Master Dissertation

Meta-learning in movement prediction 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

Abstract

Keywords: Meta-learning, `LSTM`, `CNN`, Aperiodic time-series data, Foreign exchange

Acknowledgements

I would like 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 leading me to JAIST and for everything.

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 (like finance, energy, and transportation), but also in terms of methods for analyzing and forecasting them (like 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.

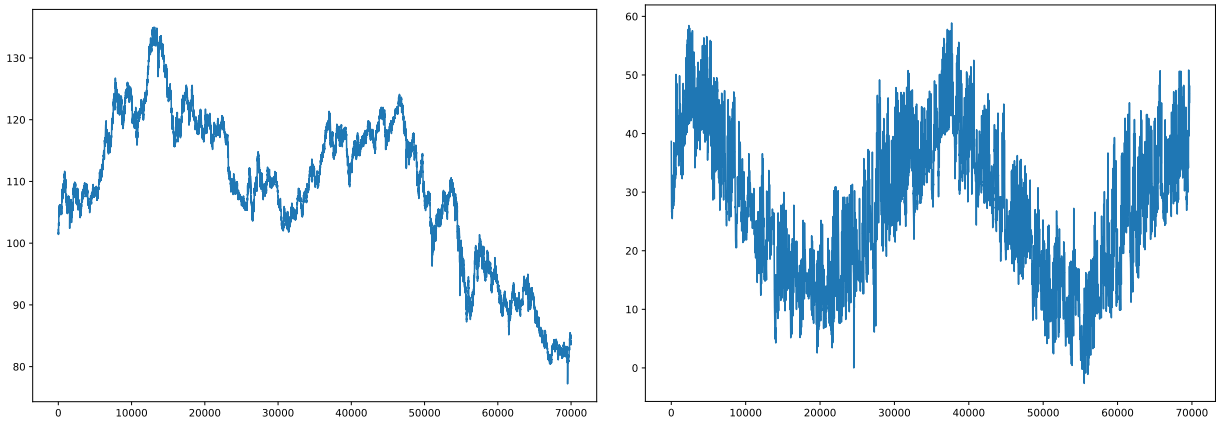
The two main approaches used in analyzing and predicting time-series data are fundamental analysis and technical analysis [2]. While fundamental analysis concentrates on examining external factors that are hard to capture from past price changes, such as the economic strategies and policies of nations 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 nature of news data. As a result, academics frequently 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 Autoregressive model (1951) [3], Moving average model (2000) [4], Long short-term memory model - `LSTM` (1997) [5], and Transformers (2017) [6].

The aforementioned techniques, together with their variations, 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. As a result, analysis and prediction may be done quickly and accurately. This is especially true for existing techniques that use deep neural networks to try to decompose periods [7–9]. 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 their 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`) [10] shows a very 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. As a result, 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 has resulted in notable modifications to the data's characteristics, which are also referred to as concept drift scenarios [11]. Mathematically, this results in aperiodic time-series data with a continuous and incredibly wide 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 to overcome when 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

In three aforementioned challenges, the first challenge is the most difficult to solve based on technical analysis methods because political and social information is difficult to extract through historical data of a data set. However, according to the efficient market hypothesis [12], historical data itself clearly reflects information about market fluctuations. That means, **by carefully examining the transaction history, one can completely grasp market fluctuations and forecast the future** without using political and social information. In addition, [13, 14] studies also point out 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.**

When faced with the second challenge, ensemble learning is often used to mitigate the effects of variance [15–17]. Basically, ensemble learning works by splitting the data into multiple parts and assuming a fixed level of variance across these parts, then simulating that variation. This approach is reasonable because it is difficult for a single model to capture all the variability in the data over a long 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` [18], `LSTM` [5], and attention mechanism [6,19,20] 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 [21,22]. 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. The mentioned studies belong to two popular approaches in analyzing time-series data: Deep feature-based approach and Frequency decomposition approach. In addition, Chapter 2 also presents ML, 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

2.1 Deep feature-based approach

2.1.1 Convolutional neural network

2.1.2 Recurrent Neural Network

2.1.3 Long Short-Term Memory

2.2 Frequency decomposition approach

2.2.1 A transformer-based method

2.2.2 NHITS

Neural Hierarchical Interpolation for Time Series Forecasting (`NHITS`) [1] 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 2.1).

