Pattern Recognition 141 (2023) 109604

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Augmented bilinear network for incremental multi-stock time-series classification

Mostafa Shabani [a,*], Dat Thanh Tran [b], Juho Kanniainen [b], Alexandros Iosifidis [a]

[a] DIGIT, Department of Electrical and Computer Engineering, Aarhus University, Denmark
[b] Unit of Computing Sciences, Tampere University Tampere, Finland

ARTICLE INFO

Article history:
Received 16 December 2022
Revised 4 March 2023
Accepted 7 April 2023
Available online 10 April 2023

Keywords:
Deep learning
Low rank tensor decomposition
Limit order book data
Financial time-series analysis

ABSTRACT

Deep Learning models have become dominant in tackling financial time-series analysis problems, overturning conventional machine learning and statistical methods. Most often, a model trained for one market or security cannot be directly applied to another market or security due to differences inherent in the market conditions. In addition, as the market evolves over time, it is necessary to update the existing models or train new ones when new data is made available. This scenario, which is inherent in most financial forecasting applications, naturally raises the following research question: *How to efficiently adapt a pre-trained model to a new set of data while retaining performance on the old data, especially when the old data is not accessible?* In this paper, we propose a method to efficiently retain the knowledge available in a neural network pre-trained on a set of securities and adapt it to achieve high performance in new ones. In our method, the prior knowledge encoded in a pre-trained neural network is maintained by keeping existing connections fixed, and this knowledge is adjusted for the new securities by a set of augmented connections, which are optimized using the new data. The auxiliary connections are constrained to be of low rank. This not only allows us to rapidly optimize for the new task but also reduces the storage and run-time complexity during the deployment phase. The efficiency of our approach is empirically validated in the stock mid-price movement prediction problem using a large-scale limit order book dataset. Experimental results show that our approach enhances prediction performance as well as reduces the overall number of network parameters.

1. Introduction

Deep Learning has become prominent in tackling challenges in financial time series analysis [1–9]. Since the inception of electronic trading systems, large amounts of trade data have become available and accessible to many people. With large-scale data, conventional approaches based on linear and non-linear models trained with convex optimization have become less efficient in terms of computational complexity as well as prediction performance. Significant improvements in computing hardware coupled with increasing amounts of large-scale datasets have enabled the research community to harness the power of deep neural networks combined with stochastic optimization. The shift towards an end-to-end Deep Learning paradigm not only allows us to tackle larger and more complex problems but also enables us to focus on more important aspects of the real-world problems, such as the feasibility and efficiency when applying learning models to real-world financial problems [10].

Although it is less demanding in terms of memory complexity compared to non-linear models trained with convex optimization, optimizing deep neural networks using stochastic optimization is still time-consuming, since in order to train a well-performing network often requires a high number of iterations of mini-batch based optimization. The high computational cost is also associated with high energy usage, which can significantly affect the profitability of a Deep Learning solution. On the microeconomic level, since the data throughput is large and the frequency of changes in the data distribution is high, especially in highly liquid markets, we are faced with a continuous influx of data. On the macroeconomic level, the market continuously evolves to accommodate different phases of the economy. These unique features of the financial market can easily make a data-driven system obsolete, requiring it to be frequently updated with recent data. The necessity of frequent updates coupled with the high cost of updates has made computa-

* Corresponding author.
E-mail addresses: mshabani@ece.au.dk (M. Shabani), thanh.tran@tuni.fi (D.T. Tran), juho.kanniainen@tuni.fi (J. Kanniainen), ai@ece.au.dk (A. Iosifidis).

https://doi.org/10.1016/j.patcog.2023.109604

tional complexity a critical factor when utilizing deep neural networks in financial analysis or forecasting systems.

There have been different approaches to reduce computational complexity when training deep neural networks, such as designing novel low-complexity network architectures [11–14], replacing existing ones with their low-rank counterparts [15–18], or adapting the pre-trained models to new tasks, i.e., performing Transfer Learning (TL) [19–21] or Domain Adaptation (DA) learning [22–25]. Among these approaches, model adaptation is the most versatile since a method in this category is often architecture-agnostic, being complementary to other approaches. In financial analysis or forecast settings, the ability to reuse existing pre-trained models can play an important role. The need to update a system on a regular basis arises in many situations in finance [26–28]. Instead of training a new model from scratch with the new data, an efficient adaptation method will allow us to quickly adjust the existing model using the new data with less computational burden. Not only can this approach reduce the operating costs, thereby increasing the profit of the system, but also create the flexibility of more frequent updates. Besides the computational efficiency, model re-usability might also improve the modeling performance of a system. For example, data might be abundant for highly liquid markets/securities but might be scarce for less liquid markets/securities. As a result, we might not have sufficient data of an illiquid stock to train a deep neural network from scratch without overfitting. By taking advantage of models pre-trained on other stocks, one can obtain better performance even with a small amount of data in new stocks.

In this paper, we consider the following research problem: given a new task $\mathcal{T}_{\text{new}}$ defined on financial time-series data and a neural network $\mathcal{N}_{\text{old}}$, which has been trained on previously collected financial time-series data to solve a task $\mathcal{T}_{\text{old}}$ that is relevant to $\mathcal{T}_{\text{new}}$, the objective is to efficiently generate a new network $\mathcal{N}_{\text{new}}$ based on $\mathcal{N}_{\text{old}}$ that generalizes well for the new task $\mathcal{T}_{\text{new}}$ without having access to the data defining $\mathcal{T}_{\text{old}}$, and without major impact on the performance of $\mathcal{N}_{\text{old}}$ in $\mathcal{T}_{\text{old}}$. The tasks are often expressed via their datasets, and we refer to the datasets of $\mathcal{T}_{\text{old}}$ and $\mathcal{T}_{\text{new}}$ as $\mathcal{D}_{\text{old}}$ and $\mathcal{D}_{\text{new}}$, respectively. We refer to the above-described problem as the *Incremental Multi-Stock Time-Series Analysis* problem. While multiple time-series analysis tasks can be formulated in this context, we focus on the case where the $\mathcal{T}_{\text{old}}$ corresponds to a mid-price direction prediction task defined on a set of stocks. This task can be formulated as a time-series classification problem. Historic data of $\mathcal{T}_{\text{old}}$ forming $\mathcal{D}_{\text{old}}$ is used to train the Deep Learning model $\mathcal{N}_{\text{old}}$. Given a pre-trained $\mathcal{N}_{\text{old}}$ and historic data of a (set of) new stock(s) forming $\mathcal{D}_{\text{new}}$, we would like to exploit the knowledge encoded in $\mathcal{N}_{\text{old}}$ to effectively be able to predict the direction of mid-price movements of the stocks belonging to both $\mathcal{T}_{\text{old}}$ and the stock(s) defining the new task $\mathcal{T}_{\text{new}}$.

The research problem described above is highly relevant to stock market data analysis since one gets access to historic data of new stocks in different time periods, and historic data used to train existing models can be either absent or so big that merging data in $\mathcal{D}_{\text{old}}$ and $\mathcal{D}_{\text{new}}$ to train $\mathcal{D}_{\text{new}}$ can be impractical due to their large size. While TL and DA have been widely adopted to solve related problems, they are not well-suited for addressing the problem of interest in our study. This is due to that they either require the use of both $\mathcal{D}_{\text{old}}$ and $\mathcal{D}_{\text{new}}$ to *adapt* the parameters of $\mathcal{N}_{\text{old}}$ to the new task, or they create a second model $\mathcal{N}_{\text{new}}$ the parameters of which are initialized to those of $\mathcal{N}_{\text{old}}$ and are further fine-tuned using $\mathcal{D}_{\text{new}}$, thus leading to an inefficient solution.

In this paper, we propose a method for performing network augmentation by learning auxiliary neural connections that are complementary to the existing ones. This allows the new model to preserve prior knowledge and rapidly adapt to the new tasks, leading to improvements in performance. By using a low-rank approximation for the auxiliary connections, our method obtains additional efficiency in terms of overall operational cost. We demonstrate our approach with the Temporal Attention-augmented Bilinear Layer (TABL) [29] network architecture, which achieves state-of-the-art performance in the stock mid-price direction prediction. In addition, we also demonstrate that the proposed approach can generalize to Convolutional Neural Networks (CNN), which is another type of neural network architecture that is widely used in financial time series analysis [1,2]. Using a large-scale Limit Order Book (LOB) dataset, our empirical study shows that the proposed method can indeed improve both prediction performance and operational efficiency of TABL networks as well as CNN networks, compared to TL and DA approaches.

The contributions of the paper are summarized as follows:

- We describe a new research problem which is motivated by situations frequently arising in financial time-series analysis problems;
- We propose a solution to the research problem based on the TABL network, the effectiveness of which is demonstrated through experiments on three frequently occurring scenarios;
- We show that the proposed approach can be extended to other neural network types by providing a solution based on CNN;
- We conducted comprehensive experiments using two datasets from distinct markets with varying characteristics, based on different real-world scenarios, in order to demonstrate the efficacy of our solution.

The remainder of the paper is organized as follows. Related works in financial time series analysis are briefly presented in Section 2. The proposed method is described in Section 3. The complexity analysis of the proposed method is provided in Section 3.3. In Section 4, we describe in detail our experimental protocol and present the empirical results. Section 5 includes supplementary experiments conducted on a comparable dataset. Section 6 concludes our work and discusses potential future research directions.

2. Related work

While econometric models can provide certain statistical insights with great transparency [30,31], deep neural networks following the end-to-end data-driven learning paradigm led to significant improvements on the performance of financial time series prediction tasks. In [1], a methodology based on Convolutional Neural Networks (CNN) was proposed for mid-price direction prediction. Deep Learning models utilizing CNN to capture the spatial structure of the LOB and Long Short Term Memory (LSTM) units to capture time dependencies were proposed in [2,32]. A multilayer Perceptron was used in [5] in the form of a spatial neural network. Adaptive input normalization jointly optimizing the input data normalization in the form of a neural layer with the parameters of the Deep Learning used for classification was proposed in [4,33].

2.1. Model Re-usability

To address model re-usability, TL enables exploiting information from a pre-trained model that was trained on a source dataset, referred to as the base model hereafter, to improve the performance of the target model being trained on a target dataset [34]. Domains of the source and target datasets can be different, however, they must be related to each other. In cases where the source and target domains are highly dissimilar, negative transfer issues can appear, i.e., transferring knowledge adversely affects the performance of the target model [35]. In [36], the Dynamic Time Warping method (DTW) [37] was used to find similarities between

the source and target datasets to avoid the negative transfer issue in TL of deep CNNs for time series classification tasks. Autonomous transfer learning (ATL) was proposed in [38] to produce a domain invariant network and handle the problem of concept drifts [39] both in the source and target domains. A hybrid algorithm based on TL, i.e. the Online Sequential Extreme Learning Machine with Kernels (OS-ELMK), and ensemble learning for time series prediction was proposed in [40] to handle the problem of wide variability between old data (base dataset) and new data (target dataset). In [41], a fuzzy regression TL method was proposed, using the Takagi-Sugeno fuzzy regression model to transfer knowledge from a source domain to a target domain. Data augmentation was used to improve the performance of TL for stock price direction prediction in [21]. A TL framework for predicting stock price movements that uses relationships between stocks to construct effective input was proposed in [42]. In [43], a TL architecture using encoder-decoder was proposed to learn market-specific trading strategies. Online transfer learning [44,45] tries to handle the transfer learning problem within an online learning process. In these types of problems, we use the knowledge from some source domains to improve the performance of an online learning task in the target domain. Incremental Learning (IL) [46] is another approach that can be used when we are dealing with a data stream and the prediction performance is reduced due to changes in the feature space of the new task. These approaches try to update some characteristics of fixed network structure to handle the problem of the new task. In [47], neural networks with dynamically evolved capacity (NADINE) are proposed, which can update the network structure and improve the prediction performance based on changes in the learning environment.

DA is another model re-usability approach. DA methods are transductive TL approaches with the assumption that the distributions of the source dataset and target dataset are different [48]. In [49], a feature learning approach was proposed that provides domain-invariant features. In [50], a Deep Adaptation Network (DAN) architecture was proposed that uses maximum mean discrepancy (MMD) [51] to find a domain-invariant feature space. In [24], it was shown that supervised DA can be seen as a two-view Graph Embedding. In [52], a method was proposed that combines DA techniques and drift handling mechanism to solve the multi-stream classification problem under multisource streams. In multistream classification [52,53], we have two datasets of source and target stream data which come from the same domain. The source stream dataset consists of labeled data, while the target stream dataset is unlabeled. The task is to predict the class labels of the target stream data and address challenges related to infinite length and concept drift. In this type of problem, we have access to the data of both the source and target data. One of the major differences between TL and DA is that DA requires all the data of both the source and target domains. As the source and target models need to be trained jointly, it is memory-intensive since the parameters of both models need to be updated [34]. When there is a major difference between the distribution of the base and target datasets, the performance of the base model may deteriorate. Multi-domain learning [54] methods incorporate the properties of multi-task learning [55] and domain adaptation. In multi-domain learning, the goal is to handle the same problem for different domains. In [56], an adaptive method for multi-domain learning is proposed that reduced the required base model parameters based on the complexity of the different domains coming from image classification problems.

2.2. Temporal attention-augmented bilinear layer

We will demonstrate our model augmentation approach with an instantiation of the Temporal Attention-augmented Bilinear Layer (TABL) network [29], which has been proposed as an efficient and effective neural network architecture for financial time-series classification. In the following, we describe the working mechanism of the TABL network architecture, providing necessary details to understand our method described in Section 3.

