Tensorized LSTM with Adaptive Shared Memory for Learning Trends in Multivariate Time Series

Dongkuan Xu,[1]* Wei Cheng,[2]* Bo Zong,[2] Dongjing Song,[2] Jingchao Ni,[2]
Wenchao Yu,[2] Yanchi Liu,[2] Haifeng Chen,[2] Xiang Zhang[1]
[1]The Pennsylvania State University, {dux19, xzz89}@psu.edu
[2]NEC Laboratories America, Inc., {weicheng, bzong, dsong, jni, wyu, yanchi, haifeng}@nec-labs.com

Abstract

The problem of learning and forecasting underlying trends in time series data arises in a variety of applications, such as traffic management, energy optimization, etc. In literature, a trend in time series is characterized by the slope and duration, and its prediction is then to forecast the two values of the subsequent trend given historical data of the time series. For this problem, existing approaches mainly deal with the case in univariate time series. However, in many real-world applications, there are multiple variables at play, and handling all of them at the same time is crucial for an accurate prediction. A natural way is to employ multi-task learning (MTL) techniques in which the trend learning of each time series is treated as a task. The key point of MTL is to learn task relatedness to achieve better parameter sharing, which however is challenging in trend prediction task. First, effectively modeling the complex temporal patterns in different tasks is hard as the temporal and spatial dimensions are entangled. Second, the relatedness among tasks may change over time. In this paper, we propose a neural network, DeepTrends, for multivariate time series trend prediction. The core module of DeepTrends is a tensorized LSTM with adaptive shared memory (TLASM). TLASM employs the tensorized LSTM to model the temporal patterns of long-term trend sequences in an MTL setting. With an adaptive shared memory, TLASM is able to learn the relatedness among tasks adaptively, based upon which it can dynamically vary degrees of parameter sharing among tasks. To further consider short-term patterns, DeepTrends utilizes a multi-task 1dCNN to learn the local time series features, and employs a task-specific sub-network to learn a mixture of long-term and short-term patterns for trend prediction. Extensive experiments on real datasets demonstrate the effectiveness of the proposed model.

Introduction

A large amount of time series data has been generated from various domains, such as traffic management (Lai et al. 2018), energy optimization (Rangapuram et al. 2018) and algorithmic trading (Wang et al. 2019a). Great efforts had previously been made for prediction on specific data points, which however could deliver very little information about the semantics and dynamics of the underlying process generating the time series (Lin, Guo, and Aberer 2017). In many applications, it is arguably even more crucial to learn and forecast the evolving trend which measures the intermediate behaviour, i.e., upward or downward pattern of time series. For example, the trend prediction has been popularly used in capacity planning, energy optimization, etc. Besides, in some applications, such as algorithmic trading, it is more achievable to predict the trend of stock price rather than forecast the market absolute values (Xu and Cohen 2018).

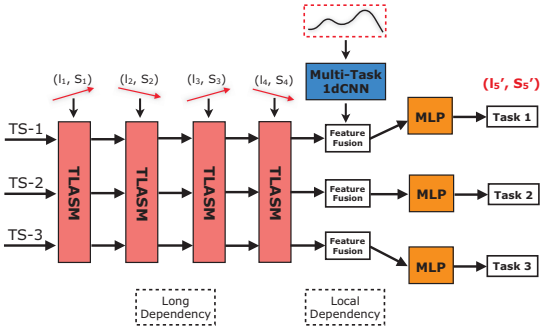


Figure 1: The overview of the proposed deep architecture for learning trends in multivariate time series.

In literature, there have been few efforts to learn and predict trends in time series data. In (Lin, Guo, and Aberer 2017), Lin et al. proposed a hybrid neural network to predict trend in univariate time series. In their work, a trend in time series is characterized by the slope and duration of the up/down movement of time series. The task is then to predict the slope and duration of the next coming trend given the historical data of time series over a period of time. Note that the existing approaches on time series prediction cannot be directly applied to the trend prediction as the task needs to predict both the duration and the slope of the subsequent trend, that are entangled.

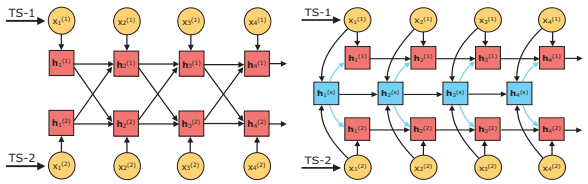
However, in many real-world applications, there are multiple variables at play, and handling all of them at the same time is crucial for an accurate prediction, because the trend pattern of one time series may influence others. For this

*This work was done when the first author was a summer intern at NEC Labs America. Wei Cheng is the corresponding author.

problem, a natural way is to take advantage of the multi-task learning (MTL) (Caruana 1997). In the MTL setting, the trend learning of each time series is considered as a task and different tasks are performed jointly. MTL can help improving the performance of tasks when they are related, and it also saves the computation cost by sharing model architectures (parameters) between related tasks. However, the MTL model may suffer significant degeneration in performance when tasks are less related to each other (Dong and De Melo 2018b; Ma, Li, and Hong 2019). Fig. 2a shows a basic MTL model for modeling the temporal patters of two time series, where each time series has its own parameters to generate hidden representations and the hidden representations of different time series influence each other by additional shared parameters. Compared to models without parameter-sharing, it introduces inductive bias into the shared architecture (Liu, Qiu, and Huang 2016). When tasks are unrelated, the inductive biases in different tasks will have conflicts and hurt task performance. To alleviate this problem, Liu et al. proposed a memory enhanced model that decouples the hidden representations into the task specific patterns and the shared ones (Liu, Qiu, and Huang 2016). The architecture of it is shown in Fig. 2b, in which an external memory is designed to share information among different tasks. However, the shared memory can not model task relatedness for better parameter-sharing. Another challenge comes from the temporal dynamics in different tasks. In many cases, the relatedness among tasks may change over time.

To address the above challenges, we propose a deep architecture, DeepTrends, for learning trends in multivariate time series as shown in Fig. 1. DeepTrends jointly learns both local and global contextual features for predicting the trend of time series. Its core module is a tensorized LSTM with adaptive shared memory (TLASM) (Fig. 3) to learn the sequential dependency of historical trends, which carries the information about long-term trend evolving. To further consider short-term dependency, DeepTrends utilizes a multi-task 1dCNN to learn the features of local raw time series, which delivers the information about the abrupt changing behavior of the trend evolution.

