# End to End Time Series Analysis

**Anonymous Author(s)**
Affiliation
Address
`email`

## Abstract

Time series plays an important role in practical applications, including stock predictions, traffic scheduling, clinical diagnosis. Such applications face two challenges: on one hand, most relationships are nonlinear , on the other hand, sampled data could always be noisy and incomplete. In the last few years, the newly developed deep learning based approaches started appearing in the community and have shown superior performances than the traditional models. This work gives a review of the most popular deep learning models and other algorithms of end-to-end learning for time series problems. While the neural networks have shown promising results on static data, such as natural language processing and computer vision, applying them with respect to time is still a challenging work. An overall review is provided to discuss the details of the models, learning features and tasks.

## 1 Introduction

With widespread adoption of electronic records, there is an increasing availability of large amounts of time series data and the need of performing accurate forecasting of future behavior in several scientific and applied domains. Time series usually contain temporal dependencies which may cause two otherwise identical samples to have different behavior. This characteristic demands the definition of robust and efficient techniques able to infer from observations the stochastic dependency between past and future. The forecasting domain used to be dominated by linear statistical methods such as ARIMA, which requires complete data and has to impute missing values before prediction[2][1]. More recently, with the possibility to model historical data as a nonlinear function and approximate complex latent relationships, machine learning models have established themselves as serious contenders to classical statistical models[5][4][3]. Here we give a review of those end-to-end (E2E) models for time series problems, where E2E refers to training a possibly complex learning system represented by a single model (specifically a Deep Neural Network) that represents the complete target system, bypassing the intermediate layers usually present in traditional pipeline designs.

In this work, we mainly investigate various end-to-end time series tasks: traditional time series forecasting, i.e., predicting the trend of a stock, time series classification (TSC), where we address to build a classifier to label the time series samples, and time series clustering. Formally, in the area of time series problems, given a sequence $\left\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(n)}\right\} = X$ of observed states, where each element $\mathbf{x}^{(t)} \in \mathbb{R}^m$ is a vector describing the physical status at time step $t$ such that $\left\{x_1^{(t)}, \cdots, x_m^{(t)}\right\} = \mathbf{x}^{(t)}$, the E2E problem aims to predict the value of or label $\mathbf{x}^{(n+\Delta)}$ denoted by $\hat{\mathbf{y}}$ for some prediction horizon $\Delta$ [6][7], which is usually a fixed value in most scenarios.

The rest of this paper is structured as follows: Section 2 talks about some supervised learning settings. Section 3 introduces basic types of neural network modules that are commonly used for time series tasks and for complicated model build-ups. Section 4, 5, 6 describe some tasks using end-to-end deep learning to perform modeling, classification and clustering. Section 7 ensues, illustrating another work from Facebook. At the end, Section 8 concludes this work.

## 2 Supervised Learning Setting

In end-to-end time series analysis, supervised learning consists in modeling, on the basis of a finite set of observations, the relation between a set of input variables and one or more output variables, which are considered somewhat dependent on the inputs. To model an input/output mapping, i.e. to achieve E2E learning, the general approaches rely on the availability of a collection of data points typically referred to as training set that we denoted early by $X$, as well as the output value $Y$.

## 3 Neural Networks

For end-to-end time series, there are countless variations of deep learning models and it would not be possible to cover all the proposed methods. Hence, for simplicity, we focus on three basic types of neural networks known as *Multilayer Perceptrons (MLP)*, *Convolutional Neural Networks (CNN)* and *Recurrent Neural Networks (RNN)*, as well as some additional ones developed from those models. The reason for generally selecting the following models is that they enjoy the popularity along with the lucidity in the community.

### 3.1 Multilayer Perceptrons

MLPs are one of the first proposed architectures in artificial neural networks (ANN) and are considered to be the simplest type of neural network architectures. Though there are a significant number of studies on MLPs for time series problems, MLPs consst of three mainly components: input layer, hidden layer and output layer.

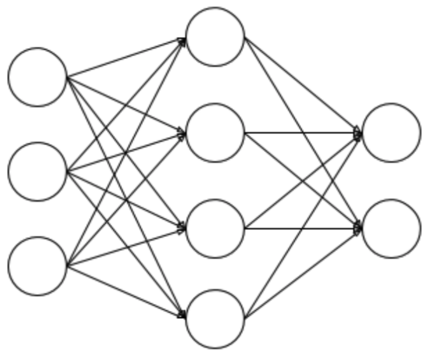

Figure 1: The basic structure of multilayer perceptrons. Image via UCB EECS189.

Each node (neuron) in the hidden layers and in the output layers computes a weighted sum of its inputs $\mathbf{x}$ with the weight $\mathbf{w}$ and the bias term $b$. To approximate nonlinear relationships, on top of that, there is a nonlinear activation function $\sigma$ for the nodes that is usually layer-wise shared. Concretely, the computation on the node can be expressed as

$$\sigma(\mathbf{w}^\top \mathbf{x} + b).$$

Let $L$ be the number of layers in an MLP, then the entire model computes the function

$$\boldsymbol{\sigma}_L \left(\mathbf{W}_L \sigma_{L-1} \left(\cdots \boldsymbol{\sigma}_2 \left(\mathbf{W}_2 \boldsymbol{\sigma}_1 \left(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right) \cdots\right) + \mathbf{b}_L\right),$$

where $\mathbf{W}$ and $\mathbf{b}$ are the matrix of weights and the vector of bias in the layer $l$ respectively.

The important issues for MLPs are the hyperparameters, the activation functions and the training method, which have been mentioned in the previous lectures of the course and will not be repeated here. For time series forecasting, previous studies MLPs can be powerful and universal due to its good generalization ability. However, with a comparatively simpler structure, in most cases MLP perform worse than other complex networks, e.g. LSTM [8][9].

