Computational (AI) Models : Concepts, Developments and Beyond*

John Sum
Institute of Technology Management
National Chung Hsing University
Taichung 402, Taiwan

April 17, 2025

Abstract

One should be noted that an AI model is in essence a computational model. Given an input vector with numbers, the AI model computes the output vector with numbers. In this article, the key concepts on computational models are presented with introducing a few computational AI models. Perceptron learning rule and backpropagation learning rule are introduced. Simulation results are shown to illustrate the properties of those learning rules which are associated with their computational models, namely simple Perceptron and multilayered Perceptron (MLP). The concepts of model complexity and weight decay are remarked. Furthermore, recurrent networks are introduced and highlighted their potential applications in problems regarding sequence generations. For the applications of language processing, the difficulties in getting a numerical vector for a word or a phrase is highlighted. Subsequently, aspects of deep neural networks are introduced with highlights on the introduction of rectified linear unit (ReLU) as neurons in the model and the Softmax as the output neurons. The development process of a computational model and its applications is presented. Finally, comparisons among a human brain, an electronic brain and a computer are delineated.

*This report is prepared for the MBA and EMBA students in the National Chung Hsing University who have minimum mathematical foundation and who have taken either the course on *AI and Machine Learning* (MBA) or the course on *Intelligent Technology Management* (EMBA) in 2025 Spring.

Contents

List of Figures

List of Tables

1 McCulloch-Pitts Neuronal Networks

McCulloch-Pitts model is the first mathematical model proposed by W. McCulloch and W. Pitts in 1943 [2]. This model abstracts the *all-or-none* property of a neuron – *If the stimulus feeding to a neuron is larger enough, the neuron fires.*

1.1 McCulloch-Pitts Neuron Model

Consider a neuron with n inputs and let $x_1, \cdots, x_n$ be the inputs. $x_i \in \{0, 1\}$ for $i = 1, \cdots, n$. Let $y = f(\mathbf{x})$ be the neuron output. The output is defined as follows :

$$f(\mathbf{x}) = h\left(\sum_{i=1}^{n} w_i x_i - b\right), \tag{1}$$

where

$$h(u) = \begin{cases} 1 & \text{if } u > 0, \\ 0 & \text{if } u \leq 0. \end{cases} \tag{2}$$

1.2 2-Input-1-Output M-P Neuron

Figure 1 shows a model with two inputs. In the figure, w_1 and w_2 are called the synaptic weights. They act like scaling factors controlling the effects of the inputs to the neuron. b is called the threshold (or bias). If the weighted sum of the inputs is larger than the threshold b, the neuron fires (equivalently, $f(\mathbf{x}) = 1$. The $h(\cdot)$ in the neuron is a step function as defined in (2).

One should be noted that the inputs are all binary variables which are non-negatives. If the value of w_1 (resp. w_2) is positive, the effect of x_1 (resp. x_2) to the neuron is *excitatory*. If the value of w_1 (resp. w_2) is negative, the effect of x_1 (resp. x_2) to the neuron is *inhibitory*.

1.3 Geometrical Interpretation

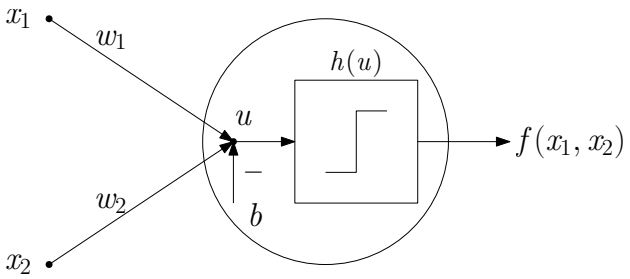
For a two-input-one-output neuron as shown in Figure 1, its mathematical model can simply be given by

$$f(x_1, x_2) = h(\underbrace{w_1 x_1 + w_2 x_2 - b}_{u(x_1, x_2)}). \tag{3}$$

Here, the function $u(x_1, x_2)$ inside $h(\cdot)$ is called a decision boundary. It partitions a 2-D plane into two parts. On one side of the decision boundary, $h(u(x_1, x_2)) > 1$. On the other side, $h(u(x_1, x_2)) < 0$.

Figure 2 shows the two examples, in which

$$\begin{aligned} u(x_1, x_2) &= x_1 + x_2 - 1. \\ u(x_1, x_2) &= -2x_1 + \frac{8}{3} x_2 - 1. \end{aligned}$$



$$f(x_1, x_2) = h(\underbrace{w_1 x_1 + w_2 x_2 - b}_{u}).$$

$$h(u) = \begin{cases} 1 & \text{if } u > 0 \\ 0 & \text{if } u \leq 0. \end{cases}$$

Figure 1: A McCulloch-Pitts model of a neuron with two inputs. Here, w_1 and w_2 are the synaptic weights; b is called the bias and $h(\cdot)$ is a step function. If the value of w_1 (resp. w_2) is positive, the effect of x_1 (resp. x_2) to the neuron is excitatory. If the value of w_1 (resp. w_2) is negative, the effect of x_1 (resp. x_2) to the neuron is inhibitory.

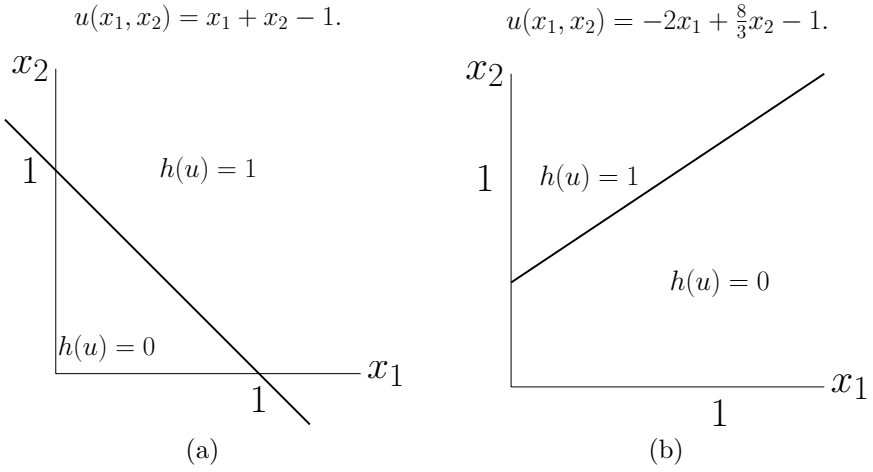
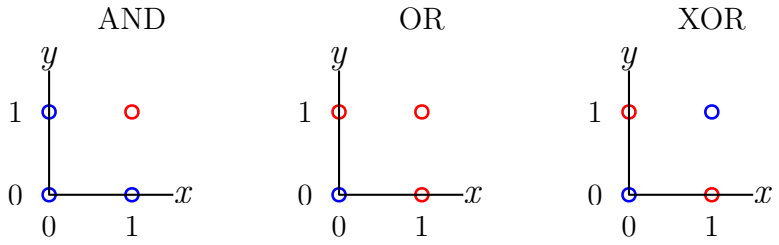


Figure 2: Two exemplar decision boundaries. (a) $u(x_1, x_2) = x_1 + x_2 - 1$. (b) $u(x_1, x_2) = -2x_1 + 8x_2/3 - 1$.



AND

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

Figure 3: Geometrical interpretation of three logical operations, namely logical AND (left), logical OR (middle) and XOR (right), and their truth tables.

1.4 Logical Operations Realization

For a single two-input McCulloch-Pitts neuron with specified values for w_1, w_2 and b, one can use the neuron to perform some logical operations. Figure 3 shows the geometries of three logical operations and their truth tables.

1.4.1 AND: $w_1 = w_2 = 1$, $b = 1.5$

For $w_1 = w_2 = 1$, $b = 1.5$, the neuronal model is given by

$$f(x_1, x_2) = h(x_1 + x_2 - 1.5). \tag{4}$$

As $x_1, x_2 \in \{0, 1\}$, $f(x_1, x_2) = 1$ if and only if $x_1 = x_2 = 1$. The neuron as defined by (4) performs logical AND. It acts as an AND gate.

1.4.2 OR: $w_1 = w_2 = 1$, $b = 0.5$

For $w_1 = w_2 = 1$, $b = 0.5$, the neuronal model is given by

$$f(x_1, x_2) = h(x_1 + x_2 - 0.5). \tag{5}$$

As $x_1, x_2 \in \{0, 1\}$, $f(x_1, x_2) = 0$ if and only if $x_1 = x_2 = 0$. The neuron as defined by (5) performs logical OR. It acts as an OR gate.

1.4.3 NAND: $w_1 = w_2 = -1$, $b = -1.5$

For $w_1 = w_2 = -1$, $b = -1.5$, the neuronal model is given by

$$f(x_1, x_2) = h(-x_1 - x_2 + 1.5). \tag{6}$$

As $x_1, x_2 \in \{0, 1\}$, $f(x_1, x_2) = 0$ if and only if $x_1 = x_2 = 1$. The neuron as defined by (6) performs logical NAND. It acts as an NAND gate.

1.4.4 NOR: $w_1 = w_2 = -1$, $b = -0.5$

For $w_1 = w_2 = -1$, $b = -0.5$, the neuronal model is given by

$$f(x_1, x_2) = h(-x_1 - x_2 + 0.5). \tag{7}$$

As $x_1, x_2 \in \{0, 1\}$, $f(x_1, x_2) = 1$ if and only if $x_1 = x_2 = 0$. The neuron as defined by (7) performs logical NOR. It acts as an NOR gate.

1.4.5 XOR Operation

For the above logical operations, their successes rely on proper designs of their decision boundaries given by

$$w_1 x_1 + w_2 x_2 - b = 0. \tag{8}$$

For each of the above logical operations, only one decision boundary is needed. As highlighted in [3], a single two-input McCulloch-Pitts neuron is unable to perform XOR operation. To do so, three two-input McCulloch-Pitts neurons are needed.

Figure 4 shows the network of three neurons which performs the XOR operation. Two neurons are needed in the (so-called) hidden layer. The outputs of the hidden neurons feed their output to the output neuron. The neurons in the hidden layer are defined to perform logical OR and logical NAND. Let $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$ be the outputs of the hidden neurons. By (4), (5) and (6), we get that

$$\begin{aligned} f_1(x_1, x_2) &= h(x_1 + x_2 - 0.5), \\ f_2(x_1, x_2) &= h(-x_1 - x_2 + 1.5), \\ f(x_1, x_2) &= h(x_1 + x_2 - 1.5). \end{aligned}$$

That is to say, with the settings of $w_{11} = w_{12} = 1$, $b_1 = 0.5$, $w_{21} = w_{22} = -1$, $b_2 = -1.5$, $\alpha_1 = \alpha_2 = 1$, $\beta = 1.5$ for the three two-input McCulloch-Pitts neurons as shown in Figure 4, XOR can be implemented.

1.5 Network of McCulloch-Pitts Neurons

To go beyond, one can claim that all multiple-input-multiple-output binary system can be implemented by a network of two-input McCulloch-Pitts neurons. In view of the processing in each neuron, these networks are basically computational models. Given an input $\mathbf{x}$, the network simply computes the outputs in accordance with the computations of the neurons in the network. A network of two-input McCulloch-Pitts neurons is essentially a *computational model*. Precisely, it is a *multiple-binary-input-multiple-binary-output computational model*[1].

[1] Note that this model is a special class of models. For the input (resp. output) value is not limited to binary, the model is simply called multiple-input-multiple-output (MIMO) model

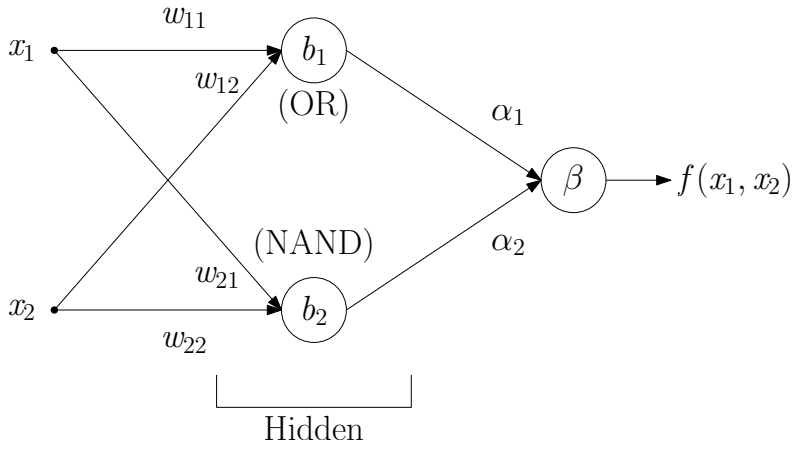


Figure 4: A network of three two-input McCulloch-Pitts neurons performs XOR operation. The neurons in the hidden layer are defined to perform logical OR and logical NAND. The parameters of the model are set to be $w_{11} = w_{12} = 1$, $b_1 = 0.5$, $w_{21} = w_{22} = -1$, $b_2 = -1.5$, $\alpha_1 = \alpha_2 = 1$, $\beta = 1.5$

1.5.1 Decision Network

To play Tic-Tac-Toe, one needs to block the opponent to fill up a line. If a line has already filled up with two opponent symbols, we should fill in the reminding un-filled cell with our symbol. To make this decision, Figure 5 shows a network of McCulloch-Pitts neurons for this decision making – *Should a symbol be put on the cell corresponding to* x_1, x_2 *or* x_3*?*

1.5.2 3-Input-4-Output Neuronal Network

To accomplish this, an AI model with three inputs and four outputs can be designed. In it, there are even types of neurons. Some of them are single-input-single-output neurons (a-type and b-type). Some of them are two-input-single-output neurons (f-type and g-type) and some of them are three-input-single-output neurons (c-type, d-type and e-type). Their mathematical models are given as follows :

$$f_a(x_i) = h(-x_i - 0.5), \tag{9}$$
$$f_b(x_i) = h(x_i - 0.5), \tag{10}$$
$$f_c(y_1, y_2, y_3) = h(y_1 + y_2 + y_3 - 1.5), \tag{11}$$
$$f_d(y_1, y_2, y_3) = h(-y_1 - y_2 - y_3 + 2.5), \tag{12}$$
$$f_e(y_1, y_2, y_3) = h(y_1 + y_2 + y_3 - 1.5), \tag{13}$$
$$f_f(f_c, f_d) = h(f_c + f_d - 1.5), \tag{14}$$
$$f_g(f_f, z_i) = h(f_e - z_i - 0.5), \tag{15}$$

(equi. system).

for $i = 1, 2, 3$. The outputs are defined as follows :

$$o_1 = f_g(f_f, z_1), \;\; o_2 = f_g(f_f, z_2), \;\; o_3 = f_g(f_f, z_3), \;\; o_4 = f_e(y_1, y_2, y_3). \quad (16)$$

If $o_i = 1$, fill in the cell x_i with a symbol. If $o_4 = 1$, the game is over.

While there are one-input-one-output neurons and two-input-one-output neurons in this network, we can replace them by using three-input-one-output neurons. The idea is straight forward. To implement an one-input-one-output neuron, we can set the weights of two inputs to zeros as shown in Figure 6. This idea can be extended to N-input-one-output neuron. The network as shown in Figure 5 can be implemented as a multilayered network as shown in Figure 7. With proper design on the values for the weights and biases, this network is able to (but not limited to) replicate the functionalities of the network as shown in Figure 5.

1.5.3 9-Input-10-Output Neuronal Network

Note that there are eight lines to be diagnosed – three rows, three columns and two diagonals. Decision on the next move in a tic-tac-toe game can be solved by a (bigger) network consisting of nine inputs and ten outputs. This big network is basically a consolidation of eight of the above network of McCulloch-Pitts neurons. Each network makes decision on the next possible move for a line.

1.6 Model Complexity

The complexity of a neuronal model is normally determined by the number of neurons and the number of parameters in the model. For comparison, the number of neurons and the number of parameters in the neuronal networks presented above are depicted in Table 1.

1.6.1 Number of Model Parameters

For the model denoted as $(N_0 - N_1 - \cdots - N_L)$ in Table 1 is a multilayered network with N_0 inputs and N_L outputs. $N_1, \cdots, N_{L-1}$ are the number of neurons in each layers. The total number of neurons is clearly $\sum_{k=1}^{L} N_k$ and the total number of parameters is $\sum_{k=1}^{N} N_k \left(N_{k-1} + 1\right)$. In which, $N_k N_{k-1}$ is the number of connections (i.e. parameters) between the k-layer neurons and the $(k-1)$-layer neurons.

1.6.2 Other Factors

It should be noted that number of parameters is just one issue concerning the model complexity. To figure out how complexity a model, computational time and the memory space required for a model to get an output given an input are two factors. The computational time and memory space required for a model to learn from a data are another two factors. These factors will be delineated in Section 2.5.

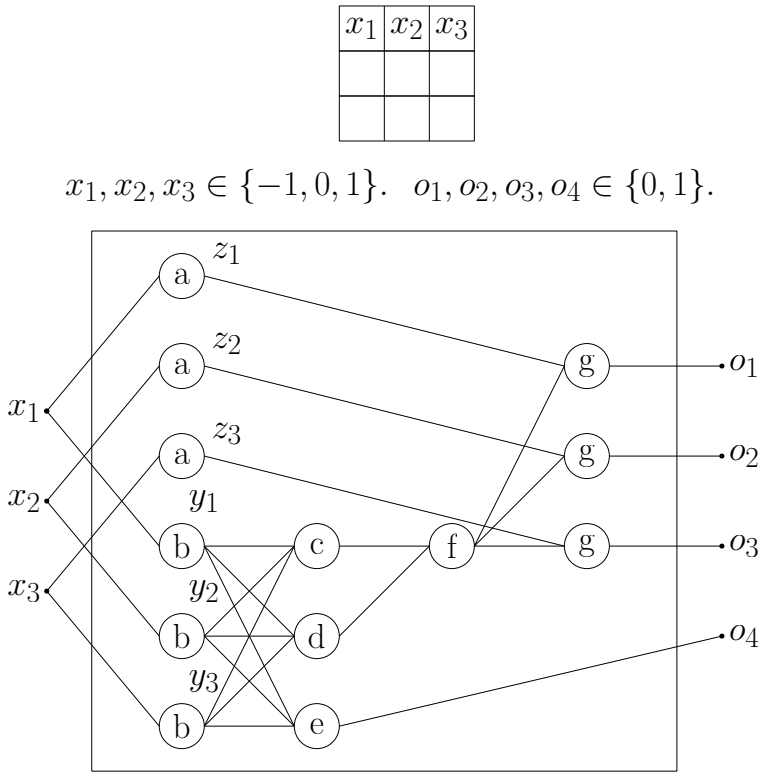
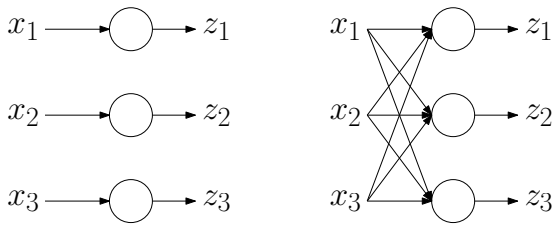


Figure 5: A network of McCulloch-Pitts Neurons could make decision for a step in a Tic-Tac-Toe game – Should a symbol be put on the cell corresponding to x_1, x_2 or x_3? In this network, there are seven different types of neurons. Some of them are single-input-single-output neurons (a-type and b-type). Some of them are two-input-single-output neurons (f-type and g-type) and some of them are three-input-single-output neurons (c-type, d-type and e-type). Here, if $o_4 = 1$, it means *game over*. Note that there are eight lines to be diagnosed – three rows, three columns and two diagonals. Decision on the next move in a tic-tac-toe game can be solved by a network consisting of eight of this network of McCulloch-Pitts Neurons.



$$z_1 = h(-x_1 + 0x_2 + 0x_3 - 0.5)$$
$$z_2 = h(0x_1 - x_2 + 0x_3 - 0.5)$$
$$z_3 = h(0x_1 + 0x_2 - x_3 - 0.5)$$

Figure 6: Implementation of three one-input-one-output neurons by three-input-three-output network. For the redundant weights, we simply set them to be zeros.

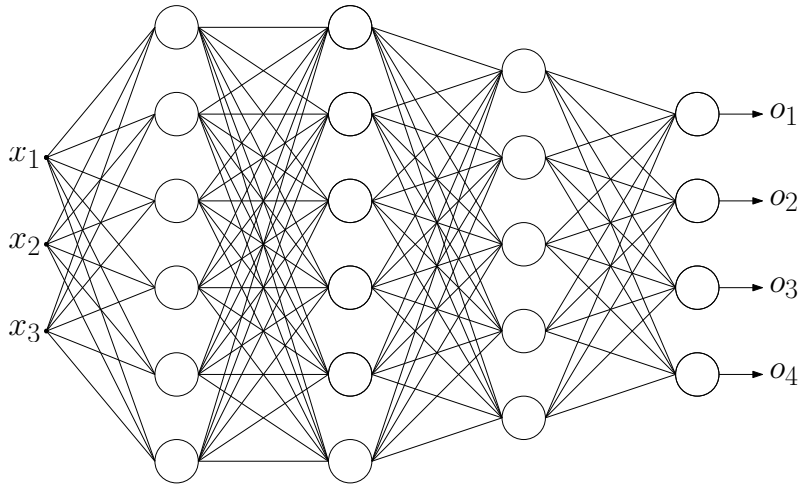


Figure 7: A three-input-four-output multilayered network of all N-input-one-output neurons. Note that this network is also called a computational model. With proper design on the values for the weights and biases, this network is able to (but not limited to) replicate the functionalities of the network as shown in Figure 5.

