mathematics

MDPI

Article

Adaptive Hard Parameter Sharing Method Based on Multi-Task Deep Learning

Hongxia Wang, Xiao Jin, Yukun Du, Nan Zhang and Hongxia Hao *

School of Statistics and Data Science, Nanjing Audit University, Nanjing 211815, China; hxwang@nau.edu.cn (H.W.); mg2108123@stu.nau.edu.cn (X.J.); mg2108129@stu.nau.edu.cn (Y.D.); mg2108117@stu.nau.edu.cn (N.Z.)
* Correspondence: hxhao@nau.edu.cn

Abstract: Multi-task learning (MTL) improves the performance achieved on each task by exploiting the relevant information between tasks. At present, most of the mainstream deep MTL models are based on hard parameter sharing mechanisms, which can reduce the risk of model overfitting. However, negative knowledge transfer may occur, which hinders the performance improvement achieved for each task. In this paper, for situations when multiple tasks are jointly trained, we propose the adaptive hard parameter sharing method. On the basis of the adaptive hard parameter sharing method, the number of nodes in the network is dynamically updated by setting a continuous gradient difference-based sign threshold and a warm-up training iteration threshold through the relationships between the parameters and the loss function. After each task fully utilizes the shared information, adaptive nodes are used to further optimize each task, reducing the impact of negative migration. By using simulation studies and instance analyses, we demonstrate theoretical proof that the performance of the proposed method is better than that of the competing method.

Keywords: multi-task learning; continuous gradient difference threshold; warm-up; training iteration threshold; information sharing; adaptive nodes

MSC: 68T07; 68W99

check for updates

Citation: Wang, H.; Jin, X.; Du, Y.; Zhang, N.; Hao, H. Adaptive Hard Parameter Sharing Method Based on Multi-Task Deep Learning. *Mathematics* **2023**, *11*, 4639. https://doi.org/10.3390/math11224639

Academic Editor: Florin Leon

Received: 20 October 2023
Revised: 8 November 2023
Accepted: 9 November 2023
Published: 14 November 2023

Copyright: © 2023 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

Multi-task learning plays a significant role in various fields including environmental science, economics, and air pollution. It aims to utilize the effective information between tasks to enhance the learning performance of multiple related tasks [1]. Multitask learning offers a unique opportunity for the application of artificial intelligence (AI)-based machine learning models, particularly those utilizing neural networks. Groundbreaking studies by Gao et al. [2], Nieto et al. [3], and Park et al. [4] have already proven the efficacy of network models in this regard. Deep learning models, by discarding the traditional feature engineering process intrinsic to machine learning, provide a robust means to decipher complex nonlinear relationships, thus becoming a focal point of current research, as highlighted by LeCun et al. [5] and Schmidhuber et al. [6].

As a promising field in machine learning, the goal of MTL is to improve the performance achieved on multiple related learning tasks by using the useful information that is present between them [1]. In the era of deep learning, MTL can be transformed into a network that is designed to learn and share representations from multi-task supervision signals. If related tasks share complementary information or act as regularizers for each other, they may yield improved performance, as seen in the work of Vandenhende et al. [7]. For a given model, we use the correlations between multiple tasks to conduct joint training for these tasks to better profile our original tasks and improve the model's learning ability for multiple tasks by sharing the obtained information.

MTL based on deep learning can be traced back to 1993. Caruana [8] proposed the hard parameter sharing model for neural networks and proposed several possible mechanisms for MTL based on empirical research. Deep learning frameworks combined with MTL mainly include the following two types of methods: (1) The Hard parameter sharing method based on multi-task deep learning. Hard parameter sharing allows for multiple tasks to share certain model parameters, enjoying the benefit of reduced storage costs. The hard parameter sharing model shares all parameter information at each layer. For instance, Collobert and Weston [9] integrated tasks such as part-of-speech tagging, chunking, named entity recognition, and word similarity into a single language model, leveraging features automatically learned from other tasks to enhance the performance of semantic role labeling. Long et al. [10] proposed a multi-linear relationship network that shares the first five convolutional layers of AlexNet, with task-specific fully connected layers designated for different tasks. Ruder et al. [11] introduced a hard parameter sharing model for multi-task learning. Although the hard-sharing mechanism is beneficial in many scenarios, its efficacy diminishes when the connections between tasks are less tight. (2) The soft parameter sharing method based on multi-task deep learning. The soft parameter sharing mechanism requires consideration of task relevance. Compared to the hard parameter sharing mechanism, soft parameter sharing can achieve significant optimization effects even for tasks with looser connections [12]. Soft parameter sharing can fully leverage advantages in multi-task learning such as latent data augmentation, eavesdropping mechanisms, attention concentration, and regularization mechanisms [12]. The soft parameter sharing model can regularize the distance between model parameters to encourage parameter similarity. For example, Duong et al. [13] utilized the L_2 distance for regularization, while Misra et al. [14] introduced a cross-stitch network structure to learn linear combinations of relative position features from different tasks. Mrini et al. [15] proposed a progressive soft parameter sharing method for multi-task learning, using a weighted loss function to simultaneously optimize multiple tasks, thereby enhancing the performance of multiple tasks.

Deep learning has become the main method in many fields and is very powerful, but it is vulnerable to noise and abnormal tasks, which significantly reduce its performance. Zhang and Yang [16] proposed that it is advisable to design a flexible and robust deep multi-task model. At present, deep MTL still possesses some shortcomings. When multiple tasks are jointly trained, improving the performance of the model on one task damages the performance achieved on tasks with different needs; this phenomenon is called negative migration or destructive interference. Minimizing negative migration is a key objective of an MTL method. Based on this, we propose an MTL method with adaptive nodes derived from the joint losses of multiple tasks. The innovation of this method is that, when the joint training processes of multiple tasks reach certain conditions, the qualified nodes are copied, and the multi-task training process is combined with the single-task training procedure. While fully enjoying the benefits of joint multi-task training, our method dynamically updates the network structure, further optimizes each task, and then greatly reduces the impact of negative migration. Consequently, it makes the overall prediction results better while incurring a small time cost.

The remainder of this article is organized as follows. Section 2 is the main part of the paper. In this section, the new model framework, its theoretical value, and its specific algorithm are presented. In Section 3, we report some numerical results obtained on simulated data. Our method is applied to two real data instances in Section 4. In Section 5, we present the proofs of the main theorems. Conclusions are stated in Section 6.

2. Models and Methods

2.1. Adaptive Hard Parameter Sharing Model

Based on the hard parameter sharing model, this section proposes an adaptive hard parameter sharing model. The main idea is to convert the last shared layer of the neural

network into an adaptive node layer, where nodes can be copied to obtain better prediction results.

The hard parameter sharing model is shown in Figure 1. This model mainly includes three parts: an input layer, a shared layer, and an output layer. The adaptive hard parameter sharing model is shown in Figure 2. This model includes four layers: an input layer, a shared layer, an adaptive node layer, and an output layer.

In this section, we first introduce the concepts of single-task nodes and multi-task nodes. A single-task node refers to a node in the adaptive node layer that is connected with a single node in the output layer, and a multi-task node refers to a node in the adaptive node layer that is connected with multiple nodes in the output layer. As shown in Figure 2, $h_p^{(f)}$ and $h_p^{(f)*}$ are two single-task nodes, and $h_{j^{(f)}}^{(f)}$ is a multi-task node.

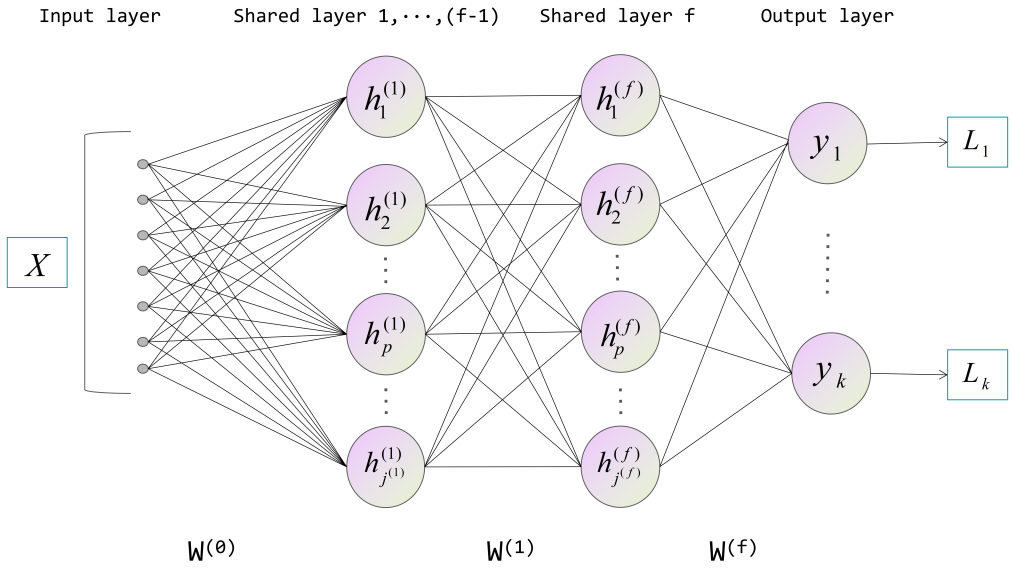


Figure 1. Hard parameter sharing model.

