Perceptron and Beyond*

Yiming Teng† and Kaichen Yin‡
Department of Physics, Université Paris-Saclay, 91405 Orsay, France
(Dated: June 13, 2025)

I. INTRODUCTION

The perceptron proposed by Frank Rosenblatt in 1957 [7] represents one of the most significant milestones in the history of machine learning. Designed to be a binary classifier, the perceptron serves as a foundational building block for understanding more complex machine learning algorithms. The Mark I perception [8], in particular, represents a pioneering attempt to mechanize pattern recognition, elucidating the potential of a computer to imitate human-like decision-making, thus marking the beginning of the thrilling development of artificial intelligence.

This lab report focuses on the implementation of the Mark I perceptron, exploring its architecture, learning process, and application on picture recognition (square-circle classification). By simulating its behavior, we aim to understand the principles of supervised learning and the limitations inherent to single-layer networks. Furthermore, we also generalized the architecture to a multi-layered case. We replaced the original international training algorithm with a modern backpropagation (BP) algorithm, improved the performance of our model, and applied our prototype to the task of neutron/γ discrimination with the dataset provided during the course.

Our simulations demonstrate the capability of the original Mark I and our improved architecture on the classification task over linearly separable datasets. Through this implementation, we delve into the theoretical significance of the perceptron in the evolution of artificial intelligence while laying the groundwork for exploring more advanced neural network models.

II. THEORETICAL FRAMEWORK

A. Prototypical Mark I perceptron

1. Achitecture

The original Mark I perception described in [8] consists of four key points:

1. A retina consisting of light-sensitive units outputting a unit positive excitatory signal or a unit negative inhibitory signal once stimulated.
2. An association layer, each unit has a fixed number of input connections from the retina, but the connected retina points are randomly selected from the sensory units. Each association unit delivers an output pulse of a certain magnitude once the algebraic sum of the signal received from the retina exceeds a threshold $\boldsymbol{\theta}^a \equiv \{\theta_i^a\}$.
3. A binary response unit connected to all association layer units, which is turned to be at state-1 once the sum of the signals it received exceeds another threshold θ^r and at state-0 once the criterion is not satisfied.
4. The training of this system amounts to adjusting the output threshold θ^r and outputting magnitude for each association layer unit.

It's worth noting that the Mark I perception was intended to be realized as a massive electric circuit, and the archaic terminology used in the original literature is very electric-engineering-oriented. Nevertheless, we can translate the description to a modern and mathematically precise one:

- A retina sensory unit: An element of the input data vector $\boldsymbol{v}$, let's denote its dimension as n.
- Random excitatory or inhibitory connections from the retina to the association layer: This neural network contains one hidden layer whose dimension is m. The values received by each unit correspond to an element of the m-dimensional vector $\boldsymbol{h}^{\text{in}}$ defined to be

$$\boldsymbol{h}^{\text{in}} \equiv \mathbf{W}\boldsymbol{v}, \tag{1}$$

  where $\mathbf{W}$ is an $m \times n$ weight matrix whose elements are randomly sampled from $\{+1, -1\}$ upon the initialization of the network. Notice that for the prototypical perceptron, the propagation of the data from the input layer to the hidden layer only serves as a dimensionality reduction; hence we will set $\dim \boldsymbol{h} < \dim \boldsymbol{v}$.
- Each association unit i outputs a signal once the sum of the stimulations it receives exceeds a certain threshold θ_i^a: The activation function of each unit in the hidden layer is Heaviside step-function $\vartheta(h_i^{\text{in}} - \theta_i^a)$, and the output of the hidden layer is denoted to be

$$\boldsymbol{h}^{\text{out}} = \{\vartheta(h_i^{\text{in}} - \theta_i^a)\} \equiv \vartheta(\boldsymbol{h}^{\text{in}} - \boldsymbol{\theta}^a) \tag{2}$$