Table 1: Complexity of some McCulloch-Pitts neuronal networks. The last model denoted as $(N_0 - \cdots - N_L)$ is a N_0-input-N_L-output network.

Model	**No. of Neurons**	**No. of Parameters**
AND	1	3
OR	1	3
NAND	1	3
NOR	1	3
XOR	3	9
Figure 5	13	36
Figure 7	21	125
$(N_0 - \cdots - N_L)$	$\sum_{k=1}^{L} N_k$	$\sum_{k=1}^{L} N_k \left(N_{k-1} + 1\right)$

1.7 Digital Computer and M-P Networks

Note that a computer is essentially constructed by a network of AND, OR, NAND, NOR and XOR logic gates to perform both logical and arithmetics operations. As a network of two-input McCulloch-Pitts neurons can perform the operations as the logic gates, a digital computer can thus be implemented by these two-input McCulloch-Pitts neurons. In this regard, a connection between computer and brain was established. *A human brain can do more than a digital computer*.

1.7.1 Number of Processing Nodes

For a digital computer, a processing node refers to a logic gate which is a two-input-one-output system. For a McCulloch-Pitts neuron, it is an N-input-one-output processing node. In terms of the number of processing nodes, a McCulloch-Pitts neuronal network could be structural simpler than a digital computer.

An obvious example is on the number of inputs. A logic gate can only accept two inputs, while a McCulloch-Pitts neuron can accept more than two inputs. For the logical operation with three inputs and its output '1' if and only if all three inputs are '1', two AND logic gates are needed for this operation. Using McCulloch-Pitts neuron, we need only one. The network complexity could be reduced.

1.7.2 Beyond Digital Computations

Moreover, McCulloch-Pitts neuron accepts scalar inputs instead of binary. This neuron can be designed to solve problems with scalar inputs. Therefore, a network of McCulloch-Pitts neurons can be designed to solve 2-class classification problems – object recognition problems in which only two classes of objects are to be recognized. Along this line of thought, multiple networks

Table 2: Interpretations of the variables and parameters in a M-P neuron.

Input $x_i = 1$	Electric pulse stream of a fixed firing rate r.
Input $x_i = 0$	No pulse stream received.
Output $f(\cdot) = 1$	Electric pulse stream of a fixed firing rate r.
Output $f(\cdot) = 0$	No pulse stream generated.
$w_i > 0$	Excitatory synapse.
$w_i = 0$	No connection.
$w_i < 0$	Inhibitory synapse.

of McCulloch-Pitts neurons can thus be applied to general object recognition problems with multiple classes of objects to be recognized. Furthermore, the model of McCulloch-Pitts neuron was applied in signal processing [4].

1.8 M-P Network as a Computational Model

It is no doubt that a network of McCulloch-Pitts neurons is essentially a computational model. As long as all the neuronal models have been defined, the operations of the network are defined accordingly. Each neuron simply performs a computation and gives results. The computational models developed along this line are called *Perceptrons* which are developed and advocated by Frank Rosenblatt in the 1950s to 1960s [5, 6, 7].

In the example delineated in Figure 5, all parameters in the network are pre-defined by me. One question is then aroused. *What if the parameters are not given, is it possible to develop a learning algorithm for this model to get these parameters?* The answer is clearly YES. The learning rule associated with *Perceptrons* were later named as *Perceptron learning rule* in [3].

1.9 Interpretation of an M-P Neuron

One question regarding the M-P neuron is on the interpretations of the input and the output. If $x_1 = 1$, the McCulloch-Pitts receives a electric pulse stream of a fixed firing rate, say r. If the output of a McCulloch-Pitts neuron is one, the neuron generates a stream of electric pulses with firing rate r to the subsequent neurons. Table 2 summaries the physical meanings of the parameters in an M-P neuron.

1.10 Learning Classification

Figure 8 shows the use of a single 2-input-1-output McCulloch-Pitts neuron for data classification. The data in Group I is indicated by a circle and the data in Group II is indicated by a square. The main problem is to find the parameters w_1, w_2 and b for the decision boundary $u(x_1, x_2)$.

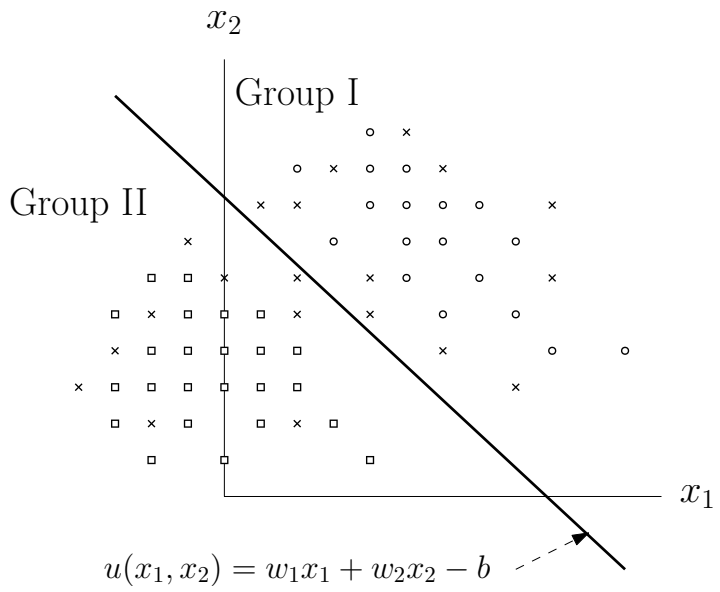


Figure 8: Use of a single 2-input-1-output McCulloch-Pitts neuron for data classification. The data in Group I is indicated by a circle and the data in Group II is indicated by a square. The data indicated by a cross is an unlabelled data. One purpose of getting the decision boundary is to use it to label those unlabelled data.

Table 3: Values of $|d_k - f(\mathbf{x}_k)|$.

d_k	$f(\mathbf{x}_k)$	$\lvert d_k - f(\mathbf{x}_k)\rvert$
0	0	0
0	1	1
1	0	1
1	1	0

1.10.1 Step 1: Indexing, labeling and assessment

To solve this classification problem, the first step is to assign indices and labels for the data.

Indexing and labeling. Suppose the total number of data is N. We assign each data a unique index. For the k^{th} data, $\mathbf{x}_k = (x_{k1}, x_{k2})$ and d_k be respectively the coordinate and label of the k^{th} data. Its label d_k is defined as follows[2] :

$$d_k = \begin{cases} 1 & \text{if } \mathbf{x}_k \text{ is in } \textit{Group I}, \\ 0 & \text{if } \mathbf{x}_k \text{ is in } \textit{Group II}. \end{cases} \tag{17}$$

Assessment (Performance Criteria). To assess how good a neuron with parameters w_1, w_2 and b can perform, we need to define a reasonable assessment measure. One can define the measure as the total prediction errors[3].

$$E(w_1, w_2, b) = \sum_{k=1}^{N} |d_k - f(\mathbf{x}_k)|, \tag{18}$$

where $f(\mathbf{x}_k)$ is the prediction of the neuron on the group to which the data $\mathbf{x}_k$ belongs. If the prediction is identical to the actual label, $|d_k - f(\mathbf{x}_k)| = 0$. Otherwise, $|d_k - f(\mathbf{x}_k)| = 1$. The values of $|d_k - f(\mathbf{x}_k)| = 1$ are depicted in Table 3 for clarification. In other words, $d_k - f(\mathbf{x}_k)| = 0$ *if only if the prediction is correct*. So, $E(w_1, w_2, b)$ is the total prediction errors of a neuron with model parameters w_1, w_2 and b. If $E(w_1, w_2, b) = 0$, the neuron with parameters w_1, w_2 and b is an optimal model.

Next, in search of (w_1, w_2, b)**.** With the above labelling, the second step is to develop a method to find the values w_1, w_2 and b their corresponding *total prediction errors* $E(w_1, w_2, b)$ is a minimum. Here, two methods are introduced, namely brute-force search and Perceptron learning.

[2]Note that this labelling is arbitrary. One can define $d_k = 0$ if $\mathbf{x}_k$ is in *Group I* and $d_k = 1$ if $\mathbf{x}_k$ is in *Group II*.

[3]One should be noted that the total prediction errors $E(w_1, w_2, b)$ is a non-differentiable function. Obtaining a learning rule which minimizes this function is not easy.

1.10.2 Step 2: Brute-force search

Its key idea is to search all possible combinations of (w_1, w_2, b). For instance,

$$\begin{aligned} w_1 &= -5, -4.99, -4.98, \cdots, 4.98, 4.99, 5. \\ w_2 &= -5, -4.99, -4.98, \cdots, 4.98, 4.99, 5. \\ b &= -5, -4.99, -4.98, \cdots, 4.98, 4.99, 5. \end{aligned}$$

In such case, the total number of combinations of (w_1, w_2, w_3) is 1001^3. It is more than 10^9 combinations. For each (w_1, w_2, b), we feed in the data one by one to the inputs of the neuron and then calculate the neuronal output. Finally, the performance of this neuron $E(w_1, w_2, b)$ is calculated. Repeating the process for all 1001^3 combinations, we will have 1001^3 values of $E(w_1, w_2, b)$. In the end, those models with zero prediction error are the optimal models.

It is clear that *brute-force search* is not an efficient method to obtain an optimal model. For the number of parameters is larger, this method is infeasible. However, for some learning problems, this method is still a key for the search of model parameters.

1.10.3 Step 2: Perceptron learning

Long in the history, developing an efficient learning rule for a network of M-P neurons has been a challenging problem. Perceptron learning is one learning developed by Frank Rosenblatt in the 1950s [5, 6, 7]. For Perceptron learning, there are two modes of learning : batch mode and online mode.

Batch mode. For the batch mode, the M-P neuron predicts the labels for all N data. That is to say, the M-P neuron calculates $f(\mathbf{x}_k)$ for $k = 1, \cdots, N$. Then, these predictions are then compared with the actual labels to get $(d_k - f(\mathbf{x}_k))$ for $k = 1, \cdots, N$. Subseqently, the parameters w_1, w_2 and b are updated based on the following equations.

$$w_1(t+1) = w_1(t) + \mu \sum_{k=1}^{N} (d_k - f(\mathbf{x}_k)) x_{k1}, \tag{19}$$

$$w_2(t+1) = w_2(t) + \mu \sum_{k=1}^{N} (d_k - f(\mathbf{x}_k)) x_{k2}, \tag{20}$$

$$b(t+1) = b(t) - \mu \sum_{k=1}^{N} (d_k - f(\mathbf{x}_k)), \tag{21}$$

where $w_1(0)$, $w_2(0)$ and $b(0)$ are arbitrary numbers. In (19), (20) and (21), the factor μ is called the learning step size which value is usually set to be a small number, say $\mu = 0.001$.

Online mode. In contrast to the batch mode learning, the update of w_1, w_2 and b is conducted one data at a time. Once a data $(\mathbf{x}_t, d_t)$ is *randomly selected* from the dataset, the M-P neuron calculates the prediction $f(\mathbf{x}_t)$ and then $(d_t - f(\mathbf{x}_t))$. Subsequently, the parameters w_1, w_2 and b are updated based on the following equations.

$$\begin{aligned} w_1(t+1) &= w_1(t) + \mu_t(d_t - f(\mathbf{x}_t))x_{t1}, && (22)\\ w_2(t+1) &= w_2(t) + \mu_t(d_t - f(\mathbf{x}_t))x_{t2}, && (23)\\ b(t+1) &= b(t) - \mu_t(d_t - f(\mathbf{x}_t)), && (24) \end{aligned}$$

where μ_t is a small number corresponding for the learning step size at time t, say $\mu_t = 0.01/t$. Besides, $w_1(0)$, $w_2(0)$ and $b(0)$ are arbitrary numbers. It can be shown that with proper setting[4] on μ_t, the online learning rule as stated in (22), (23) and (24) is able to get (precisely, *converge to*) an optimal model for the classification problem.

1.10.4 Reinforcement Interpretation

Let $\mathbf{w}(t) = (w_1(t), w_2(t), b(t))^T$ and $\mathbf{x}_t = (x_{t1}, x_{t2}, -1)^T$. The online learning rule can be rewritten in a compact form.

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \mu_t(d_t - f(\mathbf{x}_t, \mathbf{w}(t)))\mathbf{x}_t. \quad (25)$$

As $d_t, f(\mathbf{x}_t, \mathbf{w}(t)) \in \{0, 1\}$, $\mathbf{w}(t)$ only updates when d_t and $f(\mathbf{x}_t, \mathbf{w}(t))$ are different. Therefore, we can get that

$$\mathbf{w}(t+1) = \begin{cases} \mathbf{w}(t) & \text{if } d_t = f(\mathbf{x}_t, \mathbf{w}(t)), \\ \mathbf{w}(t) + \mu_t\mathbf{x}_t & \text{if } d_t = 1 \text{ and } f(\mathbf{x}_t, \mathbf{w}(t)) = 0, \\ \mathbf{w}(t) - \mu_t\mathbf{x}_t & \text{if } d_t = 0 \text{ and } f(\mathbf{x}_t, \mathbf{w}(t)) = 1. \end{cases} \quad (26)$$

The model is *reinforced* to change if its answer is not correct. This is a reinforcement learning interpretation for the online Perceptron learning.

1.11 Illustrative Examples

Either for the batch mode learning as stated in (19), (20) and (21) or the online mode learning as stated in (22), (23) and (24), one should see that the update of the model parameters is relied on those data whose predictions are incorrect.

1.11.1 Separable data

To illustrate the behavior of the Perceptron learning rule, a set of two groups of data are randomly generated and shown in Figure 9. In this dataset, 100 data are belongs to *Group I* and 100 data are belongs to *Group II*. It is clear from Figure 9 that these two groups of data are separable.

[4]The conditions are that $\sum_{t=1}^{\infty} \mu_t = \infty$ and $\sum_{t=1}^{\infty} \mu_t^2 < \infty$.

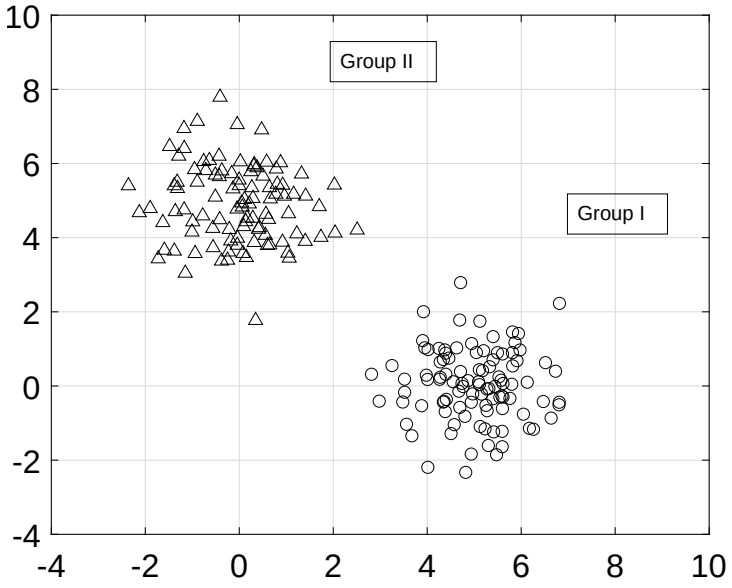


Figure 9: Two groups of data which are separable.

1.11.2 Settings

To determine the model parameters w_1, w_2 and b, the online mode Perceptron learning rule as stated in (22), (23) and (24) is applied with $\mu_t = 0.005$ for all t and the maximum of iteration is set to be 2000. Two initial conditions are simulated : (a) $w_1(0) = w_2(0) = b = 1$ and (b) $w_1(0) = w_2(0) = b = 0$.

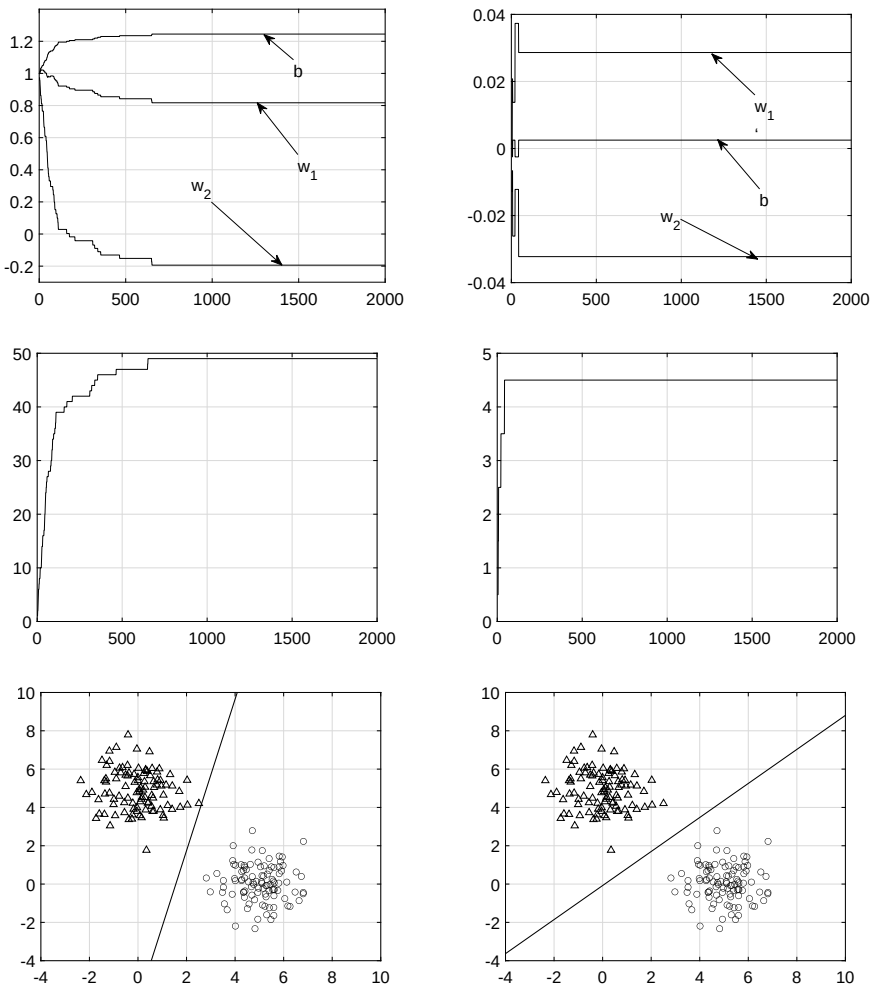
1.11.3 Results

The top panel of the Figure 10 shows the changes of the parameters w_1, w_2 and b obtained by the online Perceptron learning rule over time $t = 1, \cdots, 2000$. The middle panel of the Figure 10 shows the changes of the prediction errors $\sum_{k=1}^{t} |d_k - f(\mathbf{x}_k)|$ over time from $k = 1$ to $k = t$. The decision boundaries obtained are shown in the bottom panel of the Figure 10.

It should be noted that the results shown in Figure 10 could be slightly difference if the same experiment is repeated. It is because of the online learning. In each step, the data to be selected is random. Therefore, sequence of data being selected for update in an experiment is clearly different from the sequence of data being selected in another experiment. The results shown in Figure 10(a) or Figure 10(b) are corresponding to one experiment, not for all.

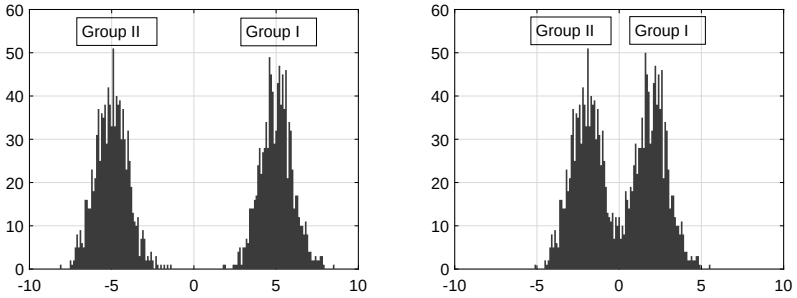
1.11.4 Comments

Applying Perceptron learning rule for a single M-P neuron, one needs to set the values for the initial conditions of w_1, w_2 and b. Besides, the learning rate μ and



(a) $w_1(0) = w_2(0) = b(0) = 1$. (b) $w_1(0) = w_2(0) = b(0) = 0$.

Figure 10: Changes of the parameters w_1, w_2 and b over time for the dataset as shown in Figure 9. (a) With the initial condition $w_1(0) = w_2(0) = b(0) = 1$, the parameters converge to $w_1 = 0.8019$, $w_2 = -0.2029$ and $b = 1.2500$ after $t \geq 750$. (b) With the initial condition $w_1(0) = w_2(0) = b(0) = 0$, the parameters converge to $w_1 = 0.0286$, $w_2 = -0.0322$ and $b = 0.0025$ after $t \geq 100$. **Top:** Changes of parameters. **Middle:** Prediction errors. **Bottom:** Decision boundary.



