Sales Forecasting with Partial Recurrent Neural Networks: Empirical Insights and Benchmarking Results

Moritz Müller-Navarra  
Hamburg University of Technology  
muellernavarra@gmail.com

Stefan Lessmann  
Institute of Information Systems  
Humboldt-University of Berlin  
stefan.lessmann@hu-berlin.de

Stefan Voß  
Institute of Information Systems  
University of Hamburg  
stefan.voss@uni-hamburg.de

Abstract
Partial Recurrent Neural Networks (PRNN) belong to the family of Artificial Neural Networks. Due to their specific architecture, PRNN are well-suited to forecast time series data. Their ability to outperform well-known statistical forecasting models has been demonstrated in some application domains. However, the potential of PRNN in business decision support and sales forecasting in particular has received relatively little attention. The paper strives to close this research gap. In particular, the paper provides a managerial introduction to PRNN and assesses their forecasting performance vis-à-vis challenging statistical benchmarks using real-world sales data. The sales time series are selected such that they encompass several characteristic patterns (e.g., seasonality, trend, etc.) and differ in shape and length. Such heterogeneity is commonly encountered in sales forecasting and facilitates a holistic assessment of PRNN, and their potential to generate operationally accurate forecasts.

1. Introduction

Sales forecasting is an integral part of operations and supply chain management [26]. For example, manufacturers require an accurate estimate of future demand to inform production planning [15, 17]. In the same way, retailers use sales forecast to guide purchasing decisions and minimize capital binding costs [5, 16]. Finally, in supply chain management, forecasting is commonly employed as a lens through which to study the business value of sharing (demand) information between retailers and manufacturers. That is, information related to the demand of end-customers helps manufacturers to boost their forecasting accuracy and reduce the well-known bull-whip effect [10, 20, 27].

Depending upon the nature of the business, the sales forecasting task can be performed by human planners, statistical models, or a combination thereof [18]. This paper focuses on statistical forecasting models and examines the potential of a family of methods called Partial Recurrent Neural Networks (PRNN), which have up to know received relatively little attention in the sales forecasting literature.

Developing and evaluating technologies that support managerial decision making are established topics in the Information Systems (IS) literature. Recently, concepts such as Big Data Analytics have received much attention. In fact, they represent one of the major trends in IS (and beyond) today, and might substantially change the face of corporate decision support as we know it [11]. This paper is in line with such concepts in the sense that we propose a methodology that extracts systematic patterns from past sales and facilitates predicting future sales.

In our analysis, we draw inspiration of Hevner’s et al. design science paradigm [9]. Advanced PRNN forecasting models represent our design artefact. The main focus of the paper is a rigorous assessment of this artefact as a tool for business planning. To that end, we perform an empirical benchmark, which is the prevailing approach in forecasting [18].

A variety of forecasting models are available. Our motivation to focus on PRNN is threefold: first, compared to conventional feed-forward neural networks, PRNN have received less attention in the literature and in business forecasting in particular. Second, their specific architecture makes them well-suited to handle time series data (i.e., the standard format of sales data). We explain this feature in Section 2. Third, recent studies demonstrate that PRNN can be a highly effective forecasting method in domains such as Electricity Consumption and Wind Speed [19, 4]. This indicates that they may also be useful for business forecasting [7, 21]. For example, business applications in the retail industry often require a large number of time series to be
forecast. This requires a technique that automatically adjusts to a given time series. As we demonstrate in our study, PRNN possess this ability. Furthermore real-world sales series often exhibit non-linear patterns (due to seasonality, trend, the introduction of new product models, etc.). Classical statistical time series methods such as exponential smoothing or Box/Jenkins models are unable to accommodate nonlinearity directly. Standard feed-forward neural networks, on the other hand, are well-adapted to model nonlinear patterns, but are less suited to account for time-related dependencies. PRNN offer both features. They account for nonlinearity and are well-suited to model time series data [4, 7, 8, 14, 21].

