ARMA Cell: A Modular and Effective Approach for Neural Autoregressive Modeling

Anonymous authors
Paper under double-blind review

Abstract

The autoregressive moving average (ARMA) model is a classical, and arguably one of the most studied approaches to model time series data. It has compelling theoretical properties and is widely used among practitioners. More recent deep learning approaches popularize recurrent neural networks (RNNs) and, in particular, long short-term memory (LSTM) cells that have become one of the best performing and most common building blocks in neural time series modeling. While advantageous for time series data or sequences with long-term effects, complex RNN cells are not always a must and can sometimes even be inferior to simpler recurrent approaches. In this work, we introduce the ARMA cell, a simpler, modular, and effective approach for time series modeling in neural networks. This cell can be used in any neural network architecture where recurrent structures are present and naturally handles multivariate time series using vector autoregression. We also introduce the ConvARMA cell as a natural successor for spatially-correlated time series. Our experiments show that the proposed methodology is competitive with popular alternatives in terms of performance while being more robust and compelling due to its simplicity.

1 Introduction

Despite the rapidly advancing field of deep learning (DL), linear autoregressive models remain popular for time series analysis among academics and practitioners. Especially in economic forecasting, datasets tend to be small and signal-to-noise ratios low, making it difficult for neural network approaches to effectively learn linear or non-linear patterns. Although research in the past has touched upon autoregressive models embedded in neural networks (e.g., Connor et al., 1991; 1994), existing literature in fields guided by linear autoregressive models such as econometrics mainly focuses on hybrid approaches (see Section 2). These hybrid approaches constitute two-step procedures with suboptimal properties and often cannot even improve over the pure linear model. The DL community took a different route for time-dependent data structures, popularizing recurrent neural networks (RNNs), as well as adaptions to RNNs to overcome difficulties in training and the insufficient memory property (Hochreiter & Schmidhuber, 1997) of simpler RNNs. In particular, methods like the long short-term memory (LSTM) cell are frequently used in practice, whereas older recurrent approaches such as Jordan or Elman networks seem to have lost ground in the time series modeling community (Jordan, 1986; Elman, 1990). This can be attributed to the more stable training and the insensitivity to information lengths in the data of more recent recurrent network approaches such as the LSTM cell.

While often treated as a gold standard, we argue that these more complex RNN cells (such as the LSTM cell) are sometimes used only because of the lack of modular alternatives and that their long-term dependencies or data-driven forget mechanisms might not always be required in some practical applications. For example, in econometrics, including a small number of lagged time series values or lagged error signals in the model is usually sufficient to explain most of the variance of the time series. Similar, sequences of images (i.e., tensor-variate time series) such as video sequences often only require the information of a few previous image frames to infer the pixel values in the next time step(s). In addition, current optimization routines allow practitioners to train classical RNN approaches without any considerable downsides such as vanishing or exploding gradients.

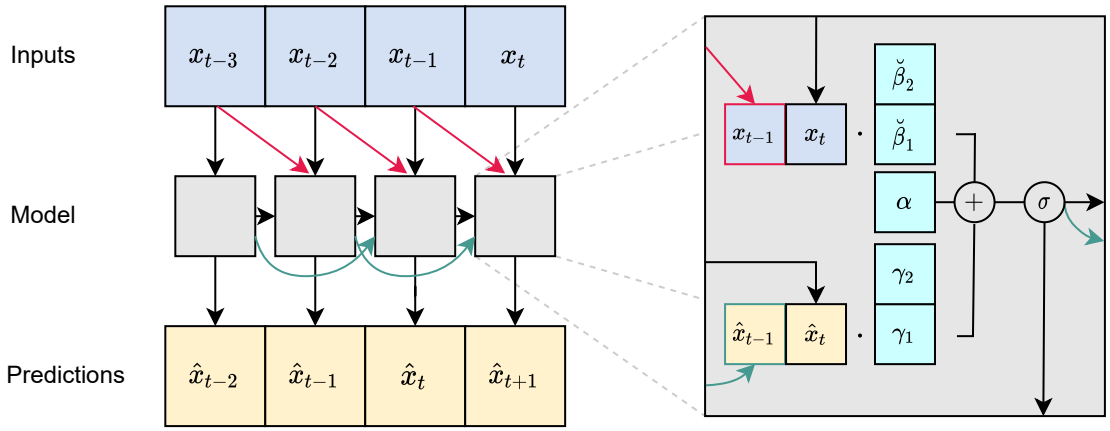

Figure 1: Left: Graphical visualizations of how predictions are computed in a univariate ARMA(2,2) cell using the time series values x from the current and previous time points as well as past model predictions $\hat{x}$. Right: Zooming in on the rightmost model cell from the left picture to show the computations of the ARMA cell with parameters as defined in equation 3.

Our contributions In this work, we propose a new type of RNN cell (cf. Figure 1) that can be seen as a natural connection between the classical time series school of thoughts and DL approaches. To analyze how the ARMA modeling philosophy can improve neural network predictions, we

- embed ARMA models in a neural network cell, which has various advantages over classical approaches (see Section 4);
- further exemplify how this proposal can be extended to convolutional approaches to model tensor-variate time series such as image sequences (Section 4.3);
- demonstrate through various numerical experiments that our model is on par with or even outperforms both its classical time series pendant as well as the LSTM cell in various settings; architectures ranging from shallow linear to deep non-linear time series models;
- provide a fully-tested, modular and easy-to-use `TensorFlow` (Abadi et al., 2016) implementation with a high-level syntax almost identical to existing RNN cells to foster its usage and systematic comparisons. It is available on Github[1]

The goal of this paper is further to make practitioners aware of an alternative to commonly used RNN cells, highlight that short-term recurrence can be sufficient in various time series applications, and that a simpler parameterized lag structures can even outperform data-driven forget mechanisms.

We start by discussing related literature in the following section. A short mathematical background is given in Section 3, followed by our proposed modeling approach in Section 4. We investigate practical aspects of our method in Section 5 and summarize all ideas and results in Section 6.

2 Related literature

Many advancements in the field of (deep) time series modeling have been made in recent years, in part inspired by autoregressive approaches from statistical modeling (e.g., DeepAR Salinas et al., 2020). As our proposal addresses the connection between classical methods and DL on the level of single building blocks in a network architecture, we focus on literature in classical time series analysis and fundamental RNN modeling

[1] We have attached the anonymized code as supplementary material. Link to the repository will be provided upon publication.

approaches in deep learning. More complex architectures and approaches for deep time series modeling can, e.g., be found in Han et al. (2019).

Traditional autoregressive approaches Autoregressive integrated moving average (ARIMA) models (see, e.g., Shumway & Stoffer, 2000) are a general class of linear models for forecasting a time series. They are characterized by a linear function of lags of the dependent variable (response) and lags of the forecasting errors. Boosted by the seminal paper of Box et al. (2015) and further popularized by their simplicity yet practical efficacy, ARIMA models have been extensively studied in statistics, econometrics, and related fields. A special case of ARIMA models are autoregressive moving average (ARMA) models, which are not "integrated", i.e., no differencing steps are required to obtain a stationary mean function. To overcome the linear restrictions of the AR(I)MA model and to account for non-linear patterns observed in real-world problems, several classes of non-linear models have been introduced. Examples include the bilinear model of Granger & Andersen (1978); Rao (1981), the threshold autoregressive model by Tong & Lim (2009), the smooth transition autoregressive model by Chan & Tong (1986), or the Markov switching autoregressive model by Hamilton (2010). Although these approaches have advantages over the linear methods, these models are usually developed to capture very specific non-linear patterns, and hence their generalization is limited. Many common extensions of AR(I)MA models such as seasonal ARIMA (SARIMA) or ARIMA with exogenous variables (ARIMAX) exist (Box et al., 2015). We here only focus on the basic ARMA model, but the proposed approaches in this paper can be easily extended to include the respective peculiarities of real-world time series such as non-stationarity, seasonality, and (non-)linear exogenous variables. Several authors have recognized analogies between ARMA models and RNNs (Connor et al., 1991; 1994; Saxén, 1997), which we will discuss in the following.