Specifically, TLASM leverages the tensorized LSTM to model the complex temporal patterns in different tasks, based upon which, an adaptive shared memory is designed to learn the task relatedness and dynamically integrate the shared information from related tasks into the representation of each individual task. The adaptive shared memory consists of multiple layers of sub-networks. TLASM learns the sub-network connections between different layers for information routing. In this way, one learning task can share more parameters with more related ones by selecting a similar sub-network. Each task is associated with one task specific unit at each time step for dynamical information routing. The idea of sub-network routing has been used in recent work (Misra et al. 2016; Ma et al. 2018; Ma, Li, and Hong 2019). However, none of them is designed for the sequential model. Moreover, time series data often involves a mixture of long-term and short-term patterns (Lai et al. 2018). In DeepTrends, TLASM is employed to model the long-term dependency within the sequence of historical trends. Since



(a) Basic MTL (b) Enhanced MTL

Figure 2: Two architectures for modeling the temporal patterns of two time series with multi-task learning.

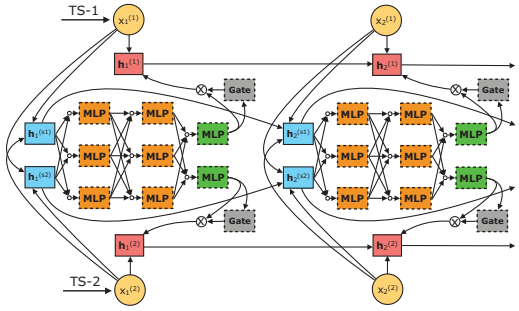


Figure 3: The architecture of the proposed TLASM for modeling the temporal patterns of two time series.

CNN is good at extracting patterns of local salience by applying a local connectivity between neurons (Lai et al. 2018; Lan et al. 2019), DeepTrends further employs a multi-task 1dCNN to extract salient features from local raw time series data, so as to model the short-term dependency between local time series data and the subsequent trend. A task-specific sub-network is then designed to integrate the long- and short-term dependency. To summarize, the major contributions are as follows.

- We propose DeepTrends, a multi-task deep learning model for learning trends in multivariate time series, which considers both long- and short-term dependency.
- We design TLASM, which is the first neural network capable to jointly model the temporal patterns of multivariate time series and achieve flexible parameter sharing.
- We perform extensive experiments on real datasets and the results demonstrate the effectiveness of DeepTrends.

The Problem

We extend the problem setting in (Lin, Guo, and Aberer 2017) into a multivariate one. n time series is denoted by $\mathbf{X} = (\mathbf{x}^1, \cdots, \mathbf{x}^n)^\top = (\mathbf{x}_1, \cdots, \mathbf{x}_T) \in \mathbb{R}^{n \times T}$, where $\mathbf{x}^i = (x_1^i, \cdots, x_T^i)^\top \in \mathbb{R}^T$ is the i-th time series and $\mathbf{x}_t = (x_t^1, \cdots, x_t^n)^\top \in \mathbb{R}^n$ represents the vector of n time series at time step t. T is the number of time steps. The historical trend sequence of $\mathbf{X}$ is the union of the trend sequence over each time series and denoted by $\mathcal{T} = \{\langle l_k^1, s_k^1 \rangle\} \cup \cdots \cup \{\langle l_k^n, s_k^n \rangle\}$, where $\{\langle l_k^i, s_k^i \rangle\}$ is the trend sequence of the i-th time series. $\langle l_k^i, s_k^i \rangle$ is the k-th trend of the i-th time series and describes a function over a subsequence (or a segment)

of the i-th time series. l_k^i and s_k^i represent the duration and slope of the k-th trend in the i-th time series respectively. l_k^i is measured in terms of the time range covered by the k-th trend of the i-th time series. Both l_k^i and s_k^i are continuous values. Trends of $\mathbf{X}$ are time ordered and non-overlapping. The durations of all the trends in each time series address $\sum_k l_k^i = T$. Details of how to extract trends in raw time series are discussed in the experiment section.

The local time series data delivers the information about the abrupt changing behavior of the trend evolution. The local data $w.r.t$ each historical trend is defined as the time series data with window size w. The local data of $\mathbf{X}$ is denoted by $\mathcal{L} = \{\langle x_{t_k-w}^1, \cdots, x_{t_k}^1 \rangle\} \cup \cdots \cup \{\langle x_{t_k-w}^n, \cdots, x_{t_k}^n \rangle\}$, where $\langle x_{t_k-w}^i, \cdots, x_{t_k}^i \rangle$ is the local data of the k-th trend of the i-th time series and t_k is the ending time of the k-th trend. Given $\mathcal{T}$ and $\mathcal{L}$, the goal is to learn the trends in multivariate time series for forecasting the subsequent trend of each time series, i.e., $\left\langle \hat{l}^1, \hat{s}^1 \right\rangle, \cdots \left\langle \hat{l}^n, \hat{s}^n \right\rangle$.

Data instances are built by combining the historical trend sequence, local raw time series data and the subsequent trend. All data instances are split into training set (80%), validation set (10%) and test set (10%). To generate trends, we adopt the l_1 trend filtering for multivariate time series (Kim et al. 2009). The objective function is

$$\sum_{t=1}^{T} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|_2^2 + \mu \sum_{t=2}^{T-1} \|\hat{\mathbf{x}}_{t-1} - 2\hat{\mathbf{x}}_t + \hat{\mathbf{x}}_{t+1}\|_2, \quad (1)$$

where $\mathbf{x}_t \in \mathbb{R}^n$ is the time series data at time step t and $\hat{\mathbf{x}}_t$ is the estimate. Using similar idea in the group Lasso, it couples together changes in the slopes of individual entries at the same time index, so the trend component found tends to show simultaneous trend changes. Note that even the trends in multivariate time series are asynchronous, the trend will be split into smaller pieces and maintains the predictive power (Kim et al. 2009). In the objective function, μ is a parameter to control the number of generated trends. The smaller μ is, the more fine-grained the trends are. The specific value of μ depends on the user's need.

The TLASM Network

We first introduce the basic LSTM, followed by how to extend it into the tensorized one with adaptive shared memory.

