


Article

Shallow Fully Connected Neural Network Training by Forcing Linearization into Valid Region and Balancing Training Rates

Jea Pil Heo ¹, Chang Gyu Im ¹, Kyung Hwan Ryu ^{2,*}, Su Whan Sung ^{1,*} , Changkyoo Yoo ³ and Dae Ryook Yang ⁴

¹ Department of Chemical Engineering, Kyungpook National University, Daegu 41566, Korea; fheod@knu.ac.kr (J.P.H.); tgi1945@knu.ac.kr (C.G.I.)

² Department of Chemical Engineering, Sunchon National University, Suncheon 57922, Korea

³ Department of Environmental Engineering, Kyung Hee University, Yongin 17104, Korea; ckyoo@khu.ac.kr

⁴ Department of Chemical and Biological Engineering, Korea University, Seoul 02841, Korea; dryang@korea.ac.kr

* Correspondence: khryu@scnu.ac.kr (K.H.R.); suwhansung@knu.ac.kr (S.W.S.)

Abstract: A new supervisory training rule for a shallow fully connected neural network (SFCNN) is proposed in this present study. The proposed training rule is developed based on local linearization and analytical optimal solutions for linearized SFCNN. The cause of nonlinearity in neural network training is analyzed, and it is removed by local linearization. The optimal solution for the linearized SFCNN, which minimizes the cost function for the training, is analytically derived. Additionally, the training efficiency and model accuracy of the trained SFCNN are improved by keeping estimates within a valid range of the linearization. The superiority of the proposed approach is demonstrated by applying the proposed training rule to the modeling of a typical nonlinear pH process, Boston housing prices dataset, and automobile mileage per gallon dataset. The proposed training rule shows the smallest modeling error and the smallest iteration number required for convergence compared with several previous approaches from the literature for the case study.

Keywords: neural network; training rule; local linearization; optimal solution; pH system modeling



Citation: Heo, J.P.; Im, C.G.; Ryu, K.H.; Sung, S.W.; Yoo, C.; Yang, D.R. Shallow Fully Connected Neural Network Training by Forcing Linearization into Valid Region and Balancing Training Rates. *Processes* **2022**, *10*, 1157. <https://doi.org/10.3390/pr10061157>

Academic Editor: Jie Zhang

Received: 14 May 2022

Accepted: 7 June 2022

Published: 9 June 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Artificial neural networks (ANNs) have been widely used in many research areas [1–18]. One of the most popular ANNs for modeling dynamic systems is the shallow fully connected neural network (SFCNN). In theory, it is capable of describing any complex nonlinear dynamics accurately using a sufficient number of hidden nodes [15]. Despite its capability, its application is limited by a huge computational load and convergence problems during supervisory training. Many researchers have developed a variety of supervisory training rules using gradient descent, Levenberg–Marquardt, quasi-Newton, and conjugate direction methods [16–21].

Gradient descent is one of the most popular algorithms for performing optimizations and is currently the most common method of training neural networks due to its effectiveness in dealing with enormous data. However, it inherently requires a massive number of learning iterations and has disadvantages in that it is vulnerable to getting stuck in saddle points under non-convex optimization problems [19,22]. Some algorithms were proposed to deal with this problem using momentum and adaptive learning rate, including the Nesterov accelerated gradient (NAG) [20], root mean square propagation (RMSProp), and adaptive moment estimation (ADAM) [21]. Although these approaches are widely used in neural network training and have proven their performance, their slow convergence remains a challenge for real-time applications [23]. An optimization method based on Hessian of training loss (e.g., Newton's method and the Levenberg–Marquardt method) is free from slow convergence from massive iteration and getting stuck in a saddle point. Nevertheless, difficulty in computing exact Hessian eigenvalues of a large size model [24]

places a method outside the mainstream of neural network training. In order to get over this hurdle, research teams proposed a diverse study focusing on computing a limited number of eigenvalues [25–27] and estimating information (i.e., curvature, sharpness, etc.) in the absence of sufficient information about an eigenvalue spectrum [28–34]. In our previous study, a training method to reduce the searching space as much as the number of the output weights in the high-dimensional nonlinear optimization problem has been proposed to alleviate the computational load [35]. Although this approach shows remarkable improvement compared with full-dimensional optimization approaches, much room remains to be improved as it cannot analytically estimate the optimal input weights.

In this paper, an improved supervisory training rule for an SFCNN using local linearization and the analytic optimal solution of a linearized SFCNN is proposed. First, the cause of nonlinearity in neural network training is analyzed, and it is locally linearized. Next, the optimal solution is derived that minimizes the cost function for training. A new cost function is proposed to keep the estimate within a valid domain of linearization. A new training rule is finally derived based on the derived optimal solution. The performance of the proposed approach is verified by applying the proposed training rule to the modeling of a typical nonlinear pH process, Boston housing prices dataset, and automobile mileage per gallon dataset. The proposed method is compared with four widely used methods: the gradient descent with momentum, stochastic gradient descent, Levenberg–Marquardt methods, and adaptive moment estimation. From the simulation results, it can be confirmed that the iteration number and the modeling error are the smallest when compared to the existing approaches.

The remaining contents of this manuscript are represented as follows: In Section 2, the proposed training method is introduced and derived based on local linearization and analytical optimal solution. Section 3 presents the performance of the proposed method compared with previous methods, including the gradient descent with momentum, the stochastic gradient descent, the adaptive momentum algorithm, and the Levenberg–Marquardt method. Finally, the conclusion and further extension of the proposed method are discussed in Section 4.

2. Proposed Supervisory Training Rule for SFCNN

Let us consider the SFCNN structure represented in Figure 1. The neural network is mathematically formulated as follows:

$$\hat{z}_j(q) = \sum_{i=1}^m v_{j,i} u_i(q), \quad j = 1, 2, 3, \dots, l \quad (1)$$

$$\hat{h}_j(q) = \frac{1}{1 + \exp(-\alpha \hat{z}_j(q))}, \quad j = 1, 2, 3, \dots, l \quad (2)$$

$$\hat{y}_k(q) = \sum_{j=1}^l w_{k,j} \hat{h}_j(q), \quad k = 1, 2, 3, \dots, n \quad (3)$$

where $v_{j,i}$ and $w_{k,j}$ are the input weight and the output weight, respectively; $\hat{h}_j(q)$ and $\hat{y}_k(q)$ denote the output of the j -th hidden node and the k -th output node, respectively; $u_i(q)$ is the input of the i -th input node; q donates the q -th sampling of the process input and output.

