Production Systems and Information Engineering
Volume 14 (2), pp. 77-93
doi: https://doi.org/10.32968/psaie.2026.2.6

VISUALIZING THE TRAINING PROCESS OF MLP

FADI ATALLAH
University of Miskolc, Hungary
Institute of Information Technology
fadiatallah1997@gmail.com

LÁSZLÓ KOVÁCS
University of Miskolc, Hungary
Institute of Information Technology
laszlo.kovacs@uni-miskolc.hu

Abstract. Multilayer Perceptrons (MLPs) are extensively used in many machine learning applications; yet, because of their high dimensionality and complexity, their training process is still largely opaque. In order to better understand how MLPs learn and modify their internal representations, this study focuses on visualizing the evolution of decision boundaries during training. We gain insights into neural networks' learning dynamics and convergence behavior by analyzing how the decision regions change as training progresses. This work also contributes to the broader field of Explainable AI (XAI) by offering a visual approach to interpreting neural network decision-making, addressing the common critique of MLPs being black-box models.

Keywords: Multilayer Perceptron, Neural Network, Black-Box Model, Explainable AI, Feedforward, Activation Functions, Principal Component Analysis, Training Visualization

1. Introduction

Neural networks, particularly Multilayer Perceptrons (MLPs), have become fundamental components in modern machine learning applications [1]. They perform exceptionally well in challenging tasks, including decision-making systems, natural language processing, and picture recognition. However, despite their effectiveness, MLPs are often perceived as black-box models [2] due to the difficulty in interpreting their internal decision-making processes. Unlike traditional machine learning models such as decision trees or linear classifiers, where decision rules are explicitly defined, MLPs rely on high-dimensional weight transformations and nonlinear activation functions to learn patterns in data. Particularly in crucial areas like healthcare, finance, and autonomous systems, this lack of interpretability raises questions regarding neural network-based systems' dependability, equity, and trustworthiness.

One way to improve interpretability is through Explainable AI (XAI), a field dedicated to developing techniques that make machine learning models more

understandable to humans [3], [4], [5]. Activation visualization, decision boundary visualization, and feature importance analysis are a few examples of XAI methods [5]. Among these methods, decision boundary visualization offers a simple means of seeing how an MLP divides several classes in the input space. Decision boundaries define the regions where the model assigns specific class labels, and their evolution during training reveals how the model adjusts its internal parameters to improve classification accuracy.

This research focuses on visualizing the transformation of decision boundaries throughout the training process of an MLP. By capturing and analyzing how these boundaries change over time, we can gain deeper insights into the learning dynamics of neural networks. Specifically, I aim to explore the following questions:

- How do decision boundaries evolve as the network learns from data?
- What patterns emerge in decision boundary transformations during different training phases, such as early learning, convergence, and potential overfitting?
- How do training parameters, such as learning rate and network architecture, influence decision boundary formation?

Understanding these aspects is crucial for improving training strategies, diagnosing issues like overfitting or underfitting, and enhancing model interpretability. By providing a visual and conceptual framework for analyzing MLP training, this work contributes to the broader efforts in making neural networks more transparent and explainable.

2. Background

In this chapter, the MLP model, the Black Box model, and the Explainable AI will be discussed in detail.

2.1. Multilayer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) is a class of artificial neural networks (ANNs) that consists of an input layer, one or more hidden layers, and an output layer [6]. Each layer is composed of artificial neurons (or nodes) that process information by applying weighted sums and activation functions to input data. The input layer is the first layer of the MLP that receives raw input data. Each neuron in this layer corresponds to a feature in the input dataset and passes the values forward without transformation [1]. The hidden layers are the intermediate layers which perform complex feature extraction and transformation. Each neuron in a hidden layer receives input from the previous layer, applies a weighted sum followed by an activation function, and passes the result to the next layer. The number of hidden layers and neurons per layer significantly impacts the network's learning capacity and performance. The final layer of the MLP is the output layer that produces the model's predictions. The number of neurons in this layer depends on the nature of the problem. A SoftMax or sigmoid activation function is commonly used for classification tasks, while regression problems typically use a linear activation function. Figure 1 shows the layers of the MLP Neural Network.

MLPs are widely used for tasks including function approximation, regression, and classification and are an essential part of deep learning models. The activation functions in MLPs, such as the sigmoid, ReLU (Rectified Linear Unit), or tanh, introduce non-linearity [7], enabling the network to learn complex patterns and

decision boundaries. The choice of activation function can significantly impact model performance, as different functions influence gradient flow and convergence speed differently.

Training an MLP involves minimizing a loss function using optimization algorithms like stochastic gradient descent (SGD) or Adam. The process includes forward propagation, where inputs pass through the network to produce an output, and backpropagation, where gradients are computed and used to update weights. Efficient weight optimization while avoiding overfitting and convergence problems is a significant difficulty in MLP training. Methods including dropout, batch normalization, and early stopping are frequently used to improve generalization and stability.

2.2. Black-Box Model

Neural networks, including MLPs, are often called "black-box" models due to their complex, non-transparent internal operations [8]. Unlike traditional machine learning models such as decision trees or linear regression, where the decision-making process is more interpretable, MLPs rely on numerous hidden layers and nonlinear transformations that make it difficult to understand how specific inputs directly contribute to outputs.