Recurrent neural network approaches When modeling sequential data such as time series, RNNs allow previous states to influence the current one, which presents a natural extension of classical perceptron-type connections (Rumelhart et al., 1986). When considering a feedforward neural network with a single hidden layer, such cyclical connections can be established by concatenating the output of the previous time step to the inputs of the next, yielding an Elman network (Elman, 1990). Similarly, a Jordan network is obtained by concatenating the previous hidden layer to the subsequent input (Jordan, 1986). An often-cited shortcoming of these so-called "simple recurrent networks" is their inferiority when learning long-term dependencies, which can be challenging due to vanishing or exploding gradients (e.g., Goodfellow et al., 2016). A variety of methods have since been developed to compensate for this shortcoming, including modeling multiple time scales simultaneously, adding leaky connections or allowing for longer delays within the recurrent connections. Most prominent are gated recurrent units (GRU; Cho et al., 2014) and long short-term memory cells (LSTM; Hochreiter & Schmidhuber, 1997). LSTM cells introduce self-loops that allow the gradient to flow without vanishing even for long durations using an input and a forget gate. GRU cells are similar to LSTM cells but only use a single gating unit to simultaneously control the forgetting and updating mechanism. Both methods have been shown to effectively tackle problems associated with long-term dependencies.

Combining classical time series approaches with neural networks A natural approach to allow for both linear autoregression and flexible non-linearity as provided by RNNs is to combine the two modeling techniques. Various hybrid approaches have been proposed in the past. One of the most common ways to combine the two paradigms is to fit a (seasonal) AR(I)MA model to the time series and subsequently train a neural network to explain the remaining variance in the residuals of the first stage mode (Aslanargun et al., 2007; Fathi, 2019; Tseng et al., 2002; Zhang, 2003). Other approaches specify a (time-delayed) neural network using the information from a preceding linear model by detrending and deseasonalizing in accordance with the first stage model (Taskaya-Temizel & Ahmad, 2005; Zhang & Qi, 2005).

State space models (SSMs) represent another popular model class in time series analysis (Aoki, 1990). The main advantages of SSMs are their generalized form and possible applications on complex, non-linear time series. A combination of SSMs and RNNs was proposed by Rivals & Personnaz (1996) as well as Suykens et al. (1995) using a network with a state layer between two hidden layers. This yields the so-called state space neural network (Amoura et al., 2011; Zamarreño & Vega, 1998). A more general representation of neural networks as a dynamical system is proposed by Hauser et al. (2019) and a deep combination of SSMs

and RNNs was further suggested by Rangapuram et al. (2018). While it is possible to represent ARMA models using SSMs, existing neural network SSM approaches in the literature do not aim for a general and modular building block, but propose specific networks using a fixed architecture.used for the mapping from features to SSM parameters.

Error Correction Models (ECMs) are commonly used to forecast cointegrated time series, i.e., time series that have a common long-term stochastic trend. For such time series, the information contained in the levels of the data is lost during a differencing step, making ECMs more suitable compared to ARIMA models. Mvubu et al. (2020) introduced a neural variant of ECMs, the Error Correction Neural Networks.

While some hybrid methods such as the combination of graph neural networks with an LSTM network (Smyl, 2020) have been shown to excel in time series forecasts, the efficacy of hybrid approaches combining AR(I)MA models and RNNs is often not clear (Taskaya-Temizel & Ahmad, 2005; Terui & Van Dijk, 2002). Khashei & Bijari (2011) propose a two-stage procedure that is guaranteed to improve the performance of the two single models, but as for all other existing hybrid approaches their technique requires fitting two separate models and hence cannot be combined or extended in a straightforward manner.

Recurrent convolutional approaches For spatio-temporal sequence forecasting problems, an extension to fully-connected RNN cells are RNN cells that apply convolutional operations to the spatially distributed information of multiple time series. As for time series applications, popular approaches use long-memory mechanisms, e.g., convolutional GRU adaptations (Tian et al., 2019) or the ConvLSTM (Shi et al., 2015) as an extension of the LSTM cell for spatio-temporal data.

3 Background and notation

In the following, we introduce our notation and the general setup for modeling time series. We will address univariate time series $x_t \in \mathbb{R}$ for time points $t \in \mathbb{Z}$ as well as multi- and tensor-variate time series, which we denote as $\boldsymbol{x}_t$ and $\boldsymbol{X}_t$, respectively.

ARMA model The ARMA(p, q) model (Box et al., 2015) for $p, q \in \mathbb{N}_0$ is defined as

$$x_t = \alpha + \sum_{i=1}^{p} \beta_i x_{t-i} + \sum_{j=1}^{q} \gamma_j \varepsilon_{t-j} + \varepsilon_t \alpha + h_t + \varepsilon_t, \tag{1}$$

where x_t represents the variable of interest defined for $t \in \mathbb{Z}$ and is observed at time points $t = 1, \ldots, T$, $T \in \mathbb{N}$[2]. $\alpha, \beta_1, \ldots, \beta_p, \gamma_1, \ldots, \gamma_q$ are real valued parameters and $\varepsilon_t \overset{iid}{\sim} \mathcal{F}(\sigma^2)$ is an independent and identically distributed (iid) stochastic process with pre-specified distribution $\mathcal{F}$ and variance parameter $\sigma^2 > 0$. By setting $q = 0$ or $p = 0$, the ARMA model comprises the special cases of a pure autoregressive (AR) and a pure moving average (MA) model, respectively. The class of ARMA models is, in turn, a special case of ARIMA models, where differencing steps are applied to obtain a stationary mean function before fitting the ARMA model. As stationarity is also a fundamental assumption for RNNs to justify parameter sharing (Goodfellow et al., 2016), we focus on the class of ARMA models in this work, i.e., assume that differencing has already been applied to the data. series is characterized by a constant mean and variance and a time invariant autocorrelation structure. $\alpha_0, \ldots, \alpha_p$ and $\beta_1, \ldots, \beta_q$ are model parameters and p and q characterize the number of lags of the dependent variable and the forecasting errors included in the model, respectively.

VARMA model The univariate ARMA model can be generalized to a multivariate version – the vector autoregressive moving average (VARMA) model – by adapting the principles of the ARMA model for multivariate time series. The VARMA(p, q) model (Tiao & Box, 1981) for $p, q \in \mathbb{N}_0$ is defined as

$$\boldsymbol{x}_t = \boldsymbol{\alpha} + \sum_{i=1}^{p} \boldsymbol{B}_i \boldsymbol{x}_{t-i} + \sum_{j=1}^{q} \boldsymbol{\Gamma}_j \boldsymbol{\varepsilon}_{t-j} + \boldsymbol{\varepsilon}_t \tag{2}$$

[2]It is common to define a time series for time points $t = 1, \ldots, T$ to describe its current value and recent history, while time series dynamics are assumed to originate from time points prior to $t = 1$, hence $t \in \mathbb{Z}$

where $\boldsymbol{x}_t, t \in \mathbb{Z}$ represents a vector of time series observed at time points $t = 1, \ldots, T$. $\boldsymbol{B}_i$ and $\boldsymbol{\Gamma}_j$ are time-invariant $(k \times k)$-matrices, where $k \in \mathbb{N}$ represents the number of individual time series. $\boldsymbol{\varepsilon}_t$ is a k-dimensional iid stochastic process with pre-specified k-dimensional distribution $\mathcal{F}(\boldsymbol{\Omega})$ and covariance matrix $\boldsymbol{\Omega}$. By setting $q = 0$, the VARMA model comprises the special cases of a pure autoregressive (VAR) model, which is the most common VARMA model used in applications. Similar to the ARMA model being a special case of the ARIMA model class, the VARMA model is a special case of the VARIMA model class, representing only stationary time series.