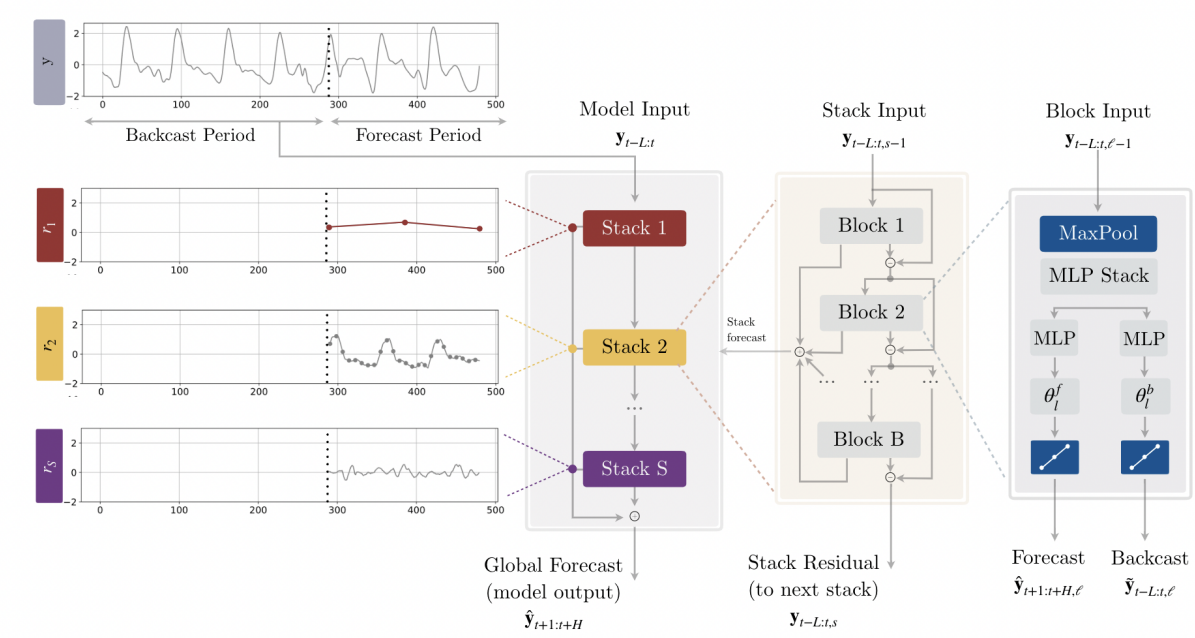


Figure 2.1: `NHITS` architecture [1].

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 2.1). 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.

$$\mathbf{y}^{(p)}_{t-L:t,l} = \mathbf{MaxPool}\left(\mathbf{y}_{t-L:t,l-1}, k_l\right) \tag{2.1}$$

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^f_l, θ^b_l (equation 2.4). 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^f_l is used to forecast future values and θ^b_l is used to regress the input values.

$$\theta_l^b = \textbf{FullyConnected}^b \left(\mathbf{y}_{t-L:t,l}^{(p)} \right) \tag{2.2}$$

$$\theta_l^f = \textbf{FullyConnected}^f \left(\mathbf{y}_{t-L:t,l}^{(p)} \right) \tag{2.3}$$

$$\tag{2.4}$$

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 2.6). 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.

$$\hat{\mathbf{y}}_{t-L:t,l} = g\left(\theta_l^b\right) \tag{2.5}$$

$$\hat{\mathbf{y}}_{t+1:t+H,l} = g\left(\theta_l^f\right) \tag{2.6}$$

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 (2.7).

$$\mathbf{y}_{t-L:t,l+1} = \mathbf{y}_{t-L:t,l-1} - \hat{\mathbf{y}}_{t-L:t,l} \tag{2.7}$$

Summing the forecast values of B blocks as in equation 2.8, we get the forecast value of a stack. Finally, summing the forecast values of the stacks as in equation 2.9, we get the predicted forecast value of the entire network.

$$\hat{\mathbf{y}}^{s}_{t+1:t+H} = \sum_{l=1}^{B} \hat{\mathbf{y}}_{t+1:t+H,l} \tag{2.8}$$

$$\hat{\mathbf{y}}_{t+1:t+H} = \sum_{s=1}^{S} \hat{\mathbf{y}}^{s}_{t+1:t+H} \tag{2.9}$$

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`.