---

* Lab Report for Mathematical Statistical Methods at Université Paris-Saclay
† yiming.teng@universite-paris-saclay.fr
‡ kaichen.yin@universite-paris-saclay.fr

- Different units of the association layer output signals of different magnitudes: The signal magnitude that the output binary unit received from the unit i of the hidden layer is weighted by an element w_i of another $1 \times m$ weight matrix $\mathbf{w}$, hence the value received by the binary unit can be characterized as

$$x = \mathbf{w}\boldsymbol{h}^{\text{out}}. \tag{3}$$

- The output unit has binary states depending on whether the stimulation exceeds a threshold θ_r: The output value of the neural network is given by

$$\hat{y} \equiv \vartheta(x - \theta^r). \tag{4}$$

- The training keeps the weight matrix $\mathbf{W}$ and the hidden layer thresholds $\boldsymbol{\theta}^a$ fixed but varies the weight matrix $\mathbf{w}$ and the output threshold θ^r according to our training algorithm.

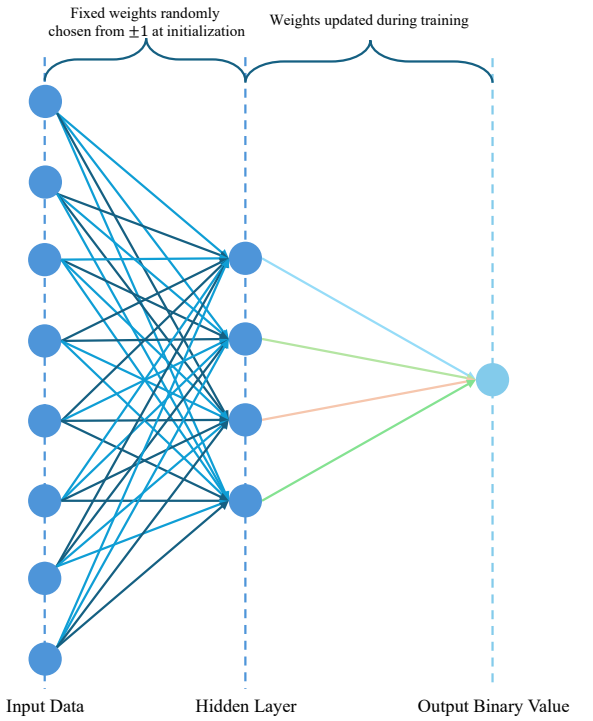

FIG. 1. The architecture of our Mark I perceptron equivalent model.

Following this dictionary, the architecture of the perceptron-equivalent neural network that we built is demonstrated in Figure 1. It's worth emphasizing that although this architecture superficially contains a single hidden layer (and we indeed name it as a "hidden layer"), the fact that we don't adjust the weight matrix $\mathbf{W}_1$ unfortunately, sentences that this network is *de facto* a single-layer feed-forward network and naming the intermediate association layer as a "hidden layer" is an abuse of terminology. We shall discuss the generalization of this structure to a real single-hidden-layer (double-layer) feed-forward network later in section II B.

2. *Training algorithm*

Denote the size of our training dataset $\{(\boldsymbol{v}_t, y_t)\}$ as T, where $\boldsymbol{v}_t$ is an input data vector, and $y_t = \pm 1$ is the label associated to it. By training the perceptron, we mean to find suitable values for $\mathbf{w}$ and θ^r so that the objective function

$$F(\mathbf{w}, \theta^r) \equiv \frac{1}{T}\sum_{t=1}^{T} \max[0, -y_t \vartheta(\mathbf{w}\vartheta(\mathbf{W}\boldsymbol{v}_t - \boldsymbol{\theta}^a) - \theta^r)] \tag{5}$$

defined over the dataset takes the minimal value [4].

