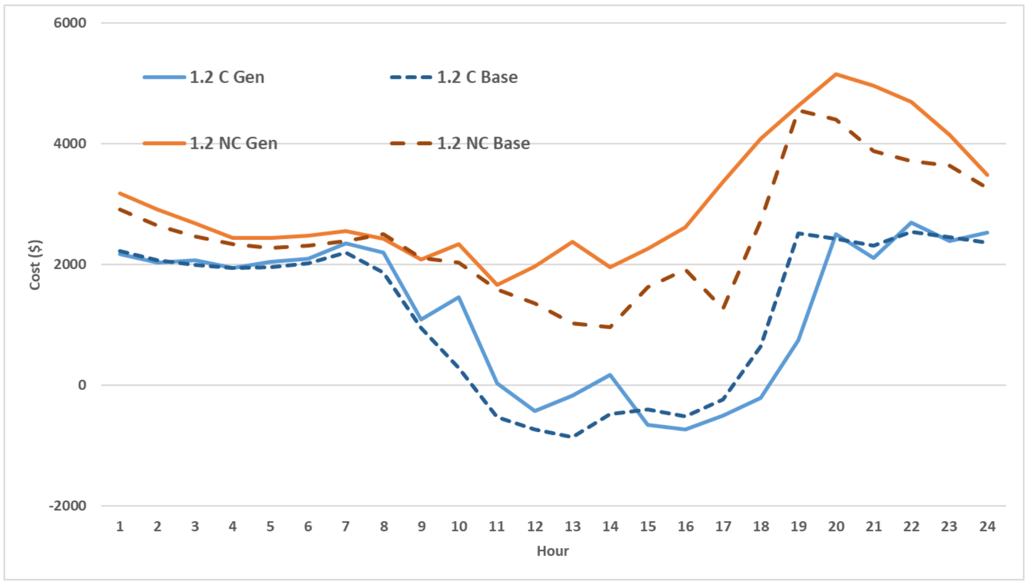

Figure 6.21: Shoulder Abnormal OPF Cost, Solar-to-Load Ratio = 1.2, Hourly Average

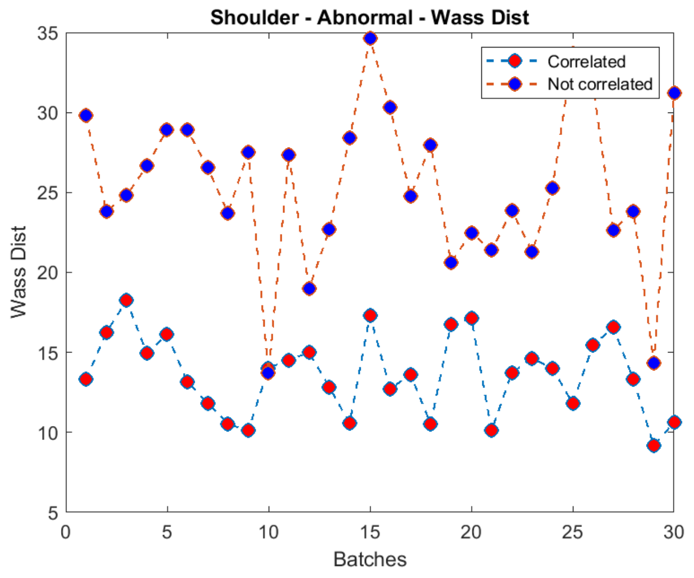

Figure 6.22: Shoulder Abnormal Wasserstein Distance, Solar-to-Load Ratio = 0.6

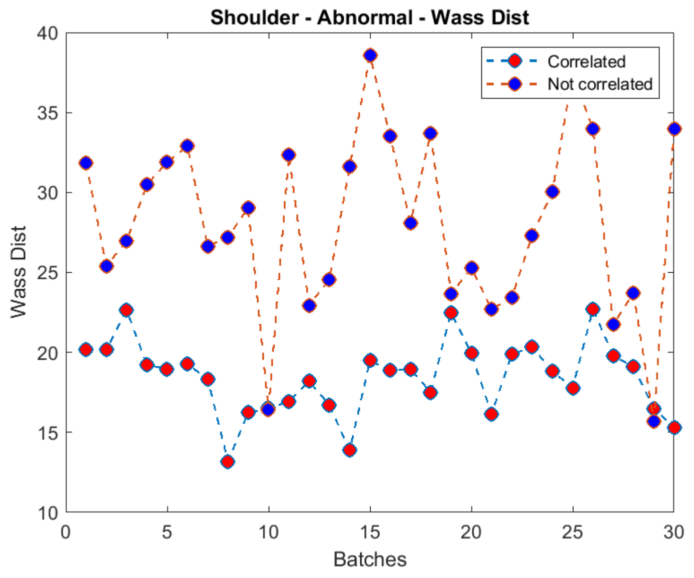

Figure 6.23: Shoulder Abnormal Wasserstein Distance, Solar-to-Load Ratio = 1.2

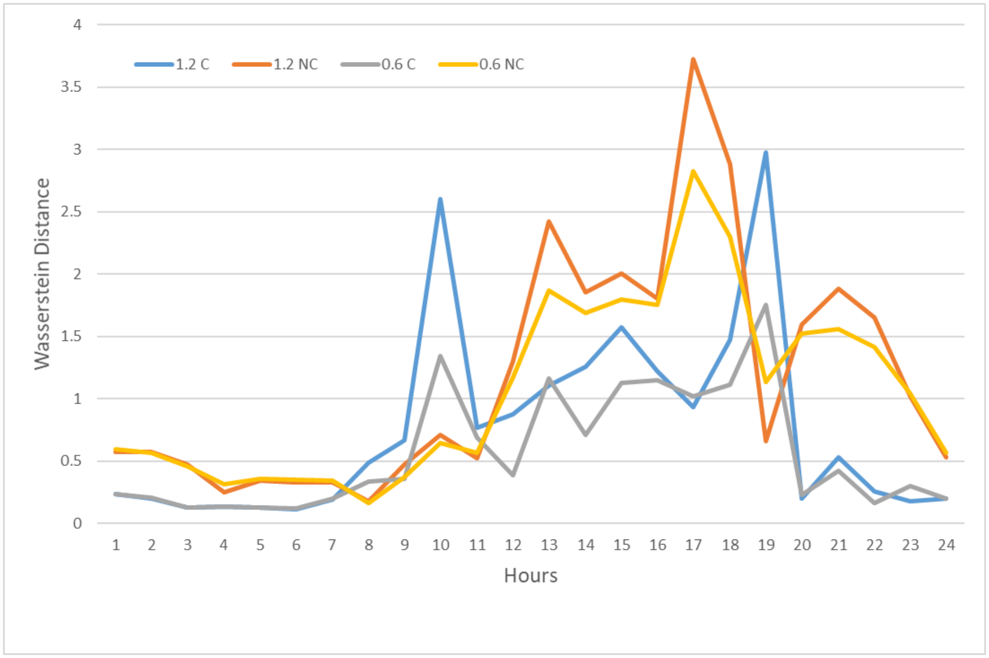

Figure 6.24: Hourly Abnormal Wasserstein Distance for Shoulder Season

Winter - Abnormal Scenarios

For the abnormal winter season, the PDFs of the voltage angle of the buses for abnormal correlated and uncorrelated scenarios are shown in Fig. 6.25 and 6.26, respectively, for a solar-to-load ratio of 0.6. The overlap between the PDFs for correlated scenarios and their baseline is much less than for abnormal summer and shoulder seasons, especially for mid-day hours. Moreover, the average OPF costs for abnormal winter scenarios, depicted in Fig. 6.27 for a solar-to-load ratio of 0.6 and in Fig. 6.28 for a solar to load ratio of 1.2, show that the performance of correlated scenarios is only slightly better than uncorrelated generation, especially for a solar-to-load ratio of 1.2.

This is better understood by observing the hourly winter OPF cost results shown in Fig. 6.29 and Fig. 6.30. It can be observed that correlated scenario generation follows its baseline closely during the start and the end of the day, but deviates from the baseline during the hours of 9 AM to 6 PM. During these hours the generated profiles overestimate the OPF cost. This is due to the generated correlated solar scenarios having lower power output than the ones in the baseline. This can be explained by the fact that most of the abnormal scenarios for winter have very low solar generation throughout the day. Since the correlated GAN is predisposed to generating scenarios like these, the effect of the days with higher solar peaks gets diluted out.

Conversely, uncorrelated solar scenarios, due to the DTW-based classifying method, have a baseline which contains fewer days with higher load than the baseline for correlated scenarios. This unrealistic representation accounts for the higher OPF costs for both uncorrelated baseline and its generated scenarios.

The Wasserstein distances, in Fig. 6.31 and Fig. 6.32 also follow the OPF cost trends, as observed in Fig. 6.33, with peaks between baseline and generated scenarios being observed for some midday hours (9 AM to 6 PM). Nevertheless, for a lower solar-to-load ratio of 0.6, correlated scenarios tend to perform better. For a solar-to-load ratio of 1.2, with higher solar penetration, the advantage of correlated scenarios over uncorrelated ones becomes lower due to difficulty in representing solar profiles appropriately.

Summer - Normal Scenarios

The PDFs of the voltage angle of the buses, for a solar-to-load ratio of 0.6, for normal correlated and uncorrelated scenarios are shown in Fig. 6.34 and 6.35, respectively. It can be observed from the figures that the performance of correlated scenarios is not much better than uncorrelated scenario generation. Similarly, under normal conditions, the cost difference between normal and generated scenarios is minimal (Fig. 6.36 - Fig. 6.41). However, uncorrelated scenarios do seem to perform better than

