Volume 1, Issue 1

Journal of Chemical Engineering and Energy Materials (JCEEM)

Journal homepage: https://www.jceem.com/

DOI: 10.22034/jceem.2025.543002.1013

Modeling a Chemical Reactor with Artificial Neural Networks

Ahmad Hassan Khalil
Department of chemical engineering, King Fahd University of Petroleum and Minerals, Saudi Arabia

Article info
Received: 23.02.2025
Accepted: 30.30.2025
Available Online: 30.03.2025
Checked for Plagiarism: Yes

Keywords:
Chemical reactor, dynamic differential equation, artificial neural networks, weight matrices, software model.

A B S T R A C T

Today, the use of artificial intelligence in modeling chemical reactors has many advantages. In this project, the inputs and outputs of a nonlinear multi-input and multi-output system have been used to create an intelligent dynamic model. Therefore, the selection of artificial neural networks of the multilayer perceptron type is suitable for this purpose. Along with this type of modeling, it is necessary to use an appropriate method for predictive control of the mentioned model. Correctable regression models that use the rules for adjusting the weight matrices of the communication paths between the model neurons have been used in this project. These rules are used to train the system to control and achieve the desired output at later times. Learning in this system is also of the supervised learning type. In this way, the dynamic differential equation of the system is available. Therefore, the desired values for the target variable that the system should achieve are specified for future times and the output of the system using a predictive controller should always achieve these goals. The system studied in this project is a chemical reactor that is used to continuously mix reactant chemicals with defined concentrations and amounts and produce a product substance with a time-varying concentration, where the desired amount of this concentration at a specific time is the desired goal that the system should achieve. Also, instead of a real system, a software model is used to collect input and output data and finally, the results of this successful modeling prove the ability of intelligent modeling methods as presented in this research. Correctable regression models that use the rules for adjusting the weight matrices of the communication paths between model neurons are used in this project.

Introduction

Modeling is used in various sciences to analyze and solve problems in a system. The use of modeling methods is especially important in chemical systems and processes. One of the most important applications of process models is the analysis of experimental data and its use to characterize the chemical process. The use of modeling improves the understanding of the process, predicts the behavior of the process under different conditions for control and design, and also optimizes the system. The most important modeling methods include the following:

1- Experimental modeling: By conducting numerous experiments, different variables are measured and a relationship between them is obtained by plotting the variables [1].

2- Mathematical modeling: Using the basic laws of the problem, the problem is formulated and the governing mathematical equations of the system variables are obtained.

Since experimental modeling requires conducting experiments under different conditions that may be difficult to achieve experimentally and are also not cost-effective, mathematical modeling is preferred in most cases. The modeling steps include the following steps:

- ✓ Identifying the process [2].
- ✓ Investigating parameters and variables and collecting necessary information.
- ✓ Determining the relationships between parameters and variables with the help of

Corresponding Author: **Ahmad Hassan khalil (ahmad.hassan.khalil@gmail.com)*

equations and governing laws in the form of mathematical equations.
- ✓ Applying analysis conditions and simplifying and solving mathematical equations with different methods and finally the model.
- ✓ Comparing the model results and experimental results to evaluate the model.

The importance of using neural networks

Achieving an accurate model for complex processes requires solving many and complex equations. Therefore, new and convenient methods should be used. One of these useful methods, in which there is no need to analyze and write long equations to identify the system, is neural networks. In this method, for modeling and identifying the system, it is only necessary to have sufficient experimental data from the system, which is used to train the network. If the network outputs approach this experimental data with a small error, it means that the network training has been completed, and in other words, this network represents the relevant system and can be used in various applications, including process control [3].

Methodology

Today, with the significant increase in the volume of chemical and biological data, the need to use computers and appropriate software to analyze them is increasing. MATLAB software, with its high capabilities in this field, has attracted the attention of chemical scientists. Understanding the capabilities of MATLAB software for evaluating, examining, and analyzing chemical and biological data is very important in this regard. With the help of MATLAB software, chemical processes such as distillation, reaction kinetics, and mass transfer can be modeled and simulated. MATLAB also provides users with tools for importing data and processing them from various sources, including process control systems, various experiments, and simulations. This software can design and implement chemical process control systems, and for example, the MATLAB optimization toolbox is used to optimize process variables and meet specific performance criteria. On the other hand, MATLAB can calculate and design processes such as determining heat and type of materials and estimate the cost of the process. MATLAB is also used in thermodynamic calculations, such as calculating equilibrium compositions and phase diagrams. Assessing the safety of the process and its environmental impacts, such as evaluating scenarios for the release of toxic substances and greenhouse gases, are other uses of MATLAB in chemical engineering. Automation of various processes and tasks in chemical engineering, including the task of data processing, simulation and analysis, is also done through MATLAB, which greatly improves efficiency and reduces the percentage of errors. Integration of various processes and components in chemical engineering, such as linking process simulation with control systems and other software tools, are also considered capabilities of MATLAB. The use of MATLAB in process safety and risk analysis is also very important. MATLAB software can be used for risk assessment and safety analysis of chemical processes. MATLAB can also suggest appropriate strategies to reduce potential risks. Chemists use MATLAB to simulate chemical processes, simulate chemical stability, model microchemistry, calculate chemical functions, analyze experimental data, and generally conduct advanced chemical research. Chemical process simulations can also be performed with MATLAB, including stable and unstable chemical processes, polar and magnetic potentials, surface chemistry, stable and unstable microchemistry, and polar and magnetic potential microchemistry. MATLAB can also be used to analyze chemical data using signal and image processing algorithms. On the other hand, MATLAB can be used to solve problems related to the thermodynamic transfer of heat, mass, or fluids. MATLAB also helps in reactor design and process control. How to code and solve chemical engineering problems with MATLAB is a knowledge that all students of this field need. They must first have a proper understanding of these problems and how to solve them, and then be able to use the software to solve them [4].

Results

1- Modeling a Single Neuron: First, we will examine the characteristics of a single neuron and how to model it. The main role of a biological neuron is to sum its inputs until the sum of the inputs exceeds a certain threshold, and thus produce an output. The inputs to the neuron enter through dendrites, which are connected to the outputs of other neurons by connection points (synapses). Synapses change the effectiveness of the signals received. The cell body receives all the inputs and fires a signal when the sum of the inputs exceeds the threshold. In short:

- ✓ The output of a neuron is either active (one) or inactive (zero).
- ✓ Output depends only on the inputs. The amount of inputs must be sufficient to activate the neuron's output.