Because MLPs rely on high-dimensional feature representations and hierarchical information abstraction, they are fundamentally black-box. Features are changed in ways that are difficult for humans to understand as data moves through the levels. Although this abstraction is useful for capturing complex patterns, it presents difficulties in situations where the explainability and reliability of the model are required.

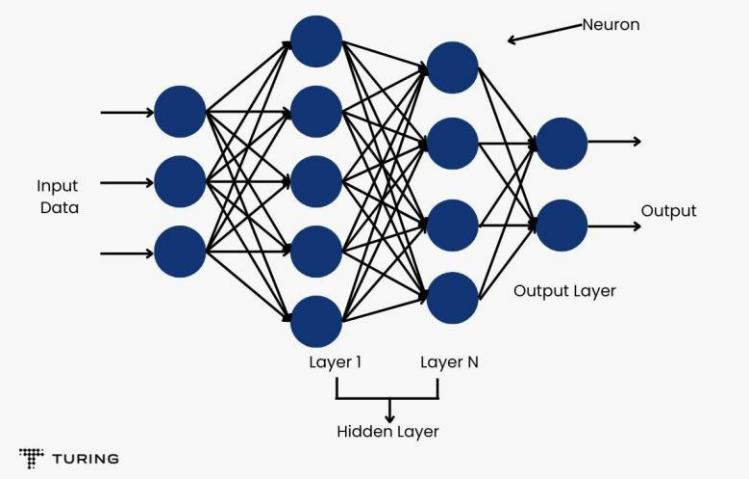


Figure 1. MLP Neural Network

Because MLPs rely on high-dimensional feature representations and hierarchical information abstraction, they are fundamentally black-box. Features are changed in ways that are difficult for humans to understand as data moves through the levels. Although this abstraction is useful for capturing complex patterns, it presents difficulties in situations where the explainability and reliability of the model are required.

The lack of interpretability raises concerns about trust, accountability, and fairness in AI-driven decision-making, especially in critical applications such as healthcare, finance, and autonomous systems. For example, in medical diagnostics, an AI

model's decision regarding a patient's condition must be transparent and justifiable to doctors and patients alike [9].

2.3. Explainable AI (XAI)

Explainable AI (XAI) is a field dedicated to making AI models more interpretable and understandable to humans. XAI aims to provide insights into how models make decisions, allowing users to trust and validate their outputs [10]. The growing complexity of deep learning models, including MLPs, has made explainability a crucial area of research, particularly in regulated industries and high-stakes applications.

Explainable AI (XAI) is a field dedicated to making AI models more interpretable and understandable to humans. XAI aims to provide insights into how models make decisions, allowing users to trust and validate their outputs [10]. The growing complexity of deep learning models, including MLPs, has made explainability a crucial area of research, particularly in regulated industries and high-stakes applications.

Several techniques have been proposed to enhance the explainability of MLPs. Feature importance analysis helps identify which input features contribute most significantly to predictions, allowing domain experts to assess model validity. Layer-wise relevance propagation (LRP) assigns relevance scores to different layers and neurons, providing a visual representation of how information flows through the network [11]. Gradient-based methods, such as saliency maps [12] and Grad-CAM [13], highlight areas in input data that influence the model's decisions.

One key approach in XAI is visualizing how an MLP learns and evolves its decision boundaries during training. By plotting decision regions and tracking their changes over epochs, researchers can gain valuable insights into how neural networks generalize patterns from data. Visualizations can help diagnose model weaknesses, detect biases, and improve network architecture.

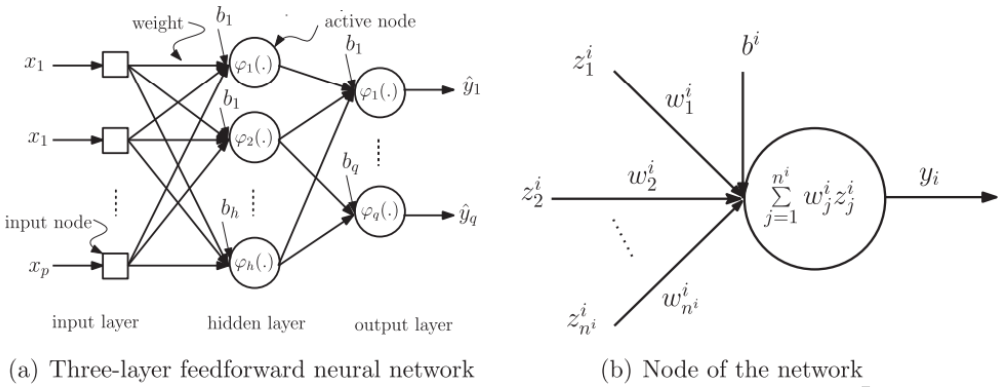
In the context of this research, I focus on leveraging visualization techniques to track the training process of MLPs, aiming to bridge the gap between black-box models and interpretability. Through these visual representations, I seek to provide a clearer understanding of how neural networks adapt to data, ultimately contributing to the broader objectives of XAI.

3. MLP Model

In this chapter, I will describe the steps of the MLP model training and some activation functions.

3.1. Feedforward

MLP uses feedforward neural network architectures, which learn to map a fixed-size input (for example, an image) to a fixed-size output (for example, a probability for each of several categories). To go from one layer to the next, a set of units computes a weighted sum of their inputs from the previous layer and passes the result through a non-linear function [14].



