Master's Thesis

Meta-learning in movement prediction problem of aperiodic time-series data

NGUYEN BAO LONG

Supervisor: **Ryo Maezono**

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
Information Science

3/2025

Abstract

Analysis and forecasting on time-series data have received great attention from both the research community and businesses due to its popularity and the great benefits it brings in terms of economics and academia. However, while research on periodic time-series data has been expanded and achieved many positive results, aperiodic time-series data such as foreign exchange, stock price has not been studied in depth. Compared to periodic time-series data, this type of data has more complex properties, creating its own difficulties and needs to be solved by specially designed methods.

This paper proposes `Temporal-ML`, a new approach that combines Model-Agnostic Meta-Learning (`MAML`) and Bidirectional Long Shot-Term Memory (`BiLSTM`) to solve the problem of movement prediction (upward, downward) of aperiodic time-series data. The goal of the algorithm is to combine the ability to selectively extract temporal dependencies of `BiLSTM` and the ability to synthesize models with high generalization of `MAML`. Accordingly, `Temporal-ML` not only efficiently extracts temporal dependencies but also extract correlations between different datasets in the context of multi-source data.

In our experiments, we compare `Temporal-ML` and `NHITS` (state-of-the-art (SOTA) model in 2023) on two aperiodic time-series datasets `USD/JPY` (foreign exchange rate between US dollar and Japanese yen, sampled hourly from 2000 to 2024) and `multi-fx` (foreign exchange rate of 60 currency pairs between 18 countries, sampled daily from 2014 to 2024). The results on the two aperiodic datasets show the superiority of the proposed algorithm over `NHITS` on all classification metrics. On the periodic data, `Temporal-ML` achieves results equivalent to the baseline model. These results demonstrate the superiority of the proposed algorithm on aperiodic data as well as the equivalence of the SOTA model on periodic data.

Additionally, this work conducts an ablation study to deeply analyze the influence of each component in `Temporal-ML`. Based on the obtained results, the thesis proves the role of each component as well as the rationality in choosing algorithms in the combination process.

In summary, this work emphasizes the importance of designing a specific method for aperiodic time-series data. The discoveries in this thesis not only help to solve the difficulties in the process of analyzing and predicting aperiodic time-series data, but also directly promote the research community in finding effective solutions on this type of data. Consequently, the research can move from movement prediction to value-specified prediction.

Keywords: Aperiodic time-series data, Meta-learning, `BiLSTM`, `BiLSTM+CNN`, `Transformer`, Foreign exchange

Acknowledgements

I would love to express my gratitude to my thesis advisors, Prof. Ryo Maezono and Associate Prof. Kenta Hongo for their advice, support, and the opportunities they have given to me during my Master's year in JAIST.

The research topic on movement prediction of aperiodic time-series data was brought to me by Assistant Prof. Ichiba Tomohiro, so I would also like to thank him for introducing me to this problem and for the guidance he provided.

This journey would never have begun without the enthusiastic support and guidance from Prof. Le Hoai Bac. Thank you for introducing me to JAIST and for everything.

In addition, I would like to thank everyone in Maezono and Hongo laboratory, fellow students and the laboratory staff, for helping me out whenever I run into trouble with the administrative documents or health.

Lastly, I wish to thank my parents for their never-ending support, motivation, and phone calls, and for helping me to stay connected with my family and friends at home.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

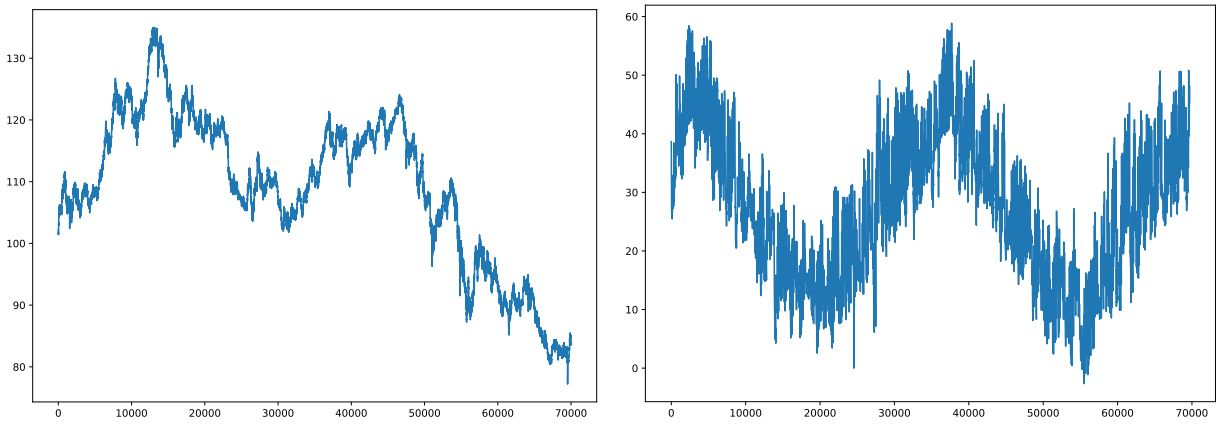
Time-series data is gaining popularity alongside numerical, categorical, and text data not only in terms of data sources (e.g., finance, energy, and transportation), but also in terms of methods for analyzing and forecasting them (e.g., frequency decomposition and methods based on historical data memorization). This makes sense as planning, organizing, and developing business strategies are greatly aided by the analysis and prediction of time-series data.

Two main approaches used in analyzing and predicting time-series data are fundamental analysis and technical analysis [3]. While fundamental analysis concentrates on examining external factors that are hard to capture from past price changes, such as the economic strategies and policies of nations and businesses, technical analysis relies entirely on historical price changes to forecast future trends.

It is challenging to achieve efficient automation of basic analytical methods due to the unstructured of news data. As a result, academics usually concentrate on creating techniques for technical analysis. The forecasting process can now be intelligently automated with the help of machine learning and deep learning techniques, such as Auto-regressive model (1951) [4], Moving average model (2000) [5], Long short-term memory model - `LSTM` (1997) [6], and Transformers (2017) [7].

The aforementioned techniques, are mostly applicable to periodic time-series data, such as traffic, weather, etc. Over time, this kind of data clearly shows a seasonal or cyclical character. Consequently, analysis and prediction can be performed quickly and accurately. This is especially true for existing techniques that use deep neural networks to decompose periods [8–10]. Nevertheless, **aperiodic time-series data** (e.g., stock prices, foreign exchange rates, etc.) has been the subject of very few investigations.

As can be seen in figure 1.1, the primary distinction between aperiodic and periodic time-series data is the lack of a distinct periodicity. In fact, because of seasonal fluctuations in electricity demand, the *Temperature oil* (target variable) attribute of the Electricity Transformer Temperature dataset (`ETT-m2`) [11] shows a noticeable periodicity over time in figure 1.1b. In the meanwhile, a number of external factors influence the *Close price* between the US dollar and the Japanese yen (figure 1.1a). This exchange rate can be significantly impacted by recent economic news. Thus, there is no periodicity in this exchange rate.



(a) Exchange ratio between USD and JPY. (b) *Oil temperature* in `ETT-m2` dataset.

Figure 1.1: 70.000 samples of aperiodic (1.1a) and periodic (1.1b) time-series dataset.

Aperiodic time-series data's primary feature is that it is heavily impacted by external factors, including production strategy and the political and economic policies of nations or businesses. This results in notable modifications to the data's characteristics, which are also referred to as concept drift scenarios [12]. Mathematically, this results in aperiodic time-series data with a continuous and unstable variance. Additionally, because the data no longer exhibits periodicity, it becomes extremely challenging to extract characteristics from it using a sliding window.

In conclusion, there are three primary obstacles in dealing with aperiodic time-series data: (1) External influences including politics, economy, and society have a significant impact on the data; (2) Data has a massive and non-stationary variance; and (3) Data's strong non-periodicity makes feature extraction challenging. **In this study, we focus on solving the three challenges mentioned above in the problem of trend prediction (upward or downward) on aperiodic time-series data.**

1.2 Motivation

The first of the three issues listed above is the most challenging to resolve using technical analysis techniques as it is challenging to extract political and social information from historical data in a data collection. However, according to the efficient market hypothesis [13], historical data itself clearly reflects information about market movements. This implies that, **by carefully examining the transaction history, one can completely grasp market fluctuations and forecast the future** without aggregating political and social information. In addition, studies [14, 15] indicate the dependence between the financial indicators of a certain company and the indicators of other companies. In other words, **integrating analysis of multiple sources of information within the same domain can yield valuable analytical insights for the indicator of interest.**

Ensemble learning is frequently utilized to mitigate the impacts of variation while dealing with the second challenge [16–18]. This type of learning involves dividing the data into several parts, assuming a certain amount of variation in each part, and then learn that variation. This approach is reasonable because it is challenging for a single model to capture all of the variability in the data over an extended period of time. **By analyzing and synthesizing information from multiple sub-models, ensemble learning provides a multidimensional view of the data, which helps the overall model adapt well to strong variance changes.**

Up to now, many models have been proposed to be able to extract features on data in general and time-series data in particular. Typically, Convolutional neural network - `CNN` [19], `LSTM` [6], and attention mechanism [7, 20, 21] have been proposed to extract local features, remember short-term and long-term features, and emphasize vectors in the feature matrix, respectively. **By selectively using these methods, hidden constraints in aperiodic time-series data can be extracted effectively**, helping to overcome the third challenge.

On the other hand, Meta-learning (ML) algorithms are known for their ability to increase the generalization and adaptability of a model on a limited dataset [22, 23]. During training, the optimization process of ML can be viewed as an efficient synthesis of models across sub-tasks (sub-datasets). Based on this ability, **a machine learning models trained by ML algorithms are expected to efficiently synthesize information from multiple data sources/time periods as well as adapt well to the continuous change of variance.**

1.3 Outline

This work is divided into 6 chapters:

- Chapter 1 provides the research context, problem statement, challenges and motivation.
- Chapter 2 presents related works, advantages and disadvantages of each study. In addition, ML is also presented in this chapter, providing the foundation for the proposed method in Chapter 3.
- Chapter 3 presents a new approach for aperiodic time-series data based on the `LSTM`/`LSTM+CNN` feature extraction method and ML model optimization method.
- Chapter 4 sets up experiments, evaluation methods for the proposed method and baseline model.
- Chapter 5 analyzes the results achieved during the experiment.
- Chapter 6 summarizes the main contributions of the study as well as presents future development directions.

Chapter 2

Related Works

In this chapter, we analyze several approaches in handling time-series data, indicating the advantages and disadvantages of each approach. The related studies range from old methods such as ensemble model, deep feature-based approach, to recent methods which aim to decompose the input signal to frequency bands.

2.1 Ensemble models

The variance on time-series data is not constant and often varies greatly over time. This variation is often referred to as the concept of drift scenarios [12]. To increase the stability of the prediction system in the context of continuously varying data variance, instead of using a single model, the ensemble approach proposes using multiple models and combining them to generate predictions.

Instead of training a learner from training data, ensemble methods train a set of learners to solve the same problem. The general form of the ensemble model $g(\mathbf{x})$ is expressed by the aggregation of models $\{h_i, i = 1 \dots T\}$ as follows:

$$g(\mathbf{x}) = \sum_{i=1}^{T} \alpha_i h_i(\mathbf{x}), \text{ with } \sum_{i=1}^{T} \alpha_i = 1 \tag{2.1}$$

In equation 2.1 the summation operation is mentioned abstractly, referring to the process of ensemble. h_i is called the base learner. The base learner can be created from any base learning algorithm (e.g., decision tree, neural network, etc.,). Most ensemble methods use the same base learning algorithm with different training data to train the base learners. This results in base learners of the same type (i.e., the base learners are all decision trees or neural networks, etc.,).

Ensemble models are often better at generalizing than base learners because they combine weak learners, which can predict only slightly better than random, to built a strong learner, which can predict with much better accuracy. Hence, base learners are often called weak learners. Indeed, when increasing levels of noise in the data, ensemble models always give better predictions than a single model.

