Comparing Boosting and Deep Learning Methods on Multivariate Time Series for Retail Demand Forecasting

Georgios Theodoridis, Athanasios Tsadiras

Comparing Boosting & Deep Learning methods on Multivariate Time Series for Retail Demand Forecasting

Georgios Theodoridis and Athanasios Tsadiras

Aristotle University of Thessaloniki, Greece
ttgeorgios@econ.auth.gr & tsadiras@econ.auth.gr

Abstract. Retail demand forecasting is an inherently complex problem as many different time-related factors as well as the correlation of demands in between each and every retail product have to be taken into account. In technical terms, retail demand forecasting is a multivariate timeseries forecasting problem where every single timeseries has to be not only analyzed but also predicted. Hence, added complexity is introduced necessitating the use of advanced methods with machine/deep learning backgrounds. Boosting models, such as XGBoost and LightGBM, are perfect choices with extensive bibliographic background and have been widely used to tackle multivariate timeseries forecasts. Simultaneously, recent advancements in deep neural networks have introduced new promising architectures that are yet to be applied on many different scenarios. Therefore, within this paper, two of those architectures with different core components are introduced, analyzed and applied. The Temporal Convolutional Network based on Convolution and the Temporal Fusion Transformer based on the Transformer architecture, which uses self-attention, are compared to boosting methods as well as standard statistical approaches, namely Exponential Smoothing and Seasonal ARIMA. The results indicate that the deep learning networks are the better choices contributing to the notion that deep learning has extraordinary capabilities in relation to large scale, complex and noisy data and that the aforementioned newly adopted designs are excellent choices for multivariate timeseries forecasting.

Keywords: Time Series Forecasting, Multivariate Forecasting, Retail Demand Forecasting, Boosting Models, Deep Neural Networks.

1 Introduction

Demand forecasting is a major challenge for retailers as it is the input for many operational decisions, especially for perishable goods with a high deterioration rate [11]. Retailers offer a vast variety of products via many different stores. Consequently, numerous forecasts are necessary daily to predict the demand of every item in every store. Therefore, it is of great importance to introduce methods that, regardless of scale and complexity, perform accurate forecasts.

Time Series Forecasting (TSF) is the fundamental process of predicting future values after analyzing chronologically ordered past values and is applied in many different fields such as energy consumption [3], financial analysis [17], sales forecasting

[18], anomaly detection [19], database optimization [5, 13] etc. TSF problems are inherently complex as there are time-related factors that must be taken into account; trend, seasonality and correlation of values based on time-distance [20].

Classical statistic models focus on univariate TSF, meaning the prediction of the future values of a single variable by observing the past values of that variable alone. But in today's reality of big data and especially in relation to retail demand, multivariate predictions are nothing sort of a necessity, hence the usage of complex machine learning models and, recently, deep learning techniques is required. Multivariate TSF may also be categorized in two main classes:

1. The usage of multiple past variables to predict one future variable, for example the prediction of electricity prices based on past electricity prices as well as past temperatures, past gas prices etc. We can define this category as many-to-one.
2. The usage of multiple past variables to predict multiple future variables, for example predicting the future sales of 5 products based on the past sales of all 5 products interchangeably. We can define this category as many-to-many.

The focus of this paper are multivariate TSFs of the latter category as retail demand forecasting is a many-to-many multivariate TSF. The contributions offered by the current paper are the following:

- Bibliographic enrichment of recently proposed DNNs with promising architectures.
- A comparison of the aforementioned DNNs with well established Boosting methods that are often considered the golden standard in machine learning practices.
- A benchmark comparison using popular statistical methods (ETS, ARIMA) that have, historically, been the default approaches in many different scenarios, including retail demand forecasting.

2 Background

Multivariate TSF is characterized by nonlinearity and difficulties in detecting overall trends and seasonality [28]. Therefore, many specialized models of different mathematical background have been developed to tackle the complexity of multivariate problems.