TABL combines the ideas of bilinear projection and attention mechanism. The input to a TABL is a multivariate time-series $\mathbf{X} \in \mathbb{R}^{D\times T}$, with T denoting the number of instances combined in the temporal dimension to form the time-series and D denoting the dimensionality of the instances forming the time-series. Given an input to the TABL, it generates an output multivariate time-series $\mathbf{Y} \in \mathbb{R}^{D'\times T'}$, where T' and D' denote the transformed number of instances and their dimensionality, respectively. This is performed through five computation steps:

1. A feature transformation of the input $\mathbf{X}$ is performed using a weight matrix $\mathbf{W}_1 \in \mathbb{R}^{D'\times D}$, producing the intermediate feature matrix $\bar{\mathbf{X}} \in \mathbb{R}^{D'\times T}$:
$$\bar{\mathbf{X}} = \mathbf{W}_1\mathbf{X}. \tag{1}$$
2. The relative importance of the instances forming the time-series is computed by:
$$\mathbf{E} = \bar{\mathbf{X}}\mathbf{W}, \tag{2}$$
where $\mathbf{W} \in \mathbb{R}^{T\times T}$ is a structured matrix that has fixed diagonal elements equal to $1/T$. By learning non-diagonal elements, the matrix $\mathbf{W}$ expresses weights encoding the pair-wise instance importance in the transformed feature space $\mathbb{R}^{D'}$, while the self-importance of all instances is set to be equal.
3. The importance scores stored at the elements e_{ij} of $\mathbf{E}$ are normalized in a row-wise manner to produce an attention mask $\mathbf{A} \in \mathbb{R}^{D'\times T}$ formed by the attention scores. This is done using the soft-max function:
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T}\exp(e_{ik})}. \tag{3}$$
Each row of the attention matrix $\mathbf{A}$ sums up to 1, distributing the importance scores along the time dimension for each dimension of the transformed input time-series data.
4. The attended features $\tilde{\mathbf{X}} \in \mathbb{R}^{D'\times T}$ are computed by applying the attention mask to $\bar{\mathbf{X}}$. To enable a soft attention, a learnable parameter λ is used to weight the contribution of the attended $\bar{\mathbf{X}} \odot \mathbf{A}$ and the original features $\bar{\mathbf{X}}$:
$$\tilde{\mathbf{X}} = \lambda(\bar{\mathbf{X}} \odot \mathbf{A}) + (1-\lambda)\bar{\mathbf{X}}. \tag{4}$$
λ is constrained to have a value between $[0, 1]$.
5. The final attended features $\tilde{\mathbf{X}}$ are linearly transformed in the second tensor mode by weight matrix $\mathbf{W}_2 \in \mathbb{R}^{T\times T'}$, shifted by the bias $\mathbf{B} \in \mathbb{R}^{D'\times T'}$, and activated by a nonlinear element-wise activation function $\phi(\cdot)$, e.g., the Rectified Linear Unit (ReLU) function:
$$\mathbf{Y} = \phi\left(\tilde{\mathbf{X}}\mathbf{W}_2 + \mathbf{B}\right). \tag{5}$$

A schematic illustration of the above computation steps is provided in Fig. 1 (top).

A TABL network is created by combining multiple TABLs or by combining TABLs with Bilinear Layers (BL). A BL is a layer in which the temporal attention branch is not used, or equivalently the value of parameter λ is set to zero.

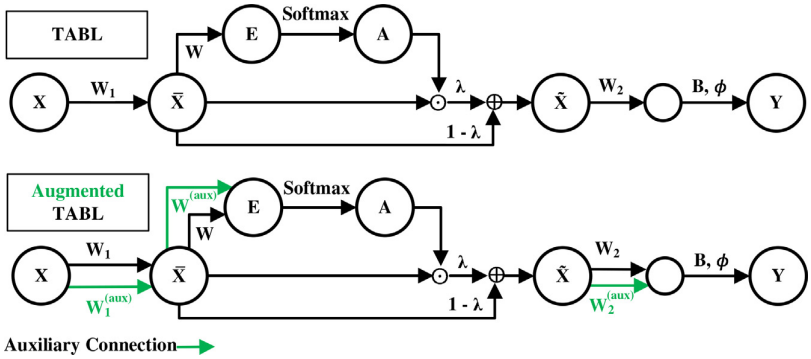

Fig. 1. Illustration of original TABL [29] (top) and the proposed Augmented TABL (bottom). Auxiliary connections are shown in green color. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3. Proposed method

The proposed method is based on two main ideas, i.e. model augmentation and low-rank approximation. Although the method presented in this Section is described in detail for the TABL network architecture, this approach can be easily generalized to other architectures, for example, the Convolutional Neural Network architecture as we will show later in this section.

Based on the problem definition described in the Introduction, we want to use the information encoded in a $\mathcal{N}_{\text{old}}$ that was trained on $\mathcal{D}_{\text{old}}$ to improve the prediction performance for $\mathcal{D}_{\text{new}}$ with the following restrictions:

1. without using the data in $\mathcal{D}_{\text{old}}$, as this data may not be available or may be very big leading to very computationally costly training for $\mathcal{D}_{\text{new}}$,
2. without major impact on the performance of $\mathcal{N}_{\text{old}}$ on the original task, i.e., $\mathcal{T}_{\text{old}}$,
3. without increasing the memory and the computation requirements much for operating on both tasks $\mathcal{T}_{\text{old}}$ and $\mathcal{T}_{\text{new}}$, after $\mathcal{N}_{\text{new}}$ is trained and deployed.

This is achieved by augmenting the parameters $\mathbf{\Theta}_{\mathcal{N}_{\text{old}}}$ of $\mathcal{N}_{\text{old}}$ with auxiliary parameters $\mathbf{\Theta}_{\text{aux}}$. That is, $\mathcal{N}_{\text{new}}$ is constructed by adding additional connections to the pre-trained neural network $\mathcal{N}_{\text{old}}$. In order to retain the prior knowledge in $\mathcal{N}_{\text{old}}$, its parameters $\mathbf{\Theta}_{\mathcal{N}_{\text{old}}}$ are fixed and we only optimize the auxiliary parameters $\mathbf{\Theta}_{\text{aux}}$ to learn additional information needed to perform well on the stocks defining $\mathbf{T}_{\text{new}}$.

There are different strategies to incorporate auxiliary connections to a pre-trained model. One of the most common approaches in transfer learning is to add new hidden layers after the last hidden layer, thereby extending the network's depth and replacing the prediction layer, also known as the penultimate layer. Instead of modifying the pre-trained model's topology, i.e., the number of layers and the size of each layer, we propose to augment the pre-trained model with auxiliary connections that are parallel to the existing ones, thereby keeping the original architecture design unchanged. The intuition behind this approach is that the architectural choice of a pre-trained model is often validated and obtained by extensive experimentation process. It has been shown that many state-of-the-art network architectures such as ResNet-50 [57] or DenseNet121 [58] are not specific to a dataset, but perform well in many similar problems. Thus, by respecting the architectural choices of the pre-trained network $\mathcal{N}_{\text{old}}$, we can avoid the time-consuming process of validating architectural choices for the auxiliary connections.

3.1. Augmented TABL

The proposed model augmentation method is illustrated in Fig. 1. As can be seen from this figure, we incorporate into the pre-trained TABL three auxiliary connections (depicted in green color), which are parameterized by $\mathbf{W}_1^{(\text{aux})}$, $\mathbf{W}^{(\text{aux})}$ and $\mathbf{W}_2^{(\text{aux})}$, respectively. The dimensions of auxiliary parameters are exactly the same as those that they are augmenting. Particularly, $\mathbf{W}_1^{(\text{aux})} \in \mathbb{R}^{D' \times D}$ is added to augment $\mathbf{W}_1$; $\mathbf{W}^{(\text{aux})} \in \mathbb{R}^{T \times T}$ is added to augment $\mathbf{W}$; and $\mathbf{W}_2^{(\text{aux})} \in \mathbb{R}^{T \times T'}$ is added to augment $\mathbf{W}_2$. The transformations produced by the augmented TABL are described by the following equations:

$$\bar{\mathbf{X}} = \mathbf{W}_1\mathbf{X} + \mathbf{W}_1^{(\text{aux})}\mathbf{X}, \tag{6}$$

$$\mathbf{E} = \bar{\mathbf{X}}\mathbf{W} + \bar{\mathbf{X}}\mathbf{W}^{(\text{aux})}, \tag{7}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T}\exp(e_{ik})}, \tag{8}$$

$$\tilde{\mathbf{X}} = \lambda(\bar{\mathbf{X}} \odot \mathbf{A}) + (1-\lambda)\bar{\mathbf{X}}, \tag{9}$$

$$\mathbf{Y} = \phi\big(\tilde{\mathbf{X}}\mathbf{W}_2 + \tilde{\mathbf{X}}\mathbf{W}_2^{(\text{aux})} + \mathbf{B}\big). \tag{10}$$

In Eq. (6), the intermediate feature matrix that is formed using the $\mathbf{W}_1$ of pre-trained model is added to the transformed features matrix that is produced using auxiliary parameter $\mathbf{W}_1^{(\text{aux})}$. The matrix $\mathbf{E}$ (Eq. (7)) which shows the relative importance of instances is produced by summing up the output of Eq. (6) with both $\mathbf{W}$ and $\mathbf{W}^{(\text{aux})}$. In the Eq. (10), the final attended features that are produced in Eq. (9) are multiplied by both $\mathbf{W}_2$ and $\mathbf{W}_2^{(\text{aux})}$ and added together. The output of this summation is shifted by $\mathbf{B}$ and activated by an activation function. As can be seen from Eqs. (6)–(10), for each computation step that involves a weight matrix, we retain the transformations used by the pre-trained TABL, while learning additional (complementary) transformations through the auxiliary parameters ($\mathbf{W}_1^{(\text{aux})}$, $\mathbf{W}^{(\text{aux})}$, $\mathbf{W}_2^{(\text{aux})}$). By making the auxiliary computation steps mimic those in the original TABL, we respect the architectural design of the pre-trained model. In addition, since only the auxiliary parameters are updated during the optimization process, intermediate knowledge of the pre-trained model (preserved in its weight matrices) is still retained and we only learn auxiliary information to improve performance in the new task at hand.

3.2. Low rank augmented TABL

Besides model augmentation, our method exploits the idea of low-rank approximation. That is, we enforce a constraint (an upper-bound K) to the rank of each auxiliary weight matrix by representing it as a two-factor decomposition. Specifically, we define the auxiliary weight matrices as:

$$\mathbf{W}_1^{(\text{aux})} = \mathbf{L}_1\mathbf{R}_1, \tag{11}$$

$$\mathbf{W}^{(\text{aux})} = \mathbf{L}\mathbf{R}, \tag{12}$$

$$\mathbf{W}_2^{(\text{aux})} = \mathbf{L}_2\mathbf{R}_2, \tag{13}$$

where $\mathbf{L}_1 \in \mathbb{R}^{D'\times K}, \mathbf{R}_1 \in \mathbb{R}^{K\times D}, \mathbf{L} \in \mathbb{R}^{T\times K}, \mathbf{R} \in \mathbb{R}^{K\times T}, \mathbf{L}_2 \in \mathbb{R}^{T\times K}, \mathbf{R}_2 \in \mathbb{R}^{K\times T'}$, with $K \leq \min(D, D', T, T')$ being a hyperparameter value.

By constraining the value of K to a small number, we enforce the weight matrices $\mathbf{W}_1^{(\text{aux})}$, $\mathbf{W}^{(\text{aux})}$ and $\mathbf{W}_2^{(\text{aux})}$ to have low ranks. The advantage of the low-rank approximation is two-fold. The first advantage is the reduction in computational and memory complexities during the optimization process. This will be analyzed in detail in the next subsection. In addition to improvements in complexity, the use of low-rank approximation can also have a regularization effect, thus improving the learning performance of the neural network. This is because low-rank approximation reduces the degrees of freedom in a given transformation, i.e., the number of parameters to be estimated, thus, it potentially reduces the overfitting effect when sufficient training data is not available.

To summarize, to take advantage of the pre-trained model to solve the new task $\mathcal{T}_{\text{new}}$ defined by the dataset $\mathcal{D}_{\text{new}}$, we solve the following optimization objective:

$$\arg\min_{\substack{\mathbf{L}_1^{(l)},\mathbf{L}^{(l)},\mathbf{L}_2^{(l)},\\ \mathbf{R}_1^{(l)},\mathbf{R}^{(l)},\mathbf{R}_2^{(l)}\\ l=1,\dots,L}} \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\mathcal{N}_{\text{new}}(x_i, y_i)), \tag{14}$$

where $(x_i, y_i) \in \mathcal{D}_{\text{new}}$ denotes the ith training time-series and the corresponding target in the new dataset, L denotes the number of TABLs in the new model, and $\mathcal{L}$ denotes the loss function.

3.3. Complexity analysis

In this subsection, we provide our analysis on the complexity of the proposed method, as well as some notes on the implementation details. The first thing we should point out about the complexity of our method is that the inference complexity of the new model $\mathcal{N}_{\text{new}}$ is exactly the same as that of the pre-trained model $\mathcal{N}_{\text{old}}$. In other words, after the transfer learning process, the new model induces the same operating cost as the old model. This is because of the distributive property of matrix multiplications in Eqs. (6), (7), (10). Let us denote:

$$\begin{aligned}\mathbf{W}_1^{(\text{new})} &= \mathbf{W}_1 + \mathbf{W}_1^{(\text{aux})},\\ \mathbf{W}^{(\text{new})} &= \mathbf{W} + \mathbf{W}^{(\text{aux})},\\ \mathbf{W}_2^{(\text{new})} &= \mathbf{W}_2 + \mathbf{W}_2^{(\text{aux})}.\end{aligned} \tag{15}$$

Equations (6), (7) and (10) become:

$$\bar{\mathbf{X}} = \mathbf{W}_1^{(\text{new})}\mathbf{X}, \tag{16}$$

$$\mathbf{E} = \bar{\mathbf{X}}\mathbf{W}^{(\text{new})}, \tag{17}$$

$$\mathbf{Y} = \phi\big(\tilde{\mathbf{X}}\mathbf{W}_2^{(\text{new})} + \mathbf{B}\big). \tag{18}$$

After training, we can simply compute Eq. (15) once and only store the values of $\mathbf{W}_1^{(\text{new})}, \mathbf{W}^{(\text{new})}, \mathbf{W}_2^{(\text{new})}$ for inference. In this way, we do not need to retain both the values of $\mathbf{W}_1, \mathbf{W}, \mathbf{W}_2$ of the pre-trained model and the values of $\mathbf{L}_1, \mathbf{R}_1, \mathbf{L}, \mathbf{R}, \mathbf{L}_2, \mathbf{R}_2$ of the auxiliary connections, which may consume more storage space than $\mathbf{W}_1^{(\text{new})}, \mathbf{W}^{(\text{new})}, \mathbf{W}_2^{(\text{new})}$. Since the matrix dimensions in Eqs. (16)–(18) for the new model are exactly the same as those in Eqs. (1), (2) and (5) for the pre-trained model, our method does not introduce any additional complexity during the inference/deployment phase.

