Opening Black Box: Potentiation Learning Artificial Neural Network

Author: Hasan Can Beydili

22250101008@subu.edu.tr

Abstract

In this paper, I aim to present a novel artificial neural network algorithm capable of learning complex relationships while maintaining a simple model architecture, contributing to the field of Explainable Artificial Intelligence (XAI).

This algorithm uses no gradient descent, no backpropagation, no loss functions, no calculus, no epoch, no hidden layer and no unnecessary neurons.

The algorithm was designed to address the gap in the literature regarding the lack of an architecture that is "as transparent as logistic regression and as powerful as Multi Layer Perceptrons (MLP)."

In the realm of artificial intelligence, Long-Term Potentiation serves as a powerful metaphor for understanding how neural systems can store and process information. Long-Term Potentiation is a biological process wherein synaptic connections between neurons are strengthened through repeated stimulation, forming the basis for long-term memory and learning in the brain. Similarly, the Potentiation Learning Artificial Neural Network incorporates a mechanism analogous to Long-Term Potentiation to facilitate learning and memory storage.

Long-Term Potentiation, a phenomenon in neuroscience, refers to the long-lasting strengthening of synaptic connections between neurons. It occurs when synapses are repeatedly activated or strongly stimulated, resulting in enhanced neural transmission. This process plays a crucial role in learning and memory formation and typically involves chemical and structural changes in synaptic connections.

1 – Introduce

1.1 – Introduction

Consider a neural network architecture that eliminates the need for gradient calculation processes inherent in traditional backpropagation and bypasses the repetitive iterative steps. This novel approach offers a more interpretable alternative. Moreover, it redefines the concept of the 'hidden layer' with an innovative design, providing a fresh perspective on this traditionally opaque component.

In the existing literature, research in the field of XAI predominantly focuses on analyzing and improving the interpretability of models trained with conventional MLP architectures [1][2][3]. However, this study introduces a fundamentally new learning architecture that has the potential to address these interpretability challenges at their core.

This paper presents the Potentiation Learning Artificial Neural Network (PLAN), a neural network architecture built upon these principles. Through empirical evidence and examples from various projects, I aim to substantiate these claims. The goal is to develop models that are as transparent as logistic regression, while maintaining the computational power and complexity of multi-layer perceptrons.

2 – Architectures

2.1 – What is Logistic Regression & Multi Layer Perceptron ?

2.1.1 – Logistic Regression

Definition

Logistic regression is a statistical model that establishes a linear relationship between the input features and the target variable. It is primarily used for classification tasks, especially binary classification, by predicting the probability of a target class using the logistic (sigmoid) function. Unlike more complex models, logistic regression does not incorporate non-linear transformations or hidden layers.

Mathematical Representation:

$$P(y = 1 \mid x) = \sigma(w^T x + b), \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

where ‘w’ represents the feature weights, ‘x’ the input features, and ‘σ’ the sigmoid function.

Why It Does Not Learn Complex Relationships

Logistic regression’s inability to learn complex relationships stems from its architecture:

1. No Activation Transformations: Logistic regression relies solely on a linear combination of input features, followed by a sigmoid function to output probabilities.

The absence of non-linear activation functions prevents the model from capturing non-linear dependencies in the data.

2. No Hidden Layers: Without hidden layers, the model lacks the hierarchical feature extraction capabilities necessary for learning complex patterns.

Interpretability

Logistic regression is considered a highly interpretable model due to the following characteristics:

- **Direct Weight Interpretation:** Each weight (wi) corresponds to the impact of a specific feature on the target variable. A positive weight indicates a positive correlation, while a negative weight indicates the opposite.
- **Feature Importance:** The magnitude of the weights provides a straightforward measure of feature importance.
- **Decision Boundary:** The linear nature of the decision boundary makes it easy to understand and visualize.
- **Simplicity:** The model's architecture ensures that predictions are easily explainable, even to non-technical audiences.

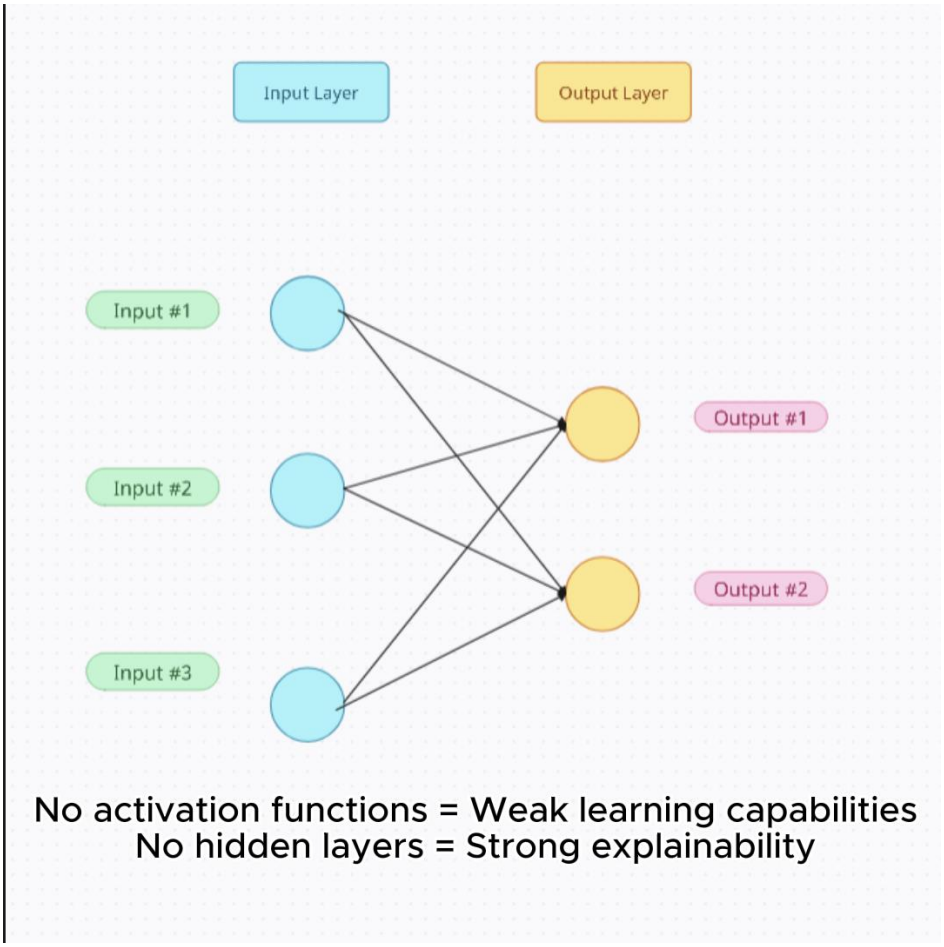

Figure 1: Example Logistic Regression Architecture

2.1.2-Multilayer Perceptron (MLP)

Definition

The Multilayer Perceptron is a type of artificial neural network capable of learning non-linear relationships between input features and the target variable. It consists of an input layer, one or more hidden layers, and an output layer, with each layer applying a combination of linear transformations and non-linear activation functions. It is also a machine learning algorithm with a broad range of applications, extensively utilized in GPT (Generative Pretrained Transformers) models and other popular Large Language Models (LLMs).

Mathematical Representation (for one hidden layer):

$$h(x) = \varphi(W_1 x + b_1), \quad y = \sigma(W_2 h(x) + b_2)$$

where 'ϕ' denotes a non-linear activation function (e.g., ReLU, tanh), and W1,W2 are weight matrices.

How It Learns Complex Relationships

MLP overcomes the limitations of logistic regression through:

1. **Non-linear Transformations**: Activation functions in hidden layers enable the network to model non-linear relationships in the data.
2. **Hidden Layers**: These layers perform hierarchical feature extraction, allowing the model to learn more abstract and intricate patterns.

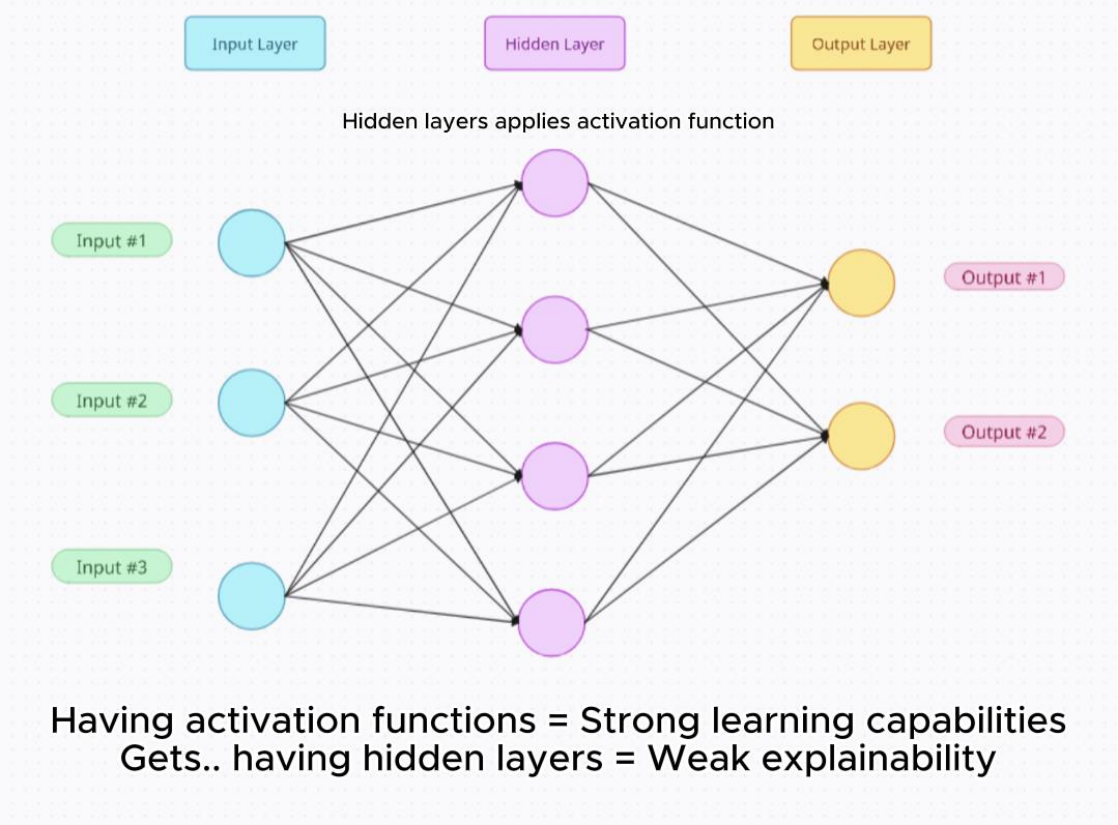

Figure 2: Example MLP Architecture

At this point, I would like to draw your attention to something: In logistic regression, the absence of activation functions prevents strong learning, and the reason it is considered interpretable is the lack of hidden layers. On the other hand, the strong learning capabilities of MLP (Multilayer Perceptron) are due to the presence of activation functions, while the reason it is considered non-interpretable is the presence of hidden layers. In other words, what we truly need are activation functions, not hidden layers. However, it is widely accepted that hidden layers are necessary for the application of activation functions. By stepping outside this conventional understanding, I have built a framework that demonstrates a learning architecture can possess strong learning capabilities while remaining interpretable, even without hidden layers and by applying activation functions directly.

2.2 – What is Potentiation Learning Artificial Neural Network ?