(a) Separable. (b) Non-Separable.

Figure 11: Separable and non-separable data. For separable dataset, it is able to find an M-P neuron its total prediction errors is zero. For non-separable dataset, the minimum total prediction errors must be non-zero.

the maximum number of iterations have to be set. Different initial conditions of w_1, w_2 and b might give different values of the convergent w_1, w_2 and b, i.e. different models. For the learning rate μ and the maximum number of iteration, the smaller the value of μ will lead to larger number of iterations. The settings of all these factors are basically determined by trial-and-error, i.e. by the experience of the developer.

1.12 Pitfall of a Network of M-P Neurons

A pitfall of the network of McCulloch-Pitts neurons is clearly on the development of a learning rule for multilayered M-P neuronal networks. For the case of single M-P neuron, the learning rule as stated in (22), (23) and (24) is able to let the neuron to attain an optimal for two-class linear separable classification problems. For a classification problem which is not linear separable, learning rule is difficult to be developed as the neuronal output is a step function.

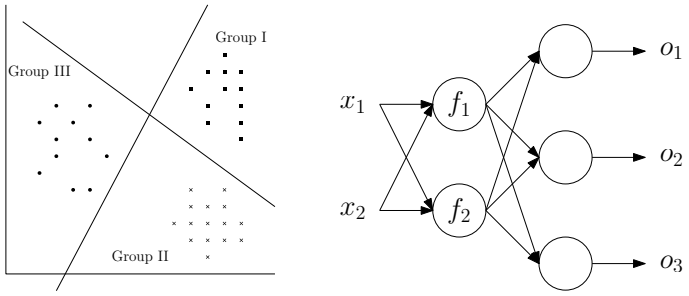
Figure 11 shows two examples. For either example, a good 1-input-1-output M-P neuron can be defined as follows :

$$f(x) = h(x), \text{ i.e. } w = 1,\ b = 0.$$

For the dataset as shown in Figure 11a, this model gives perfect predictions to all data, i.e. $E(1, 0) = 0$. However, for the dataset as shown in Figure 11b, $E(1, 0) > 0$.

1.13 Network of M-P Neurons for 3-Class Data

Applying the network of M-P neurons, it could be difficult to get a learning rule for a 3-class data classification problem. Figure 12 shows the distributions of the three classes of data and the Perceptron model which is able to solve this classification problem. The M-P neurons in the first layer perform the two



(a) Three classes. (b) Perceptron model.

Figure 12: Three-Class classification problem. (a) Geometrical illustration of the distributions of the three classes of data. (b) The Perceptron model which can solve this classification problem.

decisions as indicated in Figure 12(a). Once the decision boundaries have been obtained, the neurons at the output layer simply perform the logical operations depicted below.

$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	o_1	o_2	o_3	Group
0	0	0	0	0	–
0	1	1	0	0	I
1	0	0	0	1	III
1	1	0	1	0	II

It is clear that the Perceptron model as shown in Figure 12(b) can be designed to solve the 3-class classification problem. However, the learning rule for the update of the model parameters is not easily defined. Nevertheless, learning rule for the update of the model parameters in a multilayered M-P neuronal network is even difficult.

2 Sigmoidal Neuronal Networks

For a multilayered network of McCulloch-Pitts neurons, as shown in Figure 7, developing a learning rule for this network is difficult as the neuronal function is non-differentiable. Techniques from functional approximation and parametric estimation are not applicable, as those techniques require the (transfer function) model is differentiable.

In this regard, Paul Werbos in 1974 [8] suggested replacing the McCulloch-Pitts neuron by a differentiable function its shape is similar to an M-P neuron. Later, Rumelhart, Hinton and Williams [9] independently in 1986 suggested the same replacement. While Paul Werbos did not specify which differentiable function for a neuron model, Rumelhart, Hinton and Williams specifically introduced the sigmoid function as the neuron model. By that, the output of a

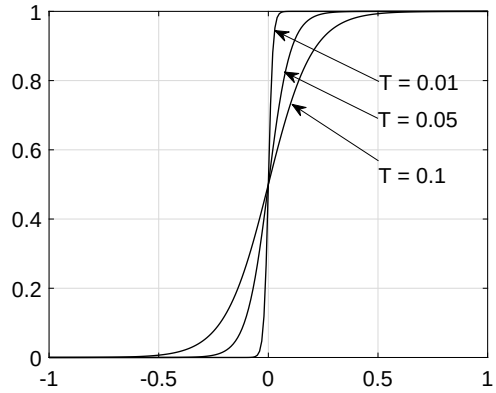


Figure 13: Sigmoid function could be considered as a relaxation of the step function in the McCulloch-Pitts model. The plots show the output versus the value of u. If $T \to 0$, the output is identical to a step function as in the McCulloch-Pitts model.

neuron is given by

$$f(x_1, x_2) = \frac{1}{1 + \exp\left(-u(x_1, x_2)/T\right)} \tag{27}$$

$$= \frac{1}{1 + \exp\left(-(w_1 x_1 + w_2 x_2 - b)/T\right)} \tag{28}$$

where $u(x_1, x_2) = w_1 x_1 + w_2 x_2 - b$ as usual and the factor T is called the temperature. Figure 13 shows the plots of the output of a neuron against the input u for $T = 0.01$, $T = 0.05$ and $T = 0.1$.

2.1 Sigmoid Neuron

For a neuron with n inputs, i.e. $\mathbf{x} = (x_1, \cdots, x_n)^T$, the output of a neuron is modeled as follows :

$$f(\mathbf{x}) = \frac{1}{1 + \exp\left(-(\sum_{i=1}^{n} w_i x_i - b)\right)}. \tag{29}$$

It should be noted that the temperature factor T is absorbed (redundant) in the parameters, i.e. $w_i \leftarrow w_i/T$ and $b \leftarrow b/T$.

2.2 Interpretation of a Sigmoid Neuron

Similar to that of a McCulloch-Pitts neuron, the physical meaning of the inputs, the outputs and the weights can be interpreted. In contrast to the M-P neuron, the value of an input to a sigmoid neuron is the *firing rate* of the impulse stream received from the input neuron. The sign of a weight w_i indicates if the connection is excitatory or inhibitory. The output of a neuron is the *firing rate*

of the impulse steam to be generated. This interpretation is usually called the *rate coding* system.

2.3 Multilayered Perceptron (MLP)

Therefore, the multilayered neuronal network with this sigmoid neuron is then called a multilayered Percertron (MLP) or back-propagation network (BPN). From a mathematical function point of view, MLP is just a multiple-input-multiple-output function which can be denoted as $\mathbf{f}(\mathbf{x}, \mathbf{w})$, where $\mathbf{x}$ is the input and $\mathbf{w}$ is the vector of the function parameters.

2.4 Backpropagation (BP) Learning

For a sigmoid multilayered Percetron (MLP), the learning algorithm as proposed by Rumelhart *et al* is called backpropagation (BP). The learning algorithm is basically a gradient descent algorithm in search of the model parameters $\mathbf{w}$) in which its prediction errors $E(\mathbf{w})$ is a minimum. Here, the parameters of an MLP is denoted as a vector $\mathbf{w}$.

Without loss of generality, we assume that there is only one output neuron in the MLP in the following presentation. The gradient descent learning for an MLP is given as follows :

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \mu \frac{\partial E(\mathbf{w}(t))}{\partial \mathbf{w}}, \tag{30}$$

where $E(\mathbf{w}(t))$ is the total prediction errors as follows :

$$E(\mathbf{w}) = \sum_{k=1}^{N} (d_k - f(\mathbf{x}_k, \mathbf{w}))^2. \tag{31}$$

Here, one should be noted that the total prediction errors as stated in (31) is different from that defined in (18). The prediction errors as stated in (31) is the sum-square-errors (SSE).

2.4.1 Batch Mode

Given a set of N data, the update of the parametric vector $\mathbf{w}$ can be conducted by the following algorithm. At step t, the outputs and their gradient vectors of the MLP for the N data are calculated. The update of $\mathbf{w}(t)$ to $\mathbf{w}(t+1)$ is obtained by the following update equation.

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \mu \sum_{k=1}^{N} (d_k - f(\mathbf{x}_k, \mathbf{w}(t))) \frac{\partial f(\mathbf{x}_k, \mathbf{w}(t))}{\partial \mathbf{w}}, \tag{32}$$

where μ is a small constant namely the learning step size. Batch mode learning is suitable for a small size MLP with small size of dataset[5].

[5]Batch mode learning is applicable to the case that (i) the number of model parameters is not large and (ii) the number of data in the dataset, i.e. N, is not large. If the number

2.4.2 Online Mode

For the online mode learning, a data $(\mathbf{x}_t, d_t)$ is randomly selected from the dataset. The update of $\mathbf{w}(t)$ to $\mathbf{w}(t+1)$ is obtained by the following update equation.

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \mu_t(d_t - f(\mathbf{x}_t, \mathbf{w}(t)))\frac{\partial f(\mathbf{x}_t, \mathbf{w}(t))}{\partial \mathbf{w}}, \tag{33}$$

where μ_t is a small constant namely the learning step size. As the online Perceptron learning, this online learning converges if μ_t satisfies the conditions that

$$\text{(a) } \sum_{t=1}^{\infty} \mu_t = \infty \text{ and (b) } \sum_{t=1}^{\infty} \mu_t^2 < \infty. \tag{34}$$

Two points should be remarked. First, as the memory and computational requirement for a step of online learning is small, it is suitable for handling a larger scale MLP with large scale dataset. Second, the online learning as stated in (33) is also called the *stochastic gradient descent* algorithm.

2.4.3 Online Learning for a 2-Input-1-Output Neuron

For a two-input-one-output neuron, $\mathbf{w} = (w_1, w_2, b)$, the learning algorithm as stated in (30) can be stated as follows :

$$\begin{aligned} w_1(t+1) &= w_1(t) + \mu_t e(t) f'(\mathbf{x}_t, \mathbf{w}(t)) x_{t1} & (35)\\ w_2(t+1) &= w_2(t) + \mu_t e(t) f'(\mathbf{x}_t, \mathbf{w}(t)) x_{t2} & (36)\\ b(t+1) &= b(t) - \mu_t e(t) f'(\mathbf{x}_t, \mathbf{w}(t)), & (37) \end{aligned}$$

where

$$\begin{aligned} e(t) &= d_t - f(\mathbf{x}_t, \mathbf{w}(t)\\ f'(\mathbf{x}_t, \mathbf{w}(t)) &= f(\mathbf{x}_t, \mathbf{w}(t))(1 - f(\mathbf{x}_t, \mathbf{w}(t))). \end{aligned}$$

In vector form,

$$\underbrace{\begin{bmatrix} w_1(t+1) \\ w_2(t+1) \\ b(t+1) \end{bmatrix}}_{\mathbf{w}(t+1)} = \underbrace{\begin{bmatrix} w_1(t) \\ w_2(t) \\ b(t) \end{bmatrix}}_{\mathbf{w}(t)} + \mu_t e(t) f'(\mathbf{x}_t, \mathbf{w}(t)) \begin{bmatrix} x_{t1} \\ x_{t2} \\ -1 \end{bmatrix}. \tag{38}$$

Again, the factor μ_t is the learning step at the time t. If μ_t satisfies the conditions as stated in (34), it can be shown that BP learning can obtain a model (w_1, w_2, b) such that its $E(\mathbf{w})$ is a minimum.

---

of model parameters is large, say millions, or the number of data in the dataset is large, say millions, batch model learning is normally not applicable as the RAM or main memory might not be large enough to support such batch mode learning.

2.4.4 Illustrative Examples

Here, we apply the above 2-input-1-output neuron for the two-class classification problem as presented in Section 1.11. As sigmoid function is continuous function, the decision boundary is defined as follows :

$$w_1 x_1 + w_2 x_2 - b = 0. \tag{39}$$

By (39), we can get that

$$f(\mathbf{x}, \mathbf{w}) \begin{cases} > 1/2 & \text{if } w_1 x_1 + w_2 x_2 - b > 0, \\ = 1/2 & \text{if } w_1 x_1 + w_2 x_2 - b = 0, \\ < 1/2 & \text{if } w_1 x_1 + w_2 x_2 - b < 0. \end{cases} \tag{40}$$

Thus, we can label the data by the output value of the sigmoidal neuron.

Again, we investigate the results for the cases that $w_1(0) = w_2(0) = b(0)) = 1$ and $w_1(0) = w_2(0) = b(0)) = 0$. The changes of $\mathbf{w}(t)$, cumulative errors and the decision boundary are shown in Figure 14.

2.5 Model Complexity

In essence, the model complexity of a multiple-input-multiple-output MLP is the same as a multiple-input-multiple-output McCulloch-Pitts Perceptron. For clarification, Figure 15 shows a MLP with N_0 input nodes and N_L output nodes.

2.5.1 Numbers of Neurons and Parameters

For the model as shown in Figure 15, the number of neurons is given by

$$\text{No. of Neurons} = \sum_{k=0}^{L} N_k. \tag{41}$$

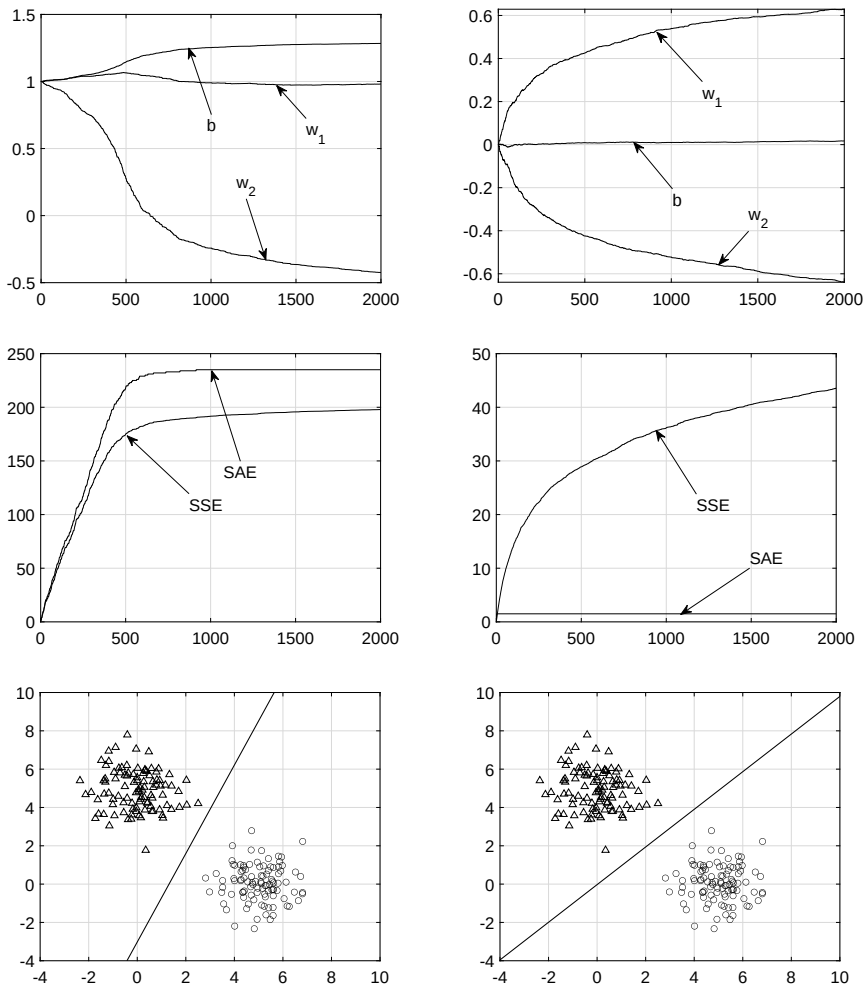
The number of parameters is given by

$$\text{No. of Parameters} = \sum_{k=1}^{L} N_k (N_{k-1} + 1). \tag{42}$$

2.5.2 Forward Pass Computation Complexity

As for the k^{th} layer, where $k = 1, \cdots, L$, there are N_{k-1} multiplications and N_{k-1} additions for computing the u_{kl} in the l^{th} neuron in the k^{th} layer. Then, the value u_{kl} is passed to the sigmoid function $\phi(u_{kl})$ for get the output of this neuron. For scalar computation, the total time spent in a forward pass is given by

$$T_C = \sum_{k=1}^{L} N_k N_{k-1} \left(T_M + T_A\right) + \sum_{k=1}^{L} N_k T_0, \tag{43}$$



(a) $w_1(0) = w_2(0) = b(0) = 1$. (b) $w_1(0) = w_2(0) = b(0) = 0$.

Figure 14: Changes of the parameters w_1, w_2 and b over time for the dataset as shown in Figure 9. (a) With the initial condition $w_1(0) = w_2(0) = b(0) = 1$. (b) With the initial condition $w_1(0) = w_2(0) = b(0) = 0$. Here, $\mu = 0.01$. **Top:** Changes of parameters. **Middle:** Prediction errors. **Bottom:** Decision boundary. One should be noted that the sum of absolute errors (SAE) converges but the parameters $w_1(t)$, $w_2(t)$ and $b(t)$ do not converge in both cases.

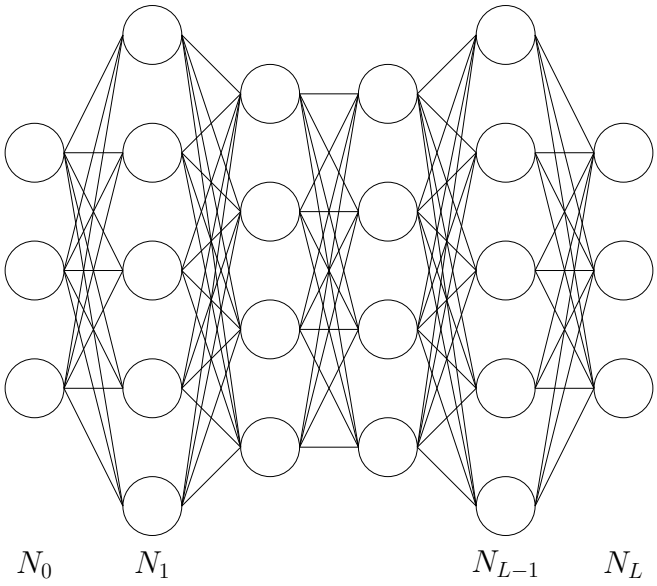


Figure 15: A MLP with N_0 input and N_L output. This model consists of $\sum_{k=1}^{L} N_k$ neurons and $\sum_{k=1}^{L} N_k(N_{k-1}+1)$ number of parameters. For a forward pass to get an output, the computational time T_C is proportional to the number of parameters.

where T_M is the time for a floating point multiplication, T_A is the time for a floating point addition (resp. subtraction) and T_0 is the time for get the value of $\phi(u_{kl})$. In practice, $T_M \approx T_A$ and T_0 can be ignored. We can get that

$$\begin{aligned} T_C &\approx 2T_M \times (\text{No. of Parameters}) \\ &= 2 \times \text{FLOPS}^{-1} \times (\text{No. of Parameters}), \end{aligned} \tag{44}$$

where FLOPS is the number of floating point computations per second. The computational time T_C for a forward pass is proportional to the number of parameters.

2.5.3 Forward Pass Memory Complexity

In a forward pass, it is clear that memory space is needed for storing the parameters, i.e. $\sum_{k=1}^{L} N_k(N_{k-1}+1) \times 4$ bytes. Given an input $\mathbf{x}_t$, the memory space for storing the neuronal outputs for the k^{th} layer is clear $N_k \times 4$ bytes. Therefore, the minimum number of memory space for a forward pass is given by

$$M_{min} = 4 \times \left(1 + \sum_{k=1}^{L} N_k(N_{k-1}+1) + \max_{k=0,\cdots,L} \{N_k\} \right), \tag{45}$$

where the unit is in byte.

2.5.4 Backward Pass Memory Complexity

While learning, the memory space needed is a lot more as all neuronal outputs have to be stored. In this regard, the size of the memory space is given by

$$M_{max} = 4 \times \left(N_0 + 1 + \sum_{k=1}^{L} N_k(N_{k-1}+2) \right), \tag{46}$$

where N_0 is the number of inputs.

2.6 Comments on Sigmoid MLP

Here, let me present a few comments regarding this sigmoidal neuron.

2.6.1 Logistic Regression

As a matter of fact, the sigmoidal neuronal function is in essence the logistic function which is widely used in statistical analysis. Applying multiple-input-one-output sigmoid neuron for solving a two-class classification problem is equivalent to solving a logistic regression problem.

2.6.2 Divergence of $\mathbf{w}(t)$

As shown in Figure 14(**Middle**), the parameters do not converge as t increases. In contrast to a multiple-input-one-output McCulloch-Pitts neuron, its parameters converge as t increases. A reason for the divergence of the parameters is due to the fact that the number of feasible multiple-input-one-output neurons for solving the two-class classification is infinite. If a decision boundary given by

$$w_1 x_1 + w_2 x_2 - b = 0$$

is able to solve the problem, all the decision boundary given by

$$\kappa(w_1 x_1 + w_2 x_2 - b) = 0 \ \ (\kappa > 0)$$

must be able to solve the same problem.

