Multistep Multivariate Scenario Generation and Forecasting for Power Systems using Machine Learning

by

Mohammed Alhazmi

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

Approved October 2021 by the
Graduate Supervisory Committee:

Anamitra Pal, Chair
Raja Ayyanar
Keith Holbert

ARIZONA STATE UNIVERSITY

December 2021

ABSTRACT

The penetration of renewable energy in the power system has grown considerably in the past few years. While this use may come with an abundance of advantages, it also introduces new challenges in operating the 100+ years old electrical network. Fundamentally, the power system relies on a real-time balance of generation and demand. However, renewable resources such as solar and wind farms are not available throughout the day. Furthermore, they introduce temporal variability to the generation process due to metrological factors, making the balance of generation and demand precarious. Utilities use standby units with reserve power and high ramp-up, ramp-down capabilities to ensure balance. However, such solutions can be very costly. An accurate scenario generation and forecasting of the stochastic variables (load and renewable resources) can help reduce the cost of these solutions.

The goal of this research is to solve the scenario generation and forecasting problems using state-of-the-art machine learning techniques and algorithms. The training database is created using publicly available data obtained from NREL and the Texas-2000 bus system. The IEEE-30 bus system is used as the test system for the analysis conducted here. The conventional generators of this system are replaced with solar farms and wind farms. The ability of four machine learning algorithms in addressing the scenario generation and forecasting problems are investigated using appropriate metrics.

The first machine learning algorithm is the convolutional neural network (CNN). It is found to be well-suited for the scenario generation problem. However, its inability to capture certain intricate details about the different variables was identified as a possible drawback. The second algorithm is the long-short term memory-variational auto-encoder (LSTM-VAE). It generated scenarios that are very similar to the actual scenarios indicating that it is suitable for solving the forecasting problem. The third algorithm is the conditional generative adversarial network (C-GAN). It was extremely effective in generating scenarios when the number of variables were small. However, its scalability was found to be a concern. The fourth algorithm is the spatio-temporal graph convolutional network (STGCN). It was found to generate representative correlated scenarios effectively.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Anamitra Pal for providing me with an opportunity to pursue research under his guidance as a member of his research group, the Pal Lab. He has guided me at all stages of my research journey and has been a constant source of motivation. I would like to extend my gratitude to Dr. Keith Holbert and Dr. Raja Ayyanar for agreeing to be a member of my thesis committee and helping me in my research with valuable feedback and insights. I thank all the members of the Pal Lab for the help that they have offered me during my master's journey. A special thanks to Behrouz Azimian and Dhaval Dalal for their valuable contributions to select portions of my research.

I would like to thank both my parents and my sisters for their support during this journey. They have been a constant source of joy and positive energy. I would like to thank Fatmah Alyami for being a source of joy, laughter, and the goodness in humanity

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

Over the past 50 years, penetration of renewable energy has been increasing in the electricity supply mix. Figure 1-1 shows renewable generation in 2021 for some of the countries of the world. Renewable energy has increased by more than 8% compared to 2020 to reach an all-time high of 8,300 TWh in 2021[1]. China solely is responsible for 50% of this global growth with more than double the renewable generation of the United States and Europe. This accelerated growth is the highest since the 1970s and it is largely due to the advancement in wind and solar generation as they constitute two-thirds of this rapid growth.

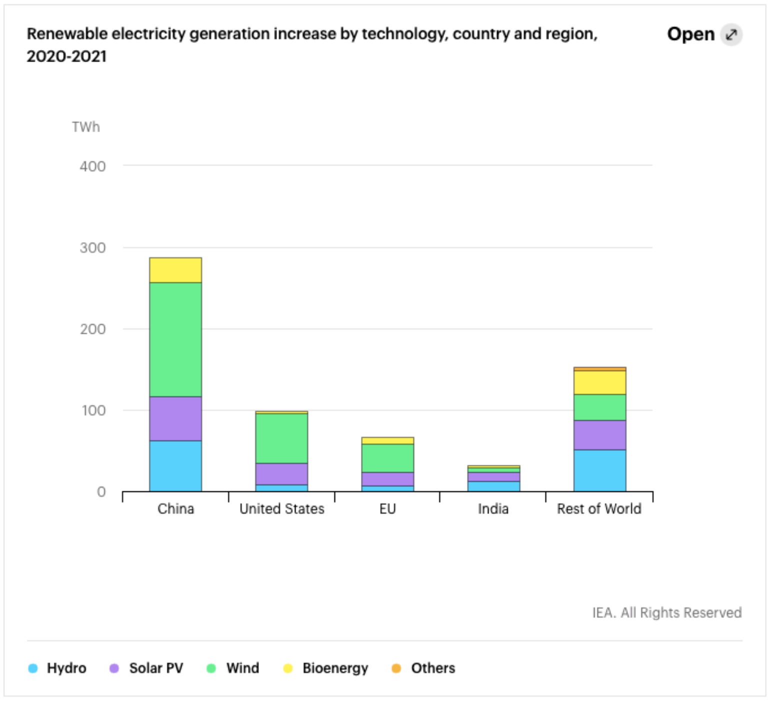


Figure 1-1: Renewable electricity generation increase by technology, country and region, 2020-2021 [1]

In order for the electric grid to accommodate the large and growing penetration of renewable energy, it has to be more flexible. The old framework where the load is changing, and the conventional generators are ready to follow that change is no longer valid. Renewable energies such as solar farms and wind farms have their own *ebbs and flows* irrespective of the load behavior. This puts a huge strain on the 100+ years old power network that was originally designed around the idea of controllable and predictable generation. If nothing is done, this *new* variability on the generation side can result in cascading failures, including blackouts [2][13].

The rapid growth and utilization of renewable resource brings about many difficulties and challenges in operating and controlling the electrical network effectively and efficiently. These difficulties include low fault ride through capabilities, high fault currents[14], low power quality, low system inertia, and high variability [15]. In this work, we address the high variability difficulty by generating accurate forecasts of generation and demand so that the power system operators can use them to operate the network safely and allocate planned resources appropriately. However, this is not easy because of the inherent nature and stochasticity of renewable resources such as wind and solar – their output depends on metrological and geographical factors which have no apparent relation to the load demand.

1.1 Motivation

The variability and intermittency of renewable energy poses a challenge to the operation of the power system using conventional (currently used) methods. For example, power system operators presently use reserve power and units with fast ramp-up/down capabilities to compensate for the fast fluctuations of solar, wind, and other renewable sources[15]. While these solutions may work now, they will incur a huge investment as more renewable resources penetrate the gird with widespread adoption of wind and solar. Accurate forecasting of renewable generation can have financial savings and save lives alike. Being able to predict and forecast renewable energies allows (a) designers to design plants economically and safely, and (b) operators to optimally plan and manage network devices and resources. Finally, the reduction of the uncertainties and variability of renewable resources improves the stability of the network and allows for higher penetration level of renewables [7].

Realization of the different scenarios that can be generated by diverse renewable resources and loads allows for better system reliability. By considering those representative and extreme scenarios, operators are able to better assess the behavior of the network. This enables them (the operators) to take the correct operational strategies which will avoid system collapse or cascading failures.

1.2 Objectives

The objective of this thesis is to utilize state-of-the-art machine learning algorithms for the problem of multistep, multivariate scenario generation and forecasting and evaluate the results of these algorithms. In order to achieve this objective, different algorithms must be integrated, and their architectures and objectives reformulated to include the physics of the electrical network. Some of those algorithms have been used in univariate scenario generation or forecasting. However, including more variables, extending the problem to include different time horizons, and scaling them for larger networks are some of the novel aspects of this study. The analysis conducted here also accounts for the spatial and temporal correlations existing between the different variables, which have often been ignored in prior research.

The advancement of machine learning and the computing power that we have today enables us to design models with complicated architectures that are able to approximate nonlinear relations and generalize these functions to unseen data. The different models designed will be tested to see if they are able to capture inherent details of the power system variables and the results will be compared using known metrics.

The goal of this thesis is to make the following contributions:

1. Reformulate existing machine learning models for multistep, multivariate scenario generation and forecasting.
2. Test scalability of the models.
3. Detailed evaluation of the results generated by the different models.
4. Investigate the pros and cons of the models.

1.3 Structure of the Thesis

The rest of the thesis is organized as follows.

Chapter 2 goes over the definitions of forecasting and scenario generation and explains the problem formulation of multistep, multivariate forecasting and scenario generation. It also reviews some of the prominent literature work that has been done on forecasting and scenario generation using statistical and machine learning algorithms.

Chapter 3 describes and elucidates some of the machine learning models used in this work. Specifically, it describes the inner workings of Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Generative Models (Generative

Adversarial Networks and Variational Auto-Encoders) and Graph Models (Graph Convolutional Neural Networks).

Chapter 4 builds the different machine learning models, generates results, and tests the performance of the models identified in Chapter 3.

Chapter 5 concludes the research and discusses the results and merits of each model used in this work. It also gives future directions for this work.

CHAPTER 2

PROBLEM FORMULATION AND LITERATURE REVIEW

2.1 Problem Formulation

In this work, we solve two problems both of which have unique attributes. The first is the *forecasting* problem. Forecasting in machine learning is the ability to use historical data to try and make informed estimates and predictions extrapolated from those historical data. Forecasting is a deterministic problem where using the same input over and over will generate the same expected output over and over since the input data has not changed. This can be defined as:

$$y = ax_1 + bx_2 + cx_3 \tag{2.1}$$

Where x_1, x_2, x_3 are the historical data and a, b, c are constants.

The second problem, namely the *scenario generation* problem, is a more challenging problem since the output is a collection of predictions. The same input can be used to generated different outputs. While (2.1) can still be used to predict the output, the assumption that a, b, c are constants is no longer valid as they are sampled from distributions, resulting in different outputs for the same input.

At the same time, in an electrical network we have different types of variables such as loads, conventional generators, solar farms, and wind farms. To include the impacts of these variables and make the two problems more realistic, we consider *multivariate* scenario generation and forecasting. This results in the following new formulation:

$$y_{t+1} = a(x_t + x_{t-1}) + b(w_t + w_{t-1}) + c(y_t + y_{t-1}) \qquad 2.2)$$

Where $x_{t,t-1}, w_{t,t-1}, y_{t,t-1}$ are the historical data for each of the variables and a, b, c are constants in the case of forecasting and sampled from a distribution in the case of scenario generation.

Day ahead planning and forecasting is an indispensable part of power system operations. It allows utilities to reserve and prepare units with high ramp up and ramp down capabilities to compensate for the uncertainties and the fluctuations that are inherent to renewable (mainly solar and wind) generation. Consequently, we reformulate the above problem to *multistep*, multivariate scenario generation and forecasting so that it is more suitable for power system operation. In this thesis, we focus on a 24-hour time horizon.

2.2 Literature Review

2.2.1 Forecasting

Univariate forecasting for solar, wind, and other renewable resources has been explored previously using statistical and machine learning algorithms. In[16], the use of physical model information, characteristics of PV panels, and simple statistical models was explored. However, the relative error exceeded 10% which was due to the use of bad forecasted metrological data and the need for more complex statistical model to account for the different intercorrelations between the input variables. In [17], the authors have also explored the use of metrological data and machine learning to forecast one-hour ahead solar output. The combination of long short-term memory (LSTM) and XGBoost models was shown to be a good candidate for addressing the problem effectively. The paper also tested the model with different input horizons (1-hour to 24-hours) and 1-hour look ahead.

In [18], the authors have addressed the forecasting problem by using deterministic and probabilistic models with the latter models improving the results of forecasting as they bode well with the stochastic nature of solar power. Authors of [19] used seasonal autoregressive integrated moving average (SARIMA) model for a multistep forecasting of solar output. autocorrelation coefficient (ACF) was used to determine the order of the terms in the model and the time lags needed to generate good predictions of solar output.

While deterministic models were able to perform relatively good for solar forecasting [18], the same cannot be said about wind power forecasting due to the higher variability and stochasticity in wind. In [20] a Markov Chain model was used to predict day-ahead wind power output. The results presented had high accuracies, but the model needed significant amount of data preprocessing and clustering based on windspeed. The use of deep belief networks (DBNs) for this purpose have also been considered. In [21], a DBN model was developed to forecast wind power at the rate of 10 minutes for 24 hours. K-means clustering was used to identify the largest sample in the data that has influence on the forecasting accuracy and only that sample was used.

While there is an abundant of research on solar or wind forecasting, only a few have addressed the two variables together. In [22], the use of a software called SPSS is used to forecast wind and solar outputs simultaneously. The paper does not provide any information about the machine learning models incorporated in the software to generate the forecasting results. Other machine learning algorithms that have been investigated to solve the multivariable problem coupled the algorithms with other techniques. For example, neural networks and fuzzy logic was used in [23] to solve long-term multivariate input and multivariate output effectively for both solar and wind.

However, the problem of multistep, multivariate forecasting for day-ahead operation that is scalable to real networks has not been looked into yet.

2.2.2 Scenario Generation

Scenario generation in a univariate setting, i.e., wind and solar separately, using machine learning algorithms has been researched extensively in[24]. It used improved conditional generative adversarial networks (C-GANs) with Wasserstein distance to generate univariate scenarios. Initially it used GANs to create representative scenarios of either solar or wind. For both cases, the model was trained separately on solar or wind datasets but not jointly. It then used C-GANs to generate extreme scenarios by categorizing the historical data based on the average power of each day.

The problem of multivariate scenario generation has not received as much attention. One of the most prominent works is [25] which developed a probabilistic model using copula joint distribution and Spearman rank. The model generated scenarios for two wind farms and two solar farms simultaneously.

Similar to the forecasting problem, multistep, multivariate scenario generation for day-ahead operation that is scalable to real networks has not been looked into yet.

CHAPTER 3

MACHINE LEARNING MODELS

3.1 Convolutional Neural Networks (CNNs)

CNNs are one of the most basic and widely used Artificial Intelligence (AI) architectures. They have gained popularity due to their ability to perform well in multitude of tasks such as classification of videos and images, natural language processing, and prediction in regression problems. They are also relatively easy to implement [26].

CNNs are a deep learning algorithm that takes a matrix as an input, e.g., $x \in \mathbb{R}^{H*W*D}$, where H, W, D are the height, width, and depth of the matrix, respectively, and outputs a 3-dimentional matrix with different channels. Each channel provides important yet distinct information about the input. The information embedded in each of the channels are the results of training learnable filters (kernels) as seen in Figure 3-1.

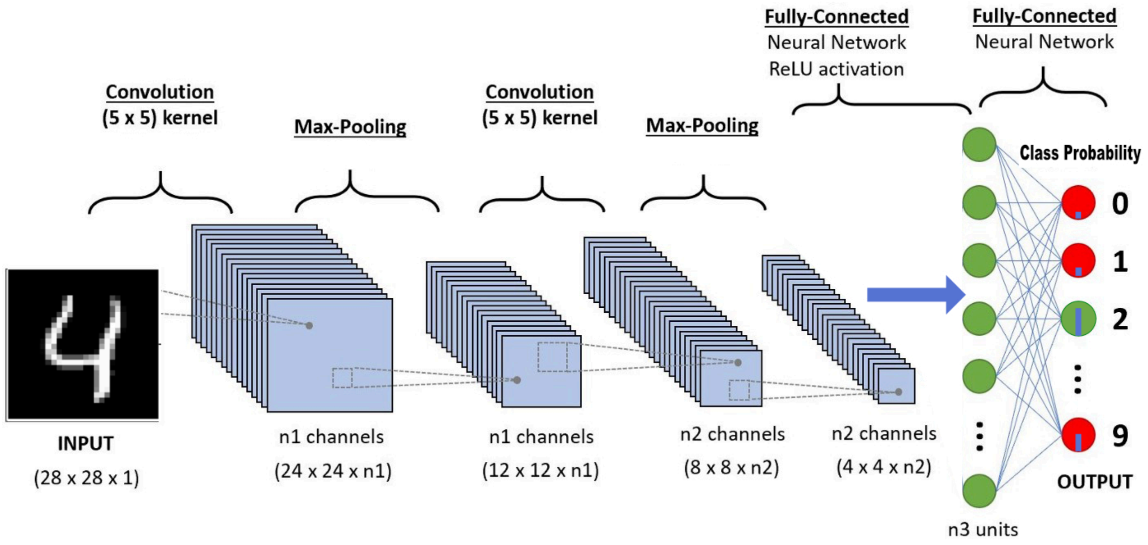


Figure 3-1: CNN Architecture

The filters are multiplied by the input sequentially and then moved based on the stride value (usually 1) until a smaller version of the initial input is created. This is done multiple times and each time the filters focus on a smaller section of the input and thereby extract higher-level information (see Figure 3-2).

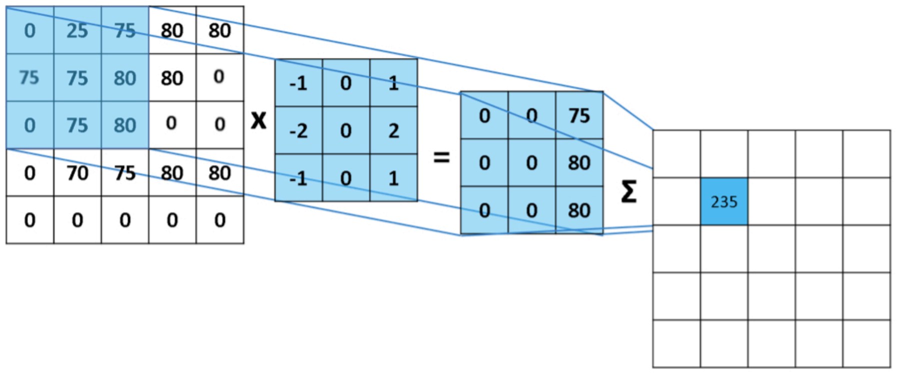


Figure 3-2: Filters or Kernels Applied to CNNs

CNNs are generally followed by fully connected layers in classification and regression problems as can be seen in Figure 3-1. The fully connected layers have their own learnable parameters (weights and biases). The output of the linear layers can be the raw output of the algorithm for a regression problem or a probability distribution of the classes in case of a classification problem.

There are two major parts to the learning process of machine learning algorithms, in general, and CNN, in particular. The first is the *feed-forward* phase where the input goes through the different layers (CNNs and fully connected layers alike), and the learnable parameters are applied to generate the output. The result is then compared to the true result and the error is calculated (usually) through the mean squared error (MSE). The

MSE calculates the squared difference between the true and predicted output over the samples.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_I - \hat{y}_i)^2 \tag{3.1}$$

During the second phase, called *back-propagation*, the learnable parameters are calculated and updated through techniques such as stochastic gradient descent (SGD). SGD updates the parameter for each training sample x and their corresponding label y with a regularized version of the newly learned parameters so that the distribution does not shift suddenly based on one sample of the training:

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x; y) \tag{3.2}$$

In (3.2), the partial derivative is taken of the layers with respect to their weights and multiplied by the learning rate to update the layers' new weights. Learning rate is the regularization mentioned earlier in this chapter that controls how much we want to learn. In Figure 3-3, learning rate is represented by the black arrow – a small learning rate will allow us to move slowly until the training loop is over and we still might be far away from the optimal solution. A large learning rate on the other hand, might let the weights (represented by the black dot) roll and pass the optimal solution back and forth.

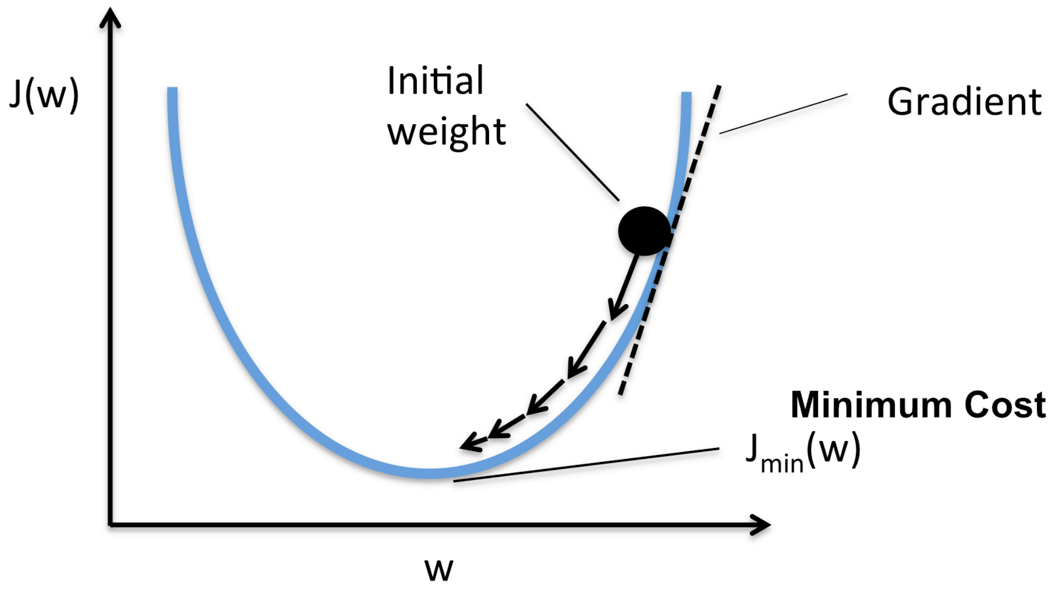


Figure 3-3: Performing Gradient Descent for Weights

Activation Functions:

Each of the outputs of the intermediate layers is usually activated using an activation function such as Rectified Linear Unit (ReLU), hyperbolic tangent (tanh) or a simple linear layer to improve the generalization of the network. ReLU is one of the most widely used activation functions in neural networks [27] and therefore, it is used in this thesis to activate the fully connected layers. ReLU function can be seen in Figure 3-4 and the mathematical formulation is shown in (3.3).

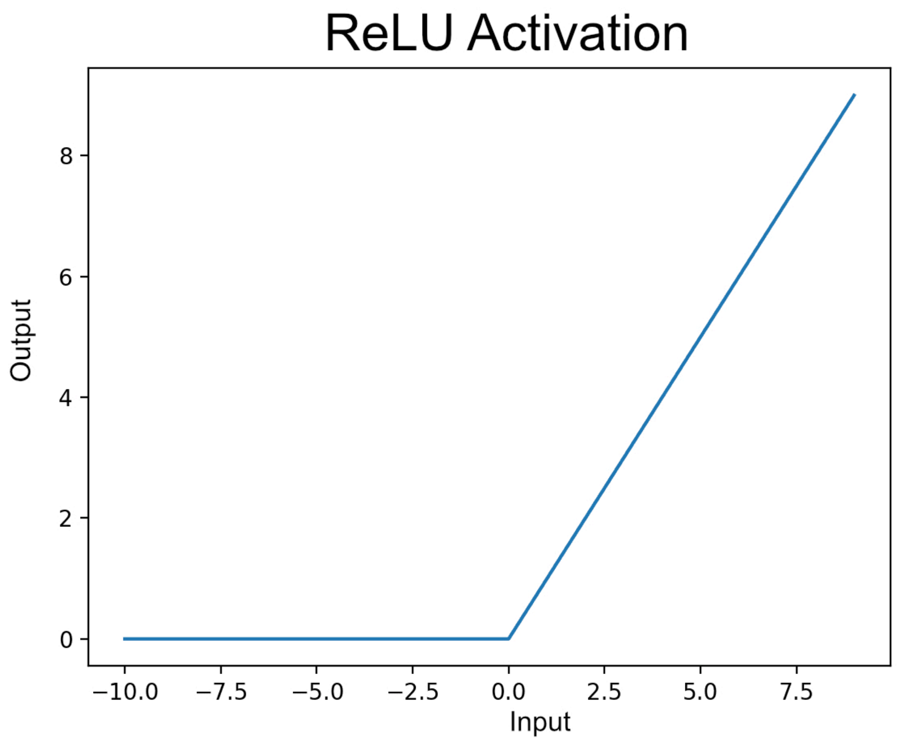


Figure 3-4: ReLU Activation Function

$$g(x) = \begin{cases} x, when\ x\ \geq 0 \\ 0, when\ x\ \leq 0 \end{cases} \tag{3.3}$$

Pooling Layer

The pooling layers operate on each channel independently and divide the input matrix of size $H \times W$ into subregions of smaller sizes and perform operations on them (see Figure 3-5). In the work presented in this thesis, max-pooling is used. The pooling operation divides the input features into subregions, operate on those subregions to summarize those features. In Figure 3-5, a max-pooling layer is passed through the input and we see that the pooling layer divides the input features to 4 regions. It reduces each subregion to its maximum features. We see the 5 is the maximum value in the blue region, 9 in the green region, 8 and 9 in the yellow and red region respectively.

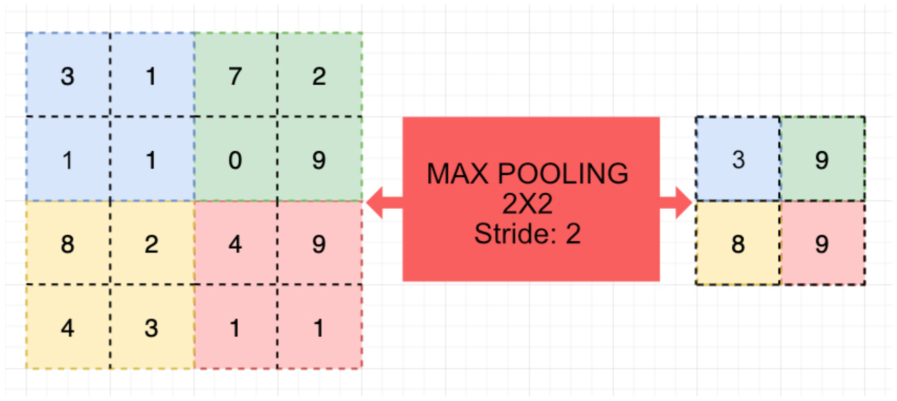


Figure 3-5: Max-pooling Applied to CNN

3.2 Variational Auto Encoders (VAE)

To understand VAE, we need to explain Auto-Encoders. Auto-Encoders are a type of generative models that try to reconstruct the original input by reducing the high dimensionality of the input to a latent vector through a neural network (encoder part of the architecture). Then, a second neural network (namely, decoder) that has the opposite architecture of the encoder network, tries to map that low dimensional vector to a reconstruction of the input, see Figure 3-6. In AEs, the goal is to generate images that are very similar to the input images. That is, once trained, AEs generate similar representation or a reconstruction of the input.