4 ARMA-based neural network layers

ARMA models have been successfully used in many different fields and are a reasonable modeling choice for time series in many areas. This section introduces a neural network cell version of the ARMA mechanism. While very similar to Elman or Jordan networks, the proposed cell exactly resembles the ARMA computations and can be used in a modular fashion in any neural network architecture where recurrent structures are present. Emulating the ARMA logic in a recurrent network cell has various advantages. It allows to 1) recover estimated coefficients of classical ARMA software (see Supplementary Material B.1 for an empirical investigation of the convergence), but can also be used to fit ARMA models for large-scale or tensor-variate data (which is otherwise computationally infeasible), 2) modularly use the ARMA cell in place for any other RNN cell, 3) combine ARMA approaches with other features from neural networks such as regularization and thereby seamlessly extend existing time series models, and 4) model hybrid linear and deep network models that were previously only possible through multi-step procedures. As shown in our numerical experiments section, an ARMA cell can further lead to comparable or even better prediction performance compared to modern RNN cells.

4.1 ARMA cell

An alternative formulation of the ARMA model can be derived by incorporating the error term through the predictions $\hat{x}_t \coloneqq x_t - \varepsilon_t, t \in \mathbb{Z}$. Thus, equation 1 can be defined in terms of its intercept, the model predictions $\hat{x}_t$ and the actual time series values x_t. It follows:

$$\hat{x}_t = \alpha + \sum_{i=1}^{\max(p,q)} \breve{\beta}_i x_{t-i} - \sum_{j=1}^{q} \gamma_j \hat{x}_{t-j} \quad \text{with } \breve{\beta}_i = \begin{cases} \beta_i + \gamma_i & \text{for } i \leq \min(p,q), \\ \beta_i & \text{for } i > q \text{ and } p > q, \\ \gamma_i & \text{for } i > p \text{ and } p < q. \end{cases} \tag{3}$$

Using equation 3, we can implement the ARMA functionality as an RNN cell. More specifically, the recurrent cell processes the p lagged time series values as well as the q predicted outputs of the previous time steps and computes a linear combination with parameters $\breve{\beta}_i$ and γ_j. After adding a bias term, the final output $\hat{x}_t$ is given by a (non-linear) activation function σ of the sum of all terms. Figure 1 gives both a higher-level view of how predictions are computed in the ARMA cell as well as a description of how the cell is defined in detail. In addition to the classical ARMA computations in the cell, the activation function σ allows to straightforwardly switch between a linear ARMA model and a non-linear version.

4.2 Extensions

The ARMA cell in Figure 1 can be used in a modular fashion similar to an LSTM or GRU cell. In the following, we will thus present how this idea can be used to generate more complex architectures using multiple units or by stacking cells. Both options also allow bypassing the linearity assumptions of ARMA models.

Multi-unit ARMA cell Similar to feedforward neural networks, an RNN layer can also contain multiple units. Each unit receives the same input but can capture different effects due to the random initialization of weights. The outputs of each unit are then concatenated. In the left panel of Figure 2, a multi-unit architecture allows combining different activation functions to simultaneously capture linear and non-linear effects. Using a multi-unit ARMA cell thereby seamlessly provides the possibility to combine a linear with

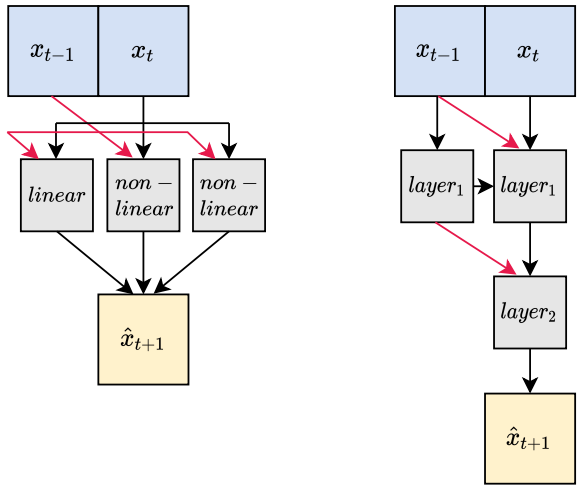

Figure 2: Visualization of an ARMA cell with multiple units representing a mixture of linear and non-linear ARMA models by using different activation functions (left) and a network with stacked ARMA cells creating a more complex model class by transforming inputs by subsequent ARMA cells (right).

a non-linear ARMA model. We refer to models having a single hidden ARMA layer with one or more units as *ShallowARMA* models.

Stacked ARMA To allow for higher levels of abstraction and increased model complexity, the ARMA modeling strategy does not only allow for multiple units in a single layer, but users can also stack multiple layers in series. This is achieved by returning a sequence of lagged outputs from the previous layer, as depicted on the right of Figure 2. Models with more than one hidden ARMA layer are referred to as *DeepARMA* models in the following.

4.3 ConvARMA

Similar to the ConvLSTM network (Shi et al., 2015), it is possible to model spatial dependencies and process tensor-variate time series $\boldsymbol{X}_t \in \mathbb{R}^{n_1 \times \ldots \times n_d}, n_1, \ldots, n_d \in \mathbb{N}, d \in \mathbb{N}$ by using convolution operations within an ARMA cell. The resulting ConvARMA(p, q) cell for $p, q \in \mathbb{N}_0$ and $t \in \mathbb{Z}$ is defined as

$$\boldsymbol{I}_t = \sum_{i=1}^{p} \boldsymbol{W}_i * \boldsymbol{X}_{t-i}, \quad \boldsymbol{C}_t = \sum_{j=1}^{q} \boldsymbol{U}_j * \hat{\boldsymbol{X}}_{t-j}, \quad \hat{\boldsymbol{X}}_t = \sigma\left(\boldsymbol{I}_t + \boldsymbol{C}_t + \boldsymbol{b}\right), \tag{4}$$

where $*$ represents the convolution operator, $\boldsymbol{W}_i \in \mathbb{R}^{k_1 \times \ldots \times k_{d-1} \times n_d \times c}, i = 1, \ldots, p$ and $\boldsymbol{U}_j \in \mathbb{R}^{k_1 \times \ldots \times k_{d-1} \times c \times c}, j = 1, \ldots, q$ are the model's kernels of size $k_1 \times \ldots \times k_{d-1}$, $\boldsymbol{b} \in \mathbb{R}^c$ is a bias term broadcasted to dimension $n_1 \times \ldots \times n_{d-1} \times c$ and σ an activation function. By convention, the last dimension of the input represents the channels, and c denotes the number of filters of the convolution. The inputs of the convolution are padded to ensure that the spatial dimensions of the prediction $\hat{\boldsymbol{X}}_t$ and the state remain unchanged. In other words, the ConvARMA cell resembles the computations of an ARMA model, but instead of simple multiplication of the time series values with scalar-valued parameters, a convolution operation is applied. Figure 3 shows an abstract visualization of the computations in a ConvARMA cell. To follow the AR(I)MA modeling logic in the spatial dimensions, a ConvARMA cell can further incorporate spatial differences in all directions. A possible extension of the cell proposed in equation 4 could further be to allow for non-linear recurrent activations as done for, e.g., the ConvLSTM cell.