In addition, the computational cost of creating multiple base learners and combining them is not too large compared to building a single model because when creating a single model, we often have to create many versions of that model during tuning. This is equivalent to generating multiple base learners in the ensemble approach. The process of combining features of an ensemble model is usually not too expensive because the combination strategy is usually very simple (e.g., voting, averaging).

Based on the process of generating base learners, we refer to two main approaches to ensemble methods: `Bagging` and `Boosting` models. Where, `Bagging` generates base learners in parallel while `Boosting` generates base learners sequentially.

2.1.1 Bagging model

The main motivation of `Bagging` is based on exploiting the independence of each base learner. `Bagging` assumes that the model error can be significantly reduced by combining independent base learners. Contrary to the initial conception, the word "bagging", although referring to dividing data into bags, does not come from the word "bag" but is an abbreviation of the word "Bootstrap AGGregatING". As mentioned in the name, `Bagging` consists of two main ideas: "bootstrap" and "aggregation".

`Bagging` uses bootstrap distribution to generate different datasets for training base learners. In other words, it applies bootstrap sampling technique to obtain samples and train base learners on separate samples. Specifically, given n training data points, samples containing n data points are generated by sampling with replacement from n original data points. By repeating this process T times, T samples are generated.

During the aggregation process, `Bagging` uses voting for the classification problem and averaging for the regression problem to aggregate output from base learners. Specifically, a `Bagging` model with T base learners $\{h_i, i = 1 \dots T\}$ in the regression problem will predict instance $\mathbf{x}$ as follows:

$$\hat{y} = \frac{1}{T} \sum_{i=1}^{T} h_i(\mathbf{x}) \tag{2.2}$$

2.1.2 Boosting model

The idea of `Boosting` comes from sequentially improving the model's prediction performance on data that it previously predicted incorrectly. This improvement is done by adjusting the distribution of the data set. We will present a toy example to illustrate this idea.

Given a data set $\mathcal{D}$ drawn from the space $\mathcal{X}$. The space $\mathcal{X}$ consists of three parts $\mathcal{X}_1, \mathcal{X}_2$ and $\mathcal{X}_3$ with equal proportions. Suppose that model h_1 after training on $\mathcal{D}$ can predict well on data samples belonging to $\mathcal{X}_1, \mathcal{X}_2$ and has poor performance on data samples belonging to $\mathcal{X}_3$. At this point, we can correct the error of h_1 by training a model h_2 on the data set $\mathcal{D}'$. In which, $\mathcal{D}'$ is the calibrated dataset from $\mathcal{D}$ to emphasize the samples $x \in \mathcal{X}_3$. After training, h_2 can overcome the weakness of h_1 on the space $\mathcal{X}_3$. By combining these two models, the output model is expected to perform well on all three sub-spaces of $\mathcal{X}$. However, suppose that this model only performs well on $\mathcal{X}_1, \mathcal{X}_3$. At this point, we continue to calibrate the training dataset to obtain $\mathcal{D}''$.

In a nutshell, `Boosting` works by training base learners sequentially and then combining them to make predictions. Later learners focus on improving the errors of earlier learners.

The limitation of ensemble models lies in the process of synthesizing base learners. This process is based on two mechanisms: voting and averaging, corresponding to the classification problem and the regression problem. **It is not difficult to see that these two mechanisms are really simple and rigid. That makes the ensemble approach, although improving the generalization ability compared to training a single model, the generalization ability is not really high.**

2.2 Deep feature-based approach

2.2.1 Convolutional neural network

The starting point of Convolution neural network (`CNN`) is originated from convolution operation. Therefore, when talking about `CNN`, it is impossible not to mention convolution operation. Suppose we use a sensor to locate a moving object. This sensor returns a single value $x(t)$, representing the position of the object at time t. We can retrieve this position at any time. Unfortunately, the returned value of our device is not really accurate but noisy. Therefore, to obtain a more accurate result at a time t, we have to aggregate $x(t)$ and its previous values: $x(a), a = t-1, t-2, etc.$ In fact, the closer a is to t, the more it affects $x(t)$. Therefore, a weighting function is required to weight the values of $x(a)$.

Now, the estimated value of the object's position at time t is calculated using the function $s(t)$ as follows:

$$s(t) = \int_{a=-\infty}^{\infty} x(a)w(t-a)da \tag{2.3}$$

$$= (x * w)(t) \tag{2.4}$$

The above operation is called *convolution* operation. In discrete domains, the convolution operation works as follows:

$$(x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.5}$$

The key point of this operation is to show a local relationship between the object under consideration ($x(t)$) and the objects $x(a)$ with a near t. Inspired by this, the research [19] proposed a `CNN` network, using two components input (corresponding to x) and kernel (corresponding to w) and *cross-correlation* operation instead of the traditional convolution operation. However, [19] still calls it a convolution neural network. Basically, the cross-correlation operation looks quite similar to the convolution operation except for a small change in the index of the input and kernel for convenience in implementation. Specifically, the cross-correlation operation works as follows:

$$(x * w)(t) = \sum_{a=-\infty}^{\infty} x(t+a)w(a) \tag{2.6}$$

It is worth noting that `CNN` can aggregate information not only in one dimension (in our example, the time dimension), but can also aggregate information in two dimensions (for conventional RGB or gray scale images) or three dimensions (for spectral images consisting of multiple single-shot images stacked on top of each other). Here, we present two common applications of `CNN` for one-dimensional and two-dimensional aggregation on time-series data and image data. Note that these data are discrete, so the integral sign in the original equation is replaced by a summation sign. In addition, since farther away positions/timestamps have less influence on the current position/timestamp, a commonly used assumption in `CNN` is that the influence is totally local. That is, the kernel has non-zero values within the influence range and zero values outside the influence range. It can be simply visualized the kernel as a window that slides along the direction of the data to

synthesize information.

Denote I, K as the input and kernel, respectively. For time-series data, the kernel can only slide along the time direction of the data, so the convolution operation is denoted as `Conv1D`. Suppose the data under consideration is the daily close price of a company's stock, the kernel K specifies the local influence over 5 days. At day i, the feature $F(i)$ is synthesized as follows:

$$F(i) = (I * K)(i) = \sum_{m=1}^{5} I(i+m)K(m) \tag{2.7}$$

For normal image data, K becomes a matrix that determines the influence of local pixels on the pixel under consideration and can be slid in both directions from left to right and from top to bottom. Therefore, the convolution operation is denoted as `Conv2D`. Suppose this influence is a rectangular region of size $(M \times N)$, at pixel (i, j), the feature $F(i, j)$ is synthesized as follows:

$$F(i,j) = (I * K)(i,j) = \sum_{m}^{M} \sum_{n}^{N} I(i+m, j+n)K(m,n) \tag{2.8}$$

During training, `CNN` attempts to learn a reasonable influence level to efficiently extract the spatial and temporal features of the input data. The effectiveness of `CNN` comes from two main ideas. First, based on the assumption that local features are more meaningful to the object under consideration than global features, `CNN` reduces its parameter search space from a set of parameters that interact comprehensively with all inputs to a set of parameters that interact only with local features (kernels) but are more meaningful. Suppose the neural network has m inputs and n outputs. By using a kernel of size $k \ll m$, the time complexity of the computation is reduced from $\mathcal{O}(mn)$ to $\mathcal{O}(kn)$ but the efficiency of the network is increased. Second, during learning, `CNN` uses only one kernel. This reduces the space complexity from $\mathcal{O}(mn)$ in traditional neural networks to $\mathcal{O}(k)$.

However, for time-series data in general and aperiodic time-series data in particular, the data properties at any given time depend not only on recent times but also on long-term times in the past. Therefore, `CNN` is only used as a method to support feature extraction in the process of solving problems on time-series data.

2.2.2 Recurrent Neural Network

Instead of operating on a pixel grid like `CNN`, Recurrent neural network (`RNN`) [24] operates on a sequence of values $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}$. Compared to `FullyConnected`, `RNN` can operate on sequences of any length without increasing the size of the network. In other words, `RNN` is able to work on long sequences much more efficiently than traditional neural networks.

Similar to `CNN`, `RNN` also uses a network architecture that allows parameter sharing. However, `CNN` shares parameters within a data sample (the same kernel operates on local regions of a data sample). `RNN` shares parameters by using the same set of parameters for different data samples. Therefore, when a data sample is input to the model, the model output contains information about this data sample and all previous data samples. This makes it possible and efficient to extend the model and apply it to data samples of different lengths.

Specifically, let the function f be the `RNN` network. The function f uses the parameter set θ to extract information from state $\mathbf{s}^{(t-1)}$ to generate state $\mathbf{s}^{(t)}$ as follows:

$$\mathbf{s}^{(t)} = f\left(\mathbf{s}^{(t-1)}; \theta\right) \tag{2.9}$$

Note that, $\mathbf{s}^{(t)}$ is an abstract variable, implying that current information depends on past information. For any sequence of length τ, the equation 2.9 can be implemented $\tau - 1$ times to compress the information in the sequence into $\mathbf{s}^{(t)}$ states.

In typical `RNNs`, the state variable $\mathbf{s}$ is represented by information extracted from the network $\mathbf{h}$ and the data $\mathbf{x}$. Specifically, the current state $\mathbf{h}^{(t)}$ is extracted based on the previous state $\mathbf{h}^{(t-1)}$ (which contains information about the entire previous sequence) and the current data sample $\mathbf{x}^{(t)}$:

$$\mathbf{h}^{(t)} = f\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta\right) \tag{2.10}$$

The equation 2.10 can be rewritten in the form of the $g^{(t)}$ model as follows:

$$\mathbf{h}^{(t)} = g^{(t)}\left(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \ldots, \mathbf{x}^{(2)}, \mathbf{x}^{(1)}; \theta\right) \tag{2.11}$$

As a result, it can be seen that the model g does not need to update its parameters

when learning each sample $x^{(t)}$ like a traditional neural network. Instead, it can use information from a sequence $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}$ to update its parameters through the model f. Thus, using the model f also improves the model's generalization ability to sequences of any length.

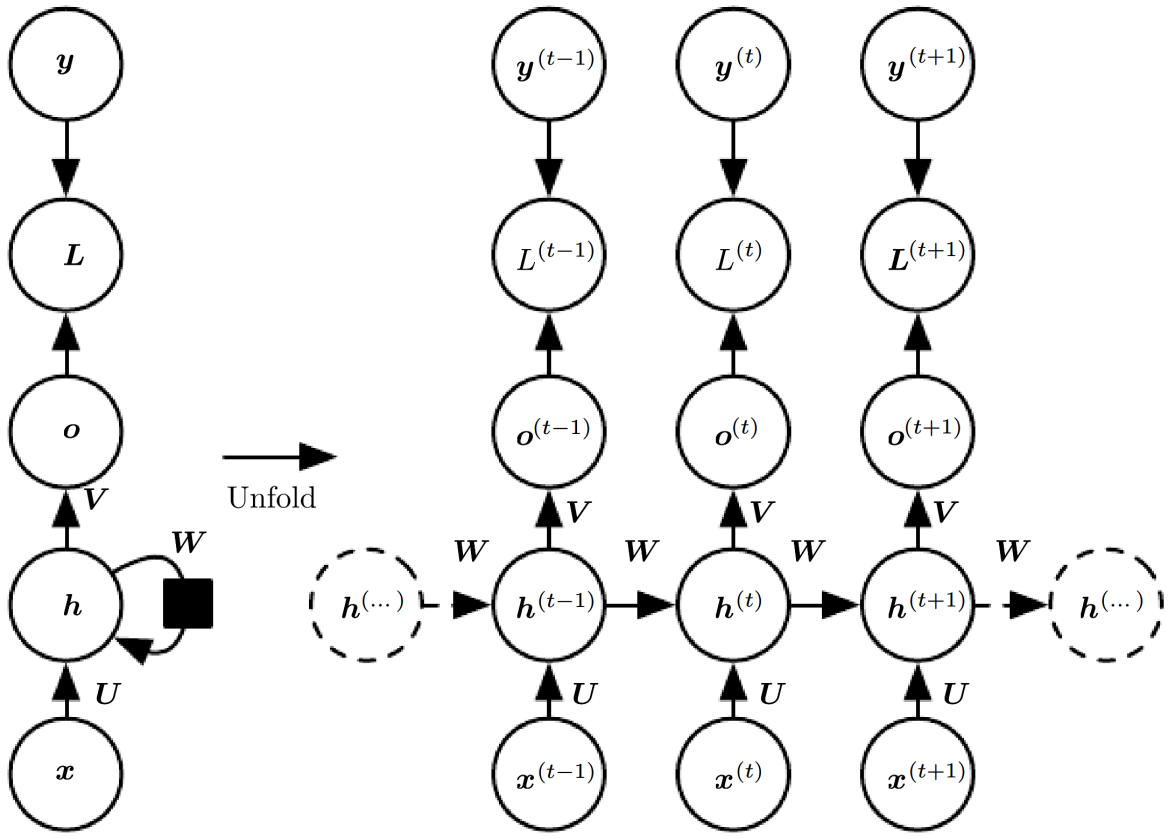


Figure 2.1: An example of computational graph of `RNN` [1].