We will now analyze the cause of the nonlinearity of the neural network. Note that $\hat{z}_j(q)$ of Equation (1) is linear with respect to the input weight. Meanwhile, $\hat{h}_j(q)$ and $\hat{y}_k(q)$ of Equations (2) and (3) are nonlinear with respect to the input weight. It now becomes apparent that the cause of the nonlinearity originates from the activation function of Equation (2). Therefore, it can be concluded that the optimal weights for the training can be analytically estimated if the activation function and the output layer are linearized.

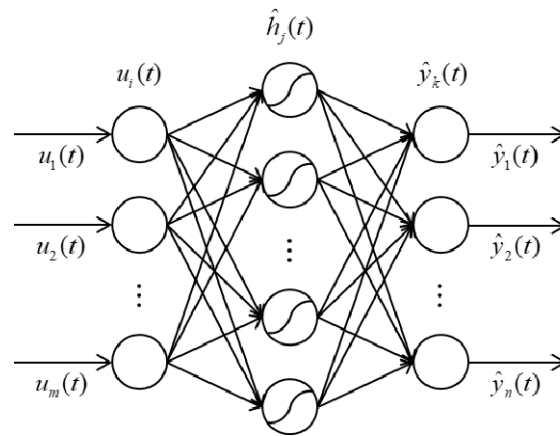


Figure 1. The considered SFCNN structure.

2.1. Derivation of the Proposed Training Rule

In the present study, the following linearization for the activation function is adopted for the training. It is linearized at the estimates of the previous iteration in the supervisory training.

$$\hat{h}_j(q) \approx \hat{h}_j^{lin}(q) = \left. \frac{d\hat{h}_j}{dz_j} \right|_{\hat{z}_j(q)=\hat{z}_j^{iter-1}(q)} (\hat{z}_j(q) - \hat{z}_j^{iter-1}(q)) + \hat{h}_j^{iter-1}(q) \quad (4)$$

where a superscript *iter* is an iteration number in the supervisory training and the superscript *lin* means a linearized function. The output layer is linearized by Equation (5):

$$\hat{y}_k(q) \approx \hat{y}_k^{lin}(q) = \sum_{j=1}^l \left((w_{k,j} - w_{k,j}^{iter-1}) \hat{h}_j^{iter-1}(q) + w_{k,j}^{iter-1} (\hat{h}_j(q) - \hat{h}_j^{iter-1}(q)) + w_{k,j}^{iter-1} \hat{h}_j^{iter-1}(q) \right) \quad (5)$$

In addition, the following cost function of the multi-objective quadratic programming problem is used for supervisory training in this research. The proposed method will estimate the input and output weights at each iteration number of *iter* by minimizing the cost function based on the linearized SFCNN from the previous iteration number of *iter* – 1.

$$\begin{aligned} \min_{v_{j,i}, w_{k,j}} Q &= 0.5 \sum_{q=1}^N \sum_{k=1}^n \left(y_k(q) - \hat{y}_k^{lin}(q) \right)^2 + 0.5 R_h \sum_{q=1}^N \sum_{j=1}^l \left(\hat{h}_j(q) - \hat{h}_j^{lin}(q) \right)^2 \\ &+ 0.5 R_y \sum_{q=1}^N \sum_{k=1}^n \left(\hat{y}_k(q) - \hat{y}_k^{lin}(q) \right)^2 + 0.5 R_v \sum_{j=1}^l \sum_{i=1}^m \left(v_{j,i}^{iter-1} - v_{j,i} \right)^2 \\ &+ 0.5 R_w \sum_{k=1}^n \sum_{j=1}^l \left(w_{k,j}^{iter-1} - w_{k,j} \right)^2 \end{aligned} \quad (6)$$

Here, $y_k(q)$ is the measurement of the process output, and $\hat{y}_k^{lin}(q)$ is the model output of the linearized SFCNN; $\hat{h}_j(q)$ and $\hat{y}_k(q)$ denote the output of the hidden node and that of the output node of the SFCNN; $\hat{h}_j^{lin}(q)$, $\hat{y}_k^{lin}(q)$ denote the output of the hidden node and that of the output node of the linearized SFCNN. The constants, R_h , R_y , R_v , and R_w to balance the five terms of the right-hand side of the cost function are updated at each iteration, as will be discussed later.

It should be noted that a number of important terms for more efficient training are included in the cost function as follows: The first term on the right-hand side is the square modeling error between the measurement of the process output ($y_k(q)$) and the SFCNN model output ($\hat{y}_k^{lin}(q)$). Most previous training methods train the SFCNN by minimizing the modeling error term.

The second and third terms are to minimize the distance between the signals of the SFCNN ($\hat{h}_j(q)$ and $\hat{y}_k(q)$) and those of the linearized SFCNN ($\hat{h}_j^{lin}(q)$ and $\hat{y}_k^{lin}(q)$) to keep

the training within the valid range of the linearization. If the cost function does not include the second and the third term, the weights of the SFCNN obtained by minimizing the cost function would be placed outside of the valid range of the linearization, causing problems, such as the divergence and saturation phenomenon of the activation functions during the training.

The fourth and fifth terms are to minimize the distances between the optimal weights of $iter$ and the weights of $iter - 1$ to keep the training from changing the input and output weights too much. These terms play an important role in preventing singularity problems during the training using the Hessian matrix. In most cases, the number of the weights of the SFCNN is huge (equivalent to overparameterization) and easily causes a singularity problem. Therefore, it is a must then to include the fourth and fifth terms in the cost function. The previous training rule using the Levenberg–Marquardt optimization method, known as the fastest method, adds a small diagonal matrix to the Hessian matrix to prevent a singularity problem during the training. The addition of the small diagonal matrix is equivalent to adding the fourth and fifth penalty terms of Equation (6) in the cost function, keeping the estimates from changing too much [36,37]. However, the Levenberg–Marquardt method has no direct instruments to guarantee the validity of the linearization because it does not consider the terms, such as the second and the third penalty terms, during the training. Thus, it cannot overcome the potential problem that training goes too far because of an unacceptable accuracy of linearization. This is one of the significant differences between the proposed method and previous approaches.

