Attention-Based Mechanism for Extracting Multivariate Features for Time Series Forecasting Problem

Sheng-Tzong Cheng, Chia-Hsuan Lin, and Ya-Jin Lyu
Department of Computer Science and Information Engineering
National Cheng Kung University, Tainan, Taiwan.

Abstract- With the advent of big data, the world's data has shown explosive growth, and the growth rate of data is much faster than the speed at which humans can analyze the data. Therefore, it is crucial to use effective methods to enable humans to find out the hidden correlations or regular patterns from these seemingly messy data, to help our human life more convenient and progressive.

Moreover, time series forecasting problems not only need to find the correlation between the target feature and other non-target features, but also need to consider the time correlation, and use this as a basis to infer and predict the target result at a future time point. In addition, time series data usually have non-linear and unstable characteristics, which makes the general linear regression method unable to effectively solve this problem.

For time series forecasting problems, many deep learning structures exploit attention-based mechanisms in recent years. The Dual-Stage Attention-Based Recurrent Neural Network (DA-RNN) embeds the attention mechanism into the Long Short-Term Memory (LSTM). This paper proposes three models using Long-term Recurrent Convolutional Network (LRCN), self-attention mechanism, and Dilated Causal Convolutional Neural Network (DC-CNN). In this paper, we consider weather and stock forecasting datasets to show that our method can achieve lower errors with other deep learning models that use attention mechanisms. Our work can be applied to various intelligent operations and industry..

Keywords – Time Series Forecasting Problems, Deep Learning, Attention Mechanism

I. INTRODUCTION

With the advent of big data, the world's data have shown explosive growth, and the growth rate of data is much faster than the speed at which humans can analyze the data. Therefore, the world is continuing to study how to use effective methods to enable humans to find out the hidden correlations or regular patterns from these seemingly messy data, to help our human life more convenient and progressive. Among these problems, the time series forecasting problem is also one of the branches. Time series forecasting problems not only need to find the correlation between the target feature and other non-target features, but also need to consider the time correlation, and use this as a basis to infer and predict the target result at a future time point. In addition, time series data usually have non-linear and unstable characteristics, which makes the general linear regression method unable to effectively solve this problem.

The attention mechanism originally appeared in the field of natural language processing (NLP). The purpose is to allow the machine to learn the structure of the sentence faster by understanding the attention of each word in the sentence to other words. So that it has better performance in various NLP tasks, e.g., Vaswani et al. [1] proposed a self-attention mechanism to solve the problem of sentence translation. Later, the attention mechanism was gradually used in other fields to achieve the effect of removing noise from data. For example, in the computer vision field, Xu et al. [2] proposed a set of attention mechanisms for image analysis to help machines improve accuracy in image caption generation problems. In time series forecasting, Qin et al. [3] embedded the attention mechanism into LSTM; Cheng et al. [4] added CBAM and HBAM attention mechanisms to the model to improve the extraction effect.

There are different calculation methods in the attention mechanism, e.g., Graves et al. [5] proposed content-based attention, and uses cosine() to calculate the similarity; Bahdanau et al. [6] proposed to use tanh() with weight to calculate the similarity; Luong et al. [7] proposed the use of softmax() and dot-product to calculate; Vaswani et al. [1] proposed scaled dot-product method. Among the models of time series prediction, the attention used by DARNN [3]

is softmax() with concatenating method; A_SeriesNet [4] used AveragePooling() plus MaxPooling() with sigmoid() method.

II. PROPOSED METHODS

This chapter will describe our extracting method in detail. First, we will explain the problem to be solved and its definition. Next, we will describe our Type & Time extracting method. And then, we will show the three models based on the Type & Time extracting method that we proposed.

In the time series forecasting problem, there are many different types of data. In this paper, the time series problem we want to solve is to use multi-dimensional input data, including target series and non-target series data, to infer the prediction results at a single time point or a continuous time point.

Below is the definition of our data:

- Target Series: The data of the target value we want to predict in the previous period.
- Non-target Series: The data of the previous period that are highly correlated with the target value we want to predict.

