Potentiation Learning Artificial Neural Network

Author: Hasan Can Beydili

tchasancan@gmail.com

Abstract

No gradient descent, no backpropagation, no error functions, no calculus, no epoch, no bias, no hidden layers and no matrix multiplications for training. Yet, faster training and promising results.

Allow me to introduce Potentiation Learning Artificial Neural Network, in this research, I have developed an architecture that holds a high potential for solving the energy consumption problem in artificial intelligence. A new artificial neural network architecture designed to address classification problems typically tackled by conventional Multi Layer Perceptron's. The Potentiation Learning Artificial Neural Network algorithm performs learning in an 'cumulative' rather than 'calculative' manner.

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

Introduction

Imagine a neural network architecture that eliminates the need for gradient calculation procedures inherent in the classic backpropagation algorithm and bypasses all the repetitive iterative processes. This new approach offers a faster and more comprehensible alternative. Additionally, it redefines the concept of the 'hidden layer' with a distinctive design that approaches this traditionally opaque area in novel ways.

In this paper, I introduce Potentiation Learning Artificial Neural Network, a neural network architecture that embodies these principles. Throughout this article, I will provide empirical evidence and examples from various projects to substantiate these claims. Our goal is to significantly reduce training times and develop models that are not only efficient but also compact in size.

Synaptic Strengthening in Potentiation Learning Artificial Neural Network:

In biological systems, Long-Term Potentiation is characterized by the persistent strengthening of synaptic connections following high-frequency stimulation. This process involves intricate biochemical pathways that lead to long-lasting changes in synaptic efficacy.

The Potentiation Learning Artificial Neural Network's emulates this by enhancing the connections between its computational units when they are frequently activated together. This synaptic-like strengthening allows the network to robustly encode patterns in the data, akin to how Long-Term Potentiation solidifies memory traces in the brain.

Activity-Dependent Plasticity:

Long-Term Potentiation's dependency on the activity patterns of neurons ensures that only significant and repeated stimuli lead to synaptic strengthening. This selective process is essential for efficient memory formation and recall.

Potentiation Learning Artificial Neural Network's employs a similar strategy, where the strengthening of connections is driven by the activity levels of its nodes. This means that frequently co-activated nodes within the network reinforce their connections, enabling Potentiation Learning Artificial Neural Network to selectively learn and retain critical information, much like how Long-Term Potentiation prioritizes important neural signals.

Dynamic Learning and Adaptation:

In biological systems, Long-Term Potentiation allows for the dynamic balancing of memory retention and plasticity, providing a mechanism for long-term stability while still allowing for adaptive learning.

Potentiation Learning Artificial Neural Network’s mirrors this balance by dynamically adjusting the strength of its internal connections based on the learning experience. This allows the network to retain core knowledge while still being adaptable to new data inputs, effectively integrating the Long-Term Potentiation-like processes into its learning framework.

When updating a Potentiation Learning Artificial Neural Network model, instead of relying on traditional methods like retraining with old and new data as seen in other artificial neural network architectures, Potentiation Learning Artificial Neural Network’s capability for model updating offers exceptional efficiency. This process leverages the ability to enhance the model's performance by incorporating new data while minimizing the need for extensive retraining.

Memory Consolidation and Retrieval:

Long-Term Potentiation facilitates the consolidation of long-term memories in the brain, making it easier to retrieve stored information when similar stimuli are encountered.

Similarly, Potentiation Learning Artificial Neural Network’s mechanism of connection strengthening ensures that learned information is consolidated within its architecture. This enhanced connectivity allows Potentiation Learning Artificial Neural Network to efficiently recall and generalize from its learned experiences, enabling it to respond accurately to new inputs that are similar to previously encountered patterns.

By incorporating Long-Term Potentiation-like principles, the Potentiation Learning Artificial Neural Network’s achieves a sophisticated approach to learning and memory storage. This approach not only strengthens its ability to process and store information but also aligns closely with the foundational principles observed in biological neural systems.

Furthermore, in addition to all these, the algorithm is incredibly fast due to not using matrix multiplication for training, no loops and no calculus anymore. It just use basic level linear algebra.

This article begins with an introduction to the architecture and proceeds to discuss the underlying concepts and mathematics of the Potentiation Learning Artificial Neural Network algorithm.

All written, visual, and conceptual work in this article belongs to the author.

All the codes in this article:
https://github.com/HCB06/Anaplan/tree/main/Welcome_to_PLAN/Codes

The Potentiaton Learning Artificial Neural Network (PLAN) is bifurcated into two distinct branches: the first is the " **Potentiaton Learning Artificial Neural Network**," which is capable of **linear separability**, and the second is the "**Deep Potentiaton Learning Artificial Neural Network**" designed to handle **non-linear separability**. In the initial section of this work, I will detail the training, mathematical formulation, testing, and evaluation of the basic model, supported by textual and visual representations. Subsequently, I will explore the more sophisticated architecture of the "Deep Potentiaton Learning Artificial Neural Network," which demonstrates enhanced capabilities in handling complex, non-linear data distributions.

A. Potentiation Learning Artificial Neural Network Architecture:

Potentiaton Learning Artificial Neural Network enables nearly perfect linear separations without requiring any hyperparameters.

2 - Concepts:

Potentiation Layer:

The Potentiation Layer is output layer of Potentiation Learning Artificial Neural Network architecture. All features directly given to Potentiation Layer’s connections, this will allow features that occur more frequently in the connections to be summed later and represented by larger numbers in memory, effectively achieving the generalization process on its own.

Activation Potentiation:

Activation Potentiation is a list containing string expressions that specify the activation functions for Deep Potentiation Learning Artificial Neural Networks. (This will be discussed in more detail in the section on Deep Potentiation Learning Artificial Neural Networks.)

Long-Term Depression (LTD):

Long-Term Depression (LTD) is a process that plays a crucial role in the memory and learning mechanisms of biological neural networks. Essentially, it induces a reduction in the strength of certain synaptic connections (weights). While excessive LTD can lead to neurological issues, when maintained at a stable level, it serves as a biological mechanism that enhances learning. (Deep Potentiation Learning Artificial Neural Network hyperparameter)

In Potentiation Learning Artificial Neural Network's, there are no hidden layers because the architecture is designed to operate without the traditional 'black box' approach of hidden processes. Each layer's function and transformation are transparent and well-defined. This structure simplifies the understanding and debugging of the network while maintaining high performance.

To illustrate how these components work together, let's delve into an example application where Potentiation Learning Artificial Neural Network demonstrates its unique capabilities.

---

3 – Training And Math:

Let's consider we have a handwritten digit classification dataset consisting of 28x28 pixel images, with 10 classes (from 0 to 9).

We define 10 neurons for potentiation layer, then our weight matrix should be (10,784) (10 = rows, 784 = columns). In this scenario, the weights affecting the first neuron in the potentiation layer vector resulting from the matrix-vector multiplication will be all the columns in the first row of the 10 rows. To understand, let's recall the matrix-vector multiplication; in the current scenario, it is worth noting that all elements of the input vector are multiplied by all elements in the first row of the weight matrix and summed up. The first output obtained from this result will represent the first neuron in our hidden layer. The continuation of this process to compute the other neurons means moving down to the next row in the weight matrix and repeating the multiplication and summation with the input. This brings us to the point where we can control the information influencing all neurons in the hidden layer ourselves. ***In fact, each row of the weight matrices holds the information of each neuron separately.*** So, with this technique, we govern the hidden layer and eliminate its secrecy, and we call this layer the **"potentiation layer"**. Thanks to this layer, we encode the brightness values of the pixels and their positions in the photograph we provided as input into a certain neuron of our initial weight matrix.

Configuration of Potentiation Layer in Potentiation Learning Artificial Neural Network's:

In our Potentiation Learning Artificial Neural Network, the potentiation layer is structured based on the number of classes in the task. The weight matrix weights defining this layer has:

Rows: Equal to the number of classes, representing class-specific neurons.

Columns: Equal to the number of input features, reflecting input dimensionality.

This setup ensures each class has dedicated weights for effective and efficient classification. The programming language to be used in the article is **GNU Octave**, which has MATLAB syntax.

```
weights = ones(10,784); % Matrix formed by ones


% Info: 10 is class count, 784 is feature count.
```

We define the weight for potentiation layer(weights) to be constant and equal to 1 instead of randomly generating them within a certain range, because we will never optimize weight values; we will adjust parameters instead.

It is quite simple to understand compared to other artificial neural network architectures.

Assuming we have performed the same operations for the other classes, we can now write the artificial neural network structure *during the learning process*. Let's consider that we will input the first data point (photograph) belonging to class 1:

```
inputLayer = normalization(inputLayer); % inputs in range 0 - 1

%% POTENTIATION LAYER %%

  weights(class,:) = inputLayer;


[1,0,0,0
 0,0,1,1]


% Here, the first row, representing the first neuron,
% hold the features of the first class,

% while the last row,representing the last neuron,
% hold the features of the second class.
```

Let's explain the underlying principle of the potentiation layer with a simple example.

In a situation where each feature is completely distinct from the others, the weight matrix of our potentiation layer would take a diagonal form. To illustrate this, let’s consider a simple network with three classes.

```
 [1,0,0
  0,1,0
  0,0,1]

% The column corresponding to the 1st row belongs to the 1st class.
% The column corresponding to the 2nd row belongs to the 2nd class.
% The column corresponding to the 3rd row belongs to the 3rd class.
% Emphasis.
```

The matrices constituting the potentiation potentiation layer should typically be in this “diagonal form”**. If you remember, we said that the reason for this is that the single feature coming from the input layer searches for the neuron that belongs from top to bottom. Rows represent neurons.**

```
% x = Single pixel value of photo

[x*1,0,0  % x * 1 = x
  0,1,0
  0,0,1]

 [1,0,0   % x * 0 = x
  x*0,1,0
  0,0,1]

 [1,0,0
  0,1,0
  x*0,0,1] % x * 0 = 0

% Then x pixel represents a feature belonging to the first class.
```

For a data point labeled ‘1’, only the 1st class is ‘1’, while all other classes are ‘0’. **There’s no need to even use an error function because the error is already ‘0’ and we also don’t need even softmax activation function. Softmax is completly optional.** We have now trained a data point with a ‘0’ error as quickly as possible.

So, how can we do this for other data samples? Let’s continue reading to find out.

Training a data sample belonging to a new class and training data samples belonging to the same class:

As I mentioned in the abstract of the paper, Potentiation Learning Artificial Neural Network algorithm performs learning in an 'cumulative' rather than 'calculative' manner.