Based on the idea of recurrent function f presented above, we can design many `RNN` network architectures. Here, we present a simple design with a computational graph as shown in 2.1. Accordingly, the network takes in a sequence of values $\mathbf{x}$ with the goal of mapping these values to outputs o matching $\mathbf{y}$. To do this, the network uses three main sets of parameters: U to map $\mathbf{x}$ to the hidden feature space; W to map the hidden feature of the previous timestamp to the hidden feature space of the current timestamp; V to map the hidden feature to the space containing unnormalized output values. The predicted value o is obtained after the normalization process. The whole process is shown as follows:

$$\mathbf{a}^{(t)} = \mathbf{b} + W\mathbf{h}^{(t-1)} + U\mathbf{x}^{(t)} \quad (\mathbf{b} \text{ is bias vector}) \tag{2.12}$$

$$\mathbf{h}^{(t)} = \tanh\left(\mathbf{a}^{(t)}\right) \tag{2.13}$$

$$o^{(t)} = \text{softmax}\left(\mathbf{c} + V\mathbf{h}^{(t)}\right) \quad (\mathbf{c} \text{ is bias vector}) \tag{2.14}$$

The model loss is calculated using the `CrossEntropyLoss` function (equation 2.15). This error function is used to update the parameters of the function f via back-propagation through time (BPTT).

$$L\left(\left\{o^{(t)}\right\}_{t=1}^{\tau}, \left\{y^{(t)}\right\}_{t=1}^{\tau}\right) = \sum_{t=1}^{\tau} L^{(t)}\left(o^{(t)}, y^{(t)}\right) \tag{2.15}$$

The BPTT technique is implemented by taking the derivative of L with respect to the weights across timestamps from $t = \tau$ to $t = 1$. For convenience in presentation, we rewrite the forward process as follows:

$$\mathbf{h}^{(t)} = f\left(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}, W_h\right) \tag{2.16}$$

$$o^{(t)} = g\left(\mathbf{h}^{(t)}, W_o\right) \tag{2.17}$$

Accordingly, the derivative of the loss function with respect to W_o and W_h is calculated as follows:

$$\frac{\partial L}{\partial W_o} = \sum_{t=1}^{\tau} \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial W_o} \tag{2.18}$$

$$= \sum_{t=1}^{\tau} \frac{\partial L^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial W_o} \tag{2.19}$$

$$\frac{\partial L}{\partial W_h} = \sum_{t=1}^{\tau} \frac{\partial L^{(t)}}{\partial W_h} \tag{2.20}$$

$$= \sum_{t=1}^{\tau} \frac{\partial L^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial W_h} \tag{2.21}$$

In which,

$$\frac{\partial \mathbf{h}^{(t)}}{\partial W_h} = \frac{\partial f\left(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}, W_h\right)}{\partial W_h} + \frac{\partial f\left(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}, W_h\right)}{\partial \mathbf{h}^{(t-1)}} \frac{\partial \mathbf{h}^{(t-1)}}{\partial W_h} \tag{2.22}$$

$$= \frac{\partial f\left(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}, W_h\right)}{\partial W_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f\left(\mathbf{x}^{(j)}, \mathbf{h}^{(j-1)}, W_h\right)}{\partial \mathbf{h}^{(j-1)}} \right) \frac{\partial f\left(\mathbf{x}^{(i)}, \mathbf{h}^{(i-1)}, W_h\right)}{\partial W_h} \tag{2.23}$$

Once we obtain the derivatives as in equation 2.19 and 2.23, we can use gradient-based optimization strategies to update the parameters in the network.

In the forward and backward processes, the computation steps are performed sequentially because they must follow the input order. Therefore, the time complexity of `RNN` is $\mathcal{O}(\tau)$. Since the later derivative values must be stored to compute the previous derivative values (according to the Chain rule), the space complexity of the algorithm is also $\mathcal{O}(\tau)$. Therefore, for long sequences, the computation can be extremely expensive.

2.2.3 Long Short-Term Memory

Suppose the data are in 1-dimensional space $\mathbb{R}$ and the function f in equation 2.23 is of the form $\sigma(w(h+x))$. Then, the product becomes:

$$\prod_{j=i+1}^{t} \frac{\partial \sigma(w(h+x))}{\partial h} = \prod_{j=i+1}^{t} w\sigma'(w(h+x)) \tag{2.24}$$

$$= w^{t-i} \prod_{j=i+1}^{t} \sigma'(w(h+x)) \tag{2.25}$$

In the case of large t and $w \neq 1$, this product will either decrease to 0 or increase to infinity exponentially. Therefore, when working on long sequences, `RNN` is susceptible to the *vanishing gradient* or *exploding gradient* problem. Indeed, the training process of `RNN` requires the application of the Chain rule for all states from $t = \tau$ to $t = 1$. This process involves a lot of matrix and vector multiplication as illustrated in equation 2.25. Therefore, if the elements in that matrix or vector are less than 1, the derivative with respect to time will converge to 0. Conversely, if the elements are greater than 1, the derivative will approach infinity over time.

To solve the above problem, [6] proposed Long Short-Term Memory (`LSTM`). `LSTM` is developed based on three main ideas: Using cell state for each timestamp; Using gates to regulate information flow; Uninterrupted derivative flow technique.

First, each cell of the network maintains a separate state $\mathbf{c}^{(t)}$. This cell state actively synthesizes information obtained from the input $\mathbf{x}^{(t)}$ and information from previous states. This process retains important information and discards unimportant information. As a result, the `LSTM` network can compress more valuable information when learning through a sequence instead of trying to remember all the information like traditional `RNN`.

Second, `LSTM` uses three forget gates, input and output gates, and input node values to regulate the information flow. At timestamp t, these operators take in an extended vector consisting of the hidden state of the previous state $\mathbf{h}^{(t-1)}$ and the current input $\mathbf{x}^{(t)}$. Specifically, they work as follows:

$$\mathbf{f}^{(t)} = \sigma\left(W_f\left[\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}\right]\right) \tag{2.26}$$

$$\mathbf{i}^{(t)} = \sigma\left(W_i\left[\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}\right]\right) \tag{2.27}$$

$$\hat{\mathbf{c}}^{(t)} = \tanh\left(W_c\left[\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}\right]\right) \tag{2.28}$$

$$\mathbf{o}^{(t)} = \sigma\left(W_o\left[\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}\right]\right) \tag{2.29}$$

It is easy to see that the values of these three gates are all in the range $[0, 1]$. Therefore, they have the effect of decreasing or maintaining the level of information in the memory state. Specifically, $\mathbf{f}^{(t)}$ selects and weakens the ineffective information of the previous cell state $\mathbf{c}^{(t-1)}$ by performing element wise multiplication. $\mathbf{i}^{(t)}$ selects the current input information $\hat{\mathbf{c}}^{(t)}$ in a similar way and then updates the current cell state $\mathbf{c}^{(t)}$ by performing element wise addition with $\mathbf{c}^{(t-1)}$ (after removing redundant information). This process takes place in the following equation:

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \otimes \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \otimes \hat{\mathbf{c}}^{(t)} \tag{2.30}$$

The prediction process uses the hidden state $\mathbf{h}^{(t)}$. $\mathbf{h}^{(t)}$ is generated by adjusting the current cell state value $\mathbf{c}^{(t)}$ based on the value of the output gate:

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \otimes \tanh\left(\mathbf{c}^{(t-1)}\right) \tag{2.31}$$

Third, in the BPTT process, instead of following the timestamps through $\mathbf{h}$ like `RNN`, `LSTM` derives the derivatives through $\mathbf{c}$. The operations performed on $\mathbf{c}$ do not involve any weights. Therefore, they do not have a residual amount of weights like the 2.25 equation. In this way, `LSTM` **successfully mitigates the vanishing gradient problem of** `RNN`.

2.3 Frequency decomposition approach

Neural Hierarchical Interpolation for Time Series Forecasting (`NHITS`) [2] is designed to target the prediction of long-horizon time-series data by decomposing the input signal into discrete frequency bands. The structure of `NHITS` consists of S stacks, each stack consisting of B consecutive blocks. At each block, a Multi-layer perceptron (`MLP`) uses historical data to predict itself and future data (see figure 2.2).

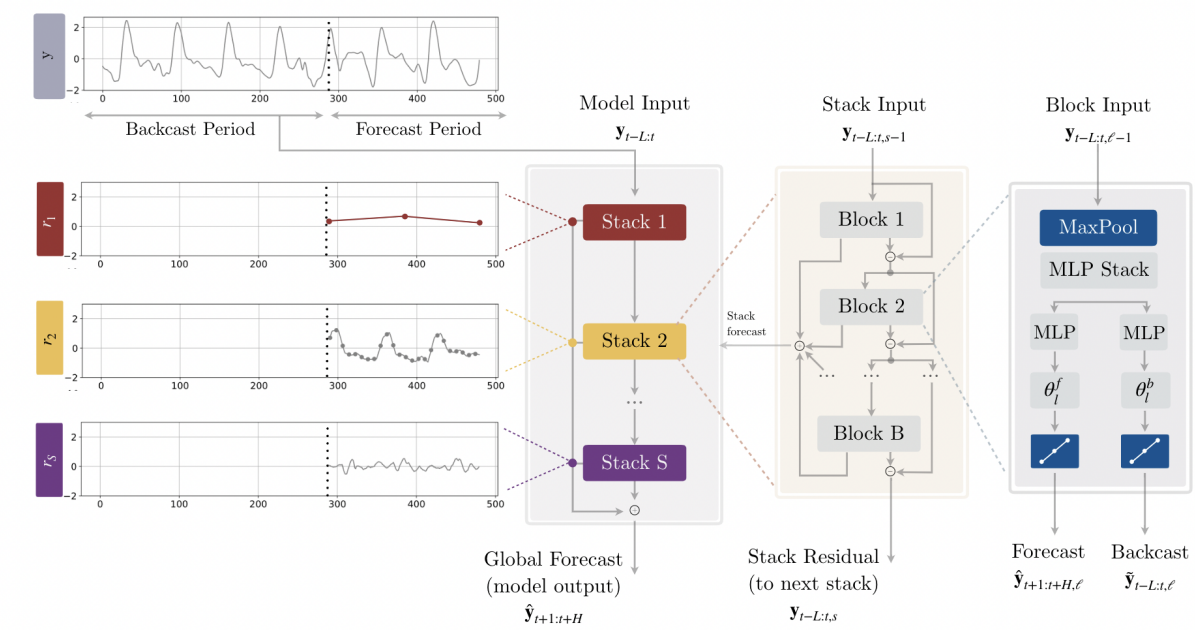


Figure 2.2: `NHITS` architecture [2].

Specifically, at block l, with L historical samples ($\mathbf{y}_{t-L:t,l-1}$), an `MLP` use three techniques, Multi-rate signal sampling, Non-linear regression, and Hierarchical interpolation, respectively, to regress past data and predict future data.