2.2.1 – Potentiation Learning Artificial Neural Network (PLAN)

Definition

PLAN is a supervised classification algorithm aimed at possessing both the interpretability of logistic regression and the ability to learn complex relationships akin to MLP. Unlike conventional methods, this algorithm performs the 'learning' process in a fundamentally different manner. In a classical MLP architecture, a technique called 'backpropagation' initializes weights randomly, and then the error function is minimized through 'iterative gradients'. However, in PLAN, weights are not initialized randomly. Instead, the weights corresponding to each class are formed through the accumulation of input samples representing that class. This approach is more aligned with the concept of 'Long Term Potentiation' from neuroscience, which is based on the principle that connections between recurring elements are strengthened. As a result, a single input sample, processed through various activations, is then aggregated to form a highly complex, nonlinear new input. Since this new input retains the same dimension as the original, we can still identify which input element represents which feature of the data. Furthermore, since this newly transformed input layer is directly added to the weights of the corresponding class (output), nonlinear learning is achieved. And because the input dimension remains the same, we can easily interpret and analyze relationships between input and output by observing the connections, just as in logistic regression.

In this architecture, the choice of activation functions and the number of transformations is crucial. Therefore, the activation functions and their combinations are determined through a learning process that experiments with different options based on a specific metric (not necessarily the loss function, but it could even be accuracy). The goal is to iteratively explore and identify the most suitable combination for the given problem. I called this combination '**activation potentiation'**.

Mathematical Representation:

$$\boldsymbol{Activation}(\boldsymbol{x_{train}}) \leftarrow \boldsymbol{x_{train}}$$

$$w \mathrel{+}= \boldsymbol{y_{train}^{T}} \cdot \boldsymbol{x_{train}}$$

$$\boldsymbol{Accuracy} = \frac{\sum_{i=1}^{N} \mathbf{1}(\boldsymbol{y_{pred}} = \boldsymbol{y}_{train})}{\boldsymbol{N}}$$

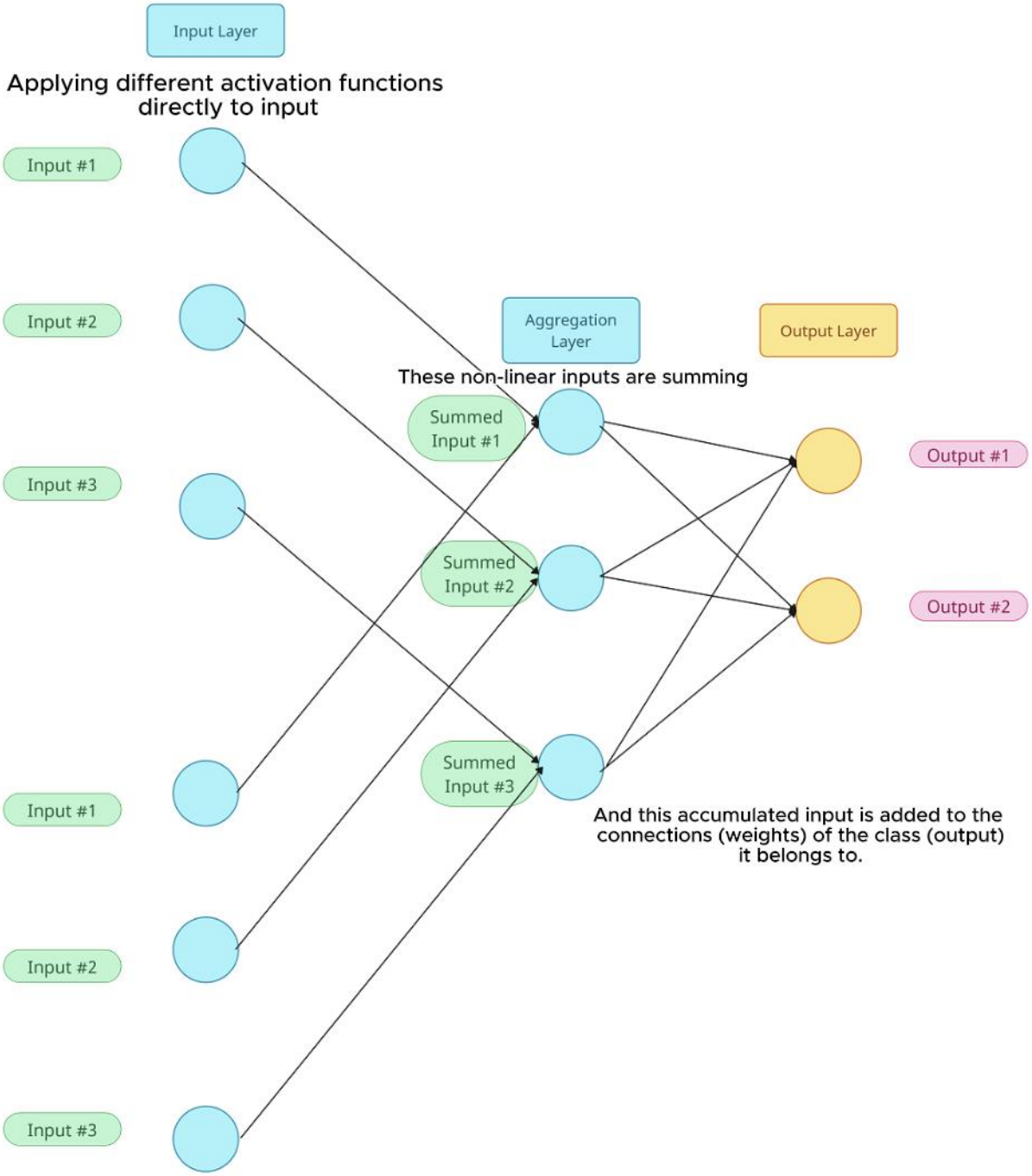

Figure 3: Example PLAN Architecture

2.2.2 – Optimization of PLAN

We are not trying to explore all these activation combinations one by one. The backpropagation optimization algorithm cannot be used in PLAN because, in PLAN, the weights are determined by a fixed rule in which the activations of inputs are accumulated according to a specific method, where each input's corresponding class connections are updated. In PLAN, the learnable parameters are not the weights but rather the combinations of activation functions. This is a discrete optimization problem. Therefore, I have modified the optimization approach by using a metaheuristic genetic algorithm, NEAT[4], to evolve and optimize the activation combinations of PLAN. I have named this approach PLANEAT.

First, we train each activation individually using PLAN's cumulative learning method, and in the end, we have as many models as the number of activations in our pool. Currently, for each activation, we have corresponding A and W. We then make predictions on the training data with these models, obtaining the accuracy values for each model. At this point, we have A, W, and accuracy values for each model. Each A and W is treated as a genome, and the entire population is passed to the PLANEAT algorithm for evolution. The genomes are divided into two groups based on their fitness performance: good and bad genomes. Based on certain mutation and crossover parameters, new activation combinations are selected through a selection process, where the best genome combines with other good genomes, replacing the bad genomes in the next generation. During this process, small mutations also occur, and weights can be involved in the crossover, though this is optional. This is because, in PLAN, the weights are already learnable based on activation combinations, but weight evolution can sometimes speed up the process. After all these steps, the new population makes predictions again, recording new accuracy values, and this process is repeated until the maximum number of generations is reached. The key point here is that, unlike the standard NEAT algorithm, the population size is determined based on the total number of activations in the pool rather than being manually set. This is because, unlike traditional NEAT, PLANEAT does not initialize weight parameters randomly. The initial population formed using PLAN's cumulative weight accumulation technique provides a strong starting point for PLANEAT, helping us overcome the performance and time limitations of classical NEAT and enabling a strong population to kick-start the process.

2.2.3 – Fitness of PLANEAT

PLANEAT optimization evolves these activation combinations by maximizing a fitness function that I refer to as WALS (Weighted Accuracy-Loss Score). This function calculates loss using categorical cross-entropy. To ensure numerical stability, a small positive value is added to both the loss and its impact. The final fitness score is then computed by multiplying accuracy with its impact and adding a term that accounts for the relationship between loss impact and loss. This formulation allows for a dynamic optimization process, balancing both accuracy and loss in an adaptive manner. Beyond the impact values, this fitness function has a unique characteristic. The **accuracy impact** and **loss impact** values are provided externally, allowing the model trainer to adjust their relative influence. When training a classification model, ensuring correct classification is the highest priority. Therefore, in the early generations, if the accuracy impact is nonzero, the function primarily focuses on maximizing accuracy. This prioritization of accuracy ensures that the model learns to classify correctly from the start. However, as the loss value becomes excessively small, the function triggers adaptive responses, aggressively minimizing loss and reinforcing further minimization. In such cases, the function adjusts its focus to further optimize the loss by giving it more weight. The accuracy and loss impact values serve to adjust the relative emphasis on each component. When **accuracy impact** is nonzero, maximizing accuracy takes priority, while the **loss impact** is given more focus once the loss is sufficiently minimized. This flexible adjustment helps guide the evolutionary process, ensuring both the correctness and efficiency of the model. In an evolutionary algorithm, this fitness function is maximized to drive optimal model performance.

Initial Evaluation of Activations:

$$\boldsymbol{Activation}(\boldsymbol{x_{train}}) \leftarrow \boldsymbol{x_{train}}$$

$$w \mathrel{+}= \boldsymbol{y_{train}^{T}} \cdot \boldsymbol{x_{train}}$$

Fitness Calculation:

$$\boldsymbol{Loss} = -\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{y}_{train} \log\left(\boldsymbol{y_{pred}}\right)$$

$$\boldsymbol{Accuracy} = \frac{\sum_{i=1}^{N} \mathbf{1}\left(\boldsymbol{y_{pred}} = \boldsymbol{y}_{train}\right)}{\boldsymbol{N}}$$

$$\boldsymbol{Fitness} = \left(\boldsymbol{accuracy} \cdot \boldsymbol{accuracy_{impact}}\right) + \left(\frac{\boldsymbol{loss_{impact}}}{\boldsymbol{loss}} \cdot \boldsymbol{loss_{impact}}\right)$$

Cross-over:

$$\boldsymbol{P}_{\text{new}} = \left(\boldsymbol{\alpha} \cdot \boldsymbol{A_a^k} + (\mathbf{1} - \boldsymbol{\alpha}) \cdot \boldsymbol{A_b^k}, \boldsymbol{w_c^{\text{new}}}\right)$$

Mutations:

$$\boldsymbol{A_k} \rightarrow \boldsymbol{A_k'},\ \boldsymbol{w_c} \rightarrow \boldsymbol{w_c} + \boldsymbol{\epsilon},\ \boldsymbol{\epsilon} \sim \boldsymbol{\mathcal{N}}\left(\mathbf{0}, \boldsymbol{\sigma^2}\right)$$

Per Generations:

$$\boldsymbol{P^{(g+1)}} = \text{Evolve}\left(\boldsymbol{P^{(g)}}\right),\ \boldsymbol{P^{*}} = \boldsymbol{argmax}_{\boldsymbol{P \in P^{(G)}}} \text{Fitness}(\boldsymbol{P})$$

