Machine Learning for Power Grid Intrusion Detection

Yutian Mei
yt.mei@mail.utoronto.ca

Gabriel Chan
gab.chan@mail.utoronto.ca

Yujie Qin
yujie.qin@mail.utoronto.ca

Zhengmao Ouyang
zhengmao.ouyang@mail.utoronto.ca

Abstract

The incorporation of automated control and communication into the power grid allows for greater control, yet introduces cybersecurity vulnerabilities. Recently, the Electric Power and Intelligent Control (EPIC) testbed was created to provide realistic data collection environment for the purpose of building Intrusion Detection System (IDS) attack detection models using Machine Learning (ML). This study looks to design a deep learning (DL) IDS approach using the EPIC_A dataset, specifically a Deep Neural Network (DNN), and provides an evaluative comparison against several baseline non-neural network (non-NN) architectures. However, most of the non-NN models performed similarly or better than the DNN. Still, current studies provide support for the efficiency of the DNN. Future work should look to further optimize the DNN model via a more comprehensive approach to designing, tuning, and training.

1 Introduction

The evolution of traditional power systems into smart grids has introduced new layers of efficiency, flexibility, and automation—but it has also exposed critical infrastructure to sophisticated cyberthreats. Smart grids rely on real-time data exchanges between sensors, controllers, and control centers, which makes them vulnerable to attacks that can disrupt power control such as False Data Injection (FDI), Time Delay Attacks (TDA), and Denial of Service (DoS) [1].

The integrity and confidentiality of these systems are paramount, and so intrusion detection systems (IDSes) that use various methods like rules-based, machine-learning, and statistical methods are vital to secure these smart grids against cyberattacks [2]. One major concern however is that these IDSes require real-world datasets to be effective, but realistic datasets are often proprietary and difficult to attain due to the amount of work to accurately label the data. This in turn limits the development and benchmarking of effective IDSes [2][3]. Simulation models have been used in place of high-fidelity datasets, but are often unable to fully replicate the patterns that occur in power systems and communication systems in real life [2].

To address this, the Electric Power and Intelligent Control (EPIC) testbed was developed and used to generate high-fidelity datasets that simulate more realistic grid scenarios under both normal and attack conditions [2][4]. The EPIC testbed is a modern smart grid platform designed to emulate real-world power system operations using actual electrical equipment. Various attacks can be performed on this testbed from which data is collected, resulting in generated datasets offering a more authentic and valuable resource than synthetic datasets typically used in IDS research [2][4]. To that end, the paper utilizes a dataset offered by [2], denoted as "EPIC_A".

Machine learning (ML) and deep learning (DL) approaches have emerged as particularly effective tools for intrusion detection. ML models provide automated pattern recognition and anomaly detection capabilities that surpass rule-based systems [6][7]. However, limitations such as the vast

amount of dynamic traffic flow limit the effectiveness of ML models, especially as data becomes higher-dimensional, making it even harder to extract meaningful features [7].

Recently, deep learning architectures—such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Deep Neural Networks (DNNs)-have emerged, and are especially well-suited due to their ability to extract hierarchical feature representations from high-dimensional input data. They are able to handle large amounts of data and can more accurately perform predictions through the use of their hidden layers [1][3][7]. A recent systematic review [4] highlights that DNNs are widely adopted for intrusion detection across domains like IoT, cloud, and smart infrastructure, often outperforming traditional ML models in accuracy and scalability. The DNN model presented in Vinayakumar et al. [3] demonstrates a high-performing IDS framework that can outperform classical machine learning classifiers. By leveraging such models and training them on realistic datasets like EPIC_A, researchers can design intelligent IDS pipelines that continuously learn and adapt to effectively protect critical smart grid infrastructure.

2 Problem Formulation

2.1 Data

As previously mentioned, the EPIC_A dataset comprises data taken from the eponymous testbed, designed for the purpose of cyberattack simulation on realistic end-to-end large-scale power systems models with fleshed out generator, transmission, and distribution components [2]. Simulated attacks in this dataset focus on the generation stage, and mainly leverage two important principles important to power systems. The first is that a generator introduced into a power system needs to be brought to the exact frequency as said power system, a phenomenon known as **"generator synchronization"**. The second is that a generator in a synced group which fails to maintain its rotation speed will begin to act like a motor and draw power from the grid in a phenomenon known as **"reverse power flow"**.