That is to say, for a good decision boundary $\mathbf{w}'$, $\kappa\mathbf{w}'$ will give better results if $\kappa > 1$. To this end, the optimal parametric vector $\mathbf{w}^*$ must be $\kappa\mathbf{w}'$ with $\kappa \to \infty$. This is one reason why $\mathbf{w}(t)$ must diverge.

2.6.3 Weight Decay (Forgetting Factor)

To solve the divergence problem, one simple approach is to design the learning rule with so-called *weight decay* as follows :

$$\begin{aligned} w_1(t+1) &= w_1(t) + \mu_t \left(e(t) f'(\mathbf{x}_t, \mathbf{w}(t)) x_{t1} - \alpha\, w_1(t)\right) && (47)\\ w_2(t+1) &= w_2(t) + \mu_t \left(e(t) f'(\mathbf{x}_t, \mathbf{w}(t)) x_{t2} - \alpha\, w_2(t)\right) && (48)\\ b(t+1) &= b(t) - \mu_t \left(e(t) f'(\mathbf{x}_t, \mathbf{w}(t)) - \alpha\, b(t)\right), && (49) \end{aligned}$$

where

$$\begin{aligned} e(t) &= d_t - f(\mathbf{x}_t, \mathbf{w}(t)\\ f'(\mathbf{x}_t, \mathbf{w}(t)) &= f(\mathbf{x}_t, \mathbf{w}(t))(1 - f(\mathbf{x}_t, \mathbf{w}(t))). \end{aligned}$$

Normally, the decay factor α is set to be a small positive number.

We investigate the results for the case that $w_1(0) = w_2(0) = b(0)) = 1$. The changes of $\mathbf{w}(t)$, prediction errors and the decision boundary are shown in Figure 16. In the figure, MAE and MSE stand for the mean absolute errors and mean square errors as defined as follows :

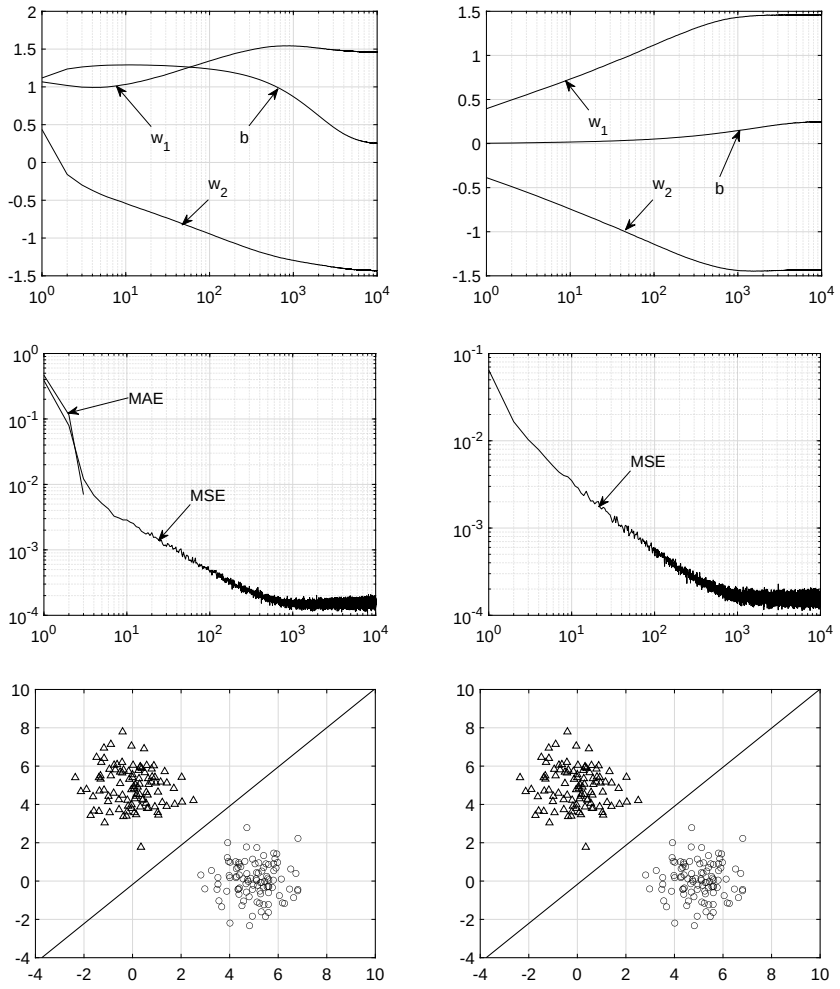
$$MAE(k) = \frac{1}{T} \sum_{t=(k-1)T+1}^{kT} |d_t - y(\mathbf{x}_t, \mathbf{w}(t))|, \quad (50)$$

$$MSE(k) = \frac{1}{T} \sum_{t=(k-1)T+1}^{kT} (d_t - f(\mathbf{x}_t, \mathbf{w}(t)))^2, \quad (51)$$

where

$$y(\mathbf{x}_t, \mathbf{w}(t)) = \begin{cases} 1 & \text{if } f(\mathbf{x}_t, \mathbf{w}(t)) > 0.5 \\ 0 & \text{if } f(\mathbf{x}_t, \mathbf{w}(t)) \le 0.5 \end{cases} \quad (52)$$

and T is the size of the time-window.



(a) $w_1(0) = w_2(0) = b(0) = 1$. (b) $w_1(0) = w_2(0) = b(0) = 0$.

Figure 16: Changes of the parameters w_1, w_2 and b over time for the dataset as shown in Figure 9. Here, $\mu_t = 0.001$, $\alpha = 0.0001$ and the total number of learning steps is $10000 \times 20 \times 200 = 4 \times 10^7$. The initial conditions are set to be (a) $w_1(0) = w_2(0) = b(0) = 1$ and (b) $w_1(0) = w_2(0) = b(0) = 0$. The results shown in the middle panel are mean absolute errors (MAE) and mean square errors (MSE) over every 4000 steps. **Top:** Changes of parameters. **Middle:** Mean prediction errors. **Bottom:** Decision boundary. One should be noted that the horizontal axis in both the top and the middle panels is in log-scale.

2.6.4 MSE Versus No. of Data

From the middle panels in Figure 16, one should observe that the prediction error of a model decreases as the number of training steps increases. As each update requires a data to be fed in, the number of training steps can be in analog to the number of data fed.

Thus, the middle panels in Figure 16 could be showing the performance of a model with respect to the size of a dataset. In between $t = 10^1$ to 10^2, the performance drops in accordance with the so-called *Power Law*[6], i.e.

$$\log(\text{MSE}(N)) = \kappa_0 - \kappa_1 \log(N), \tag{53}$$

where κ_0 and κ_1 are constants.

Clearly, *Power Law* can only be observed when the model has not yet learnt enough. Once the model has learnt enough, say after $t = 10^3$, the change of the model performance does not fit for (53).

2.6.5 MSE Versus No. of Data for an MLP for MNIST

Power law manifests in the training of an MLP of the structure 784-100-100-10 for the MNIST dataset. Figure 17 shows the $\log(MSE)$ against the $\log(epoch)$ for a training of the MLP in the MNIST handwritten character recognition problem. The changes of *training MSE* follows scaling law in between the 2^{nd} epoch to the 30^{th} epoch. That is to say,

$$\log(MSE(epoch)) = \kappa_0 - \kappa_1 \log(epoch),$$

where κ_0 and κ_1 are constants. For $epoch \geq 100$, $MSE(epoch)$ has no significant change.

2.6.6 Scaling Law for (Stochastic) Gradient Descent

In Section 2.6.4 and Section 2.6.5, one should realize that the changes of the training MSE follows the so-called *scaling law*. Precisely, it is the *power law*. As a matter of fact, the property of *power law* has long been mentioned in the learning algorithms which are developed based on (stochastic) gradient descent[7]. It is known that the convergence rate of a (stochastic) gradient descent is $\mathcal{O}(1/t)$. That is to say,

$$MSE(t) \leq \overline{MSE}(t) = \mathcal{O}(1/t). \tag{54}$$

In other words, the upper bound on the change of $MSE(t)$, denoted as $\overline{MSE}(t)$, is given by

$$\log(\overline{MSE}(t)) = \kappa_0 - \kappa_1 \log(t), \tag{55}$$

where κ_0 and κ_1 are constants.

[6] It should be noted that *power law* is one so-called *scaling law*. There are many *scaling law* in the literatures. Another one is called *Moore's Law*. For a fixed area, the number of transistors (N) could be made double in every fixed number of months, i.e. $log(N(Year)) =$ const. $+ \gamma$ Year.

[7] *Power law* is a common statistical property that can be found in many naturally evolved social networks [10], such as friendship network and Internet.

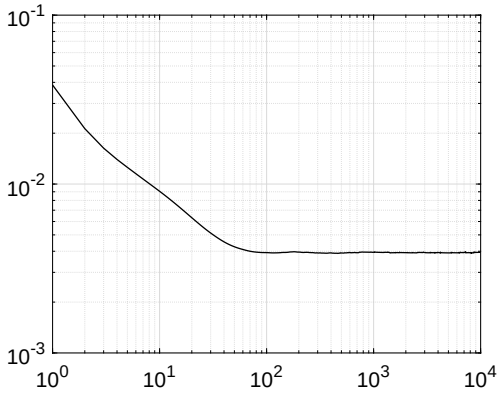


Figure 17: The $\log(MSE)$ against the $\log(epoch)$ for a training of the MLP in the MNIST handwritten character recognition problem. Here the network is an MLP of the structure 784-100-100-10. The online learning rate is 0.01. In each epoch, $60,000$ training samples are fed in. Therefore, the total number of steps in this online training is $6 \times 10^4 \times 10^4$.

2.6.7 Trial-and-Error Factors

From the context presented in this section, one should realize that the success of an MLP learning rule relies on at least seven factors. They are

1. the structure of the computational model, i.e. the network structure which includes the number of layers, the number of neurons in each layer and the mathematical model of a neuron;
2. the objective function (respectively, assessment measure or performance criteria) to be minimized;
3. the learning algorithm which can be developed by the method of gradient descent, the method of gradient descent with momentum or others;
4. the number of iterations (resp. stopping criteria) and the *validation* method;
5. the learning step μ_t;
6. the weight decay (equivalently, forgetting) factor and
7. the initial conditions of the model parameters, i.e. $\mathbf{w}(0)$.

All these factors can only be determined by many (or even numerous) trial-and-errors. For the stopping criteria, it can be a fixed number or it can be determined by the validation error.

3 Beyond Decision Making

The above presentations focus on the use of a multilayered network for solving a decision problem[8]. It is interesting to ask if these computational models are able to learn to generate a time sequence of outputs. Two types of sequence generations are usually considered.

1. Fixed number of outputs, say $y(t+1), y(t+2), \cdots, y(t+N)$. N is a predefined fixed number. In time series analysis, this problem is called N-step prediction. Predicting the average daily temperatures (resp. stock price) in the future seven days is one example.

2. Dynamic number of outputs. Text generation by an LLM is an example of this type.

3.1 Time Sequence Prediction/Generation

Let say, we have collected a sequence of closing prices of a stock in a consecutive N trading days. We would like to find a computational model which is able to predict the upcoming T trading days closing prices given the historical closing prices up to today. To do so, one approach is to design a computational model as shown in Figure 18.

3.2 Sequence Generation

By the same token, generation a sequence of data can be accomplished by applying a computational model $f(\mathbf{p}, \mathbf{w})$, as shown in Figure 19.

3.3 Time Window

For sequence generation problems, two factors determine the complexity of a problem to be investigated. They are the input time window width and the output time window width. Normally, the term *time window width* is simply called the *time window*. For a model with input window M and output window N, it needs M previous inputs to predict N future outputs.

Let say, in the time t, the model is going to predict values of the upcoming N instance. The inputs to the model are $p_t, p_{t-1}, \cdots, p_{t-M+1}$. The predictions are denoted by $\hat{p}_{t+1}$ to $\hat{p}_{t+N}$.

Accordingly, many types of sequence generation problems can be defined and depicted in Table 4.

[8]Here, a decision problem under our definition is that its outputs are binary numbers, i.e. $o_i \in \{0, 1\}$ for $i = 1, \cdots, n$. As a matter fact, either multilayered McCulloch-Pitts neuronal network or multilayered Perceptron could be applied in some multiple-input-multiple-output function approximation problems.

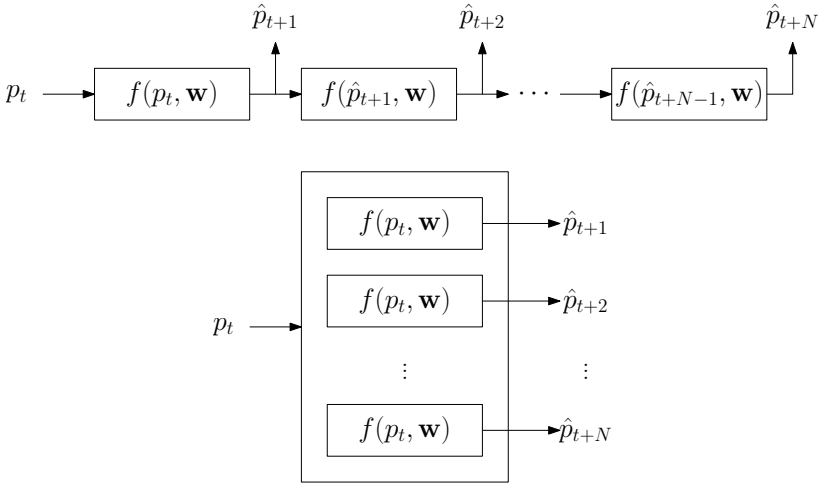


Figure 18: Computational model for a time sequence data prediction. p_t is the true value at time t. Based on this value, the computational model is applied to predict the values from $\hat{p}_{t+1}$ to $\hat{p}_{t+N}$ given the value of p_t. The prediction window width is fixed at N.

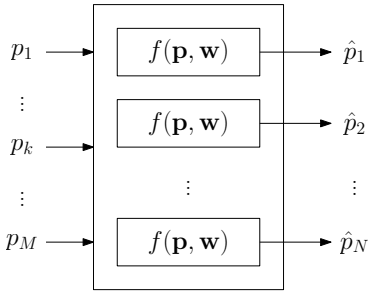


Figure 19: A computational model for sequence generation. Here, $\mathbf{p} = (p_1 \cdots p_M)^T$ are inputs to the model. The computational model thus generates the outputs $\hat{p}_1 \cdots \hat{p}_N$ in response to the inputs $p_1, \cdots, p_M$. The prediction window width is fixed at M.

Table 4: Different types of sequence generation problems.

Input	Output	Example	Problem Complexity
1	1	Stock price prediction	Simple
M	1	Stock price prediction	Simple
1	N	N-Step prediction	Difficult
M	N	Temperature prediction	Medium
M	N	Text generation	Difficult
Dynamic	N	Text generation	V Difficult
M	Dynamic	Text generation	VV Difficult
Dynamic	Dynamic	Text generation	VVVVV Difficult

If a number is shown for an input (resp. output), the window is fixed.

3.4 Text Generation

From Figure 19, one can note that a computational model $f(\cdot, \mathbf{w})$ could be designed (equivalently, trained) to generate a sequence of texts if $\mathbf{p}$ is a sequence of text-input (i.e. prompt) of M words. The outputs is a sequence of N words. The key idea is to design a computational model with recurrent connections, as shown in Figure 20.

3.4.1 Word Embedding

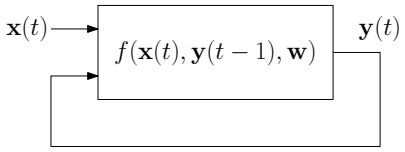
For text generation problems, one key problem is how to encode a word in a numerical value(s). For more than a decade, this problem has been a tough research problem. Eventually, *word embedding* algorithms for English words and Chinese words have been developed. Each word is encoded by a multi-dimensional numerical vector, Figure 21.

3.4.2 Language Dependent

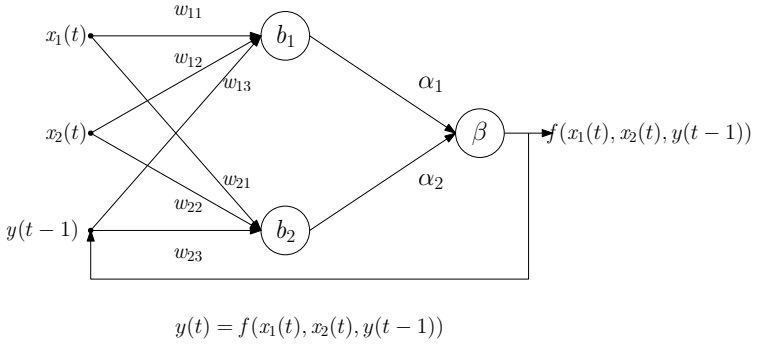
Word embedding algorithms for different languages are clearly end up with different large language models (LLMs). As a word embedding algorithm is designed for a particular language, different word embedding algorithms are needed to be design for different languages. It could be reason why an English-oriented LLM performs different from a Chinese-oriented LLM.

3.4.3 Non-Explainable

From its text generation ability, each LLM could *demonstrate* that it has learnt some regularities in text generation. However, these regularities cannot be found or explained by the structure and the parameters of the computational model. Thus, these LLMs are not explainable.

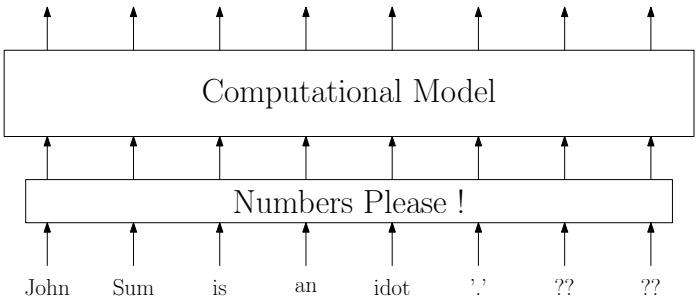


(a) Block diagram.



(b) Simple recurrent MLP.

Figure 20: The block diagram of a recurrent network (a), with output feeding back to the input of the network. The precise model in the square box in (a) can be any model. If the model is defined as an MLP, its structure is shown in (b). This structure is commonly called a recurrent MLP (RMLP). Its learning rule is simple as compared with other recurrent networks. The symbol $\mathbf{w}$ is the model parametric vector referring the collections of all model parameters. Again, the key is to find the parameters for the computational model so that the network is able to generate the sequence of data $\{\mathbf{x}(t), y(t)\}_{t=1}^{T}$ and the initial condition $y(0)$. The computational model structure of all LLMs is typically designed along this idea.



Prompt: John Sum is an idot.

Figure 21: Each word or each phrase is converted to a vector of multiple numerical numbers to be input to the computational model. The method of converting each word or phrase to a vector of numbers is called *word embedding*. Note that word embedding has been a research topic in natural language processing. Many word embedding methods have thus been developed. Clearly, word embedding is language-specific.

4 Deep Neural Networks

Sigmoid neuronal networks as introduced in Section 2 have been a major type of AI models for use in the 1980s to the 1990s. While the idea of multiple layers was introduced, the models for applications could only be designed as a single layer or two layers network; and the number of neurons in a layer is not large.

4.1 Multiple Layers

Owing to the advancement in computational power of a computer from the late 1990s to the 2010s, larger scale neural network models were introduced from the late 1990s to the 2010s [1, 11, 12, 13, 14]. These models have normally more than five layers and some might have more than ten layers. The number of neurons in a layer can have more than hundreds neurons. These models are called *deep neural networks* [15, 16] and the learning theory for these deep neural networks is called *deep learning*.

4.2 Large-Scale

For application purposes, these deep neural networks have some structures which are different from the multilayered Perceptrons (MLP). First, the scale of a deep neural network is much larger than a conventional MLP in the 1990s. A deep neural network could consist of ten thousands or even millions number of model parameters. The neural network consists of large number of layers and the number of neurons in a layer could be hundreds to thousands.

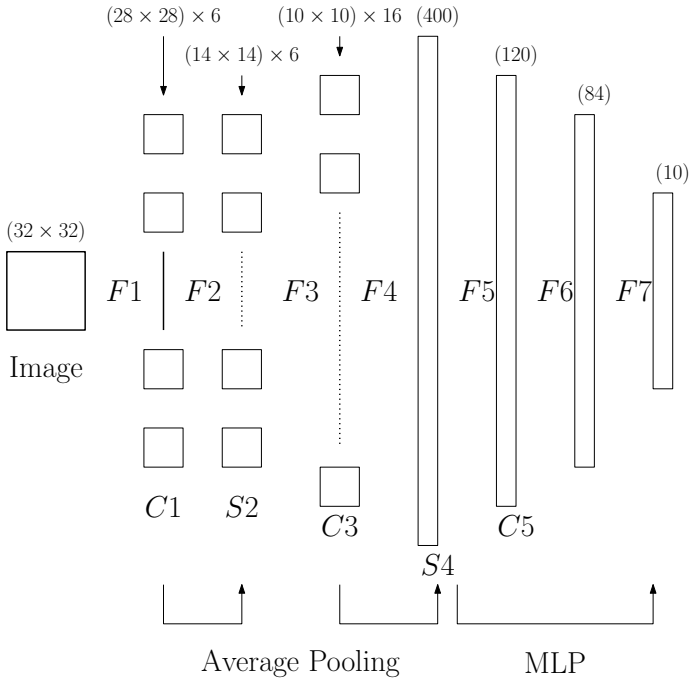


Figure 22: Model structure of the LeNet 5 in [1]. The F1, F3 and F5 layers perform convolution. So, these layers are called convolution layers as well. Note that only the neurons in the F5 and F6 layers are sigmoidal neurons. The neurons in the other layers are not.