Given a one-dimensional time series, i.e., $x = \{x_1, x_2, \dots, x_T\} \in \mathbb{R}^{1\times T}$, where T is the length of the previous period. And given N non-target series, i.e., $X = \{x^1, x^2, \dots, x^N\} \in \mathbb{R}^{N\times T}$, with $x_t = \{x_t^1, x_t^2, \dots, x_t^N\} \in \mathbb{R}^{1\times N}$ denotes as a vector of non-target series at time t.

Given a target series, i.e., $y = \{y_1, y_2, \dots, y_T\} \in \mathbb{R}^{1\times T}$, which indicates the data of the target value we want to predict in the previous T time points.

We aim to learn a function $F(\cdot)$ that can find out the nonlinear mapping relationship between target series and predict values, non-target series and predict values. Below is the function presentation:

$$y_{T+1} = F\ (x^1, \dots, x^N, y_1, \dots, y_T) \tag{1}$$

A. *Type & Time extracting method*

In this paper, our main purpose is to extract more detailed features for the non-target data. To achieve this goal, we extract the non-target data's Type and Time separately. We will use three techniques to achieve this, namely Type Convolution, Time Convolution, and Self-Attention.

1) *Type Convolution*

In Figure 1 we split the non-target data into each type series. According to each type series, our model needs to learn the weight value of each corresponding kernel, so that the extracted feature value can remove the more interference when calculating the similarity. Then, each type series and the corresponding kernel are doing inner product to get a new channel. Finally, all channels are merged, and a new feature map extracted from each type is obtained.

$$O_{Type-Conv} = [x^i \odot W_i\ ; \dots\ ;\ x^N \odot W_N], \tag{2}$$

Where $O_{Type\text{-}Conv}$ is the output of type convolution, x^i denotes the i^{th} type series and x^N denotes the final type series. W_i and W_N are the weight of the i^{th} kernel and the weight of the final kernel, respectively. $\odot$ denotes the inner product operation.

2) *Time Convolution*

In Figure 2 like type convolution, we split the non-target data into each time series. According to each time series, our model also needs to learn the weight value of each corresponding kernel. Then, each Time series and the corresponding kernel are doing inner product to get a new channel. Finally, all channels are merged, and a new feature map extracted by each time point is obtained.

$$O_{Time-Conv} = [x_t \odot W_t\ ; \dots\ ;\ x_T \odot W_T], \tag{3}$$

Where $O_{Time\text{-}Conv}$ is the output of time convolution. x_t denotes the t time point series and x_T denotes the last time point series. W_t and W_T are the weight of the t time point's kernel and the weight of the last time point's kernel respectively.

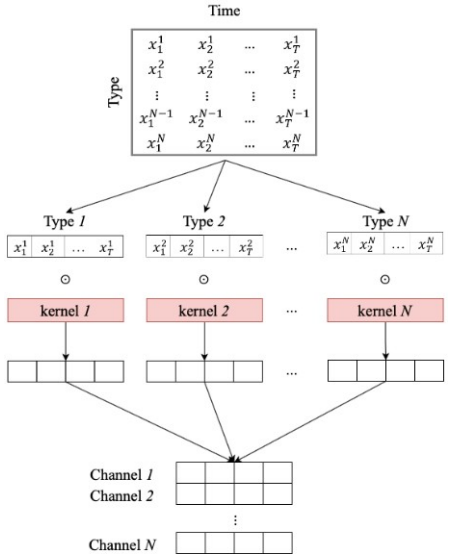

Figure 1. Overview of the type convolution

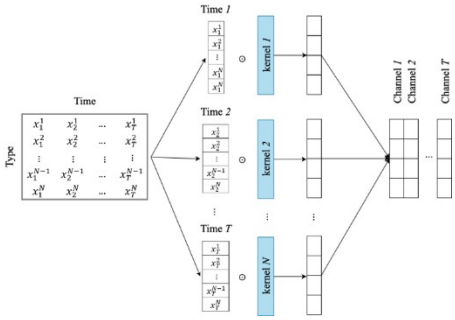

Figure 2. Overview of the time convolution

3) Self-Attention

First, the query and key are multiplied, and then the similarity between the two is calculated through the softmax function, and the output value will fall in the interval of 0~1. To avoid that the multiplied value is too large, resulting in distortion of the output of the softmax function, a scaling operation is added. After obtaining the similarity between the two, multiply the value to get the denoised data. Figure 3 shows the mechanism of the self-attention, and below is the formula:

