Sparsity-invariant Convolution for Forecasting Irregularly Sampled Time Series

Krisztian Buza[1,2][0000−0002−7111−6452]

[1] Artificial Intelligence Laboratory
Institute Jozef Stefan
Ljubljana, Slovenia
[2] BioIntelligence Group, Department of Mathematics-Informatics
Sapientia Hungarian University of Transylvania
Targu Mures, Romania
buza@biointelligence.hu

Abstract. Time series forecasting techniques range from ARIMA over exponential smoothing to neural approaches, such as convolutional neural networks. However, most of them were designed to work with regularly sampled and complete time series, i.e., time series which can be represented as a sequence of numbers *without* missing values. In contrast, we consider the task of forecasting *irregularly* sampled time series in this paper. We argue that, compared with "usual" convolution, sparsity-invariant convolution is better suited for the case of irregularly sampled time series, therefore, we propose to use neural networks with sparsity-invariant convolution. We perform experiments on 30 publicly-available real-world time series datasets and show that sparsity-invariant convolution significantly improves the performance of convolutional neural networks in case of forecasting irregularly sampled time series. In order to support reproduction, independent validation and follow-up works, we made our implementation (software code) publicly available at https://github.com/kr7/timeseriesforecast-siconv .

Keywords: time series forecasting · convolutional neural network · sparsity-invariant convolution

1 Introduction

Due to its prominent applications in medicine [9], retail [15], finance [10] and other domains [4], time series forecasting has been a key area of research. Time series forecasting techniques range from the well-known autoregressive models [2] over exponential smoothing [8] to approaches based on deep learning [12], [19]. Among the numerous methods, a prominent family of methods include forecast techniques based on convolutional neural networks (CNNs) [1], [16].

In ideal cases, under the assumption that a reliable sensor is installed, measurements are made continuously or periodically with constant time between subsequent observations. In such cases, time series can be represented as sequences of numbers without missing values. However, in real-world applications,

sensors are not fully reliable due to various reasons ranging from the capacity of the battery over weather conditions to hardware failures and signal transmission issues. In other cases (e.g. measuring the blood pressure at the family doctor), the measurements are done semi-regularly, thus the time between consecutive observations varies. Consequently, many real-world time series, for example in healthcare [22], finance [17] and meteorology [14], are irregularly sampled.

The vast majority of conventional time series forecasting techniques, including methods based on deep learning, consider time series as sequences of numbers *without* missing values. This corresponds to the assumption that the time series are regularly sampled and complete. Lim and Zohren [12] point out that "deep neural networks typically require time series to be discretised at regular intervals, making it difficult to forecast datasets where observations can be missing or arrive at random intervals." They also note that the domain of irregularly sampled time series is understudied.

In this paper we propose to use convolutional neural networks (CNNs) with sparsity-invariant convolution for forecasting irregularly sampled time series. In experiments on 30 publicly available real-world datasets from various domains, we show that CNNs with sparsity invariant convolution outperform "usual" CNNs, both in terms of mean squared error and mean average error, when it comes to forecasting irregularly sampled time series.

The reminder of the paper is organized as follows. In Section 2, we provide a short discussion of related works. We describe our approach in Section 3, followed by the experimental evaluation is Section 4. Finally, we conclude in Section 5.

2 Related Work

Convolutional neural networks have been widely used for classification and forecasting of time series, see e.g. [1], [3], [6]. Works that are most closely related to ours fall into two categories: (i) methods based on convolutional neural networks for time series forecasting and (ii) approaches based on sparsity-invariant convolution.

As for the works in the former category, we refer to the recent surveys of Lim et al. [12], Sezer et al. [17] and Torres et al. [19] and we point out that our approach is orthogonal to such works in the sense that one could replace convolutional layers in any convolutional network by sparsity-invariant convolution if the data (or its hidden representation, i.e., the input of a convolutional layer within the network) is sparse.

Regarding the works on sparsity-invariant convolution, we note that the operation of sparsity-invariant convolution has originally been introduced in the depth completion (a.k.a. 3D reconstruction) community [20] where a few pixels of an image are associated with distance information, thus the distance information corresponds to sparse data [23]. While various models with sparsity-invariant convolution have been shown to outperform their counterparts with "usual" convolution, see e.g. [13], to the best of our knowledge, ours is the first work that studies sparsity-invariant convolution in the domain of time series.

