J Inf Process Syst, Vol.20, No.3, pp.345~359, June 2024
https://doi.org/10.3745/JIPS.04.0312
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Improving Deep Learning Models Considering the Time Lags between Explanatory and Response Variables

Chaehyeon Kim and Ki Yong Lee*

Abstract

A regression model represents the relationship between explanatory and response variables. In real life, explanatory variables often affect a response variable with a certain time lag, rather than immediately. For example, the marriage rate affects the birth rate with a time lag of 1 to 2 years. Although deep learning models have been successfully used to model various relationships, most of them do not consider the time lags between explanatory and response variables. Therefore, in this paper, we propose an extension of deep learning models, which automatically finds the time lags between explanatory and response variables. The proposed method finds out which of the past values of the explanatory variables minimize the error of the model, and uses the found values to determine the time lag between each explanatory variable and response variables. After determining the time lags between explanatory and response variables, the proposed method trains the deep learning model again by reflecting these time lags. Through various experiments applying the proposed method to a few deep learning models, we confirm that the proposed method can find a more accurate model whose error is reduced by more than 60% compared to the original model.

Keywords

Deep Learning, Model Optimization, Regression Model, Time Lag

1. Introduction

In real life, some events affect other events with a certain time lag, rather than immediately. For example, temperature, cloud cover, and humidity affect precipitation after a certain period. Advertising expenditure also affects sales volume with a certain time lag. Similarly, the marriage rate affects the birth rate after a certain period of time. We call the time difference in which one event affects another event the time lag. Therefore, when we analyze the relationships between events, it is very important to consider the time lags between those events, if present.

Recently, with the advent of deep learning techniques, many phenomena in social sciences, economics, climate, natural science, etc. have been successfully modeled using deep learning models that are based on artificial neural networks—e.g., multilayer perceptron (MLP), convolutional neural network (CNN), and long short-term memory (LSTM). These deep learning models represent the complex relationships between explanatory variables (i.e., independent or input variables) and response variables (i.e.,

Manuscript received March 14, 2022; first revision June 16, 2022; accepted July 16, 2022.
* **Corresponding Author:** Ki Yong Lee (kiyonglee@sookmyung.ac.kr)
Dept. of Computer Science, Sookmyung Women's University, Seoul, Korea (7chaeny25@sookmyung.ac.kr, kiyonglee@sookmyung.ac.kr)
Current affiliation for author, Chaehyeon Kim, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA.

dependent or output variables). However, most of those models assume that explanatory variables affect the response variable immediately and do not consider the time lags between explanatory and response variables. For example, Fig. 1(a) shows an example of a training dataset used to train a deep learning model, where $X_1, X_2, \dots, X_n$ are explanatory variables and Y is the response variable. As described in Fig. 1(a), most of existing deep learning models assume that the current value of Y is affected by the current values of $X_1, X_2, \dots, X_n$. However, as shown in Fig. 1(c) and 1(d), an explanatory variable may affect the response variable with a certain time lag. For example, Fig. 1(c) indicates that the international oil prices affect the system marginal prices (SMP) with a time lag of about half a year. Similarly, Fig. 1(d) implies that the number of marriages affects the number of births with a time lag of about a year. In these cases, it would be more desirable to consider the time lags between explanatory and response variables to build a more accurate model. For example, Fig. 1(b) depicts that the values of X_1 and X_2 affect the value of Y after 2 observations and 1 observation, respectively. However, most existing deep learning models do not automatically detect such time lags between explanatory and response variables, which degrades the model performance. Although we have proposed the initial idea of identifying the time lags between explanatory and response variables in [1], [1] considers only simple linear regression models.

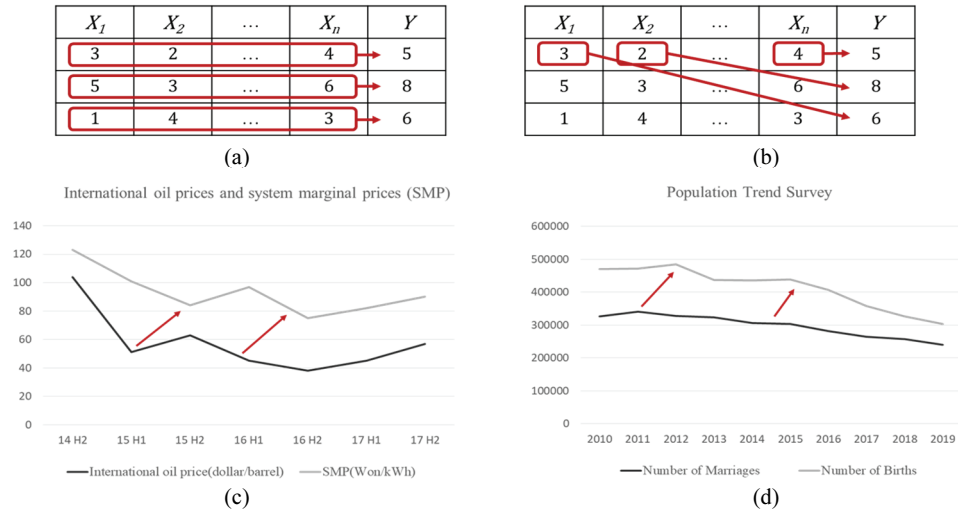

Fig. 1. Examples of time lags between explanatory and response variables: (a) An example of a training dataset, (b) The time lags between X_1, X_2, ..., X_n and Y, (c) The time lags between the oil price and SMP, and (d) The time lags between the number of marriages and the number of births.