## 3.2 Convolutional Neural Networks

CNN is a type of neural networks that consists some specific layers (convolutional and pooling layers) for convolutional operations. CNN enjoys all the benefits that MLP has while not requiring to learn from lag observations which MLP does, while significantly reducing the number of weights. Though having been used in various fields, like object segmentation, style conversion, automatic coloring, what CNN can really do is just function as a feature extractor.

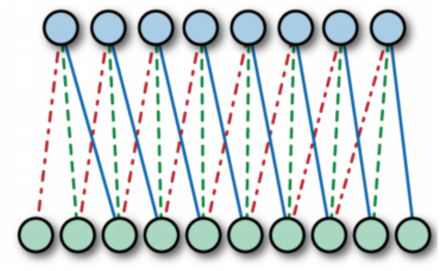

Figure 2: The basic structure of convolutional layers. Image via UCB EECS189.

A convolutional layer takes a $W \times H \times D$ dimensional input $I$ and convolves it with a $w \times h \times D$ dimensional filter $G$. Mathematically, the convolution operator is defined as

$$(\mathbf{I} * \mathbf{G})[x, y] = \sum_{a=0}^{w-1} \sum_{b=0}^{h-1} \sum_{c \in \{1 \cdots D\}} I_c[x+a, y+b] \cdot G_c[a, b].$$

From the operation of convolution, it can be seen that any unit in the output image is only related to a part of the input image. In traditional neural networks, since they are all fully connected, any unit of output must be affected by all inputs. In this way, the recognition effect of the potential features will be greatly reduced. Specifically speaking, each area has its own unique characteristics, which we wish not to be affected by other areas.

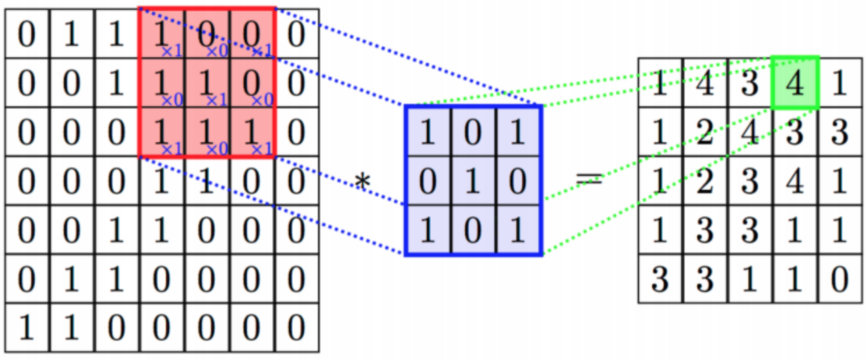

Figure 3: Convolving a filter with an image, where $W = H = 7$, $D = 1$ and $w = h = 3$. Image via UCB EECS189.

## 3.3 Recurrent Neural Networks

The most popular model that have been used for modeling sequential data is the Recurrent Neural Network [10][11]. In RNN, the activations from each time step are stored in the internal state of the model in order to provide the memory at that moment.

Had there not been the cycle arrow in the left part of fig. 4, the wrapped RNN would be an MLP, which is made up by the input, the hidden component and the output. Let $U$, $V$ and $W$ be the weight matrices between a input state and the hidden state ($x_t$ and $h_t$), between a hidden state and the output state at the same time step ($h_t$ and $y_t$), and between hidden states ($h_{t-1}$ and $h_t$) respectively. With

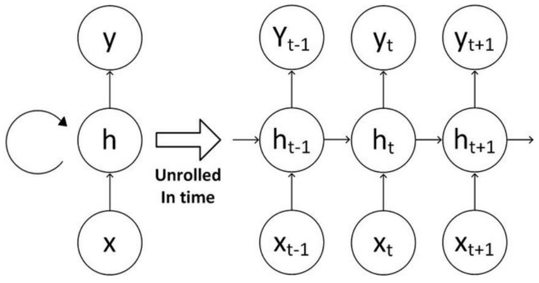

Figure 4: A conceptual visualization of the recurrent nature of an RNN, where $x$ is the input data for the input layer, $h$ represents the hidden layers that carry internal state (memory) through the time, and $y$ is the model output. Image by Savvas Varsamopoulos.

$b$ and $\sigma$ remain to denote the bias term between hidden layers and the activation function severally, additionally letting $c$ be the bias term between the hidden layer and the output layer, we have the equations in the computational graph as:

$$\begin{aligned} h_t &= \sigma\left(Ux_t + Wh_{t-1} + b\right) \\ y_t &= \sigma\left(Vh_t + c\right) \end{aligned}$$

Note that $\sigma$ may differ regarding disparate targets of the network structures, which ranges from sigmoid, tanh, ReLU to softmax.

Now we will introduce back-propagation through time (BPTT), a classic method for RNN training, the essence of which in fact is still the back propogation algorithm. Since RNN processes time series data, the algorithm is time series related and is called back propagation with time. The core idea of BPTT is the same as that of BP, continuously searching for the optimal point along the negative gradient direction of the parameters that need to be optimized. However, all $U$, $W$, and $V$ here are shared at various positions in the sequence, i.e., we are updating the same parameters during backpropagation through all the laters $h_t$, $h_{t-1}$, ... Since the sequence has a loss function at every moment, the final loss $L$ is:

$$L = \sum_{t=1}^{n} L_t,$$

which leads us to the partial derivatives of those weights. For $V$, we have

$$\frac{\partial L}{\partial V} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial \tilde{y}_t} \frac{\partial \tilde{y}_t}{\partial o_t} \frac{\partial o_t}{\partial V}.$$