3 Our Approach

We begin this section with a formal definition of our task followed by the description of sparsity-invariant convolution in the context of time series.

3.1 Problem Formulation

Given an observed time series $x = (x_1, \ldots, x_l)$ of length l, we aim at predicting its subsequent h values $y = (x_{l+1}, \ldots, x_{l+h})$. We say that h is the forecast horizon and y is the target. Furthermore, we assume that a dataset D is given which contains n time series with the corresponding target:

$$D = \{(x^{(i)}, y^{(i)})_{i=1}^{n}\}. \tag{1}$$

We use D to train neural networks for the aforementioned prediction task. We say that $x^{(i)}$ is the input of the neural network.

In our experiments, we assume that an independent dataset D^* is given which can be used to evaluate the predictions of our model. Similarly to D, dataset D^* contains pairs of input and target time series. D^* is called the test set.

The above sequence $x_1, \ldots, x_l$ corresponds to a regularly sampled time series. To account for the fact that time series may be irregularly sampled, we allow for missing values in the aforementioned time series, i.e., each x_i within $x = (x_1, \ldots, x_l)$ is either a real number or a symbol indicating that the value is missing. We assume that time series are represented at a relatively high resolution so that all the actual observations may be mapped to one of the symbols in the sequence $x_1, \ldots, x_l$, while most symbols of the sequence denote missing values. In real-world applications, the ratio of missing values may be as high as 90% or even more [5].

3.2 SiConv: Sparsity-invariant Convolution

Considering convolutional neural networks working with data containing missing values, it may be useful to encode the missing values by zeros because in this case the multiplication of a missing value (i.e., a zero) by the corresponding weight results in zero and therefore the missing values are ignored in the weighted sum calculated by the convolutional layer, which is an intuitive behaviour. Furthermore, treating missing values as zeros is also inline with dropout, a widely-used regularisation for deep neural networks [18]. For these reasons, we follow the wide-spread convention of denoting missing values by zeros [13], [20].

The intuition behind sparsity-invariant convolution [20] is to normalize the output of the convolutional layer according to the number of its non-missing inputs. As we focus on time series forecasting in this paper, we describe sparsity-invariant convolution, denoted as SiConv for simplicity, in the context of time series. Denoting the input of SiConv as $x^{in} = (x_1^{in}, \ldots, x_l^{in})$, the size of the

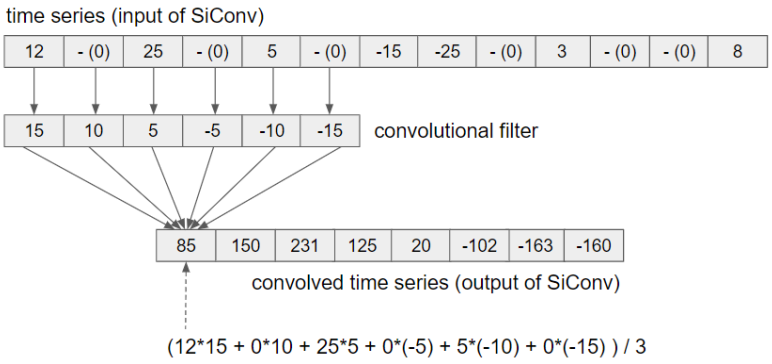

Fig. 1. Illustration of sparsity-invariant convolution (SiConv). The input and output time series of SiConv are shown in the top and bottom, respectively. The convolutional filter in the center is expected to detect a decreasing trend. Compared to "usual" convolution, the difference is that the output values are divided by the number of non-missing input values.

convolutional filter as s, the output $x^{out} = (x_1^{out}, \ldots, x_{l-s+1}^{out})$ of SiConv can be calculated as follows:

$$x_i^{out} = \begin{cases} \frac{b + \sum_{j=0}^{s} w_j x_{i+j}^{in}}{z_i} & \text{if } z_i \neq 0 \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

where

$$z_i = \sum_{j=0}^{s} \mathcal{I}(x_{i+j}^{in} \neq 0), \tag{3}$$

while b and w_j denote the bias and weights of the convolutional filter; $\mathcal{I}$ is an indicator function that returns 1 if its argument is true, otherwise it returns zero, therefore z_i is the number of the non-zero (i.e., non-missing) inputs of SiConv within the i-th convolutional window. SiConv is illustrated in Fig. 1.