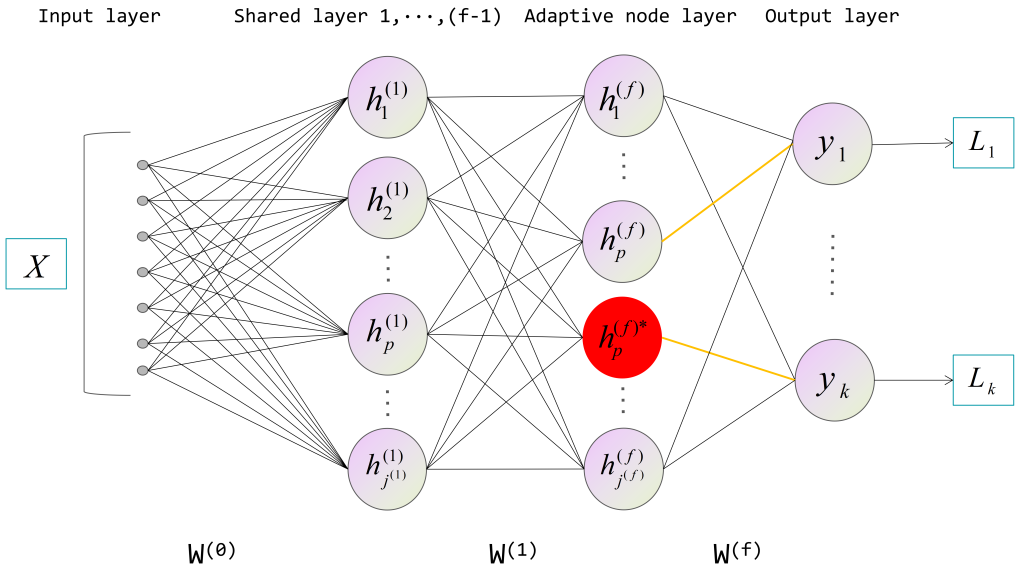


Figure 2. Adaptive hard parameter sharing model.

2.2. *Adaptive Hard Parameter Sharing Method*

Suppose that the given space is large enough to simultaneously contain solutions for multiple tasks. Assuming that there are k tasks, the output layer has k nodes in total.

This is recorded as $\boldsymbol{Y} = (y_1, y_2, \cdots, y_k)$. Shared layer $i (i = 1, 2, \cdots, f-1)$ has $j^{(i)}$ nodes, denoted as $\boldsymbol{H}^{(i)} = (h_1^{(i)}, h_2^{(i)}, \cdots, h_{j^{(f)}}^{(i)})$, and layer f is the adaptive node layer. There are $j^{(f)}$ nodes before the replication nodes, which is denoted as $\boldsymbol{H}^{(f)} = (h_1^{(f)}, h_2^{(f)}, \cdots, h_{j^{(f)}}^{(f)})$. Note that the weight matrix from shared layer $i-1$ to shared layer i is $\boldsymbol{W}^{(i-1)}$, $\boldsymbol{W}^{(i-1)} = (\boldsymbol{W}_1^{(i-1)}, \boldsymbol{W}_2^{(i-1)}, \cdots, \boldsymbol{W}_{j^{(i)}}^{(i-1)})$, $\boldsymbol{W}_p^{(i-1)} = (w_{p1}^{(i-1)}, w_{p2}^{(i-1)}, \cdots, w_{pj^{(i-1)}}^{(i-1)})^{\mathsf{T}} (p = 1, \ldots, j^{(i)})$, and the offset matrix is $\boldsymbol{b}^{(i-1)} = (b_1^{(i-1)}, b_2^{(i-1)}, \cdots, b_{j^{(i)}}^{(i-1)})$.

The gradient dissimilarity sign is the weight of a node $h_p^{(f)}$ in the adaptive node layer and the weight of the connection $\boldsymbol{W}_p^{(f-1)}$ in the upper shared layer.

Definition 1. *During the backpropagation process, an activation function is chosen whose value domain is contained in* $[0, +\infty)$ *or* $(-\infty, 0]$. *For* $\boldsymbol{W}_p^{(f-1)}$, $sign(\frac{\partial L_1}{\partial \boldsymbol{W}_p^{(f-1)}})$, $sign(\frac{\partial L_2}{\partial \boldsymbol{W}_p^{(f-1)}})$, $\cdots$, *and* $sign(\frac{\partial L_k}{\partial \boldsymbol{W}_p^{(f-1)}})$ *are not all equal, where L represents loss function. This situation is called gradient anisotropy.*

Theorem 1. *For each layer with activation function "sigmoid", for the same i, there is "*$sign(\frac{\partial L_i}{\partial w_{p1}^{(d-1)}}) = sign(\frac{\partial L_i}{\partial w_{p2}^{(d-1)}}) = sign(\frac{\partial L_i}{\partial w_{pj^{d-1}}^{(d-1)}})$*, " (*$d = 1, 2 \cdots, f-1$*) during the backpropagation process.*

For simplicity, a hard parameter sharing model with two tasks and two shared layers is used as an example of an improved model to illustrate our idea. The overall loss function for both tasks is as follows:

$$L = L_1 + L_2 \tag{1}$$

Figure 3 shows the hard parameter sharing model with two tasks and two shared layers. Figure 4 shows the adaptive hard parameter sharing model with two tasks and two shared layers.

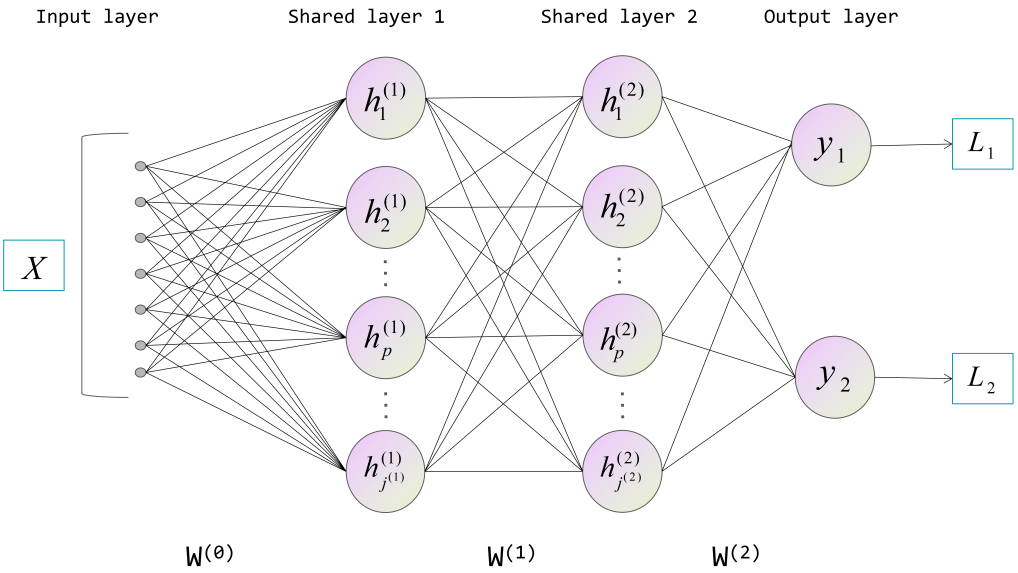


Figure 3. Hard parameter sharing model with two tasks and two shared layers.

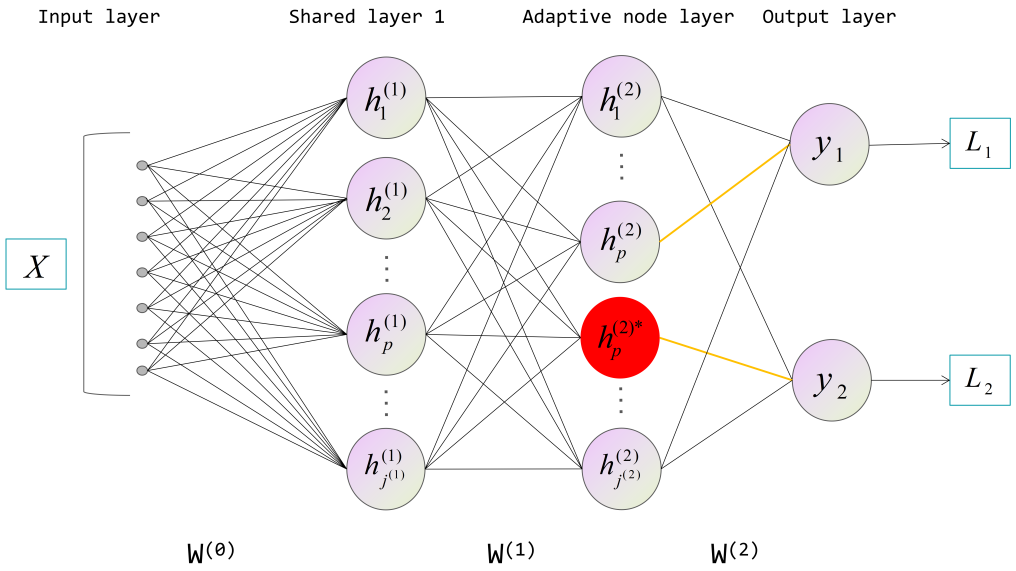


Figure 4. Adaptive hard parameter sharing model with two tasks and two shared layers.