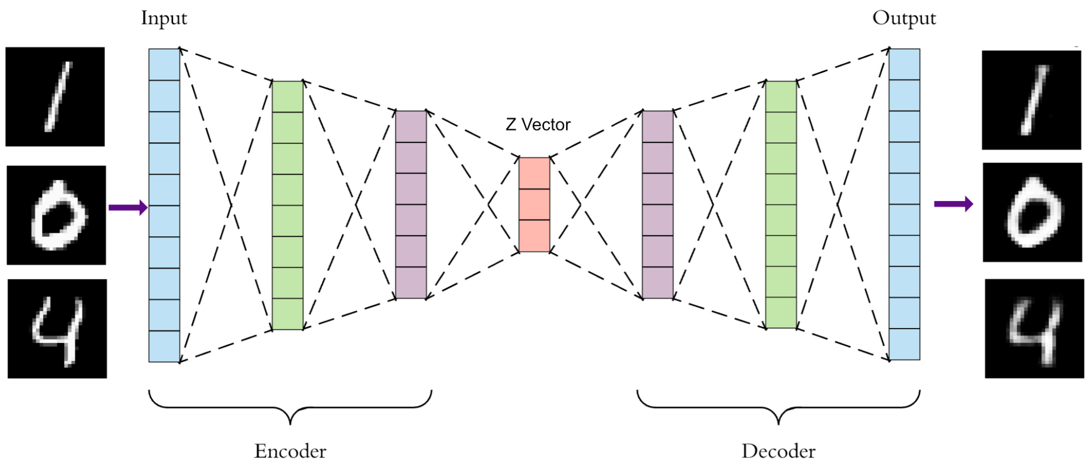


Figure 3-6: Auto-Encoders

VAEs follow the same general structure of AEs, namely, the encoder, decoder, and the latent representation between the encoder-decoder network. The main difference lies in the fact that now we want to reconstruct our input (similar to AEs) while also learning the different distributions of our data. The encoder tries to encode the high dimensional input features into means and variances that describe the different data distributions. The decoder inputs are randomly sampled from those learned distributions (from the encoder part). The decoder generates new data based on the sampled information, see Figure 3-7. The loss function is shown in (3.4). It consists of two parts, the first part, $\log p(x|z)$, is similar to AE's loss which is the reconstruction loss between input and output that we want to minimize. The second term is the Kullback-Leibler (KL) divergence, which makes sure that the distributions learned in z space are not far away from a normal distribution.

$$E_{q(z|x)} \log p(x|z) - KL(q(z|x)||p(z)) \tag{3.4}$$

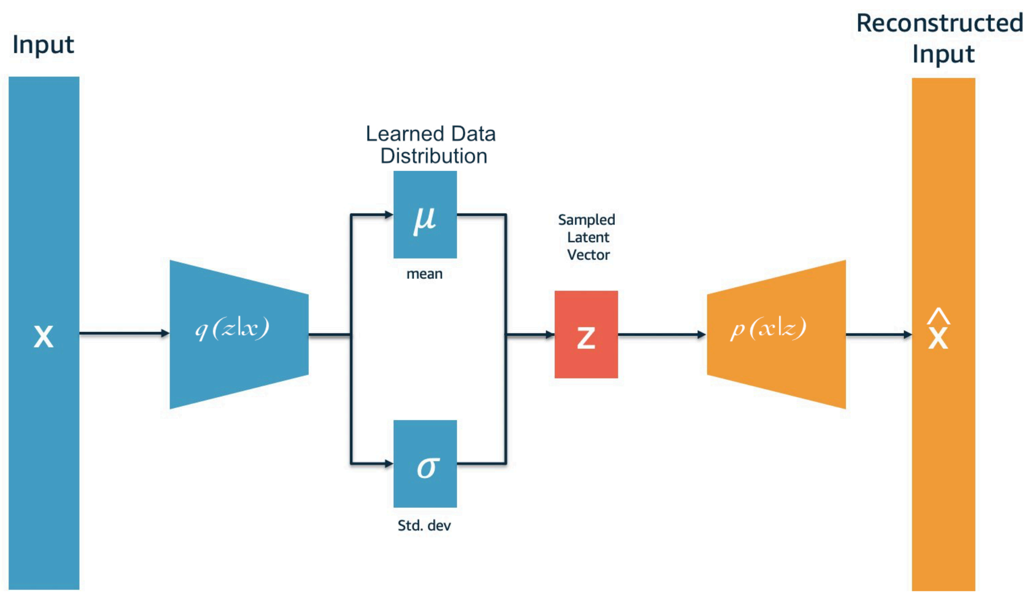


Figure 3-7: Architecture of Variational Auto Encoders

3.3 Long Short-Term Memory networks (LSTM)

LSTMs are used in state-of-the-art natural language processing such as search autocompletion in Google, recommendation system in Amazon, and Netflix[28], [29]. They perform exceptionally when dealing with sequential data due to their ability to retain long-term dependencies. A single LSTM cell (see Figure 3-8) encloses what is referred to as “gates” and takes in an input X_t, information from the previous time step $C_t - 1$, and $h_t - 1$, and creates outputs h_t and C_t for the next cell if the information still needs to be processed.

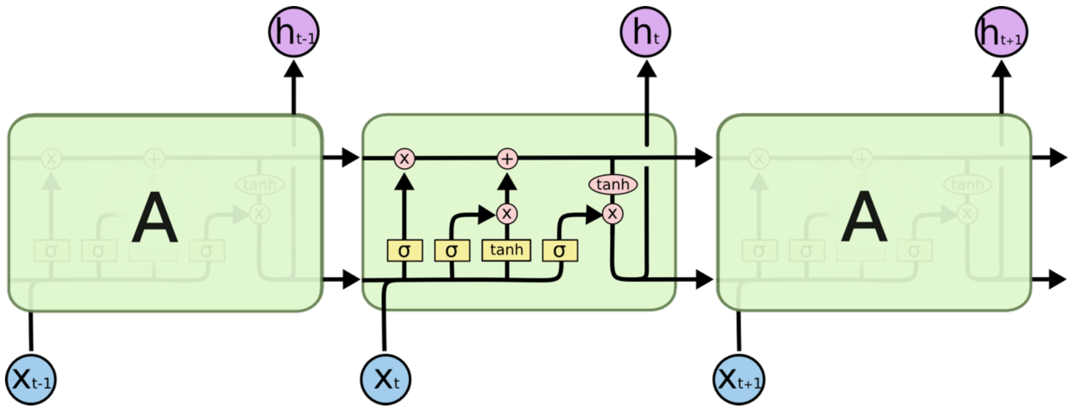
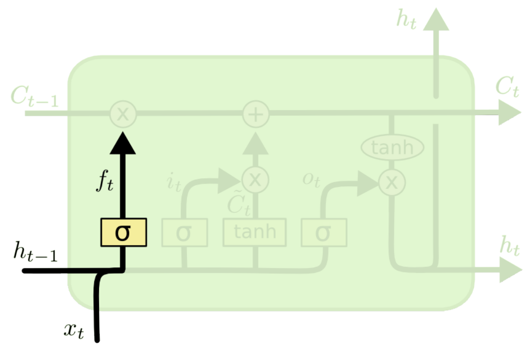


Figure 3-8: LSTM Cells [30]

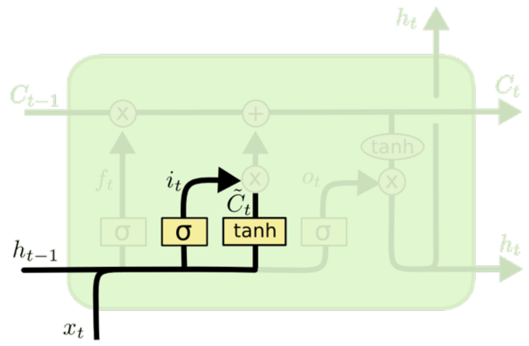
The information going through the LSTM cell is processed through gates and the first gate is the *forget gate*. This gate basically decides what information from the previous step in $C_t - 1$ is to be forgotten based on the new information X_t. This is done through the sigmoid function (highlighted in yellow in Figure 3-9) which squashes the information between 0 and 1 with 0 being forget that information and 1 being completely remember that information.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 3-9: LSTM Forget Gate [30]

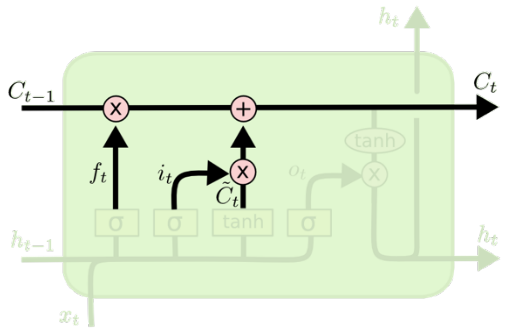
The second gate, namely the *remember gate*, is concerned with what is currently important and must be remembered so it can be added to C_t to build a long-term memory. Both the input X_t and the information from the previous time step $h_t - 1$, are passed through a sigmoid function and a tanh function. The sigmoid function is used similar to the previous step, i.e., to decide which information is relevant and must be remembered. However, the tanh function is used to prioritize the information in X_t by squashing it between -1 and 1, see Figure 3-10.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 3-10: Remember Gate [30]

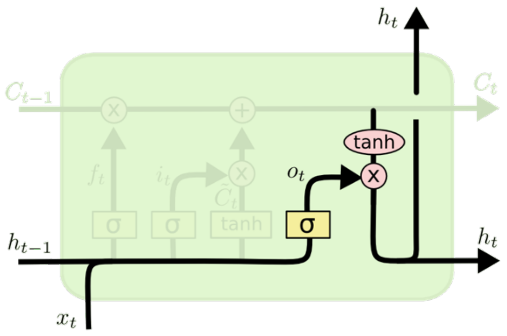
The output is then added to $C_t - 1$ and passed to the next cell C_t accumulating more long-term information, see Figure 3-11.



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

Figure 3-11: Remember Gate (Cell State) [30]

The last gate in the process is the *output gate*. It basically takes the summarized version of the long-term memory, C_t, pushes the values between -1 and 1, reduces the current time input to probabilities using the sigmoid function, and multiplies the two to generate the output h_t, see Figure 3-12.



$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 3-12: Output Gate [30]

3.4 Conditional Generative Adversarial Networks (C-GANs)

GANs are a clever way of setting up two neural networks with opposing objectives to play against each other to see if they are able to learn how the other operates and find equilibrium. GAN algorithms essentially use two blocks, a *generative* block and a *discriminator* block, both of which are represented by a deep neural network. The generator takes in a random vector and tries to create an output that is similar to the input dataset. The discriminator tries to distinguish between the real input data and the synthetic data created by the generator (see Figure 3-13). The GAN algorithm is analogous to a two-player game with opposing and adversarial objectives.

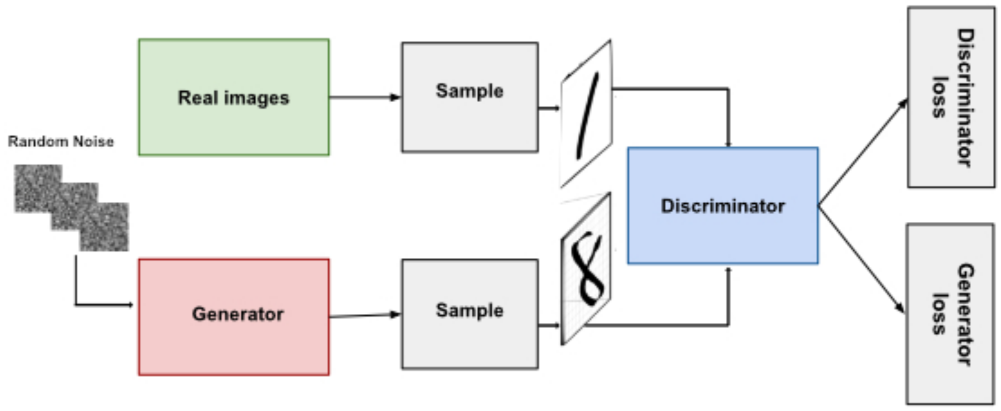


Figure 3-13: GAN General Architecture

$$min_{\theta_g} max_{\theta_d} [\mathrm{E}_{x \sim pdata} log D_{\theta_d}(x) \; + \; \mathrm{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)] \qquad (3.5)$$

In the loss function for a GAN shown above, the discriminator, described by the parameters (θ_d), tries to maximize the loss function ($max_{\theta_d}[\mathrm{E}_{x \sim pdata} log D_{\theta_d}(x)]$). That is, it tries to correctly identify real images from fake images generated by the generator block. The generator, described by the parameters (θ_g), tries to minimize the second term of the equation ($min_{\theta_g}[\mathrm{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)]$. That is, it tries to fool the discriminator into thinking that those synthetic images are real images. Both θ_d and θ_g are some form of neural networks.

3.5 Graph Convolutional Neural Network (GCNN)

GCNNs are unstructured CNNs where the spatial relations are not limited to nearby data. In reality, two adjacent loads can be connected to the same circuit but also, they can be connected to different sources that do not meet at all. That is where CNNs fail and GCNNs succeed.

Spatio-Temporal Graph Convolutional Networks (STGCN)

STGCN [31] is a special type of GCNN where the application is a regression problem and not a classification problem. STGCN was introduced in 2017, where it was applied to forecast traffic. STGCN consists of two main blocks, see Figure 3-14:

1- Spatio-Temporal Convolutional Block: The input time series is passed through a temporal gated convolutional network to extract temporal features and a spatial graph convolutional network where the spatial relations of the network are utilized to help in prediction.

2- Output Layer: The output layer is a linear Bayesian layer to generate the scenarios.

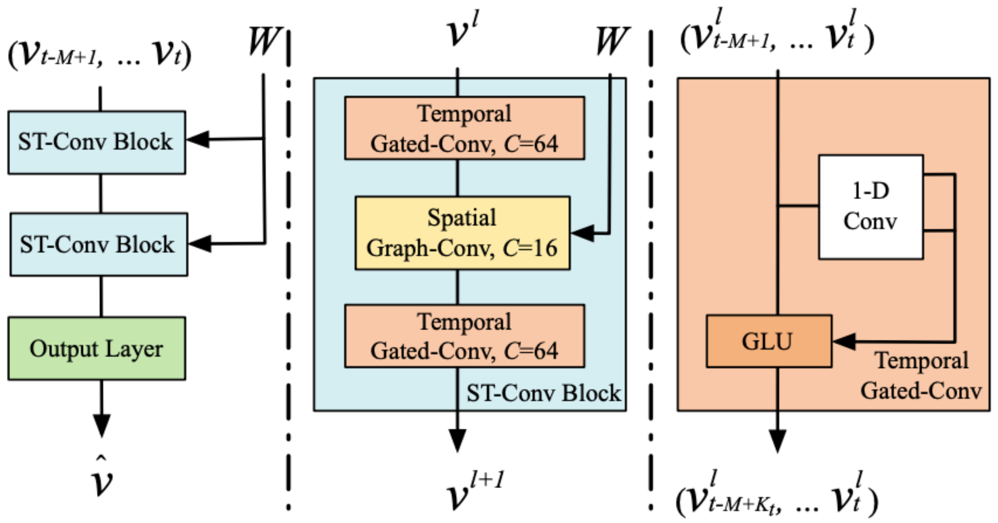

$(v_{t-M+1}, \dots v_t)$
W
ST-Conv Block
ST-Conv Block
Output Layer
$\hat{v}$
v^l
W
Temporal Gated-Conv, C=64
Spatial Graph-Conv, C=16
Temporal Gated-Conv, C=64
ST-Conv Block
v^{l+1}
$(v^l_{t-M+1}, \dots v^l_t)$
1-D Conv
GLU
Temporal Gated-Conv
$(v^l_{t-M+K_t}, \dots v^l_t)$

Figure 3-14: STGCN General Architecture [31]

Temporal Gated Convolutional Network:

The temporal layer simply applies the following equation:

$$\Gamma * \mathcal{T}\, Y = P \odot \sigma(Q) \tag{3.6}$$

Where P is the raw data of the time series, Q is the raw data of the time series passed through a 1-D CNN to summarize and learn temporal relations and σ is the sigmoid function that squashes the results between 0 and 1 and acts as a gate to control the flow of data. The elements of the original time series (P) is then multiplied by the elements of the summarized version of the time series (Q). The result of this temporal layer is an evaluation of the importance of each input in the time series.

Spatial-Graph Convolution:

In the [31], for the traffic forecasting application, the term in the parentheses in (3.7) was used to normalize a given adjacency matrix since 1 and 0 or completely connected and unconnected nodes, did not stabilize the learning process.

$$\Theta *_{\mathcal{G}} (x) = \theta(\widetilde{D}^{-0.5}\widetilde{W}\widetilde{D}^{-0.5})x \tag{3.7}$$

However, since the connection in an electrical network is characterized by the Ybus matrix (admittance matrix) and since the Ybus actually quantifies how strong and how week is the connection between each node, it was logical to replace that normalization with the Ybus so that the new formulation becomes:

$$\Theta *_{\mathcal{G}} (x) = \theta(Ybus)x \tag{3.8}$$

The above, (3.8), simply states that the output of the temporal layer is multiplied by the Ybus and the result is multiplied with theta (a neural network with learnable parameters).

ST-Conv Block:

The final result of each spatio-temporal block is expressed below:

$$v^{l+1} = \Phi_{\theta}(\Gamma_1^l * \mathcal{T}\, ReLU\, (\Theta^l *_{\mathcal{G}} \left(\Gamma_1^l * \mathcal{T} v^l\right))) \tag{3.9}$$

Starting with the rightmost parentheses, we see that the time series input (v^l) is passed through the temporal gated convolutional layer. The result is then passed through the spatial-graph convolution. The result is then activated by a ReLU function and passed through another temporal gated convolutional layer. Finally, the result is passed through a Bayesian network to generate the desired scenarios.

3.6 Error Metrics

RMSE: Root Mean Squared Error (RMSE) is a standard measure of error used to quantify the prediction error of a model and it is defined as:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{3.10}$$

The problem is, this metric in itself cannot determine the quality of scenario generation models. This is because scenarios are supposed to be different and sometimes extreme. Therefore, this metric is employed to check if the model struggles across the different variables (solar, wind, and load), but is not used to compare the accuracy of the different algorithms identified in this thesis.

Mean Absolute Error: Mean Absolute Error is another error metric between paired time series, and it is employed in the same way as the RMSE.

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{3.11}$$

3.7 Autocorrelation Coefficient Function (ACF)

Time observations can be correlated with other forms of lagged versions of the same time series. ACF quantifies how those data points relate to the preceding time steps. It essentially measures the self-similarity of the times series between different time-lags.

$$p_k = \frac{\sum_{t=k+1}^{T}(r_t - \bar{r})(r_{t-k} - \bar{r})}{\sum_{t=1}^{T}(r_t - \bar{r})^2} \tag{3.12}$$

For example, time series with adjacent measurements have similar values and distant measurements have values with higher deviations. That is, the ACF tends to decrease over larger time-lags. A positive value in the ACF means that the data points for this particular time lag stay consistently above or below the mean of the time series, while a negative value means that the data points alternate across the mean. In this thesis, ACF is used to see if the generated scenarios and the original input signals behave similarly over different time lags.

3.8 Cumulative Distribution Function (CDF)

CDF is defined mathematically as:

$$F_X(x) = P(X \leq x) \tag{3.13}$$

It is the probability that the variable X takes a value less than x. CDF is used to compare the real and generated scenarios.

CHAPTER 4

PROPOSED ARCHITICURES

In this chapter we are going to prepare our dataset, define some preprocessing techniques and apply them our data, prepare the test system, and design the architecture of the different ML models. For each of the models, we are going to show different seasonal results for wind and solar farms, weekday and weekend results for load and discuss the results. Lastly, we will compare the different algorithms and their performances, discuss the weak points and superiority of each model and their application.

4.1 Data Preprocessing

Machine learning models depend heavily on the quality and the quantity of the dataset utilized. In order to build our training dataset, we need realistic data for solar farms, wind farms, and loads. For solar and wind farms, the NREL integration dataset [32] was used. It provides one-year worth of hourly data for different solar and wind farms in the US. Due to the lack of load datasets, the information of the synthetic 2000-bus Texas system [33] was utilized. It provides an hourly load reading for one year. Hence, we have a total of 8,760 data points for each variable.

4.2 The Test Model

The IEEE-30 bus system was chosen to test the different machine learning models identified in Chapter 3. The two conventional generators located at buses 1 and 2 of this system were replaced by solar farms and wind farms, respectively. The loads of each of the buses were compared to the average of each of the load buses in the Texas 2000 bus test case and the mapping of the buses was done accordingly.

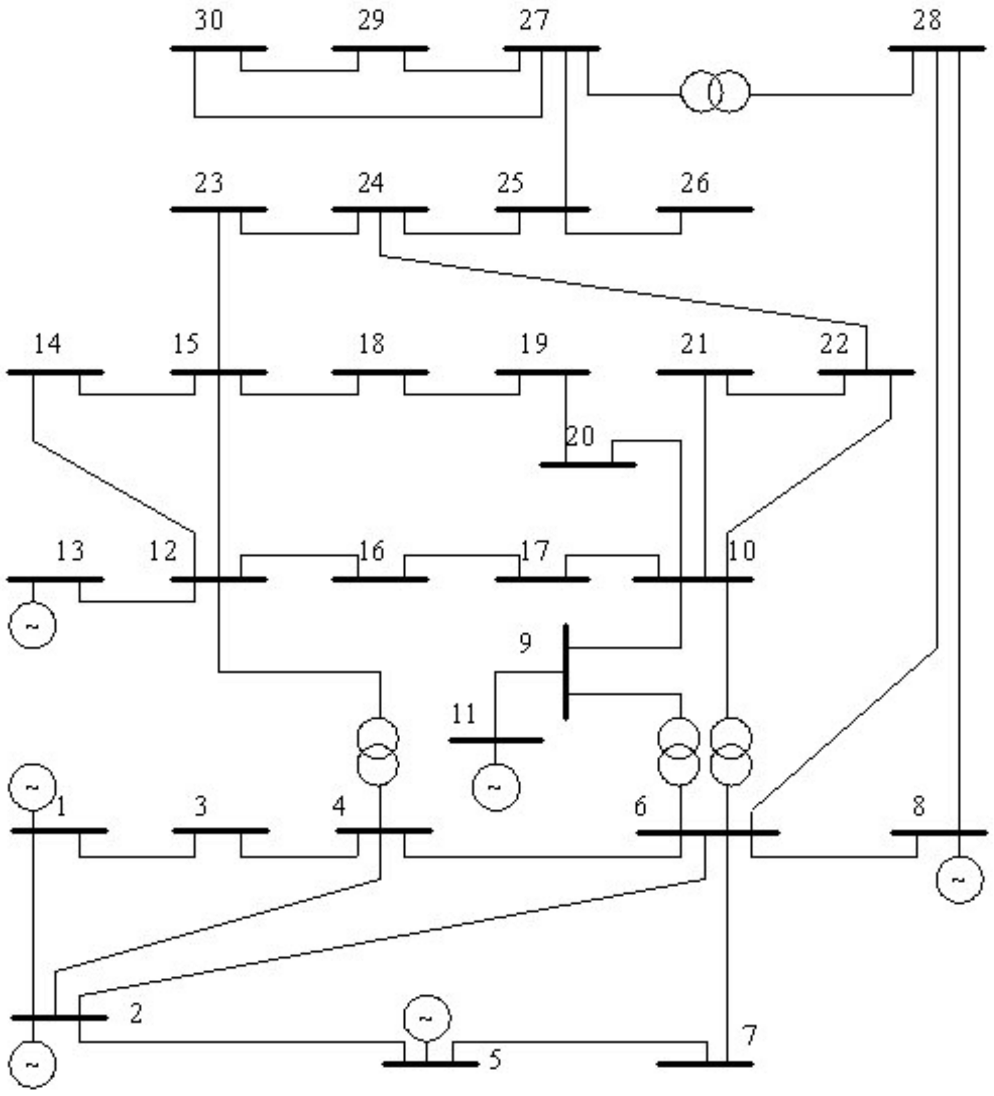


Figure 4-1: IEEE-30 Bus System

4.3 Data Normalization

Data normalization is a very important and an integral step of any regression problem. It makes models process data faster [34] and learn quicker than when raw data are used. It also avoids explosion of learnable parameters in neural networks during training. Therefore, we need to normalize our data before using them in training or prediction. The equation used to normalize the data is shown below:

$$z = \frac{x - \mu}{\sigma} \tag{4.1}$$

Where z is the normalized value, x is the raw data, μ is the mean of the data and σ is the standard deviation of the data. This normalization is done variable wise in a multivariate problem. In this setting, for each variable, we calculate its mean and standard deviation and apply the normalization for that variable.

4.4 Label Encoding

Label encoding is a process to convert categorical classes into numerical forms as machine learning algorithms are unable to work with labeled data. One form of label encoding is integer encoding where each category is converted into an integer [35]. For example, if we have 3 different labels for the different classes, each label can be given an integer value between 1 and 3. However, this might create an ordering problem since three is bigger than one. The algorithm might translate that as class "3" has higher probability of happening compared to class "1". The second method is to use One-Hot Encoding [35]. One-Hot Encoding is the process of labeling the different classes with 0 and 1. So for the same 3 class example, one hot encoding would generate a vector of size 3 and the correct class will be 1 and all the other classes will be given 0.

4.5 CNN Model

CNNs have performed exceptionally when it comes to image classification problems due to the way the filters process spatial patterns in pictures. They also have great merits when it comes to forecasting because the same filters can be used to process limited temporal and spatial relations. However, using a combination of CNN and Multi-Layer Perceptron (MLP) can only result in the same output over and over given a fixed input. This is due to the fact that all the learnable parameters (weights and biases of the neurons and filters), are set after the model is trained. However, that is not the purpose of scenario generation as we want to generate different scenarios given the same input. The way to overcome this problem is to use a distribution over the weights for the last layer. That will allow for the CNN to capture and learn temporal and spatial relation for different types of variables and then generate scenarios at the last layer by sampling from the weights of the last layer instead of using set parameters that generate the same output repeatedly given an input.

4.5.1 Data

To test the algorithm, we are using the IEEE-30 bus test case illustrated previously. The input data to the algorithm has the following structure:

- Original input matrix has 2 dimensions where the rows are the time steps and the columns are the number of variables (e.g., 30 variables in the IEEE-30 bus system), in the network.
- The input data preprocessor function will normalize the data, generate input and output data sequences, and train, validate and test data sequences. The input and output sequence matrices have 3 and 2 dimensions, respectively, where the input matrix has the final shape of (number_of_samples, input_time_window, number of variables), see Figure 4-2, and the output matrix has the shape of (number_of_samples, output_time_window*number of variables).

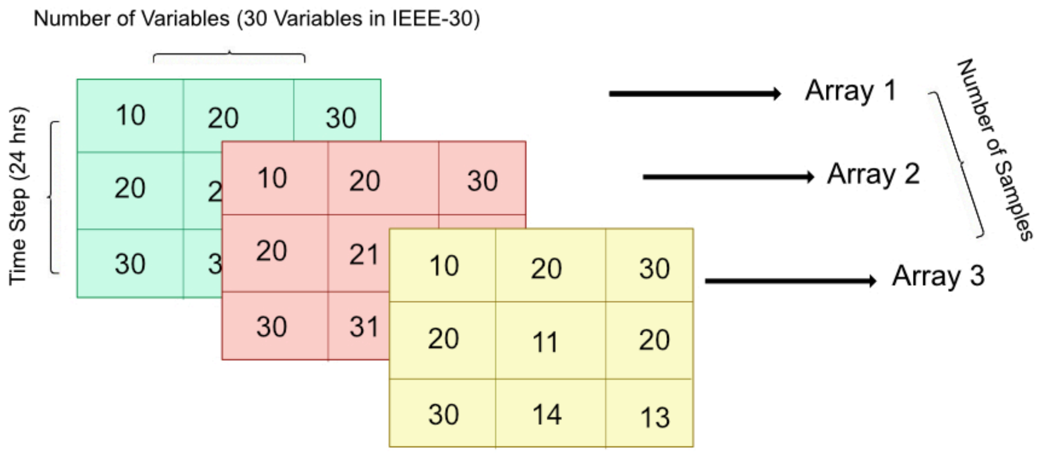


Figure 4-2: Data Input to CNN Model

4.5.2 Hyperparameters

Table 4-1: CNN Hyperparameters

Input Horizon	48
Output Horizon	24
Stride	24
Number of Epochs	1000
CNN	
Filters	64
Filter Size	1x2
Maxpooling	2x2
Linear Layer	
1st Layer Size	800 neurons
Activation Function	Relu
2nd Layer Size	30 neurons
Activation Function	Linear
Learning Rate	0.001
Optimizer	Adam
Loss	Mean Squared Error
Data	
Train	80%
Validation	10%
Testing	10%

The above-mentioned hyperparameters were the best parameters found during the training of the neural network.

4.5.3 Results

After training the CNN model to generate scenarios, we sample from those scenarios and analyze if the model is able to generate representative scenarios. First, we compare sample summer and winter scenarios for the solar farms. In order to generate the scenarios, we need to repeatedly input distinct summer and winter power profiles for the solar farms. The same scenarios used here will be used for the other algorithms for benchmarking and comparison. Figure 4-3-Figure 4-8 show the results.