Note, however, that the optimal solution of Equation (6) can be estimated analytically because the activation function and the output layer are linearized as Equations (4) and (5). In order to derive the analytic solution, consider the following equations derived from Equations (1) and (4)–(6).

$$\begin{aligned} \frac{\partial Q}{\partial v_{a,b}} &= - \sum_{q=1}^N \sum_{k=1}^n \left(y_k(q) - \hat{y}_k^{lin}(q) \right) \frac{\partial \hat{y}_k^{lin}(q)}{\partial v_{a,b}} - R_v \sum_{q=1}^N \sum_{j=1}^l \left(\hat{h}_j(q) - \hat{h}_j^{lin}(q) \right) \frac{\partial \hat{h}_j^{lin}(q)}{\partial v_{a,b}} \\ &\quad - R_w \sum_{q=1}^N \sum_{k=1}^n \left(\hat{y}_k(q) - \hat{y}_k^{lin}(q) \right) \frac{\partial \hat{y}_k^{lin}(q)}{\partial v_{a,b}} + A \end{aligned} \quad (7)$$

$$A = -R_v \sum_{j=1}^l \sum_{i=1}^m (v_{j,i}^{iter-1} - v_{j,i}) \text{ if } j = a \text{ and } i = b, A = 0 \text{ if } j \neq a \text{ or } i \neq b$$

$$\begin{aligned} \frac{\partial Q}{\partial w_{a,b}} &= - \sum_{q=1}^N \sum_{k=1}^n \left(y_k(q) - \hat{y}_k^{lin}(q) \right) \frac{\partial \hat{y}_k^{lin}(q)}{\partial w_{a,b}} - R_w \sum_{q=1}^N \sum_{k=1}^n \left(\hat{y}_k(q) - \hat{y}_k^{lin}(q) \right) \frac{\partial \hat{y}_k^{lin}(q)}{\partial w_{a,b}} + B \end{aligned} \quad (8)$$

$$B = -R_w \sum_{k=1}^n \sum_{j=1}^l \left(w_{k,j}^{iter-1} - w_{k,j} \right) \text{ if } k = 1 \text{ and } j = b, B = 0 \text{ if } k \neq a \text{ or } j \neq b$$

$$\begin{aligned} \frac{\partial^2 Q}{\partial v_{a,b} \partial v_{c,d}} &= - \sum_{q=1}^N \sum_{k=1}^n \left(y_k(q) - \hat{y}_k^{lin}(q) \right) \frac{\partial^2 \hat{y}_k^{lin}(q)}{\partial v_{a,b} \partial v_{c,d}} + \sum_{q=1}^N \sum_{k=1}^n \frac{\partial \hat{y}_k^{lin}(q)}{\partial v_{a,b}} \frac{\partial \hat{y}_k^{lin}(q)}{\partial v_{c,d}} \\ &\quad - R_v \sum_{q=1}^N \sum_{j=1}^l \left(\hat{h}_j(q) - \hat{h}_j^{lin}(q) \right) \frac{\partial \hat{h}_j^{lin}(q)}{\partial v_{a,b} \partial v_{c,d}} + R_v \sum_{q=1}^N \sum_{j=1}^l \frac{\partial \hat{h}_j^{lin}(q)}{\partial v_{a,b}} \frac{\partial \hat{h}_j^{lin}(q)}{\partial v_{c,d}} \\ &\quad - R_w \sum_{q=1}^N \sum_{k=1}^n \left(\hat{y}_k(q) - \hat{y}_k^{lin}(q) \right) \frac{\partial \hat{y}_k^{lin}(q)}{\partial v_{a,b} \partial v_{c,d}} + R_w \sum_{q=1}^N \sum_{k=1}^n \frac{\partial \hat{y}_k^{lin}(q)}{\partial v_{a,b}} \frac{\partial \hat{y}_k^{lin}(q)}{\partial v_{c,d}} + C \end{aligned} \quad (9)$$

$$C = R_v \text{ if } k = a = c \text{ and } j = b = d, D = 0 \text{ if } k \neq c, k \neq c, j \neq b, \text{ or } j \neq d$$

$$\begin{aligned} \frac{\partial^2 Q}{\partial w_{a,b} \partial w_{c,d}} &= - \sum_{q=1}^N \sum_{k=1}^n \left(y_k(q) - \hat{y}_k^{lin}(q) \right) \frac{\partial^2 \hat{y}_k^{lin}(q)}{\partial w_{a,b} \partial w_{c,d}} + \sum_{q=1}^N \sum_{k=1}^n \frac{\partial \hat{y}_k^{lin}(q)}{\partial w_{a,b}} \frac{\partial \hat{y}_k^{lin}(q)}{\partial w_{c,d}} \\ &\quad - R_w \sum_{q=1}^N \sum_{k=1}^n \left(\hat{y}_k(q) - \hat{y}_k^{lin}(q) \right) \frac{\partial \hat{y}_k^{lin}(q)}{\partial w_{a,b} \partial w_{c,d}} + R_w \sum_{q=1}^N \sum_{k=1}^n \frac{\partial \hat{y}_k^{lin}(q)}{\partial w_{a,b}} \frac{\partial \hat{y}_k^{lin}(q)}{\partial w_{c,d}} + D \end{aligned} \quad (10)$$

$$D = R_w \text{ if } k = a = c \text{ and } j = b = d, D = 0 \text{ if } k \neq a, k \neq c, j \neq b, \text{ or } j \neq d$$

$$\frac{\partial^2 Q}{\partial v_{a,b} \partial w_{c,d}} = - \sum_{q=1}^N \sum_{k=1}^n \left(y_k(q) - \hat{y}_k^{lin}(q) \right) \frac{\partial^2 \hat{y}_k^{lin}(q)}{\partial v_{a,b} \partial w_{c,d}} + \sum_{q=1}^N \sum_{k=1}^n \frac{\partial \hat{y}_k^{lin}(q)}{\partial v_{a,b}} \frac{\partial \hat{y}_k^{lin}(q)}{\partial w_{c,d}} - R_w \sum_{q=1}^N \sum_{k=1}^n \left(\hat{y}_k(q) - \hat{y}_k^{lin}(q) \right) \frac{\partial \hat{y}_k^{lin}(q)}{\partial v_{a,b} \partial w_{c,d}} + R_w \sum_{q=1}^N \sum_{k=1}^n \frac{\partial \hat{y}_k^{lin}(q)}{\partial v_{a,b}} \frac{\partial \hat{y}_k^{lin}(q)}{\partial w_{c,d}} \quad (11)$$