The benefit of the effective single-layer nature of the perceptron prototype is that for any t, $(\mathbf{w}, \theta^r) \mapsto (\mathbf{w}\vartheta(\mathbf{W}\boldsymbol{v}_t - \boldsymbol{\theta}^a) - \theta^r)$ is linear and thus convex [5]. Moreover, the max-function and the *vartheta* function all preserve convexity; hence F is convex yet not differentiable. Nevertheless, this optimization problem can be solved by a stochastic subgradient descent technique whose implementation is a neat iteration algorithm; our pseudocode for it is shown in Algorithm 3.

Algorithm 1 Perceptron algorithm

```
Set an appropriate learning rate r
Randomly initialize W, θ^a, w_0, θ^r_0
for t = 1, ..., T do
    /* Calculate the actual output */
    ŷ_t = ϑ(w_{t-1} ϑ(W v_t − θ^a_{t-1}) − θ^r_{t-1})
    /* Update weights and the output threshold */
    w_t = w_{t-1} + r(y_t − ŷ_t) ϑ(W v_t − θ^a)
    θ^r_t = θ^r_{t-1} + r(y_t − ŷ_t)
end for
```

3. *Limitations*

It's obvious that the real dataset fed in the single-layer network corresponding to Mark I perceptron can be denoted as $\{(\boldsymbol{h}_t^{\text{out}}, y_t)\} \equiv \{(\vartheta(\mathbf{W}\boldsymbol{v}_t - \boldsymbol{\theta}^a), y_t)\}$. Suppose there are N hidden-layer units, then $\mathbf{w} \in \mathbb{R}^N$ and each $\boldsymbol{h}_t^{\text{out}} \in \mathbb{R}^N$. Our training algorithm aims to find a suitable $\mathbf{w} \in \mathbb{R}^N$ and $\theta^r \in \mathbb{R}$ such that

$$\boldsymbol{h}_t^{\text{out}} \mapsto \vartheta(\mathbf{w}\boldsymbol{h}_t^{\text{out}} - \theta^r) = y_t, \quad \forall t \in [1, T]. \tag{6}$$

Note $\mathbf{w}\boldsymbol{x} - \theta^r = 0$ just describes a hyperplane in $\mathbb{R}^N$, where $\mathbf{w}^T$ corresponds to a normal vector of it. Therefore, $\mathbf{w}\boldsymbol{h}_t^{\text{out}} - \theta^r > 0$ means $\boldsymbol{h}_t^{\text{out}}$ locates in the half-space separated the hyperplane where $\mathbf{w}^T$ is point at, while $\mathbf{w}\boldsymbol{h}_t^{\text{out}} - \theta^r < 0$ indicates $\boldsymbol{h}_t^{\text{out}}$ is on the other side.

This observation leads to the conclusion that the prototypical perceptron is a linear classifier; it's more applicable if the more points in the input dataset $\{\boldsymbol{h}_t^{\text{out}}\}$ can be

separated by a hyperplane in $\mathbb{R}^N$. It's indeed proved by [6] that a perceptron will converge on any linearly separable dataset after finite training steps. On the contrary, once the dataset is not linearly separable, the training won't converge, and a limited accuracy is expected after however many training epochs.

B. Improved perceptron: Double-layer feed-forward network

1. Universial approximation theorem

[3] generally and rigorously discussed the limitations of the conventional single-layer perceptron architecture and concluded that any minor modifications to the model within the effective single-layer structure can not influence the fact that it's only applicable to a very narrow range of problems. Nevertheless, following practices on multi-layer architectures implied a stronger capability inherited by these models, and in 1989, [1] rigorously proved the incredible potential of multi-layer networks known as one of the earliest universal approximation theorems: Multi-layer feed-forward networks with as few as one hidden-layer and using arbitrary non-polynomial activation functions can approximate almost any function of interest to any desired accuracy, provided sufficiently many hidden units are available.

Since the architecture of the Mark I perceptron is indeed a single-hidden-layer (double-layer) feed-forward network once the weight matrix $\mathbf{W}$ and hidden-layer thresholds $\boldsymbol{\theta}^a$ are updated during training, we can keep the architecture topology in Figure 1 invariant to improve the model so that it suits the universal approximation theorem in the large-hidden-layer-size limit; hence this network is expected to behave a better classification ability for a finite hidden-layer size. However, the original iterative optimization algorithm is futile for training these parameters, and we need a new algorithmic strategy.