Multi-rate signal sampling. `MaxPool` layer with kernel size l compresses the original data into $\mathbf{y}^{(p)}_{t-L:t,l}$ (equation 2.32). When k_l is large, the amount of data considered in a sliding window is also large, the network will pay more attention to input signal with long wavelengths (low frequencies). When k_l is small, the obtained features are of short wavelengths (high frequencies). By using `MaxPool` layer, `MLP` will be able to work on a certain frequency band, which helps to increase the efficiency of frequency decomposition. Not only stopping at frequency band decomposition, `MaxPool` essentially reduces the size of the input signal, helping to save memory during training and inference.

$$\mathbf{y}^{(p)}_{t-L:t,l} = \mathbf{MaxPool}\left(\mathbf{y}_{t-L:t,l-1}, k_l\right) \tag{2.32}$$

Non-linear regression. After compressing the data, `NHITS` learns the interpolation coefficients using stacked perceptron layers with a nonlinear activation function (`FullyConnected`). The goal of `FullyConnected` is to generate the vectors θ_l^f, θ_l^b (equation 2.34). These are the two interpolation vectors forecast and backcast, which are used to aggregate the output values of block l using the interpolation function $g(\cdot)$. In which, θ_l^f is used to forecast future values and θ_l^b is used to regress the input values.

$$\theta_l^b = \textbf{FullyConnected}^b \left(\mathbf{y}_{t-L:t,l}^{(p)} \right) \tag{2.33}$$

$$\theta_l^f = \textbf{FullyConnected}^f \left(\mathbf{y}_{t-L:t,l}^{(p)} \right) \tag{2.34}$$

Hierarchical interpolation. To forecast H future values, a conventional neural network must redesign the number of output neurons. For transformer models, to increase the number of output samples, it is necessary to increase the number of input samples so that the cross-attention layers can work effectively. This makes the traditional training process very resource-consuming when there is a need to predict large H. `NHITS` solves this problem by using an interpolation equation with pre-prepared interpolation coefficients in the **Non-linear regression** step (equation 2.36). The dimensionality of the interpolation coefficients in each stack is determined by the expressiveness ratio r_l: $|\theta_l| = \lceil r_l H \rceil$. Typically, the expressiveness ratio will be very small in the first stacks and gradually increase towards the end. Accordingly, the stacks can simulate frequencies from low to high. In addition, by using the interpolation function, `NHITS` does not require too much hardware to train the neural network in the case of large H.

$$\hat{\mathbf{y}}_{t-L:t,l} = g\left(\theta_l^b\right) \tag{2.35}$$

$$\hat{\mathbf{y}}_{t+1:t+H,l} = g\left(\theta_l^f\right) \tag{2.36}$$

The output of block l is the forecast value $\hat{\mathbf{y}}_{t+1:t+H,l}$ and the backcast value $\hat{\mathbf{y}}_{t-L:t,l}$. Input of block $l+1$ is the difference between its backcast and its output (2.37).

$$\mathbf{y}_{t-L:t,l+1} = \mathbf{y}_{t-L:t,l-1} - \hat{\mathbf{y}}_{t-L:t,l} \tag{2.37}$$

Summing the forecast values of B blocks as in equation 2.38, we get the forecast value of a stack. Finally, summing the forecast values of the stacks as in equation 2.39, we get the predicted forecast value of the entire network.

$$\hat{\mathbf{y}}^{s}_{t+1:t+H} = \sum_{l=1}^{B} \hat{\mathbf{y}}_{t+1:t+H,l} \tag{2.38}$$

$$\hat{\mathbf{y}}_{t+1:t+H} = \sum_{s=1}^{S} \hat{\mathbf{y}}^{s}_{t+1:t+H} \tag{2.39}$$

By concatenating stacks, each receiving the remainder of the previous stack, **the above architecture is expected to decompose the data into different frequency bands (weekly, daily, even hourly)**. In practice, `NHITS` performs very well for highly periodic datasets such as electricity consumption, weather, traffic. **However, we are aiming for an aperiodic time-series dataset, which has very low, or even non-existent, periodicity**. This poses a huge challenge for `NHITS`.

2.4 Optimization-based Meta-learning

Meta-learning (ML) is a training method that allows a learning model to gain experience by performing many different tasks in the same task distribution. This equips the machine learning model with the ability to generalize highly and adapt quickly to new tasks after only a few training steps with limited training data [22, 23]. With this ability, ML is widely used in tasks that require the ability to fast adapt to new data (e.g. personalization of learning models [25–27], domain adaptation in online learning [**?**, 28]).

For the traditional learning model training method, we train the prediction model $\hat{y} = f_\theta(\mathbf{x})$ on the dataset $\mathcal{D}_t = \{(\mathbf{x}_i, y_i)\}$ of task t. Denote $\mathcal{L}$ is the error function, ϕ is the prior knowledge, the goal of the training process is to minimize the error function on the dataset $\mathcal{D}$ by finding the parameter θ that satisfies:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\mathcal{D}_t; \theta, \phi) \tag{2.40}$$

The ML approach is to try to learn a good prior knowledge ϕ. This is achieved by learning a task distribution $\mathcal{T}$ [22]. Once a good prior knowledge is learned, it can be used for new tasks in the same task distribution $\mathcal{T}$. Mathematically, denote $\mathcal{L}(\mathcal{D}_t, \phi)$ is the error function that represents the effectiveness of using ϕ in training the model on task T, the ML goal is expressed as follows:

$$\min_{\phi} \mathop{\mathbb{E}}_{t\sim\mathcal{T}} \mathcal{L}(\mathcal{D}_t, \phi) \tag{2.41}$$

2.4.1 Model-Agnostic Meta-Learning

For the optimization-based approach, a basic ML algorithm, typically Model-Agnostic Meta-Learning (`MAML`), is learned on multiple tasks t drawn from the same task distribution $\mathcal{T}$ [22]. The data for each task is divided into a support set $\mathcal{D}_t^{support}$ (usually small, around 20%) and a query set $\mathcal{D}_t^{query}$. During the learning process, inner and outer optimization are performed alternately. The goal of inner optimization is to attempt to solve task t by finding an optimal set of parameters θ_t^* on the support set via ϕ:

$$\theta_t^* = \theta_t(\phi) = \arg\min_{\theta} \mathcal{L}_t^{task}\left(\phi, \mathcal{D}_t^{support}\right) \tag{2.42}$$

Where, ϕ is the result of the outer optimization process, which acts as the initial value of θ_t. $\mathcal{L}_t^{task}$ is the error function of the model on the support set of task t.

The goal of outer optimization is to find the optimal prior knowledge ϕ^*, which makes learning a new task in the distribution $\mathcal{T}$ fast and efficient. Specifically, the algorithm uses the optimal parameter sets θ_t^* to perform on the corresponding query set. The errors of the entire model are then aggregated to perform the outer optimization process:

$$\begin{aligned}\phi^* &= \arg\min_{\phi} \sum_t \mathcal{L}_t^{meta}\left(\theta_t^*, \mathcal{D}_t^{query}\right) \\ &= \arg\min_{\phi} \sum_t \mathcal{L}_t^{meta}\left(\theta_t(\phi), \mathcal{D}_t^{query}\right)\end{aligned} \tag{2.43}$$

By performing the above training method, the ϕ^* model will have a high level of generalization across different tasks, and can quickly respond to a new task after only a few training steps.

In the inference phase, the initial values for the model parameters are assigned ϕ^*. The model is then adapted quickly to the support set and performed on the query set. The results on the query set are the model output.

2.4.2 Meta-SGD

As for the `Meta-SGD` algorithm, study [29] shows that using a small, fixed local learning rate over time or a decreasing local learning rate over time is only suitable for the context of training a model with a large dataset over a long period of time. In the context of limited labeled data but the model needs to adapt quickly to new data sets, such hyperparameter selection method is no longer suitable.

The [29] study also proposes a new approach that allows self-tuning and local hyper-

parameter optimization. Accordingly, in addition to optimizing prior knowledge (ω), the algorithm also considers the inner learning rate α as an learnable parameter. By initializing α as a matrix of the same size as θ, the algorithm aims to update both the direction and the learning rate for each weight element in θ by adjusting α at the outer optimization. Subsequently, tasks use α by multiplying (element-wise product) this quantity by the local error function derivative. Accordingly, `Meta-SGD` not only makes learning models adapt quickly on local data sets, but also contributes greatly to personalizing the learning model for each task.

One disadvantage of optimization-based ML method lies in solving equation 2.43 which requires massive overhead to compute and maintain a Hessian matrix. Even so, ML algorithms still achieve a high accuracy in handling many problem in which the quick adaptation or effective model synthesis are required.

Chapter 3

Methodology

We propose `Temporal-ML`, a ML-based method which consists of two main components that work in parallel: (1) - Temporal feature extraction; (2) - Models' parameter synthesis. The overview of the method is illustrated in figure 3.1 and the detail is presented in algorithm 1. In the *Temporal feature extraction* section, we propose to use the features extracted from `BiLSTM` network. In the *Effective synthesis of model's parameters* section, `MAML` is utilized to synthesize the parameters of the models.

Due to the contribution of `BiLSTM` features, we expect to effectively extract hidden temporal features from aperiodic data. By using `MAML` in the weight aggregation process, the proposed method is expected to be a reasonable and effective alternative to traditional ensemble models in effectively synthesizing external factors, minimizing the impact of variance variation, and efficiency capturing hidden long-term dependencies in the past.

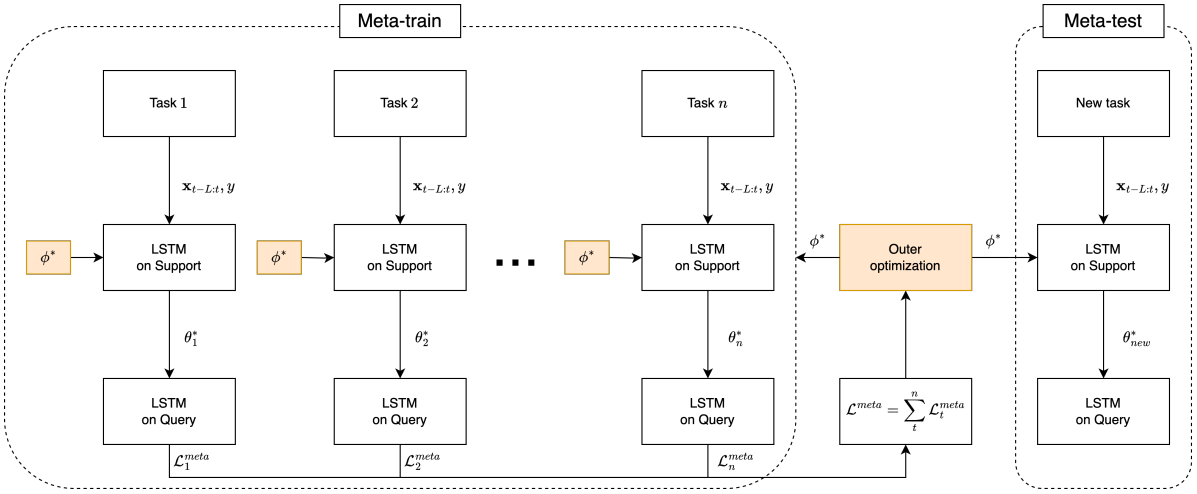


Figure 3.1: The full-flow of meta-training and meta-testing on multi-fx data. Each currency pair is regarded as a task.

