Time Series Forecasting Overview

Domen Mohorčič
Faculty of Computer and
Information Science
University of Ljubljana
Email: dm4492@student.uni-lj.si

***Abstract*—Time series is a special type of data, collected at even time intervals for longer periods of time. Time series forecasting is the task of predicting those sequences into the future. The task is important for commercial and environmental reasons, because with accurate predictions we can prepare for the forecasted scenario and either take its full advantage or mitigate it. A number of approaches have been proposed for time series forecasting. The earliest ones were simple linear autoregression models, but lately a lot of different approaches have been proposed: matrix factorization, transformers, deep, recurent, convolutional, and graph neural networks. We replicate the results from different papers, compare different models on the same dataset, and confirm that the current state-of-the-art model is indeed the best at time series forecasting.**

I. Introduction

Time series is a sequence data, collected at even time intervals. It is ordered and usually has multiple real-valued observations for each timestamp. Time series forecasting (TSF) is a task of predicting the next values in such series. It is important in the fields of climate modeling, medicine, commercial decision making, and finance. With climate modeling [6] we can predict and analyze the average temperature increase over time and forecast possible climate events that would have a huge impact on our way of living. The most common climate forecasting is weather forecasting. We use it every day to dress appropriately. Financial firms use TSF to predict when and where markets will move. With a good model they can make a lot of money by knowing when to buy and when to sell. In medicine TSF can be used for seizure prediction [2] and helping patients before they have one. And in commercial decision making the TSF can be used to predict the sales of a specific item. If the predicted sales go up, the company can prepare for it by having more stock to ensure they will not run out of specific item.

TSF task can be divided into categories based on two different criteria: forecast horizon and number of variables predicted.

The forecast horizon divides the problem into two categories: single- and multi-horizon forecasting. In single-horizon forecasting the model predicts only one step at a time. Multi-horizon forecasting tries to predict multiple steps into the future. Even though single-horizon forecasting methods are more accurate than multi-horizon ones, multi-horizon forecasting is more valuable. Decision makers use multi-horizon forecasting as it allows them to make better and more informed decisions as it gives more context and provides possible trends in the future. A single-horizon method can be converted into a multi-horizon one with iteration, but will be less accurate than a direct multi-horizon method. Because of all the advantages of multi-horizon over single-horizon forecasting, multi-horizon forecasting is the ultimate goal of TSF.

Number of variables predicted is the next criteria. It divides TSF problem into two categories: univariate and multivariate. Univariate TSF focuses on predicting one variable, while multivariate TSF tries to predict multiple variables at the same time. The multivariate methods take into account multiple variables, but can also predict only one variable, if others are known ahead of time. Multivariate methods are preferred, since we can get better predictions if we can learn the correlation between multiple variables. Another problem here is the availability of variables. If we predict only one variable but base our prediction on many, we need to have values for these variables in the future. If those values do not exist at the time of prediction, a multivariate model that predicts all variables is used, but can be less accurate. Univariate models have no such problems, as they base their next predictions automatically on previously forecasted values.

The aim of this report is to make an overview of the TSF in the last decade. In Chapter III we take a closer look at different models. In Chapter IV we take a quick look at most popular TSF datasets and metrics and in Chapter V we test the models. In this report we will focus on multi-horizon univariate forecasting models.

II. Related work

The prediction of time series began in the previous century. The first methods used were Autoregression (AR) and Moving Average (MA). AR and MA models use linear combination of previous observations and depend on the domain expertise. In 1968 Box, Jenkins, and MacGregor combined these two methods into Autoregressive Integrated Moving Average (ARIMA) model [1]. The Equation 1 represents the ARIMA model, where X_t is the prediction at time t, ϵ_t is the error at time t, and α and β are just linear coefficients. It can accurately forecast for short periods of time and tends to converge to a single value after long forecasting periods. The ARIMA model is a univariate single-horizon method and will be used as a baseline.

$$X_t - \alpha_1 X_{t-1} - \cdots - \alpha_p X_{t-p} = \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} \quad (1)$$

In the recent years a lot of different approaches have been tried. In 2016 Yu, Rao, and Dhillon proposed the use of matrix factorization technique called Temporal Regularized Matrix Factorization (TRMF) in paper [14]. The method was developed with intentions to be highly scalable, and can also deal with highly noisy or missing data. It builds on the idea to use graph regularization for matrix factorization (MF), but uses well-known time series models to describe temporal dependencies and as a regularization. It was tested against other MF methods and performed significantly better. It also proved to be very good for missing data imputation.