However, the ones for $U$ and $W$ would be more complicated. Here we let $t = 3$ for simplicity:

$$\begin{aligned} \frac{\partial L_3}{\partial W} &= \frac{\partial L_3}{\partial \tilde{y}_3} \frac{\partial \tilde{y}_3}{\partial o_3} \frac{\partial o_3}{\partial s_3} \frac{\partial s_3}{\partial W} + \frac{\partial L_3}{\partial \tilde{y}_3} \frac{\partial \tilde{y}_3}{\partial o_3} \frac{\partial o_3}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial W} + \frac{\partial L_3}{\partial \tilde{y}_3} \frac{\partial \tilde{y}_3}{\partial o_3} \frac{\partial o_3}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W}, \\ \frac{\partial L_3}{\partial U} &= \frac{\partial L_3}{\partial \tilde{y}_3} \frac{\partial \tilde{y}_3}{\partial o_3} \frac{\partial o_3}{\partial s_3} \frac{\partial s_3}{\partial U} + \frac{\partial L_3}{\partial \tilde{y}_3} \frac{\partial \tilde{y}_3}{\partial o_3} \frac{\partial o_3}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial U} + \frac{\partial L_3}{\partial \tilde{y}_3} \frac{\partial \tilde{y}_3}{\partial o_3} \frac{\partial o_3}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial U}, \end{aligned}$$

where $s_t$ is the hidden state at time $t$. While picking sigmoid or tanh as $\sigma$, the first derivative will be between 0 and 1. As the time series continues to deepen, the cumulative multiplication of decimals will cause the gradient to become smaller and smaller until it is close to 0, which will cause the gradient to disappear, i.e. *gradient vanishing*. It needs to be pointed out here that since the first derivative of tanh has a larger range and is symmetrical about 0, the convergence speed is faster than sigmoid, and the gradient disappears slower than sigmoid. If the activation function is ReLU, when the input is positive, the gradient is always 1, so there will be no problem of the gradient vanishing after accumulation, but during the accumulation process, the gradient will become larger and larger, which is easy to cause the problem known as *gradient exploding*. When the input is negative, the gradient is always 0, which will cause the problem of gradient disappearance. At present, the gradient problem is not only solved by the activation function, but is usually solved by combining parameter initialization and batch regularization.

Though theoretically RNN works smoothly as a sequential data approximator, the major challenge with a typical generic RNN is that these networks remember only a few earlier steps in the sequence and thus are not suitable when a longer sequence is involved just like we have discussed above. A type of neural network architecture that solves the problem is Long Short-Term Memory.

### 3.4 Long Short-Term Memory

LSTM is a special kind of Recurrent Neural Network (RNN) with the capability of remembering the values from the past for the purpose of future use [12]. A key reason for the success of LSTM in time series problems is its ability to filter and keep memorization from the earlier stage by the recurrent connected units (*cells*) and the gates along with two horizontal lines incorporated, which carry the memory.

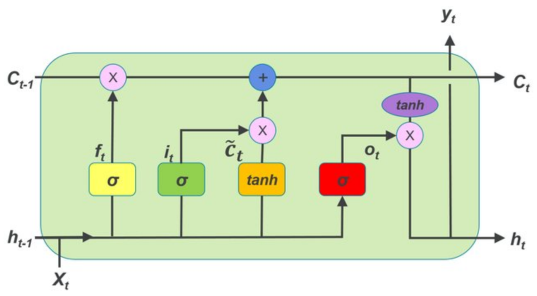

Figure 5: A cell of Long Short-Term Memory Neural Network, where the colored components are different components of the network. Image by Ismail et al.

Regarding a typical LSTM, there are three kinds of gates: forget, input and output. Based on fig. 5, we will describe its internal elements on an individual basis. which are surrounding and connecting by various of lines. First is the *forget gate*, the leftmost part of the cell, i.e. the yellow rectangle and the lines through it, which absorb the hidden state from the last cell $h^{(t-1)}$ as well as the newly input $X^{(t)}$. The sigmoid activate function output $f^{(t)} \in [0, 1]$, representing the extent to which the past memory remains in the current cell:

$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right),$$

where $U$, $W$, $b$ have the same meaning as the ones in RNN.

*Input gate*, built up by the green and the orange activation functions and the lines through them, controls the extent to which $X^{(t)}$ would flow into the cell, which is expressed by two parts:

$$i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right),$$
$$a^{(t)} = \tanh\left(W_a h^{(t-1)} + U_a x^{(t)} + b_a\right).$$

Before moving to the *Output gate*, we have to firstly take care of the current cell state $C^{(t)}$ and the effects of the former two gates:

$$C^{(t)} = C^{(t-1)} \odot f^{(t)} + i^{(t)} \odot a^{(t)}.$$

Here $\odot$ denotes the Hadamard product.

Then with the new born cell state, the rightmost part of the cell, i.e. the *output gate*, will generate the hidden state $h^{(t)}$ and the predicton $\hat{y}^{(t)}$ as follows:

$$o^{(t)} = \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right),$$
$$h^{(t)} = o^{(t)} \odot \tanh\left(C^{(t)}\right),$$
$$\hat{y}^{(t)} = \sigma\left(V h^{(t)} + c\right).$$

## 3.5 Conclusions of different neural networks

So what makes LSTM a better model than others? Here we will give an intuitive understanding. The sequence problem emphasizes the order of sequence, which also leads to the concept of context. In a weather forecasting problem, the weather of tomorrow may be related to the combination formed by the recent ones, or it may be all the weathers before it, and with the help of memory units such as the state $s_t$ of the RNN, the output of a sequence position is mathematically related to the input of all previous sequences. Though due to the multiplicative problem of the gradient of the vanilla RNN, the influence of the previous sequence is almost zero, and this is later corrected by LSTM as an additive problem. The mathematical foundation of RNN can be considered as a Markov chain, and the subsequent value is determined by the probability of the former and some parameters.

As for CNN, the convolution kernel of which emphasizes the window in space, it is extracting the features in another way. This window is the same as the sequence problem in that it also considers the front and back, but RNN does not and will not consider the space up and down issues. Similar to the price of stocks, there will be no multiple output (prices) on the same input. There is only one line in the entire space. Such data density itself is not suitable for CNN. We can imagine such a scenario: using one of CNN The convolution kernel processes $m$ samples of a 1-dimensional feature, you will easily understand that it is essentially a fully connected neural network (FCN).