$$O_{SA} = softmax\left(\frac{QK^T}{\sqrt{n}}\right)V, \quad (4)$$

Where O_{SA} denotes the output of self-attention. Q, K and V is the query, key, and value, respectively. T denotes the transpose.

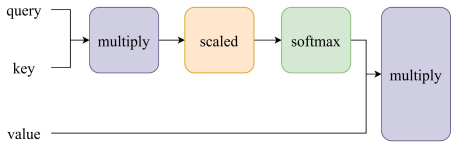

Figure 3. The self-attention mechanism

B. Type & Time CNN-LSTM

Our first model architecture is based on the encoder-decoder and the CNN-LSTM architecture. Use the encoder-decoder based on LSTM and the encoder-decoder based on GRU to learn feature extraction from giving non-target data $X = \{x^1, x^2, \ldots, x^N\} \in \mathbb{R}^{N \times T}$ and target data $y = \{y_1, y_2, \ldots, y_T\} \in \mathbb{R}^{1 \times T}$:

$$O_{lstm} = f_{lstm}(X, y), \quad (5)$$

$$O_{gru} = f_{gru}(X, y), \tag{6}$$

where O_{lstm} and O_{gru} are the output of the LSTM-based encoder-decoder and GRU-based encoder-decoder, respectively. f_{lstm} and f_{gru} are the LSTM function and GRU function.

We concatenate the output of the LSTM-based encoder-decoder and the GRU-based encoder-decoder, and then uses the convolutional layer to capture the best result of the two to obtain the final characteristics that affect the output:

$$O_{conv} = W_{conv}[O_{lstm}; O_{gru}] + b_{conv}, \tag{7}$$

where O_{conv} denotes as the output of convolutional layer and W_{conv} can control the weight between O_{lstm} and O_{gru}. $[O_{lstm}; O_{gru}]$ means that concatenate O_{lstm} and O_{gru}. This method allows our model to obtain two results from two different networks, LSTM and GRU, and select both to learn features that have a greater impact on the output. When the problem we encounter is suitable to be solved by LSTM, the influence of LSTM on the output will be relatively large; conversely, when problems suitable for GRU are encountered, GRU will have greater dominance over the output.

In the input data section, we will also input the target data into the decoder, so that our model can learn the correlation between target data and non-target data for the output at the same time. This is very helpful for improving the accuracy of our output. Figure 4 is our overall architecture diagram.

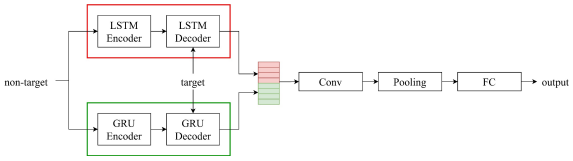

Figure 4. The architecture of extracting method based on the CNN-LSTM

Under our LSTM-based encoder-decoder architecture, we use the CNN-LSTM architecture mentioned in section II, and pair it with our multivariate features and time extraction method. First, in the encoder part, when the non-target is input to our model, it will go with two blocks for feature extraction. One is responsible for the extraction between types, and the other is responsible for the extraction of time. Then it will be input into the self-attention proposed by Transformer to do the correlation calculation, to strengthen the data with strong correlation in the extracted features, and weaken the data with weak correlation, so that the model can remove the interference of noise.

$$O_{Type} = f_{SA}(f_{Type}(X)), \tag{8}$$

$$O_{Time} = f_{SA}(f_{Time}(X)), \tag{9}$$

where f_{Type} and f_{Time} denote as the type extract function and the time extract function, and f_{SA} is the self-attention function. O_{Type} and O_{Time} are the output of the type extract function and the time extract function, respectively.