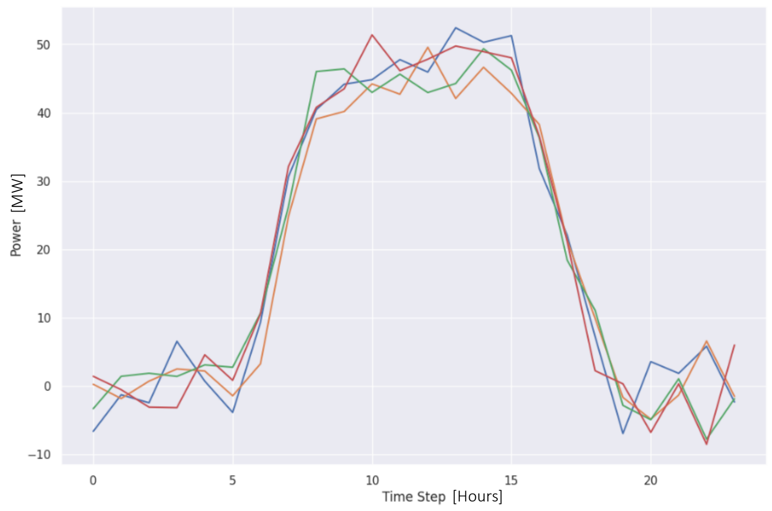


Figure 4-3: Solar Generated Scenarios During Summer

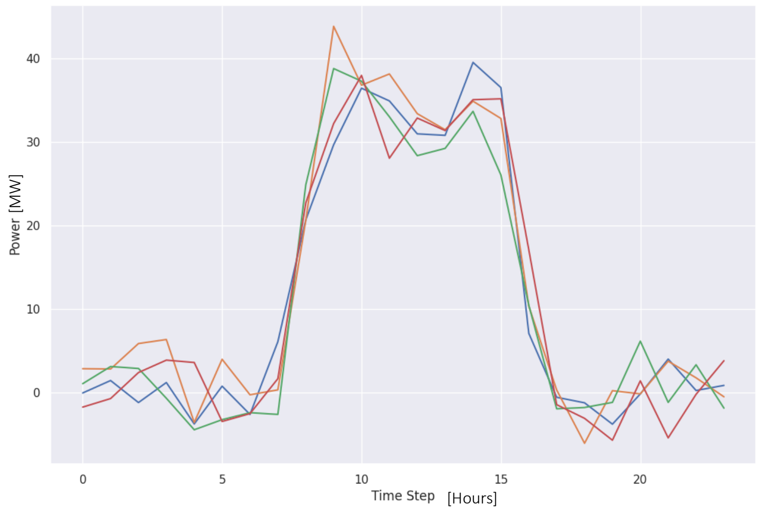


Figure 4-4: Solar Generated Scenarios During Winter

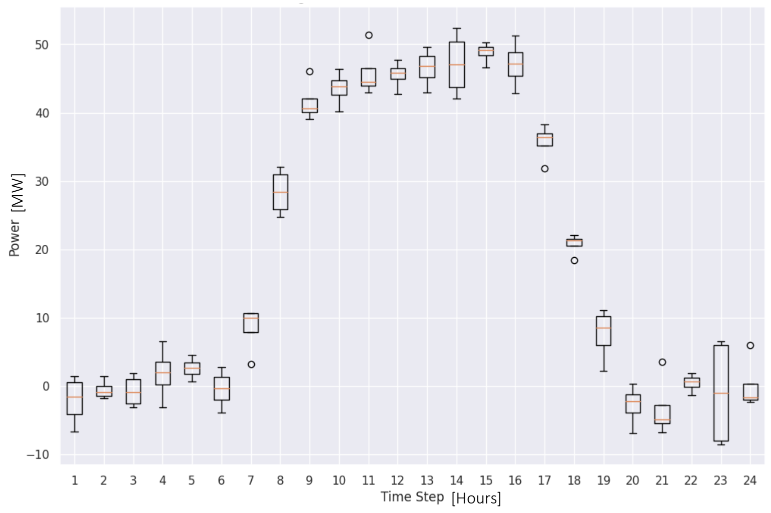


Figure 4-5: Average Hourly Power for Solar Farms in Summer Scenarios

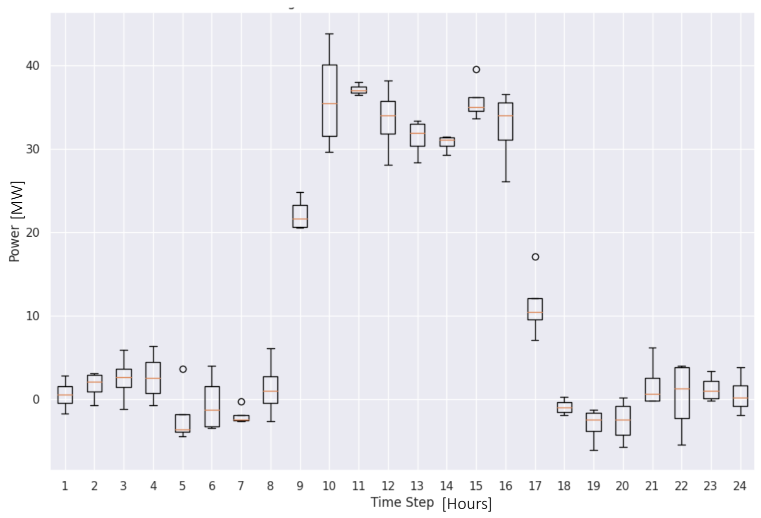


Figure 4-6: Average Hourly Power for Solar Farms in Winter Scenarios

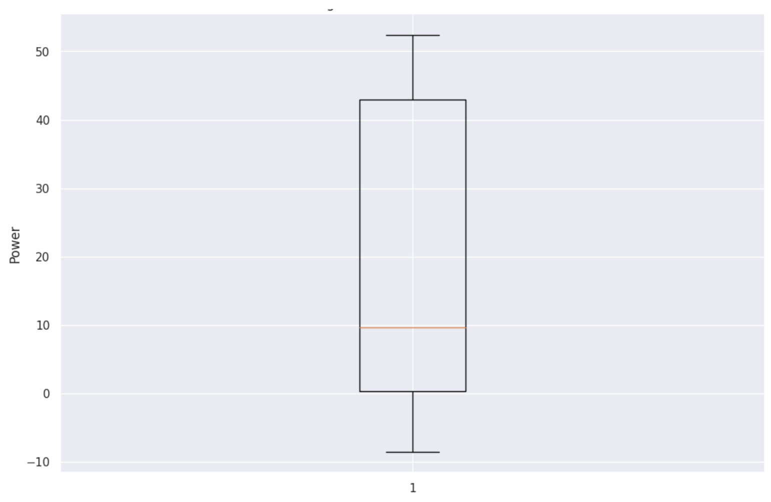


Figure 4-7: Average Daily Power for Summer Scenario

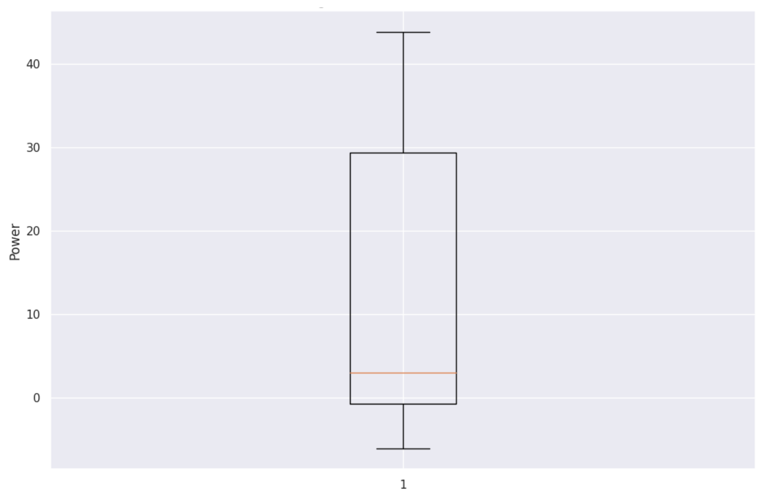


Figure 4-8: Average Daily Power for Winter Scenario

Looking at those scenarios, we notice that CNN is able to generate reasonable scenarios that are able to capture the intricate seasonal behavior of solar farms in summer and winter. We notice that in summer scenarios (see Figure 4-3), the solar farms generate more for longer hours. It starts producing as early as 5 A.M, reaches its capacity at 8 A.M and generally stays at that level for most of the day, before dying down as late as 6 or 7 P.M. During the winter scenario (see Figure 4-4), we notice that the level of production has decreased by 30% during the day and that the duration has also shrunk significantly. From the average hourly production (see Figure 4-5 and Figure 4-6), we notice a higher variability during the day. This could be due to the presence of more cloudy days during the winter season (see Figure 4-6). Looking at the average production for the generated scenarios (see Figure 4-7 and Figure 4-8), it is clear that summer scenarios have a higher average production compared to winter scenarios. This is expected and conforms to a solar farm's production behavior.

Feeding-in summer and spring wind scenarios generates the following results (see Figure 4-9-Figure 4-14).

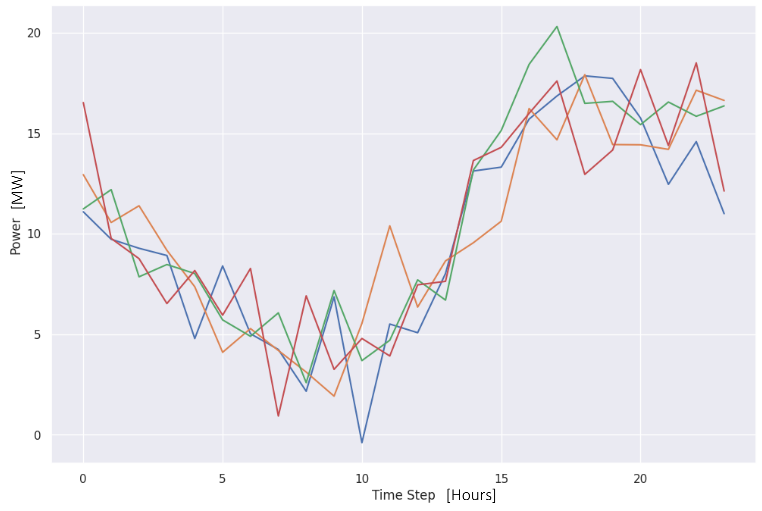


Figure 4-9: Wind Generated Scenarios During Summer

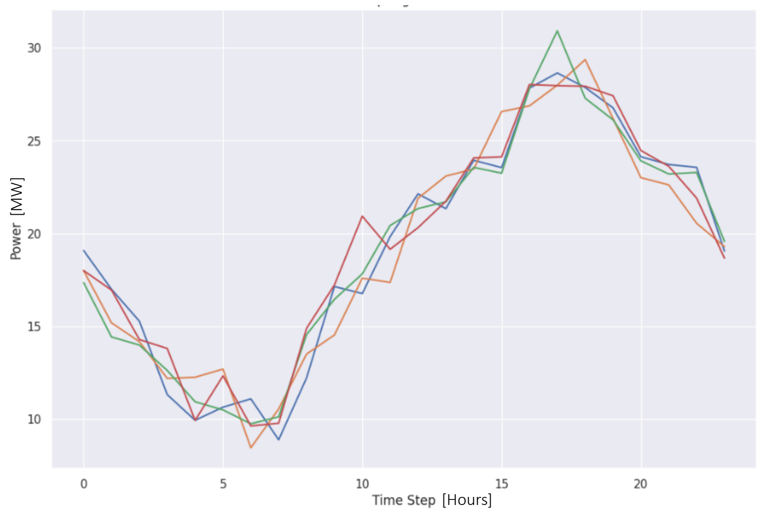


Figure 4-10: Wind Generated Scenarios During Spring

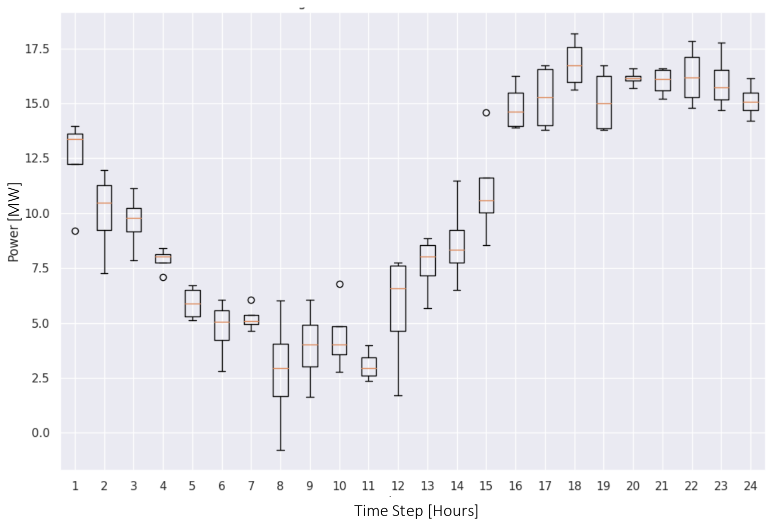


Figure 4-11: Average Hourly Power for Wind Farms in Summer Scenarios

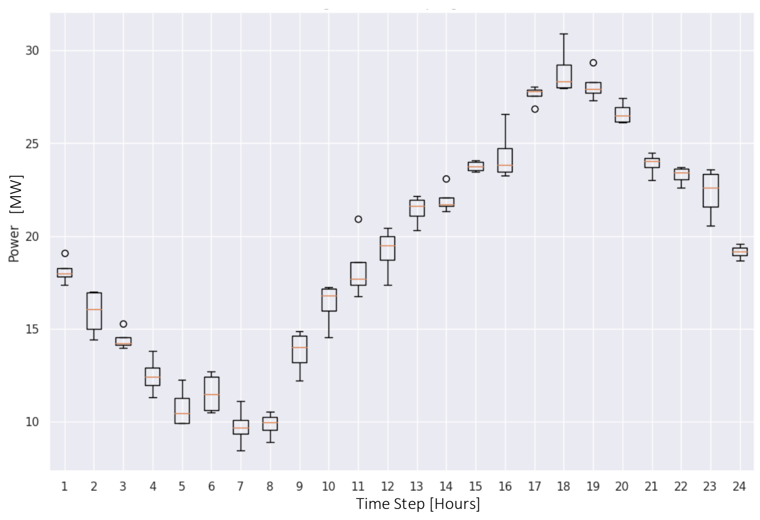


Figure 4-12: Average Hourly Power for Wind Farms in Spring Scenarios

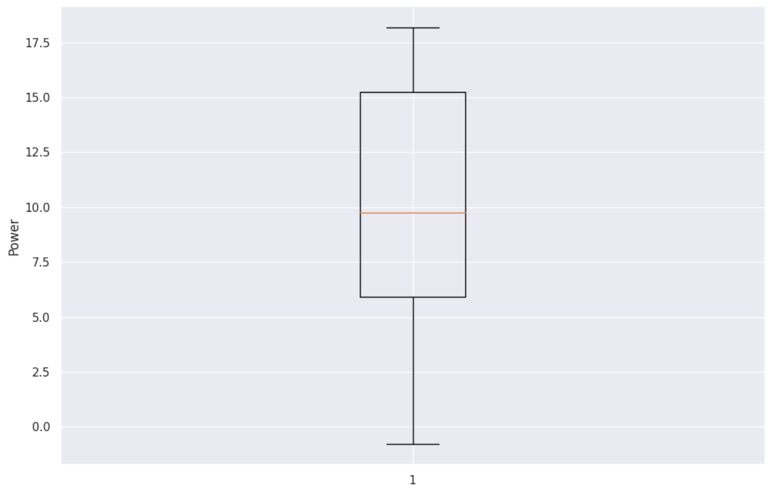


Figure 4-13: Average Daily Power for Summer Scenario

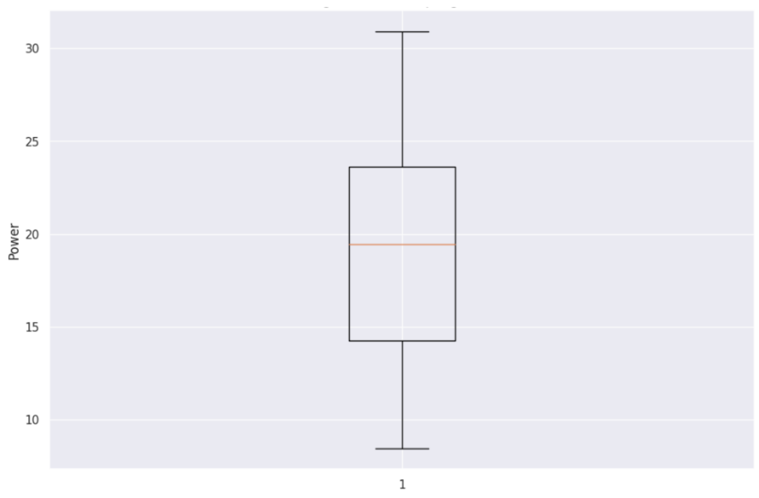


Figure 4-14: Average Daily Power for Spring Scenario

For a wind farm producing during the spring season (see Figure 4-10), the algorithm is able to capture the pattern of high-power production. However, the variability that is inherent to wind farms is not particularly captured as can be seen by the hourly average power production (see Figure 4-12). This is not the case for a wind farm producing in the summer season (see Figure 4-9), where the algorithm is able to capture the lower production level during the summer season as well as the variability of wind (see Figure 4-11). Examining the average power production for summer (see Figure 4-13) and spring (see Figure 4-14) scenarios, we notice a lower average production during summer and a higher production level during spring, which is what we expect during those two seasons. Examining load scenarios during weekdays and weekdays, and feeding-in corresponding inputs, we get the following outputs (see Figure 4-15-Figure 4-20).

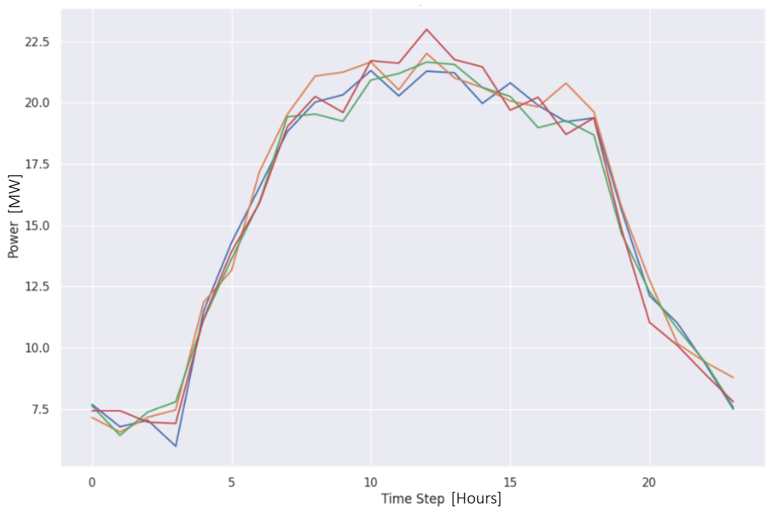


Figure 4-15: Load Generated Scenarios During Weekday

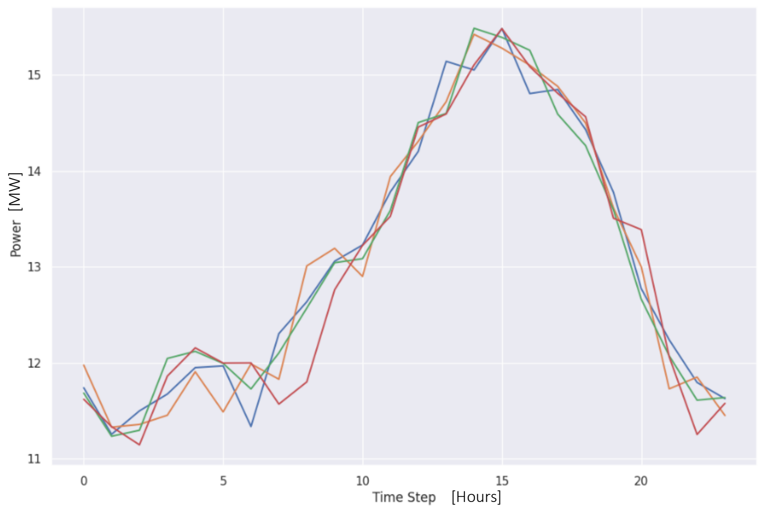


Figure 4-16: Load Generated Scenarios During Weekend

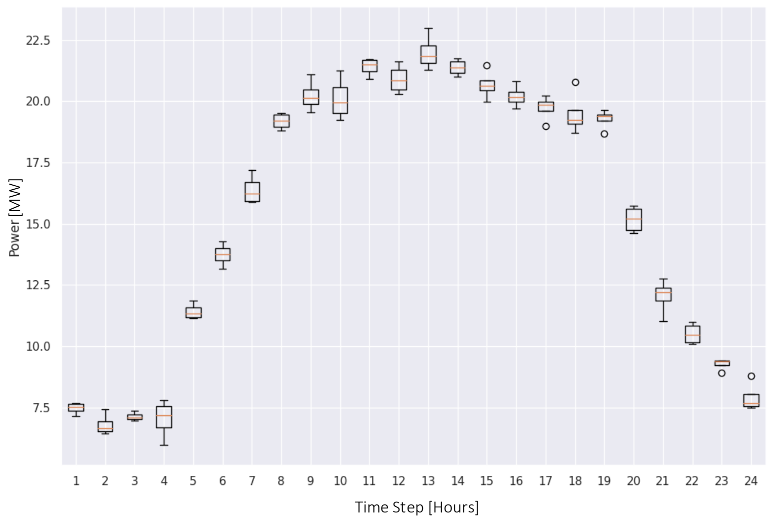


Figure 4-17: Average Hourly Demand for Load Weekday Scenarios

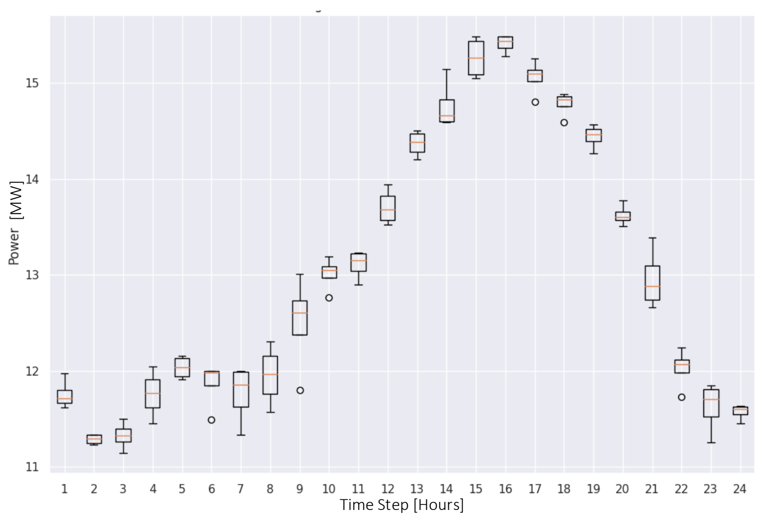


Figure 4-18: Average Hourly Demand for Load Weekend Scenarios

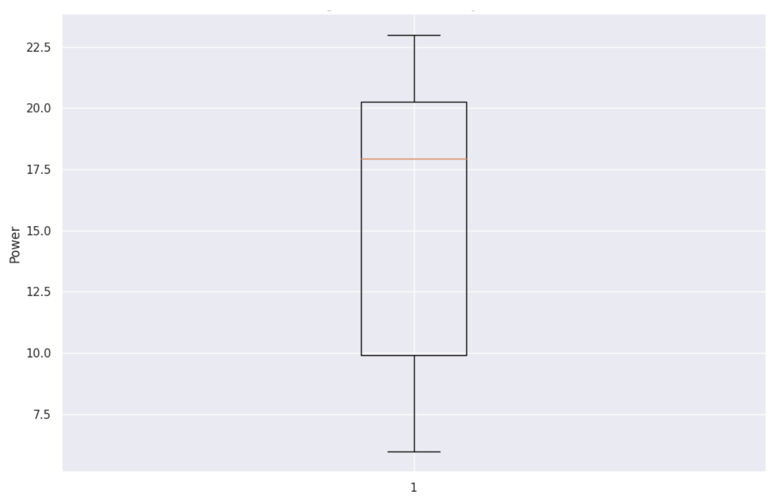


Figure 4-19: Average Daily Demand for Weekday Scenario

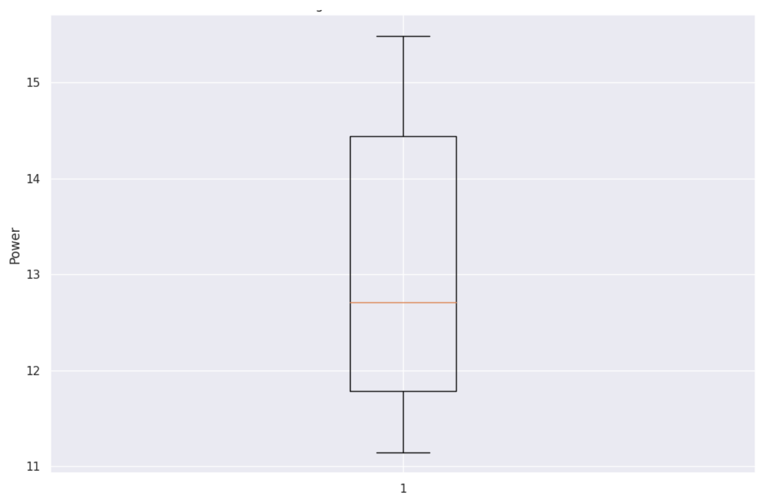


Figure 4-20: Average Daily Demand for Weekend Scenario

Investigating the behavior of the load during weekdays and weekends, we find that the algorithm is able to generate different scenarios that are in line with our expectations. Given a weekday load profile, the model is able to capture the pattern of the load where there is a heavy consumption during the morning and afternoon hours, and it drops during the nighttime (see Figure 4-15). The behavior changes a little bit during the weekends, where the load rise starts later in the day and starts decreasing later in the afternoon (see Figure 4-16). The scenarios generated are still not able to capture the variability of the load and that is evident by the hourly average consumptions (see Figure 4-17 and Figure 4-18). The scenarios for the weekday load consumption is generally higher when compared to weekends and that also is aligned with our expectations (see Figure 4-19 and Figure 4-20).

4.5.4 Autocorrelation and Error Metrics

Sampling randomly generated scenarios and comparing them temporally to real scenarios using the ACF gives us insights into the quality of the generated scenarios, particularly if they capture the temporal behavior of each variable.

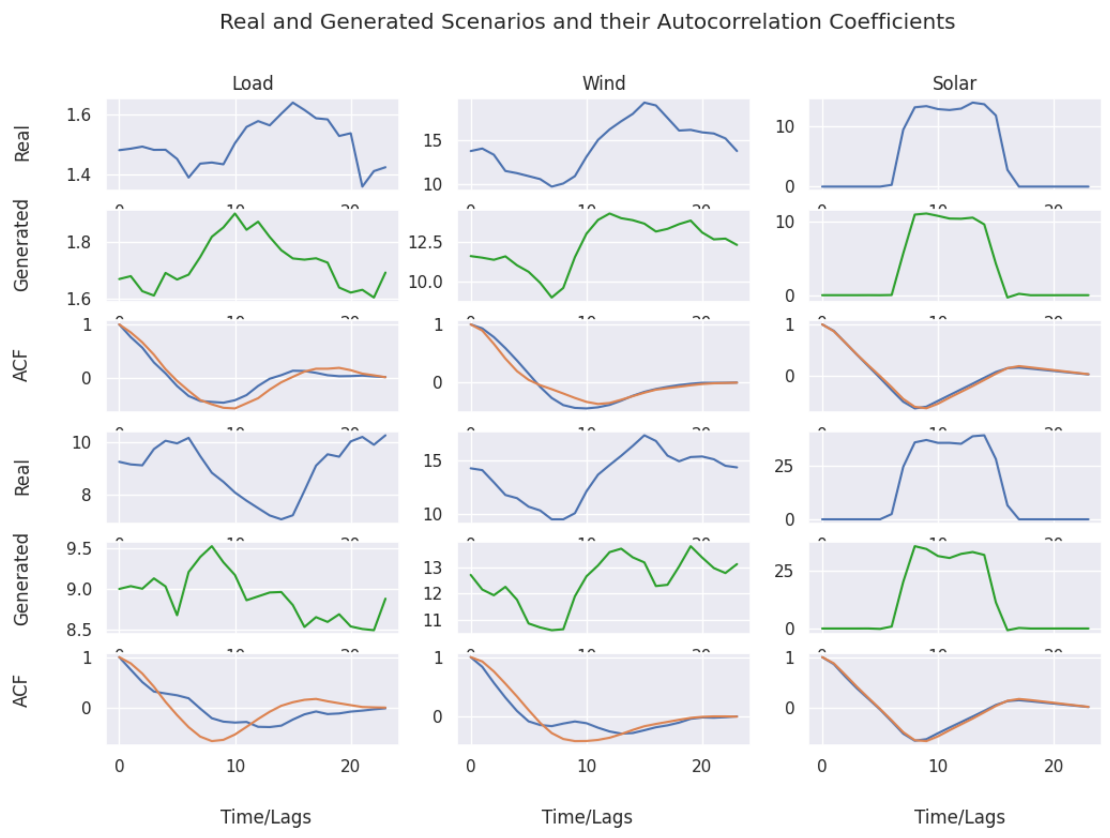


Figure 4-21: Autocorrelation of Real and Generated Scenarios

Looking at the ACF plots in Figure 4-21, we see that the algorithm generally tries to generate scenarios that mimic the temporal behavior of the real scenarios, especially the solar scenarios, but struggles with some of the load and wind scenarios. This can be seen from the ACF (bottom left corner on Figure 4-21) and the ACF next to it.

To further validate the quality of the generated scenarios compared to the real scenarios, we look into the CDFs (see Figure 4-22-Figure 4-24) of the generated scenarios compared to the real ones.