As for the ConvLSTM or ConvGRU cell, the ConvARMA cell can be included in an autoencoder architecture for sequence-to-sequence modeling or extended to e.g., allow for warping, rotation, and scaling (Shi et al., 2017).

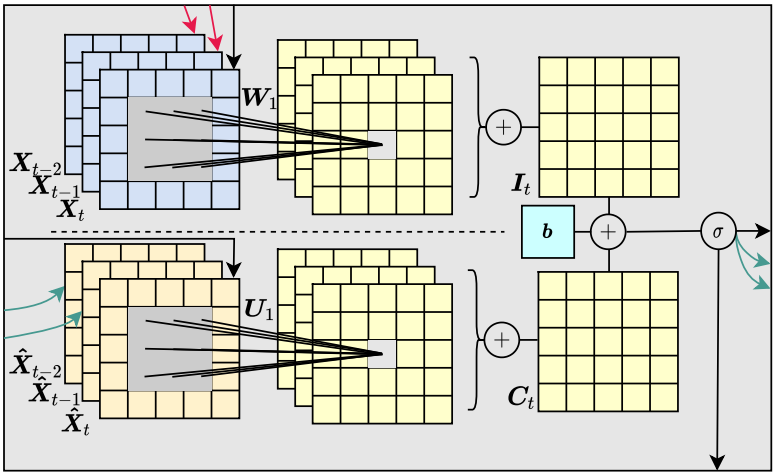

Figure 3: Exemplary visualization of a single-filter ConvARMA cell processing matrix-variate time series (with a single channel) with three lags (upper left) and matrix-variate predictions with three lags (bottom left) using convolutions and combining the results into a single matrix prediction (bottom/top right) with additional bias term b and activation function σ (center right).

4.4 Limitations

As for other autoregressive approaches, our approach is limited in its application if the time series are very short or if a large number of lags p is required to approximate the underlying data generating process well. We note, however, that due to the model's recurrent definition, past time points $t - i$ for $i > p$ also influence the model's predictions. It is therefore often not necessary to define a large lag value p, even if autocorrelation is high. Despite the ARMA cell's simplicity, this also shows that its predictions are not always straightforward to interpret.

5 Numerical experiments

In this section, we examine the performance of our ARMA cell in a variety of synthetic and benchmark experiments. We examine how it compares to classical time series approaches as well as to a similar complex neural architectures. Note that our experiments are not designed to be a benchmark comparison with current state-of-the-art time series forecasting frameworks. These rather complex architectures include many different components, such as automated pre-processing and feature generation, and thus do not necessarily allow to make a statement about the performance of a single recurrent cell therein. Instead, we aim for a comparison with other fundamental modeling building blocks [3].

Methods For (multivariate) time series, we compare a shallow and a deep variant of the ARMA cell against the respective (V)ARMA model and neural models. For the latter, we consider LSTM, GRU, and Simple RNN cells, again each in a their shallow and deep variants. Hyperparameter optimization is done using a grid search with predefined parameter spaces for the number of units for all network layers and lags for ARMA-type models. All other hyperparameters of network layers are kept fixed with defaults that do not favor one or the other method. For moving images, we compare ConvARMA against a naïve approach of repeating the last image and a ConvLSTM network with parameter specifications that are defined as similar as possible to the one of ConvARMA. Further details on the specification of the architectures can be found in the Supplementary Material B.4.

[3] We provide code to reproduce all experiments in the supplementary material

Table 1: Comparisons of different methods (rows) and different data generating processes (columns) for univariate time series using the average RMSE ± the standard deviation of 10 independent runs. The best performing method is highlighted in bold, the second-best in italics.

model	ARMA	TAR	SGN	NAR	Heteroskedastic
ARMA	2.04±0.35	2.92±3.20	2.36±2.67	2.67±4.92	1.19±0.15
ShallowARMA	**1.96±0.09**	**1.09±0.12**	1.10±0.07	*1.02±0.05*	**1.11±0.06**
DeepARMA	1.97±0.10	*1.24±0.47*	**1.05±0.06**	1.02±0.04	*1.11±0.06*
LSTM	1.98±0.10	1.38±0.42	1.19±0.16	1.02±0.04	1.15±0.09
DeepLSTM	2.02±0.10	1.46±0.55	1.17±0.13	1.02±0.04	1.16±0.08
GRU	*1.96±0.10*	1.28±0.31	*1.09±0.08*	**1.02±0.04**	1.13±0.07
DeepGRU	1.99±0.09	1.24±0.36	1.09±0.12	1.02±0.04	1.12±0.06
Simple	1.99±0.09	1.29±0.31	1.14±0.09	1.04±0.04	1.13±0.08
DeepSimple	2.01±0.11	1.47±0.57	1.15±0.10	1.03±0.04	1.16±0.10

Performance measures We compare time series predictions using the root mean squared error (RMSE) for uni- and multivariate time series forecasts, and the cross-entropy loss for next frame video predictions. We provide further performance measures for our comparisons in the Supplementary Material D.

5.1 Simulation study

We start with a variety of synthetic data examples using time series models defined in Lee et al. (1993). Simulations include linear and non-linear, as well as uni- and multivariate time series. All time series are of length 1000 and split into 70% train and 30% test data. The data generating processes follow Lee et al. (1993) and include an ARMA process (ARMA), a threshold autoregressive model (TAR), an autoregressive time series which is transformed using the sign operation (SGN), a non-linear autoregressive series (NAR), a heteroscedastic MA process (Heteroscedastic), a vector ARMA (VARMA), a non-linear multivariate time series with quadratic lag structure (SQ) and an exponential multivariate autoregressive time series (EXP). The exact specification of the data generating processes can be found in the Supplementary Material B.2.

Results The results in Table 1 suggest that the ShallowARMA approach emulating an ARMA model in a neural network works well for all linear- and non-linear datasets. In terms of robustness, the lower RMSE and high standard deviation of the ARMA model on the ARMA process shows that fitting an ARMA model in a neural network with stochastic gradient descent can, in fact, be more robust than the standard software (Hyndman & Khandakar, 2008; Seabold & Perktold, 2010). While the classical ARMA did match the performance of its neural counterpart it some cases, the average RMSE is worse, as it did not converge in all runs, even for the linear time series. The performance of the DeepARMA approach is slightly worse compared to the ShallowARMA in most cases. The performance of LSTM, GRU, and the Simple RNN are all similar, with all methods matching the ARMA cells in some cases, and falling slightly behind in others. As expected, the classical ARMA approach does not work well for non-linear data generating processes (TAR, SGN, NAR) and yields unstable predictions underpinned by the large standard deviations in RMSE values.

For multivariate time series results of the simulation are summarized in Table 2. The results again show that the ShallowARMA model matches the performance of the classical VARMA model for a dataset that is also based on a VARMA process. For other types of data generation, the ShallowARMA model and DeepARMA model work similarly well. Both outperform the other neural cells, which in turn yield better results than the VARMA baseline.

In summary, findings suggest that ARMA cells work well for simpler linear and non-linear data generating processes while being much more stable than a classical ARMA approach. In Supplementary Material B.1, we further study the empirical convergence of a single unit single hidden layer ARMA cell, which is mathematically equivalent to an ARMA model for given values of p and q.