Our two extraction feature blocks can be deepened in multiple layers to achieve a better extraction effect. To avoid the problem of vanishing gradient caused by the deepening of the network, we use the Residual method to add the original data and the result extracted through multiple layers:

$$Res_{Type} = O_{Type} + X, \tag{10}$$

$$Res_{Time} = O_{Time} + X, \tag{11}$$

Where Res_{Type} and Res_{Time} are the output of Residual method of Type and Time , respectively.

Then, before entering the LSTM network, use the batch normalize layer to normalize the data to prevent the addition of the value from being too large, which will cause the gradient to explode. Finally, the results of feature extraction through time and feature extraction through type are combined, and then input to the decoder for calculation.

$$O_{encoder} = LSTM(normalize(Res_{Type})) + LSTM(normalize(Res_{Time})), \tag{12}$$

Where $O_{encoder}$ is the output of the encoder.

In the decoder, the input data is not only the output of the encoder, but also target data. When processing the results output by the encoder, the same is done through the attention-based CNN-LSTM architecture, and then the non-target results are output through pooling and fully connected layer.

$$O_{non-target} = W_{FC}(AvgPool(LSTM(f_{SA}(Conv(X))))) \quad , \tag{13}$$

Where $O_{non\text{-}target}$, W_{FC}, $AvgPool$ are the output of non-target, weight of fully-connected layer, and average pooling layer, respectively.

In the target data part, we use LSTM to extract its feature value in the previous period:

$$f_t = \sigma(W_f[h_{t-1}; x_t] + b_f), \tag{14}$$

$$i_t = \sigma(W_i[h_{t-1}; x_t] + b_i), \tag{15}$$

$$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o), \tag{16}$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh(W_c[h_{t-1}; x_t] + b_c) \quad , \tag{17}$$

$$h_t = o_t \otimes \tanh(c_t) \, , \tag{18}$$

where $t \leq T$. f_t, i_t and o_t represent the forget gate, the input gate, and the output gate of the LSTM unit at time t, respectively. And they will generate the cell state c_t and the hidden state h_t to memorize the past information. $\otimes$ denotes as the elementwise multiply, and σ denotes as the sigmoid function.

And then we input the non-target result into the LSTM unit at the last time point:

$$f_{T+1} = \sigma(W_f[h_T; O_{non-target}] + b_f), \tag{19}$$

$$i_{T+1} = \sigma(W_i[h_T; O_{non-target}] + b_i), \tag{20}$$

$$o_{T+1} = \sigma(W_o[h_T; O_{non-target}] + b_o), \tag{21}$$

$$c_{T+1} = f_{T+1} \otimes c_T + i_{T+1} \otimes \tanh(W_c[h_T; O_{non-target}] + b_c) \quad , \tag{22}$$

$$h_T = o_{T+1} \otimes \tanh(c_{T+1}). \tag{23}$$

It not only learn the time relevance of the main target data, but also the relevance between non-target data and output. Figure 5 shows that the LSTM-based encoder-decoder architecture.

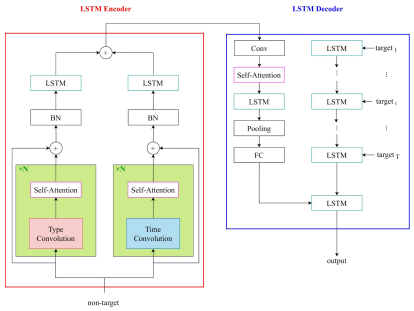

Figure 5. Architecture of the LSTM-based encoder-decoder

C. Type & Time Position-Encoding

In the CNN-LSTM based model, we use LSTM and GRU to capture the before and after changes of the data in continuous time, but these two architectures make our model parameters very large, and it will take too much time to train a large amount of data. Therefore, our second model proposes a lightweight architecture to improve this problem.

Our second model architecture is based on the position-encoding concept proposed by Transformer. Using position-encoding technology to replace LSTM and GRU, our model can easily endow the input data with the concept of time by encoding the data, so that the model can also capture continuous time factors during training.