The complexity during the training phase is dependent on the implementation details. Here we should note that there are two strategies to implement the proposed augmented TABL:

- **Implementation Strategy 1 (IS1)**: during the forward pass, using Eqs. (11)–(13), we first compute $\mathbf{W}_1^{(\text{aux})}$, $\mathbf{W}^{(\text{aux})}$ and $\mathbf{W}_2^{(\text{aux})}$ from $\mathbf{L}_1, \mathbf{R}_1$, $\mathbf{L}$, $\mathbf{R}$, $\mathbf{L}_2$ and $\mathbf{R}_2$. After that, using Eq. (15), we can compute $\mathbf{W}_1^{(\text{new})}$, $\mathbf{W}^{(\text{new})}$, $\mathbf{W}_2^{(\text{new})}$, which are then used in Eqs. (16)–(18) to compute the output of the augmented TABL.
- **Implementation Strategy 2 (IS2)**: in this case we do not compute explicitly the auxiliary weight matrices $\mathbf{W}_1^{(\text{aux})}$, $\mathbf{W}^{(\text{aux})}$ and $\mathbf{W}_2^{(\text{aux})}$ but we take advantage of the size of their low-rank approximations. Specifically, Eqs. (6), (7) and (10) are computed as follows, with the computation order from left to right and by respecting the priority of the parentheses:

$$\bar{\mathbf{X}} = \mathbf{W}_1\mathbf{X} + \mathbf{L}_1(\mathbf{R}_1\mathbf{X}), \tag{19}$$

$$\mathbf{E} = \bar{\mathbf{X}}\mathbf{W} + (\bar{\mathbf{X}}\mathbf{L})\mathbf{R}, \tag{20}$$

$$\mathbf{Y} = \phi\big(\tilde{\mathbf{X}}\mathbf{W}_2 + (\tilde{\mathbf{X}}\mathbf{L}_2)\mathbf{R}_2 + \mathbf{B}\big). \tag{21}$$

Each of the above-mentioned strategies has its own advantages in terms of computational and memory complexity. Adopting the first implementation strategy (IS1) will lead to a faster forward-backward pass compared to the second strategy (IS2). This is because during stochastic optimization, we often update multiple samples in the same forward-backward pass, thus leading to $\mathbf{X}$ having another large dimension, which corresponds to the mini-batch size. Since each equation in IS2 involves two matrix multiplications with $\mathbf{X}$ or $\bar{\mathbf{X}}$ or $\tilde{\mathbf{X}}$, IS2 requires more calculations compared to IS1. The computational complexity estimates of IS1 and IS2 can be found in the Appendix.

While adopting IS1 can lead to a faster forward-backward pass compared to IS2, this strategy also requires a higher amount of memory, which might not be feasible when training large networks on a GPU with limited memory. This is because using IS2, during the forward pass, we do not need to keep the intermediate outputs of $\mathbf{W}_1\mathbf{X}$ in Eq. (19), $\bar{\mathbf{X}}\mathbf{W}$ in Eq. (20) and $\tilde{\mathbf{X}}\mathbf{W}_2$ in Eq. (21) for the gradient update computation in the backward pass. On the other hand, when using IS1 we need to keep all intermediate outputs in the forward pass in order to compute the gradient updates for the backward pass. In addition, since the auxiliary weight matrices are explicitly computed in the forward pass of IS1, we do not obtain any memory reduction from using the low-rank approximation as is the case for IS2.

3.4. Augmented convolution layer

As we mentioned in the beginning of this section, the proposed approach can be easily generalized to other architectures since most neural networks rely on linear or multilinear transformations.

Table 1
Distribution of classes in train and test dataset in FI-2010. Class 0 refers to the stock price remains the same, class 1 to the situation where the price goes up, and class 2 where the price goes downs.

Target stock	Train dataset			Test dataset		
	Class 0	Class 1	Class 2	Class 0	Class 1	Class 2
Stock 1	9,013	8,198	8,060	2,108	2,507	2,440
Stock 2	28,487	11,235	11,394	16,712	4,629	4,482
Stock 3	40,489	10,228	10,452	26,067	4,136	3,974
Stock 4	22,258	9,983	10,263	17,326	4,661	4,333
Stock 5	61,116	21,991	21,790	38,829	9,458	8,756
All Stocks	161,363 (56%)	61,635 (21%)	61,959 (21%)	101,042 (67%)	25,391 (16%)	23,985 (15%)

For Convolutional Neural Networks (CNNs), an augmented convolution layer can be formed by incorporating low-rank auxiliary filters to the pre-trained filters in a convolution layer. Since the filters in a convolution layer can be represented as a 3-mode (1D convolution layer) or 4-mode (2D convolution layer) tensor, the low-rank auxiliary filters can be represented in the Canonical Polyadic (CP) form [59], similar to the low-rank CNN proposed in [16]. Let us denote $\mathcal{W}$ as the pre-trained convolution filters in a convolution layer, and $\mathbf{X}$ as the input time-series. Similar to an augmented TABL, The augmented convolution layer has the following form:

$$\mathbf{Y} = \mathbf{X} \circledast \mathcal{W} + \mathbf{X} \circledast \mathcal{W}^{(\text{aux})}, \tag{22}$$

where $\mathbf{Y}$ denotes the output of the convolution layer, $\circledast$ denotes the convolution operation, and $\mathcal{W}^{(\text{aux})}$ denotes the low-rank auxiliary filters.

The convolutional architectures for time-series often consist of 1D convolution layers. Thus, $\mathcal{W}$ is a 3-mode tensor, i.e., $\mathcal{W} \in \mathbb{R}^{N \times D \times t}$ with N denotes the number of filters, D denotes the number of time-series and t denotes the kernel size. Using the CP form, $\mathcal{W}^{(\text{aux})}$ can be represented as:

$$\mathcal{W}^{(\text{aux})} = \sum_{k=1}^{K} \mathbf{w}_1(k) \otimes \mathbf{w}_2(k) \otimes \mathbf{w}_3(k), \tag{23}$$

where $\mathbf{w}_1(k) \in \mathbb{R}^{N \times 1 \times 1}$, $\mathbf{w}_2(k) \in \mathbb{R}^{1 \times D \times 1}$, $\mathbf{w}_3(k) \in \mathbb{R}^{1 \times 1 \times t}$. $\otimes$ denotes the outer product and K denotes the rank hyperparameter.

4. Experiments

4.1. Dataset and experimental protocols

In order to evaluate the effectiveness of the proposed method in the research problem defined in Section 1, we conducted experiments in the problem of stock mid-price direction prediction using the information appearing in Limit Order Books (LOB). Mid-price is the average value between the best bid (buy) price and the best ask (sell) price of a given stock. Although this quantity is just a virtual quantity, i.e., at a given time instance, no transaction can happen at the mid-price, the direction of mid-price change can capture the dynamics of a given stock, reflecting the supply and demand in the market. For this reason, mid-price direction prediction is a popular problem when analyzing LOB information [60].

The dataset used in our experiments is a public LOB dataset known as FI-2010 [61], which consists of more than four million limit orders during the period of ten working days (from 1st of June to 14th of June 2010). The dataset contains limit orders for five companies traded in the Helsinki Exchange, operated by Nasdaq Nordic. At each time instance, the dataset provides the quotes of the top ten levels of the LOB, i.e., the top ten best bid prices and volumes and top ten best ask prices and volumes. The top quotes form 40 different values corresponding to the bid and ask prices and volumes of the top 10 LOB levels. Regarding the labels of the mid-price, at any given time instance, the database provides the direction (stationary, increasing, decreasing) of the mid-price after the next H time instances, where $H \in \{10, 20, 30, 50, 100\}$ denotes the prediction horizon.

We followed a similar experimental protocol as in [29]: the input to all of the evaluated models was formed from the 10 most recent limit order events, which consists of 10 instances of 40 dimensions standardized using z-score normalization. As has been shown in [62], the adoption of information from all 10 LOB levels is important for achieving high performance in learning-based methods. All models were trained to predict the mid-price direction after 10 events, i.e., $H = 10$. Regarding the train and test datasets, we used time-series of the first seven days for training and time-series of the last three days for testing. From the training set, we used the last 10% of the time-series samples for validation purposes. In addition, we also evaluated our model augmentation strategy in an online learning setting, which is described in detail in Section 4.4. All models were optimized using the Adam optimizer with an initial learning rate of 0.01. The learning rate was reduced whenever the validation loss reached a plateau.

Since FI-2010 is an imbalanced dataset with the majority of labels belonging to the stationary class, F1-score is used as the main performance metric, similar to prior works [2,4,29]. The class distributions of 5 stocks individually and the overall class distributions for the dataset can be seen in Table 1. In addition to F1-score, we also report average accuracy, precision and recall. Finally, we adopted the same weighted entropy loss function as in [29] to alleviate the effects of class imbalance:

$$L = -\sum_{c=1}^{3} \frac{\beta}{N_c} y_c \log(\tilde{y}_c), \tag{24}$$

where the N_c is the number of samples in cth class, and the y_c and $\tilde{y}_c$ are the true and predicted probabilities of cth class respectively. β is a constant with a value of $1e6$. All experiments were run 5 times and we report the mean and standard deviation of 5 runs for each performance metric. Because of the random model parameter initialization and the stochastic nature of the Backpropagation algorithm, it is often the case that we obtain different network parameters, thus different performances at different runs. The use of the standard deviation of the performance achieved over different runs helps us evaluate how stable and reliable our model is given that it is trained using a stochastic optimization process.

4.2. Experimental setup 1

In order to simulate an experiment following the definition of the research problem defined in Section 1, we conducted experiments for both TABL and CNN architectures with the following setup:

- For each stock S in the database, we consider the data belonging to S as the new dataset $\mathcal{D}_{\text{new}}$ and the data belonging to the remaining four stocks as the old dataset $\mathcal{D}_{\text{old}}$.
- We use the notation `TABL-base` and `CNN-base` to denote the models that were trained on the $\mathcal{D}_{\text{old}}$ dataset. This is the pre-trained model $\mathcal{N}_{\text{old}}$ in our problem formulation. This model,

when used directly to evaluate on the new dataset $\mathcal{D}_{\text{new}}$, can serve as a simple baseline to compare with our method.
- A TL approach taking advantage of the pre-trained model (`TABL-base` and `CNN-base`) and finetuning it on the new dataset $\mathcal{D}_{\text{new}}$ is also used as a baseline. We denote models following this TL approach as `TABL-fine-tune` and `CNN-fine-tune`.
- In order to simulate a scenario where only $\mathcal{D}_{\text{new}}$ is available, we train TABL and CNN models from scratch (random initialization) using $\mathcal{D}_{\text{new}}$, which we refer to as `TABL-target` and `CNN-target` respectively.
- The augmented TABL models obtained by our method with two implementation strategies are denoted as `aTABL-IS1` and `aTABL-IS2`, respectively. Similarly, the augmented CNN model denoted as `aCNN`. Here we should note that, similar to `aTABL`, there are also two implementation strategies for `aCNN`. In our experiments, we simply used the second implementation strategy for `aCNN`.
- In addition to the above, we also train a TABL model from scratch using both $\mathcal{D}_{\text{old}}$ and $\mathcal{D}_{\text{new}}$. This model, denoted as `TABL`, represents the scenario where we have access to both the old and new datasets at once. Similarly, `CNN` is used to denote the model that was trained using both $\mathcal{D}_{\text{old}}$ and $\mathcal{D}_{\text{new}}$.

To find the best network architecture for the pre-trained models (`TABL-base`), several configurations for hidden layers were validated. We followed the design in [29], i.e., all hidden layers except the last one are Bilinear layers and the prediction layer is a TABL. Grid search was used for hyperparameter tuning. The results of ablation study for hidden layers can be seen in Appendix B. The best network architectures for each pre-trained model corresponding to each experiment can be seen in Fig. 6. For the CNN architecture, we adopted a conventional design pattern of 1D CNN for time-series, which is shown in Fig. 2.

Tables 2 and 3 provide the prediction performance of all models on the test sets defined on each experiment. The results are grouped based on the target stock S. At the bottom of Tables 2 and 3, we report the average performance over the five target stocks. In addition, the last column shows the maximum rank value (chosen through validation) associated with our methods. As we have described two implementation strategies for the augmented TABL in Section 3, we also conducted the experiments and report the performance obtained by using these implementation strategies, denoted as `aTABL-IS1` and `aTABL-IS2` in Table 2.