$$\frac{\partial \hat{y}_k^{lin}(q)}{\partial v_{a,b}} = \sum_{j=1}^l w_{k,j}^{iter-1} \frac{\partial \hat{h}_j^{lin}(q)}{\partial v_{a,b}} \quad (12)$$

$$\frac{\partial \hat{h}_j^{lin}(q)}{\partial v_{a,b}} = \left. \frac{d \hat{h}_j}{d \hat{z}_j} \right|_{z_j(q)=\hat{z}_j^{iter-1}(q)} \frac{\partial \hat{z}_j^{lin}(q)}{\partial v_{a,b}} \quad (13)$$

$$\frac{\partial \hat{z}_j^{lin}(q)}{\partial v_{a,b}} = u_b(q) \text{ if } j = a, \quad \frac{\partial \hat{z}_j^{lin}(q)}{\partial v_{a,b}} = 0 \text{ if } j \neq a \quad (14)$$

$$\frac{\partial \hat{y}_k^{lin}(q)}{\partial w_{a,b}} = h_b^{iter-1}(q) \text{ if } k = a, \quad \frac{\partial \hat{y}_k^{lin}(q)}{\partial w_{a,b}} = 0 \text{ if } k \neq a \quad (15)$$

From Equations (14) and (15), it is clear that the second derivatives of $\hat{z}_j^{lin}(q)$ and $\hat{y}_k^{lin}(q)$ are zeros, which means that all the derivatives of Q more than twice are zeros when we consider Equations (7)–(13). Therefore, the cost function Q becomes a quadratic form in terms of the input and output weights, resulting in the following quadratic equation:

$$\nabla Q = \nabla^2 Q \Big|_{iter-1} (\theta - \theta^{iter-1}) + \nabla Q \Big|_{iter-1} \quad (16)$$

where θ is a vector composed of the input and the output weights. The ∇Q and $\nabla^2 Q$ are the gradient and Hessian matrix of Q , respectively. Finally, the analytic optimal solution for the cost function of Equation (6) can be derived from Equation (16) by setting $\nabla Q = 0$. Then, the following training rule for the cost function is obtained from the optimal solution.

$$\theta^{iter} = \theta^{iter-1} - \left(\nabla^2 Q \Big|_{iter-1} \right)^{-1} \nabla Q \Big|_{iter-1} \quad (17)$$

All the numerical problems in adopting the cost function of Equation (6) except determining the constants of R_h , R_y , R_v , and R_w , are now solved.

2.2. Determining Hyperparameters

The constant, R_h , R_y , R_v , and R_w , are update each iteration by:

$$R_{v,iter} = R_{iter} \frac{\sum_{i=0}^{i=iter} \left(\|\partial Q / \partial v\|_{F,i}^2 \lambda^{iter-i} \right)}{\sum_{i=0}^{i=iter} \left(\|\partial Q / \partial w\|_{F,i}^2 \lambda^{iter-i} \right) + \sum_{i=0}^{i=iter} \left(\|\partial Q / \partial v\|_{F,i}^2 \lambda^{iter-i} \right)} \quad (18)$$

$$R_{w,iter} = R_{iter} \frac{\sum_{i=0}^{i=iter} \left(\|\partial Q / \partial w\|_{F,i}^2 \lambda^{iter-i} \right)}{\sum_{i=0}^{i=iter} \left(\|\partial Q / \partial w\|_{F,i}^2 \lambda^{iter-i} \right) + \sum_{i=0}^{i=iter} \left(\|\partial Q / \partial v\|_{F,i}^2 \lambda^{iter-i} \right)} \quad (19)$$

$$R_{h,iter} = R_{v,iter} \quad (20)$$

$$R_{y,iter} = R_{w,iter} \quad (21)$$

where $\|\bullet\|_F$ and $\lambda^{iter-i} < 1$ donate the Frobenius norm and a forgetting factor, respectively. The subscript $iter$ is the iteration number. R_{iter} controls the overall speed of the convergence. If R_{iter} is larger, the convergence rate decreases and the robustness increases, and vice versa.

In this research, R_{iter} is updated as follows: if the modeling error of the $iter$ iteration decreases compared to that of the $iter-1$ iteration, it decreases R_{iter} by $R_{iter} = R_{iter-1}/\alpha$, $\alpha > 1$, and updates θ^{iter} by Equation (17) to increase the convergence rate. Otherwise, it increases R_{iter} by $R_{iter} = \beta R_{iter-1}$, $\beta > 1$ to increase the robustness and does not update θ^{iter} as $\theta^{iter} = \theta^{iter-1}$. This is similar to the Levenberg–Marquardt method. However, the update of $R_{v,iter}$ and $R_{w,iter}$ is totally different. The update rule of Equations (18) and (19) play an important role in equalizing the two gradients of the cost function with respect to the input weights and the output weights. If the gradient for the output weights of $\|\partial Q/\partial w\|_F^2$ is bigger than the gradient for the input weights of $\|\partial Q/\partial v\|_F^2$, the proposed method adjusts the weights toward equalizing the two gradients by penalizing the movements of the output weights more than those of the input weights according to Equations (18)–(21). This feature can improve the efficiency of the training significantly because unbalanced training between the input weights and the output weights slows down the convergence rate and causes saturation of the activation function, resulting in unacceptable local minima. This is the other uniqueness of the proposed method. The previous training rules, such as gradient descent and Levenberg–Marquardt, cannot incorporate this feature

3. Results and Discussions

In order to verify the performance of the proposed training method, the study performed three case studies, including a highly nonlinear pH process, the Boston house price and automobile mileage per gasoline datasets, which are widely used in neural network training problems. In the present study, the performances of four different training methods, including the gradient descent with momentum (GDM), the Levenberg–Marquardt (LM), the stochastic gradient descent (SGD), and the adaptive momentum (ADAM) algorithms, are compared with the proposed method. All the neural network pieces of training are conducted via MATLAB. In the present study, the learning rate scheduler with an initial learning rate of 0.001, provided in the MATLAB embedded function, is used for the GDM, SGD, and ADAM methods. For the LM and the proposed methods, the initial damping parameter (R_0) of 1.0 and the update factors, α and β of 1.2 and 10.0 are used, respectively.