Figure 6 shows the architecture of the extracting method based on the position-encoding. In the encoder part, our non-target data will enter the type convolution and time convolution of extracting type and time respectively through

our extracting method. However, before entering the time convolution, non-target data will be added with position-encoding to assign the time factor to the original data to achieve the effect of replacing the LSTM.

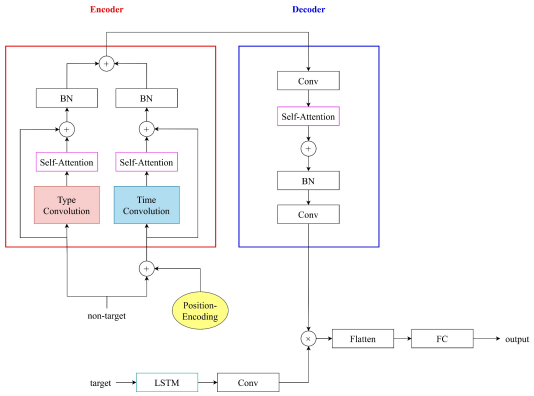

Figure 6. The architecture of the extracting method based on the position-encoding

$$O_{encoded-X} = X + P, \tag{24}$$

where $O_{encoded-X}$ denotes the encoded non-target data, and *P* means the position-encoding.

The features extracted by the two convolutions are used to remove the noise in the data through self-attention, and then added to the original non-target data to normalize, and then input to our decoder after the addition. In the decoder part, our approach is like that of the encoder, but after normalizing, we use a convolution layer to select the best part of the features extracted by the encoder-decoder.

In extracting the features of the target data, we use a simple LSTM to achieve the purpose of reducing parameters. In the same way as the encoder-decoder, a layer of convolution is used to select the best learned feature and multiply it with the features extracted by the encoder-decoder. Finally, after flattening the features learned through the non-target data and the target data, we use the fully connected layer to output our results.

For our non-target data to have a time factor, we need to incorporate the time information into the non-target data. The position $p = \{1, 2, \dots, T\} \in \mathbb{R}^{1\times T}$ is converted into the same dimension $\mathbb{R}^{N\times T}$ as the non-target data through one-hot encoding, and the error caused by the size of the number can be reduced.

$$P = one - hot(p), \tag{25}$$

where p denotes the original position with dimension $\mathbb{R}^{1\times T}$, and P denotes the transformed position with dimension $\mathbb{R}^{N\times T}$.

In Figure 7.After the conversion, we multiply the weight learned from the model to adjust the proportion of each time point. In terms of non-target data, another set of weights is also multiplied, and then the two are added together to get the data with the time factor added.

$$O_{encoded-X} = W_p \times P + W_X \times X, \tag{26}$$

where W_p and W_X denote the weight value for the position-encoding and the non-target data, respectively.

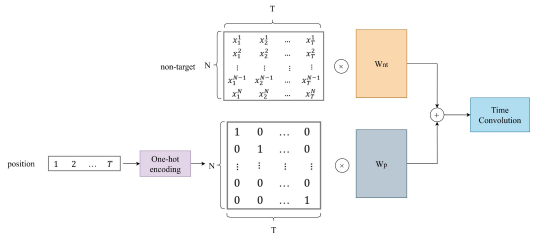

Figure 8. Overview of the position-encoding

D. Type & Time DC-CNN

Our third model is based on the second model. The encoder-decoder architecture is removed, and the number of blocks is modified and deepened, so that our model can improve the prediction accuracy and achieve only a slight increase in the number of parameters. According to the requirements, Figure 9 is our overall architecture diagram.