The efficiency of synapses in transmitting incoming signals to the cell body can be modeled using a coefficient that is multiplied by the neuron's inputs. Stronger synapses, which transmit more signals, have much larger coefficients, while weaker synapses have smaller coefficients.

This model first calculates the weighted sum of its inputs and then compares it with its internal threshold level and if it exceeds it, it is activated. Otherwise, it remains inactive. Since the inputs pass through the neuron to produce the output, we call this system "feedback". We need to show this operation mathematically. If the number of inputs is n, then each input line has its own weight coefficient. The modeled neuron calculates its inputs. First, it multiplies the first input by the weight coefficient of the line connecting that input. Then it repeats the same operation for the second input and the other inputs and finally sums up all the resulting values. In short:

Input 1 × Weight for line 1 = Sum of inputs
Input 2 × Weight for line 2+
Input n × Weight for line n+

$$\sum_{i=1}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + \ldots + w_n x_n$$

The above sum must be compared with the threshold value of the neuron in question. Compared to the threshold, if the sum obtained exceeds the threshold value, then the output of the neuron will be equal to one, and if the sum is less than the threshold, the output will be equal to zero.

Threshold Function

On the other hand, the threshold value can be first subtracted from the weighted sum obtained and then the resulting value is compared with zero. If the resulting value is positive, the output of the neuron will be one, otherwise, it will be zero. It should be noted that the entire threshold function has not changed, but in this case a step jump has taken place in the region. The threshold actually adds a skew to the weighted sum. Another way to achieve this would be to completely remove the threshold from the neuron body and instead add an input with a constant value of 1. This input would always remain active and its connection line weight would be considered equal to the negative threshold value. In this case, instead of subtracting the threshold value from the sum of the neuron inputs, we could multiply the additional constant input +1 by its weight, which is equal to the negative threshold value, and add it to the other inputs, which we call the bias calculation. In this way, we call the value the bias or one-sidedness. Both approaches are similar and acceptable [5].

If we call the output y, the following relationship expresses the first approach.

$$y = f_h\left[\sum_{i=1}^{n} w_i x_i - \theta\right]$$

While fh is a step function, this function is actually called the "Hoyside" function.

$$f_h(x) = 1 \text{ اگر } \quad x > 0$$

$$f_h(x) = 0 \text{ اگر } \quad x \le 0$$

In this way, our purpose is fulfilled. Note that the output of the function is only the values 1 and 0. In other words, the neuron is either active or inactive. If we use the second solution, namely calculating the bias, we choose another input with the number 0 and always set its value to 1. In this case, the weight coefficient of the new input will be equal to the bias value. Therefore, the above function will become as follows:

$$y = f_h\left[\sum_{i=0}^{n} w_i x_i\right]$$

Note that the lower limit of the sigma sign has changed from 1 to 0, and the value of 0x will always be +1. Their model was suggested in much the same way as discussed, through research into the behavior of brain neurons. It is important to discuss the specifics of this model further. Their model is a very simple device that compares the weighted sum of its inputs to determine an output with a threshold. The model does not take into account the complex structure and timing of the activity of real neurons, and it has none of the complex features of biological neurons. That is why we call it a model, not a replica of a biological neuron. Now we need to see how this simple model can be used. The way neurons communicate with each other is important, but to understand what is happening in the complex real world, it is better to first consider only one layer of neurons, so that we can study the outputs of the neurons under certain inputs.

The model neurons, which are connected in a simple way, were named perceptron by Frank Rosenblatt in 1962. He was the first to simulate the model neurons on digital computers and to formally analyze them. In his book Principles of Neural Dynamics, Rosenblatt described perceptrons as simplified networks in which some features of real neural systems were exaggerated and others were ignored. He admitted that the model was by no means an exact copy of neural systems and that he was dealing with only a basic model. The goal here is to discover the properties of models that learn their behavior from much simpler forms of natural neural systems, which are usually built on a smaller scale [6].

Learning in Simple Neurons

We need a way to learn in our neural models. Connecting these neurons together may create networks that can do something, but to do something useful we need to be able to train them in some way. As mentioned earlier, what makes these models useful is their ability to learn. Also, to make the models easy to understand, the learning methods should be as simple as possible. As is common in most neural computations, our source for simulations will be real neural systems. Children are often encouraged to get good math results and scolded for crossing the street without paying

attention to their surroundings. Dogs are given food treats for obeying commands. In general, good behavior is rewarded and bad behavior is punished. The same principle can be applied to artificial networks. We should reward desirable behaviors and discourage undesirable behaviors. If we have two sets of objects, say a set of different written shapes A and a set of different written shapes B, we might want our neuron to distinguish A Form B. We might want our neuron to output a 1 when it sees A and a 0 when it sees B. We need to think about our neuron model and examine its behavior to see how we can incorporate the concept of learning into our simple design. The guiding principle is to let the neuron learn from its mistakes. If the answer is incorrect, we want to reduce the probability of this error in the future, and if the answer is correct, we do not change the situation [7].

If we initially randomly determine the weight coefficients of the neuron's communication lines, that is, in fact, it is the starting state and the neuron does not know anything and has not yet been trained, then we can input a letter A into the neuron. The neuron calculates the weighted sum of its inputs and compares it with a threshold value. If the calculated value is greater than the threshold, the neuron will give an answer of 1, otherwise the output will be zero. The probability that the answer is correct by chance is 50%. Because the inputs to the neuron can only exceed the threshold value by chance. Suppose, the neuron gives a correct answer. In this case, no action is required [8].