From the experimental results in Tables 2 and 3, we can clearly see that on average, the proposed model augmentation approach leads to performance improvements compared to the standard finetuning approach for both TABL and CNN network architectures. In addition, by adapting the pre-trained models on the new data using the proposed augmentation approach, we indeed observed performance improvements compared to the scenario when the pre-trained models are not adapted to the new data, i.e., `TABL-base` and `CNN-base`. This was not always the case for the standard finetuning approach since we observed performance degradation when comparing the performance of `CNN-fine-tune` and `CNN-base`. According to Table 2, when comparing the performance of `TABL-base` and `TABL`, it appears that `TABL-base` outperforms `TABL` on stocks 1, 2, and 4. The outcome is somewhat unexpected as the training set for the target stock is included in the training set for the `TABL`, so its performance should intuitively be better than `TABL-base`, which is not trained on the target stock dataset. There could be several explanations for this phenomenon. One possibility is the size of the training set for target stocks 3 and 5 (see Table 1), which is larger than that of the other stocks, enabling better prediction

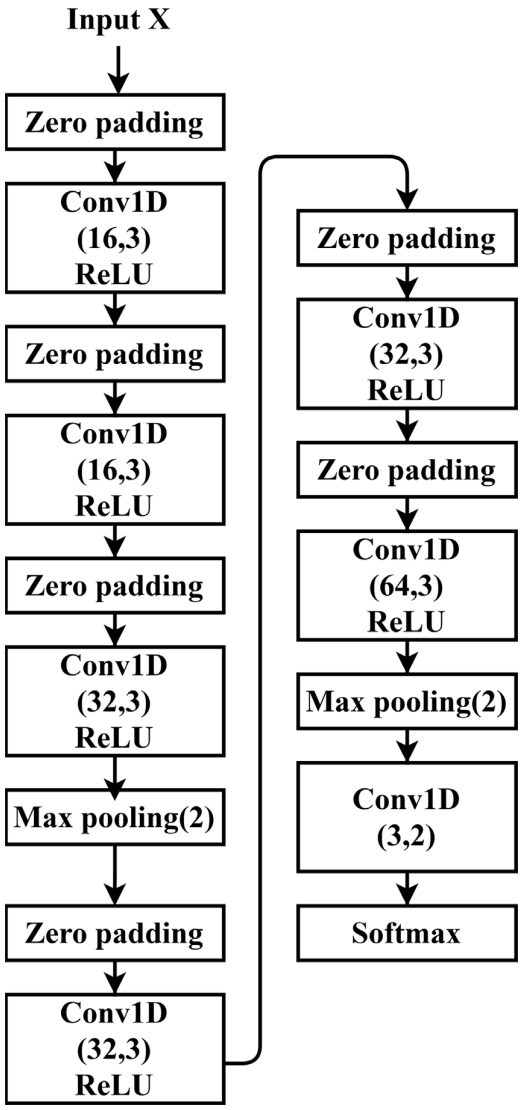

Fig. 2. The CNN architecture with 7 convolution layers used in our experiments. Here the notation Conv1D (a, b) refers to a convolution layer having a filters, each of which has a kernel size of b.

performance on the target stock. Consequently, the `TABL` method has better performance on stocks 3 and 5 than `TABL-base`. Another potential explanation for `TABL-base`'s better performance on stocks 1, 2, and 4 is the possibility of presence of noise or random fluctuations in those stocks, which had a detrimental effect on the `TABL`'s performance. Overall, Table 2 indicates that, on average, `TABL` outperforms `TABL-base`. The experiments conducted with `TABL-target` and `CNN-target` models indicate that relying solely on the target stock for training is insufficient to achieve competitive performance in most cases. Although the training set size for stock 5 is the largest (see Table 1), which may explain why `CNN-target` performed better for that stock than for others, it is not the best performing approach for either CNN or TABL models. Notably, the `aTABL-IS2` achieved significantly better performance for stock 5. Regarding the two implementation strategies conducted for the TABL architectures, we can see that both strategies lead to very similar results. The differences stem from the stochastic nature of the optimizer as well as the initialization of the network parameters. As previously discussed, the two imple-

Table 2
Performance of TABL networks (Mean ± Standard Deviation) on the test sets of the Experimental Setup 1.

Target	Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Max Rank
Stock 1	TABL	62.40 ± 0.10	62.10 ± 0.10	62.10 ± 0.10	62.08 ± 0.10	-
	TABL-base	63.96 ± 0.20	65.01 ± 0.30	64.45 ± 0.30	63.98 ± 0.20	-
	TABL-fine-tune	64.97 ± 0.40	66.66 ± 0.50	65.73 ± 0.40	64.94 ± 0.40	-
	TABL-target	4707 ± 0.20	4695 ± 0.10	4685 ± 0.20	4683 ± 0.20	-
	aTABL-IS1	**65.41 ± 0.40**	**68.68 ± 0.70**	**66.49 ± 0.40**	**65.28 ± 0.40**	**8**
	aTABL-IS2	**65.37 ± 0.30**	**68.07 ± 0.50**	**66.34 ± 0.30**	**65.28 ± 0.30**	**9**
Stock 2	TABL	78.44 ± 0.10	74.64 ± 0.20	66.26 ± 0.00	69.43 ± 0.00	-
	TABL-base	79.48 ± 0.40	77.21 ± 0.01	66.49 ± 0.10	70.31 ± 0.20	-
	TABL-fine-tune	79.95 ± 01.00	79.97 ± 03.00	65.22 ± 0.10	69.91 ± 0.70	-
	TABL-target	7252 ± 0.10	6432 ± 0.10	6326 ± 0.10	6375 ± 0.10	-
	aTABL-IS1	**81.07 ± 0.30**	**83.48 ± 01.00**	**65.72 ± 0.10**	**71.11 ± 0.20**	**4**
	aTABL-IS2	**80.86 ± 0.30**	**82.62 ± 01.40**	**65.77 ± 0.20**	**70.98 ± 0.20**	**1**
Stock 3	TABL	84.94 ± 0.10	75.65 ± 0.30	66.22 ± 0.10	70.03 ± 0.20	-
	TABL-base	82.96 ± 01.30	71.14 ± 02.50	66.17 ± 0.20	68.25 ± 01.20	-
	TABL-fine-tune	84.61 ± 01.30	76.35 ± 04.10	64.79 ± 0.50	69.04 ± 01.10	-
	TABL-target	7575 ± 0.30	5930 ± 0.20	6451 ± 0.10	6141 ± 0.20	-
	aTABL-IS1	**85.01 ± 01.90**	**77.74 ± 06.70**	**65.60 ± 0.30**	**70.02 ± 01.90**	**1**
	aTABL-IS2	**85.82 ± 0.40**	**80.01 ± 01.90**	**65.26 ± 0.10**	**70.58 ± 0.50**	**4**
Stock 4	TABL	77.55 ± 0.10	71.76 ± 00.20	64.25 ± 00.20	66.41 ± 0.10	-
	TABL-base	76.27 ± 03.20	69.47 ± 05.30	66.98 ± 0.60	67.74 ± 02.40	-
	TABL-fine-tune	80.96 ± 0.40	80.66 ± 0.90	65.33 ± 0.80	70.20 ± 0.80	-
	TABL-target	7507 ± 0.10	6677 ± 0.20	6538 ± 0.20	6602 ± 0.20	-
	aTABL-IS1	**81.38 ± 0.50**	**80.87 ± 01.60**	**66.55 ± 0.30**	**71.32 ± 0.40**	**1**
	aTABL-IS2	**81.45 ± 0.30**	**81.10 ± 01.30**	**66.70 ± 0.30**	**71.48 ± 0.20**	**1**
Stock 5	TABL	76.94 ± 0.10	70.03 ± 00.20	59.90 ± 0.20	63.39 ± 0.20	-
	TABL-base	67.72 ± 0.40	56.83 ± 0.7	59.82 ± 0.30	57.81 ± 0.10	-
	TABL-fine-tune	77.35 ± 0.90	68.61 ± 01.50	64.57 ± 0.70	66.16 ± 01.20	-
	TABL-target	7455 ± 0.20	6347 ± 0.20	6320 ± 0.10	6330 ± 0.10	-
	aTABL-IS1	**78.50 ± 02.90**	**71.53 ± 06.00**	**63.59 ± 03.20**	**66.47 ± 04.10**	**4**
	aTABL-IS2	**78.77 ± 02.10**	**72.24 ± 04.00**	**64.43 ± 01.40**	**67.18 ± 02.60**	**6**
Average	TABL	76.05 ± 08.30	70.84 ± 05.40	63.75 ± 02.70	66.27 ± 03.50	-
	TABL-base	74.08 ± 08.00	67.93 ± 07.60	64.78 ± 02.90	65.62 ± 04.90	-
	TABL-fine-tune	77.57 ± 07.50	74.45 ± 06.50	65.13 ± 00.50	68.05 ± 02.40	-
	TABL-target	6920 ± 06.90	6003 ± 05.20	6424 ± 02.10	6026 ± 02.50	-
	aTABL-IS1	**78.27 ± 07.60**	**76.46 ± 06.20**	**65.59 ± 01.20**	**68.84 ± 02.80**	**-**
	aTABL-IS2	**78.45 ± 07.80**	**76.81 ± 06.30**	**65.70 ± 00.90**	**69.10 ± 02.70**	**-**

mentation strategies for the TABL architecture have varying computational complexities. In cases where the two strategies produce similar prediction performances, the efficiency of model training can be a deciding factor in selecting the optimal model. Evaluating the time complexity of a model can be done by assessing the number of floating point operations (FLOPs) required to run a single instance of the model, while the space complexity can be evaluated through the number of parameters. Table 4 displays the number of FLOPs and training parameters for each stock, comparing the aTABL-IS1 and aTABL-IS2 models from Table 2. Although the network structures of the augmented models for each stock are the same, the implementation strategy and model rank significantly affect the efficiency of the models. For stocks 1, 2, 3, and 5, there are major differences in both FLOPs and number of parameters, which can be attributed to the rank of the best model for each strategy. However, for stock 4, as the rank for both strategies is the same, the numbers of parameters for both models are equal, and the difference in FLOPs is due to the implementation strategies which in this case the aTABL-IS2 has the higher number of FLOPs.

The rank value, which is a hyperparameter of the proposed augmentation method, can influence the efficiency of learning complementary knowledge from the new data. As mentioned before in this section, we chose the rank of the augmented models by running experiments on a range of different rank values and selected the one that produces the best F1-score on the training set. Figures 4 and 5 show the effect of different ranks on the test performance of the proposed method. From these figures, it can be seen that there is no clear relation between the rank value and the generalization performance of the model. However, we can see that good performance can be obtained from low values of the rank.

To have a better understanding of the prediction performance on each stock, we provide in Fig. 3 the confusion matrices obtained by using the best model on each stock from the models listed in Tables 2 and 3. As can be seen, for most of the stocks, the stationary class (label 0) is the most populated one. The results show that for all stocks the majority of the predictions are to the stationary class (label 0), while the models can distinguish between the stationary class and the two other classes (up and down corresponding to labels 1 and 2, respectively). Focusing on the classes up and down, we can see that when a time-series is classified to these two classes, it is correct in most of the cases, while misclassifications are mostly directed to the stationary class. When excluding the stationary class, i.e., considering only the cases when the models predict that the mid-price will move up or down which are the cases that would lead to an action, distinguishing between class up (label 1) and class down (label 2) is very accurate. Another performance measure based on the provided confusion matrix is the win-rate, which is calculated by assuming that trades would only be placed when the forecast indicates that the stock price will change. As the prediction of changes is more difficult and important than the prediction of stationary class, the win-rate shows the performance of the proposed method in predicting class 1 and class 2. Table 5 shows the win-rate of all stocks' confusion matrices. As can be seen, our method achieves high performance in predicting the changes.

Our proposed method offers the advantage that the augmented model can be utilized for $\mathcal{T}_{\text{old}}$, and its performance on the $\mathcal{D}_{\text{old}}$

Table 3
Performance of CNNs (Mean ± Standard Deviation) on the test sets of the Experimental Setup 1.

Target	Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Max Rank
Stock 1	CNN	53.93±0.013	53.69±0.014	53.05±0.012	52.45±0.013	-
	CNN-base	52.94±0.005	52.4±0.005	51.78±0.005	50.78±0.006	-
	CNN-fine-tune	55.07±0.0	54.99±0.005	54.67±0.005	54.52±0.006	-
	CNN-target	45.97±0.02	45.48±0.03	45.45±0.03	45.30±0.03	-
	aCNN	**54.74±0.001**	**55.6±0.002**	**54.92±0.001**	**54.77±0.001**	**2**
Stock 2	CNN	77.24±0.013	73.43±0.033	64.93±0.005	67.96±0.009	-
	CNN-base	76.89±0.003	72.19±0.007	65.07±0.002	67.76±0.003	-
	CNN-fine-tune	78.71±0.009	77.13±0.024	64.73±0.002	68.86±0.007	-
	CNN-target	68.35±0.011	59.39±0.13	59.85±0.11	59.61±0.09	-
	aCNN	**78.48±0.008**	**75.73±0.022**	**65.37±0.001**	**69.01±0.006**	**1**
Stock 3	CNN	82.93±0.013	71.77±0.034	64.92±0.006	67.67±0.01	-
	CNN-base	**83.03±0.01**	**71.64±0.025**	**64.99±0.002**	**67.77±0.009**	-
	CNN-fine-tune	76.45±0.042	64.18±0.029	61.4±0.034	60.97±0.042	-
	CNN-target	73.51±0.031	57.21±0.028	62.71±0.023	59.24±0.034	-
	aCNN	82.64±0.008	70.49±0.021	65.32±0.003	67.55±0.007	**1**
Stock 4	CNN	76.9±0.022	70.95±0.041	66.74±0.005	68.16±0.013	-
	CNN-base	74.3±0.028	66.6±0.035	66.16±0.017	66.06±0.026	-
	CNN-fine-tune	79.11±0.028	79.98±0.057	63.44±0.034	67.86±0.037	-
	CNN-target	75,27±0.021	67,76±0.034	63,89±0.03	65,51±0.035	-
	aCNN	**79.42±0.0**	**74.93±0.002**	**67.17±0.0**	**70.13±0.001**	**2**
Stock 5	CNN	67.34±0.032	56.72±0.014	56.75±0.019	56.19±0.003	-
	CNN-base	67.89±0.024	56.29±0.022	56.24±0.012	56.06±0.016	-
	CNN-fine-tune	67.52±0.055	60.78±0.034	53.0±0.021	52.69±0.025	-
	CNN-target	**66.91±0.044**	**55.69±0.04**	**59.50±0.024**	**57.06±0.03**	-
	aCNN	67.99±0.006	56.49±0.006	57.11±0.002	56.69±0.002	2
Average	CNN	72.15±0.11	66.27±0.089	61.78±0.057	63.19±0.071	-
	CNN-base	71.82±0.105	64.56±0.085	61.23±0.058	62.24±0.072	-
	CNN-fine-tune	72.63±0.089	68.56±0.102	59.59±0.054	61.3±0.074	-
	CNN-target	71.01±0.062	60.02±0.09	61.48±0.061	60.35±0.059	-
	aCNN	**72.77±0.101**	**66.54±0.092**	**62.0±0.051**	**63.6±0.068**	**-**

Table 4
Number of FLOPs and training parameters (#Params) of aTABL-IS1 and aTABL-IS2 models in Table 2.

Target	Stock 1		Stock 2		Stock 3		Stock 4		Stock 5	
Method	aTABL-IS1	aTABL-IS2	aTABL-IS1	aTABL-IS2	aTABL-IS1	aTABL-IS2	aTABL-IS1	aTABL-IS2	aTABL-IS1	aTABL-IS2
Rank	8	9	4	1	1	4	1	1	6	4
FLOPs	160,439	245,439	137,356	125,827	142,486	204,862	191,659	200,939	222,299	275,639
#Params	6,159	6,554	2,879	1,658	2,318	3,939	3,238	3,238	4,904	5,744

Fig. 3. Confusion matrices for the best model on the test set of each stock among the models in Tables 2 and 3. Class 0 refers to the stock price remains the same, class 1 to the situation where the price goes up, and class 2 is where the price goes downs.