(a) Three-layer feedforward neural network (b) Node of the network

Figure 2. Feedforward in Neural Network

As shown in Figure 2, each node is connected to all nodes in the next layer, and each connection has its weight and bias values. Initially, the bias values are zero and will be changed during the training process, but the weight values cannot be zero because the neural network will be useless if the weights are zero. The weight value is to make an input value more or less critical.

The feedforward process starts with feeding the input data, and each input neuron transmits its value to the neurons in the next layer. Then, each neuron in the hidden layer computes a weighted sum of its inputs, as in equation (3.1).

$$z^{(l)} = \sum_{i=1}^{n} w_i^{(l)} a_i^{(l)} + b_i^{(l)} \quad (3.1)$$

Where z is the neuron value, l is the layer index, n is the number of the input values, w is the weight, a is the input value, and b is the bias value.

After that, each neuron applies an activation function on its value and transmits it to the next layer to ensure non-linearity in the transformation.

$$a^{(l)} = f(z^{(l)}) \quad (3.2)$$

In the end, the final layer processes the transformed data and produces an output. For classification tasks, a SoftMax function is typically used to output probabilities, and for regression tasks, the output is a continuous value without activation or using a linear activation.

3.2. Activation Functions

Activation Functions play a crucial role in artificial neural networks by converting an input signal into an output signal, subsequently serving as input to the next layer in the hierarchy [15]. Activation functions are essential components in multilayer perceptrons (MLPs), significantly contributing to the network's ability to learn and represent complex patterns. By introducing non-linearity into the model, these functions allow the MLP to approximate intricate relationships between inputs and outputs effectively. In the absence of activation functions, even a deep network made up of multiple layers would essentially behave like a single-layer linear model, severely restricting its capacity to capture complex, non-linear relationships inherent in real-world data. The choice of activation function can substantially impact the network's training dynamics and overall effectiveness, making it a critical consideration in the design of neural architectures [16]. Thus, including non-linear activation functions is not just advantageous; it is fundamental for unlocking the full potential of deep learning models in tackling complex tasks across diverse applications.

Different activation functions are used to solve different kinds of problems, as each function is better suited to a specific type of problem. I will describe the most used functions.

1- Sigmoid Function

It is the most widely used activation function as it is non-linear. The sigmoid function transforms the values in the range 0 to 1. It can be defined as:

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.3}$$

The sigmoid function is continuously differentiable and has a smooth S-shaped function.

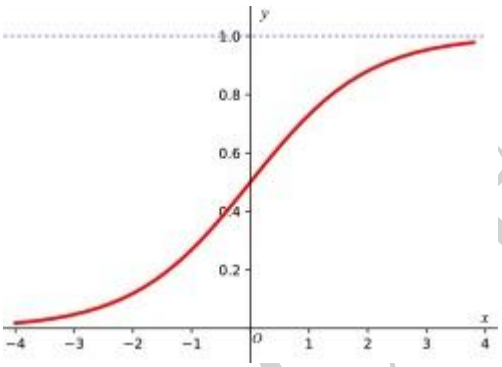


Figure 3. Sigmoid Function

Also, the sigmoid function is not symmetric about zero, which means that the signs of all neuron output values will be the same. Scaling the sigmoid function can improve this issue.

2- Hyperbolic Tangent function (TANH) Function

Tanh function is similar to the sigmoid function but symmetric around the origin. This results in different signs of outputs from previous layers, which will be fed as input to the next layer. It can be defined as:

$$f(x) = \frac{2}{1+e^{-2x}} - 1 \tag{3.4}$$

The Tanh function is continuous and differentiable; its values lie in the range -1 to 1. Compared to the sigmoid function, the gradient of the Tanh function is more steep. Tanh is preferred over the sigmoid function as its gradients are zero-centered and are not restricted to varying in a specific direction.

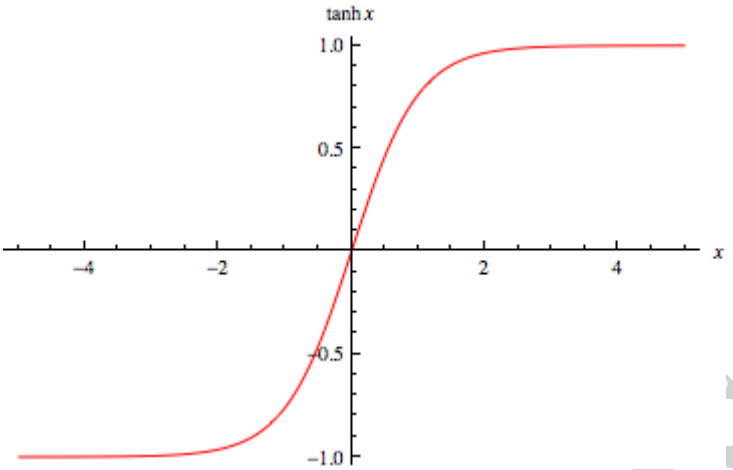


Figure 4. Tanh Function

3- Rectified Linear Unit (ReLU)
ReLU is a non-linear activation function that is widely used in neural networks. The success of ReLU
is based on its superior training performance over other activation functions, such as the sigmoid and the hyperbolic tangent. It can be defined mathematically as:

$$f(x) = \max(0, x) \tag{3.5}$$

ReLU is linear (identity) for all positive values and zero for all negative values. However, the output values range from 0 to infinity. One of the advantages of using ReLU is its simplicity. It can output a true zero value, allowing the activation of hidden layers in neural networks to contain one or more true zero values, increasing the representational sparsity of the network.