After performing the same operations for data samples labeled ‘2’ belonging to the second class, we will merge the trained matrices for the first class with the trained matrices for the second class. We can accomplish this merging by adding the matrices together:

Note: In order to make it easier to understand in this paper, weights are described to be saved to and loaded from a file directory. You can implement this within the program, which will make it faster.

If we think 1 labeled data sample is our first training sample:

```
if class == 1 % [[After the initial training,
  % save the matrices to a file following the steps]]

  weights = sprintf('weights/weights.mat');
   save(weights, 'weights');


else % if [[the matrices are already saved, retrieve them and add them
together.
 % When it's a new class, we won't perform addition but merging.
 % For the same class, we'll add them together, thus reinforcing
 % existing features and adding new ones to the matrix.]]

  newWeights = weights;

  weights = sprintf('weights/weights.mat');
    load(weights);

  weights += newWeights;

  weights = sprintf('weights/weights.mat');
   save(weights, 'weights');


end
```

Actually, there is no addition operation here because we separate the rows in such a way that they do not affect each other, so the addition operation here only allows merging. What we will obtain are matrices trained for two separate classes. It would be more logical to do this for other classes as well and then merge this modular structure at the end. If we want to include a new data sample in a model trained for a specific class, what we need to do is to follow the steps above again to determine the range of rows containing the information for the class to which the data sample belongs, and then add the information from the trained matrix (since both will already have the same row range). So, it’s about combining the trained model with the model trained for a single data sample. This calculations mathematical formation is:

$$\mathbf{W}_{\text{final}} = \sum_{c=1}^{C} \mathbf{W}'_c \times N_c$$

```
%% FOR OTHER CLASSES %%

  % class1_OldWeights = [0,0,0,0
                         0,0,1,1]


          +                 they are merging. = [1,1,0,0
                                                  0,0,1,1]

  % class2_NewWeights = [1,1,0,0
                         0,0,0,0]

%% FOR SAME CLASSES %%


  % class1_OldWeights = [0,0,0,0
                         0,0,1,1]


          +                  they are summing. = [0,0,0,0
                                                   0,0,2,2]

  % class1_NewWeights = [0,0,0,0
                         0,0,1,1]
```

At the beginning of the text, we were talking about Long-Term Potentiation. When training new data belonging to the same class, we strengthen the connections, meaning we add +feature to the values of the pixels in that class's rows. In the potentiation layer, for indices with the same pattern, we add +feature, while for indices with a new pattern. In other words, we are aggregating the model trained for a single data point with the model trained for multiple data points. This underscores the importance of having equally distributed data when preparing data for similar classes. It is important to gather data with equal distribution specifically for similar classes to directly influence the F1 score value of the model. **Because this architecture not only resembles Long-Term Potentiation in biological neural networks but also resembles how biological neural networks store learned information in memory**, the presence of similar features across multiple classes can dull the model's generalization ability and cause confusion, just like in humans. Humans learn best what they repeat the most. Just like in humans, the more these connections are strengthened (+feature, +feature, +feature), a bias based on experience is formed, allowing one feature's connection to be triggered without considering the presence of other features. So, it creates a bias based on experience, just like in humans. **In this architecture, the model's generalization ability is parallel to equal data distribution**.

Here its a mathematical reason:

```
% x = Same feature element (pixel) for the 1st and 2nd class.
% y = Pixel
% z = Pixel
% x = 255
% y = 255
% z = 255

% TRAINING:

 [1,1,0  % A feature matrix trained with an equal number of samples for
each class..
  1,1,1
  0,0,1]

% TEST (The output expected is for the 2nd class.):

[x*1,1,0  % x * 1 = x
  x*1,1,1 % x * 1 = x
  x*0,0,1] % x * 0 = 0

for 1st class = x = 255
for 2nd class = x = 255
for 3rd class = x = 0

% The other features are examined:

[1,y*1,0  % y * 1 = y
  1,y*1,1 % y * 1 = y
  0,y*0,1] % y * 0 = 0

for 1st class = x + y = 510
for 2nd class = x + y = 510
for 3rd class = x + y = 0

% The other features are examined:

[1,1,z*0  % z * 0 = 0
  1,1,z*1 % z * 1 = z
  0,0,z*1] % z * 1 = z

for 1st class = x + y + z = 510
for 2nd class = x + y + z = 765
for 3rd class = x + y + z = 255

[x*1,y*1,z*0  % (x * 1) + (y * 1) + (z * 0) = 510 (1st neuron).
  x*1,y*1,z*1 % (x * 1) + (y * 1) + (z * 1) = 765 (2nd neuron).
  x*0,y*0,z*1] % (x * 1) + (y * 1) + (z * 1) = 255 (3rd neuron).

% IT IS UNDERSTOOD THAT THERE IS AN INPUT BELONGING TO THE 2ND CLASS.
% LET'S ADD A NEW EXAMPLE BELONGING TO THE 1ST CLASS TO THE MODEL.
```

```
% TRAINING:

 [2,1,0  % One more example has been added for the 1st class.
  1,1,1
  0,0,1]

% TEST (The output expected is for the 2nd class.):

 [x*2,y*1,z*0  % (x * 2) + (y * 1) + (z * 0) = 765 (1st neuron).
  x*1,y*1,z*1 % (x * 1) + (y * 1) + (z * 1) = 765 (2nd neuron).
  x*0,y*0,z*1] % (x * 1) + (y * 1) + (z * 1) = 255 (3rd neuron).

% BIAS HAS LED TO CONFUSION.
% TO SOLVE:

% TRAINING:

 [2,1,0  % One more example has been added for the 2nd class. (Samples
equalized again)
  2,2,1
  0,0,1]

% TEST (The output expected is for the 2nd class.):

[x*2,y*1,z*0  % (x * 2) + (y * 1) + (z * 0) = 765 (1st neuron).
  x*2,y*2,z*1 % (x * 2) + (y * 2) + (z * 1) = 1275 (2nd neuron).
  x*0,y*0,z*1] % (x * 1) + (y * 1) + (z * 1) = 255 (3rd neuron).

% IT IS UNDERSTOOD ONCE AGAIN THAT THERE IS AN INPUT BELONGING TO THE 2ND
CLASS.
```

Let's display all the training steps together on a script before the test:

```
%% WEIGHT INITIALIZATION: %%

  weights = ones(10,784);



  %% POTENTIATION LAYER %%

  inputLayer = normalization(inputLayer) %% inputs in range 0 - 1
  weights(class,:) = inputLayer;


%% MERGING/SUMMING - SAVING WEIGHTS: %%

  if class == 1 % [[After the initial training,
   save the matrices to a file following the steps]]
```

```
    weights = sprintf('weights/weights.mat');

     save(weights, 'weights');


  else % if [[the matrices are already saved, retrieve them and add them
together.

     % When it's a new class, we won't perform addition but merging.
     % For the same class, we'll add them together, thus reinforcing
     % existing features and adding new ones to the matrix.]]

    newWeights = weights;

    weights = sprintf('weights/weights.mat');

      load(weights);


    weights += newWeights;

    weights = sprintf('weights/weights.mat');

     save(weights, 'weights');

  end
```

4 - Test:

Potentiation Learning Model Testing And Predictions

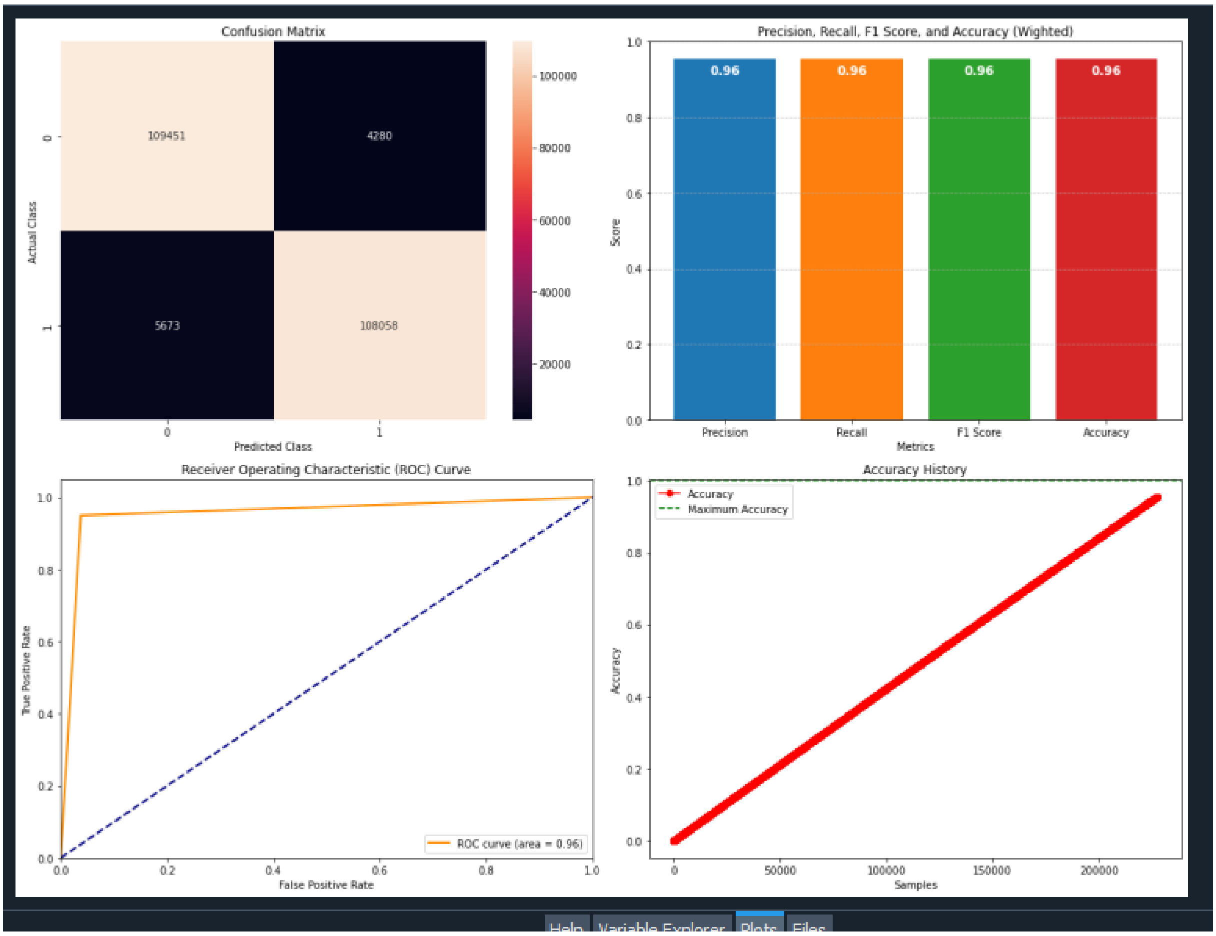
If it's in the testing and prediction phase, it simply involves multiplying the input used during training.

```
%% Only matrix multiplications:

outputLayer = weights * inputLayer;
```

5 – Results For Potentiation Learning Neural Networks:

All the codes for this application:
https://github.com/HCB06/Anaplan/tree/main/Welcome_to_PLAN/Codes.

