Computational (AI) Models

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

April 1, 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). Furthermore, recurrent networks are introduced and highlighted their potential applications in problems regarding sequence generations. Subsequently, aspects of deep neural networks are introduced and comparisons among a human brain, an electronic brain and a computer are delineated.

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

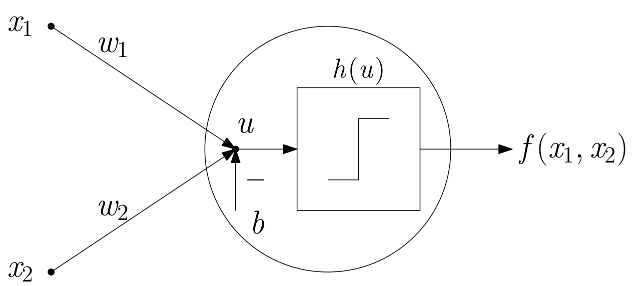
For a two-input-one-output neuron as shown in Figure 1, its mathematical model can simply be given by

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

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.

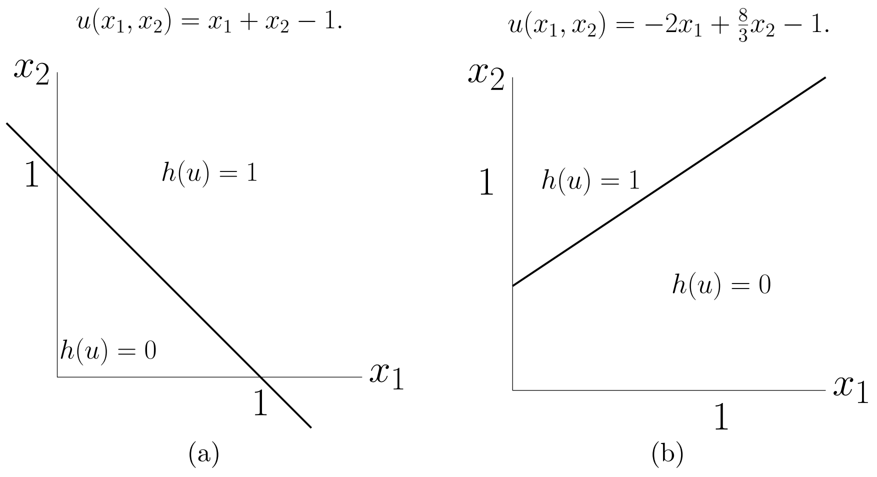
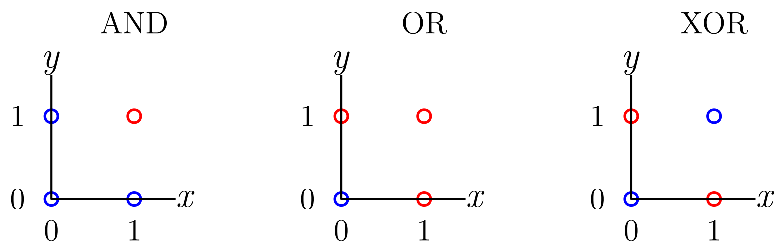


Figure 2: Two exemplar decision boundaries. (a) $u(x_1, x_2) = x_1 + x_2 - 1$. (b) $u(x_1, x_2) = -2x_1 + 8x_2/3 - 1$.



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

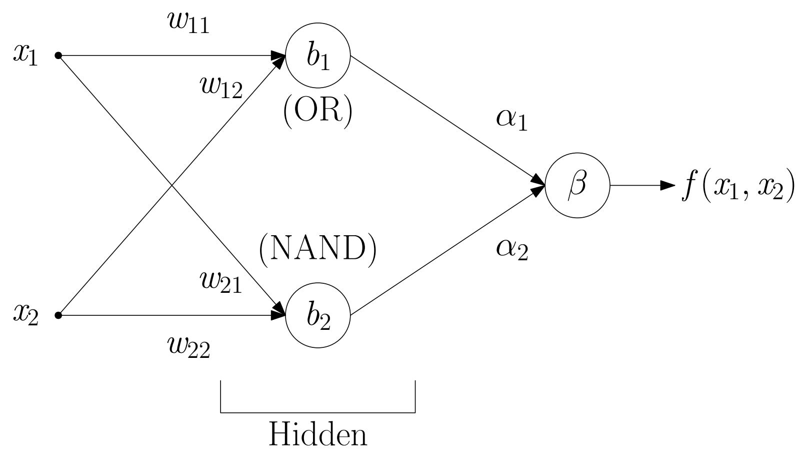


Figure 4: A network of three two-input McCulloch-Pitts neurons performs XOR operation. The neurons in the hidden layer are defined to perform logical OR and logical NAND. The parameters of the model are set to be $w_{11} = w_{12} = 1$, $b_1 = 0.5$, $w_{21} = w_{22} = -1$, $b_2 = -1.5$, $\alpha_1 = \alpha_2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

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 :

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

(equi. system).

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

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

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.

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 $\sum_{k=1}^{N} N_k (N_{k-1} + 1)$. 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.