3.1. A pH Process

A pH process known as one of the highly nonlinear chemical processes is simulated to demonstrate the advantages of the proposed method compared with previous ones. Figure 2 shows the pH process where the weak acid influent of phosphoric acid (H_3PO_4) is treated by the strong base of sodium hydroxide (NaOH) in a continuously stirred tank reactor. Here, the feed flow rate (F) and the reactor working volume (V) are 2 L/min and 10 L, respectively. The total concentrations of the phosphoric acid in the influent stream and the sodium hydroxide in the titrating stream are 0.05 and 0.2 mol/L, respectively. The initial total ion concentrations of the phosphoric acid and the sodium hydroxide in the reactor are 0.05 and 0.05 mol/L, respectively. For the detailed material balance equations and an equilibrium, equations are referred to previously published works [38].

The process is perturbed by the titrating stream (u) of uniformly distributed random noises between 0 and 1 L/min, and the sampling time is 1 min. This study assumes that the obtained pH data is corrupted by measurement noises of ± 0.05 . The data used in the present study is represented in Figure 3. Moreover, u and pH data before the 525th samples are used for training, and data after the 525th samples are used for validation. The input nodes of the neural network consist of $u(t-1)$, $u(t-2)$, $pH(t-1)/12$, $pH(t-2)/12$, and one bias. The output of the neural network is $pH(t)/12$. The number of hidden nodes is 3 or 7. The training of a pH process is performed for 20 cases with randomly generated input data to verify the performance of the proposed method more reliably.

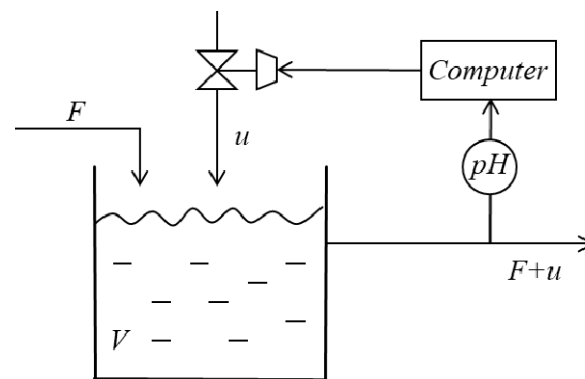


Figure 2. Schematic diagram of the pH process.

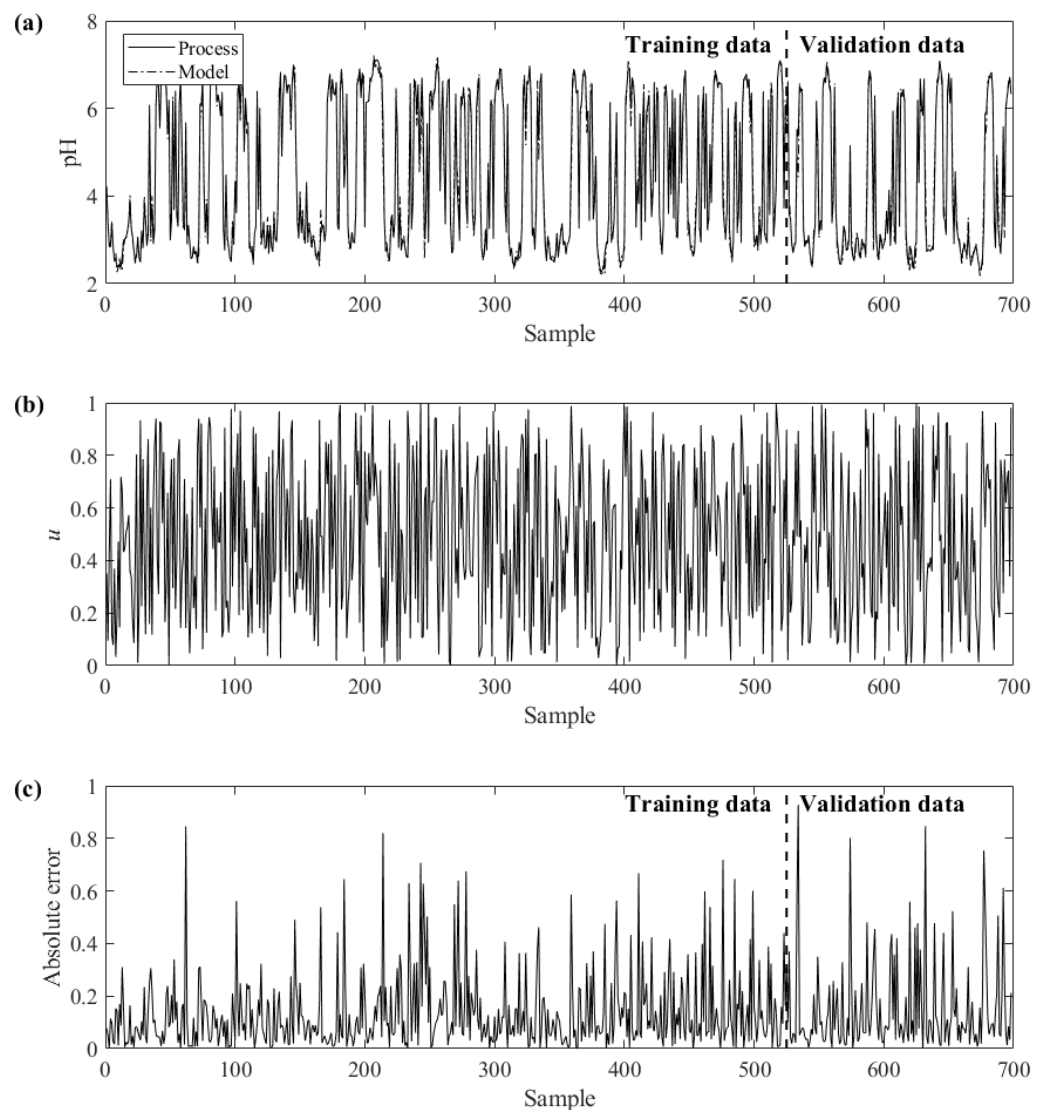


Figure 3. Process data and performance: (a) process output and model output obtained from neural network with 7 hidden nodes, (b) process input, and (c) absolute error between measured and predicted values.