We note that in case if there are no missing values in the input of SiConv, SiConv is equivalent to "usual" convolution up to the scaling factor $z_i = s$. On the other hand, for those segments of the time series, where all the inputs of SiConv are missing values (zeros) within the convolutional window, the output of SiConv is zero which denotes a missing value according to our encoding. For these reasons, we may use SiConv not only in the first convolutional layer, but in subsequent convolutional layers as well, especially in case of highly sparse input. In such cases, each convolutional layer with SiConv decreases the ratio of missing values in the hidden representation.

Representing irregularly sampled time series at a relatively high resolution results in sequences containing lot of missing values. In such cases, SiConv is more suited than "usual" convolution by its design. Therefore, we propose to use SiConv in neural networks for forecasting irregularly sampled time series.

4 Experimental Evaluation

The goal of our experiments is to examine the effect of SiConv on the forecast performance.

4.1 Experimental Settings

Datasets We performed experiments on 30 real-world time series datasets from various domains. These datasets are publicly available in *The UEA & UCR Time Series Classification Repository*.[3] The datasets used in our experiments are listed in the first column on Tab. 1 and Tab. 2. For all the datasets, we considered a forecast horizon of $h = 16$. We trained the models to predict the last h values of the time series based on its previous values.

Baselines In order to assess the contribution of SiConv, in all our experiments, we trained three versions of the same convolutional neural network: (i) with SiConv, (ii) with "usual" convolution and (iii) with "usual" convolution and linear interpolation of the missing values.

Experimental Protocol We performed experiments according to the 10-fold cross-validation protocol. That is: we initially partition the data into 10 splits.[4] Out of them, 9 splits are used as training data, while the remaining one is used as test data. The process is repeated 10-times: in each round of the cross-validation, a different split plays the role of the test set.

Evaluation Metrics We evaluated the predicted time series both in terms of mean squared error (MSE) and mean absolute error (MAE). MSE and MAE were calculated as follows:

$$MSE = \sum_{(x,y)\in D^*} \sum_{\substack{y_i \in y, \\ y_i \neq 0}} \frac{(\hat{y}_i - y_i)^2}{N} \tag{4}$$

$$MAE = \sum_{(x,y)\in D^*} \sum_{\substack{y_i \in y, \\ y_i \neq 0}} \frac{|\hat{y}_i - y_i|}{N} \tag{5}$$

where D^* denotes the test dataset, x is an instance of the test set and y is the corresponding target (roughly speaking: x contain the "past" values of the

[3] https://www.timeseriesclassification.com

[4] In our case, each *time series dataset* contains several *time series*. For example, the ECG5000 dataset contains in total 5000 time series, and each of these 5000 time series have a length of 140. In order to avoid data leakage [7], when partitioning data, an entire time series is assigned to one of the splits. For each time series belonging to the test split, we aim to predict its last h values. The segment we aim to predict is *unknown* to the model.

Table 1. Mean squared error (averaged over 10 folds) ± its standard deviation in case of our approach (SiCNN) and the baselines (CNN, linCNN) at a sparsity level of $SL = 80\%$. Lower values indicate better performance. For each dataset, the best approach is underlined. For SiCNN, we provide two symbols in the form of ∘/∘ which denote whether the difference between CNN and linCNN is statistically significant (•) or not (∘) according to paired t-test at significance level of $p = 0.01$.