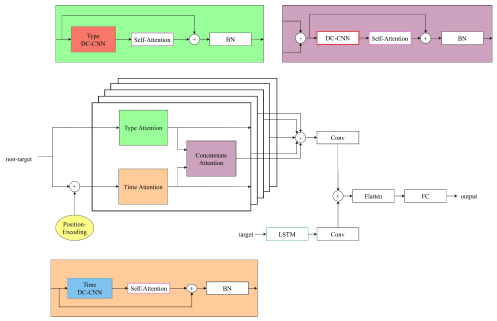

Figure 9. The architecture of extracting method based on the DC-CNN

First, the main block of extracting the non-target data is that data enter type attention to perform feature extraction and denoising for type. On the convolution part, we changed to use the DC-CNN architecture, so that our model parameters can not only be reduced, but also can be retrieved based on time. In addition, before entering time attention, it is added with position-encoding, and convolution also uses DC-CNN instead. Then, input the features extracted by the two into concatenate attention, and extract the features learned from the two at this stage again.

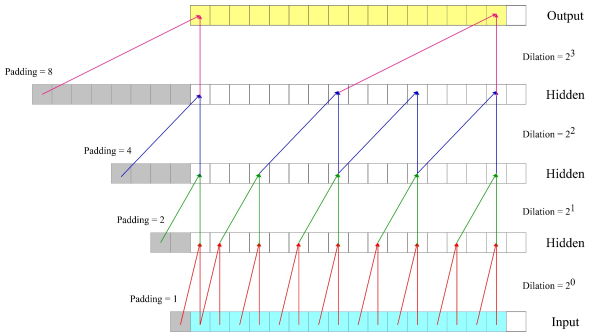

Figure 10. Overview of the dilated causal convolution

The DC-CNN architecture was originally applied in the field of image recognition, but because of its computing mechanism, it is currently widely used in the topic of capturing continuous time features. It includes two major mechanisms, dilated convolution, and causal convolution. These two mechanisms allow us to use CNN to also perform feature extraction on data based on time, eliminating the complex architecture of RNN so that our model can reduce the amount of calculation parameters. In Figure 9, The dilated convolution is mainly to adjust the current operation node. In addition to the current information, it can also refer to the position of the input data, from $\{2^0, 2^1, \ldots, 2^N\}$. Therefore, in our multi-layered architecture, the position of the reference information can be advanced according to the increase in the number of layers to achieve the characteristics of long-term and short-term learning.

III. EXPERIMENTAL RESULTS

A. Datasets and Environment

This chapter will describe our experimental results in details. We use four public datasets to verify our models. Including two weather forecast datasets: SML 2010 indoor temperature and solar radiation prediction, two stock forecast datasets: Tesla stock price and Microsoft stock price. In Table 1, “Time Range” shows the collection time of the datasets. 'Features' represents the dimension of the non-target data of each dataset. “Train”, “Validation” and “Test” indicate the number of training sets, validation sets and test sets that are cut out from each dataset.

The SML 2010 indoor temperature dataset is collected from a monitor system mounted in a house. It corresponds to about forty days of monitoring data, and each data collection time is fifteen minutes apart.And the target is indoor temperature. The solar radiation prediction dataset is meteorological data from the HI-SEAS weather station from four months. Our target series is solar radiation.

The time range of the Tesla stock price dataset is from June 2010 to March 2017. And the target we want to predict is the average price calculated based on the daily highest price and lowest price. The data collected by Microsoft stock price dataset is from March 1986 to August 2016. Each data has the same non-target series as the Tesla stock price dataset. Target series is also to predict the average price per day.

We use computers with Nvidia GTX1050 to train our models. Our OS is Windows 10, and the platform uses Tensorflow 2.1.0 GPU / Keras 2.3.1.

Table 1. List of four datasets

Dataset	Time Range	Features	Train	Validation	Test
SML 2010	40 days	17	3,200	400	537
solar radiation	2016/09 – 2016/12	5	26,160	3,270	3,256
Tesla	2010/06 – 2017/03	5	1,302	200	200
Microsoft	1986/03 – 2016/08	5	6,484	600	600

B. Training Procedure and Evaluation Metrics

In this paper, we use the mean absolute error, denotes as *MAE*, as our loss function:

$$MAE = \frac{1}{T}\sum_{t=1}^{T}|y_t - f_t|, \tag{27}$$

where f_t is our predicted value and y_t is the true value at time t.