2.3 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 [21, 22]. 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 [23–25], domain adaptation in online learning [**?**, 26]).

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:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\mathcal{D}_t; \theta, \phi) \tag{2.10}$$

The ML approach is to try to learn a good prior knowledge ϕ. This is achieved by learning a task distribution $\mathcal{T}$ [21]. 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:

$$\min_{\phi} \mathop{\mathbb{E}}_{t\sim\mathcal{T}} \mathcal{L}(\mathcal{D}_t, \phi) \tag{2.11}$$

2.3.1 Model-Agnostic Meta-Learning (MAML)

For the optimization-based approach, a basic ML algorithm, typically `MAML`, is learned on multiple tasks t drawn from the same task distribution $\mathcal{T}$ [21]. 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:

$$\theta_t^* = \theta_t(\phi) = \arg\min_{\theta} \mathcal{L}_t^{task}\left(\phi, \mathcal{D}_t^{support}\right) \tag{2.12}$$

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:

$$\begin{aligned}\phi^* &= \arg\min_{\phi} \sum_t \mathcal{L}_t^{meta}\left(\theta_t^*, \mathcal{D}_t^{query}\right) \\ &= \arg\min_{\phi} \sum_t \mathcal{L}_t^{meta}\left(\theta_t(\phi), \mathcal{D}_t^{query}\right)\end{aligned} \tag{2.13}$$

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.

2.3.2 Meta-SGD

As for the `Meta-SGD` algorithm, study [27] 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 [27] 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 2.13 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 parts 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 use two types of features from `LSTM` and `LSTM+CNN` networks. In the *Effective synthesis of model's parameters* section, we use `MAML` to synthesize the parameters of the models.

Due to the contribution of `LSTM` and `LSTM+CNN` features, we expect to effectively extract hidden 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.

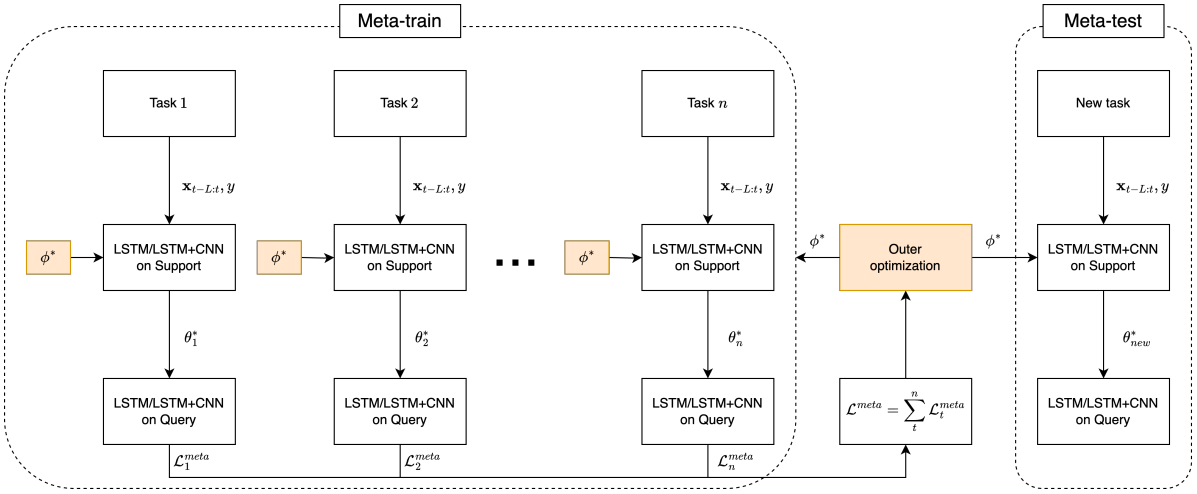


Figure 3.1: The full-flow of meta-training and meta-testing on multi-fx data. Each currency pair is regarded as a task.