In year 2018 Lai et al. proposed a neural network called LSTNet in [3]. The network is a combination of recurrent neural network (RNN) and a convolutional neural network (CNN). It uses convolution to extract short-term patterns in the data. Recurrent components capture periodic patterns, and the whole model incorporates auto-regressive model in parallel to keep the linear component. The method was tested against other models on four different standard datasets and performed better than all of them.

The DeepAR model was designed and proposed by Salinas, Flunkert, and Gasthaus in 2019 in [9]. It is a method for producing probabilistic forecasts, and it is an auto-regressive RNN. It was build on the ideas from paper [10], posted in 2014, where authors Sutskever, Vinyals, and Le introduced a sequence to sequence RNN model for translation purposes. DeepAR model uses inputs as well as last predictions to forecast values. When compared to other models it performed better, but on some datasets yielded comparable results.

In 2017 a novel transformer approach was introduced by Vaswani et al. in [11]. They proposed a model which relies on self-attention rather than a reference window and uses encoder-decoder structure to generate text. In theory self-attention provides an infinite window length, meaning the model can always find relevant text it encountered somewhere before and generate text with relevant context. In 2020 researchers Lim et al. in [4] proposed a temporal fusion transformer, a model based on the original transformer, but for TSF purposes. They called it log-sparse transformer. It uses recurrent layers for local processing and multi-head self-attention layers for long-term dependencies to create multi-horizon predictions. The model performed better than all of its predecessors.

A year later, in 2021, Informer was introduced by Zhou et al. in [15] as another upgrade to the original transformer, which had a few issues: quadratic time complexity, memory bottleneck for long inputs, and slow prediction times for long outputs. Informer implemented probsparse self-attention to reduce learning time complexity from $O(n^2)$ to $O(nlogn)$. It also has smaller feature map, which enabled faster calculation of predictions. Informer performed better than other models both on univariate and multivariate predictions.

In 2020 researchers Wu et al. proposed a novel graph neural network (GNN) approach for TSF and presented it in [13]. GNN assumes that state of a node depends on the state of its neighbors, so a graph structure with predefined relationships between variables was required in models, known as spatial-temporal GNN. Wu et al. proposed a MTGNN model where each node is a variable and the model learns connections and the graph structure by itself. The model performed better than other GNN models for single-horizon forecasting and comparable for multi-horizon forecasting.

In 2020 researchers Oreshkin et al. presented a deep neural network (DNN) called N-Beats in [8]. It is unique in a way that it uses only fully connected layers and nothing else. It consists of building blocks that output a forecast and a backcast to help future building blocks with forecasting. The model was designed to be interpretable, so it uses periodic and small degree polynomial functions as mapping functions, which mimic seasonality and trend. The model was compared to other M competition [12] models and consistently performed better.

Liu et al. revisited convolutional neural networks (CNN) in late 2021 and proposed a new model, called SCINet, in [5]. SCINet model is a hierarchical model that captures temporal dependencies at multiple resolutions. The model splits the data into even and odd and learns trends on them separately. This procedure is repeated recursively, so the model learns from small and local to large and global trends. The model is currently the state-of-the-art and preformed better at all standard TSF datasets.

III. Models

In this section we will take a look at some of the above mentioned methods in detail. We will describe in detail the workings of TRMF, DeepAR, Informer, MTGNN, and SCINet.

A. *TRMF*

TRMF method is a MF model. It was proposed to handle large number of observations through longer periods of time. It was designed to handle climate data, where there are potentially thousands of sensors measuring every hour over a decade. Some sensors can break, so the data can have missing values. Normal models cannot deal with missing data, so some other preprocessing must be performed.

A natural way of modeling high dimensional data is in the form of a matrix, where rows are sensors and columns correspond to time points. The MF method tries to factorize input matrix into two smaller matrices, $Y = FX$. The TRMF model tries to minimize the expression in Equation 2, where Ω is a set of observed entries at time t and variable i and $Y_{i,t}$ is the observation. Matrices X and F represent time-varying variables and features for each time series respectively. λ represents regularization coefficient and R represents regularization function.

$$\sum_{(i,t)\in\Omega} (Y_{i,t} - f_i^T x_t)^2 + \lambda_f R_f(F) + \lambda_x R_x(X) \tag{2}$$