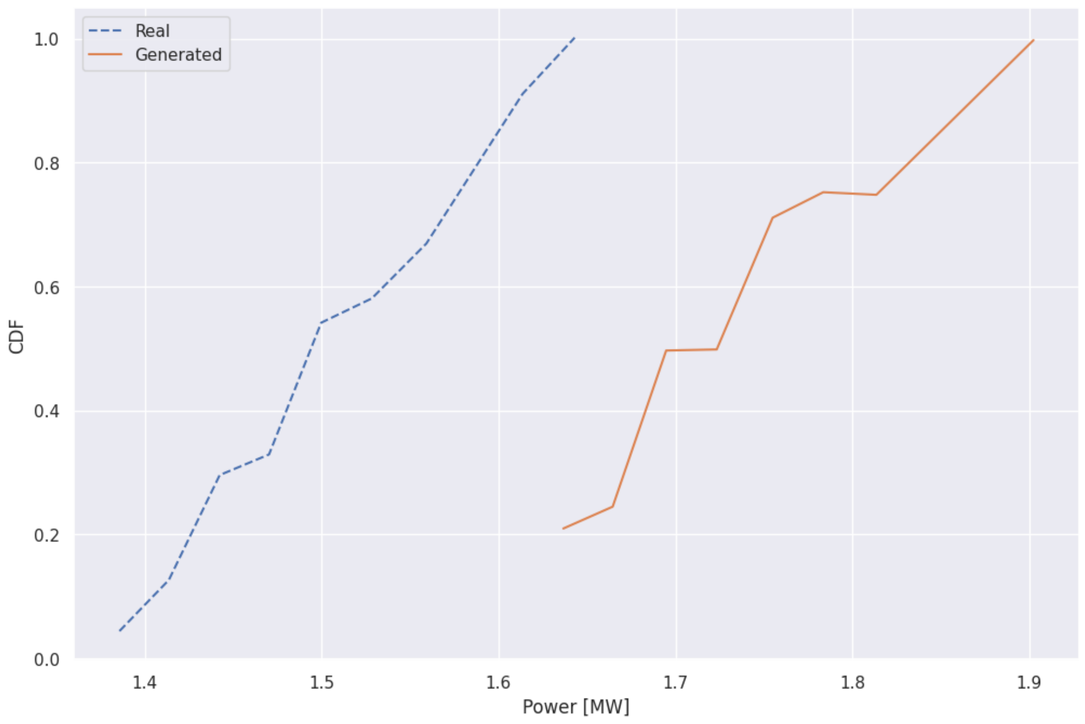


Figure 4-22: CDF of Real and Generated Load Scenarios

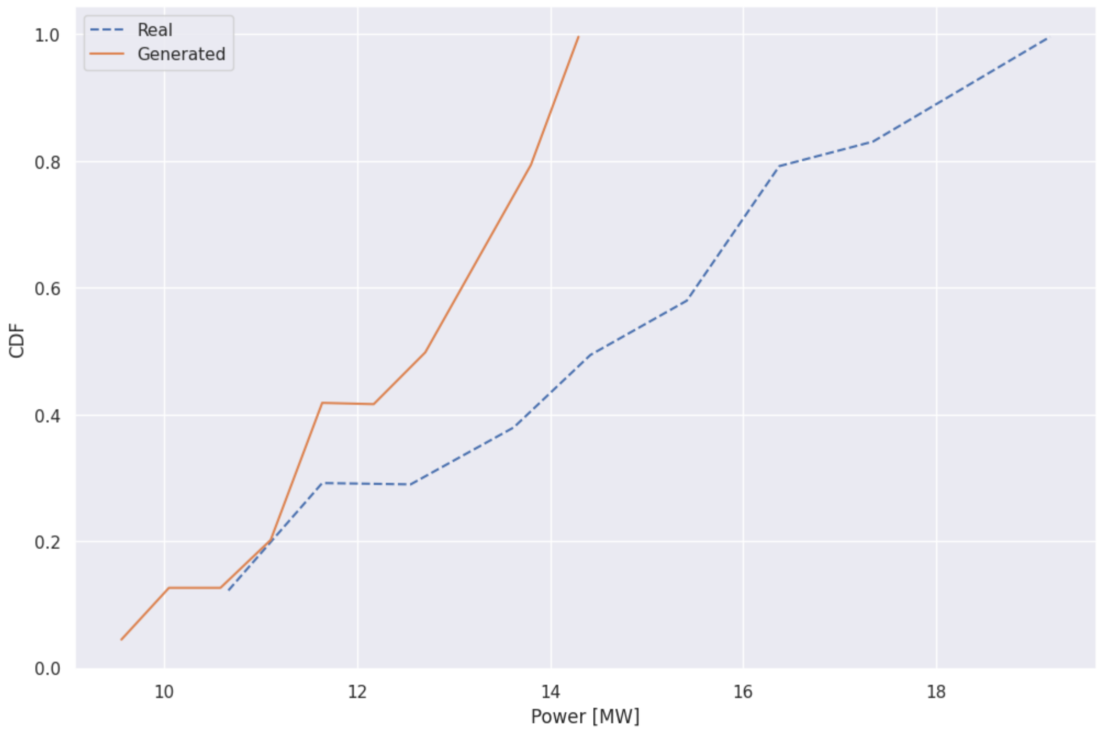


Figure 4-23: CDF of Real and Generated Wind Scenarios

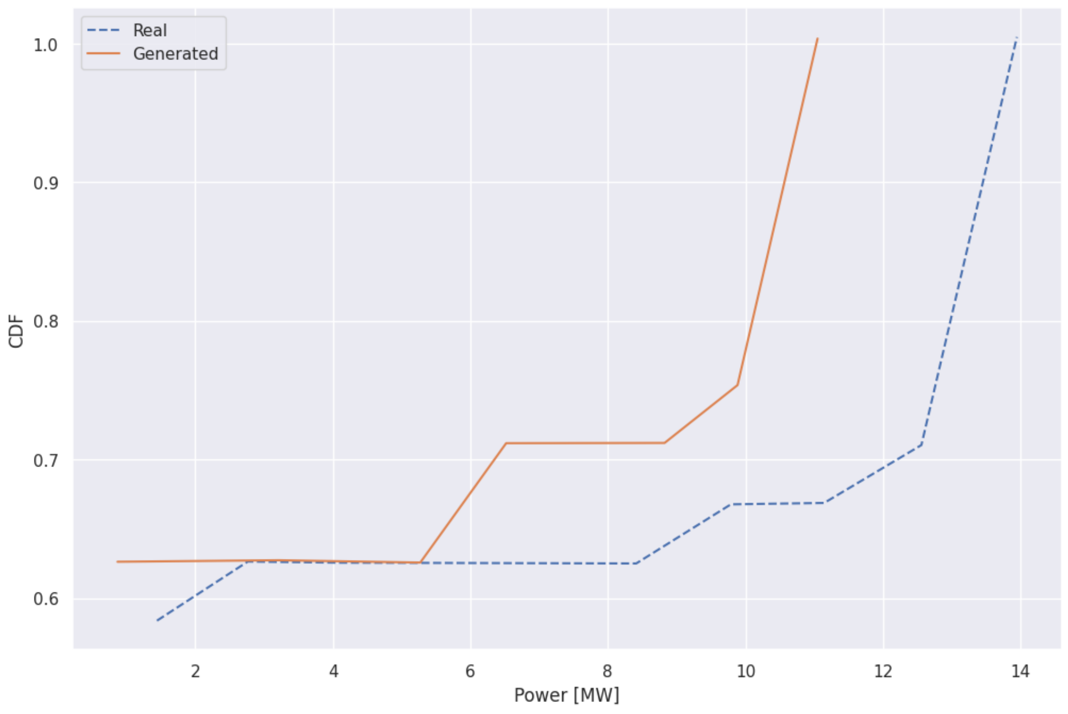


Figure 4-24: CDF of Real and Generated Solar Scenarios

The CDF profiles for the different variables further reinforces the belief that has been established by the ACF, which is that the algorithm struggles more to create wind and load scenarios due to the higher variability of wind and load present in the data.

4.5.5 Error Metrics:

The standard errors shown below are used to check which variable the model struggles with the most. As it was evident by the previous results, the model struggles the most with wind and load scenarios compared to solar scenarios, and this is reflected in the error metric as well.

Table 4-2: MAE and RMSE Error for CNN

	Load	Solar	Wind
MAE	1.994	0.322	2.938
RMSE	12.960	0.135	17.094

4.6 LSTM-VAE

LSTMs have gained their popularity from their ability to process sequential data effectively even with arbitrary hyperparameters. They are able to subsume sequential data due to their structure that was explained previously in Chapter 3.3. However, LSTMs, like a neural network's learnable parameters, are fixed. Hence, for the same input only one output is generated and that is not the purpose of scenario generation. So, to overcome this issue, VAE structure is employed. LSTM is used in the encoder part to summarize the input sequential data of the different variables. The VAE then tries to learn the different means and variances of those variables. The VAE samples from the different categorical embeddings and an LSTM that has the reverse architecture to the one used in the encoder, is used to decode and generate new data in the decoder. An architecture of the model is shown in Figure 4-25.

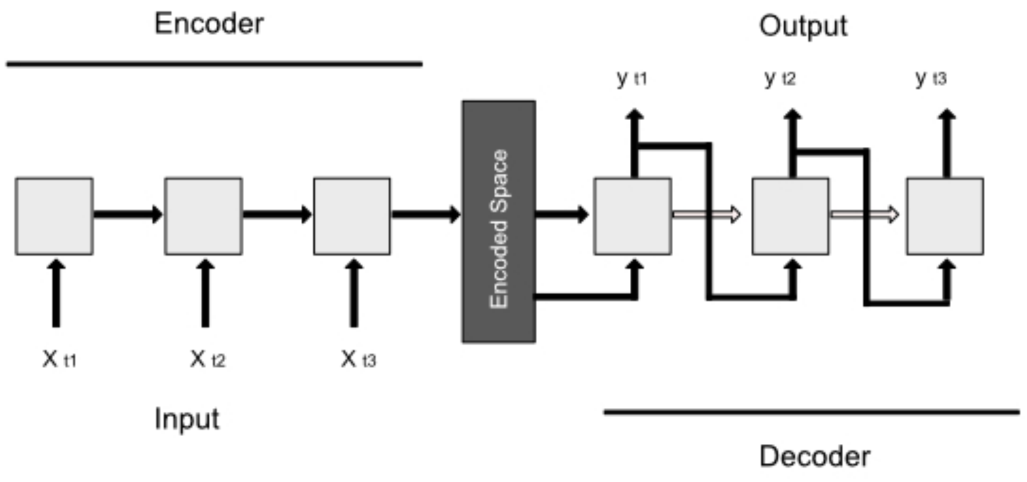


Figure 4-25: LSTM Model Architecture

4.6.1 Data:

To test the algorithm, we use the IEEE-30 bus test case illustrated previously. The input data to the algorithm has the following structure:

- Original input matrix has 2 dimensions where the rows are the time steps and the columns are the number of variables (e.g., 30 variables in the IEEE-30 bus system), in the network.
- The input data preprocessor function will normalize the data, generate input and output data sequences, and train, validate and test the data sequence. The input and output sequence matrices have 3 and 2 dimensions, respectively, where the input matrix has the final shape of (input_time_window, number_of_samples, number of variables) and the output matrix has the shape of (output_time_window, number of variables).

4.6.2 Hyperparameters:

Table 4-3: LSTM-VAE Hyperparameters

Input Horizon	120
Output Horizon	24
Number of Recurrent Layers	1
Batch size	35
Hidden Size	44
Number of Epochs	200
Teaching vs Recursive	Mixed (30% Teacher Forcing)
Learning Rate	0.005963
Optimizer	Adam
Loss	Mean Squared Error
Data	
Train	80%
Validation	10%
Testing	10%

A random search was done over the different hyperparameters to finetune the results of the network with the objective of minimizing the loss function (see Figure 4-26). The objective of this random search is to find the best combination of hyperparameters to minimize the loss. The highlighted line in Figure 4-26 shows the best combination of hyperparameter.

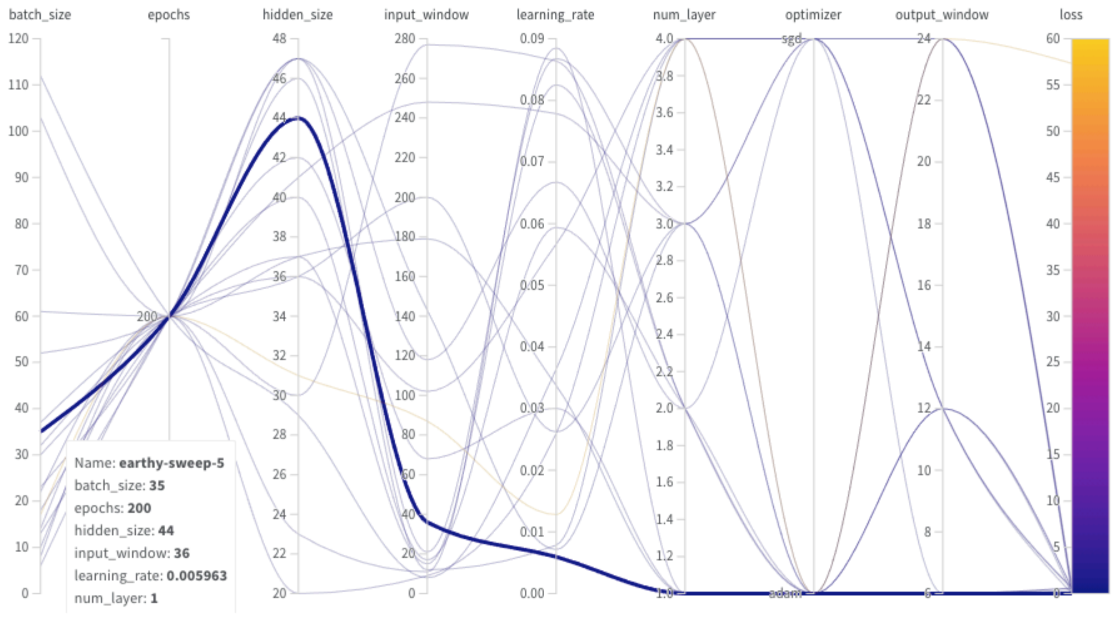


Figure 4-26: Hyperparameter Search for LSTM-VAE

4.6.3 Results:

Now that the LSTM-VAE model is trained, we are able to sample different scenarios for given inputs to evaluate the quality of those scenarios with what we know from the characteristics of solar and wind farms, and loads. We also evaluate them with correlation and error metrics. First, we generate scenarios based on random inputs from our dataset. In this case we do not have control over picking seasonal data. The aim is to generate random scenarios and the results obtained for each given input are as follows.

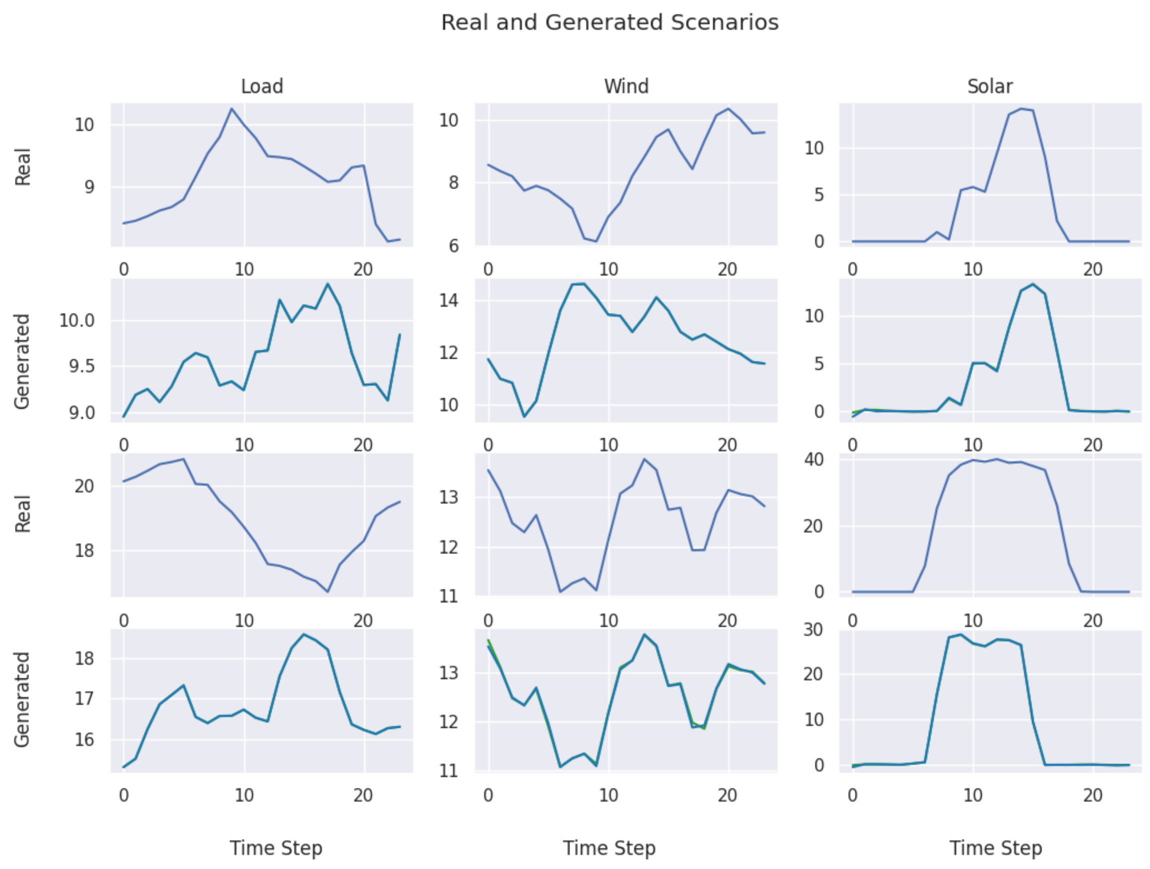


Figure 4-27: Real and Generated Sample Scenarios

A distinct feature of the LSTM-VAE algorithm as seen in Figure 4-27 is that although it is not able to generate scenarios as we had hoped, it does succeed in solving the forecasting problem. This is because there is little to no variability when sampling one or two scenarios from a given input. However, we will consider generating more scenarios in subsequent analysis (Figure 4-28-Figure 4-43) to see if that is able to create more variability.

We first try the same input for solar farms for both summer and winter seasons and compare the results.

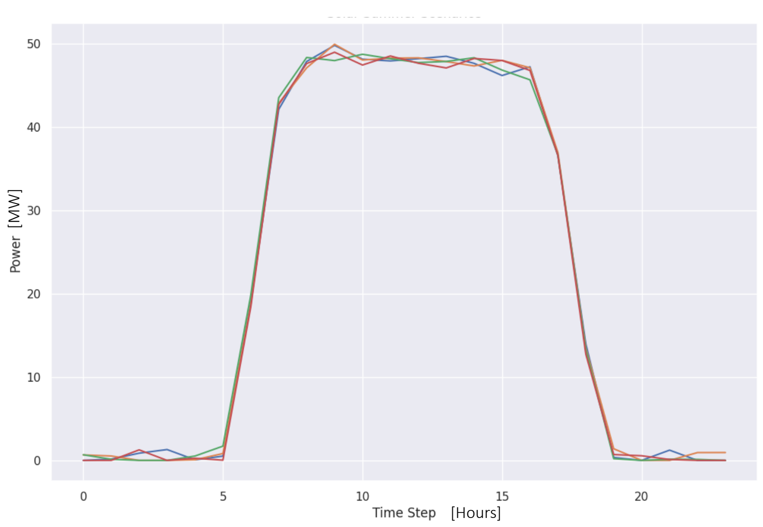


Figure 4-28: Solar Generated Scenarios During Summer

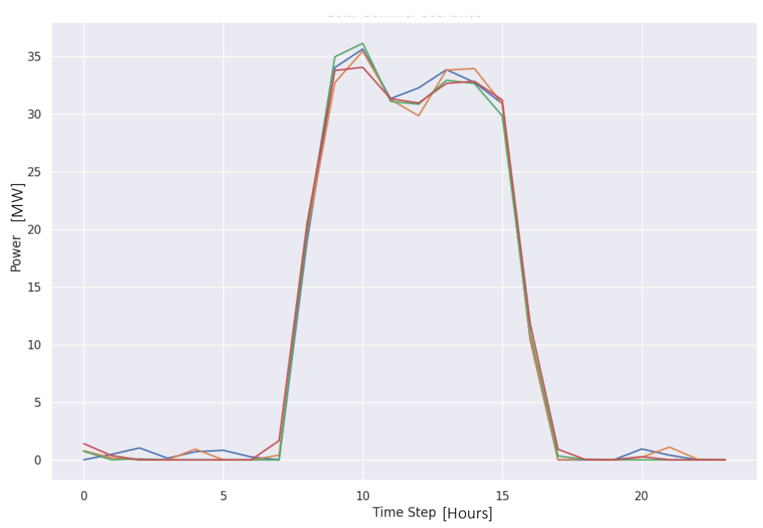


Figure 4-29: Solar Generated Scenarios During Winter

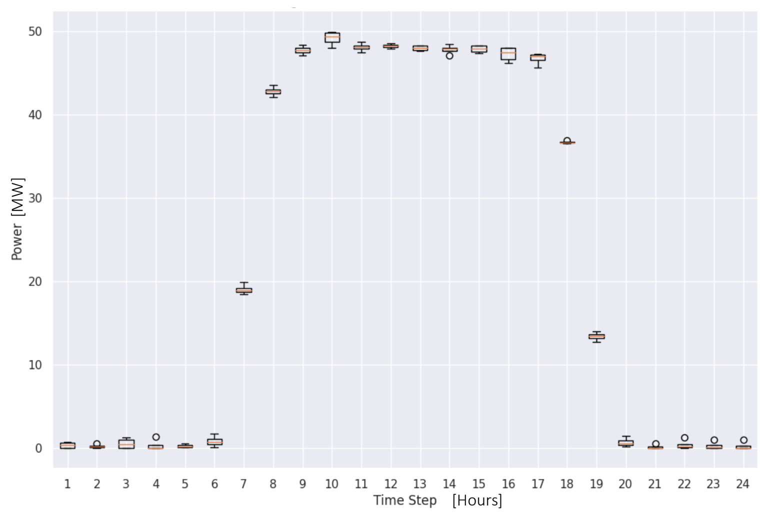


Figure 4-30: Average Hourly Power for Solar Summer Scenarios

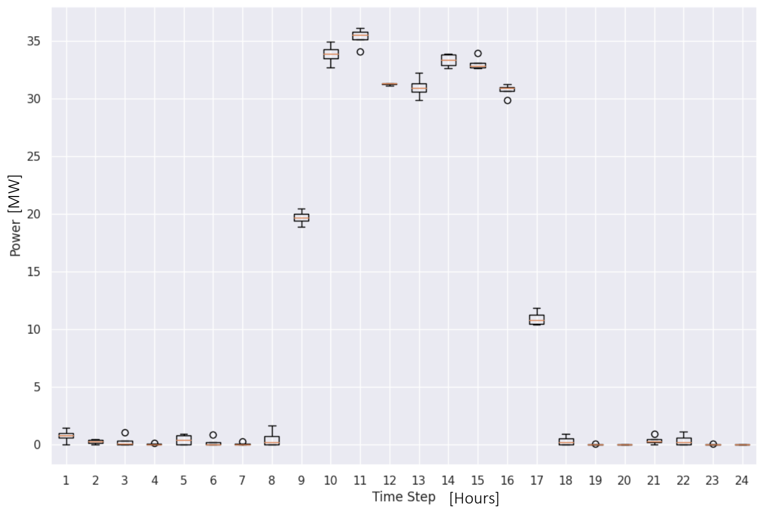


Figure 4-31: Average Hourly Power for Solar Winter Scenarios

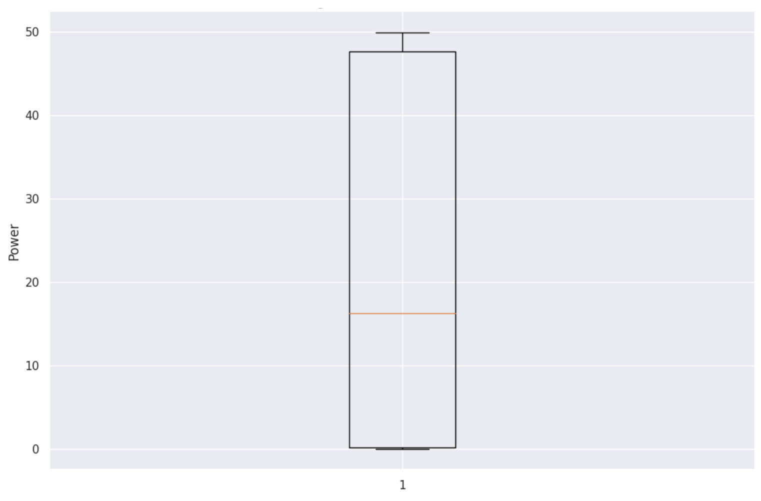


Figure 4-32: Average Daily Power for Summer Scenario

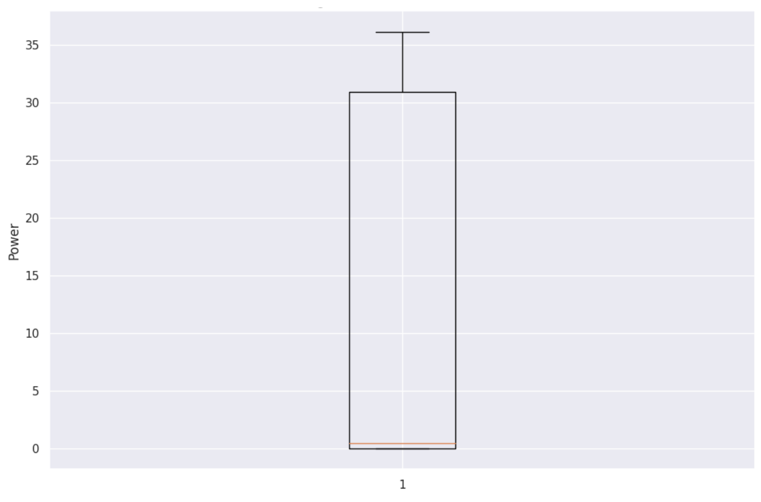


Figure 4-33: Average Daily Power for Winter Scenario

Sampling more scenarios from a given input does generate a little variability to our scenarios as illustrated by summer and winter scenarios for solar farms (seeFigure 4-28-Figure 4-33). Similarly, just like the CNN model, the LSTM-VAE model is able to capture different seasonal behavior for different inputs without being explicitly told. In summer scenarios (see Figure 4-28), the solar farm generates more power and for extended periods of time during the day and early evenings. In winter (see Figure 4-29), the production level drops by 36% and the production hours starts later in the day and drops much earlier compared to summer scenarios. We also notice that unlike the CNN model, LSTM-VAE is able to capture small details about solar farms such as the fact that the production level does not fall below zero (see Figure 4-28). In the average power production per hour (see Figure 4-30 and Figure 4-31), we see little to no variation. This is due to the ability of LSTMs to retain temporal information accurately. Comparing the average power for the generated scenarios during summer and winter (see Figure 4-32 and Figure 4-33), we see that the solar farm produces much more electricity during the summer while the output is close to zero in the winter.

Moving the attention to wind farms, and sampling different scenarios from summer and spring we see the following outputs (see Figure 4-34-Figure 4-39).