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 $\mathcal{D}_t^{support}$

$$\begin{aligned}
\mathbf{x}' &= \textbf{FullyConnected}\left(\mathbf{x}\right) \\
\mathbf{h}_{LSTM} &= \textbf{BidirectionalLSTM}\left(\mathbf{x}'\right) \\
\mathbf{h}_{CNN} &= \textbf{Convolution1D}\left(\mathbf{x}'\right) \\
\hat{y} &= \begin{cases} \textbf{FullyConnected}\left(\mathbf{h}_{LSTM}\right) \\ \textbf{FullyConnected}\left(\textbf{Concatenate}\left(\mathbf{h}_{LSTM}, \mathbf{h}_{CNN}\right)\right) \end{cases}
\end{aligned}$$

    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 needs to 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 of `MAML`. In case the data includes a single dataset, it is necessary to divide this dataset into subsets corresponding to separate tasks. 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 two different methods: via `LSTM` and `LSTM+CNN`. For the method using `LSTM`, 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 `LSTM` network to selectively extract long-term temporal dependencies ($\mathbf{h}_{LSTM}$). To fully exploit the long-term time constraints, we use `BidirectionalLSTM` to extract from both sides of $\mathbf{x}$ (equation 3.2)

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

$$\mathbf{h}_{LSTM} = \mathbf{BidirectionalLSTM}\left(\mathbf{x}'\right) \tag{3.2}$$

Inspired by study [28], we use `LSTM+CNN` features, which combines the features extracted from `LSTM` and `CNN`. Specifically, in addition to using `LSTM`, $\mathbf{x}'$ is also fed into `CNN` to extract local temporal features $\mathbf{h}_{CNN}$ (equation 3.3). Next, $\mathbf{h}_{LSTM}$ and $\mathbf{h}_{CNN}$ are concatenated (equation 3.4) and then passed to the classification head of the neural network (equation 3.5).

$$\mathbf{h}_{CNN} = \mathbf{Convolution1D}\left(\mathbf{x}'\right) \tag{3.3}$$

$$\mathbf{h} = \mathbf{Concatenate}\left(\mathbf{h}_{LSTM}, \mathbf{h}_{CNN}\right) \tag{3.4}$$

$$\hat{y} = \mathbf{FullyConnected}\left(\mathbf{h}\right) \tag{3.5}$$

The `LSTM` network maintains cell-state values to selectively store long-term dependencies. This is very suitable for solving time-series data problems. On the other hand, future values often depend heavily on recent historical values. We propose to use the `CNN` network to emphasize local features, thereby directing part of the model's attention to certain time points. Therefore, the proposed method not only remembers long-term features but also highlights short-term features.

3.3 Effective synthesis of models' parameters

We use `MAML` as presented in the algorithm 1 to train and aggregate the weights of the models at the tasks. As mentioned in section 2.3, parameter optimization in the ML approach is to solve the two equations 2.12 and 2.13 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:

$$\begin{cases} \theta_t^{(0)} & = \phi_{r-1} \\ \theta_t^{(e)} & = \theta_t^{(e-1)} - \alpha \nabla_\theta \mathcal{L}_t^{task} \left(\theta_t^{(e-1)}, \mathcal{D}_t^{support} \right) \end{cases} \tag{3.6}$$

In which, ϕ_{r-1} is the result of the $r-1$ outer optimization process, α is the inner learning rate.

Next, the outer optimization process is performed by aggregating the losses on the query set of the tasks and optimizing on it (equation 3.7).

$$\begin{cases} \phi_0 = \text{Random Initialization} \\ \phi_r = \phi_{r-1} - \beta \nabla_\phi \sum_{t=1}^{n} \mathcal{L}_t^{meta} \left(\theta_t^*(\phi), \mathcal{D}_t^{query} \right) \end{cases} \tag{3.7}$$

Where, β is the outer learning rate.

Assuming the algorithm runs E steps in inner optimization, the derivative quantity at equation 3.7 is rewritten as follows (the notations of dataset are removed):

$$\begin{aligned} \nabla_\phi \sum_{t=1}^{n} \mathcal{L}_t^{meta} \left(\theta_t^*(\phi) \right) &= \nabla_\phi \sum_{t=1}^{n} \mathcal{L}_t^{meta} \left(\theta_t - \alpha \nabla_\theta \mathcal{L}_t^{task} \left(\theta_t \right) \right) \\ &= \sum_{t=1}^{n} \frac{\partial \mathcal{L}_t^{meta} \left(\theta_t^{(E)} \right)}{\partial \theta_t^{(E)}} \frac{\partial \theta_t^{(E)}}{\partial \phi} \\ &= \sum_{t=1}^{n} \nabla_\theta \mathcal{L}_t^{meta} \left(\theta_t^{(E)} \right) \prod_{j=0}^{E-1} \left[\mathbb{I} - \alpha \nabla_\theta^2 \mathcal{L}_t^{task} \left(\theta_t^{(j)} \right) \right] \end{aligned} \tag{3.8}$$

The presence of the product of second order derivatives in the equation 3.8 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 [23–25, 27, 29] 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 [2]. 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, the equation 3.7 can be considered an optimization-based method of synthesizing sub-models, which helps to take advantage of the feature extraction capabilities of each model. In other words, the synthesized model can extract features at a deeper level, significantly improving the prediction ability compared to traditional ensemble models.