So in this way, you may ask why can't DNN/MLP be used to replace RNN? In fact, quite same as the answer to why DNN can't be used to replace CNN, CNN and RNN also use a set of parameters as a window to scan the entire sample set, instead of using one like DNN, which requires large-scale parameter network to connect this sample.

In summary, The main purpose of RNN/LSTM is to promise the continuity of data, i.e. let the data have context, while CNN is trying to treat and learn from the data as a whole. However, modern neural network design is not only relying on CNN or RNN, but is usually a more complex (and mathematically robust) reasonable structure. In the following section, we will talk about different tasks with some distinct models.

## 4 Time Series Forecasting

Almost all the neural networks mentioned above are suitable for performing time series forecasting tasks. Benefited from the availability of universal approximation, neural networks have almost dominated this field. Several different deep learning approaches can be found in the literature for performing forecasting tasks [13][14][15].

LSTM, by its nature utilizes the temporal characteristics of any time series signal, hence forecasting end-to-end time series is a well-studied and successful implementation of LSTM. Meanwhile, series combination models, especially linear/nonlinear hybrid models (e.g. ARIMA/LSTM), are not uncommon in the literature [16][17].

When it come to the metrics of the experiments, besides the MAE and MSE that have been mentioned in the previous lectures, here we introduce a few more common metrics for time series analysis.

Root mean square error (RMSE) is the standard deviation of the prediction errors, can be expressed as

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}.$$

Mean absolute percentage error gives a good idea of the relative error, whose formula is given by

$$\mathrm{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \cdot 100\%.$$

This approach, however, is facing the problem when the sequences can have small denominators that leads to a zero or nearly zero division. Since MAPE puts a heavy penalty on negative errors, as a consequence, following this metric the system may prefer the model with a lower prediction. This drawback can be avoided by taking the logarithm of the accuracy ratio.

Last, we have R-squared

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}},$$

where $SS_{\text{res}}$ is the is the sum of squared residuals from the predicted values and $SS_{\text{tot}}$ is the sum of squared deviations of the dependent variable from the sample mean. As the fraction of the total sum of squares that is explained by the model, this mechanic shows whether the model fits the observed samples and how good it fits, where a high $R^2$ stands for a high correlation.

Here we introduce one recent proposed work for time series forecasting[38][39]. The work is named N-BEATS, which implements both single-variable many-to-many prediction. The network structure has two functions, one is called restoration, and the other is prediction. The role of restoration is to perform effective time series feature extraction and give full play to the advantages of deep learning automatic feature learning. The block design of the entire network is similar to the idea of iterative decision tree, where the network is used to learn residuals, and the neural network first learns simple rules, and then learns complex rules.

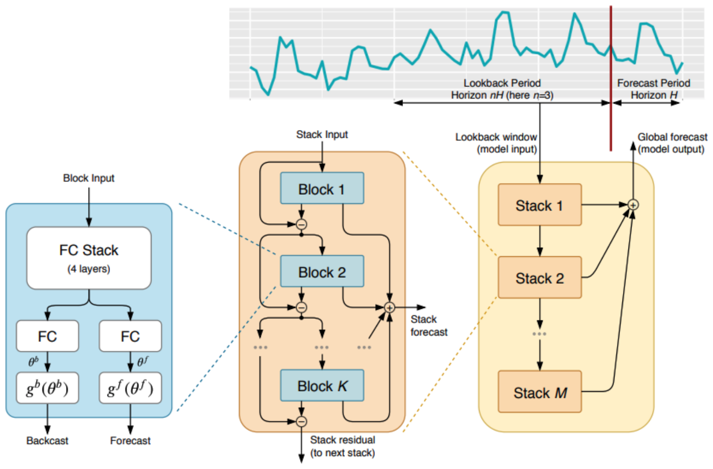

Figure 6: The architecture of N-BEATS. Image by Oreshkin et al. [38]

Each block is divided into two parts. The first part is used to generate backward expansion coefficients and forward expansion coefficients. After going through four fully connect layers, the original time series input then goes through the projection layer, after which generates the subsequent expansion coefficient $\theta_l^b$ and the previous expansion coefficient $\theta_l^f$. The linear projection layer is a simple linear transformation. To summarize the work flow up to now,

$$\mathbf{h}_{\ell,1} = \mathrm{FC}_{\ell,1}(\mathbf{x}_\ell), \quad \mathbf{h}_{\ell,2} = \mathrm{FC}_{\ell,2}(\mathbf{h}_{\ell,1}), \quad \mathbf{h}_{\ell,3} = \mathrm{FC}_{\ell,3}(\mathbf{h}_{\ell,2}), \quad \mathbf{h}_{\ell,4} = \mathrm{FC}_{\ell,4}(\mathbf{h}_{\ell,3}),$$
$$\boldsymbol{\theta}_\ell^b = \mathrm{LINEAR}_\ell^b(\mathbf{h}_{\ell,4}), \quad \boldsymbol{\theta}_\ell^f = \mathrm{LINEAR}_\ell^f(\mathbf{h}_{\ell,4}).$$

Then for the connection between blocks, which are connected by the residual parts, we have the formula

$$\mathbf{x}_\ell = \mathbf{x}_{\ell-1} - \widehat{\mathbf{x}}_{\ell-1}, \quad \widehat{\mathbf{y}} = \sum_\ell \widehat{\mathbf{y}}_\ell.$$

The target of the following block, intuitively speaking, is learning the residual from the transportation of the last block.

As for the interpretability, they proposes two structures, one of which named generic architecture does not depend on specific experience knowledge. The change can be seen as a waveform reconstruction in time. Since there is no additional restriction on the $V_l^f$, $\hat{y}_l$ has a poor interpretability. Thus the other is to meet the interpretability requirements by introducing expert experience, that is, to achieve interpretability by introducing the trend and seasonality of the time series, just like the traditional statistical models. The specific method is to artificially set the $V$ matrix multiplied by the $\theta$ so that the matrix meets the requirements of trend and seasonality. The enlightenment from this aspect is that the introduction of expert experience can be realized by designing the numerical form of the multiplication matrix. This is a great work which combines both traditional models and deep learning ones. At the end, they calculate the loss by MAPE as we mentioned early.