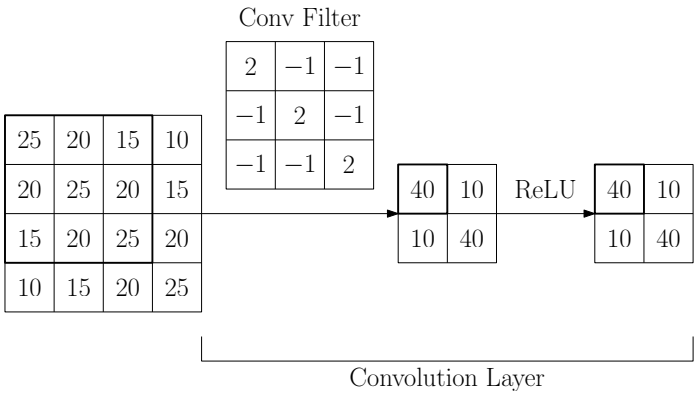


Figure 23: The working principle of a convolution layer with ReLU neurons. Here, the receptive field of a convolution filter is of size 3×3.

4.3 Convolution Layers

Second, convolution layers are usually added in a deep neural network for solving image processing or pattern recognition problems. These convolution layers mimic the biological properties of the early image processing in a human visual system. This idea was first appeared in *Cognitron* and *Neocognitron* introduced in [17, 18] in the 1970s, in *LeNet*[9] introduced in [1] in the 1990s and later in *AlexNet* introduced in [12] in the 2010s.

Figure 23 shows the working principle of a convolution layer with ReLU neurons. Here, the receptive field of a convolution filter is of size 3×3. For an image with size $M \times M$, the output image after convolution is of size $(M-2) \times (M-2)$. In the example as shown in Figure 23, the size of the input image is 4×4. Thus, the output image is of size 2×2.

4.4 Rectified Linear Neuron (ReLU)

Third, a new neuron model called rectified linear neuron as shown in Figure 24 is employed. Rectified linear neuron was first investigated in [19, 20] and later applied in the neural network models for pattern recognitions [17, 18]. The mathematical model for a rectified linear neuron is given by

$$f(x) = \max\{0, x\}. \tag{56}$$

As a comparison, the interpretations of the input and output values of different neurons are depicted in Table 5.

[9] Its structure is shown in Figure 22.

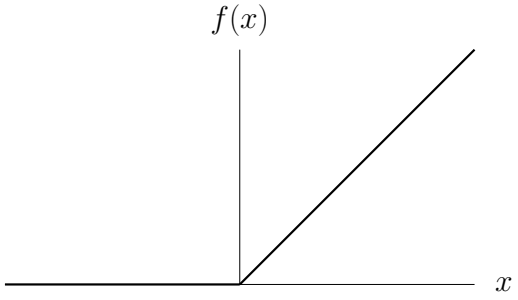


Figure 24: A rectified linear unit neuron. Its transfer function is defined as $f(x) = \max\{0, x\}$.

Table 5: Comparisons of different neuron models.

Model	Input	Output
McCulloch-Pitts	Pulses of firing rate r	Pulses of firing rate r
Sigmoid neuron	Pulses of firing rate x	Pulses of firing rate y
ReLU neuron	Pulses of firing rate x	Pulses of firing rate y
Electronic neuron	Voltage x	Voltage y

4.5 Tackling Vanishing Gradient by ReLU

In the research of deep neural network, the main reason for using rectified linear neuron is to tackle the so-called *vanishing gradient problem*. Consider a MLP with structure as shown in Figure 25. The output is given by

$$\begin{aligned} f(x_1, x_2) &= \phi(\alpha_1 z_1(x_1, x_2) + \alpha_2 z_2(x_1, x_2) - \beta) && (57) \\ z_1(x_1, x_2) &= \phi(w_{11}x_1 + w_{12}x_2 - b_1) && (58) \\ z_2(x_1, x_2) &= \phi(w_{21}x_1 + w_{22}x_2 - b_2), && (59) \end{aligned}$$

where $\phi(\cdot)$ is the sigmoid function defined as follows :

$$\phi(s) = \frac{1}{1 + \exp(-s)}.$$

4.5.1 Derivative of $\phi(s)$

One should note that the derivative of $\phi(s)$ with respect to s is given as follows :

$$\begin{aligned} \phi'(s) &= \frac{-1}{(1 + \exp(-s))^2} (-\exp(-s)) \\ &= \frac{\exp(-s)}{(1 + \exp(-s))^2} \\ &= \phi(s)(1 - \phi(s)). \end{aligned}$$

As $0 \leq \phi(s) \leq 1$, we can get that

$$0 \leq \phi'(s) \leq 1/4. \tag{60}$$

The derivative of $\phi(s)$ is bounded by 0 and 1/4. This property on $\phi(s)$ and $\phi'(s)$ becomes the key reason leading to the vanishing gradient problem in a deep sigmoid network.

4.5.2 Learning for a Two-Layer MLP

Consider the online model learning for the update the parameter w_{ij} in the input layer, one needs to compute the following learning equation.

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) + \mu_t e(t) g_0(t) \alpha_i(t) \sum_{i=1}^{2} g_i(t) x_{tj} \\ &= w_{ij}(t) + \mu_t \sum_{i=1}^{2} e(t) g_0(t) \alpha_i(t) g_i(t) x_{tj} \\ &= w_{ij}(t) + \mu_t \sum_{i=1}^{2} \left[g_0(t) g_{1i}(t) \right] e(t) \alpha_i(t) x_{tj}, \end{aligned} \tag{61}$$

where

$$\begin{aligned} e(t) &= (d_t - f(\mathbf{x}_t, \mathbf{w}(t))) & (62) \\ g_0(t) &= \phi(u_0(t))(1 - \phi(u_0(t))) & (63) \\ g_i(t) &= \phi(u_i(t))(1 - \phi(u_i(t))) & (64) \end{aligned}$$

for $i = 1, 2$ and

$$\begin{aligned} u_0(t) &= \alpha_1(t) z_1(x_{t1}, x_{t2}) + \alpha_2(t) z_2(x_{t1}, x_{t2}) - \beta(t), \\ u_1(t) &= w_{11}(t) x_{t1} + w_{12}(t) x_{t2} - b_1(t), \\ u_2(t) &= w_{21}(t) x_{t1} + w_{22}(t) x_{t2} - b_2(t). \end{aligned}$$

4.5.3 Vanishing Gradient

Recall from (60) that $0 \leq \phi(s)(1 - \phi(s)) \leq 1/4$. Thus, the values $g_0(t)$ and $g_i(t)$ must be smaller than 1/4. Thus, $g_0(t)g_1(t)$ or $g_0(t)g_2(t)$ must be smaller than 1/16. Subsequently, the update of a parameter deep in the network might be vanished if the number of layers is large.

4.5.4 ReLU

Take LeNet 5, Figure 22, as an example. If all neurons in the model are sigmoidal neurons, the parameters at the F6 and F7 layers (closer to output layer) will bigger changes in each step of learning. The parameters at the F1 and F2 (closer to the inputs) will get very small changes in each step of learning.

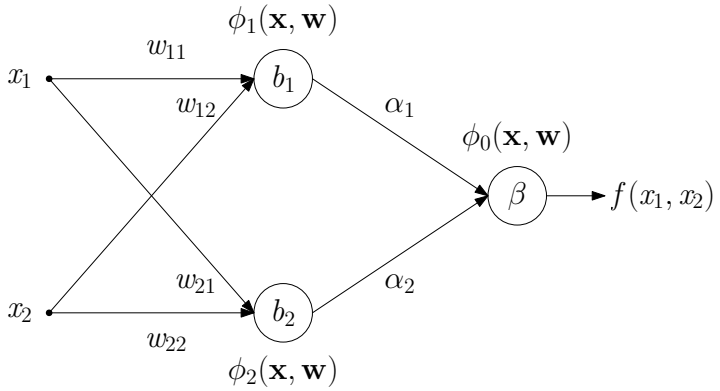


Figure 25: A MLP with three neurons. Note that $f(\mathbf{x}, \mathbf{w}) = \phi_0(\mathbf{x}, \mathbf{w})$.

If the neurons in the Figure 25 are replaced by ReLUs, the update of w_{ij} stated in (61) is replaced by the following update equation.

$$w_{ij}(t+1) = w_{ij}(t) + \mu_t \sum_{i=1}^{2} [g_0(t) g_i(t)]\, e(t) \alpha_i(t) x_{tj}, \tag{65}$$

where

$$q_0(t) = \begin{cases} 1 & \text{if } \alpha_1(t)\phi_1(t) + \alpha_2(t)\phi_2(t) - \beta(t) > 0, \\ 0 & \text{if } \alpha_1(t)\phi_1(t) + \alpha_2(t)\phi_2(t) - \beta(t) \leq 0, \end{cases} \tag{66}$$

and

$$q_i(t) = \begin{cases} 1 & \text{if } w_{i1}(t)x_{t1} + w_{i2}(t)x_{t2}(t) - b(t) > 0, \\ 0 & \text{if } w_{i1}(t)x_{t1} + w_{i2}(t)x_{t2}(t) - b(t) \leq 0, \end{cases} \tag{67}$$

for $i = 1, 2$. in this regard, the update of $w_{ij}(t)$ can be written as follows :

$$w_{ij}(t+1) = \begin{cases} w_{ij}(t) + \mu_t \sum_{i=1}^{2} e(t)\alpha_i(t)x_{tj} & \text{if } q_0(t) = q_i(t) = 1, \\ w_{ij}(t) & \text{if } q_0(t) = 0 \text{ or } q_i(t) = 0. \end{cases} \tag{68}$$

A weight w_{ij} updates if and only if the output of $\phi_i(t) > 0$ and $\phi_0(t) > 0$.

4.6 Softmax Output Neurons

Apart from application of ReLU neurons, Softmax neurons are applied in the output layer of a deep neural network. To illustrate the processing of Softmax neurons, Figure 26 shows a network with four neurons. The neurons in the first layer are ReLU neurons. The neurons in the output layer are Softmax neurons. The definition of a Softmax neuron is defined as follow :

$$f_i(\mathbf{x}, \mathbf{w}) = \frac{\exp(-\phi_i(\mathbf{x}, \mathbf{w}))}{\sum_{j=1}^{n} \exp(-\phi_j(\mathbf{x}, \mathbf{w}))}. \tag{69}$$

It is clear $0 \leq f_i(\mathbf{x}, \mathbf{w}) \leq 1$ for $i = 1, \cdots, n$ if there are n outputs.

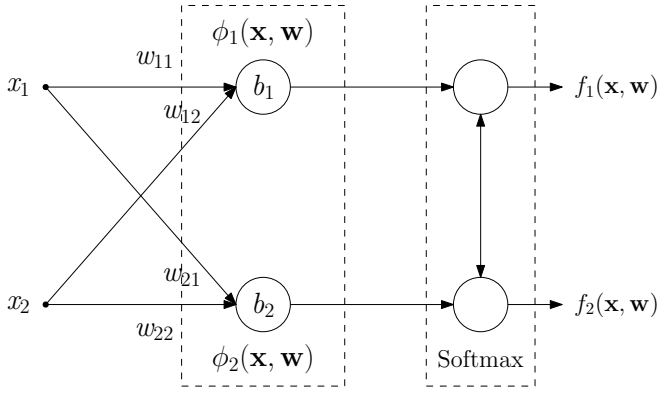


Figure 26: A MLP with four neurons, in which two of them are ReLU neurons and the other two are Softmax neurons. Note that $f_1(\mathbf{x}, \mathbf{w})$ and $f_2(\mathbf{x}, \mathbf{w})$ are the outputs.

Applying Softmax neurons for a n-class classification problem, the label $\mathbf{d}_k$ of the input vectors $\mathbf{x}_k$ has to be defined as follows :

$$\mathbf{d}_k = (0, \cdots, 0, \underbrace{1}_{i}, 0, \cdots, 0)^T \tag{70}$$

if $\mathbf{x}_k$ is belongs to the i^{th} class.

4.7 Application of GPU

While the rectified linear neurons are applied and the convolution layers are defined, the learning algorithm developed for a deep neural network is still computational intensive. In the end, training a deep neural network and sometimes in the use of a deep neural network require the use of GPU(s). Application of (multiple) GPU becomes a critical factor. For a large-scale deep neural network, GPU might also be applied in application.

Typically, the time spent on a floating point multiplication in a CPU is hundreds time longer than the time spent on a floating point multiplication in a GPU. If an GPU is able to support vector and matrix computation, the computational time will be much shorter.

4.8 Dedicated Processing Unit

For some deep neural networks with complicated architectures, specialized processing units are sometimes designed and implemented for applications. Those applications include the usages of object recognition models for auto-driving.

4.8.1 XPU

Those specialized processing units are named *tensor processing unit* (TPU), *image processing unit* (IPU) and *neural processing unit* (NPU). Those processing units are designed not specific for graphical processing. Thus, their designs are clearly different from the design of a GPU.

4.8.2 Support Real-Time Applications

No matter what, the ultimate purpose of these dedicated processing units is to accelerate the processing power of a computational model when it is being used for real-time applications.

5 Computational (AI) Model Development

To summary, Figure 27 shows the key steps in a computational (AI) model development. It includes at least three important steps, namely the hypothetical computational model, the performance criteria and the learning algorithm.

5.1 The Model & The Performance Criteria

The initial step is clearly on the computational model hypothesized. Two different types of models can be hypothesized, namely predictive model and generative model. These models are usually associated with different performance criterion.

5.1.1 Predictive Model

For a model simply for predicting the output $\mathbf{f}(\mathbf{x}, \mathbf{w})$ given an input $\mathbf{x}$, a predictive (resp. decision) model like MLP is good enough. For this type of models, the performance criteria can simply be defined as the *mean square errors* given by

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{k=1}^{N} \|\mathbf{d}_k - \mathbf{f}(\mathbf{x}_k, \mathbf{w})\|^2, \tag{71}$$

where $\{(\mathbf{x}_k, \mathbf{d}_k)\}_{k=1}^N$ is the given training dataset. In practice, a computational model with smaller $\|\mathbf{w}\|$ usually gives better results. In this regard, the performance criteria can be defined as follows :

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{k=1}^{N} \|\mathbf{d}_k - \mathbf{f}(\mathbf{x}_k, \mathbf{w})\|^2 + \kappa_1\|\mathbf{w}\|^2, \tag{72}$$

where $0 < \kappa_1 \ll 1$ is a positive constant (equivalently, weighting factor).

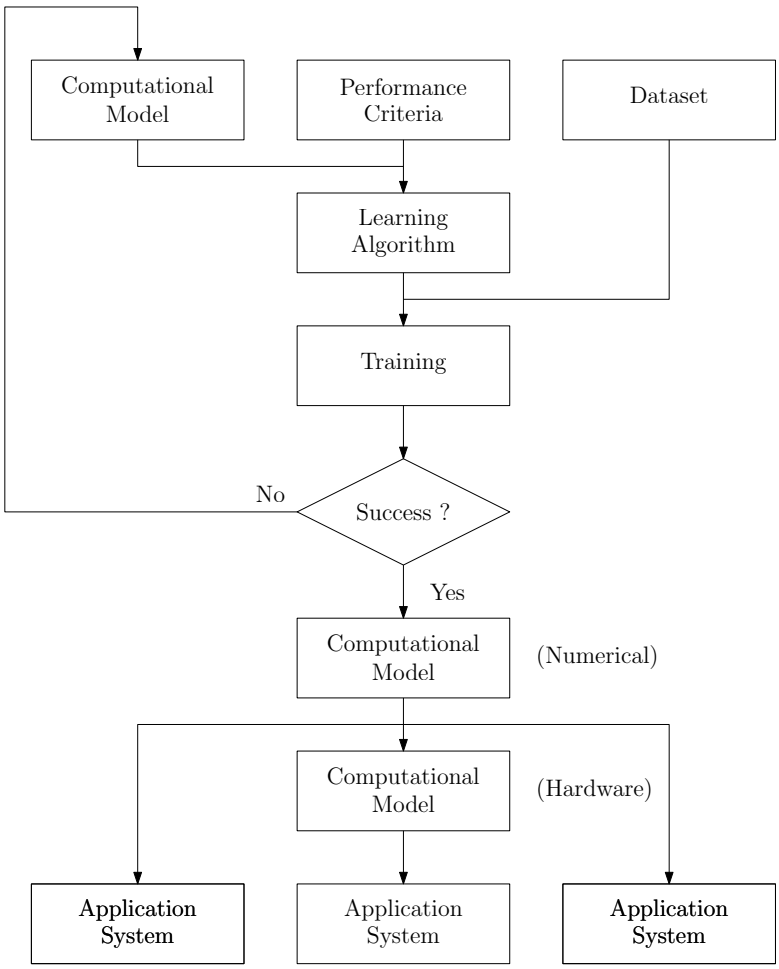


Figure 27: Development process for an AI computational model. Initially, a thought computational model is hypothesized. Together with the model, a performance criteria is defined for the problem. Depending on the nature of the model, the criteria could be the mean square errors (MSE) or maximum likelihood. Once the performance criteria has been defined, the learning algorithm for the model can be derived. Training is clearly yet another computational task which can be conducted by a computer with or without GPUs. After the learning process is completed successfully, a *numerical computational model* is obtained. If not, search for another computational model and then repeat the process. If using a computer (resp. a cloud) is not efficient enough to use the trained model, a *hardware* computational model might be built. Finally, application systems can be built on top of these computational model.

5.1.2 Generative Model

For a generative AI model, it could be a stochastic model like *factor analysis* (FA) and *structural equation model* (SEM). With the model, a performance criteria is defined. Depending on the nature of the model, the criteria could be the mean square errors (MSE) for a deterministic model or the *maximum likelihood* for a stochastic model.

A generative model is essentially a probabilistic model. Given set of training data, say $\{(\mathbf{x}_k, \mathbf{d}_k)\}_{k=1}^N$, we can have the distribution of the samples over a high-dimensional space, denoted as $P(\{(\mathbf{x}_k, \mathbf{d}_k)\}_{k=1}^N)$. The purpose of a generative model with parametric vector $\mathbf{w}$ is to generate a set of N data whose distribution $P(\{(\mathbf{x}'_k, \mathbf{y}'_k)\}_{k=1}^N, \mathbf{w})$ is identical to the distribution of the samples.

The probability of the model giving output $\mathbf{y}$ is given by $P(\mathbf{y}|\mathbf{x}, \mathbf{w})$. The *likelihood* of a generative model with $\mathbf{w}$ giving $\mathbf{d}_k$ with input $\mathbf{x}_k$ is thus $P(\mathbf{d}_k|\mathbf{x}_k, \mathbf{w})$. Therefore, the *likelihood* of the generative model given $\mathbf{d}_k$ with input $\mathbf{x}_k$ is thus $P(\mathbf{d}_k|\mathbf{x}_k, \mathbf{w})$ for $k = 1, \cdots, N$ is given by

$$\text{Likelihood} = P\left(\{(\mathbf{x}_k, \mathbf{d}_k)\}_{k=1}^N \middle| \mathbf{w}\right) = \prod_{k=1}^N P(\mathbf{d}_k|\mathbf{x}_k, \mathbf{w}). \tag{73}$$

Hence, the performance criteria can be defined as the log-likelihood function.

$$\mathcal{L}(\mathbf{w}) = \sum_{k=1}^N \log P(\mathbf{d}_k|\mathbf{x}_k, \mathbf{w}). \tag{74}$$

The meaning of *likelihood* could be understood in the following. If a generative model is with parametric vector $\mathbf{w}$, its probability (i.e. likelihood) of generating the dataset $\{(\mathbf{x}_k, \mathbf{d}_k)\}_{k=1}^N$ is given by (73).

If it is assumed that the model parameters follow certain probability distribution $P(\mathbf{w})$, the performance criteria $\mathcal{L}(\mathbf{w})$ can be re-defined as follows :

$$\mathcal{L}(\mathbf{w}) = \sum_{k=1}^N \log P(\mathbf{d}_k|\mathbf{x}_k, \mathbf{w}) + \kappa_1 \log\left(P(\mathbf{w}\right), \tag{75}$$

where $0 < \kappa_1 \ll 1$ is a weighting factor.