During the backpropagation process, we use the last shared layer as the adaptive node layer of the adaptive hard parameter sharing model. The weight vector of the upper layer connected by nodes $h_p^{(2)}(p = 1, 2, \cdots, j^{(2)})$ in this layer is $\boldsymbol{W}_p^{(1)} = (w_{p1}^{(1)}, w_{p2}^{(1)}, \cdots, w_{pj^{(1)}}^{(1)})$. For the overall loss function, the gradient is calculated as follows:

$$\frac{\partial L}{\partial \boldsymbol{W}_p^{(1)}} = \frac{\partial L_1}{\partial \boldsymbol{W}_p^{(1)}} + \frac{\partial L_2}{\partial \boldsymbol{W}_p^{(1)}} = \frac{\partial L_1}{\partial y_1}\frac{\partial y_1}{\partial h_p^{(2)}}\frac{\partial h_p^{(2)}}{\partial \boldsymbol{W}_p^{(1)}} + \frac{\partial L_2}{\partial y_2}\frac{\partial y_2}{\partial h_p^{(2)}}\frac{\partial h_p^{(2)}}{\partial \boldsymbol{W}_p^{(1)}}, \tag{2}$$

where $\frac{\partial L_1}{\partial \boldsymbol{W}_p^{(1)}} > 0$, $\frac{\partial L_2}{\partial \boldsymbol{W}_p^{(1)}} < 0$, $\frac{\partial L}{\partial \boldsymbol{W}_p^{(1)}} > 0$.

We update the parameters using Adam, the formulas for which are as follows:

$$\boldsymbol{W}_p^{(1)} = \boldsymbol{W}_p^{(1)} - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + c}, \tag{3}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{4}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \tag{5}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{6}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{7}$$

where η denotes the learning rate, t signifies the update step, g_t indicates the gradient, m_t represents the exponential weighted moving average of g_t, v_t denotes the exponential weighted moving average of the squared g_t, $\hat{m}_t$ stands for the bias-corrected m_t, and $\hat{v}_t$ indicates the bias-corrected v_t, while c is a constant added to ensure numerical stability, typically set to 10^{-8}. β_1 and β_2 represent the exponential decay rates.

According to the Adam algorithm, for task y_1, since $\frac{\partial L_1}{\partial \boldsymbol{W}_p^{(1)}} > 0$ is the same as $\frac{\partial L}{\partial \boldsymbol{W}_p^{(1)}} > 0$, the loss function L_1 decreases. For task y_2, the loss function L_2 increases because $\frac{\partial L_2}{\partial \boldsymbol{W}_p^{(1)}} < 0$, while $\frac{\partial L}{\partial \boldsymbol{W}_p^{(1)}} > 0$. For the overall multi-task, the overall multi-task loss function $L = L_1 + L_2$ decreases.

Similarly, when $\frac{\partial L}{\partial \boldsymbol{W}_p^{(1)}} < 0$, for task y_1, the loss function L_1 increases, while for task y_2, the loss function L_2 decreases. The overall loss function L decreases.

To reduce the impact of the negative migration phenomenon in MTL, we replicate node $h_p^{(2)}$, and the new node is denoted as $h_p^{(2)*}$. We set the initial value of $\boldsymbol{W}_p^{(1)*}$ equal to $\boldsymbol{W}_p^{(1)}$. The connection weights of $h_p^{(2)}$ and the output layer are $w_{1p}^{(2)}$ and $w_{2p}^{(2)}$, respectively. After replication, the connection weight of $h_p^{(2)}$ from a multi-task node to a single-task node with the output layer is $w_{1p}^{(2)}$; $h_p^{(2)*}$ is also a single-task node, and its connection weight with the output layer is $w_{2p}^{(2)}$. The Adam optimization algorithm is also used to update the parameters related to the new model.

The lowest points of the two curves in Figure 5 represent the minimal value points of the loss functions of the two tasks. The green dashed line represents the initial values of the two task iterations, and the red dashed line represents the change in the loss function with respect to the parameters after starting the node-replication process. Assuming that the overall loss function of the adaptive hard parameter sharing model is $L^* = L_1^* + L_2^*$, the gradient of the loss function L_1^* over $\boldsymbol{W}_p^{(1)}$ for task y_1 is $\frac{\partial L_1^*}{\partial \boldsymbol{W}_p^{(1)}} = \frac{\partial L^*}{\partial \boldsymbol{W}_p^{(1)}}$, which is greater than the gradient $\frac{\partial L}{\partial \boldsymbol{W}_p^{(1)}}$ of the overall multi-task loss function L over $W_p^{(1)}$ in the original model due to the increase in the step size. The loss function L_1 decreases even more (see the red line in Figure 5). For task y_2, the loss function L_2^* decreases according to the Adam optimization algorithm (see the blue line in Figure 5). Theorem 2 shows that the overall loss function $L^* = L_1^* + L_2^*$ is smaller than L_1, and the Proof of Theorem 2 is given in Section 5.

Theorem 2. *If the number of consecutive gradient differences > the continuous gradient dissimilarity threshold φ and the number of iterations > the warm-up iteration threshold λ, then we obtain $L^* < L$.*

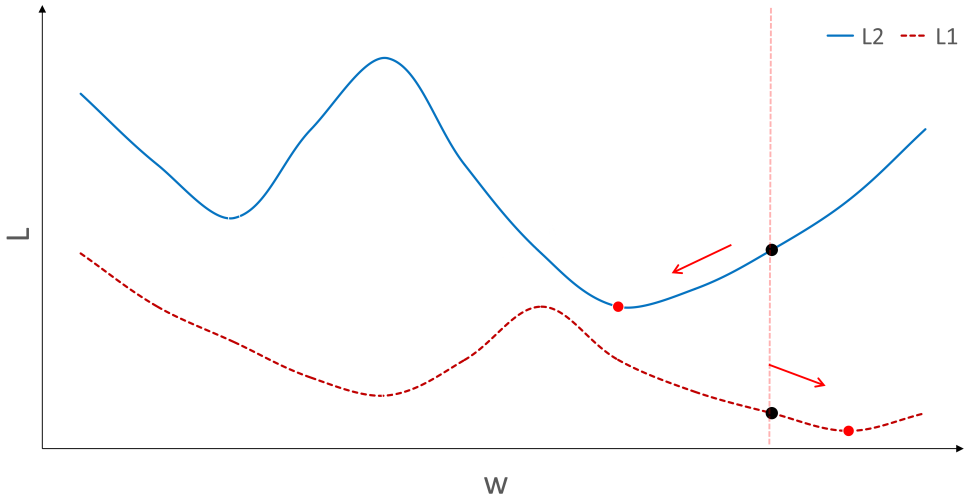


Figure 5. Variation in the loss function of the adaptive hard parameter sharing model with respect to the parameters.

Algorithm 1 describes the processes of constructing the adaptive hard parameter sharing method, setting the gradient anisotropy threshold φ and offsetting the warm-up iteration threshold λ in detail. T is the total number of iterations when the iteration terminates at round $T-1$, where $V^{(f-1)}$ equals the zero vector again.

The threshold value λ is set to prevent the model from performing node replication too early, and each task can make full use of the mutually useful information to learn and take advantage of MTL. The threshold value φ is set to prevent the model parameters

from falling into local optima. When continuous gradient dissimilarity occurs, the model parameters are monotonically constant with respect to the loss function for a long time. If the number of dissimilarities is not less than the threshold value and node replication is performed when the number of model warm-up training iterations satisfies the imposed condition, the probability that the end point of the iterative process is the global optimal solution is greater.

Algorithm 1 Adaptive hard parameter sharing algorithm

Require: Continuous gradient dissimilarity threshold φ, warm-up iteration threshold λ. Loss functions L_1, L_2, the adaptive node layer $\boldsymbol{H}^{(f)} = (h_1^{(f)}, \cdots, h_{j^{(f)}}^{(f)})$ and adaptive node count vector $\boldsymbol{V}^{(f)} = (v_1, \cdots, v_{j^{(f)}})^{\mathsf{T}} = \vec{\boldsymbol{0}}$

1: **for** $t = 1, ..., T$ **do**
2: **for** $i = 1, ..., j^{(i)}$ **do**
3: calculate gradient $\frac{\partial L_1}{\partial \boldsymbol{W}_i^{(f-1)}}$ and $\frac{\partial L_2}{\partial \boldsymbol{W}_i^{(f-1)}}$
4: If $sign(\frac{\partial L_1}{\partial \boldsymbol{W}_i^{(f-1)}}) \neq sign(\frac{\partial L_2}{\partial \boldsymbol{W}_i^{(f-1)}})$
let $v_i = v_i + 1$; otherwise, $v_i = 0$
5: If $t \geq \lambda$ and $v_i \geq \varphi$
copy node $h_i^{(f)}$; return to step 1
end
6: **end**
7: **return** $(\boldsymbol{y_1}, \boldsymbol{y_2})$

Remark 1. *All nodes of the adaptive node layer can be replicated, and $k (k > 2)$ tasks still apply. The nodes of each adaptive node layer are connected to at least one task and at most $j^{(n)} * k$ tasks. According to the specific actual situation and demands, different conditions and methods should be used for the heterogeneous gradient replication nodes.*

3. Simulation Studies

For the sake of simplicity, the following models are used in this section.

$$\boldsymbol{Y} = (\boldsymbol{X}\boldsymbol{W}^{(0)} + \boldsymbol{b}^{(0)})\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)} + \boldsymbol{\varepsilon}. \tag{8}$$

***scenario* 1**: $\boldsymbol{X} \sim U([0,1])^5$. $\boldsymbol{W}^{(0)}$ is a 5∗4-dimensional matrix, where $\boldsymbol{W}_{ij}^{(0)} \sim U[0,4]$, $i = 1,2,3,4,5$, $j = 1,2,3,4$. $\boldsymbol{W}^{(1)}$ is a 4∗2-dimensional matrix, where $\boldsymbol{W}_{ij}^{(1)} \sim U[0,4]$, $i = 1,2,3,4$, $j = 1,2$. $\boldsymbol{b}^{(0)}$ is a $n * 4$-dimensional matrix, where $\boldsymbol{b}_{ij}^{(0)} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2,3,4$. $\boldsymbol{b}^{(1)}$ is a $n * 2$-dimensional matrix, and $\boldsymbol{b}_{ij}^{(1)} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2$. $\boldsymbol{\varepsilon}$ is an $n * 2$-dimensional matrix, where $\varepsilon_{ij} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2$. $\boldsymbol{Y}$ is an $n * 2$-dimensional matrix.