In standard MF squared Frobenius norm is used as a regularization. It assumes no dependencies among variables, so it is not suitable. Another technique is graph regularization for temporal dependencies, where we specify dependency pattern with a lag set L such that all edges of distance $l \in L$ share the same weight w_l. In such a graph nodes represent the observations. The graph regularization is defined in Equation 3, where G represents graph and μ represents regularization coefficient.

$$R(X \mid G, \mu) = \frac{1}{2} \sum_{l \in L} \sum_{t; t>l} w_l (x_t - x_{t-l})^2 + \frac{\mu}{2} \sum_t \|x_t\|^2 \quad (3)$$

It has few problems, since $w_t = 0$ would minimize the regularization. Authors took the ideas from it and proposed to use a TSF model M to describe dependencies. Such model is described in Equation 4, where Θ represents parameters of the model and ϵ_t are the residuals.

$$x_t = M_\Theta(x_{t-l} : l \in L) + \epsilon_t \quad (4)$$

The new regularizer thus encourages the structure, introduced by M, and the resulting regularizing function is in Equation 5. $T_M(X \mid \Theta)$ is the negative log likelihood of observing a particular value at time t given model M and its parameters Θ.

$$R_x(X) = T_M(X \mid \Theta) = -logP(x_1, \ldots, x_T \mid \Theta) \quad (5)$$

In Equation 2 the regularization term for X is substituted with Equation 5. We treat Θ as another set of variables, and the Equation 2 can be solved by an alternating minimization over F, X, and Θ. The forecasting is done by using model M to predict future latent embedding of input and use F to obtain forecasts.

B. *DeepAR*

The DeepAR model is a RNN autoregression model that makes probabilistic forecasts. The model is also a sequence-to-sequence model, meaning minimal feature engineering is required to capture the group behaviour. It makes probabilistic forecasts based on Monte Carlo estimations that are consistent throughout time. The model also has the ability to forecast time series that it has not seen during training, as it infers its predictions by learning from similar variables.

The goal of DeepAR is to model the conditional distribution $P(z_{i,t_0}, \ldots, z_{i,t_T} \mid z_{i,1}, \ldots, z_{i,t_0-1}, x_{i,1}, \ldots, x_{i,T})$, where $z_{i,t}$ are values of time series i at time t and $x_{i,t}$ are covariates. The model distribution is given by a product of likelihood factors for each time t, given by the Equation 6. $h_{i,t}$ is the value of the network at time t, Θ represents model parameters, and θ is a function of the network input.

$$Q_\Theta(z_{i,t_0:T} \mid z_{i,1:t_0-1}, x_{i,1:T}) = \Pi_{t=t_0}^{T} l(z_{i,t} \mid \theta(h_{i,t}, \Theta)) \quad (6)$$

The likelihood model l determines the noise model of the data and should be chosen to match the statistical properties of the data. DeepAR model predicts all parameters of θ for the next time point, so the noise model can change over time. With Gaussian likelihood model, which is parametrized with mean μ and variance σ^2, the likelihood function becomes the probability density function of normal distribution. The parameters of the model can be learned by maximizing the sum of the log-likelihood function.

The covariates $x_{i,t}$ can be item-dependent, time-dependent, or both. They are used to provide additional information about the item or the time point to the model. The covariates are standardized to have zero mean and unit variance.

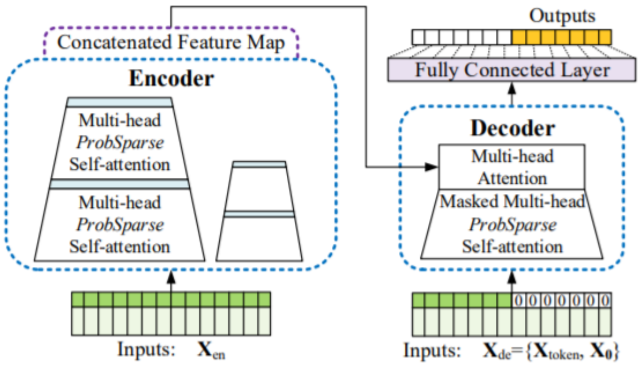
Concatenated Feature Map
Encoder
Multi-head ProbSparse Self-attention
Multi-head ProbSparse Self-attention
Inputs: X_{en}
Outputs
Fully Connected Layer
Decoder
Multi-head Attention
Masked Multi-head ProbSparse Self-attention
Inputs: $X_{de}=\{X_{token}, X_0\}$

Fig. 1: The architecture of the Informer model. The encoder learns feature representation while the decoder takes care of forecasting.