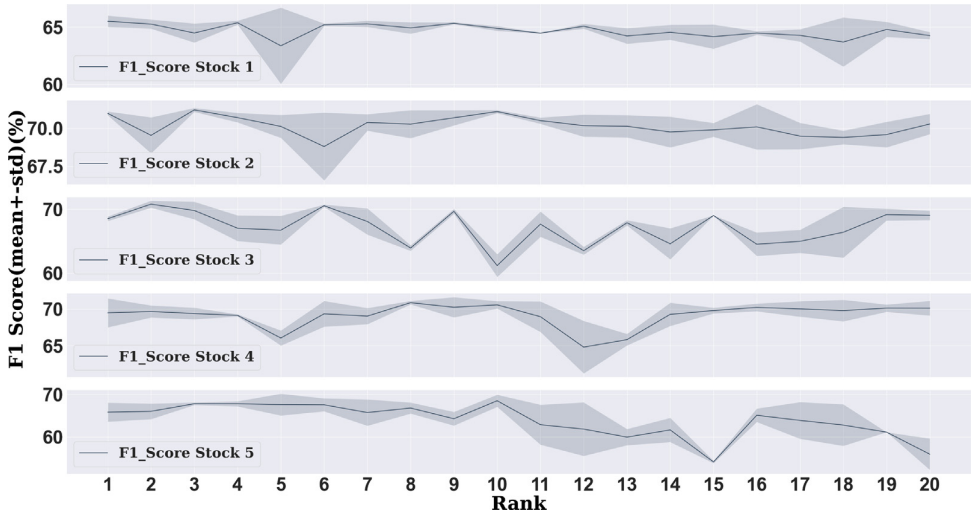

Fig. 4. Performance of **aTABL-IS1** for values of K (rank of the model's weight matrices) between 1 to 20. The shadow of each line shows the standard deviation of results for each rank.

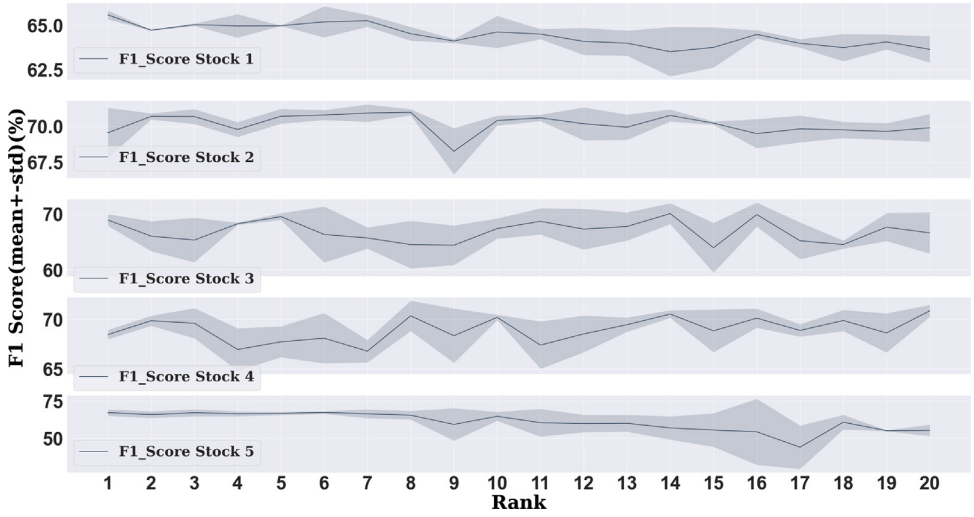

Fig. 5. Performance of **aTABL-IS2** for values of K (rank of the model's weight matrices) between 1 to 20. The shadow of each line shows the standard deviation of results for each rank.

Table 5
Win-rate measured on the test set of each stock.

	Stock 1	Stock 2	Stock 3	Stock 4	Stock 5
Win-Rate(%)	74.6	85.5	79.8	82.3	64.6

dataset does not significantly decrease when compared to that of the base model. To evaluate this, we calculated the performance metrics of augmented and fine-tuned models for each $\mathcal{D}_{\text{old}}$ dataset related to a target stock, and compared their performance with that of the base model. The results are presented in Table 8, where the target column specifies the test dataset for each block of target stocks. The method column shows the base model trained on $\mathcal{D}_{\text{old}}$, indicated by "base", and the "augmented" and "fine-tune" models are the models of our proposed method and finetuning approach respectively. We selected the best augmented models from Tables 2 and 3 to compare with other methods. Based on the results, it can be observed that the augmented model generally has slightly lower performance than the base model. The fine-tuned model experiences a significant reduction in performance compared to the base model for all target stocks. Here it should be further noted that, by construction of the proposed solution, it is possible to predict for the $\mathcal{T}_{\text{old}}$ without using auxiliary connections. In this case, the model reverts to its base model providing the same performance as the based model, whereas the fine-tuned model loses this capability as its weights are altered during the finetuning process.

4.3. Experimental setup 2

To have a better real-world understanding of the advantages of the proposed method, we define another experimental setup as follows: the old dataset $\mathcal{D}_{\text{old}}$ consists of three stocks from the FI-2010 database and the new dataset $\mathcal{D}_{\text{new}}$ contains the remaining two stocks. We only have access to a pre-trained model $\mathcal{N}_{\text{old}}$ that has been trained on $\mathbf{T}_{\text{old}}$ but not to its training data $\mathcal{D}_{\text{old}}$. The objective is to build a model or a set of models that work well not

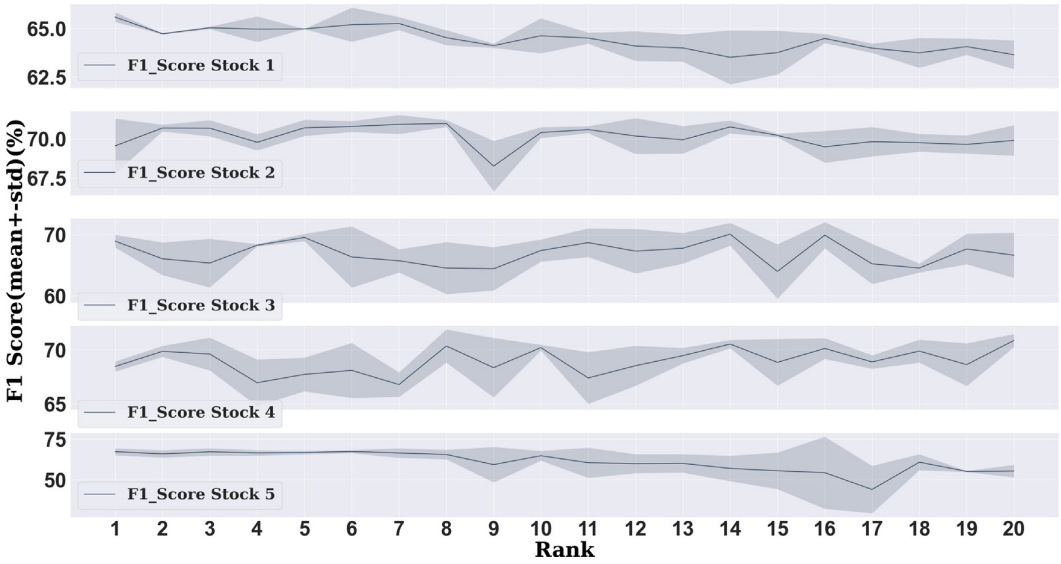

Fig. 6. The best base model network topology for each target stock. Each block corresponds to a layer (BL or TABL) and indicates the the output dimensions of the layer. The output with shape of (3,1) is a column vector with the 3 probability-like outputs of the network corresponding to the three classes. The top topology (`TABL`) refers to the case training is conducted using the training sets of all stocks.

Table 6
Performance of TABL architectures (Mean ± Standard Deviation) measured on the test set in the Experiment Setup 2.

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	#Params
`TABL`	77.54 ± 01.20	69.26 ± 03.30	65.80 ± 01.10	67.33 ± 02.10	17,914
`TABL-fine-tune`	79.17 ± 01.30	72.56 ± 01.10	66.17 ± 03.10	68.77 ± 02.30	53,742
aTABL-IS1	**80.31 ± 03.10**	**73.13 ± 02.40**	**66.26 ± 02.10**	**69.73 ± 03.00**	**26,476**
aTABL-IS2	**80.56 ± 02.30**	**75.80 ± 01.10**	**66.47 ± 03.10**	**70.00 ± 03.20**	**25,636**

only for the stocks in $\mathcal{D}_{\text{new}}$ but also for the stocks in $\mathcal{D}_{\text{old}}$. There are three approaches to tackle this problem:

- We simply keep the pre-trained model `TABL-base` or `CNN-base` and use it for all five stocks. The results related to this approach are denoted with `TABL-base` and `CNN-base`, respectively.
- For each new stock in $\mathcal{D}_{\text{new}}$, we make a copy of the `TABL-base` or `CNN-base` model and fintune it using the data coming from the new stock. The fine-tuned model is used to generate predictions for the new stock. This approach requires the storage of three models: the pre-trained model trained on $\mathcal{D}_{\text{old}}$ and two models fine-tuned on data of the two new stocks $\mathcal{D}_{\text{new},4}$ and $\mathcal{D}_{\text{new},5}$. The results related to this approach are denoted with `TABL-fine-tune` and `CNN-fine-tune`.
- Using the proposed method, in which for each of the new stocks an augmented model is created by adapting the pre-trained model using the corresponding datasets $\mathcal{D}_{\text{new},4}$ and $\mathcal{D}_{\text{new},5}$, respectively. Using our approach, we can store the pre-trained model $\mathcal{N}_{\text{old}}$ and only the auxiliary parameters for augmented models trained on $\mathcal{D}_{\text{new},4}$ and $\mathcal{D}_{\text{new},5}$. The results related to this approach are denoted with `aTABL-IS1` and `aTABL-IS2`, corresponding to two implementations of augmented TABL, and the results for augmented CNNs are denoted with `aCNN`.

With this experimental setup, we compare not only the prediction performance of three different approaches but also the storage cost associated with them. Tables 6 and 7 show the prediction performance as well as the total number of parameters that need to be stored for each case. From Table 6, we can easily observe that the proposed method not only outperforms the finetuning approach in terms of performance but also in storage cost. In practice, when we have N new stocks in our portfolio, the fine-tuning approach would require additional storage of N models. On the other hand, for every new stock, our approach only requires a small fraction of additional storage for the auxiliary parameters. Even though the proposed augmentation has slightly inferior performance compared to the standard finetuning approach using CNNs, we still observe performance gains compared to the baseline CNN in Table 7. Regarding the storage cost, the proposed method always leads to storage savings compared to the finetuning approach.

4.4. Online learning experimental setup

Since the proposed model augmentation approach can also be used in an online learning setting in which new data of the same stock is generated through time, we also conducted experiments simulating this setting. More specifically, data from the first five days were used to train the base models. In order to make predictions for the eighth, ninth, and tenth days, there are three approaches:

- We simply use the base models (`TABL-base` and `CNN-base`).
- We fine-tune the base models using data from the sixth and seventh days before making predictions for the last three days (results denoted with `TABL-fine-tune` and `CNN-fine-tune`).
- use our model augmentation method with the data from the sixth and seventh days to adapt the baseline models (results denoted with `aTABL-IS1`, `aTABL-IS2` and `aCNN`).

Results obtained by following these three approaches are shown in Tables 9 and 10 and indicate that the proposed model augmen-

Table 7
Performance of CNN architectures (Mean ± Standard Deviation) measured on the test set in the Experiment Setup 2.

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	#Params
`CNN`	72.33 ± 0.025	62.83 ± 0.024	62.43 ± 0.007	62.36 ± 0.015	17,091
CNN-fine-tune	**73.89 ± 0.007**	**64.18 ± 0.01**	**63.71 ± 0.003**	**63.89 ± 0.005**	**51,273**
`aCNN`	73.27 ± 0.01	63.62 ± 0.014	62.56 ± 0.005	62.94 ± 0.006	30,427

Table 8
Performance of `Base`, `Augmented`, and `Fine-tune` methods on the $\mathcal{D}_{\text{old}}$ dataset of each target stock.

Target	Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Stock 1	**Base**	80.64	74.52	66.14	69.40
	Augmented	81.94	80.02	63.74	68.81
	Fine-tune	79.08	72.41	62.51	65.95
Stock 2	**Base**	81.00	76.43	66.38	70.12
	Augmented	80.75	76.74	65.40	69.45
	Fine-tune	75.98	66.76	62.23	64.06
Stock 3	**Base**	79.59	76.70	66.50	70.15
	Augmented	75.80	70.14	61.58	64.55
	Fine-tune	74.94	68.87	60.61	63.51
Stock 4	**Base**	80.67	76.03	66.32	69.88
	Augmented	78.09	71.98	61.48	65.10
	Fine-tune	74.21	64.93	57.67	60.08
Stock 5	**Base**	81.21	78.17	66.98	71.03
	Augmented	79.37	73.31	66.95	69.54
	Fine-tune	76.43	68.38	66.52	67.28

Table 9
Performance of CNN architectures (Mean ± Standard Deviation) measured on the test set in the online learning experimental setup.

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Max Rank
`CNN-base`	74.05±0.016	64.56±0.021	63.9±0.008	64.13±0.014	-
`CNN-fine-tune`	74.94±0.005	65.8±0.007	63.79±0.003	64.69±0.002	-
aCNN	**75.24±0.013**	**66.26±0.02**	**63.94±0.003**	**64.93±0.007**	1

Table 10
Performance of TABL architectures (Mean ± Standard Deviation) measured on the test set in the online learning experimental setup.

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Max Rank
`TABL-base`	75.15±0.02	66.61±0.024	65.43±0.003	65.58±0.015	-
`TABL-fine-tune`	76.79±0.001	68.33±0.002	66.12±0.001	67.13±0.001	-
aTABL-IS1	**79.88±0.02**	**76.16±0.045**	**64.93±0.006**	**68.68±0.018**	9
aTABL-IS2	**80.99±0.006**	**78.15±0.024**	**65.61±0.005**	**69.92±0.004**	8

tation method is also effective in adapting a pre-trained model in an online learning manner, yielding better performance compared to the baseline models as well as the fine-tuned models.

Here we should note that, in general, deep learning models may achieve different performances when trained using different random parameter initializations and when tested on data with different characteristics. We can see that the standard deviations of the performance of the models in this experiment, as well as in all other experiments reported above, are small in value. This shows that the adopted models provide reliable performance in terms of the stochastic nature of the Backpropagation algorithm.