2. Backpropagation algorithm

The Mark 1 Perceptron stands out for its notable feature: it includes a hidden layer, making it a two-layer neural network. However, the weights and threshold parameters of this hidden layer are randomly initialized and fixed, remaining unchanged throughout the subsequent training process. This characteristic provides the Mark 1 Perceptron with enhanced classification capabilities compared to single-layer neural networks. However, it also introduces a limitation—the weights in the hidden layer are not trainable, leading to considerable randomness. The backpropagation algorithm can be employed to address this limitation and optimize the hidden layer parameters.

The backpropagation (BP) algorithm is a well-known algorithm for training neural networks, which uses the chain rule of derivatives to calculate gradients for each layer of the network. This algorithm mainly consists of four key steps: forward propagation, backward propagation, and gradient descent.

For a general backpropagation algorithm, we consider an l-layer neural network. For the layer k , input is $\boldsymbol{x^{(k-1)}}$, the weights and thresholds are $\boldsymbol{w^k}$ and $\boldsymbol{\theta^k}$,and $\boldsymbol{h^k = w^k x^{(k-1)} + \theta^k}$. The activation function of this layer is $f(\boldsymbol{h})$, and the loss function is $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$.

For the training of a single sample in this neural network, the backpropagation algorithm can be expressed as [2]:

Algorithm 2 Backpropagation Algorithm

Input: Network with l layers, input x, target output y, learning rate r.
Initialize: Weights $\boldsymbol{w}$ and biases $\boldsymbol{\theta}$ randomly.
Forward propagation:
for each layer k from 1 to l **do**
  $\boldsymbol{h^{(k)} = w^{(k)} x^{(k-1)} + \theta^{(k)}}$
  $\boldsymbol{x^{(k)} = f(h^{(k)})}$
end for
Output: $\hat{\boldsymbol{y}} = \boldsymbol{x^{(l)}}$.
Compute loss: $Loss = L(\hat{\boldsymbol{y}}, \boldsymbol{y})$.

Backward propagation:
Output layer gradient $\boldsymbol{g^{(l)} = \nabla_{\hat{y}} L(\hat{y}, y) \odot f'(h^{(l)})}$.
for each layer k from $l-1$ to 1 **do**
  $\boldsymbol{g^{(k)} = (w^{(k+1)})^T g^{(k+1)} \odot f'(h^{(k)})}$
end for

Gradient Descent:
for each layer k from 1 to l **do**
  $\boldsymbol{w^{(k)} = w^{(k)} - r g^{(k)} (x^{(k-1)})^T}$
  $\boldsymbol{\theta^{(k)} = \theta^{(k)} - r g^{(k)}}$
end for

In this algorithm, "$\odot$" between two matrices is called the element-wise product or Hadamard product [2].

For the Mark 1 Perceptron, to introduce the Backpropagation algorithm, we choose a continuous loss function as

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{2} \sum_{t=1}^{T} ||\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t||^2, \tag{7}$$

and select the activation function as the sigmoid function

$$f(\boldsymbol{h}) = sigmoid(\boldsymbol{h}) = \frac{1}{1 + e^{-\boldsymbol{h}}}. \tag{8}$$

Therefore, the training algorithm for the Mark 1 Perceptron with the BP algorithm is as follows:

Algorithm 3 Perceptron algorithm with BP

Set an appropriate learning rate r
Randomly initialize $\mathbf{W}$, $\boldsymbol{\theta}^a$, $\mathbf{w}_0$, θ_0^r
for $t = 1, ..., T$ **do**
    `/* Forward propagation */`
    $\boldsymbol{x}_t^{(1)} = sigmoid(\mathbf{W}\boldsymbol{x}_t - \boldsymbol{\theta}_{t-1}^a)$
    $\hat{y}_t \quad = sigmoid(\mathbf{w}_{t-1}\boldsymbol{x}_t^{(1)} - \theta_{t-1}^r)$