1.7 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.*

x_1	x_2	x_3

$$x_1, x_2, x_3 \in \{-1, 0, 1\}. \quad o_1, o_2, o_3, o_4 \in \{0, 1\}.$$

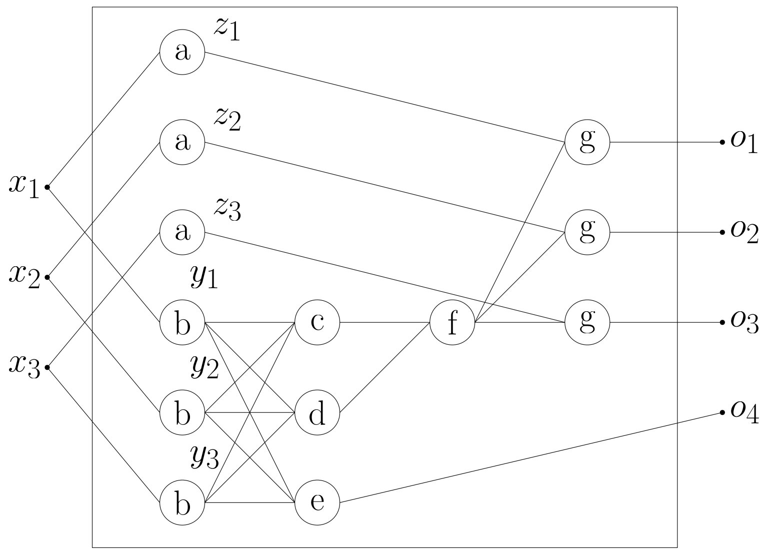
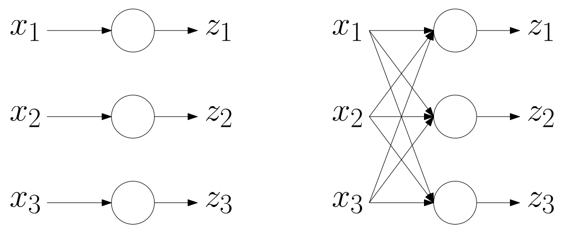


Figure 5: A network of McCulloch-Pitts Neurons could make decision for a step in a Tic-Tac-Toe game – Should a symbol be put on the cell corresponding to x_1, x_2 or x_3? In this network, there are seven different types of neurons. Some of them are single-input-single-output neurons (a-type and b-type). 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 an one-input-one-output neuron by three-input-one-output neuron. For the redundant weights, we simply set them to be zeros.

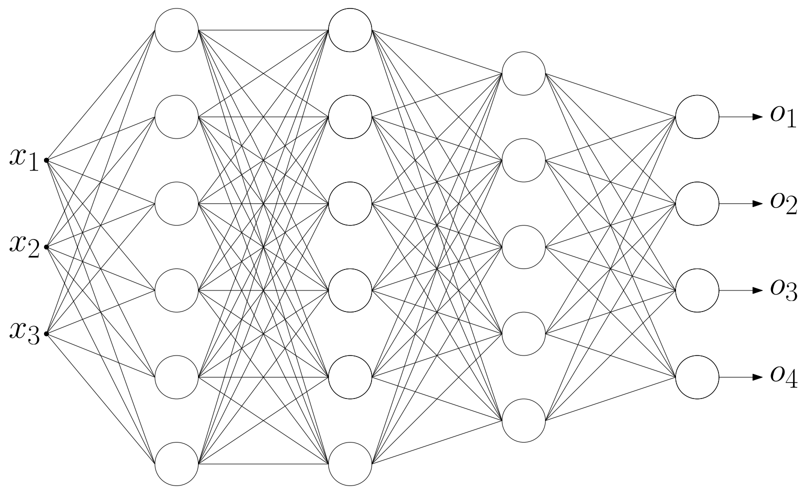


Figure 7: A 3-input-4-output multilayered network of all N-input-one-output neurons. Note that this network is also called a computational model. With proper design on the values for the weights and biases, this network is able to (but not limited to) replicate the functionalities of the network as shown 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
$(N_0 - \cdots - N_L)$	$\sum_{k=1}^{L} N_k$	$\sum_{k=1}^{L} N_k \left(N_{k-1} + 1\right)$

1.7.1 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 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

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.

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 $u(x_1, x_2)$.

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

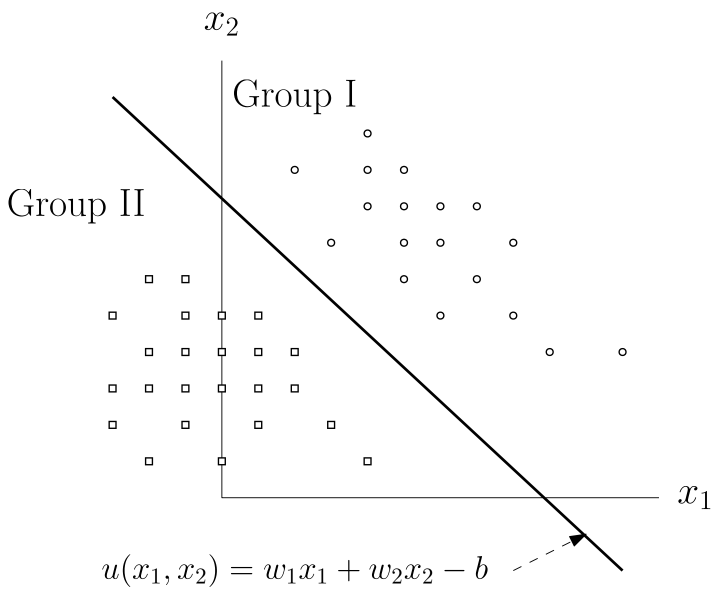


Figure 8: Use of a single 2-input-1-output McCulloch-Pitts neuron for data classification. The data in Group I is indicated by a circle and the data in Group II is indicated by a square.