You can train and test models wtih Potentiation Learning Artificial Neural Network architecture. All datasets in this article:
https://github.com/HCB06/Anaplan/tree/main/Welcome_to_Anaplan/ExampleCodes



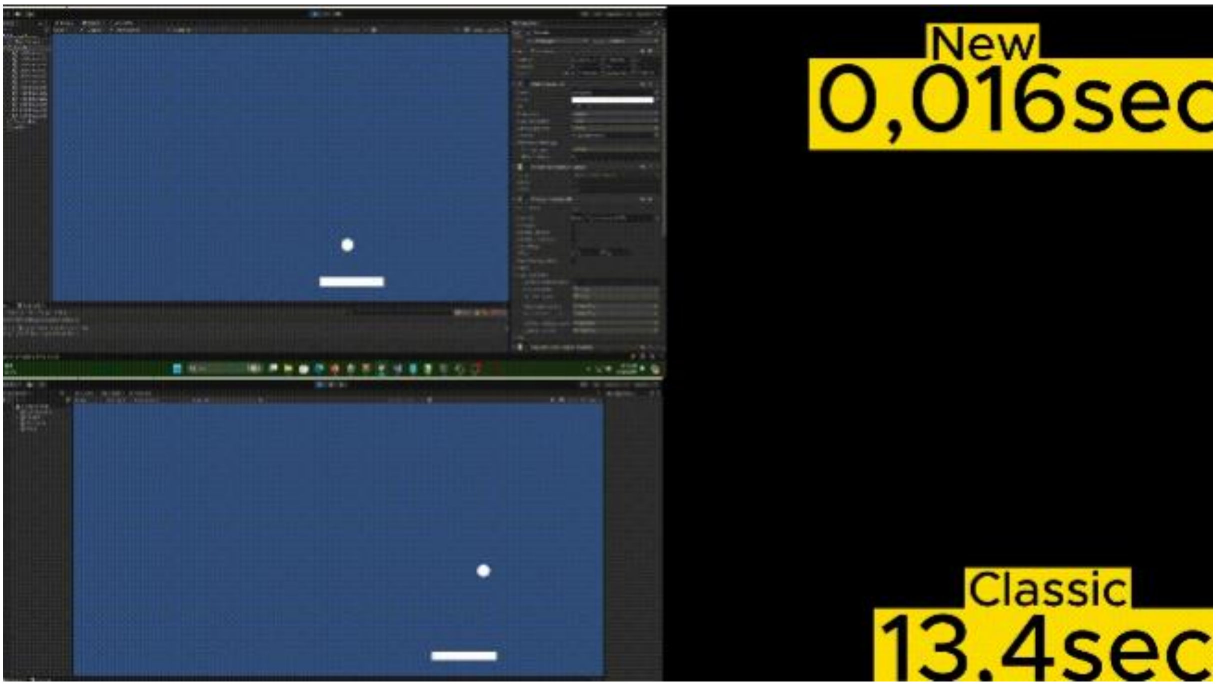
Dataset from: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Also I created this Unity demo for autonomous vehicles lane tracking AI with Potentiation Learning Artificial Neural Network algorithm:

My YouTube video(Tr): *https://www.youtube.com/watch?v=ApCKXqSMqT4&t=507s*

And with output layer normalization finalized: *https://www.linkedin.com/posts/hasan-can-beydili-77a1b9270_ke%C5%9Ffetti%C4%9Fim-e%C4%9Fitim-mimarisini-kullanan-modelimi-activity-7190854824471543808-CWhb?utm_source=share&utm_medium=member_desktop*

Training speed and overall performance comprasion with Tensorflow’s Dense ANN’s:

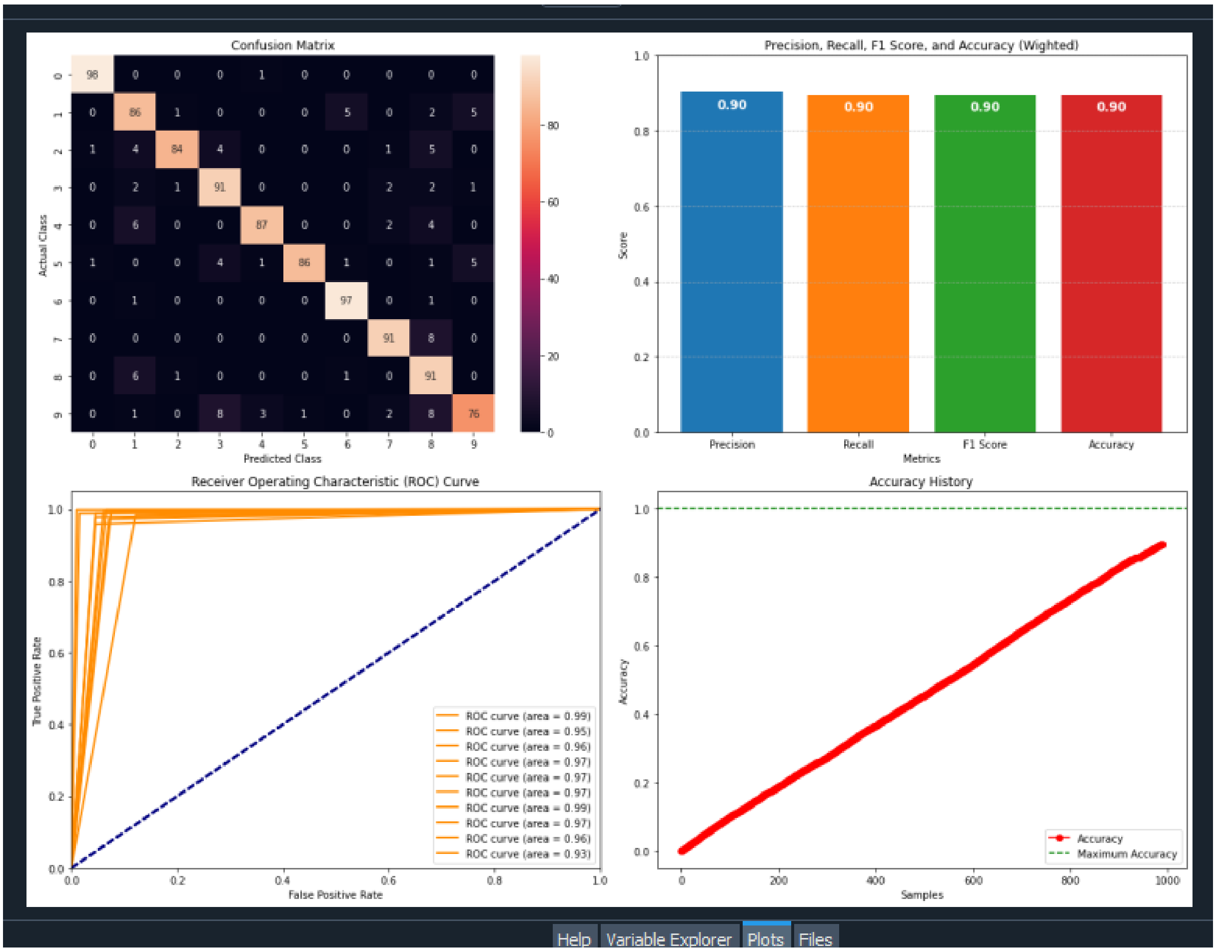


My YouTube video(Tr): *https://www.youtube.com/watch?v=tH7pgPSbyUo&t=1s*

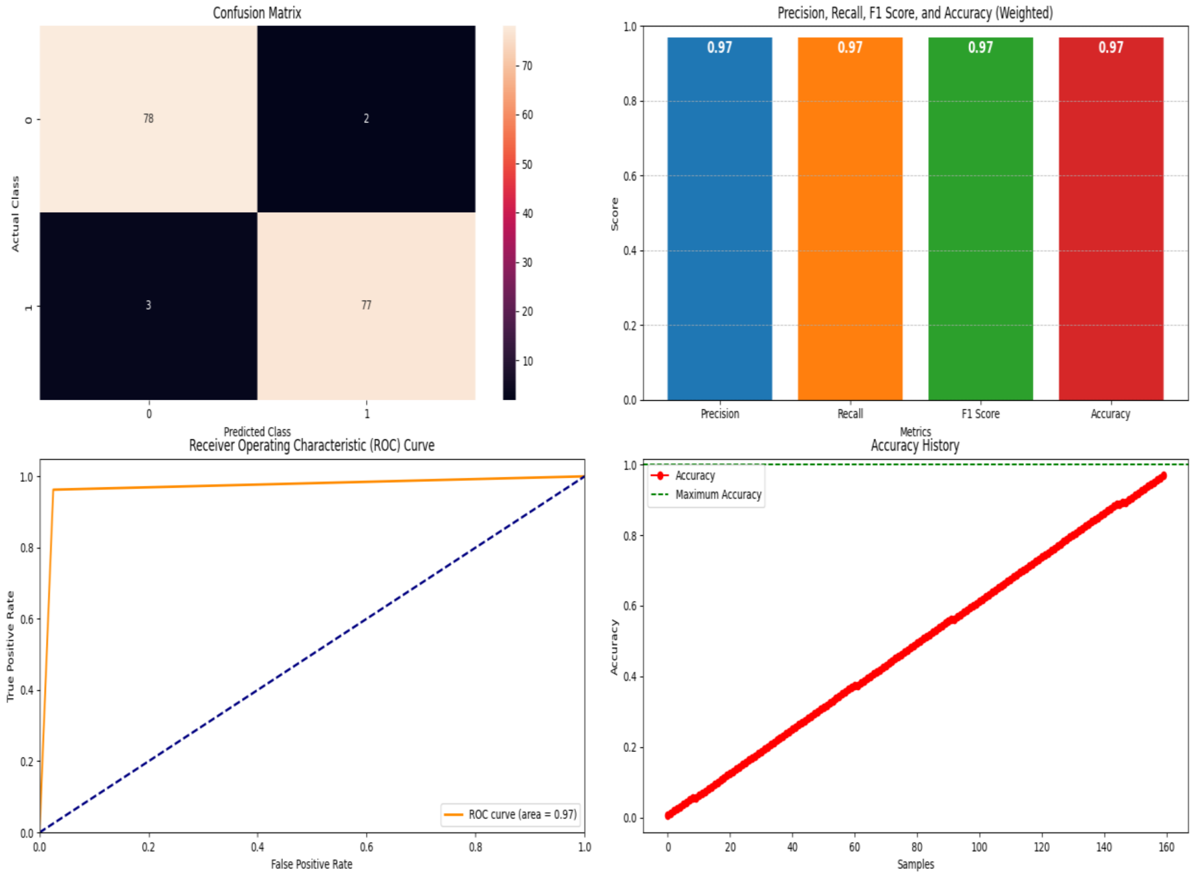
Shortly: *https://www.linkedin.com/posts/hasan-can-beydili-77a1b9270_brick-breaker-oyunu-yap%C4%B1yorum-burada-anlatt%C4%B1%C4%9F%C4%B1m-activity-7196972136014434307-6rRQ?utm_source=share&utm_medium=member_desktop*