    `/* Backward propagation*/`
    $g_2 = (\hat{y}_t - y_t) \cdot y_t(1 - y_t)$
    $\boldsymbol{g_1} = \mathbf{w}_t^T \cdot g_2 \cdot \boldsymbol{x}_t^{(1)} \odot (1 - \boldsymbol{x}_t^{(1)})$

    `/*Update weights and threshold*/`
    $\mathbf{w}_t = \mathbf{w}_{t-1} - rg_2(\boldsymbol{x}_t^{(1)})^T$
    $\theta_t^r = \theta_{t-1}^r - rg_2$
    $\mathbf{W}_t = \mathbf{W}_{t-1} - r\boldsymbol{g_1}(\boldsymbol{x_t})^T$
    $\boldsymbol{\theta_t^a} = \boldsymbol{\theta_{t-1}^r} - r\boldsymbol{g_1}$
end for

III. SIMULATION RESULT

A. Mark I perceptron

1. *Square-circle classification*

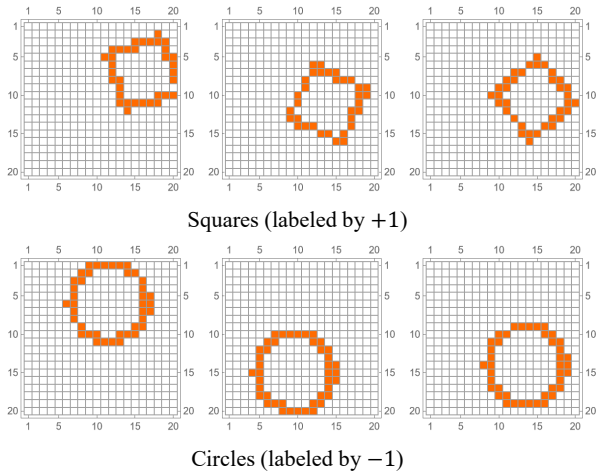

FIG. 2. Illustration of the input data of our perceptron.

We design a square-circle classification task for our prototypical Mark I perceptron model. The input data are 20×20 binarized matrices representing rasterized pictures of a randomly tilted square or a circle, and a square is labeled by +1. In contrast, a circle is tagged with −1 (as illustrated in Figure 2). We generated 10000 matrices for square and the same number for circle in the file `Square_dataset_generation.m` and stored them in `square_dataset_size20_num10000.mat`.

Following the setup discussed in section II A, we implemented this model in the file `perceptron_Mark_1.m`. The squares and circles are randomly arranged into a 20000-large dataset for training. The learning rate is set to $r = 0.05$. During training, we focus on keeping track of the error rate defined by

$$\text{error rate} \equiv \frac{1}{T}\sum_{t=1}^{T}|y_t - \hat{y}_t|. \tag{9}$$

The evolution of the error rate as the training epoch grows is plotted in Figure 3.

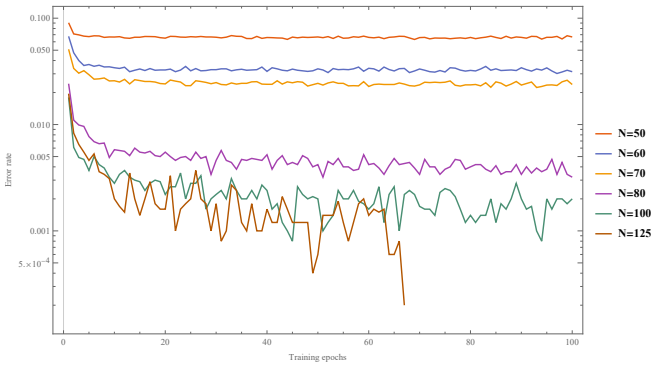

FIG. 3. Error rate curve for different hidden-layer sizes N for the square-circle discrimination task.