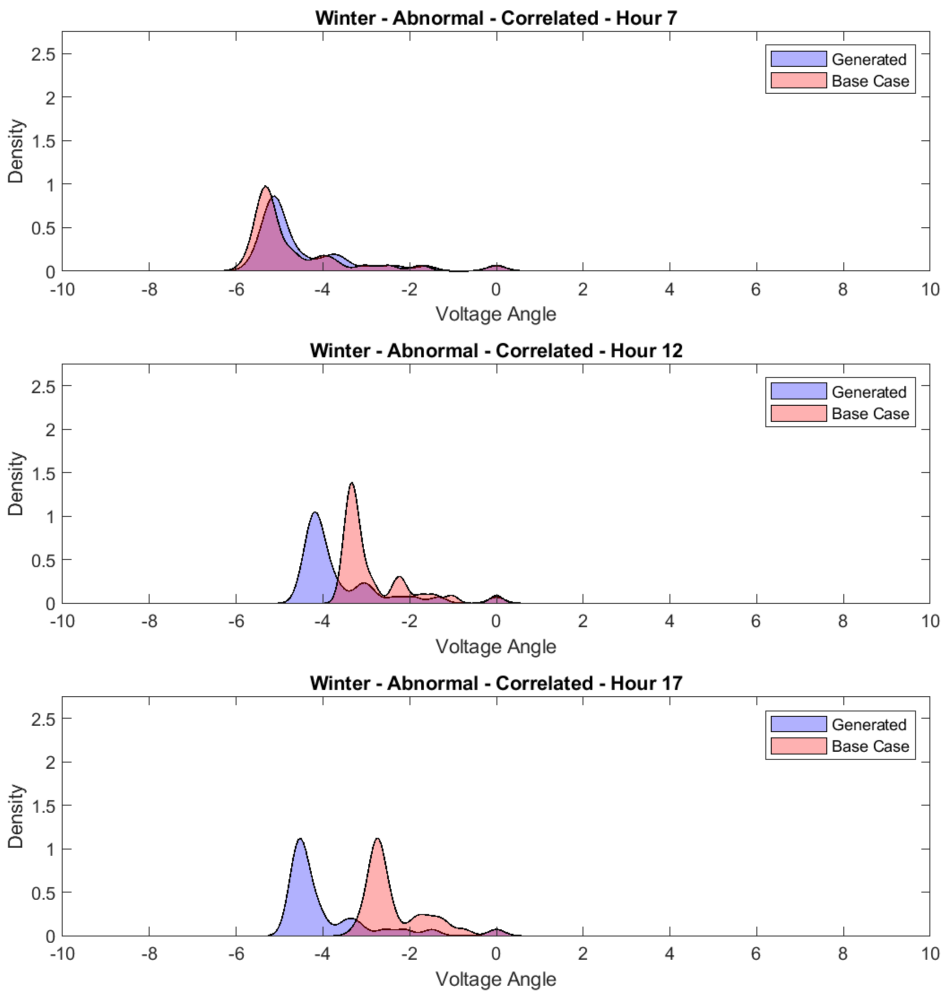

Figure 6.25: Voltage Angle PDF, Winter - Abnormal, Correlated

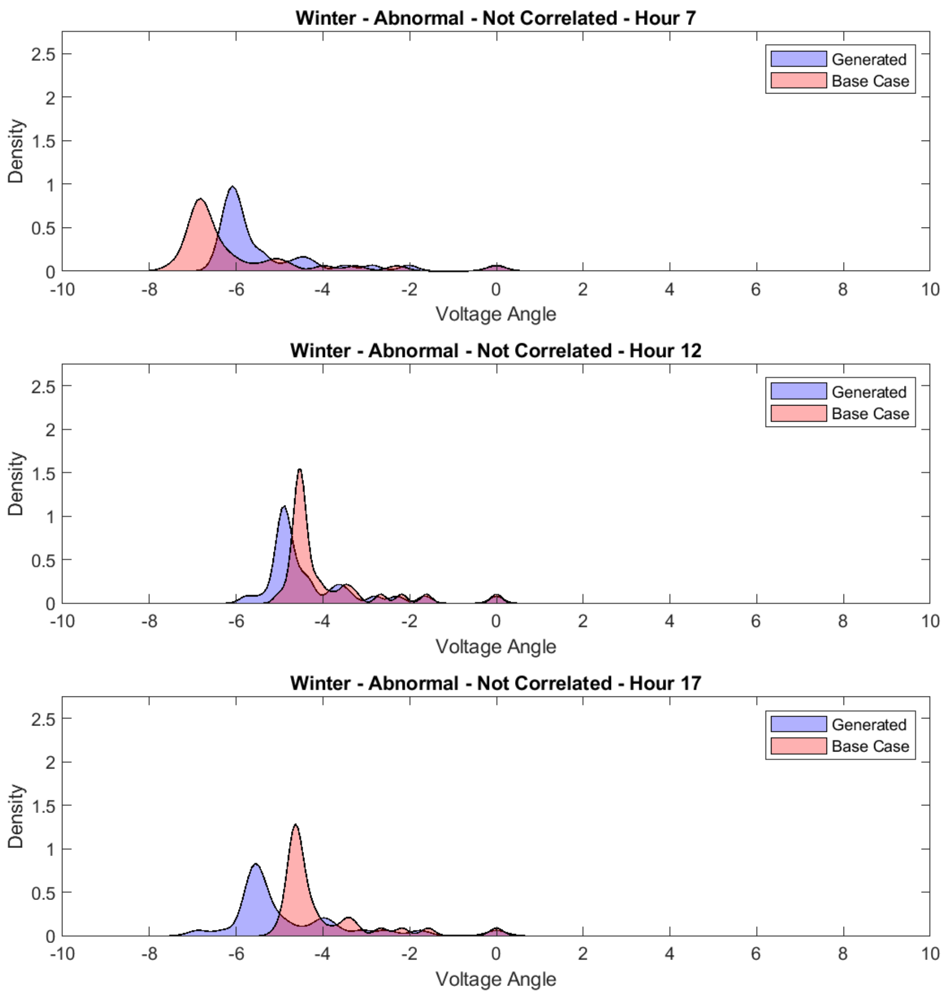

Figure 6.26: Voltage Angle PDF, Winter - Abnormal, Uncorrelated

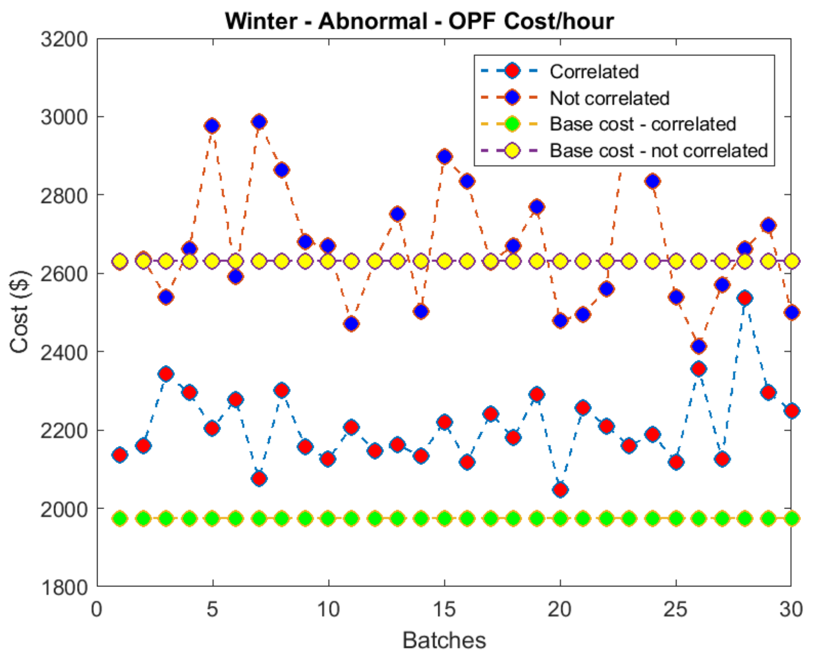

Figure 6.27: Winter Abnormal OPF Cost, Solar-to-Load Ratio = 0.6

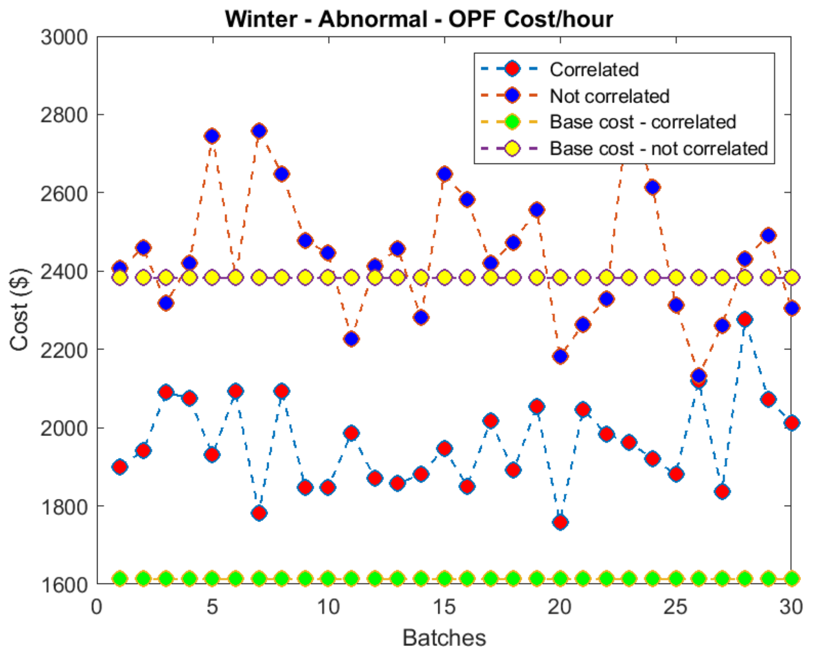

Figure 6.28: Winter Abnormal OPF Cost, Solar-to-Load Ratio = 1.2

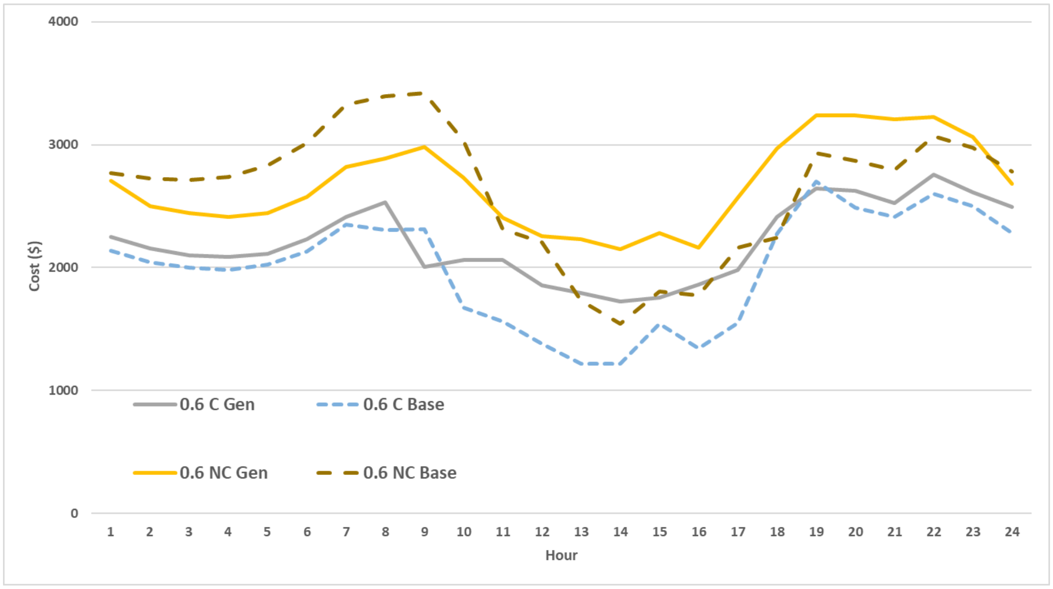

Figure 6.29: Winter Abnormal OPF Cost, Solar-to-Load Ratio = 0.6, Hourly Average

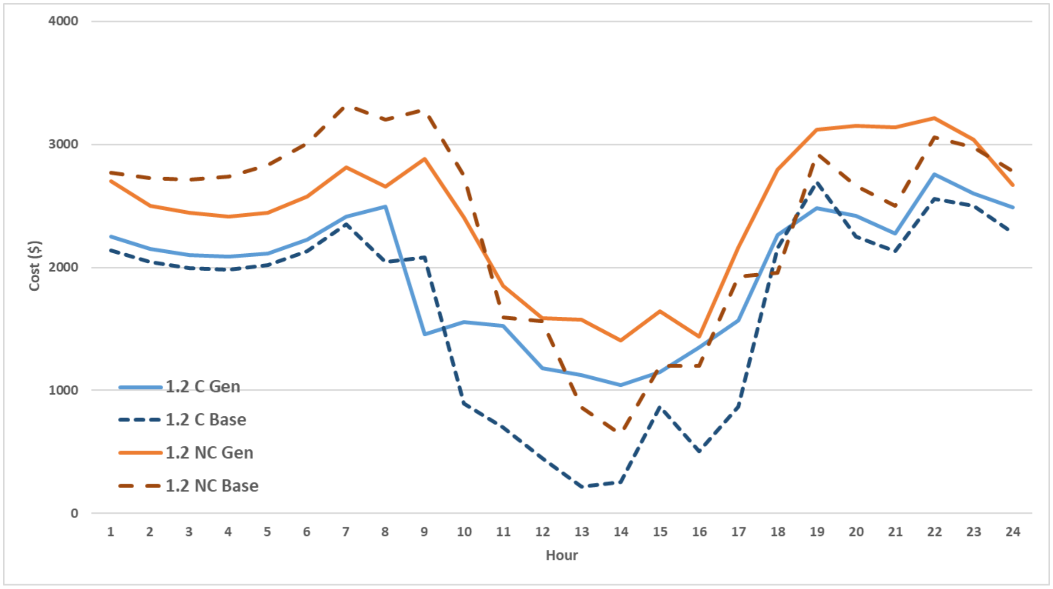

Figure 6.30: Winter Abnormal OPF Cost, Solar-to-Load Ratio = 1.2, Hourly Average

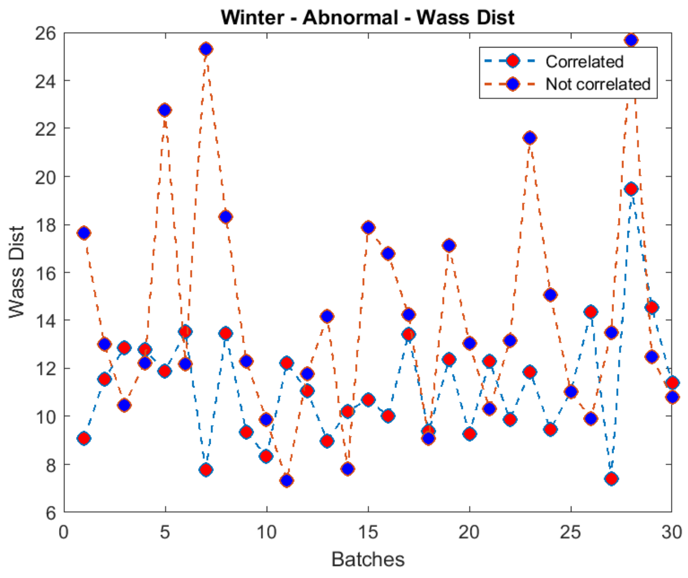

Figure 6.31: Winter Abnormal Wasserstein Distance, Solar-to-Load Ratio = 0.6

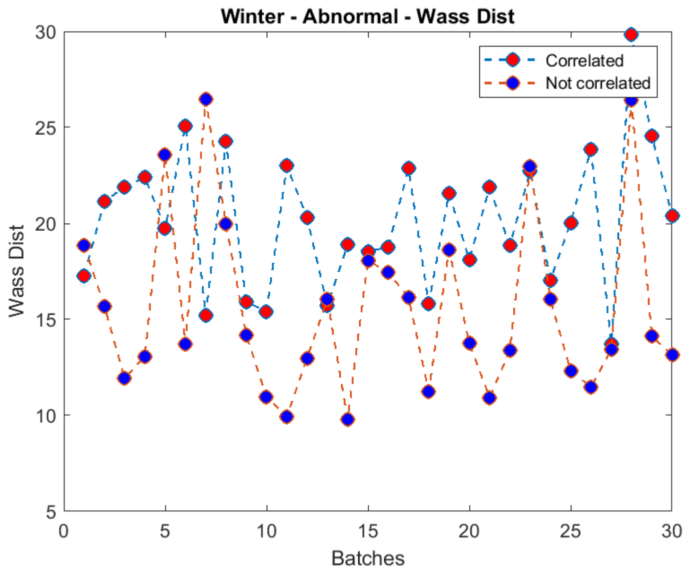

Figure 6.32: Winter Abnormal Wasserstein Distance, Solar-to-Load Ratio = 1.2

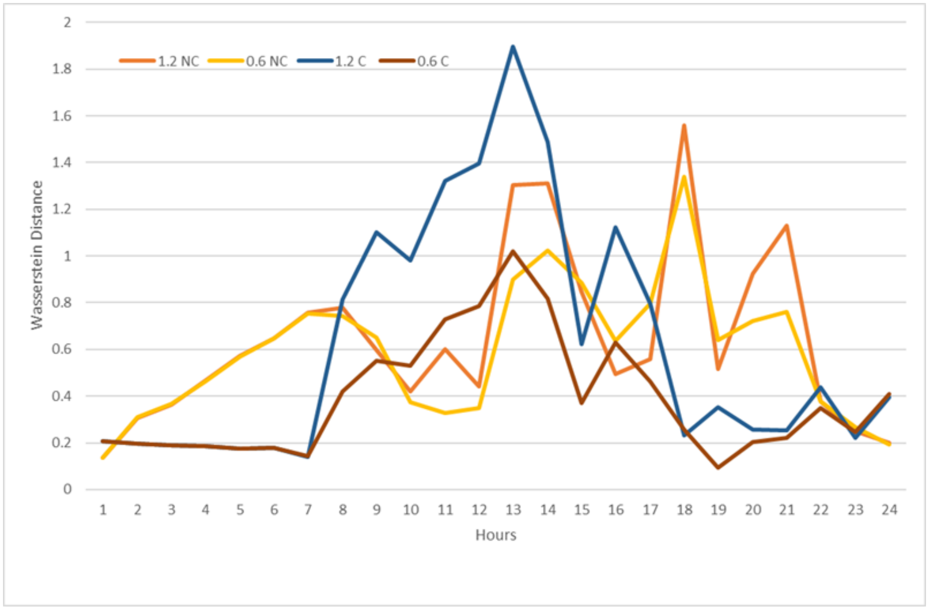

Figure 6.33: Hourly Abnormal Wasserstein Distance for Winter Season

correlated scenarios. This is explained by the fact that there is not much variation in both the solar and load profiles and the univariate GANs are able to better train on the available data. The correlated baseline is higher for both solar-to-load ratios because the average peak load is higher than for the uncorrelated baseline.

The Wasserstein distance also follows this trend, with the correlated scenarios displaying higher values than uncorrelated ones for both solar-to-load ratios. Nevertheless, the difference between correlated and uncorrelated generation scenarios is minimal.

Shoulder - Normal Scenarios

Voltage angle PDFs of the buses, for normal correlated and uncorrelated scenarios are shown in Fig. 6.42 and 6.43, respectively. It can be observed from the figures that the performance of correlated scenarios is only slightly better than uncorrelated scenario generation. Similarly, the normal shoulder scenarios are very close to one another,