***scenario* 2**: $\boldsymbol{X}$ is an n∗5-dimensional matrix, where $\boldsymbol{X}_{ij} \sim N(0,1)$, $i = 1,\cdots,n$, $j = 1,2,3,4,5$. $\boldsymbol{W}^{(0)}$ is a 5∗4-dimensional matrix, where $\boldsymbol{W}_{ij}^{(0)} \sim N(0,1)$, $i = 1,2,3,4,5$, $j = 1,2,3,4$. $\boldsymbol{W}^{(1)}$ is a 4∗2-dimensional matrix, where $\boldsymbol{W}_{ij}^{(1)} \sim N(0,1)$, $i = 1,2,3,4$, $j = 1,2$. $\boldsymbol{b}^{(0)}$ is a n∗4-dimensional matrix, where $\boldsymbol{b}_{ij}^{(0)} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2,3,4$. $\boldsymbol{b}^{(1)}$ is an $n * 2$-dimensional matrix, and $\boldsymbol{b}_{ij}^{(1)} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2$. $\boldsymbol{\varepsilon}$ is an $n * 2$-dimensional matrix, where $\varepsilon_{ij} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2$. $\boldsymbol{Y}$ is an $n * 2$-dimensional matrix.

***scenario* 3**: $\boldsymbol{X}$ is an $n * 5$-dimensional matrix, where $\boldsymbol{X}_{ij} \sim E(10)$, $i = 1,\cdots,n$, $j = 1,2,3,4,5$. $\boldsymbol{W}^{(0)}$ is a 5∗4-dimensional matrix, where $\boldsymbol{W}_{ij}^{(0)} \sim N(0,1)$, $i = 1,2,3,4,5$, $j = 1,2,3,4$. $\boldsymbol{W}^{(1)}$ is a 4∗2-dimensional matrix, where $\boldsymbol{W}_{ij}^{(1)} \sim N(0,1)$, $i = 1,2,3,4$, $j = 1,2$.

$\boldsymbol{b}^{(0)}$ is an $n*4$-dimensional matrix, where $\boldsymbol{b}_{ij}^{(0)} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2,3,4$. $\boldsymbol{b}^{(1)}$ is an $n*2$-dimensional matrix, and $\boldsymbol{b}_{ij}^{(1)} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2$. $\boldsymbol{\varepsilon}$ is an n*2-dimensional matrix, where $\varepsilon_{ij} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2$. $\boldsymbol{Y}$ is an $n*2$-dimensional matrix.

***scenario* 4**: $\boldsymbol{X}$ is an $n*5$-dimensional matrix, where $\boldsymbol{X}_{ij} \sim F(5,10)$, $i = 1,\cdots,n$, $j = 1,2,3,4,5$. $\boldsymbol{W}^{(0)}$ is a 5∗4-dimensional matrix, where $\boldsymbol{W}_{ij}^{(0)} \sim N(0,1)$, $i = 1,2,3,4,5$, $j = 1,2,3,4$. $\boldsymbol{W}^{(1)}$ is a 4∗2-dimensional matrix, where $\boldsymbol{W}_{ij}^{(1)} \sim N(0,1)$, $i = 1,2,3,4$, $j = 1,2$. $\boldsymbol{b}^{(0)}$ is an n∗4-dimensional matrix, where $\boldsymbol{b}_{ij}^{(0)} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2,3,4$. $\boldsymbol{b}^{(1)}$ is an n∗2-dimensional matrix, and $\boldsymbol{b}_{ij}^{(1)} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2$. $\boldsymbol{\varepsilon}$ is an n∗2-dimensional matrix, where $\varepsilon_{ij} \sim N(0,1)$, $i = 1,2,\cdots,n$, $j = 1,2$. $\boldsymbol{Y}$ is an n*2-dimensional matrix.

The algorithm program constructed in this paper includes the following. The *CPU* is an r7-4800H, the GPU is an *RTX*2060, the operating system is 64-bit Windows, and the development language is Python. All methods are implemented based on the Pytorch framework. The numbers of neurons in the input layer, shared layer, adaptive node layer, and output layer are set to 5, 4, 3 and 2, respectively.

For scenario 1, scenario 2, scenario 3, and scenario 4, the optimal hyperparameters can be obtained by adjusting the parameters as shown in Table 1.

Table 1. Hyperparameters obtained for four scenarios.

Hyperparameters	Uniform	Standard Normal	Exponential	F Distribution
φ_1	6	6	6	7
φ_2	6	7	5	9
φ_3	4	8	8	5
λ_1	50	50	50	60
λ_2	20	20	25	30
λ_3	10	20	15	20

Here, φ_1 is the continuous gradient dissimilarity threshold for the primary replicated node of the hard parameter sharing model. φ_2 is the continuous gradient anisotropy threshold for the secondary replicated node of the adaptive hard parameter sharing model of the primary replicated node. φ_3 is the adaptive hard parameter sharing model of the replicated secondary node; it provides the tertiary replication node with consecutive gradient anisotropy thresholds. λ_1 is the hard parameter sharing model for the primary replication node with a warm-up training iteration threshold. λ_2 is the adaptive hard parameter sharing model of replicated primary node that implements the warm-up training iteration threshold for the secondary replicated node. λ_3 is the adaptive hard parameter sharing model of the replicated secondary node; it provides the warm-up training iteration threshold for the tertiary replicated node.

For the above scenarios, separate comparison experiments are set up with the hard parameter sharing model, and to ensure the validity of the comparison, the total number of iterations of the original model is equal to the cumulative total number of iterations of the new model before and after each node replication step.

To assess how well our method performs, we compare the hard parameter sharing model with the adaptive hard parameter sharing model. To ensure the validity of the comparison, the total number of iterations of the original model is equal to the cumulative total number of iterations before and after each replication of the new model.

For the above four simulation scenarios, the sample size n is set to 1000, 2000, and 3000. The overall datasets are randomly divided into a training set and testing set with a ratio of 7:3, and part of the data in the training set are used as the validation set to select

the optimal model. The test set is used to evaluate the performance and robustness of the model, and the mean values of the results obtained over 100 rounds are used as the experimental results.

The performance improvement value p is calculated as follows:

$$p = \frac{h - a}{h} \times 100\%, \tag{9}$$

where h denotes the mean MSE for the hard parameter sharing model over 100 simulations and a denotes the mean MSE for the adaptive hard parameter sharing model over 100 simulations.

Table 2 shows the results of the comparison experiments for scenario 1, scenario 2, scenario 3, and scenario 4. As the sample size increases, the mean MSE of both models decrease. This can be seen from the p values, which exceed 13%, thus demonstrating the superior performance of the adaptive hard parameter sharing model in these four scenarios.

Table 2. The mean MSE (10^{-4}) and performance improvements (%) for both models over 100 simulations.

	$n = 1000$			$n = 2000$			$n = 3000$		
	h	a	p	h	a	p	h	a	p
Uniform	0.314	0.224	28.7	0.284	0.184	35.2	0.258	0.142	44.9
Standard normal	1.604	1.382	13.8	1.230	1.038	15.6	0.894	0.764	14.5
Exponential	0.627	0.483	22.9	0.554	0.436	21.3	0.526	0.418	20.5
F distribution	2.763	2.186	20.9	2.368	1.895	20.0	1.962	1.650	15.9

Table 3 represents the elapsed times required for 1000 samples by both models over 100 simulations. From Table 3, the average elapsed times required by the adaptive hard parameter sharing model on the three simulated datasets are within 9 s of those of the hard parameter sharing model. It is easy to see that the overall performance of the adaptive hard parameter sharing model is better than that of the hard parameter sharing model, indicating that the new model has good prediction and estimation capabilities compared to those of the original model, and it incurs a small time cost.

Table 3. Average elapsed times (s) required for 1000 samples by both models over 100 simulations.

	h	a
Uniform	85	92
Standard normal	97	102
Exponential	89	93
F distribution	102	111

Figures 6–9 are the MSE comparisons between the adaptive hard parameter sharing model and the hard parameter sharing model with a sample size of 1000 for the four scenarios. The first three marker points in Figures 6–9 represent the first, second, and third node replications of the adaptive hard parameter sharing model, respectively. The last two marker points represent the lowest MSE of the two models. From Figures 6–9, it can be seen that the MSE of the new model is always lower than that of the original model during the iterative process, which illustrates the superior performance of the adaptive hard parameter sharing model.