C. *Informer*

The Informer model is a version of the transformer model, which was proposed for natural language processing tasks. The architecture of the Informer model can be seen on Figure 1. The model was designed to operate on long sequences, which is better for TSF problem, as we can forecast further in the future. The transformer uses an encoder-decoder architecture and introduces self-attention for long sequence analysis, but is inefficient due to quadratic time complexity. Some newer models proposed sparse and log-sparse self-attention, which solved time complexity problem.

The advantage of Informer is prob-sparse self-attention. The original self-attention, proposed by transformer paper, is presented in Equation 7, where Q is the query, K is the key, V is the value, and d is the input dimension. The query's attention is defined in Equation 8, where the expression in the first sum represents the probability of key based on some query $p(k_j \mid q_i)$. The self-attention combines the values and makes outputs based on computing that probability.

$$A(Q, K, V) = Softmax\left(\frac{QK^t}{\sqrt{d}} V\right) \quad (7)$$

$$A(q_i, K, V) = \sum_j \frac{k(q_i, k_j)}{\sum_l k(q_i, k_l)} \quad (8)$$

If $p(k_j \mid q_i)$ approaches uniform distribution, the self-attention becomes redundant. We can use Kullback-Lieber divergence to distinguish between distributions and find important queries. The result is query's sparsity measurement, M. The higher the value of M, the more important the query is. The proposed prob-sparse self-attention is presented in

Equation 9, where $\overline{Q}$ is a sparse matrix that contains only top queries under the sparsity measurement.

$$A(Q, K, V) = Softmax\left(\frac{\overline{Q}K^t}{\sqrt{d}}V\right) \tag{9}$$

As a consequence of prob-sparse self-attention, the feature map has redundant values of V. Authors use distilling, which suppresses redundant values and makes a more focused feature map.

The decoder and forecasting part of the model takes the input sequence as input instead of starting token like in NLP tasks. The decoder predicts outputs by one forward procedure. This generative style decoding allows the Informer model to make predictions in linear time complexity, which is great improvement over previous transformer models.

D. MTGNN

The MTGNN model is a GNN multivariate model. A typical GNN model relies heavily on a predefined graph structure, which assumes the dependencies between variables are known. Another problem is during learning: GNN models tend to optimize message passing while not changing the underlying graph structure. The authors of MTGNN model propose a graph learning layer, which learns the directed graph structure from provided data and is able to optimize it during learning.

The MTGNN model consists of a graph learning layer, graph convolution modules, temporal convolution modules, and an output module. The graph learning layer learns the graph adjacency matrix iteratively to learn relationships among time sequences. The proposed adjacency matrix is unidirectional since time-varying variables may not influence each other symmetrically. The matrix is calculated by sub-graph training, where in each iteration nodes are grouped randomly and the algorithm learns the sub-graph structure. This gives all nodes in a large graph an equal chance to connect. The random sampling of nodes for sub-graph learning is also used to reduce memory and time complexity.

The graph convolution module aims to combine node's information with the information of its neighbors. For that the authors propose a mix-hop propagation layer, which consists of two steps: information propagation and information selection. The information propagation formula can be seen in Equation 10, where $H^{(k)}$ represents hidden states at step k, A is the weight matrix, and β is the coefficient for retaining original states. The information selection formula can be seen in Equation 11, where W represents the weights and acts as a feature selector.

$$H^{(k)} = \beta H_{in} + (1 - \beta)AH^{(k-1)} \tag{10}$$

$$H_{out} = \sum_{i=0}^{K} H^{(k)}W^{(k)} \tag{11}$$

The temporal convolution module applies a set of convolution filters to extract temporal features. The module uses two dilated inception layers, one uses tangent hyperbolic activation function and acts as a filter and the other uses a sigmoid activation function and controls the amount of information the filter can pass to the next module.

Before the output module another layer is added to standardize the information. The output module consists of two standard convolution layers that have the number of outputs equal to forecasting horizon.

E. SCINet

The SCINet model is short for Sample Convolution and Interaction Networks model. Its main idea is extracting and exchanging information at different temporal resolutions. The model is build out of SCI-Blocks, which downsample the input data into two sub-sequences and extracts features using convolution filters from both of them. Subsampling is performed to learn both long- and short-term dependencies.

