Classification of Cyber Attacks in Smart Grids using Deep Neural Networks

Atharva Swami
Computer Science & Engineering
Indian Institute of Technology, Dharwad
Karnataka, India
atharvaswami@gmail.com

***Abstract*—The current research in smart grids is depending on the availability of time series data. The available data-sets are not publicly accessible due to security and privacy issues. This has been preventing the research community from effectively apply the machine learning algorithms to improve the stability of the power system. Therefore, the synthetic data generation has been a promising solution for addressing the issues of power system applying different algorithms. Most of the recent works has focused the system dynamics to generate the data-sets. Furthermore, with the integration of cyber space and physical space in the smart grid, it is easy for an adversary to affect the normal operation of the primary equipment by attacking cyberspace. Therefore, timely response to attacks is especially critical for the stable operation of the smart grid. In this work, we compared the performance of different neural network models to classify existing data into different types of attacks for detection purpose. Moreover, we proposed an approach to generate synthetic data using a deep Generative Adversarial Network (GAN) model. We choose the parameters of the neural networks in a way to optimize the models in the best possible way. After evaluating the models using our approach, the results show that we can successfully classify cyber-attacks with an accuracy of 90% using the synthetic data that we generated, which is converging to the original distribution.**

***Index Terms*—IoT:Internet of Things, GAN:Generative Adversarial Network, SVM:Support Vector Machine**

I. Introduction

A smart grid is an electrical grid network that is used for power transmission that provides reliable, efficient electricity transmission and distribution with Internet of Things (IoT) technology. IoT is a large network where the information is sensed through the smart devices such as phasor measurement units (PMUs) and smart meters. The information is processed by the control units for some applications like state estimation and event management.

Though the smart grid is robust and reliable using the smart monitoring devices, still the existing techniques are not sufficient to detect the cyber-attacks. The attackers are able to access the smart devices and manipulate the information to create false events in the power system such as power blackouts, generator outage, and line outage. Therefore, timely response to attacks is especially critical for the stable operation of the smart grid. Not only detection of the cyber-attacks will counter these attacks but we also need to classify the attacks so that the necessary types of measures can be taken to counter them. Due to the wide variety of cyber-attacks, it is difficult for the control units to respond uniformly and quickly to all attacks. It is particularly important to classify attacks since only when the attack is identified, the accurate response to the attack can be achieved.

II. Smart Grid Data-set:

The provided data-set contains 37 total scenarios, which include 8 Natural-events, 1 No-event, and 28 Attack-events. Each event comprises thousands of monitoring observations captured by 4 PMUs working at 120 samples, one control panel logging system, and a Snort intrusion detection system (IDS). Any PMU provides 29 status variables, thus the set of all PMUs provides $116(29 \times 4)$ variables that become the features that directly describe the power system behavior, while the Snort IDS and the logs provide other 12 additional elements to be used for spotting anomalous events in the overall power control system. This, results in 128 basic elements describing the system activity, to be used for detecting and recognizing attacks, plus the class (or supervisory signal) consisting of a label reporting the involved attack type or the absence of anomalies (normal activity). These basic elements will be the starting point for mining the more meaningful space or time-related features that will be useful in detecting attacks.
Types of Scenarios:

1) Short-circuit fault – this is a short in a power line and can occur in various locations along the line, the location is indicated by the percentage range.
2) Line maintenance –one or more relays are disabled on a specific line to do maintenance for that line.
3) Remote tripping command injection (Attack) – this is an attack that sends a command to a relay that causes a breaker to open. It can only be done once an attacker has penetrated outside defenses.
4) Relay setting change (Attack) – relays are configured with a distance protection scheme and the attacker changes the setting to disable the relay function such that the relay will not trip for a valid fault or a valid command.
5) Data Injection (Attack) – here we imitate a valid fault by changing values to parameters such as current, voltage, sequence components, etc. This attack aims to blind the operator and causes a blackout.

We divided the above 5 main types of cyber-attacks into 6 categories as described below according to the given scenarios.