An initial and intuitive approach is to treat TSF as a generic regression problem whilst converting "time" into input features and using well known machine learning models as predictors such as Support Vector Regression (SVR) [22], Random Forest [9] and Gradient Boosting models like LightGBM [25]. The inherent advantage of this approach is the preestablished research and bibliographic robustness of said regression models. On the contrary, large multivariate datasets may include complex nonlinear correlations that are tedious to detect by equating TSF to general regression and simply applying generic regression models without tweaking their architecture.

Meanwhile, deep neural network approaches that were initially designed to solve image and/or language recognition problems have now gained popularity in general

TSF problems. Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) are mathematically great choices offering models such as Long-Short Term Memory (LSTM) [10] and Convolutional LSTM (ConvLSTM) [24]. These models are able to detect complex dependencies as they are designed to learn from both short- and long-term time correlations. A potential disadvantage is handling aperiodic and noisy data as these models depend on the existence of seasonality. Recently, to resolve the aforementioned issues, a new network architecture has been proposed, the Temporal Convolutional Network (TCN) [2]. It is described as straightforward in its convolution architecture and has the ability to create both deep and wide networks that are more resilient to aperiodic inputs.

Current advancements in hardware technology, mainly GPUs, reveal the capability and benefits of running deep learning processes in parallel. RNN's are unable to parallelize their internal calculations as they, by design, read and produce predictions in sequence. Hence, the Transformer model [27] was proposed that depends on multi-head self-attention which can easily be parallelized. Since then, many different novel designs based on self-attention have surfaced including the Temporal Fusion Transformer (TFT) [16] which improves the original concept by allowing the use of known past and future covariates as inputs and focusing on predicting multiple outputs.

3 Model Analysis

In the following paragraphs an overview of every model studied within this paper will be presented, focusing more on Deep Neural Networks (DNNs) and their architecture.

3.1 Multivariate Boosting

XGBoost (eXtreme Gradient Boosting) is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable [4]. It implements parallel tree boosting (also known as GBDT, GBM) under the Gradient Boosting ensemble learning framework. LightGBM is another boosting framework that uses tree-based learning algorithms. It is considered a predecessor to the XGBoost framework that achieves faster training speed and higher efficiency [15]. LightGBM uses histogram-based algorithms [14] which bucket continuous feature values into discrete bins and grows trees leaf-wise (best-first) [23]. It is currently considered one of the standard machine learning frameworks and has been heavily analyzed [12, 1].

In terms of multivariate TSF, both boosting algorithms use time dependent covariates such as numerical values representing the day of the week, month, year etc. or boolean values revealing if the current day is a weekday, holiday etc. to explain the time axis as well as the values of each series as input. By design, the output of the regressor is one dependent variable, therefore this approach can only solve a many-to-one multivariate TSF. To forecast many-to-many time series, n number of separate predictors need to be implemented where n equals the number of time series in the current problem. Simply put, instead of solving the initial problem directly, this method solves numerous many-to-one problems. In fact, via this process, every general

regression machine learning model may be used to solve many-to-many multivariate TSFs.

A seemingly obvious disadvantage is the huge number of predictors necessary for larger scale TSFs but, in reality, both the execution time is faster and the size in memory is lower compared to complex DNNs that solve the problem directly.

3.2 TCN

A Temporal Convolutional Network (TCN) [2] addresses TSF problems with the usage of convolution. At its core, TCN applies multiple dilated, causal 1D convolutions on its input to create an equally sized output. A convolution is causal when an output at time t is convolved only with elements from time t and earlier. For univariate TSF, 1D (one dimensional) convolutions are straightforward to explain; the only dimension is time with every timestep being assigned a single value. One would assume that multivariate TSF problems would require 2D convolutional layers and technically they are equivalent to a 2D convolution with kernel size (k, n), where n is static and equal to the number of timeseries, but they are still 1D in the sense that the window only moves along a single axis, the time axis, and the kernel size is essentially just k.