We find that for small values of the hidden-layer size N, the training never converges, and the error rate tends to fluctuate around a fixed value (we shall name it "stable error rate") as the training epoch grows. N getting larger drastically lowers the stable error rate, and we find that a critical value N_c exists above which the training converges in a rather small number of steps. For our case, $N_c = 125$ and the corresponding error rate curve are shown in Figure 3.

The intriguing behavior of the error rate function deserves further analysis. We highlight the key feature of Figure 3:

- The existence of a critical value N_c.

It's tempting to interpret this phenomenon as a sort of phase transition on the linear separability, and we will argue for it semi-quantitatively. Notice that our model setting always has $\dim \boldsymbol{v} > \dim \boldsymbol{h}$, hence the propagation of the data through the first connecting weights $\boldsymbol{v} \mapsto \boldsymbol{h} = \vartheta(\mathbf{W}\boldsymbol{v} - \boldsymbol{\theta}^a)$ is effectively a dimensionality reduction and is inevitably associated with an information loss. Intuitively, such an information loss will "blur" the distribution of the original data $\{\boldsymbol{v}_t\}$, lowering the linear separability of the data $\{\boldsymbol{h}_t^{\text{out}}\}$ which the core of the perceptron is classifying in $\mathbb{R}^N$.

To verify our conjecture, we projected the output data $\{\boldsymbol{h}_t^{\text{out}}\}$ onto the hyperplane spanned by the weight vector $\mathbf{w}^T$ obtained by 100 training epochs and a randomly chosen vector $\boldsymbol{w}_\perp$ orthogonal to $\mathbf{w}^T$ obtained by Gram-Schmidt orthogonalization for different hidden-layer sizes N. This is also implemented in `perceptron_Mark_1.m`. Our result is shown in Figure 4, which irrefutably supports our hypothesis, and $N_c = 125$ indeed corresponds

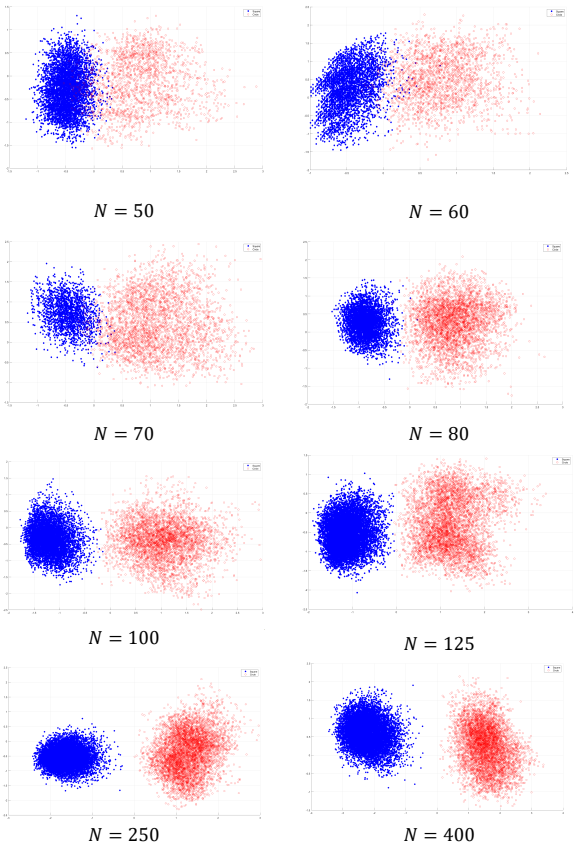

FIG. 4. Scatter plot of the data $\{\boldsymbol{h}_t^{\text{out}}(N)\}$ projected on the hyperplane spanned by $\boldsymbol{w}_\perp$ and $\mathbf{w}^T$ demonstrating the emerging linear separability as N increases. Blue spots represent squares in the data, while red ones correspond to circles.

to a critical size where $\{\boldsymbol{h}_t^{\text{out}}\}$ starts to be linearly separable.