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. 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.

[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.

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

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

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.

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.

$$
\begin{aligned}
w_1(t+1) &= w_1(t) + \mu \sum_{k=1}^{N} (d_k - f(\mathbf{x}_k)) x_{k1}, & (19)\\
w_2(t+1) &= w_2(t) + \mu \sum_{k=1}^{N} (d_k - f(\mathbf{x}_k)) x_{k2}, & (20)\\
b(t+1) &= b(t) - \mu \sum_{k=1}^{N} (d_k - f(\mathbf{x}_k)), & (21)
\end{aligned}
$$

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.

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

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

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

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

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

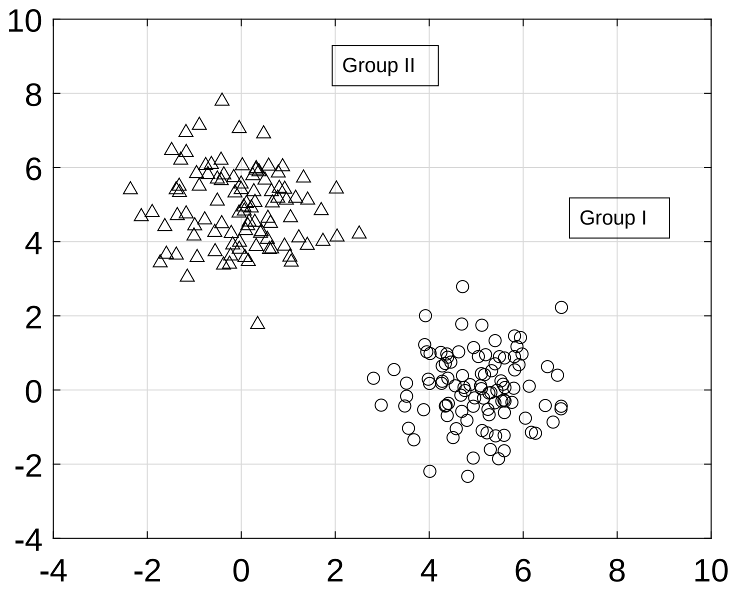


Figure 9: Two groups of data which are separable.

1.11 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.

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

Figure 10(**Top**) shows the changes of the parameters w_1, w_2 and b obtained by the online Perceptron learning rule over time $t = 1, \cdots, 2000$. Figure 10(**Middle**) 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 Figure 10(**Bottom**).

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 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.

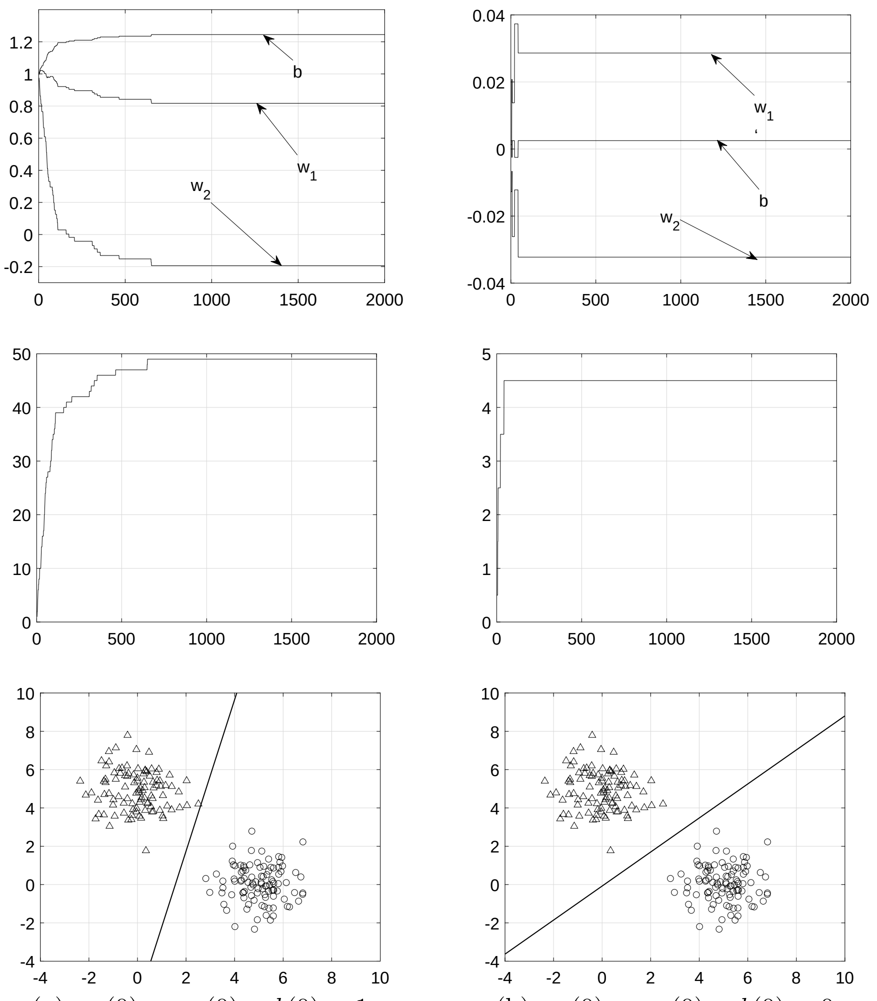
Figure 11 shows two examples. For either example, a good 1-input-1-output M-P neuron can be defined as follows :