Dataset	CNN	linCNN	SiCNN
Adiac	0.0220 ± 0.0036	0.1347 ± 0.0400	0.0224 ± 0.0034 ∘/•
ArrowHead	0.0397 ± 0.0122	0.3289 ± 0.1052	0.0376 ± 0.0092 ∘/•
BeetleFly	0.4711 ± 0.2805	0.4872 ± 0.2996	0.4671 ± 0.2687 ∘/∘
BirdChicken	0.3893 ± 0.2224	0.6731 ± 0.2880	0.3480 ± 0.1709 ∘/•
BME	0.0699 ± 0.0209	0.0586 ± 0.0163	0.0735 ± 0.0202 ∘/∘
CincECGTorso	0.0750 ± 0.0231	0.0612 ± 0.0195	0.0702 ± 0.0204 ∘/∘
DiatomSizeReduction	0.0133 ± 0.0040	0.7714 ± 0.1710	0.0144 ± 0.0050 ∘/•
ECG200	0.2918 ± 0.1194	0.2980 ± 0.1414	0.2780 ± 0.1069 ∘/∘
ECG5000	0.7821 ± 0.0671	1.9665 ± 0.1601	0.6645 ± 0.0414 •/•
ECGFiveDays	0.0371 ± 0.0054	0.0661 ± 0.0075	0.0305 ± 0.0039 •/•
FacesUCR	1.7111 ± 0.1581	1.7151 ± 0.1620	1.6909 ± 0.1076 ∘/∘
FiftyWords	0.1670 ± 0.0447	0.2988 ± 0.0684	0.1664 ± 0.0466 ∘/•
GunPoint	0.0675 ± 0.0381	0.3287 ± 0.1501	0.0763 ± 0.0387 ∘/•
Haptics	1.5011 ± 0.5241	20.124 ± 8.8588	1.3942 ± 0.5036 ∘/•
InlineSkate	0.1426 ± 0.0631	0.3934 ± 0.1382	0.1504 ± 0.0705 ∘/•
Lightning2	0.1598 ± 0.0837	0.3014 ± 0.0898	0.1511 ± 0.0703 ∘/•
Lightning7	0.4505 ± 0.2166	0.5815 ± 0.3696	0.4549 ± 0.2800 ∘/•
Mallat	0.0187 ± 0.0012	0.7913 ± 0.8176	0.0183 ± 0.0016 ∘/∘
MedicalImages	0.1324 ± 0.0403	0.1565 ± 0.0638	0.1240 ± 0.0410 ∘/∘
MoteStrain	0.7120 ± 0.0916	0.9816 ± 0.1341	0.6299 ± 0.0957 •/•
OSULeaf	0.3063 ± 0.0724	0.6940 ± 0.1614	0.2786 ± 0.0627 •/•
Phoneme	2.6407 ± 0.2944	2.5916 ± 0.5279	2.3227 ± 0.3087 •/∘
Plane	0.0932 ± 0.0435	0.3603 ± 0.1098	0.0896 ± 0.0479 ∘/•
PowerCons	1.6712 ± 0.3403	2.4377 ± 0.7643	1.5449 ± 0.3347 ∘/•
Symbols	0.1063 ± 0.0161	0.6541 ± 0.0592	0.0819 ± 0.0138 •/•
SwedishLeaf	0.1392 ± 0.0166	0.2674 ± 0.0670	0.1278 ± 0.0195 ∘/•
Trace	0.0113 ± 0.0027	0.3073 ± 0.0526	0.0085 ± 0.0023 •/•
TwoLeadECG	0.0441 ± 0.0059	0.1507 ± 0.0292	0.0336 ± 0.0041 •/•
WordSynonyms	0.6195 ± 0.0467	0.7266 ± 0.0846	0.4990 ± 0.0565 •/•
Worms	0.8910 ± 0.2081	1.3738 ± 0.4375	0.9053 ± 0.1951 ∘/•

Table 2. Mean absolute error (averaged over 10 folds) ± its standard deviation in case of our approach (SiCNN) and the baselines (CNN, linCNN) at a sparsity level of $SL = 80\%$. Lower values indicate better performance. For each dataset, the best approach is underlined. For SiCNN, we provide two symbols in the form of ∘/∘ which denote whether the difference between CNN and linCNN is statistically significant (•) or not (∘) according to paired t-test at significance level of $p = 0.01$.