Our analysis strongly implicates that the hidden layer's size in the prototypical perceptron will severely affect its performance on classification tasks by reducing the linear separability of the data processed by the core architecture of the classifier.

2. *Neutron/γ discrimination*

We now apply this perceptron to discriminate neutron and γ-ray signal provided in the database in the file `perceptron_Mark_NeutronGamma.m`. We again tracked the error rate evolution for different hidden-layer sizes during training, and the result is plotted in Figure 5.

Compared to the previous square-circle classification task, here the error rate evolution exemplifies features:

- The error rates get stabilized after only a few training epochs.

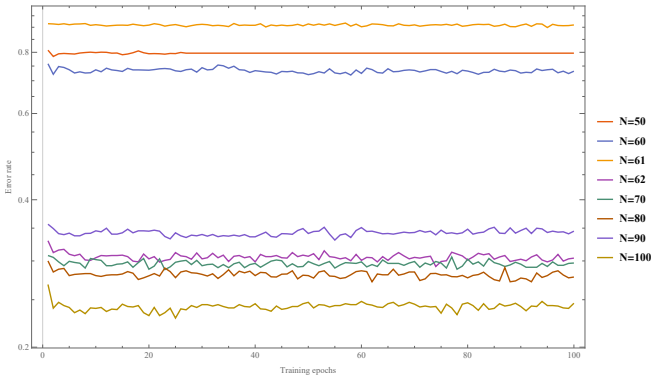

FIG. 5. Error rate curve for different hidden-layer sizes N for the neutron/γ discrimination task.

- An error rate gap between $N = 61$ and $N = 62$.
- Error rate can never be optimized to zero for whatever values of N and training epochs.

These observations lead to the conjecture that this dataset is not completely linearly separable, and there's a sudden linear separability increase from $N = 61$ to $N = 62$.

This conjecture here can again be verified by projecting the output data from the hidden layer onto a hyperplane constructed as before. The scatter plot is shown in Figure 6, and we indeed see a sudden increase in separability for $N = 62$, and the data remain linearly un-separable for larger N.

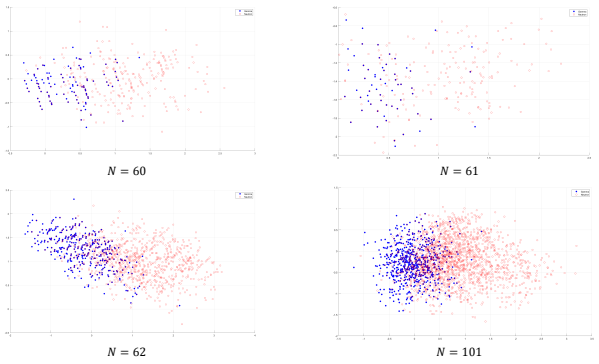

FIG. 6. Scatter plot of the output data from the hidden layer onto a hyperplane in the neutron/γ discrimination task. Blue spots represent γ-ray data, whereas red ones are neutron data.

B. Double-layer feed-forward network

To further optimize the weights and threshold parameters of the first layer of the Mark 1 Perceptron and improve its classification performance, we utilize the backpropagation algorithm described in section II B for training. To enable a direct comparison with

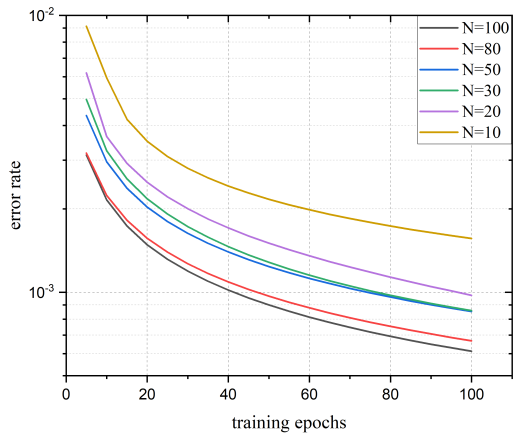

FIG. 7. Error curve of the Mark 1 Perceptron for the square-circle classification task after applying the BP algorithm.

the original algorithm, we conducted the same square-circle classification task using the identical dataset, `square_dataset_size20_num10000.mat`. Since the sigmoid function was used as the activation function, with its output ranging between 0 and 1, we assigned the following labels: squares are labeled as 0, and circles are labeled as 1.