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

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

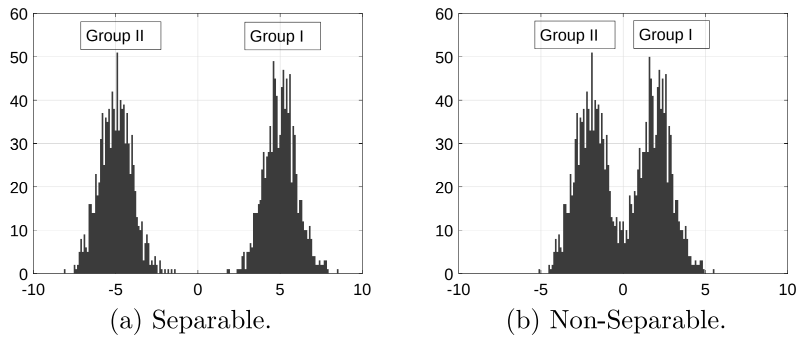
Applying the network of M-P neurons, it could be difficult to get a learning rule for a 3-class data classification problem. Figure 12 shows the distributions of the three classes of data and the Perceptron model which is able to solve this classification problem. The M-P neurons in the first layer perform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



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

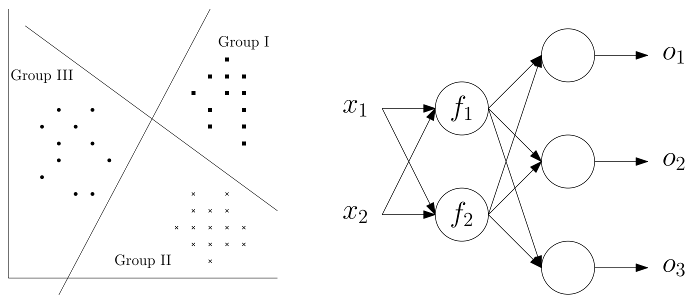
(b) $w_1(0) = w_2(0) = b(0) = 0$.

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 $(w_1(0), w_2(0), b(0)) = (1, 1, 1)$, the parameters converge to $(0.8019, -0.2029, 1.2500)$ after $t \geq 750$. (b) With the initial condition $(w_1(0), w_2(0), b(0)) = (0, 0, 0)$, the parameters converge to $(0.0286, -0.0322, 0.0025)$ 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.



(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.

$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	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.

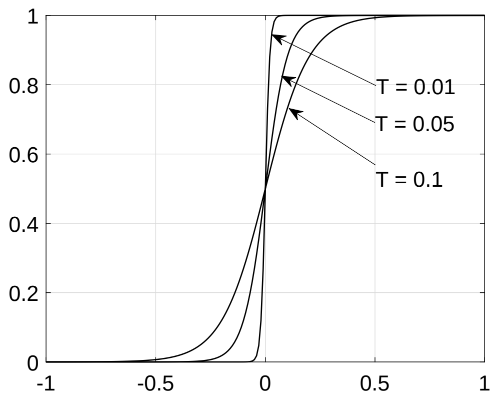


Figure 13: Sigmoid function could be considered as a relaxation of the step function in the McCulloch-Pitts model. The plots show the output versus the value of u. If $T \to 0$, the output is identical to a step function as in the McCulloch-Pitts model.

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 neuron is given by

$$\begin{aligned} f(x_1, x_2) &= \frac{1}{1 + \exp\left(-u(x_1, x_2)/T\right)} && (27) \\ &= \frac{1}{1 + \exp\left(-(w_1 x_1 + w_2 x_2 - b)/T\right)} && (28) \end{aligned}$$

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 $\mathbf{w}$. Thus, the online 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.

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

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

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.

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 :

$$w_1x_1 + w_2x_2 - b = 0. \tag{39}$$

By (39), we can get that

$$f(\mathbf{x}, \mathbf{w}) \begin{cases} > 1/2 & \text{if } w_1x_1 + w_2x_2 - b > 0, \\ = 1/2 & \text{if } w_1x_1 + w_2x_2 - b = 0, \\ < 1/2 & \text{if } w_1x_1 + w_2x_2 - b < 0. \end{cases} \tag{40}$$

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 Comments on Sigmoid MLP

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

2.5.1 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.