Trained and tested with digits dataset in sckitlearn library and comprasion wtih Tensorflow's fully connected ANN's

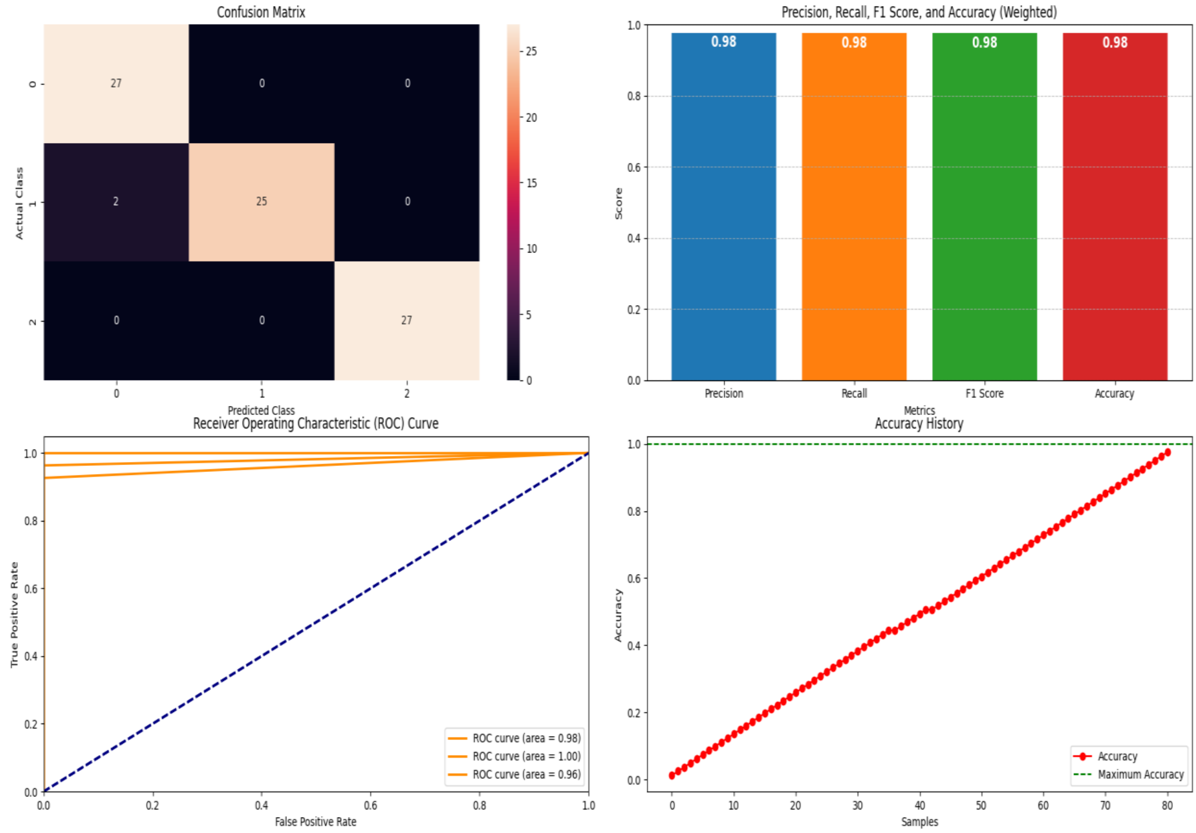
My YouTube video(Tr): https://www.youtube.com/watch?v=uT5mYvUVdmA&t=2s



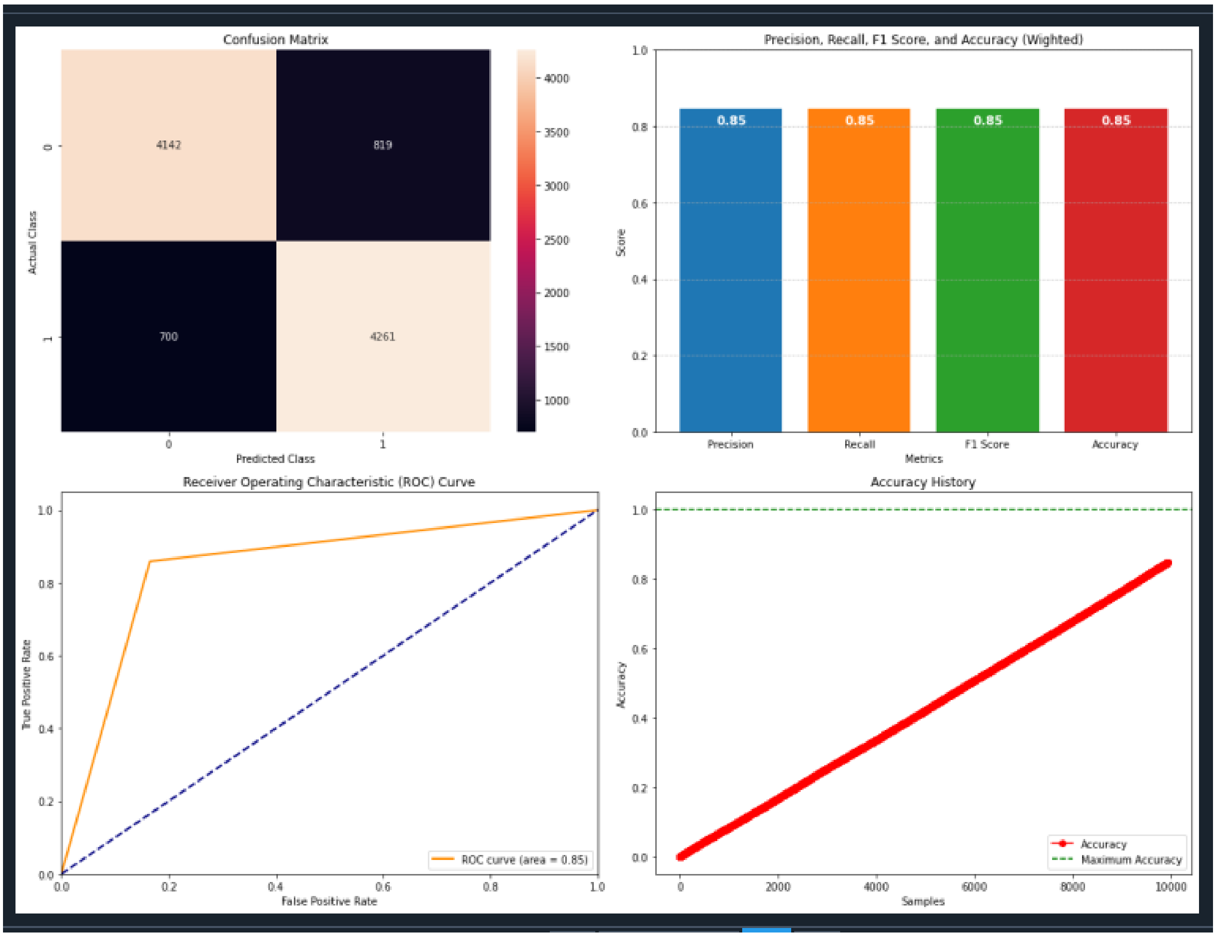
Dataset From: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html



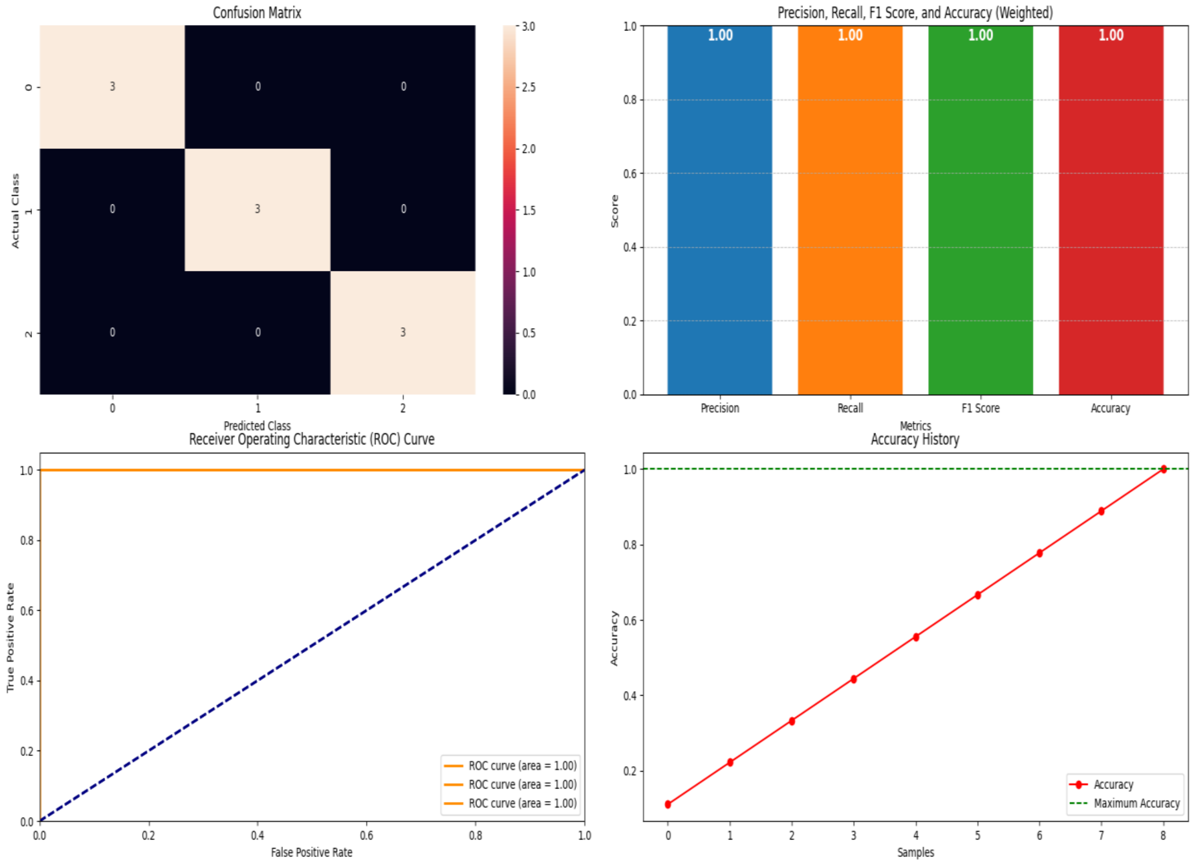
Dataset From*:* *https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data*



Dataset From*:* [*https://www.geeksforgeeks.org/wine-dataset/*](https://www.geeksforgeeks.org/wine-dataset/)



Dataset From (6000 Features Selected):
https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews



Dataset From: https://www.kaggle.com/datasets/uciml/iris

More models will be added to the Pyerual Jetwork Library GitHub page: https://github.com/HCB06/Anaplan/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.