Each data point is a 9-tuple, whose features are captured from Wireshark during testbed simulations. These features reflect communicated measurements and messages that can be modified to carry out cyberattacks. In a False Data Injection Attack (FDIA), the value of one or more features is manipulated to be incorrect before the measurement or message containing it is received. Thus, generator synchronization and prevention of reverse power flow are stopped. In a Time Delay Attack, a specific feature called VSD1_Speed is manipulated to delay generator synchronization or the prevention of reverse power flow. The dataset comprises 13448 Normal samples, 3356 FDIA samples, and 5555 TDA samples. Mathematically, it may be expressed:

$$\mathbb{D} = \{(x_n, v_n), 1 \leq n \leq 22359\}$$

where each x_n represents a 9-tuple of features in $\mathbb{R}^9$, and $v_n \in \mathbb{V} = \{\text{Normal, FDIA, TDA}\}$. The imbalanced nature of the dataset is typical of cybersecurity applications.

2.2 Model and Learning Algorithm

The main architecture studied in this paper is the DNN, which is a fully connected neural network (NN) with at least two hidden layers. Let $\boldsymbol{w}$ be a vector representation of all the weights in the DNN. Then, the DNN has a mathematical representation as a function:

$$f_{\boldsymbol{w}} : \mathbb{R}^9 \mapsto \mathbb{V}$$

Where m represents the number of weights in the DNN, $\boldsymbol{w} \in \mathbb{R}^m$, and thus the algorithm A used to train the DNN can be conceptualized as a search through the hypothesis space:

$$\mathbb{H} = \{f_{\boldsymbol{w}} \mid \boldsymbol{w} \in \mathbb{R}^m\}$$

for an optimal set of weights $\boldsymbol{w}$ such that the output of $f_{\boldsymbol{w}}$ over some training subset of $\mathbb{D}$ minimizes a given loss function L. Here, the typical cross-entropy loss function is used.

The Decision Tree, Logistic Regression (LR), K-Nearest-Neighbour (KNN), and Random Forest (RF) ML models are also considered for the sake of a baseline comparison against a variety non-NN architectures which can reliably be trained on limited hardware resources and time constraints. Their mathematical representations are similar.

2.3 Evaluation

Ultimately, the performance of each model is evaluated on some testing subset of $\mathbb{D}$ to produce the metrics of classification accuracy, precision, and recall. For a class i, let P_i be the number of data points classified as i, let C_i be the number of points in class i, and let TP_i be the number of points *correctly* classified as i. Then,

$$\text{Classification Accuracy} = \frac{\sum_i TP_i}{\sum_i C_i},\ \text{Precision}_i = \frac{TP_i}{P_i},\ \text{Recall}_i = \frac{TP_i}{C_i}$$

The classification accuracy is the percentage correctly classified points overall, whereas the precision for a class i is the percentage of data points classified as i that actually have label i, and the recall for a class i is the percentage of data points having label i which were classified as i. Precision and recall are imperative measures in an imbalanced dataset.

The goal of this study is to design a DNN model that generalizes well on the EPIC_A dataset, balancing classification accuracy, precision, and recall. This will be done using a systematic determination of DNN hyperparameters, and the final results will be compared to non-NN models. Each distinct set of hyperparameters contains model training as a subproblem, which decomposes into the usual data, model, algorithm formulation as described above.

3 Design

The process used to design the DNN model follows a structured determination of hyperparameters, based on the experiments outlined in [3], with the training/validation/test split over $\mathbb{D}$ as 60%, 20%, and 20%. The training data is also passed through SKLearn's StandardScaler. The Normal, FDIA, and TDA labels are numerized as 0, 1, and 2, respectively.

First, the optimal combination of hidden layers, hidden layer widths, dropout values is determined. The dataset contains just nine features, and thus the number of hidden layers is kept as 2, 3, or 4. The widths of hidden layers, similar to [3], are kept as decreasing consecutive powers of 2, with the first hidden layer having a width in the set $\{512, 256, 128, 64\}$. Dropout values were tested from the set $\{0, 0.01, 0.1\}$. The following shows the validation accuracy of the epoch with lowest validation loss, averaged over 3 trials. The best dropout value was found to be 0.1, so only results with dropout 0.1 are shown for brevity.

