Predicting Smart Grid Stability

Mamadou MARONE
Double degree student at ETSIT-UPM
Madrid, Spain

Inass RACHIDI
Double degree student at ETSIT-UPM
Madrid, Spain

***Abstract*—In this project, we address the problem of predicting the stability of smart electric grids using bio-inspired learning algorithms. With the rise of renewable energy sources, smart grids have become complex bidirectional networks where consumers also act as suppliers, making stability assessment challenging. We leverage a differential equation-based model to generate data and train neural networks to mimic this model. Our approach utilizes genetic algorithms for optimizing the neural network's architecture and spiking neural networks for enhanced training. This paper presents our methodology, experiments, and results, demonstrating the effectiveness of bio-inspired algorithms in predicting grid stability with improved accuracy.**

I. Introduction

The integration of renewable energy sources has transformed traditional unidirectional power grids into complex, bidirectional smart grids. In these systems, end-users not only consume but also produce and supply energy, necessitating advanced methods for managing supply and demand. Grid stability, a critical concern, depends on dynamically balancing power production and consumption while considering economic factors like energy pricing.

Traditional models, such as the Decentral Smart Grid Control (DSGC) differential equation-based model, provide a framework for predicting grid stability. However, the complexity of these models often requires simplifications that limit their accuracy and practical applicability.

II. Motivation & Objectives

This study aims to overcome the limitations of traditional models by employing machine learning techniques, specifically neural networks, to predict grid stability more accurately. First, we utilized a simple neural network classifier optimized using a Genetic Algorithm. Subsequently, we developed a model based on a Spiking Neural Network.

Our objectives are twofold: to investigate the effectiveness of the Genetic Algorithm (GA) in optimizing neural network architecture and to evaluate the performance of Spiking Neural Networks in solving our problem. Essentially, we aim to determine whether bio-inspired learning methods can be effectively employed to develop a model for our specific problem.

III. Methodology

A. Data

The dataset "Electrical Grid Stability Simulated Dataset" that we used for our experiments was obtained through simulations using a Differential Stability Grid Control (DSGC) model. In the context of a 4-node star network (one supplier node and three consumer nodes), the DSGC model involves a set of differential equations that describe the dynamics of the grid based on the reaction times, nominal power, and price elasticity coefficients of the participants.

For node i (where $i = 1$ for the supplier and $i = 2, 3, 4$ for the consumers):

$$\frac{d\theta_i}{dt} = \omega_i$$

$$M_i \frac{d\omega_i}{dt} + D_i\omega_i = P_i - \sum_j E_{ij} \sin(\theta_i - \theta_j)$$

where:

- θ_i is the phase angle of node i
- θ_j is the phase angle of node j
- ω_i is the angular frequency deviation of node i
- M_i is the inertia coefficient of node i
- D_i is the damping coefficient coefficient of node i
- P_i is the nominal power produced or consumed at node i such as $P_1 = -(P_2 + P_3 + P_4)$
- E_{ij} is the coupling strength between nodes i and j

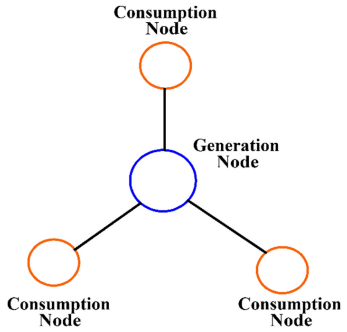

Fig. 1. 4-node star

The original dataset contains 10,000 observations. As the reference grid is symetric, the dataset can be augmented in 3! times, representing a permutation of the three consumers occupying three consumer nodes. The augmented version has then 60,000 observations. It also contains 12 primary predictive features and two dependent variables.

Predictive features:

- 'tau1' to 'tau4': the reaction time of each network participant,

- 'p1' to 'p4': nominal power produced (positive) or consumed (negative) by each network participant,
- 'g1' to 'g4': price elasticity coefficient for each network participant.

Dependent variables:

- 'stab': the maximum real part of the characteristic differentia equation root (if positive, the system is linearly unstable; if negative, linearly stable);
- 'stabf': a categorical (binary) label ('stable' or 'unstable').