Chapter 4

Numerical Experiment

4.1 Dataset description

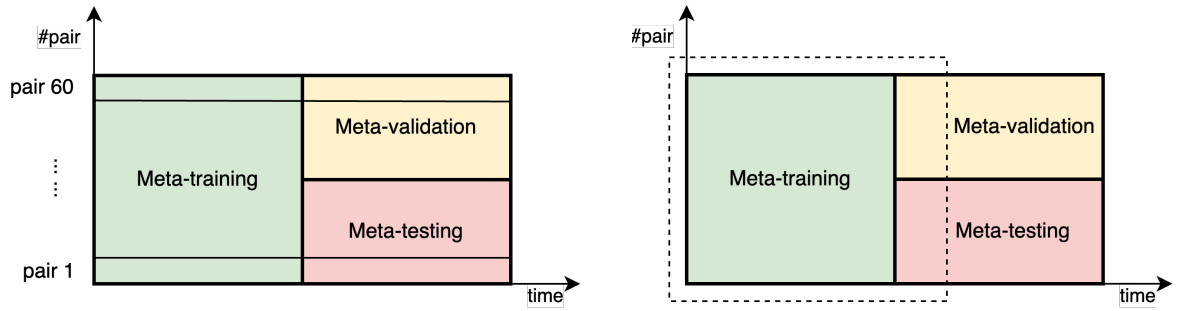
Foreign exchange (FX) in particular and financial indices in general are typical data types for aperiodic time-series data. Therefore, we choose this type of data to test the model. Specifically, we configure two datasets using FX data. `USD/JPY` dataset consists of only data of the exchange rate between US dollar and Japanese yen. The data is sampled hourly from 2000 to 2024, including the attributes of 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`) [10] and Weather (`WTH`) [30]. `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`.

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



(a) (b)

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

4.2 Data pre-process

4.2.1 Temporal-ML

Như đã đề cập trong section 3.1, để sử dụng ML, các tập dữ liệu cần phải được cấu trúc thành các task riêng biệt. Trong mỗi task, tập support chiếm 20% dữ liệu, tập query chiếm 80% dữ liệu. Đối với `multi-fx`, mỗi cặp tiền tệ được chia thành 2 tập dữ liệu, ứng với 2 tasks. Do đó, 60 cặp tiền tệ tạo thành 120 tasks. Đối với các tập dữ liệu còn lại, mỗi tập được chia nhỏ ra thành các chuỗi thời gian, mỗi chuỗi tương ứng với một task. Trong cả quá trình, 50% số tasks được sử dụng trong quá trình meta-training, 25% số tasks được sử dụng trong quá trình meta-validation, số tasks còn lại được sử dụng trong meta-testing. Thống kê dữ liệu cho các task theo tập dữ liệu được trình bày trong bảng 4.1.

Vì các tập dữ liệu đều là time-series, dữ liệu trong quá trình huấn luyện không được tiết lộ bất cứ thông tin gì về tương lai. Do đó, 120 tasks của dữ liệu `multi-fx` được chia như hình 4.1a. Đối với các tập dữ liệu còn lại, dữ liệu sử dụng trong quá trình huấn luyện, kiểm thử, kiểm tra phải đảm bảo thứ tự thời gian từ quá khứ đến tương lai.

Quá trình tiền xử lý bằng `z-score` theo đó được thực hiện trên toàn bộ dữ liệu huấn luyện và các tập support của dữ liệu kiểm thử và kiểm tra của từng cặp tiền tệ (see figure 4.1b) vì đây là các dữ liệu đã được biết trước. Cụ thể, tiền xử lý thuộc tính X được thực

hiện như sau:

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

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

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

μ_X và σ_X thu được từ phương trình 4.1 và 4.2 được sử dụng để chuẩn hóa cho thuộc tính X trong query set của quá trình meta-validation và meta-testing.