Especially the absence of any matrix multiplication during training makes the algorithm particularly special and noteworthy.

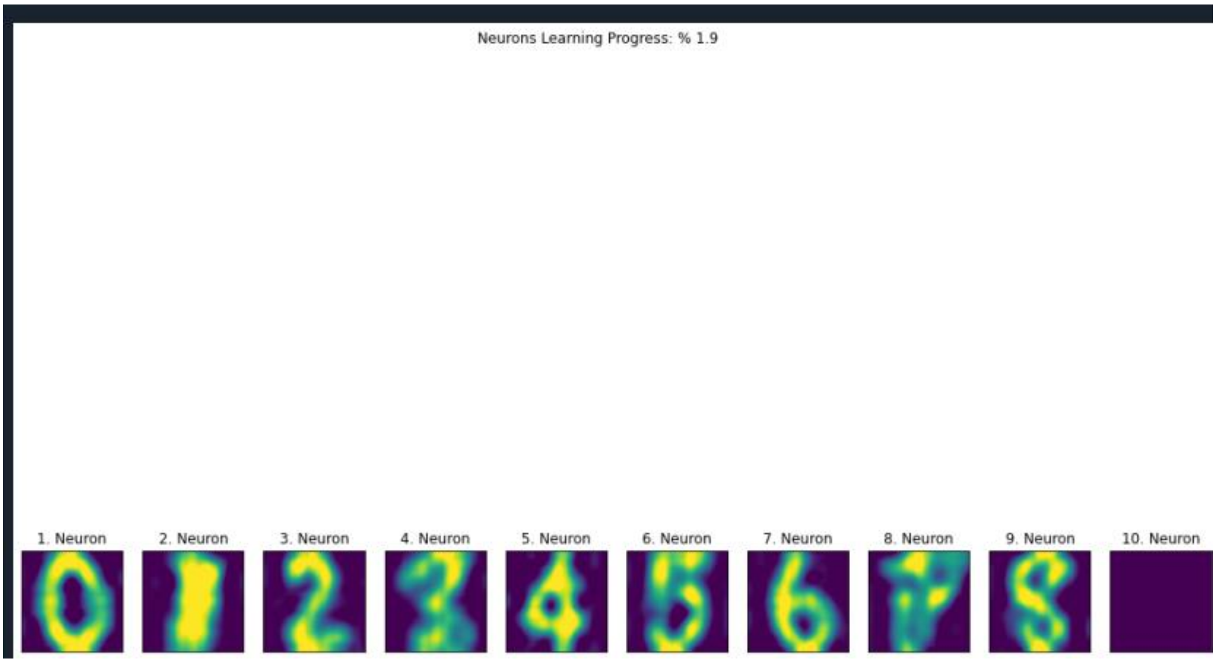
Performance on Different Dataset Sizes:

Through my research, I have observed that Potentiation Learning Artificial Neural Network performs particularly well on large datasets. The architecture's ability to handle extensive input features and capture complex patterns makes it suitable for scenarios with abundant data. For instance, when applied to large datasets, Potentiation Learning Artificial Neural Network’s can efficiently extract and process the underlying features, leading to high classification accuracy.

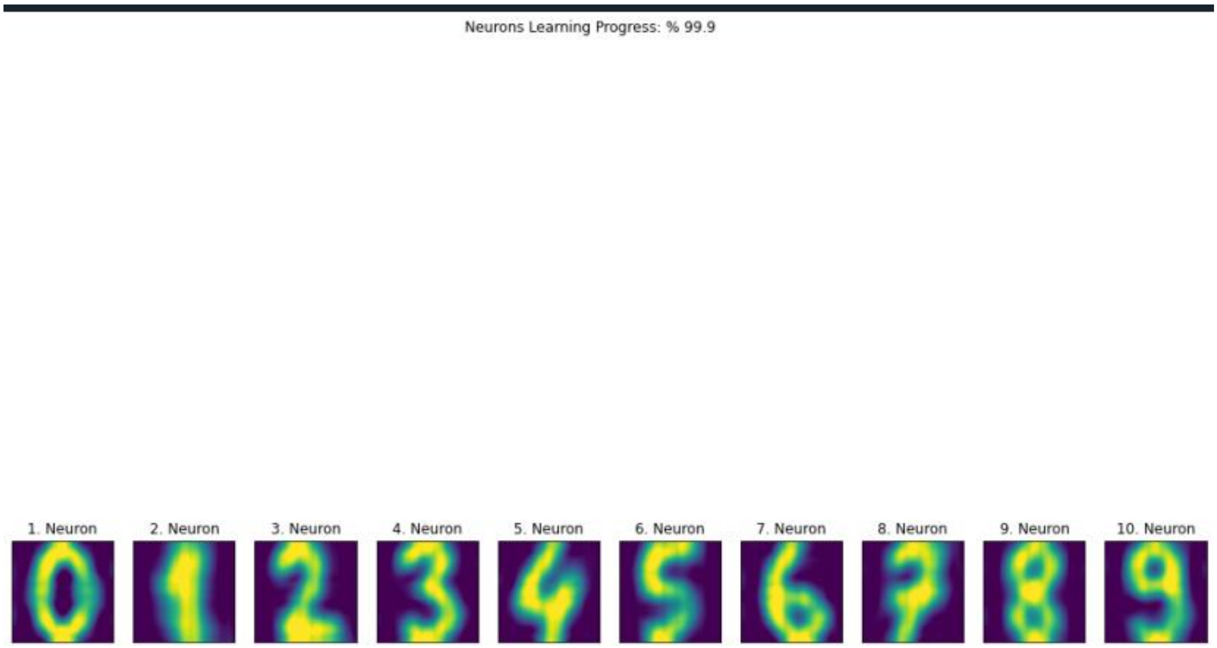
Conversely, when working with smaller datasets, such as the Iris dataset, the architecture shows a heightened sensitivity to the specific samples used for training. This sensitivity impacts the generalization performance significantly. The reliance on specific data points means that the model's performance can vary greatly with different training subsets, highlighting the need for careful sample selection and Potentiationly more sophisticated techniques to ensure robust generalization. I set the test size to 10% specifically for the Iris dataset.

Also, while it exhibits near-perfect performance on datasets with a high number of features, it tends to struggle with datasets containing fewer features. However, in terms of overall performance, the model architecture trained without any hyperparameter tuning (direct injection) consistently achieved impressive results, never dropping below an 0.85 accuracy rate across all datasets.

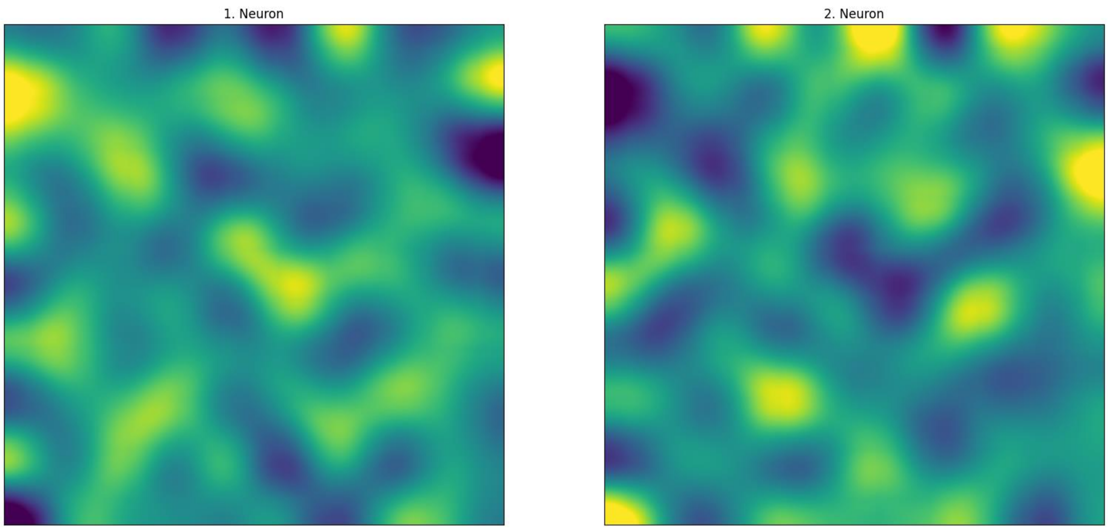
I conducted tests particularly on text, images, and other diverse data types, and achieving consistently above a certain standard in all of them is quite impressive. This underscores how widely applicable the Potentiation Learning Artificial Neural Network algorithm can be to a general audience.