In terms of evaluating the accuracy of our models, we use three evaluation metrics to calculate. They are the root-mean- square error denotes as *RMSE*, the coefficient of determination denotes as R^2 and the mean absolute percentage error denotes as *MAPE*:

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_t - f_t)^2}, \tag{28}$$

$$R^2 = 1 - \frac{\sum_{t=1}^{T}(y_t - f_t)^2}{\sum_{t=1}^{T}(y_t - \bar{y})^2}, \tag{29}$$

$$MAPE = \frac{100}{T}\sum_{t=1}^{T}\left|\frac{y_t - f_t}{y_t}\right|, \tag{30}$$

where the RMSE and MAPE are as small as possible, the accuracy of the forecasting value means higher. But the closer of the R^2 is to 1, the better of the forecasting value.

C. Experimental Results

In our experiments, we implemented and compared the input data of the first ten time points to predict the results of the next time point and the results of the five time points after the prediction. We will show the comparison results of our three models with DARNN [3], SeriesNet [8], A_SeriesNet [4], and TA_SeriesNet[9]. Each result is an average of ten training results. The number after each model in the table indicates the number of residuals.

1) Forecasting One Time Point

In Table 2 from the results of SML 2010, it can be found that our three models have higher accuracy for room temperature prediction with a high linear correlation than the other four models, showing that our Type & Time extraction method is very reliable in predicting linear data.

Table 2. The result of the forecasting one time-point on weather dataset

Models	SML 2010			Solar radiation		
	RMSE	R^2	MAPE	RMSE	R^2	MAPE
DARNN	0.049	0.9995	0.16	66.231	0.9457	26.261
SeriesNet (5)	0.694	0.9144	2.81	63.334	0.9504	14.388
A_SeriesNet (5)	0.033	**0.9998**	0.113	62.518	0.9516	13.371
TA_SeriesNet (5)	0.043	0.9996	0.149	63.389	0.9502	19.715
Type & Time CNN-LSTM (4)	0.029	**0.9998**	0.101	68.005	0.9427	16.659
Type & Time position-encoding	0.029	**0.9998**	0.101	**61.007**	**0.9539**	11.133
Type & Time DC-CNN (5)	**0.028**	**0.9998**	**0.099**	61.493	0.9532	**10.429**

In the results of solar radiation, we can first observe the shortcomings of the Type & Time CNN-LSTM model. The effect on a large amount of data is worse than other models; but for our other two models, we can find that even under a large amount.

In Table 3 we show the results of the test set of seven models in the Tesla stock and Microsoft stock data sets. From the results of tesla stock which shows that our Type & Time extraction method can capture both linear and nonlinear data in detail. The value of the change.

Table 3. The result of the forecasting one time point on stock dataset

Models	Tesla stock			Microsoft stock		
	RMSE	R^2	MAPE	RMSE	R^2	MAPE
DARNN	6.159	0.9435	2.115	1.058	0.9532	1.555
SeriesNet (5)	12.03	0.7828	4.398	3.032	0.5821	5.302
A_SeriesNet (5)	4.251	0.9731	1.43	0.826	0.9719	1.187
TA_SeriesNet (5)	5.975	0.9461	2.045	0.815	0.971	1.184
Type & Time CNN-LSTM (4)	4.129	0.9746	1.36	0.832	0.9704	1.2
Type & Time position-encoding	4.319	0.9722	1.448	0.839	0.9698	1.215
Type & Time DC-CNN (5)	**3.988**	**0.9762**	**1.337**	**0.807**	**0.9723**	**1.181**

In the results of Microsoft stock, it can be observed that our Type & Time DC-CNN model is the best among them, showing that this model can smoothly capture the changes in values during the long-term stock training data changes; although the results of the other two models are a little worse than A_SeriesNet, they are also better than DARNN and SeriesNet. It shows that the more complex Type & Time.

2) Forecasting Five Time Points

In Table 4, we can find our Type & Time DC-CNN model. Under such a large amount of training data, whether it predicts short-term or mid-term results, it has better performance than other models. And our Type & Time CNN-LSTM model also shows that for a large amount of training data, the prediction results will be worse than other models.