4.2.2 Baseline model

Nghiên cứu này sử dụng thuật toán `NHITS` [1] làm baseline model. `NHITS` sử dụng toàn bộ thuộc tính để dự đoán xu hướng tiếp theo cho một thuộc tính trong một tập dữ liệu. Thuật toán hướng đến phân rã các giải tần và kết hợp lại để dự đoán tương lai. Đối với các tập dữ liệu `USD/JPY`, `ETT-m2`, và `WTH`, dữ liệu tại mỗi tập được chia thành training set, validation set, và test set với tỷ lệ tương ứng 6:2:2.

Chúng tôi tiền xử lý dữ liệu huấn luyện bằng `z-score` như đã đề cập ở subsection 4.2.1. Sau đó, μ_X và σ_X thu được từ phương trình 4.1 và 4.2 được sử dụng để chuẩn hóa cho thuộc tính X trong validation set trong quá trình tuning. Khi đã chọn được các siêu tham số tốt nhất, dữ liệu của training set và validation set được chuẩn hóa lại từ đầu, sau đó được huấn luyện với bộ siêu tham số tốt nhất để dự đoán test set.

Đối với tập dữ liệu `multi-fx`, vì `NHITS` không có cơ chế tổng hợp mô hình trên các tập dữ liệu khác nhau, chúng tôi lặp lại quy trình huấn luyện (huấn luyện, tuning, testing trên 60%, 20%, 20% dữ liệu, respectively) cho từng cặp tiền tệ, sau đó tổng hợp metrics trên các cặp tiền tệ để thu được đánh giá cuối.

4.3 Metric evaluation

Nghiên cứu sử dụng accuracy, precision, recall, và F1 để đánh giá các mô hình:

$$acc = \frac{TP + TN}{TP + FP + TN + FN}$$
$$P = \frac{TP}{TP + FP}$$
$$R = \frac{TP}{TP + FN}$$
$$F1 = \frac{2PR}{P + R}$$

Trong đó, TP, TN, FP, FN lần lượt là số mẫu *true_positive, true_negative, false_positive, false_negative.*

4.3.1 Temporal-ML

Giả sử thuật toán `Temporal-ML` được đánh giá trên tập dữ liệu $\mathcal{D}$ gồm a thuộc tính và được chia thành n tasks. Quá trình đánh giá được thực hiện bằng cách để mô hình dự đoán xu hướng của từng thuộc tính với đầu vào là tất cả các thuộc tính, sau đó thực hiện hai bước tổng hợp: (1) - Tổng hợp trên task; (2) - Tổng hợp trên thuộc tính.

Tổng hợp trên task. Xét metric m khi mô hình dự đoán thuộc tính k bất kỳ $(1 \leq k \leq a)$. Sau khi dự đoán thuộc tính k, chúng ta thu được n giá trị: $\{m_1^{(k)}, \ldots, m_n^{(k)}\}$. Quá trình tổng hợp metrics m của n tasks được thực hiện như sau:

$$\bar{m}^{(k)} = \frac{1}{n} \sum_{i=1}^{n} m_i^{(k)} \tag{4.4}$$
$$s_m^{(k)} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (m_i^{(k)} - \bar{m}^{(k)})^2} \tag{4.5}$$

Tổng hợp trên thuộc tính. Mô hình thực hiện dự đoán tất cả a thuộc tính và thu được a metrics: $\left\{\left(\bar{m}^{(k)} \pm s_m^{(k)}\right)\right\}_{k=1}^{a}$. Quá trình tổng hợp metric m của a thuộc tính diễn ra như sau:

$$\bar{m} = \frac{1}{a} \sum_{k=1}^{a} \bar{m}^{(k)} \tag{4.6}$$
$$s_m = \sqrt{\frac{1}{a} \sum_{k=1}^{a} (s_m^{(k)})^2} \tag{4.7}$$

4.3.2 Baseline model

`NHITS` cũng được đánh giá trên tập dữ liệu $\mathcal{D}$ với a thuộc tính và phải thực hiện dự đoán trên từng thuộc tính. Xét metric m, sau quá trình dự đoán thuộc tính k, ta thu được a giá trị: $\{m_1^{(k)}, \ldots, m_a^{(k)}\}$. Quá trình tổng hợp metric m của a thuộc tính được thực hiện như sau:

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

Như đã đề cập ở trên, `NHITS` không có cơ chế tổng hợp mô hình khi kiểm thử trên dữ liệu `multi-fx`. Do đó, chúng tôi kiểm thử `NHITS` trên từng cặp tiền tệ trong `multi-fx` và tổng hợp metrics giống như `Temporal-ML`.