1) Data Injection
    a) SLG fault replay - attack0
2) Remote Tripping Command Injection
    a) Command injection against single relay - attack1
    b) Command injection against multiple relays - attack2
3) Relay Setting Change
    a) Disabling relay function - single relay disabled & fault - attack3
    b) Disabling relay function - two relays disabled & fault - attack4
    c) Disabling relay function - two relay disabled & line maintenance - attack5

Attack	Scenarios
attack0	7, 8, 9 , 10, 11, 12
attack1	15, 16, 17, 18
attack2	19, 20
attack3	21, 22, 23, 24, 25, 26, 27, 28, 29, 30
attack4	35, 36, 37, 38
attack5	39, 4

So, we filtered out the attack events from the main data-set and added a new column classifying the attacks from 0 to 5. Now we have 129 columns in the data-set out of which 128 are the features and the last column is the classifier. Then we performed data cleaning on the above data-set by removing the infinity and NULL values. After that, we divided the data-set into training and testing data which contained similar percentages of all events. To get better results we then scaled down the data. This makes it easier for the deep learning models to generate or classify the data.

III. Classification Models:

We designed many deep neural network models to classify the above test data-set into the 6 types of cyber attacks and check their accuracy. The details of each model are given below. Finally, we use the model with the greatest accuracy for classification.

A. Logistic Regression

Logistic regression is a statistical analysis method used to predict a data value based on prior observations of a data set. It predicts a dependent data variable by analyzing the relationship between one or more existing independent variables. In this model, we got a score of 43.07%.

B. Support Vector Machine (SVM)

The SVMs are primarily used for classification in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. In this model, we got stuck into an infinite loop while training the model with the complete data-set. This happened because we were using too many training examples for our SVM implementation. SVMs are based around a kernel function. Most implementations explicitly store this as an NxN matrix of distances between the training points to avoid computing entries over and over again. So, then we used a small data-set of size equal to the number of features(i.e. 128) and got a score of 26.72%

C. Artificial Neural Network (ANN)

ANN considers classification as one of the most dynamic research and application areas. The major disadvantage in using ANN is to find the most appropriate grouping of training, learning, and transfer functions for classifying the data sets with a growing number of features and classified sets. In this model, we used a 2 layered neural network containing 128 neurons in the middle layer and 6 neurons in the output layer(as there are 6 attack classes). The ReLU activation function is used for the input and the middle layer whereas the sigmoid activation function is used in the output layer for classification. Then the model was compiled using a stochastic gradient descent optimizer with sparse categorical cross-entropy loss function and accuracy metrics. Finally, training the model for 1000 epochs we got a test accuracy of 67.57%. This gave a much better score than the previous machine learning models.

D. Convolutional Neural Network (CNN)

CNN is a class of artificial neural networks, most commonly applied to classify images. The flatten layer in CNN is used to make the multidimensional input one-dimensional, commonly used in the transition from the convolution layer to the fully connected layer. The flattened output is fed as the input to the fully connected neural network having varying numbers of hidden layers to learn the non-linear complexities present with the feature representation. Therefore to use CNN for the classification of the attacks, we reshaped the 1-dimensional features array (size=128) into a 3-dimensional features array(size=(8,16,1)). This conversion was done because images are generally used in 3-dimensional array form(length, breadth, and color) while classification using CNN. A Conv2D layer was used as the input layer with 32 filters and a 3X3 kernel. The ReLU activation function was used in the input layer and the middle layer whereas the Softmax activation function was used in the output layer. Even in this model, we used 128 neurons in the middle layer and 6 neurons in the output layer(as there are 6 attack classes). The data was max pooled and flattened before passing to the dense hidden layer. We used the MaxPooling2D layer to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network. The model was compiled using the adam optimizer with sparse categorical cross-entropy loss function and accuracy metrics. Finally, training the model for 1000 epochs we got a test accuracy of 72.26% which is even better than the ANN model test accuracy.

E. CNN-RNN Variant of the Autoencoder Model

We propose an approach using a complex deep neural network (DNN) framework able to uncover meaningful information from the power system monitoring data regarding both relations among the different observations and their correlations over time, represented as spatial and temporal dependencies, constituting more expressive and representative features to be used for attack detection. This is accomplished through the usage of two sparse autoencoders (SAEs) whose encoder-decoder functions are replaced with two DNNs, respectively a convolutional and a recurrent one, providing the ability to capture the preceding correlations between their inputs.