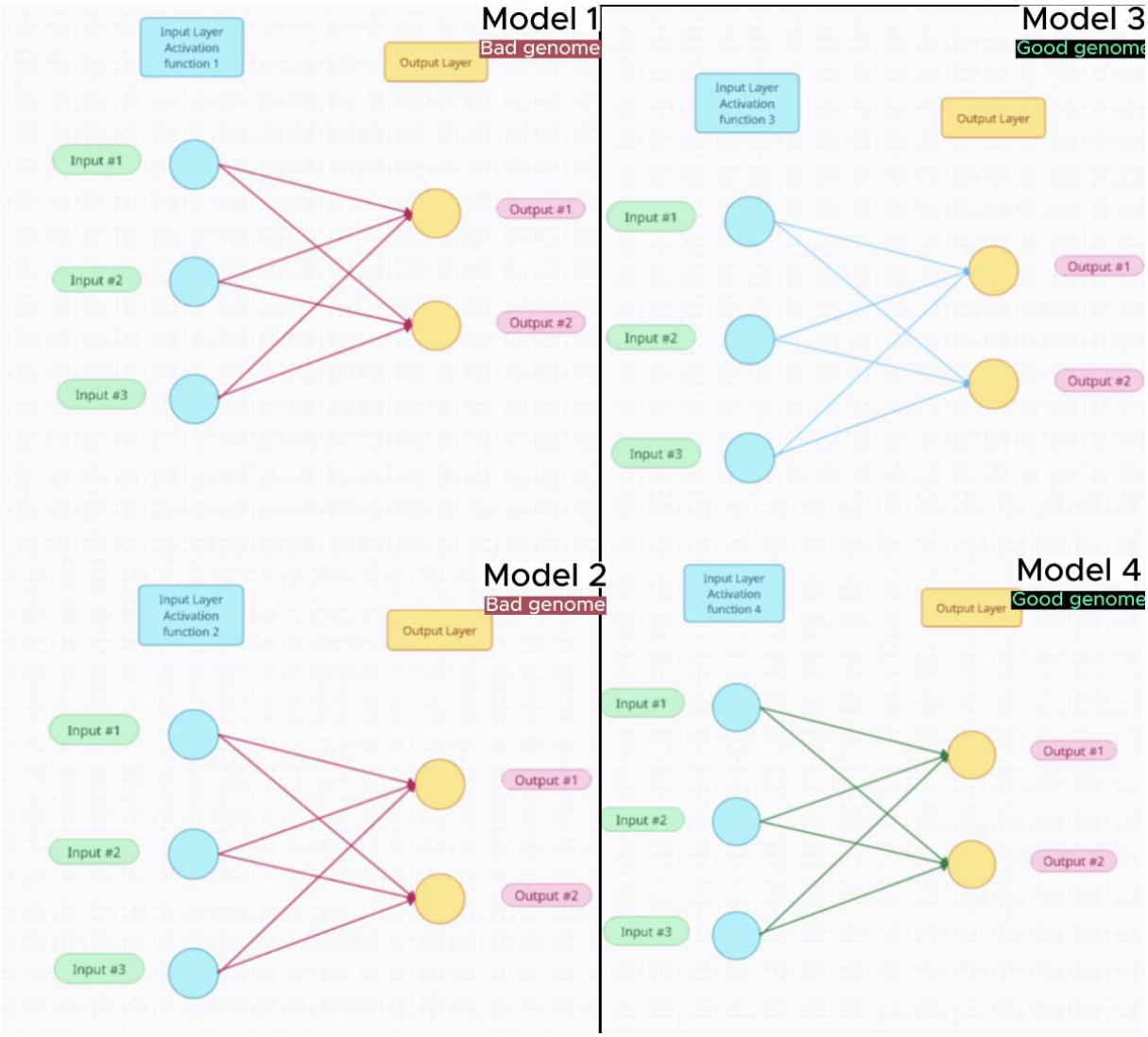

Figure 4: Before Evolution

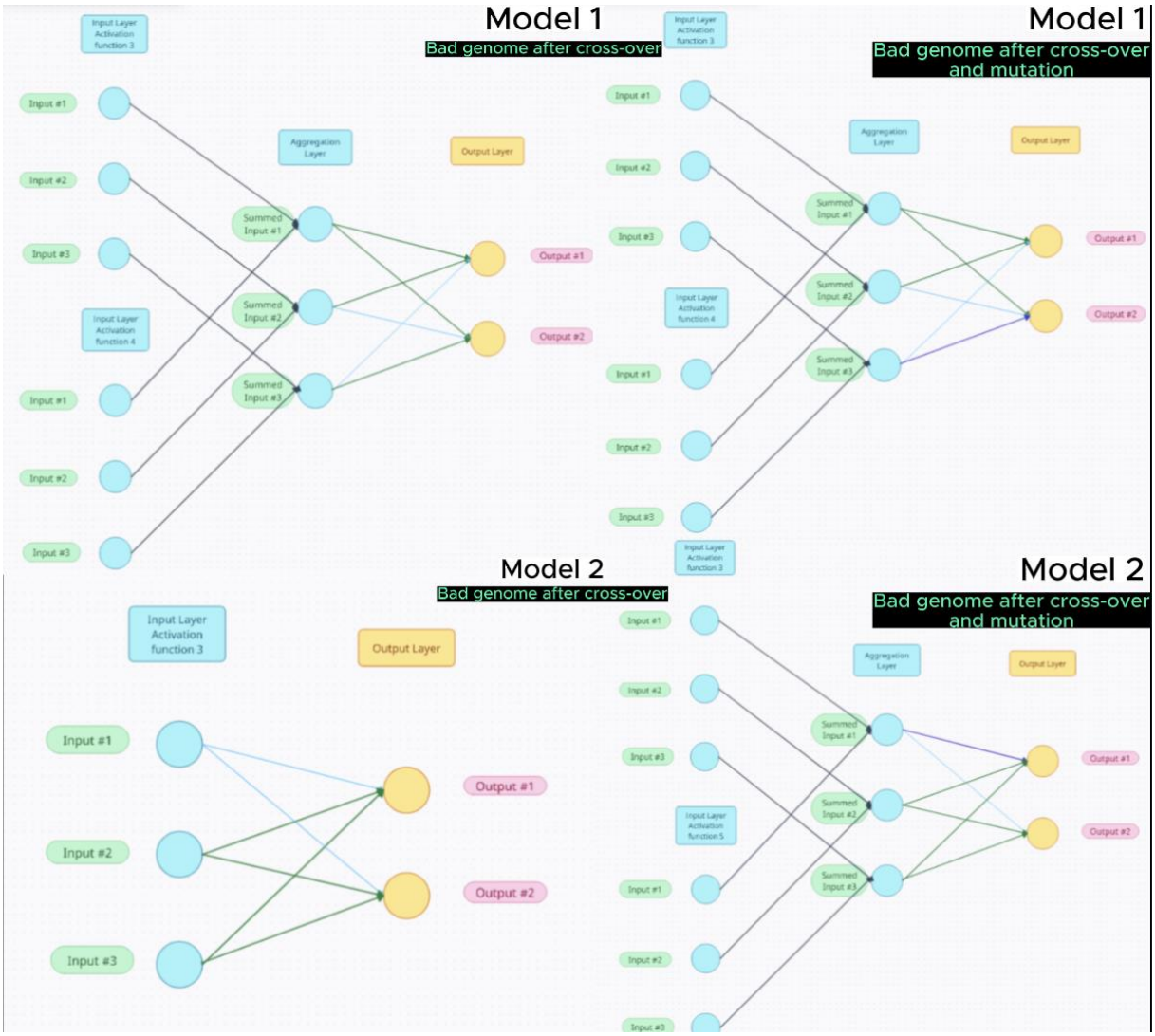

Figure 5: After Evolution

3 – Experiments & Comparisons:

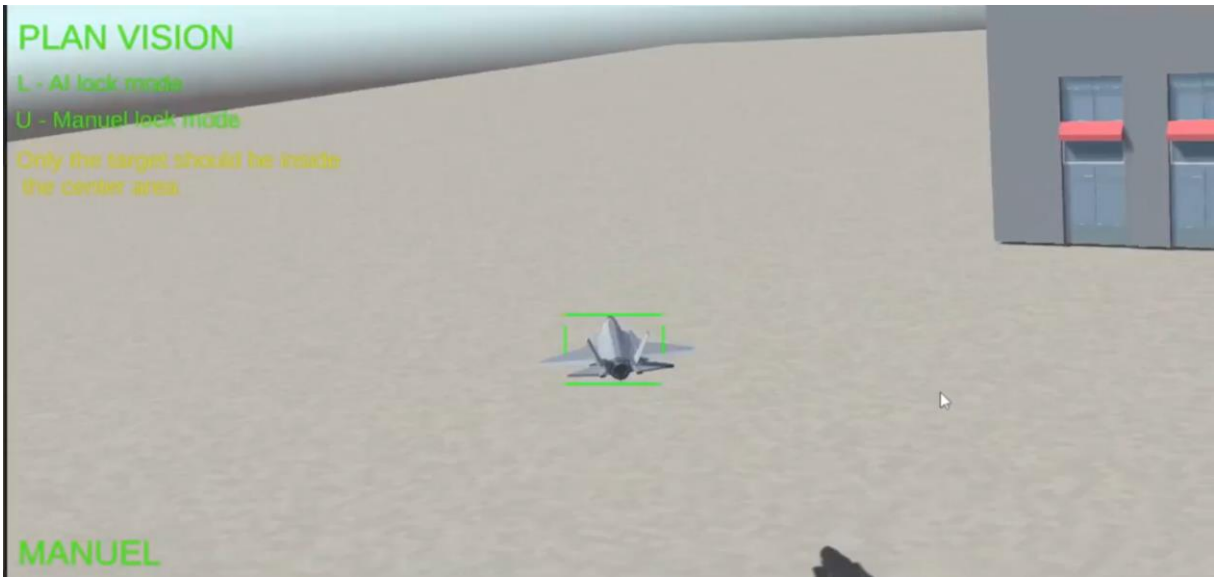

Figure 6: Performance of PLAN on target tracking

https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/ExampleCodes/PLAN%20Vision/---%20SMART%20LOCKING%20SYSTEM%20---.txt

In PLAN, neurons directly memorize what each class looks like. These memories enable the system to identify which class the input data belongs to.