The Basic LSTM Network

The LSTM network is a powerful approach to learn the long-term dependency of sequential data (Xu et al. 2019b; 2019a). The calculation process of the LSTM unit (applied to each time step) is described in Eqs. (2)-(4). Given a sequence of input data $\mathbf{x}_1, \mathbf{x}_2, ... \in \mathbb{R}^n$, a memory cell $\mathbf{c}_t \in \mathbb{R}^d$ and a hidden state $\mathbf{h}_t \in \mathbb{R}^d$ are calculated for each input data by the following equations.

$$\begin{bmatrix} \tilde{\mathbf{c}}_t \\ \mathbf{f}_t \\ \mathbf{i}_t \\ \mathbf{o}_t \end{bmatrix} = \begin{bmatrix} \tanh \\ \sigma \\ \sigma \\ \sigma \end{bmatrix} (\mathbf{W}[\mathbf{x}_t \oplus \mathbf{h}_{t-1}] + \mathbf{b}), \quad (2)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \quad (3)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \quad (4)$$

where $\mathbf{W} \in \mathbb{R}^{4d \times (N+d)}$ and $\mathbf{b} \in \mathbb{R}^{4d}$ are parameters. $\mathbf{f}_t$, $\mathbf{i}_t$, $\mathbf{o}_t \in \mathbb{R}^d$ are called forget, input, output gates respectively and their values are in the range of [0,1]. These gates control how much information to keep/throw away. $\sigma(\cdot)$, $\oplus$ and $\odot$ represent element-wise $sigmoid$ function, concatenation operator and element-wise multiplication respectively. The LSTM unit can be rewritten as follows, where θ represents all the parameters.

$$(\mathbf{h}_t, \mathbf{c}_t) = LSTM(\mathbf{h}_{t-1}, \mathbf{c}_{t-1}, \mathbf{x}_t, \theta) \quad (5)$$

Intuitively, we can take the concatenation of the duration l_k^i and slope s_k^i as the input data x_k^i, and feed this concatenation in each trend of all time series into LSTM to learn the long-term trend dependency. After feeding the trend sequence $\mathcal{T}$ into LSTM, the hidden state $\mathbf{h}_t$ at the last time step is used as the overall representation of the trend sequences.

Individual time series typically presents different dynamics. However, as the basic LSTM blindly blends the information of all time series into the hidden state $\mathbf{h}_t$, it is intractable to further learn the time series-specific representations. Besides, the relatedness among the trend learning tasks of different time series can not be modeled by the hidden state mixing multivariate data, thus potentially hurting the trend learning task performance.

Tensorizing Hidden States

We tensorize the hidden states to learn the time series-specific representation, such that the hidden representation of each time series can be learned exclusively based on the data from that time series. The idea of tensorizing hidden states has been used in some recent work (He et al. 2017; Guo, Lin, and Antulov-Fantulin 2019) and has shown its advantages for sequential tasks.

The intuition behind tensorizing hidden states is that we represent the hidden state as a matrix $\mathbf{H}_t = (\mathbf{h}_t^1, \cdots, \mathbf{h}_t^n)^\top$, where $\mathbf{h}_t^i \in \mathbb{R}^{d_0}$ is the hidden state vector specific to the i-th time series. The data used to generate $\mathbf{h}_t^i$ is exclusively related to the i-th time series.

Given the newly coming data $\mathbf{x}_t \in \mathbb{R}^n$ and the previous state matrix $\mathbf{H}_{t-1}$, the hidden state is updated as follows.

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_c \circledast \mathbf{x}_t + \boldsymbol{\mathcal{U}}_c \otimes_n \mathbf{H}_{t-1} + \mathbf{B}_c), \quad (6)$$

where $\tilde{\mathbf{C}}_t = (\tilde{\mathbf{c}}_t^1, \cdots, \tilde{\mathbf{c}}_t^n)^\top$ has the same shape of the hidden state matrix $\mathbf{H}_{t-1}$. The element $\tilde{\mathbf{c}}_t^i \in \mathbb{R}^{d_0}$ corresponds to the hidden state update of the i-th time series.

$\mathbf{W}_c = (\mathbf{w}_c^1, \cdots, \mathbf{w}_c^n)^\top \in \mathbb{R}^{n \times d_0}$ is the input-to-hidden transition matrix, where $\mathbf{w}_c^i \in \mathbb{R}^{d_0}$. $\mathbf{W}_c \circledast \mathbf{x}_t$ captures the information from the input data and is defined by

$$\mathbf{W}_c \circledast \mathbf{x}_t = (\mathbf{w}_x^1 x_t^1, \cdots, \mathbf{w}_x^n x_t^n)^\top. \quad (7)$$

$\mathcal{U}_c = (\mathbf{U}_c^1, \cdots, \mathbf{U}_c^n)^\top \in \mathbb{R}^{n \times d_0 \times d_0}$ is the hidden-to-hidden transition tensor, where $\mathbf{U}_c^i \in \mathbb{R}^{d_0 \times d_0}$. $\mathcal{U}_c \otimes_n \mathbf{H}_{t-1}$ captures the information from the previous state matrix:

$$\mathcal{U}_c \otimes_n \mathbf{H}_{t-1} = (\mathbf{U}_c^1 \mathbf{h}_{t-1}^1, \cdots, \mathbf{U}_h^n \mathbf{h}_{t-1}^n)^\top, \tag{8}$$

where $\otimes_n$ indicates the tensor product along the axis of n.

From the view of MTL, tensorizing hidden states transforms the hidden state update of multivariate time series into multiple independent tasks, each of which corresponds to a time series. Thus, it helps learning time series-specific representations. However, it can not model the task relatedness.

Adaptive Shared Memory

We propose an adaptive shared memory to model task relatedness. Our goal is to make more related tasks share more model architecture/parameters and less related ones share less. Similar idea has been used in recent work (Misra et al. 2016; Ma et al. 2018; Ma, Li, and Hong 2019), however, none of them focuses on the recurrent neural networks for sequence modeling.

Fig. 3 illustrates the architecture of TLASM, in which the red cells are task-specific units and the middle parts are the adaptive shared memory that consists of multiple layers of parallel sub-networks. In the adaptive shared memory module, the first layer consists of multiple independent LSTMs (blue blocks), followed by a bunch of sub-networks (yellow rectangles) consist of multiple multilayer perceptrons (MLPs). The last layer (green blocks) is the task-specific MLPs to collect information for specific task. The connection between the sub-networks is a weighted average with attention mechanism. All the independent LSTMs and sub-networks are shared by all prediction tasks. The adaptive shared memory learns the connections between the sub-networks to encode the architecture space, which generates different sub-network routings. It achieves a flexible parameter sharing by learning to select a similar sub-network routing for related tasks. Besides, because it includes LSTMs as the first layer to read information from time series at each time step, the adaptive shared memory is able to model the task relatedness that may change over time.