Dataset	CNN	linCNN	SiCNN
Adiac	0.1107 ± 0.0094	0.2780 ± 0.0335	0.1124 ± 0.0073 ∘/•
ArrowHead	0.1438 ± 0.0222	0.4636 ± 0.1077	0.1430 ± 0.0178 ∘/•
BeetleFly	0.5735 ± 0.1761	0.5923 ± 0.1956	0.5652 ± 0.1853 ∘/∘
BirdChicken	0.4880 ± 0.1510	0.6739 ± 0.1666	0.4825 ± 0.1222 ∘/•
BME	0.1636 ± 0.0257	0.1631 ± 0.0194	0.1696 ± 0.0259 ∘/∘
CincECGTorso	0.1697 ± 0.0292	0.1716 ± 0.0205	0.1625 ± 0.0262 ∘/∘
DiatomSizeReduction	0.0865 ± 0.0069	0.8229 ± 0.1091	0.0902 ± 0.0097 ∘/•
ECG200	0.3903 ± 0.0642	0.3676 ± 0.0650	0.3691 ± 0.0589 ∘/∘
ECG5000	0.6298 ± 0.0270	1.0300 ± 0.0351	0.5746 ± 0.0183 •/•
ECGFiveDays	0.1461 ± 0.0111	0.2101 ± 0.0176	0.1326 ± 0.0090 •/•
FacesUCR	0.9919 ± 0.0399	1.0092 ± 0.0325	0.9893 ± 0.0221 ∘/∘
FiftyWords	0.2911 ± 0.0283	0.4238 ± 0.0563	0.2898 ± 0.0253 ∘/•
GunPoint	0.1772 ± 0.0396	0.4871 ± 0.1294	0.1860 ± 0.0345 ∘/•
Haptics	0.7583 ± 0.0634	3.2173 ± 0.8438	0.7309 ± 0.0630 ∘/•
InlineSkate	0.2500 ± 0.0328	0.4578 ± 0.1057	0.2574 ± 0.0432 ∘/•
Lightning2	0.2796 ± 0.0486	0.4604 ± 0.0716	0.2828 ± 0.0456 ∘/•
Lightning7	0.4437 ± 0.0935	0.5169 ± 0.1302	0.4418 ± 0.1008 ∘/•
Mallat	0.1068 ± 0.0034	0.7263 ± 0.3777	0.1061 ± 0.0043 ∘/•
MedicalImages	0.2402 ± 0.0214	0.2701 ± 0.0357	0.2340 ± 0.0176 ∘/•
MoteStrain	0.6083 ± 0.0314	0.7850 ± 0.0620	0.5651 ± 0.0331 •/•
OSULeaf	0.4249 ± 0.0572	0.6612 ± 0.0809	0.4140 ± 0.0491 ∘/•
Phoneme	1.1828 ± 0.0501	1.1544 ± 0.1294	1.0860 ± 0.0557 •/∘
Plane	0.2075 ± 0.0380	0.4073 ± 0.0633	0.2064 ± 0.0355 ∘/•
PowerCons	0.9842 ± 0.0896	1.1170 ± 0.2191	0.9371 ± 0.0848 ∘/∘
SwedishLeaf	0.2832 ± 0.0178	0.3940 ± 0.0537	0.2722 ± 0.0185 ∘/•
Symbols	0.2508 ± 0.0159	0.7049 ± 0.0399	0.2160 ± 0.0150 •/•
Trace	0.0842 ± 0.0110	0.5317 ± 0.0519	0.0743 ± 0.0116 ∘/•
TwoLeadECG	0.1631 ± 0.0082	0.3100 ± 0.0334	0.1426 ± 0.0075 •/•
WordSynonyms	0.5762 ± 0.0210	0.6537 ± 0.0505	0.5127 ± 0.0278 •/•
Worms	0.7596 ± 0.0846	0.9599 ± 0.1548	0.7687 ± 0.0805 ∘/•

time series, and y contains its "future" values), y_i is one of the values to be forecast (we assume that missing values are denoted by zeros) and $\hat{y}_i$ is the corresponding prediction of the model. N is the number of non-missing values in the test dataset:

$$N = \sum_{(x,y)\in D^*} \sum_{y_i \in y} \mathcal{I}(y_i \neq 0) \tag{6}$$

Both MSE and MAE were calculated in each of the 10 folds of the cross-validation. We report the average and standard deviation of MSE and MAE in Tab. 1 and Tab. 2.

We used paired t-test at significance level (p-value) of 0.01 in order to assess whether the observed differences between our approach SiCNN and its competitors, CNN and linCNN are statistically significant or not.

Implementation We implemented our neural networks in Python using the PyTorch framework. In order to support reproduction and follow-up works, we made our implementation publicly available in a github repository.[5] Our code can be executed in Google Collaboratory[6].