Because the model was successful, but if the answer was zero, we need to increase the weighted sum, so that the next time it encounters the letter A, it gives the correct answer, 1. We do this by increasing the weight coefficients of the neuron's connection lines. So, to encourage the probability of getting an answer of 1, we increase the weights. For the letter B, we want the neuron to produce the number zero. That is, we want the weighted sum of the inputs to be less than the threshold value. Therefore, whenever the neuron encounters the letter B, we want to reduce its weight coefficients so that it will produce the number zero when it sees the letter B in the future. This means that to train the network, we need to increase the weight coefficients when the neuron is active and decrease the coefficients when the neuron is inactive. This is achieved by adding the value of the inputs to the corresponding coefficients when the neuron is active and subtracting the value of the inputs from the coefficients when the neuron is inactive. This will be our learning rule. It is worth noting that only the inputs that are active at that time are affected, and this is obvious. Because the passive inputs have no effect on the sum of the weights, and changing them has no effect in that particular case. This learning rule is another form of the rule proposed in 1949 by Donald Hebb and known as Hebb's learning rule. Hebb based his rule on the exclusive strengthening of active communication lines on his studies of real neural systems. In the slightly modified form of Hebb's rule that we use, the exclusive changing property of active communication lines is preserved, but these lines are both strengthened and weakened. We can do this because we know in advance the desired results and therefore see in which direction we should change the weight coefficients. Because this learning is guided by having the desired result in hand, we call this type of training "supervised learning" or "supervised learning." This is the dominant method for training pervasive models today. This learning went unquestioned until 1951. In that year, Marvin Minsky and Dean Edmundszky built a “neural network.” The massive device used 300 light bulbs, a large number of motors and clutches, and a gyro pilot from a World War II bomber to turn 40 control knobs. The arrangement of these knobs represented the machine’s memory. Minsky and Edmunds spent a long time watching the machine in action, watching how it adjusted the knobs and moved so many parts at once. The tangled system of connections was full of weak solder joints and misalignments, but the random nature of the system allowed it to continue working even when some of the light bulbs (neurons) failed. This mechanical invention was probably the first realization of neural networks [9].

Perceptron Learning Algorithm

The learning method described in this section can be represented as the following algorithm. This algorithm can be coded to build perceptron networks on computers with any programming language.

Perceptron Learning Algorithm

1- Determine the initial coefficients and threshold values.

Let $w_i(t), (0 \le i \le n)$ be the weight of input i at time t and θ be the output threshold value. Let w_0 be equal to $-\theta$ and x_0 always equal to 1.

Let $w_i(0)$ be a small random number. Then, initialize all weights and thresholds.

2- Provide the desired input and output.

Provide the inputs $x_n, \ldots x_2, x_1, x_0$ and the desired output d(t) to the model.

3- Calculate the actual output.

$$y(t) = f_h\left[\sum_{i=0}^{n} w_i(t)x_i(t)\right]$$

4- Change the weight coefficients.

$w_i(t+1) = w_i(t)$ if the output was correct.

$w_i(t+1) = w_i(t) + x_i(t)$ if the actual output was zero and the desired output was 1 (Class A).

$w_i(t+1) = w_i(t) - x_i(t)$ if the actual output was 1 and the desired output was zero (Class B).

Note that if the model answer is correct, the weights do not change. Also, the weight coefficients of those lines that do not contribute to the wrong answer do not change. Because their coefficient values are added to the input value of the lines, which is zero. Therefore, they remain unchanged.

This is the basic perceptron algorithm, but several modifications have been proposed to this basic algorithm. The first modification is to introduce a multiplier smaller than one in the formula for changing the weight coefficients. This slows down the rate of change of the weight coefficients and thus the network gets closer to the solution in shorter steps. This modification changes the fifth step of the algorithm as follows:

5- Adjust the weight coefficients (modified form).

$w_i(t+1) = w_i(t)$ if the output was correct.

$w_i(t+1) = w_i(t) + \eta x_i(t)$ if the actual output was zero and the desired output was 1 (class A).

$w_i(t+1) = w_i(t) - \eta x_i(t)$ if the actual output was 1 and the desired output was zero (class B).

Where $(0 \le \eta \le 1)$ is η, a positive feedback factor that controls the speed of adjustment. A similar algorithm was proposed by Widrow and Hough. They found that it is better to adjust the weights more when the difference between the actual output and the desired output is large and to adjust them less when the difference is small. They proposed a rule for learning called the Widrow-Hough delta rule. This rule calculates the difference between the sum of the weights and the desired output and considers it as an error. The adjustment of the weights is then done in proportion to this error [10].

The error value F can be written as:

$$\Delta = d(t) - y(t)$$

Where d(t) is the desired output of the system and y(t) is the actual output. This formula itself controls the matter of adding or subtracting the weight coefficients. Because if the desired output is 1 and the actual output is zero, $\Delta = +1$. Therefore, the weight coefficients are increased. Conversely, if the desired output is 0 and the actual output is +1, $\Delta = -1$ becomes and the weight coefficients are reduced. It should be noted that if the decision is correct, the weights do not change. Because $d(t) - y(t) = 0$

The final learning algorithm is basically similar to the initial perceptron learning algorithm. Only the fourth step of the perceptron algorithm changes as follows:

Adjust the weight coefficients (Widrow-Hoff delta rule)

$$\Delta = d(t) - y(t)$$

$$w_i(t+1) = w_i(t) + \eta \Delta x_i(t)$$

$$d(t) = \begin{cases} +1 \\ 0 \end{cases}$$

Where A is a positive feedback factor that controls the rate of modulation.

Widrow called the neurons that used this algorithm adlines (adaptive linear neurons). He also connected a large number of adlines together and called the structure a modaline.

Another approach that has been proposed is to use bipolar digits +1 and -1 instead of binary +1 and zero. Using binary digits means that lines with zero inputs are not trained. While using bipolar digits gives all lines a chance to be trained. This simple idea increases the speed of reaching the answer, but it often causes confusion in the writing. Because some writers use binary digits and some use bipolar digits. In fact, both methods are the same and using one or the other depends on personal taste.

Learning and adjusting weights in Adeline

When a neural network like Adeline is in the training or learning phase, three factors must be considered:

- ✓ The inputs used are selected from a training set, such that the desired response of the system to these inputs is known.
- ✓ The actual response generated is compared to an input pattern with the desired response and the existing error (the deviation of the actual response from the desired response) is calculated.
- ✓ The weights are adjusted in such a way that the error is reduced.