Algorithm 1 Temporal-ML

Initialize ϕ_0

for round $r = 1, 2, ...$ **do** ▷ Outer loop

  Sample a subset T_r of m tasks

  for task $t \in T_r$ **do** ▷ Inner loop

    Initialize inner weight $\theta_t^{(0)} \leftarrow \phi_{r-1}$

    Forward: For all data point $\mathbf{x}$ in $\mathcal{D}_t^{support}$

$$\begin{aligned} \mathbf{x}' &= \textbf{FullyConnected}\left(\mathbf{x}\right) \\ \mathbf{h}_{LSTM} &= \textbf{BiLSTM}\left(\mathbf{x}'\right) \\ \hat{y} &= \textbf{FullyConnected}\left(\mathbf{h}_{LSTM}\right) \end{aligned}$$

    Backward and aggregate meta loss: ▷ Inner optimization

$$\begin{aligned} &\mathcal{L}_t^{task}\left(\theta_t^{(e-1)}, \mathcal{D}_t^{support}\right) = \textbf{CrossEntropyLoss}\left(\mathbf{y}, \hat{\mathbf{y}}\right) \\ &\theta_t^{(e)} \leftarrow \theta_t^{(e-1)} - \alpha \nabla_\theta \mathcal{L}_t^{task}\left(\theta_t^{(e-1)}, \mathcal{D}_t^{support}\right) \\ &\mathcal{L}^{meta} \leftarrow \mathcal{L}^{meta} + \mathcal{L}_t^{meta}\left(\theta_t^{(e)}, \mathcal{D}_t^{query}\right) \end{aligned}$$

  Outer optimization at round r: ▷ Outer optimization

$$\phi_{r+1} \leftarrow \phi_r - \beta \nabla_\phi \mathcal{L}^{meta}$$

3.1 Data preparation

`Temporal-ML` uses ML algorithms to train the model. Therefore, the data must be reorganized so that the ML algorithms can work. In case the data includes many different datasets belonging to the same field, each dataset will be considered a task. **The goal of using multiple datasets of the same domain is to utilize the corelation between them** as indicated in study [14, 15]. In case the data includes a single dataset, it is necessary to divide this dataset into subsets corresponding to separate tasks. **In this way, we aim to enhance the learning of information about external factor reflected in single dataset** [13]. In summary, the prepared dataset includes n tasks: $\mathcal{D} = \{\mathcal{D}_t\}_{t=1}^{n}$. The data at each task is divided into support and query sets: $\mathcal{D}_t = \left\{\mathcal{D}_t^{support}, \mathcal{D}_t^{query}\right\}$.

A data sample consists of pairs of values $(\mathbf{x}_{t-L:t}, y)$. In which, $\mathbf{x}_{t-L:t}$ includes L historical values from time t back; $y \in \{0, 1\}$ is the data label, showing the decreasing or increasing trend of the data sample x_{t+1} compared to x_t. Depending on each problem and

the implementation, the elements in $\mathbf{x}_{t-L:t}$ can be a matrix or a vector. For example, for stock data, $\mathbf{x}_{t-L:t}$ can contain L vectors $\vec{x}_i$ = (open, low, high, close) or can be a vector of close price values only.

3.2 Temporal feature extraction

Feature extraction is performed using `BiLSTM`. Specifically, each element in the matrix $\mathbf{x}_{t-L:t}$ (abbreviated as $\mathbf{x}$) is passed through a `FullyConnected` layer whose output is larger than the dimension of $\vec{x}_i, i \in [t-L, t]$ to obtain vector $\vec{x}'_i$ (equation 3.1). Accordingly, the data characteristics are expressed more deeply and clearly. These features are then passed through the `BiLSTM` network (equation 3.2) to selectively extract long-term temporal dependencies ($\mathbf{h}_{BiLSTM}$). At the end, $\mathbf{h}_{BiLSTM}$ is passed through a classification head of the neural network (equation 3.3) to predict the label of $\mathbf{x}$.

$$\mathbf{x}' = \mathbf{FullyConnected}\,(\mathbf{x}) \tag{3.1}$$

$$\mathbf{h}_{BiLSTM} = \mathbf{BiLSTM}\,(\mathbf{x}') \tag{3.2}$$

$$\hat{y} = \mathbf{FullyConnected}\,(\mathbf{h}_{BiLSTM}) \tag{3.3}$$

`LSTM` and `BiLSTM` both maintain cell-state values to selectively store temporal dependencies, as well as to mitigate the vanishing gradient problem during training. Therefore, these networks are well suited for solving time-series data problems. However, for long input sequences, `LSTM` has difficulty in exploiting global context (the entire input sequence) and often faces the problem of local information loss (the further the input, the easier it is to forget). These two problems are effectively solved by `BiLSTM`. With the ability to extract and synthesize information from both sides of the input sequence, `BiLSTM` **can capture the entire context with minimal information loss for distant inputs. In the context of the movement prediction of aperiodic time-series data problem,** `BiLSTM` **is expected to successfully capture continuous value variation as well as not miss important information at distant inputs**. Therefore, it is a more reasonable choice in extracting temporal constraints.

3.3 Effective aggregation of models' parameters

We use `MAML` as presented in the algorithm 1 to train and aggregate the weights of the models of all tasks via the `CrossEntropyLoss` (equation 3.4).

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{\dim(\mathbf{y})} \sum_{i=1}^{\dim(\mathbf{y})} (\mathbf{y}_i \log \hat{\mathbf{y}}_i) + (1 - \mathbf{y}_i) \log (1 - \hat{\mathbf{y}}_i) \tag{3.4}$$

As mentioned in section 2.4, parameter optimization in the ML approach is to solve the two equations 2.42 and 2.43 using optimization methods on the support set and query set. Specifically, the optimization process includes many global steps (outer optimization), performed on all tasks participating in training. Each global step includes many local steps (inner optimization) performed on each individual task. At global step r, the eth local optimization process on the support set of task t proceeds as follows:

$$\begin{cases} \theta_t^{(0)} & = \phi_{r-1} \\ \theta_t^{(e)} & = \theta_t^{(e-1)} - \alpha \nabla_\theta \mathcal{L}_t^{task} \left(\theta_t^{(e-1)}, \mathcal{D}_t^{support} \right) \end{cases} \tag{3.5}$$

In which, ϕ_{r-1} is the result of the $r-1$ outer optimization process, α is the inner learning rate, $\mathcal{L}_t^{task}$ evaluates the use of $\theta_t^{(e-1)}$ on $\mathcal{D}_t^{support}$.

Next, the outer optimization process is performed by aggregating the losses on the query set of the tasks and optimizing on it (equation 3.6).

$$\begin{cases} \phi_0 = \text{Random Initialization} \\ \phi_r = \phi_{r-1} - \beta \nabla_\phi \sum_{t=1}^{n} \mathcal{L}_t^{meta} \left(\theta_t^*(\phi), \mathcal{D}_t^{query} \right) \end{cases} \tag{3.6}$$

Where, β is the outer learning rate, $\theta_t^*(\phi)$ is the local optimized weight obtained from equation 3.5, $\mathcal{L}_t^{meta}$ evaluates the use of $\theta_t^*(\phi)$ on $\mathcal{D}_t^{query}$.

Assuming the algorithm runs E steps in inner optimization, the derivative quantity at equation 3.6 is rewritten as follows (the notations of dataset are removed):

$$
\begin{aligned}
\nabla_\phi \sum_{t=1}^{n} \mathcal{L}_t^{meta}\left(\theta_t^*(\phi)\right) =& \nabla_\phi \sum_{t=1}^{n} \mathcal{L}_t^{meta}\left(\theta_t - \alpha \nabla_\theta \mathcal{L}_t^{task}\left(\theta_t\right)\right) \\
=& \sum_{t=1}^{n} \frac{\partial \mathcal{L}_t^{meta}\left(\theta_t^{(E)}\right)}{\partial \theta_t^{(E)}} \frac{\partial \theta_t^{(E)}}{\partial \phi} \\
=& \sum_{t=1}^{n} \nabla_\theta \mathcal{L}_t^{meta}\left(\theta_t^{(E)}\right) \prod_{j=0}^{E-1} \left[\mathbb{I} - \alpha \nabla_\theta^2 \mathcal{L}_t^{task}\left(\theta_t^{(j)}\right)\right]
\end{aligned} \tag{3.7}
$$

The presence of the product of second order derivatives in the equation 3.7 makes the derivation process complicated because it requires a lot of overhead to maintain Hessian matrices. Therefore, the number of computational steps to find θ^* needs to be limited. In practice, methods using ML [25–27, 29, 30] often choose $E \in [1, 5]$.

Hybrid ensemble models have been widely used in time-series processing problems and have been experimentally proven to be more accurate than standard time-series models because they can synthesize the strengths of many sub-models [3]. However, current ensemble model synthesis forms are still very rigid because they can only synthesize based on the final results (e.g. voting mechanism of bagging models) and semi-final results (e.g. stacking models). **From the perspective of ensemble models, equation 3.6 can be considered an optimization-based method of synthesizing sub-models, which helps to take advantage of the feature extraction capabilities of each model. Moreover, equation 3.6 produces a fast-adaptive model which is extremely suitable for the context of unstable variance in data**. In other words, the synthesized model can quickly adapt to data with new variance as well as extract features at a deeper level, which significantly improves the prediction ability compared to traditional ensemble models.

Chapter 4

Numerical Experiment

Chapter 4 describes experiments to evaluate `Temporal-ML` and compare it with `NHITS` on large and popular datasets. The experiments are performed on aperiodic data consisting of a single dataset and aperiodic data consisting of multiple datasets to verify the ability to extract market information as well as the ability to aggregate information from multiple sources. Periodic data is also used to increase the objectivity of the evaluations. In addition, we perform the removal/replacement of components of `Temporal-ML` to analyze the capabilities of these components in detail.

4.1 Dataset description

Foreign exchange in particular and financial indices in general are typical data types for aperiodic time-series data. Therefore, we choose this type of data to test the model. Specifically, we configure two datasets using foreign exchange data. `USD/JPY` dataset consists of only data of the exchange rate between US dollar and Japanese yen. The data is sampled hourly from 2000 to 2024, including 4 attributes: *open, low, high, and close price*. **By carefully examining the transaction history as well as effectively aggregating data characteristics, we expect to forecast the movement of price without utilizing external information.**

`multi-fx` dataset consists of 60 currency pairs made of 18 countries: Australia, Canada, Switzerland, Denmark, EU, United Kingdom, Hong Kong, Iceland, Japan, Norway, New Zealand, Singapore, Sweden, Turkey, United States, Mexico, China, South Africa. The data has similar attributes to `USD/JPY` and sampled daily from 2014 to 2024. **By integrating analysis of multiple sources of information in the domain of FX, we expect to obtain valuable analytical insights for the indicator of interest.**

In addition, we used two periodic datasets: Electricity Transformer Temperature

(ETT-m2) [11] and Weather (WTH) [31]. ETT-m2 dataset consists of 7 data fields, measuring the parameters of a transformer in a province of China every 15 minutes from July 2016 to July 2018. WTH dataset consists of 12 data fields, recording the weather parameters at the Weather Station of the Max Planck Biogeochemistry Institute in Jena, Germany every 10 minutes in 2020. These two datasets exhibit very strong periodicity, which NHITS has been very successful in predicting. Experiments on them provide a more comprehensive comparison of the proposed method's capabilities against NHITS.

4.2 Temporal-ML

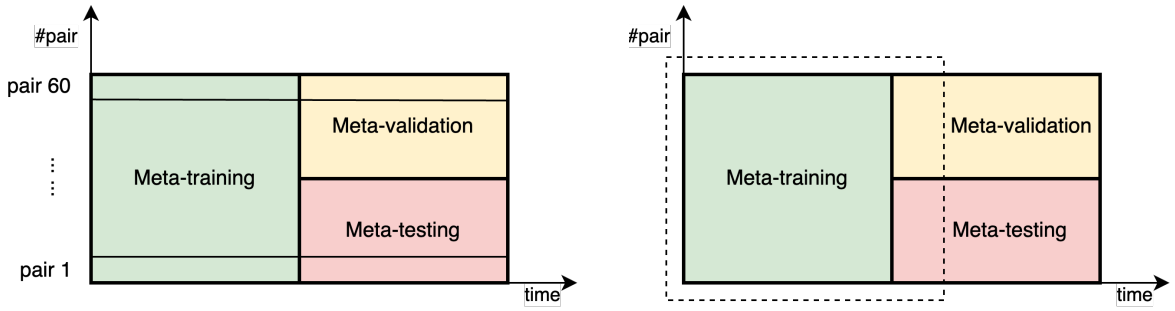
4.2.1 Data pre-processing

As mentioned in section 3.1, to use ML, the datasets need to be structured into separate tasks. In each task, the support set accounts for 20% of the data, the query set accounts for 80% of the data. For multi-fx, each currency pair is divided into 2 datasets, corresponding to 2 tasks. Therefore, 60 currency pairs form 120 tasks. For the remaining datasets, each dataset is divided into time series, each corresponding to a task. During the whole process, 50% of the tasks are used in the meta-training process, 25% of the tasks are used in the meta-validation process, and the remaining tasks are used in meta-testing. Statistics for tasks by dataset are presented in table 4.1.