4.5. Trading simulation

An efficient trading system requires complex trading decisions and strategies. Forecasting price movements is an important part of a trading system. However, a more important component in a profitable system is the order placing strategy, i.e., when to place an order, at what price and volume to place the order and so on. The design of an efficient order placing strategy is out of the scope of this paper. This paper focuses on improving the prediction performance of mid-price direction movements, which can help develop more accurate trading systems.

To evaluate the profitability of the proposed model, we defined a simple long-only trading system with a naive order placing rule as follows: when the prediction is "up" we will buy one share at best ask price and hold until the predicted label changes to "down" (we do nothing for stationary predictions). At this point, we will sell at the best bid price to take the transaction costs on bid-ask spread into account (the pricing scheme of a particular broker, the trading tier, and the cost of data subscription are not included). To calculate the actual cumulative return, we use the un-normalized ask and bid price values which are not included in the public FI-2010 dataset [61]. We do this because the normalization of the public data at the element-level leads to prices which can have negative values. The plots in the Fig. 7 show the cumulative returns of trades for the test dataset of each stock for the TABL and the best aTABL models in Table 2. The return of each trade is calculated as follows:

$$Returns = (Exit_{Price}/Entry_{Price}) - 1, \tag{25}$$

where the $Exit_{Price}$ is the best bid price and $Entry_{Price}$ is the best ask price of the above trading strategy. The results in Fig. 7 show that trading based on the predictions of TABL and aTABL is profitable. The plots of stocks 4 and 5 show that the predictions of aTABL lead to better cumulative returns. The demonstrated results are based on a naive trading strategy. In reality, there are many other factors that affect the actual profit of a trading system. These include the cost of research and development, the cost of running and maintaining the trading system, the additional transaction cost

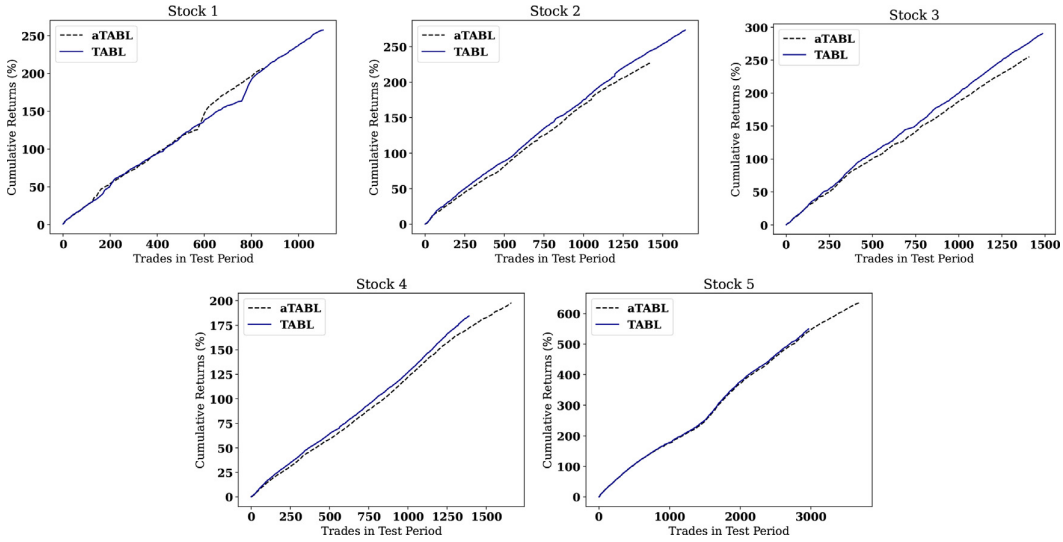

Fig. 7. Cumulative Returns (%) of Trades in Test Period.

on top of the bid-ask spread, market impact of transactions, and the scalability of the trade are very important factors in a live trading system. A practical trading system is designed, developed and evaluated using a complex procedure considering many different information besides the forecasting ability of a model. This trading simulation is designed to show the potential of the proposed method in developing a trading system by relaxing other factors that affect trading.

5. Supplementary experiments

In addition to evaluating our proposed method on the FI-2010 dataset, we also conducted experiments on US stock market to demonstrate the effectiveness of our method on a more volatile market. The US-2015 dataset shares the same characteristics as the FI-2010 dataset, with the difference being that it includes five securities - Apple, Facebook, Google, Intel, and Microsoft - that are traded on the Nasdaq stock market. The dataset was collected over a period of 10 days, from 15 September 2015 to 25 September 2015. As with the FI-2010 dataset, the experimental protocol for the US-2015 dataset was exactly the same. We used the same approach for data preprocessing and the same methodology for training and evaluating our models. Specifically, we formed the input to all models from the 10 most recent limit order events, consisting of 10 instances of 40 dimensions standardized using z-score normalization. The purpose of the supplementary experiments on the US-2015 dataset was to evaluate the generalization performance of our proposed method on a different and more volatile market. By using a different dataset, we aimed to demonstrate the robustness of our method in analyzing stock market data from different markets.

Table 11 displays the class distributions of the train and test sets for the US-2015 dataset. It is noteworthy that compared to the FI-2010 dataset, which is highly imbalanced, the US-2015 dataset exhibits a significantly more balanced distribution among the three classes of up, down, and stationery. This feature of the US-2015 dataset suggests that it pertains to a more volatile market, as compared to the relatively stable market represented by the FI-2010 dataset. The balanced class distribution of the US-2015 dataset is expected to pose a different and potentially more challenging task for the models to classify the mid-price direction accurately. Therefore, evaluating the proposed method on the US-2015 dataset allows us to assess the robustness and generalization capability of the models, and provides insights into their performance under more diverse and challenging market conditions.

In accordance with the procedures described in Sections 4.2, 4.3, and 4.4, we utilized the prediction horizon of 10 events and maintained consistency in the splitting of the training and testing sets across all experimental setups. The results of the supplementary experiments are presented in the following section.

5.1. Experimental setup 1

The experiments in this section replicate the experimental setup of Section 4.2, using the same procedure. Results for both TABL and CNN architectures are presented in Tables 12 and 13. According to the results presented in Table 12 for the TABL architectures, the proposed method exhibits better performance compared to both `TABL-fine-tune` and `TABL-target` models for all five target stocks. On average, the proposed method outperforms all other approaches. However, for target stock 4 the `TABL` model shows slightly better performance. This is not surprising, as training on both $\mathcal{D}_{\text{old}}$ and $\mathcal{D}_{\text{new}}$ datasets from scratch increases the likelihood of achieving better performance. Table 13 displays the performance of different approaches on each of the five target datasets (stocks), and the average performance of all approaches for the CNN architecture. The results indicate that, on average, `aCNN` outperforms the other existing approaches. However, for target stocks 4 and 5, the proposed method did not perform better than other approaches, which is similar to the observation made for stock 4 in Table 12.

The effectiveness of the proposed method can also be demonstrated using a confusion matrix. Figure 8 shows the confusion matrices of the best models presented in Table 12 and 13 for each stock. The results indicate that the predictions of the “up” and “down” classes are mostly accurate, with misclassifications mainly labeled as “stationary”. For the “stationary” class, the confusion matrices of stocks 3, 4, and 5 reveal mostly correct predictions. However, for stocks 1 and 2, the best models struggle to distinguish between the “stationary” class and the other two classes. Although this could be considered a limitation, it is worth noting that the misclassifications may not be a major problem when the true class is “stationary”.

Table 11
Distribution of classes in train and test dataset in US-2015. Class 0 refers to the stock price remains the same, class 1 to the situation where the price goes up, and class 2 where the price goes downs.

	Train dataset			Test dataset		
Target stock	Class 0	Class 1	Class 2	Class 0	Class 1	Class 2
Stock 1	32,322	51,054	50,825	13,151	22,474	21,890
Stock 2	32,425	52,192	49,585	11,206	23,388	22,922
Stock 3	57,471	38,363	38,368	24,907	16,324	16,288
Stock 4	62,088	36,540	35,574	23,267	16,738	17,511
Stock 5	47,892	42,891	43,419	17,190	19,756	20,570
All Stocks	232,198 (34.6%)	221,040(32.9%)	217,771(32.5%)	89,721(31.2%)	98,680(34.3%)	99,181(34.5%)

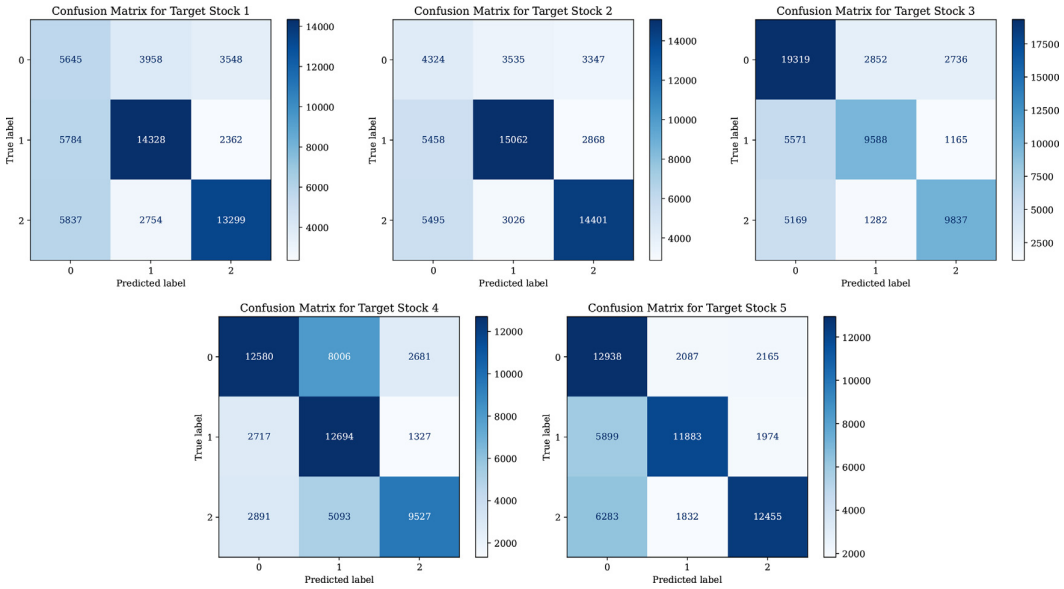

Fig. 8. Confusion matrices for the best model on the test set of each stock among the models in Tables 12 and 13. Class 0 refers to the stock price remains the same, class 1 to the situation where the price goes up, and class 2 is where the price goes downs.

5.2. Experimental setup 2

The aim of this experiment is to assess the performance of our proposed method on stocks of the more volatile US market in the US-2015 dataset. To conduct this evaluation, we use the real-world experimental setup described in Section 4.3. By doing so, we demonstrate the efficacy of our approach and its potential for real-world application. Results of the evaluation for both TABL and CNN architectures are presented in Tables 14 and 15. The results demonstrate that our proposed method for both TABL and CNN architectures outperforms the two other possible approaches. Specifically, the `aTABL-IS1` approach achieves better performance with fewer parameters, resulting in faster training times. For the CNN approach, the `aCNN` model produces the best prediction performance, although it requires more parameters than the other approaches due to its higher rank value. This indicates that `aCNN` is more accurate but it is not as fast as the other methods.

5.3. Online learning experimental setup

In this experiment, we investigate a critical aspect of the effectiveness of our proposed method, which is its performance when applied to stock data that is generated over time. The results of this experiment will provide valuable insights into the suitability of our approach for real-world applications. The experimental setup for online learning, which was described in Section 4.4, was replicated for this experimental setup. The outcomes of these experiments are presented in Tables 16 and 17. Table 16 displays the experimental outcomes of TABL architectures, demonstrating that `aTABL-IS1` and `aTABL-IS2` are highly effective and efficient in addressing the online learning problem. the proposed approach outperforms other existing methods while using fewer training parameters. In Table 17, the results of CNN architectures are presented. The `aCNN` method exhibits superior performance compared to the `CNN-fine-tune`, while also achieving nearly the same performance as the base model with a smaller number of training parameters.

5.4. Cross-market experimental setup

In addition to the experiments conducted in Sections 4, 5.1, 5.2, and 5.3 that demonstrate the efficacy of our proposed method in tackling the research problem described in Section 1, we also assess the performance of our method on an experimental setup where the base and target datasets have significantly different characteristics but are associated with the same prediction task. The use of both US-2015 and FI-2010 datasets in this experiment is suitable since they depict distinct stocks with unique characteristics that are aligned with the research objectives. The FI-2010 and US-2015 datasets were used interchangeably as the $\mathcal{D}_{\text{old}}$ and $\mathcal{D}_{\text{new}}$ to assess the impact of learned information from each dataset on the prediction of the other. To maintain consistency throughout the experiments presented in this paper, pre-trained models $\mathcal{N}_{\text{old}}$ were trained on the $\mathcal{D}_{\text{old}}$, and the TL (`TABL-fine-tune`, `CNN-fine-tune`) and augmented models (`aTABL-IS1`, `aTABL-IS2`, `aCNN`) for both TABL and

Table 12
Performance of TABL networks (Mean ± Standard Deviation) on the test sets of the Experimental Setup 1 (US-2015 dataset).