As mentioned earlier, this type of learning is called supervised learning. Because the outputs are known in advance and the network is guided to produce correct outputs. Unlike this type of learning, in unsupervised learning, the outputs are not specified in advance and the network is allowed to be in appropriate states. Unsupervised learning is not our focus in this project. According to what was said about supervised learning, the requirements for this type of learning are the existence of a suitable rule for adjusting the weights and a suitable error function, which was previously discussed under the name of Hebb's learning rule. The concept of this theory can be summarized in the form of a rule as follows: if an output is active and at the same time the input corresponding to one of the weights is also active, then increase the weight corresponding to this input. Numerically, this rule can be interpreted in several ways. One way is to assume that the inputs (x) and outputs (y) are binary numbers 0 and 1. A weight (w) is increased only when the output is 1 and the input corresponding to this weight is also 1. If the weights are initially set to 0, their values are

calculated as follows after using a set of training patterns as input:

$$w_i = \sum_{p=1}^{P} x_{ip} y_p$$

Where P is the number of patterns presented in the training phase. The above summation expression can also be written in matrix form, in which the weight matrix is obtained by multiplying the matrix of input patterns and the output matrix corresponding to these patterns [11].

$$[W]=[X]^t[Y]$$

In the formula $[X]^t$ is the transpose of the matrix [X]. For example, consider an Adeline where four different combinations of inputs and outputs of the AND function (y=x1^x2) are used as the training set. This AND function, as you can see, is an AND function with two inputs. The statement value table for this example is as follows. In this table, all combinations of inputs and their corresponding output values are shown.

P	X_1	X_2	Y
۱	۰	۰	۰
۲	۰	۱	۰
۳	۱	۰	۰
۴	۱	۱	۱

Table 1- Proposition value table for AND function with inputs 0 and 1

If the network is trained by presenting input and output pairs in the order given in the above value table from top to bottom, then the first input pattern used is P=1 as 00, which has a corresponding output of 0. Thus, the arrays of matrices [X] and [Y] are as follows:

$$[X]=\begin{bmatrix}0&0\\0&1\\1&0\\1&1\end{bmatrix}\quad [X]^t=\begin{bmatrix}0&0&1&1\\0&1&0&1\end{bmatrix}\ and\ [Y]=\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

$$[W]=\begin{bmatrix}0&0&1&1\\0&1&0&1\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\end{bmatrix}=\begin{bmatrix}1\\1\end{bmatrix}$$

Therefore, the weight values are w1=1 and w2=1. If the training set is presented to the network again with these new weight values, the corresponding weighted sum (i.e. net) will be as shown in Table 2 below, where $net=\sum_{i=1}^{2} W_i X_i$ is:

X_2	X_1	net	Y
۰	۰	۰	۰
۰	۱	۱	۰
۱	۰	۱	۰
۱	۱	۲	۱

Table 2- Input values and calculated value for net and output

The value obtained for net is converted by a nonlinear output function into a value for the output y. These nonlinear functions can be one of the types of sign functions (sign function), step, logarithmic or Boolean logic operations such as X_1ANDX_2, X_1OR X_2 or NOT X_1 which are represented as $\neg X_۱, X_۱ \vee X_۲, X_۱ \wedge X_۲$ respectively. This type of nonlinear function is called a hard limiter.

In the following, other examples that have logical functions with two inputs, like the previous example, are used to introduce different learning algorithms. Of course, such simple examples cannot represent the types of problems that are usually

solved by neural networks, and the main reason for this is that in these examples the training set is completely determined and deterministic. In this case, all possible input patterns are presented to the network and the network is only expected to respond correctly to these inputs. This means that the network does not need to generalize its knowledge to correctly classify completely new inputs. Using binary values of 0 and 1 does not always give good results in using this mechanism to update the weights. This can be shown by the function $Y = X_1 \wedge \neg X_2$. The propositional value table for this function is as follows:

P	X_1	X_2	Y
۱	۰	۰	۰
۲	۰	۱	۰
۳	۱	۰	۱
۴	۱	۱	۰

Table 3- Table of the value of the proposition related to the function $Y = X_1 \wedge \neg X_2$

As before, the weights are obtained by multiplying the following two matrices:

$$[W] = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

When the training set is presented to the network again, the weighted sum and outputs (assuming a bias of -0.5) are as follows:

X_1	X_2	net	Y
۰	۰	۰	۰
۰	۱	۰	۰
۱	۰	۱	۱
۱	۱	۱	۱

Table 4- Statement Value Table for Function Including -0.5 for Bias

The reason for this error is that the learning rule described earlier only allows the values of the weights to increase. A more general approach to the previous rule is to allow the values to decrease. One way to do this is to decrease the value of a weight when its corresponding input is active and its output is inactive, or vice versa [12]. This can be done by using +1 and -1 instead of 0 and 1 (as in Adaline) and calculating the weights as the product of the matrix of input patterns and the matrix corresponding to the output responses. For example, the statement value table for function $Y = X_1 \wedge \neg X_2$ looks like this:

P	X_1	X_2	Y
۱	-۱	-۱	-۱
۲	-۱	+۱	-۱
۳	+۱	-۱	+۱
۴	+۱	+۱	-۱

Table 5- Proposition value table for function $Y = X_1 \wedge \neg X_2$ using bipolar digits

To obtain the weights using Hebbian learning, the following procedure is used:

$$[W]=\begin{bmatrix}-1 & -1 & +1 & +1\\ -1 & +1 & -1 & +1\end{bmatrix}\begin{bmatrix}-1\\ -1\\ +1\\ -1\end{bmatrix}=\begin{bmatrix}+2\\ -2\end{bmatrix}$$

By reusing the input pattern with the latest set of weights, the output values are obtained as follows:

X_1	X_2	net	Y
-۱	-۱	۰	۰
-۱	+۱	-۴	۰
+۱	-۱	۴	۱
+۱	+۱	۰	۰

Table 6- Statement value table for output inputs and net value and bias -0.5

Correct outputs are obtained by an appropriate value for the compensation value and a hard limiter. Such a form of the Hebb learning rule is not always successful in training the network. It is successful only when the inputs are orthogonal or linearly independent. This is because the Hebb learning rule does not consider the actual output values and only works with the desired output values. If the weights are adjusted to values that depend on the amount of error between the actual and desired outputs, the aforementioned limitation will be overcome. The stated error is called delta and the new learning rule is called the Delta-Widow-Hoff rule, which was examined in the previous section [13]. In the following, we will reexamine this rule for bipolar digits.

Changing the Perceptron Model

How to solve the problem of the inability to solve linearly inseparable problems?