The intuition behind multiple LSTMs included in the adaptive shared memory is that there are different shared hidden feature spaces for the tasks and each LSTM corresponds to one of them. Suppose the 1st layer includes p standard LSTMs and the 2nd layer includes q MLPs. After feeding all the trend sequence data into these LSTMs, the outputs at time step t are

$$(\mathbf{h}_t^{(1)}, \mathbf{c}_t^{(1)}) = LSTM_1(\mathbf{h}_{t-1}^{(1)}, \mathbf{c}_{t-1}^{(1)}, \mathbf{x}_t, \theta^{(1)}) \tag{9}$$

$$\cdots$$

$$(\mathbf{h}_t^{(p)}, \mathbf{c}_t^{(p)}) = LSTM_p(\mathbf{h}_{t-1}^{(p)}, \mathbf{c}_{t-1}^{(p)}, \mathbf{x}_t, \theta^{(p)}). \tag{10}$$

For the sub-network routing between the 1st layer and 2nd one, we use a weighted average of the 1st layer's outputs:

$$\begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \vdots \\ \mathbf{e}_q \end{bmatrix} = \begin{bmatrix} \alpha_{11}\mathbf{I}_{11} & \cdots & \alpha_{1p}\mathbf{I}_{1p} \\ \vdots & \ddots & \vdots \\ \alpha_{q1}\mathbf{I}_{q1} & \cdots & \alpha_{qp}\mathbf{I}_{qp} \end{bmatrix} \begin{bmatrix} \mathbf{h}_t^{(1)} \\ \mathbf{h}_t^{(2)} \\ \vdots \\ \mathbf{h}_t^{(p)} \end{bmatrix}, \tag{11}$$

where $\alpha_{ij} \geq 0$ is the weight, $\sum_j \alpha_{ij} = 1$ and $\mathbf{I}_{ij}$ is the identity matrix. α_{ij} represents the degree of connection between sub-networks and is learned by an attention module:

$$\alpha_{ij} = \frac{\exp\{\mathbf{w}_i^\top \tanh(\mathbf{V}_i \mathbf{h}_t^{(j)})\}}{\sum_{k=1}^{q} \exp\{\mathbf{w}_i^\top \tanh(\mathbf{V}_i \mathbf{h}_t^{(k)})\}}, \tag{12}$$

where $\mathbf{w}_i \in \mathbb{R}^{d_\alpha}$ and $\mathbf{V}_i \in \mathbb{R}^{d_\alpha \times d}$ are parameters. The number of attentions used between two layers equals to the number of sub-networks in the latter layer. Similarly, we design the sub-network routing between other layers. The attentions between different layers are different. As the number of sub-networks in the final layer equals to the number of tasks, i.e., the number of time series, we can get the output of the last layer as $\mathbf{R}_t = (\mathbf{r}_t^1, \cdots, \mathbf{r}_t^n)^\top$, where $\mathbf{r}_t^i \in \mathbb{R}^{d_r}$ is the information read from the shared memory for the i-th time series.

With the tensorized hidden states mechanism and the adaptive shared memory, we propose TLASM. The intuition behind TLASM is that the hidden state of time series is influenced by both the information from that time series and the information from related ones. Specifically, each time series has its own memory $\mathbf{c}_t^i \in \mathbb{R}^{d_0}$ storing the time series-specific information and the adaptive shared memory $\mathbf{r}_t^i \in \mathbb{R}^{d_r}$ storing the information of related time series. When generating the hidden state of the i-th time series $\mathbf{h}_t^i$, it needs to read information from both the two memories $\mathbf{r}_t^i$ and $\mathbf{c}_t^i$.

The calculation process of the TLASM unit is described in Eqs. (13)-(16). As a standard LSTM neural network, TLASM has the forget gate $\mathbf{F}_t$, input gate $\mathbf{I}_t$, output gate $\mathbf{O}_t$ and the memory cell $\mathbf{C}_t$ in the update process. Given the input data $\mathbf{x}_1, \mathbf{x}_2, ... \in \mathbb{R}^n$, the cell matrix $\mathbf{C}_t$ and the state matrix $\mathbf{H}_t$ are calculated as follows.

$$\begin{bmatrix} \mathbf{F}_t \\ \mathbf{I}_t \\ \mathbf{O}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \end{bmatrix} (\mathbf{W} \circledast \mathbf{x}_t + \mathcal{U} \otimes_n \mathbf{H}_{t-1} + \mathbf{B}), \tag{13}$$

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t, \tag{14}$$

$$\mathbf{G}_t = \sigma(\mathcal{U}_c \otimes_n \mathbf{C}_t + \mathcal{U}_r \otimes_n \mathbf{R}_t), \tag{15}$$

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t + \mathbf{G}_t \odot (\mathcal{U}_g \otimes_n \mathbf{R}_t)), \tag{16}$$

where $\mathbf{W} \in \mathbb{R}^{3n \times d_0}$, $\mathcal{U} \in \mathbb{R}^{3n \times d_0 \times d_0}$, $\mathbf{B} \in \mathbb{R}^{3n \times d_0}$, and $\mathcal{U}_c, \mathcal{U}_r, \mathcal{U}_g \in \mathbb{R}^{n \times d_0 \times d_0}$ are parameters. $\tilde{\mathbf{C}}_t$ is the updated state matrix. $\mathbf{F}_t, \mathbf{I}_t, \mathbf{O}_t \in \mathbb{R}^{n \times d}$ are the gates in the form of matrix. $\mathbf{G}_t \in \mathbb{R}^{n \times d}$ is a fusion gate. It selects a part of the information read from the shared memory, which is merged with the time series-specific memory into a new one for each task (Eq. (16)). Similar idea of memory fusion strategy has been used in (Liu, Qiu, and Huang 2016).

Similar to the case of tensorizing hidden states (Eq. (6)), the tensor-dot operations ensure the data used to generate the gates (Eq. (13)) and the memory cell matrix (Eq. (14)) of each time series are exclusively from the corresponding time series. TLASM can also be considered as a set of parallel LSTMs, each of which processes one time series and then merges via the adaptive shared memory.

Deep Architecture with TLASM for Learning Trends in Multivariate Time Series

The overview of the proposed deep architecture is shown in Fig. 1. The sequences of historical trends of all time series are fed into TLASM to learn the long-term trend evolving. A multi-task 1dCNN is applied to the local raw data of each time series to extract local features. The outputs of TLASM and 1dCNN are further fed into a task-specific subnetwork to get the final trend prediction of that time series.