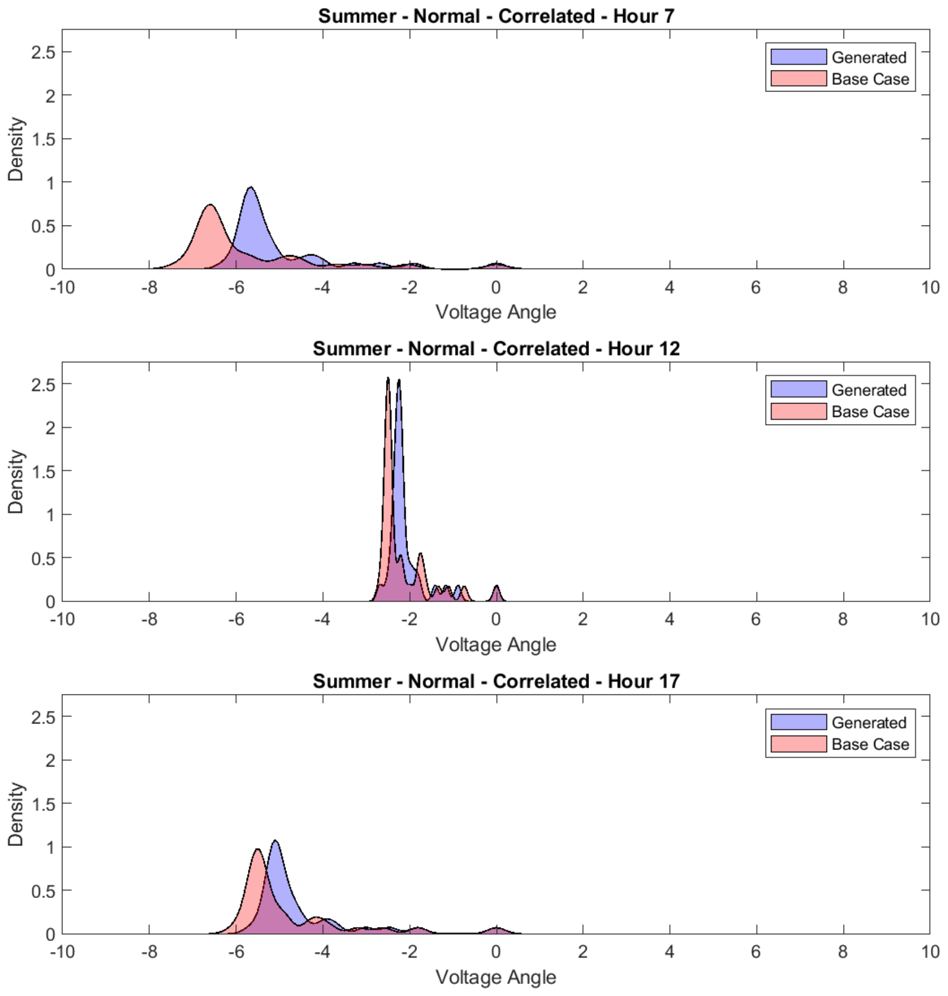

Figure 6.34: Voltage Angle PDF, Summer - Normal, Correlated

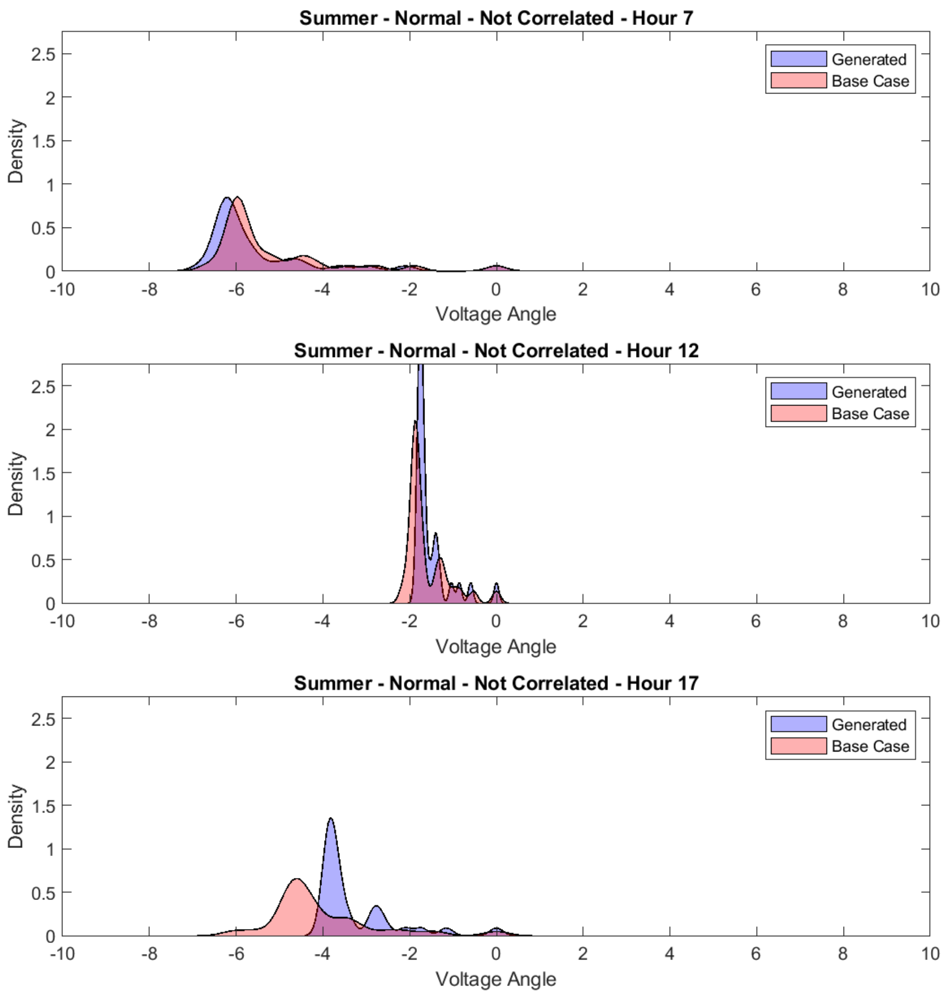

Figure 6.35: Voltage Angle PDF, Summer - Normal, Uncorrelated

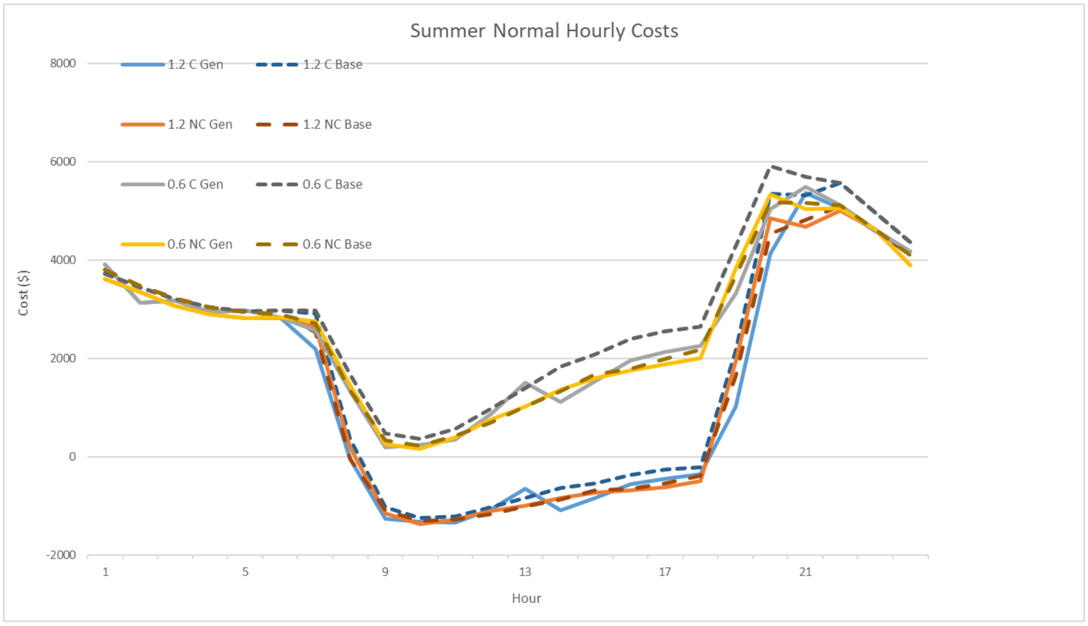

Figure 6.36: Summer Normal OPF Cost, Hourly Average

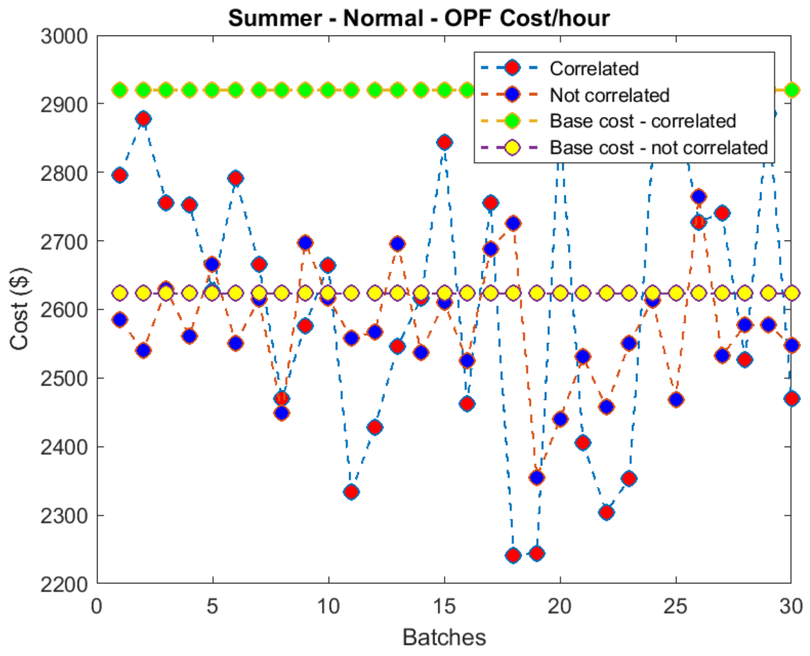

Figure 6.37: Summer Normal OPF Cost, Solar-to-Load Ratio = 0.6

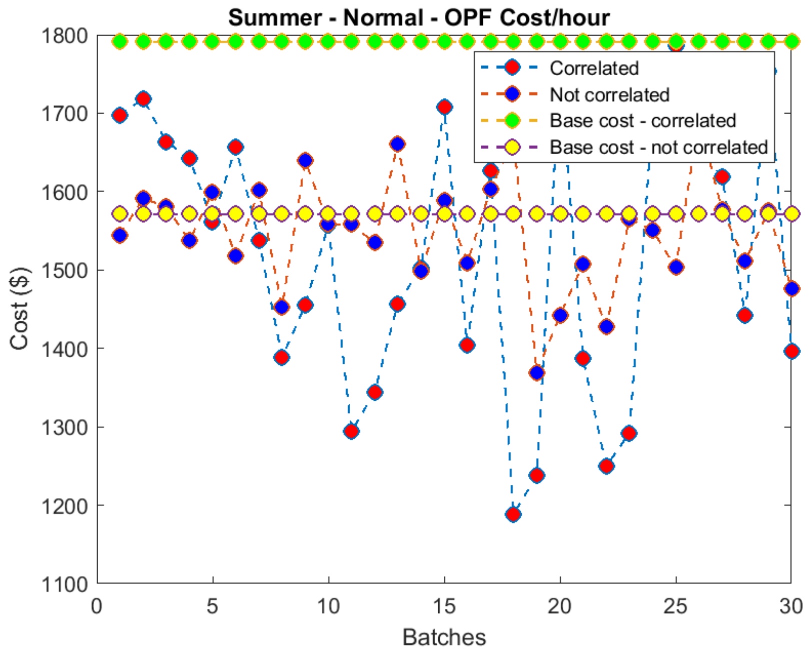

Figure 6.38: Summer Normal OPF Cost, Solar-to-Load Ratio = 1.2

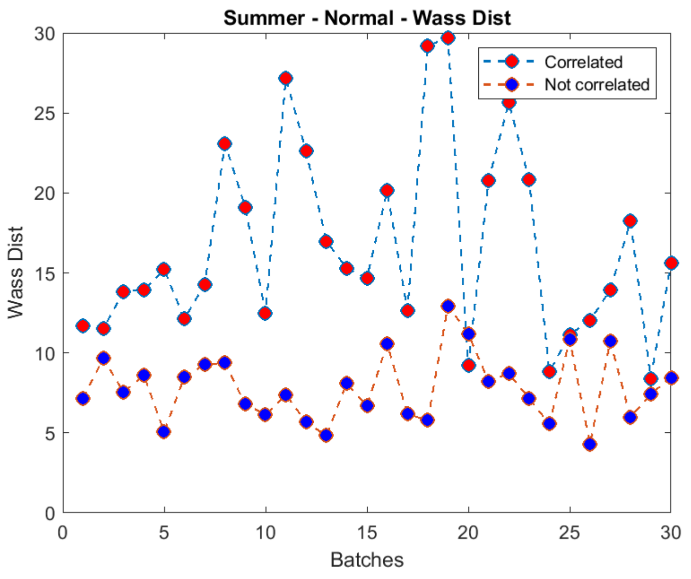

Figure 6.39: Summer Normal Wasserstein Distance, Solar-to-Load Ratio = 0.6

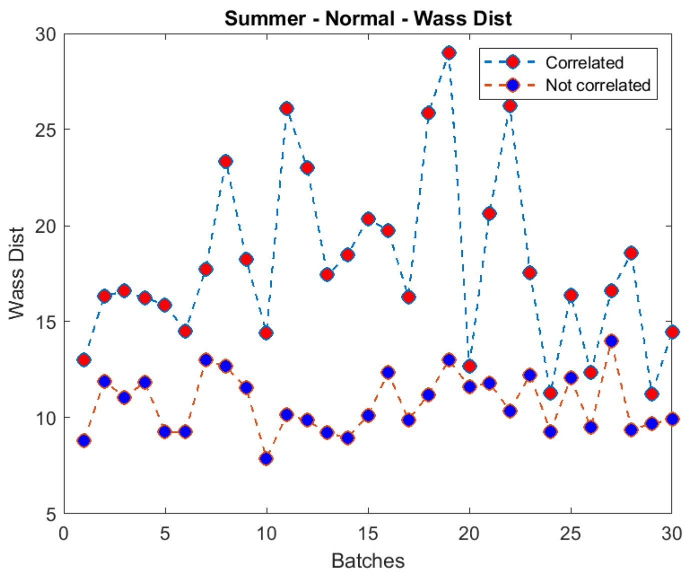

Figure 6.40: Summer Normal Wasserstein Distance, Solar-to-Load Ratio = 1.2

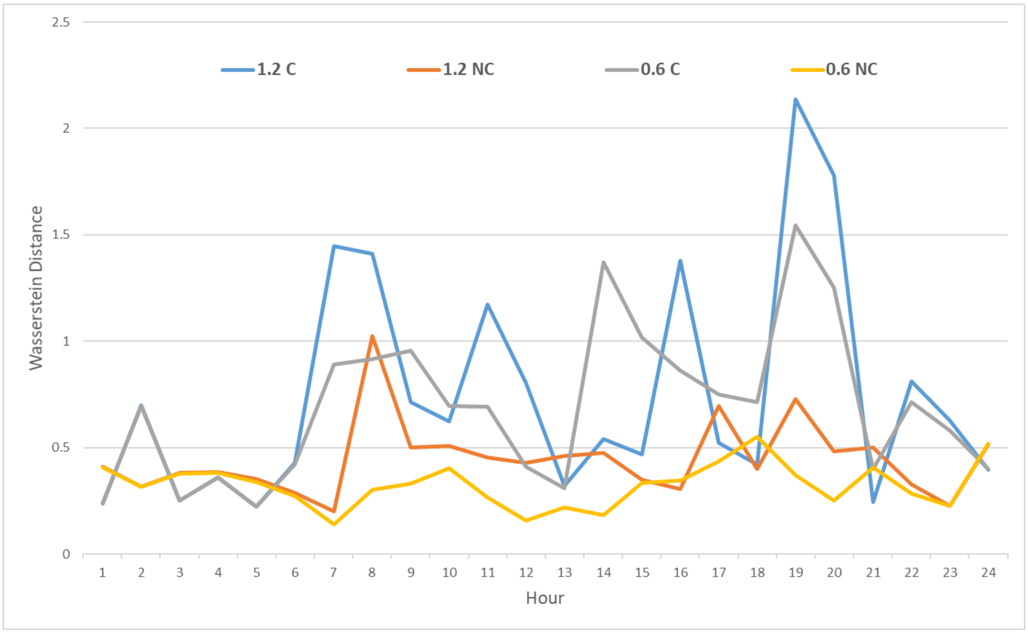

Figure 6.41: Hourly Normal Wasserstein Distance for Summer Season

as can be seen in Fig. 6.44 - Fig. 6.49, whether generated in a correlated manner or not. Since there are not many variations in the normal days, both correlated and uncorrelated GANs are able to pick up on the temporal features of the profiles very well.

The Wasserstein distance also follows this trend, with the correlated scenarios having marginally lower numbers than uncorrelated scenario generation.