Table 1: Varying DNN Structures vs. Validation Accuracy of the Epoch with Lowest Validation Loss, Averaged Over 3 Trials

# Hidden Layers	Layer Widths	Avg. Validation Accuracy
2	$9 \to 64 \to 32 \to 3$	0.8841
2	$9 \to 128 \to 64 \to 3$	0.8900
2	$9 \to 256 \to 128 \to 3$	0.8912
2	$9 \to 512 \to 256 \to 3$	0.8928
3	$9 \to 256 \to 128 \to 64 \to 3$	0.8984
3	$9 \to 512 \to 256 \to 128 \to 3$	0.8928
4	$9 \to 256 \to 128 \to 64 \to 32 \to 3$	0.8957
4	$9 \to 512 \to 256 \to 128 \to 64 \to 3$	0.8960

Here, 3 hidden layers having widths 256, 128, and 64 were chosen. Increasing the number of hidden layers from 2 to 3 leads to better validation accuracy, but the improvement diminishes with more layers. Also, more hidden layers comes with longer training time and higher risk of overfitting. Increasing the number of neurons in the hidden layers generally leads to better validation accuracy, suggesting that a higher capacity model can capture more complex patterns in the data. But further increasing does not help much as the model complexity approaches saturation. Using a small amount of dropout can improve validation accuracy, and helps to prevent overfitting. The small improvement in performance indicates that no serious overfitting has occurred.

In performing these experiments, the use of batch normalization, as well as the choice of activation function and optimizer were also considered. While batch normalization helped accelerate the training process to some extent requiring less epochs to converge, no significant improvement in

model performance was witnessed, hence batch normalization was not utilized. ReLU, tanh, and sigmoid activation functions were also tested, with no significant change in model performance. ReLU is well-known for its simplicity and effectiveness that often leads to faster training, and thus was kept. For the choice of optimizer in training function, Adam undoubtedly outperformed SGD. It has several advantages, including adaptive learning rates and incorporating momentum, which help accelerate convergence and make it robust to noisy gradients.

Lastly, the batch size and learning rate were tuned. The batch sizes were selected from the set $\{16, 32, 64, 128, 256, 512\}$, and the learning rate was taken from the set $\{0.0001, 0.001, 0.01\}$. Averaged against 3 trials, the following shows the validation accuracy of the epoch with lowest validation loss. For brevity, only results with learning rate 0.001 are shown, which was ultimately the chosen learning rate.

Table 2: Varying Batch Sizes vs. Validation Accuracy of the Epoch with Lowest Validation Loss, Averaged Over 3 Trials

Batch Size	Avg. Validation Accuracy	Batch Size	Avg. Validation Accuracy
16	0.8931	128	0.8962
32	0.8947	256	0.9001
64	0.8984	512	0.8914

Overall, a small/large batch size should come with a low/high learning rate. A proper choice of learning rate and batch size provides a balance between convergence speed and model performance. Even though a batch size of 256 gives the best validation accuracy statistic, a batch size of 128 is chosen to pair with the learning rate of 0.001 for improved generalization.

The final structure of the DNN is given as:

Table 3: Final DNN Structure

Layer	Type	# Units	Activation
0-1	fully connected input	$9 \rightarrow 256$	ReLU
1-2	dropout		
2-3	fully connected	$256 \rightarrow 128$	ReLU
3-4	dropout		
4-5	fully connected	$128 \rightarrow 64$	ReLU
5-6	dropout		
6-7	fully connected output	$64 \rightarrow 3$	-

The DNN is trained using a batch size of 128, a learning rate of 0.001, and a dropout of 0.1 with the Adam optimizer. The implementation is done using PyTorch.

Non-neural network (non-NN) models were implemented using SKLearn, and their hyperparameters were tuned using GridSearchCV. These models were tuned and tested on the data over three trials, and the determined hyperparameters are given Table 4.

The specific sets of hyperparameters for each model over which GridSearchCV was performed are available in Table 6 in the Appendix.

Table 4: Non-NN Model Hyperparameters

Model	Hyperparameters
LR	'C': {1, 10}, 'penalty': l2, 'solver': lbfgs
DT	'criterion': {entropy, gini}, 'max_depth': None, 'min_samples_split': 10
KNN	'metric': euclidean, 'n_neighbours': {5,7,9}, 'weights': uniform
RF	'max_depth: {20, None}, 'min_samples_leaf: 2, min_samples_split: {5,10}, 'n_estimators' : {100,200}

4 Results

A final evaluation is performed on each of the DNN and non-NN models over test subsets of $\mathbb{D}$, which accounts for a 20% split of the EPIC_A dataset. The DNN and non-NN models were conducted by separate individuals in separate experiments - the testing data for the DNN was created using

SKLearn's train_test_split function with random_seed set to 0. The non-NN models were tested over three trials with random_seed set to 1,2, and 3 - results were averaged over the three trials.