Table 2: Comparisons of different methods (rows) and different data generating processes (columns) for multivariate time series using the average RMSE ± the standard deviation of 10 independent runs. The best performing method is highlighted in bold, the second-best in italics.

	VARMA	EXP	SQ
VARMA	**1.00±0.03**	3.35±0.69	1.90±0.14
ShallowARMA	*1.00±0.03*	*3.14±0.74*	**1.75±0.12**
DeepARMA	1.01±0.03	**3.10±0.75**	*1.76±0.11*
LSTM	1.01±0.04	3.26±0.73	1.83±0.16
DeepLSTM	1.03±0.04	3.35±0.68	1.86±0.14
GRU	1.02±0.04	3.19±0.75	1.80±0.13
DeepGRU	1.01±0.04	3.22±0.80	1.82±0.18
SIMPLE	1.02±0.03	3.29±0.70	1.80±0.12
DeepSimple	1.02±0.03	3.30±0.84	1.83±0.13

Table 3: Comparison of different univariate and multivariate forecasting approaches (rows) for different datasets (columns) based on the average RMSE ± the standard deviation of 10 independent runs. The best performing method is highlighted in bold, the second-best in italics.

		m4	traffic	electricity	exchange
univ.	ARMA	1.58±0.00	0.98±0.00	1.19±0.00	1.18±0.00
	ShallowARMA	**1.57±0.01**	0.97±0.00	1.14±0.01	*1.03±0.00*
	DeepARMA	*1.57±0.01*	**0.94±0.01**	**1.10±0.02**	**1.03±0.00**
	LSTM	1.71±0.14	*0.96±0.01*	1.15±0.07	1.03±0.01
	DeepLSTM	1.96±0.38	0.97±0.02	1.12±0.05	1.04±0.01
	GRU	1.61±0.03	0.97±0.02	*1.11±0.02*	1.04±0.01
	DeepGRU	1.61±0.02	0.97±0.02	1.11±0.03	1.04±0.00
	Simple	1.72±0.15	1.00±0.01	1.12±0.01	1.04±0.00
	DeepSimple	1.76±0.17	1.00±0.02	1.11±0.02	1.04±0.01
multiv.	ARMA	1.72±0.00	*1.06±0.00*	1.46±0.00	1.33±0.00
	ShallowARMA	*1.68±0.01*	**1.06±0.00**	1.37±0.03	*1.10±0.00*
	DeepARMA	**1.67±0.01**	1.08±0.01	1.32±0.03	1.10±0.00
	LSTM	1.92±0.15	1.15±0.01	2.07±1.12	**1.10±0.01**
	DeepLSTM	2.11±0.25	1.15±0.02	1.26±0.05	1.18±0.25
	GRU	1.91±0.25	1.15±0.00	1.25±0.04	1.10±0.00
	DeepGRU	1.88±0.12	1.15±0.01	*1.23±0.02*	1.10±0.00
	Simple	1.89±0.06	1.16±0.00	1.25±0.03	1.10±0.00
	DeepSimple	1.90±0.08	1.16±0.01	**1.22±0.01**	1.10±0.00

5.2 Benchmarks

In order to investigate the performance of our approach for real-world time series with a potentially more complex generating process, we compare the previously defined models on various time series benchmark datasets.

5.2.1 Univariate and multivariate time series

We use the m4 (Makridakis et al., 2018), traffic (Yu et al., 2016), electricity (Yu et al., 2016) and exchange (Lai et al., 2018) dataset, all openly accessible and commonly used in time series forecast benchmarks. Further background on every dataset and details on pre-processing can be found in the Supplementary Material B.3. As all datasets come with multiple time series, we use these datasets both for testing the performance on univariate and multivariate time series. For univariate time series, this is done by training a model for every dimension and averaging the results over the different multivariate dimensions.

Univariate time series Results of univariate benchmarks are summarized in Table 3. The comparisons suggest that the two ARMA cells and the other neural cells perform equally well on the Exchange dataset, but the ARMA cells outperform on the other datasets. The classical ARMA model is competitive for the m4 dataset, but again worse than its neural pendant on Traffic, Electricity, and Exchange.

Multivariate time series For the multivariate time series benchmarks, we observe that model performance is in general worse than when performing hyperparameter optimization and model training for each time series individually, as done for the univariate time series benchmark. Finding architectures better suited to the individual time series seems to outweigh the additional information from observing the comovement of multiple time series simultaneously. In the comparison of different forecasting approaches for multivariate dimensions, the performance of the ARMA cells is either notably better than the other neural cells but on par with the classical ARMA model (Traffic), better than the ARMA model but on par with the other neural cells (Exchange, Electricity), or outperforms all other approaches (m4).

5.2.2 Tensor-variate time series

Finally, we compare the ARMA and LSTM approach on tensor-variate time series such as image or sensor-grid sequences. These benchmarks are performed with different layers and filter sizes to investigate the difference in performance for different RNN cell complexity.

Datasets To compare the models' performance, we use five different datasets. A common dataset for next video frame prediction is the Moving MNIST dataset (MovMNIST; Srivastava et al., 2015) which contains video sequences of two digits moving randomly inside a frame. Similarly, the Noisy and Shifted squares datasets (Noisy, Shifted) used to investigate the properties temporal convolutions (Chollet et al., 2015) consist of smaller and bigger squares moving through a pre-defined window at different speeds. In addition, we analyze two spatio-temporal datasets, the Taxi NYC and Bike NYC datasets (NYTaxi, NYBike; as, e.g., used in Lin et al., 2020). These consist of hourly taxi and bike movements in New York City quantified as the number of inflows and outflows of each sector in a grid view of the city. A more detailed description of all datasets can be found in Supplementary Material B.3.

Table 4: Comparison of different forecasting approaches (rows; with numbers corresponding to the quadratic filter sizes of each layer) for different datasets (columns) based on the average cross-entropy (standard deviation in brackets) over 10 different initializations. The best-performing method is highlighted in bold.

	MovMNIST	Noisy	Shifted	NYTaxi	NYBike
ConvARMA 5-3-1	**0.063±0.001**	**0.094±0.002**	**0.082±0.002**	**0.281±0.001**	**0.285±0.001**
ConvLSTM 5-3-1	0.076±0.038	0.116±0.002	0.109±0.004	0.285±0.001	0.290±0.002
ConvARMA 3-1	**0.072±0.003**	**0.149±0.057**	**0.151±0.065**	**0.288±0.000**	**0.292±0.000**
ConvLSTM 3-1	0.093±0.002	0.161±0.002	0.154±0.002	0.289±0.000	0.295±0.000
ConvARMA 3	**0.075±0.000**	**0.120±0.002**	**0.112±0.001**	0.289±0.000	**0.296±0.000**
ConvLSTM 3	0.103±0.018	0.167±0.002	0.159±0.001	**0.289±0.000**	0.296±0.001
Baseline	0.509	1.041	1.135	0.375	0.391

Results Table 4 summarizes the comparisons of tensor-variate forecasts. Similar to the univariate and multivariate time series applications, the tensor-variate version of the ARMA cell outperforms its LSTM pendant in all configurations on Moving MNIST as well as the noisy and shifted datasets. For the Taxi and Bike data, both methods perform on par while being notably better than the baseline.