Table 4.1: Statistics on datasets.

Dataset	Attribute	Task	Sample	Sample/Task
USD/JPY	4	60	150,175	2,503
multi-fx	4	120	154,440	1,287
ETT-m2	7	48	69,680	1,452
WTH	12	40	35,064	877

Since the datasets are all time-series, the training data should not reveal any information about the future. Therefore, the 120 tasks of the multi-fx dataset are divided as shown in 4.1a. For the remaining datasets, the data used in the training, testing, and validation must be in chronological order from past to future.



(a) (b)

Figure 4.1: (a): Splitting data for `multi-fx`. (b): Pre-process for `multi-fx` (dash line rectangle accounts for all training and 20% data of validation and testing tasks).

The `z-score` preprocessing is then performed on the entire training data and the support sets of the test and validation data for each currency pair (see figure 4.1b) since these are known data. Specifically, the preprocessing of attribute X is performed as follows:

$$\mu_X = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (n \text{ là số mẫu của thuộc tính } X) \tag{4.1}$$

$$\sigma_X = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_X)^2} \tag{4.2}$$

$$X_{new} = \frac{X - \mu_X}{\sigma_X} \tag{4.3}$$

μ_X and σ_X obtained from the equations 4.1 and 4.2 are used to normalize the attribute X in the query set of the meta-validation and meta-testing processes.

4.2.2 Metric evaluation

The study uses *Accuracy, Precision, Recall*, and *F1-score* to evaluate the models (equations 4.4, 4.5, 4.6, 4.7). In which, *Accuracy* is used to calculate the ratio between the number of correctly classified samples and the total number of predicted samples, *Precision* measures the confidence level of the model when it labels a data sample, *Recall* tests the rate of misclassified data samples of a classifier, *F1-score* checks the level of bias of the model on major labels by taking the harmonic average over *Precision* and *Recall*.

$$acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.4}$$

$$P = \frac{TP}{TP + FP} \tag{4.5}$$

$$R = \frac{TP}{TP + FN} \tag{4.6}$$

$$F1 = \frac{2PR}{P + R} \tag{4.7}$$

$$\tag{4.8}$$

Where, TP, TN, FP, FN are the number of *true_positive, true_negative, false_positive, false_negative*, respectively.

Suppose the `Temporal-ML` algorithm is evaluated on a dataset $\mathcal{D}$ consisting of a attributes and divided into n tasks. The evaluation is performed by letting the model predict the movement of each attribute with all attributes as input, then performing two aggregation steps: (1) - Aggregation on tasks; (2) - Aggregation on attributes. This evaluation process is designed based on [2].

Aggregation on tasks. Consider metric m when the model predicts any attribute k ($1 \leq k \leq a$). After predicting attribute k, we obtain n values: $\{m_1^{(k)}, \ldots, m_n^{(k)}\}$. The process of synthesizing metrics m of n tasks is performed as follows:

$$\bar{m}^{(k)} = \frac{1}{n} \sum_{i=1}^{n} m_i^{(k)} \tag{4.9}$$

$$s_m^{(k)} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (m_i^{(k)} - \bar{m}^{(k)})^2} \tag{4.10}$$

Aggregation on attributes. The model predicts all a attributes and obtains a metrics: $\left\{ \left(\bar{m}^{(k)} \pm s_m^{(k)} \right) \right\}_{k=1}^{a}$. The process of synthesizing metric m of a attributes is as follows:

$$\bar{m} = \frac{1}{a} \sum_{k=1}^{a} \bar{m}^{(k)} \tag{4.11}$$

$$s_m = \sqrt{\frac{1}{a} \sum_{k=1}^{a} (s_m^{(k)})^2} \tag{4.12}$$

4.2.3 Fine-tuning

In our implementation, we use a `FullyConnected` layer of 16 units with a `ReLU` activation function to represent deeper and more explicit features. This feature is then passed to a `BiLSTM` block. The `BiLSTM` block consists of 32 hidden units, the last two outputs of which are concatenated to form a final vector. This vector is then passed through a binary classification layer with a `Sigmoid` activation function.

Table 4.2: Search space for fine-tuning our method.

Hyper-parameter	Seach space
Inner batch size (samples/batch)	{32}
Inner training step	{3}
Outer batch size (tasks/batch)	{5}
Outer training step	{100}
Lookback window	{10, 20, 30}
Inner learning rate	{0.001, 0.00325, 0.0055, 0.00775, 0.01}
Outer learning rate	{0.001, 0.00325, 0.0055, 0.00775, 0.01}

The fine-tuning process of ML algorithms involves many hyper-parameters such as inner batch size, outer batch size, inner training step, outer training step,... To make fine-tuning easy, we fix most of the parameters and only fine-tune the size of lookback window, inner and outer learning rate. Details are presented in table 4.2.

4.3 Baseline model

4.3.1 Data pre-processing

This study uses the `NHITS` algorithm (AAAI 2023) as the baseline model. `NHITS` uses the entire set of attributes to predict the next trend for an attribute in a dataset. The algorithm aims to decompose the frequency bands and combine them to predict the future. For the `USD/JPY`, `ETT-m2`, and `WTH` datasets, the data in each set is divided into a training set, a validation set, and a test set with a ratio of 6:2:2, respectively.

We preprocess the training data with `z-score` as mentioned in subsection 4.2.1. Then, μ_X and σ_X obtained from 4.1 and 4.2 equations are used to normalize the attribute X in the validation set during tuning. Once the best hyper-parameters are selected, the training and validation data are renormalized from scratch, and then trained with the best set of hyper-parameters to predict the test set.

For the `multi-fx` dataset, since `NHITS` does not have a mechanism to aggregate models across different datasets, we repeat the training process (training, tuning, testing on 60%, 20%, 20% of the data, respectively) for each currency pair, then aggregate metrics across currency pairs to obtain the final evaluation.

In addition, a stochastic prediction model is also used for comparison. The stochastic model generates predicted values of 0 and 1 according to a uniform distribution. The process of synthesizing metrics of this model is similar to `NHITS`.

4.3.2 Metric evaluation

`NHITS` also uses similar metrics to `Temporal-ML` and is evaluated on the dataset $\mathcal{D}$ with a attributes and must perform predictions on each attribute. Considering metric m, after predicting attribute k, we obtain a values: $\{m_1^{(k)}, \ldots, m_a^{(k)}\}$. The process of synthesizing metric m of a attributes is performed as follows:

$$\bar{m} = \frac{1}{a} \sum_{k=1}^{a} m^{(k)}$$

$$s_m = \sqrt{\frac{1}{a} \sum_{i=1}^{n} (m^{(k)} - \bar{m})^2}$$

As mentioned, `NHITS` does not have a mechanism for model aggregation when testing on `multi-fx` data. Therefore, we test `NHITS` on each currency pair in `multi-fx` and aggregate metrics like `Temporal-ML`.

Table 4.3: Search space for fine-tuning NHITS.

Hyper-parameter	Seach space
Random seed	{1}
Number of stacks	{3}
Number of blocks in each stack	{1}
Activation function	{ReLU}
Batch size	{256}
Epoch	{500}
Lookback window	{10, 20, 30}
Pooling kernel	{[2,2,2], [4,4,4], [8,8,8], [8,4,1], [16,8,1]}
Stacks' coefficients	{[168,24,1], [24,12,1], [180,60,1],[40,20,1], [64,8,1]}
Number of MLP layers	{1,2}
Learning rate	{0.001, 0.00325, 0.0055, 0.00775, 0.01}

4.3.3 Fine-tuning

We rely on [2] to define the search space (table 4.3) for parameter fine-tuning, as well as to fine-tune the model architecture. For parameters not mentioned in the table, we use the default values of the `NHITS` implementation in the `NeuralForecast` library [32]. The best results of fine-tuning process are selected and reported in this study.

4.4 Fairness in evaluation

Let m be the number of data samples in each task, the total data of training, validating, and testing is: mn. For `NHITS`, the model is trained, validated, and tested on 60%, 20%, 20% of the data, respectively:

$$\begin{aligned} \text{Train: } & 0.6mn \\ \text{Validation: } & 0.2mn \\ \text{Testing: } & 0.2mn \end{aligned}$$

For `Temporal-ML`, the knowledge enrichment process for the model uses the entire training data and the support set of the testing/validation data. The testing/validation process is performed on the query set:

$$\begin{aligned} \text{Train: } & 0.5mn + 20\% \left(\frac{m}{2}n\right) = 0.6mn \\ \text{Validation: } & 80\% \left(\frac{m}{2}\frac{n}{2}\right) = 0.2mn \\ \text{Testing: } & 80\% \left(\frac{m}{2}\frac{n}{2}\right) = 0.2mn \end{aligned}$$

Therefore, with the above data division, we achieve fairness in the number of data samples during testing.

4.5 Ablation experiment

We evaluate each component in `Temporal-ML` by removing or replacing it from the algorithm and compare the results to the original algorithm. Specifically, we conduct experiments on two groups of algorithms: (1) - Model without `BiLSTM`; (2) - Model without ML. Regarding the evaluation of these models, ML-based models are evaluated as

Temporal-ML, the remaining models are evaluated as NHITS.

4.5.1 Model without BiLSTM

We study the temporal feature extraction ability of the algorithms by replacing BiLSTM in Temporal-ML with BiLSTM+CNN, Transformer algorithms [7]. **The goal of replacing BiLSTM with BiLSTM+CNN is to test the local feature extraction ability of the extractor** by combining the features of BiLSTM and CNN then concatenating them and performing classification (equations 3.2, 4.13–4.15).

$$\mathbf{h}_{CNN} = \textbf{Convolution1D}\left(\mathbf{x}'\right) \tag{4.13}$$

$$\mathbf{h} = \textbf{Concatenate}\left(\mathbf{h}_{LSTM}, \mathbf{h}_{CNN}\right) \tag{4.14}$$

$$\hat{y} = \textbf{FullyConnected}\left(\mathbf{h}\right) \tag{4.15}$$

Transformer is composed of two encoders. Each encoder uses three attention heads, and the key vector dimension is 256. The FeedForward part of the encoder uses two Convolution1D layers and one LayerNormalization. After stacking the two encoder layers, the feature matrix is passed through a Flatten layer and then fed into the classification head. Transformer is a widely used and excellent performing neural network. **The goal of using this network is to test the performance of attention mechanism on aperiodic time-series data**.

4.5.2 Model without ML

To study the impact of MAML as well as optimization-based ML algorithms on Temporal-ML, we remove this component and only use the pure feature extraction methods introduced above: BiLSTM, BiLSTM+CNN, Transformer. Through this, **the ability to effectively synthesize the models of MAML and optimization-based ML algorithms in the problem of movement prediction of aperiodic time-series data will be clarified**.

Chapter 5

Results & Discussions

5.1 Main results

The main results of this work are shown in table 5.1. Accordingly, our proposed method outperforms NHITS on all metrics on both aperiodic datasets (USD/JPY, multi-fx). On periodic datasets (ETT-m2, WTH), our model performs almost the same or better than the baseline model in some cases.