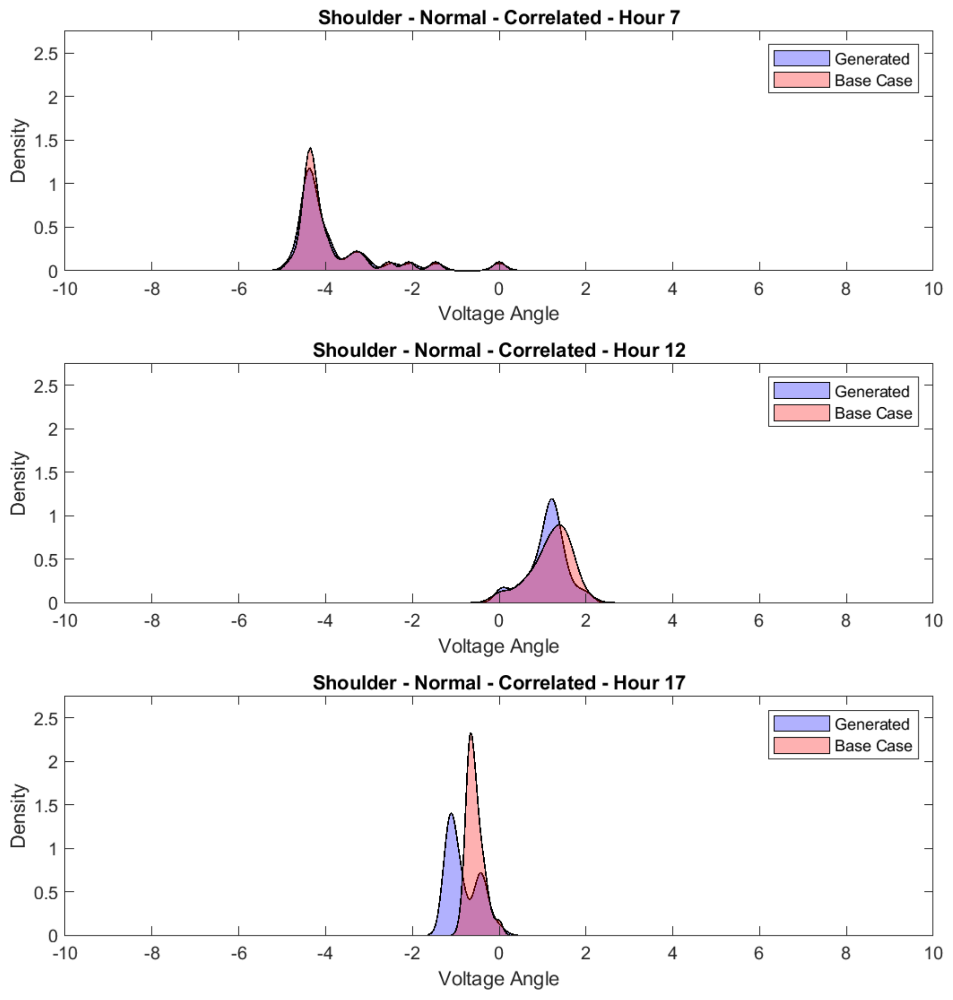

Figure 6.42: Voltage Angle PDF, Shoulder - Normal, Correlated

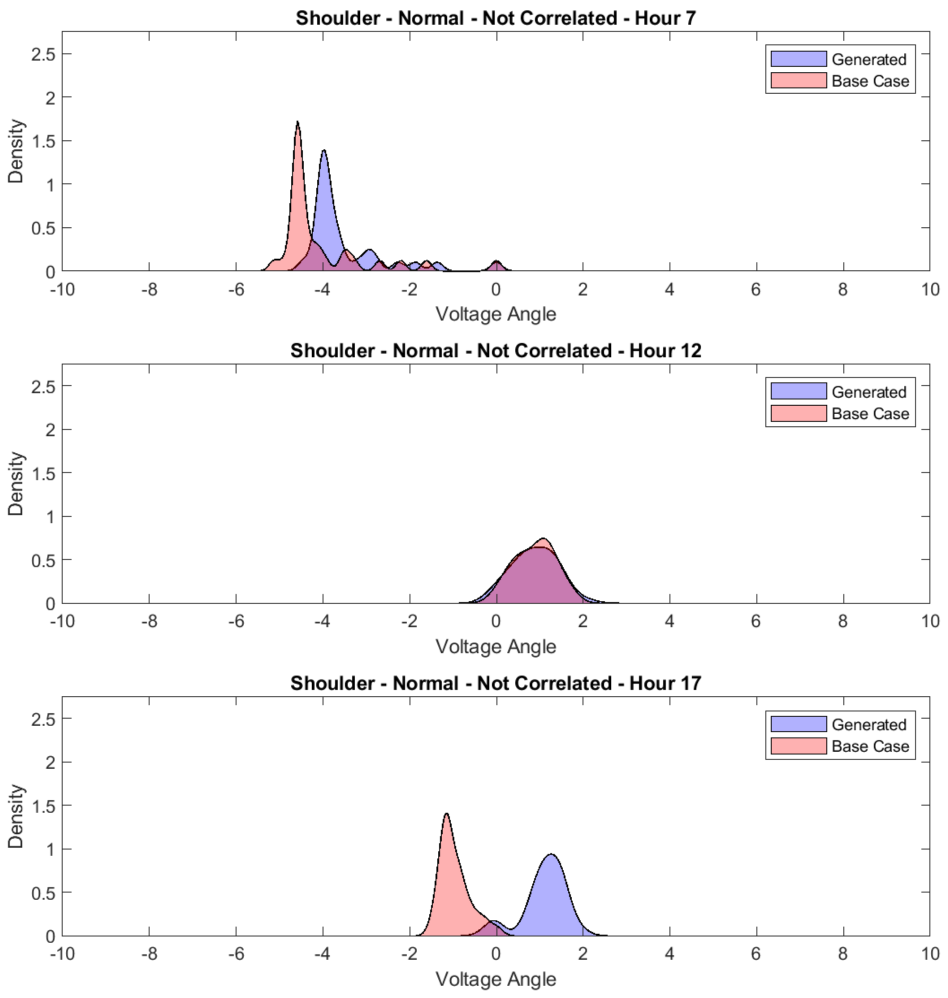

Figure 6.43: Voltage Angle PDF, Shoulder - Normal, Uncorrelated

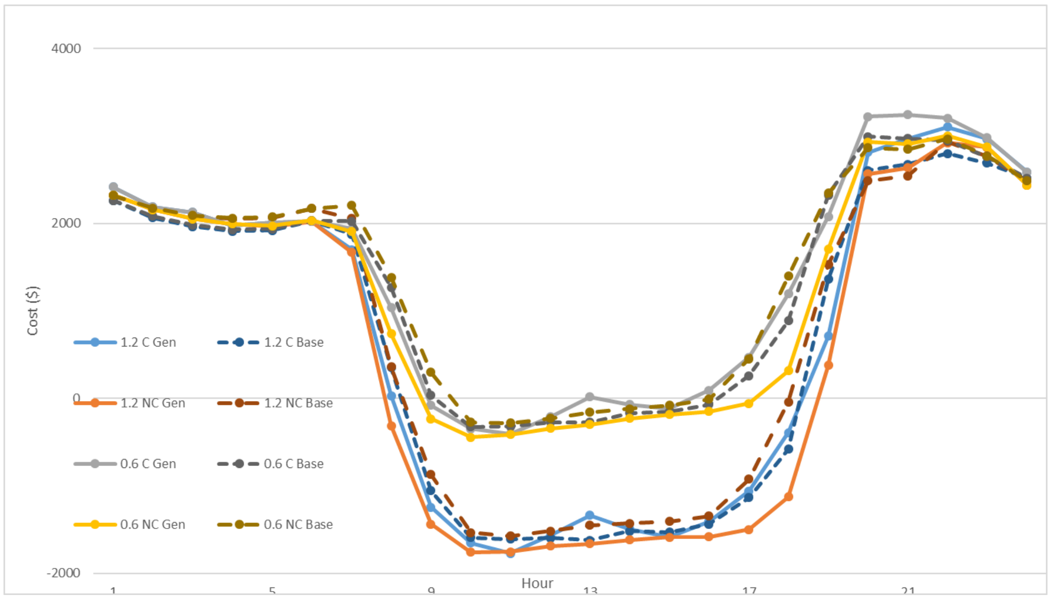

Figure 6.44: Shoulder Normal OPF Cost, Hourly Average

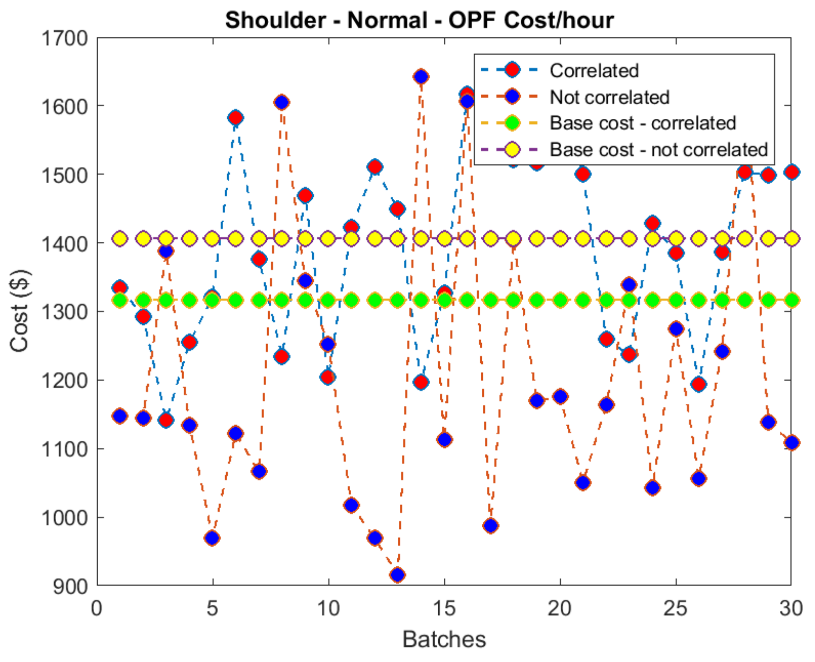

Figure 6.45: Shoulder Normal OPF Cost, Solar-to-Load Ratio = 0.6

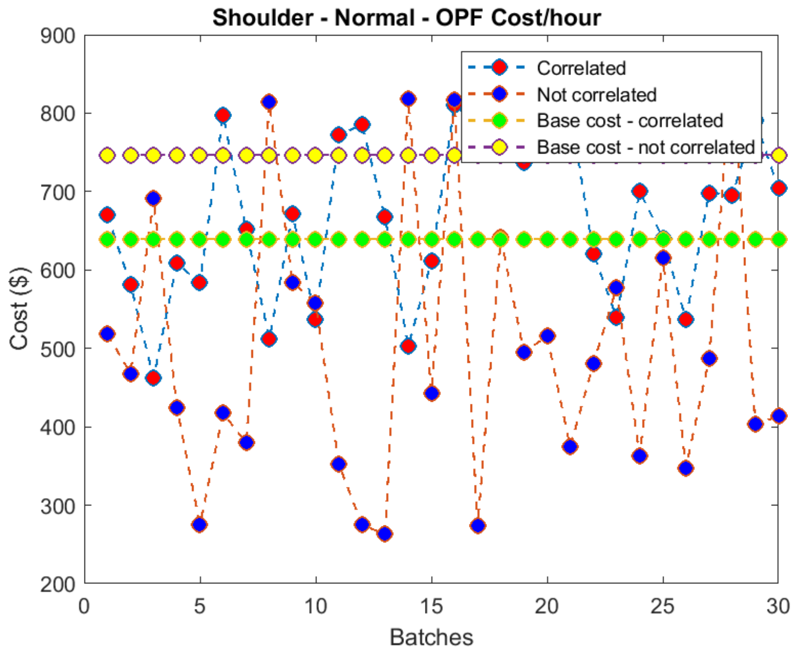

Figure 6.46: Shoulder Normal OPF Cost, Solar-to-Load Ratio = 1.2

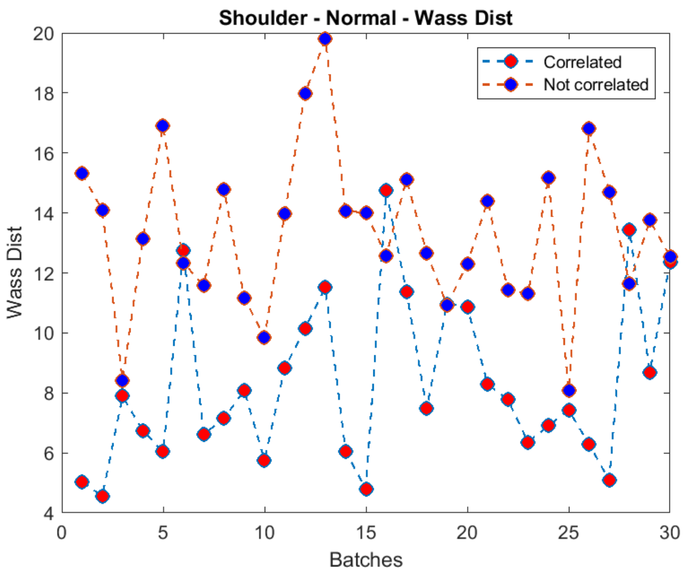

Figure 6.47: Shoulder Normal Wasserstein Distance, Solar-to-Load Ratio = 0.6

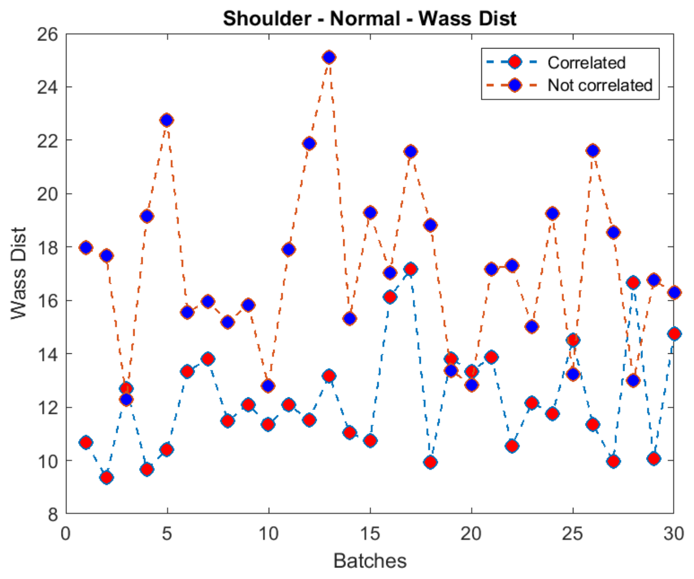

Figure 6.48: Shoulder Normal Wasserstein Distance, Solar-to-Load Ratio = 1.2

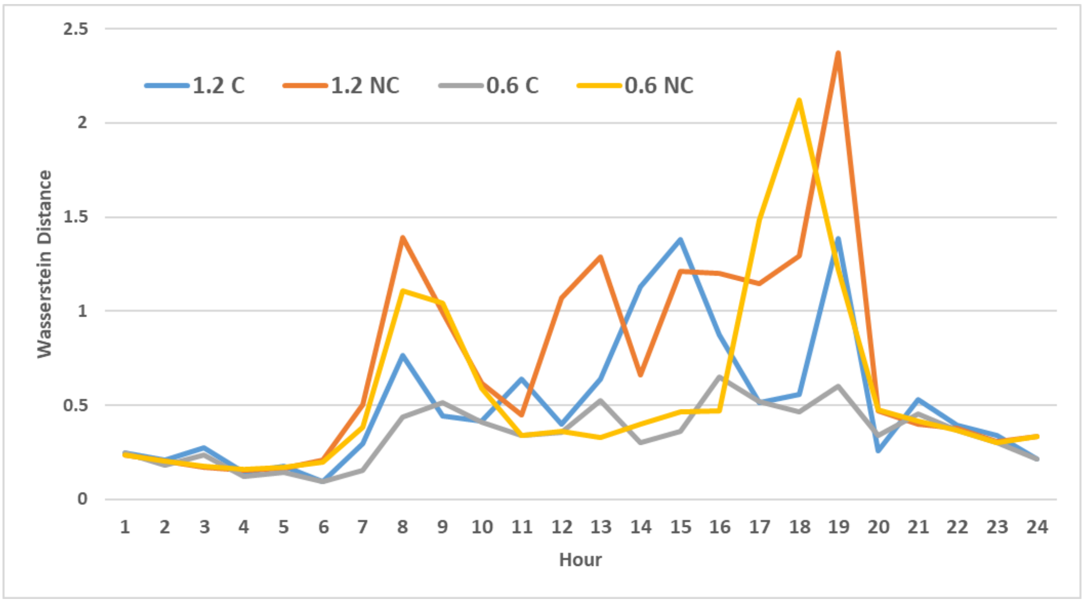

Figure 6.49: Hourly Normal Wasserstein Distance for Shoulder Season

Winter - Normal Scenarios

Fig. 6.50 and 6.51 depict the correlated and uncorrelated PDFs of the voltage angles for normal winter scenarios with a solar-to-load ratio of 0.6. It can be observed from the figures that the performance of correlated scenarios is slightly better than uncorrelated scenario generation. At the same time, correlated winter scenarios perform better than uncorrelated winter ones as they are closer to their respective baselines (Fig. 6.52 - Fig. 6.57). That being said, the overall difference is not big between the OPF cost of the scenarios themselves and to their baselines. Since these scenarios specify normal conditions, the variability between the scenarios is minimal and all the GANs are able to reproduce them with a high degree of confidence.

Interestingly enough, for a solar-to-load ratio of 0.6, the performance of the correlated scenarios is better than uncorrelated scenarios. The situation, however, is reversed for a solar-to-load ratio of 1.2. This is explained by observing the solar profiles generated by both the GANs. When the data are separated into abnormal and normal cases for uncorrelated profiles, the solar abnormal data mostly contain samples that have very low solar production and lesser variability. On the other hand, for correlated generation, abnormal solar data show higher peaks and more variability, which when combined with the added complexity of generation of two variables, results in the OPF cost and the Wasserstein distance being farther off from the baseline. This is especially the case for some hours, such as hour 17 and 19, where the generated solar profiles have low generation on average while the baseline solar profiles have peaks. This effect is more prominent in the higher solar-to-load ratio case.