We implemented this model in the file `perceptron_Mark1_BP.m`. To improve training speed and reduce the error rate, we carefully selected the learning rate to be $r = 0.5$. The evolution of the error rate as the number of training epochs increases is shown in Figure 7.

Compared to the results in Figure 3, the error evolution curve using the BP algorithm demonstrates significant improvements.

Firstly, the fluctuations in the curve throughout the training process are significantly reduced compared to Figure 3, and it exhibits an overall monotonic decreasing trend. This indicates that the BP algorithm can suppress randomness and enhance the stability of the optimization process.

Secondly, for smaller hidden layer sizes, such as $N = 20, 30$, the BP algorithm achieves an error rate as low as 10^{-3}, whereas the original Mark 1 struggles to reduce the error rate below 10^{-2} under such conditions. This highlights the algorithm's ability to significantly improve the classification performance of perceptrons with smaller hidden layers.

These observations collectively demonstrate the efficiency and stability of the BP algorithm in enhancing the training performance of the Mark 1 Perceptron. We briefly summarize the improvements brought by the BP algorithm as follows:

- Suppressing fluctuations and randomness during the training process, therefore improving the stability of the optimization process;
- Enhancing the performance of networks with small size of hidden layer, demonstrating good classification performance even with $N = 30$.

IV. CONCLUSION

In this report, we explored the implementation of the Mark I perceptron and its improvement through a double-layer feed-forward network trained with the back-propagation algorithm. Our findings highlight the fundamental limitations of the single-layer perceptron, such as its reliance on linear separability, and the critical impact of hidden layer size on classification performance.

The enhanced model, trained with backpropagation, demonstrated significant improvements in both training stability and accuracy, especially for datasets with higher complexity. These results underline the importance of optimization algorithms and multi-layer architectures in overcoming the limitations of early neural network models.

The perceptron remains a foundational concept in machine learning, offering valuable insights into supervised learning principles and paving the way for modern neural network advancements. This study emphasizes its historical and practical significance, while showcasing the potential of contemporary techniques to extend its applications to more complex problem domains.

V. ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to Prof. Pierre Desesquelles for his exceptional teaching of the Mathematical & Statistical Methods course, which introduced us to a wide range of mathematical computations and statistical techniques. His lectures have inspired us to apply these interesting and powerful methods in our own projects.

We are also grateful to classmates in the MSM course for their insightful discussions and helpful communication.

Finally, we would like to thank our family and friends for their encouragement and understanding throughout our journey of studying abroad in France.

Appendix A: Data Availability

All the source code for this project can be accessed through the repository `https://github.com/BlueMagpieKC/Project_Perceptron.git`

[1] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2(5):359–366, January 1989.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[3] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 1969.

[4] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive computation and machine learning. The MIT Press, Massachusetts, second edition, 2018.

[5] A function $f(\boldsymbol{x})$ is convex if $\forall \alpha \in [0,1]$, $f(\alpha\boldsymbol{x}+(1-\alpha)\boldsymbol{y}) \leq \alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{y})$.

[6] Albert J. Novikoff. On Convergence Proofs for Perceptrons. *Office of Naval Research*, January 1963.

[7] Frank Rosenblatt. The Perceptron—a perceiving and recognizing automaton. *Cornell Aeronautical Laboratory*, Report 85-460-1, January 1957.

[8] Frank Rosenblatt. Perceptron Simulation Experiments. *Proceedings of the IRE*, 48(3):301–309, March 1960.