4.3.3 Fairness in evaluation

Gọi m là số mẫu dữ liệu trong mỗi tasks, tổng số dữ liệu của quá trình huấn luyện, kiểm thử, và kiểm tra là: mn. Đối với `NHITS`, mô hình được huấn luyện, kiểm thử, kiểm tra trên 60%, 20%, 20% dữ liệu, respectively:

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

Đối với `Temporal-ML`, quá trình làm giàu knowledge cho mô hình sử dụng toàn bộ dữ liệu huấn luyện và tập support của dữ liệu kiểm thử/kiểm tra. Quá trình kiểm thử/kiểm tra được thực hiện trên tập query:

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

Do đó, với cách chia dữ liệu như trên, chúng tôi đạt được sự công bằng về số lượng mẫu dữ liệu trong quá trình kiểm thử.

4.4 Hyper parameters tuning

4.4.1 Temporal-ML

Trong cài đặt của chúng tôi, chúng tôi sử dụng một lớp `FullyConnected` gồm 16 units với hàm kích hoạt `ReLU` để biểu diễn các đặc trưng sâu hơn và rõ ràng hơn. Sau đó, đặc trưng này được truyền song song đến các hai khối `BidirectionalLSTM` và `CNN`. Khối `BidirectionalLSTM` bao gồm 32 hidden units, các đầu ra được nối dài để tạo thành một vector cuối. Khối `CNN` bao gồm hai layer `CNN` có số filter lần lượt là 32 và 64. Kernel được sử dụng trong các layer có kích thước 3×3. Theo sau mỗi layer `CNN` là một layer `MaxPooling` sử dụng kernel kích thước 2×2. Kết thúc khối `CNN` là một layer `Flatten`. Trong trường hợp chỉ sử dụng các đặc trưng dài hạn, đầu ra của khối `BidirectionalLSTM` sẽ được sử dụng để phân lớp. Nếu sử dụng thêm các đặc trưng cục bộ, đầu ra của khối `BidirectionalLSTM` và `CNN` sẽ được nối lại rồi truyền qua một layer phân lớp nhị phân với hàm kích hoạt `Sigmoid`.

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, 0.005, 0.01, 0.05}
Outer learning rate	{0.001, 0.005, 0.0015, 0.0055}

Quá trình fine-tune của thuật toán ML liên quan đến nhiều siêu tham số như inner batch size, outer batch size, inner training step, outer training step,... Để dễ dàng fine-tune, chúng tôi cố định hầu hết các tham số và chỉ fine-tune kích thước của lookback window, inner và outer learning rate. Chi tiết được trình bày trong bảng 4.2.

4.4.2 Baseline model

Đối với mô hình `NHITS`, chúng tôi dựa trên [1] để định nghĩa không gian tìm kiếm cho việc fine-tune tham số, cũng như kiến trúc mô hình. Đối với các tham số không được đề cập trong bảng, chúng tôi sử dụng giá trị mặc định của cài đặt `NHITS` trong thư viện `NeuralForecast` [31]. Kết quả tốt nhất của các lần fine-tune được chọn ra và báo cáo trong nghiên cứu này.

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	{5, 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 MLF layers	{1,2}
Learning rate	{0.001, 0.002, 0.005, 0.01, 0.02}

Chapter 5

Results

5.1 Main results

5.2 Feature extraction ability

5.3 Sub-model aggregation ability

Chapter 6

Conclusion & Future works

6.1 Conclusion

6.2 Future works

Bibliography

[1] 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.

[2] 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.

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

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

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

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

[27] 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.

[28] Q.-H. Vo, H.-T. Nguyen, B. Le, and M.-L. Nguyen, "Multi-channel lstm-cnn model for vietnamese sentiment analysis," in *2017 9th international conference on knowledge and systems engineering (KSE)*, pp. 24–29, IEEE, 2017.

[29] 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.

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

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