SCINet builds on the idea of dilated causal convolution. It was first proposed for generating raw audio in WaveNet in [7]. Dilated causal convolution layers use a stack of convolution layers with exponentially enlarged dilation factors. We can think of it as a convolution where filters are applied locally. Dilated convolutions enable models to have larger lookback windows with fewer convolution layers. The architecture of SCINet can be seen on Figure 2.

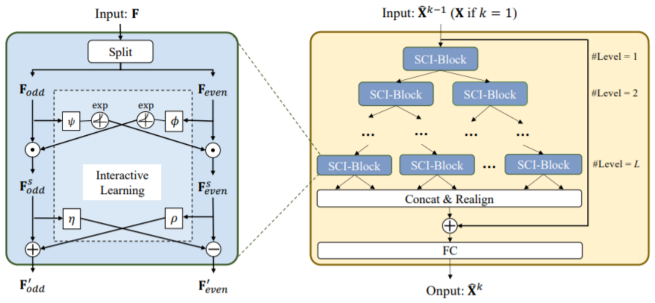

Fig. 2: The architecture of SCINet. On the left is a SCI-Block, and on the right is the SCINet model. We can see that the input of a SCI-Block gets separated and processed interdependently. The imput gets separated multiple times through the model which enables the learning on different resolutions.

The SCINet model is hierarchical and is build using SCI-Blocks. Each SCI-Block splits the data into F_{odd} and F_{even}. It uses different kernels to learn features from each sequence separately. To compensate for possible information loss while downsampling authors propose interactive learning strategy. F_{odd} and F_{even} are projected to hidden states with two different 1D convolution functions, ψ and ϕ. Then they are transformed using exponential function and perform element-wise product with swapped odd and even sequences, as shown in Equation 12.

$$\begin{aligned} F^s_{odd} &= F_{odd} \cdot exp(\psi(F_{even})) \\ F^s_{even} &= F_{even} \cdot exp(\phi(F_{odd})) \end{aligned} \tag{12}$$

Dataset	T	D	L
Electricity	26,304	321	1 hour
Traffic	17,544	862	1 hour
Solar	52,560	137	10 minutes
Exchange	7,588	8	1 day

TABLE I: Description of datasets. T represents number of timestamps, D the number of variables, and L sample rate of data.

Next, the two scaled feature vectors are again projected to another set of hidden states with two different 1D convolution functions, ρ and η, and are added or subtracted from F^s_{odd} and F^s_{even} to get output sequences, $F^{'}_{odd}$ and $F^{'}_{even}$. The entire process is repeated recursively until we get a tree-like structure. Authors report that the best results are obtained with fewer than five levels of SCI-Blocks.

The output of the bottom SCI-Blocks is concatenated by reversing the splitting operation and then added to the original sequence. A fully connected layer is used to output final predictions. The authors also propose that the output of one SCINet could be fed into another to achieve better prediction accuracy.

IV. Data and Metrices

In TSF there exist a few popular datasets for models to be tested on. Most used ones were introduced by Lai et al. in [3]. The datasets are electricity, traffic, solar, and exchange. Their descriptions are available in Table I.

The electricity dataset includes the kWh consumption of electricity for 321 clients from years 2011 to 2014. Data points reflect hourly consumption. Traffic dataset describes road occupancy rates for 862 roads in San Francisco Bay area, California, USA. Data was collected for 48 months and has granularity of 1 hour. Solar energy dataset measures solar production rates for 137 solar power plants in Alabama, USA. The data was collected every 10 minutes in 2006. Final common dataset is exchange, which provides daily exchange rates for 8 different currencies from 1990 to 2016.

Another newer dataset was introduced by [15]. The Electricity Transformer Temperature (ETT) dataset includes one target variable, oil temperature, and has six other variables, denoting power load values. The data contains two years of values, measured at 15 minute intervals, on two electricity transformers in China. The dataset has three variations, ETTh1, ETTh2, ETTm1. h1 and h2 versions have sequences with 1 and 2 hour difference respectively. The m1 version has 15 minute sample rate.