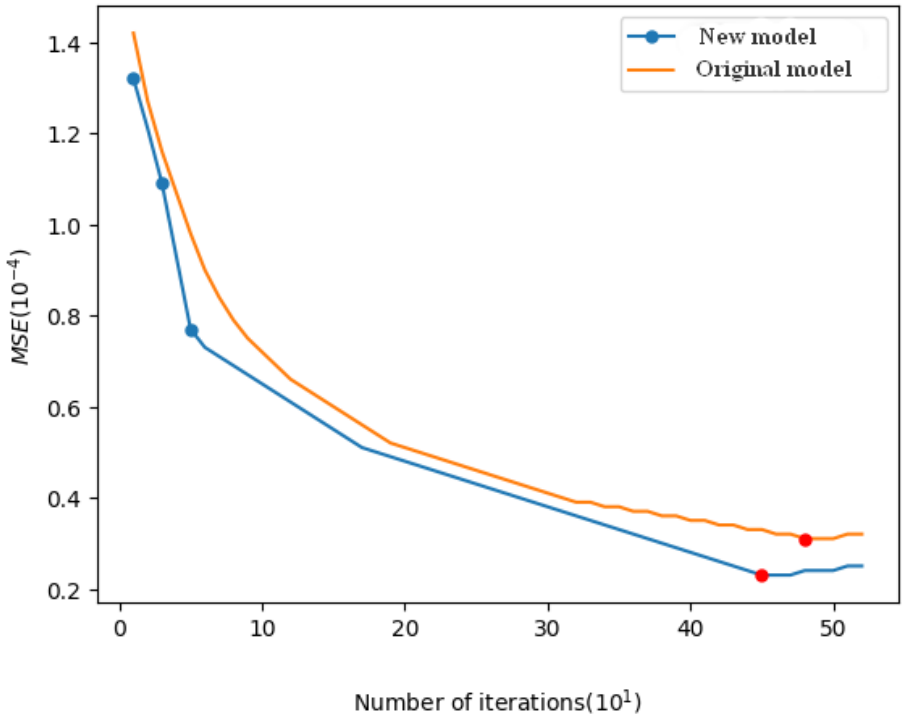


Figure 6. *MSE* comparison between the new model and the original model for scenario 1.

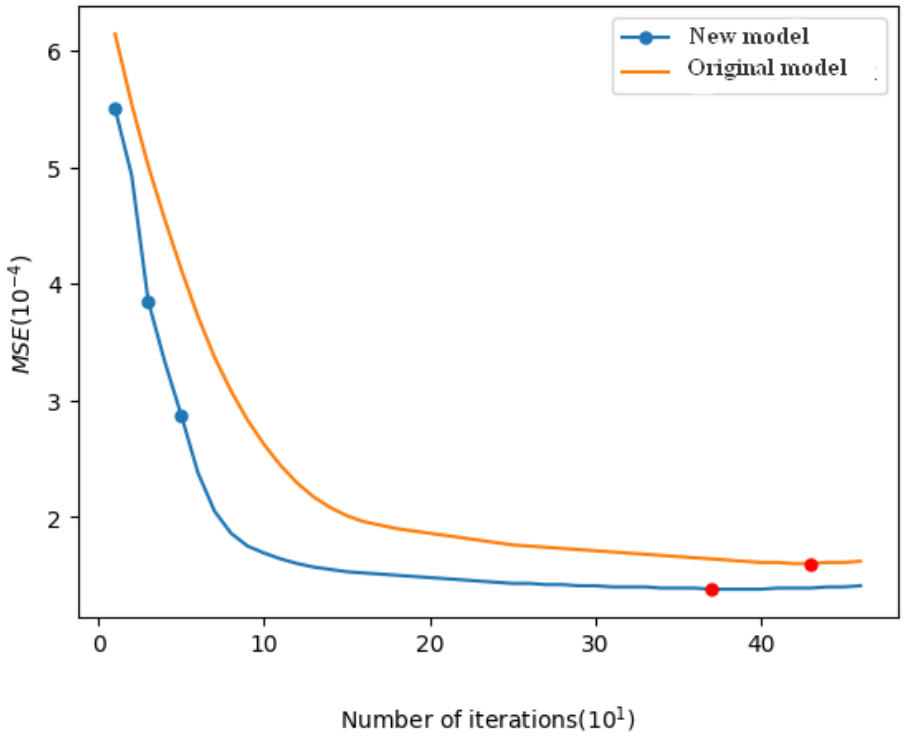


Figure 7. *MSE* comparison between the new model and the original model for scenario 2.

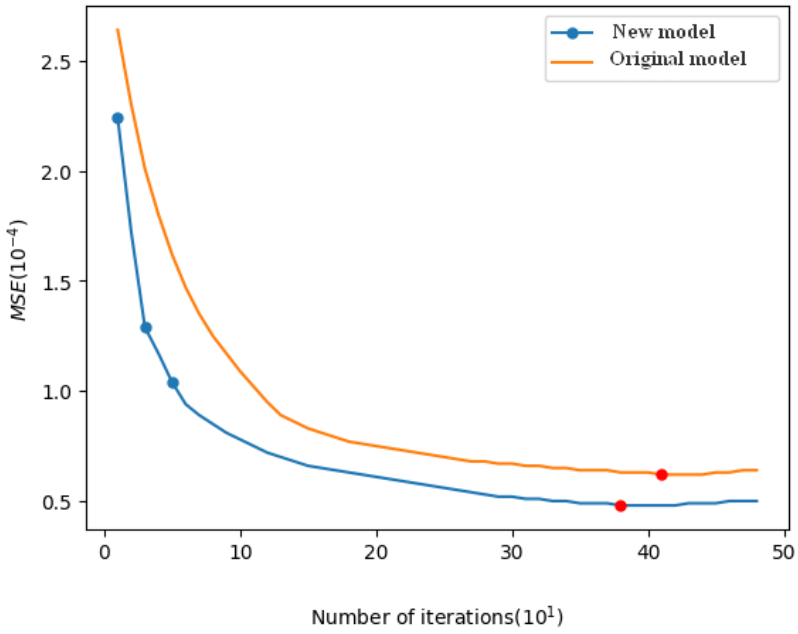


Figure 8. *MSE* comparison between the new model and the original model for scenario 3.

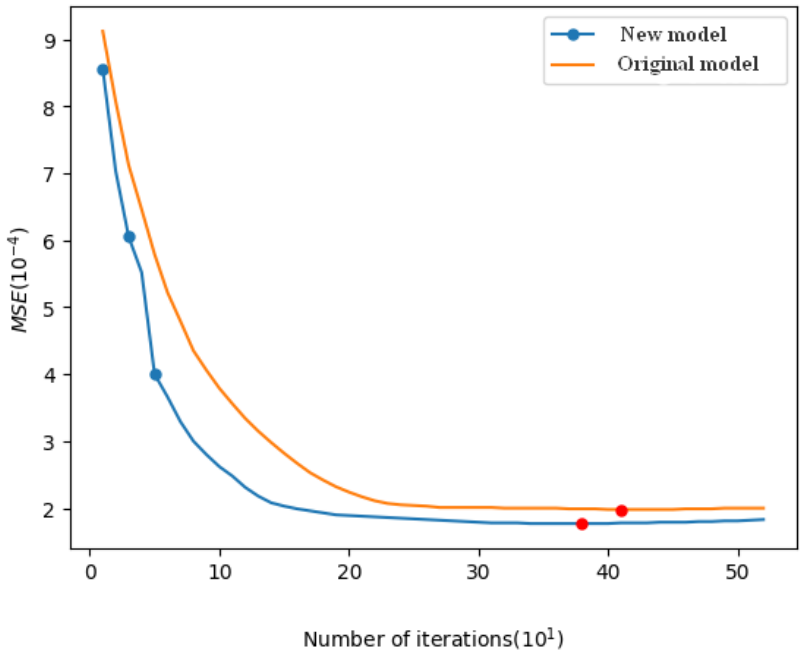


Figure 9. *MSE* comparison between the new model and the original model for scenario 4.

4. Instance Analyses

4.1. Two Tasks Instance

This section considers the air pollution data of Nanjing, which are obtained from the Chinese meteorological data network and the true air environmental data center. The data contain the $PM_{2.5}$ concentration ($\mu g/m^3$), PM_{10} concentration (mg/m^3), CO concentration (mg/m^3), SO_2 concentration ($\mu g/m^3$), NO_2 concentration ($\mu g/m^3$), and $O3_8h$ ($\mu g/m^3$) for each day from 2 December 2013 to 23 April 2023, considering the coordinated control of $PM_{2.5}$ and $O3_8h$ ($\mu g/m^3$). We set the $PM_{2.5}$ concentration and $O3_8h$ ($\mu g/m^3$) as the outputs y_1 and y_2, respectively. The PM_{10} concentration, CO concentration, SO_2 concentration, and NO_2 concentration are input variables x_1, x_2, x_3, and x_4, respectively. The sample size n is 3454.