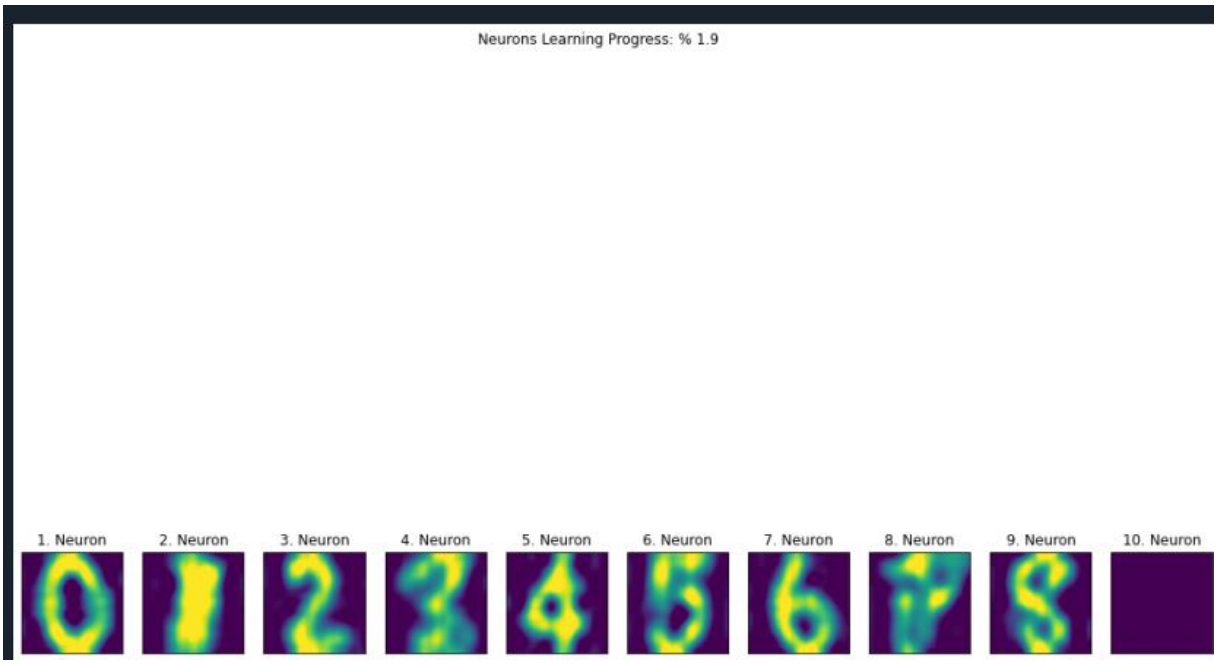

Figure 7: Performance of PLAN

Output neurons (Learning at %1.9 with digits dataset)

https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/Example Codes/PLAN%20Vision/digits(plan).py

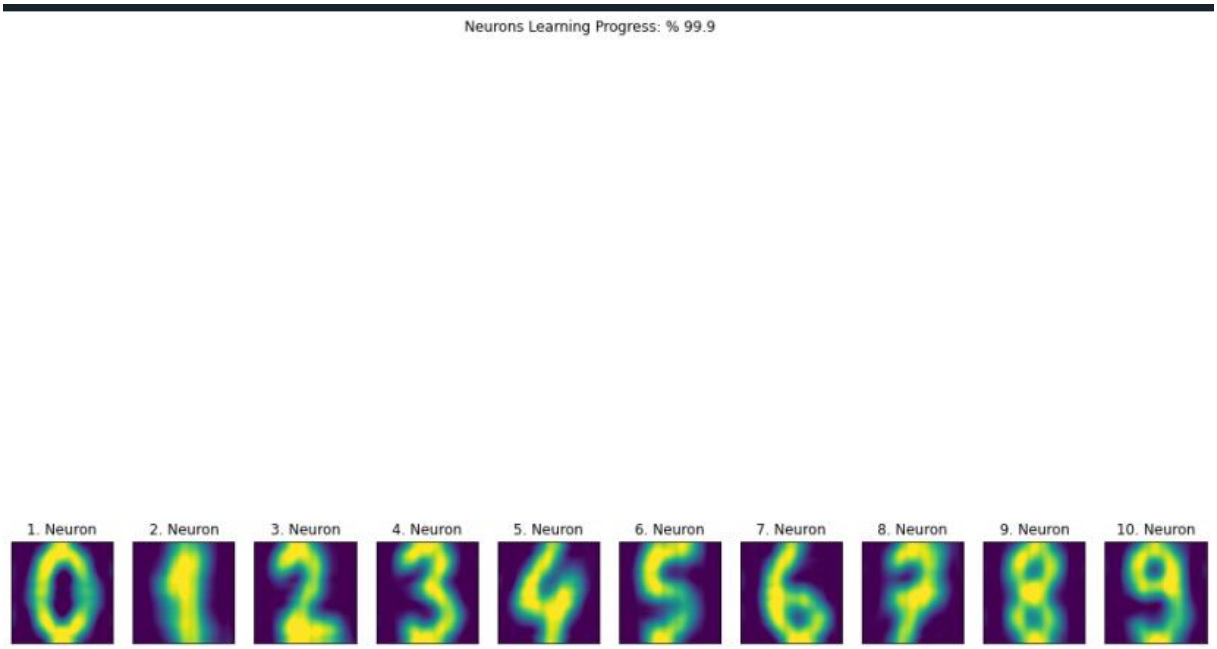

Figure 8: Performance of PLAN

Output neurons (Learning at %99.9 with digits dataset)

https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/Example Codes/PLAN%20Vision/digits(plan).py

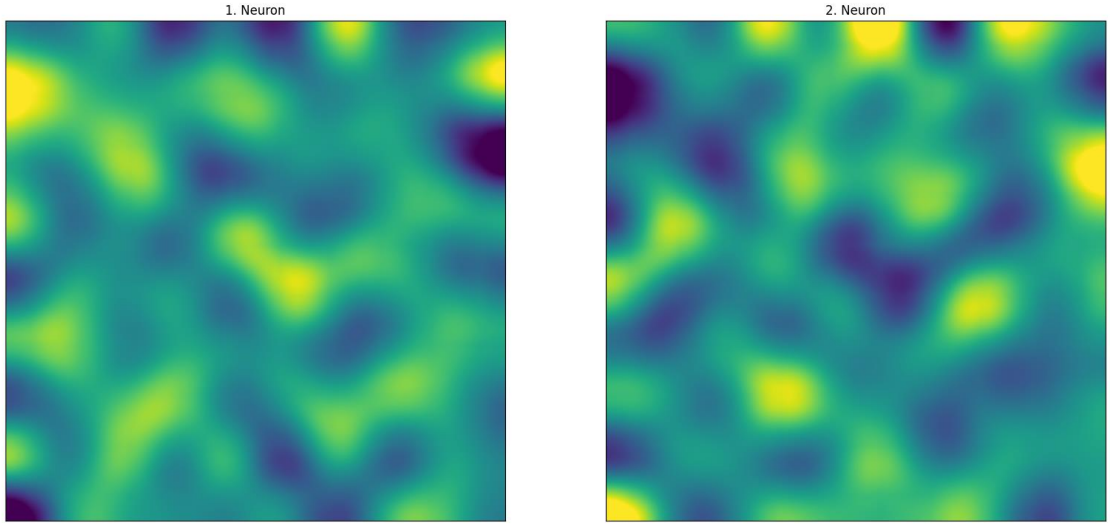

Figure 9: Performance of PLAN

***Output neurons (Learning at %100 with imdb 50k movie reviews dataset - selected feature count:* 100)**

https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/Example Codes/NLPlan/imdb.py

I conducted tests particularly on text, images, and other diverse data types, and achieving consistently above a certain standard in all of them is quite impressive. This underscores how widely applicable the Potentiation Learning Artificial Neural Network algorithm can be to a general audience.

Neurons can learn diffirences of classes:

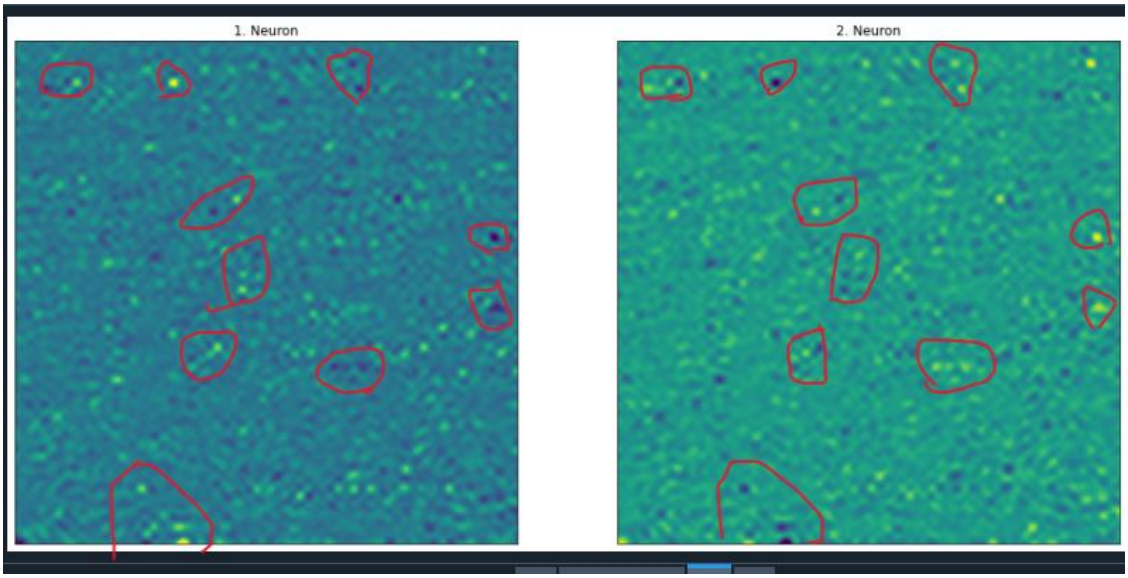

Figure 10: Performance of PLAN

Output neurons (Learning at %100 with imdb 50k movie reviews dataset - selected feature count: 6084)

https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/ExampleCodes/NLPlan/imdb.py

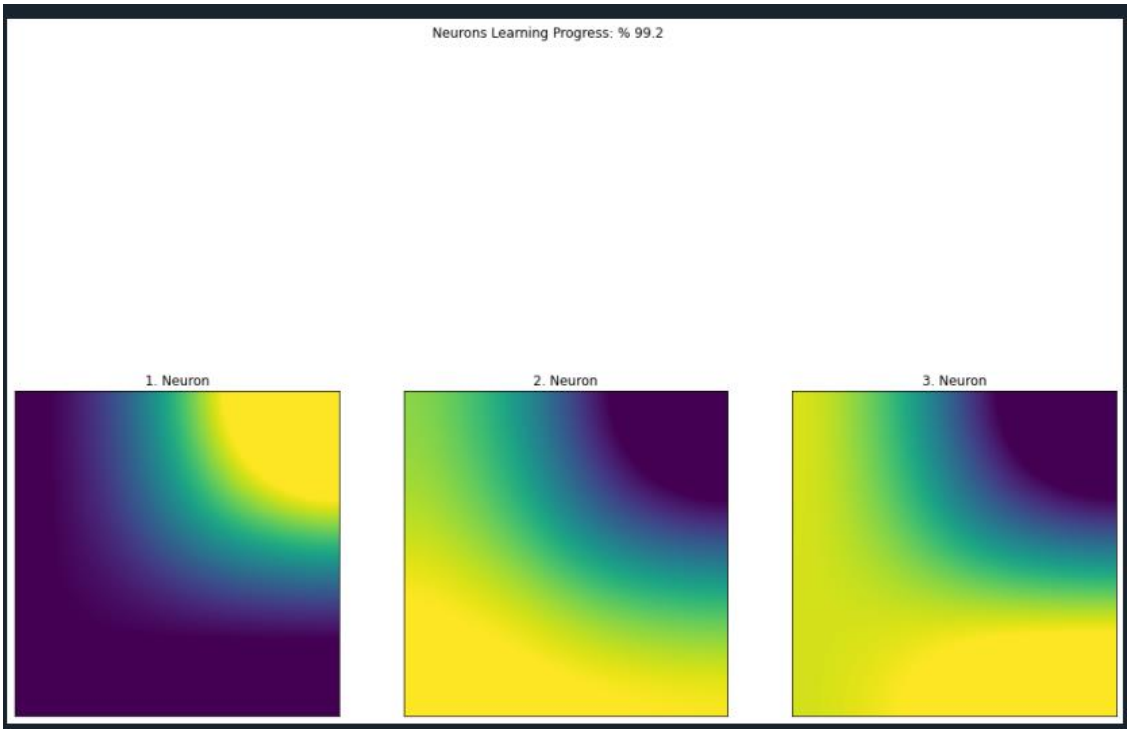

Figure 11: Performance of PLAN

Output neurons (Learning at %99.2 with iris dataset)

https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/ExampleCodes/iris(plan).py

Neurons can autonomously learn distinctive features by mimicking Long-Term Potentiation. This ability can be utilized to analyze differences between classes.