5.2 Learning Algorithm Development

Once the performance criteria $\mathcal{L}(\mathbf{w})$ has been defined, the problem of getting a $\mathbf{w}$ in which $\mathcal{L}(\mathbf{w})$ is a local minimum is equivalent to an optimization problem. Thus, numerical algorithms for optimization problems can thus be applied to design the learning algorithms. Two common approaches for the development of a large computational model learning are *gradient descent* (GD) and *gradient descent with momentum* (GDM).

5.2.1 Gradient Descent (GD)

With the performance criteria defined, a learning algorithm can be defined based on the ideas of gradient descent (GD) or gradient descent with momentum (GDM). The learning algorithm developed by gradient descent is given by

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \mu_t \frac{\partial \mathcal{L}(\mathbf{w}(t))}{\partial \mathbf{w}}, \tag{76}$$

where μ_t is the learning step size at time t.

5.2.2 Gradient Descent with Momentum (GDM)

The learning algorithm developed by GDM is given by

$$\begin{aligned} \mathbf{v}(t) &= (1-\kappa_t)\mathbf{v}(t-1) + \kappa_t \frac{\partial \mathcal{L}(\mathbf{w}(t))}{\partial \mathbf{w}} && (77) \\ \mathbf{w}(t+1) &= \mathbf{w}(t) - \mu_t \mathbf{v}(t), && (78) \end{aligned}$$

where κ_t and μ_t are learning step sizes at time t.

5.2.3 Pseudo Brute-Force Search

Gradient descent and gradient descent with momentum are commonly applied in very large scale computational model, say the number of parameters is larger. For a small scale computational model, pseudo brute-force search like stochastic search and MCMC methods can be applied.

Before 2010s, the idea of stochastic search or MCMC method could only be applied in small size computational model. With the application of GPU in AlexNet [12], many text-to-image computational models like *generative adversarial network* (GAN) [21, 22] are subsequently developed based on this approach.

5.3 Training

Training is clearly yet another computational task which can be conducted by a computer with or without GPUs. If the learning algorithm is computationally intensive, special computing platform might be needed. In this regard, building a platform with many computers and GPUs might be needed.

In the end of training, the model obtained can be implemented for use. If the model obtained cannot meet an acceptance level of performance, the developer needs to re-design the computational model and go through the previous steps again[10].

[10]The time in search of a computational model which can give acceptable performance is highly uncertain. It could range from days to years. During the search period, developers have to go through many trial-and-error. The number of trial-and-errors is not anticipated.

5.4 Implementations of a Pre-Trained Model

After the training process is completed and the acceptance level of performance has achieved, a *numerical computational model* is obtained.

5.4.1 Software Implementation

The numerical computational model with the trained parameters can thus be implemented as a program running in a cloud platform as a service for the application developers to develop application systems. Today, many object recognition CNNs and LLMs are deployed as software on a cloud.

Apart from deploying a CNN dor LLM on a cloud, some CNNs and LLMs can be installed in a powerful desktop computer if their model sizes are small and they are open-source programs. A disadvantage is that a user will need to manually download the latest version of the computational model.

5.4.2 Hardware Implementation

It is clear that software implementation of a computational model with millions of parameters can support non-real-time applications. However, for a real-time application like auto-driving, software implementation is definitely not efficient as the decision of the computational model must be made in less than 0.01s or less. In this regard, specialised *hardware* must be built to implement the computational model. A disadvantage of hardware implementation is that the version of the hardware computational model cannot be updated once it has been made. Exemplar models include those CNNs which are applied in object recognition for auto-driving systems.

5.5 Application System Development

Once a stable computational model has been released. Application system can be developed by using the computational model as its core. Today, many pre-trained computational models are available and many of them have been applied in applications. Some of these computational models are depicted in Table 6. Some exemplar computational models and their applications are depicted in Table 7.

5.5.1 User Interface Design

For the applications like Google Assistant, Amazon Alexa and Microsoft Copilot, their system architectures are shown in Figure 28. To support the intelligent services, specialized computing platforms with large number of GPUs are needed.

5.5.2 Transfer Learning for Fine-Tune

For certain applications, application system developers might need to *fine-tune* the so-called *pre-trained* computational model fitting for such applications. For

Table 6: List of pre-trained computational models (Update: April 5, 2025).

Developer	Founded	Text (LLM)	Image	Video	Music	Reasoning	Agent[b]
Adobe	1982	–	Firefly	–	–	–	??
Amazon	1994	Titan	Titan	Titan	–	–	??
Anthropic	2021	Claude	–	–	–	–	Claude 3.5 Sonnet
BigScience[a]	2021	Bloom	–	–	–	–	??
Canva	2013	–	Magic Studio	–	–	–	??
Cohere	2019	Cohere	–	–	–	–	??
Deep AI	2017	√	√	√	√	–	??
DeedSeek	2023	DeedSeek V3	–	–	–	DeekSeek R3	??
Google	1998	Gemini	Imagen	Veo	MusicLM	Gemini Pro	??
Meta	2004	Llama	Imagine	Emu Video	MusicGen	Llama 3.2	??
Midjourney	2022	–	Midjourney	–	–	–	??
Mistral AI	2023	Mistral	–	–	–	–	??
OpenAI	2015	ChatGPT	DALL-E	Sora	Jukebox	o3	Operator
Stability AI	2019	StableLM	Stable Diffusion	√	√	–	??
TII[b]	2020	Falcon	–	–	–	–	??
xAI	2023	Grok-3	Aurora	–	–	Grok-3	??

[a] BigScience Collaborative Initiative. [b]Technology Innovation Institute(TII), UAE.

[b] It is clear that agentic AI model development is an inevitable trend. With agentic AI system, a manager of a firm is able to design the task to be completed by an agentic AI. If it succeeds, adminstration and operational staffs who are working for survey, analysis and documentation could be largely eliminated.

Table 7: Exemplar computational models and their applications. Development of an application system on top of these computational models could be a difficult task. The development is not just user interface development. It might involve the processes of fine-tune, transfer learning, hardware design and others.

Model (Developer)	**Application (Developer)**
Image Processing	
AlexNet (UoT)	Auto-Drive (Tesla)[(a)]
VGG (Oxford)	Auto-Drive (Tesla)[(a)]
ResNet (Microsoft)	Auto-Drive (Tesla)[(a)]
GoogLeNet (Google)	Auto-Drive (Tesla)[(a)]
AlexNet (UoT)	Medical Image Diagnosis[(b)]
VGG (Oxford)	Medical Image Diagnosis[(b)]
ResNet (Microsoft)	Medical Image Diagnosis[(b)]
GoogLeNet (Google)	Medical Image Diagnosis[(b)]
Natural Language Processing	
Neural Machine Translation (Google)	Google Translate (Google)[(d)]
DeepSeek (DeepSeek)	LM Studio (LM Studio)[(c)]
Llama-7B (Meta)	LM Studio (LM Studio)[(c)]
Mistral-7B (Mistral)	LM Studio (LM Studio)[(c)]
ChatGPT (OpenAI)	Copilot (Microsoft)[(d)]
Amazon LLM (Amazon)	Alexa (Amazon)[(d)]
Gemini (Google)	Search (Google)[(d)]
Gemini (Google)	Google Assistant (Google)[(d)]
Text-to-Image	
DALL·E (OpenAI)	Copilot (Microsoft)[(d)]

[(a)] Specialized processors are built to implement such computational model. [(b)] Many research groups have developed application systems for medical image diagnosis. [(c)] Apart from LM Studio, many tech firms have developed similar application systems for use in a powerful desktop computer like AIPC. [(d)] Those application systems are developed in a cloud-based platform.

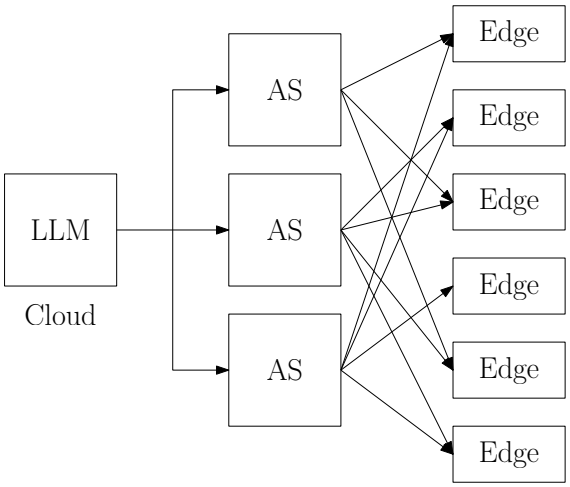


Figure 28: System architecture of AI applications deployment. On the cloud or server side, in which the computational models are installed, specialized computing platforms with large number of GPUs are needed. Here, LLM stands for large language model and AS stands for application system. The edge corresponds to a user device. It could be a desktop computer, a notebook computer, a pad, a cell phone, a car system, the devices in a home network or any device being used in the user side.

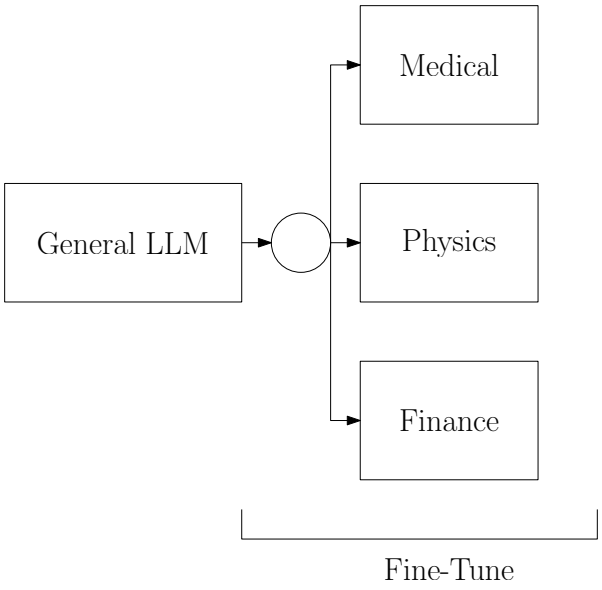


Figure 29: Idea behind mixture of experts computational models. Here, the computational model is applied in three domains, namely medical, physics and finance. During the fine-time stage, the corresponding dataset is fed in to train the specific fine-tune model. During fine-tune of an expert, the general LLM is considered as a fixed model. Finally, the selector (the circle) is trained to give correct selection of expert(s) to generate the output.

a computational model with multiple layers, a developer could *fine-tune* the parameters at the few layers near the output instead of fine-tune the whole computational model. Besides, a developer can simply apply a partial *pre-trained* computational model as a foundation and add a MLP as a output model to generate the required outputs.

5.5.3 Mixture of Experts (MoE)

Clearly, an AI computational model should be applied to a wider range of applications. Training a wide-range applicable computational model is time consuming and even not performance well. Therefore, one approach is to apply a pre-trained model to adapt to specific expert domains, say in the area of cancer and in the area of financial investment. Figure 29 shows the generic idea of a computational model with mixture of experts.

Fine-Tune an expert : As a result, multiple fine-tune experts can be obtained. Each expert, i.e. again an AI model, can tackle the problems specialized in its own domain. Once a question has been asked to the AI system, the system could identify the domain in which the question is belongs to and then trigger

the corresponding expert AI subsystem to response to the question.

Training time reduction : As each fine-tune training is limited to a specific domain, the time spent could be shortened. Besides, fine-tune training for all expert subsystems can be conducted in parallel. In the end, the time spent on training a model with mixture of experts scenario is much shortened than a single general model.

Old idea : One should be noted that the idea of MoE has been applied in many engineering system designs long before 1970s [23, 24]. A key design challenge is to determine under what input condition which expert has to take action.

6 Brain, Electronic Brain and Computer

Long in the history, many scholars have attempted to make an artificial brain which can replicate the behaviors of a human brain. Pereceptron is one of them. Perceptron could be considered as an electronic brain. For the current released LLMs, the artificial brain is made of computers. Therefore, it is needed to make comparisons among a human brain, an electronic brain and a computer. Their comparisons are depicted in Table 8.

6.1 Processing Unit

In a human brain, the processing units are clearly the biological neurons. For those electronic brains, the processing units are electronic neurons. An electronic neuron is basically an electric circuit with electronic components. An electronic neuron is usually designed to implement the behavior of a sigmoid neuron as stated in (29) and shown in Figure 13. For an AI system implemented on a computer, its processing units are clearly the logic gates in the computer.

6.2 Model Structure

In regard to the model structure, human brain is a network of biological neurons. For an electronic brain, it is a network of electronic components which implements a pre-designed computational model. For an AI system running on a computer, its structure is also a (software) computational model.

6.3 Detail Structure & Signal Flow

While a human brain is a network of biological neurons, its detail structure and signal flow are largely unknown. They are still under research. On the other hand, the detail structure and signal flow in an electronic brain (resp. a computer) are known as an electronic brain is designed by engineers based on pre-designed circuits.

6.4 Speak, Listen and See

Today, an electronic brain (resp. computer) can be designed to connect with loudspeaker, microphone and camera to speak, listen and see. In terms of environmental interactions, an electronic brain and a computer can behave the same as a human brain.

6.5 Intelligence

From intelligence point of view, it is commonly agreed that a human brain is intelligent. For an electronic brain or a computer, they are pre-designed or pre-programmed for solving problems. Thus, an electronic brain and a computer should not be considered as having intelligence.

6.6 Learning Mechanism

For a human brain, actual neuron-level learning mechanism is still under research. One common believe is that the properties of some synapses might change during a learning process. However, the changes in a (biological level) neuronal network in relation to a (psychological level) reinforcement learning is still unclear and under research.

For an electronic brain and a computer, their learning mechanisms are pre-designed and hence programmed. For an electronic brain, some changes might be found in some electronic components if it is in the process of learning. For a computer, there is no any change in the properties of the logic gates during learning.

6.7 Electronic Brain

One should be noted that the issues depicted in this table are not just interested in the area of AI, they are interested in the areas of psychology, cognitive science, neuroscience, brain science and philosophy. It is not to mention the area in electrical engineering and computer science which are willing to build an electronic brain (resp. non-biological brain) functioning as a human brain.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[3] M. Minsky and S. Papert, *Perceptrons : An introduction to computational geometry.* MIT Press, 1969.

Table 8: Comparisons among a human brain, an electronic brain and a computer.

	Human Brain	Electronic Brain	Computer
Processing unit	Neuron	Electronic neuron	Logic gate
Structure	Net. of neurons	Comp. model	Comp. model
Detail structure	Under research	Known	Known
Detail signal flow	Under research	Known	Known
Speak	Yes	Yes	Yes
Listen	Yes	Yes	Yes
See	Yes	Yes	Yes
Intelligence	Under research	Programmed	Programmed
Learning	Under research	Programmed	Programmed

During learning, the properties of some synapses in a human brain change. For an electronic brain, physical properties of some electronic components might change. However, there is nothing change in a computer even an AI model is under learning. One should be noted that the issues depicted in this table are not just interested in the area of AI, they are interested in the areas of psychology, cognitive science, neuroscience, brain science and philosophy. It is not to mention the area in electrical engineering and computer science which are willing to build an electronic brain functioning as a human brain.

[4] B. Widrow, "Generalization and information storage in networks of Adaline neurons," in *Self-Organizing Systems*. Spartan Books, 1962, pp. 435–461.

[5] F. Rosenblatt, "The Perceptron: a probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, no. 6, p. 386, 1958.

[6] ——, "Perceptron simulation experiments," *Proceedings of the IRE*, vol. 48, no. 3, pp. 301–309, 1960.

[7] ——, *Principles of Neurodynamics: Perceptions and the theory of brain mechanisms*. Spartan, 1962.

[8] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," *PhD Dissertation, Harvard University*, 1974.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[10] A.-L. Barabási and E. Bonabeau, "Scale-free networks," *Scientific American*, vol. 288, no. 5, pp. 50–9, 2003.

[11] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[14] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[16] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[17] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological Cybernetics*, vol. 20, no. 3-4, pp. 121–136, 1975.

[18] ——, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.

[19] H. K. Hartline and C. H. Graham, "Nerve impulses from single receptors in the eye." *Journal of Cellular & Comparative Physiology*, vol. 1, no. 2, pp. 277–295, 1932.

[20] H. K. Hartline, "Intensity and duration in the excitation of single photoreceptor units." *Journal of Cellular & Comparative Physiology*, pp. 229–247, 1934.

[21] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[22] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," *arXiv preprint arXiv:1612.07828*, 2016.

[23] D. G. Lainiotis, "Partitioning: A unifying framework for adaptive systems, I: Estimation," *Proceedings of the IEEE*, vol. 64, no. 8, pp. 1126–1143, 1976.

[24] ——, "Partitioning: A unifying framework for adaptive systems, II: Control," *Proceedings of the IEEE*, vol. 64, no. 8, pp. 1182–1198, 1976.

[25] M. D. McDonnell, M. D. Tissera, T. Vladusich, A. van Schaik, and J. Tapson, "Fast, simple and accurate handwritten digit classification by training shallow neural network classifiers with the 'Extreme Learning Machine' algorithm," *PloS ONE*, vol. 10, no. 8, p. e0134254, 2015.

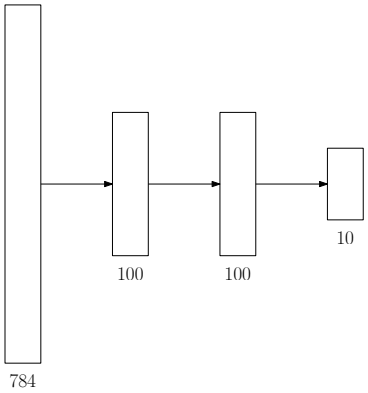


Figure 30: Structure of an MLP for MNIST handwritten digit recognition. The input image is of size 28×28. This MLP could be considered as a specialized convolution neural network (CNN), in which the first layer consists of 100 convolution filters of size 28×28. The second layer consists of 100 convolution filters of size 10×10 and the output layer consists of 10 convolution filters of size 10×10.

Appendix

A MLP as a CNN

Applying an MLP with structure 784-100-100-10 for handwritten digit recognition problem, the inputs is a small image of size 28×28 gray scale pixels. To recognize the digit, the image is fed to the inputs. Then, the 100 neurons in the first layer calculate their neuronal outputs and pass the outputs to the next layer. Figure 30 shows the structure of the MLP.

A.1 Convolution in the First Layer

Essentially, these 100 neurons are 100 convolution filters. The receptive field of each filter is of size 28×28. Its parameters determine the property of its convolutionary property. In contrast to a convolution neural network (CNN) like LeNet5 and AlexNet, he receptive field of a convolution filter is of size $M \times M$, where M is 3 or typically 5. Besides, the number of convolution filters is much smaller than 100.

A.2 Convolution in the Second layer

The neurons in the second layer in the MLP can also be considered as performing 100 convolutionary filtering. Each neuron gets 10×10 inputs from the first layer and then calculates the convolutionary result for its output.

A.3 Convolution in the Output Layer

Finally, the neurons at the output layer again perform convolutionary filtering on the 10×10 outputs from the second layer.

A.4 MLP vs CNN

By that, one should realize that the difference between a MLP and a CNN is on the size of a convolution filter and the number of outputs to be generated in a layer. Consider the image size of a handwritten digit with 28×28, the total number of outputs in the first layer is 24×24 if the size of a convolution filter is 5×5.

B Limitation of a Simple ReLU Pereptron

To illustrate a limitation of using ReLU as a neuronal model, a two-input-one-output simple network is constructed and being trained to get a decision boundary for the dataset as shown in Figure 9.

B.1 Model and Learning

The simple two-input-one-output Perceptron with ReLU output node is defined as follows :

$$f(\mathbf{x}, \mathbf{w}) = \max\{0, w_1 x_1 + w_2 x_2 - b\}, \tag{79}$$

where $\mathbf{w} = (w_1, w_2, b)^T$ and $\mathbf{x} = (x_1, x_2, 1)^T$. The learning objective is defined as the mean square errors given by

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^{N} \left(d_k - f(\mathbf{x}_k, \mathbf{w})\right)^2. \tag{80}$$

The learning algorithm is defined in the sense of stochastic gradient descent.

$$\mathbf{w}(t+1) = \begin{cases} \mathbf{w}(t) + \mu_t (d_t - f(\mathbf{x}_t, \mathbf{w}(t)))\mathbf{x}_t & \text{if } \mathbf{w}^T(t)\mathbf{x}_t > 0, \\ \mathbf{w}(t) & \text{if } \mathbf{w}^T(t)\mathbf{x}_t \leq 0, \end{cases} \tag{81}$$

where $(\mathbf{x}_t, d_t)$ is a data randomly selected from the dataset at time t.

B.2 Settings

In the simulations, $\mu_t = 0.01$ for all t and the total number of learning steps is $10000 \times 200 = 2 \times 10^6$. The initial conditions are set to be (a) $w_1(0) = w_2(0) = b(0) = 1$ and (b) $w_1(0) = w_2(0) = b(0) = 0$. The results are shown in Figure 31.

B.3 Result Highlights

The results shown on the left column are based on the initial condition that $w_1(0) = w_2(0) = b(0) = 1$ and the results shown on the left column are based on the initial condition that $w_1(0) = w_2(0) = b(0) = 0$. In accordance with the results, a few points are noted and listed below.