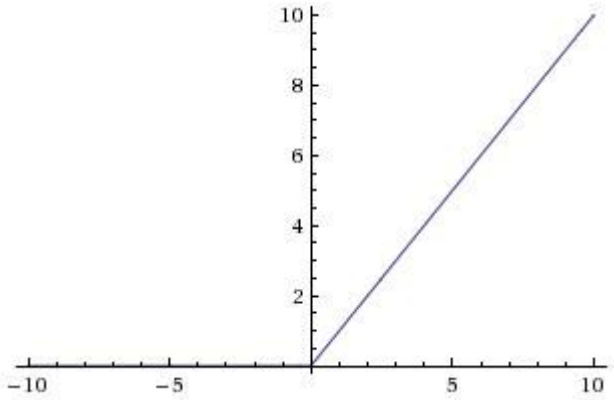


Figure 5. ReLU Function

4. Implementation and Results

4.1. Dataset

In this paper, I use a built-in data set from the Scikit project, an open source named IRIS data set. It contains four attributes named Sepal-length, Sepal-width, Petal-

length, and Petal-width, and three classes or species named Sentosa, Versicolor, and Virginic. Each class has 50 samples. The total samples of iris data set are 150 [17].

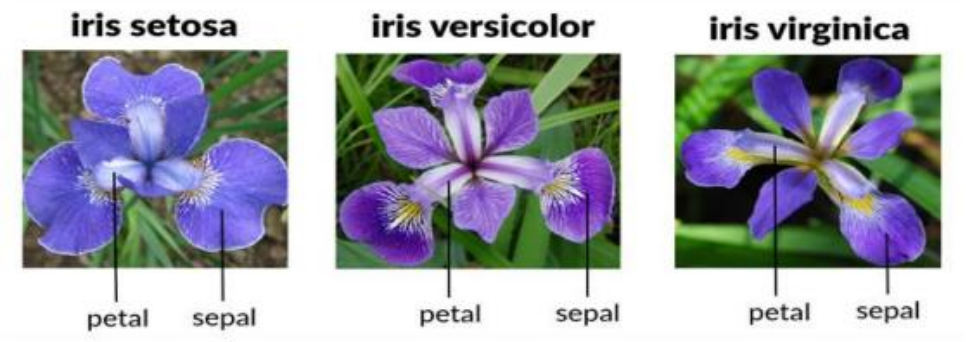


Figure 6. Three species of IRIS flower

There are five variables in this data set:
1. Sepal length: (It is used as input data, which is measured by centimeters) The range of the values is [4.3, 7.9]
2. Sepal width: (It is used as input data, which is measured by centimeters) The range of the values is [2.0, 4.40]
3. Petal length: (It is used as input data, which is measured by centimeters) The range of the values is [1.0, 6.9]
4. Petal width: (It is used as input data, which is measured by centimeters) The range of the values is [0.1, 2.5]
5. Class: (It is used as target data measured by centimeters) The class variable contains three groups or species named iris Sentosa, iris Versicolor, and iris Virginic.

4.2. Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a widely used dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional representation while retaining as much variance as possible. PCA achieves this by identifying the principal components—orthogonal directions of maximum variance in the dataset—and projecting the data onto these components [18]. Using PCA has a lot of benefits like:

- Dimensionality Reduction: PCA reduces the number of features while preserving essential information, making it easier to visualize and analyze data, especially in cases where high-dimensional data is challenging to interpret.
- Noise Reduction: PCA filters out minor variations and noise by focusing on the principal components with the highest variance, improving model robustness.
- Computational Efficiency: Lower-dimensional data requires fewer computations, which speeds up training and inference in machine learning models, including MLPs.

The first step in PCA analysis is subtracting the mean values from dataset Z to properly normalize and assign equal weights to the original dataset during the analysis phase [19].

$$X_{i,j} = Z_{i,j} - \overline{X_j} \quad (4.1)$$

Where $X_{i,j}$ is the standardized dataset, $Z_{i,j}$ is the data variable j in the sample unit i and $\overline{X_j}$ is the sample mean for the variable j.
The next step is to compute the covariance matrix. The covariance of two variables measures their correlation. The matrix has all-possible covariance pairs between m variables, and its diagonal includes all variances. Equation (4.2) represents the covariance of the standardized dataset X.

$$C_x = \frac{1}{n-1} XX^T \quad (4.2)$$

Then, Compute the eigenvalues λ_i and eigenvectors v_i of C, equation (4.3). The eigenvectors represent the principal components, and the corresponding eigenvalues indicate their significance.

$$C_{v_i} = \lambda_i v_i \quad (4.3)$$

After that, Select the top k eigenvectors (corresponding to the largest eigenvalues) to form the projection matrix $W = v_1, v_2, \dots\dots, v_k]$
In the end, Project the original data onto the new k-dimensional space.