B. Genetic Algorithm for Neural Network Optimization

Before conducting experiments, we will dive into the concept of genetic algorithms and their relevance to optimizing neural networks, with a specific focus on addressing the challenges posed by our electric grid stability detection problem. Genetic algorithms, drawing inspiration from natural selection and evolution, present a promising methodology for efficiently navigating the vast solution space inherent in neural network architectures.

We will examine whether genetic algorithms are aptly suited for this endeavor, leveraging their capacity to simultaneously explore multiple potential solutions and adapt dynamically to the intricacies of our problem. By emulating the evolutionary process, genetic algorithms can iteratively refine neural network architectures, selecting and modifying candidate solutions based on their performance.

C. Spiking Neural Networks

Spiking Neural Networks (SNNs) are a type of artificial neural network that more closely mimic the way biological brains operate. Unlike traditional artificial neural networks (ANNs) that process information using continuous values and perform computations in a synchronized manner, SNNs use discrete events known as spikes to communicate between neurons, introducing a temporal dimension to their operation. Each neuron in an SNN accumulates incoming spikes and fires a new spike when its accumulated input reaches a certain threshold. This spiking mechanism allows SNNs to efficiently process temporal and dynamic data, making them highly suitable for real-time applications such as predicting smart grid stability. By leveraging the timing and frequency of spikes, SNNs can capture complex patterns and interactions in data, offering advantages in energy efficiency and computational power compared to traditional ANNs.

IV. Experiments

A. Genetical Algorithm: Experimental Setup and Evaluation

We begin by optimizing the neural network's architecture using a genetic algorithm. This bio-inspired technique iteratively evolves a population of neural network configurations, selecting the best-performing models and introducing variations through crossover and mutation. Key parameters optimized include the number and size of hidden layers, activation functions, number of iteration (number of generation), the mutation percentage and the number of solution in each generation.

1) Implementation of the model adapted for GA optimization: To apply the genetic algorithm to neural network architecture optimization, we began by initializing a dictionary of matrices representing the different layers of the neural network. These matrices were then flattened into vectors, which were concatenated to form a final vector that mimics the genes of an individual solution.

To evaluate a given gene as a solution for the neural network, the corresponding vector was converted back into a dictionary of matrices and used with an activation function to perform a forward pass in the neural network.

2) Optimization process: At each generation, the vectors leading to the best solutions were selected as parents to generate offspring. These offspring vectors shared sections with the selected parents, with the goal of achieving better performance. Additionally, some values in the vectors of the offspring were modified with a given probability, known as the mutation percentage.

To select the best parents, we needed to choose a fitness function to optimize across generations. Given that we were performing binary classification, we chose accuracy as the fitness function.

3) Configurations & Evaluation : We also recorded the time required to optimize the parameters and saved the evolution of accuracy for both the training and validation sets across generations in a CSV file.

Following this setup, we experimented with various configurations related to both the neural network architecture and the genetic algorithm itself. This approach allowed us to analyze the effectiveness of the genetic algorithm for neural network optimization. Additionally, it provided a heuristic method to optimize these parameters to achieve the best performance for solving our problem.

In the Result section we will present the outcomes of these experiments and make discussion about them.

B. Spiking Neural Networks

1) Data Preprocessing: In the data preprocessing stage, the dataset is first shuffled to ensure that the data is well-mixed before splitting into training and testing sets. This randomness helps to avoid any potential bias in the data order. The features and the target variable are then separated, with the features stored in one variable and the target classification labels in another. To ensure uniformity in feature scales, standardization is applied. This involves removing the mean and scaling to unit variance, which is crucial for effective model training.