Therefore, in this paper, we propose an extension of existing deep learning models, which automatically finds the time lags between explanatory variables $X_1, X_2, \dots, X_n$ and the response variable Y. The proposed method then reflects the found time lags to build a more accurate deep learning model. For each explanatory variable $X_i (i = 1,2, \dots, n)$, the proposed method finds out which of the past w values of X_i minimizes the error between the model and the current value of Y, and uses the found values to determine the time lag between X_i and Y. After determining the time lags between $X_1, X_2, \dots, X_n$ and Y in this way, the proposed method trains the deep learning model again by reflecting these time lags to build a more accurate model. As one of the most important advantages, the proposed method can be applied to any deep learning model based on artificial neural networks in general without additional

restrictions. Through various experiments applying the proposed method to popular and representative deep learning models including MLP, CNN, and LSTM on synthetic and real datasets, we show that the proposed method can find a more accurate model whose error is reduced by more than 60% compared to a model that does not consider the time lags between explanatory and response variables. Our contributions are summarized as follows:

- For the first time, we propose a method to improve the performance of a given deep learning model by considering the time lags between explanatory and response variables.
- We propose a new layer that can be added to an existing deep learning model that automatically identifies the time lags between explanatory and response variables
- We show the performance improvement of the proposed method by applying the proposed method to popular deep learning models on synthetic and real datasets.

The rest of the paper is organized as follows. In Section 2, we present related work on time lags and briefly describe three popular and representative deep learning models used in the experiments. Section 3 describes the proposed method for automatically identifying the time lags between explanatory and response variables. Section 4 presents experimental results on synthetic and real datasets to show the effectiveness of the proposed method. Finally, Section 5 concludes the paper.

2. Related Work

2.1 Considering the Time Lags between Explanatory and Response Variables

The term "time lag" refers to the time interval between two related events occurring, such as the time interval between marriage and childbirth and the time interval between cloud formation and rainfall. As mentioned in Section 1, the marriage rate affects the birth rate a few years later. In this case, if we build a model that predicts the birth rate from the marriage rate without considering the time lag between them, we will end up with a model with low accuracy. However, there has been little research on building a model by automatically identifying the time lags between explanatory variables $X_1, X_2, \dots, X_n$ and the response variable Y, and even recent studies proceed without considering time lags [2]. Previous studies considering the time lags between explanatory and response variables are largely classified into the following three categories:

1) Set the time lags between explanatory and response variables arbitrarily through prior knowledge: For example, [3] builds a birth rate prediction model that predicts this year's birth rate from some statistics from a few years ago (e.g., 1 or 2 years ago). These types of methods do not automatically identify the time lags between explanatory and response variables, but set the time lags manually. However, setting the time lags arbitrarily does not guarantee the model with optimal performance.
2) Build a model for each possible time lag and then select the model with the best performance: For example, [4] builds a linear regression model by including a variable representing the time lag between explanatory variables and the response variable. However, [4] needs to set the value of that variable for each possible time lag, which makes it difficult to apply in practice. More specifically, if we have n explanatory variables $X_1, X_2, \dots, X_n$ and the number of possible time lags for each explanatory variable is w, then we need to build a total of w^n models and then find the best model among them. Furthermore, this method is only applicable to linear regression models.

3) Assign different weights to several time lags and find the best combination of weights that gives the highest performance: For example, [5] constructs candidate combinations of weights for several time lags (e.g., 1 month, 2 months, …, T months) and selects the best combination of weights based on the square of correlation coefficient R^2 of the linear regression model. However, [5] needs to construct a lot of candidate combinations of weights and measure R^2 for each combination. More specifically, if we assign weights P_1, P_2, …, P_T for T time lags (e.g., 1 month, 2 months, …, T months), where $P_1 + P_2 + \ldots + P_T = 1$, there can be a prohibitively large number of candidate combinations of weights (e.g., $P_1 = 0.1$, $P_2 = 0.2$, …, $P_T = 0.5$). Furthermore, this method only considers linear regression models and does not consider deep learning models.

Therefore, to date, there is no special way to systematically identify the time lags between explanatory and response variables, especially for deep learning models. This motivates our work.

2.2 Deep Learning Models

This section briefly describes three popular and representative deep learning models (i.e., MLP, CNN, and LSTM) considered in this paper.

2.2.1 MLP

A MLP is the simplest form of a deep learning model, which consists of a number of layers. These are an input layer, several hidden layers, and an output layer. Fig. 2 shows the architecture of an MLP. Each layer also consists of a number of nodes (or neurons) and each node in one layer connects to every node in the following layer with some weights. Each node in the input layer corresponds to an explanatory variable $X_i (i = 1, 2, \ldots, n)$. Each node in a hidden layer transforms the values from the previous layer with a weighted sum followed by an activation function, which can be represented as $y = f(w_1 x_1 + w_2 x_2 + \cdots + w_m x_m + \beta)$, where m is the number of nodes in the previous layer, $x_1, x_2, \ldots, x_m$ represent the values from the previous layer, $w_1, w_2, \ldots, w_m$ represent the weights between the nodes in the previous layer and this node, β represents the bias of this node, f represents the activation function (e.g., linear activation, ReLU, sigmoid, hyperbolic tangent), and y represents the output value of this node. The output layer receives the output values from the last hidden layer and transforms them into a final output value, which corresponds to the response variable Y.

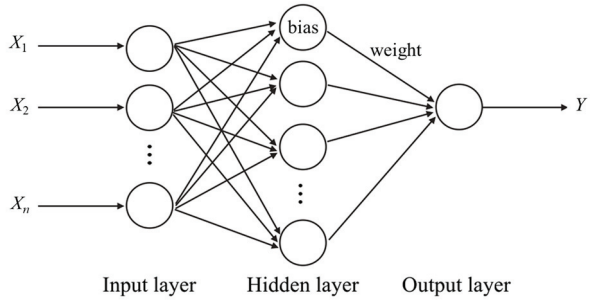

Fig. 2. The architecture of MLP.

Given a training dataset where each instance has the form $(x_1, x_2, \ldots, x_n, y)$, where $x_1, x_2, \ldots, x_n$ are the values of the explanatory variables $X_1, X_2, \ldots, X_n$ and y is the value of the response variable Y, an MLP

is trained to find the optimal parameters (the weight between nodes and the bias of each node) that minimize the error between y and the output of the MLP for the input values $x_1, x_2, \ldots, x_n$. The MLP is simple but widely used because of its ability to approximate complex functions between explanatory and response variables.