The air pollution prediction model is as follows:

$$\boldsymbol{Y} = \sigma(\sigma(\sigma(\boldsymbol{X}\boldsymbol{W}^{(0)} + \boldsymbol{b}^{(0)})\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)})\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}) + \boldsymbol{\varepsilon}, \tag{10}$$

where the input $\boldsymbol{X}$ is a 3454∗4-dimensional matrix, $\boldsymbol{W}^{(0)}$ is a 4∗4-dimensional matrix, $\boldsymbol{W}^{(1)}$ is a 4∗3-dimensional matrix, $\boldsymbol{W}^{(2)}$ is a 3∗2-dimensional matrix, $\boldsymbol{b}^{(0)}$ is a 3454∗4-dimensional matrix, $\boldsymbol{b}^{(1)}$ is a 3454∗3-dimensional matrix, and $\boldsymbol{b}^{(2)}$ is a 3454∗2-dimensional matrix. $\boldsymbol{\varepsilon}$ is a 3454∗2-dimensional matrix. σ is the sigmoid activation function. The output $\boldsymbol{Y}$ is a 3454∗2-dimensional matrix.

For the air pollution data in Nanjing, the optimal hyperparameters can be obtained by adjusting the parameters as shown in Table 4.

Table 4. Hyperparameters obtained for air pollution data.

Hyperparameters	Air Pollution
φ_1	15
φ_2	8
φ_3	7
λ_1	70
λ_2	20
λ_3	20

The *MSE* of the adaptive hard parameter sharing model and the hard parameter sharing model of air pollution are given in Table 5. The adaptive hard parameter sharing model is 9.1 percentage points more effective than the hard parameter sharing model. The time required by the adaptive hard parameter sharing model is 321 s, and the time required by the hard parameter sharing model is 310 s; thus, the new model takes 11 s longer than the original model.

Figure 10 shows the *MSE* comparison between the adaptive hard parameter sharing model and the hard parameter sharing model for air pollution data in Nanjing. The meaning of the marker points in Figure 10 is the same as that in the simulation scenarios. The difference is that this figure shows the *MSE* change trends of the two models before the first node replication, which are essentially the same. Figure 10 shows that the *MSE* of the new model is almost always lower than that of the original model during the iterative process, which confirms the good performance of the new model.

Table 5. MSE values (10^{-2}), elapsed times (s), and performance improvements (%) of both models.

	MSE	***Time***	***P***
Original model	1.3829	310	9.1
New model	1.2584	321	

4.2. Three Tasks Instance

This section considers the student performance data obtained from the Kaggle website. The data contain information on the math performance of high school students, including their grades and demographic information. The data were collected from three high schools in the United States and include seven variables: race/ethnicity, parental level of education, lunch, test preparation course, math score, reading score, and writing score. The math score, reading score, and writing score are the output variables y_1, y_2, and y_3, respectively, while the race/ethnicity, parental level of education, lunch, and test preparation course are the input variables x_1, x_2, x_3, and x_4, respectively. The sample size n is 1000.

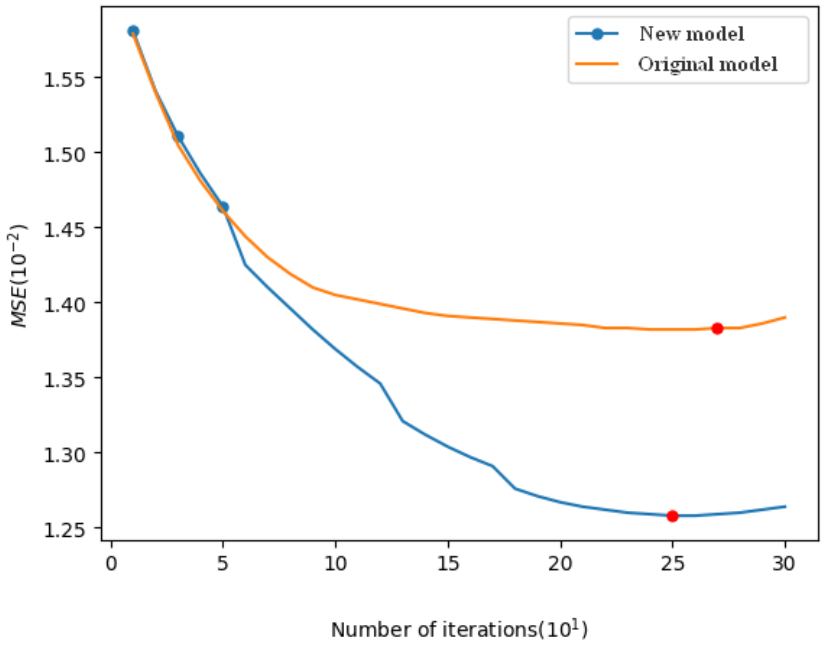


Figure 10. *MSE* comparison between the new model and the original model for air pollution data in Nanjing.

The student performance prediction model is as follows:

$$\boldsymbol{Y} = \sigma(\sigma(\sigma(\boldsymbol{X}\boldsymbol{W}^{(0)} + \boldsymbol{b}^{(0)})\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)})\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}) + \boldsymbol{\varepsilon}, \tag{11}$$

where the input $\boldsymbol{X}$ is a 1000∗4-dimensional matrix, $\boldsymbol{W}^{(0)}$ is a 4∗4-dimensional matrix, $\boldsymbol{W}^{(1)}$ is a 4∗4-dimensional matrix, $\boldsymbol{W}^{(2)}$ is a 4∗3-dimensional matrix, $\boldsymbol{b}^{(0)}$ is a 1000∗4-dimensional matrix, $\boldsymbol{b}^{(1)}$ is a 1000∗4-dimensional matrix, and $\boldsymbol{b}^{(2)}$ is a 1000∗3-dimensional matrix. $\boldsymbol{\varepsilon}$ is a 1000∗3-dimensional matrix. σ is the sigmoid activation function. The output $\boldsymbol{Y}$ is a 1000∗3-dimensional matrix.

For the student performance data, the optimal hyperparameters can be obtained by adjusting the parameters as shown in Table 6.

Table 6. Hyperparameters obtained for student performance data.

Hyperparameters	Student Performance
φ_1	14
φ_2	6
φ_3	6
λ_1	65
λ_2	30
λ_3	35

The *MSE* of the adaptive hard parameter sharing model and the hard parameter sharing model of student performance are given in Table 7. The adaptive hard parameter sharing model is 16.9 percentage points more effective than the hard parameter sharing model. The time required by the adaptive hard parameter sharing model is 120 s, and the time required by the hard parameter sharing model is 111 s; thus, the new model takes 9 s longer than the original model.

Table 7. MSE values (10^{-3}), elapsed times (s), and performance improvements (%) of both models.

	MSE	*Time*	*P*
Original model	6.862	111	16.9
New model	5.703	120	

Figure 11 shows the *MSE* comparison between the adaptive hard parameter sharing model and the hard parameter sharing model for student performance in the United States.The meaning of the marker points in Figure 11 is the same as that in the simulation scenarios. The difference is that this figure shows the *MSE* change trends of the two models before the first node replication, which are essentially the same. Figure 11 shows that the *MSE* of the new model is almost always lower than that of the original model during the iterative process, which confirms the good performance of the new model.

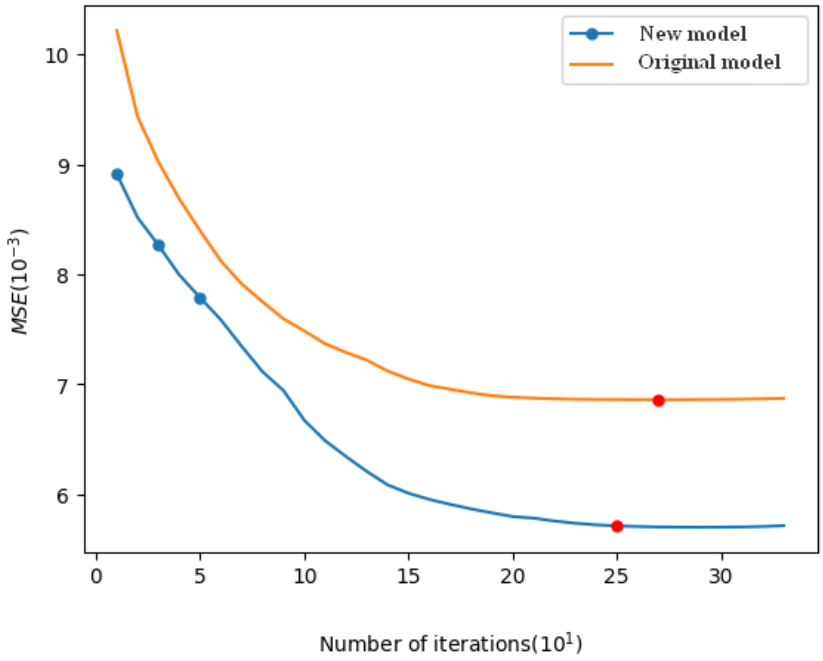


Figure 11. *MSE* comparison between the new model and the original model for student performance.

5. Proof

In this section, we give the Proofs of Theorems 1 and 2.

Proof of Theorem 1. Assume that there are $f+1$ layers and k tasks in the hard parameter sharing model. $\boldsymbol{H}^{(i)} = (h_1^{(i)}, h_2^{(i)}, \cdots, h_{j^{(i)}}^{(i)})$ represents the output vector of $j^{(i)}$ nodes in layer i, and $\boldsymbol{W}_p^{(i-1)} = (w_{p1}^{(i-1)}, w_{p2}^{(i-1)}, \cdots, w_{pj^{(i-1)}}^{(i-1)})^{\mathsf{T}}$ represents the weight vector connecting the p nodes in layers i and $i-1$.