Potentiation layer (Learning at %1.9 with digits dataset)

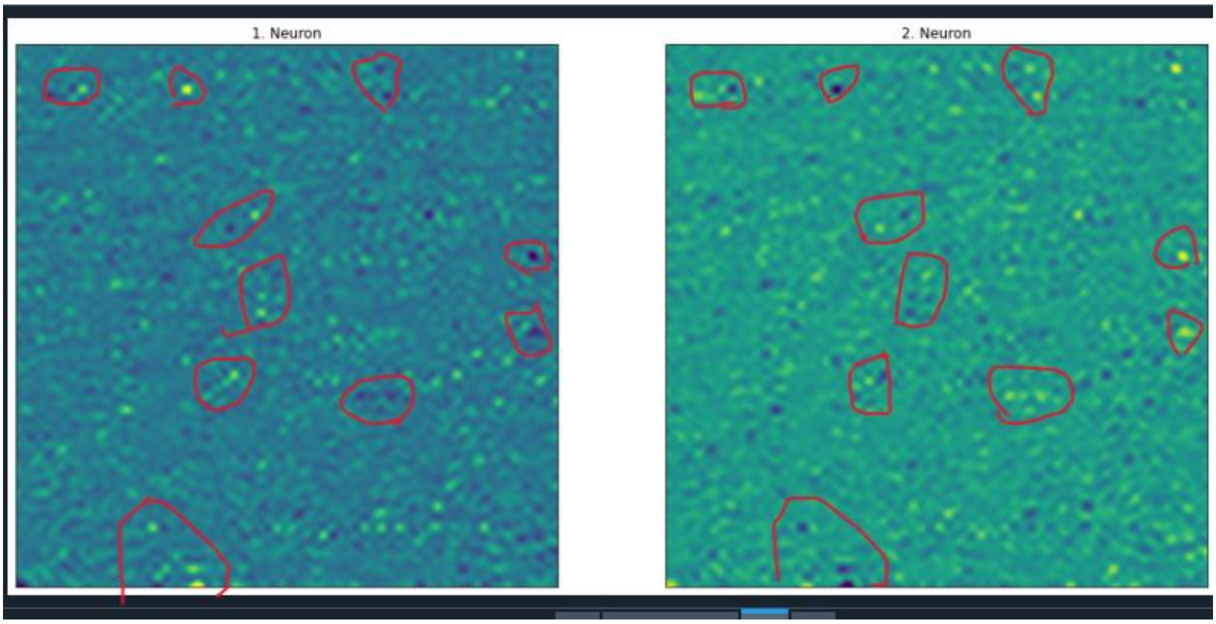


Potentiation layer (Learning at %99.9 with digits dataset)

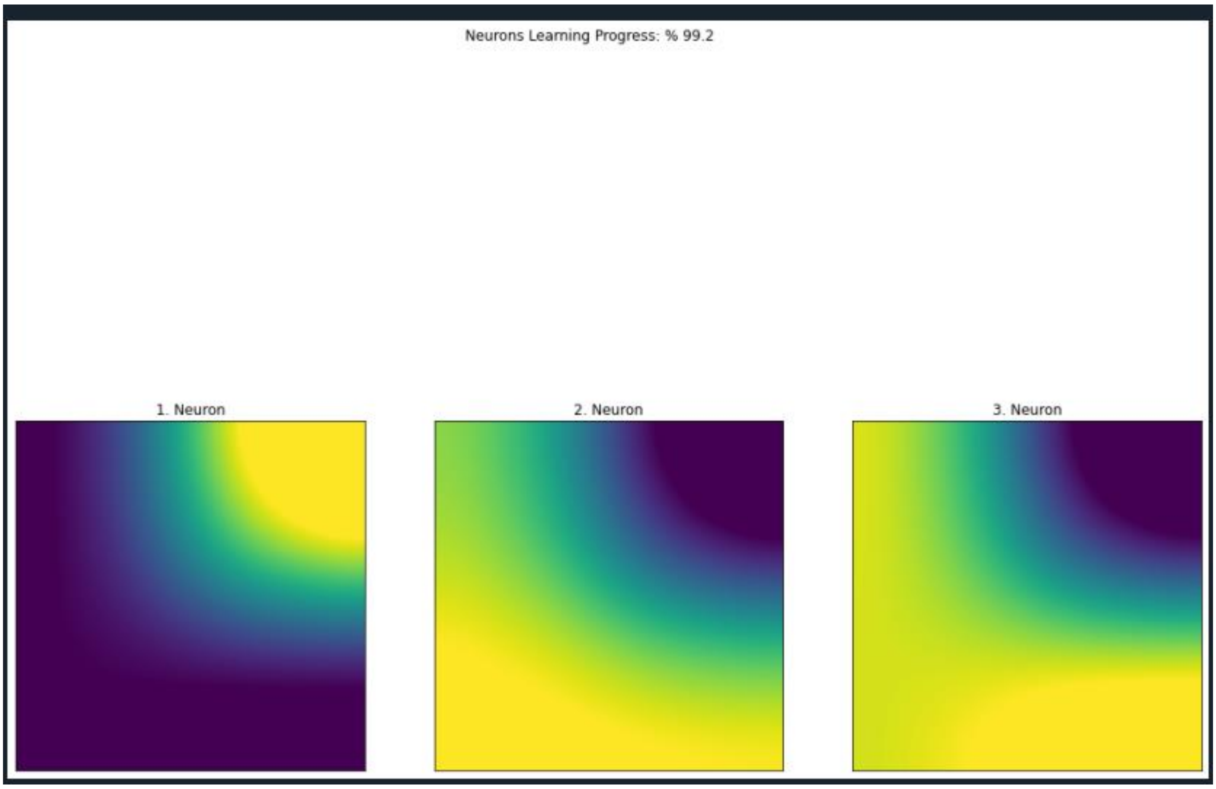


Potentiation layer (Learning at %100 with imdb dataset - selected feature count: 100)

Neurons can learn diffirences of classes:



Potentiation layer (Learning at %100 with imdb dataset - selected feature count: 6084)

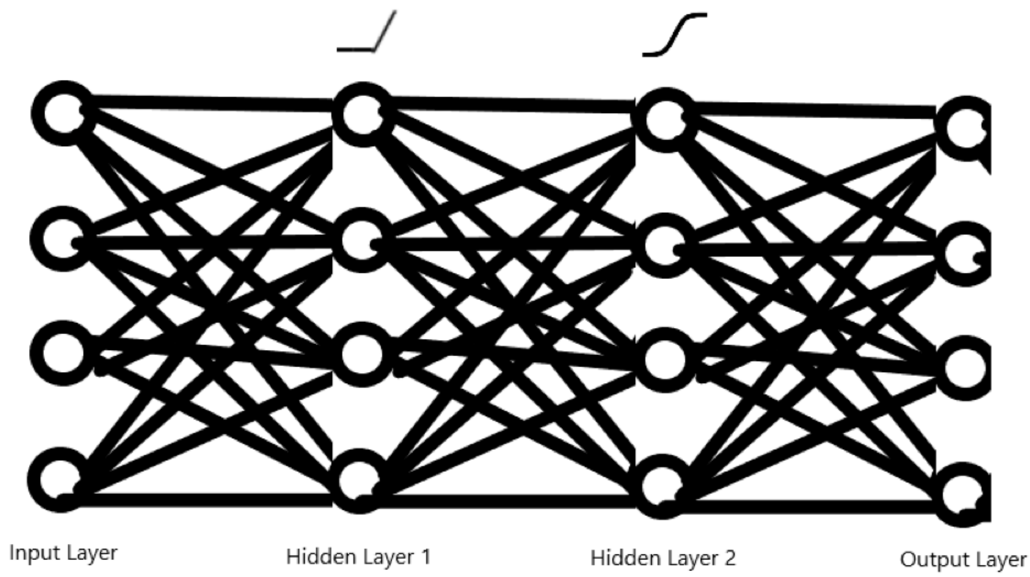


Potentiation layer (Learning at %99.2 with iris dataset)

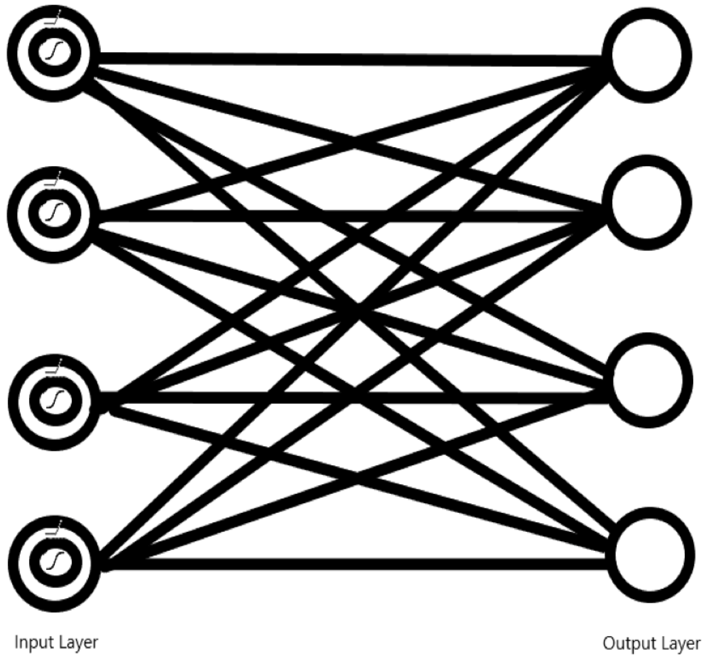
Neurons can autonomously learn distinctive features by mimicking Long-Term Potentiation. This ability can be utilized to analyze differences between classes.

B. Deep Potentiation Learning Artificial Neural Networks:

The 'depth' in Deep Potentiation Learning Artificial Neural Networks is fundamentally different from that in other artificial neural networks. While other neural networks possess **horizontal depth**, Deep Potentiation Learning Artificial Neural Networks are characterized by **vertical depth**.