6 Conclusion and Outlook

We provided a modular and flexible neural network cell to model time series in a simply parameterized fashion and as an alternative to commonly used RNN cells such as the LSTM cell. We further extended this

approach to vector autoregression and autoregressive models for tensor-variate applications. Our numerical experiments show that the ARMA cell 1) performs well on univariate, multivariate, and tensor-variate time series; 2) matches or even outperforms the LSTM, GRU, and a Simple RNN cell in linear and non-linear settings, and; 3) shows more robust convergence for classical ARMA formulations compared to a standalone implementation. While the focus of this paper was to investigate the performance of the ARMA cell in comparison to a basic ARMA model and a vanilla neural cells, an interesting future experiment is to examine whether state-of-the-art forecasting frameworks such as DeepAR (Salinas et al., 2020) can be improved by replacing the LSTM cells with ARMA cells.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. pp. 265–283, 2016.

Karima Amoura, Patrice Wira, and Said Djennoune. A state-space neural network for modeling dynamical nonlinear systems. In *IJCCI (NCTA)*, pp. 369–376, 2011.

Masanao Aoki. *State space modeling of time series*. Universitext. Springer, Berlin, Germany, August 1990.

Atilla Aslanargun, Mammadagha Mammadov, Berna Yazici, and Senay Yolacan. Comparison of ARIMA, neural networks and hybrid models in time series: tourist arrival forecasting. *Journal of Statistical Computation and Simulation*, 77(1):29–53, 2007.

George E.P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

K. S. Chan and H. Tong. On estimating thresholds in autoregressive models. *Journal of Time Series Analysis*, 7(3):179–190, 1986.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Francois Chollet et al. Keras, 2015. URL `https://github.com/fchollet/keras`.

Jerome Connor, Les Atlas, and Douglas Martin. Recurrent networks and NARMA modeling. *Advances in Neural Information Processing Systems*, 4, 1991.

J.T. Connor, R.D. Martin, and L.E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5(2):240–254, 1994. doi: 10.1109/72.279188.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. ISSN 0364-0213. doi: https://doi.org/10.1016/0364-0213(90)90002-E. URL `https://www.sciencedirect.com/science/article/pii/036402139090002E`.

Oussama Fathi. Time series forecasting using a hybrid ARIMA and LSTM model. *Velvet Consulting*, pp. 1–7, 2019.

Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

Clive W.J. Granger and Allan Andersen. On the invertibility of time series models. *Stochastic Processes and their Applications*, 8(1):87–92, 1978.

James D. Hamilton. Regime switching models. In *Macroeconometrics and time series analysis*, pp. 202–209. Springer, 2010.

Zhongyang Han, Jun Zhao, Henry Leung, King Fai Ma, and Wei Wang. A review of deep learning models for time series prediction. *IEEE Sensors Journal*, 21(6):7833–7848, 2019.

Michael Hauser, Sean Gunn, Samer Saab Jr, and Asok Ray. State-space representations of deep neural networks. *Neural Computation*, 31(3):538–554, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 26(3):1–22, 2008. doi: 10.18637/jss.v027.i03.

Michael I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 531–546. Hillsdale, NJ: Erlbaum, 1986.

Mehdi Khashei and Mehdi Bijari. A novel hybridization of artificial neural networks and ARIMA models for time series forecasting. *Applied Soft Computing*, 11(2):2664–2675, 2011.

Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 95–104, 2018.

Tae-Hwy Lee, Halbert White, and Clive W.J. Granger. Testing for neglected nonlinearity in time series models: A comparison of neural network methods and alternative tests. *Journal of Econometrics*, 56(3): 269–290, 1993.

Haoxing Lin, Rufan Bai, Weijia Jia, Xinyu Yang, and Yongjian You. Preserving dynamic attention for long-term spatial-temporal prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 36–46, 2020. doi: 10.1145/3394486.3403046.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4):802–808, 2018. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2018.06.001. URL https://www.sciencedirect.com/science/article/pii/S0169207018300785.

Mhlasakululeka Mvubu, Emmanuel Kabuga, Christian Plitz, Bubacarr Bah, Ronnie Becker, and Hans G. Zimmermann. On error correction neural networks for economic forecasting. In *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, pp. 1–8, 2020. doi: 10.23919/FUSION45008.2020.9190244.

Syama Sundar Rangapuram, Matthias W. Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. *Advances in Neural Information Processing Systems*, 31, 2018.

T. Subba Rao. On the theory of bilinear time series models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 43(2):244–255, 1981.

Isabelle Rivals and Léon Personnaz. Black-box modeling with state-space neural networks. In *Neural Adaptive Control Technology*, pp. 237–264. World Scientific, 1996.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2019.07.001. URL https://www.sciencedirect.com/science/article/pii/S0169207019301888.

Henrik Saxén. On the equivalence between ARMA models and simple recurrent neural networks. In *Applications of Computer Aided Time Series Modeling*, pp. 281–289. Springer, 1997.

Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Xingjian Shi, Zhihan Gao, Leonard Lausen, Hao Wang, Dit-Yan Yeung, Wai kin Wong, and Wang chun Woo. Deep learning for precipitation nowcasting: A benchmark and a new model, 2017.

Robert H. Shumway and David S. Stoffer. *Time series analysis and its applications*, volume 3. Springer, 2000.

Slawek Smyl. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1):75–85, 2020.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using LSTMs. In *International conference on machine learning*, pp. 843–852. PMLR, 2015.

Johan A.K. Suykens, Bart L.R. De Moor, and Joos Vandewalle. Nonlinear system identification using neural state space models, applicable to robust control design. *International Journal of Control*, 62(1):129–152, 1995.

T. Taskaya-Temizel and K. Ahmad. Are ARIMA neural network hybrids better than single models? In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 5, pp. 3192–3197 vol. 5, 2005. doi: 10.1109/IJCNN.2005.1556438.

Nobuhiko Terui and Herman K. Van Dijk. Combined forecasts from linear and nonlinear time series models. *International Journal of Forecasting*, 18(3):421–438, 2002.

Lin Tian, Xutao Li, Yunming Ye, Pengfei Xie, and Yan Li. A generative adversarial gated recurrent unit model for precipitation nowcasting. *IEEE Geoscience and Remote Sensing Letters*, 17(4):601–605, 2019.

George C. Tiao and George E.P. Box. Modeling multiple time series with applications. *Journal of the American Statistical Association*, 76(376):802–816, 1981.

Howell Tong and Keng S. Lim. Threshold autoregression, limit cycles and cyclical data. In *Exploration Of A Nonlinear World: An Appreciation of Howell Tong's Contributions to Statistics*, pp. 9–56. World Scientific, 2009.

Fang-Mei Tseng, Hsiao-Cheng Yu, and Gwo-Hsiung Tzeng. Combining neural network model with seasonal time series ARIMA model. *Technological Forecasting and Social Change*, 69(1):71–87, 2002.

Hsiang-Fu Yu, Nikhil Rao, and Inderjit S. Dhillon. Temporal regularized matrix factorization for high-dimensional time series prediction. In *NIPS*, pp. 847–855, 2016.

Jesús M. Zamarreño and Pastora Vega. State space neural network. properties and application. *Neural Networks*, 11(6):1099–1112, 1998.