The dilated convolution of a 1D sequence $x \in R^n$ and a filter $f : \{0, \dots, k-1\} \rightarrow R$ is defined as the operation F on element s of the sequence so that:

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-d \cdot i} \quad (1)$$

where d is the dilation factor, k is the kernel size and $s - d \cdot i$ is the direction in the past. Essentially, dilation adds a fixed step in between every two consecutive filter taps.

To actually train and forecast via convolution, as the output has to be equal to the input, the network will equate its output to the values of the timeseries shifted forward by the forecast window, hence the tail of said output is the forecast.

The architecture of the current TCN follows that of [2] and shall be broken down in three levels as visualized in Fig. 1:

1. The network consists of sequential Residual blocks that contain a branch leading out to a series of transformations $\mathcal{F}$, whose outputs are added to the input x of the block:

$$o = Activation(x + \mathcal{F}(x)) \quad (2)$$

Every block has a dilation factor that increases exponentially (base b) per block. The number of blocks m is decided automatically so that full history coverage can be achieved, meaning that every output value has been affected by every input value.

2. Every Residual block consists of 2 Convolutional layers that are followed by a potential weight normalization, a ReLU activation to achieve non-linearity and a spatial dropout layer. Assuming input x with n timeseries (meaning depth of n), the number of filters f, used during convolution on it, will change the depth of the out-

put to f. More specifically, the first layer of the first block will have an input depth n and an output depth f. During the element-wise addition as defined in (2) the shapes of x and $\mathcal{F}(x)$ will be incompatible. For that reason, a 1×1 Convolution is applied on x to adjust its dimensions accordingly. Similarly, the last convolution layer of the last block will receive an input of depth f and it has to output the forecast, therefore a tensor of depth n. The ReLU activation is skipped during that last output as the forecast is allowed to contain negative values.

3. The Convolutional layers of every block apply filters with kernel size k dilated by d.

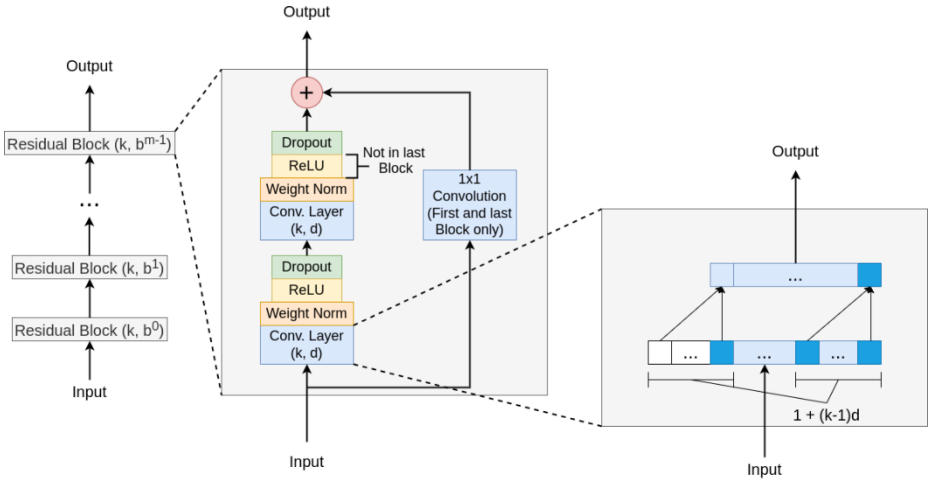

Fig. 1. The TCN architecture in three levels with kernel size as k, dilation exponent base b, Residual block number m and dilation factor d.