The most common metrics used in TSF are mean absolute error (MAE), mean square error (MSE), normalized deviation (ND), root mean square error (RMSE), normalized root mean square error (NRMSE), root relative square error (RSE), and empirical correlation coefficient (CORR). The equations for each metric can be seen in Equation 13.

$$\begin{aligned}
MAE &= \frac{1}{N}\sum_{i=1}^{N}|y_i - f(x_i)| \\
MSE &= \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2 \\
ND &= \frac{\sum_{i=1}^{N}|y_i - f(x_i)|}{\sum_{i=1}^{N}|y_i|} \\
RMSE &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2} \\
NRMSE &= \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2}}{\frac{1}{N}\sum_{i=1}^{N}|y_i|} \\
RSE &= \frac{\sqrt{\sum_{i=1}^{N}(y_i - f(x_i))^2}}{\sqrt{\sum_{i=1}^{N}(y_i - \overline{y})^2}} \\
CORR &= \frac{1}{N}\sum_{j=0}^{N}\frac{\sum_{i=0}^{d}(y_{i,j} - \overline{y})(x_{i,j} - \overline{x})}{\sum_{i=0}^{d}(y_{i,j} - \overline{y})^2(x_{i,j} - \overline{x})^2}
\end{aligned} \tag{13}$$

V. Results

In this section we will present the results from evaluating models against the reported metrics. The evaluated models are TRMF, DeepAR, and SCINet. ARIMA model is used in the final comparison between models.

Authors of **TRMF** model performed evaluation on electricity, traffic, and synthetic datasets. For the synthetic dataset the authors randomly generated matrix F and x_i following predefined model with lags. True values are obtained by $y_t = Fx_t + \epsilon$, where $\epsilon \sim N(0, 0.01)$. The electricity and traffic datasets were first standardized. The authors used ND and NRMSE metrics for forecasting evaluation. Their results compared to mine are in Table II.

	Syntetic		Electricity		Traffic	
	ND	NRMSE	ND	NRMSE	ND	NRMSE
TRMF	0.373	0.487	0,255	1,397	0.187	0.423
Reproduced	0.330	0.443	0.562	0.941	0.57	0.618

TABLE II: Reproduced results for TRMF model.

We can see that for syntetic dataset out results were a little bit better than the authors, but that was probably because the data was randomly generated and the authors didn't provide reproducible code. But we were unable to reproduce the results on the electricity and traffic datasets. Our experiments achieved higher ND score, but lower NRMSE score for electricity dataset. On the traffic dataset our evaluations performed worse. Even though the datasets were both standardized, the authors newer explicitly stated their train and test splits. We used the same prediction horizon and lookback window, but had problems with using the same lags as authors. We used $[1, 24]$ as lags instead of $[1, \dots, 24, 168, \dots, 191]$ because the combined gradient was too large and hte regularization parameters could not be set low enough. We performed rolling

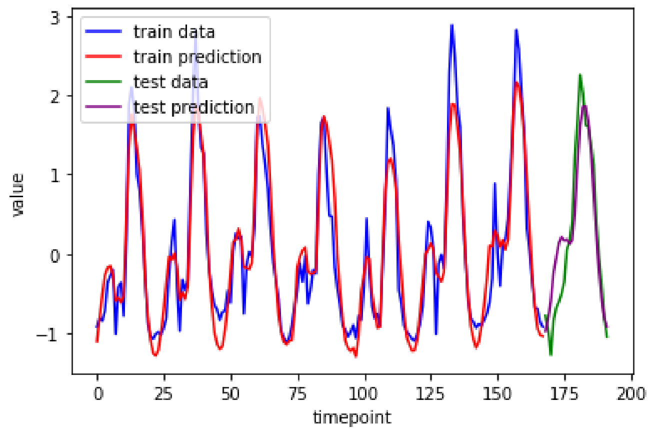

Fig. 3: Performance of TRMF model on random client in electricity dataset. We can see that the model can capture trends and even forecast with relative percision.

cross validation on the last two years of data, but we cannot be certain that the authors did the same. On Figure 3 we can see an example forecast for TRMF model on electricity dataset for a random client. The model learned the features of the data and made a good prediction, which follows the real trend.

Authors of **DeepAR** trained and tested their model on five datasets: three public and two private. The public datasets were parts, traffic, and electricity, and the two private datasets were ec and ec-sub, weekly item sales from Amazon. We had access only to traffic and electricity datasets, so we tested the DeepAR model on them. They used the same metrics as authors of TRMF, except they replaced the NRMSE with RMSE. Their results compared to mine are in Table III.

	Electricity		Traffic	
	ND	RMSE	ND	RMSE
DeepAr	0.07	1.00	0,17	0,42
Reproduced	0.369	0.445	0.669	0.988

TABLE III: Reproduced results for DeepAR model.