Figures 3 and 4 show one of the training results. As illustrated in Figure 3c, the proposed method has a higher modeling accuracy for the pH process under training and validation results. The comparison result between the proposed method and previous

methods is represented in Figure 4. As confirmed in Figure 4, the mean square error (MSE) values of the previous approach (except the LM method) after 200 iterations are much bigger than those of the proposed method after just 10 iterations. The study multiply performs neural network training for the pH process in order to verify a reliable performance. The results of training error, the number of iterations to converge, and the computation time obtained from the repeated simulation (20 times with different datasets generated from simulations with randomly generated input values) are summarized in Tables 1 and 2. As enumerated in Table 1, the proposed method shows an outstanding modeling performance when comparing the training errors after 100,000 iterations. In order to make a reasonable comparison of the convergence rate between the diverse methods, this study compares the number of iterations and computational time for the convergence of reaching within $\pm 1.0\%$ of the MSE value in Table 1. The required number of iterations and computation time (except ADAM) is smaller than the previous methods, as represented in Table 2. Despite the fact that ADAM can compute many iterations in a very short time, it has a very poor modeling performance compared to the proposed method. Consequently, it can be concluded that the proposed method provides the fastest convergence rate and the smallest modeling error compared to all the previous approaches.

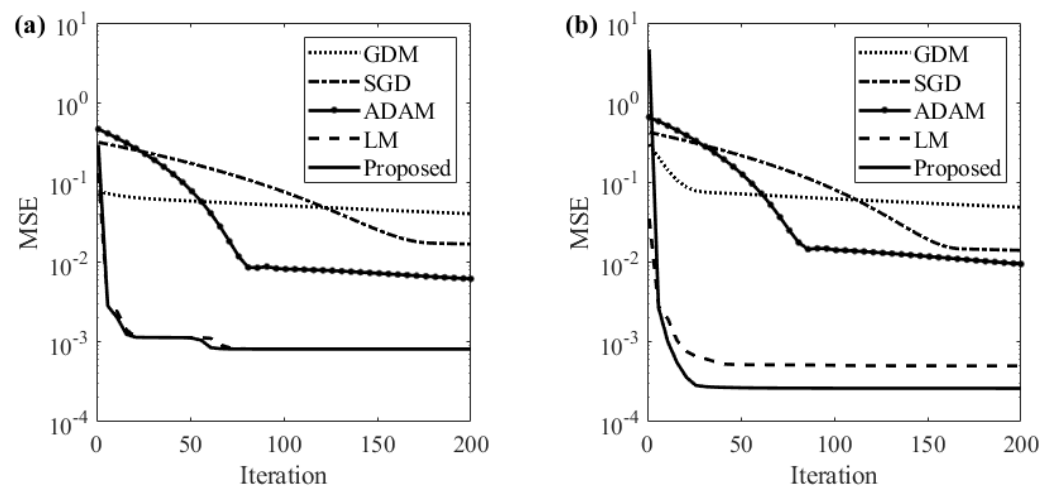


Figure 4. Error convergences of the proposed training method and the previous methods (GDM: gradient descent with momentum, SDG: stochastic gradient descent, ADAM: adaptive moment estimation, and LM: Levenberg–Marquardt) up to 200 iterations: (a) with 3 hidden nodes and (b) 7 hidden nodes.

Table 1. Training performance for the pH process after 100,000 iterations—training errors (mean squared error; MSE).

Method	Hidden Layer with 3 Nodes		Hidden Layer with 7 Nodes	
	Mean	Standard Deviation	Mean	Standard Deviation
GDM	7.319×10^{-3}	4.417×10^{-4}	4.046×10^{-3}	1.529×10^{-4}
SGD	1.697×10^{-3}	7.664×10^{-5}	1.681×10^{-3}	7.235×10^{-5}
ADAM	1.668×10^{-3}	7.142×10^{-5}	1.668×10^{-3}	7.142×10^{-5}
LM	1.201×10^{-3}	3.603×10^{-4}	3.068×10^{-4}	6.079×10^{-5}
Proposed	1.140×10^{-3}	3.284×10^{-4}	2.825×10^{-4}	4.940×10^{-5}

Table 2. Training performance for the pH process after 100,000 iterations—number of iterations and computation time of hidden layer with 7 nodes.

Method	Number of Iterations to Converge		Computation Time to Converge (s)	
	Mean	Standard Deviation	Mean	Standard Deviation
GDM	94,494	843.1	130.4	16.4
SGD	6318.3	2815.3	29.0	12.5
ADAM	637.57	180.25	3.82	1.23
LM	145.55	63.79	8.68	3.74
Proposed	107.40	44.99	7.94	3.30

3.2. Representative Datasets for Neural Network Training: Boston Housing Price and Automobile Mileage Per Gallon

To ensure the general performance of the proposed method, the study performs SFCNN training with the Boston housing prices [39] dataset and automobile mileage per gallon (MPG) [40] dataset, which are commonly used datasets for testing neural network training. This study uses 75% and 25% of the dataset as the training set and validation set, respectively. Data for training and validation are randomly selected. The used neural network takes 20 nodes into account. The training results of both the Boston housing prices and automobile MPG are depicted in Figure 5.