2.2.2 CNN

A CNN [6] is a type of deep learning model originally proposed to process image data. Since the original MLP basically processes one-dimensional data, when image data are used as input, the image data should be flattened and made into one-dimensional data. In this process, the spatial and topological information of the image is lost. A CNN takes 2- or 3-dimensional data as raw input and builds up layers of features while maintaining the spatial and topological information of the input data. Consequently, a CNN is typically used to identify spatial-invariant patterns.

As shown in Fig. 3, a CNN is composed of multiple layers of convolution layers and pooling layers [7]. In the first convolution layer of the CNN in Fig. 3, several feature maps are obtained using multiple filters (or kernels), each of which detects some specific type of feature over the input data. Next, the first pooling layer reduces the dimensionality of each feature map. The two common pooling methods are max and average pooling. The second convolution layer again obtain a number of feature maps from the output of the first pooling layer using multiple filters. The second pooling layer then again reduces the size of each feature map generated by the second convolution layer. Next, the output of the second pooling layer is flattened into one-dimensional data, and finally, one or more fully connected layers are applied with some activation function.

Although CNNs are widely used for classification problems [8], they can also be used for regression problems [9]. For a classification problem, a softmax layer is typically used as the output layer of a CNN to output the predicted probability of each class. On the other hand, for a regression problem, a fully-connected layer can be used as the output layer of a CNN, where each node outputs an arbitrary continuous value. In this case, the CNN extracts features from the input data using convolution layers and pooling layers and then outputs continuous values to predict continuous response variables.

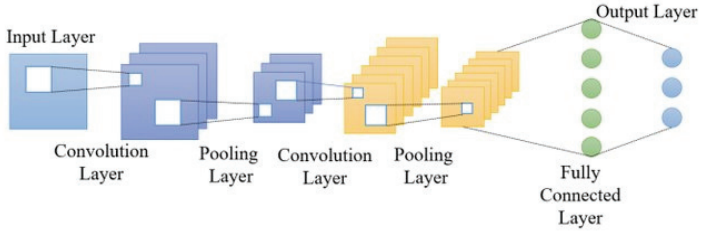

Fig. 3. The architecture of CNN. Adapted from [7].

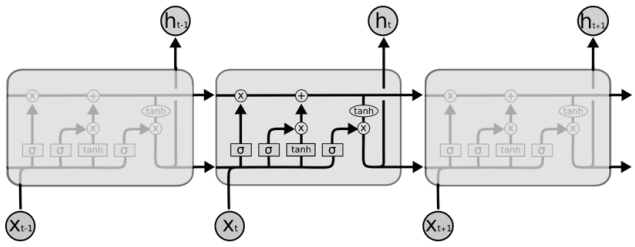

Fig. 4. The architecture of LSTM.

2.2.3 LSTM

LSTM [10] is one of the recurrent neural network (RNN) models where the output of nodes can be fed back to the same nodes. The LSTM is a widely used deep learning model because it can solve the long-term dependency problem of the traditional RNN. LSTM is mainly used to process sequence data such as time series, natural language text, and speech. Fig. 4 shows the architecture of LSTM. Unlike the traditional RNN, each unit of LSTM is composed of an input gate, a forget gate, and an output gate. The LSTM solves the long-term dependency problem by adding a cell state to each unit in the network. The cell state C_t stores long-term information from the previous time steps. The LSTM uses the three gates to remove or add information to C_t. The forget gate determines which information from the previous cell state C_{t-1} is forgotten. The forget gate receives the previous hidden state h_{t-1} and the current input x_t of the unit, and outputs a vector f_t, where each element is the output of the sigmoid function and has a value between 0 and 1. If the value of an element is 0, all the information of the corresponding element in C_{t-1} is completely forgotten. If it is 1, all the information of the corresponding element in C_{t-1} is remembered. Next, the input gate determines how much of the current input will be used. The input gate receives h_{t-1} and x_t, and outputs a vector i_t, where each element is the output of the sigmoid function and represents how much we will update the corresponding element in C_{t-1}. Along with this, new information that could be added to C_{t-1}, denoted by $\tilde{C}_t$, is obtained by passing h_{t-1} and x_t through the tanh activation function. Then, the new cell state C_t is obtained by $C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$, where $\circ$ is the Hadamard (or element-wise) product. Finally, the output gate determines how much of the new cell state C_t is output into the new hidden state h_t. The output gate receives h_{t-1} and x_t, and outputs a vector o_t, where each element is the output of the sigmoid function and represents how much we will output the corresponding element in C_t. Then, the new hidden state h_t is obtained by $h_t = o_t \circ \tanh(C_t)$.

3. Proposed Method

In this section, we present the proposed extension for existing deep learning models, which automatically finds the time lags between explanatory and response variables. The proposed method consists of two steps: (1) identifying the time lags and (2) building a model reflecting the identified time lags. We first define the problem and then describe the proposed method in detail.

3.1 Problem Definition

We assume that a training dataset has the form shown in Fig. 5. In Fig. 5, each row represents a data instance, where $X_1, X_2, \ldots, X_n$ are explanatory variables and Y is the response variable. The symbol to the left of each row (i.e., $k-w+1$, …, $k-2$, $k-1$, k) indicates the sequence number of the corresponding data instance. ${X_i}^{(k)}$ and $Y^{(k)}$ represent the values of X_i and Y of the k-th data instance, respectively. Thus, the k-th data instance can be expressed as $({X_1}^{(k)}, {X_2}^{(k)}, \ldots, {X_n}^{(k)}, Y^{(k)})$. The problem of building a model that represents the relationship between $X_1, X_2, \ldots, X_n$ and Y can be viewed as a problem of finding a function f that satisfies $Y = f(X_1, X_2, \ldots, X_n)$.

However, most of the existing approaches to building a model $Y = f(X_1, X_2, \ldots, X_n)$ assume that $Y^{(k)}$ is determined by ${X_1}^{(k)}, {X_2}^{(k)}, \ldots, {X_n}^{(k)}$. In other words, they try to find f that minimizes the error between

$Y^{(k)}$ and $f(X_1{}^{(k)}, X_2{}^{(k)}, \ldots, X_n{}^{(k)})$. However, as mentioned in Section 1, some events in real life affect other events with a certain time lag. For example, in Fig. 5, $Y^{(k)}$ may be determined by $X_1{}^{(k-2)}$, $X_2{}^{(k-1)}$, …, $X_n{}^{(k)}$, rather than $X_2{}^{(k)}$, $X_2{}^{(k)}$, …, $X_n{}^{(k)}$. In this case, we say that there is a time lag of 2 between X_1 and Y, that there is a time lag of 1 between X_2 and Y, and that there is no time lag between X_n and Y.