The reproduced results for electricity dataset were a bit unusual. Reproduced ND score was higher than the one reported by the authors, but the reproduced RMSE score was much lower. The results for traffic dataset are worse as well, but at least both metrics are worse by a factor. Higher metric values may be due to training time. The authors state that training on electricity and traffic datasets took 7 and 3 hours respectively, while ours took around 10 minutes. We used the same network structure, but limited training to 200 epochs with early stopping if results have not improved for 20 epochs. The authors did not mention such training parameters.

When looking deeper into the article we noticed that they used slightly different RMSE formula, and their naming of metrics is not consistent. In the text they mention normalized RMSE or NRMSE and present a modified NRMSE formula, but they use RMSE when presenting results. If we calculate NRMSE for each dataset, we get much lower values than the ones presented by authors.

The authors of **SCINet** model tested their model on all previously described datasets, but included only ETT results in their published paper. Unpublished version of the paper includes results for electricity, exchange, traffic, and solar datasets, which we tried to replicate. For their purposes authors chose metrics RSE and CORR. In Table IV we can see reproduced results for electricity, solar, and exchange datasets. We did not reproduce traffic dataset due to memory constraints.

	Electricity		Solar		Exchange	
	RSE	CORR	RSE	CORR	RSE	CORR
SCINet	0.0957	0.9276	0.4032	0.9413	0.0426	0.9591
Reproduced	0.1041	0.9218	0.4079	0.9112	0.0445	0.9273

TABLE IV: Reproduced results for SCINet model.

We can see that we were able to reproduce all of their results with a reasonable margin of error. The reproduced scores are a little bit worse than the ones authors presented, but are close. Such results are expected, as authors trained their model for 100 epochs while we trained it only for 30.

All of the previously tested models were also trained on ETT dataset to compare them directly, since different authors used different metrics. We used the ETTh1 dataset, which has granularity of 1 hour. The authors of SCINet were kind enough to test DeepAR and ARIMA on ETTh1 dataset, so we can compare our findings to theirs. For baseline we use the ARIMA model. The DeepAR and TRMF models have the same architecture as the authors used for their testing. The models were tested with univariate forecasting, except for TRMF, which requires multiple time series sequences. The TRMF model was evalueated only on the one sequence other models were tested on. The reproduced results can be seen on Table V. The results the authors of SCINet provided are in Table VI.

	Horizon	24	48	168	336	720
ARIMA	MAE	0.324	0.248	0.240	**0.201**	**0.303**
	MSE	0.155	0.098	0.087	**0.065**	**0.143**
TRMF	MAE	1.043	0.937	0.900	0.899	1.080
	MSE	1.141	0.940	0.879	0.860	1.254
DeepAR	MAE	0.146	0.195	0.238	0.246	0,343
	MSE	0.036	0.062	0.088	0.092	0.168
SCINet	MAE	**0.134**	**0.170**	**0.222**	0.253	0.348
	MSE	**0.031**	**0.050**	**0.082**	0.102	0.181

TABLE V: Results on ETTh1 dataset. The best results are in **bold**.

	Horizon	24	48	168	336	720
ARIMA	MAE	0.284	0.424	0.504	0.593	0.766
	MSE	0.108	0.175	0.396	0.468	0.659
DeepAR	MAE	0.280	0.327	0.422	0.552	0.707
	MSE	0.107	0.162	0.239	0.445	0.658
SCINet	MAE	0.132	0.173	0.222	0.242	0.343
	MSE	0.031	0.051	0.081	0.094	0.176

TABLE VI: Results on ETTh1 dataset from the authors of SCINet.

If we first take a look at SCINet's authors results at Table VI we can see that SCINet is the clear winner, with DeepAR

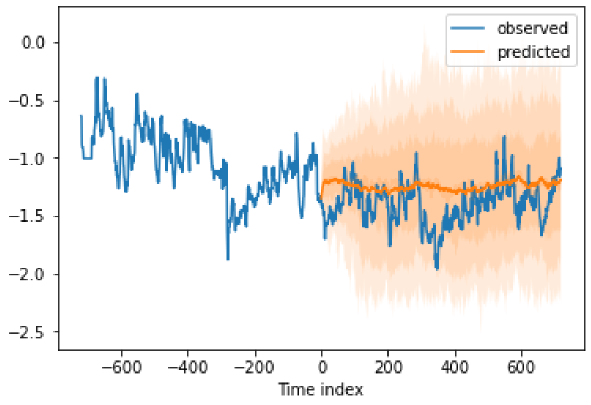

Fig. 4: Performance of DeepAR model on OT variable in ETTh1 dataset. Since DeepAR is a probabilistic mode, we can see how the prediction distribution changes through time, but does not get larger, meaning the model is consistent with future predictions.

taking second place and ARIMA third. Since ARIMA is the most simple of the three models, the results are expected. We can also see that the MAE and MSE scores are higher for longer forecasting windows.