Table 4. The result of the forecasting five time points on the weather dataset

Models	SML 2010			Solar radiation		
	RMSE	R^2	MAPE	RMSE	R^2	MAPE
DARNN	1.756	0.3122	4.486	73.145	0.9336	35.982
SeriesNet (5)	0.851	0.8709	3.505	70.962	0.9375	22.125
A_SeriesNet (5)	0.084	0.9982	0.289	68.52	0.9417	20.545
TA_SeriesNet (5)	0.103	0.9973	0.35	68.538	0.9419	39.981
Type & Time CNN-LSTM (4)	0.081	0.9982	0.279	75.602	0.9289	30.363
Type & Time position-encoding	0.085	0.9982	0.284	68.206	0.9423	**18.711**
Type & Time DC-CNN (5)	**0.077**	**0.9985**	**0.271**	**67.88**	**0.9428**	19.231

In Table 5 from the results of Tesla stock. Our other two models are slightly inferior to other models in predicting short-term results, showing that they cannot smoothly capture accurate changes in time, so they are still worse than other models in predicting mid-term results.

IV.CONCLUSION

In this paper, we propose a feature extraction method called Type & Time, which can perform detailed feature extraction based on time and feature type, combined with the attention mechanism, making this method compared to the comprehensive extraction methods used by other models have better performance. In addition, we have used three models to verify the advantages of our Type & Time, namely the more traditional Type & Time CNN-LSTM with the most parameters, the lightest Type & Time position-encoding, and the Type & Time DC-CNN that can best extract the time change correlation.

Finally, regarding the number of our model parameters, the best-performing Type & Time DC-CNN has only better parameters than DARNN. Replacing DC-CNN with the DDSTCN architecture used in A_SeriesNet can reduce the number of parameters and doesn't affect the accuracy of the model.

Table 5. The result of the forecasting five time points on the stock dataset

Models	Tesla stock			Microsoft stock		
	RMSE	R^2	MAPE	RMSE	R^2	MAPE
DARNN	8.235	0.8925	2.829	1.522	0.8967	2.318
SeriesNet (5)	12.451	0.7607	4.474	3.235	0.5286	5.617
A_SeriesNet (5)	7.687	0.9045	2.669	1.516	0.8977	2.29
TA_SeriesNet (5)	8.758	0.8829	2.965	**1.433**	**0.912**	**2.119**
Type & Time CNN-LSTM (4)	**7.481**	**0.9102**	**2.553**	1.569	0.8902	2.344
Type & Time position-encoding	7.657	0.9054	2.666	1.575	0.888	2.378
Type & Time DC-CNN (5)	7.536	0.908	2.615	1.437	0.9075	2.162

REFERENCES

[1] Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. ArXiv, abs/1706.03762.

[2] Xu, Kelvin, et al. "Show, attend and tell: Neural image caption generation with visual attention." International conference on machine learning. PMLR, 2015.

[3] Qin, Y.; Song, D.; Cheng, H.; Cheng, W.; Jiang, G.; Cottrell, G. A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17); AAAI Press: Palo Alto, CA, USA, 2017; pp. 2627–2633.

[4] Cheng Y, Liu Z, Morimoto Y. Attention-Based SeriesNet: An Attention-Based Hybrid Neural Network Model for Conditional Time Series Forecasting. Information. 2020; 11(6):305. https://doi.org/10.3390/info11060305

[5] Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural turing machines." arXiv preprint arXiv:1410.5401 (2014).

[6] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).

[7] Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." arXiv preprint arXiv:1508.04025 (2015).

[8] Cho, Kyunghyun, et al. "On the properties of neural machine translation: Encoder-decoder approaches." arXiv preprint arXiv:1409.1259 (2014).

[9] Cheng, Yepeng, and Yasuhiko Morimoto. "Triple-Stage Attention-Based Multiple Parallel Connection Hybrid Neural Network Model for Conditional Time Series Forecasting." IEEE Access 9 (2021): 29165-29179.