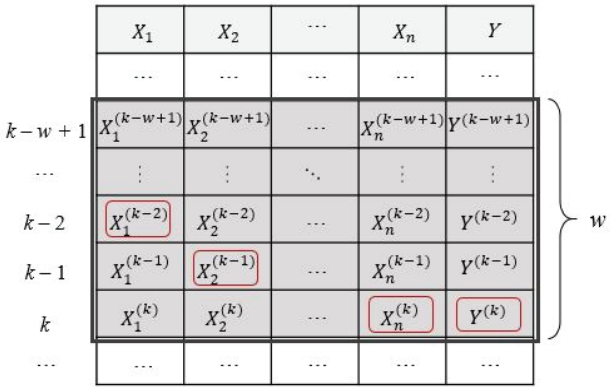

Fig. 5. The form of a training dataset.

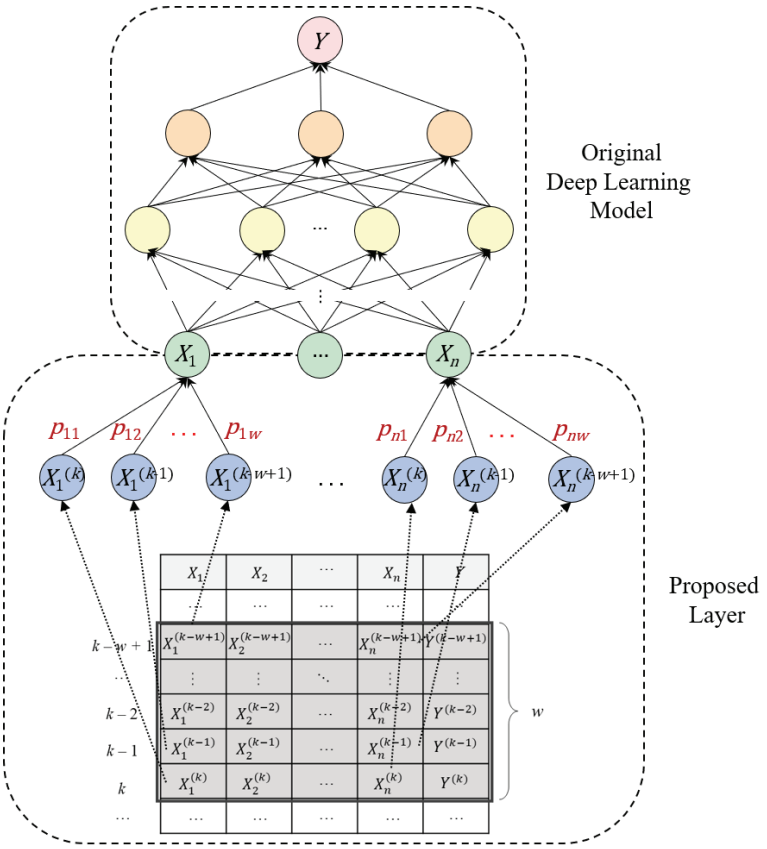

Fig. 6. The proposed extension to an existing deep learning model.

Given a training dataset in the form shown in Fig. 5 and a deep learning model f_θ, where θ is the parameters of f, this paper addresses the problem of finding the time lags between each of $X_1, X_2, \dots, X_n$ and Y, and the optimal parameters θ^* of f_θ. In other words, our goal is to find the time lags $t_1, t_2, \dots, t_n$ and the optimal model f_{θ^*} that minimize the error between $Y^{(k)}$ and $f_{\theta^*}({X_1}^{(k-t_1)}, {X_2}^{(k-t_2)}, \dots, {X_n}^{(k-t_n)})$, where $t_i (i = 1, 2, \dots, n)$ is the time lag between X_i and Y. Here, we assume that the time lag between $X_i (i = 1, 2, \dots, n)$ and Y are at most w, i.e., $0 \le t_i \le$ w. As mentioned in Section 1, the proposed method can be applied to any deep learning model that is based on artificial neural networks without additional restrictions. Thus, f_θ can be any deep learning model, e.g., MLP [11], CNN [12], and LSTM [13].

3.2 Overview

Given a training dataset where $X_1, X_2, \dots, X_n$ are explanatory variables and Y is the response variable, and a deep learning model f_θ, the proposed method finds the time lag between X_i and Y for each X_i, and the optimal parameters θ^* of f_θ considering the found time lags between X_i and Y. For this purpose, we extend the given deep learning model by adding an additional layer to find the time lags between X_i and Y. Fig. 6 shows the proposed extension to an existing deep learning model. In Fig. 6, the upper part corresponds to an existing deep learning model that represents the relationship between $X_1, X_2, \dots, X_n$ and Y without considering the time lags between them. Here, the nodes $X_1, X_2, \dots, X_n$ represents the explanatory variables $X_1, X_2, \dots, X_n$, respectively, and the topmost node Y represents the response variable Y. Note that any deep learning model based on artificial neural networks can be at the upper part. The lower part in Fig. 6 corresponds to the additional layer proposed in this paper to identify the time lag between each explanatory variable X_i and Y. Each node X_i in the upper part is connected to the nodes $X_i^{(k)}, X_i^{(k-1)}, \dots, X_i^{(k-w+1)}$ in the proposed layer. The nodes $X_i^{(k)}, X_i^{(k-1)}, \dots, X_i^{(k-w+1)}$ in the proposed layer represent the most recent w values of the explanatory variable X_i. The weight $p_{ij} (i = 1, 2, \dots, n, j = 1, 2, \dots, w)$ between the node X_i and the node $X_i^{(k-j+1)}$ indicates how much $X_i^{(k-j+1)}$ contributes to the value of X_i that minimizes the error of the model. All the weights p_{ij} are learnable and determined through training, and these weights will be used to identify the time lag between each X_i and Y. In the next subsection, we present how to identify the time lag between each X_i and Y using the proposed layer.