2.5.2 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

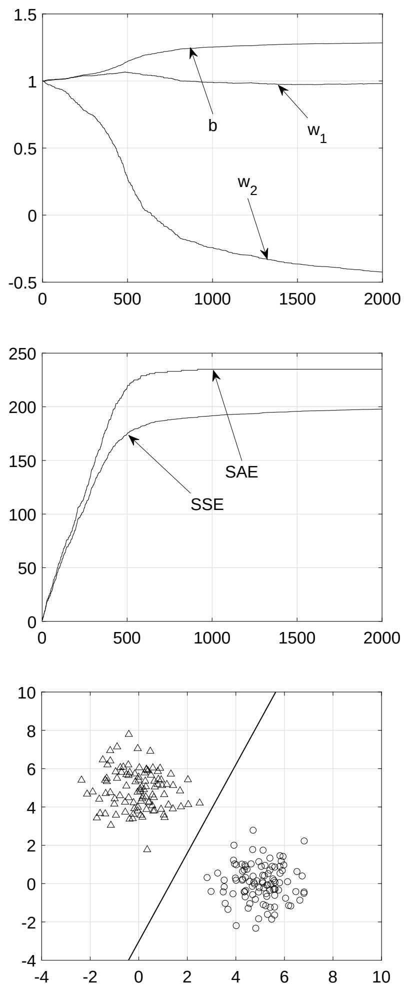
$$w_1x_1 + w_2x_2 - b = 0$$

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

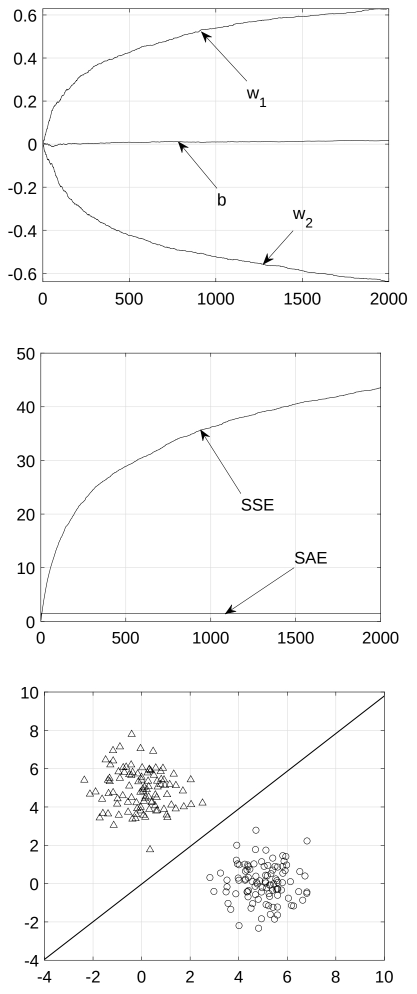
$$\kappa(w_1x_1 + w_2x_2 - b) = 0 \;\; (\kappa > 0)$$

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.



(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 $(w_1(0), w_2(0), b(0)) = (1, 1, 1)$. (b) With the initial condition $(w_1(0), w_2(0), b(0)) = (0, 0, 0)$. 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.

2.5.3 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 :

$$\begin{aligned} w_1(t+1) &= w_1(t) + \mu_t \left(e(t)f'(\mathbf{x}_t, \mathbf{w}(t))x_{t1} - \alpha\, w_1(t)\right) & (41)\\ w_2(t+1) &= w_2(t) + \mu_t \left(e(t)f'(\mathbf{x}_t, \mathbf{w}(t))x_{t2} - \alpha\, w_2(t)\right) & (42)\\ b(t+1) &= b(t) - \mu_t \left(e(t)f'(\mathbf{x}_t, \mathbf{w}(t)) - \alpha\, b(t)\right), & (43)\end{aligned}$$

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 15. In the figure, MAE and MSE stand for the mean absolute errors and mean square errors as defined as follows :

$$MAE = \frac{1}{T}\sum_{t=T+1}^{2T} |d_t - y(\mathbf{x}_t, \mathbf{w}(t))|, \quad (44)$$

$$MSE = \frac{1}{T}\sum_{t=T+1}^{2T} (d_t - f(\mathbf{x}_t, \mathbf{w}(t)))^2, \quad (45)$$

where

$$y(\mathbf{x}_t, \mathbf{w}(t)) = \begin{cases} 1 & \text{if } f(\mathbf{x}_t, \mathbf{w}(t)) > 0.5 \\ 0 & \text{if } f(\mathbf{x}_t, \mathbf{w}(t)) \le 0.5. \end{cases} \quad (46)$$