2) Model Architecture: The Spiking Multi-Layer Perceptron (SpikingMLP) model architecture combines elements of traditional artificial neural networks with spiking neuron models, aiming to simulate the complex behavior of biological neurons while enabling gradient-based learning. It consists of two dense layers for linear transformations and feature extraction, followed by two Leaky Integrate-and-Fire (LIF) neuron layers. In these LIF neuron layers, the continuous input data is interpreted as spikes over time, accumulating these spikes to generate membrane potentials, and emitting spikes based on a

threshold mechanism. This dynamic spiking behavior allows the network to process information temporally and capture the dynamics of the input data. The parameters of the LIF neuron layers, such as leakiness and threshold, control the behavior of the neurons and modulate their sensitivity to input spikes. Additionally, the surrogate gradient method is employed to approximate gradients and facilitate backpropagation through the network, enabling the training of spiking neural networks despite the non-differentiability of spikes.

3) Model Training: The training process involves iterating over multiple epochs. In each epoch, the model is set to training mode, and performance metrics such as total loss and accuracy are tracked. During each iteration, the model processes batches of data, computes predictions, calculates the loss, and updates the model parameters using an optimization algorithm. The training accuracy and average loss are calculated and printed after each epoch to monitor the model's learning progress. This step is crucial for ensuring that the model improves over time and converges to an optimal solution.

4) Model Evaluation: Model evaluation is conducted using cross-validation to ensure robustness and generalizability, especially in the context of imbalanced datasets. The dataset is split into several folds, typically using techniques like stratified k-fold, to ensure that each fold maintains the same class distribution as the original dataset. This approach helps mitigate the impact of class imbalance, ensuring that the model is trained and validated on representative samples from each class. During evaluation, performance metrics such as accuracy, precision, recall, and F1 score are computed and aggregated across all folds. Additionally, a confusion matrix is generated to analyze the results better.

V. Results and Discussion

A. Genetic Algorithm Optimization

Our experiments demonstrate that the genetic algorithm effectively optimizes the neural network architecture to achieve high accuracy in detecting instability in the electric grid. The performance of the optimization depends on various parameters related to the genetic algorithm, which can be adjusted to balance accuracy and computational time.

The figure below shows the results obtained with different configurations. Each configuration changes only one parameter at a time to observe its impact on performance and computational duration.

In the first three configurations, the objective was to determine the optimal architecture by varying the number of neurons in the hidden layers. As expected, these changes affected both performance and computation time. In the first and third configurations (C1 & C3), we observed underfitting by comparing the accuracy on the training and testing sets, as shown in the following figure (Fig. 2). The second configuration (C2) proved to be the best in terms of neural network architecture, although it was less optimal in terms of computational time.

Parameters	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**
NUM_NEU_1	8	12	10	12	12	12	12
NUM_NEU_2	4	6	6	6	6	6	6
NUM_GEN	500	500	500	500	500	500	650
MUT_PER	10	10	10	10	10	25	10
SOL_PER_POP	16	16	16	8	32	16	16
ACTIV.	Sig.	Sig.	Sig.	Sig.	Sig.	Sig.	Rel.
ACCURACY(%)	75.3	82.6	80.9	63.8	83.2	78.2	76.8
DURATION(h)	2	2.3	1.9	1	3.84	3.2	2.2

TABLE I
CONFIGURATION ATTRIBUTES AND THEIR CORRESPONDING ACCURACY AND COMPUTATIONAL TIME

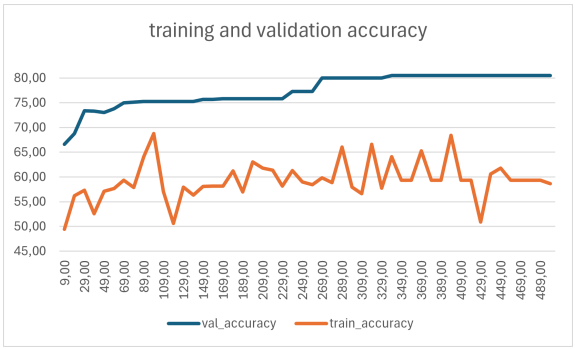

Fig. 2. Train & Validation Accuracy along generation

Next, we examined the impact of offspring size or the number of solutions at each generation in configurations C4 and C5. Initially, it is evident that computational time increases proportionally with the offspring size. Comparing the performances:

- Configuration C4 (SOL_PER_POP = 32) achieved an accuracy of 83.2% with a computational time of 3.2 hours.
- Configuration C5 (SOL_PER_POP = 8) achieved an accuracy of 63.8% with a computational time of 1 hour.
- Configuration C6 (SOL_PER_POP = 16) achieved an accuracy of 82.6% with a computational time of 2.3 hours.