3.3 Identifying Time Lags

In order to identify the time lag between each X_i and Y, we need to obtain all $p_{ij} (i = 1, 2, \dots, n, j = 1, 2, \dots, w)$ through training. For this purpose, we train the whole model shown in Fig. 6 as follows. First, we use the loss function used to train the original deep learning model (at the upper part in Fig. 6) without modification. When training the whole model in Fig. 6, we use w consecutive rows in the training dataset as one training data instance. Recall that w is the maximum time lag. For example, the first training data instance is from the 1st row to the w-th row in the training dataset, the second training data instance is from the 2nd row to the (w + 1)-th row, and so on. In other words, we slide a window of size w row by row on the training dataset and use each as a training data instance. Let the current training instance be from the (k – w + 1)-th row to the k-th row. At this time, each node $X_i^{(k-j+1)}$ in the proposed layer takes $X_i^{(k-j+1)}$ in the training dataset as input and outputs it as it. As mentioned in Section 3.2, we use a weight

p_{ij} between the node X_i and the node $X_i^{(k-j+1)}$ to represent how much $X_i^{(k-j+1)}$ contributes to the value of X_i that minimizes the error of the model. For p_{ij} to express this meaning, we modify the output of the node X_i, denoted by X_i^{output}, as follows:

$$X_i^{output} = \sum_{j=1}^{w} p_{ij} X_i{}^{(k-j+1)} \tag{1}$$

Also, for p_{ij} to represent its share in the total contribution, we impose the following constraints on p_{ij} during training, as in [14]:

$$0 \le p_{ij} \le 1 \tag{2}$$

$$\sum_{j=1}^{w} p_{ij} = 1 \tag{3}$$

where $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, w$. To make all p_{ij} satisfy the constraints in Eq. (2) and Eq. (3) during training, we update each p_{ij} once more after $p_{i1}, p_{i2}, \dots, p_{iw}$ are updated at each epoch by the optimizer. We use the following two techniques to make all p_{ij} satisfy the constraints in Eqs. (2) and (3).

Normalization [15]: We use the following equation to normalize $p_{i1}, p_{i2}, \dots, p_{iw}$ to values in the range of [0, 1] and make their sum 1.

$$p_{ij} = \frac{p_{ij}}{\sum_{j=1}^{w} p_{ij}} \tag{4}$$

Softmax function [16]: We use the softmax function to make $p_{i1}, p_{i2}, \dots, p_{iw}$ be in the range of [0, 1] and ensure that they add up to 1.

$$p_{ij} = \frac{e^{p_{ij}}}{\sum_{j=1}^{w} e^{p_{ij}}} \tag{5}$$

Once the training of the whole model in Fig. 6 is completed using the method described so far, we determine the time lag between each $X_i (i = 1, 2, \dots, n)$ and Y as follows: among the weights $p_{i1}, p_{i2}, \dots, p_{iw}$ between nodes X_i and $X_i{}^{(k)}, X_i{}^{(k-1)}, \dots, X_i{}^{(k-w+1)}$ after training, let p_{ij} be the largest weight $(1 \le j \le w)$. This means that $X_i{}^{(k-j+1)}$ contributed the most to the value of X_i that minimizes the error of the model among $X_i{}^{(k)}, X_i{}^{(k-1)}, \dots, X_i{}^{(k-w+1)}$. In this case, we determine that the time lag between X_i and Y is j, i.e., $t_i = j$, because the error of the model is minimized when $X_i{}^{(k-j+1)}$ is used rather than the other values. We will present the performance of the proposed method when each of Eqs. (4) and (5) is used in Section 4. In the next subsection, we present how to build the final model reflecting the identified time lags.

3.4 Building a Model Reflecting Time Lags

Once we obtain the time lags $t_1, t_2, \dots, t_n$ between the explanatory variables $X_1, X_2, \dots, X_n$ and the response variable Y, we train the original deep learning model (at the upper part in Fig. 6) again alone to obtain the final model that considers the time lags between $X_1, X_2, \dots, X_n$ and Y. In order to build the final

deep learning model reflecting the identified time lags, we transform the original training dataset, where each data instance has the form $(X_1^{(k)}, X_2^{(k)}, \ldots, X_n^{(k)}, Y^{(k)})$, into a new training dataset, where each data instance has the form $(X_1^{(k-t_1)}, X_2^{(k-t_2)}, \ldots, X_n^{(k-t_n)}, Y^{(k)})$. In other words, we shift the values of each explanatory variable X_i downward by its time lag t_i. For example, if the time lag between X_1 and Y is 2 and the time lag between X_2 and Y is 1, we shift the values of X_1 and X_2 downward by 2 and 1, respectively, as shown in Fig. 7. In this way, $X_1^{(k-t_1)}, X_2^{(k-t_2)}, \ldots, X_n^{(k-t_n)}$ are associated with $Y^{(k)}$ instead of $X_1^{(k)}, X_2^{(k)}, \ldots, X_n^{(k)}$. After obtaining the transformed training dataset, we also make the nodes $X_1, X_2, \ldots, X_n$ in Fig. 6 output their input as it is instead of outputting X_i^{output} (i.e., Eq. (1)). Finally, we train the original deep learning model with the transformed training dataset. Note that in this case, we use each row of the transformed training dataset as one training data instance.

Through the proposed method described in Section 3, we can easily obtain the time lags between $X_1, X_2, \ldots, X_n$ and Y, and the deep learning model reflecting them. Thus, we do not need to build w^n models for all possible combinations of time lags and find the model with the best performance among them.