3.3 TFT

The Temporal Fusion Transformer [16] is a DNN designed to learn both long- and short-term relationships from time-varying inputs. It employs a sequence-to-sequence layer of LSTMs to learn short-term correlations and a multi-head attention block for long-term dependencies. Gating mechanisms are also present to skip potentially useless parts of the network, as well as variable selection networks that, for multivariate inputs, assign weights to each timeseries for a given timestep. The input accepted by TFT is not only past values with potential covariates but also future known values - future covariates. The prediction performed by TFT is probabilistic, using a quantile forecast to asses best-worst case scenarios which are extremely practical in application. If such analysis is not desired, as within this study, the 50% quantile is outputted. Fig. 2 visualizes the TSF architecture based on [16] but with the lack of Static Encoders as Static Metadata are not examined.

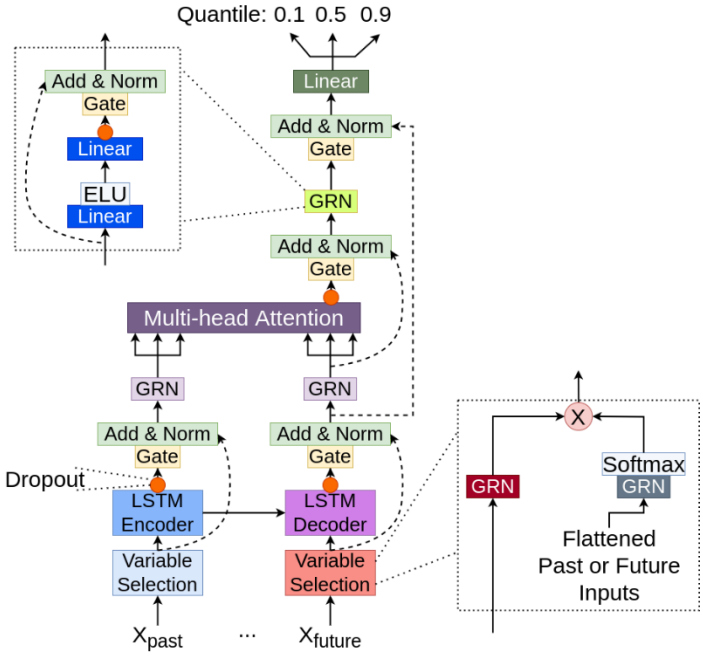

Fig. 2. The TFT architecture.

The Gated Residual Network (GRN) of the TFT gives the model the flexibility to apply non-linear processing only where needed. Given input a:

$$GRN_{\omega}(a) = LayerNorm(a + GLU_{\omega}(\eta)) \tag{3}$$

where GLU is the gate, ω denotes weight sharing as weight is shared throughout the GRN and $\eta \in R^{d_{model}}$ (d_{model} is the hidden state size which is common across TSF) is the end result of two Linear transformations with an ELU [6] activation function in between. The GLU is then defined as:

$$GLU_{\omega}(\eta) = \sigma\left(W_{3,\omega}\,\eta + b_{3,\omega}\right) \odot \left(W_{4,\omega}\,\eta + b_{4,\omega}\right) \tag{4}$$

where the weights and biases are indexed starting with 3 as 1 and 2 are used in the Linear transformation, $\odot$ is element-wise Hadamard product. This gate acts as a cut-off as it may output values close to 0 and based on (3) only the initial input will move on.

Within the variable selection network, the weights are calculated based on the flattened vector of all the past or future inputs (depending on encoding/decoding), meaning all the values of every timeseries for every past/future timestep.

Attention, in general, is defined as the scaling of values $V \in R^{N \times d_V}$ based on relationships between keys $K \in R^{N \times d_{attn}}$ and queries $Q \in R^{N \times d_{attn}}$. The scaled dot-product attention, which is a common choice [27], is:

$$Attention(Q, K, V) = Softmax(QK^T / \sqrt{d_{attn}}) \times V \tag{5}$$

In multi-head attention, multiple copies of the attention module are used in parallel. Each head captures different relationships between the input values. To be able to extract feature importance and interpret the predictions, TFT uses a modified multi-head attention that shares values in each head, and employs additive aggregation of all heads [16].