TLASM for Learning Long-Term Trend Evolving The trend sequences of all time series, i.e., $\mathcal{T} = \{\langle l_k^1, s_k^1 \rangle\} \cup \cdots \cup \{\langle l_k^n, s_k^n \rangle\}$, are fed into TLASM to learn long-term trend evolving. Specifically, we concatenate l and s of all time series as the input data $\mathbf{x}_t = (l_t^1, s_t^1, \cdots, l_t^n, s_t^n)^\top \in \mathbb{R}^{2n}$. The output of TLASM, $\mathbf{H} \in \mathbb{R}^{2n \times d}$, is the transformed representation of all trend sequences. We denote $L^i(\mathcal{T})$ as the part of $\mathbf{H}$ that corresponds to the i-th time series.

Multi-task 1dCNN for Learning Local Features To extract the features of local time series data, DeepTrends employs a multi-task 1dCNN module which enjoys the classic architecture of the shared-bottom MTL. In the module, a low-level subnetwork is shared by all time series and each time series has its own subnetwork built on top of the shared one. All these subnetworks consist of multiple stacked layers of 1d convolutional, activation and pooling operations. The elements of local data $\mathcal{L} = \{\langle x_{t_k-w}^1, \cdots, x_{t_k}^1 \rangle\} \cup \cdots \cup \{\langle x_{t_k-w}^n, \cdots, x_{t_k}^n \rangle\}$ are fed into the multi-task 1dCNN module. The output that corresponds to the i-th time series is denoted by $C^i(\mathcal{L})$.

Task-Specific Sub-networks Motivated by the success of multi-task learning (Ma, Li, and Hong 2019; Liu, Johns, and Davison 2019), we design a task-specific sub-network for the trend learning of each time series. The outputs of TLASM and multi-task 1dCNN, i.e., $L^i(\mathcal{T})$ and $C^i(\mathcal{L})$, are concatenated and fed into the task-specific sub-network. The output of the sub-network for the i-th time series is

$$\left\langle \hat{l}^i, \hat{s}^i \right\rangle = f^i(L^i(\mathcal{T}) \oplus C^i(\mathcal{L})), \tag{17}$$

where $f^i(\cdot)$ represents an MLP that consists of m layers of neurons. The output of the k-th layer can be expressed as $\mathbf{y}_k = \varphi(\mathbf{W}_k^i(\mathbf{y}_{k-1}) + \mathbf{b}_k^i)$, where φ is the leaky ReLU activation function and $\mathbf{W}_k^i$, $\mathbf{b}_k^i$ are parameters.

Objective Function Given the trend sequences $\mathcal{T}$ and the local data $\mathcal{L}$, the objective function is

$$J = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{z^i}\sum_{k=1}^{z^i}\left[(\hat{l}_k^i - l_k^i)^2 + (\hat{s}_k^i - s_k^i)^2\right] + \lambda P_{nn}, \tag{18}$$

where z^i is the number of trends in the i-th time series and P_{nn} is the penalization term for the parameters to prevent our model from over-fitting. λ is a hyper-parameter.

Experiment

We use three datasets and seven baselines for experiments.

Table 1: Description of the datasets

Dataset	Traffic			Exchange-Rate			Solar-Power		
μ	10	40	100	2	6	20	400	600	800
# Trends	5187	3755	1499	2312	1508	925	13957	8554	4856
# Time Steps	17544			7588			52560		
# Time Series	9			7			9		

Table 2: Comparison of baseline methods

Method	Models Dependency		Parameter Sharing		
	Long-Term	Short-Term	Hard	Soft	Adaptive
CNN	×	✓	×	×	×
LSTM	✓	×	×	×	×
CLSTM	✓	×	×	×	×
TreNet	✓	✓	×	×	×
LSTM-m	✓	×	✓	×	×
TreNet-m	✓	✓	✓	×	×
ME-LSTM	✓	✓	×	✓	×
DTrends-L	✓	✓	✓	×	×
DTrends-C	✓	✓	×	×	✓
DeepTrends	✓	✓	×	×	✓

Datasets, Baselines and Experiment Setting

The three public datasets are summarized in Table 1.

- Traffic[1]: The collection of 48 months hourly data from the California Department of Transportation. The data describes the road occupancy rates (between 0 and 1) measured by different sensors on San Francisco Bay area freeways.
- Exchange-Rate: The collection of the daily exchange rates of Australia, British, Canada, Switzerland, Japan, New Zealand and Singapore ranging from 1990 to 2016.
- Solar-Power[2]: The production records of solar power in 2006. The records are sampled every 10 minutes from 137 PV plants in Alabama State. We take the records of nice plants of them.

To conduct a comprehensive comparison, we select three values of μ for each dataset to generate the trend sequence with different granularities. The details of the generated trends are summarized in Table 1.

We compared our model with the seven baseline methods. The comparison between them is shown in Table 2. CNN, LSTM, CLSTM and TreNet are originally designed for univariate data. We apply them to each time series independently to predict the trends. LSTM-m, TreNet-m and ME-LSTM adopt multi-task learning and predict the trends of different time series jointly.

- CNN: The model applies 1dCNN to the local raw data of each time series independently to predict trends.
- LSTM-u: The model applies LSTM to the trend sequence of each time series independently to predict trends.

[1] http://pems.dot.ca.gov/

[2] https://www.nrel.gov/grid/solar-power-data.html

- CLSTM: The model feeds the features learnt by CNN over the raw time series into LSTM to predict trends (Ballas et al. 2016).
- TreNet: The model is designed for univariate time series. It uses LSTM and 1dCNN to capture the dependency in the trend sequence and the local raw data respectively to predict trends (Lin, Guo, and Aberer 2017).
- LSTM-m: The model feeds the trend sequences of all time series into a LSTM to predict the trends jointly.
- TreNet-m: The model extends TreNet to model multivariate data. LSTM and 1dCNN are applied to the trend sequences and the local raw data of all time series respectively to predict the trends of different time series jointly.
- ME-LSTM: The model augments LSTM with an external memory to capture the long term dependency (Liu, Qiu, and Huang 2016) and employs multi-task 1dCNN to capture local features.

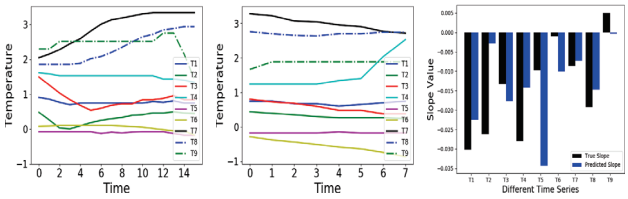
To evaluate the effectiveness of different components of DeepTrends, we do ablation study on its variants. DTrends-L is the variant that replaces TLASM with the basic LSTM. DTrends-C is the variant that replaces the multi-task 1dCNN with a basic 1dCNN to extract local features. We evaluate the performance in terms of root mean square error (RMSE) and root relative squared error (RRSE) (Lai et al. 2018). Lower RMSE/RRSE indicates higher performance.