## 5 Time Series Classification

Time series classification (TSC), the problem to label the time series samples, has become a popular task of time series [18]. Given the need to accurately classify time series data, researchers have proposed hundreds of methods to solve this task, such as a nearest neighbor classifier with Dynamic Time Warping (DTW) [19][20], which has been proved to be a promising baseline. In short, TSC is trying to compare the similarities and label them.

However, in this area, the lengths of the two time series that need to be compared for similarity may not be the same. In the field of speech recognition, the speaking speed of different people is different. Since the speech signal has considerable randomness, even if the same person makes the same tone at different times, it is impossible to have a complete length of time. Moreover, the pronunciation speed of different phonemes in the same word is also different. For example, some people will drag the sound "A" very long, or pronounce "i" very short. In these complex situations, the distance (or similarity) between two time series cannot be effectively obtained using the traditional Euclidean distance.

Hence, DTW is proposed as a substitution. DTW can calculate the similarity of two time series, which is especially suitable for time series of different lengths and different rhythms (such as audio sequences in which different people read the same word). DTW will automatically warping and warp the time series, that is, local zoom on the time axis. So that the shapes of the two series are as consistent as possible, and the maximum possible similarity is obtained. DTW uses the dynamic programming method to calculate the time warping. It can be said that the application of the dynamic programming method to the time warping problem is DTW. More details about DTW have been covered in the homework.

Regarding end-to-end learning as the topic of this notes, here we introduce some CNN based methods from related studies [21][22][23]. After revolutionizing the performances of several tasks in computer vision like object classifications, face verifications and etc., CNN has been introduced to this topic to address TSC problems because of its success in representation learning. One of the recent works inspired by computer vision encodes the time series data as images and then apply CNN as a classifier [24], as CNN can treat the raw input data as a 1-D image then read it and store it as important element. Compared with $k$-NN with DTW, such a deep learning framework can learn a hierarchical feature representation from raw data automatically, matching the concept of end-to-end learning perfectly [25].

Here we talk about a work combining LSTM, convolutional network with attention mechanism: ALSTM-FCN [26].

The attention mechanism is a technique often used in neural translation of text, coming up as a solution for sequence tasks. Recent works have argued that attention mechanisms, without any recurrence, can be effective in end-to-end modeling tasks [27][28]. The fully convolutional block consists of three stacked temporal convolutional blocks with filter sizes of 128, 256, and 128 respectively. Each convolutional block is identical to the convolution block in the CNN architecture. Each block consists of a temporal convolutional layer, which is accompanied by batch normalization and followed by a ReLU activation function. Finally, global average pooling is applied after the final convolution block.

Simultaneously, the time series input is conveyed into a dimension shuffle layer. The transformed time series from the dimension shuffle is then passed into the LSTM block. The LSTM block, comprising of either a general LSTM layer or an Attention LSTM layer, is followed by a dropout. The output of the global pooling layer and the LSTM block is concatenated and passed onto a softmax classification layer.

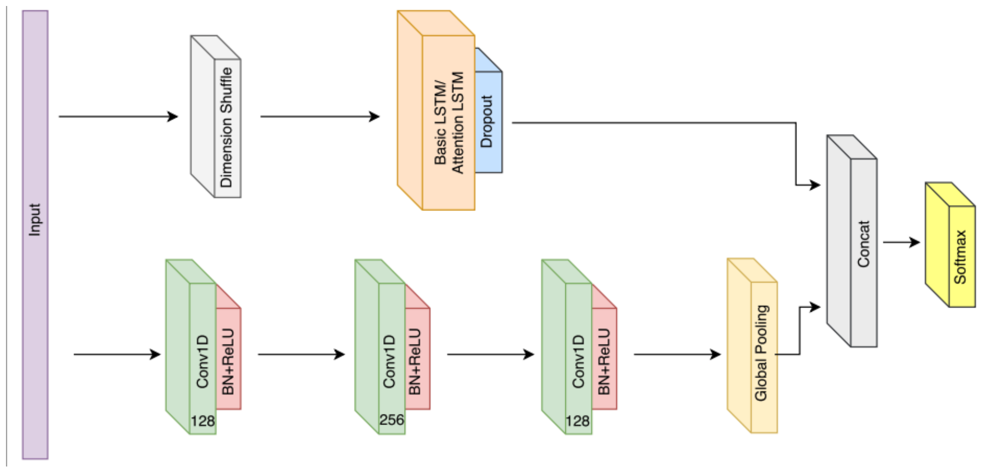

Figure 7: The LSTM-FCN architecture. LSTM cells can be replaced by Attention LSTM cells to construct the ALSTM-FCN architecture. Image by Karim et al.

## 6 Anomaly Detection

An anomaly, or an outlier, is a sample that performing significantly different from the rest in the set, or in the sequence for temporal samples. Anomaly detection refers to retrieve, detect even predict those abnormal issues in time series sequences. Reliable temporal uncertainty estimation is a critical industrial demand, e.g. business and properties monitoring for Yahoo and Microsoft [29][30]. Due to the complexity and the comprehensiveness of the designs, a business supporting anomaly detection system always showed a far more complicated framework. In this section we will briefly introduce some representative systems.

### 6.1 Extensible GenericAnomaly Detection System (EGADS)

Proposed by a team from Yahoo, EGADS is the first comprehensive system for anomaly detection that is flexible, accurate, scalable and extensible. Though it is not an end-to-end learning model, we still put it here as it properly define the basic concepts for anomaly detection systems and is open source[1].

Some recent works based on deep learning have compared themselves against EGADS and have made remarkable improvements on both real and synthetic data [31].