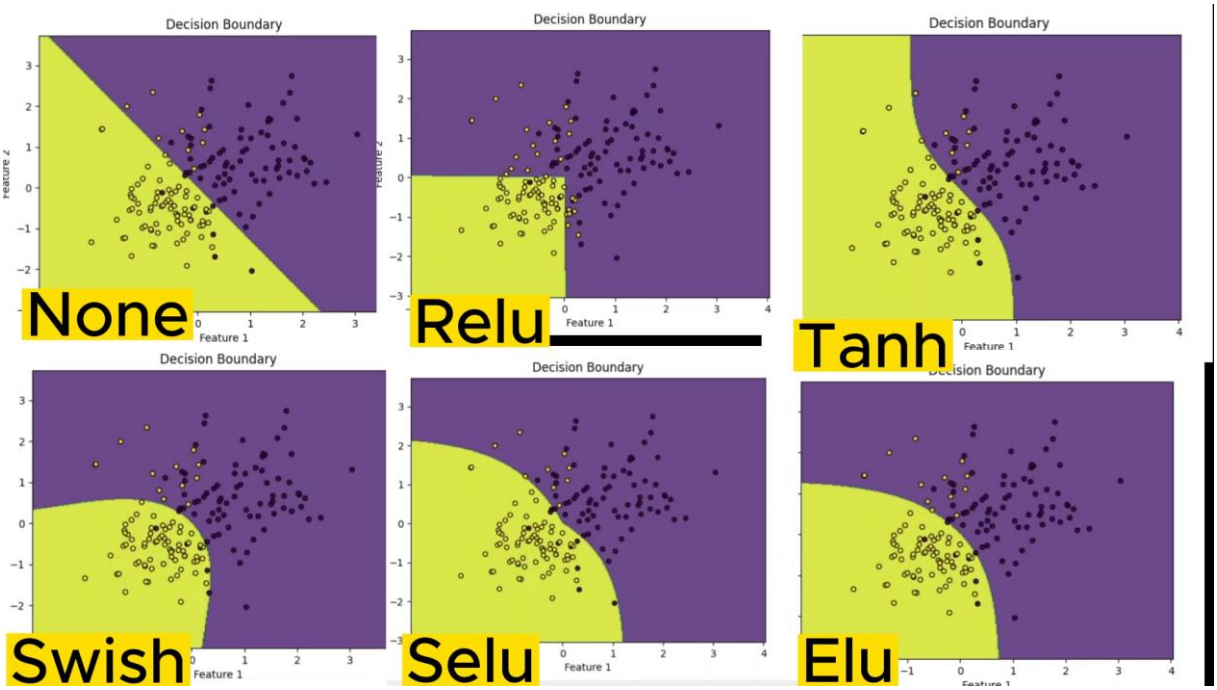

Figure 12: Decision boundaries of the model for different activation functions in aggregation layer: (None = no activation function(linear))

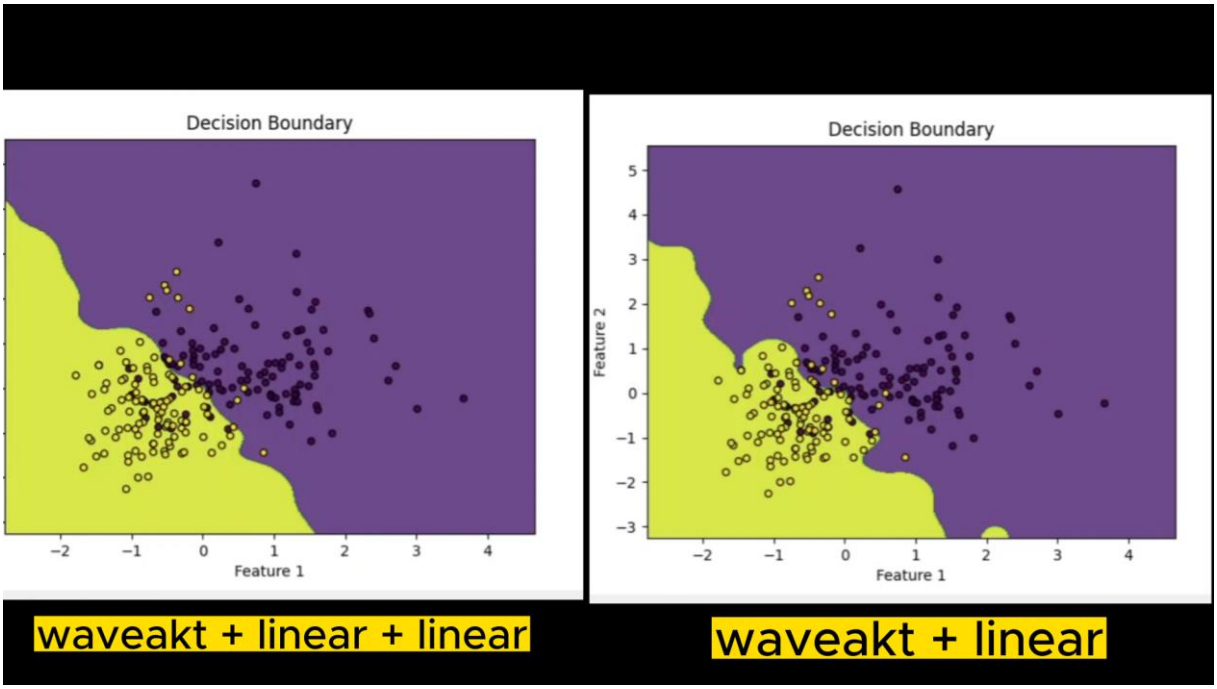

Figure 13: Activation functions can be combined to mitigate each other's weaknesses, thereby adapting to the shape of the data and uncovering er features.

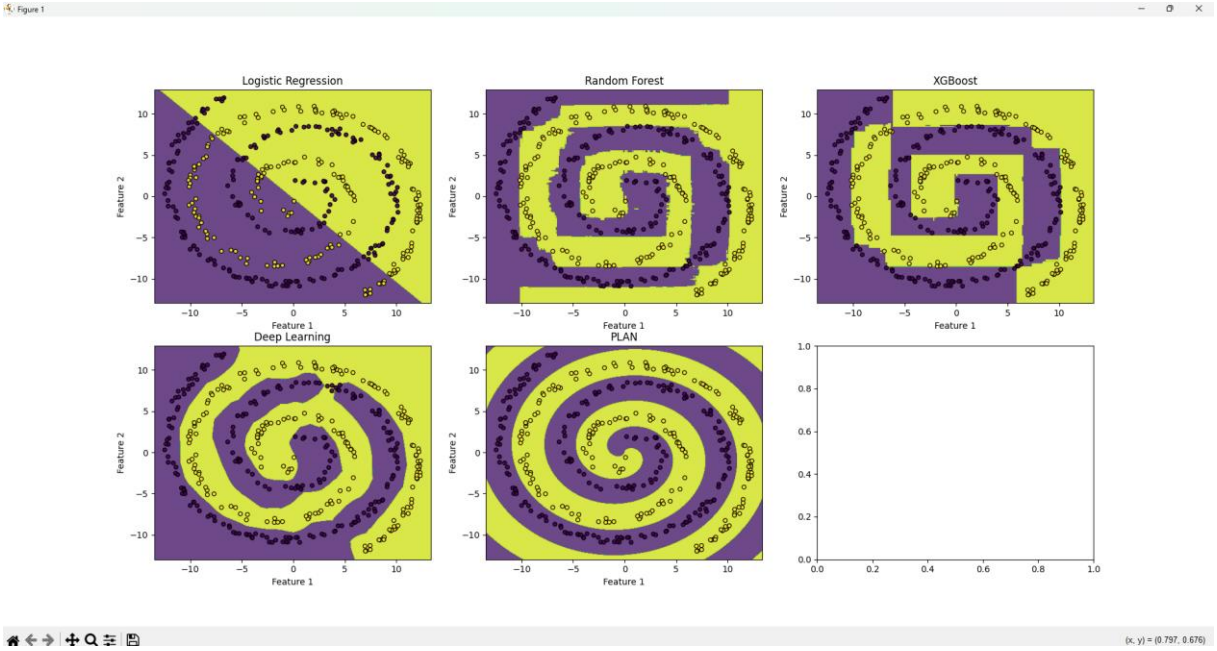

Figure 14: (PLAN's Spiral activation on Test dataset)

https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/Example Codes/spiral(all_algorithms).py

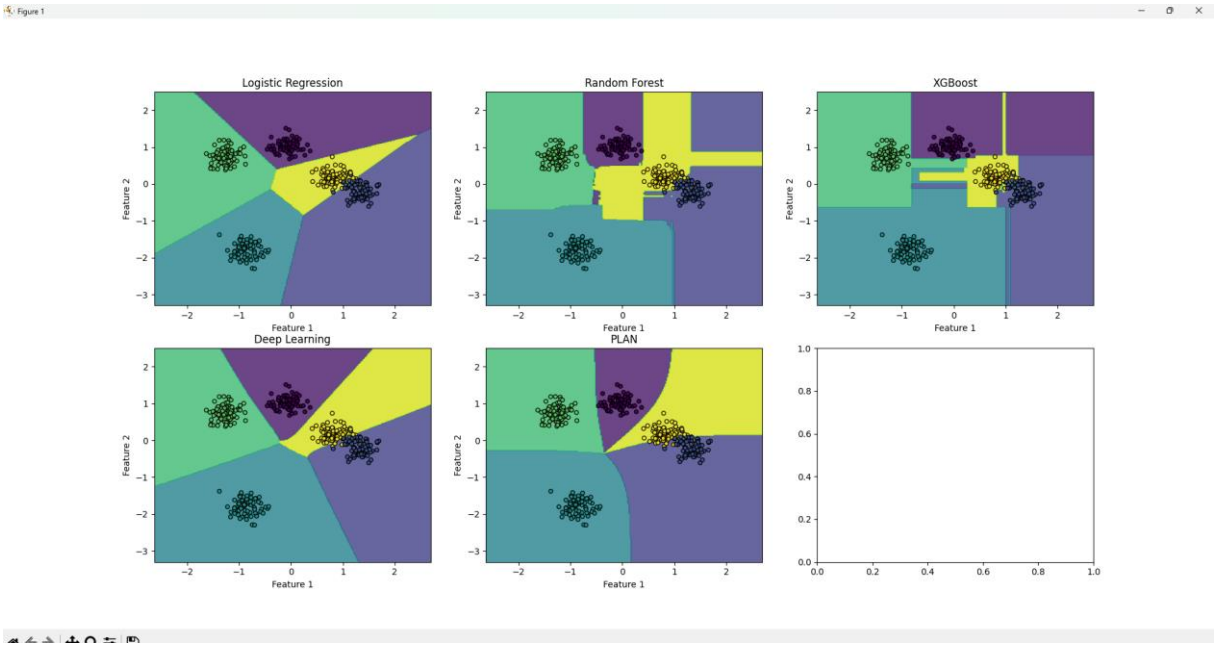

Figure 15: (Scikit-learn's blobs dataset)

https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/Example Codes/blobs(all_algorithms).py