2.5.4 Performance Versus Number of Data

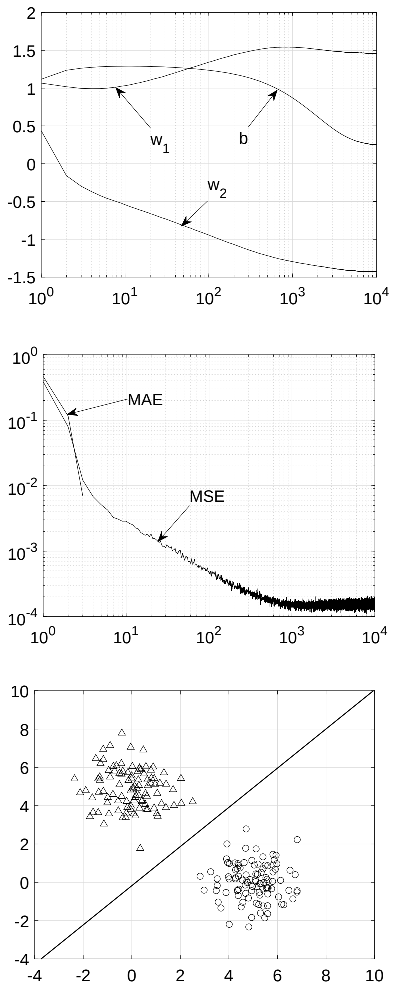
From the middle panels in Figure 15, one should observe that the prediction error of a model decreases as the number of training steps increases. As each update requires a data to be fed in, the number of training steps can be in analog to the number of data fed.

Thus, the middle panels in Figure 15 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*[5], i.e.

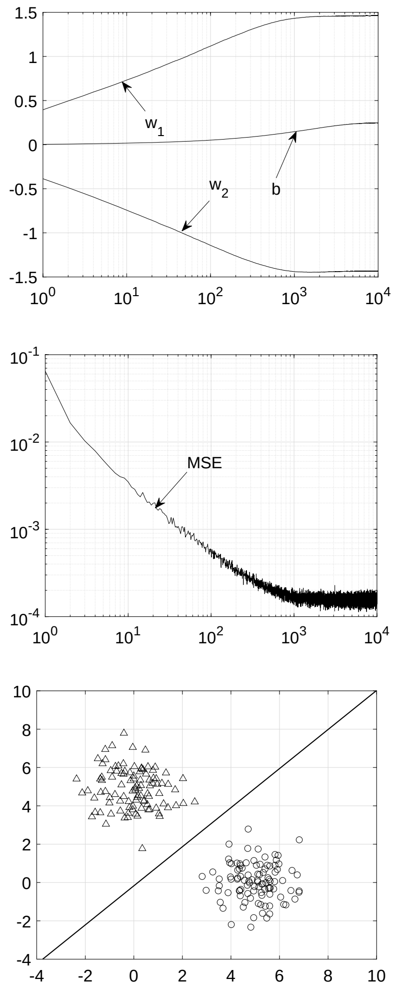
$$\log(\text{MSE}(N)) = \text{const.} - \gamma \log(N). \quad (47)$$

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 (47).

[5]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 *Mores 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.



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



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

Figure 15: 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.

2.5.5 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 six factors. They are

1. the network structure,
2. the objective function (respectively, assessment measure) to be minimized,
3. the number of iterations (resp. stopping criteria),
4. the learning step μ_t,
5. the weight decay (equivalently, forgetting) factor and
6. the initial condition of the parameters, i.e. $\mathbf{w}(0)$.

All these factors can only be obtained by trial-and-error.

3 Beyond Decision Making

The above presentations focus on the use of a multilayered network for solving a decision problem[6]. 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 16.

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 17.

[6]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.

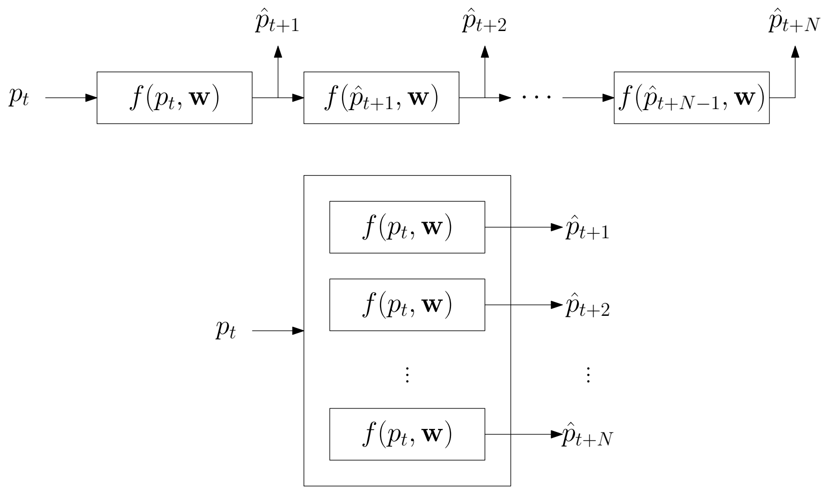
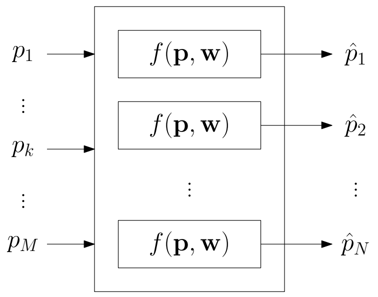


Figure 16: Computational model for a time sequence data prediction. p_t is the true value at time t. Based on this value, the computational model is applied to predict the values from $\hat{p}_{t+1}$ to $\hat{p}_{t+N}$ given the value of p_t. The prediction window width is fixed at N.



p_1
$\vdots$
p_k
$\vdots$
p_M
$f(\mathbf{p}, \mathbf{w})$
$f(\mathbf{p}, \mathbf{w})$
$\vdots$
$f(\mathbf{p}, \mathbf{w})$
$\hat{p}_1$
$\hat{p}_2$
$\vdots$
$\hat{p}_N$

Figure 17: 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.