Table 5: Final Evaluation Results of DNN and non-NN Models

Model	Acc.	Pr. Normal	R. Normal	Pr. FDIA	R. FDIA	Pr. TDA	R. TDA
DNN	0.896	0.933	0.900	0.823	0.823	0.855	0.927
LR	0.750	0.774	0.856	0.740	0.239	0.695	0.788
DT	0.911	0.920	0.937	0.873	0.842	0.912	0.890
KNN	0.907	0.935	0.913	0.850	0.854	0.877	0.923
RF	0.933	0.934	0.958	0.907	0.844	0.943	0.922

Overall, the DNN is exhibits accuracy around 90% in classifying Normal data points, with similar precision and recall scores. It performs similarly well with the TDA class, although the precision is relatively low compared to the recall. This indicates the model is more likely to produce false negatives with TDA data. More significantly, it is consistently weaker in classifying FDIA datapoints, with statistics in the low eighties for both precision and recall. However, statistics for the FDIA class are generally lower across all considered models, indicating that the patterns comprising data under this label might be much more subtle or complex than other classes. Indeed, four FDIA attacks were simulated in [2], whereas for the TDA class just two attacks were given.

It is notable that some of the more sophisticated non-NN models, including the DT, KNN, and RF models, actually outperform the DNN on many of the evaluation metrics. Even though the derivation of DNN hyperparameters was systematic, this shows that a broader range of structures could have been considered. It is also possible that other metrics could have been more useful during hyperparameter selection, since decisions were based only on the validation accuracy of the epoch with lowest validation loss. Since the non-NN models were run on an 80% training and 20% test split, rather than a 60% training, 20% validation, and 20% test split, these models may have had more data on which to train, resulting in their better, albeit non-comparable performance. Another interpretation is that a neural network approach was in fact not needed for the EPIC_A dataset, and that a simpler approach would have sufficed. The performance of the logistic regression classifier is as expected, since it is not designed to capture non-linear relationships in data.

5 Concluding Remarks

This study sought the determination of a DNN structure for effective classification on the EPIC_A dataset, comprising nine features over Normal conditions, as well as FDIA and TDA attacks on generation control in a realistic EPIC testbed. Taking after the process in [3], many hyperparameters were tested, including varying the number of hidden layers, layer widths, dropout values, batch normalization, activation, optimizer, batch size, and learning rate. Combinations of hyperparameters were evaluated using a 60% of the data for training and 20% for validation over three trials, with the average best validation accuracy over three trials taken to be the point of comparison against other combinations. Ultimately, a DNN with three hidden layers having widths 256, 128, and 64, with dropout 0.1, no batch normalization, ReLU activation, Adam optimizer, batch size 128, and learning rate 0.001 was chosen, which was finally evaluated with a 20% testing split of the EPIC_A dataset.

This DNN was compared to non-NN structures, including decision tree, logistic regression, SVM, and KNN models, which underwent performance testing three trials, each trial including a GridCVSearch for hyperparameters on a 80% train and 20% test split. With the exception of the logistic regression classifier, these models performed around the same or even better than their DNN counterpart on classification accuracy, as well as precision and recall over the different classes. Greater exploration of the DNN hyperparameter space, better training criteria, and consistency in the data splitting as well as testing between the non-NN and DNN models could have resulted in the desired result, where the DNN shows a classification improvement over the considered alternatives. It is also possible that non-NN structures were wholly sufficient to capture the patterns in the data alone. These are all points that future work would take into consideration.

References

[1] R. Kimanzi, P. Kimanga, D. Cherori, and P. K. Gikunda, "Deep Learning Algorithms Used in Intrusion Detection Systems – A Review," Feb. 26, 2024, arXiv: arXiv:2402.17020. doi: 10.48550/arXiv.2402.17020.

[2] H. C. Tan, M. Adeeb Hossain, D. Mashima, and Z. Kalbarczyk, "High-fidelity Intrusion Detection Datasets for Smart Grid Cybersecurity Research," in 2024 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), Sep. 2024, pp. 340–346. doi: 10.1109/SmartGridComm60555.2024.10738043.