DeepAD, an anomaly detection model that leverages a plethora of basic models, has three main phases as illustrated in fig. 9: time series forecasting, merge predictions, and anomaly detector. At the first phase both a statistical model (ARIMA) and a nonlinear model (LSTM) will be trained, the results of which will then be applied and combined to mix the advantages of those basic models. Then the anomaly detector ensued, computing a dynamic threshold at each time step on the past scaled squared error. This is a generic anomaly detection framework that does not utilize the prior knowledge neither for training nor for decision making.

Meanwhile, there are some studies focusing on simpler scenarios, e.g. holiday prediction for Uber trips [32][33]. In the following work, the Uber team build their model with an encoder-decoder framework for inherent temporal pattern capturing and a prediction network that takes both the encoder-decoder output and potential external features to guide the final prediction.

Before fitting the prediction model, it is necessary to first perform pre-training to fit an encoder that can extract useful and representative embedding from the time series. The goal has two directions: (i) to ensure that the learned embedding provides useful features for prediction; (ii) to prove that abnormal input can be captured in embedding and further spread to the prediction network.

[1]Source code available at https://github.com/yahoo/egads.

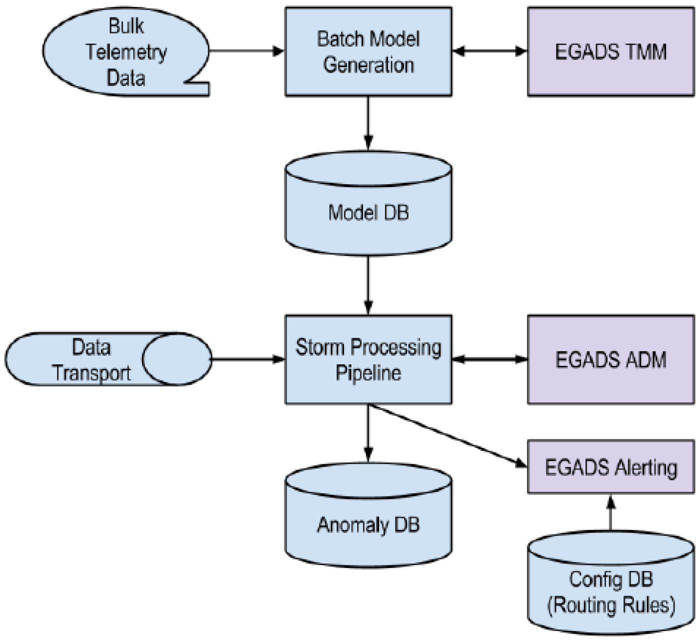

Figure 8: The architecture for EGADS-YMS. Image by Laptev et al.

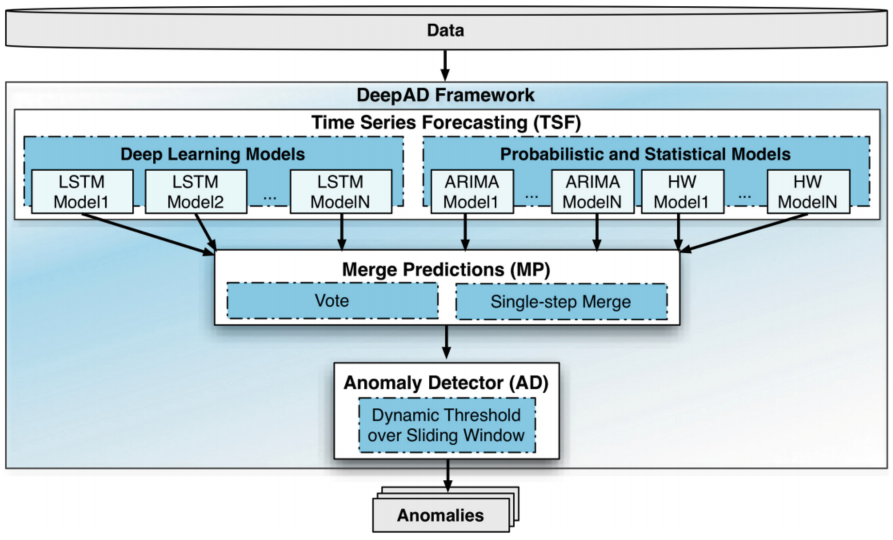

Figure 9: The framework overview of DeepAD. Image by Buda et al.

When predicting potential samples with very different patterns from the training set, it could be thought as the uncertainty is captured. Here the method is tend to train an encoder, which extracts representative features from a time series. In a sense, a decoder can recover the time series from the encoding space. At test time, the coding quality of each sample will provide insights on how close it is to the training set. Another insight is that first they use the encoder-decoder network to fit all the training time series with the latent embedding space, and then evaluate the gap between the test data and the training data in the embedding space.

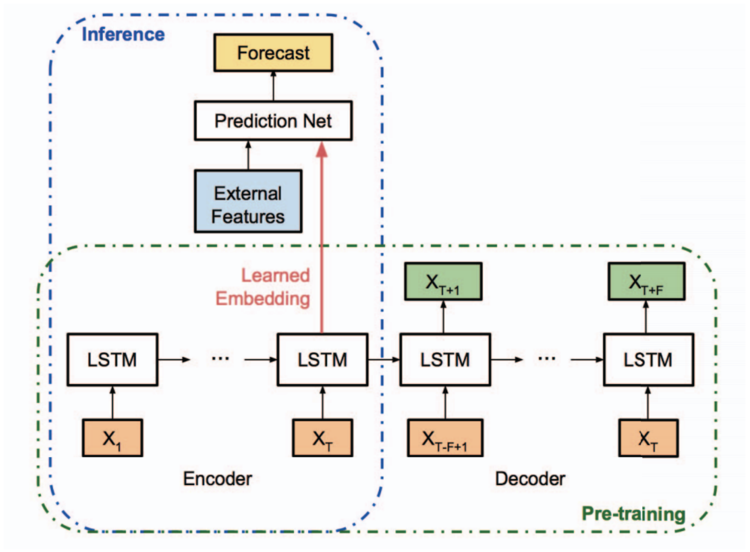

Figure 10: The neural network architecture proposed by Uber team. Image by Zhu and Laptev.