4 Performance Evaluations

4.1 The Dataset

To evaluate the aforementioned TFS methods within the retail demand forecasting field, a sales dataset is sourced from [26]. It contains the daily sales of 50 items in 10 different stores over the timespan of 5 years. Therefore, in terms of timeseries analysis, the dataset can be deconstructed as a multivariate set of 500 timeseries, the sales of every item for every store. The forecasting problem is predicting future sales based on the historical sales of every item, hence a many-to-many multivariate forecast. To properly stress test the models and investigate their ability to withhold long term correlations, a relatively large history window is provided to every model, a year, so as to forecast 30 timesteps (days) in the future. The sales statistics for every timeseries is displayed in Table 1. It's important to note that the values have been 0-1 scaled and the timeseries are split in train, validation and test sets so that the train timespan is 3 years and both the validation and the test set are 1 year.

Table 1. Basic statistic values for every sale in every store.

Metric	Values
Mean	0.23
Std	0.12
25% quantile	0.13
Median	0.20
75% quantile	0.30

The 75% percentile is just 0.3 while the max is by design 1. This might indicate the existence of numerous outliers or the fact that some items in some stores might sell in way bigger quantities which does make logical sense, for example milk vs bicycles. To investigate this, the boxplot of the mean sales of every item-store combination (the 500 timeseries) is graphed in Fig. 3. It seems that, even though more popular items do exist, on average the sales are consistent throughout without outlier mean values. Subsequently, we can conclude that the timeseries do have sale spikes individually. Moreover, a reasonable thought is that those spikes might happen during specific days or months. Fig. 4 displays the boxplots per weekday (a) and month (b). The results indicate that sale spikes aren't weekday or month dependent ergo they are possibly outliers that will be challenging to detect. This adds an extra layer of difficulty to our forecasts as it will test the resiliency of each method to noise. To assist each model in

its endeavor to handle said noise, a single covariate timeseries is added that explicitly states, for every day within the timespan, the weekday, the month, the year and if the current day is some form of holiday based on the US holiday calendar as the dataset originates from there.

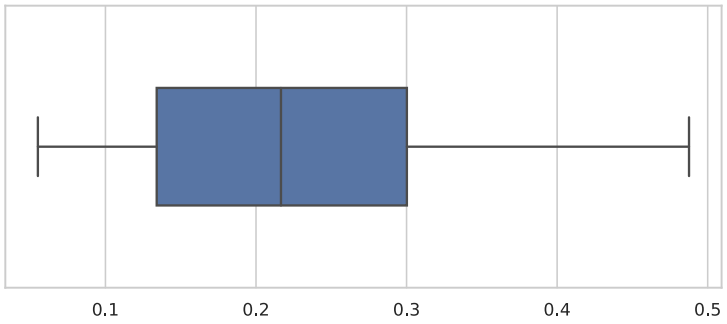

Fig. 3. Boxplot of mean values of every timeseries.

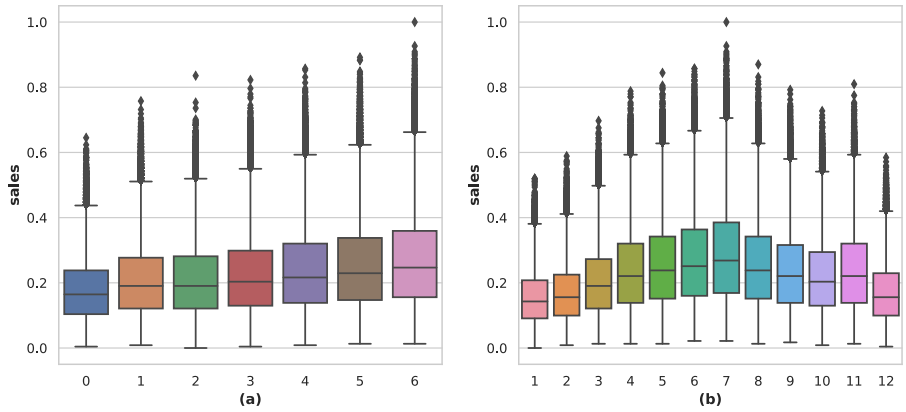

Fig. 4. Boxplot of sales per weekday (a) and month (b) for every item in every store.