3.3 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 $\hat{p}_{t+N}$.

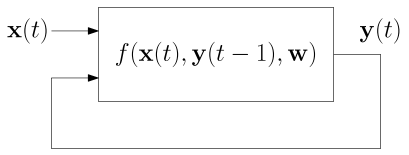
Accordingly, many types of sequence generation problems can be defined and depicted in Table 4.

3.4 Text Generation

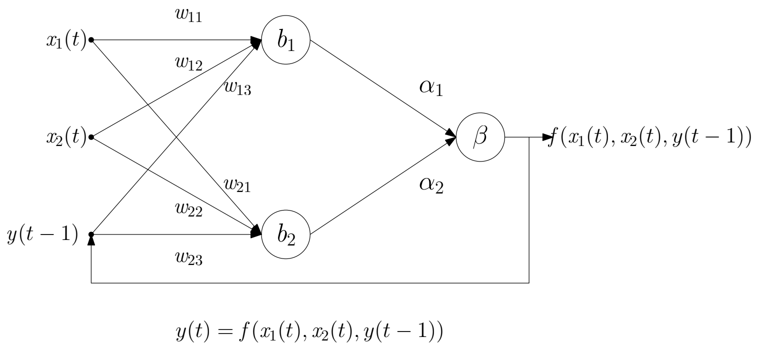
From Figure 17, one can note that a computational model $f(\cdot, \mathbf{w})$ could be designed (equivalently, trained) to generate a sequence of texts if $\mathbf{p}$ is a sequence of text-input (i.e. prompt) of M words. The outputs is a sequence of N words. The key idea is to design a computational model with recurrent connections, as shown in Figure 18.

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.



(a) Block diagram.



(b) Simple recurrent MLP.

Figure 18: 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 **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.

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.

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, 10, 11, 12]. 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* [13, 14] 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.

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

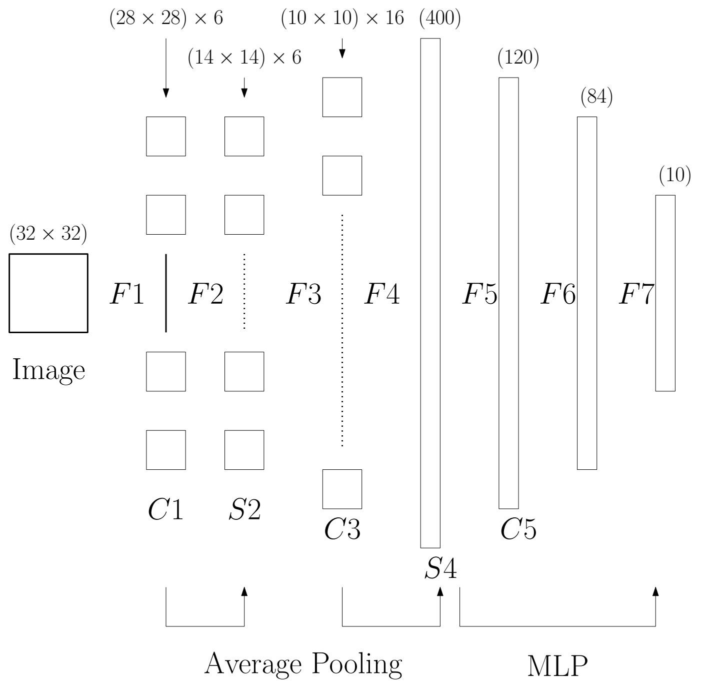


Figure 19: Model structure of the LeNet 5 in [1]. The F1, F3 and F5 layers perform convolution. So, these layers are called convolution layers as well. Note that only the neurons in the F5 and F6 layers are sigmoidal neurons. The neurons in the other layers are not.

system. This idea was first appeared in *Cognitron* and *Neocognitron* introduced in [15, 16] in the 1970s, in *LeNet*[7] introduced in [1] in the 1990s and later in *AlexNet* introduced in [11] in the 2010s.

4.4 Rectified Linear Neuron (ReLU

Third, a new neuron model called rectified linear neuron as shown in Figure 20 is employed. Rectified linear neuron was first investigated in [17, 18] and later applied in the neural network models for pattern recognitions [15, 16]. The mathematical model for a rectified linear neuron is given by

$$f(x) = \max\{0, x\}. \tag{48}$$

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

[7]Its structure is shown in Figure 19.

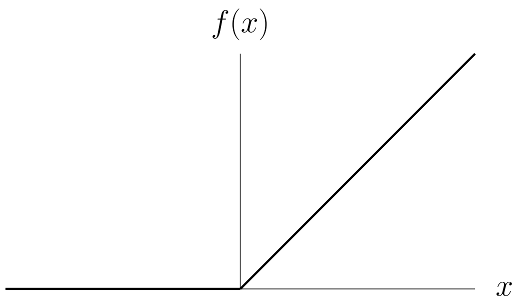


Figure 20: A rectified linear unit neuron. Its transfer function is defined as $f(x) = \max\{0, x\}$.