Although the convolutional and recurrent networks have proven to be effective also without being used in combination with the autoencoders, their performance tends to decrease if they are trained on an unbalanced data set. Furthermore, for a high number of hidden layers, such networks are affected by the well-known problem of the vanishing gradient. LSTM and gated recurrent unit (GRU) are powerful alternatives capable to overcome this issue. [2] A single LSTM unit includes a cell of memory maintaining information for sufficiently time intervals. A set of gates are used to control the information flow and to make decisions about what data should be put into memory, and when they need to be forgotten. GRUs are similar to LSTMs but they use a simplified structure comprising fewer gates to control the flow of information. On the contrary, the stacked autoencoder-based deep network solves many issues related to the training of networks enclosing a high number of layers and offers a powerful tool that can be used also in different fields of application.

In Deep Learning, CNNs have proven to be the best candidate in extracting spatial insights from data, especially in computer vision and image processing fields. The usage of several stacked convolutional layers in CNNs, used to create a hierarchy of progressively more abstract representations of the input data, constitutes their main strength for mining spatial dependencies. Similarly, RNNs are the most suitable neural network typology for modeling temporal dependencies in time-series data.

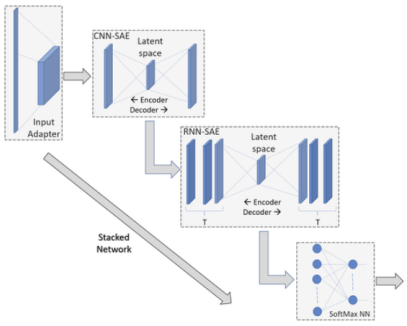

Fig. 1. CNN-RNN variant of the Autoencoder Model

To have time-stamps or to connect the data, we use the LSTM model. In this model, we have also used CNN so that we can extract the compressed features from the flattened layer. The main use of the CNN model is to do feature engineering and find the special features of the dataset. Take those extracted features and use them as the input for the LSTM model. As this is categorical data we need to use the softmax activation function for classifying the outputs. Now to use the LSTM model, we not only need the extracted features but we also need the timestamps.

Also due to the unavailability of time-series data, we modified the current attack data in such a way that can be used as the input to the LSTM model. Sort the whole dataset according to the attack type. Now choose each consecutive 10 rows of the data and assume to be of the same time frame(same day). Then convert both the X_training and X_testing data into 3-dimensional data (days, entries-per-day, no-of-features). Now get the y_training and y_testing data from the original dataset and convert it into categorical data. Design the decoder part using 2 LSTM layers and a softmax activation function. In the input layer set the return_sequences=true to send 10 timestamps each at a time. Put the learning rate as e^{-5} to avoid getting stuck at local minima. Then use the adam optimizer to compile the model. Finally, after modifying the above dataset and testing it on this model, we get a score of 89%.

IV. Attacker Model:

A. Generative Adversarial Networks:

For research in smart grids, there are not many time-series datasets available due to issues related to data collection, security, and privacy. So initially this prevented us from effectively applying deep learning algorithms to significantly improve the distribution-level accuracy of predictions and protect the grids from cyber attacks. Thus we decided to come up with generating synthetic data for addressing the above issues. But doing so using by simulating the underlying network and system dynamics is difficult, time-consuming, error-prone, and oftentimes infeasible in many problems. In this work, we propose an approach to generate a synthetic dataset by utilizing deep generative adversarial networks (GAN) to learn the conditional probability distribution of essential features in the real dataset and generate samples based on the learned distribution. [3]

GANs are a deep learning architecture for training powerful generator models. A generator model is capable of generating new artificial samples that plausibly could have come from an existing distribution of samples. GANs are comprised of both generator and discriminator models. The generator is responsible for generating new samples from the domain, and the discriminator is responsible for classifying whether samples are real or fake (generated). Importantly, the performance of the discriminator model is used to update both the model weights of the discriminator itself and the generator model. This means that the generator never actually sees examples from the domain and is adapted based on how well the discriminator performs. An intuitive explanation of the objective function is that the generator is trying to produce fake samples

while the discriminator is trying to detect the counterfeits. Competition in this game drives both the generator and the discriminator to improve their methods until the fake samples are indistinguishable from the real data.