4.3. Training Visualization

Since the MLP Neural Network is widely used in many applications, the need to understand the training process and the effects of changing the crucial MLP values (activation function, number of hidden layers…, etc.) is increased. Also, we should understand how missing or noisy data in the data set will affect the trained model.
First, I used a custom layer for the first hidden layer. The custom layer offers several benefits, particularly when we need more control or customization than standard layers can provide. Custom layers allow us to implement unique operations not supported by standard layers, such as adding non-standard mathematical transformations, implementing custom normalization methods, or creating specialized activations. We also can define and control trainable weights and biases explicitly. This is useful when we want non-standard weight initialization, constraints, or weight sharing across layers.
For the loss function, which is a mathematical function that measures how well a machine learning model's predictions match the actual target values, I used the Sparse Categorical Crossentropy function. The Sparse Categorical Crossentropy loss function is a variant of the categorical crossentropy loss used for classification problems with multiple classes. It is beneficial when your target labels are integer-encoded rather than one-hot encoded. Also, it works well with the SoftMax activation function in the output layer.
I made multiple tests to check the differences if I changed the active function, deleted data, or added noisy data to the data set.

4.3.1. ReLU Activation Function

In this section, I will describe the results of using the ReLU activation function. Figure 7 shows the result of the trained model after 10 epochs. We can see that many of the points are still wrongly classified, and we need more training steps to get a model with a higher accuracy. The accuracy value of this model was 0.8148,

and the loss value was high, 0.8036.

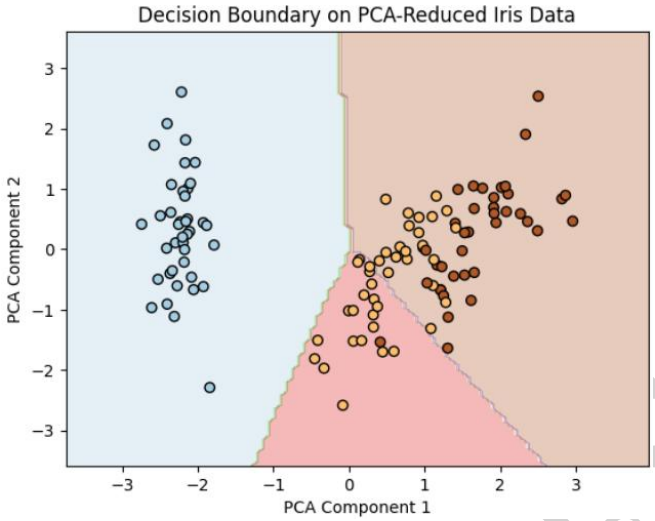


Figure 7. Trained model after 10 epochs using ReLU

After 50 epochs, the accuracy value got much higher (0.9630), and the loss value was lower (0.1738). By using some of the data from the data set as a validation set, the results were not bad; the validation accuracy value was 0.9167, and the validation loss was 0.2755, which means that the model still needs to be trained more to check if it will cause an overfitting problem or not. We can see in Figure 8 that the decision boundaries were changed a lot and arranged in a way that classifies more points correctly.

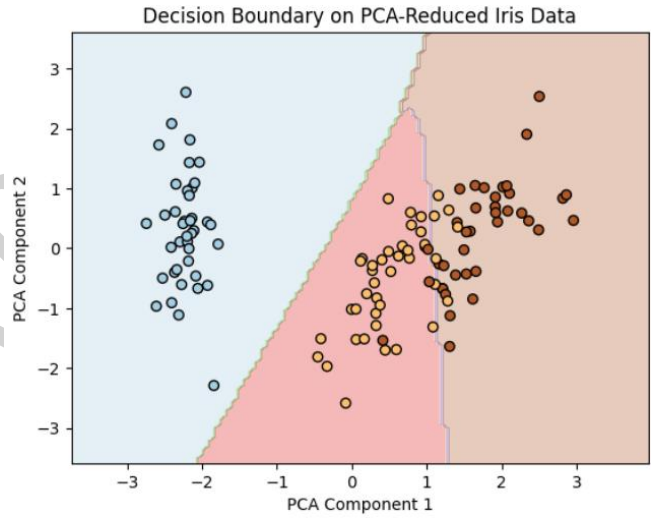


Figure 8. Trained model after 50 epochs using ReLU

I trained the model for more epochs until 100 epochs to check if the accuracy and loss values would change or not. In the last 10 epochs, the accuracy value reached 0.9815, but at the end of the training, it went back to 0.9630. The loss value decreased during the training until it reached 0.0716. Figure 9 (a) shows the decision boundaries when the accuracy equals 0.9722, and (b) shows the decision boundaries when the accuracy equals 0.9815. We cannot see any difference between both figures because the change is so small, which we can see with the help of the training process visualization.

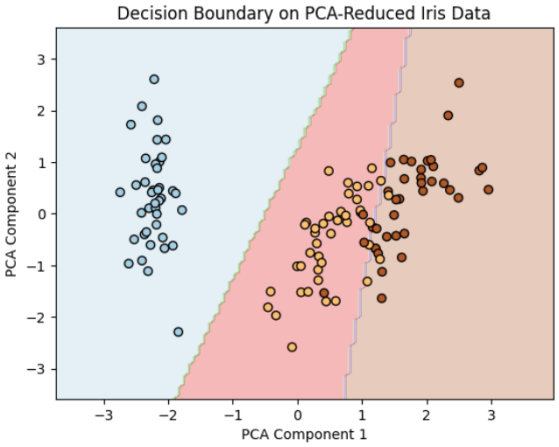
From the previous results, we can say that visualizing the decision boundaries during the training process can help us see how the model improves after multiple steps. Still, it will not show the small differences between two consecutive epochs. The results in Table 1 show that the model did not overfit the data, and the decision boundaries plot shows the same result.