1. For $w_1(0) = w_2(0) = b(0) = 0$, the model is unable to learn. So that, the results are not shown in the figure.

2. The decision boundary cuts on the dataset with label '0'. An explanation for this phenomena is presented shortly in the next subsection.

3. Our results reveal that the simple Perceptron with ReLU as output neuron is unable to get the decision boundary in the same way as the simpler Perceptron with either McCulloch-Pitts output neuron or sigmoidal output neuron.

A reason why the decision boundary cuts on the dataset with label '0' is presented in the next subsection.

B.4 1D Data Illustration

From the highlights in the last subsection, one question is why the decision boundary cuts on the dataset with label '0'. Owing to figure out the reason, we consider a simple classification problem with two sets of 1D data with the following labels.

$$d_k = \begin{cases} 0 & \text{if } x_k \in [-1, 0] \\ 1 & \text{if } x_k \in (0, 1], \end{cases} \tag{82}$$

for $k = 1, \cdots, N$. Moreover, the data distributed evenly in the interval $[-1, 1]$. That is to say, the probability density function for x is that $P(x) = 1/2$.

Assume that the simple Perceptron has one input and one output. The output node is defined as a ReLU. So, the mathematical model for this simpler Perceptron is given by

$$f(x, b) = \max\{0, x - b\}. \tag{83}$$

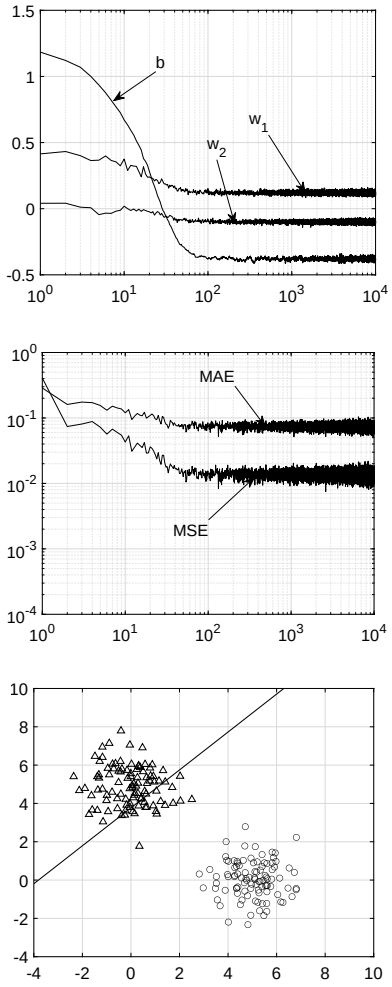
Note that we confine the model only has one parameter to be trained. With (83), the performance criteria is defined as the mean square error.

For $-1 \leq b \leq 0$ and $N \to \infty$, we get that

$$\begin{aligned} \mathcal{L}(b) &= (1/2)\int_b^0 (x-b)^2 dx + (1/2)\int_0^1 (x-b-1)^2 dx. \\ &= \frac{(b+1)^3 - 2b^3}{6} \end{aligned} \tag{84}$$

and its gradient is given by

$$\frac{\partial \mathcal{L}(b)}{\partial b} = \frac{\left((b+1)^2 - 2b^2\right)}{2}$$



$$w_1(0) = w_2(0) = b(0) = 1.$$

Figure 31: Changes of the parameters w_1, w_2 and b over time for the dataset as shown in Figure 9. Here, $\mu_t = 0.01$ and the total number of learning steps is $10000 \times 200 = 2 \times 10^6$. The initial conditions are set to be $w_1(0) = w_2(0) = b(0) = 1$. For the case that $w_1(0) = w_2(0) = b(0) = 0$, the model is unable to learn. The results shown in the middle panel are mean absolute errors (MAE) and mean square errors (MSE) over every 4000 steps. **Top:** Changes of parameters. **Middle:** Mean prediction errors. **Bottom:** Decision boundary. One should be noted that the horizontal axis in both the top and the middle panels is in log-scale.

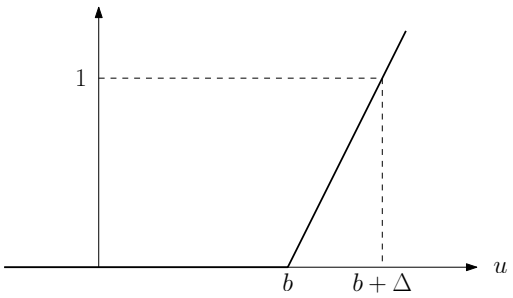


Figure 32: The transfer function of a ReLU with parameters b and Δ.

Clearly, $\partial\mathcal{L}(b)/\partial b$ is a decreasing function of b for $-1 \leq b \leq 0$. Thus, $\partial\mathcal{L}(b)/\partial b = 0$ if and only if $b = 1 - \sqrt{2} < 0$. At that, the MSE is given by $\mathcal{L}(1-\sqrt{2}) = (3 - 2\sqrt{2})/3$.

Here, one should be noted that the decision boundary specified by b is a negative value. It cuts the group of data with label '0'. By that, one can infer the reason why the decision boundaries as shown in Figure 31 cut the group of data with label '0', which is marked by triangle in the bottom panel.

C Network of ReLUs

To implement a network of ReLU neurons to perform a logical operation, at least three ReLU neurons are needed. Recall that the transfer function of a ReLU is stated in (56) and shown in Figure 24. Precisely, we can define the ReLU with parameters.

$$f(x_1, x_2, w_1, w_2, b, \Delta) = \max\left\{0, \frac{w_1 x_1 + w_2 x_2 - b}{\Delta}\right\}. \tag{85}$$

Let $u = w_1 x_1 + w_2 x_2$. The shape of $\max\{0, (u-b)/\Delta\}$ is shown in Figure 32. By that, one can get that

$$\begin{aligned} g(u, b, \Delta) &= \max\left\{0, \frac{u-b}{\Delta}\right\} - \max\left\{0, \frac{u-(b+\Delta)}{\Delta}\right\} \\ &= \max\left\{0, \max\left\{0, \frac{u-b}{\Delta}\right\} - \max\left\{0, \frac{u-(b+\Delta)}{\Delta}\right\}\right\}. \end{aligned} \tag{86}$$

C.1 Saturating Linear Neuron

$g(u, b, \Delta)$ is a saturating linear function given by

$$g(u, b, \Delta) = \begin{cases} 1 & \text{if } u \geq b + \Delta, \\ (u-b)/\Delta & \text{if } b < u < b + \Delta, \\ 0 & \text{if } u \leq b. \end{cases} \tag{87}$$

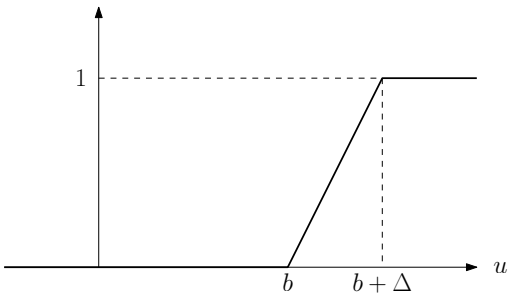


Figure 33: The shape of the saturating linear function $g(u, b, \Delta)$ as stated in (86) or (87).

Its shape is shown in Figure 33. Thus, we can get a saturating linear neuron by a network of three ReLU neurons. In this regard, realization of a logic gate can be implemented by a network of ReLU neurons.

C.2 Logical Operations Realization

With reference to the realization of logical operations based on networks of McCulloch-Pitts neurons as presented in Section 1.4, realization of logical operations based on networks of ReLU neurons can be done similarly.

C.2.1 AND: $w_1 = w_2 = 1$, $b = 1.5$, $\Delta = 0.01$

For $w_1 = w_2 = 1$, $b = 1.5$ and $\Delta = 0.01$, the neuronal model is given by

$$g(x_1, x_2, 1, 1, 1.5, 0.01) = \begin{cases} 1 & \text{if } x_1 + x_2 \geq 1.51, \\ 100(x_1 + x_2 - b) & \text{if } 1.5 < x_1 + x_2 < 1.51, \\ 0 & \text{if } x_1 + x_2 \leq 1.5. \end{cases} \quad (88)$$

As $x_1, x_2 \in \{0, 1\}$, $g(x_1, x_2, 1, 1, 1.5, 0.01) = 1$ if and only if $x_1 = x_2 = 1$. The neuron as defined by (88) performs logical AND. It acts as an AND gate.

C.2.2 OR: $w_1 = w_2 = 1$, $b = 0.5$, $\Delta = 0.01$

For $w_1 = w_2 = 1$, $b = 0.5$ and $\Delta = 0.01$, the neuronal model is given by

$$g(x_1, x_2, 1, 1, 0.5, 0.01) = \begin{cases} 1 & \text{if } x_1 + x_2 \geq 0.51, \\ 100(x_1 + x_2 - b) & \text{if } 0.5 < x_1 + x_2 < 0.51, \\ 0 & \text{if } x_1 + x_2 \leq 0.5. \end{cases} \quad (89)$$

As $x_1, x_2 \in \{0, 1\}$, $g(x_1, x_2, 1, 1, 0.5, 0.01) = 0$ if and only if $x_1 = x_2 = 0$. The neuron as defined by (89) performs logical OR. It acts as an OR gate.

C.2.3 NAND: $w_1 = w_2 = -1$, $b = -1.5$, $\Delta = 0.01$

For $w_1 = w_2 = -1$, $b = -1.5$ and $\Delta = 0.01$, the neuronal model is given by

$$\begin{aligned} & g(x_1, x_2, -1, -1, -1.5, 0.01) \\ = & \begin{cases} 1 & \text{if } -x_1 - x_2 \geq -1.49, \\ 100(-x_1 - x_2 - b) & \text{if } -1.5 < -x_1 - x_2 < -1.49, \\ 0 & \text{if } -x_1 - x_2 \leq -1.5. \end{cases} \\ = & \begin{cases} 1 & \text{if } x_1 + x_2 \leq 1.49, \\ 100(-x_1 - x_2 - b) & \text{if } 1.49 < x_1 + x_2 < 1.5, \\ 0 & \text{if } x_1 + x_2 \geq 1.5. \end{cases} \end{aligned} \tag{90}$$

As $x_1, x_2 \in \{0, 1\}$, $g(x_1, x_2, -1, -1, -1.5, 0.01) = 0$ if and only if $x_1 = x_2 = 1$. The neuron as defined by (90) performs logical NAND. It acts as an NAND gate.

C.2.4 NOR: $w_1 = w_2 = -1$, $b = -0.5$, $\Delta = 0.01$

For $w_1 = w_2 = -1$, $b = -0.5$ and $\Delta = 0.01$, the neuronal model is given by

$$\begin{aligned} & g(x_1, x_2, -1, -1, -0.5, 0.01) \\ = & \begin{cases} 1 & \text{if } -x_1 - x_2 \geq -0.49, \\ 100(-x_1 - x_2 - b) & \text{if } -0.5 < -x_1 - x_2 < -0.49, \\ 0 & \text{if } -x_1 - x_2 \leq -0.5. \end{cases} \\ = & \begin{cases} 1 & \text{if } x_1 + x_2 \leq 0.49, \\ 100(-x_1 - x_2 - b) & \text{if } 0.49 < x_1 + x_2 < 0.5, \\ 0 & \text{if } x_1 + x_2 \geq 0.5. \end{cases} \end{aligned} \tag{91}$$

As $x_1, x_2 \in \{0, 1\}$, $g(x_1, x_2, -1, -1, -0.5, 0.01) = 1$ if and only if $x_1 = x_2 = 0$. The neuron as defined by (91) performs logical NOR. It acts as an NOR gate.

C.2.5 XOR Operation

Recall from Figure 4 that an XOR can be implemented by an OR gate, an NAND and an AND gate. As presented above, each of these logic gate can be implemented by a network of three ReLU neurons. Thus, XOR can be implemented by nine ReLU neurons.

C.2.6 Digital Computer Implementation

Similar to that of using McCulloch-Pitts neurons, one can infer that a digital computer can also be implemented by entirely ReLU neurons.

C.3 Classification Problems

In principle, we can train a network of three ReLU neurons to solve a 2-class classification problem. By fixing the value of Δ to a small positive value, we

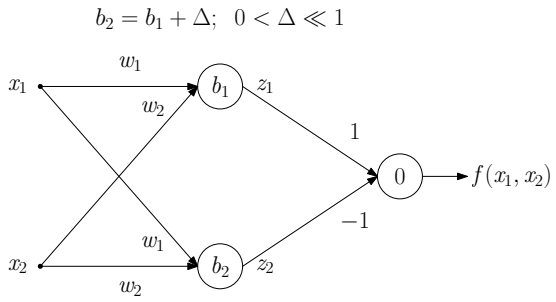


Figure 34: An application of a network of three ReLU neurons in 2-class classification problem. Here, Δ is a predefined small positive number, say $\Delta = 0.01$. The parameters w_1, w_2 and b_1 are shared among the neurons in the hidden layer. z_1 and z_2 are the output of the neurons in the hidden layer. The parameters of the output neuron are fixed. $f(x_1, x_2) = \max\{0, z_1 - z_2\}$, where $z_1 = g(x_1, x_2, w_1, w_2, b, \Delta)$ and $z_2 = g(x_1, x_2, w_1, w_2, b + \Delta, \Delta)$.

still need to find out nine model parameters. An efficient approach is to set the both hidden ReLU neurons with sharing parameters, as shown Figure 34. In this regard, the total number of model parameters is just three if Δ is pre-set to a small positive number. They are w_1, w_2 and b_1. The transfer function of the output neuron is a saturating linear function as follows :

$$f(\mathbf{x}, \mathbf{w}) = \max\{0, \underbrace{g(x_1, x_2, w_1, w_2, b, \Delta)}_{z_1} - \underbrace{g(x_1, x_2, w_1, w_2, b + \Delta, \Delta)}_{z_2}\}, \quad (92)$$

where $\mathbf{x} = (x_1, x_2)^T$, $\mathbf{w} = (w_1, w_2, b)^T$, $z_1 = g(x_1, x_2, w_1, w_2, b, \Delta)$ and $z_2 = g(x_1, x_2, w_1, w_2, b + \Delta, \Delta)$.

C.3.1 Learning Algorithm

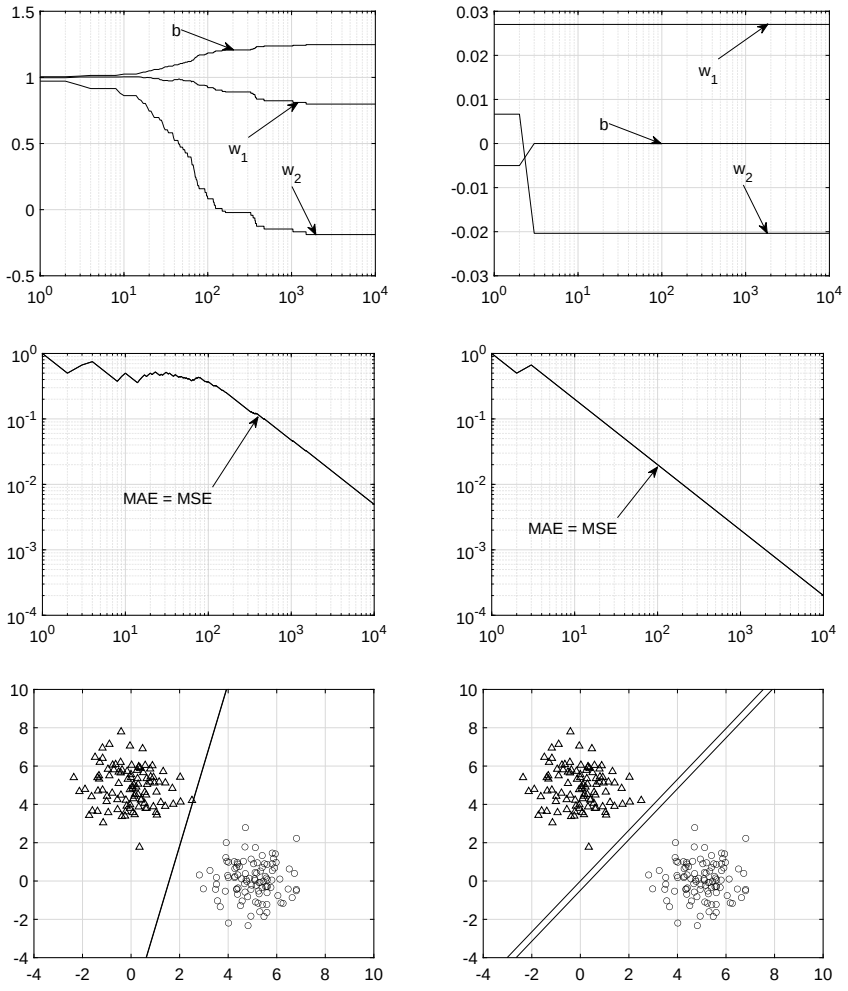
The learning algorithm for the parameters w_1, w_2 and b can thus be derived by gradient descent in similar way as for the MLP.

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \mu_t \left(d_t - f(\mathbf{x}_t, \mathbf{w}(t))\right) \begin{bmatrix} \mathbf{x}_t \\ 1 \end{bmatrix}, \quad (93)$$

where $(\mathbf{x}_t, d_t)$ is a sample randomly selected from the dataset at the t^{th} training step. One should be noted that the learning rule stated in (93) is similar to the online Perceptron learning rule as stated in (25).

C.3.2 Illustrative Examples

Figure 35 shows the results in which the learning rule (93) is applied with $\Delta = 0.01$ and $\mu_t = 0.005$ for all t. Two initial conditions are investigated : (a) $w_1(0) = w_2(0) = b(0) = 1$ and (b) $w_1(0) = w_2(0) = b(0) = 0$. The value



(a) $w_1(0) = w_2(0) = b(0) = 1$. (b) $w_1(0) = w_2(0) = b(0) = 0$.

Figure 35: Changes of the parameters w_1, w_2 and b over time for the dataset as shown in Figure 9. Here, $\Delta = 0.01$, $\mu_t = 0.005$ and the total number of learning steps is 10000. The initial conditions are set to be (a) $w_1(0) = w_2(0) = b(0) = 1$ and (b) $w_1(0) = w_2(0) = b(0) = 0$. The value of $MAE(t)$ is defined as $MAE(t) = t^{-1} \sum_{\tau=1}^{t} |d_\tau - f(\mathbf{x}_\tau, \mathbf{w}(\tau)|$. The value of $MSE(t)$ is defined as $MSE(t) = t^{-1} \sum_{\tau=1}^{t} (d_\tau - f(\mathbf{x}_\tau, \mathbf{w}(\tau))^2$. **Top:** Changes of parameters. **Middle:** Mean prediction errors. **Bottom:** Decision boundary. One should be noted that the horizontal axis in both the top and the middle panels is in log-scale.

of $MAE(t)$ is defined as $MAE(t) = t^{-1} \sum_{\tau=1}^{t} |d_\tau - f(\mathbf{x}_\tau, \mathbf{w}(\tau)|$. The value of $MSE(t)$ is defined as $MSE(t) = t^{-1} \sum_{\tau=1}^{t} (d_\tau - f(\mathbf{x}_\tau, \mathbf{w}(\tau))^2$.

C.4 Nonlinear Decision Boundary Problems

The model as stated in (92) can only solve the classification problems with a linear decision boundary. For a 2-class classification problem with nonlinear decision boundary, a network similar to that of MLP has to be designed. Instead of using sigmoid neurons as the neurons, the network of ReLU neurons as stated in (92) is applied. If the number of these neuronal networks is sufficient enough, we can get a multi-layer network for finding the nonlinear decision boundary.

D Performances versus Model Size

To investigate the performances of a computation model versus its model size, a number of simulations have been conducted for the MNIST hand-written character recognition problem.

D.1 MNIST : Written Digit Recognition Problem

The MNIST handwritten digit dataset was downloaded[11]. To convert the dataset in a form that the MATLAB is able to load, two helper functions are used[12]. MNIST dataset consists of ten classes of handwritten digit images, from 0 to 9. Each image is of size 28×28 pixels. Each pixel is encoded by its gray scale which is an integer ranging from 0 to 255. In the dataset, there are 60,000 training images and 10,000 testing images. A survey on the performance of various classifiers on MNIST can be found in [25].

For these experiments, the grey level of each pixel is normalized from the range $[0, 256]$ to the range of $[0, 1]$. The label of a digit is encoded as a 10 dimensional binary vector. For MLP, $\mathbf{y}_k \in \{0, 1\}^{10}$. For instance, the label of digit '0' is defined as follows :

$$0 \to 1000000000.\ 1 \to 0100000000.\ \cdots\ 8 \to 0000000010.\ 9 \to 0000000001.$$

Thus, we can design an MLP with 784 inputs and 10 outputs to solve this handwritten digit recognition problem.

D.2 Computation Model – MLP 784-N_1-N_2-10

The MLPs to be investigated consist of two hidden layers and one output layer. The number of inputs is 784 and the number of outputs is 10. If the number of neurons in the first hidden layer is N_1 and the number of neurons in the second hidden layer is N_2, the The total number of model parameters N_p is given by

$$N_p = 784 \times N_1 + N_1 \times N_2 + 10 \times N_2 + (N_1 + N_2 + 10)\,. \tag{94}$$

[11] From `http://yann.lecun.com/exdb/mnist/`.