In our experiments, all the models are trained by Adam algorithm (Kingma and Ba 2014). The adaptive shared memory consists of 4 layers of sub-networks. The 1st one consists of 4 LSTMs. The number of MLPs in the last layer equals to the number of time series. Other layers consist of 4 MLPs. All MLPs are 2-layer fully-connected networks. In multi-task 1dCNN, both the shared 1dCNN and the task-specific one have two stacked convolutional layers, and they have 64 filters of size 2 and 4, 32 filters of size 2 and 4 respectively. The task-specific sub-networks are 2-layer fully-connected networks. The window size T, local data size w and λ are grid-searched from $\{8, 16, 32, 64\}$, $\{4, 8, 16, 32\}$ and $\{0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$ respectively. d_0 d_0, d_α, d_r and the hidden size of MLP layers are grid-searched from $\{8, 16, 32, 64\}$. The learning rates of all models are random-searched within $[0.00005, 0.1]$ in log-scale. The code of DeepTrends is publicly available[3]

Prediction Results

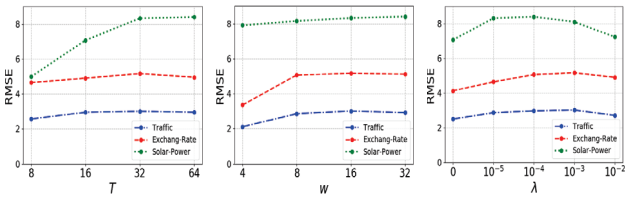
Table 3 shows the prediction performances. It is observed that DeepTrends achieves the best performance on both the duration and the slope predictions. Compared to CNN and LSTM, TreNet shows better performance. This is because TreNet can model both the long- and short-term dependencies in the trend of univariate time series. TreNet outperforms CLSTM, indicating the validity of feeding the extracted trends into LSTM to capture the sequential dependency of historical trends. LSTM-m outperforms LSTM on the Solar-Power dataset but not on the Traffic dataset. This is because different time series are related in Solar-Power, which benefits the MTL used in LSTM-m, and the low relatedness between time series in Traffic makes LSTM-m suffer degeneration in performance. DeepTrends adopts an adaptive shared memory that can model the relatedness flexibly, which helps DeepTrends outperform LSTM and LSTM-m on both datasets. Similar cases can be observed between TreNet and TreNet-m. ME-LSTM outperforms LSTM, indicating the advantage of enhanced memory when modeling the temporal patterns of multiple time series. DeepTrends outperforms ME-LSTM, which indicates the advantage of adaptive shared memory over enhanced memory. By comparing DeepTrends with DTrends-L and DTrends-C, we see the benefits of tensorizing hidden states and multi-task 1dCNN (compared to basic 1dCNN) to extract local features respectively.

[3]https://github.com/DerronXu/DeepTrends/tree/master



(a) Local data (b) Future data (c) True slope v.s. Predicted slope

Figure 4: Visualization of the trend slope prediction by DeepTrends on the Traffic dataset.



(a) Varying T (b) Varying w (c) Varying λ

Figure 5: Parameter sensitivity study of DeepTrends.

Visualization of Trend Slope Prediction

To gain further insight about the long- and short-term dependencies in the trend learning, we visualize the trend slope prediction by DeepTrends on the Traffic dataset as shown in Fig. 4. It is observed that, for the T6 time series, the local data shows a downward trend overall. In the subsequent future time period, the time series value of T6 decreases as shown in Fig. 4b. This indicates the validity of local data used for trend prediction. DeepTrends predicts the trend direction of T6 successfully (Fig. 4c). For T7, its local data shows an upward trend, however, its future data shows a downward trend as shown in Fig. 4b. This is because the long-term dependency in trend sequence also influences the subsequent trend. As shown in Fig. 4c, DeepTrends predicts the trend slope of T7 accurately, which verifies the ability of the TLASM network to learn the long-term trend evolving.

Table 3: Trend (duration & slope) prediction results on each dataset.