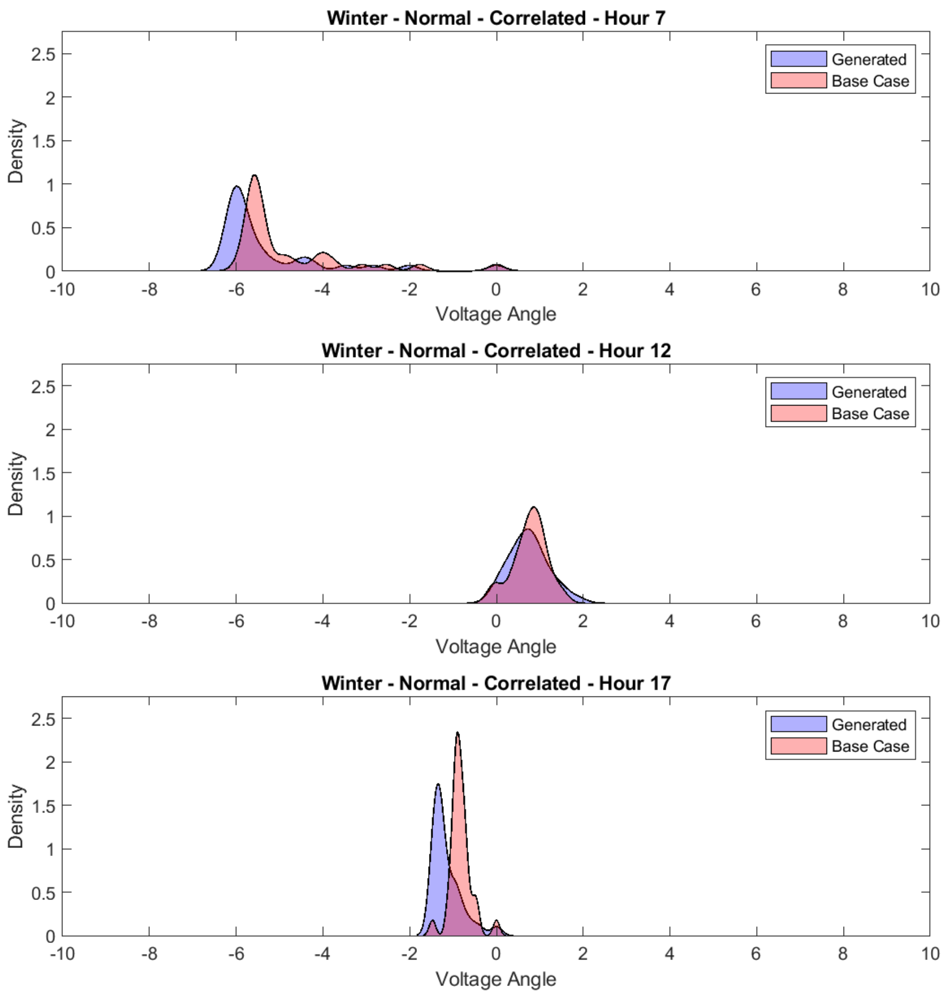

Figure 6.50: Voltage Angle PDF, Winter - Normal, Correlated

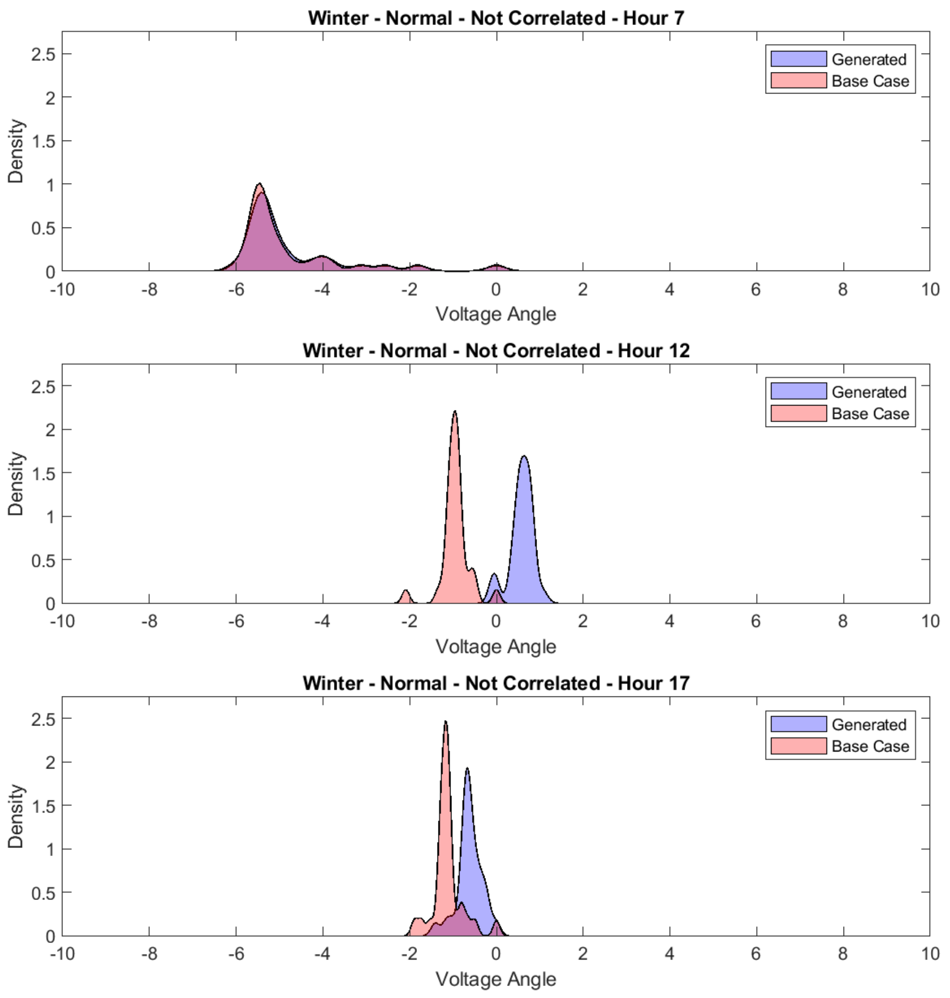

Figure 6.51: Voltage Angle PDF, Winter - Normal, Uncorrelated

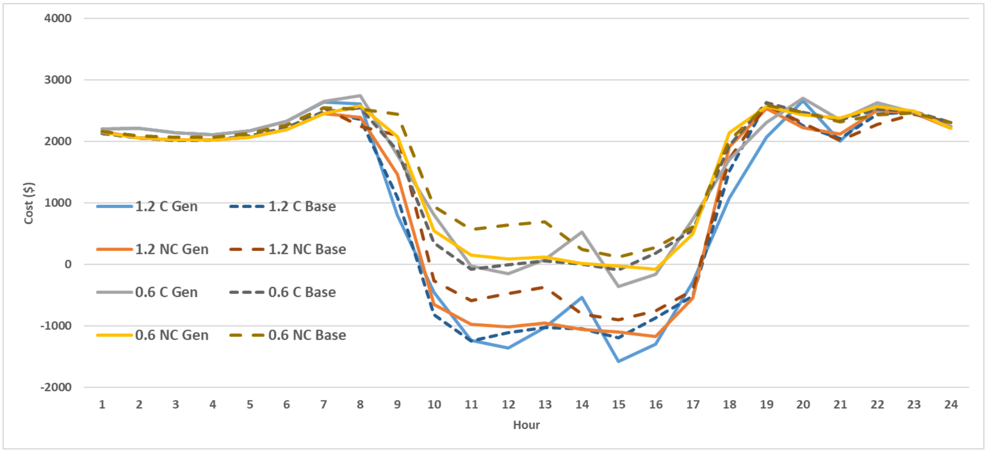

Figure 6.52: Winter Normal OPF Cost, Hourly Average

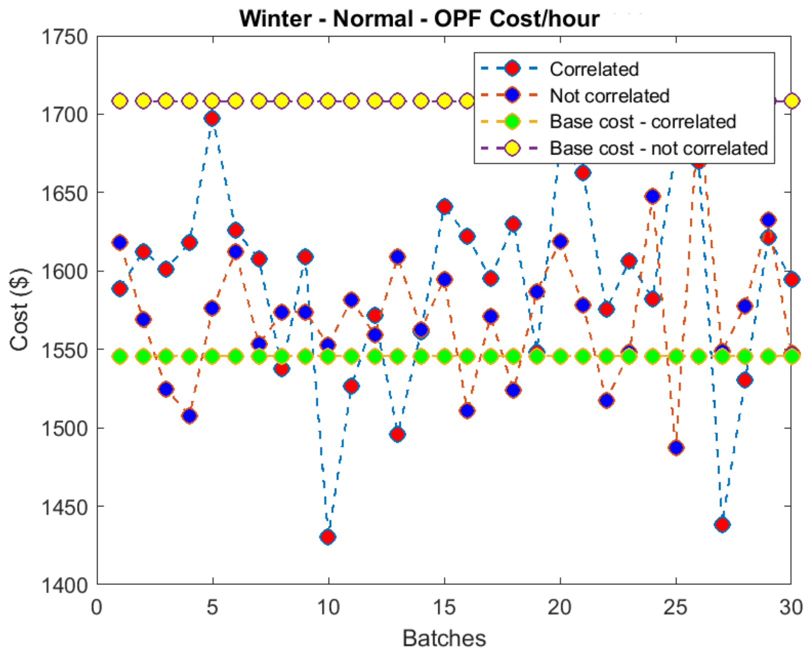

Figure 6.53: Winter Normal OPF Cost, Solar-to-Load Ratio = 0.6

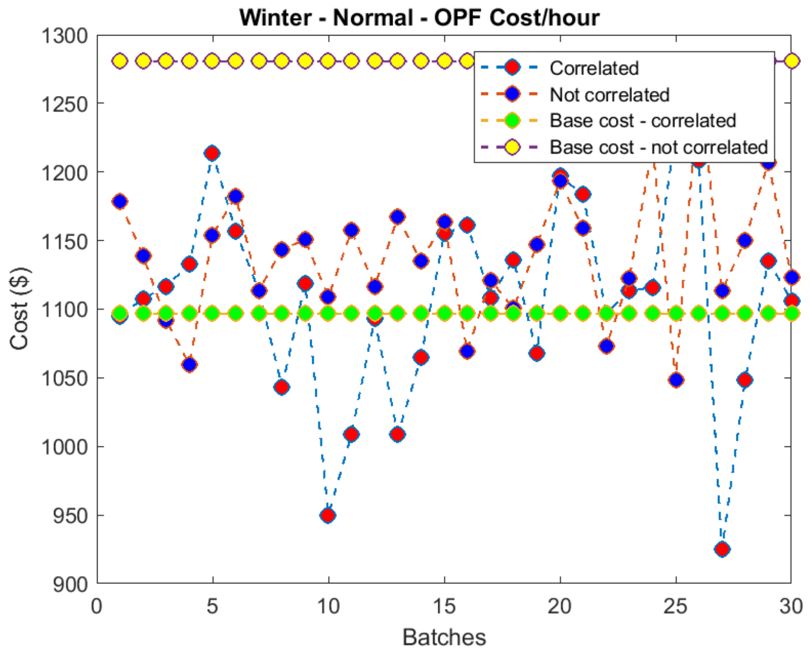

Figure 6.54: Winter Normal OPF Cost, Solar-to-Load Ratio = 1.2

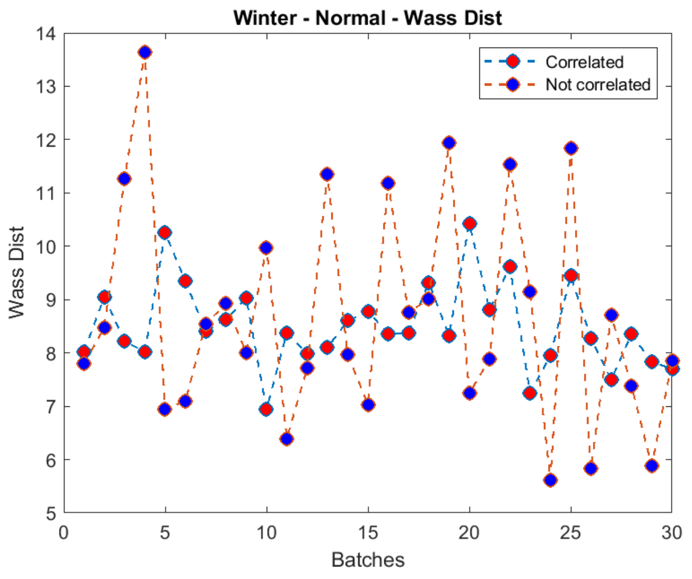

Figure 6.55: Winter Normal Wasserstein Distance, Solar-to-Load Ratio = 0.6

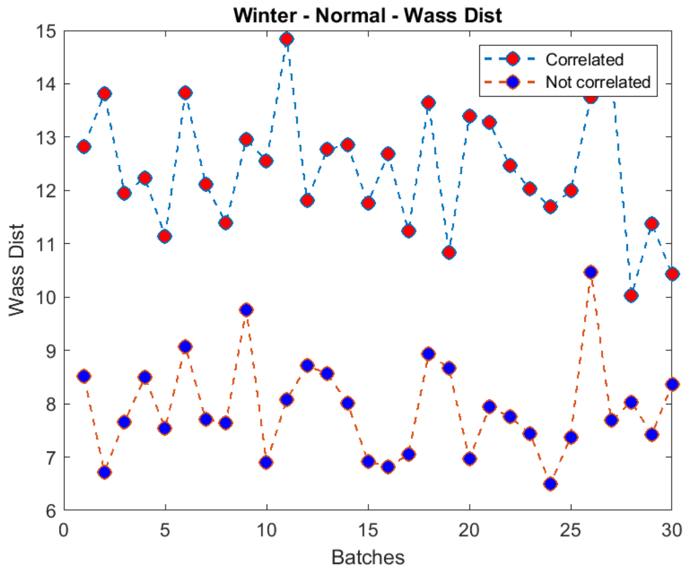

Figure 6.56: Winter Normal Wasserstein Distance, Solar-to-Load Ratio = 1.2

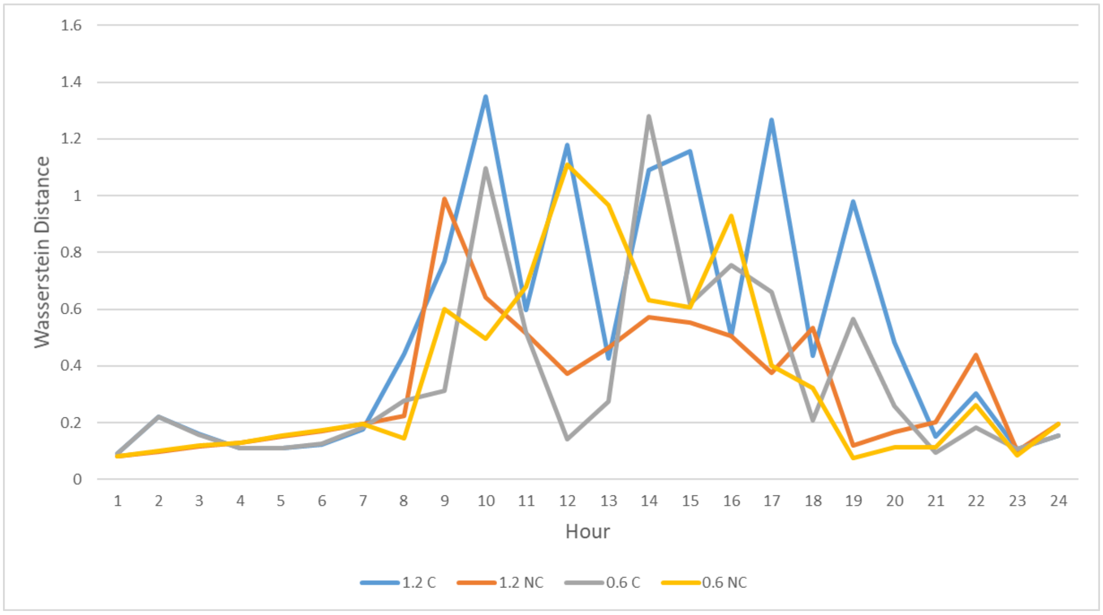

Figure 6.57: Hourly Normal Wasserstein Distance for Winter Season

Chapter 7

DISCUSSION

Given the results described in Chapter 6, it can be safely said that the proposed recurrent GAN is able to generate both solar and load profiles similar to the sequences present in the original dataset to a very satisfactory degree. The statistical analysis using auto-correlation and PSD verifies that the real and synthesized sequences depict similar temporal behavior. Further verification on the IEEE 30-bus system complements the performance of the created GANs.

The conditional training of the GAN gives the user more control over the type of scenarios being generated. With the correct labels, the GAN can be used to generate scenarios for any season and for both abnormal and normal conditions.

From the OPF cost and the Wasserstein distance results, it can be observed that the correlated GAN generally performs better than the GANs trained for uncorrelated scenarios. This is particularly apparent in the OPF results of abnormal profiles, with the abnormal days demonstrating a strong correlation between solar and load sequences. The uncorrelated GANs are not able to capture the real-world relationship between both variables and generate sequences with OPF costs and voltage angle distributions far off from its baseline. This is easily observed in Table 7.1 and Table 7.2, with the Wasserstein distance values between the PDFs of the voltage angles for generated scenarios vs. the baseline being lower for correlated scenarios than uncorrelated ones. This difference is also apparent in the OPF costs, with a large difference between correlated and uncorrelated scenarios.

The performance of the GANs for correlated scenario generation for the shoulder season is better than that for the uncorrelated scenario generation by a significant margin for all solar-to-load ratios. Moreover, the correlated GAN also performs well for normal conditions for the shoulder season, unlike for normal winter and summer where it performs marginally worse (see Table 7.3 and Table 7.4). This can be attributed to the variation of solar and load samples within the shoulder season itself, with correlation between the variables enabling the GANs to generate more realistic scenarios. Since the fluctuations in sequences, especially for load demand profiles, is low for normal conditions in summer and winter, the correlated GAN has a performance similar to or slightly worse than uncorrelated GANs. However, this is not the case for shoulder season where the profiles vary much more in peak and average values, and the performance of the correlated GAN is significantly better.

Table 7.1: Abnormal, Solar to Load Ratio = 0.6

	Season	Correlated	Uncorrelated
Wasserstein Dist.	Summer	0.58	1.59
	Shoulder	0.57	1.06
	Winter	0.39	0.59
Avg. Cost Per Hour ($)	Summer	3308	2452
	Shoulder	1749	3515
	Winter	2211	2674

For normal winter and summer conditions, uncorrelated scenarios tend to perform slightly worse than correlated scenarios. Nevertheless, the difference between correlated and uncorrelated scenarios against one another, and against their respective baselines is not significant. This can be attributed to the fact that there are fewer fluctuations between normal scenarios themselves compared to abnormal profiles. The univariate GANs are then able to train much better on the available data

Table 7.2: Abnormal, Solar to Load Ratio = 1.2

	Season	Correlated	Uncorrelated
Wasserstein Dist.	Summer	0.85	1.66
	Shoulder	0.77	1.17
	Winter	0.62	0.65
Avg. Cost Per Hour ($)	Summer	2363	1698
	Shoulder	1246	3035
	Winter	1961	2443

due to the lesser complexity of the sample space. It can, therefore, be said that this is a trade-off between better generation of abnormal scenarios vs. normal scenarios.

The OPF costs by themselves are not representative of the degree of similarity of the GANs' scenario generation ability to real cases, unless compared to a baseline. However, it can be seen in Tables 7.1 - 7.4 that the difference between OPF costs of correlated and uncorrelated scenarios do tend to generally diverge with larger difference in Wasserstein distances between the PDFs of the voltage angles.

Additionally, note that for a higher solar-to-load ratio of 1.2, the Wasserstein distances are slightly higher than for a lower solar-to-load ratio of 0.6. Since the solar profiles show greater fluctuations than load profiles, with a higher percentage of solar penetration, the variability of the data increases. This is reflected in the Wasserstein distances for all the seasons and for both normal and abnormal operating conditions.