At first, it is thought that the solution to the problem might be to use more than one perceptron, so that each perceptron separates a small part of the space and from their sum linearly inseparable classes are successfully separated. This approach to the XOR elimination problem is shown in Figure 1. At first glance, it does not seem like a problem, but with some care we find that this combination of perceptron's is never able to learn. Each neuron in this combination, as before, calculates the weighted sum of its inputs, compares it with a threshold, and returns an answer of one or zero [14].

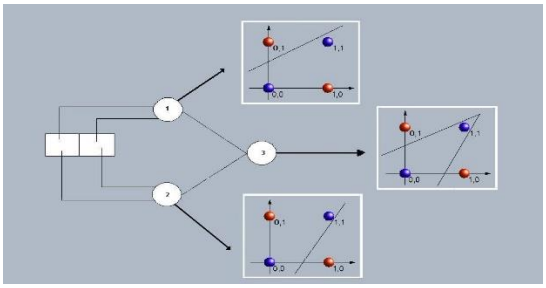

Figure 1. Combining perceptrons can solve the XOR problem

Perceptron 1 recognizes the pattern (1, 0), perceptron 2 recognizes the pattern (0, 1), and by combining these two perceptron's, perceptron 3 can correctly distinguish the input pattern, but this perceptron's must be pre-trained. They can never learn this way of classification themselves.

In the neurons of the first layer, the inputs will be the original inputs of the network, but the inputs of the second layer are the outputs of the first layer. This means that the perceptron's of the second layer do not know which of the original inputs are active and which are off. They are only aware of their inputs, which are actually the outputs of the first layer. Since learning means strengthening the connections between active inputs and active neuron units, it is impossible to strengthen the correct parts of the network. Because the real inputs are actually hidden by the middle layer of output units. The dual active and inactive state of the neurons does not give any indication of the amount necessary to adjust the

weight coefficients, and therefore it is not possible to adjust them [15].

The weight inputs that activate the neuron slightly should not change as much as the inputs that activate the neuron completely, but we have no information about their state. In other words, the threshold staircase function eliminates the information necessary for learning the network. This problem is called the "quota assignment" problem. Because the network cannot distinguish which of the input weights to increase and which to decrease, it cannot determine the necessary changes to improve the answer in subsequent rounds.

The solution to this problem, which arises due to the step nature of the function and the thresholding process, is to change this function and use its nonlinear form with a slight difference. If we smooth it somewhat, so that it is on or off almost as before, but slopes in the middle, to provide information about its input, we will be able to determine the amount of strengthening or weakening of the corresponding weight coefficients. In this way, the network will acquire the ability to learn as needed. In both cases, if the sum of the weighted inputs greatly exceeds the threshold, the output value will be practically one. Conversely, if it is much less than the threshold value, the output will be zero, but where the weighted sum and the threshold are approximately equal, the neuron's output will have a size between the two extremes. This means that the neuron's output can be related to its inputs in a more useful and informative way. Thus, we modified our model to solve a specific problem according to the type of problem. Our problem was a stepped threshold function that hid the inputs from the outputs. However, we have retained most of the features of the previous model. For example, each neuron calculates the weighted sum of the inputs as before and compares it with the threshold, but the output is no longer just one and zero, but is located on a spectrum, although the new threshold function behaves in many ways almost like a stepped function, especially at the ends of the spectrum. Now it is necessary to use a nonlinear threshold function. Because if the perceptron layers all use linear functions, their power will not exceed that of a single-layer perceptron model. The reason is that each perceptron layer performs only linear operations on its inputs. Linear operations can eventually be combined into a single operation. For example, scaling is a linear operation because it affects all components equally. If a network scales the inputs by a factor of 5 in the first layer and by a factor of 2 in the second layer, this is exactly the same as if a single layer scales the inputs by a factor of 10.

New Model

Newer perceptron's are organized in layers. They are naturally called multilayer perceptron's. The new model has three types of layers: the input layer, the outer layer, and the layer between them that is not directly connected to the input data and the output results. In fact, this layer is called the hidden layer. Each unit in the hidden layer and the output layer acts like a perceptron. The units in the input layer are simply responsible for distributing the input values to the next layer and therefore do not perform any calculations. By changing the nonlinear function from a step to a sigmoid and adding hidden layers, we inevitably have to change the model learning rule. Our new model must be able to recognize more complex patterns. We will examine the new rule in more detail below [16].

EP is the value of the error function of the pattern p. tpj is the desired output of pattern p at node j and opj is the actual output at that node. wij is the weighting coefficient of the connection line from node i to node j. Suppose that the error function is proportional to the square of the difference between the actual output and the desired output for all patterns fed to the network.

$$E_p = \frac{1}{2}\sum_j \left(t_{pj} - o_{pj}\right)^2$$

Using the $\frac{1}{2}$ coefficient simplifies the calculation somewhat and makes this function similar to other error functions. The incentive value per unit, or in other words, the net input j for pattern p, can be written as:

$$net_{pj} = \sum_i w_{ij} o_{pi}$$

In other words, the amount of excitation of each unit is similar to a single-layer perceptron equal to the weighted sum of its inputs. The output of each unit j is equal to the amount of the threshold function fj acting on the weighted sum of its inputs. This function was a step function in the case of the perceptron, while in the case of the multilayer perceptron it is a sigmoid function.

$$o_{pj} = f_j\left(net_{pj}\right)$$

Using the chain rule of differentiation, we can write the following relationship:

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ij}}$$

$$\frac{\partial net_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k w_{kj} o_{pk} = \sum_k \frac{\partial wik}{\partial wij} o_{pk} =$$

Because in all cases $\frac{\partial w_{ik}}{\partial w_{ij}}$ is equal to zero, except when k = i, in which case it is equal to one. The amount of error change can be defined as a function of the net input per unit as follows:

$$-\frac{\partial E_p}{\partial net_{pj}} = \delta_{pj}$$

Therefore, the above relationship becomes as follows:

$$-\frac{\partial E_p}{\partial w_{ij}} = \delta_{pj} o_{pi}$$

As a result, reducing the error value Ep will mean changing the weight coefficients proportional to $\delta_{pj} o_{pi}$. In other words:

$$\Delta_p w_{ij} = \eta \delta_{pj} o_{pi}$$

What we need to know now is the value of A per unit. If we know this, we can reduce E. Using the above relationship and the chain rule, we can write:

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial net_{pj}}$$

$$\frac{\partial o_{pj}}{\partial net_{pj}} = f'_j\left(net_{pj}\right)$$

Now consider the first term in the above equation. Using the first equation, we can find the derivative of Ep with respect to opj.

$$\frac{\partial E_p}{\partial o_{pj}} = -\left(t_{pj} - o_{pj}\right)$$

Therefore:

$$\delta_{pj} = f'_j\left(net_{pj}\right)\left(t_{pj} - o_{pj}\right)$$

The above expression is useful for the outer layer units. Because both the desired outputs and the actual outputs are known, but it is not suitable for the hidden layer units [17]. Because the value of their desired outputs is not available. If the unit j is one of the hidden layer units, we proceed using the derivative chain rule as follows:

$$\frac{\partial E_p}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial}{\partial o_{pj}} \sum_i w_{ik} o_{pi}$$

Using equations 2 and 3 and considering that, like equation 5, the partial derivative of the second expression is equal to one in only one case, the following equation is obtained:

$$\frac{\partial E_p}{\partial o_{pj}} = -\sum_k \delta_{pk} w_{jk}$$

Finally, by substituting equation 14 into equation 9, the following equation is obtained:

$$\delta_{pj} = f'_j\left(net_{pi}\right)\sum_k \delta_{pj} w_{jk}$$

The above equation shows the amount of change in the error function with respect to the weight coefficients in the network and provides a method for reducing the error function. The value of the function is proportional to the error values of the subsequent units δ. Therefore, the error values must first be calculated in the units of the outer layer and then propagated to the previous layers of the network to change the weight coefficients of the previous units. In fact, propagating errors to the previous layers is the reason for using the term "backpropagation" in networks. Equations 12 and 15 show how multilayer networks are trained. One of the advantages of using the sigmoid function as a nonlinear function is its similarity to the step function. Therefore, it should have similar behavior. The sigmoid function is defined as follows.

$$f\left(net\right) = \frac{1}{1 + e^{-knet}}$$

If we take its derivative with respect to (net), we get the following result:

$$f'\left(net\right) = \frac{ke^{-knet}}{\left(1 + e^{-knet}\right)^2}$$
$$= k\ f\left(net\right)\left(1 - f\left(net\right)\right) = k\ o_{pj}\left(1 - o_{pj}\right)$$

Therefore, the derivative of the function is simply a simple function of the output.

Multilayer Perceptron Algorithm

The multilayer perceptron algorithm that uses the back-propagation learning rule is described below. This algorithm requires nonlinear functions that are continuously differentiable. In other words, the functions must be smooth. We have chosen to use the sigmoid function A because of its simplicity of derivation [18].

Multilayer Perceptron's

A multilayer perceptron structure consists of a number of neurons and biases that are connected by connecting lines. A single neuron consists of a sum operator and an activity function. All the inputs to a neuron, defined as numerical values, are summed together after being multiplied by their respective connection weights and fed into a function called the activity function, thus creating the neuron's outputs. Neurons are connected by connections, each of which has a connection weight or value. These connection weights are also defined as numerical values, like the inputs and outputs of a neuron. The output produced by each neuron is used as input to the next neuron, in the order mentioned. The biases entering each neuron are also constant numbers, whose input value is usually considered to be one, and their connection weights determine the threshold value of the activity of the neuron

associated with them. Figures 1 and 2 show a single neuron and a multilayer perceptron, respectively.

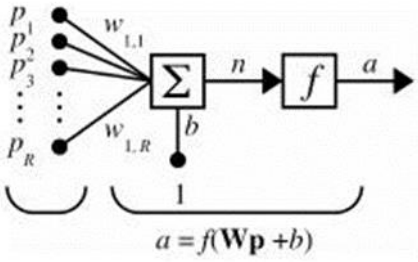

Figure 2. An example of a neural neuron

In perceptrons, neurons are usually grouped and arranged in different layers. Also, during training, the activity functions remain fixed and only the connection weights are changed arbitrarily for adjustment.

Intelligent Model of Dynamic Systems (Data Collection)

The modeled input and output data are obtained by training a mathematical model that is intended to simulate the behavior of the system. In this type of simulation, either an intelligent system such as artificial neural networks or fuzzy logic systems is used, or a training method that is included in classification, among intelligent training techniques, such as genetic algorithms, is used for this purpose. Such modeling is called intelligent modeling, the results of which are usually used in a separate space or domain [19].

Finally, in modeling based on adjusting input and output data, the system operates based on adjusting effective signals such as temperature, pressure values that must be predicted and estimated at future times. Such signals are called output signals and the rest of the signals that are effective in achieving the desired output in the system are called input signals. It is important to note that this type of definition of inputs to the system is different from the definition of control inputs. A control input is a command signal issued by the controller. In closed-loop systems, the input control signal is considered as one of the inputs used to model the reactor. In almost all reaction vessels and reactors designed for industrial processes, including chemical reactors, the dynamic properties and behavior of the system are more striking and prominent than its static properties. Therefore, we classify them as dynamic systems. In a dynamic system, the output values at present and future times are not only dependent on the inputs of that system, but the output values of the system at previous times are also considered another factor determining the output values at present and future times. Therefore, to model dynamic systems, we need a dynamic model with the ability to return the model's output information at present as a control input for controlling the output at future times. After successfully modeling a dynamic system in a discrete space, the values of the output variables at previous moments are used as inputs to the model to predict and estimate the current output values. As an example of the use of control input, consider a model with feedback output information used as a predictive input for a nonlinear single-input-single-output dynamic system [20].

Testing the correctness of the model performance

After the modeling is completed, the model should be tested for performance. It is clear that testing the performance of the model is different from training it. Since our model must be able to predict the behavior of the system at desired future times, in testing the performance of the model, previously estimated output values should be considered as output values with a time delay and used as control inputs that feed back to the system. The following equation is used for the first stage of estimation. In this stage, all the output values (y) are available at previous moments. Since to predict the output at a particular moment, say ys(k+2), we need ys(k+1), but in fact there is no recorded data for it. Therefore, the estimated value, in the first stage, is used for prediction. In other words:

$$y_s(k+2) = f(u(k-nu+2), \ldots, u(k+1), y(k-ny+2), \ldots, y_s(k+1))$$

In other words, it is tested during the period when the system has understood its desired behavior. Nowadays, there are software packages that are used to design and train intelligent models in the field of predictive control. The results obtained from the system performance evaluation by this software are sometimes obtained without considering the characteristics of reversible systems with feedback. In such cases, the outputs recorded in the system performance evaluation section are used as time-delayed inputs instead of the previously estimated outputs, or only the following equation is used to check the system performance. Therefore, the results provided by this software must be reviewed by the designer.