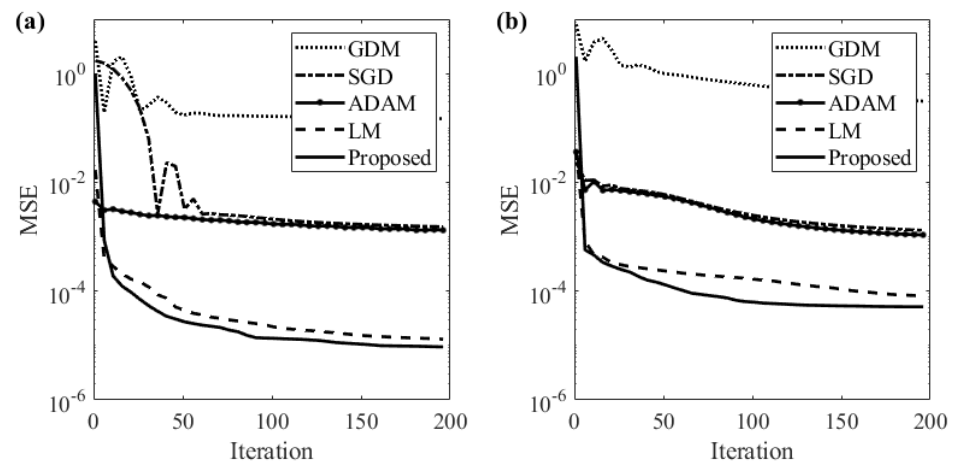


Figure 5. Training error convergences of the proposed training method and the previous methods: (a) Boston housing prices dataset and (b) automobile mileage per gallon dataset.

As represented in Figure 5 of both cases, the proposed training method shows the fastest and most stable convergence. The proposed method can achieve a better result than the final results of GDM, SDG, and ADAM just within 10 iterations. In order to analyze the reliability of the training performance, the training is performed with 20 cases obtained from a randomly chosen training dataset. The average and standard deviation values of the training error and the number of iterations required for convergence are summarized in Tables 3 and 4. The case study uses early stopping, where training stops at the point where the validation performance starts to degrade continuously over 15 iterations. It can be confirmed that the proposed approach shows overwhelming training performances, especially compared with GDM, SGD, and ADAM.

Table 3. Boston housing prices and automobile MPG training performances using early stopping (validation patient of 15)—training errors (MSE).

Method	Boston Housing Prices		Automobile MPG	
	Mean	Standard Deviation	Mean	Standard Deviation
GDM	Not converge	Not converge	Not converge	Not converge
SGD	1.127×10^{-3}	1.066×10^{-3}	1.405×10^{-3}	1.215×10^{-3}
ADAM	5.858×10^{-4}	4.368×10^{-4}	6.417×10^{-4}	7.005×10^{-5}
LM	1.001×10^{-4}	2.653×10^{-5}	2.477×10^{-4}	4.010×10^{-5}
Proposed	7.393×10^{-5}	1.649×10^{-5}	2.407×10^{-4}	2.990×10^{-5}

Table 4. Boston housing prices and automobile MPG training performances using early stopping (validation patient of 15)—number of iterations.

Method	Boston Housing Prices		Automobile MPG	
	Mean	Standard Deviation	Mean	Standard Deviation
GDM	Not converge	Not converge	Not converge	Not converge
SGD	4330	5123	2833	4437
ADAM	4097	2579	3156	5506
LM	29.9	2.4	30.7	3.3
Proposed	26.6	2.0	29.1	2.3

4. Conclusions

A novel supervisory training rule for shallow, fully connected neural networks (SFCNN) using local linearization and an analytic optimal solution for a linearized SFCNN is proposed. The proposed cost function includes a number of important terms to keep the training within a valid region of the linearization and to prevent the training from changing the input and output weights too much, solving problems, such as divergence, singularity, and the saturation phenomenon of the activation functions. The performance of the proposed approach is verified by applying it to a highly nonlinear pH process, the Boston housing price dataset, and the automobile mileage per gallon dataset. It shows a superior modeling performance with the smallest modeling error and the smallest iteration number required for convergence compared with the previous approaches for the case study.

This research took a shallow neural network with just one hidden layer into account for developing the training methods. Currently, the research area of machine learning is expanding to various fields, such as computer vision and natural language processing. In accordance with this purpose, a deep neural network with a large number of hidden layers is generally used. Thus, as a future work, the proposed method will be extended to DNN.