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 ReLU to Tackle Vanishing Gradient

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 21. The output is given by

$$\begin{aligned}
f(x_1, x_2) &= \phi(\alpha_1 z_1(x_1, x_2) + \alpha_2 z_2(x_1, x_2) - \beta) && (49)\\
z_1(x_1, x_2) &= \phi(w_{11}x_1 + w_{12}x_2 - b_1) && (50)\\
z_2(x_1, x_2) &= \phi(w_{21}x_1 + w_{22}x_2 - b_2), && (51)
\end{aligned}$$

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

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

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

$$\begin{aligned}
w_{ij}(t+1) &= w_{ij}(t) + \mu_t e(t) g_0(t) \alpha_i(t) \sum_{i=1}^{2} g_{1i}(t) x_{tj}\\
&= w_{ij}(t) + \mu_t \sum_{i=1}^{2} e(t) g_0(t) \alpha_i(t) g_{1i}(t) x_{tj} && (52)
\end{aligned}$$

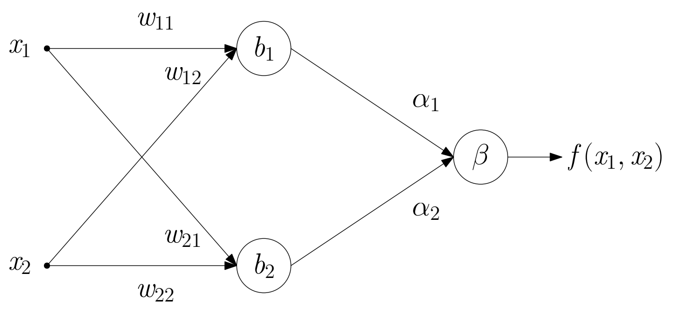


Figure 21: A MLP with three neurons.

where

$$\begin{aligned} e(t) &= (d_t - f(\mathbf{x}_t, \mathbf{w}(t))) \quad (53)\\ g_0(t) &= \phi(u_0(t))(1 - \phi(u_0(t))) \quad (54)\\ g_{1i}(t) &= \phi(u_i(t))(1 - \phi(u_i(t))) \quad (55) \end{aligned}$$

for $i = 1, 2$ and

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

Note that

$$0 \leq \phi(s)(1 - \phi(s)) \leq 1/4.$$

The value $g_0(t)$ and $g_{1i}(t)$ must be smaller than $1/4$. Hence, the factor $g_0(t)g_{11}(t)$ or $g_0(t)g_{12}(t)$ 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.

Take LeNet 5, Figure 19, 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.

4.6 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.

5 Computational (AI) Model Development

To summary, Figure 22 shows the key steps in a computational (AI) model development. It includes at least three important steps, namely the hypothetical

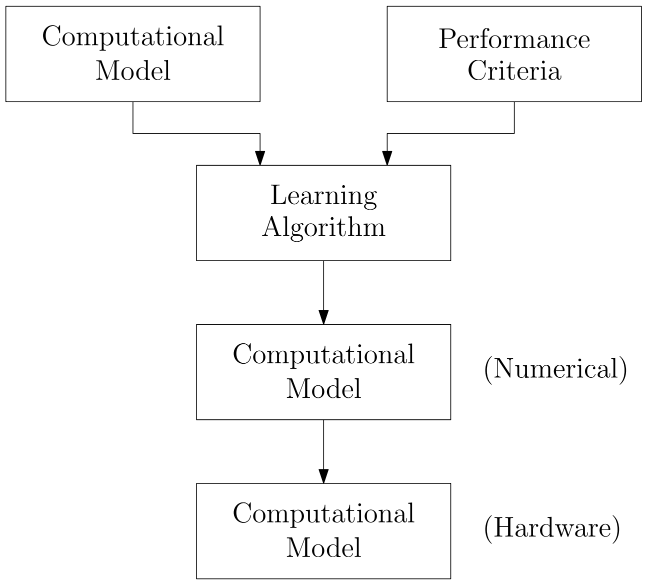


Figure 22: Process of a computational model development. Initially, a thought computational model is hypothesized. Together with the model, a performance criteria is defined for the problem. Depending on the nature of the model, the criteria could be the mean square errors (MSE) or maximum likelihood. Once the performance criteria has been defined, the learning algorithm for the model can be derived. Learning is clearly yet another computational task which can be conducted by a computer with or without GPUs. After the learning process is completed, a *numerical computational model* is obtained. If using a computer (resp. a cloud) is not efficient enough to use the model, a *hardware* computational model might be built.

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. 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.

5.2 Learning Algorithm

Once the performance criteria has been defined, the learning algorithm for the model can be derived. Learning is clearly yet another computational task which can be conducted by a computer with or without GPUs. If the learning task is computational intensive, special computing platform might be needed. In this regard, building a platform with many computers and GPUs might be needed.

5.3 Implementations

After the learning process is completed, a *numerical computational model* is obtained. If using a computer (resp. a cloud) is not efficient enough to use the model, a *hardware* computational model might be built for real-time applications. It is all dependent on the usage of the model being developed.

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 6.

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.

Table 6: Comparisons among human brain, electronic brain and 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	Yes	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.

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.

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.

[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] 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.

[11] 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.

[12] 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.

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

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

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

[16] ——, “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.

[17] 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.

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