4.2 Model Fine-tuning and Training

Each model previously introduced has a list of potential hyperparameters that can be fine-tuned to optimize a forecasting score. Fine-tuning is performed via repeated train-validation.

At this point it's important to note some computational costs. Whilst the LightGBM and XGBoost models compute quite fast, the same cannot be said for the TCN and TFT models. To yield results within reasonable timespans on a single Nvidia Tesla P100 GPU some maximums need to be set and they have to be applied on every model so as to preserve fairness of comparison. Ideally, every model is trained and validated on the entirety of the training and validation set accordingly. To reduce these exhaustive training/validation iterations, a maximum number of 50 iterations is

set. The algorithm will pick a random valid starting point (same seed for all models) within the timespans and start training/validating from that point on. Then this process is repeated 49 more times. Another limitation is the usage of random search within the hyperparameter space, which is limited to 100 iterations. The maximum number of estimators for boosting algorithms and epochs for neural networks is set to be 400 and early stopping is applied every 50 rounds to return the best performing model.

Table 2. Hyperparameters used and selected (**bold**) during fine-tuning.

XGBoost	LightGBM
The dropout rate: 0, **0.1**, 0.3.	The dropout rate: 0, **0.1**, 0.3.
The learning rate: 0.001, **0.01**, 0.1.	The learning rate: 0.001, 0.01, **0.1**.
The hessian: 0.5, **1**, 2.	The hessian: **0.5**, 1, 2.
The gamma: 0.01, 0.1, **1**.	The number of leaves: 100, **200**, 300.
The maximum depth: 5, 10, 20, **None**.	The maximum depth: 5, 10, 20, **None**.
TCN	**TFT**
The dropout rate: **0**, 0.1, 0.3.	The dropout rate: **0**, 0.1, 0.3.
The learning rate: **0.001**, 0.01, 0.1.	The learning rate: **0.001**, 0.01, 0.1.
The kernel size: 2, **3**, 4.	The size of the state: **8**, 16, 32.
The number of filters: 2, 3, **4**.	The number of LSTM layers: 1, **2**, 3.
The dilation base: **2**, 3, 4.	The number of attention heads: 2, **4**, 8.
The use of weight norm.: True, **False**.	Multi-head attention on both encoder and decoder: **True**, False.

The hyperparameters of each model are presented in Table 2 with optimal in bold. The search minimizes the MSE loss. The optimizers used during training are not part of the hyperparameters; DART [21] is used for boosting and Adam [8] for the neural networks. The batch size is also kept static and equal to 64.

4.3 Results

After fine-tuning each model, they are tested on the test set. In addition to every method used, an Exponential Smoothing (ETS) as well as a Seasonal ARIMA (SARIMA – via the Auto-ARIMA method [7]) are also performed in a univariate way; for every single timeseries the models are fine-tuned and then tested. This process sets a more advanced benchmark compared to possible naïve methods such as a typical moving average as well as portraying how effective classical statistical models, that are heavily used to this day, really are in a multivariate scenario. Table 3 presents the results using multiple metrics and for each metric the mean score (top) as well as the median score (bottom) are displayed as the models forecast multiple times for all 500 timeseries.

Table 3. Forecasting results of each method based on different metrics (mean on top, median on bottom).