Dataset		Traffic			Exchange-Rate			Solar-Power			Traffic			Exchange-Rate			Solar-Power		
		Duration									Slope								
		μ			μ			μ			μ			μ			μ		
Methods	Metrics	10	50	170	2	6	20	400	600	800	10	50	170	2	6	20	400	600	800
CNN	Rmse	2.1215	3.9815	12.4465	3.5391	6.1490	10.0548	10.0505	15.4735	20.4078	0.1016	0.0270	0.0275	0.0157	0.0120	0.0084	0.0196	0.0170	0.0248
	Rrse	1.0265	1.3997	2.0688	26.5035	29.3988	31.4994	2.4176	2.3791	2.5307	0.8938	1.6943	1.1357	43.6662	2.4127	2.9410	1.4979	1.4894	1.5817
LSTM	Rmse	2.0466	3.2263	9.0031	3.0915	5.2540	9.0721	8.9872	13.0804	18.5547	0.1539	0.0317	0.0208	0.0158	0.0181	0.0304	0.0134	0.0161	0.0170
	Rrse	0.9583	0.8745	0.9000	1.8089	1.7093	1.3682	1.1360	1.1401	1.2885	8.2372	14.9347	1.5882	9.8686	1.6277	1.2297	3.8386	2.1871	2.7085
CLSTM	Rmse	2.2803	3.9689	10.3119	3.4449	5.8263	9.3573	8.9829	13.6282	19.1766	0.1062	0.0542	0.0679	0.0164	0.0156	0.0273	0.0142	0.0162	0.0243
	Rrse	1.4100	0.9491	1.6693	1.9105	1.7658	14.4338	1.1832	1.6952	1.2246	0.9191	2.1959	2.1812	10.9342	1.6568	1.7672	1.8320	1.3968	1.6294
TreNet	Rmse	1.9428	3.0801	9.4828	3.0865	5.2595	**8.1762**	9.0520	12.9362	18.7838	0.0888	0.0244	0.0258	0.0143	0.0126	0.0148	0.0129	0.0160	0.0351
	Rrse	0.8959	0.7253	0.8561	1.8188	1.7208	1.3823	1.1911	1.1852	1.2383	0.7257	1.1628	1.0147	3.7117	1.8401	1.1758	1.3026	1.1661	1.0655
LSTM-m	Rmse	2.3541	3.2469	9.8074	3.9940	5.2879	8.3240	8.0060	12.7251	19.2483	0.1293	0.0314	0.0346	0.0142	0.0489	0.0770	0.0122	0.0195	0.0729
	Rrse	0.8733	0.7145	0.8474	1.3344	1.6351	1.3863	0.8539	0.9683	1.1861	1.5978	6.7605	1.2966	4.8976	1.6653	1.2051	1.8516	1.3556	1.5496
TreNet-m	Rmse	2.2255	3.3440	10.6698	3.9548	7.2710	8.7669	7.9072	12.8988	19.3374	0.0770	0.0231	0.0367	0.0141	0.0226	0.0280	0.0153	0.0451	0.0840
	Rrse	0.8530	**0.7074**	0.8994	1.2471	1.2502	1.2278	0.8433	0.9657	1.2025	0.6983	**1.0532**	**1.0065**	4.0165	1.6324	1.1032	1.3940	1.1386	1.1136
ME-LSTM	Rmse	1.7984	3.0551	11.7528	3.0501	5.6678	8.3554	7.5384	12.1377	19.4335	0.0949	0.0288	0.0255	0.0166	0.0123	**0.0094**	0.0126	0.0155	0.0209
	Rrse	0.6829	0.7654	0.9819	1.7857	1.1985	1.2548	0.8897	0.8400	1.0669	0.8163	1.8951	1.0729	3.7765	2.4440	2.0697	1.7924	1.3367	1.0595
DTrends-L	Rmse	2.2076	3.3460	11.4092	3.7655	5.4440	8.6764	7.7359	12.3016	20.7890	0.0788	0.0242	0.0255	0.0144	0.0150	0.0884	0.0134	0.0186	0.0284
	Rrse	0.8191	0.7724	0.8953	1.2020	1.1670	1.0719	0.8037	0.8973	1.1171	0.6419	1.2139	1.1829	3.5058	1.6496	1.1138	2.7451	1.5100	1.3490
DTrends-C	Rmse	1.8120	3.0253	10.5177	3.1496	5.8344	8.3924	7.5047	9.8812	15.6963	0.0890	0.0268	0.0188	0.0142	0.0122	0.0251	0.0122	0.0160	0.0163
	Rrse	0.6595	0.7762	0.8543	1.1326	1.1973	1.0718	0.8293	0.5995	0.82948	0.6937	1.6592	1.1818	3.3471	1.8354	**1.0965**	1.3082	1.1578	1.0543
DeepTrends	Rmse	**1.7587**	**3.0182**	**9.4343**	**2.9920**	**5.1705**	8.2209	**7.4439**	**8.3247**	**13.5987**	**0.0763**	**0.0202**	**0.0139**	**0.0140**	**0.0111**	0.0196	**0.0119**	**0.0152**	**0.0154**
	Rrse	**0.6087**	0.7544	**0.8412**	**1.1232**	**1.1615**	**1.0668**	**0.8011**	**0.5647**	**0.7114**	**0.6040**	1.1728	1.1543	**2.1804**	**1.3109**	1.1288	**1.2893**	**1.1013**	**1.0448**

For T3, there are two sub-trends (one downward and one upward) in its local data. For its future data, T3 shows a slightly downward trend. This is because the trend of T3 is influenced by both long- and short-term dependencies, which is also captured by DeepTrends (Fig. 4c).

Parameter Sensitivity

We study the sensitivity of DeepTrends with respect to the time window size T, the local data size w and the hyper-parameter λ. Figs. 5a-5c show the RMSE of duration prediction on Traffic (μ=50), Exchange-rate (μ=6), Solar-Power (μ=600) by changing one parameter while fixing others. It is observed that as T and w increase, RMSE decreases in general until an optimal value. However, a large λ may hurt the performance. $T \in$ [32,64], $w \in$ [8,16,32] and $\lambda \in [10^{-5},10^{-4},10^{-3}]$ give the optimal results. Thus it is reasonable to set them to 32, 16 and 10^{-4} respectively. Moreover, the non-zero choices of λ verify the importance of the penalization term in the objection function.

Related Work

There have been intensive interests in developing prediction methods on specific data points for its practical importance (Rangapuram et al. 2018; Liang et al. 2019). A lot of work in time series forecasting is based on deep learning techniques because of the rapid development of deep neural networks (Dong and De Melo 2018a; 2019; Xu et al. 2019c; Wang et al. 2019b). For example, a dual-stage attention-based RNN was proposed to predict time series by capturing long-term temporal dependency and selecting relevant time series (Qin et al. 2017). The social media data and stock price signals were exploited jointly for stock movement prediction in a deep generative model (Xu and Cohen 2018). Some others explored the combinations of recurrent neural networks with dilation, residual connections and attention (Chang et al. 2017; Kim, El-Khamy, and Lee 2017). More recently, the adversarial training was leveraged to improve the stock movement prediction using an attentive LSTM (Feng et al. 2019). Based on backward and forward residual links and a deep stack of fully-connected layers, a deep neural architecture for interpretable time series forecasting was proposed (Oreshkin et al. 2019). However, the prediction on specific data points could deliver limited information about the semantics and dynamics of time series (Bufo et al. 2014; Lin, Guo, and Aberer 2017). It is more desirable to learn and forecast the evolving trend which measures the upward or downward pattern of time series. There are few efforts to learn trends in time series data. A hybrid neural network was proposed to predict trend in univariate time series (Lin, Guo, and Aberer 2017), in which a trend is characterized by the slope and duration of the up/down movement of time series.

In many real-world applications, time series are multivariate. Because the trend pattern of one time series may influence others, handling all of them jointly is crucial for an accurate prediction. Multi-task learning (MTL) is a natural way to resolve this problem (Caruana 1997). MTL has been broadly used in image classification (Liu, Johns, and Davison 2019), natural language processing (Liu et al. 2019) and video classification (Ma, Li, and Hong 2019). The most commonly used architecture of MTL is the shared-bottom model, where several low-level sub-networks are shared by all tasks and each task has its own task-specific sub-network (Ruder12 et al. 2019). However, the MTL model may suffer significant degeneration in performance when tasks are less related to each other (Ma, Li, and Hong 2019). To resolve this problem, a memory enhanced model was proposed to decouple the hidden representations into the task specific patterns and the shared ones (Liu, Qiu, and Huang 2016). Some recent work utilized sub-network routing for a flexible parameter sharing to consider task relatedness (Misra et al. 2016; Ma et al. 2018; Ma, Li, and Hong 2019). However, none of them focuses on the recurrent neural networks for sequence modeling.