Table 1. Trained model results using ReLU

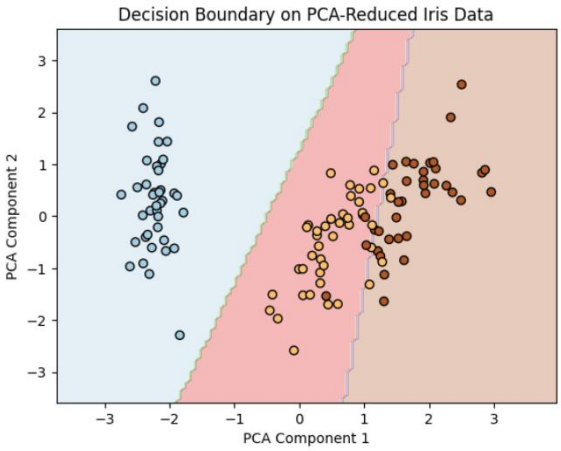
	10 Epochs	**50 Epochs**	**100 Epochs**
Accuracy	0.8148	0.9630	0.9630
Loss	0.8036	0.1738	0.0716
Val-accuracy	0.7500	0.9167	1.0000
Val-loss	0.8628	0.2755	0.0832

After that, I did another test to see the effect of deleting some data from the data set. I randomly deleted 20% of the data set and continued the process like in the previous test. Figure 10. shows the results of the trained model after 10, 50, and 100 epochs. The model was improving slower than the previous test as we had less data to use in training.

We can see in Table 2 how the accuracy and loss values were changing during the training process. The values show that we faced an overfitting problem as the accuracy at the end of the training was 1, but the validation accuracy was 0.8, which tells us that the model was overtrained by the training set and will not be able to detect new data correctly. However, using the visualization function, we were not able to see and detect the overfitting problem as we see that there are spaces in the figure which can be evaluated if we tried a new point, but it was not true in practice. So, the training process visualization is not efficient to detect overfitting.



(a) Accuracy = 0.9722

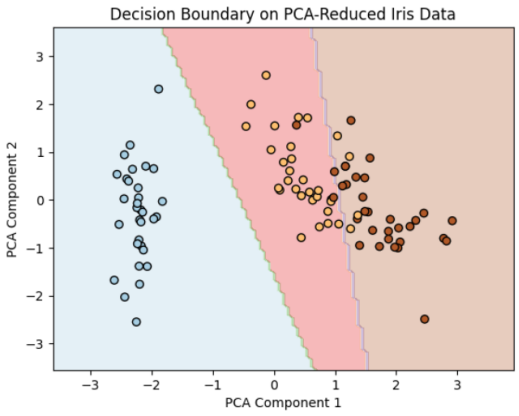


(b) Accuracy = 0.9815

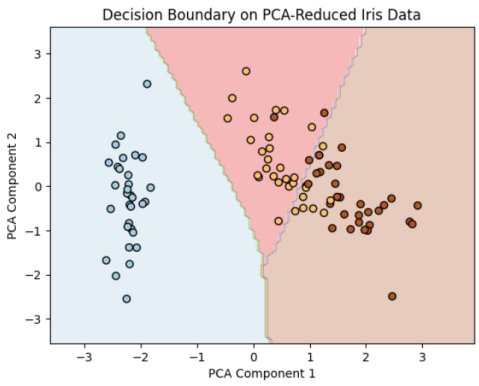
Figure 9. The difference between the decision boundaries

Table 2. Trained Model results using ReLU after deleting data

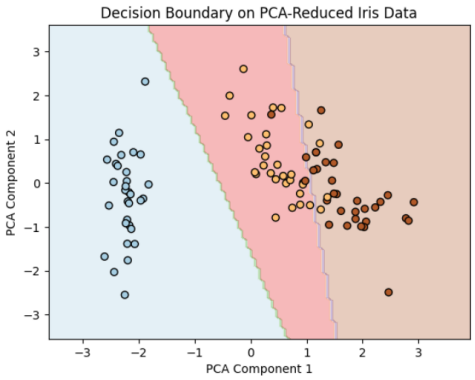
	10 Epochs	**50 Epochs**	**100 Epochs**
Accuracy	0.7791	0.9302	1.0000
Loss	0.8184	0.2375	0.0498
Val-accuracy	0.8000	0.9000	0.8000
Val-loss	0.8574	0.3825	0.2080



(a) After 10 Epochs



(b) After 50 Epochs



(c) After 100 Epochs

Figure 10. Visualizing the training process using ReLU after deleting data

Also, I wanted to check the effect of adding noisy data to the data set on the training process. After randomly adding some noisy data, I started to train the MLP model using the steps mentioned before. The model did not reach the point where we can say that it is an efficient model to use in this situation, as the highest accuracy value reached was 0.8682, which is not a good value. By using the visualizing function, we can see that the model is not efficient as there are a lot of data points that were classified wrongly, like in Figure 11.