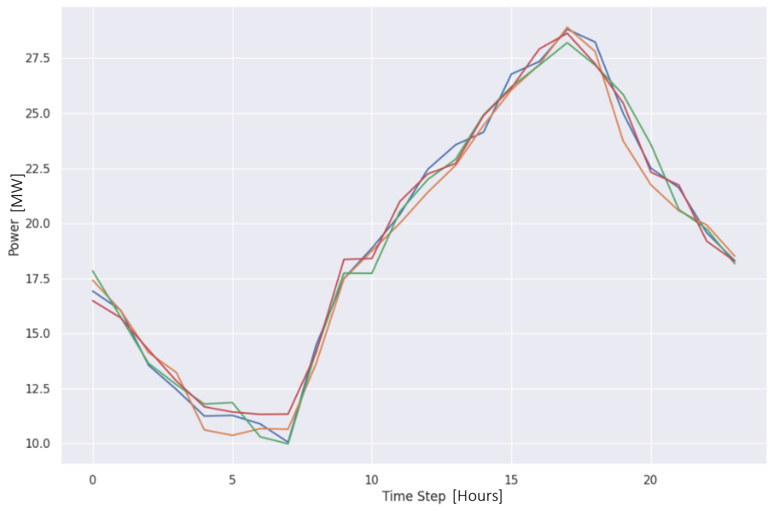


Figure 4-34: Wind Generated Scenarios During Spring

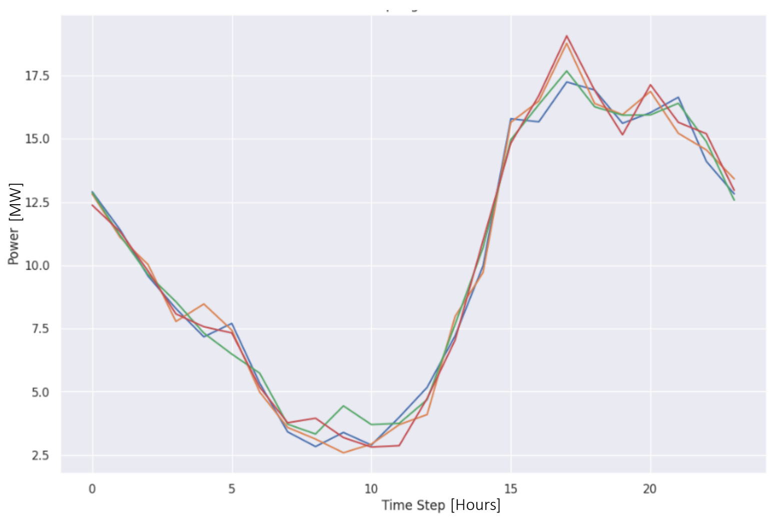


Figure 4-35: Wind Generated Scenarios During Summer

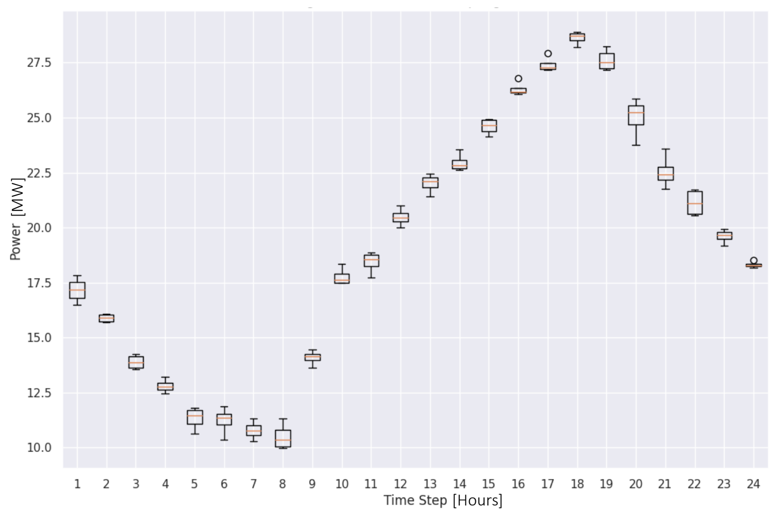


Figure 4-36: Average Hourly Power for Wind Spring Scenarios

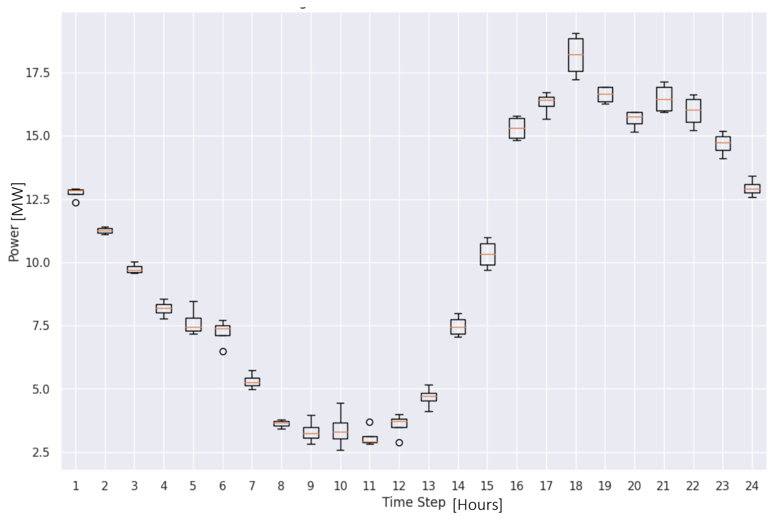


Figure 4-37: Average Hourly Power for Wind Summer Scenarios

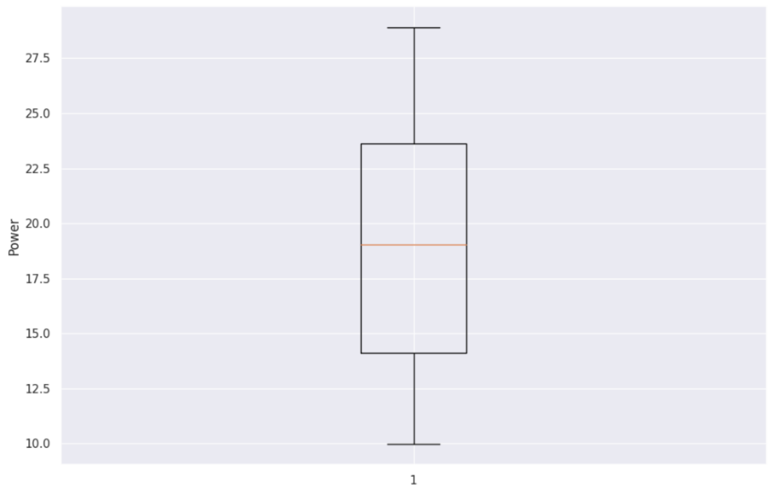


Figure 4-38: Average Daily Power for Spring Scenario

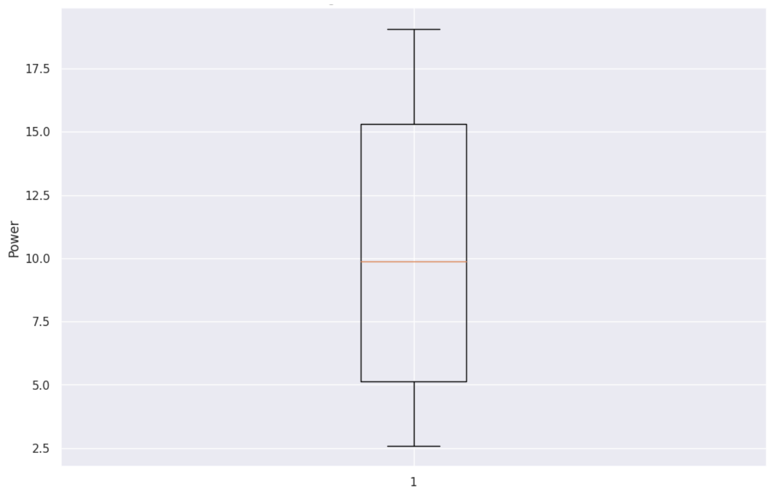


Figure 4-39: Average Daily Power for Summer Scenario

Although the model is able to distinguish between different seasons and generate scenarios accordingly, it still suffers from generating scenarios that look very similar to each other, i.e., with little variations (see Figure 4-34 and Figure 4-35). The spring scenarios tend to have higher production level compared to the summer scenarios and the summer scenarios tend to have a bigger dip early in the day. Generally, the average power of the generated scenarios during spring is higher compared to the average power production during summer season (see Figure 4-38 and Figure 4-39).

The load scenarios during weekday and weekend are very distinct (see Figure 4-40 and Figure 4-41) and the model is able to capture that but the same invariability between scenarios is observed (see Figure 4-42 and Figure 4-43).

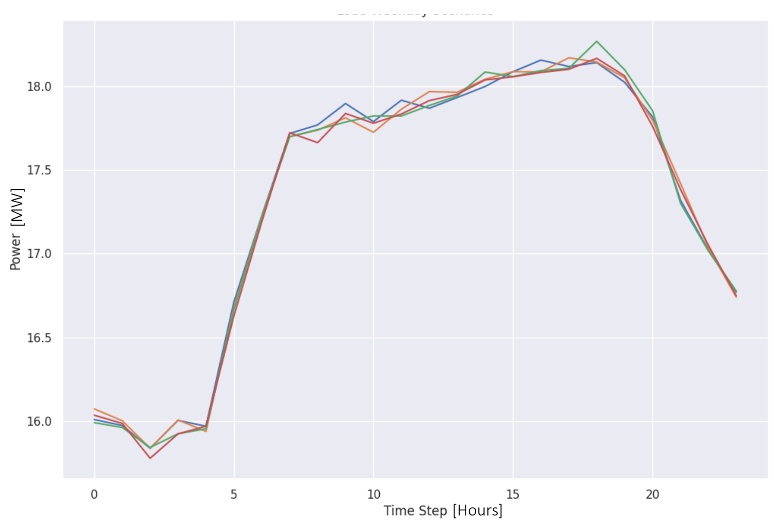


Figure 4-40: Load Generated Scenarios During Weekday

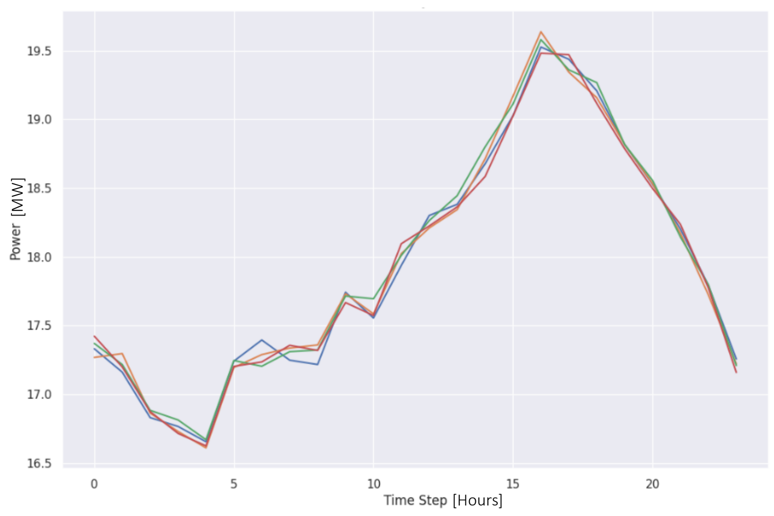


Figure 4-41: Load Generated Scenarios During Weekend

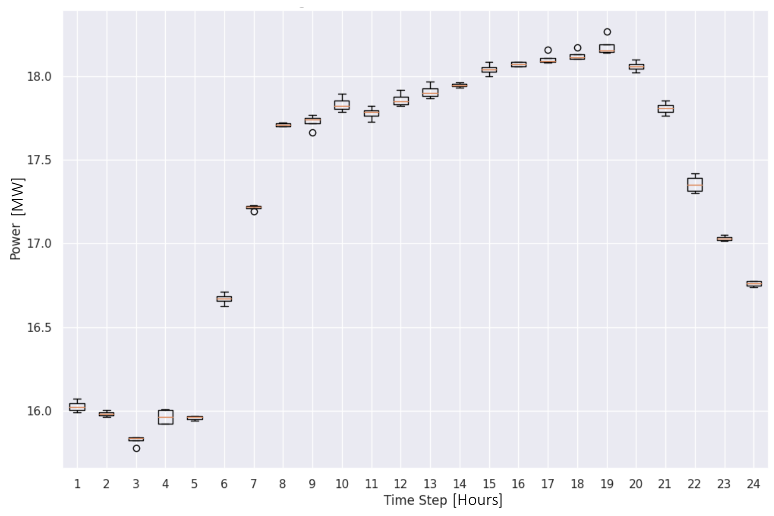


Figure 4-42: Average Hourly Demand for Load Weekday Scenarios

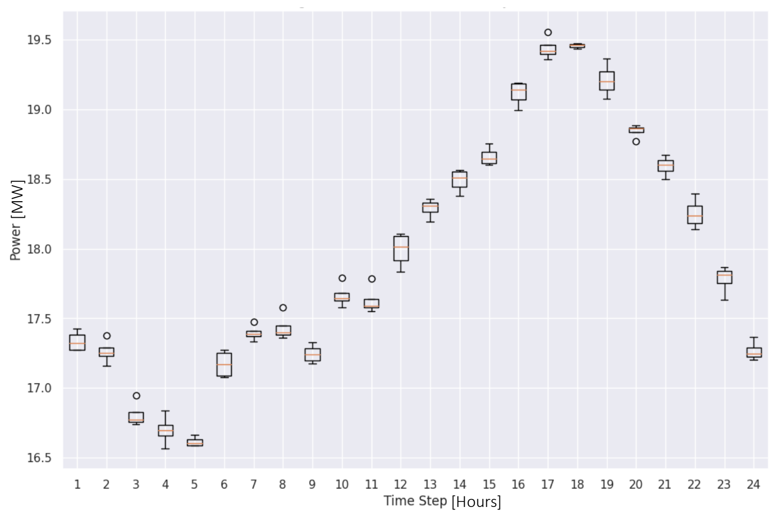


Figure 4-43: Average Hourly Demand for Load Weekend Scenarios

4.6.4 Autocorrelation and Error Metrics:

Sampling generated scenarios and comparing them temporally to real scenarios using the ACF gives us insights into the quality of the generated scenarios, particularly if they capture the temporal behavior of each variable.

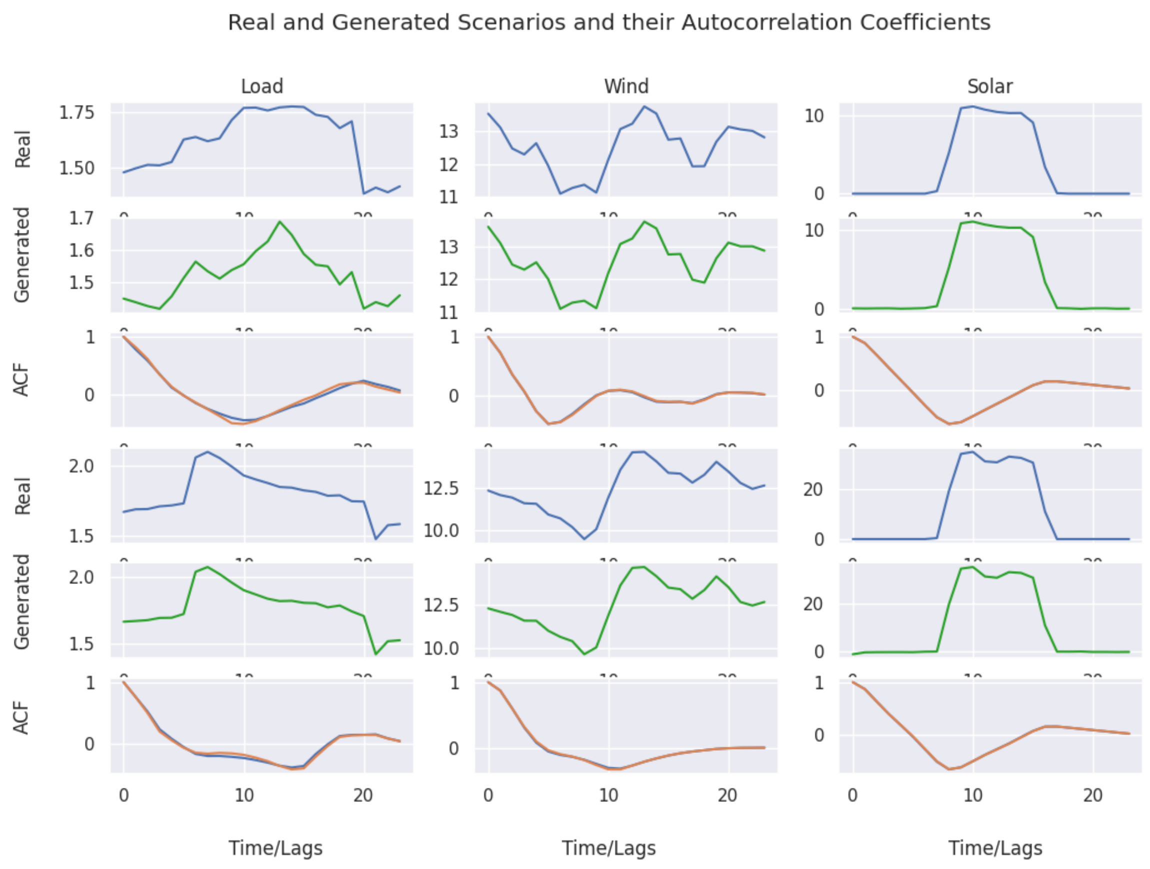


Figure 4-44: Autocorrelation of Real and Generated Scenarios

Looking at the ACF plot (see Figure 4-44), we see that the algorithm mimics the temporal behavior of all input scenarios perfectly and that further verifies our assumption that LSTM-VAE solves the forecasting problem rather than the scenario generation problem. With all the variables, ACF of the real and generated scenarios are undistinguishable and the model does not struggle with any type of variable in particular.

Next we look at the CDFs (see Figure 4-45 and Figure 4-47) of both the real and generated data to see if they mimic the temporal behavior perfectly as seen from the ACF.

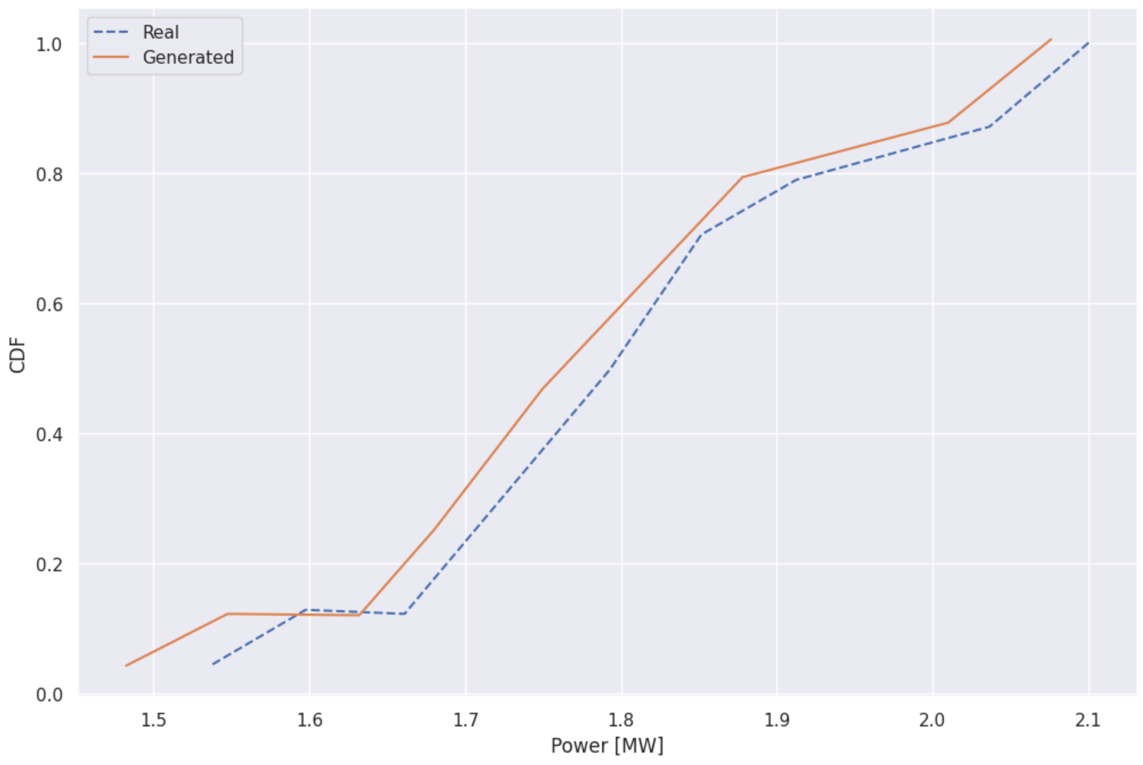


Figure 4-45: CDF of Real and Generated Load Scenarios

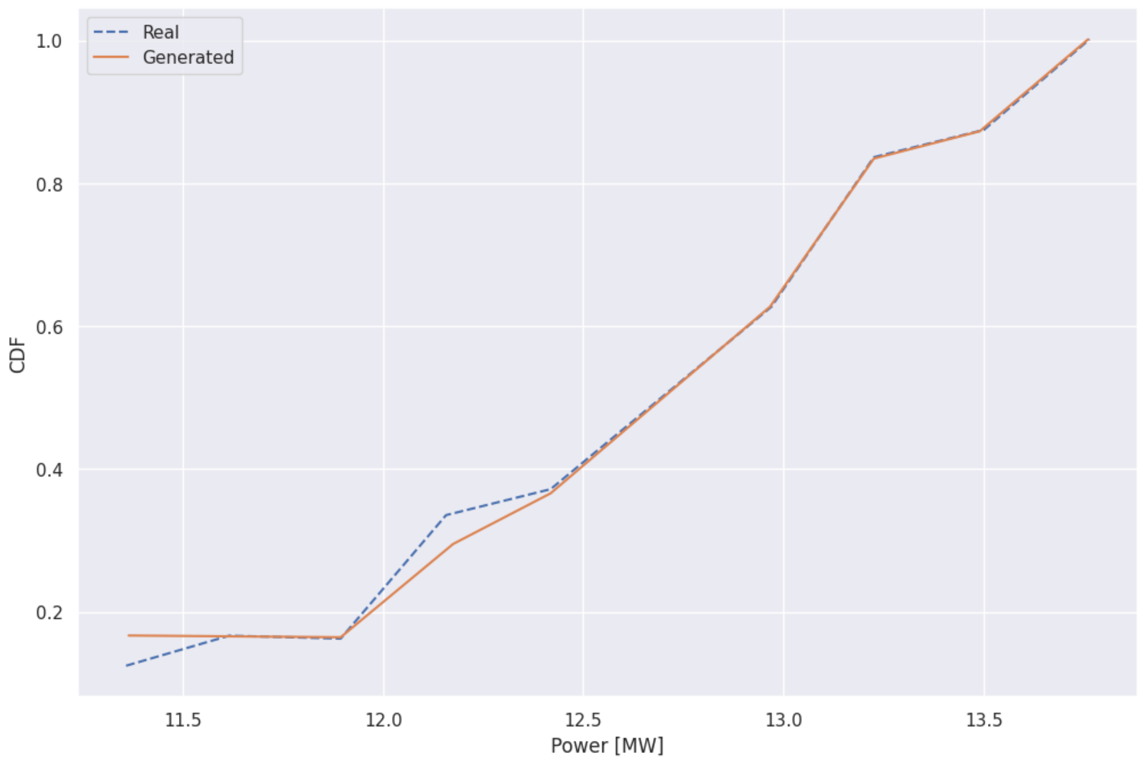


Figure 4-46: CDF of Real and Generated Wind Scenarios

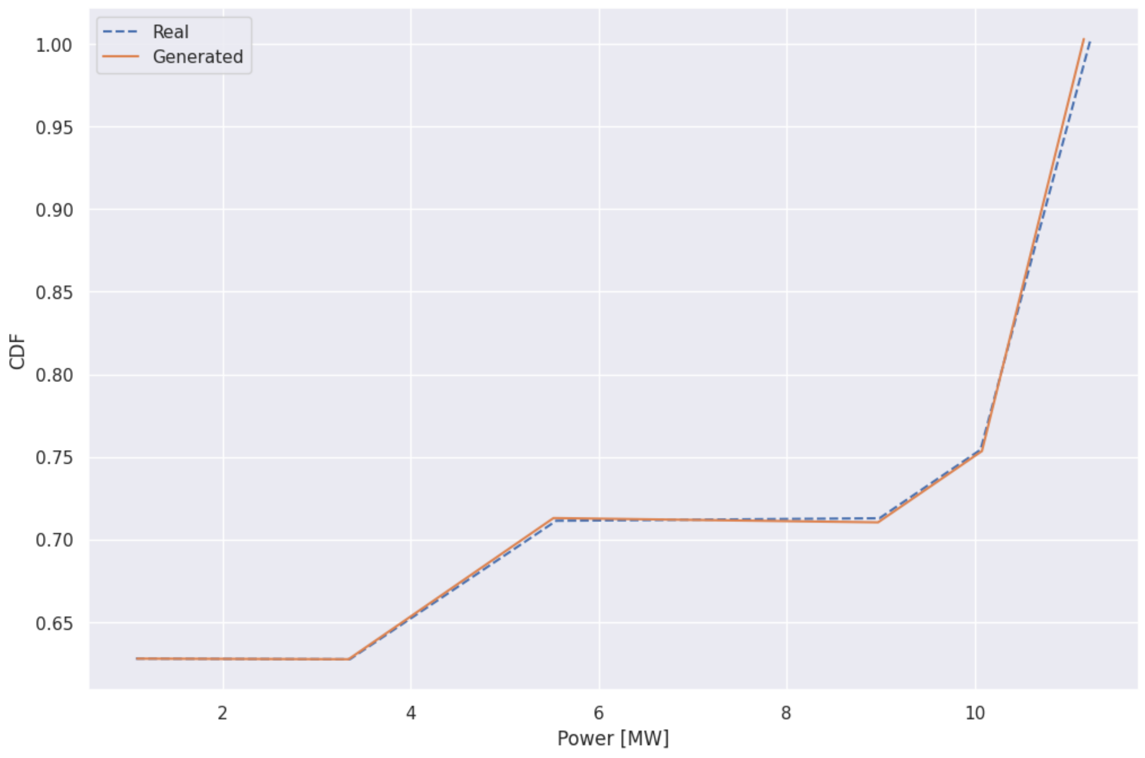


Figure 4-47: CDF of Real and Generated Solar Scenarios

It is evident from the above plots that the CDF of real and generated data are a match.

4.6.5 Error Metrics:

The results of MAE and RMSE shown below further validate the inference that the LSTM-VAE model is suitable for forecasting across all variables.

Table 4-4: MAE and RMSE for LSTM-VAE

	Load	Solar	Wind
MAE	0.397	0.193	0.296
RMSE	0.264	0.051	0.176

4.7 Conditional GANs

GANs are the epitome of generation models. They are able to generate images, sequences, codes, sentences, and much more [36]. However, GANs are infamous when it comes to training [37]. The generator needs to learn the dataset distribution and fool the discriminator. However, more often than not, the generator finds an easy way of fooling the discriminator, but not by actually mimicking the distribution of the dataset. Sampling from this generator would yield noise or in some cases zeros, which is known as model collapse as the generator is not able to learn the data distribution but is able to outsmart the discriminator. There are abundant ways to facilitate the training process and mitigate the model collapse problem such as using Wasserstein distance and improved GANs. Although these methods work, their scalability is still a concern because the generator needs to learn different distributions.

To put this into test, we start with a simple problem where the generator has to learn three different variables. We evaluate the results to check if it is successful. Subsequently, we will try to increase the number of variables and conditions to 10 (2 wind farms, 2 solar farms and 6 loads) and 40 (4 seasons per variable), respectively. If the model is able to perform well, only then we will test it on the IEEE-30 bus system with 30 variables and 120 conditions.

4.7.1 Data:

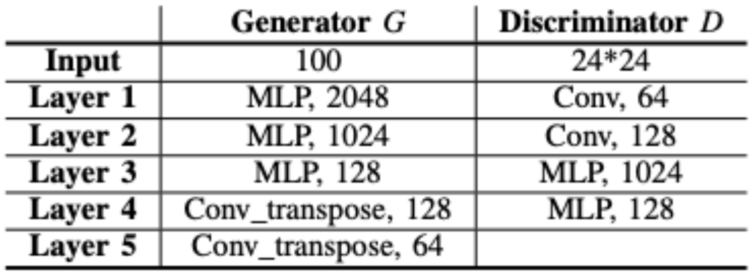
To test the algorithm, we are initially training and testing on three variables. The input data to the algorithm has the following structure:

- Original input matrix has 2 dimensions where the rows are the time steps and the columns are the number of variables, namely three (wind, solar, and load).

- Since GANs are not able to distinguish between the different variables, we also need integer representation for those variables. This is as follows: 0 = wind, 1 = solar, 2 = load
- The input data preprocessor function will normalize the data, generate input and output data sequences, and train, validate and test data sequence, as well as one-hot encode the integer representation of the variables. The input and output sequence matrices have 2 and 2 dimensions, respectively, where the input matrix has the final shape of (number_of_samples, input_time_step+encoding_of_the_variable) and the output matrix has the shape of (number_of_samples, output_time_step)

4.7.2 Hyperparameters:

The hyperparameters were taken from [24].



	Generator *G*	**Discriminator** *D*
Input	100	24*24
Layer 1	MLP, 2048	Conv, 64
Layer 2	MLP, 1024	Conv, 128
Layer 3	MLP, 128	MLP, 1024
Layer 4	Conv_transpose, 128	MLP, 128
Layer 5	Conv_transpose, 64	

Figure 4-48: C-GANs Hyperparameters [24]

On the left-hand side of each column in Figure 4-48, we see the type of layer used and on the right-hand side we see the size of the output. MLP is a linear layer and Conv is a convolutional layer (CNN) and Conv_transpose is a deconvolution layer that has the opposite structure than the Conv layer.

4.7.3 Results:

Unlike the previous models, the input to the trained model is not a time series of solar, wind, or load. Instead, the input to the model is a random latent vector z sampled from a normal distribution and an encoded value for the variable (0=wind, 1=solar, and 2=load), which makes it very difficult to control the output. We only have control using the stated conditions (wind, solar, or load). For the previous algorithms, we used to input a summer scenario for solar or a weekday for load and expect results that behave similarly to the input. For C-GANs we sample and evaluate the results to see if they conform to a certain seasonal characteristic for the given variable.