Nonlinear Predictive Control

In nonlinear predictive controllers that are used to predict the behavior of nonlinear systems, we are forced to use nonlinear models. As an example of nonlinear predictive control, if in a discrete space at time k, the output of the system is known as y(k) and the experimental data u obtained from the experiment is considered as the control command signal at that time k, then we will have that the control command is defined by an optimization method for the system. This method is based on the minimization of a performance function that includes the prediction error. The following equation is an example of a performance function, represented by J, which is commonly used in the control of systems with predictive neural networks:

$$J(k)=\sum_{i=1}^{N}[y_s(k+i)-y_d]^2+\rho[u'(k)-u(k-1)]^2.$$

Where y_s and y_d are the predicted and desired outputs of the system, respectively, and u' and u are the experimental and actual inputs, respectively [21].

Characteristics of the reactor under study

The reactor under study is a tank for continuous mixing of two reactants. The diagram of the mixing process of reactants to produce product is shown in the figure below.

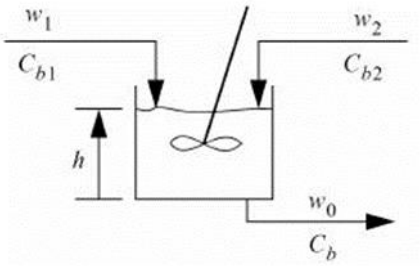

Figure 3. Schematic diagram of the reactor under study

Two fluid streams enter the reactor with concentrations

$$\frac{dC_b(t)}{dt}=(C_{b1}-C_b(t))\frac{w_1(t)}{h(t)}-(C_{b2}-C_b(t))\frac{w_2(t)}{h(t)}-\frac{k_1C_b(t)}{(1+k_2C_b(t))^2}$$

$C_{b_2}=0/1\left(kg/m^3\right)$ و $C_{b_1}=24/9$ و $\left(kg/m^3\right)$,

respectively. The weight rates of the inlet streams are also called w1 and w2. The reactor will have another fluid stream with concentration c_b and weight rate w_0 as the outlet [22]. Another important variable to consider is the liquid height in the reactor (h). A simple mathematical model of the system using mass balance equations is given by:

$$\frac{dh(t)}{dt}=w_1(t)+w_2(t)-0.2\sqrt{h(t)}$$

The flow concentration in the outlet and the liquid height, which can be measured from equation $w_\circ=0.2\sqrt{\mathrm{h}}$, are the output variables that must be measured and adjusted (Figure 4). The general form of the system is as follows:

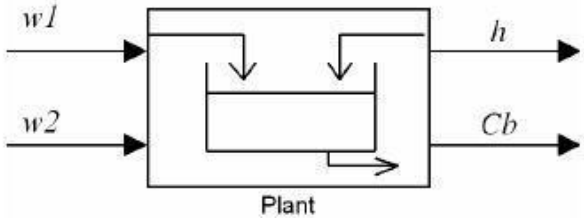

Figure 4. Chemical reactor as a multi-input-multi-output system

Intelligent model of the studied reactor

In this study, modeling has been carried out for a dynamic system in order to achieve predictive control goals. The studied reactor, according to the described characteristics, can have two control inputs w1 and w2 [23]. Since in predictive control, a control command is often used, one of the control inputs must be considered constant. Therefore, we consider the weighted rate of the second input flow with concentration $24/9\,kg/m^3$ as a constant amount $0/1\left(m/s^3\right)$. As a result, from now on, we will not use the weighted rate variable of the second input flow as a control input signal in modeling. The number 2 is considered for the model order and each of the values of n_u and n_y.

$$[\hat{C}_b(k+1), \hat{h}(k+1)] = F[w_1(k-1), w_1(k), C_b(k-1), C_b(k), h(k-1), h(k)]$$

$$\hat{C}_b(k+1) = F_1[w_1(k-1), w_1(k), C_b(k-1), C_b(k), h(k-1), h(k)]$$

$$\hat{h}(k+1) = F_2[w_1(k-1), w_1(k), C_b(k-1), C_b(k), h(k-1), h(k)]$$

$$\hat{C}_b(k+1) = F_1[w_1(k-1), w_1(k), \hat{C}_b(k-1), \hat{C}_b(k), \hat{h}(k-1), \hat{h}(k)]$$

$$\hat{h}(k+1) = F_2[w_1(k-1), w_1(k), \hat{C}_b(k-1), \hat{C}_b(k), \hat{h}(k-1), \hat{h}(k)]$$

In which, the values with the letter A are predicted/estimated values. Sometimes, among the factors affecting the system modeling, one of the outputs is mistakenly ignored [24].

Conclusion

In this project, modeling of dynamic systems with the aim of predicting their output was studied using artificial neural networks in order to control unwanted results and drive the system response towards the desired goals. This type of control is of interest because the output of the system is predicted by the artificial neural network before reaching a stage that is not desirable for us and prevents it from reaching this undesirable stage. As the basis of nonlinear predictive control, the accuracy and reliability of nonlinear models for predicting the outputs of systems are of particular importance. Multilayer perceptrons are used as a network for training and predicting the behavior of the system and a mixing tank with two steady flows and a series of two reacting chemicals are used as the dynamic system studied in this project. The system training and testing data were not obtained from a real system. In fact, the data were collected using a computer model of the system under study. In this study, the ability of the intelligent tool to predict the behavior of a nonlinear multi-input-multi-output reactor has been demonstrated. We also saw that a set of important points and variables must be considered simultaneously in intelligent modeling. Even if only one or a few of the above variables are the desired system response. In this way, one can be confident in the validity of the responses obtained from the said model, especially when this model is designed for predictive control purposes. Some important points in intelligent modeling for predictive control purposes are listed as follows:

1- Fixed system inputs in modeling can be assumed constant, provided that they remain constant during training and simulation.