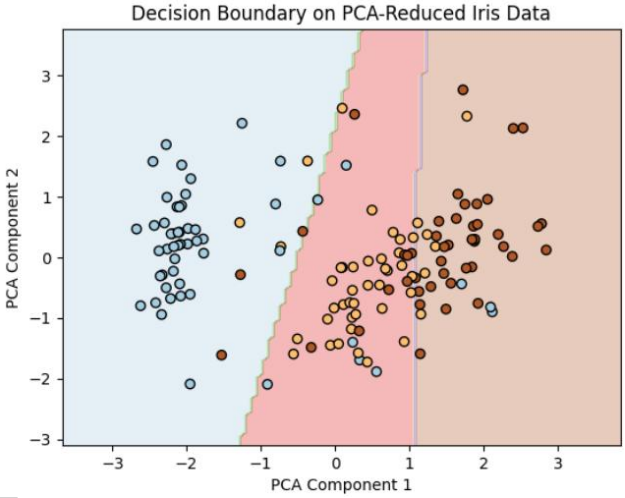


Figure 11. Noisy data effect on the training process

Table 3. Model results after adding noisy data

	10 Epochs	**50 Epochs**	**100 Epochs**
Accuracy	0.6899	0.8605	0.8682
Loss	0.8552	0.4935	0.3832
Val-accuracy	0.6000	0.8000	0.8000
Val-loss	0.9513	0.7020	0.5306

4.3.2. Sigmoid Activation Function

As there are many activation functions that we can use during the training process

of MLP, it is important to know the benefits of using each activation function and check which one is most suitable in this situation. For that, I created a new test by changing the activation function from ReLU to Sigmoid, checked the results, and visualized the training process like I did before.

After 10 epochs of training, we can see that the model is not trained well yet as the accuracy value was 0.6574 and the loss value was 0.9861, and we can see in Figure 12 that the model just found two classes instead of three, and almost half of the data points were classified wrongly, which is the same result of the accuracy of the model.

After 100 epochs, the model improved a lot but was still not efficient as it had a big value for the loss, which equaled 0.2286. In compression between using ReLU and Sigmoid, ReLU got a higher accuracy value and lower loss value, so we can say that ReLU was better than Sigmoid for this problem. Also, by checking Figure 13, we can see the difference between the decision boundaries for the model that used ReLU (Figure (a)) and the model that used Sigmoid (Figure (b)). We could decide which one is more accurate as Figure 13 (a) could classify more data points correctly and has a bigger area for classification.

Table 4. Trained model results using Sigmoid

	10 Epochs	**50 Epochs**	**100 Epochs**
Accuracy	0.6574	0.8889	0.9630
Loss	0.9861	0.4034	0.2286
Val-accuracy	0.5000	0.8333	0.9167
Val-loss	1.0502	0.5499	0.3751

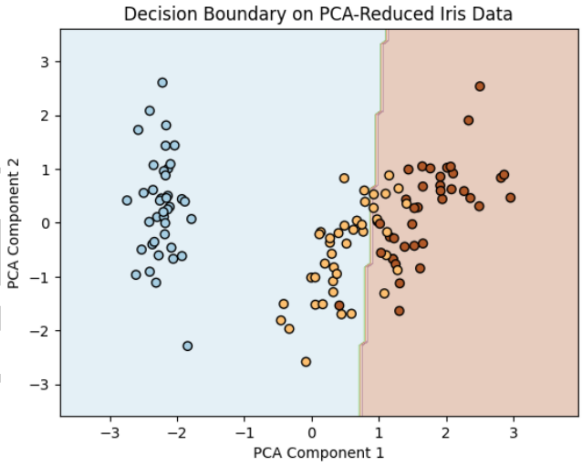
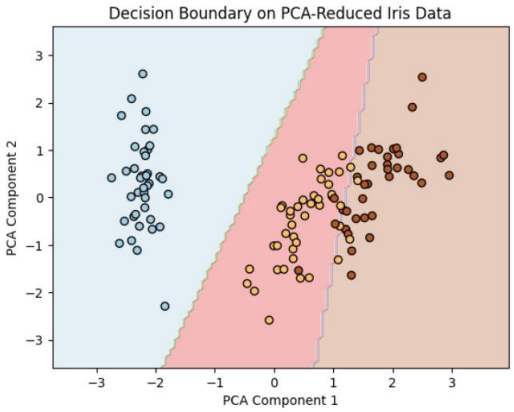
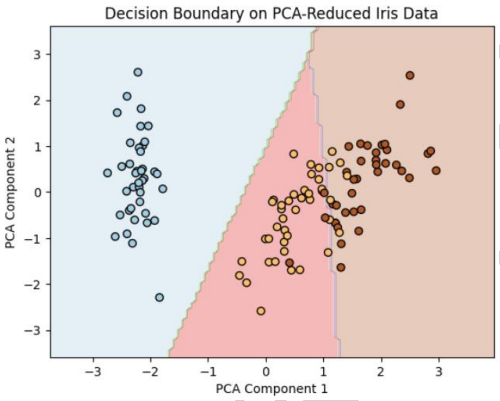


Figure 12. Trained model after 10 epochs using Sigmoid



(a) ReLu



(b) Sigmoid

Figure 13. Comparison between using ReLU and Sigmoid

5. Conclusion

After doing a lot of tests and checking the efficiency of visualizing the training process, I came up with multiple points:

1- Visualizing the training process can help us to understand how the model is training and how the decision boundaries are changing during the training process.
2- Visualizing the training process is good for checking if there are noisy data or outliers in the data set.
3- We can use the visualizing function to check the difference between using different activation functions and choosing the best one.
4- Visualizing the decision boundaries of the model cannot help to check overfitting.
5- It is not efficient to use the visualizing function to see the oscillation in the model during the training.