Method (rank) / Metrics	ETS (6)	SARIMA (5)	XGBoost (4)	LightGBM (3)	TCN (2)	**TFT (1)**
MAE	0.031	0.028	0.024	0.022	0.021	**0.021**
	0.030	0.026	0.023	0.022	0.021	**0.021**
sMAPE	20.16	18.36	16.26	15.48	14.67	**14.38**
	19.38	17.35	15.37	14.06	13.60	**13.41**
MSE	17	13	9	8	7	**7**
$(\times 10^{-4})$	14	11	8	7	6	**6**
RMSE	0.039	0.034	0.030	0.027	0.026	**0.026**
	0.038	0.033	0.029	0.026	0.026	**0.025**
R^2	-0.423	-0.232	0.101	0.210	0.289	**0.320**
	-0.402	-0.071	0.140	0.276	0.339	**0.349**

After inspecting the results, it is clear that both boosting and DNN methods are superior to the standard statistical methods. It should be noted that multivariate versions of ARIMA do exist but, as an attempt was made to apply them on the current TSF problem, it was quickly revealed that they were extremely inefficient for the large number of timeseries present. Consequently, the usage of more advanced algorithms for bigger scale multivariate TSF is nothing sort of warranted. By comparing boosting methods to DNNs, the results are in favor of the neural networks. With TFT having the best results and TCN being a close second, the future of these architectures but also DNNs in general seems bright for multivariate TSF problems. As per their design, the neural network approaches are able to extract dependencies and correlations even in extremely large, complex and noisy timeseries. That said, their training and, more importantly, fine-tuning is marginally slower and resource heavier than the boosting counterparts. Using a single Nvidia Tesla P100 GPU, the boosting methods were able to be fine-tuned in under two hours whilst the DNNs needed more than six. Nonetheless, both TFT and TCN are excellent predictors and the better approach for the current retail demand forecasting problem.

5 Conclusions

Throughout this paper, the retail demand forecasting problem is addressed. Retail demand forecasting is essentially a multivariate timeseries forecasting problem that introduces large numbers of timeseries that are mutually dependent.

Within this scope, newly suggested DNN architectures have been introduced, analyzed, tuned, trained and tested. At the same time, highly praised and frequently used machine learning regressors based on boosting methods are also tuned, trained and tested.

The results indicate that neural networks and deep learning is an excellent approach to multivariate TSF and by extension retail demand forecasting. Both the

Temporal Fusion Transformer and the Temporal Convolutional Network surpass XGBoost and LightGBM as well as staple statistical methods, namely Exponential Smoothing and Seasonal ARIMA.

The main conclusion is that deep learning techniques are exceptionally accurate even under the complexity of multiple noisy timeseries. New and improved designs, such as TFT and TCN, are able to overthrow the current machine learning standards in multivariate TSF. Secondly, the boosting algorithms are a great alternative when resources are limited yielding great overall results. Lastly, statistical methods seem to fall behind and, as the number of data increases worldwide, their usage might seem obsolete. They are a great tool for quickly analyzing any given timeseries but as soon as more variables are introduced with increased complexity, more advanced models should be considered instead.

As for future work, new DDN preprocessing methods may be introduced to denoise the data, potentially perform timesries classification to inspect which items are highly correlated and consequently tweak the architecture of the neural networks so as to process said information to yield better results.

Acknowledgements. The authors would like to acknowledge the support provided by the IT Center of the Aristotle University of Thessaloniki (AUTh) throughout the progress of this research work.