During the forward propagation process,

$$\boldsymbol{Z}^{(1)} = \boldsymbol{X} \cdot \boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}, \tag{12}$$

$$\boldsymbol{H}^{(1)} = \sigma(\boldsymbol{Z}^{(1)}), \tag{13}$$

$$\boldsymbol{Z}^{(2)} = \boldsymbol{H}^{(1)} \cdot \boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}, \tag{14}$$

$$\boldsymbol{H}^{(2)} = \sigma(\boldsymbol{Z}^{(2)}), \tag{15}$$

$$\cdots$$

$$\widehat{\boldsymbol{Y}} = \boldsymbol{H}^{(f)} = \sigma(\boldsymbol{Z}^{(f)}) = \sigma(\boldsymbol{H}^{(f-1)} \cdot \boldsymbol{W}^{(f-1)} + \boldsymbol{b}^{(f-1)}). \tag{16}$$

Loss function:

$$L = \sum_{i=1}^{k} L_i. \tag{17}$$

Since the support of the sigmoid function is $(0,1)$, for any $i(i = 1,2,\cdots,n)$ and $j(j = 1,2,\cdots,j^{(i)})$, $h_{j^{(i)}}^{(i)} > 0$.

During the backpropagation process, according to the chain derivation rule, $\boldsymbol{W}_p^{(d-1)} = (w_{p1}^{(d-1)}, w_{p2}^{(d-1)}, \cdots, w_{pj^{(d-1)}}^{(d-1)})^{\mathsf{T}}$ is the connected weight vector between the upper layer and the p-th node of the d-th layer. $j^{(d-1)}$ is the number of nodes of layer $d-1$ in the model. The gradient of the loss function L_i of task i is

$$\bigtriangledown w_p^{(d-1)} = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial \boldsymbol{Z}^{(n)}} \cdot \frac{\partial \boldsymbol{Z}^{(n)}}{\partial \boldsymbol{H}^{(n-1)}} \cdot \frac{\partial \boldsymbol{H}^{(n-1)}}{\partial \boldsymbol{Z}^{(n-1)}} \cdots \frac{\partial \boldsymbol{H}^{(d+1)}}{\partial \boldsymbol{Z}^{(d+1)}} \cdot \frac{\partial \boldsymbol{Z}^{(d+1)}}{\partial h_p^{(d)}} \cdot \frac{\partial h_p^{(d)}}{\partial z_p^{(d)}} \cdot \frac{\partial z_p^{(d)}}{\partial w_p^{(d-1)}}. \tag{18}$$

Set

$$a = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial \boldsymbol{Z}^{(n)}} \cdot \frac{\partial \boldsymbol{Z}^{(n)}}{\partial \boldsymbol{H}^{(n-1)}} \cdot \frac{\partial \boldsymbol{H}^{(n-1)}}{\partial \boldsymbol{Z}^{(n-1)}} \cdots \frac{\partial \boldsymbol{H}^{(d+1)}}{\partial \boldsymbol{Z}^{(d+1)}} \cdot \frac{\partial \boldsymbol{Z}^{(d+1)}}{\partial h_p^{(d)}} \cdot \frac{\partial h_p^{(d)}}{\partial z_p^{(d)}}, \tag{19}$$

then,

$$\begin{aligned} \bigtriangledown w_p^{(d-1)} &= a(\frac{\partial z_p^{(d)}}{\partial w_{p1}^{(d-1)}}, \frac{\partial z_p^{(d)}}{\partial w_{p2}^{(d-1)}}, \cdots, \frac{\partial z_p^{(d)}}{\partial w_{pj^{(d-1)}}^{(d-1)}})^{\mathsf{T}} \\ &= a(h_1^{(d-1)}, h_2^{(d-1)}, \cdots, h_{j^{(d-1)}}^{(d-1)})^{\mathsf{T}} \\ &= a(\sigma(z_1^{(d-1)}), \sigma(z_2^{(d-1)}), \cdots, \sigma(z_{j^{(d-1)}}^{(d-1)}))^{\mathsf{T}}. \end{aligned}$$

Thus, $sign(\frac{\partial L_i}{\partial w_{p1}^{(d-1)}}) = sign(\frac{\partial L_i}{\partial w_{p2}^{(d-1)}}) = sign(\frac{\partial L_i}{\partial w_{pj^{(d-1)}}^{(d-1)}}) = sign(a)$, and the proof of Theorem 1 is completed. □

Remark 2. *This proof is applicable to sigmoid activation functions. It is also applicable to activation functions whose value range includes [0,+∞) or (−∞, 0], such as the rectified linear unit (relu) and softmax functions.*

Proof of Theorem 2. We take two tasks as examples, where layer $f+1$ is the output layer and layer f is the adaptive node layer.

For the hard parameter sharing network, the overall loss function is $L = L_1 + L_2$.

$\boldsymbol{W}^{(f-1)} = (\boldsymbol{W}_1^{(f-1)}, \boldsymbol{W}_2^{(f-1)}, \cdots, \boldsymbol{W}_{j^{(f-1)}}^{(f-1)})$ denotes the weight matrix of layer $f-1$ connected to layer f, where $\boldsymbol{W}_1^{(f-1)} = (w_{i1}^{(f-1)}, w_{i2}^{(f-1)}, \cdots, w_{ij^{(f-1)}}^{(f-1)})^{\mathsf{T}}$ denotes the weight vector of the $i-th$ node in layer $f-1$ connected to layer f. The number of nodes in layer $f-1$ is $j^{(i-1)}$, and the node vector of the adaptive node layer is denoted as $H^{(f-1)} = (h_1^{(f-1)}, h_2^{(f-1)}, \cdots, h_{j^{(f-1)}}^{(f-1)})$.

The gradient of the overall two tasks loss function L for the parameter vector $\boldsymbol{W}_i^{(f-1)}$ is

$$\frac{\partial L}{\partial \boldsymbol{W}_i^{(f-1)}} = \frac{\partial L_1}{\partial y_1}\frac{\partial y_1}{\partial {h_i}^{(f)}}\frac{\partial {h_i}^{(f)}}{\partial \boldsymbol{W}_i^{(f-1)}} + \frac{\partial L_2}{\partial y_2}\frac{\partial y_2}{\partial {h_i}^{(f)}}\frac{\partial {h_i}^{(f)}}{\partial \boldsymbol{W}_i^{(f-1)}}. \tag{20}$$

The gradient of the loss function L_1 of task y_1 with respect to the parameter vector $\boldsymbol{W}_i^{(f-1)}$ is

$$\frac{\partial L}{\partial \boldsymbol{W}_i^{(f-1)}} = \frac{\partial L_1}{\partial y_1}\frac{\partial y_1}{\partial {h_i}^{(f)}}\frac{\partial {h_i}^{(f)}}{\partial \boldsymbol{W}_i^{(f-1)}}. \tag{21}$$

The gradient of the loss function L_2 of task y_2 with respect to parameter $\boldsymbol{W}_i^{(f-1)}$ is

$$\frac{\partial L_2}{\partial \boldsymbol{W}_i^{(f-1)}} = \frac{\partial L_2}{\partial y_2}\frac{\partial y_2}{\partial {h_i}^{(f)}}\frac{\partial {h_i}^{(f)}}{\partial \boldsymbol{W}_i^{(f-1)}}. \tag{22}$$

We update the parameter $\boldsymbol{W}_i^{(f-1)}$ concerning the overall two tasks:

$$\boldsymbol{W}_i^{(f-1)} = \boldsymbol{W}_i^{(f-1)} - \eta\frac{\partial L}{\partial \boldsymbol{W}_i^{(f-1)}}. \tag{23}$$

By the condition $\frac{\partial L}{\partial \boldsymbol{W}_i^{(f-1)}} > 0$, $\boldsymbol{W}_i^{(f-1)}$ iterates in the negative gradient direction, the value of $\frac{\partial L}{\partial \boldsymbol{W}_i^{(f-1)}}$ decreases, and then the overall loss function L decreases according to the principle of the gradient descent algorithm.

For task y_1, since $\frac{\partial L_1}{\partial \boldsymbol{W}_i^{(f-1)}} > 0$, the update direction L_1 for parameter $\boldsymbol{W}_i^{(f-1)}$ is the same as the update direction of the overall loss function L for parameter $\boldsymbol{W}_i^{(f-1)}$. Therefore, L_1 decreases.

For task y_2, since $\frac{\partial L_2}{\partial \boldsymbol{W}_i^{(f-1)}} < 0$, the update direction of this task loss function L_2 for parameter $\boldsymbol{W}_i^{(f-1)}$ is opposite to the update direction of the overall loss function L for parameter $\boldsymbol{W}_i^{(f-1)}$. Therefore, L_2 rises.

Assume that L_1 changes by d_1 units and that L_2 increases d_2 units; then, L_1 decreases $(d_1 - d_2)$ units.