4.2 Experiments on Datasets from Various Domains

In order to assess the contribution of SiConv relative to "usual" convolution in various domains, first, we consider a simple convolutional network containing a single convolutional layer with 25 filters, followed by a max pooling layer with window size of 2, and a fully connected layer with 100 units. We set the size of convolutional filters to 9. The number of units in the output layer corresponds to the forecast horizon, as each unit is expected to predict one of the numeric values of the target time series. We trained the networks for 1000 epochs. We used mean squared error loss and the Adam optimizer [11] with a batch size of 16. As mentioned previously, we varied the type of convolutinal layer, therefore the variants of this simple convolutional neural network are denoted as SiCNN, CNN and linCNN with SiConv, "usual" convolution and "usual" convolution combined with linear interpolation of missing values, respectively.

As the time series of the aforementioned datasets do not contain missing values, we randomly selected 80% of the values of each time series and replaced them by missing values. This is meant by sparsity level $SL = 80\%$ in the caption of Tab. 1 and Tab. 2.

As one can see in Tab. 1 and Tab. 2, in the majority of the examined cases, SiCNN outperforms it counterparts with "usual" convolution. Moreover, in many cases SiCNN is statistically significantly better than CNN and linCNN. In particular, SiCNN significantly outperforms linCNN on 22 datasets in terms of MSE, and on 23 datasets in terms of MAE. Furthermore, SiCNN is significantly better than CNN on 9 and 7 datasets in terms of MSE and MAE, respectively. On the

[5] https://github.com/kr7/timeseriesforecast-siconv
[6] https://colab.research.google.com/

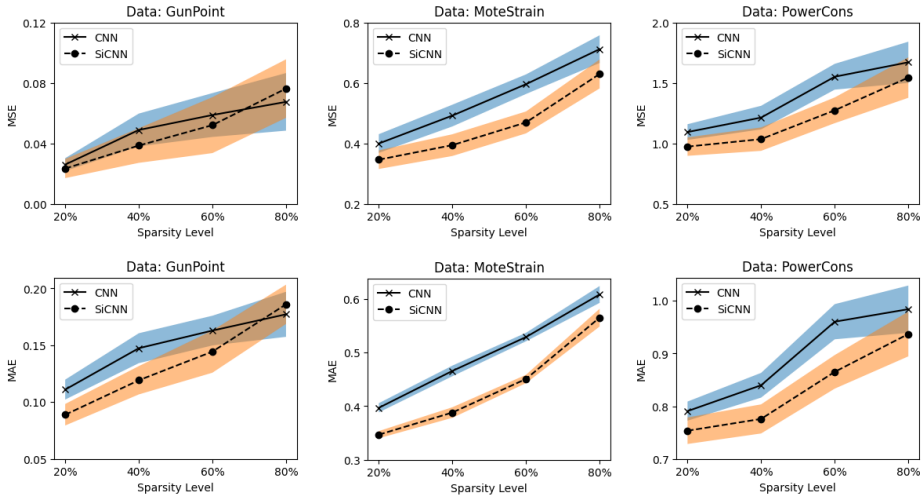

Fig. 2. MSE and MAE of our approach (SiCNN) and the baseline (CNN) as function of sparsity level. The width of the colored area corresponds to the standard deviation of MSE and MAE.

other hand, in those few cases when CNN or linCNN outperform SiCNN, the difference is not significant statistically.

It is interesting to note that CNN with linear interpolation of missing values worked worse that CNN without such an interpolation. This may be attributed to the fact that the actual trend between two observations is not linear and once the interpolation is performed, the network is not able to distinguish between true values and interpolated values.

4.3 The Effect of Sparsity Level

Next, we examine the performance of the simple convolutional network from the previous section in case of various levels of sparsity. For simplicity, we consider three datasets: GunPoint, MoteStrain and PowerCons. As one can see in Fig. 2, we observed similar trends in case of all the three datasets. In particular, with increasing level of sparsity, both the error of our approach (SiCNN) and that of the baseline (CNN) grows which is expected because it is inherently more difficult to predict future values of a time series in case if more of its values are missing. However, we point out that our approach, SiCNN, consistently outperforms CNN for all the examined levels of sparsity both in terms of MSE and MAE, except for the GunPoint dataset at sparsity level of 80%.