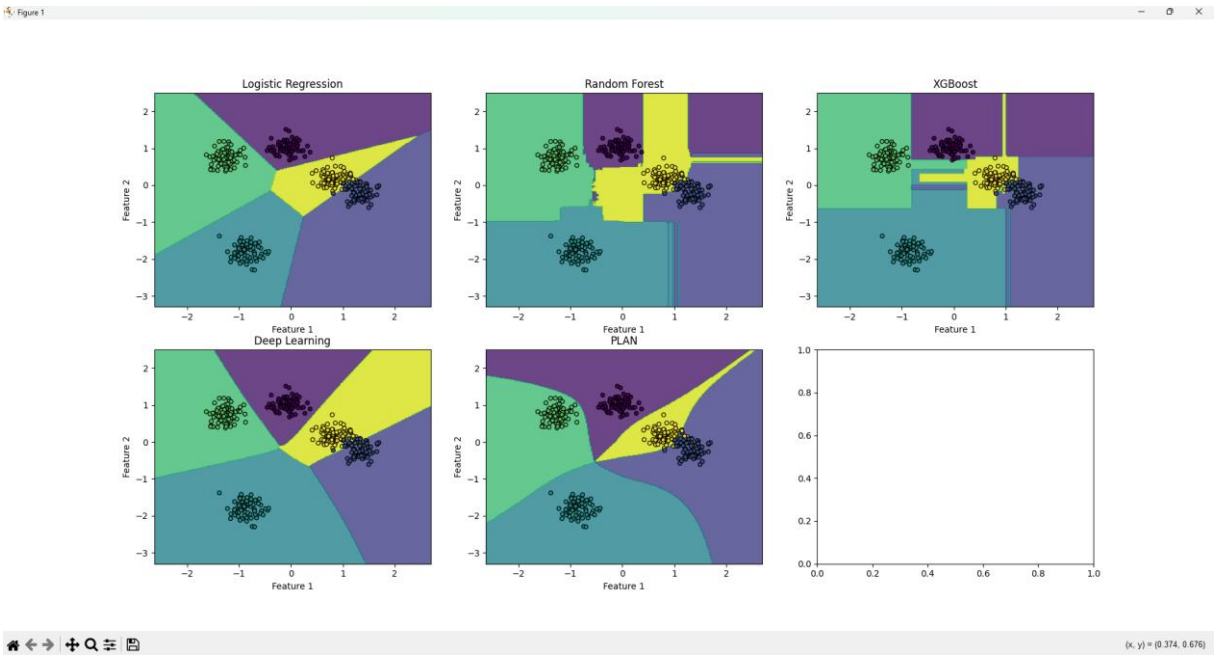

Figure 16: (Scikit-learn's blobs dataset with different activation combinations)
https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/Example Codes/blobs(all_algorithms).py

More models will be added to the PyerualJetwork Library GitHub page:
https://github.com/HCB06/PyerualJetwork/tree/main

The results of implementing the Potentiation Learning Artificial Neural Network architecture have been promising, though there are areas for improvement. While the current performance is not flawless, the Potentiation for future advancements is significant.

3.1 – Circles Dataset

In a world where obtaining clean data is challenging, it has been demonstrated that PLAN achieves better generalization with less training data compared to other classification algorithms:

For the robustness test against noise, we use the circles dataset available in scikit-learn. Based on the information in the bottom right corner of the first image, the model training results are as follows:

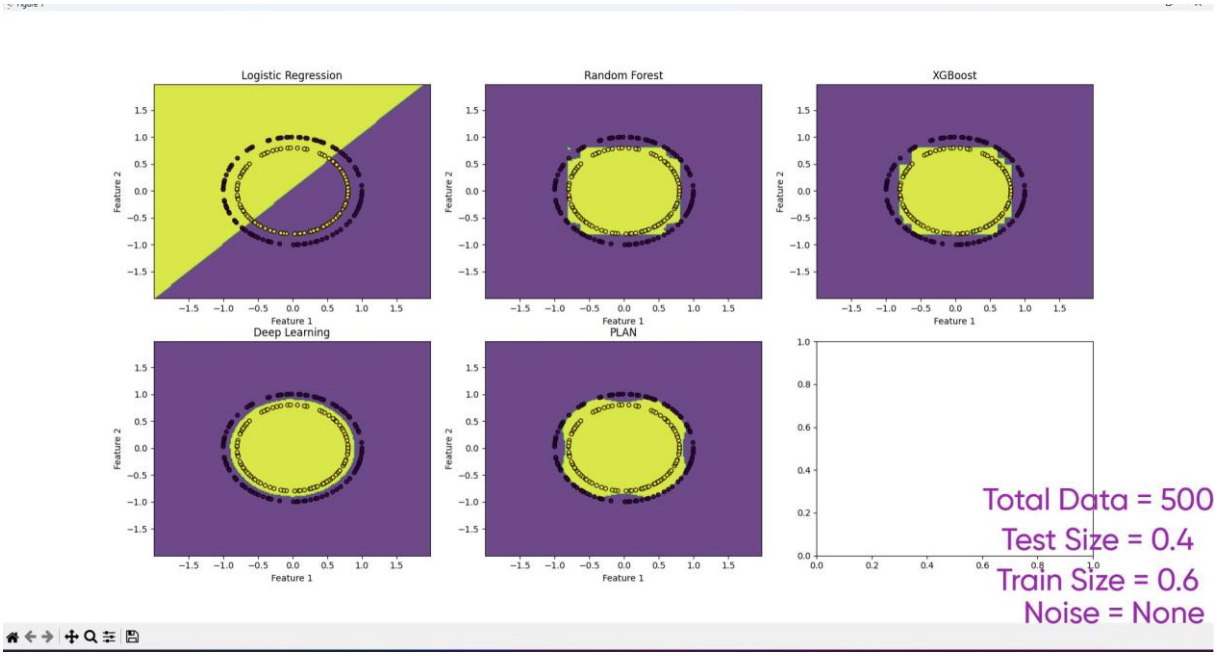

Figure 17: (Scikit-learn's circles dataset performance)

https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/Example Codes/circles(all_algorithms).py

- **Logistic Regression Test Accuracy: 0.4700**
- **Random Forest Test Accuracy: 0.9750**
- **XGBoost Test Accuracy: 0.9900**
- **Learning Test Accuracy: 1.0000**
- **PLAN Test Accuracy: 1.0000**

Both MLP and the PLAN algorithm performed well in this scenario. However, real-world data often contains complex relationships that cannot be separated as cleanly. Therefore, I introduced some noise to the second image and retrained all models without adjusting any hyperparameters. The results for the second image are as follows:

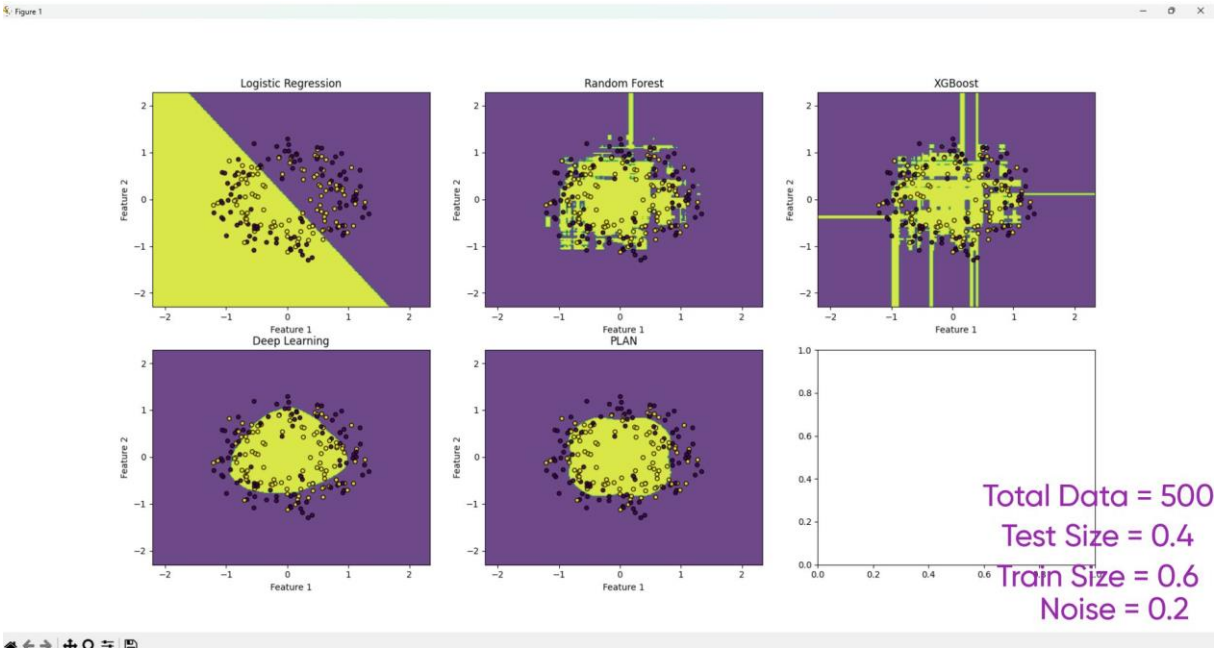

Figure 18: (Scikit-learn's circles dataset performance)

- **Logistic Regression Test Accuracy: 0.5150**
- **Random Forest Test Accuracy: 0.5800**
- **XGBoost Test Accuracy: 0.5650**
- **MLP Test Accuracy: 0.6500**
- **PLAN Test Accuracy: 0.7050**

In this case, PLAN achieved the highest accuracy with the same hyperparameters. For the third image, I increased the noise level to 0.3. The results for the third image are:

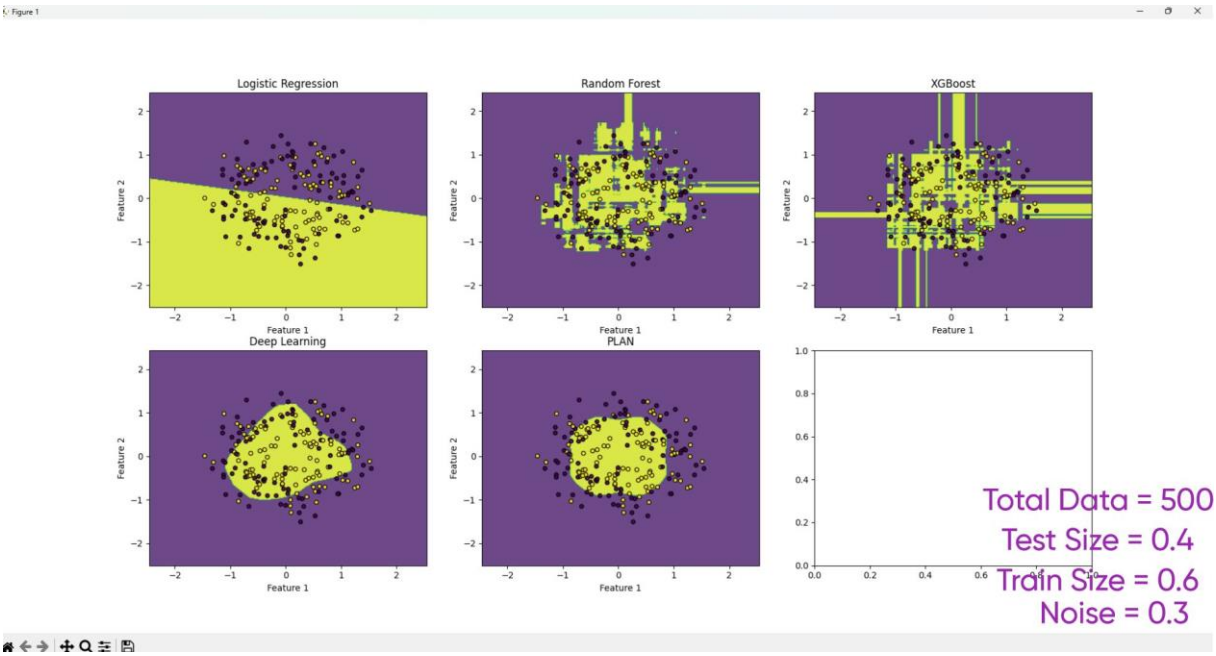

Figure 19: (Scikit-learn's circles dataset performance)

- **Logistic Regression Test Accuracy: 0.5200**
- **Random Forest Test Accuracy: 0.5350**
- **XGBoost Test Accuracy: 0.5200**
- **MLP Test Accuracy: 0.6000 - 0.6100**
- **PLAN Test Accuracy: 0.6500**

The ranking remained unchanged. The ability to adapt to noisy data is commendable. This aspect could potentially simplify the hyperparameter tuning process for model developers.