However, since the input is a random variable, the need for a Bayesian layer is no longer a must and as the input is random (and not a real scenario), we can generate distinct scenarios from different seasons and days simultaneously. This can be seen as a huge advantage for scenario generation as the algorithm is not governed by the input time series and can generate representative and extreme scenarios at the same time.

Looking at generated scenarios for solar (see Figure 4-49), we can see that the results are different from our previous generated scenarios. We see that the output is higher than all previously generated scenarios but at the same time, the duration of the power generated during the day is less compared to the previous results, the average power generated (see Figure 4-51) is less and we can also see higher variability (see Figure 4-50) in the results, especially during ramp-up and ramp-down, which was not captured by the previous algorithms.

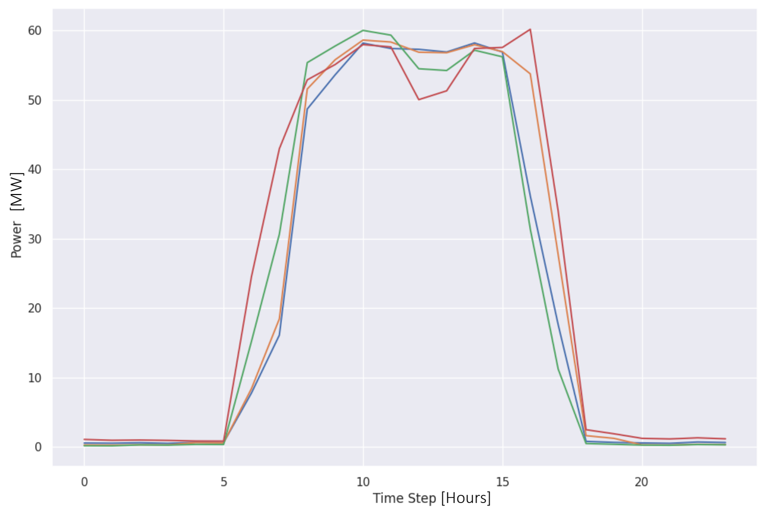


Figure 4-49: Solar Generated Scenarios During Summer

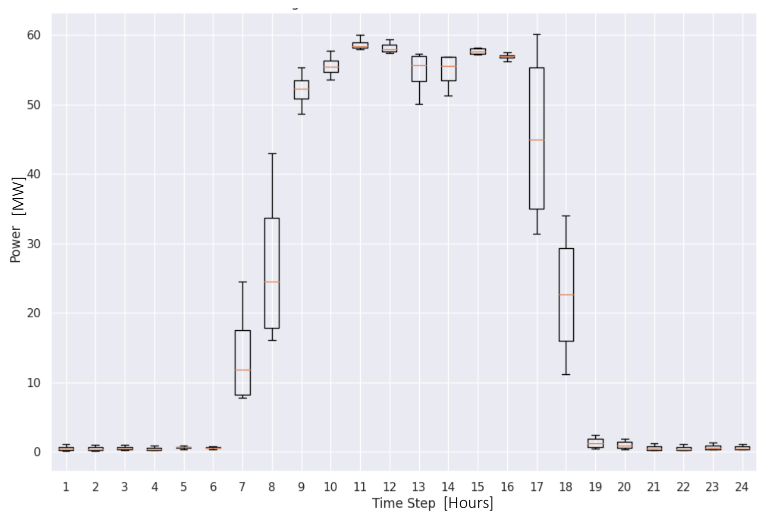


Figure 4-50: Average Hourly Power for Solar Summer Scenarios

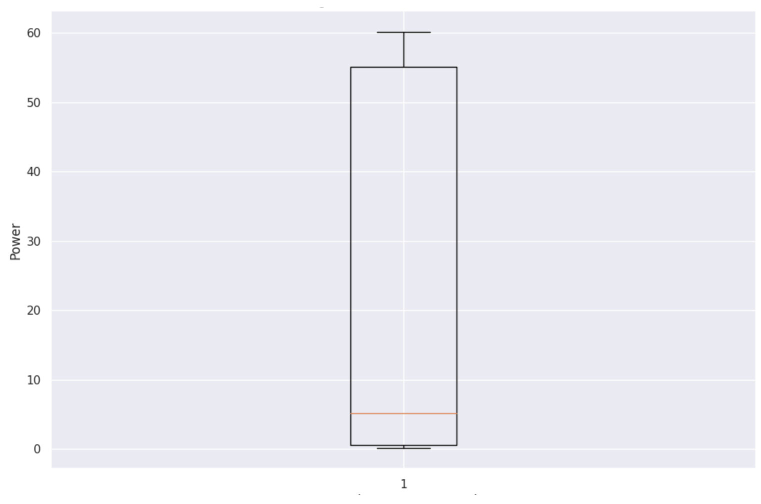


Figure 4-51: Average Daily Power for Summer Scenario

Looking at wind or load generated scenarios does not give us a lot of information about seasonal or weekly data. Generating data for wind scenarios results in the scenarios shown below (see Figure 4-52-Figure 4-54). The same conclusion can be drawn as the solar scenarios regarding variability and the difficulty to categorize these scenarios into a single season.

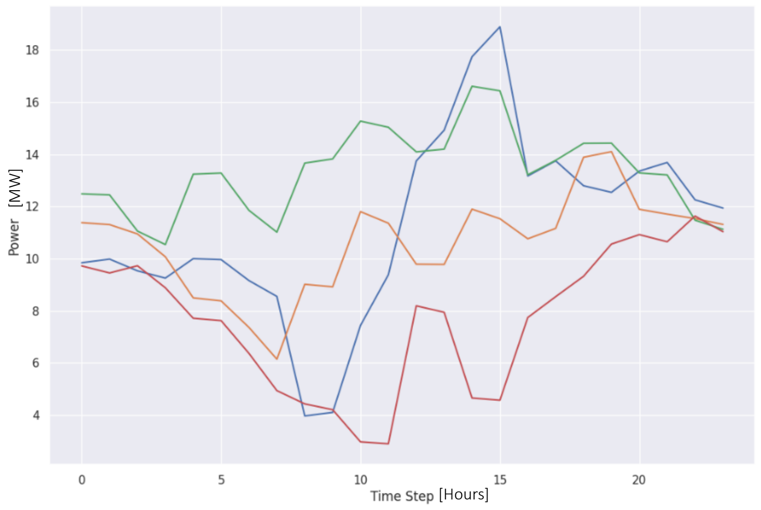


Figure 4-52: Wind Generated Scenarios

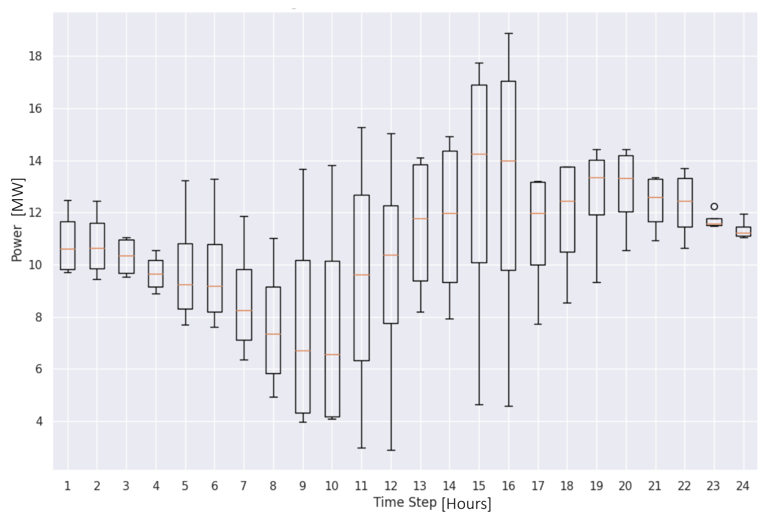


Figure 4-53: Average Hourly Power for Wind Scenarios

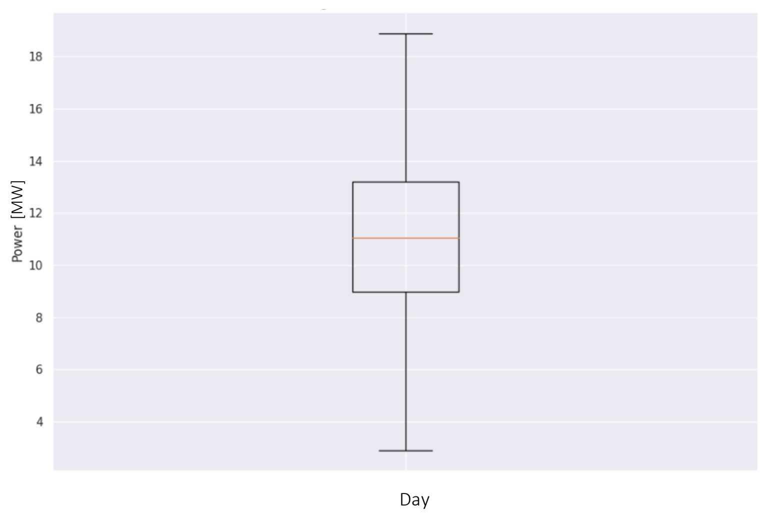


Figure 4-54: Average Daily Power for Wind Scenario

Nonetheless, the C-GAN works and is able to generate distinct scenarios which was the ultimate goal of this test. Next, we scale up the data and investigate the ability of the model to work with 10 variables and 40 conditions. The hyperparameters of the generator and the discriminator do not change. The only change happens in the number of inputs and the one-hot encoding of the different variables.

Sampling outputs from the scaled-up version of the model gives the results shown below (see Figure 4-55-Figure 4-57).

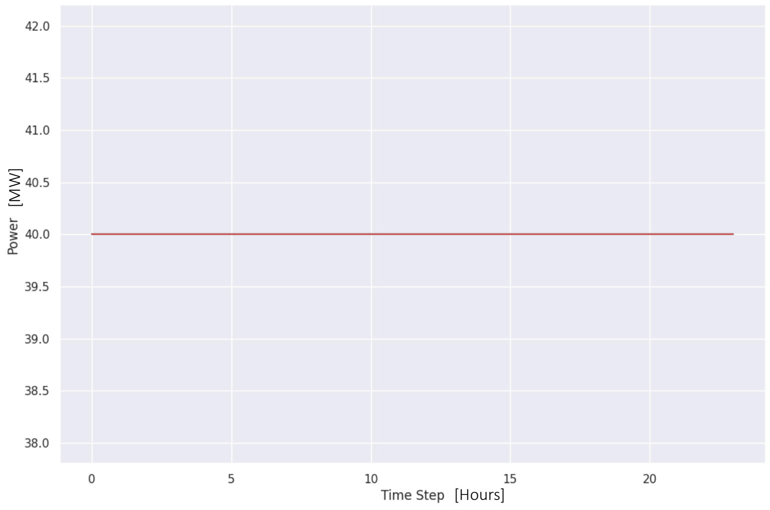


Figure 4-55: Solar Generated Scenarios

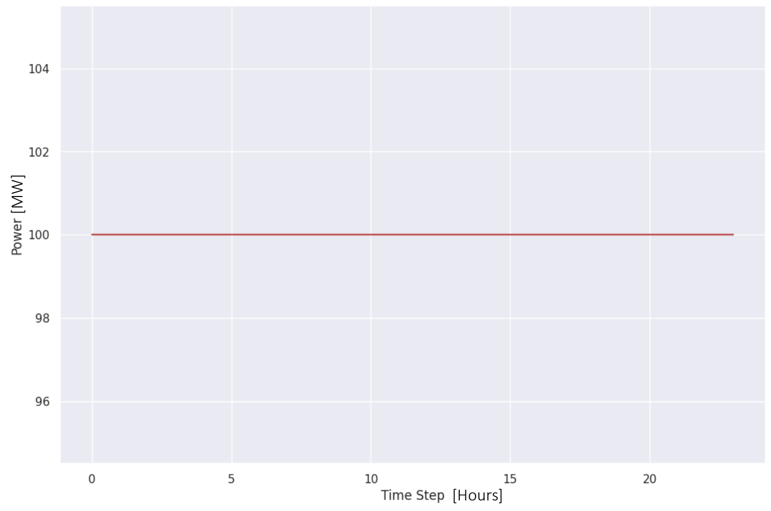


Figure 4-56: Load Generated Scenarios

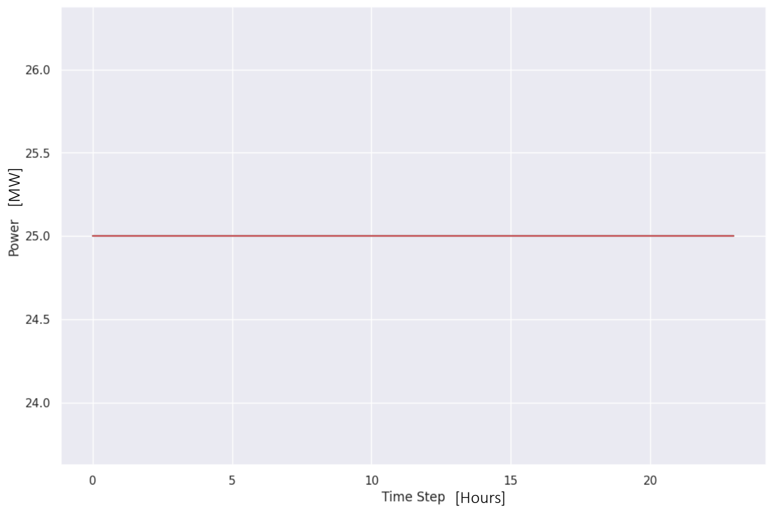


Figure 4-57: Wind Generated Scenarios

It is clear from Figure 4-55-Figure 4-57 that the model has failed and collapsed during training due to the number of conditions that the generator tries to learn, especially when those distributions overlap. To further investigate this issue, we sample results during the training process of the generator for the initial 3 variable case and the scaled-up 10 variable case. The results obtained are as follows.

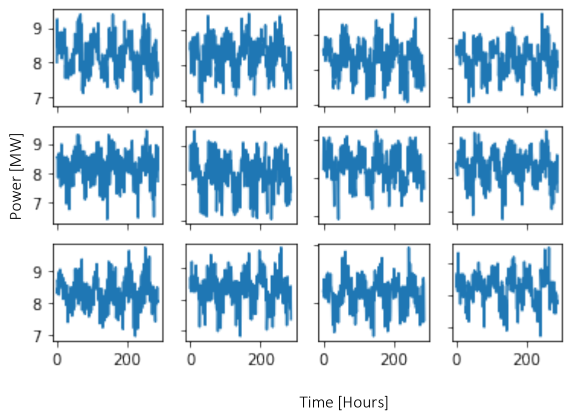


Figure 4-58: Sampled Output During Training (Epoch = 50)

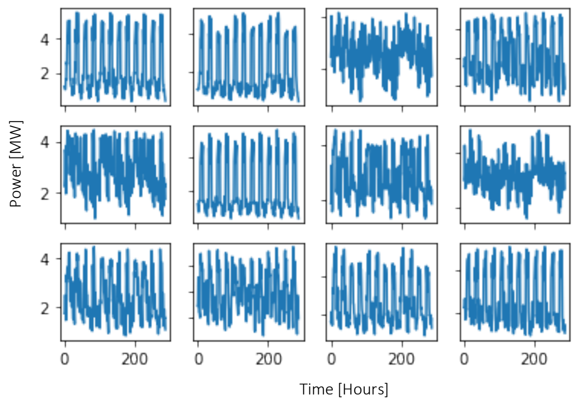


Figure 4-59: Sampled Output During Training (Epoch = 100)

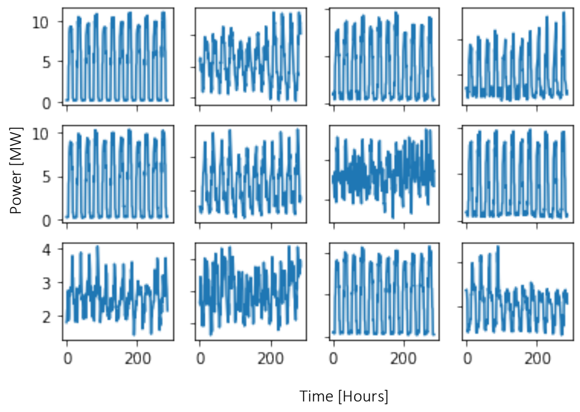


Figure 4-60: Sampled Output During Training (Epoch = 150)

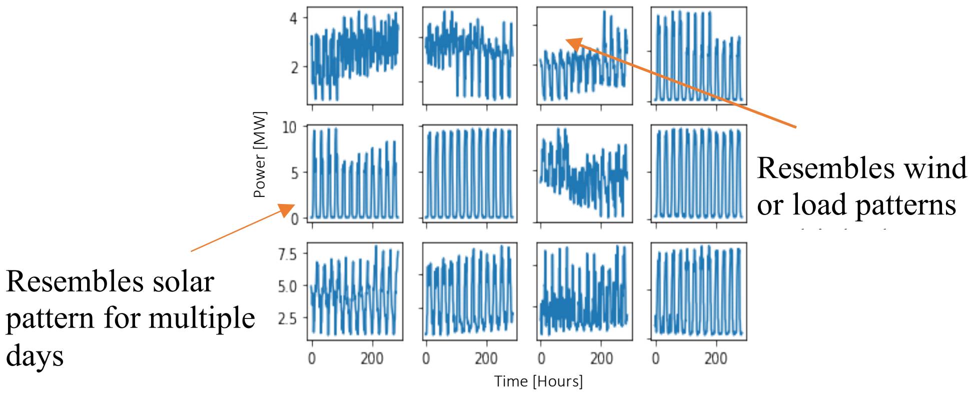


Figure 4-61: Sampled Output During Training (Epoch = 200)

Figure 4-58-Figure 4-61 show the progression of the training process for the three-variable case. We are able to see that the generator starts with random variables but over the training epochs, it is able to learn the different distributions of the variables. Looking closely at the generated scenarios, one can also distinguish solar scenarios from wind or load (see Figure 4-61).

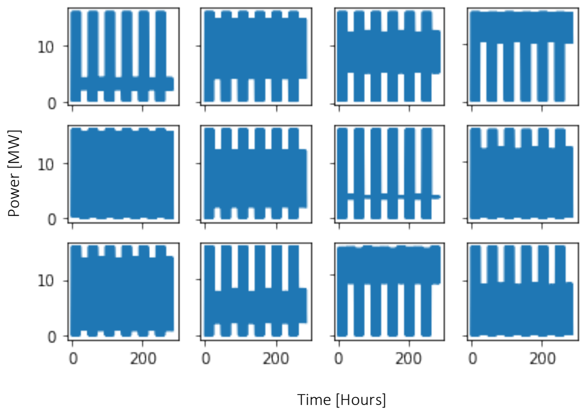


Figure 4-62: Sampled Output During Training (Epoch = 50)

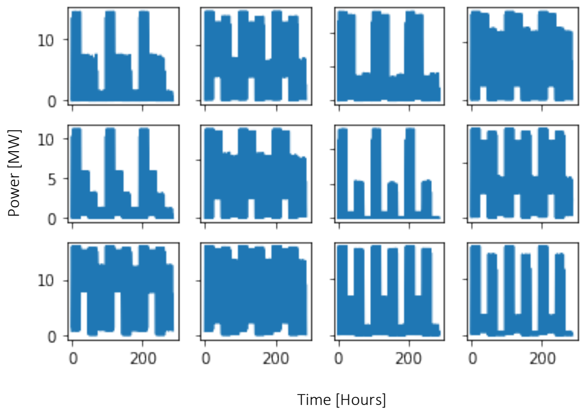


Figure 4-63: Sampled Output During Training (Epoch = 100)

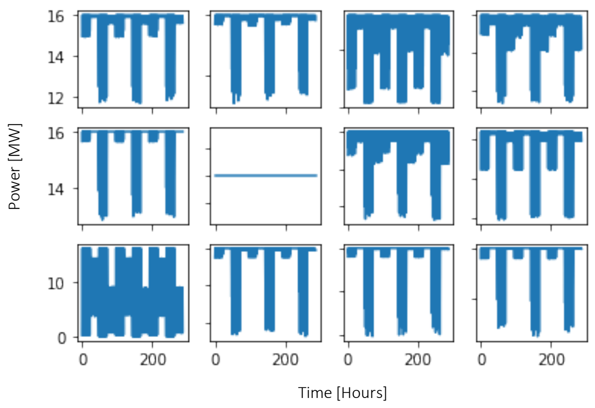


Figure 4-64: Sampled Output During Training (Epoch = 150)

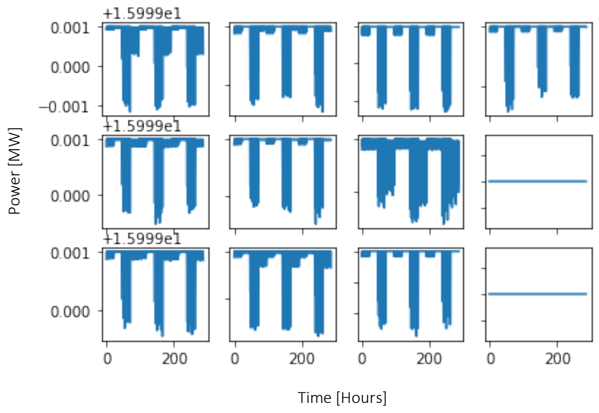


Figure 4-65: Sampled Output During Training (Epoch = 200)

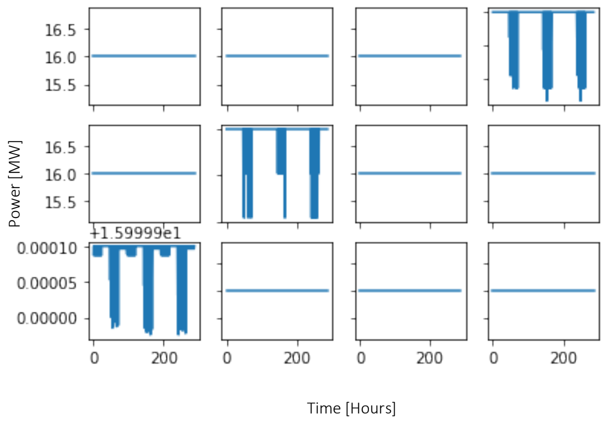


Figure 4-66: Sampled Output During Training (Epoch =250)

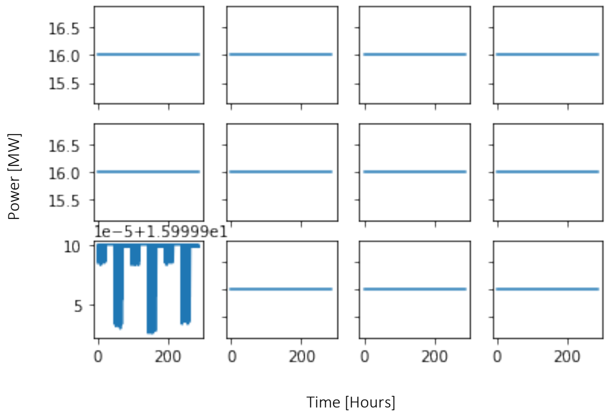


Figure 4-67: Sampled Output During Training (Epoch = 300)

Figure 4-62-Figure 4-67 show the progression of the training process for the 10 variable case. We are able to see that the generator starts with random variables, and over the training epochs, tries to learn the different distributions of the variables. However, over time, it collapses and starts generating constant values that do not relate to the original dataset.

4.8 STGCN

STGCN is a type of graph neural network that does sequential temporal and spatial manipulations over the data based on the raw input data and an adjacency matrix. Ybus is used as the adjacency matrix in this thesis. The algorithm tries to do a node level (bus level) predictions for all the nodes in the system simultaneously.

4.8.1 Data:

To test the algorithm, we use the IEEE-30 bus test case illustrated previously. The input data to the algorithm has the following structure:

- Original input matrix has 2 dimensions where the rows are the time steps and the columns are the number of variables (e.g., 30 variables in the IEEE-30 bus system), in the network.
- The input data preprocessor function will normalize the data, generate input and output data sequences, and train, validate and test data sequence. The input and output sequence matrices have 3 and 2 dimensions, respectively, where the input matrix has the final shape of (input_time_step , number_of_samples, number of variables) and the output matrix has the shape of (output_time_step, number of variables).

4.8.2 Hyperparameter:

Input Horizon	24
Output Horizon	24
Batch size	50
Number of Epochs	200
Learning Rate	0.001
Optimizer	Adam
Loss	Mean Squared Error
Data	
Train	80%
Validation	10%
Testing	10%

These parameters were obtained from [31].

4.8.3 Results:

After training the model, we first test it visually to see if the model is able to generate distinct scenarios that have similar temporal patterns. Inputting random sampled signals and sampling two scenarios we obtain the following outputs.

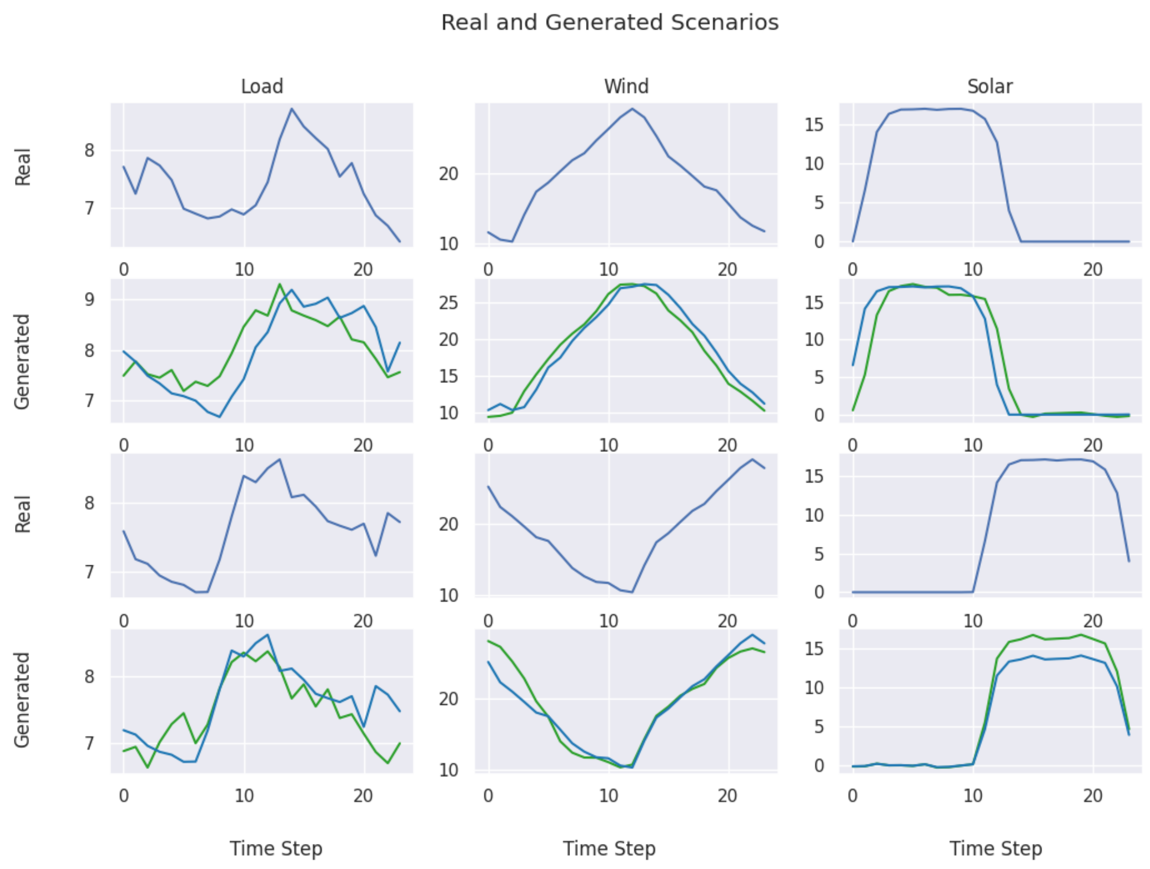


Figure 4-68: Sample Generated Scenarios

Unlike the LSTM-VAE model, it is very clear from Figure 4-68 that the model is able to generate distinct scenarios that follow the same temporal pattern as the input. Now we need to validate and compare the model with the baseline inputs that were used in the previous algorithms.

Testing the model on summer and winter solar scenarios yields the following outputs.