4.4 Experiments with Deep Convolutional Networks

After considering various neural networks, such as multi-layer perceptions, multi-scale convolutional neural networks and residual networks, as well as other time

Table 3. Mean squared error and mean absolute error (averaged over 10 folds) ± their standard deviation in case of various neural architectures with different types of convolution on the GunPoint, MoteStrain and PowerCons datasets at a sparsity level of $SL = 80\%$. Lower values indicate better performance. For each architecture, the best approach is underlined. For our approach (SiConv), we provide two symbols in the form of ∘/∘ which denote whether the difference compared to the same architecture with (i) "usual" convolution and (ii) "usual" convolution combined with linear interpolation of missing values (abbreviated as "+lin.") is statistically significant (•) or not (∘) according to paired t-test at significance level of $p = 0.01$.

Architecture	Convolution	GunPoint	MoteStrain	PowerCons
		Mean Squared Error		
simple CNN	"usual"	0.0675±0.0381	0.7120±0.0916	1.6712±0.3403
	"usual"+lin.	0.3287±0.1501	0.9816±0.1341	2.4377±0.7643
	SiConv	0.0763±0.0387∘/•	0.6299±0.0957•/•	1.5449±0.3347∘/•
ResNet	"usual"	0.2741±0.0872	0.7546±0.1289	1.3002±0.2679
	"usual"+lin.	0.5516±0.1041	0.9328±0.1650	2.1404±0.4537
	SiConv	0.0866±0.0434•/•	0.6724±0.1357•/•	1.1735±0.2438∘/•
FCN	"usual"	0.1135±0.0351	0.6053±0.1393	1.2299±0.2153
	"usual"+lin.	0.2567±0.0672	0.7628±0.1670	1.6392±0.4356
	SiConv	0.0509±0.0361•/•	0.5901±0.1363•/•	1.1687±0.2438•/•
		Mean Absolute Error		
simple CNN	"usual"	0.1772±0.0396	0.6083±0.0314	0.9842±0.0896
	"usual"+lin.	0.4871±0.1294	0.7850±0.0620	1.1170±0.2191
	SiConv	0.1860±0.0345∘/•	0.5651±0.0331•/•	0.9371±0.0848∘/∘
ResNet	"usual"	0.4038±0.0603	0.5945±0.0231	0.9035±0.0657
	"usual"+lin.	0.6040±0.0590	0.6941±0.0415	1.0675±0.1257
	SiConv	0.1961±0.0347•/•	0.5409±0.0252•/•	0.8460±0.0617•/•
FCN	"usual"	0.2491±0.0289	0.5444±0.0369	0.8809±0.0555
	"usual"+lin.	0.4066±0.0553	0.6487±0.0498	0.9200±0.1046
	SiConv	0.1218±0.0310•/•	0.5355±0.0362•/•	0.8470±0.0649•/•

series classifiers, namely: "time series based on a bag-of features", elastic ensemble, 1-nearest neighbor bag-of-SFA-symbols in vector space, shapelet ensemble, flat-COTE (COTE) and 1-nearest neighbor with dynamic time warping, Wang et al. [21] found that their "fully convolutional network" (FCN) "achieves premium performance", i.e., FCN outperforms all the aforementioned models. According to their observations, the difference between FCN and its competitors were statistically significant, expect for the second-best model, a residual network, denoted as ResNet. Therefore we decided to experiment with deep neural networks based on these FCN and ResNet architectures.

As the aforementioned networks were designed for time series classification, we had to adapt them for time series forecasting. In particular, both in case of FCN and ResNet, we removed the final global average pooling layer and replaced it by a fully connected layer in which the number of units corresponds to the forecast horizon, as each unit is expected to predict one of the numeric values

of the target time series, just like in case of the simple convolutional networks considered in Section 4.2.

Tab. 3 shows the performance of deep neural networks on the GunPoint, MoteStrain and PowerCons datasets in case of a sparsity level of $SL = 80\%$. As one can see, using SiConv instead of "usual" convolution improves performance both in case of ResNet and FCN.

5 Conclusion and Outlook

This paper focused on forecasting irregularly sampled time series with convolutional neural networks. We proposed to use sparsity-invariant convolution for this task. We performed experiments on 30 real-world time series datasets from various domains with a simple convolutional neural network. Additionally, we examined the effect of sparsity on the prediction error. We also experimented with a more advanced neural architecture, called "fully convolutional network" and a variant of ResNet that had been found to be particularly promising in case of time series previously. Our results show that convolutional neural networks with sparsity-invariant convolution systematically outperform their counterparts with "usual" convolution.