B. Define a Discriminator Model

The first step is to select a sample dataset for the model. We used the same attack dataset here which we used earlier to train different deep learning models to classify the cyber attacks. The next step is to define the discriminator model. The model must take a sample from our problem, such as a vector with 129 elements, and output a classification prediction as to whether the sample is real or fake. This is a binary classification problem.

- Inputs: A vector with size 129 containing all the 128 features and the corresponding attack type.
- Outputs: Binary classification, whether the sample is real (or fake).

The discriminator model will have one hidden layer with 25 nodes and we will use the ReLU activation function and an appropriate weight initialization method called He weight initialization. The output layer will have one node for the binary classification using the sigmoid activation function. The model will minimize the binary cross-entropy loss function, and the Adam version of stochastic gradient descent will be used because it is very effective.

We could start training this model now with real examples with a class label of one and randomly generated samples in the range -1 and 1 for all the elements of a sample with a class label of zero(this function will act as our fake generator model initially). Next, we create a function to train and evaluate the discriminator model. This is achieved by manually enumerating the training epochs and for each epoch generating a half batch of real examples and a half batch of fake examples, and updating the model on each, e.g. one whole batch of examples. The model can then be evaluated on the generated examples and we can report the classification accuracy on the real and fake samples. We train the model for 1,000 batches and use 128 samples per batch (64 fake and 64 real).

Running the example generates real and fake examples and updates the model, then evaluates the model on the same examples and prints the classification accuracy. In this case, the model rapidly learns to correctly identify the real examples with perfect accuracy and is very good at identifying the fake examples with around 99% accuracy. Training the discriminator model is straightforward. The goal is to train a generator model, not a discriminator model, and that is where the complexity of GANs truly lies.

C. Define a Generator Model

The next step is to define the generator model. The generator model takes as input a point from the latent space and generates a new sample, e.g. a vector with size 129 containing all the 128 features and the corresponding attack type. A latent variable is a hidden or unobserved variable, and a latent space is a multi-dimensional vector space of these variables. We can define the size of the latent space for our problem and the shape or distribution of variables in the latent space. This is because the latent space has no meaning until the generator model starts assigning meaning to points in the space as it learns. After training, points in the latent space will correspond to points in the output space, e.g. in the space of generated samples.

We will define a latent space of 50 dimensions and use the standard approach in the GAN literature of using a Gaussian distribution for each variable in the latent space. We will generate new inputs by drawing random numbers from a standard Gaussian distribution, i.e. mean of zero and a standard deviation of one.

- Inputs: Point in latent space, e.g. a 50-element vector of Gaussian random numbers.
- Outputs: A vector with size 129 containing all the 128 features and the corresponding attack type.

The generator model will be small like the discriminator model. It will have a single hidden layer with five nodes and will use the ReLU activation function and the He weight initialization. The output layer will have 129 nodes for the 129 elements in a generated vector and will use a linear activation function. A linear activation function is used because we know we want the generator to output a vector of real values and the scale will be [-1, 1] for all the elements as we have normalized the data while using the discriminator.

The first step is to generate new points in the latent space. We can achieve this by calling the randn() NumPy function for generating arrays of random numbers drawn from a standard Gaussian. The array of random numbers can then be reshaped into samples: that is n rows with 50 elements per row. Then create a function to implement this and generate the desired number of points in the latent space that can be used as input to the generator model. Next, we can use the generated points as input in the generator model to generate new samples. Running the example generates random points from the latent space, uses this as input to the generator, and generates fake samples from our 2-dimensional data domain. As the generator has not been trained, the generated points are complete rubbish, as we expect, but we can imagine that as the model is trained, these points will slowly begin to resemble the real data. Later we will need to use the generator model in this way to create samples for the discriminator to classify.

D. Training the Generator Model

The weights in the generator model are updated based on the performance of the discriminator model. When the discriminator is good at detecting fake samples, the generator is updated more, and when the discriminator model is relatively poor or confused when detecting fake samples, the generator model is updated less. This defines the zero-sum or adversarial relationship between these two models. There may be many ways to implement this using the Keras API, but perhaps the simplest approach is to create a new model that subsumes or encapsulates the generator and discriminator models.