Table 5.1: Classification results (%) of Temporal-ML and NHITS. Best results per metrics are boldfaced. (*): Our method.

		accuracy	*precision*	*recall*	$F1-score$
USD/JPY	NHITS	58.46	58.24	57.65	55.82
	Temporal-ML*	$\mathbf{70.33 \pm 1.69}$	$\mathbf{70.73 \pm 1.82}$	$\mathbf{70.26 \pm 1.93}$	$\mathbf{69.28 \pm 2.50}$
multi-fx	NHITS	53.51 ± 5.02	53.90 ± 7.68	53.64 ± 4.05	50.20 ± 7.35
	Temporal-ML*	$\mathbf{66.26 \pm 7.45}$	$\mathbf{67.08 \pm 8.97}$	$\mathbf{65.76 \pm 7.79}$	$\mathbf{64.06 \pm 10.00}$
ETT-m2	NHITS	**71.88**	**66.86**	**62.49**	**62.69**
	Temporal-ML*	71.14 ± 9.13	62.21 ± 7.51	58.75 ± 5.30	57.61 ± 6.74
WTH	NHITS	74.18	68.05	65.04	65.40
	Temporal-ML*	$\mathbf{74.97 \pm 2.63}$	$\mathbf{69.13 \pm 3.44}$	$\mathbf{65.98 \pm 3.96}$	$\mathbf{66.00 \pm 3.80}$

Specifically, on the two aperiodic datasets USD/JPY and multi-fx, Temporal-ML improves by 12% to 14% on metrics compared to the baseline model. On the other hand, for periodic data, the accuracy of Temporal-ML is less than 1% lower, and the remaining metrics are 4-5% lower than NHITS on ETT-m2 data. For strongly cyclical data such as WTH, our algorithm outperforms NHITS by about 1%. In addition, Temporal-ML also achieves a 1-3% lower dispersion on metrics than NHITS when operating in a multi-source data context. Overall, the difference between metrics of the same method is not too large, so there is no problem of data bias and the model can be evaluated based on accuracy alone.

In table 5.1, the results are aggregated as the average of all features, so they are

highly representative. However, these results lack detail in the evaluation. Therefore, we analyze the results of the algorithms on each attribute of the aperiodic datasets in table 5.2. Accordingly, when `NHITS` predicts the attributes *hight, low, close price* on both datasets, the results are only very close to the random prediction model. This shows that prediction on aperiodic data is really difficult. However, on both datasets, the accuracy of `NHITS` often reaches the largest value when predicting *open price* and is significantly larger than the prediction results of the remaining attributes. Therefore, when taking the average, the results of `NHITS` seem to be quite high.

Table 5.2: Accuracy (%) of `NHITS` on each attribute of `USD/JPY` and `multi-fx` dataset.

		Open	*High*	*Low*	*Close*
`USD/JPY`	Random	49.91	49.91	50.33	50.66
	`NHITS`	75.03	53.74	51.57	53.50
`multi-fx`	Random	49.92 ± 2.19	49.93 ± 2.22	49.94 ± 2.17	49.85 ± 2.13
	`NHITS`	58.36 ± 7.13	50.66 ± 4.28	51.17 ± 3.21	53.84 ± 4.60

On the other hand, `Temporal-ML` only fails to predict *low price* of `USD/JPY` (accuracy approaching random prediction level). All other features of both datasets achieve high convergence and are significantly higher than `NHITS`. This demonstrates the superiority of the proposed algorithm on aperiodic data.

To clarify the capabilities of `Temporal-ML`, based on the results obtained from ablation experiments (section 4.5), we analyze the proposed algorithm on two aspects: (1) - Temporal feature extraction ability; (2) - Models' parameters aggregation ability.

5.2 Temporal feature extraction ability

From the tables 5.1 and 5.2, it is not difficult to see that the temporal feature extraction ability of `Temporal-ML` is much better than that of `NHITS`. Indeed, the feature extraction and synthesis process of `NHITS` shows that this method is trying to simulate the frequency resolution process of Fourier transform. The frequency band features are the most important information for periodic data. Therefore, the predictions of `NHITS` can easily achieve high accuracy on this type of data. However, it is very difficult to achieve frequency resolution on aperiodic data. Furthermore, the time-series data, although filtered through multiple stacks to extract low-to-high frequency bands, the filtering techniques used in the network are extremely simple (using only `FullyConnected` with nonlinear activation functions and `Pooling` layers). This is what makes `NHITS` a major obstacle on `multi-fx`, `USD/JPY` datasets and aperiodic data in general.

On the other hand, our proposed method uses deep neural networks for feature extrac-

tion. BiLSTM and RNN have generally demonstrated their capabilities in many problems involving temporal constraints, that they can easily obtain deep hidden features. Therefore, on both periodic and aperiodic data, our method achieves results approximately equal to or better than NHITS.

To better understand the capabilities of BiLSTM, we performed an ablation study on this component of Temporal-ML as mentioned in the subsection 4.5.1 and presented the results in table 5.3. Accordingly, when BiLSTM is replaced by other components (BiLSTM+CNN, Transformer) for temporal feature extraction, the results on all four metrics of most datasets are lower than those obtained by Temporal-ML. Specifically, on two aperiodic datasets (USD/JPY and multi-fx), the algorithms using BiLSTM+CNN and Transformer to extract features give 2-10% lower in prediction ability than Temporal-ML. On periodic data, MAML(Transformer) still gives much lower results than Temporal-ML. However, MAML(BiLSTM+CNN) gives results that are almost asymptotically close to the proposed algorithm. On all four datasets, the standard deviation of the metrics of the proposed algorithm is generally 1-5% smaller than the remaining algorithms.

Table 5.3: Classification results (%) of Temporal-ML and models with BiLSTM replaced. Best results per metrics are boldfaced. (*): Our method.

		accuracy	*precision*	*recall*	$F1-score$
USD/JPY	Temporal-ML*	$\mathbf{70.33 \pm 1.69}$	$\mathbf{70.73 \pm 1.82}$	$\mathbf{70.26 \pm 1.93}$	$\mathbf{69.28 \pm 2.50}$
	MAML(BiLSTM+CNN)	68.75 ± 1.78	69.15 ± 1.55	68.73 ± 1.77	69.14 ± 2.17
	MAML(Transformers)	60.67 ± 4.09	59.74 ± 6.24	59.32 ± 3.79	53.15 ± 6.19
multi-fx	Temporal-ML*	$\mathbf{66.26 \pm 7.45}$	$\mathbf{67.08 \pm 8.97}$	$\mathbf{65.76 \pm 7.79}$	$\mathbf{64.06 \pm 10.00}$
	MAML(BiLSTM+CNN)	66.08 ± 8.19	66.56 ± 9.23	65.41 ± 8.66	63.78 ± 10.83
	MAML(Transformers)	56.03 ± 6.08	56.32 ± 6.01	55.63 ± 5.01	52.33 ± 7.08
ETT-m2	Temporal-ML*	71.14 ± 9.13	62.21 ± 7.51	58.75 ± 5.30	57.61 ± 6.74
	MAML(BiLSTM+CNN)	$\mathbf{71.40 \pm 9.10}$	$\mathbf{63.25 \pm 8.82}$	$\mathbf{60.22 \pm 8.42}$	$\mathbf{59.43 \pm 9.24}$
	MAML(Transformers)	68.69 ± 8.85	58.54 ± 4.38	57.85 ± 4.71	57.07 ± 5.14
WTH	Temporal-ML*	$\mathbf{74.97 \pm 2.63}$	$\mathbf{69.13 \pm 3.44}$	$\mathbf{65.98 \pm 3.96}$	$\mathbf{66.00 \pm 3.80}$
	MAML(BiLSTM+CNN)	74.41 ± 2.71	68.17 ± 4.60	65.50 ± 4.82	65.63 ± 4.47
	MAML(Transformers)	72.28 ± 3.06	62.35 ± 8.73	60.90 ± 4.97	58.65 ± 4.72

According to this result, it can be seen that using BiLSTM brings high and stable results in the prediction process. Using networks such as CNN, Transformer not only create negative improvements but also disturbs the learning process or reduces the efficiency of the whole model. For using BiLSTM+CNN, the learning process is disturbed because BiLSTM itself has selected effective time dependencies during the extraction process while CNN only extracts features but does not select. This leads to the instability of BiLSTM+CNN's features. If CNN extracts good features, the network will perform well (e.g., metrics on ETT-m2). On the contrary, the network will perform worse than using only BiLSTM (e.g., metrics on USD/JPY, multi-fx, WTH).

In addition, we analyze the convergence of Temporal-ML and the above algorithms on

two aperiodic datasets. The results are presented in figures 5.1a and 5.1b. In general, in the process of predicting attributes of `USD/JPY` and `multi-fx`, `Temporal-ML` both give high accuracy and fast convergence. Particularly for predicting *close price* on `multi-fx`, `MAML(BiLSTM+CNN)` gives better accuracy and convergence than the proposed algorithm. Surprisingly, `MAML(Transformer)` only converges when predicting *open price* of both foreign exchange datasets. We believe that in order to use a complex model like `Transformer`, a large amount of data with a large number of inner epochs is required. This makes `Transformer` not a good choice when combined with ML algorithms.

With the above analysis, we conclude that `BiLSTM` is not only a suitable choice for exploring the variability of aperiodic time-series data but also a suitable algorithm to combine with ML algorithms when working on time-series data in general. **By using `BiLSTM`, we have effectively extracted market fluctuations and predicted the future**.

5.3 Models' parameters aggregation ability

In the process of synthesizing features to make the final prediction, `NHITS` only uses simple addition and subtraction. This makes sense in the context of synthesizing information based on the frequency bands extracted from the input data. Indeed, data related to weather, demand for airplanes or electricity show very strong periodic characteristics because weather and human consumption behavior, although they may vary over time, still show a very clear seasonality (e.g., cold weather in winter, high demand for airplanes during holidays). Therefore, the frequency of this type of data is a good feature and can be easily extracted. After decomposing the input data into frequency bands, we can simply synthesize them to be able to predict the future. In addition, synthesizing the frequency bands by an addition can be considered very simple compared to current deep learning algorithms, which work on the hidden feature space of the data. Therefore, the synthesis process of `NHITS` on aperiodic data encounters many difficulties because the input information for the process is not good and the synthesis method is ineffective. Consequently, it can be seen that aperiodic time-series data becomes a fatal weakness for `NHITS` and frequency decomposition-based methods in general.

Dealing with the complexity of non-periodic data requires a complex structure that can efficiently aggregate features. As mentioned in the Motivation section (section 1.2), analyzing and synthesizing information from multiple data sources in the same domain can provide useful information for predicting metrics of interest. `Temporal-ML` with the ability to synthesize information based on the optimization process of ML algorithms, allows the model to capture multi-dimensional information from different data sources.

which helps produce effective predictions.

To evaluate the capabilities of ML algorithms in `Temporal-ML`, we remove this component from the proposed algorithm and then run experiments to compare with the original results (subsection 4.5.2). We use ML algorithms for the purpose of synthesizing multi-source information. Therefore, comparisons are only performed on the `multi-fx` dataset because this dataset contains many sub-datasets from many different sources, which helps to accurately evaluate the information synthesis capability. The results are summarized in table 5.4. Accordingly, with the information synthesis capability, `Temporal-ML` achieves 2-12% higher in all metrics than algorithms that do not use ML in testing process. In addition, the dispersion of the metrics is also lower from 1-7%, proving that the difference in the task prediction process is much better than the remaining algorithms.

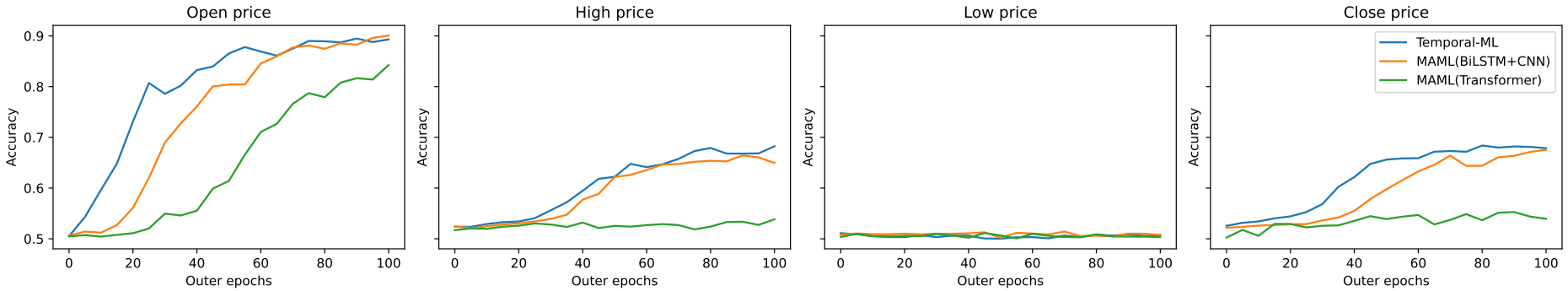
The better performance of `Temporal-ML` on `multi-fx` is understandable because all the remaining methods in the 5.4 table do not have the ability to synthesize information like the proposed algorithm, but rely entirely on past data to predict the future. Therefore, not only `BiLSTM`, `BiLSTM+CNN` but also `Transformer` cannot take advantage of the correlation between data sources.