Speaking of hyperparameters, the est model I developed using the PLAN algorithm, which includes a total of 10 aggreagation layers. However, do not confuse these aggreagation layers with hidden layers in MLP. They require significantly less computational cost. They do not involve matrix-vector multiplications. Therefore, despite performance considerations, this remains the most efficient algorithm on this list. Since PLAN is a type of artificial neural network, tuning its parameters is akin to cooking; a small mistake in seasoning can significantly impact the result. A balanced list of activations, akin to proper seasoning, is essential. Removing or adding one activation can drastically alter accuracy.

In the fourth image, I reduced the total data from 500 to 50. In this scenario, all algorithms except Random Forest and PLAN were eliminated (hyperparameters remained the same as in the first image). The results are:

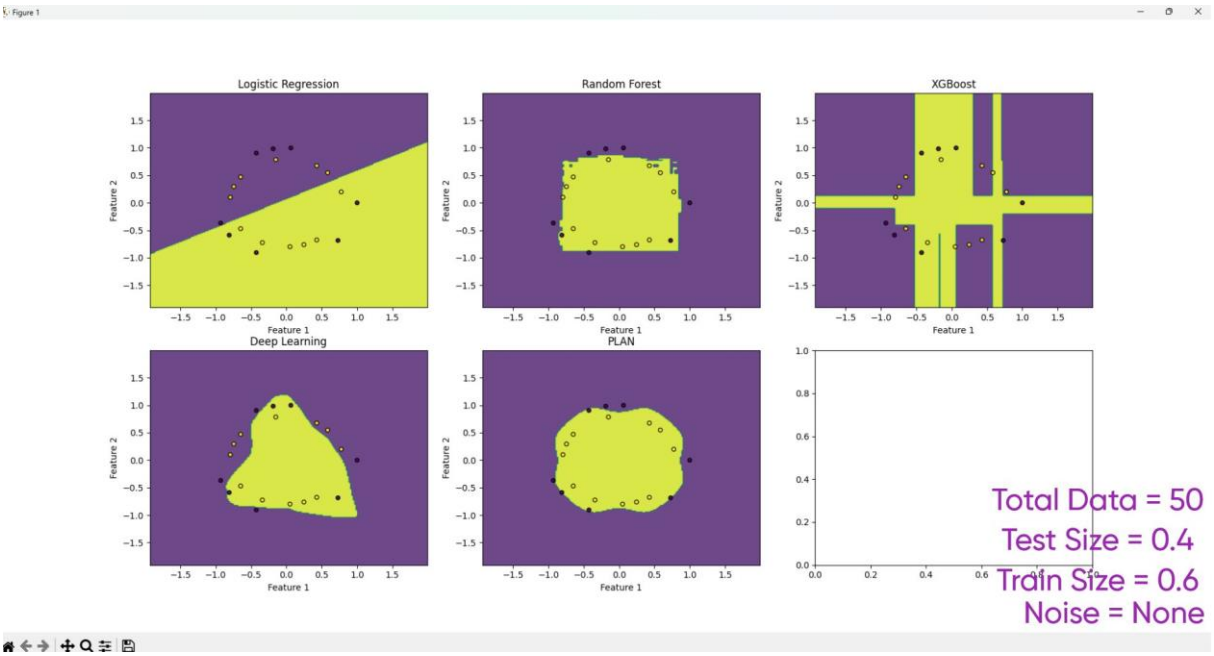

Figure 20: (Scikit-learn's circles dataset performance)

- **Logistic Regression Test Accuracy: 0.5000**
- **Random Forest Test Accuracy: 0.9000**
- **XGBoost Test Accuracy: 0.4500**
- **MLP Test Accuracy: 0.6000**
- **PLAN Test Accuracy: 0.9000**

Random Forest performed well, but it also struggled when the total data was reduced to 25. The results for the fifth image are:

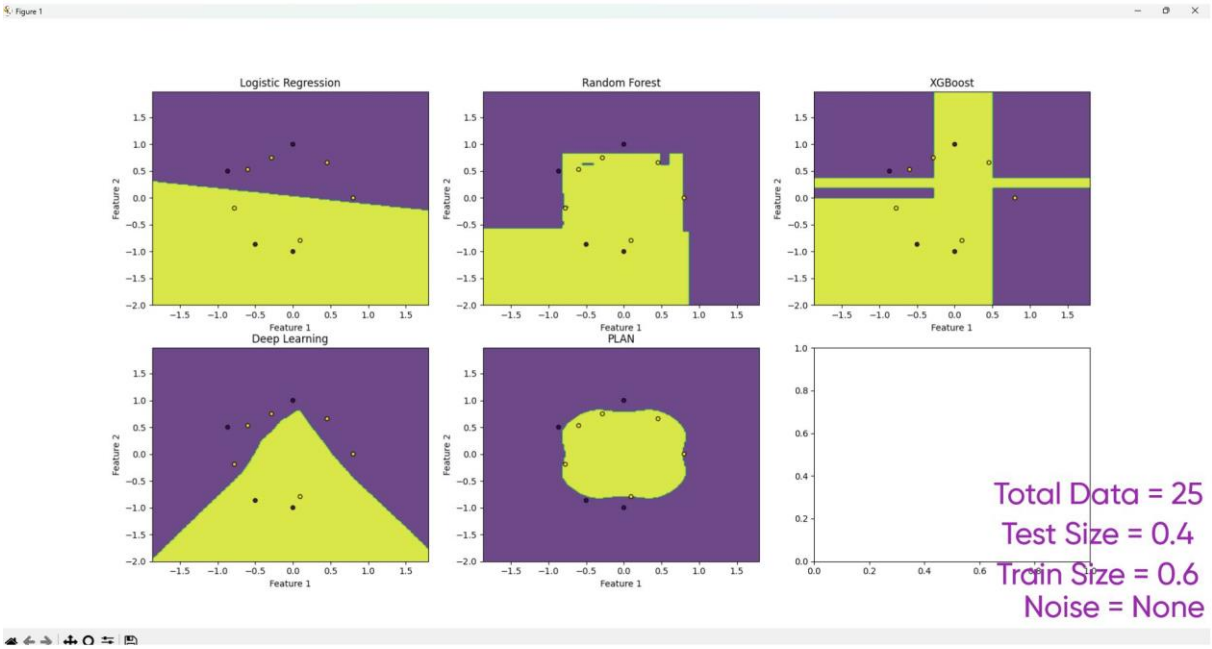

Figure 21: (Scikit-learn's circles dataset performance)

- **Logistic Regression Test Accuracy: 0.4000**
- **Random Forest Test Accuracy: 0.7000**
- **XGBoost Test Accuracy: 0.5000**
- **MLP Test Accuracy: 0.3000**
- **PLAN Test Accuracy: 0.8000**

In the sixth image, I changed the training data to 10% and the test data to 90%, with a total of 500 data points:

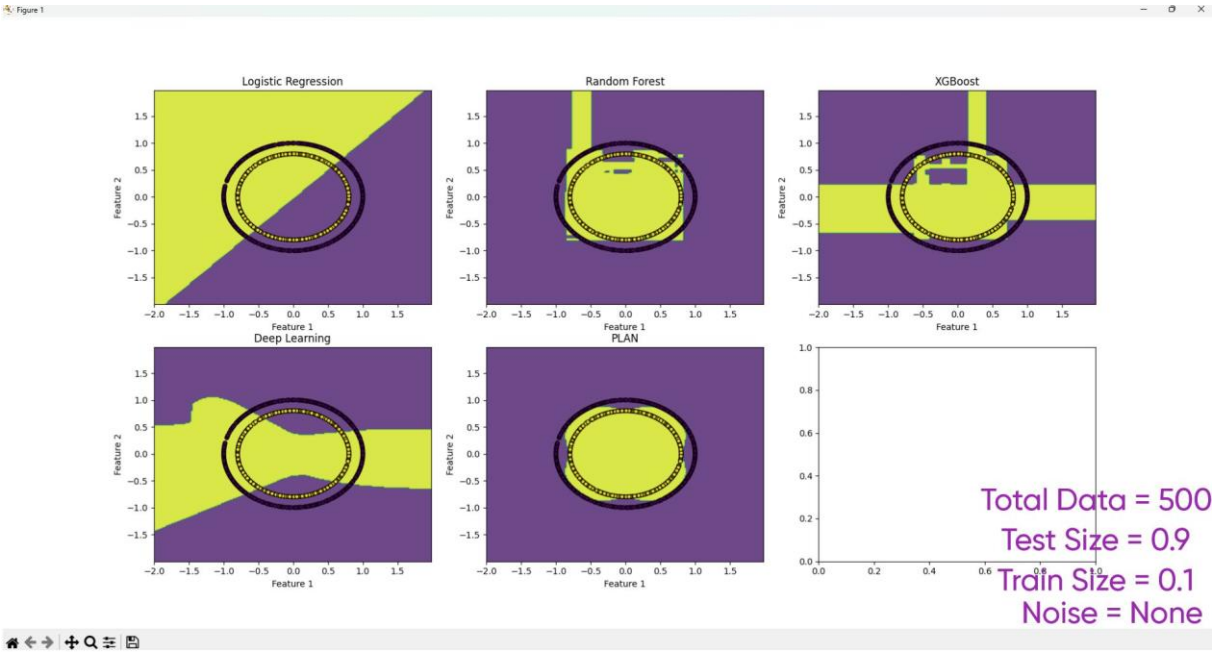

Figure 22: (Scikit-learn's circles dataset performance)

- **PLAN Test Accuracy: 1.0000**
- **MLP Test Accuracy: 0.5089**
- **Random Forest Test Accuracy: 0.8311**
- **XGBoost Test Accuracy: 0.6689**
- **Logistic Regression Test Accuracy: 0.4911**

Code:

https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/Example Codes/circles(all_algorithms).py

3.2 – Heart Diease Dataset

In this comparison, we will examine the robustness of different algorithms using the Heart Disease dataset, which is derived from real-world data.

The first criterion is the ability to produce consistent results across at least five consecutive code compilations. The hyperparameters of the models have been adjusted accordingly to ensure stable outcomes.

The second criterion is consistency. For example, a model's success with a large dataset but failure with a smaller dataset is considered inconsistent.