After replicating node $h_i^{(f)}$, the new node $h_i^{(f)*}$ is a single-task node associated with task y_2, and the original node $h_i^{(f)}$ is a single-task node associated with task y_1. We set $\boldsymbol{W}_i^{(f-1)(1)} = \boldsymbol{W}_i^{(f-1)(2)} = \boldsymbol{W}_i^{(f-1)}$ for the overall loss function $L^* = L_1^* + L_2^*$ of the adaptive hard parameter sharing model, where $\boldsymbol{W}_i^{(f-1)(1)}$ is the parameter vector associated with task y_1, and the gradient of the loss function L_1^* of y_1 with respect to $\boldsymbol{W}_i^{(f-1)(1)}$ is

$$\frac{\partial L_1^*}{\partial \boldsymbol{W}_i^{(f-1)(1)}} = \frac{\partial {L_1}^*}{\partial y_1}\frac{\partial y_1}{\partial {h_i}^{(f)}}\frac{\partial {h_i}^{(f)}}{\partial \boldsymbol{W}_i^{(f-1)(1)}}. \tag{24}$$

$\boldsymbol{W}_i^{(f-1)(2)}$ is the parameter vector associated with task y_2, and the gradient of the loss function L_2^* of y_2 with respect to $\boldsymbol{W}_i^{(f-1)(2)}$ is

$$\frac{\partial L_2^*}{\partial \boldsymbol{W}_i^{(f-1)(2)}} = \frac{\partial {L_2}^*}{\partial y_2}\frac{\partial y_2}{\partial {h_i}^{(f)}}\frac{\partial {h_i}^{(f)}}{\partial \boldsymbol{W}_i^{(f-1)(2)}}. \tag{25}$$

The parameter $\boldsymbol{W}_i^{(f-1)(1)}$ is updated as follows:

$$\boldsymbol{W}_i^{(f-1)(1)} = \boldsymbol{W}_i^{(f-1)(1)} - \eta \frac{\partial L_1^*}{\partial \boldsymbol{W}_i^{(f-1)(1)}}. \tag{26}$$

The gradient of the loss function over the parameter vector in the original model is

$$\frac{\partial L}{\partial \boldsymbol{W}_i^{(f-1)}} = \frac{\partial L_1}{\partial y_1} \frac{\partial y_1}{\partial h_i{}^{(f)}} \frac{\partial h_i{}^{(f)}}{\partial \boldsymbol{W}_i^{(f-1)}} + \frac{\partial L_2}{\partial y_2} \frac{\partial y_2}{\partial h_i{}^{(f)}} \frac{\partial h_i{}^{(f)}}{\partial \boldsymbol{W}_i^{(f-1)}}. \tag{27}$$

We compare $\frac{\partial L}{\partial \boldsymbol{W}_i^{(f-1)}}$ with $\frac{\partial L_1^*}{\partial \boldsymbol{W}_i^{(f-1)}}$, where

$$\frac{\partial h_i{}^{(f)}}{\partial \boldsymbol{W}_i^{(f-1)}} = \frac{\partial h_i{}^{(f)}}{\partial \boldsymbol{W}_i^{(f-1)(1)}}, \tag{28}$$

$$\frac{\partial L_1}{\partial y_1} = \frac{\partial (\hat{y}_1 - y_1)^2}{\partial y_1} = \frac{\partial L_1^*}{\partial y_1}. \tag{29}$$

Therefore, the first term in (20) is equal to the value of (24), and because $\frac{\partial L_2}{\partial \boldsymbol{W}_i^{(f-1)}} < 0$, $\frac{\partial L^*}{\partial \boldsymbol{W}_i^{(f-1)(1)}} = \frac{\partial L_1^*}{\partial \boldsymbol{W}_i^{(f-1)(1)}} > \frac{\partial L}{\partial \boldsymbol{W}_i^{(f-1)}}$, L_1^* decreases by $d_1^* (d_1^* > d_1)$ units.

We update the parameter $\boldsymbol{W}_i^{(n-2)(2)}$ on task y_2:

$$\boldsymbol{W}_i^{(n-1)(2)} = \boldsymbol{W}_i^{(n-1)(2)} - \eta \frac{\partial L_2^*}{\partial \boldsymbol{W}_i^{(n-1)(2)}}. \tag{30}$$

Since $\frac{\partial L_2^*}{\partial \boldsymbol{W}_i^{(n-1)(2)}} > 0$, the value of $\boldsymbol{W}_i^{(n-1)(2)} > 0$ increases, and according to the principle of the gradient descent algorithm, L_2^* decreases. L_2^* is set to decrease by d_3 units.

Since $L^* = L_1^* + L_2^*$, L^* decreases by $d_1^* + d_3$ units. Therefore, L^* decreases by more units than L.

When $\frac{\partial L}{\partial \boldsymbol{W}_i^{(f-1)}} > 0$, $\frac{\partial L_1}{\partial \boldsymbol{W}_i^{(f-1)}} < 0$, $\frac{\partial L_2}{\partial \boldsymbol{W}_i^{(f-1)}} > 0$; $\frac{\partial L}{\partial \boldsymbol{W}_i^{(f-1)}} < 0$, $\frac{\partial L_1}{\partial \boldsymbol{W}_i^{(f-1)}} < 0$, $\frac{\partial L_2}{\partial \boldsymbol{W}_i^{(f-1)}} > 0$; $\frac{\partial L}{\partial \boldsymbol{W}_i^{(f-1)}} > 0$, $\frac{\partial L_1}{\partial \boldsymbol{W}_i^{(f-1)}} < 0$, $\frac{\partial L_2}{\partial \boldsymbol{W}_i^{(f-1)}} > 0$, it can be proven that L^* decreases by more units than L for the same reason. □

Remark 3. *The optimization algorithm used in this conclusion can also be other optimization algorithms (for example: the Adam algorithm) and is not limited to gradient descent.*

6. Conclusions

In this paper, the MTL method with adaptive nodes is proposed based on a multi-task deep learning method combined with a hard parameter sharing model. We set a continuous gradient dissimilarity threshold and a warm-up training iteration threshold in combination with the Adam optimization algorithm to dynamically update the model and further mitigate the negative migration phenomenon in MTL. The results of numerical simulations show that the adaptive hard parameter sharing model given in this paper possesses better estimation performance than the hard parameter sharing model. The effectiveness and feasibility of the adaptive hard parameter sharing model is then verified with two practical instances, and it requires a small time cost. Finally, the gradient congruence phenomenon and the fact that the adaptive hard parameter sharing model produces better estimation results are theoretically demonstrated.

Author Contributions: Methodology, Writing—original draft and Software X.J.; Writing—editing, H.W.; Writing—review, Y.D.; Data curation, N.Z.; Article layout, H.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Social Science Fund of China, grant number 22BTJ021 and 23CTJ028; the Jiangsu Province Self Science, grant number BK20210678; "Qinglan project" of Colleges and Universities of Jiangsu Province and Postgraduate Research and Practice Innovation Program of Jiangsu Province, grant number KYCX22_2154.

Data Availability Statement: Data are contained within the article.

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

References

1. Zhang, Y.; Yang, Q. An overview of multi-task learning. *Natl. Sci. Rev.* **2018**, *5*, 30–43. [CrossRef]
2. Gao, H.M.; Yin, L.; Ning, J. Artificial neural network model for ozone concentration estimation and Monte Carlo analysis. *Atmos. Environ.* **2018**, *184*, 129–139. [CrossRef]
3. García Nieto, P.J.; García-Gonzalo, E.; Bernardo Sánchez, A.; Miranda, A.A.R. Air quality modeling using the PSO-SVM-based approach, MLP neural network, and M5 model tree in the metropolitan area of Oviedo (Northern Spain). *Environ. Model. Assess.* **2018**, *23*, 229–247. [CrossRef]
4. Park, S.; Kim, M.; Kim, M.; Namgung, H.G.; Kim, K.T.; Cho, K.H.; Kwon, S.B. Predicting PM10 concentration in Seoul metropolitan subway stations using artificial neural network (ANN). *J. Hazard. Mater.* **2018**, *341*, 75–82. [CrossRef]
5. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
6. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef] [PubMed]
7. Vandenhende, S.; Georgoulis, S.; Van Gansbeke, W.; Proesmans, M.; Dai, D.; Van Gool, L. Multi-task learning for dense prediction tasks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *44*, 3614–3633. [CrossRef] [PubMed]
8. Caruana, R. Multitask learning: A knowledge-based source of inductive bias. In Proceedings of the Tenth International Conference on Machine Learning, Amherst, MA, USA, 27–29 June 1993; pp. 41–48.
9. Collobert, R.; Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proceedings of the 25th International Conference on Machine Learning, Helsinki, Finland, 5–9 July 2008; pp. 160–167.
10. Long, M.; Cao, Z.; Wang, J.; Yu, P.S. Learning multiple tasks with multilinear relationship networks. *Adv. Neural Inf. Process. Syst.* **2017**, 30, 1593–1602.
11. Ruder, S.; Bingel, J.; Augenstein, I.; Søgaard, A. Latent multi-task architecture learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 01, pp. 4822–4829.
12. Feng, X.; Zhao, X.; Feng, X. Event joint extraction method based on soft parameter sharing. *Comput. Appl. Res.* **2023**, *40*, 91–96.
13. Duong, L.; Cohn, T.; Bird, S.; Cook, P. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Beijing, China, 2015 ; Volume 2, pp. 845–850.
14. Misra, I.; Shrivastava, A.; Gupta, A.; Hebert, M. Cross-stitch networks for multi-task learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 3994–4003.
15. Mrini, K.; Dernoncourt, F.; Yoon, S.; Bui, T.H.; Chang, W.; Farcas, E.; Nakashole, N. A gradually soft multi-task and data-augmented approach to medical question understanding. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, Online, 3 August 2021; Volume 1, pp. 1505–1515.
16. Zhang, Y.; Yang, Q. A survey on multi-task learning. *IEEE Trans. Knowl. Data Eng.* **2021**, *34*, 5586–5609. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.