Target	Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Max Rank
Stock 1	TABL	58.29±0.09	53.61±0.07	53.87±0.07	53.49±0.07	
	TABL-base	58.39±0.09	53.9±0.07	54.13±0.07	53.82±0.07	1
	TABL-fine-tune	55.06±0.09	51.62±0.06	51.6±0.07	51.59±0.06	
	TABL-target	55.26±0.01	51.36±0.07	51.5±0.07	51.38±0.07	
	aTABL-IS1	**58.29±0.03**	**54.97±0.02**	**54.91±0.02**	**54.92±0.02**	**6**
	aTABL-IS2	**57.42±0.04**	**56.39±0.02**	**55.46±0.02**	**55.54±0.03**	**3**
Stock 2	TABL	58.63±0.09	53.82±0.07	53.77±0.07	53.78±0.07	
	TABL-base	56.84±0.08	54.23±0.07	53.63±0.07	53.56±0.07	
	TABL-fine-tune	55.05±0.09	52.23±0.08	51.63±0.08	51.64±0.08	
	TABL-target	55.66±0.07	52.61±0.06	51.96±0.05	51.96±0.05	
	aTABL-IS1	**58.79±0.03**	**55.19±0.03**	**54.73±0.03**	**54.76±0.03**	**3**
	aTABL-IS2	**58.23±0.04**	**55.53±0.03**	**54.85±0.03**	**54.81±0.03**	**6**
Stock 3	TABL	64.46±0.09	65.39±0.12	63.13±0.09	63.8±0.09	
	TABL-base	47.38±0.05	52.28±0.01	52.57±0.07	45.95±0.04	
	TABL-fine-tune	58.16±0.08	58.25±0.01	57.58±0.07	57.7±0.08	
	TABL-target	58.95±0.09	59.22±0.09	59.8±0.01	58.92±0.09	
	aTABL-IS1	**65.89±0.03**	**66.31±0.04**	**64.8±0.04**	**65.38±0.04**	**1**
	aTABL-IS2	**66.8±0.04**	**67.87±0.05**	**65.07±0.03**	**66.0±0.03**	**19**
Stock 4	TABL	**70.15±0.12**	**72.24±0.18**	**68.31±0.12**	**69.15±0.13**	
	TABL-base	68.4±0.11	70.29±0.17	66.69±0.09	67.5±0.01	
	TABL-fine-tune	58.34±0.09	59.07±0.09	59.39±0.09	58.37±0.09	
	TABL-target	57.9±0.01	59.09±0.09	59.28±0.09	57.92±0.01	
	aTABL-IS1	68.73±0.04	68.92±0.05	68.18±0.04	68.46±0.04	2
	aTABL-IS2	67.84±0.04	68.16±0.05	67.31±0.04	67.64±0.04	4
Stock 5	TABL	63.26±0.11	64.51±0.11	63.53±0.11	63.36±0.11	
	TABL-base	62.9±0.11	63.88±0.12	63.09±0.11	62.96±0.11	
	TABL-fine-tune	62.27±0.01	62.56±0.11	62.24±0.01	62.2±0.01	
	TABL-target	59.24±0.08	58.66±0.08	58.6±00.08	58.51±0.08	
	aTABL-IS1	**64.38±0.04**	**66.75±0.04**	**64.88±0.04**	**64.57±0.04**	**15**
	aTABL-IS2	**64.23±0.03**	**65.54±0.03**	**64.5±0.03**	**64.34±0.03**	**13**
Average	TABL	62.96±4.86	61.92±8.06	60.53±6.44	60.72±6.85	
	TABL-base	58.79±7.8	58.92±7.82	58.03±6.42	56.76±8.5	
	TABL-fine-tune	57.78±2.97	56.75±4.69	56.49±4.74	56.3±4.6	
	TABL-target	57.41±1.84	56.19±3.86	56.23±4.13	55.74±3.73	
	aTABL-IS1	**63.22±4.54**	**62.43±6.78**	**61.5±6.24**	**61.62±6.35**	
	aTABL-IS2	**62.91±4.82**	**62.7±6.24**	**61.44±5.83**	**61.67±6.04**	

Table 13
Performance of CNNs (Mean ± Standard Deviation) on the test sets of the Experimental Setup 1 (US-2015 dataset).

Target	Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Max Rank
Stock 1	CNN	58.73±0.03	52.74±0.05	52.39±0.02	50.0±0.02	
	CNN-base	57.0±0.1	51.43±0.09	51.56±0.07	50.26±0.06	
	CNN-fine-tune	55.49±0.09	48.95±0.09	49.54±0.07	47.45±0.04	
	CNN-target	55.38±0.03	53.29±0.02	52.9±0.02	52.95±0.02	
	aCNN	**56.4±0.03**	**53.08±0.02**	**52.99±0.02**	**52.97±0.02**	**2**
Stock 2	CNN	60.14±0.03	53.2±0.03	53.27±0.02	52.81±0.02	
	CNN-base	56.54±0.1	51.95±0.07	51.87±0.08	51.89±0.08	
	CNN-fine-tune	57.97±0.11	50.72±0.08	50.79±0.08	50.01±0.06	
	CNN-target	53.25±0.03	46.11±0.03	46.15±0.02	45.03±0.01	
	aCNN	**58.31±0.03**	**53.3±0.02**	**53.25±0.02**	**53.24±0.02**	**2**
Stock 3	CNN	58.31±0.02	57.85±0.02	58.24±0.02	57.97±0.02	
	CNN-base	43.58±0.05	48.75±0.11	47.94±0.07	42.48±0.04	
	CNN-fine-tune	57.57±0.09	57.69±0.13	55.43±0.07	56.05±0.08	
	CNN-target	59.46±0.03	59.24±0.04	58.17±0.02	58.48±0.03	
	aCNN	**61.55±0.04**	**61.11±0.04**	**60.9±0.03**	**61.0±0.04**	**3**
Stock 4	**CNN**	**69.56±0.04**	**71.18±0.06**	**67.99±0.04**	**68.78±0.04**	
	CNN-base	66.68±0.11	68.13±0.15	64.91±0.11	65.61±0.12	
	CNN-fine-tune	63.3±0.12	63.26±0.14	63.11±0.11	63.13±0.12	
	CNN-target	56.45±0.04	58.55±0.05	58.08±0.04	56.54±0.04	
	aCNN	62.03±0.04	62.26±0.04	62.8±0.04	62.17±0.04	1
Stock 5	**CNN**	**63.8±0.03**	**64.05±0.04**	**63.73±0.03**	**63.71±0.03**	
	CNN-base	61.48±0.13	61.65±0.13	61.37±0.13	61.36±0.13	
	CNN-fine-tune	61.45±0.09	61.01±0.1	61.01±0.1	61.0±0.1	
	CNN-target	60.79±0.03	60.94±0.03	60.74±0.03	60.65±0.03	
	aCNN	62.48±0.03	62.07±0.03	62.07±0.03	62.07±0.03	2
Average	**CNN**	**62.11±4.7**	**59.81±7.83**	**59.13±6.72**	**58.66±7.71**	
	CNN-base	57.06±8.58	56.39±8.19	55.53±7.23	54.32±9.22	
	CNN-fine-tune	59.16±3.16	56.33±6.28	55.98±6.02	55.53±6.78	
	CNN-target	57.07±3.06	55.63±6.04	55.21±5.81	54.73±6.12	
	aCNN	60.16±2.67	58.37±4.75	58.41±4.88	58.29±4.76	

Table 14
Performance of TABL architectures (Mean ± Standard Deviation) measured on the test set in the Experiment Setup 2 (US-2015 dataset).

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	#Params
TABL	58.47±0.11	59.04±0.12	58.35±0.11	58.45±0.11	16,784
TABL-Fine-tune	59.56±0.08	59.62±0.08	59.43±0.08	59.47±0.08	50,352
aTABL-IS1	**59.98±0.1**	**60.54±0.1**	**59.93±0.1**	**60.01±0.1**	**21,828**
aTABL-IS2	**60.66±0.1**	**61.54±0.11**	**60.69±0.1**	**60.74±0.1**	**52,796**

Table 15
Performance of CNN architectures (Mean ± Standard Deviation) measured on the test set in the Experiment Setup 2 (US-2015 dataset).

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	#Params
CNN	53.84±0.09	53.7±0.09	53.68±0.09	53.7±0.9	8,947
CNN-Fine-tune	55.19±0.08	54.98±0.08	54.68±0.08	54.29±0.07	26,841
aCNN	**56.36±0.09**	**56.09±0.09**	**55.93±0.09**	**55.7±0.08**	**45,817**

Table 16
Performance of TABL architectures (Mean ± Standard Deviation) measured on the test set in the online learning experimental setup (US-2015 dataset).

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Max Rank
TABL	60.97±0.09	60.85±0.09	60.75±0.09	60.75±0.09	
TABL-Fine-tune	58.02±0.09	59.35±0.1	58.14±0.09	58.19±0.09	
aTABL-IS1	**62.73±0.08**	**63.54±0.08**	**62.81±0.08**	**62.82±0.08**	**2**
aTABL-IS2	**62.62±0.08**	**63.36±0.09**	**62.68±0.08**	**62.7±0.08**	**4**

Table 17
Performance of CNN architectures (Mean ± Standard Deviation) measured on the test set in the online learning experimental setup (US-2015 dataset).

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Max Rank
CNN	**59.58±0.09**	**59.24±0.09**	**59.26±0.09**	**59.16±0.09**	
CNN-Fine-tune	56.74±0.08	56.35±0.08	56.3±0.08	56.04±0.08	
aCNN	58.93±0.11	58.59±0.1	58.61±0.1	58.52±0.1	1

Table 18
Performance of TABL architectures (Mean ± Standard Deviation) measured on Cross-market experiments.

Target	Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Max Rank
FI-2010	TABL	41.66±0.07	44.85±0.05	48.34±0.07	39.53±0.06	
	TABL-target	78.56±0.15	72.1±0.3	65.22±0.09	67.87±0.16	
	TABL-Fine-tune	73.24±0.12	64.05±0.19	63.06±0.09	63.01±0.12	
	aTABL-IS1	**79.42±0.22**	**74.99±0.51**	**64.3±0.12**	**68.05±0.23**	**7**
	aTABL-IS2	**79.82±0.05**	**75.38±0.12**	**65.02±0.04**	**68.79±0.06**	**5**
US-2015	TABL	38.92±0.01	51.46±0.11	40.5±0.01	33.22±0.03	
	TABL-target	62.83±0.1	63.32±0.1	62.87±0.1	62.86±0.1	
	TABL-Fine-tune	60.98±0.11	60.77±0.11	60.76±0.11	60.75±0.11	
	aTABL-IS1	62.45±0.11	63.0±0.11	62.47±0.11	62.49±0.11	2
	aTABL-IS2	**64.65±0**	**65.23±0.06**	**64.68±0**	**64.71±0.06**	**9**

Table 19
Performance of CNN architectures (Mean ± Standard Deviation) measured on Cross-market experiments.

Target	Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Max Rank
FI-2010	CNN	33.98±0.04	45.44±0.05	46.97±0.05	33.9±0.04	
	CNN-target	72.9±0.16	62.84±0.22	62.28±0.1	62.47±0.16	
	CNN-fine-tune	67.53±0.14	56.83±0.14	59.34±0.09	57.67±0.12	
	aCNN	**75.92±0.12**	**66.58±0.17**	**64.11±0.09**	**65.18±0.12**	**3**
US-2015	CNN	41.8±0.01	49.77±0.04	43.09±0.01	38.53±0.01	
	CNN-target	62.06±0.03	61.8±0.03	61.8±0.03	61.77±0.03	
	CNN-fine-tune	59.74±0.1	59.76±0.1	59.58±0.1	59.54±0.1	
	aCNN	**62.0±0.09**	**61.91±0.08**	**61.85±0.08**	**61.86±0.08**	**2**

CNN architectures were trained using the $\mathcal{D}_{\text{new}}$ by leveraging the $\mathcal{N}_{\text{old}}$. Tables 18 and 19 present the results of experiments conducted using TABL and CNN architectures. The results in each table are divided into two parts based on the target dataset, $\mathcal{D}_{\text{new}}$. The results in each part shows the performance of four different approaches: the base models referred to as TABL and CNN methods, the TABL-target and CNN-target methods that are referred to as those which were trained from scratch on the target dataset, the finetuning approach methods known as TABL-fine-tune and CNN-fine-tune, and the augmented methods named aTABL-IS1, aTABL-IS2, and aCNN. The results indicate that TABL and CNN models failed to accurately predict the

data of $\mathcal{T}_{\text{new}}$. Despite the close competition between models that were trained from scratch on the target dataset (`TABL-target`, `CNN-target`) and those using augmented methods for both TABL and CNN architectures, `aTABL-IS1`, `aTABL-IS2`, and `aCNN` consistently outperform the former. This indicates that the knowledge gained from pre-trained models of datasets with varying statistical characteristics and dynamics can be beneficial for training models on datasets of the same domain.

6. Conclusion

In this paper, we studied a new research problem defined on financial time-series data, that of efficiently training new models for stock mid-price time-series classification by exploiting knowledge in existing deep learning models trained on time-series data of different stocks or time-series data of past periods. We proposed a new method that exploits model augmentation and low-rank matrix approximation to improve the prediction performance and reduce the storage cost. Our model augmentation approach takes advantage of the learned information from the knowledge encoded in the parameters of an existing (pre-trained) model and learns auxiliary connections that are added to the pre-trained model to adapt it for the new task. The low rank approximation of auxiliary parameters regularizes the learning process and reduces the storage cost of new models. Extensive experiments on stock mid-price direction prediction tasks demonstrated that the proposed method can lead to performance improvements as well as a reduction in storage requirements during deployment. Interesting future research directions include investigating the effectiveness of the proposed approach in time-series classification problems coming from different applications, as well as the study of designing deep learning models targeting applications involving other forms of input data, like images, videos, audio, under the restrictions indicated by the adopted problem formulation.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

The research received funding from the Independent Research Fund Denmark project DISPA (Project Number: 9041-00004).

Appendix A

Let us denote by N the number of samples in a mini-batch during stochastic optimization. Below, we provide the computational complexity estimate for two implementation strategies described in Section 3.

Computational Complexity of Implementation Strategy 1

- Computing Eq. (16) requires $NDD'T$ operations.
- Computing Eq. (17) requires $ND'TT$ operations.
- Computing Eq. (18) requires $ND'TT' + 2ND'T'$ operations.

Computational Complexity of Implementation Strategy 2

- Computing Eq. (19) requires $NDD'T + NDKT + NKTD'$ operations.
- Computing Eq. (20) requires $ND'TT + ND'TK + ND'KT$ operations.
- Computing Eq. (21) requires $ND'TT' + ND'TK + ND'KT' + 2ND'T'$ operations.

Appendix B

To find the best topology for $\mathcal{N}_{\text{old}}$ for each target stock we ran experiments for different topologies and hyperparameter values. Tables B.1-B.5 show the results of the topology ablation study for $\mathcal{N}_{\text{old}}$ of each target stock.

Table B.1
Ablation study of $\mathcal{N}_{\text{old}}$ for Stock 1.

Model	F1-score
layers_[[60, 5], [200, 10]]	0.694022
layers_[[60, 10], [120, 5]]	0.687618
layers_[[60, 5], [120, 10], [50, 5]]	0.686768
layers_[[60, 10], [200, 10]]	0.68425
layers_[[50, 10], [250, 5]]	0.68172
layers_[[60, 10], [200, 5]]	0.67418

Table B.2
Ablation study of $\mathcal{N}_{\text{old}}$ for Stock 2.