If we take a look at our results at Table V the scores are different. Not much has changed for SCINet compared to their reported scores, but the other models could not be replicated. DeepAR model performed twice as better as authors of SCINet reported it to be. This is strange since our MAE metric for DeepAR is constantly around half the reported values of SCINet paper. The forecast of the DeepAR model for a horizon of 720 can be seen of Figure 4, where we can see that the model has low variance. The ARIMA model has the highest difference from reported values and has scored the best for longer horizons, which is unexpected. It was also unexpected that the results do not change much when we increase the forecasting horizon. Upon closer inspection we realised that longer forecasts for ARIMA tend to converge to one number, which in our case was around the average of the testing sequence and thus produced nearly constant MAE and MSE scores regardless of forecasting horizon length. The TRMF model is the only one which was multivariate, but we focused only on target variable. The model produces consistent scores with MAE and MSE metrics like ARIMA, and upon inspection we saw that it could not replicate the input sequence like it did when forecasting on electricity dataset. It also had the same problem as ARIMA, where it forecasted a constant value, which happened to be around the average of the testing sequence.

At the end the best model on ETTh1 dataset is SCINet, DeepAR placed second, ARIMA third, and TRMF fourth. DeepAR took second place because it performed much better for shorter horizons than ARIMA. Since SCINet is currently state-of-the-art its dominance was expected.

VI. Conclusion

The aim of this report was to present the state of TSF and currently used models. We presented in detail matrix factorization, transformer, and other deep models. We presented the most popular datasets and metrics used for TSF. We tried to replicate the results from three different papers and were largely unsuccessful. When comparing the models on the same dataset the results were expected, and we confirmed that SCINet is current state-of-the-art model for TSF.

References

[1] G. E. P. Box and G. M. Jenkins. “Some Recent Advances in Forecasting and Control”. In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 17.2 (1968), pp. 91–109. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2985674.

[2] Christian Elger and Klaus Lehnertz. “Seizure prediction by non-linear time series analysis of brain electrical activity”. In: *The European journal of neuroscience* 10 (Mar. 1998), pp. 786–9. DOI: 10.1046/j.1460-9568.1998.00090.x.

[3] Guokun Lai et al. *Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks*. 2018. arXiv: 1703.07015 [cs.LG].

[4] Shiyang Li et al. *Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting*. 2020. arXiv: 1907.00235 [cs.LG].

[5] Minhao Liu et al. “Time Series is a Special Sequence: Forecasting with Sample Convolution and Interaction”. In: *CoRR* abs/2106.09305 (2021). arXiv: 2106.09305. URL: https://arxiv.org/abs/2106.09305.

[6] Manfred Mudelsee. “Trend analysis of climate time series: A review of methods”. In: *Earth-Science Reviews* 190 (2019), pp. 310–322. ISSN: 0012-8252. DOI: https://doi.org/10.1016/j.earscirev.2018.12.005. URL: https://www.sciencedirect.com/science/article/pii/S0012825218303726.

[7] Aaron van den Oord et al. *WaveNet: A Generative Model for Raw Audio*. 2016. arXiv: 1609.03499 [cs.SD].

[8] Boris N. Oreshkin et al. *N-BEATS: Neural basis expansion analysis for interpretable time series forecasting*. 2020. arXiv: 1905.10437 [cs.LG].

[9] David Salinas, Valentin Flunkert, and Jan Gasthaus. *DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks*. 2019. arXiv: 1704.04110 [cs.AI].

[10] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: 1409.3215 [cs.CL].

[11] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[12] Vipul et al. *Makridakis Competitions*. [Online; accessed 27.12.2021]. 2021. URL: https://en.wikipedia.org/wiki/Makridakis_Competitions.

[13] Zonghan Wu et al. *Connecting the Dots: Multivariate Time Series Forecasting with Graph Neural Networks*. 2020. arXiv: 2005.11650 [cs.LG].

[14] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. "Temporal Regularized Matrix Factorization for High-dimensional Time Series Prediction". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/85422afb467e9456013a2a51d4dff702-Paper.pdf.

[15] Haoyi Zhou et al. *Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting*. 2021. arXiv: 2012.07436 [cs.LG].