From a power utility's perspective, generation of abnormal scenarios holds great value due to the rareness of abnormal events occurring throughout a calendar year. Since there is lesser data available for abnormal scenarios, dataset augmentation using the recurrent conditional GAN methodology described in this work is highly beneficial for performing contingency studies or analyzing worst-case scenarios.

Lastly, with the added penetration of renewables in power grids increasing each year, the flexible and scalable method of training the GAN ensures that it can be trained with more data samples easily. The conditional training is also possible after segregating new data samples into suitable classes and assigning the required labels. After training, the model is able to generate an unlimited number of conditioned scenarios which can be used to augment datasets to any degree.

Table 7.3: Normal, Solar to Load Ratio = 0.6

	Season	Correlated	Uncorrelated
Wasserstein Dist.	Summer	0.69	0.33
	Shoulder	0.35	0.56
	Winter	0.35	0.36
Avg. Cost Per Hour ($)	Summer	2616	2574
	Shoulder	1398	1208
	Winter	1592	1572

Table 7.4: Normal, Solar to Load Ratio = 1.2

	Season	Correlated	Uncorrelated
Wasserstein Dist.	Summer	0.75	0.45
	Shoulder	0.51	0.71
	Winter	0.52	0.33
Avg. Cost Per Hour ($)	Summer	1523	1548
	Shoulder	670	498
	Winter	1107	1139

Chapter 8

CONCLUSION

A recurrent, multivariate GAN methodology is outlined in this work for generating correlated solar generation and load demand scenarios. The correlated GAN proposed in this work holds great value for power utilities seeking to augment their dataset, especially for abnormal operating conditions, which are important for analyzing the behavior of the system for worse-case conditions. Coupled with the increase in renewable energy resources globally, and with progressively more stress being placed on the electric power grid, the ability to augment any dataset with more abnormal cases will result in more thorough reliability analyses.

Any number of required scenarios can be generated to augment multivariate time-series datasets to any desirable degree using generative models. However, the best generative models are those that are able to pick up on the nuances of the data succinctly. The correlation already present in the variables was shown to help the proposed generative model in generating more realistic sequences. This is especially true for abnormal scenarios, where the variables were shown to be interrelated to a higher degree. These correlations helped the proposed GAN model to produce sequences closer to the real samples, and not simply replicating the original dataset.

The validation of the generated scenarios was performed both in a statistical fashion, by using auto-correlation and power spectral density, to show that the generated profiles are similar to the real ones, and by implementation of the synthesized profiles on the IEEE 30-bus system. The latter enabled the observation of the response of a simulated power system when run with synthetic data and provided a way to compare the generated scenarios with real ones, and comparison of correlated scenario generation with uncorrelated scenario generation. The results verify the usage of the generated profiles in a real-world system and complement the argument in favor of correlated scenario generation.

The performance of the correlated GAN against uncorrelated load and solar GANs was investigated in detail and it was observed that for abnormal operating conditions, correlated scenario generation fares much better than uncorrelated scenario generation, owing to the ability of the model to keep the correlations intact during the generation process. Since it is usually necessary to add more abnormal scenarios to power system datasets, the proposed recurrent conditional GAN is a valuable tool for increasing the number of such samples in the database. Finally, since increased penetration of renewables in the grid increases the number of interconnected variables which might affect system stability, the generation of scenarios in a correlated manner helps in synthesising datasets with the correlations preserved.

8.1 Future Work

The following studies can be undertaken to expand upon the work presented in this thesis:

- Incorporation of spatial correlation in the training of the GAN by addition of information about the distribution of solar generation sites and/or load demand in the area.

- Making the generative model more sophisticated with the addition of embedding layers to extract the important features of the data.

- Formulation of a framework to utilize the generated scenarios in probabilistic forecasting, reliability planning, and decision-making such as for placing energy storage systems.

REFERENCES

[1] Net renewable capacity additions by technology, 2017-2023. `https://www.iea.org/reports/renewable-energy-market-update-may-2022/renewable-electricity`, 2022. Accessed: 2022-06-26.

[2] Reetam Sen Biswas, Anamitra Pal, Trevor Werho, and Vijay Vittal. Mitigation of saturated cut-sets during multiple outages to enhance power system security. *IEEE Transactions on Power Systems*, 36(6):5734–5745, 2021.

[3] Tong Wang, Mingxin Jin, Babak Jafarpisheh, Anamitra Pal, and Zengping Wang. Gain scheduled adaptive control scheme for damping ssos in pmsg-integrated power system under high wind speed variability. *Electric Power Components and Systems*, 49(9-10):953–966, 2021.

[4] Pooja Gupta, Anamitra Pal, and Vijay Vittal. Coordinated wide-area control of multiple controllers in a power system embedded with hvdc lines. *IEEE Transactions on Power Systems*, 36(1):648–658, 2020.

[5] Anamitra Pal and Meghna Barkakati. Systems and methods for assessing reliability of electrical power transmission systems, April 29 2021. US Patent App. 17/078,863.

[6] Atif Naveed Khan, Kashif Imran, Muhammad Nadeem, Anamitra Pal, Abraiz Khattak, Kafait Ullah, Muhammad Waseem Younas, and Muhammad Shahzad Younis. Ensuring reliable operation of electricity grid by placement of facts devices for developing countries. *Energies*, 14(8):2283, 2021.

[7] Hashem Albhrani, Reetam Sen Biswas, and Anamitra Pal. Identification of utility-scale renewable energy penetration threshold in a dynamic setting. In *2020 52nd North American Power Symposium (NAPS)*, pages 1–6. IEEE, 2021.

[8] Reetam Sen Biswas, Anamitra Pal, Trevor Werho, and Vijay Vittal. Fast identification of saturated cut-sets using graph search techniques. In *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2020.

[9] Chen Wang, Chetan Mishra, Reetam Sen Biswas, Anamitra Pal, and Virgilio A Centeno. Adaptive lvrt settings adjustment for enhancing voltage security of renewable-rich electric grids. In *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2020.

[10] Reetam Sen Biswas, Anamitra Pal, Trevor Werho, and Vijay Vittal. A graph theoretic approach to power system vulnerability identification. *IEEE Transactions on Power Systems*, 36(2):923–935, 2020.

[11] Malhar Padhee, Reetam Sen Biswas, Anamitra Pal, Kaustav Basu, and Arunabha Sen. Identifying unique power system signatures for determining vulnerability of critical power system assets. *ACM SIGMETRICS Performance Evaluation Review*, 47(4):8–11, 2020.

[12] Tong Wang, Jing Yang, Malhar Padhee, Jingtian Bi, Anamitra Pal, and Zengping Wang. Robust, coordinated control of sso in wind-integrated power system. *IET Renewable Power Generation*, 14(6):1031–1043, 2020.

[13] Chetan Mishra, Reetam Sen Biswas, Anamitra Pal, and Virgilio A Centeno. Critical clearing time sensitivity for inequality constrained systems. *IEEE Transactions on Power Systems*, 35(2):1572–1583, 2019.

[14] Meghna Barkakati and Anamitra Pal. A comprehensive data driven outage analysis for assessing reliability of the bulk power system. In *2019 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2019.

[15] Pooja Gupta, Anamitra Pal, Chetan Mishra, and Tong Wang. Design of a coordinated wide area damping controller by employing partial state feedback. In *2019 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2019.

[16] Chetan Mishra, Anamitra Pal, James S Thorp, and Virgilio A Centeno. Transient stability assessment of prone-to-trip renewable generation rich power systems using lyapunov's direct method. *IEEE Transactions on Sustainable Energy*, 10(3):1523–1533, 2019.

[17] Tong Wang, Jing Yang, Jiuliang Liu, Pooja Gupta, Anamitra Pal, and Jun Deng. Sdae-based probabilistic stability analysis of wind integrated power systems. In *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)*, pages 1–6. IEEE, 2018.

[18] Malhar Padhee and Anamitra Pal. Effect of solar pv penetration on residential energy consumption pattern. In *2018 North American Power Symposium (NAPS)*, pages 1–6. IEEE, 2018.

[19] Chetan Mishra, James S Thorp, Virgilio A Centeno, and Anamitra Pal. Transient stability assessment of cascade tripping of renewable sources using sos. In *2018 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2018.

[20] Chetan Mishra, James S Thorp, Virgilio A Centeno, and Anamitra Pal. Estimating relevant portion of stability region using lyapunov approach and sum of squares. In *2018 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2018.

[21] Malhar Padhee, Anamitra Pal, and Katelynn A Vance. Analyzing effects of seasonal variations in wind generation and load on voltage profiles. In *2017 North American Power Symposium (NAPS)*, pages 1–6. IEEE, 2017.

[22] Chetan Mishra, James S Thorp, Virgilio A Centeno, and Anamitra Pal. Stability region estimation under low voltage ride through constraints using sum of squares. In *2017 North American Power Symposium (NAPS)*, pages 1–6. IEEE, 2017.

[23] Rajesh Subbiah, Anamitra Pal, Eric K Nordberg, Achla Marathe, and Madhav V Marathe. Energy demand model for residential sector: a first principles approach. *IEEE Transactions on Sustainable Energy*, 8(3):1215–1224, 2017.

[24] Tong Wang, Anamitra Pal, James S Thorp, and Yuan Yang. Use of polytopic convexity in developing an adaptive interarea oscillation damping scheme. *IEEE Transactions on Power Systems*, 32(4):2509–2520, 2016.

[25] Anamitra Pal. Phasor measurement-enabled decision making. In *Synchronized Phasor Measurements and Their Applications*, pages 211–243. Springer, 2017.

[26] Tong Wang, Anamitra Pal, James S Thorp, Zengping Wang, Jizhen Liu, and Yuan Yang. Multi-polytope-based adaptive robust damping control in power systems using cart. *IEEE transactions on power systems*, 30(4):2063–2072, 2014.

[27] Anamitra Pal, Iknoor Singh, and Bharat Bhargava. Stress assessment in power systems and its visualization using synchrophasor based metrics. In *2014 North American Power Symposium (NAPS)*, pages 1–6. IEEE, 2014.

[28] Anamitra Pal. *PMU-based applications for improved monitoring and protection of power systems.* PhD thesis, Virginia Polytechnic Institute and State University, 2014.

[29] Anamitra Pal, James S Thorp, Santosh S Veda, and VA Centeno. Applying a robust control technique to damp low frequency oscillations in the wecc. *International Journal of Electrical Power & Energy Systems*, 44(1):638–645, 2013.

[30] Katelynn Vance, Anamitra Pal, and James S Thorp. A robust control technique for damping inter-area oscillations. In *2012 IEEE Power and Energy Conference at Illinois*, pages 1–8. IEEE, 2012.

[31] Anamitra Pal. *Coordinated Control of Inter-area Oscillations using SMA and LMI.* PhD thesis, Virginia Polytechnic Institute & State University, 2012.

[32] Anamitra Pal and James S Thorp. Co-ordinated control of inter-area oscillations using sma and lmi. In *2012 IEEE PES Innovative Smart Grid Technologies (ISGT)*, pages 1–6. IEEE, 2012.

[33] Anamitra Pal and James S. Thorp. Coordinated control of inter-area oscillations using sma and lmi: A robust control technique for damping low frequency oscillations, 2012.

[34] Anamitra Pal. Damping low frequency oscillations in the wecc. *Final project report, public interest energy research (PIER) program, TRP-08-06, prepared for California Energy Commission by Virginia Polytechnic Institute and State University, Blacksburg, Virginia*, 2011.

[35] Tao Hong and Alex Hofmann. Data integrity attacks against outage management systems. *IEEE Transactions on Engineering Management*, 69(3):765–772, 2021.

[36] Anubhav Nath, Reetam Sen Biswas, and Anamitra Pal. Application of machine learning for online dynamic security assessment in presence of system variability and additive instrumentation errors. In *2019 North American Power Symposium (NAPS)*, pages 1–6. IEEE, 2019.

[37] Pooja Gupta, Anamitra Pal, and Vijay Vittal. Coordinated wide-area damping control using deep neural networks and reinforcement learning. *IEEE Transactions on Power Systems*, 37(1):365–376, 2021.

[38] Behrouz Azimian, R Sen Biswas, Anamitra Pal, and Lang Tong. Time synchronized state estimation for incompletely observed distribution systems using deep learning considering realistic measurement noise. In *2021 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2021.

[39] Behrouz Azimian, Reetam Sen Biswas, Shiva Moshtagh, Anamitra Pal, Lang Tong, and Gautam Dasarathy. State and topology estimation for unobservable distribution systems using deep neural networks. *IEEE Transactions on Instrumentation and Measurement*, 71:1–14, 2022.

[40] Meiyan Li, Anamitra Pal, Arun G Phadke, and James S Thorp. Transient stability prediction based on apparent impedance trajectory recorded by pmus. *International Journal of Electrical Power & Energy Systems*, 54:498–504, 2014.

[41] Anamitra Pal, JS Thorp, Taufiquar Khan, and S Stanley Young. Classification trees for complex synchrophasor data. *Electric Power Components and Systems*, 41(14):1381–1396, 2013.

[42] Fenghua Gao, James S Thorp, Shibin Gao, Anamitra Pal, and Katelynn A Vance. A voltage phasor based fault-classification method for phasor measurement unit only state estimator output. *Electric Power Components and Systems*, 43(1):22–31, 2015.

[43] Pierre Pinson, Henrik Madsen, Henrik Aa Nielsen, George Papaefthymiou, and Bernd Klöckl. From probabilistic forecasts to statistical scenarios of short-term wind power production. *Wind Energy: An International Journal for Progress and Applications in Wind Power Conversion Technology*, 12(1):51–62, 2009.

[44] Juan M Morales, Luis Baringo, Antonio J Conejo, and Roberto Mínguez. Probabilistic power flow with correlated wind sources. *IET generation, transmission & distribution*, 4(5):641–651, 2010.

[45] Xinmin Zhang, Yuan Li, Siyuan Lu, Hendrik F Hamann, Bri-Mathias Hodge, and Brad Lehman. A solar time based analog ensemble method for regional solar power forecasting. *IEEE Transactions on Sustainable Energy*, 10(1):268–279, 2018.

[46] Duehee Lee and Ross Baldick. Load and wind power scenario generation through the generalized dynamic factor model. *IEEE Transactions on power Systems*, 32(1):400–410, 2016.

[47] Ben S Bernanke, Jean Boivin, and Piotr Eliasz. Measuring the effects of monetary policy: a factor-augmented vector autoregressive (favar) approach. *The Quarterly journal of economics*, 120(1):387–422, 2005.

[48] Xingpeng Jiang, Xiaohua Hu, Weiwei Xu, and EK Park. Predicting microbial interactions using vector autoregressive model with graph regularization. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 12(2): 254–261, 2014.

[49] Duehee Lee and Ross Baldick. Future wind power scenario synthesis through power spectral density analysis. *IEEE Transactions on Smart Grid*, 5(1):490–500, 2013.

[50] Anthony Papavasiliou, Shmuel S Oren, and Richard P O'Neill. Reserve requirements for wind power integration: A scenario-based stochastic programming framework. *IEEE Transactions on Power Systems*, 26(4):2197–2206, 2011.

[51] Guzmán Díaz, Javier Gómez-Aleixandre, and José Coto. Wind power scenario generation through state-space specifications for uncertainty analysis of wind power plants. *Applied Energy*, 162:21–30, 2016.

[52] Heejung Park and Ross Baldick. Transmission planning under uncertainties of wind and load: Sequential approximation approach. *IEEE Transactions on Power Systems*, 28(3):2395–2402, 2013.

[53] David P Larson, Lukas Nonnenmacher, and Carlos FM Coimbra. Day-ahead forecasting of solar power output from photovoltaic plants in the american southwest. *Renewable Energy*, 91:11–20, 2016.

[54] Jing Huang, Lawrence Rikus, and Yi Qin. Probabilistic solar irradiance forecasting using numerical weather prediction ensembles over australia. In *2020 47th IEEE Photovoltaic Specialists Conference (PVSC)*, pages 0554–0558. IEEE, 2020.

[55] Amedeo Buonanno, Martina Caliano, Marialaura Di Somma, Giorgio Graditi, and Maria Valenti. Comprehensive method for modeling uncertainties of solar irradiance for pv power generation in smart grids. In *2021 International Conference on Smart Energy Systems and Technologies (SEST)*, pages 1–6. IEEE, 2021.

[56] M Mohandes, S Rehman, and TO Halawani. Estimation of global solar radiation using artificial neural networks. *Renewable energy*, 14(1-4):179–184, 1998.

[57] Ahmed Ouammi, Driss Zejli, Hanane Dagdougui, and Rachid Benchrifa. Artificial neural network analysis of moroccan solar potential. *Renewable and Sustainable Energy Reviews*, 16(7):4876–4889, 2012.

[58] Juan A Lazzus, Alejandro A Perez Ponce, and Julio Marin. Estimation of global solar radiation over the city of la serena (chile) using a neural network. *Applied Solar Energy*, 47(1):66–73, 2011.