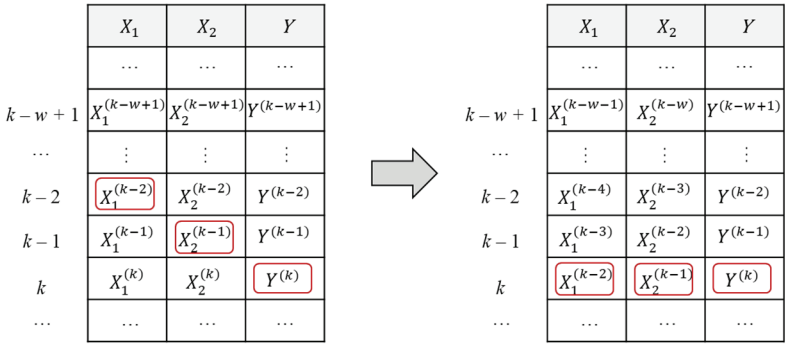

Fig. 7. An example of a transformed training dataset.

4. Experiments

4.1 Experimental Setting

In this section, we present the performance evaluation results of the proposed method. In order to show that the proposed method improves the performance of existing deep learning models by considering the time lags between explanatory and response variables, we use the following datasets:

1) Synthetic dataset: We assumed a model of the form $Y = \beta + \alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_3 + \alpha_4 X_4$, where X_1, X_2, X_3 and X_4 are explanatory variables, Y is the response variable, $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ and β are some fixed constants. We also assumed that there are a time lag of 2 between X_1 and Y, a time lag of 1 between X_2 and Y, a time lag of 4 between X_3 and Y, and a time lag of 3 between X_4 and Y, i.e., $t_1 = 2, t_2 = 1, t_3 = 4$, and $t_4 = 3$. Given the model and time lags, we randomly generated a dataset that fits the model and time lags. This dataset intends to confirm that the proposed method finds the model and time lags accurately.
2) Real dataset: We used demographic data provided by the Korean Statistical Information Service

(KOSIS) [17]. KOSIS is a database managed by Statistics Korea (KOSTAT), which contains 1,000 types of national approval statistics on the economy, society, and environment. From KOSIS, we selected the marriage rate data from January 1997 to December 2020, employment rate data for women from June 1999 to October 2021, and fertility rate data from January 1997 to December 2020, which are calculated for each month. For the employment rate data, we only used data for women in their 20s and 30s because women's childbearing age is from their mid-20s to mid-30s. As a result, this dataset consists of two explanatory variables (i.e., marriage rate and employment rate) and one response variable (i.e., fertility rate).

For both synthetic and real datasets, we split the dataset into two sets, which are training and test set. We used 80% and 20% of the dataset as the training and test set, respectively. We trained each model used in the experiments using the training set and measured the performance of each model using the test set.

As mentioned earlier, the proposed method can be applied to any deep learning model that is based on artificial neural networks. In the experiments, we used three deep learning models to show the effectiveness of the proposed method.

1) MLP: We constructed an MPL with three layers: one input layer, one hidden layer, and one output layer. The number of nodes in the input layer and hidden layer is the same as the number of explanatory variables, the number of nodes in the output layer is 1. To train the MLP, we used the mean squared error (MSE) as the loss function and the Adam optimizer with a learning rate of 0.01.
2) CNN: We also constructed a CNN with five layers: one input layer, one convolution layer, one max pooling layer, one fully connected layer, and one output layer. The convolutional layer uses a 1×1 filter with stride 1. We used ReLU as the activation function. To train the CNN, we used MSE as the loss function and the Adam optimizer with a learning rate of 0.01.
3) LSTM: Finally, we constructed an LSTM with three layers: one input layer, one LSTM layer with 3 hidden states, and one output layer. We set the size of the input layer to the number of explanatory variables. To train the LSTM, we used MSE as the loss function and the Adam optimizer with a learning rate of 0.01

We compared the performance of the above original deep learning models with that of the models optimized by the proposed method considering the time lags between explanatory and response variables. We also investigated the performance of the proposed method when we use each of Eqs. (4) and (5) described in Section 3.3. Finally, we evaluated the performance of the proposed method for varying values of w, the maximum time lag, which is the only parameter of the proposed method. The proposed method was implemented using PyTorch, and all the experiments were conducted on a PC running Ubuntu with Intel Core i7-9700 CPU, TITAN V GPU, and 16 GB memory. In the next subsection, we present how much the proposed method improves the performance of the three deep learning models (i.e., MLP, CNN, and LSTM) by considering the time lags between explanatory and response variables.

4.2 Evaluation Results

4.2.1 Performance evaluation on synthetic dataset

In this section, we present performance evaluation results on the synthetic dataset. In this experiment, we used the synthetic dataset described in Section 4.1. To measure the performance of a model, we used

MSE for the training dataset, which was averaged over 5 runs. Fig. 8 shows the experimental results on the synthetic dataset. Here, "Baseline" represents the original deep learning model that does not consider the time lags between explanatory and response variables. More specifically, given the *k*-th data instance $({X_1}^{(k)}, {X_2}^{(k)}, \dots, {X_n}^{(k)}, Y^{(k)})$, "Baseline" predicts $Y^{(k)}$ using only ${X_1}^{(k)}, {X_2}^{(k)}, \dots, {X_n}^{(k)}$ and does not use ${X_1}^{(k-t_1)}, {X_2}^{(k-t_2)}, \dots, {X_n}^{(k-t_n)}$, where $t_i (i = 1, 2, \dots, n) \geq 1$. On the other hand, "*w* = *n*" represents the model found by the proposed method when we set *w* to *n*. We varied *w* from 1 to 4. Fig. 8(a)–8(c) show the effect of the proposed method when the proposed method is applied to MLP, CNN, and LSTM described in Section 4.1, respectively. Finally, "with normalization" and "with softmax" represent the models found by the proposed method using Eqs. (4) and (5), respectively.