2- The values of different variables that are presented to the system through different signals such as temperature, pressure, etc., must be very close to each other.

3- Dynamic systems, including all reactors and reaction vessels, can be modeled by reversible models.

4- In testing and validating the results of dynamic models at desired future times, the estimated output values of the model should be used as control inputs to the system and the recorded output data should only be used to compare the results.

These results are reliable and usable when all the inputs and outputs of the system are properly defined and the type of intelligent tool used to model them is of less importance. The single-layer perceptron has shown remarkable success despite the simplicity of the model. The perceptron has demonstrated the properties we expect from a trainable system and has proven that it can distinguish classes of objects from each other if they are linearly separable. What is needed is a way to overcome the problem of linear inseparability without losing the main features of the perceptron and its overall simplicity. Addressing this need first attracted the attention of the scientific community in 1986, when Rommel Hart and McClelland invented a new form of the perceptron model called the multilayer perceptron.

Disclosure Statement

No potential conflict of interest reported by the authors.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Authors' Contributions

All authors contributed to data analysis, drafting, and revising of the paper and agreed to be responsible for all the aspects of this work.

References

[1] Mahdiraji, E. A.; Zarini, M. K. (**2025**), Application of Distributed Algorithm in a Microgrid for Optimal Energy Management and Rapid Convergence. *Qjoest*,, 6(1), 55-70.

[2] Mahdiraji, E. A.; Amiri, M. S. Shariatmadar, S. M. (**2021**), Analysis of Lightning Strikes on the Transmission Line by Considering the Frequency-Dependent Model. *Qjoest*, 2(6), 12-36.

[3] Amouzad Mahdiraji, E., Shariatmadar, S. (**2019**), A New Method for Simplification and

Reduction of State Estimation's Computational Complexity in Stability Analysis of Power Systems. *International Journal of Smart Electrical Engineering*, *08*(02): 51-58.

[4] Amouzad Mahdiraji, E., Shariatmadar, S. (**2019**), Improving the Transient Stability of Power Systems Using STATCOM and Controlling it by Honey Bee Mating Optimization Algorithm. *International Journal of Smart Electrical Engineering*, 08(03): 99-104.

[5] Mahdiraji, E. A.; Amiri, M. S.; Shariatmadar, S. M. (**2021**), Voltage Load Shedding Considering Voltage Sensitivity and Reactive Power. *Qjoest*, 2021, 2(6), 55-72.

[6] Khodadadi Zarini, M. Mahdiraji. EA. (**2024**), Review of Energy Management in Micro Grid in Power Engineering. *J. Eng. Ind. Res*, 5 (2): 90-99.

[7] Mahdiraji, E. A. (**2022**), Optimal Purchase and Sale of Energy in Electricity Markets Due to Various Uncertainties in Microgrid. *Qjoest*, 3(1), 29-39.

[8] Mahdiraji, EA. (**2024**), Effects of Fourier Transform and Modal Theory in Electrotherapeutic Signals, *Eurasian Journal of Chemical, Medicinal and Petroleum Research*. 3(1):1-6.

[9] Mahdiraji, EA. Amiri, MS. (**2021**), Adaptive Control of Network Frequency by Doubly-Fed Induction Generators Using a Data-Driven Method. *Eurasian. J. Sci. Technol*, 1(2), 75-91.

[10] Mahamedi, B., Zhu, J. G., & Hashemi, S. M. (**2016**). A setting-free approach to detecting loss of excitation in synchronous generators. *IEEE Transactions on Power Delivery*, *31*(5), 2270–2278.

[11] Samimi, A. Bozorgian, A. Samimi, M. (**2022**), An Analysis of Risk Management in Financial Markets and Its Effects, *Journal of Engineering in Industrial Research* 3 (1), 1-7, 2022

[12] Alavi, A., & Mohammadi, M. (**2020**). Effects of temperature on sulfidation in refinery furnaces. *Journal of Materials Engineering and Performance*, 29(5), 2781-2789.

[13] Chen, L., & Zhou, M. (**2021**). Mechanisms of chloride stress corrosion cracking in refinery stainless steels. *Corrosion Reviews*, 39(1), 57-68.

[14] Samimi, A. (**2024**), Risk Management in EPC Projects, *Eurasian Journal of Chemical, Medicinal and Petroleum Research*, 3 (5), 185-195

[15] Sadr, M. B. (**2025**). Analysis of Spatial Criteria of Industries in the Village Based on Environmental Standards. *Journal of Chemical Engineering and Energy Materials*, 1(1), 1-8. doi: 10.22034/jceem.2025.220371

[16] Samimi, A. (**2021**), The Need for Risk Assessment in Occupational Health, *Journal of Engineering in Industrial Research*, 2 (2), 71-76

[17] Navaei, O. (**2025**), Personalized Medicine: Tailoring Treatment Plans Based on Genetic Profile, *Eurasian Journal of Chemical, Medicinal and Petroleum Research*, 4 (2), 129-151

[19] Samimi, A. (**2020**), Investigation the Impact of Risk Management and Audit Committee on Industrial Companies, *Journal of Exploratory Studies in Law and Management*, 7 (3), 132-137

[20] Imanzadeh, M. (**2025**), Development of Solid State Electrolytes for Next Generation Lithium-Ion Batteries, *Eurasian Journal of Chemical, Medicinal and Petroleum Research*, 4 (1), 32-47

[21] Samimi, A. (**2021**), The Need for Risk Assessment in Occupational Health, *Journal of Engineering in Industrial Research*, 2 (2), 71-76

[22] Gandomi, MJ. (**2025**), Use of New Technologies for Enhanced CO2 Reduction Reactions, *Eurasian Journal of Chemical, Medicinal and Petroleum Research*, 4 (1), 15-31

[23] Sabzehali, S. (**2025**), Nanomaterials in Drug Delivery Systems: Challenges and Perspectives, *Eurasian Journal of Chemical, Medicinal and Petroleum Research*, 4 (1), 48-62

[24] Kiamansouri, M. (**2025**), Integration of Renewable Energy Sources in Oil and Gas Operations a Sustainable Future, *Eurasian Journal of Chemical, Medicinal and Petroleum Research*, 4 (1), 63-87