Specifically, a new GAN model can be defined that stacks the generator and discriminator such that the generator receives as input random points in the latent space, generate samples that are fed into the discriminator model directly, classified, and the output of this larger model can be used to update the model weights of the generator. To be clear, we are not talking about a new third model, just a logical third model that uses the already-defined layers and weights from the standalone generator and discriminator models.
Only the discriminator is concerned with distinguishing between real and fake examples; therefore, the discriminator model can be trained in a standalone manner on examples of each. The generator model is only concerned with the discriminator's performance on fake examples. Therefore, we will mark all of the layers in the discriminator as not trainable when it is part of the GAN model so that they can not be updated and overtrained on fake examples.
When training the generator via this subsumed GAN model, there is one more important change. We want the discriminator to think that the sample's output by the generator is real, not fake. Therefore, when the generator is trained as part of the GAN model, we will mark the generated samples as real (class 1). We can imagine that the discriminator will then classify the generated samples as not real (class 0) or a low probability of being real (0.3 or 0.5). The back-propagation process used to update the model weights will see this as a large error and will update the model weights (i.e. only the weights in the generator) to correct for this error, in turn making the generator better at generating plausible fake samples.

- Inputs: Point in latent space, e.g. a 50-element vector of Gaussian random numbers.
- Outputs: Binary classification, likelihood the sample is real (or fake).

Define a function to take as arguments the already-defined generator and discriminator models and create the new logical third model subsuming these two models. The weights in the discriminator are marked as not trainable, which only affects the weights as seen by the GAN model and not the standalone discriminator model. The GAN model then uses the same binary cross-entropy loss function as the discriminator and the efficient Adam version of stochastic gradient descent.
Making the discriminator not trainable is a clever trick in the Keras API. The trainable property impacts the model when it is compiled. The discriminator model was compiled with trainable layers, therefore the model weights in those layers will be updated when the standalone model is updated via calls to train_on_batch(). The discriminator model was marked as not trainable, added to the GAN model, and compiled. In this model, the model weights of the discriminator model are not trainable and cannot be changed when the GAN model is updated via calls to train_on_batch(). By training the model in such a way for 2000 epochs we get that the discriminator identifies the fake data as the real data with 100% accuracy. This means that the generator model has trained itself in such a way that the discriminator model couldn't identify between real and fake data. If we already have a strong discriminator, then training the generator in such a way will make it possible to generate fake data which is so much similar to the original data.
Instead, what is required is that we first update the discriminator model with real and fake samples, then update the generator via the composite model. This requires combining elements from the train_discriminator() function defined in the discriminator section and the train_gan() function defined above. It also requires that the generate_fake_samples() function use the generator model to generate fake samples instead of generating random numbers. After generating the fake data using the GAN model, denormalize the last classification column between 0 to 5.
Finally, we can evaluate the model by testing the generated fake data using the discriminator model. By training the model for 500 epochs we get that the discriminator identifies the fake data with 85% accuracy. This type of training in the GAN model will help us get a very strong discriminator model.

V. Results and Evaluation

Comparing all the machine learning and deep learning neural networks used, we can see that the Autoencoder model gives the best score of 89% along with the ability to use a time-series dataset. Although the dataset that we used in this paper isn't a time series dataset, we can generate one using a strong generator model of GANs. Thus after training the generated fake data by generator model of GANs on the Autoencoder model, and then testing the original data, we get a score of 90%.

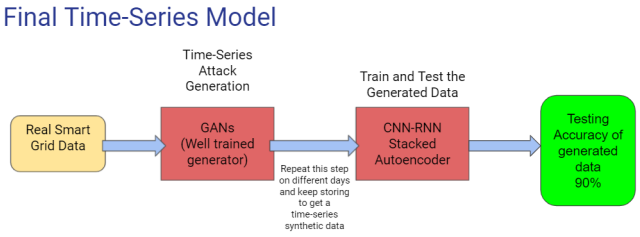

Fig. 2. Attacker-Model and Classification-Model combined

The plot of the discriminator and generator loss and accuracy as a function of epochs while training both the models simultaneously using the combined function can be seen below. Early in training, the losses oscillate as both models attempt to find their equilibrium, after which, both losses vary around a point that signifies stable training. Accuracy on the real and fake data are similar, showing that neither model is stronger than the other.
Line plots for loss and accuracy are created and saved at the end of the run. The figure contains two subplots. The top subplot shows line plots for the discriminator loss for real data (blue), discriminator loss for generated fake data (orange), and the generator loss for generated fake data (green). We can see that all three losses are somewhat erratic early in the run