Classic Deep Learning (Horizontal Depth)



Deep Potentiation Learning Artificial Neural Network (Vertical Depth)

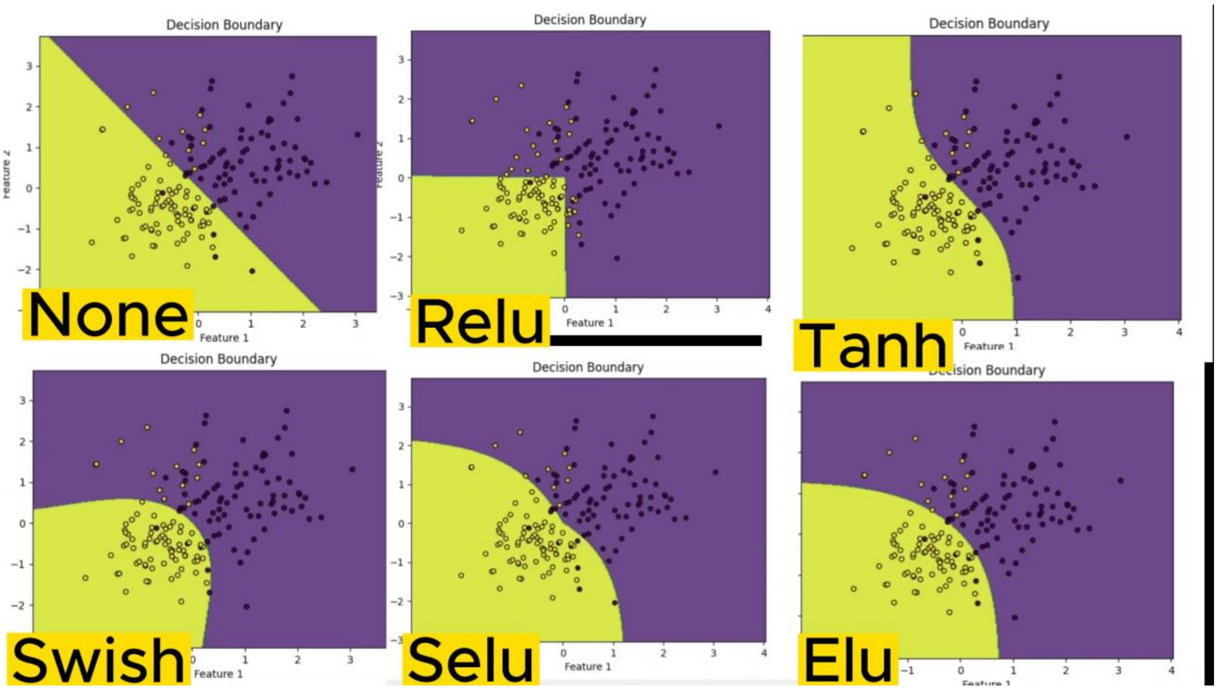
6 – Training And Math For Deep Potentiation Learning Artificial Neural Network:

The training process 26ort he Deep Potentiation Learning Artificial Neural Network mirrors that of the Potentiation Learning Artificial Neural Network. However, before adding the input values to the corresponding columns in the weight matrix, the inputs are passed through an activation function or functions, selected based on the 2D PCA distribution of our dataset. If multiple activation functions are selected, the outputs of all activations are summed before the input is recorded in the weight matrix. I have designated this list of activation functions as "activation_potentiation."

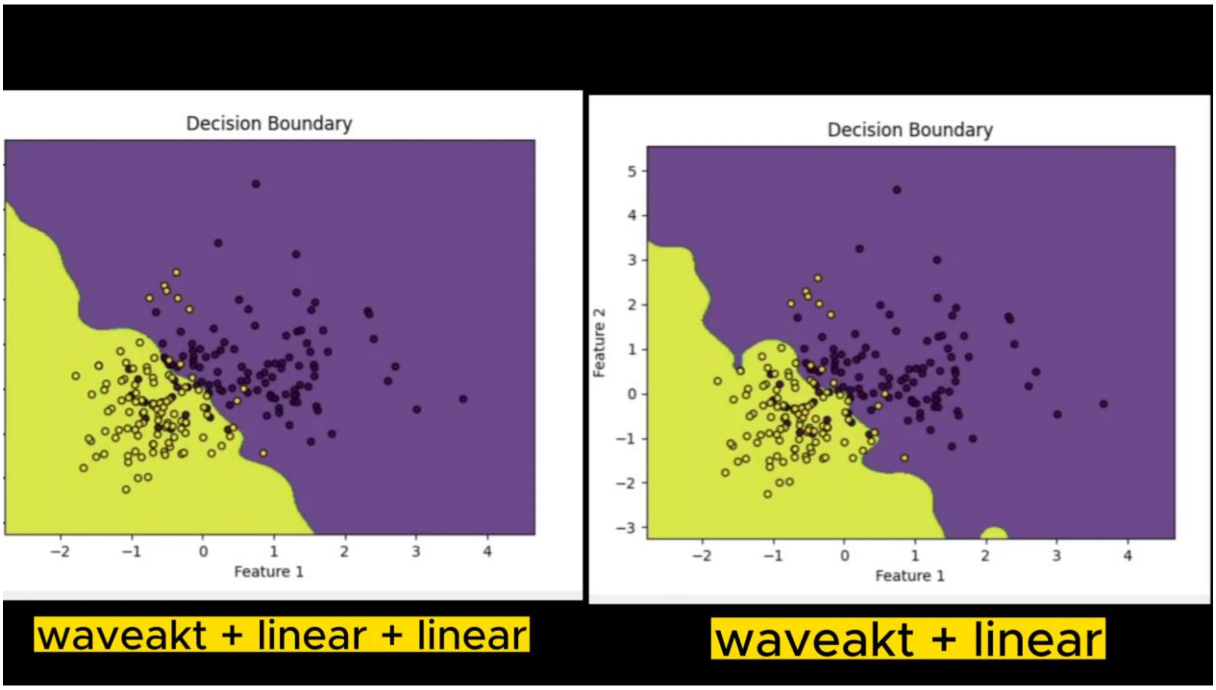
Let us recall the training process of the Potentiation Learning Artificial Neural Network:

```
inputLayer = normalization(inputLayer); % inputs in range 0 - 1

%% POTENTIATION LAYER %%

for i = 1: length(activaiton_potentiation)

      if activation_potentiation(i) == 'sigmoid' %% example

            inputLayer  = sigmoid(inputLayer); %% if activation is sigmoid
```

```
        end
end
weights(class,:) = inputLayer;
```



Decision boundaries of the model for different activation functions: (None = no activation function)



Activation functions can be combined to mitigate each other’s weaknesses, thereby adapting to the shape of the data and uncovering deeper features.

Long-Term Depression (LTD) Role For Deep Potentiation Learning Artificial Neural Network:

The concept of Long-Term Depression (LTD) was discussed in the conceptual section. In the context of the algorithm, LTD is implemented by introducing random noise equivalent to the LTD parameter before the input vector is added to the weight matrix during training. The primary goal of this approach is to prevent the model from overfitting and to enhance its generalization performance. By employing this technique, improved performance can be achieved with large and medium-sized datasets.

```
İnputLayer = normalization(inputLayer); % inputs in range 0 – 1

%% POTENTIATION LAYER %%

for i = 1: length(activaiton_potentiation)

      if activation_potentiation(i) == ‘sigmoid’ %% example

            inputLayer  = sigmoid(inputLayer); %% if activation is sigmoid

      end

end

depression_vector = rand(1, length(inputLayer))
```

```
for i = 1:LTD

      inputLayer -= depression_vector

end


weights(class,:) = inputLayer;
```

In this process, data loss is almost nonexistent, as cumulative learning takes place, making it easier to capture the most general features.

7 – Testing And Predictions For Deep Potentiation Learning Artificial Neural Network:

```
for i = 1: length(activaiton_potentiation)

      if activation_potentiation(i) == ‘sigmoid’ %% example

            inputLayer  = sigmoid(inputLayer); %% if activation is sigmoid

      end

end

outputLayer = weights * inputLayer;
```