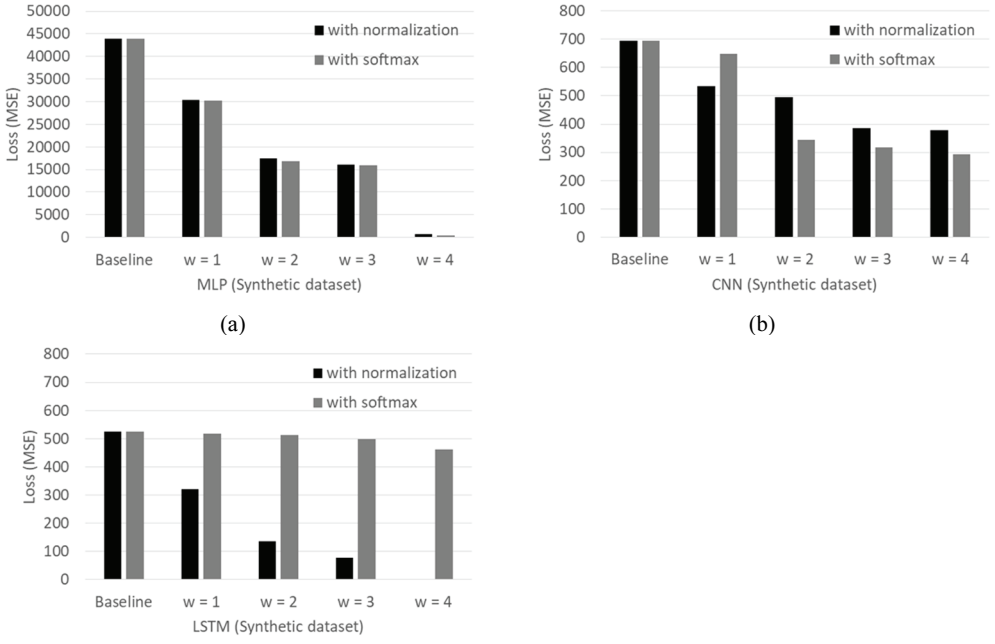

Fig. 8. Performance evaluation results on the synthetic dataset: (a) MLP, (b) CNN, and (c) LSTM.

In Fig. 8, we can see that the MSE of the models found by the proposed method are significantly reduced compared to the original deep learning model (i.e., Baseline). This means that the proposed method improves the performance of the given deep learning model by reflecting the time lags between explanatory and response variables. Also, we can see that the performance improvement of the proposed method increases as *w* increases. This is because the proposed method can find the best time lag for each explanatory variable in a wider range. However, after some good time lags are found, we can see that the performance improvement is not very large even if *w* further increases. Thus, we can confirm that the proposed method can identify the time lags between explanatory and response variables effectively.

Note that the performance of the proposed method varies depending on whether Eq. (4) or Eq. (5) is used (i.e., "with normalization" and "with softmax"). This is because Eqs. (4) and (5) have different ways of normalizing $p_{i1}, p_{i2}, \dots, p_{iw}$. More specifically, Eq. (4) assigns each p_{ij} a value that is exactly proportional to its current value, while Eq. (5) gives a much more value to p_{ij} of any value larger.

Therefore, when the time lag is not certain, Eq. (5) can assign a large weight to a wrong p_{ij} accidentally. For this reason, the softmax function can sometimes lead to reduced performance improvement. For example, in Fig. 8(c), the proposed method successfully found the optimal time lags (i.e., the loss is nearly zero) when Eq. (4) was used, while the proposed method showed only a slight performance improvement when Eq. (5) was used.

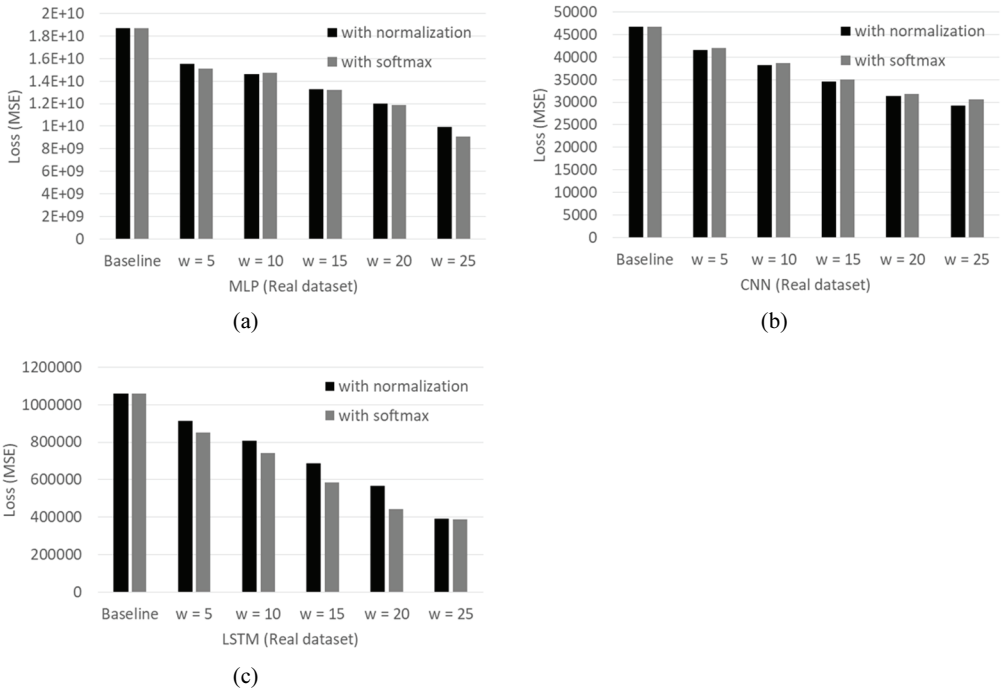

Fig. 9. Performance evaluation results on the real dataset: (a) MLP, (b) CNN, and (c) LSTM.

4.2.2 Performance evaluation on real dataset

In this section, we present performance evaluation results on the real dataset described in Section 4.1. We performed the experiment in exactly the same way as in Section 4.2.1, except that we varied *w* from 5 to 25 in steps of 5. Fig. 9(a)–9(c) show the effect of the proposed method when the proposed method is applied to MLP, CNN, and LSTM described in Section 4.1, respectively.