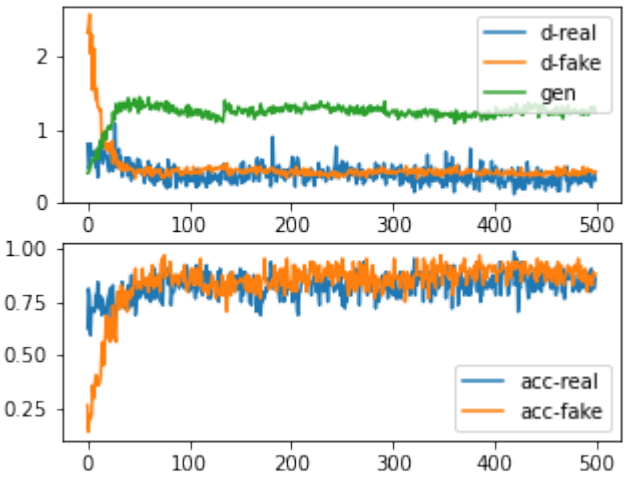

Fig. 3. Plot of each training accuracy and loss of the GAN model.

before stabilizing to epoch 100. Losses remain stable after that, although the variance increases.

This is an example of the normal or expected loss during training. Namely, discriminator loss for real and fake samples is about the same at or around 0.5, and loss for the generator is slightly higher at around 1.5. If the generator model is capable of generating plausible data, then the expectation is that those data would have been generated between after 100.

The bottom subplot shows a line plot of the discriminator accuracy on real (blue) and fake (orange) images during training. We see a similar structure as the subplot of loss, namely that accuracy starts off quite different between the two image types, then stabilizes between epochs 100 to 300 at around 70% to 80%, and remains stable beyond that, although with increased variance. This shows that the discriminator model is trained well enough to detect attack data separately from other fake data(like normal or no events data).

We have also checked the accuracy of the fake sample created using the generator model with respect to the original data by evaluating our synthetically generated data-set and measuring the maximum mean discrepancy (MMD) between real and synthetic data sets as probability distributions. Getting the MMD value as 0.15 shows that their sampling distance converges.

$$MMD = \left\{ \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} K(x_i, x_j) - \frac{2}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} K(x_i, y_j) + \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} K(y_i, y_j) \right\}^{1/2}$$

Measures the distance between two probability distributions by drawing samples. Given samples

$\{x_i\}_{i=1}^{N} \sim p(x)$ real data

$\{y_j\}_{j=1}^{M} \sim q(y)$ fake data

where $K(x, y) = \exp(-||x - y||^2/(2\sigma^2))$ is known as radial basis function (RBF) kernel

Fig. 4. Formula to Calculate the MMD

Thus we have designed a novel approach to classify cyber-attacks in smart grids using different deep neural networks.

VI. Future Works

We can perform classic machine learning tasks e.g. clustering and forecasting on both original and synthetic data sets. A potentially useful synthetic data-set shall show similar results as the original data-set in these tasks. Also, we can further strongly train the discriminator model of GANs to detect attack data from normal events or no event data.

Acknowledgment

I wish to show my appreciation to my supervisor, Prof. Kedar Khandeparkar and Smruti Dash for their continued support and encouragement. I offer my sincere appreciation for the learning opportunities provided by them.

References

[1] Liang Zhou, Lifang Han, Xuan Ouyang, Yushi Cheng, Huan Ying, Tianchen Zhang, China Electric Power Research Institute, Beijing 100192, China, College of Electrical Engineering Zhejiang University, Hangzhou, "Cyber-Attack Classification in Smart Grid via Deep Neural Network" , Research Gate, October 2018.

[2] Gianni D'Angelo, Francesco Palmieri, "International Journal of Intelligent Systems" , Volume36, Issue12, December 2021, Pages 7080-7102

[3] Chi Zhang, Sanmukh R. Kuppannagari, Rajgopal Kannan and Viktor K. Prasanna, Computer Science Department, University of Southern California, Los Angeles, Electrical Engineering Department, University of Southern California, Los Angeles, CA, "Generative Adversarial Network for Synthetic Time Series Data Generation in Smart Grids" , IEEE SmartgridComm publication, 2018