## 7 Prophet

Besides traditional models like ARIMA and those fancy neural networks, there is also a sophisticated yet multipurpose tool provided by Facebook for time series forecasting, Prophet[34]. Based on time series decomposition and machine learning fitting, Prophet can not only handle the case of some outliers in the time series problems, but also the case of some missing values, and predict the future trend of the time series almost automatically. This tool that provides programming interfaces in both R and Python enables statisticians and analysts to use their expertise for business or research purposes. In this section, we will briefly introduce the methodology of Prophet and leave some practice problems in the homework.

### 7.1 Algorithm

In the field of traditional time series analysis, there is a common analysis method called time series decomposition, where let $y_t$ be the time series. Then we can write

$$y_t = S_t + T_t + R_t$$

where $S_t$ is the seasonal component, $T_t$ is the trend component, and $R_t$ is the remainder component, all with respect to period $t$. Similarly, a multiplicative decomposition would be written as

$$y_t = S_t \times T_t \times R_t,$$

which is equivalent to

$$\ln y_t = \ln S_t + \ln T_t + \ln R_t.$$

Generally speaking, in real life and business tasks, in addition to seasonality, trend, and remainders, there are usually holiday effects, which in Prophet are expressed as

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t,$$

where $g(t)$ is the piecewise linear or logistic growth curve for modelling non-periodic changes in time series, $s(t)$ is the periodic changes (e.g. weekly/yearly seasonality), $h(t)$ is the effects of holidays (user provided) with irregular schedules, and $\varepsilon(t)$ is the error term accounts for any unusual changes not accommodated by the model.

### 7.2 Application

As a competitive time series analysis tool, Prophet has been applied for real world business predictions, where the data are highly non stationary and irregular. Here we will focus on a work in retail industry, which presents a framework capable of accurately forecasting future sales in the retail industry and classifying the product portfolio according to the expected level of forecasting reliability [35]. The proposed framework, that would be of great use for any company operating in the retail industry, is based on Facebook's Prophet algorithm and backtesting strategy. Real-world sales forecasting benchmark data obtained experimentally in a production environment in one of the biggest retail companies in Bosnia and Herzegovina is used to evaluate the framework and demonstrate its capabilities in a real-world use case scenario.

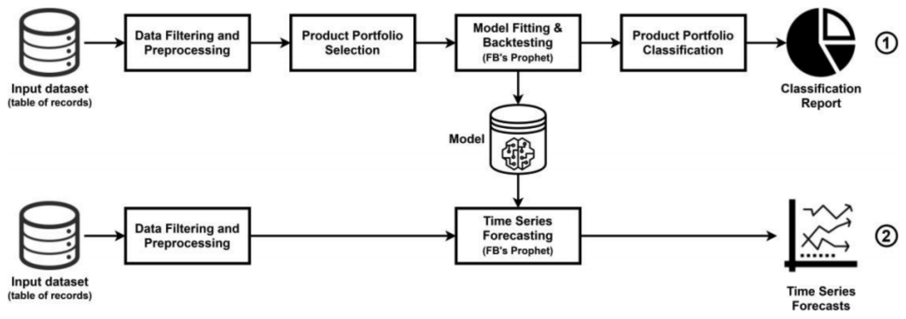

Figure 11: The proposed sales forecasting model. Image by Žunić et al.

In the work flow shown in fig. 11, after data preprocessing, Prophet is used for modelling the dynamics of sales for items in a product portfolio without using additional regressors, with the aim of generating monthly and quarterly sales forecasts. It is empirically concluded that at least 24 months of historical data is required for reliable estimation of trend and/or seasonal effects.

## 8 Conclusions

In addition to the above ones, there are also other interesting end-to-end works for time series tasks: an imputation network with residual short paths for simultaneous time series data imputation and prediction [36], Apply *Gated Restricted Boltzmann Machine* (GRBM) for video action recognition[37], and etc.

The dominance of RNN-based models for end-to-end time series analysis will probably not disappear anytime soon, mainly due to their easy adaptation to most temporal dependency problems. Meanwhile, some enhanced versions of the original LSTM or RNN models, generally integrated with hybrid learning systems started becoming more common. Here we grouped the studies and listed some symbolic models. However, it indicates that there are still latent patterns and opportunities waiting for our readers to explore because of the high uncertainty of time series.

## Appendix A Datasets

Here we offer some datasets that are widely used by the community.

- UCR Time Series Classification Archive: A superset for 48 datasets (updated in 2018 fall) for time series classification and clustering. Available at https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.
- Human Activity Recognition Using Smartphones Data Set: A dataset recording the activities while using smartphones, where the features are obtained by calculating variables from the time and frequency domain. Available at https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones#.
- Yahoo Webscope Program: A reference library of interesting and scientifically useful datasets, used for anomaly detection. Available at https://webscope.sandbox.yahoo.com/.

## References

[1] Hillmer, Steven Craig, and George C. Tiao. "An ARIMA-model-based approach to seasonal adjustment." Journal of the American Statistical Association 77.377 (1982): 63-70.

[2] De Gooijer, Jan G., and Rob J. Hyndman. "25 years of time series forecasting." International journal of forecasting 22.3 (2006): 443-473.

[3] Siami-Namini, Sima, Neda Tavakoli, and Akbar Siami Namin. "A comparison of ARIMA and LSTM in forecasting time series." 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 2018.

[4] Bontempi, Gianluca, Souhaib Ben Taieb, and Yann-Aël Le Borgne. "Machine learning strategies for time series forecasting." European business intelligence summer school. Springer, Berlin, Heidelberg, 2012.

[5] Ahmed, Nesreen K., et al. "An empirical comparison of machine learning models for time series forecasting." Econometric Reviews 29.5-6 (2010): 594-621.