Author Contributions: Conceptualization, C.Y., D.R.Y. and S.W.S.; methodology, J.P.H. and S.W.S.; formal analysis, J.P.H. and C.G.I.; writing—original draft preparation, K.H.R. and S.W.S.; writing—review and editing, K.H.R. and S.W.S.; supervision, K.H.R. and S.W.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bhat, N.V.; Minderman, P.A.; McAvoy, T.; Wang, N.S. Modeling chemical process systems via neural computation. *IEEE Control Syst. Mag.* **1990**, *10*, 24–30. [[CrossRef](#)]
2. Bhat, N.; McAvoy, T.J. Use of neural nets for dynamic modeling and control of chemical process systems. *Comput. Chem. Eng.* **1990**, *14*, 573–582. [[CrossRef](#)]
3. Chen, S.; Billings, S.A.; Grant, P.M. Non-linear system identification using neural networks. *Int. J. Control* **1990**, *51*, 1191–1214. [[CrossRef](#)]
4. Fukuda, T.; Shibata, T. Theory and applications of neural networks for industrial control systems. *IEEE Trans. Ind. Electron.* **1992**, *39*, 472–489. [[CrossRef](#)]
5. Ydstie, B.E. Forecasting and control using adaptive connectionist networks. *Comput. Chem. Eng.* **1990**, *14*, 583–599. [[CrossRef](#)]
6. Savković-Stevanović, J. Neural networks for process analysis and optimization: Modeling and applications. *Comput. Chem. Eng.* **1994**, *18*, 1149–1155. [[CrossRef](#)]
7. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, *2*, 359–366. [[CrossRef](#)]
8. Henrique, H.M.; Lima, E.L.; Seborg, D.E. Model structure determination in neural network models. *Chem. Eng. Sci.* **2000**, *55*, 5457–5469. [[CrossRef](#)]
9. Boozarjomehry, R.B.; Svrcek, W.Y. Automatic design of neural network structures. *Comput. Chem. Eng.* **2001**, *25*, 1075–1088. [[CrossRef](#)]
10. Derks, E.; Buydens, L.M.C. Aspects of network training and validation on noisy data: Part 1. Training aspects. *Chemom. Intell. Lab. Syst.* **1998**, *41*, 171–184. [[CrossRef](#)]
11. Pan, Y.; Sung, S.W.; Lee, J.H. Data-based construction of feedback-corrected nonlinear prediction model using feedback neural networks. *Control Eng. Pract.* **2001**, *9*, 859–867. [[CrossRef](#)]
12. Lee, D.S.; Jeon, C.O.; Park, J.M.; Chang, K.S. Hybrid neural network modeling of a full-scale industrial wastewater treatment process. *Biotechnol. Bioeng.* **2002**, *78*, 670–682. [[CrossRef](#)] [[PubMed](#)]
13. Dogan, E.; Sengorur, B.; Koklu, R. Modeling biological oxygen demand of the Melen River in Turkey using an artificial neural network technique. *J. Environ. Econ. Manag.* **2009**, *90*, 1229–1235. [[CrossRef](#)] [[PubMed](#)]
14. Heo, S.; Lee, J.H. Parallel neural networks for improved nonlinear principal component analysis. *Comput. Chem. Eng.* **2019**, *127*, 1–10. [[CrossRef](#)]
15. Jawad, J.; Hawari, A.H.; Zaidi, S.J. Artificial neural network modeling of wastewater treatment and desalination using membrane processes: A review. *Chem. Eng. J.* **2021**, *419*, 129540. [[CrossRef](#)]
16. Li, Y.; Jia, M.; Han, X.; Bai, X.-S. Towards a comprehensive optimization of engine efficiency and emissions by coupling artificial neural network (ANN) with genetic algorithm (GA). *Energy* **2021**, *225*, 120331. [[CrossRef](#)]
17. Bakay, M.S.; Ağbulut, Ü. Electricity production based forecasting of greenhouse gas emissions in Turkey with deep learning, support vector machine and artificial neural network algorithms. *J. Clean. Prod.* **2021**, *285*, 125324. [[CrossRef](#)]
18. Cui, Z.; Wang, L.; Li, Q.; Wang, K. A comprehensive review on the state of charge estimation for lithium-ion battery based on neural network. *Int. J. Energy Res.* **2022**, *46*, 5423–5440. [[CrossRef](#)]
19. Dauphin, Y.N.; Pascanu, R.; Gulcehre, C.; Cho, K.; Ganguli, S.; Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Adv. Neural Inf. Process. Syst.* **2014**, *27*, 1–9.
20. Nesterov, Y. A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. *Dokl. Acad. Sci. USSR* **1983**, *269*, 543–547.
21. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
22. d’Ascoli, S.; Refinetti, M.; Biroli, G. Optimal learning rate schedules in high-dimensional non-convex optimization problems. *arXiv* **2022**, arXiv:2202.04509.
23. Van Der Smagt, P.P. Minimisation methods for training feedforward neural networks. *Neural Netw.* **1994**, *7*, 1–11. [[CrossRef](#)]
24. Begum, K.G.; Rao, A.S.; Radhakrishnan, T.K. Enhanced IMC based PID controller design for non-minimum phase (NMP) integrating processes with time delays. *ISA Trans.* **2017**, *68*, 223–234. [[CrossRef](#)]
25. Sagun, L.; Evci, U.; Guney, V.U.; Dauphin, Y.; Bottou, L. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv* **2017**, arXiv:1706.04454.
26. Yao, Z.; Gholami, A.; Lei, Q.; Keutzer, K.; Mahoney, M.W. Hessian-based analysis of large batch training and robustness to adversaries. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 1–11.
27. Oymak, S.; Soltanolkotabi, M. Toward moderate overparameterization: Global convergence guarantees for training shallow neural networks. *IEEE J. Sel. Areas Inf. Theory* **2020**, *1*, 84–105. [[CrossRef](#)]
28. Keskar, N.S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; Tang, P.T.P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv* **2016**, arXiv:1609.04836.
29. Li, C.; Farkhoor, H.; Liu, R.; Yosinski, J. Measuring the intrinsic dimension of objective landscapes. *arXiv* **2018**, arXiv:1804.08838.
30. Li, H.; Xu, Z.; Taylor, G.; Studer, C.; Goldstein, T. Visualizing the loss landscape of neural nets. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 1–11.
31. Draxler, F.; Veschgini, K.; Salmhofer, M.; Hamprecht, F. Essentially no barriers in neural network energy landscape. In Proceedings of the International Conference on Machine Learning PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 1309–1318.

32. Ghorbani, B.; Krishnan, S.; Xiao, Y. An investigation into neural net optimization via hessian eigenvalue density. In Proceedings of the International Conference on Machine Learning PMLR, Long Beach, CA, USA, 10–15 June 2019; pp. 2232–2241.
33. Granzol, D.; Garipov, T.; Vetrov, D.; Zohren, S.; Roberts, S.; Wilson, A.G. Towards understanding the true loss surface of deep neural networks using random matrix theory and iterative spectral methods. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2020.
34. Gilmer, J.; Ghorbani, B.; Garg, A.; Kudugunta, S.; Neyshabur, B.; Cardoze, D.; Dahl, G.E.; Nado, Z.; Firat, O. A Loss Curvature Perspective on Training Instabilities of Deep Learning Models. In Proceedings of the International Conference on Learning Representations, Virtual, 25 April 2022.
35. Sung, S.W.; Lee, T.; Park, S. Improved training rules for multilayered feedforward neural networks. *Ind. Eng. Chem. Res.* **2003**, *42*, 1275–1278. [[CrossRef](#)]
36. Sung, S.W.; Lee, J.; Lee, I.-B. *Process Identification and PID Control*; John Wiley & Sons: Hoboken, NJ, USA, 2009; ISBN 978-0-470-82410-8.
37. Yoo, C.K.; Sung, S.W.; Lee, I.-B. Generalized damped least squares algorithm. *Comput. Chem. Eng.* **2003**, *27*, 423–431. [[CrossRef](#)]
38. Sung, S.W.; Lee, I.-B.; Yang, D.R. pH Control using an identification reactor. *Ind. Eng. Chem. Res.* **1995**, *34*, 2418–2426. [[CrossRef](#)]
39. Harrison, D., Jr.; Rubinfeld, D.L. Hedonic housing prices and the demand for clean air. *J. Environ. Econ. Manag.* **1978**, *5*, 81–102. [[CrossRef](#)]
40. Quinlan, J.R. Combining instance-based and model-based learning. In Proceedings of the Tenth International Conference on Machine Learning, Amherst, MA, USA, 27–29 July 1993; pp. 236–243.