Table 5.4: Classification results (%) of `Temporal-ML` and models without `MAML` on `multi-fx`. Best results per metrics are boldfaced. (*): Our method.

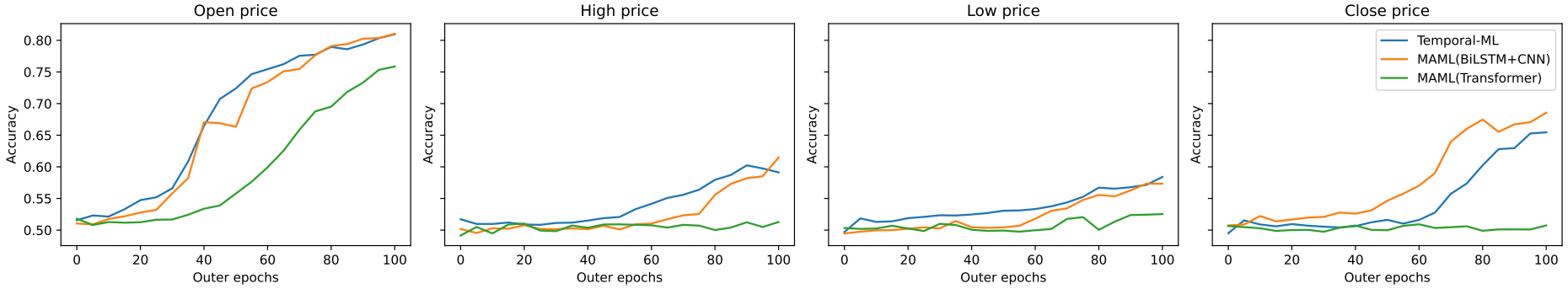
	accuracy	*precision*	*recall*	$F1-score$
`Temporal-ML`*	$\mathbf{66.26 \pm 7.45}$	$\mathbf{67.08 \pm 8.97}$	$\mathbf{65.76 \pm 7.79}$	$\mathbf{64.06 \pm 10.00}$
`BiLSTM`	63.88 ± 9.00	64.18 ± 13.39	63.66 ± 9.13	60.07 ± 13.09
`BiLSTM+CNN`	62.23 ± 8.63	62.89 ± 9.76	62.20 ± 8.34	59.91 ± 10.97
`Transformers`	58.44 ± 9.15	56.71 ± 15.10	58.22 ± 8.84	52.22 ± 13.12

The convergence of `Temporal-ML` and the models without ML is shown in Fig. 5.1c. In particular, when predicting *open price*, the proposed algorithm shows superior convergence compared to the remaining algorithms. For the attributes *high price* and *close price*, `Temporal-ML` shows signs of slowing down in the early stages of training. However, towards the end, the proposed model shows the ability to quickly surpass and has the potential to further improve the accuracy compared to the other algorithms, which have already converged.

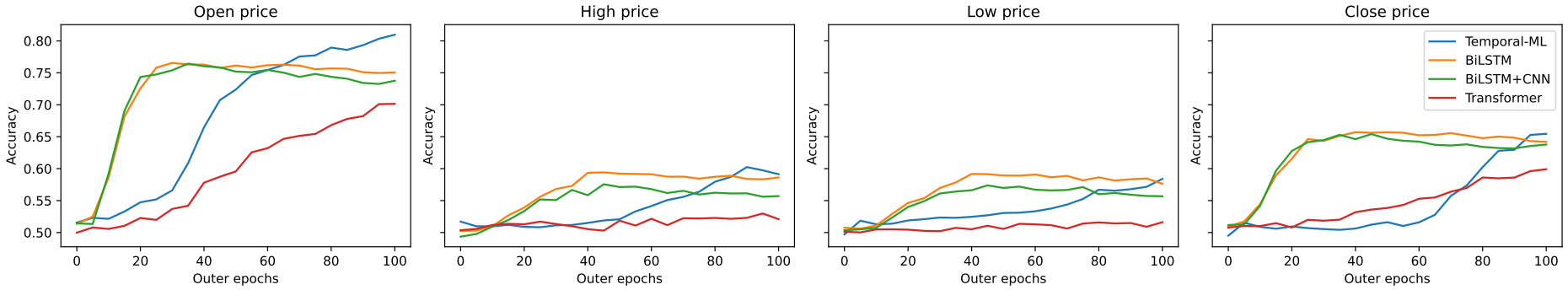
Therefore, it can be seen that, **the use of optimization-based ML algorithms in information synthesis is completely reasonable because it successfully takes advantage of the correlation between data sets and achieves high accuracy without requiring too much data as in the single-source context**.



(a) `Temporal-ML` vs. Models without `BiLSTM` on `USD/JPY`.



(b) `Temporal-ML` vs. Models without `BiLSTM` on `multi-fx`.



(c) `Temporal-ML` vs. Models without `MAML` on `multi-fx`.

Figure 5.1: Convergence process of algorithms in ablation study on all attribute of foreign exchange datasets.

Chapter 6

Conclusions & Future works

6.1 Conclusions

Problems on aperiodic time-series data pose a great challenge to current machine learning models because this data depends on both transaction history and external factors such as economic and political situations; the variance of the data changes continuously; the aperiodicity of the data is very strong. While these challenges are very data-specific and require a tailored approach, most of the current algorithms do not focused on solving these challenges but are only aimed at time-series in general. Indeed, deep learning methods such as `LSTM`, `CNN`, `Transformer` are designed to analyze general serial data. For special data types such as aperiodic time-series data, they become less effective due to the limitation in storing temporal constraints. Frequency decomposition-based methods such as `NHITS` and `bla_bla` aims to decompose the input signal into frequency bands but periodicity does not exist at all in aperiodic time-series data.

Keeping in mind the above challenges, by utilizing the time constraint extraction capability of `BiLSTM` and the efficient parameter synthesis capability of `MAML`, we propose the `Temporal-ML` algorithm with the ability to extract hidden time features as well as synthesize information from multi-source data. Based on experiments, in the process of solving the next trend prediction problem, the proposed method achieves high efficiency on two aperiodic data sets (`USD/JPY` and `multi-fx`) and equivalent efficiency on periodic data sets (`ETT-m2` and `WTH`) compared to `NHITS` (SOTA model in AAAI 2023). In addition, by performing an ablation study, we demonstrate the effectiveness of each component in `Temporal-ML`. Specifically, the ability to efficiently extract temporal constraints comes from `BiLSTM` and the ability to synthesize multi-source information comes from `MAML`. Accordingly, our algorithm effectively solves the problem of data dependence on external factors, the problem of unstable variance, and the problem of data aperiodicity.

6.2 Future works

In the future, we propose two main directions of development related to the architecture and personalization of the model.

Model architecture. Our method is developed as a component with two main components working in parallel: Temporal feature extraction and Models' parameter synthesis. This provides our method with a flexible upgradability. Our experiment only illustrates a typical case of efficient model synthesis. By using different ML algorithms such as `Reptile`, `Meta-SGD`, `iMAML`, it is possible to generate new models with higher accuracy.

Long-horizon problem. It is possible to extend this method to solve long-horizon prediction problems. Indeed, by changing the output and error of the model, it is possible to solve these problems. However, the architecture of the sub-models needs to be re-examined to better suit the new problem.

Bibliography

[1] A. C. Ian Goodfellow, Yoshua Bengio, "Deep learning," 2016.

[2] C. Challu, K. G. Olivares, B. N. Oreshkin, F. G. Ramirez, M. M. Canseco, and A. Dubrawski, "Nhits: Neural hierarchical interpolation for time series forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, pp. 6989–6997, 2023.

[3] M. Ayitey Junior, P. Appiahene, O. Appiah, and C. N. Bombie, "Forex market forecasting using machine learning: Systematic literature review and meta-analysis," *Journal of Big Data*, vol. 10, no. 1, p. 9, 2023.

[4] P. A. Moran, "Hypothesis testing in time series analysis," *Royal Statistical Society. Journal. Series A: General*, vol. 114, pp. 579–579, 7 1951.

[5] M. Rosenblatt, *Gaussian and non-Gaussian linear time series and random fields*. Springer Science & Business Media, 2000.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[8] M. Liu, A. Zeng, M. Chen, Z. Xu, Q. Lai, L. Ma, and Q. Xu, "Scinet: Time series modeling and forecasting with sample convolution and interaction," *Advances in Neural Information Processing Systems*, vol. 35, pp. 5816–5828, 2022.

[9] M. Chen, H. Peng, J. Fu, and H. Ling, "Autoformer: Searching transformers for visual recognition," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 12270–12280, 2021.

[10] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting," in *International conference on machine learning*, pp. 27268–27286, PMLR, 2022.

[11] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, pp. 11106–11115, 2021.

[12] H. Liu, Z. Wang, X. Dong, and J. Du, "Onsitnet: A memory-capable online time series forecasting model incorporating a self-attention mechanism," *Expert Systems with Applications*, vol. 259, p. 125231, 2025.

[13] E. F. Fama, "Efficient capital markets: A review of theory and empirical work," *Journal of finance*, vol. 25, no. 2, pp. 383–417, 1970.

[14] A. C. M. Andraw W. Lo, "When are contrarian profits due to stock market overreaction?," *The review of financial studies*, vol. 3, no. 2, pp. 175–205, 1990.

[15] T. S. Mech, "Portfolio return autocorrelation," *Journal of Financial Economics*, vol. 34, no. 3, pp. 307–344, 1993.

[16] M. Ali, R. Prasad, Y. Xiang, and Z. M. Yaseen, "Complete ensemble empirical mode decomposition hybridized with random forest and kernel ridge regression model for monthly rainfall forecasts," *Journal of Hydrology*, vol. 584, p. 124647, 2020.

[17] T. Zafeiriou and D. Kalles, "Intraday ultra-short-term forecasting of foreign exchange rates using an ensemble of neural networks based on conventional technical indicators," in *11th Hellenic Conference on Artificial Intelligence*, pp. 224–231, 2020.

[18] A. Sadeghi, A. Daneshvar, and M. M. Zaj, "Combined ensemble multi-class svm and fuzzy nsga-ii for trend forecasting and trading in forex markets," *Expert Systems with Applications*, vol. 185, p. 115566, 2021.

[19] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems*, vol. 2, 1989.

[20] D. Bahdanau, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[21] M.-T. Luong, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[22] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 9, pp. 5149–5169, 2021.

[23] A. Vettoruzzo, M.-R. Bouguelia, J. Vanschoren, T. Rognvaldsson, and K. Santosh, "Advances and challenges in meta-learning: A technical review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[25] F. Chen, M. Luo, Z. Dong, Z. Li, and X. He, "Federated meta-learning with fast convergence and efficient communication," *arXiv preprint arXiv:1802.07876*, 2018.

[26] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning: A meta-learning approach," *arXiv preprint arXiv:2002.07948*, 2020.

[27] B.-L. Nguyen, T. C. Cao, and B. Le, "Meta-learning and personalization layer in federated learning," in *Asian Conference on Intelligent Information and Database Systems*, pp. 209–221, Springer, 2022.

[28] N. Hu, E. Mitchell, C. D. Manning, and C. Finn, "Meta-learning online adaptation of language models," *arXiv preprint arXiv:2305.15076*, 2023.

[29] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few-shot learning," *arXiv preprint arXiv:1707.09835*, 2017.

[30] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*, pp. 1126–1135, PMLR, 2017.

[31] O. Kolle, "Weather dataset," 2008.

[32] A. Garza, "Neural forecast - user friendly state-of-the-art neural forecasting models."