[59] Mingjian Cui, Deping Ke, Yuanzhang Sun, Di Gan, Jie Zhang, and Bri-Mathias Hodge. Wind power ramp event forecasting using a stochastic scenario generation method. *IEEE Transactions on sustainable energy*, 6(2):422–433, 2015.

[60] George Sideratos and Nikos D Hatziargyriou. Probabilistic wind power forecasting using radial basis function neural networks. *IEEE Transactions on Power Systems*, 27(4):1788–1796, 2012.

[61] Hung-Chih Wu and Chan-Nan Lu. A data mining approach for spatial modeling in small area load forecast. *IEEE Transactions on Power Systems*, 17(2):516–521, 2002.

[62] Shu Fan, Luonan Chen, and Wei-Jen Lee. Machine learning based switching model for electricity load forecasting. *Energy Conversion and Management*, 49 (6):1331–1344, 2008.

[63] Yinghao Chu, Bryan Urquhart, Seyyed MI Gohari, Hugo TC Pedro, Jan Kleissl, and Carlos FM Coimbra. Short-term reforecasting of power output from a 48 mwe solar pv plant. *Solar Energy*, 112:68–77, 2015.

[64] You-Jing Zhong and Yuan-Kang Wu. Short-term solar power forecasts considering various weather variables. In *2020 International Symposium on Computer, Consumer and Control (IS3C)*, pages 432–435. IEEE, 2020.

[65] Stylianos I Vagropoulos, Evaggelos G Kardakos, Christos K Simoglou, Anastasios G Bakirtzis, and Joao PS Catalao. Ann-based scenario generation methodology for stochastic variables of electric power systems. *Electric Power Systems Research*, 134:9–18, 2016.

[66] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[67] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[68] Hazem Zein, Samer Chantaf, Rola El-Saleh, and Amine Nait-Ali. Generative adversarial networks based approach for artificial face dataset generation in acne disease cases. In *2021 4th International Conference on Bio-Engineering for Smart Technologies (BioSMART)*, pages 1–4. IEEE, 2021.

[69] Xinru Zhong, Xiujie Qu, and Chen Chen. High-quality face image super-resolution based on generative adversarial networks. In *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, volume 1, pages 1178–1182. IEEE, 2019.

[70] Hongyu Chen, Qinyin Xiao, and Xueyuan Yin. Generating music algorithm with deep convolutional generative adversarial networks. In *2019 IEEE 2nd International Conference on Electronics Technology (ICET)*, pages 576–580. IEEE, 2019.

[71] Advait Maduskar, Aniket Ladukar, Shubhankar Gore, and Neha Patwari. Music generation using deep generative modelling. In *2020 International Conference on Convergence to Digital World-Quo Vadis (ICCDW)*, pages 1–4. IEEE, 2020.

[72] Raymond Kwan How Toh and Alexei Sourin. Generation of music with dynamics using deep convolutional generative adversarial network. In *2021 International Conference on Cyberworlds (CW)*, pages 137–140. IEEE, 2021.

[73] Xiaolin Li, Changcheng Shao, Yifan Zhou, and Lei Huang. Face mask removal based on generative adversarial network and texture network. In *2021 4th International Conference on Robotics, Control and Automation Engineering (RCAE)*, pages 86–89. IEEE, 2021.

[74] Zheng-An Zhu, Yun-Zhong Lu, and Chen-Kuo Chiang. Generating adversarial examples by makeup attacks on face recognition. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2516–2520. IEEE, 2019.

[75] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *International conference on machine learning*, pages 1060–1069. PMLR, 2016.

[76] Azmi Can Özgen, Omid Abdollahi Aghdam, and Hazim Kemal Ekenel. Text-to-painting on a large variance dataset with sequential generative adversarial networks. In *2020 28th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2020.

[77] Zhiqiang Zhang, Wenxin Yu, Ning Jiang, and Jinjia Zhou. Text to image synthesis with erudite generative adversarial networks. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 2438–2442. IEEE, 2021.

[78] Ming Tao, Songsong Wu, Xiaofeng Zhang, and Cailing Wang. Dcfgan: Dynamic convolutional fusion generative adversarial network for text-to-image synthesis. In *2020 IEEE International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*, volume 1, pages 1250–1254. IEEE, 2020.

[79] Kenan Emir Ak, Joo Hwee Lim, Jo Yew Tham, and Ashraf Kassim. Semantically consistent hierarchical text to fashion image synthesis with an enhanced-attentional generative adversarial network. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3121–3124. IEEE, 2019.

[80] Amin Fadaeddini, Babak Majidi, and Mohammad Eshghi. A case study of generative adversarial networks for procedural synthesis of original textures in video games. In *2018 2nd National and 1st International Digital Games Research Conference: Trends, Technologies, and Applications (DGRC)*, pages 118–122. IEEE, 2018.

[81] Ganesh Ravindra Padalkar, Shivani Dinkar Patil, Mukta Mallikarjun Hegadi, and Nikita Kailash Jaybhaye. Drug discovery using generative adversarial network with reinforcement learning. In *2021 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–3. IEEE, 2021.

[82] Mengdi Xu, Jiandong Cheng, Yirong Liu, and Wei Huang. Deepgan: Generating molecule for drug discovery based on generative adversarial network. In *2021 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2021.

[83] Daiki Yorioka, Hyunho Kang, and Keiichi Iwamura. Data augmentation for deep learning using generative adversarial networks. In *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, pages 516–518. IEEE, 2020.

[84] Kalpana Devi Bai Mudavathu, MVP Chandra Sekhara Rao, and KV Ramana. Auxiliary conditional generative adversarial networks for image data set augmentation. In *2018 3rd International Conference on Inventive Computation Technologies (ICICT)*, pages 263–269. IEEE, 2018.

[85] Foji Chen, Feng Zhu, Qingxiao Wu, Yingming Hao, Yunge Cui, and Ende Wang. Infrared images augmentation based on images generation with generative adversarial networks. In *2019 IEEE International Conference on Unmanned Systems (ICUS)*, pages 62–66. IEEE, 2019.

[86] Yusuke Naritomi and Takanori Adachi. Data augmentation of high frequency financial data using generative adversarial network. In *2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, pages 641–648. IEEE, 2020.

[87] Shota Haradal, Hideaki Hayashi, and Seiichi Uchida. Biosignal data augmentation based on generative adversarial networks. In *2018 40th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*, pages 368–371. IEEE, 2018.

[88] Yuanming Li, Bonhwa Ku, Gwantae Kim, Jae-Kwang Ahn, and Hanseok Ko. Seismic signal synthesis by generative adversarial network with gated convolutional neural network structure. In *IGARSS 2020-2020 IEEE International Geoscience and Remote Sensing Symposium*, pages 3857–3860. IEEE, 2020.

[89] Junkai Liang and Wenyuan Tang. Sequence generative adversarial networks for wind power scenario generation. *IEEE Journal on Selected Areas in Communications*, 38(1):110–118, 2019.

[90] Yufan Zhang, Qian Ai, Fei Xiao, Ran Hao, and Tianguang Lu. Typical wind power scenario generation for multiple wind farms using conditional improved wasserstein generative adversarial network. *International Journal of Electrical Power & Energy Systems*, 114:105388, 2020.

[91] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[92] Congmei Jiang, Yongfang Mao, Yi Chai, Mingbiao Yu, and Songbing Tao. Scenario generation for wind power using improved generative adversarial networks. *IEEE Access*, 6:62193–62203, 2018.

[93] Jinhai Liu, Fuming Qu, Xiaowei Hong, and Huaguang Zhang. A small-sample wind turbine fault detection method with synthetic fault data using generative adversarial nets. *IEEE Transactions on Industrial Informatics*, 15(7):3877–3888, 2018.

[94] Chao Ren and Yan Xu. A fully data-driven method based on generative adversarial networks for power system dynamic security assessment with missing data. *IEEE Transactions on Power Systems*, 34(6):5044–5052, 2019.

[95] Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. Time-series generative adversarial networks. *Advances in neural information processing systems*, 32, 2019.

[96] Yize Chen, Yishen Wang, Daniel Kirschen, and Baosen Zhang. Model-free renewable scenario generation using generative adversarial networks. *IEEE Transactions on Power Systems*, 33(3):3265–3275, 2018.

[97] Hu Wei, Zhang Hongxuan, Dong Yu, Wang Yiting, Dong Ling, and Xiao Ming. Short-term optimal operation of hydro-wind-solar hybrid system with improved generative adversarial networks. *Applied Energy*, 250:389–403, 2019.

[98] Süleyman Aslan, Uğur Güdükbay, B Uğur Töreyin, and A Enis Çetin. Early wildfire smoke detection based on motion-based geometric image transformation and deep convolutional generative adversarial networks. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8315–8319. IEEE, 2019.

[99] Shuangyu You. Pcb defect detection based on generative adversarial network. In *2022 2nd International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, pages 557–560. IEEE, 2022.

[100] Tolga Ergen and Suleyman S Kozat. Online training of lstm networks in distributed systems for variable length data sequences. *IEEE transactions on neural networks and learning systems*, 29(10):5159–5165, 2017.

[101] Taoying Li, Miao Hua, and XU Wu. A hybrid cnn-lstm model for forecasting particulate matter (pm2. 5). *Ieee Access*, 8:26933–26940, 2020.

[102] Jiawen Jiang, Haiyang Zhang, Chenxu Dai, Qingjuan Zhao, Hao Feng, Zhanlin Ji, and Ivan Ganchev. Enhancements of attention-based bidirectional lstm for hybrid automatic text summarization. *IEEE Access*, 9:123660–123671, 2021.

[103] Hamed Jelodar, Yongli Wang, Rita Orji, and Shucheng Huang. Deep sentiment classification and topic discovery on novel coronavirus or covid-19 online discussions: Nlp using lstm recurrent neural network approach. *IEEE Journal of Biomedical and Health Informatics*, 24(10):2733–2742, 2020.

[104] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[105] Dhaval Dalal, Anamitra Pal, and Philip Augustin. Representative scenarios to capture renewable generation stochasticity and cross-correlations. *arXiv preprint arXiv:2202.03588*, 2022.

[106] Ieee 30 bus system. `https://icseg.iti.illinois.edu/ieee-30-bus-system/`. Accessed: 2022-04-12.

[107] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[108] Ray D. Zimmerman and Carlos E. Murillo-Sánchez. Matpower, June 2019. URL `https://doi.org/10.5281/zenodo.3251119`.