[6] Lippi, Marco, Matteo Bertini, and Paolo Frasconi. "Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning." IEEE Transactions on Intelligent Transportation Systems 14.2 (2013): 871-882.

[7] Venkatraman, Arun, Martial Hebert, and J. Andrew Bagnell. "Improving multi-step prediction of learned time series models." Twenty-Ninth AAAI Conference on Artificial Intelligence. 2015.

[8] Chen, Lin, et al. "Which artificial intelligence algorithm better predicts the Chinese stock market?." IEEE Access 6 (2018): 48625-48633.

[9] Cao, Jian, Zhi Li, and Jian Li. "Financial time series forecasting model based on CEEMDAN and LSTM." Physica A: Statistical Mechanics and its Applications 519 (2019): 127-139.

[10] Hüsken, Michael, and Peter Stagge. "Recurrent neural networks for time series classification." Neurocomputing 50 (2003): 223-235.

[11] Sagheer, Alaa, and Mostafa Kotb. "Time series forecasting of petroleum production using deep LSTM recurrent networks." Neurocomputing 323 (2019): 203-213.

[12] Gers, Felix A., Douglas Eck, and Jürgen Schmidhuber. "Applying LSTM to time series predictable through time-window approaches." Neural Nets WIRN Vietri-01. Springer, London, 2002. 193-200.

[13] Romeu, Pablo, et al. "Time-series forecasting of indoor temperature using pre-trained deep neural networks." International conference on artificial neural networks. Springer, Berlin, Heidelberg, 2013.

[14] Turner, Jeffrey T. Time series analysis using deep feed forward neural networks. University of Maryland, Baltimore County, 2014.

[15] Sezer, Omer Berat, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. "Financial time series forecasting with deep learning: A systematic literature review: 2005–2019." Applied Soft Computing 90 (2020): 106181.

[16] Khashei, Mehdi, and Zahra Hajirahimi. "A comparative study of series arima/mlp hybrid models for stock price forecasting." Communications in Statistics-Simulation and Computation 48.9 (2019): 2625-2640.

[17] Chaâbane, Najeh. "A hybrid ARFIMA and neural network model for electricity price prediction." International journal of electrical power & energy systems 55 (2014): 187-194.

[18] Fawaz, Hassan Ismail, et al. "Deep learning for time series classification: a review." Data Mining and Knowledge Discovery 33.4 (2019): 917-963.

[19] Berndt, Donald J., and James Clifford. "Using dynamic time warping to find patterns in time series." KDD workshop. Vol. 10. No. 16. 1994.

[20] Bagnall, Anthony, et al. "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances." Data Mining and Knowledge Discovery 31.3 (2017): 606-660.

[21] Zheng, Yi, et al. "Time series classification using multi-channels deep convolutional neural networks." International Conference on Web-Age Information Management. Springer, Cham, 2014.

[22] Cui, Zhicheng, Wenlin Chen, and Yixin Chen. "Multi-scale convolutional neural networks for time series classification." arXiv preprint arXiv:1603.06995 (2016).

[23] Wang, Zhiguang, Weizhong Yan, and Tim Oates. "Time series classification from scratch with deep neural networks: A strong baseline." 2017 International joint conference on neural networks (IJCNN). IEEE, 2017.

[24] Wang, Zhiguang, and Tim Oates. "Encoding time series as images for visual inspection and classification using tiled convolutional neural networks." Workshops at the twenty-ninth AAAI conference on artificial intelligence. Vol. 1. 2015.

[25] Pelletier, Charlotte, Geoffrey I. Webb, and François Petitjean. "Temporal convolutional neural network for the classification of satellite image time series." Remote Sensing 11.5 (2019): 523.

[26] Karim, Fazle, et al. "LSTM fully convolutional networks for time series classification." IEEE access 6 (2017): 1662-1669.

[27] Qin, Yao, et al. "A dual-stage attention-based recurrent neural network for time series prediction." Proceedings of the 26th International Joint Conference on Artificial Intelligence. 2017.

[28] Song, Huan, et al. "Attend and diagnose: Clinical time series analysis using attention models." 32nd AAAI Conference on Artificial Intelligence, AAAI 2018. AAAI press, 2018.

[29] Laptev, Nikolay, Saeed Amizadeh, and Ian Flint. "Generic and scalable framework for automated time-series anomaly detection." Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. 2015.

[30] Ren, Hansheng, et al. "Time-Series Anomaly Detection Service at Microsoft." Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019.

[31] Buda, Teodora Sandra, Bora Caglayan, and Haytham Assem. "Deepad: A generic framework based on deep learning for time series anomaly detection." Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, Cham, 2018.

[32] Zhu, Lingxue, and Nikolay Laptev. "Deep and confident prediction for time series at uber." 2017 IEEE International Conference on Data Mining Workshops (ICDMW). IEEE, 2017.

[33] Laptev, Nikolay, et al. "Time-series extreme event forecasting with neural networks at uber." International Conference on Machine Learning. Vol. 34. 2017.

[34] Taylor, Sean J., and Benjamin Letham. "Forecasting at scale." The American Statistician 72.1 (2018): 37-45.

[35] Žunić, Emir. "Application of Facebook's Prophet Algorithm for Successful Sales Forecasting Based on Real-world Data." International Journal of Computer Science & Information Technology (IJCSIT) Vol 12 (2020).

[36] Shen, Lifeng, Qianli Ma, and Sen Li. "End-to-end time series imputation via residual short paths." Asian Conference on Machine Learning. 2018.

[37] Memisevic, Roland, and Geoffrey Hinton. "Unsupervised learning of image transformations." 2007 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2007.

[38] Oreshkin, Boris N., et al. "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting." International Conference on Learning Representations. 2019.

[39] Oreshkin, Boris N., et al. "Meta-learning framework with applications to zero-shot time-series forecasting." arXiv preprint arXiv:2002.02887 (2020).