We observe that increasing the offspring size can enhance performance, but it also results in a trade-off between performance and computational time.

Finally, we investigated the impact of increasing the mutation probability and changing the activation function in configurations C6 and C7, respectively. However, both experiments resulted in poorer performance, with accuracies of 78.2% and 76.8%, respectively.

Regarding the mutation probability, this decline in performance may be attributed to the fact that excessive mutation leads to exploration rather than exploitation of the knowledge gained from evolution, potentially resulting in suboptimal accuracy.

As for the activation function, although the reason behind this observation is unclear, the Sigmoid function appeared to be more efficient than the ReLU function.

B. Spiking Neural Networks

The table below encapsulates the collective performance metrics derived from our model across all folds during the cross-validation process. These metrics serve as valuable indicators of both the overall efficacy and reliability of our model in addressing the classification task at hand.By combining the mean values for training loss and accuracy across all epochs within each fold, alongside the precision, recall, F1-score, and accuracy on the validation dataset for every fold, this concise summary offers a panoramic view of our model's performance evolution throughout the training regimen.

Fold	Tr. Loss	Tr. Accur	Accur	Precision	Recall	F1
1	0.406	89.9%	93%	92%	88.4%	90.1%
2	0.4148	89.1%	92.8%	91.7%	88%	89.9%
3	0.415	88.9%	92.3%	92.9%	85.3%	88.9%
4	0.406	89.7%	92.8%	93.7%	85.9%	89.6%
5	0.407	89.8%	92.3%	93.1%	85.1%	88.9%

TABLE II
PERFORMANCE METRICS ACROSS EACH FOLD

Tr. Accur : Training Accuracy across all the epochs of the given fold

Tr. Loss : Training Loss across all the epochs of the given fold

To get an overall summary of the model's performance, we could compute the mean value of each of the evaluation metric for all the folds

Metric	Mean	Standard deviation
Accuracy	92.6%	0.003
Precision	92.7%	0.007
Recall	86.5%	0.014
F1-score	89.5%	0.005

The performance of our Spiking Neural Network (SNN) model showcases promising results, as evidenced by the collective performance metrics derived from cross-validation.

During training, which lasted approximately 1 minute and 58 seconds, we employed a cross-validation strategy with five folds. Each fold underwent five epochs of training with a batch size of 32 and a learning rate of 0.001. Despite experimenting with hyperparameters, including the beta value for the Leaky integrate-and-fire (LIF) neurons, we found that the initial configuration yielded optimal results. Thus, we maintained the number of folds, epochs, batch size, and learning rate unchanged throughout training.

VI. CONCLUSION

This study presents a novel application of bio-inspired learning algorithms to predict smart grid stability. By optimizing neural network architecture using a genetic algorithm and training spiking neural networks, we achieved good accuracy, with the best performance provided by the spiking neural network, which also proved faster to train. However, with further hyperparameter tuning, it is possible to enhance the performance obtained using the genetic algorithm as an optimizer for neural network architecture.

Our results underscore the potential of bio-inspired techniques in enhancing the reliability and efficiency of smart grid management, as well as contributing to advancements in deep learning. Future work will focus on refining the configurations of the genetic algorithm to improve scalability and address the issue of time consumption. Additionally, further investigation into the configurations and capabilities of spiking neural networks will be conducted to fully harness their potential in this domain.

REFERENCES

[1] Mithat Önder, Muhsin Ugur Dogan, Kemal PolatKemal Polat, "Classification of smart grid stability prediction using cascade machine learning methods and the internet of things in smart grid" Neural Computing and Applications 35(24):1-19 May 2023

[2] Ahmed Fawzy Gad, "PyGAD: An Intuitive Genetic Algorithm Python Library" arXiv, 2021

[3] PAULO BREVIGLIERI, "Predicting Smart Grid Stability with Deep Learning" Kaggle, 2020