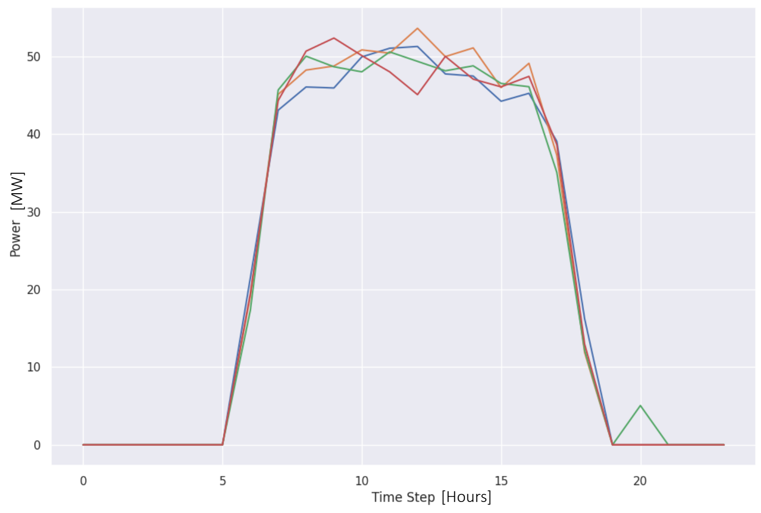


Figure 4-69: Solar Generated Scenarios During Summer

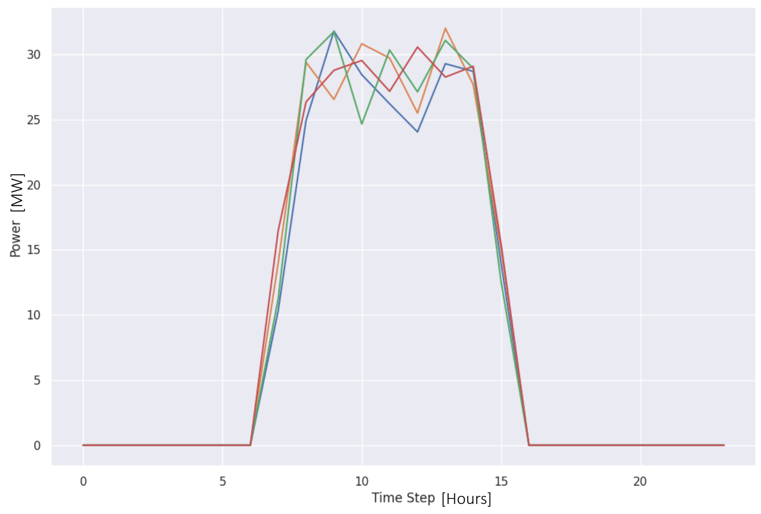


Figure 4-70: Solar Generated Scenarios During Winter

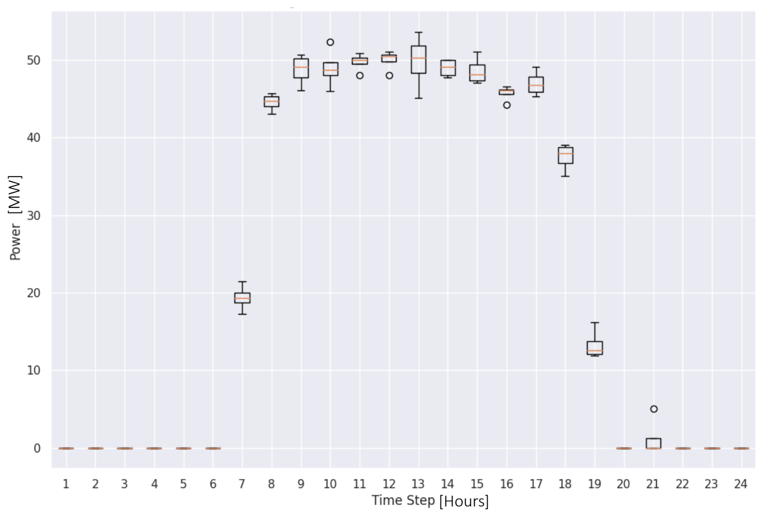


Figure 4-71: Average Hourly Power for Solar Summer Scenarios

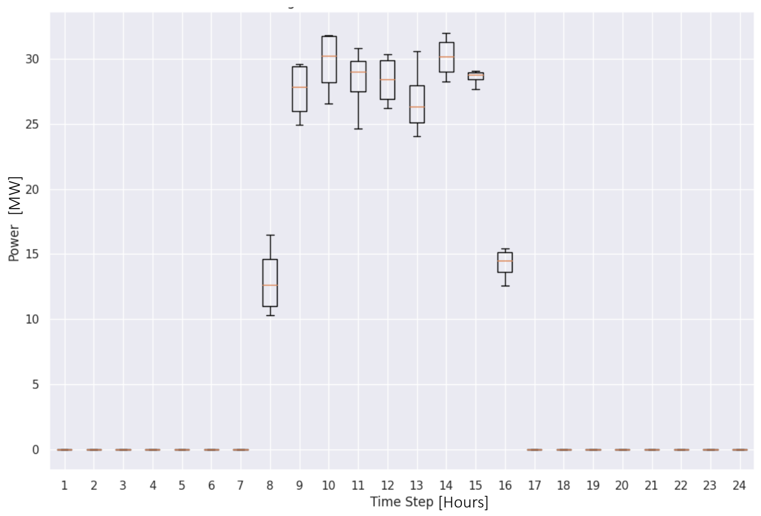


Figure 4-72: Average Hourly Power for Solar Winter Scenarios

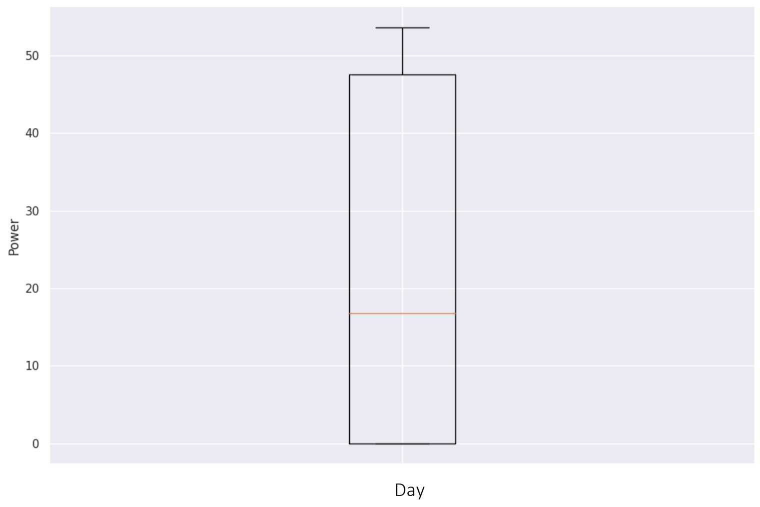


Figure 4-73: Average Daily Power for Summer Scenario

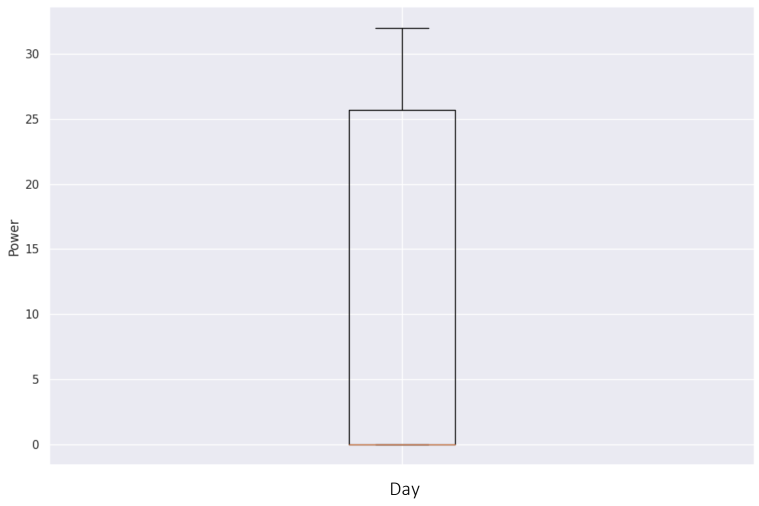


Figure 4-74: Average Daily Power for Winter Scenario

Similar to the previous algorithms (CNN and LSTM-VAE), and as seen in Figure 4-69-Figure 4-74, the (STGCN) model is able to capture key differences between solar farm outputs in summer and winter seasons accurately. However, unlike the CNNs, the model learns that the power does not go below zero (see Figure 4-69). The means of the generated scenarios are actually closer to the means of the generated scenarios using LSTM-VAE indicating that the model is also able to generate representative scenarios that are closer to the forecasted scenarios.

Examining the wind farm generated scenarios during summer and spring seasons by utilizing the same input used before for comparison, generates the following results.

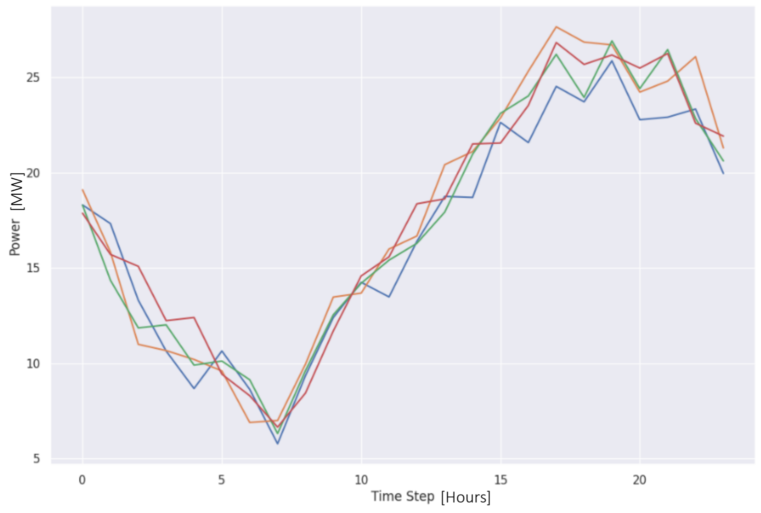


Figure 4-75: Wind Generated Scenarios During Spring

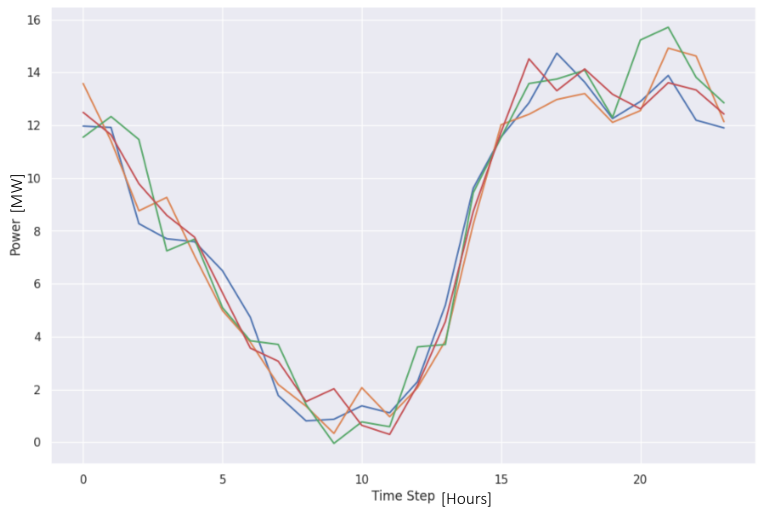


Figure 4-76: Wind Generated Scenarios During Summer

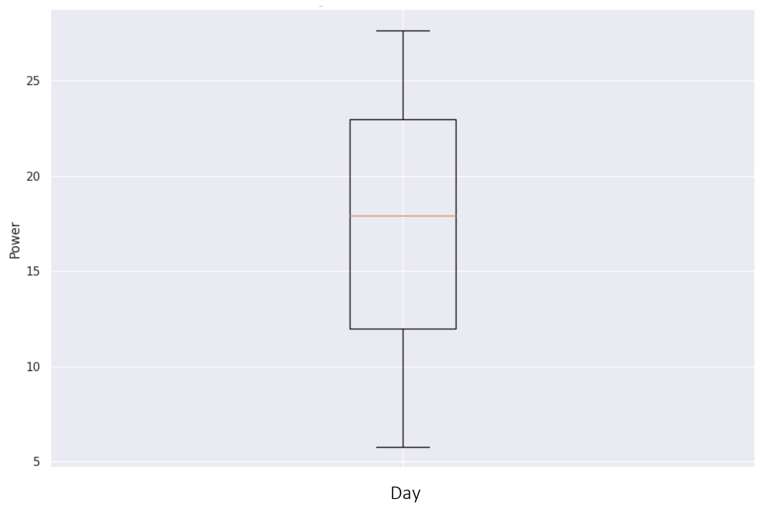


Figure 4-77: Average Daily Power for Spring Scenario

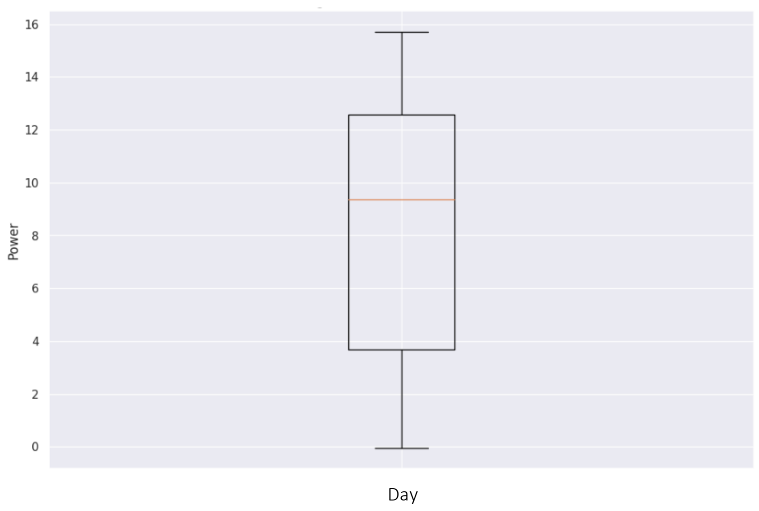


Figure 4-78: Average Daily Power for Summer Scenario

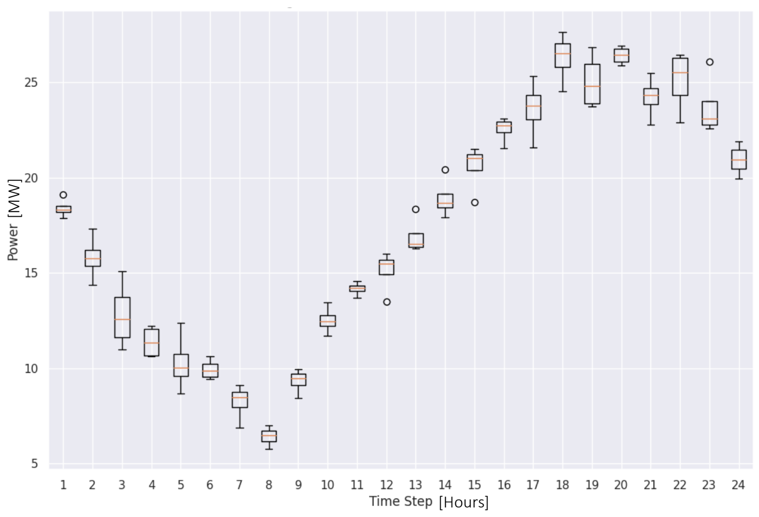


Figure 4-79: Average Hourly Power for Wind Spring Scenarios

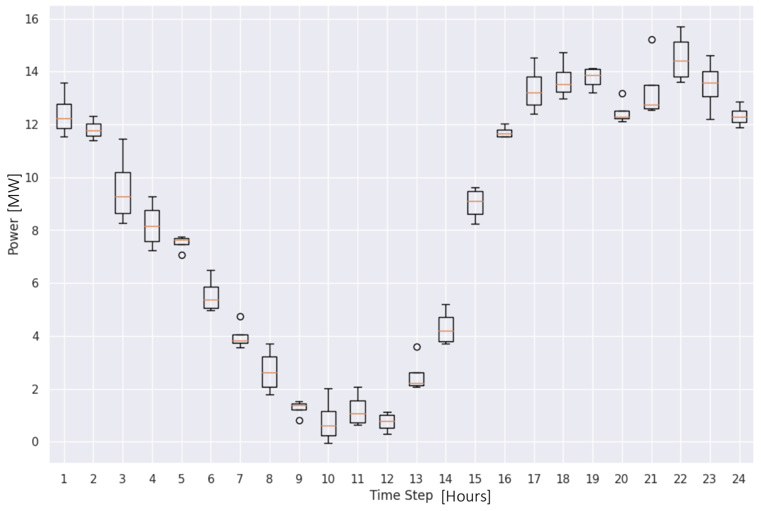


Figure 4-80: Average Hourly Power for Wind Summer Scenarios

The model is able to capture seasonal differences in the wind behavior for summer and spring seasons (see Figure 4-75-Figure 4-80). Interestingly, the model was also able to capture the big dip in power during the day for the summer scenarios (see Figure 4-76) just like the LSTM-VAE model. Note that this behavior was not captured by the CNN model.

Evaluating the performance of the model on weekday and weekend load scenarios generate the following outputs.

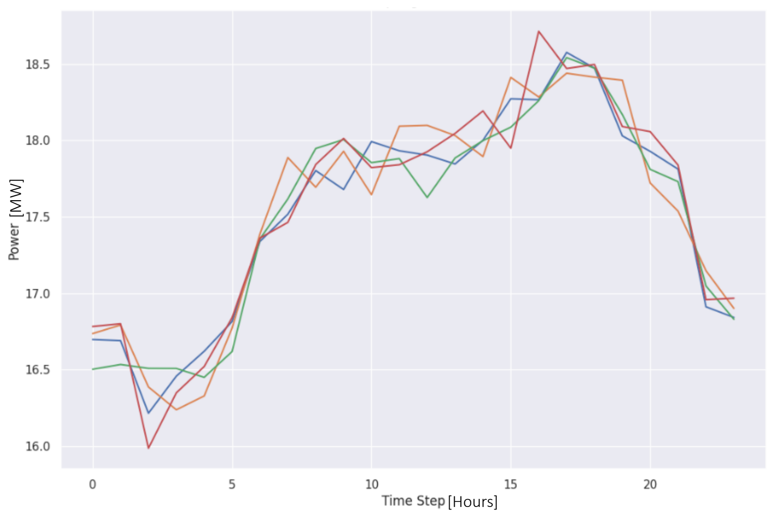


Figure 4-81: Load Generated Scenarios During Weekday

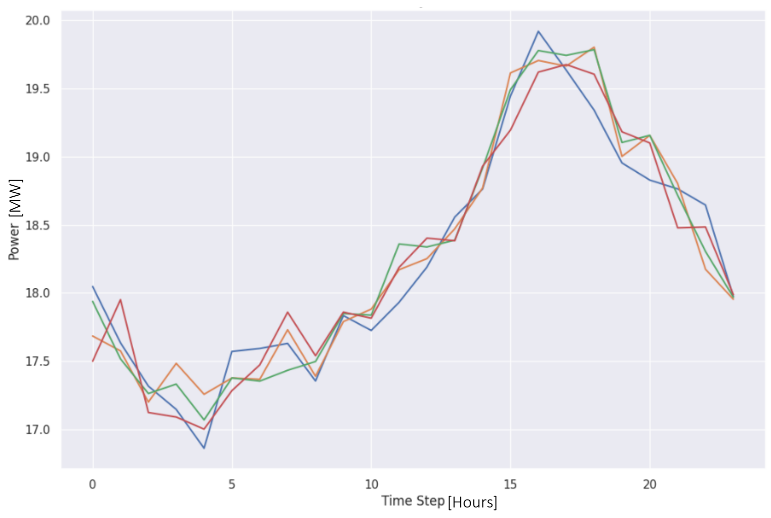


Figure 4-82: Load Generated Scenarios During Weekend

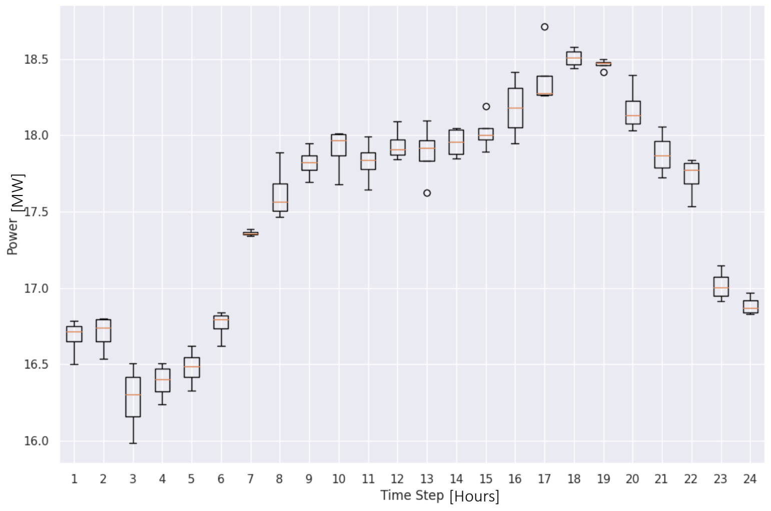


Figure 4-83: Average Hourly Demand for Load Weekday Scenarios

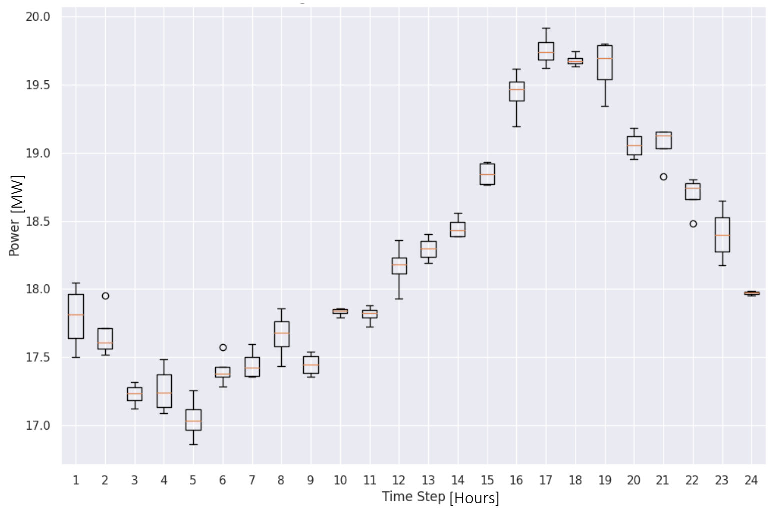


Figure 4-84: Average Hourly Demand for Load Weekend Scenarios

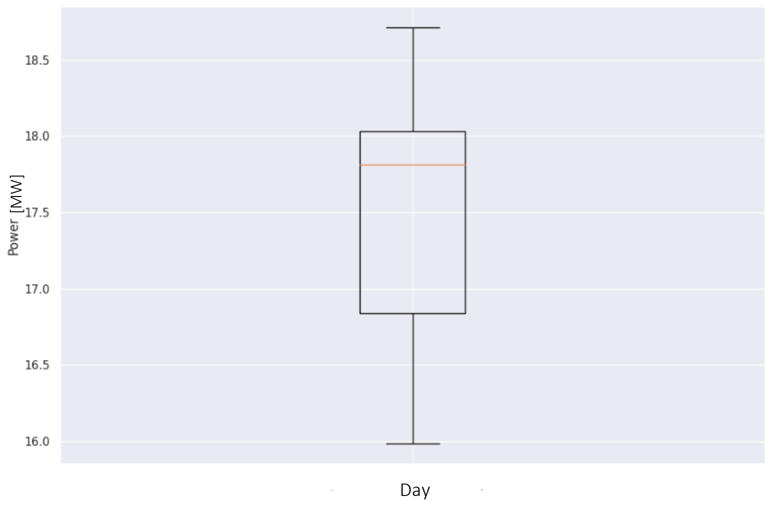


Figure 4-85: Average Daily Demand for Weekday Scenario

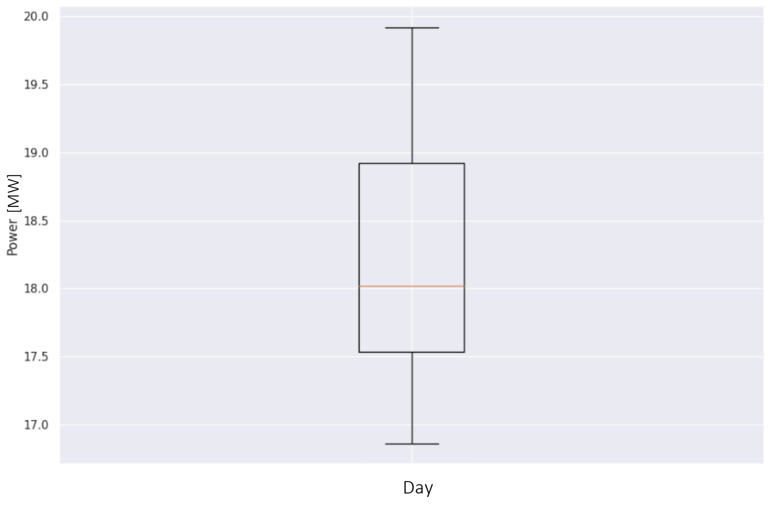


Figure 4-86: Average Daily Demand for Weekend Scenario

As can be seen from Figure 4-81-Figure 4-86, the model is able to capture the high pattern of demand during the day and the high variability early in the morning and late in the afternoon (see Figure 4-83).

4.8.4 Autocorrelation and Error Metrics:

To test for autocorrelation and to see if the generated and real scenarios are characterized by the same temporal behavior, we perform the ACF test.

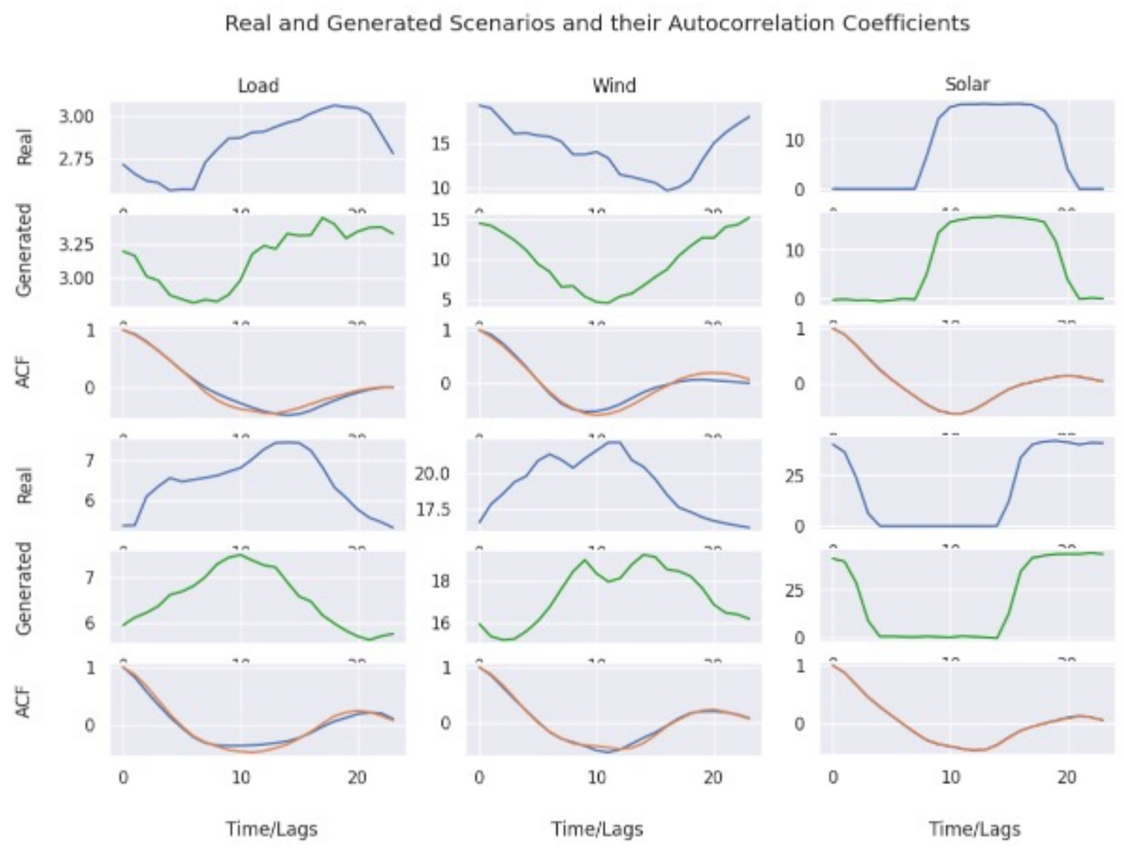


Figure 4-87: Autocorrelation of Real and Generated Scenarios

Looking at the ACF plots in Figure 4-87, we see that the algorithm mimics the temporal behavior of all input scenarios very well. This is an indication that the algorithm is suitable for forecasting problems.

Next we look at the CDFs (Figure 4-88-Figure 4-90) of both the real and generated data to see if they struggle with overfitting like we have seen with LSTM-VAE or big gaps between real and generated scenarios like in the case of CNN.