Using the Heart Disease dataset, I trained nearly all classification algorithms (Logistic Regression, Random Forest, XGBoost, MLP, and PLAN) with the same training and test data in a single script. After a lengthy optimization process, the results are as follows:

- **Logistic Regression Test Accuracy: 0.8833**
- **Random Forest Test Accuracy: 0.9000**
- **XGBoost Test Accuracy: 0.8833**
- **MLP Test Accuracy: 0.9000**
- **PLAN Test Accuracy: 0.9000**

Code:

https://github.com/HCB06/PyerualJetwork/blob/main/Welcome_to_PyerualJetwork/Example Codes/heart_disease(all_algorithms).py

When reducing the training data and increasing the test data without altering any hyperparameters, the results are:

- **Logistic Regression Test Accuracy: 0.7353**
- **Random Forest Test Accuracy: 0.7757**
- **XGBoost Test Accuracy: 0.7647**
- **MLP Test Accuracy: 0.4596**
- **PLAN Test Accuracy: 0.8235**

The dramatic drop in MLP accuracy occurs with a moderately complex model, which meets the first criterion by consistently yielding a value of 0.4596. However, this result indicates inconsistency, which is undesirable.

To address this, I simplified the model to have only one hidden layer with 100 neurons. The results are now between 0.79 and 0.81, indicating a model training process based on exact matching. However, stability is lost in this case. With the full dataset, the accuracy is 0.88.

PLAN models are significantly more robust compared to MLP models.

Additionally, even when adding more layers to Learning, the model does not robustly exceed the 0.90 accuracy threshold with the full dataset. No matter how many layers or neurons are added, the accuracy remains capped at 0.90.

When selecting a model architecture, considering that the model will continuously interact with users and that computational costs can increase exponentially, even someone with limited experience in machine learning can infer which architecture is more robust by examining the results. PLAN allows you to achieve the highest results with whatever data you have, without manual adjustments.

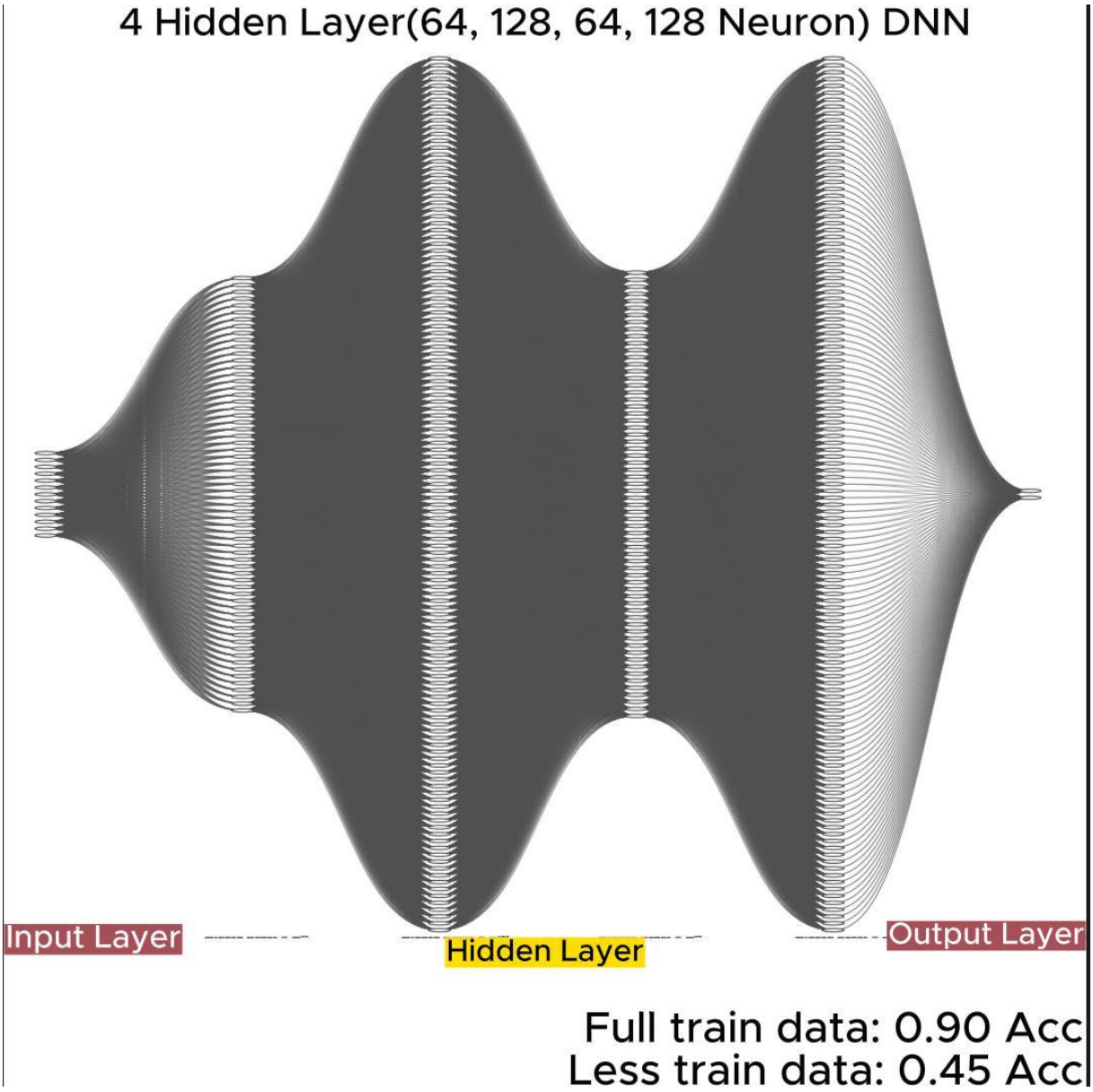

Figure 23: (Deep Neural Network(DNN) in Heart Disease dataset)

Furthermore, the PLAN architecture offers a significant advantage in model updating (adding new data). Instead of creating and training a new architecture with entirely new layers, its inherent robustness means this is not necessary.

For other comparisons:
https://github.com/HCB06/PyerualJetwork/tree/main/Welcome_to_PyerualJetwork/ExampleCodes

4 – Extra Comments

Multi-functional memories

I previously mentioned that in PLAN, neurons directly memorize what each class looks like. This gives us not only an abstract representation, such as 'Class 0' and 'Class 1,' but also a model that provides what Class 0 and Class 1 actually look like. This is a feature not found in other classification algorithms. While we typically train these models solely for classification, the fact that they also reveal what each class looks like demonstrates their utility for **data analysis.** Furthermore, the information stored in the weight matrix can be retrieved later for other purposes, allowing the model to serve multiple functions. For example, for the class representing photos labeled '1,' if we input an image of a '1,' not only will it predict '1,' but it can also be used to complete missing parts, adding additional functionality to the model. This enables a 'kill two birds with one stone' approach by embedding multiple capabilities into a single model. **For instance, PLAN models trained for classification can easily be operated in reverse. Instead of predicting the output given the input, they can estimate the input given the output. This is achievable with a very simple modification to the standard operation.**

For class predicting: weight * input layer

For input prediction: output layer * weight

```
weight matrix: [[0.5,0.7,0.2]   *   output: [0.2,0.9]   =   input: [0.37,
               [0.3,0.6,0.8]]                                        0.68,
                                                                     0.76]
```

```
0.37 ≈ 0.3
0.68 ≈ 0.6
0.76 ≈ 0.8
```

```
This is achievable in PLAN because, in PLAN, each row in the weight matrix
represents a class.
```

This is because, in PLAN, learning is achieved through the cumulative combination of features, forming the connections. In other words, the numbers in a row of the weight matrix represent the features influencing the class associated with that row. This structure allows us to feed probabilities into the output layer and predict the input with a straightforward operation. This reverse operation can be performed effectively in tasks like image and text classification and can be adapted for more specific tabular data by adding or subtracting certain bias values to the predicted input as needed.

Greater parallelism compared to MLPs

Compared to MLPs, greater parallelism can be achieved despite the fact that in state-of-the-art MLP designs, matrix-vector multiplications can be parallelized. This is due to the horizontal depth concept (depth with hidden layers), where the output of the previous layer must be awaited before the computations for the next layer can begin, making it impossible to initiate the computation of all layers simultaneously. However, in the PLAN algorithm, depth follows a vertical concept, meaning that the activation transformations applied to the input occur independently of each other. This allows all transformations to be initiated simultaneously, regardless of their number.

Evading local minimums (or maximums-Accuracy)

Another advantage is its near impossibility of getting stuck in local minima (or perhaps maxima would be a more appropriate term, as PLAN does not rely on an error function—errors are not minimized; instead, accuracy is maximized). Unlike backpropagation, PLAN is a more stochastic algorithm. Its stochastic nature stems from the fact that, unlike backpropagation, it does not follow a fixed gradient. This characteristic plays a pivotal role in discovering more creative solutions and potentially identifying different solution paths with each run.

Potentiation Learning Artificial Neural Network Library: PyerualJetwork

To facilitate broader experimentation and practical application, I have encapsulated the Potentiation Learning Artificial Neural Network architecture in a Python library called "PyerualJetwork." This library is now available for both experimental and commercial use, allowing researchers and developers to easily integrate Potentiation Learning Artificial Neural Network into their projects. You can access the "plan" module within this library for various applications. **The PyerualJetwork library made an impressive debut, garnering over 100.000 downloads in just 7 months. This clearly highlights people's interest in the PLAN algorithm and the industry's need for it.**

For those interested in exploring or utilizing this framework, the library and algorithm are hosted on GitHub. You can find them at: *GitHub - PyerualJetwork*. Additionally, for practical demonstrations and tutorials on how to use the Potentiation Learning Artificial Neural Network architecture, please visit my YouTube channel: *YouTube - Hasan Can Beydili*.

5 - Advantages and Disadvantages:

Advantages:

1. Easy to learn and implement. (non-calculus)
2. Easy to maintain and update. (non-blackbox)
3. It occupies less memory space. (only can have one weight matrix)
4. Multi-functional single model architecture. (cumulative learning)
5. Strong generalization even small training splits.
6. It meets the potential to be as explainable as logistic regression and as powerful as multilayer artificial neural networks.
7. Greater parallelism compared to MLPs
8. Evades local minimum and maximum compared to MLPs Backpropagation

Disadvantages:

1. In early research stage.
2. The duration of training some cases long.
3. Slightly diffucult optimization algorithm.

6 - Conclusion:

The Potentiation Learning Artificial Neural Network architecture, through its innovative approach to neural network design, offers a compelling alternative to traditional methods. It is more interpretable, and strong learning capabilities like MLP's makes it a significant development in the AI and XAI landscape. The availability of the Potentiation Learning Artificial Neural Network library on GitHub further encourages exploration and adoption of this architecture, paving the way for new applications and advancements in artificial intelligence.

Codes in this article:

https://github.com/HCB06/PyerualJetwork/tree/main/Welcome_to_PyerualJetwork/ExampleCodes

Potentiation Learning Artificial Neural Network library:

https://github.com/HCB06/PyerualJetwork

References

[1] AKM Bahalul Haque, A.K.M. Najmul Islam, Patrick Mikalef 2022. Explainable Artificial Intelligence (XAI) from a user perspective- A synthesis of prior literature and problematizing avenues for future research

[2] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier

[3] Bas H.M. van der Velden∗, Hugo J. Kuijf, Kenneth G.A. Gilhuijs, Max A. Viergever 2022. Explainable artificial intelligence (XAI) in deep learning-based medical image analysis

[4] Kenneth O. Stanley, Risto Miikkulainen 2002. Evolving Neural Networks through Augmenting Topologies