Refereces

[1] H. Taud and J. F. Mas, “Multilayer Perceptron (MLP),” in Geomatic Approaches for Modeling Land Change Scenarios, M. T. Camacho Olmedo, M. Paegelow, J.-F. Mas, and F. Escobar, Eds., Cham: Springer International Publishing, 2018, pp. 451–455. https://doi.org/10.1007/978-3-319-60801-3_27

[2] J. Sjöberg et al., “Nonlinear black-box modeling in system identification: a unified overview,” Automatica, vol. 31, no. 12, pp. 1691–1724, Dec. 1995, https://doi.org/10.1016/0005-1098(95)00120-8

[3] R. R. Hoffman, S. T. Mueller, G. Klein, and J. Litman, “Metrics for Explainable AI: Challenges and Prospects,” Feb. 01, 2019, arXiv: arXiv:1812.04608. https://doi.org/10.48550/arXiv.1812.04608

[4] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, and J. Zhu, “Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges,” in Natural Language Processing and Chinese Computing, J. Tang, M.-Y. Kan, D. Zhao, S. Li, and H. Zan, Eds., Cham: Springer International Publishing, 2019, pp. 563–574. https://doi.org/0.1007/978-3-030-32236-6_51

[5] A. Holzinger, A. Saranti, C. Molnar, P. Biecek, and W. Samek, “Explainable AI Methods - A Brief Overview,” in xxAI - Beyond Explainable AI: International Workshop, Held in Conjunction with ICML 2020, July 18, 2020, Vienna, Austria, Revised and Extended Papers, A. Holzinger, R. Goebel, R. Fong, T. Moon, K.-R. Müller, and W. Samek, Eds., Cham: Springer International Publishing, 2022, pp. 13–38. https://doi.org/10.1007/978-3-031-04083-2_2

[6] J. F. Mas, H. Puig, J. L. Palacio, and A. Sosa-López, “Modelling deforestation using GIS and artificial neural networks,” Environ. Model. Softw., vol. 19, no. 5, pp. 461–471, May 2004, https://doi.org/10.1016/S1364-8152(03)00161-0

[7] A. Apicella, F. Donnarumma, F. Isgrò, and R. Prevete, “A survey on modern trainable activation functions,” Neural Netw., vol. 138, pp. 14–32, June 2021, https://doi.org/10.1016/j.neunet.2021.01.026

[8] C. Rudin and J. Radin, “Why Are We Using Black Box Models in AI When We Don’t Need To? A Lesson From An Explainable AI Competition,” Harv. Data Sci. Rev., vol. 1, no. 2, Nov. 2019, https://doi.org/10.1162/99608f92.5a8a3a3d

[9] C. Rudin, “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,” Nat. Mach. Intell., vol. 1, no. 5, pp. 206–215, May 2019, https://doi.org/10.1038/s42256-019-0048-x

[10] S. Letzgus, P. Wagner, J. Lederer, W. Samek, K.-R. Müller, and G. Montavon, “Toward Explainable Artificial Intelligence for Regression Models: A methodological perspective,” IEEE Signal Process. Mag., vol. 39, no. 4, pp. 40–58, July 2022, https://doi.org/10.1109/MSP.2022.3153277

[11] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, “On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation,” PLOS ONE, vol. 10, no. 7, p. e0130140, July 2015, https://doi.org/10.1371/journal.pone.0130140

[12] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps,” Apr. 19, 2014, arXiv: arXiv:1312.6034. https://doi.org/10.48550/arXiv.1312.6034

[13] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization,” presented at the Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 618–626. Accessed: Feb. 01, 2025. [Online]. Available: https://openaccess.thecvf.com/content_iccv_2017/html/Selvaraju_Grad-CAM_Visual_Explanations_ICCV_2017_paper.html

[14] V. K. Ojha, A. Abraham, and V. Snášel, “Metaheuristic design of feedforward neural networks: A review of two decades of research,” Eng. Appl. Artif. Intell., vol. 60, pp. 97–116, Apr. 2017, https://doi.org/10.1016/j.engappai.2017.01.013

[15] S. Sharma, S. Sharma, and A. Athaiya, “ACTIVATION FUNCTIONS IN NEURAL NETWORKS,” Int. J. Eng. Appl. Sci. Technol., vol. 04, no. 12, pp. 310–316, May 2020, https://doi.org/10.33564/IJEAST.2020.v04i12.054

[16] A. D. Rasamoelina, F. Adjailia, and P. Sinčák, “A Review of Activation Function for Artificial Neural Network,” in 2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI), Jan. 2020, pp. 281–286. https://doi.org/10.1109/SAMI48414.2020.9108717

[17] S. A. Mithy, S. Hossain, S. Akter, U. Honey, and S. B. Sogir, “Classification of Iris Flower Dataset using Different Algorithms,” vol. 9, 2022, https://doi.org/10.26438/ijsrmss/v9i6.110

[18] H. Abdi and L. J. Williams, “Principal component analysis,” WIREs Comput. Stat., vol. 2, no. 4, pp. 433–459, 2010, https://doi.org/10.1002/wics.101

[19] N. Salem and S. Hussein, “Data dimensional reduction and principal components analysis,” Procedia Comput. Sci., vol. 163, pp. 292–299, Jan. 2019, https://doi.org/10.1016/j.procs.2019.12.111