We point out that sparsity-invariant convolution may be used in any convolutional neural network instead of "usual" convolution which makes this operation attractive for many applications. In order to support reproduction of our results as well as follow-up works, we published our implementation (software codes) at https://github.com/kr7/timeseriesforecast-siconv .

Acknowledgement

This work was supported by the European Union through GraphMassivizer EU HE project under grant agreement No 101093202.

References

1. Borovykh, A., Bohte, S., Oosterlee, C.W.: Dilated convolutional neural networks for time series forecasting. Journal of Computational Finance, Forthcoming (2018)
2. Box, G.E., Jenkins, G.M., Reinsel, G.C., Ljung, G.M.: Time series analysis: forecasting and control. John Wiley & Sons (2015)
3. Buza, K., Antal, M.: Convolutional neural networks with dynamic convolution for time series classification. In: International Conference on Computational Collective Intelligence. pp. 304–312. Springer (2021)
4. Chatfield, C.: Time-series forecasting. Chapman and Hall/CRC (2000)
5. Che, Z., Purushotham, S., Cho, K., Sontag, D., Liu, Y.: Recurrent neural networks for multivariate time series with missing values. Scientific reports **8**(1), 6085 (2018)
6. Chen, Y., Kang, Y., Chen, Y., Wang, Z.: Probabilistic forecasting with temporal convolutional neural network. Neurocomputing **399**, 491–501 (2020)

7. David, Z.: Information leakage in financial machine learning research. Algorithmic Finance **8**(1-2), 1–4 (2019)
8. Gardner Jr, E.S.: Exponential smoothing: The state of the art—part ii. International journal of forecasting **22**(4), 637–666 (2006)
9. Kaushik, S., Choudhury, A., Sheron, P.K., Dasgupta, N., Natarajan, S., Pickett, L.A., Dutt, V.: Ai in healthcare: time-series forecasting using statistical, neural, and ensemble architectures. Frontiers in big data **3**, 4 (2020)
10. Kim, K.j.: Financial time series forecasting using support vector machines. Neurocomputing **55**(1-2), 307–319 (2003)
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
12. Lim, B., Zohren, S.: Time series forecasting with deep learning: A survey. Philosophical Transactions of the Royal Society A **379**(2194), 20200209 (2021)
13. Ramesh, A.N., Giovanneschi, F., González-Huici, M.A.: Siunet: Sparsity invariant u-net for edge-aware depth completion. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 5818–5827 (2023)
14. Ravuri, S., Lenc, K., Willson, M., Kangin, D., Lam, R., Mirowski, P., Fitzsimons, M., Athanassiadou, M., Kashem, S., Madge, S., et al.: Skilful precipitation nowcasting using deep generative models of radar. Nature **597**(7878), 672–677 (2021)
15. Seeger, M.W., Salinas, D., Flunkert, V.: Bayesian intermittent demand forecasting for large inventories. Advances in Neural Information Processing Systems **29** (2016)
16. Sen, R., Yu, H.F., Dhillon, I.S.: Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. Advances in neural information processing systems **32** (2019)
17. Sezer, O.B., Gudelek, M.U., Ozbayoglu, A.M.: Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. Applied soft computing **90**, 106181 (2020)
18. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research **15**(1), 1929–1958 (2014)
19. Torres, J.F., Hadjout, D., Sebaa, A., Martínez-Álvarez, F., Troncoso, A.: Deep learning for time series forecasting: a survey. Big Data **9**(1), 3–21 (2021)
20. Uhrig, J., Schneider, N., Schneider, L., Franke, U., Brox, T., Geiger, A.: Sparsity invariant cnns. In: 2017 international conference on 3D Vision (3DV). pp. 11–20. IEEE (2017)
21. Wang, Z., Yan, W., Oates, T.: Time series classification from scratch with deep neural networks: A strong baseline. In: 2017 International joint conference on neural networks (IJCNN). pp. 1578–1585. IEEE (2017)
22. Yadav, P., Steinbach, M., Kumar, V., Simon, G.: Mining electronic health records (ehrs) a survey. ACM Computing Surveys (CSUR) **50**(6), 1–40 (2018)
23. Yan, L., Liu, K., Belyaev, E.: Revisiting sparsity invariant convolution: A network for image guided depth completion. IEEE Access **8**, 126323–126332 (2020)