Conclusion

In this paper, we propose a deep architecture, DeepTrends, for learning trends in multivariate time series. The core module of DeepTrends is a TLASM network, which is used to capture the long-term dependency in the historical trend sequence. Particularly, TLASM tensorizes the hidden states to model the complex temporal patterns in different tasks. An adaptive shared memory is proposed to learn the task relatedness and dynamically integrates the shared information from related tasks into the learning process of individual task. To consider the short-term dependency between the local data and the subsequent trend, a multi-task 1dCNN is designed to extract the features of local raw time series. A task-specific sub-network is further designed to integrate the long- and short-term dependency. Extensive experimental results demonstrate the effectiveness of DeepTrends.

Acknowledgement

This project was partially supported by NSF grants IIS-1707548 and CBET-1638320.

References

Ballas, N.; Yao, L.; Pal, C.; and Courville, A. 2016. Delving deeper into convolutional networks for learning video representations. In *ICLR*.

Bufo, S.; Bartocci, E.; Sanguinetti, G.; Borelli, M.; Lucangelo, U.; and Bortolussi, L. 2014. Temporal logic based monitoring of assisted ventilation in intensive care patients. In *ISoLA*, 391–403.

Caruana, R. 1997. Multitask learning. *Machine learning* 28(1):41–75.

Chang, S.; Zhang, Y.; Han, W.; Yu, M.; Guo, X.; Tan, W.; Cui, X.; Witbrock, M. J.; Hasegawa-Johnson, M.; and Huang, T. S. 2017. Dilated recurrent neural networks. *NIPS*.

Dong, X., and De Melo, G. 2018a. Cross-lingual propagation for deep sentiment analysis. In *AAAI*.

Dong, X., and De Melo, G. 2018b. A helping hand: Transfer learning for deep sentiment analysis. In *ACL*, 2524–2534.

Dong, X., and De Melo, G. 2019. A robust self-learning framework for cross-lingual text classification. In *EMNLP-IJCNLP*, 6307–6311.

Feng, F.; Chen, H.; He, X.; Ding, J.; Sun, M.; and Chua, T.-S. 2019. Enhancing stock movement prediction with adversarial training. In *IJCAI*.

Guo, T.; Lin, T.; and Antulov-Fantulin, N. 2019. Exploring interpretable lstm neural networks over multi-variable data. In *ICML*.

He, Z.; Gao, S.; Xiao, L.; Liu, D.; He, H.; and Barber, D. 2017. Wider and deeper, cheaper and faster: Tensorized lstms for sequence learning. In *NeurIPS*, 1–11.

Kim, S.-J.; Koh, K.; Boyd, S.; and Gorinevsky, D. 2009. \ell_1 trend filtering. *SIAM review* 51(2):339–360.

Kim, J.; El-Khamy, M.; and Lee, J. 2017. Residual lstm: Design of a deep recurrent architecture for distant speech recognition. *Interspeech*.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lai, G.; Chang, W.-C.; Yang, Y.; and Liu, H. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *SIGIR*, 95–104. ACM.

Lan, S.; Yu, R.; Yu, G.; and Davis, L. S. 2019. Modeling local geometric structure of 3d point clouds using geo-cnn. In *CVPR*, 998–1008.

Liang, J.; Xu, D.; Sun, Y.; and Honavar, V. 2019. Lmlfm: Longitudinal multi-level factorization machines. *arXiv preprint arXiv:1911.04062*.

Lin, T.; Guo, T.; and Aberer, K. 2017. Hybrid neural networks for learning the trend in time series. In *IJCAI*, 2273–2279.

Liu, P.; Fu, J.; Dong, Y.; Qiu, X.; and Cheung, J. C. K. 2019. Learning multi-task communication with message passing for sequence learning. In *AAAI*, volume 33, 4360–4367.

Liu, S.; Johns, E.; and Davison, A. J. 2019. End-to-end multi-task learning with attention. In *CVPR*, 1871–1880.

Liu, P.; Qiu, X.; and Huang, X. 2016. Deep multi-task learning with shared memory. In *EMNLP*.

Ma, J.; Zhao, Z.; Yi, X.; Chen, J.; Hong, L.; and Chi, E. H. 2018. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *SIGKDD*, 1930–1939. ACM.

Ma, J.; Li, Z. Z. J. C. A.; and Hong, L. 2019. Snr: Sub-network routing for flexible parameter sharing in multi-task learning. In *AAAI*.

Misra, I.; Shrivastava, A.; Gupta, A.; and Hebert, M. 2016. Cross-stitch networks for multi-task learning. In *CVPR*, 3994–4003.

Oreshkin, B. N.; Carpov, D.; Chapados, N.; and Bengio, Y. 2019. N-beats: Neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint arXiv:1905.10437*.

Qin, Y.; Song, D.; Cheng, H.; Cheng, W.; Jiang, G.; and Cottrell, G. W. 2017. A dual-stage attention-based recurrent neural network for time series prediction. In *IJCAI*, 2627–2633.

Rangapuram, S. S.; Seeger, M. W.; Gasthaus, J.; Stella, L.; Wang, Y.; and Januschowski, T. 2018. Deep state space models for time series forecasting. In *NeurIPS*, 7785–7794.

Ruder12, S.; Bingel, J.; Augenstein, I.; and Søgaard, A. 2019. Latent multi-task architecture learning. In *AAAI*.

Wang, J.; Zhang, Y.; Tang, K.; Wu, J.; and Xiong, Z. 2019a. Alphastock: A buying-winners-and-selling-losers investment strategy using interpretable deep reinforcement attention networks. In *SIGKDD*, 1900–1908. ACM.

Wang, Z.; Feng, X.; Tang, J.; Huang, G. Y.; and Liu, Z. 2019b. Deep knowledge tracing with side information. In *AIED*, 303–308. Springer.

Xu, Y., and Cohen, S. B. 2018. Stock movement prediction from tweets and historical prices. In *ACL*, 1970–1979.

Xu, D.; Cheng, W.; Luo, D.; Gu, Y.; Liu, X.; Ni, J.; Zong, B.; Chen, H.; and Zhang, X. 2019a. Adaptive neural network for node classification in dynamic networks. In *ICDM*. IEEE.

Xu, D.; Cheng, W.; Luo, D.; Liu, X.; and Zhang, X. 2019b. Spatio-temporal attentive rnn for node classification in temporal attributed graphs. In *IJCAI*, 3947–3953.

Xu, D.; Cheng, W.; Zong, B.; Ni, J.; Song, D.; Yu, W.; Chen, Y.; Chen, H.; and Zhang, X. 2019c. Deep co-clustering. In *SDM*, 414–422. SIAM.