APPENDIX A

CODE FOR TRAINING GAN (PYTHON)

```
import torch
import numpy as np
from matplotlib import pyplot as plt
from torch import nn, optim, Tensor
from torch.autograd.variable import Variable
from torchvision import transforms, datasets
from torch.utils.data import Dataset, DataLoader, TensorDataset
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from scipy.special import kl_div, rel_entr
import time
import random
import pickle

#setting which GPU to use
import os
os.environ['CUDA_VISIBLE_DEVICES']='1'

#checking GPU status
available_gpus = [torch.cuda.device(i) for i in
   range(torch.cuda.device_count())]
available_gpus
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
torch.cuda.is_available()
torch.cuda.current_device()
torch.cuda.device_count()

#setting hyperparameters
num_epochs = 2000 #epochs
learning_rate = 0.0001 #learning rate
gen_input_size = 256 #latent space
input_size = 2 #number of features
hidden_size = 128 #number of features in hidden state
num_layers = 3 #number of stacked lstm layers
num_classes = 1 #number of output classes
seq_length = 24*1 #sequence length
batch_size = 100 #batch size
dropout = 0.2 #dropout
discriminator_cycles = 3
n_labels = 3 #length of the label vector


#loading the data
data_summer = pd.read_csv("data/summer_data.csv")
data_summer = data_summer[['Solar', 'Load']]
data_shoulder = pd.read_csv("data/shoulder_data.csv")
data_shoulder = data_shoulder[['Solar', 'Load']]
data_winter = pd.read_csv("data/winter_data_2.csv")
data_winter = data_winter[['Solar', 'Load']]
labels_summer = pd.read_csv("data/DTW_Summer_Solar+Load.csv")
labels_summer = labels_summer[['505']]
labels_shoulder = pd.read_csv("data/DTW_Shoulder_Solar+Load.csv")
labels_shoulder = labels_shoulder[['372']]
labels_winter = pd.read_csv("data/DTW_Winter_Solar+Load.csv")
labels_winter = labels_winter[['604']]
data_summer.head(8)

#converting the data to numpy arrays
data_summer = data_summer.to_numpy()
data_shoulder = data_shoulder.to_numpy()
data_winter = data_winter.to_numpy()
labels_summer = labels_summer.to_numpy()
labels_shoulder = labels_shoulder.to_numpy()
labels_winter = labels_winter.to_numpy()

#concatenating all the seasonal data into one array
data_whole = np.concatenate((data_summer, data_shoulder), axis=0)
print(data_whole.shape)
data_whole = np.concatenate((data_whole, data_winter), axis=0)
print(data_whole.shape)

#making sequences of data of length = seq_length
data_modified = data_whole.reshape(-1, seq_length, input_size)
print(data_modified.shape)

#segrating the scaled data into normal/abnormal classes according to DTW
    cutoff
y_summer = np.zeros(len(labels_summer))
y_shoulder = np.zeros(len(labels_shoulder))
y_winter = np.zeros(len(labels_winter))
normal_summer = 0
normal_summer_data = []
abnormal_summer = 0
abnormal_summer_data = []
normal_shoulder = 0
normal_shoulder_data = []
abnormal_shoulder = 0
abnormal_shoulder_data = []
normal_winter = 0
normal_winter_data = []
abnormal_winter = 0
abnormal_winter_data = []
index = 0

for i in range(len(y_summer)):
    if labels_summer[i] <= 0.8:
        y_summer[i] = 1
        normal_summer += 1
        normal_summer_data.append(data_modified[index, :, 0:2])
        index += 1
    else:
        y_summer[i] = 2
        abnormal_summer += 1
        abnormal_summer_data.append(data_modified[index, :, 0:2])
        index += 1

for i in range(len(y_shoulder)):
    if labels_shoulder[i] <= 1.41:
        y_shoulder[i] = 3
        normal_shoulder += 1
        normal_shoulder_data.append(data_modified[index, :, 0:2])
        index += 1
    else:
        y_shoulder[i] = 4
        abnormal_shoulder += 1
        abnormal_shoulder_data.append(data_modified[index, :, 0:2])
        index += 1

for i in range(len(y_winter)):
    if labels_winter[i] <= 1.05:
        y_winter[i] = 5
        normal_winter += 1
        normal_winter_data.append(data_modified[index, :, 0:2])
        index += 1
    else:
        y_winter[i] = 6
        abnormal_winter += 1
        abnormal_winter_data.append(data_modified[index, :, 0:2])
        index += 1


y_summer = np.asarray(y_summer)
y_shoulder = np.asarray(y_shoulder)
y_winter = np.asarray(y_winter)

y_whole = np.concatenate((y_summer, y_shoulder), axis=0)
y_whole = np.concatenate((y_whole, y_winter), axis=0)
y_whole = np.asarray(y_whole)
print(y_whole.shape)


normal_summer_data = np.asarray(normal_summer_data)
normal_shoulder_data = np.asarray(normal_shoulder_data)
normal_winter_data = np.asarray(normal_winter_data)
abnormal_summer_data = np.asarray(abnormal_summer_data)
abnormal_shoulder_data = np.asarray(abnormal_shoulder_data)
abnormal_winter_data = np.asarray(abnormal_winter_data)

print(index)
```

```
#shuffling the data before feeding it to GAN
rng_state = np.random.get_state()
np.random.shuffle(data_modified)
np.random.set_state(rng_state)
np.random.shuffle(y_whole)

#encoding the labels
def binary_encoding(y, n_labels):
    #y_one_hot = Variable(torch.zeros(len(y), 3))
    y_binary_encoding = np.zeros((len(y), 3))
    for i in range(len(y)):
        if y[i] == 1:
            y_binary_encoding[i] = [0, 0, 0]
        if y[i] == 2:
            y_binary_encoding[i] = [0, 0, 1]
        if y[i] == 3:
            y_binary_encoding[i] = [0, 1, 0]
        if y[i] == 4:
            y_binary_encoding[i] = [1, 0, 0]
        if y[i] == 5:
            y_binary_encoding[i] = [0, 1, 1]
        if y[i] == 6:
            y_binary_encoding[i] = [1, 0, 1]
    return torch.Tensor(y_binary_encoding)

labels = binary_encoding(y_whole, n_labels)
print(labels)

#making the dataloader
dataset = TensorDataset(Tensor(data_modified), Tensor(labels))
data_loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
    shuffle=True)

num_batches = len(data_loader)
print(num_batches)

#defining the discriminator
class DiscriminatorNet(torch.nn.Module):

    def __init__(self, num_classes, input_size, hidden_size, num_layers,
        seq_length, dropout, n_labels):
        super(DiscriminatorNet, self).__init__()

        self.num_classes = num_classes #number of classes
        self.num_layers = num_layers #number of layers
        self.input_size = input_size #input size
        self.hidden_size = hidden_size #hidden state
        self.seq_length = seq_length #sequence length
        self.dropout = dropout #dropout ratio
        self.n_labels = n_labels #no of labels
        self.lstm = nn.LSTM(input_size=(input_size+n_labels),
           hidden_size=hidden_size,
                        num_layers=num_layers, batch_first=True, dropout =
                           dropout)
        self.fc = nn.Sequential(nn.Linear(hidden_size, num_classes),
           nn.Sigmoid())

    def forward(self, x):
        h_0 = Variable(torch.zeros(self.num_layers, x.size(0),
           self.hidden_size)).to(device) #hidden state
        c_0 = Variable(torch.zeros(self.num_layers, x.size(0),
           self.hidden_size)).to(device) #internal state
        #Propagate input through LSTM
        output, (hn, cn) = self.lstm(x, (h_0, c_0)) #lstm with input,
           hidden, and cell state
        hn = hn[-1,:,:] #reshaping the data for dense layer next
        out = self.fc(hn) #Final Output
        return out

#defining the generator
class GeneratorNet(torch.nn.Module):

    def __init__(self, num_classes, input_size, output_size, hidden_size,
       num_layers, seq_length, dropout, n_labels):
        super(GeneratorNet, self).__init__()

        n_out = output_size*seq_length
        self.num_classes = num_classes #number of classes
        self.num_layers = num_layers #number of layers
        self.input_size = input_size #input size
        self.hidden_size = hidden_size #hidden state
        self.seq_length = seq_length #sequence length
        #self.dropout = dropout ratio
        self.n_labels = n_labels

        self.lstm = nn.LSTM(input_size=(input_size+n_labels),
           hidden_size=hidden_size,
                        num_layers=num_layers, batch_first=True, dropout =
                           dropout) #lstm

        self.fc = nn.Sequential(nn.Linear(hidden_size, n_out), nn.Tanh())
        #self.fc = nn.Sequential(nn.Linear(hidden_size*num_layers, n_out))

    def forward(self, x):
        h_0 = Variable(torch.zeros(self.num_layers, x.size(0),
           self.hidden_size)).to(device) #hidden state
        c_0 = Variable(torch.zeros(self.num_layers, x.size(0),
           self.hidden_size)).to(device) #internal state
        # Propagate input through LSTM
        #c = self.label_emb(labels)
        #x = torch.cat([x, c], 1)
        output, (hn, cn) = self.lstm(x, (h_0, c_0)) #lstm with input,
           hidden, and internal state
        hn = hn[-1,:,:] #reshaping the data for dense layer next
        out = self.fc(hn) #Final Output
        return out

# Function for geneting noise
def noise(size):
    #n = Variable(torch.randn(size, 100))
    n = Variable(torch.randn(size, seq_length, gen_input_size))
    if torch.cuda.is_available(): return n.cuda()
    return n

#defining the generator and the discriminator
discriminator = DiscriminatorNet(num_classes, input_size, hidden_size,
   num_layers, seq_length, dropout, n_labels)
generator = GeneratorNet(num_classes, gen_input_size, input_size,
   hidden_size, num_layers, seq_length, dropout, n_labels)
if torch.cuda.is_available():
    discriminator.cuda()
    generator.cuda()

print(discriminator)
print()
print(generator)

#Defining the optimizers
d_optimizer = optim.Adam(discriminator.parameters(), lr= learning_rate)
g_optimizer = optim.Adam(generator.parameters(), lr= learning_rate)

# Loss function
loss = nn.BCELoss()

#defining the real and fake data targets to compute the loss against
def real_data_target(size):
    '''
    Tensor containing ones, with shape = size
    '''
    data = Variable(torch.ones(size, 1))
    if torch.cuda.is_available(): return data.cuda()
    return data

def fake_data_target(size):
    '''
    Tensor containing zeros, with shape = size
    '''
    data = Variable(torch.zeros(size, 1))
    if torch.cuda.is_available(): return data.cuda()
    return data

#defining the discriminator train cycle
def train_discriminator(optimizer, real_data, fake_data):
    # Reset gradients
    optimizer.zero_grad()

    # 1.1 Train on Real Data
    prediction_real = discriminator(real_data)
    # Calculate error and backpropagate
    error_real = loss(prediction_real, real_data_target(real_data.size(0)))
    error_real.backward()

    # 1.2 Train on Fake Data
    prediction_fake = discriminator(fake_data)
    # Calculate error and backpropagate
    error_fake = loss(prediction_fake, fake_data_target(real_data.size(0)))
    error_fake.backward()

    # 1.3 Update weights with gradients
    optimizer.step()

    # Return error
    return error_real + error_fake, prediction_real, prediction_fake

#defining the generator train cycle
def train_generator(optimizer, fake_data):
    # 2. Train Generator
    # Reset gradients
    optimizer.zero_grad()
    # Sample noise and generate fake data
    prediction = discriminator(fake_data)
    # Calculate error and backpropagate
    error = loss(prediction, real_data_target(prediction.size(0)))
    error.backward()
    # Update weights with gradients
    optimizer.step()
    # Return error
    return error

#main training code
start = time.time()
for epoch in range(num_epochs):

    for n_batch, (real_batch, real_label) in enumerate(data_loader):

        if torch.cuda.is_available(): real_data = real_batch.cuda()
        else: real_data = Variable(real_batch).float()
        if torch.cuda.is_available(): real_label = (real_label.unsqueeze(1)
           * torch.ones([real_data.size(0), seq_length, n_labels])).cuda()
        else: real_label = Variable(real_label.unsqueeze(1) *
           torch.ones([real_data.size(0), seq_length, n_labels])).float()
        real_data = torch.cat([real_data, real_label], dim = 2).cuda()

        #Training Discriminator
        for _ in range(discriminator_cycles):
           #Generate fake data
           fake_data = generator(torch.cat([noise(real_data.size(0)),
               real_label], dim = 2))
           fake_data = torch.reshape(fake_data ,(-1, seq_length,
               input_size)).to(device)
           fake_data = torch.cat([fake_data, real_label], dim = 2)

           d_error, d_pred_real, d_pred_fake =
               train_discriminator(d_optimizer,
                                                                real_data,
                                                                    fake_data)

        #Training Generator
        #Generate fake data
        fake_data = generator(torch.cat([noise(real_data.size(0)),
           real_label], dim = 2))
        fake_data = torch.reshape(fake_data ,(-1, seq_length,
           input_size)).to(device)
        fake_data = torch.cat([fake_data, real_label], dim = 2)

        g_error = train_generator(g_optimizer, fake_data)

    print("Epoch: %d, discriminator loss: %1.5f, generator loss: %1.5f" %
        (epoch, d_error, g_error))
end = time.time()
print(f"Runtime of the program is {end - start} seconds")

#runtime in hours
runtime = (end-start)/60
print(runtime, " minutes")
```

APPENDIX B

CODE FOR RESULT VALIDATION USING OPF (MATLAB)

```
clc

hourList = [1:1:24];
plotHourList = [7 12 17];
voltageMag = 8;
volatgeAng = 9;
SolarToLoadRatio = 0.6;
wassDistList = [];
costList = [];
costListPerHour = [];
wassDistListPerHour = [];
plotOrNot = false; %for plotting voltage angle PFDs
plotOrNot2 = false; %for plotting duck curve
saveOrNot = false; %for saving figures
saveOrNot2 = false; %for saving OPF cost, Wass Dist

seasonFlag = 'Winter'; %Shoulder, Summer, Winter
normalFlag = 'Abnormal'; %Normal, Abnormal
normalFlagLowerCase = 'abnormal'; %normal, abnormal

str = sprintf('%s - %s - Solar - 900 - New DTW.csv', seasonFlag,
    normalFlag);
dataSolarTotal = readmatrix(str);
str = sprintf('%s - %s - Load - 900 - New DTW.csv', seasonFlag,
    normalFlag);
dataLoadTotal = readmatrix(str);
str = sprintf('%s - %s - Solar - Real - New DTW 3.csv', seasonFlag,
    normalFlag);
dataSolarReal = readmatrix(str);
str = sprintf('%s - %s - Load - Real - New DTW 3.csv', seasonFlag,
    normalFlag);
dataLoadReal = readmatrix(str);

mpopt = mpoption('verbose', 0); %setting options for opf

for j = 0:29

    totalWassDist = 0;
    totalCost = 0;
    totalCostReal = 0;
    costListRealPerHour = [];
    dataSolar = dataSolarTotal(((j*30)+1):((j*30)+30),:); %reading 30
        profiles for 30 buses
    dataLoad = dataLoadTotal(((j*30)+1):((j*30)+30),:);

    temp = 1;
    if (plotOrNot == true)
        figure('Position', [400, 50, 800, 800]);
    end
```

```
for hour = hourList
    %loading the data and the test case

    testcase = loadcase(case_ieee30_modified);
    testcaseReal = loadcase(case_ieee30_modified);


    for i = 1:30 %no of buses
        testcase.bus(i,3) = testcase.bus(i,3)*(dataLoad(i,hour) -
            (SolarToLoadRatio)*dataSolar(i,hour));
        testcaseReal.bus(i,3) =
            testcaseReal.bus(i,3)*(dataLoadReal(i,hour) -
            (SolarToLoadRatio)*dataSolarReal(i,hour));
    end

    results = runopf(testcase, mpopt);
    results_real = runopf(testcaseReal, mpopt);

    %objetive funtion value
    results.f;
    results_real.f;

    [f1,x1] = ksdensity(results.bus(:,volatgeAng)); %calculating pdf
        using normal kernel estimation
    [f2,x2] = ksdensity(results_real.bus(:,volatgeAng));

    totalWassDist = totalWassDist +
        py.scipy.stats.wasserstein_distance(x1, x2, f1, f2);
    totalCost = totalCost + results.f;
    totalCostReal = totalCostReal + results_real.f;
    costListPerHour = [costListPerHour, results.f];
    costListRealPerHour = [costListRealPerHour, results_real.f];
    wassDistListPerHour = [wassDistListPerHour,
        py.scipy.stats.wasserstein_distance(x1, x2, f1, f2)];

    if (ismember(hour,plotHourList) == true) && (plotOrNot == true)

        subplot(3,1,temp)

        area(x1,f1, 'facecolor','b', 'LineWidth', 0.8, 'FaceAlpha', 0.3)
        xlim([-10 10])
        ylim([0 2.75])
        hold on

        area(x2,f2, 'facecolor','r', 'LineWidth', 0.8, 'FaceAlpha', 0.3)
        hold off
        str = sprintf('%s - %s - Correlated - Hour %d', seasonFlag,
            normalFlag, hour);
        legend('Generated', 'Base Case')
        ylabel('Density')
```

```
            xlabel('Voltage Angle')
            title(str)
            temp = temp + 1;
        end
    end

    if (saveOrNot == true)
        str = sprintf('Figures/%s_%s_correlated/%s - %s - Correlated -
            %d.png', seasonFlag, normalFlagLowerCase, seasonFlag,
            normalFlag, j);
        saveas(gcf, str)
    end

    wassDistList = [wassDistList, totalWassDist];
    costList = [costList, totalCost/24];
    costReal = totalCostReal/24;

    if plotOrNot2 == true
        figure
        hold on
        for i = [1:1:30]
            pp = dataLoad(i,:) - dataSolar(i,:);
            plot(pp, 'LineWidth', 1)
            str = sprintf('%s - %s - Correlated - Duck Curve', seasonFlag,
                normalFlag);
            title(str)
            ylabel('MW')
            xlabel('hours')
        end
    end
end

costListPerHour = reshape(costListPerHour, [24,30]).';
wassDistListPerHour = reshape(wassDistListPerHour, [24,30]).';
costListPerHourAvg = mean(costListPerHour);
wassDistListPerHourAvg = mean(wassDistListPerHour);

if saveOrNot2 == true
    str = sprintf('Data/Cost - %s_%s_correlated.csv', seasonFlag,
        normalFlagLowerCase);
    writematrix(costListPerHour, str);
    str = sprintf('Data/Wass_Dist - %s_%s_correlated.csv', seasonFlag,
        normalFlagLowerCase);
    writematrix(wassDistListPerHour, str);
end

disp('Correlated')


voltageMag = 8;
```

```
volatgeAng = 9;
wassDistList2 = [];
costList2 = [];
costListPerHour2 = [];
wassDistListPerHour2 = [];

str = sprintf('%s - %s - Solar Only - 900 - New DTW.csv', seasonFlag,
    normalFlag);
dataSolarTotal = readmatrix(str);
str = sprintf('%s - %s - Load Only - 900 - New DTW.csv', seasonFlag,
    normalFlag);
dataLoadTotal = readmatrix(str);
str = sprintf('%s - %s - Solar Only- Real - New DTW.csv', seasonFlag,
    normalFlag);
dataSolarReal = readmatrix(str);
str = sprintf('%s - %s - Load Only- Real - New DTW.csv', seasonFlag,
    normalFlag);
dataLoadReal = readmatrix(str);

mpopt = mpoption('verbose', 0); %setting options for opf

for j = 0:29

    totalWassDist = 0;
    totalCost = 0;
    totalCostReal2 = 0;
    costListRealPerHour2 = [];
    dataSolar = dataSolarTotal(((j*30)+1):((j*30)+30),:); %reading 30
        profiles for 30 buses
    dataLoad = dataLoadTotal(((j*30)+1):((j*30)+30),:);

    temp = 1;
    if (plotOrNot == true)
        figure('Position', [400, 50, 800, 800]);
    end
    for hour = hourList
        %loading the data and the test case

        testcase = loadcase(case_ieee30_modified);
        testcaseReal = loadcase(case_ieee30_modified);

        for i = 1:30 %no of buses
            testcase.bus(i,3) = testcase.bus(i,3)*(dataLoad(i,hour) -
                (SolarToLoadRatio)*dataSolar(i,hour));
            testcaseReal.bus(i,3) =
                testcaseReal.bus(i,3)*(dataLoadReal(i,hour) -
                (SolarToLoadRatio)*dataSolarReal(i,hour));
        end

        results = runopf(testcase, mpopt);
        results_real = runopf(testcaseReal, mpopt);

        %objetive funtion value
        results.f;
        results_real.f;

        [f1,x1] = ksdensity(results.bus(:,volatgeAng)); %calculating pdf
            using normal kernel estimation
        [f2,x2] = ksdensity(results_real.bus(:,volatgeAng));

        totalWassDist = totalWassDist +
            py.scipy.stats.wasserstein_distance(x1, x2, f1, f2);
        totalCost = totalCost + results.f;
        totalCostReal2 = totalCostReal2 + results_real.f;
        costListPerHour2 = [costListPerHour2, results.f];
        costListRealPerHour2 = [costListRealPerHour2, results_real.f];
        wassDistListPerHour2 = [wassDistListPerHour2,
            py.scipy.stats.wasserstein_distance(x1, x2, f1, f2)];

        if (ismember(hour,plotHourList) == true) && (plotOrNot == true)

            subplot(3,1,temp)

            area(x1,f1, 'facecolor','b', 'LineWidth', 0.8, 'FaceAlpha', 0.3)
            xlim([-10 10])
            ylim([0 2.75])
            hold on

            area(x2,f2, 'facecolor','r', 'LineWidth', 0.8, 'FaceAlpha', 0.3)
            hold off
            str = sprintf('%s - %s - Not Correlated - Hour %d', seasonFlag,
                normalFlag, hour);
            legend('Generated', 'Base Case')
            ylabel('Density')
            xlabel('Voltage Angle')
            title(str)
            temp = temp + 1;
        end
    end

    if (saveOrNot == true)
        str = sprintf('Figures/%s_%s_not_correlated/%s - %s - Not
            Correlated - %d.png', seasonFlag, normalFlagLowerCase,
            seasonFlag, normalFlag, j);
        saveas(gcf, str)
    end

    wassDistList2 = [wassDistList2, totalWassDist];
    costList2 = [costList2, totalCost/24];
    costReal2 = totalCostReal2/24;
```

```
    if plotOrNot2 == true
        figure
        hold on
        for i = [1:1:30]
            pp = dataLoad(i,:) - dataSolar(i,:);
            plot(pp, 'LineWidth', 1)
            str = sprintf('%s - %s - Not Correlated - Duck Curve',
                seasonFlag, normalFlag);
            title(str)
            ylabel('MW')
            xlabel('hours')
        end
    end
end
costListPerHour2 = reshape(costListPerHour2, [24,30]).';
wassDistListPerHour2 = reshape(wassDistListPerHour2, [24,30]).';
costListPerHourAvg2 = mean(costListPerHour2);
wassDistListPerHourAvg2 = mean(wassDistListPerHour2);

if saveOrNot2 == true
    str = sprintf('Data/Cost - %s_%s_not_correlated.csv', seasonFlag,
        normalFlagLowerCase);
    writematrix(costListPerHour2, str);
    str = sprintf('Data/Wass_Dist - %s_%s_not_correlated.csv', seasonFlag,
        normalFlagLowerCase);
    writematrix(wassDistListPerHour2, str);
end

disp('Not correlated')
```