Also in Fig. 9, we can see that the proposed method improves the performance of the given deep learning models significantly. As in the case of the synthetic dataset, the performance improvement of the proposed method increases in general as w increases. Recall that the real dataset has two explanatory variables (i.e., the marriage rate and the employment rate of women) and one response variable (i.e., the birth rate). Fig. 9 implies that the birth rate can be predicted more accurately using the marriage rate and employment rate 25 months ago rather than the current month. Thus, the experimental results confirm again that the performance of a model can be improved by considering the time lags between explanatory and response variables. In the case of the real dataset, the use of Eq. (4) or Eq. (5) does not affect the performance improvement of the proposed method significantly. If there are clear time lags between explanatory and response variables, both equations yield similar results.

5. Conclusion

In this paper, we proposed a method to improve the performance of a given deep learning model by automatically identifying the time lags between explanatory and response variables and reflecting them in the model. For this purpose, the proposed method adds a new layer to the existing deep learning model and uses this layer to find which values in the past minimize the error of the model. By using this new layer, the proposed method can find the time lags between explanatory and response variables easily without building models for all possible time lags.

We also evaluated the effect of the proposed method by applying the proposed method to three popular deep learning models (i.e., MLP, CNN, and LSTM) on synthetic and real datasets. The experimental results show that the proposed method actually improves the performance of a given deep learning model by reflecting the time lags between explanatory and response variables. For the real dataset, the proposed method reduced the error of the existing model by up to 60%. Therefore, we can conclude that the proposed method can improve the performance of existing deep learning models when there are time lags between explanatory and response variables.

Conflict of Interest

The authors declare that they have no competing interests.

Funding

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2021R1A2C1012543).

References

[1] C. Kim, E. Ryoo, and K. Y. Lee, “Deep learning model for identifying the time lag between explanatory variables and response variable in regression analysis,” in *Proceedings of Annual Conference of the Korea Information Processing Society*, Yeosu, Korea, 2021, pp. 868-871. https://doi.org/10.3745/PKIPS.y2021m11a.868

[2] J. Wu, “Prediction of birth rate in China under three-child policy based on neural network,” in *Proceedings of 2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP)*, Xi’an, China, 2022, pp. 1652-1655. https://doi.org/10.1109/ICSP54964.2022.9778548

[3] K. Kim and S. Jeon, “Scenario analysis of fertility in Korea using the fertility rate prediction model,” *The Korean Journal of Applied Statistics*, vol. 28, no. 4, pp. 685-701, 2015. https://doi.org/10.5351/KJAS.2015.28.4.685

[4] T. Hayduk and M. Walker, “The effect of advertising on sales and brand equity in small sport businesses,” *Sport Marketing Quarterly*, vol. 30, no. 3, pp. 178-192, 2021. http://doi.org/10.32731/SMQ.303.0921.02

[5] Q. Sun, C. Liu, T. Chen, and A. Zhang, “A weighted-time-lag method to detect lag vegetation response to climate variation: a case study in Loess Plateau, China, 1982–2013,” *Remote Sensing*, vol. 13, no. 5, article no. 923, 2021. https://doi.org/10.3390/rs13050923

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Boston, MA, USA, 2015, pp. 1-9.

[7] H. Gu, Y. Wang, S. Hong, and G. Gui, "Blind channel identification aided generalized automatic modulation recognition based on deep learning," *IEEE Access*, vol. 7, pp. 110722-110729, 2019. https://doi.org/10.1109/ACCESS.2019.2934354

[8] M. Aamir, Z. Rahman, W. A. Abro, M. Tahir, and S. M. Ahmed, "An optimized architecture of image classification using convolutional neural network," *International Journal of Image, Graphics and Signal Processing*, vol. 11, no. 10, pp. 30-39, 2019. https://doi.org/10.5815/ijigsp.2019.10.05

[9] S. Miao, Z. J. Wang, and R. Liao, "A CNN regression approach for real-time 2D/3D registration," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1352-1363, 2016. https://doi.org/10.1109/TMI.2016.2521800

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997. https://doi.org/10.1162/neco.1997.9.8.1735

[11] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, et al., "MLP-mixer: an all-MLP architecture for vision," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24261-24272, 2021.

[12] H. Han, "Residual learning based CNN for gesture recognition in robot interaction," *Journal of Information Processing Systems*, vol. 17, no. 2, pp. 385-398, 2021. https://doi.org/10.3745/JIPS.01.0072

[13] J. Kim, J. Park, M. Shin, J. Lee, and N. Moon, "The method for generating recommended candidates through prediction of multi-criteria ratings using CNN-BiLSTM," *Journal of Information Processing Systems*, vol. 17, no. 4, pp. 707-720, 2021. https://doi.org/10.3745/JIPS.02.0159

[14] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: training deep neural networks with weights and activations constrained to+ 1 or -1," 2016 [Online]. Available: https://arxiv.org/abs/1602.02830

[15] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. San Francisco, CA: Morgan Kaufmann, 2001.

[16] S. Gold and A. Rangarajan, "Softmax to softassign: neural network algorithms for combinatorial optimization," *Journal of Artificial Neural Networks*, vol. 2, no. 4, pp. 381-399, 1996.

[17] Statistics Korea, "KOSIS (Korean Statistical Information Service)," c2024 [Online]. Available: http://kosis.kr

Chaehyeon Kim **https://orcid.org/0000-0002-6443-1286**

She received her B.S. in both Statistics and Computer Science from Sookmyung Women's University, Seoul, Republic of Korea, in 2021. She also received her M.S. degree in Computer Science from Sookmyung Women's University, Seoul, Korea, in 2023. She is currently a PhD student in Computer and Information Science at the University of Pennsylvania. Her current research interests include databases, data mining, and deep learning.

Ki Yong Lee **https://orcid.org/0000-0003-2318-671X**

He received his B.S., M.S., and Ph.D. degrees in Computer Science from KAIST, Daejeon, Republic of Korea, in 1998, 2000, and 2006, respectively. From 2006 to 2008, he worked for Samsung Electronics, Suwon, Korea as a senior engineer. From 2008 to 2010, he was a research assistant professor of the Department of Computer Science at KAIST, Daejeon, Korea. He joined the faculty of the Division of Computer Science at Sookmyung Women's University, Seoul, in 2010, where currently he is a professor. His research interests include database systems, data mining, and big data.