[3] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep Learning Approach for Intelligent Intrusion Detection System," IEEE Access, vol. 7, pp. 41525–41550, 2019, doi: 10.1109/ACCESS.2019.2895334.

[4] C. Siaterlis, B. Genge, and M. Hohenadel, "EPIC: A Testbed for Scientifically Rigorous Cyber-Physical Security Experimentation," IEEE Transactions on Emerging Topics in Computing, vol. 1, no. 2, pp. 319–330, Dec. 2013, doi: 10.1109/TETC.2013.2287188.

[5] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A Detailed Investigation and Analysis of Using Machine Learning Techniques for Intrusion Detection," IEEE Communications Surveys & Tutorials, vol. 21, no. 1, pp. 686–728, 2019, doi: 10.1109/COMST.2018.2847722.

[6] Md. A. Talukder et al., "Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction," Journal of Big Data, vol. 11, no. 1, p. 33, Feb. 2024, doi: 10.1186/s40537-024-00886-w.

[7] M. L. Ali, K. Thakur, S. Schmeelk, J. Debello, and D. Dragos, "Deep Learning vs. Machine Learning for Intrusion Detection in Computer Networks: A Comparative Study," Applied Sciences, vol. 15, no. 4, Art. no. 4, Jan. 2025, doi: 10.3390/app15041903.

Appendix

Table 6: Structure of Data in the EPIC_A Dataset

Feature	Description
TotW	G2 MIED MMS measurement of instantaneous real power
TotVar	G2 MIED MMS measurement of instantaneous reactive power
Phv.neut.ang	G2 MIED MMS measurement of phase-to-neutral angel
A.phsA	G2 MIED MMS Measurement of Root-mean-square current, phase A
sync_start	SCADA MMS command to start synchronization
sync_complete	G1 MIED MMS command to stop synchronization
gen1_p_negative	G1 MIED MMS command indicating reverse power
gen2_p_negative	G2 MIED MMS command indicating reverse power
VSD1_speed	SLPC Modbus command to regulate G1 VSD

In the EPIC testbed, generator synchronization is ensured and reverse power flow is prevented by an automated regulatory scheme known as the "Supervisory Control And Data Acquisition" (SCADA) system, which communicates with smart programmable logic controllers (SPLCs) and microgrid intelligent electronic devices (MIEDs) to ensure proper grid functioning. Here, MIEDs are responsible for monitoring generator power output, and accordingly issue Manufacturing Message Specification (MMS) commands with to SPLCs. MIEDs also produce MMS messages with measurement data. SPLCs then act on variable speed drives (VSDs) via a protocol known as TCP Modbus to ultimately control their corresponding generators to a desired speed setpoint in rotations per minute. The primary role of the SCADA system in these experiments is to issue MMS commands to the SPLCs for starting synchronization. The diagram for this is shown in the following figure, as taken from [2].

Figure 1: Systems Diagram of Generation Control in the EPIC Testbed [2]

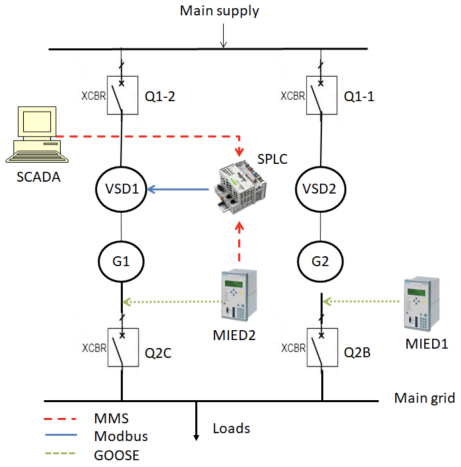

Table 7: Hyperparameter Search Space for Non-NN Models

Model	**Hyperparameters**
LR	'solver': {liblinear, lbfgs}, 'C': {0.01, 0.1, 1, 10}, 'penalty': {l2}
DT	'criterion': {gini, entropy}, 'max_depth': {3, 5, 10, None}, 'min_samples_split': {2, 5, 10}
KNN	'n_neighbours': {3,5,7,9}, 'weights': {uniform, distance}, 'metric': {euclidean, manhattan, minkowski}
RF	'n_estimators': {50,100,200}, 'max_depth': {None, 10, 20}, 'min_samples_split': {2,5,10}, 'min_samples_leaf': {1,2,4}