[12] From `http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset`.

The transfer function of both the hidden nodes and the output nodes is defined as a sigmoid function.

D.3 Typical Results

It is recalled that the convergence rate and the performance of an MLP depend on the *learning rate* and the *number of iterations*.

D.3.1 Results

Figure 36 and Figure 37 show typical results for a training.

1. For the simulation results as shown in Figure 36, the number of neurons in the first layer is 200 and the number of neurons in the second layer is 75. The learning step size is set to be 0.1 and the total number of epoches is set to be 200.

2. For the simulation results as shown in Figure 37, the number of neurons in the first layer is 250 and the number of neurons in the second layer is 150. The learning step size is set to be 0.1 and the total number of epoches is set to be 100.

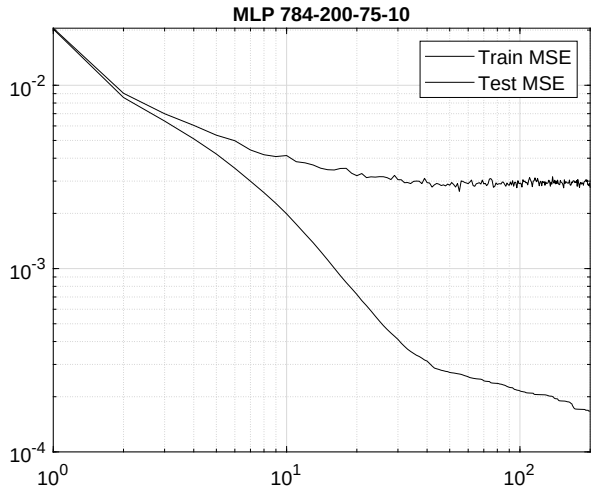
D.3.2 U-Shape on *Test MSE* and *Test ER*

While the model fits better for the training dataset and with decreasing values on the *Train MSE* and *Train ER*, the testing MSE and the testing error rate might not. Both *Test MSE* and *Test ER* show U-shape. The model with the best *Test MSE* and *Test ER* should be around the 50^{th} training epoch. This phenomena is called *overfitting*, i.e. the model fits too well to the training dataset. Thus, its fitting to the testing dataset is getting worse after some point of time. In this illustrative example, the model testing MSE and testing ER increase after around the 50^{th} training epoch.

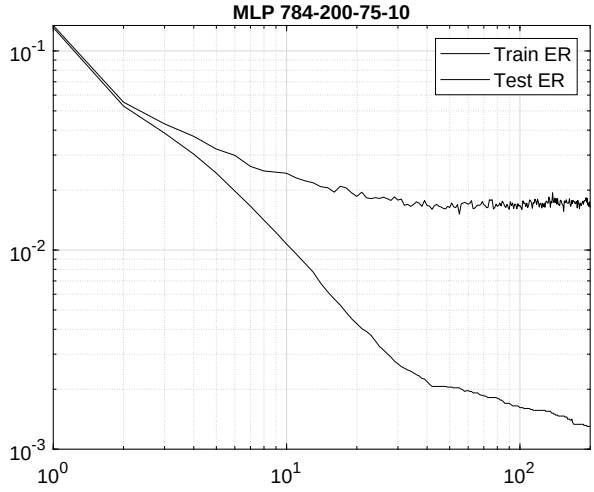
D.4 Versus Model Size N_p

To investigate the effect of model size and the performance, different MLP structures are trained and their performances are recorded. Fifteen structures are investigated. Their number of neurons in the first layer and the number of neurons in the second layer are depicted in Table 9. Figure 38 and Figure 40 show the performances of these two-hidden-layer MLPs in terms of their *training MSE* (Train MSE), *testing MSE* (Test MSE), *training error rate* (Train ER) and *testing error rate* (Test ER).

It is clear that the change of the training (resp. testing) MSE decreases in accordance with log-log linear relation. Similarly, the change of the training (resp. testing) error rate decreases in accordance with log-log linear relation. The decreasing rate with respect to the testing dataset is a way smaller than the decreasing rate for the training dataset.

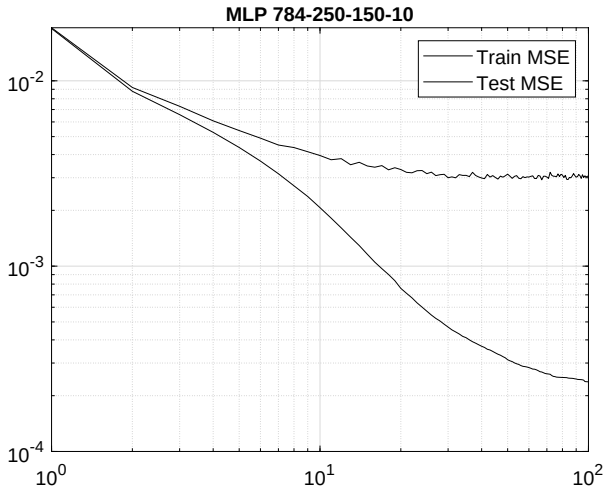


(a) MSE

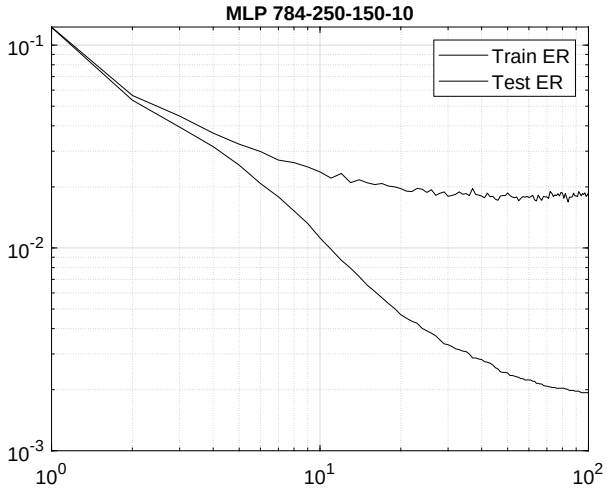


(b) Error Rate

Figure 36: Typical training results for an MLP for the MNIST written digit problem. In this example, the number of neurons in the first layer is 200 and the number of neurons in the second layer is 75. The learning step size is set to be 0.1 and the total number of epoches is set to be 200. One point should be noted. While the model fits better and decreases for the training dataset, the testing MSE and the testing error rate might not. Both *Test MSE* and *Test ER* show U-shape. The model with the best *Test MSE* and *Test ER* should be around the 50^{th} training epoch.



(a) MSE



(b) Error Rate

Figure 37: Typical training results for an MLP for the MNIST written digit problem. In this example, the number of neurons in the first layer is 250 and the number of neurons in the second layer is 150. The learning step size is set to be 0.1 and the total number of epoches is set to be 100. One point should be noted. While the model fits better and decreases for the training dataset, the testing MSE and the testing error rate might not. Both *Test MSE* and *Test ER* show U-shape. The model with the best *Test MSE* and *Test ER* should be around the 50^{th} training epoch. The *Train MSE* is around 0.0003 and *Tain ER* is 0.0022. The *Test MSE* is around 0.0030 and the *Test ER* is around 0.0177.

Table 9: Results obtained from fifteen structures of MLPs (784-N_1-N_2-100) being trained for the MNIST problem. Here, N_1 (resp. N_2) is the number of neurons at the first (resp. second) layer and N_p is the number of model parameters.

N_1	N_2	N_p	Train MSE	Test MSE	Train ER	Test ER
10	10	8070	0.0075	0.0118	0.0449	0.0715
20	20	16330	0.0037	0.0074	0.0215	0.0452
50	50	42310	0.0006	0.0046	0.0050	0.0267
100	100	89610	0.0004	0.0039	0.0032	0.0222
120	120	109930	0.0004	0.0037	0.0033	0.0208
150	150	141910	0.0004	0.0033	0.0031	0.0192
200	200	199210	0.0003	0.0033	0.0025	0.0186
50	200	51460	0.0009	0.0055	0.0073	0.0296
100	200	100710	0.0004	0.0038	0.0038	0.0210
150	200	149960	0.0004	0.0036	0.0034	0.0208
200	50	167560	0.0003	0.0032	0.0020	0.0192
200	75	172835	0.0003	0.0030	0.0019	0.0167
200	100	178110	0.0003	0.0031	0.0023	0.0180
200	150	188660	0.0003	0.0033	0.0022	0.0183
250	100	222360	0.0002	0.0030	0.0017	0.0177

From Figure 38, we can anticipate the size of a two-hidden-layer MLP whose *testing error rate* smaller or equal to 0.01 is 10^6. That is to say, the number of parameters is 10^6. By that, the two-hidden-layer MLP should be with 1000 neurons in each hidden layer.

D.5 Versus Model Structure

It is clear from Table 9 that the performance of a computation model cannot be simply determined by its size. It is also depended by its model structure. For instance, the two-hidden-layer MLPs with 200 neurons in the first layer outperform the two-hidden-layer MLPs with 50 neurons in the first layer. Here highlights three points from the results depicted in Table 9.

1. For the MLPs with $N_1 = N_2$, its performance *improves* as N_1 (equivalently, N_2) increases from 10 to 200.

2. For the MLPs with $N_2 = 200$, its performance *improves* as N_1 increases from 50 to 200.

3. For the MLPs with $N_1 = 200$, its performance *degrades* as N_2 increases from 75 to 200.

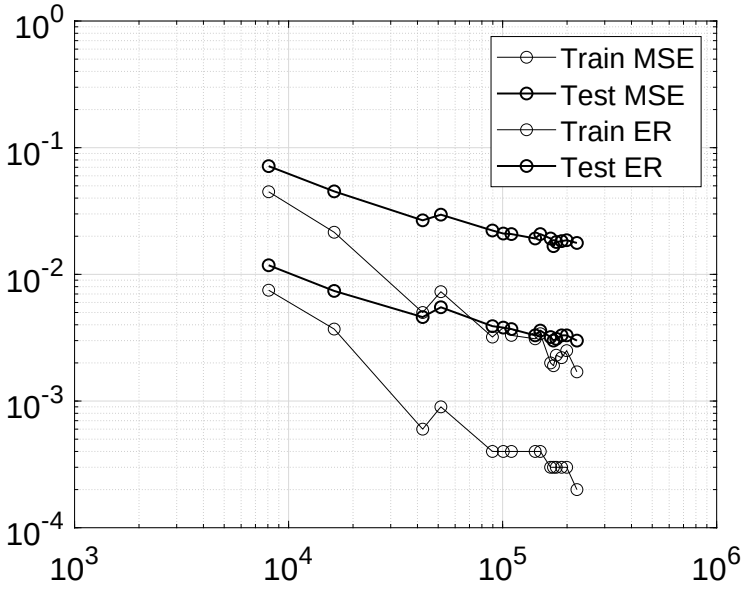


Figure 38: Performances of the two-hidden-layer MLPs versus their model sizes. It is clear that the change of the training (resp. testing) MSE decreases in accordance with log-log linear relation. Similarly, the change of the training (resp. testing) error rate decreases in accordance with log-log linear relation. The decreasing rate with respect to the testing dataset is a way smaller than the decreasing rate for the training dataset. The learning rate μ_t is set to be 0.05 and the total number of training epoches is 100. Each performance result shown in the diagram is averaged over the last 10 epoches.

Thus, one should note be confused by a simple argument that *the larger the model size the performance is better*. The performance of a computation model is largely determined by its model structure.

D.6 On Scaling Law

A common perception is that the performance of a computation model is getting better if the number of its model parameters is larger. The performance of one type of model, say a larger language model, is getting better if the size of the model increases.

D.6.1 Power Law

Empirically, it is common believed that the relation between the performance of a computation model follows a *log-log linear* (equi. *Power Law*) relation with certain factors.

$$\begin{aligned} \log(\text{Performance}(K)) &= \text{const}_0 + \text{const}_1 \log(K), \quad (95) \\ \text{Performance}(K) &= \exp(\text{const}_0)\, K^{\text{const}_1}, \quad (96) \end{aligned}$$

where K could be the number of data in the training dataset, the training time, the model size N_p and others.

D.6.2 On the Estimation of Population Mean

To estimate the population mean of a *normal distribution* with mean $\hat{p}$, an unbiased learning algorithm is given by

$$\hat{p}_t = \hat{p}_{t-1} - \frac{1}{t}\left(\hat{p}_{t-1} - p_t\right), \quad (97)$$

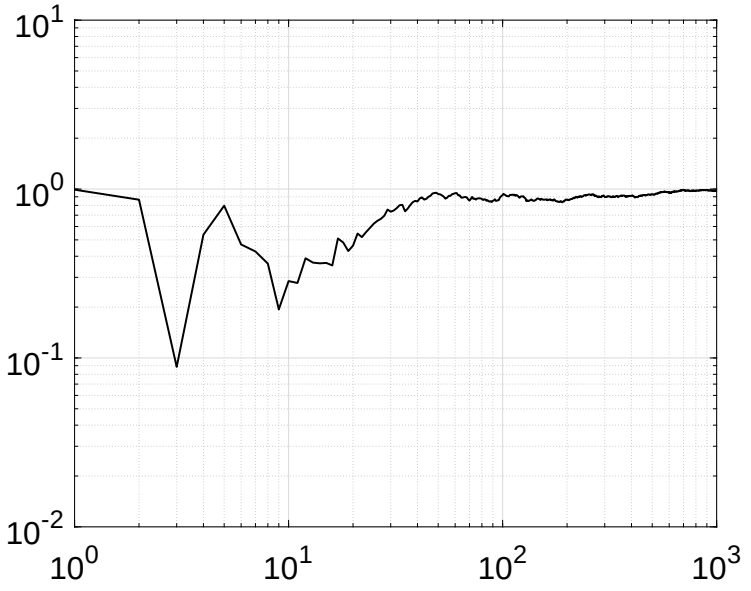
where t is the number of data input for training. Thus, t can be considered as the size of the training dataset.

To determine the *goodness* of the estimator $\hat{p}_k$, one criterion is defined as $E[(\hat{p}_t - \bar{p})^2]$. It is the deviation of the estimator $\hat{p}_t$ from the true population mean $\bar{p}$. Let $V(\hat{p}_t)$ be the *expected* variance $E[(\hat{p}_t - \bar{p})^2]$. One can get that

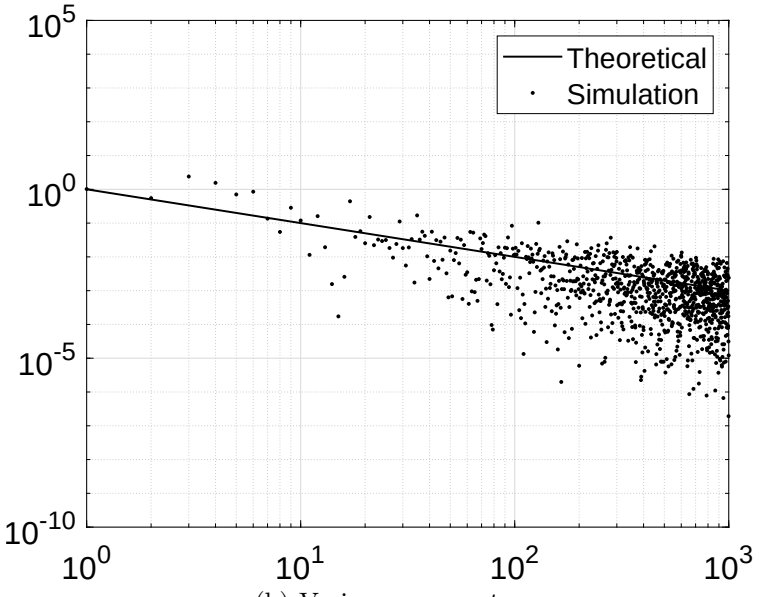
$$V(\hat{p}_t) = \frac{\bar{S}}{t^2} + \left(\frac{t-1}{t}\right)^2 V(\hat{p}_{t-1}). \quad (98)$$

D.6.3 On MLP for MNIST Problem

However the results depicted in Table 9 and the arguments presented in Section D.5 reveal that *scaling law* is not applicable to the application of MLP with structure 784-N_1-N_2-100 in the MNIST problem. Therefore, applying *scaling law* to predict the future investment on one specific type of computation models for solving a specific problem could be misleading and even meaningless.



(a) $\hat{p}_t$ versus t.



(b) Variance versus t.

Figure 39: The changes of $\hat{p}_t$ (a) and variance of $\hat{p}_t$ against t. Here, the true mean and variance of the normal distributed random variables are 1, i.e. $\hat{p} = 1$ and $\bar{S} = 1$. The dots shown in (b) are the variances obtained by $t^{-1}\sum_{k=1}^{t}(p_k - \hat{p}_k)^2)^2$. One should be noted that this variance estimator is a biased estimator.

D.7 Relations among Performance Criteria

Figure 40(a) shows the relations among different performance criteria. Basically, only the relation between *Train MSE* and *Test MSE* is known. Others are unknown.

D.7.1 *Test MSE* versus *Train MSE*

For the *Train MSE* and *Test MSE*, theoretical analysis has already shown their relation.

$$E[\text{Test MSE}] = E[\text{Train MSE}] + S_o \times F(\text{No. of Paramters}), \tag{99}$$

where $E[\text{Test MSE}]$ (resp. $E[\text{Train MSE}]$) is the expectation of the *Test MSE* (resp. *Train MSE*), S_o is the variance of the output noise and $F(\cdot)$ is a function of the number of model parameters. Usually, $F(\cdot)$ depends on the learning algorithm. Different learning algorithm might come up with different $F(\cdot)$ and hence different equation for (99).

D.7.2 Error Rate versus MSE

For the relation between MSE and error rate, it is still unknown. For some problems like the MNIST problem, we can determine their relation empirically. As shown in Figure 40(b,c), the error rate and MSE shows *log-log linear* relation[13] , i.e.

$$\log(\text{ER}) = \text{const}_0 + \text{const}_1 \times \log(\text{MSE}), \tag{100}$$

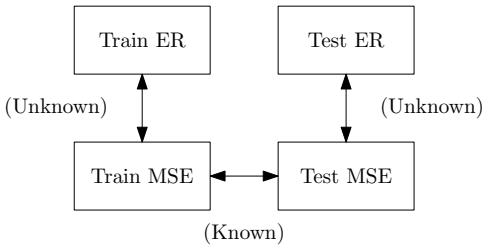
where const_0 and const_1 are constants to be determined from the plot in Figure 40(c).

D.7.3 Training Criterion vs Practical Performance

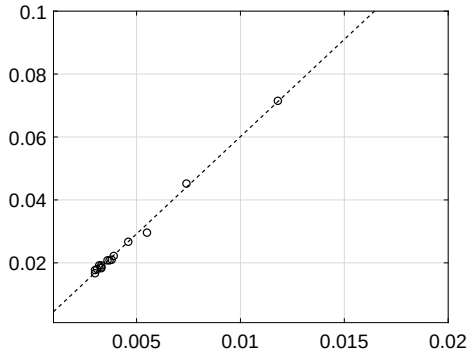
Note that the learning algorithm is developed based on *Train MSE* but not the others. Based upon these empirical findings, we conjecture that the *gradient descent learning algorithm* developed by using the *Train MSE*, as stated in (30), is applicable to search for a good MLP which has small *Test ER*.

Again, this conjecture might not be applicable to other type of models for solving other problems. In practice, development of a learning algorithm might even not based on the *Train MSE*, simply say $\mathcal{L}(\mathbf{w})$. A practical performance criterion could be the score of a LLM on math examination. The relation between training criterion $\mathcal{L}(\mathbf{w})$ and its score in a math examination is even hardly identified, both empirically and theoretically.

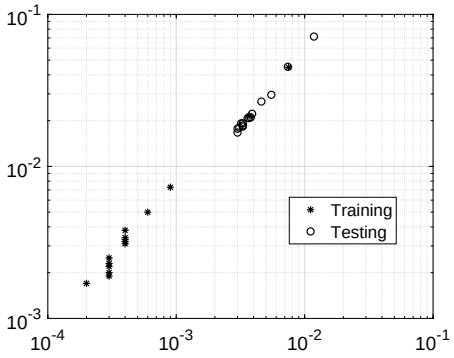
[13] It should be emphasized that the *log-log linear* relation between the *Test ER* and the *Test MSE* is similar to the *log-log linear* relation between the *Train ER* and the *Train MSE*. For other problems, their relations might be different.



(a) Relations among performance criteria.



(b) ER versus MSE (Testing).



(c) ER versus MSE (Training and testing).

Figure 40: Relations among performance criteria (a). Normally, the relation between MSE and error rate is unknown. Empirically, the error rate and MSE shows log-log linear relation as shown in (b) for the *Test ER* and *Test MSE* and (c) for the *Train ER* and *Train MSE*. Note that the learning algorithm is developed based on *Train MSE* but not the others. Based upon these empirical findings, we conjecture that the learning algorithm could search for a model which gives small *Test MSE* (resp. *Test ER*).