Model	F1-score
layers_[[60, 10], [120, 5]]	0.701169
layers_[[60, 10], [200, 5]]	0.693212
layers_[[60, 5], [120, 10], [50, 5]]	0.692069
layers_[[60, 5], [200, 10]]	0.685344
layers_[[60, 10], [200, 10]]	0.68498
layers_[[50, 10], [250, 5]]	0.683752

Table B.3
Ablation study of $\mathcal{N}_{\text{old}}$ for Stock 3.

Model	F1-score
layers_[[60, 5], [120, 10], [50, 5]]	0.701476
layers_[[60, 10], [120, 5]]	0.699135
layers_[[60, 5], [200, 10]]	0.698294
layers_[[60, 10], [200, 5]]	0.698241
layers_[[50, 10], [250, 5]]	0.698197
layers_[[60, 10], [200, 10]]	0.689825

Table B.4
Ablation study of $\mathcal{N}_{\text{old}}$ for Stock 4 .

Model	F1-score
layers_[[60, 10], [200, 10]]	0.698838
layers_[[60, 5], [120, 10], [50, 5]]	0.69745
layers_[[60, 10], [120, 5]]	0.697304
layers_[[60, 5], [200, 10]]	0.691858
layers_[[50, 10], [250, 5]]	0.687543
layers_[[60, 10], [200, 5]]	0.685277

Table B.5
Ablation study of $\mathcal{N}_{\text{old}}$ for Stock 5.

Model	F1-score
layers_[[60, 10], [200, 10]]	0.710281
layers_[[60, 5], [120, 10], [50, 5]]	0.709831
layers_[[60, 10], [120, 5]]	0.706329
layers_[[50, 10], [250, 5]]	0.706135
layers_[[60, 10], [200, 5]]	0.70475
layers_[[60, 5], [200, 10]]	0.70228

References

[1] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, A. Iosifidis, Forecasting stock prices from the limit order book using convolutional neural networks, in: IEEE Conference on Business Informatics, Vol. 1, 2017, pp. 7–12.

[2] Z. Zhang, S. Zohren, S. Roberts, DeepLOB: deep convolutional neural networks for limit order books, IEEE Trans. Signal Process. 67 (2019) 3001–3012.

[3] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, A. Iosifidis, Temporal logistic neural bag-of-features for financial time series forecasting leveraging limit order book data, Pattern Recognit. Lett. 136 (2020) 183–189.

[4] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, A. Iosifidis, Deep adaptive input normalization for time series forecasting, IEEE Trans. Neural Netw. Learn. Syst. 31 (9) (2020) 3760–3765.

[5] J.A. Sirignano, Deep learning for limit order books, Quant. Finance 19 (4) (2019) 549–570.

[6] M.F. Dixon, Sequence classification of the limit order book using recurrent neural networks, SSRN Electron. J. (2017).

[7] D.T. Tran, J. Kanniainen, M. Gabbouj, A. Iosifidis, Data-driven neural architecture learning for financial time-series forecasting, arXiv preprint arXiv:1903.06751(2019).

[8] D. Cao, Y. El-Laham, L. Trinh, S. Vyetrenko, Y. Liu, DSLOB: a synthetic limit order book dataset for benchmarking forecasting algorithms under distributional shift, CoRR (2022) abs/2211.11513, doi:10.48550/arXiv.2211.11513.

[9] A.-A. Semenoglou, E. Spiliotis, V. Assimakopoulos, Data augmentation for univariate time series forecasting with neural networks, Pattern Recognit. 134 (2023) 109132.

[10] O.B. Sezer, M.U. Gudelek, A.M. Ozbayoglu, Financial time series forecasting with deep learning: a systematic literature review: 2005–2019, Appl. Soft Comput. 90 (2020) 106181.

[11] S. Kiranyaz, T. Ince, A. Iosifidis, M. Gabbouj, Progressive operational perceptrons, Neurocomputing 224 (2017) 142–154.

[12] D.T. Tran, A. Iosifidis, Learning to rank: a progressive neural network learning approach, in: IEEE International Conference on Acoustics, Speech and Signal Processing, 2019, pp. 8355–8359.

[13] D.T. Tran, S. Kiranyaz, M. Gabbouj, A. Iosifidis, Progressive operational perceptrons with memory, Neurocomputing 379 (2020) 172–181.

[14] S. Kiranyaz, T. Ince, A. Iosifidis, M. Gabbouj, Operational neural networks, Neural Comput. Appl. (2020) 1–24.

[15] M. Jaderberg, A. Vedaldi, A. Zisserman, Speeding up convolutional neural networks with low rank expansions, arXiv preprint arXiv:1405.3866(2014).

[16] D.T. Tran, A. Iosifidis, M. Gabbouj, Improving efficiency in convolutional neural networks with multilinear filters, Neural Netw. 105 (2018) 328–339.

[17] H. Huang, H. Yu, LTNN: a layerwise tensorized compression of multilayer neural network, IEEE Trans. Neural Netw. Learn. Syst. 30 (5) (2018) 1497–1511.

[18] X. Ruan, Y. Liu, C. Yuan, B. Li, W. Hu, Y. Li, S. Maybank, EDP: an efficient decomposition and pruning scheme for convolutional neural network compression, IEEE Trans. Neural Netw. Learn. Syst. 32 (10) (2020) 4499–4513.

[19] L. Shao, F. Zhu, X. Li, Transfer learning for visual categorization: asurvey, IEEE Trans. Neural Netw. Learn. Syst. 26 (5) (2014) 1019–1034.

[20] Z. Ding, Y. Fu, Deep transfer low-rank coding for cross-domain learning, IEEE Trans. Neural Netw. Learn. Syst. 30 (6) (2018) 1768–1779.

[21] E. Fons, P. Dawson, X.-j. Zeng, J. Keane, A. Iosifidis, Augmenting transferred representations for stock classification, arXiv preprint arXiv:2011.04545(2020).

[22] L. Duan, D. Xu, I.W.-H. Tsang, Domain adaptation from multiple sources: adomain-dependent regularization approach, IEEE Trans. Neural Netw. Learn. Syst. 23 (3) (2012) 504–518.

[23] Z. Wang, B. Du, Y. Guo, Domain adaptation with neural embedding matching, IEEE Trans. Neural Netw. Learn. Syst. 31 (7) (2019) 2387–2397.

[24] L. Hedegaard, O. Sheikh-Omar, A. Iosifidis, Supervised domain adaptation; a graph embedding perspective and a rectified experimental protocol, IEEE Trans. Image Process. 30 (2021) 8619–8631.

[25] S. Wu, A. Wu, W.-S. Zheng, Online deep transferable dictionary learning, Pattern Recognit. 118 (2021) 108007.

[26] L. Yu, S. Wang, K.K. Lai, An online learning algorithm with adaptive forgetting factors for feedforward neural networks in financial time series forecasting, Nonlinear Dyn. Syst. Theory 7 (1) (2007) 51–66.

[27] R.C. Cavalcante, A.L. Oliveira, An approach to handle concept drift in financial time series based on extreme learning machines and explicit drift detection, in: International Joint Conference on Neural Networks, 2015, pp. 1–8.

[28] X. Wang, M. Han, Online sequential extreme learning machine with kernels for nonstationary time series prediction, Neurocomputing 145 (2014) 90–97.

[29] D.T. Tran, A. Iosifidis, J. Kanniainen, M. Gabbouj, Temporal attention-augmented bilinear network for financial time-series data analysis, IEEE Trans. Neural Netw. Learn. Syst. 30 (5) (2019) 1407–1418.

[30] K. Kanjamapornkul, R. Pinčák, S. Chunithipaisan, E. Bartoš, Support spinor machine, Digit. Signal Process. 70 (2017) 59–72.

[31] K. Kanjamapornkul, R. Pinčák, E. Bartoš, The study of thai stock market across the 2008 financial crisis, Physica A 462 (2016) 117–133.

[32] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, A. Iosifidis, Using deep learning for price prediction by exploiting stationary limit order book features, Appl. Soft Comput. 93 (2020) 106401.

[33] D.T. Tran, J. Kanniainen, M. Gabbouj, A. Iosifidis, Bilinear input normalization for neural networks in financial forecasting, arXiv preprint arXiv:2109.00983 (2021).

[34] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans Knowl Data Eng 22 (10) (2009) 1345–1359.

[35] M.T. Rosenstein, Z. Marx, L.P. Kaelbling, T.G. Dietterich, To transfer or not to transfer, in: NIPS 2005 Workshop on Transfer Learning, Vol. 898, 2005, pp. 1–4.

[36] H.I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.-A. Muller, Transfer learning for time series classification, in: IEEE International Conference on Big Data, 2018, pp. 1367–1376.

[37] D.J. Berndt, J. Clifford, Using dynamic time warping to find patterns in time series, in: KDD Workshop, Vol. 10, 1994, pp. 359–370.

[38] M. Pratama, M. de Carvalho, R. Xie, E. Lughofer, J. Lu, ATL: autonomous knowledge transfer from many streaming processes, in: ACM International Conference on Information and Knowledge Management, 2019, pp. 269–278.

[39] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Comput. Surv. 46 (4) (2014) 1–37.

[40] R. Ye, Q. Dai, A novel transfer learning framework for time series forecasting, Knowl. Based Syst. 156 (2018) 74–99.

[41] H. Zuo, G. Zhang, W. Pedrycz, V. Behbood, J. Lu, Fuzzy regression transfer learning in Takagi–Sugeno fuzzy models, IEEE Trans. Fuzzy Syst. 25 (6) (2016) 1795–1807.

[42] T.-T. Nguyen, S. Yoon, A novel approach to short-term stock price movement prediction using transfer learning, Appl. Sci. 9 (22) (2019) 4745.

[43] A. Koshiyama, S. Flennerhag, S.B. Blumberg, N. Firoozye, P. Treleaven, QuantNet: transferring learning across systematic trading strategies, arXiv preprint arXiv:2004.03445(2020).

[44] P. Zhao, S.C. Hoi, J. Wang, B. Li, Online transfer learning, Artif. Intell. 216 (2014) 76–102.

[45] L. Ge, J. Gao, A. Zhang, OMS-TL: a framework of online multiple source transfer learning, in: ACM International Conference on Information & Knowledge Management, 2013, pp. 2423–2428.

[46] D.A. Ross, J. Lim, R.-S. Lin, M.-H. Yang, Incremental learning for robust visual tracking, Int. J. Comput. Vis. 77 (1) (2008) 125–141.

[47] M. Pratama, C. Za'in, A. Ashfahani, Y.S. Ong, W. Ding, Automatic construction of multi-layer perceptron network from streaming examples, in: ACM International Conference on Information and Knowledge Management, 2019, pp. 1171–1180.

[48] M. Wang, W. Deng, Deep visual domain adaptation: a survey, Neurocomputing 312 (2018) 135–153.

[49] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, V. Lempitsky, Domain-adversarial training of neural networks, J. Mach. Learn. Res. 17 (1) (2016). 2096–2030

[50] M. Long, Y. Cao, J. Wang, M. Jordan, Learning transferable features with deep adaptation networks, in: International Conference on Machine Learning, 2015, pp. 97–105.

[51] A. Gretton, K.M. Borgwardt, M.J. Rasch, B. Schölkopf, A. Smola, A kernel two-sample test, J. Mach. Learn. Res. 13 (1) (2012) 723–773.

[52] M. Pratama, M. de Carvalho, R. Xie, E. Lughofer, J. Lu, ATL: autonomous knowledge transfer from many streaming processes, in: ACM International Conference on Information and Knowledge Management, 2019, pp. 269–278.

[53] X. Renchunzi, M. Pratama, Automatic online multi-source domain adaptation, Inf. Sci. (Ny) 582 (2022) 480–494.

[54] A. Bulat, J. Kossaifi, G. Tzimiropoulos, M. Pantic, Incremental multi-domain learning with network latent tensor factorization, in: AAAI, 2020, pp. 10470–10477.

[55] R. Caruana, Multitask learning, Mach. Learn. 28 (1) (1997) 41–75.

[56] A. Senhaji, J. Raitoharju, M. Gabbouj, A. Iosifidis, Not all domains are equally complex: adaptive multi-domain learning, in: International Conference on Pattern Recognition, 2020.

[57] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[58] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4700–4708.

[59] T.G. Kolda, B.W. Bader, Tensor decompositions and applications, SIAM Rev. 51 (3) (2009) 455–500.

[60] R. Cont, Statistical modeling of high-frequency financial data, IEEE Signal Process Mag. 28 (2011) 16–25.

[61] A. Ntakaris, M. Magris, J. Kanniainen, M. Gabbouj, A. Iosifidis, Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods, J. Forecast. 37 (2018) 852–866.

[62] D.T. Tran, J. Kanniainen, A. Iosifidis, How informative is the order book beyond the best levels? Machine learning perspective, NeurIPS 2021 Workshop on Machine Learning meets Econometrics, 2021.

Mostafa Shabani received his master's in information systems and currently is a PhD candidate in deep learning for financial data analysis at Aarhus University, Denmark. His research interests include machine learning for time series data analysis, especially in computational finance.

Dat Thanh Tran received the BASc degree in Automation Engineering from Hameen University of Applied Sciences, and MSc degree in Data Engineering and Machine Learning from Tampere University in 2017 and 2019 respectively. He has been a Research Assistant in Multimedia Research Group led by Professor Moncef Gabbouj at Tampere University since 2015. His current research interests include statistical pattern recognition and machine learning, especially machine learning models with efficient computation and novel neural architectures.

Dr Juho Kanniainen is Full Professor at Tampere University, Finland, where he is heading a research group Financial Computing and Data Analytics with a focus on financial data science. His papers on financial market research have been published in top-tier journals, including IEEE TNNLS and Review of Finance.

Alexandros Iosifidis is a Professor at Aarhus University, Denmark. He leads the Machine Learning and Computational Intelligence group at the Department of Electrical and Computer Engineering, and the Machine Intelligence research area at the University's Centre for Digitalisation, Big Data, and Data Analytics (DIGIT). He is a Senior Member of IEEE and a member of the IEEE Technical Committee on Machine Learning for Signal Processing. He is the Associate Editor-in-Chief of the *Neurocomputing* journal, covering the research area of neural networks, and an Associate Editor of IEEE Transactions on Neural Networks and Learning Systems and IEEE Transactions on Artificial Intelligence.