G. Peter Zhang. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50:159–175, 2003.

G. Peter Zhang and Min Qi. Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, 160(2):501–514, 2005.

A Definitions, assumptions and theoretical analysis

This shows how to rewrite the ARMA model. We start with

$$x_t = \alpha + \sum_{i=1}^{p} \beta_i x_{t-i} + \sum_{j=1}^{q} \gamma_j \varepsilon_{t-j} + \varepsilon_t$$

and use the definition of $\hat{x}_t \coloneqq x_t - \varepsilon_t$ to get

$$\hat{x}_t = \alpha + \sum_{i=1}^{p} \beta_i x_{t-i} + \sum_{j=1}^{q} \gamma_j \varepsilon_{t-j}.$$

We now replace each ε_{t-j} with $x_{t-j} - \hat{x}_{t-j}$

$$\hat{x}_t = \alpha + \sum_{i=1}^{p} \beta_i x_{t-i} + \sum_{j=1}^{q} \gamma_j (x_{t-j} - \hat{x}_{t-j}) = \alpha + \sum_{i=1}^{p} \beta_i x_{t-i} + \sum_{i=1}^{q} \gamma_i x_{t-i} - \sum_{j=1}^{q} \gamma_j \hat{x}_{t-j}.$$

We see that for all indices $i \leq \min(p, q)$ the common factor of x_{t-i} is $\beta_i + \gamma_i$, if $p > q$ and $i > q$ the factor is β_i and if $q > p$ and $i > p$ then the factor is γ_i, yielding equation 3.

B Further details and results for numerical experiments

B.1 ARMA parameter recovery

In order to investigate if the implemented cell recovers parameters of an arbitrary ARMA model with coefficients estimated in a standard ARMA software (Seabold & Perktold (2010)), we simulate (V)ARMA processes for $25,000$ time steps and all possible combinations of $p, q \in \{0, 1, \ldots, 5\}$. We then train a neural network defined by a single linear ARMA cell on the data and check the convergence against the values obtained by maximum likelihood estimation. Results confirm that the ARMA cell can recover the coefficients for different values of p and q, and also in the multivariate setting. Figure 4 visualizes one exemplary learning process.

B.2 Description of simulated data generating processes

All error terms are a Gaussian white noise $\varepsilon_t \sim \mathcal{N}(0, 1)$. The data generating processes were defined as follows:

- ARMA(2,1)

$$x_t = 0.1x_{t-1} + 0.3x_{t-2} - 0.4\varepsilon_{t-1} + \varepsilon_t$$

- Threshold autoregressive (TAR)

$$x_t = \begin{cases} 0.9x_{t-1} + \varepsilon_t & \text{for } |x_{t-1}| \leq 1, \\ -0.3x_{t-1} + \varepsilon_t & \text{for } |x_{t-1}| > 1 \end{cases}$$

- Sign autoregressive (SGN)

$$x_t = sgn(x_{t-1}) + \varepsilon_t,$$

  with

$$sgn(x) = \begin{cases} 1 \text{ for } x > 0, \\ 0 \text{ for } x = 0, \\ -1 \text{ for } x < 0 \end{cases}$$

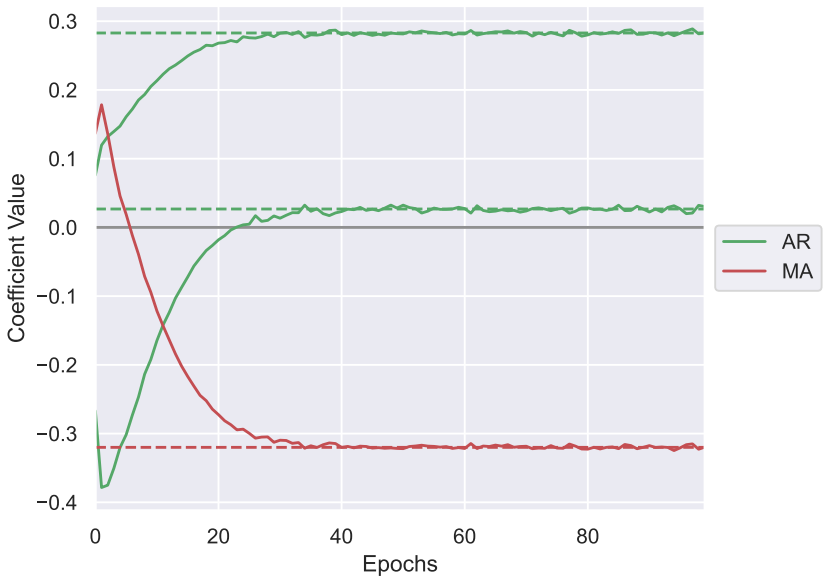

Figure 4: Optimization paths for a single linear ARMA(2,1) cell using stochastic gradient descent. After around 30 iterations, the model converges to the maximum likelihood coefficients.

- Nonlinear autoregressive (NAR)

$$x_t = \frac{0.7|x_{t-1}|}{|x_{t-1}+2|} + \varepsilon_t$$

- Heteroskedastic MA(2)

$$x_t = \varepsilon_t - 0.4\varepsilon_{t-1} + 0.3\varepsilon_{t-2} + 0.5\varepsilon_t\varepsilon_{t-2}$$

- VARMA

$$\begin{bmatrix} x_{t,1} \\ x_{t,2} \end{bmatrix} = \begin{bmatrix} 0.1 & -0.2 \\ -0.2 & 0.1 \end{bmatrix} \begin{bmatrix} x_{t-1,1} \\ x_{t-1,2} \end{bmatrix} + \begin{bmatrix} -0.4 & 0.2 \\ 0.2 & -0.4 \end{bmatrix} \begin{bmatrix} \varepsilon_{t-1,1} \\ \varepsilon_{t-1,2} \end{bmatrix} + \begin{bmatrix} \varepsilon_{t,1} \\ \varepsilon_{t,2} \end{bmatrix}$$

- Square multivariate (SQ)

$$x_{t,1} = 0.6x_{t-1} + \varepsilon_{t,1}$$
$$x_{t,2} = x_{t,1}^2 + \varepsilon_{t,2}$$

- Exponential multivariate (EXP)

$$x_{t,1} = 0.6x_{t-1} + \varepsilon_{t,1}$$
$$x_{t,2} = \exp(x_{t,1}) + \varepsilon_{t,2}$$

For the multivariate time series (VARMA, SQ, EXP), the second index of $x_{t,i}$, $i \in \{1, 2\}$, refers to the individual components.

B.3 Description of benchmark datasets

m4 Stemming from the Makridakis Competitions Makridakis et al. (2018) (see `https://en.wikipedia.org/wiki/Makridakis_Competitions` for more information), the m4 dataset contains 414 time series of hourly data. Every time series has a different starting point and a length of 748 hours. To allow for multivariate prediction, we take a subset of ten times series. overlap. We further take differences with a period of one and 24 hours to improve stationarity and reduce seasonal effects, respectively.

Traffic The traffic dataset can be downloaded from `https://archive.ics.uci.edu/ml/datasets/PEMS-SF`. It consists of 963 car lane occupancy rates with values between 0 and 1 taken from freeways in the San Francisco bay area. Time series start on the first of January 2008 and last until March 30 2009 with an observation frequency of 10 minutes. To condense the information, an hourly aggregation is used Yu et al. (2016), yielding time series of length 10,560. We use the first ten time series and observations until '2008-06-22 23:00:00', yielding a total of 4,167 observations per lane. We further apply seasonal differencing with a seasonal period of 24 hours and take first differences to reduce non-stationary behavior.

Electricity The electricity dataset can be downloaded from `https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014`. The dataset consists of electricity consumption (kWh) time series of 370 customers. Values correspond to electricity usage in a frequency of 15 minutes. In our benchmarks, we aggregate the values to hourly consumption (see also Yu et al. (2016) for justification of this approach). We use a subset of ten customers and a time range from '2014-01-01 00:00:00' to '2014-09-07 23:00:00', yielding a total of 6,000 observations per customer . We further apply seasonal differencing with a period of 24 hours to reduce seasonal effects and take the first differences for stationarity reasons.

Exchange The dataset was made available by Lai et al. (2018). The time series are exchange rates from 8 countries for days between January 1990 and May 2013. We calculate the returns of the exchange rates to receive a stationary time series.