References

1. Al Daoud, E.: Comparison between XGBoost, LightGBM and CatBoost using a home credit dataset. International Journal of Computer and Information Engineering, 13(1), 6-10 (2019).
2. Bai, S., Kolter, J. Z., & Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271 (2018).
3. C. Deb, F. Zhang, J. Yang, S. E. Lee and K. W. Shah.: A review on time series forecasting techniques for building energy consumption. Renew. and Sust. Energ. Rev. 74, 902 – 924 (2017).
4. Chen, Tianqi., Guestrin, Carlos.: XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794 (2016).
5. D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang.: Automatic database management system tuning through large-scale machine learning. In Proceedings of the 2017 ACM International Conference on Management of Data, pages 1009–1024. ACM (2017).
6. D.-A. Clevert, T. Unterthiner, S. Hochreiter.: Fast and accurate deep network learning by exponential linear units (ELUs), in: ICLR (2016).
7. Dhamo, E., & Puka, L.: Using the R-package to forecast time series: ARIMA models and Application. In INTERNATIONAL CONFERENCE Economic & Social Challenges and Problems (2010).
8. Diederik P. Kingma, Jimmy Lei Ba.: ADAM: A method for stochastic optimization. Published as a conference paper at ICLR 2015 (2015).
9. Hamidi, O., Tapak, L., Abbasi, H., Abbasi, H., Maryanaji, Z.: Application of random forest time series, support vector regression and multivariate adaptive regression splines models in prediction of snowfall (a case study of Alvand in the middle Zagros, Iran). Theor. Appl. Climatol. 134, 769–776 (2018).

10. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Comput., 9, 1735–1780 (1997).
11. Huber, J., & Stuckenschmidt, H.: Daily retail demand forecasting using machine learning with emphasis on calendric special days. International Journal of Forecasting, 36(4), 1420-1438 (2020).
12. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y.: Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems, 30 (2017).
13. L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon.: Query-based workload forecasting for self-driving database management systems. In SIGMOD, pages 631–645. ACM (2018).
14. Li, Ping, Qiang Wu, Christopher J. Burges. Mcrank: Learning to rank using multiple classi-fication and gradient boosting. Advances in Neural Information Processing Systems 20 (2008).
15. LightGBM's documentation, https://lightgbm.readthedocs.io/, last accessed 2022/2/1
16. Lim, B., Arık, S. Ö., Loeff, N., & Pfister, T.: Temporal fusion transformers for interpretable multi-horizon time series forecasting. International Journal of Forecasting, 37(4), 1748-1764 (2021).
17. M. Chen and B. Chen.: A hybrid fuzzy time series model based on granular computing for stock price forecasting, Inf. Sciences 294, 227 – 241 (2015).
18. M. H. Rafiei and H. Adeli.: A novel machine learning model for estimation of sale prices of real estate units, Journal of Construction Engineering and Management 142, 04015066, (2015).
19. M. Paolanti, D. Liciotti, R. Pietrini, A. Mancini and E. Frontoni.: Online detection of stealthy false data injection attacks in power system state estimation. IEEE Trans. on Smart Grid 9(3), 1636–1646 (2018).
20. P. Lara-Benítez, M. Carranza-García and J. C. Riquelme.: An experimental review on deep learning architectures for time series forecasting. International Journal of Neural Systems 31(03), 2130001 (2021).
21. Rashmi, K. V., Gilad-Bachrach, R.: DART: Dropouts meet Multiple Additive Regression Trees, http://arxiv.org/abs/1505.01866 (2015).
22. Sapankevych, N., Sankar, R.: Time Series Prediction Using Support Vector Machines: A Survey. IEEE Comput. Intell. Mag., 4, 24–38 (2009)
23. Shi, Haijian.: Best-first decision tree learning. The University of Waikato, (2007).
24. Shi, X., Chen, Z., Wang, H., Yeung, DY., Wong, W.K., Woo, W.C.: Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. In Proceedings of the Neural Information Processing Systems Conference, Montreal, QC, Canada, 802–810 (2015).
25. Sun, X., Liu, M., & Sima, Z.: A novel cryptocurrency price trend forecasting model based on LightGBM. Finance Research Letters, 32, 101084 (2020).
26. The Item Demand Forecasting Dataset on Kaggle, https://www.kaggle.com/c/demand-forecasting-kernels-only/data, last accessed 2022/2/1
27. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I.: Attention is all you need. Advances in neural information processing systems, 30 (2017).
28. Wan, R., Mei, S., Wang, J., Liu, M., & Yang, F.: Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting. Electronics, 8(8), 876 (2019).