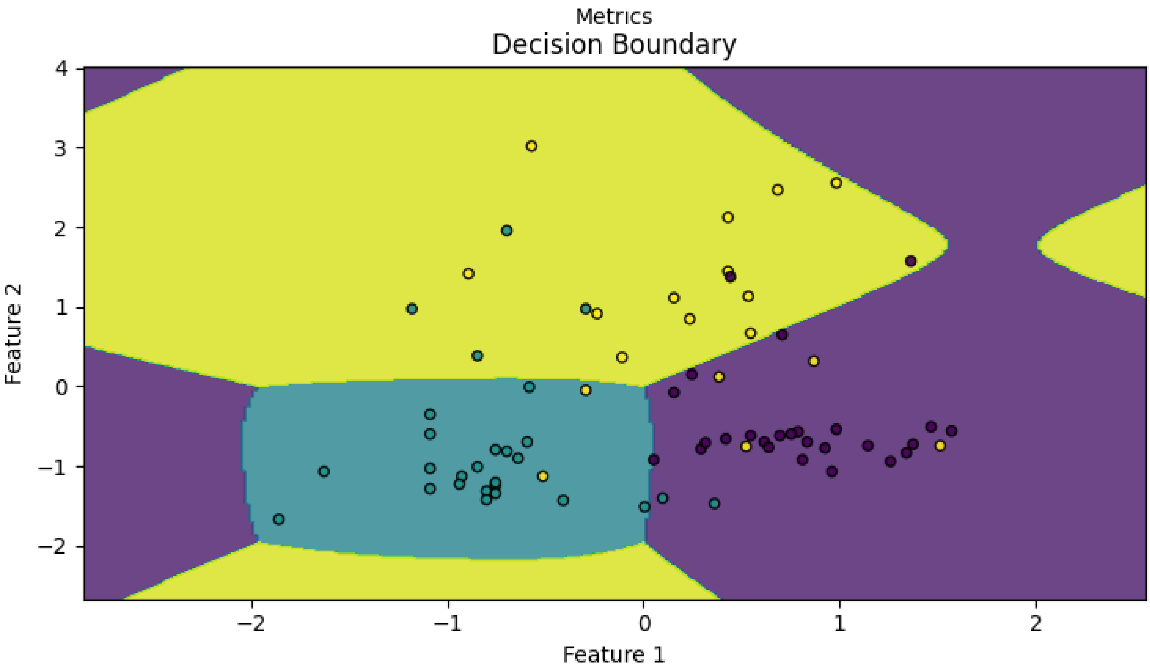


Figure1

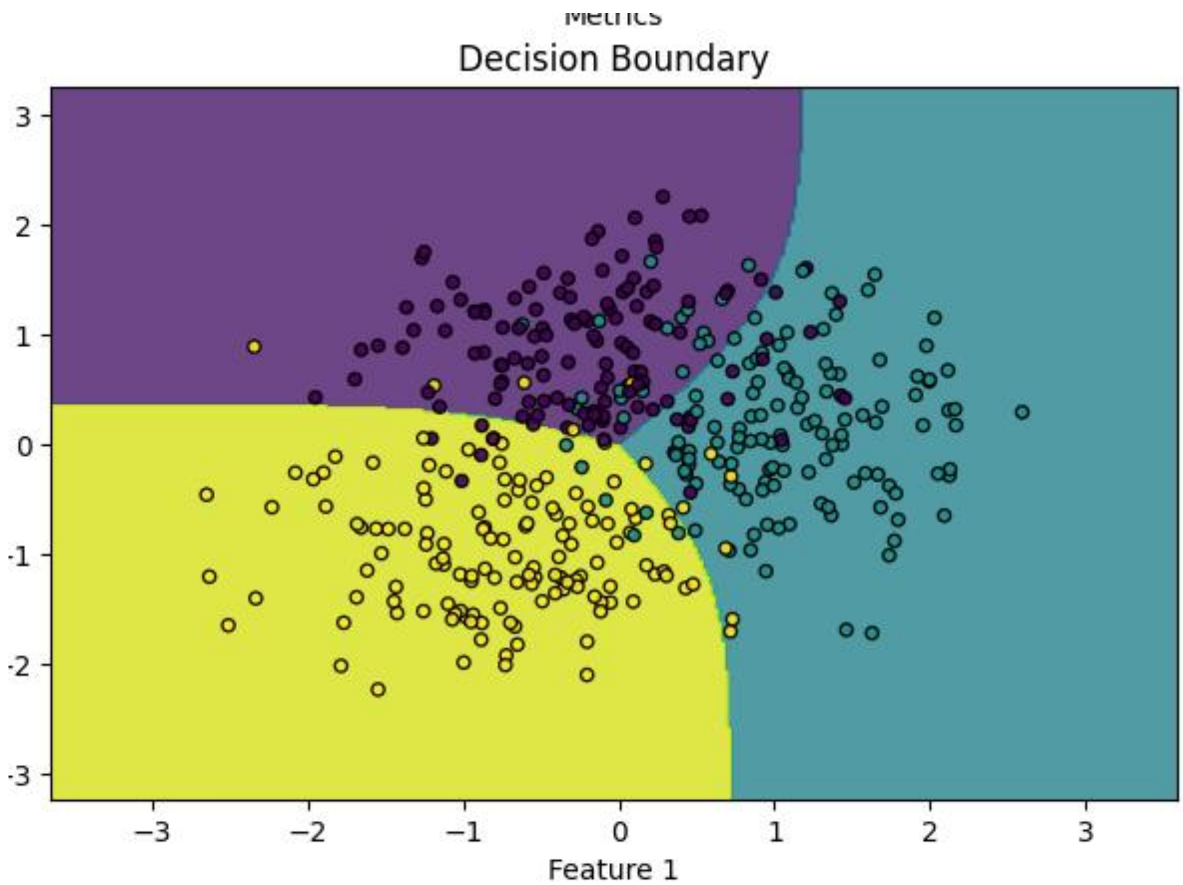


Figure2

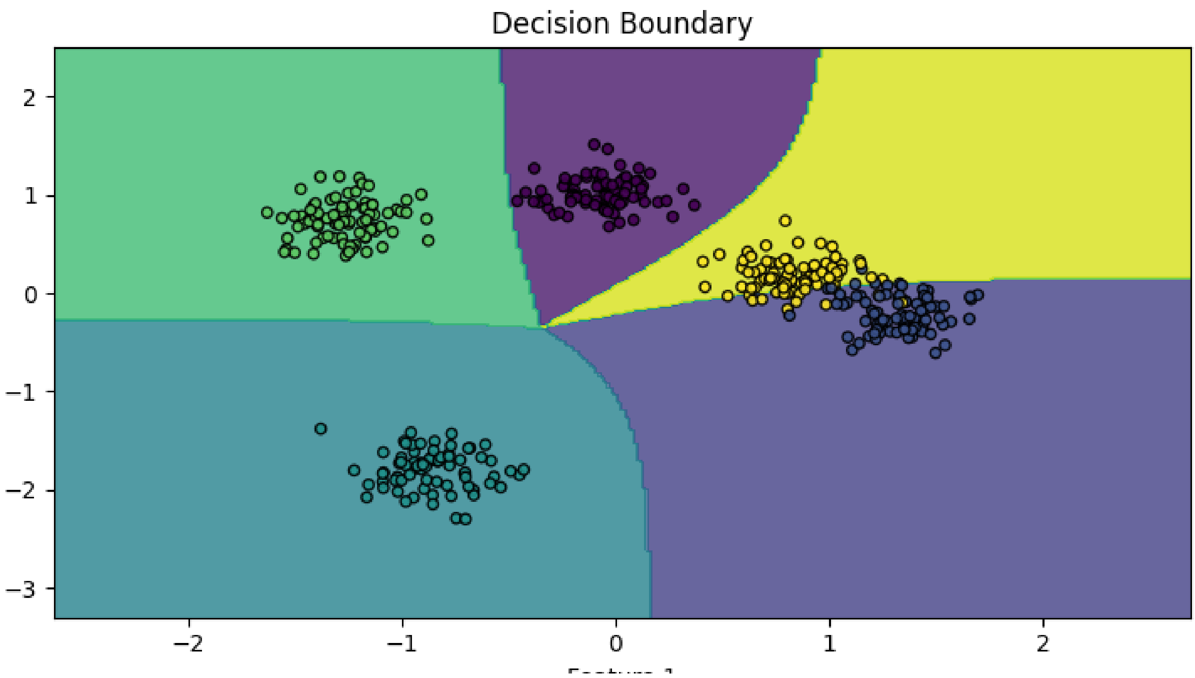


Figure3

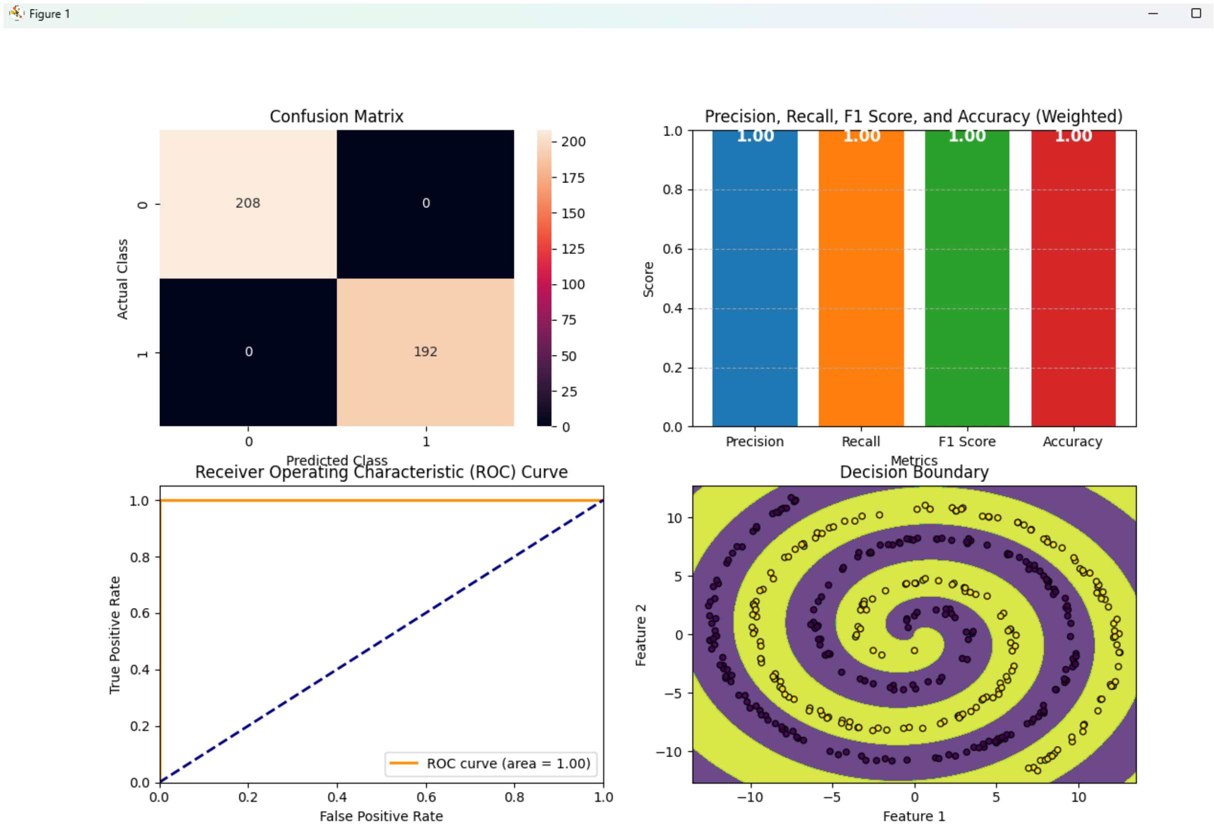


Figure4 (Spiral Dataset with %100 accuracy)

The Deep Potentiation Learning Artificial Neural Network is as simple as a perceptron yet as powerful as a deep learning algorithm. It is an extraordinary algorithm with the potential to learn deep features through a vertical learning process, rather than a horizontal one, while performing fewer mathematical operations.

Potentiation Learning Artificial Neural Network Library: Anaplan

To facilitate broader experimentation and practical application, I have encapsulated the Potentiation Learning Artificial Neural Network architecture in a Python library called "Anaplan." 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.

For those interested in exploring or utilizing this framework, the library and the GNU Octave codes referenced in this paper are hosted on GitHub. You can find them at: GitHub - Anaplan. 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.

Integration with Existing Architectures

One of Potentiation Learning Artificial Neural Network's strengths is its flexibility to integrate with existing artificial neural network architectures. For instance, instead of using a traditional fully connected layer at the end of a Convolutional Neural Network, the Potentiation Learning Artificial Neural Network framework's Potentiation Layer. This approach not only leverages the interpretability and efficiency of Potentiation Learning Artificial Neural Network but also enhances the overall model by replacing complex layers with more manageable components.

Environmental Impact

Due to its non-iterative learning process, Potentiation Learning Artificial Neural Network's offers a significant advantage in terms of computational efficiency. In scenarios where Potentiation Learning Artificial Neural Network's is widely adopted, its streamlined approach to training can lead to a substantial reduction in the computational resources required. This efficiency has the Potentiation to positively impact global warming by lowering the energy consumption associated with large-scale neural network training. By promoting a more sustainable method for training Artifical Intelligence models, Potentiation Learning Artificial Neural Network's could contribute to reducing the environmental footprint of Artificial Intelligence technologies.

8 – Advantages and Disadvantages:

Advantages:

1. Faster training times. (no loop usage, no epochs)
2. Extremly fast training times. (dont need matrix multiplicaitons for training.)
3. Easy to learn and implement. (no calculus)
4. Easy to maintain and update. (non-blackbox)
5. It occupies less memory space. (only can have one weight matrix)
6. Energy efficency (less mathematical operations)
7. Strong generalization (vertical learning architecture)

Disadvantages:

1. In early research stage

The Potentiation Learning Artificial Neural Network architecture, through its innovative approach to neural network design, offers a compelling alternative to traditional methods. Its Potentiation for faster, more interpretable, and environmentally friendly learning processes makes it a significant development in the AI 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/Anaplan/tree/main/Welcome_to_PLAN/Codes

Potentiation Learning Artificial Neural Network library: *https://github.com/HCB06/Anaplan*