Moving MNIST The dataset can be downloaded from `https://www.cs.toronto.edu/~nitish/unsupervised_video/`. It contains $10,000$ video sequences. Each sequence consists of 20 frames that show two digits moving with a random speed and direction. The digits move independently, but intersect from time to time and bounce off the edges of the frame. The resolution of the frames is (64×64) pixels and the monochrome light intensity is encoded as an 8-bit integer. We draw a random subset of $1,000$ sequences from this dataset for our experiments.

Taxi New York City and Bike New York City The datasets are available at `https://github.com/haoxingl/DSAN`. The goal is crowd flow prediction on a given spatial window in new york city. The Taxi-NYC dataset was originally taken from NYC-TLC (`https://www1.nyc.gov/site/tlc/index.page`) and the Bike-NYC dataset from Citi-Bike (`https://www.citibikenyc.com/`). Both datasets consist of 60 days of trip records. For every trip, the start and end location and time is included.

For all datasets except the Taxi New York City and Bike New York City we use the first 70% for training, and the remaining 30% for testing the model. For the neural network models, 30% of the training data is used for validation. The two New York datasets are already split in training (1920 hours), validation (576 hours), and testing (960 hours) data.

B.4 Architectures and search space

For uni- and multivariate time series, all neural networks contain one to two RNN layers of the respective RNN cell, yielding the shallow and deep versions of the models, respectively. The cells in each layer contain one to five units with a rectified linear activation function. In the ShallowARMA model, one cell is activated linearly as shown in Figure 2, resembling a hybrid model. A final fully connected layer with linear activation and appropriate output shape is used to match the dimensions of the time series. The lag values p and q are chosen from the interval $[1, 4]$. The loss function of all models is the mean squared error function. For training, the Adam optimizer is used in combination and early stopping callback to prevent overfitting. For all other model properties, the default values are used. For tensor-variate time series, batch normalization layers are added between the RNN layers, and an adaptive learning is added to improve convergence. Each layer contains 64 filters, so a 2D convolution is added to reduce the number of channels appropriately.

Table 5: Comparisons of different methods (rows) and different data generating processes (columns) for univariate time series using the average MAE ± the standard deviation of 10 independent runs. The best performing method is highlighted in bold, the second-best in italics.

model	ARMA	TAR	SGN	NAR	Heteroskedastic
ARMA	1.62±0.29	2.39±2.66	1.98±2.35	2.28±4.42	0.95±0.11
ShallowARMA	**1.55±0.07**	**0.84±0.06**	0.88±0.06	0.81±0.04	**0.88±0.05**
DeepARMA	1.57±0.08	0.96±0.38	**0.84±0.05**	0.81±0.04	*0.88±0.05*
LSTM	1.57±0.07	1.08±0.36	0.96±0.15	*0.81±0.04*	0.92±0.08
DeepLSTM	1.60±0.08	1.12±0.42	0.94±0.12	0.81±0.04	0.92±0.07
GRU	*1.56±0.09*	0.98±0.22	0.88±0.07	**0.81±0.04**	0.90±0.06
DeepGRU	1.58±0.07	*0.95±0.26*	*0.87±0.10*	0.81±0.04	0.90±0.05
Simple	1.58±0.08	0.99±0.22	0.91±0.08	0.83±0.04	0.90±0.06
DeepSimple	1.60±0.08	1.15±0.46	0.92±0.08	0.82±0.04	0.92±0.08

C Computational environment

All experiments and benchmarks were carried out on an internal cluster. Uni- and multivariate time series were trained on a server with 10 vCPUs, running on an Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz physical CPU and 48Gb allocated memory. Tensor-variate time series were trained on a server with 16 vCPUS, running on an Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz, 32Gb allocated memory, and a Nvidia GeForce RTX 2080 Ti (11Gb).

D Additional simulation and benchmark results

In the following, we provide additional results on numerical experiments by including comparisons based on the mean absolute error (MAE).

Results The results for simulated data suggest that either the ShallowARMA or DeepARMA cell perform best in most cases while on par with the GRU cell for the ARMA and NAR dataset. For the simulated multivariate time series, none of the existing neural methods outperforms the ARMA cells. For the time series benchmark datasets, the ARMA approaches outperform all other RNN approaches on m4 and Traffic. On the Electricity dataset the ARMA cells remain competitive for the univariate case, but yield larger RMSE values compared to GRU and Simple in the multivariate setting. On Exchange all methods perform equally well.

Overall the rankings of methods do not change notably when using the MAE instead of the RMSE as comparison measure.

Table 6: Comparisons of different methods (rows) and different data generating processes (columns) for multivariate time series using the average MAE ± the standard deviation of 10 independent runs. The best performing method is highlighted in bold, the second-best in italics.

MODEL	VARMA	EXP	SQ
VARMA	**0.80±0.02**	1.61±0.13	1.22±0.07
ShallowARMA	*0.80±0.03*	*1.48±0.13*	**1.19±0.06**
DeepARMA	0.81±0.03	**1.48±0.13**	*1.20±0.06*
LSTM	0.81±0.03	1.55±0.13	1.25±0.09
DeepLSTM	0.82±0.03	1.63±0.16	1.29±0.10
GRU	0.81±0.03	1.53±0.14	1.23±0.07
DeepGRU	0.81±0.03	1.54±0.13	1.23±0.09
SIMPLE	0.82±0.02	1.56±0.15	1.23±0.07
DeepSimple	0.82±0.03	1.56±0.17	1.25±0.08

Table 7: Comparison of different univariate and multivariate forecasting approaches (rows) for different datasets (columns) based on the average MAE ± the standard deviation of 10 independent runs. The best performing method is highlighted in bold, the second-best in italics.

		M4	TRAFFIC	ELECTRICITY	EXCHANGE
UNIV.	ARMA	**0.82±0.00**	0.48±0.00	0.77±0.00	0.78±0.00
	ShallowARMA	*0.83±0.00*	0.48±0.00	0.74±0.01	0.67±0.00
	DeepARMA	0.84±0.01	**0.45±0.01**	*0.70±0.02*	**0.67±0.00**
	LSTM	0.88±0.03	*0.45±0.00*	0.72±0.03	*0.67±0.00*
	DeepLSTM	0.98±0.10	0.45±0.00	**0.69±0.02**	0.67±0.00
	GRU	0.86±0.02	0.45±0.01	0.70±0.02	0.67±0.00
	DeepGRU	0.86±0.02	0.45±0.01	0.70±0.01	0.67±0.00
	Simple	0.91±0.06	0.46±0.00	0.71±0.01	0.67±0.00
	DeepSimple	0.93±0.07	0.46±0.01	0.70±0.01	0.67±0.00
MULTIV.	ARMA	*0.82±0.00*	0.48±0.00	0.77±0.00	0.78±0.00
	ShallowARMA	**0.82±0.01**	0.53±0.01	0.78±0.01	0.68±0.01
	DeepARMA	0.83±0.00	0.53±0.02	0.77±0.03	0.68±0.00
	LSTM	0.97±0.04	0.49±0.01	0.93±0.20	**0.67±0.00**
	DeepLSTM	1.06±0.08	0.49±0.02	0.74±0.03	0.69±0.06
	GRU	0.97±0.09	0.48±0.00	0.73±0.02	0.67±0.00
	DeepGRU	0.97±0.06	0.48±0.01	*0.72±0.02*	*0.67±0.00*
	Simple	0.99±0.03	**0.47±0.01**	0.73±0.02	0.67±0.00
	DeepSimple	1.01±0.05	*0.47±0.01*	**0.70±0.01**	0.67±0.00