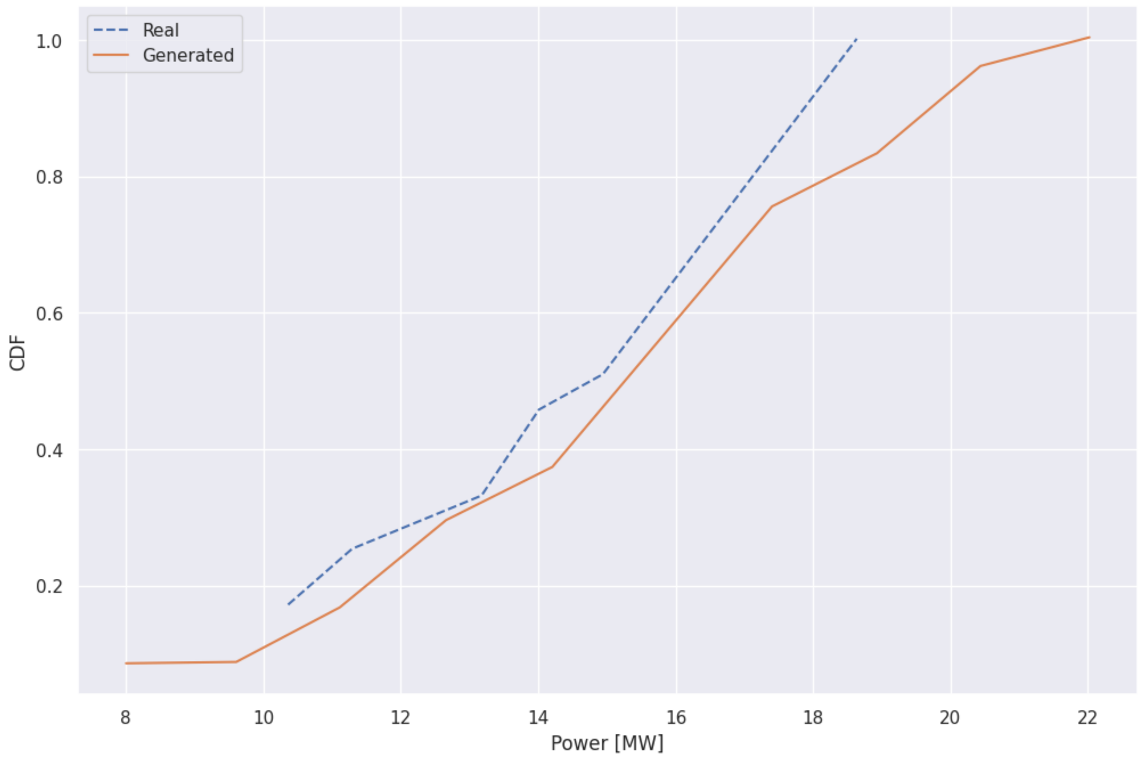


Figure 4-88: CDF of Real and Generated Load Scenarios

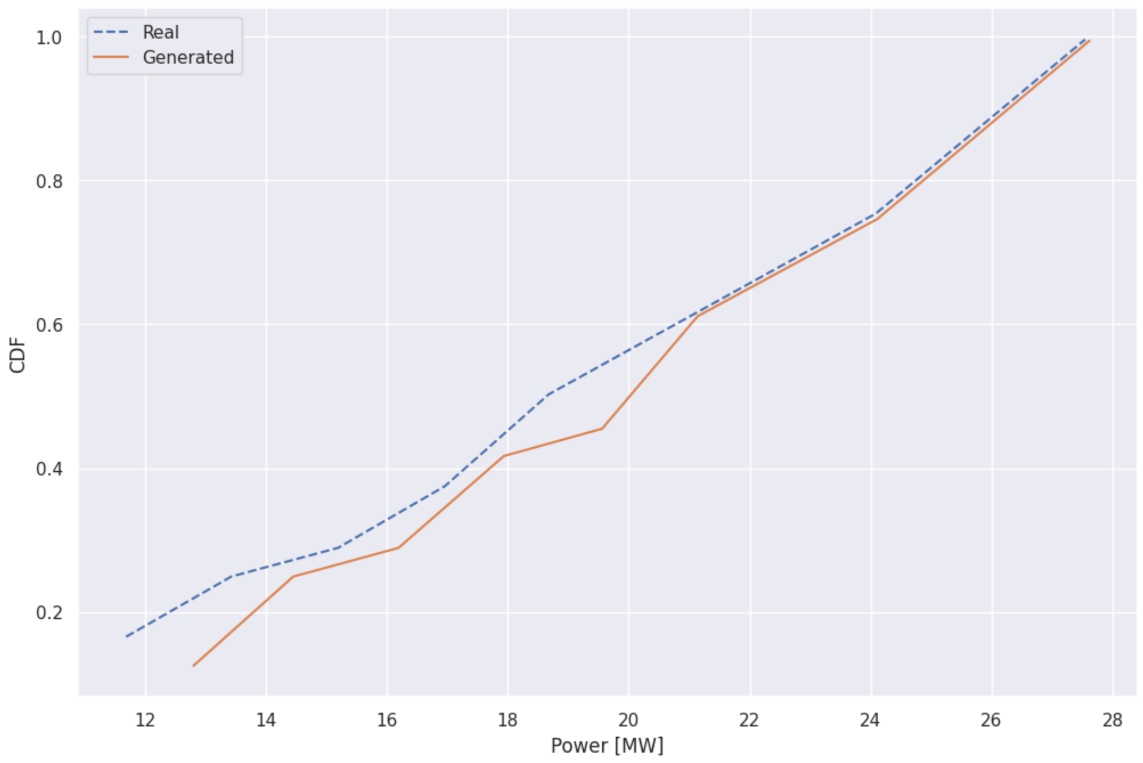


Figure 4-89: CDF of Real and Generated Wind Scenarios

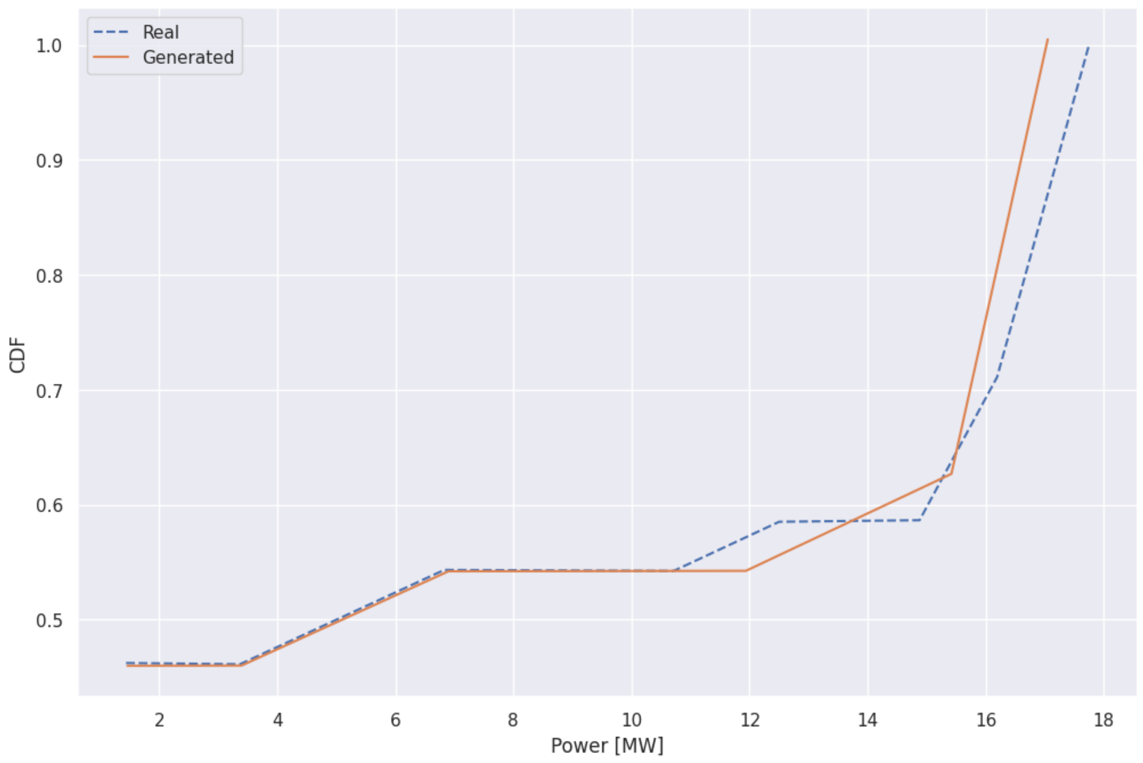


Figure 4-90: CDF of Real and Generated Solar Scenarios

It is obvious from the CDFs that there are differences between real and generated scenarios as they are not perfectly on top of each other like the results of the LSTM-VAE and there are differences between the samples. Hence, the model is able to generate representative scenarios.

4.8.5 Error Metrics:

The error results obtained below show that the algorithm has higher MAE and RMSE for load and wind when compared to solar. It also shows higher errors when compared to LSTM-VAE but lower results than CNN.

Table 4-5: MAE and RMSE Error for STGCN

	Load	Solar	Wind
MAE	1.329	0.242	2.623
RMSE	10.145	0.066	13.771

4.9 Discussion

It is obvious from the results that for generating distinct scenarios, C-GANs are the best candidate. This is simply because the initial starting point or the input to the model is a random vector z (sampled from a normal distribution), whereas all the other algorithms need an input time series. We believe that this allows the model to generate very distinct scenarios that have not been seen by the model before. While solar farms generally have higher power production in summer compared to other seasons, there are still days where those solar farms generate little to no power because of cloudy skies. The C-GANs are conditions on seasons only. Hence, the algorithm has a bigger sample pool from which results can be generated. The other algorithms are conditioned on an input, so the generated scenarios are also conditioned on that same input, which is why we do not see very distinct and extreme scenarios. The problem with C-GANs arises when we try to scale it up to more variables and conditions. The model fails strikingly as seen by the earlier results in Chapter 4.7. Hence, scaling up of the algorithm for use in electrical networks is extremely challenging. This is primarily due to the adversarial nature of the algorithm as explained in Chapter 3.4 which makes it difficult for the generator to learn the different distributions of the data and fool the discriminator.

LSTM-VAE generates results that are very close to the real scenario and this is due to the architecture of LSTM cells, their ability to learn temporal behavior, and the absence of spatial information. They are able to learn seasonal and daily behaviors and mimic them with little variations. They are perfect when it comes to multivariate and multistep forecasting as proved by the ACF, CDF, and the error metrics in Chapter 4.6; the results are near perfect for forecasting purposes.

STGCN and CNNs are better suited for scenario generation problems. They both generate representative scenarios accurately. However, CNNs lack the ability to capture certain intricate details about the different variables such as the ability to capture large dips or not allowing solar generation to go below zero. CNNs also do not incorporate the physics of the power network and the graphical relations that exist. While filters do account for spatial relations, they are not necessarily true relations (from a power systems perspective) as CNNs account for spatial relations based on proximity, and not connections due to the usage of filters (Chapter 4.5).

In terms of scalability, theoretically STGCN is the best candidate since it does not have the adversarial nature of C-GANs, and it is not limited to the rigid structure of filters in CNNs that account for special relations. The connections are also fluid since not all buses are connected to each other. Lastly, while CNNs are easier to implement compared to STGCN, CNNs require more time to train due to the large number of epochs and that can be a huge hurdle, refer Table 4-1.

CHAPTER 5

CONCLUSION AND FUTURE DIRECTIONS

The goal of this thesis was to address multivariate, multistep scenario generation and forecasting for solar, wind, and load through machine learning algorithms while accounting for the intercorrelations. State-of-the-art algorithms such as CNN, LSTM-VAE, C-GAN and STGCN were utilized to solve this problem. The IEEE-30 bus system was used for this analysis and the conventional generators were replaced by solar and wind farms. Each of the algorithms had their own pros and cons and all of those were detailed in the thesis.

C-GANs generate distinct scenarios but are very difficult to train due to the opposing objectives of the generator and discriminator networks. The model works perfectly when tested on smaller number of variables with distinct distributions. Three variables (solar, wind, and load) were initially used to train model. However, extending the model to a real system such as the IEEE-30 bus system was not possible due to the larger number of conditions for each of the variables and the overlap of the distributions of the different variables.

LSTM-VAE model was able to address the forecasting problem and generate scenarios with very little variations. The resulting scenarios were astoundingly similar to the real scenario. The correlation results and the error metrics both confirmed this observation. This makes LSTM-VAE perfect for multivariate and multistep forecasting.

Both STGCN and CNNs were able to generate representative scenarios for all the variables. However, STGCN model was able to capture very detailed information about the variables that were missed by the (simpler) CNN model.

This work lays the foundation for a hybrid framework of STGCN and LSTM-VAE that can be utilized in the day-to-day operation of the electrical network and for day-ahead planning. While an hourly reading of load and generation were used in this thesis, a smaller time scale can also be used to capture smaller variations. Moreover, a different time horizon (other than 24 hours) can also be used. The readings for load demand and generation can be set as inputs to both the models with LSTM-VAE generating accurate forecasts that can be used to allocate resources accordingly, while STGCN can generate different scenarios to test the reliability of the system. Particularly, the scenarios generated in this work can also be used for battery energy storage systems (BESS) allocation [38].

5.1 Future Directions

The work presented in this thesis can be extended in different ways:

- Transductive learning can be used in GCNN. Transductive learning in GCNN provides the ability to add nodes or change the structure of the graph completely. This opens up a new research frontier to solve problems of optimal placement of wind or solar farms based on historical data and the spatio-temporal relationships. Designers can idnetify new locations for wind and solar farms, connect them to the existing network, and run GCNN to forecast and generate scenarios to see if it is viable to place a wind or solar farm at those locations.

- Another future research direction is to apply the machine learning models at the distribution level and include other variables with unique characteristics (such as electric vehicles [39]) to see how these models perform.
- The addition of clustering techniques can help improve the results by focusing on generating unique scenarios based on those clusters. Hence, incorporating a clustering model as a precursor to the models discussed in this thesis can be explored in the future.

REFERENCES

[1] "Global Energy Review 2021," IEA, 2021. [Online]. Available: https://www.iea.org/reports/global-energy-review-2021/renewables.

[2] R. S. Biswas, A. Pal, T. Werho, and V. Vittal, "Mitigation of saturated cut-sets during multiple outages to enhance power system security," accepted for publication in IEEE Transactions on Power Systems.

[3] R. S. Biswas, A. Pal, T. Werho, and V. Vittal, "A graph theoretic approach to power system vulnerability identification," IEEE Trans. Power Syst., vol. 36, no. 2, pp. 923-935, Mar. 2021.

[4] T. Wang, J. Yang, M. Padhee, A. Pal, J. Bi, and Z. Wang, "Robust coordinated control of sub-synchronous oscillation in wind integrated power system," IET Renewable Power Gener., vol. 14, no. 6, pp. 1031-1043, Apr. 2020.

[5] C. Mishra, R. S. Biswas, A. Pal, and V. A. Centeno, "Critical clearing time sensitivity for inequality constrained systems," IEEE Trans. Power Syst., vol. 35, no. 2, pp. 1572-1583, Mar. 2020.

[6] C. Mishra, A. Pal, J. S. Thorp, and V. A. Centeno, "Transient stability assessment of prone-to-trip renewable generation rich power systems using Lyapunov's direct method," IEEE Trans. Sustainable Energy, vol. 10, no. 3, pp. 1523-1533, Jul. 2019.

[7] H. Albhrani, R. S. Biswas, and A. Pal, "Identification of utility-scale renewable energy penetration threshold in a dynamic setting," in Proc. IEEE North American Power Symposium (NAPS), Tempe, AZ, pp. 1-6, 11-13 Apr. 2021.

[8] C. Wang, C. Mishra, R. S. Biswas, A. Pal, and V. A. Centeno, "Adaptive LVRT settings adjustment for enhancing voltage security of renewable-rich electric grids," in Proc. IEEE Power Eng. Soc. General Meeting, Montreal, Canada, pp. 1-5, 2-6 Aug. 2020.

[9] A. Nath, R. S. Biswas, and A. Pal, "Application of machine learning for online dynamic security assessment in presence of system variability and additive

instrumentation errors," in Proc. IEEE North American Power Symposium (NAPS), Wichita, KS, pp. 1-6, 13-15 Oct. 2019.

[10] T. Wang, J. Yang, J. Liu, P. Gupta, A. Pal, and J. Deng, "SDAE-based probabilistic stability analysis of wind integrated power systems," in Proc. 2nd IEEE Conf. Energy Internet and Energy System Integration (EI2), Beijing, China, pp. 1-6, 20-22 Oct. 2018.

[11] M. Padhee, and A. Pal, "Effect of solar PV penetration on residential energy consumption pattern," in Proc. IEEE North American Power Symposium (NAPS), Fargo, ND, pp. 1-6, 9-11 Sep. 2018.

[12] T. Chakraborty, and A. Pal, "Controller tuning of generic positive sequence solar PV models used in interconnection studies," in Proc. IEEE North American Power Symposium (NAPS), Morgantown, WV, pp. 1-6, 17-19 Sep. 2017.

[13] M. Padhee, A. Pal, and K. A. Vance, "Analyzing effects of seasonal variations in wind generation and load on voltage profiles," in Proc. IEEE North American Power Symposium (NAPS), Morgantown, WV, pp. 1-6, 17-19 Sep. 2017.

[14] S. Wang, P. Dehghanian, M. Alhazmi, J. Su and B. Shinde, "Resilience-Assured Protective Control of DC/AC Inverters Under Unbalanced and Fault Scenarios," 2019 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT), 2019, pp. 1-5, doi: 10.1109/ISGT.2019.8791659.

[15] S. Impram, S. Varbak Nese and B. Oral, "hallenges of renewable energy penetration on power system flexibility: A survey," Energy Strategy Reviews, 2020.

[16] D. A. Snegirev, S. A. Eroshenko, R. T. Valiev and A. I. Khalyasmaa, "Algorithmic realization of short-term solar power plant output forecasting," 2017 IEEE II International Conference on Control in Technical Systems (CTS), 2017, pp. 228-231.

[17] Y. -J. Zhong and Y. -K. Wu, "Short-Term Solar Power Forecasts Considering Various Weather Variables," 2020 International Symposium on Computer, Consumer and Control (IS3C), 2020, pp. 432-435.

[18] F. Serttas, F. O. Hocaoglu and E. Akarslan, "Short Term Solar Power Generation Forecasting: A Novel Approach," 2018 International Conference on Photovoltaic Science and Technologies (PVCon), 2018, pp. 1-4.

[19] V. Kushwaha and N. M. Pindoriya, "Very short-term solar PV generation forecast using SARIMA model: A case study," 2017 7th International Conference on Power Systems (ICPS), 2017, pp. 430-435.

[20] S. M. Verma, V. Reddy, K. Verma and R. Kumar, "Markov Models Based Short Term Forecasting of Wind Speed for Estimating Day-Ahead Wind Power," 2018 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), 2018, pp. 31-35.

[21] X. Zheng, X. Qi, H. Liu, X. Liu and Y. Li, "Deep Neural Network for Short-Term Offshore Wind Power Forecasting," 2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO), 2018, pp. 1-5.

[22] Y. Zhongping et al., "Integrated wind and solar power forecasting in China," Proceedings of 2013 IEEE International Conference on Service Operations and Logistics, and Informatics, 2013, pp. 500-505.

[23] S. Makhloufi, M. Debbache and S. Boulahchiche, "Long-term Forecasting of Intermittent Wind and Photovoltaic Resources by using Adaptive Neuro Fuzzy Inference System (ANFIS)," 2018 International Conference on Wind Energy and Applications in Algeria (ICWEAA), 2018, pp.

[24] Y. Chen, Y. Wang, D. Kirschen and B. Zhang, "Model-Free Renewable Scenario Generation Using Generative Adversarial Networks," in IEEE Transactions on Power Systems, vol. 33, no. 3, pp. 3265-3275.

[25] M. Yang, W. Liu, X. Yin, Z. Cui and W. Zhang, "A Two-Stage Scenario Generation Method for Wind- Solar Joint Power Output Considering Temporal and

Spatial Correlations," 2021 6th Asia Conference on Power and Electrical Engineering (ACPEE), 2021, pp. 415-423.

[26] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1-6.

[27] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[28] J. Schmidhuber, 2017. [Online]. Available: https://people.idsia.ch/~juergen/impact-on-most-valuable-companies.html. [Accessed 2021].

[29] F. Beaufays, "The neural networks behind Google Voice transcription," 2015. [Online]. Available: https://ai.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html. [Accessed 2021].

[30] C. Olah, "Colah's Blog," 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 2021].

[31] B. Yu, H. Yin and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," arXiv preprint arXiv:1709.04875, 2017.

[32] "NREL," [Online]. Available: https://www.nrel.gov/grid/data-tools.html. [Accessed 2021].

[33] A. Birchfield, "ELECTRIC GRID TEST CASE REPOSITORY," [Online]. Available: https://electricgrids.engr.tamu.edu/electric-grid-test-cases/activsg2000/. [Accessed 2021].

[34] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller., Efficient BackProp, Springer, 1998.

[35] M. Karakaya, "Medium," 2020. [Online]. Available: https://medium.com/deep-learning-with-keras/which-activation-loss-functions-in-multi-class-clasification-4cd599e4e61f. [Accessed 2021].

[36] J. Gui, S. Zhenan, W. Yonggang, D. Tao and J. Ye, "A review on generative adversarial networks: Algorithms, theory, and applications," arXiv preprint arXiv:2001.06937 , 2020.

[37] I. Gulrajani, A. Faruk, A. Martin, D. Vincent and C. Aaron, "Improved training of wasserstein gans," arXiv preprint arXiv:1704.00028, 2017.

[38] M. Padhee, A. Pal, C. Mishra and K. A. Vance, "A Fixed-Flexible BESS Allocation Scheme for Transmission Networks Considering Uncertainties," in IEEE Transactions on Sustainable Energy, vol. 11, no. 3, pp. 1883-1897.

[39] M. Z. Zeb et al., "Optimal Placement of Electric Vehicle Charging Stations in the Active Distribution Network," in IEEE Access, vol. 8, pp. 68124-68134, 2020.

APPENDIX A

GITHUB REPO

All the codes can be found at:
https://github.com/Anamitra-Pal-Lab/Scenario-Generation-and-Forecasting

APPENDIX B

CNN PSEUDO CODE

```
# define model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu',
input_shape=(n_steps_in, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(n_output))
model.compile(optimizer='adam', loss='mse')
```

APPENDIX C

LSTM-VAE PSEUDO CODE

```
class lstm_encoder(nn.Module):
    ''' Encodes time-series sequence '''

    def __init__(self, input_size, hidden_size, num_layers = 2):

        '''
        : param input_size:     the number of features in the input
X
        : param hidden_size:    the number of features in the hidden
state h
        : param num_layers:     number of recurrent layers (i.e., 2
means there are
        :                       2 stacked LSTMs)
        '''

        super(lstm_encoder, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        # define LSTM layer
        self.lstm = nn.LSTM(input_size = input_size, hidden_size =
hidden_size,
                            num_layers = num_layers)

    def forward(self, x_input):

        '''
        : param x_input:               input of shape (seq_len, # in
batch, input_size)
        : return lstm_out, hidden:     lstm_out gives all the hidden
states in the sequence;
        :                              hidden gives the hidden state
and cell state for the last
        :                              element in the sequence
        '''

        lstm_out, self.hidden =
self.lstm(x_input.view(x_input.shape[0], x_input.shape[1],
self.input_size))

        return lstm_out, self.hidden

    def init_hidden(self, batch_size):
        '''
        initialize hidden state
        : param batch_size:    x_input.shape[1]
        : return:              zeroed hidden state and cell state
        '''

        return (torch.zeros(self.num_layers, batch_size,
self.hidden_size),
                torch.zeros(self.num_layers, batch_size,
self.hidden_size))


class lstm_decoder(nn.Module):
    ''' Decodes hidden state output by encoder '''

    def __init__(self, input_size, hidden_size, num_layers = 2):

        '''
        : param input_size:     the number of features in the input
X
        : param hidden_size:    the number of features in the hidden
state h
        : param num_layers:     number of recurrent layers (i.e., 2
means there are
        :                       2 stacked LSTMs)
        '''

        super(lstm_decoder, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        self.lstm = nn.LSTM(input_size = input_size, hidden_size =
hidden_size,
                            num_layers = num_layers)
        self.linear = nn.Linear(hidden_size, input_size)

    def forward(self, x_input, encoder_hidden_states):

        '''
        : param x_input:                    should be 2D
(batch_size, input_size)
        : param encoder_hidden_states:      hidden states
        : return output, hidden:            output gives all the
hidden states in the sequence;
        :                                   hidden gives the hidden
state and cell state for the last
        :                                   element in the sequence

        '''

        lstm_out, self.hidden = self.lstm(x_input.unsqueeze(0),
encoder_hidden_states)

        output = self.linear(lstm_out.squeeze(0))
        return output, self.hidden

class lstm_seq2seq(nn.Module):
    ''' train LSTM encoder-decoder and make predictions '''

    def __init__(self, input_size, hidden_size):

        '''
        : param input_size:     the number of expected features in
the input X
        : param hidden_size:    the number of features in the hidden
state h
        '''

        super(lstm_seq2seq, self).__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size

        self.encoder = lstm_encoder(input_size = input_size,
hidden_size = hidden_size)
        self.decoder = lstm_decoder(input_size = input_size,
hidden_size = hidden_size)
```

APPENDIX D

C-GANS PSEUDO CODE

```
class Generator(nn.Module):
    def __init__(self, channels_noise=100, channels_img=1,
features_g=64, label=3,batch_size = 32):
        super(Generator, self).__init__()
        self.label = label
        self.batch_size = batch_size
        self.fc1 = nn.Linear(channels_noise+label,features_g*16)
        self.fc2 = nn.Linear(features_g*16+label, features_g*72)
        self.deconv1 = nn.ConvTranspose2d(features_g*2+label ,
features_g, kernel_size=2, stride=2, padding=0)
        self.deconv2 = nn.ConvTranspose2d(features_g+label ,
channels_img, kernel_size=2, stride=2, padding=0)
        self.activation = nn.ReLU()

    def forward(self, z, y):
        #print("Initializing the generator")
        #print("Input Z shape", z.shape)
        #print("Input Y shape", y.shape)
        yb =y.unsqueeze(2).unsqueeze(2).to(device)
        z = torch.cat([z, y],1)
        #print("Input Z shape", z.shape)
        out = self.activation(batchnormalize(self.fc1(z)))
        #print("h1 shape", out.shape)
        out = torch.cat([out, y],1)
        #print("h1 shape", out.shape)
        out = self.activation(batchnormalize(self.fc2(out)))
        #print("h2 shape", out.shape)
        out =  out.view([-1,128,6,6])
        #print("h2 shape", out.shape)
        temp12 = (yb.to(device)*torch.ones([self.batch_size,
self.label,6, 6]).to(device))
        out = torch.cat([out, temp12],1)
        n=yb*torch.ones([self.batch_size, self.label,6,
6]).to(device)
        #print("shape of yb new",n.shape )
        #print("h2 shape", out.shape)
        out = self.activation(batchnormalize(self.deconv1(out)))
        #print("h3 shape", out.shape)
        temp22 = yb*torch.ones([self.batch_size, self.label, 12,
12]).to(device)
        out = torch.cat([out, temp22], 1)
        #print("h3 shape", out.shape)
        out = self.deconv2(out)
        #print("h4 shape", out.shape)
```

```
out = torch.sigmoid(out)
return out
```

APPENDIX E

STGCN PSEUDO CODE

```
class STGCN(nn.Module):
    """
    Spatio-temporal graph convolutional network
    Input has shape (batch_size, num_nodes, num_input_time_steps,
    num_features).
    """

    def __init__(self, num_nodes, num_features, num_timesteps_input,
                 num_timesteps_output):
        """
        :param num_nodes: Number of nodes in the graph.
        :param num_features: Number of features at each node in each
time step.
        :param num_timesteps_input: Number of past time steps fed
into the
        network.
        :param num_timesteps_output: Desired number of future time
steps
        output by the network.
        """
        super(STGCN, self).__init__()
        self.block1 = STGCNBlock(in_channels=num_features,
out_channels=64,
                                 spatial_channels=16,
num_nodes=num_nodes)
        self.block2 = STGCNBlock(in_channels=64, out_channels=64,
                                 spatial_channels=16,
num_nodes=num_nodes)
        self.block3 = STGCNBlock(in_channels=64, out_channels=64,
                                 spatial_channels=16,
num_nodes=num_nodes)
        self.last_temporal = TimeBlock(in_channels=64,
out_channels=64)
        self.fully = nn.Linear((num_timesteps_input - 20) * 64,
                               num_timesteps_output)
```