The primary objective of this paper is thus to examine the potential of PRNN for sales forecasting. To that end, we compare different types of PRNN to established benchmarks (exponential smoothing and SARIMA) using real-world sales data. Given that parameter tuning is a critical issue in neural network modeling [25] and important to automatically forecast a large number of time series, a secondary objective of the paper is to investigate the sensitivity of different types of PRNN with respect to parameter settings and to demonstrate an automatic parameter tuning method.

The paper is structured as follows: We introduce PRNN in Section 2. Next, we describe our experimental setup and report empirical results in Section 3. Finally, Section 4 concludes the paper.

2. Partial Recurrent Neural Networks

In the following we review different types of PRNN and describe how they facilitate forecasting monthly sales several steps ahead. In general, an artificial neural network is an information processing system that strives to mimic the process of human decision making. Consider, for example, a computer wholesaler such as Dell. To fulfill future customer orders, Dell needs to decide how many hard-drives to purchase in a given planning period. To that end, the planner in charge will typically first gather some relevant pieces of information (e.g., how many computers were sold last month, etc.) and then make a decision. Neural networks operate in a similar way. They receive some input information, process this information, and then produce a corresponding output value. However, prior to application, a neural network needs to ‘learn’ how to perform a certain task. To that end, one provides the network with training data that comprises pairs of inputs and corresponding output values. The network then uses these examples together with a mathematical algorithm to minimize the difference between network outputs and actual outputs.

In sales forecasting, we can use past sales to form training examples. For example, the input information for the network could consist of sales in some period $t$. The sales in the subsequent period $t+1$, could then give the corresponding (actual) output value; whereby we assume that sales in $t$ and $t+1$ have been observed in the past. Note that this approach assumes that past sales determine future sales to some extent. Therefore, it is important that the neural network is aware of the temporal relationship between inputs and outputs. PRNN achieve this through their special architecture as shown below. A PRNN consists of simple cells called neurons, which exchange information through weighted links with other neurons [24]. Every neuron has a certain activation value, which is computed by an activation function. We use the logistic function (1) for this purpose. The resulting output value of a neuron is then passed to connected neurons in subsequent layers.

$$f_{act}(net_j, \theta_j) = \frac{1}{1 + e^{-(net_j - \theta_j)}}$$

(1)

In (1), $net_i$ denotes to the propagation function and $\theta_i$ is a threshold value. The propagation function defines how the input of a neuron is computed from the output value of connected neurons and connection weights. We define $net_j$ as follows (2):

$$net_j(t) = \sum_i a_i(t) \cdot w_{ij}$$

(2)

Here, $a_i(t)$ denotes the output value of neuron $i$ at the time step $t$. $a_i$ is the activation value computed with the logistic function (1), and $w_{ij}$ the connection weight between neuron $i$ and $j$. Depending on the characteristics of the network connections, the neurons form different layers. PRNN obtain at least one context-layer, which serves as the memory of the network [24]. The different layers in such network are used for different tasks. The input-layer receives input from the environment and forwards the input to the hidden-layer (see Figures 1–3). The neurons in the hidden-layer also receive input from the context-layer. The output-layer then receives input from the hidden-layer and is accessible from the environment. This processing of an input signal is called forward pass. After performing the forward pass, the neurons in the context-layer compute their new states or context depending on their input. When a new input signal arrives, the context is passed depending on the
The neurons in the hidden-layer use the logistic function \( f \) and the propagation function \( g \) to compute their activation value. The neurons in the other layers simply use the propagation function \( g \) to compute their input, which is then equal to their activation value.

In this paper, we consider three different PRNN, namely Elman, Jordan, and multi-recurrent networks [8, 14, 28]. Figure 1 depicts the structure of Jordan Networks. The context-layer receives input from the output-layer and from itself (self-recurrent).

Elman Networks (Figure 2), differ from Jordan Networks in that the context-layer receives input not from the output- but from the hidden-layer. Note that we can design Hierarchical Elman Networks, where the number of hidden-layers can be varied [7, 21].

It is possible to design PRNN with a context-layer for every layer (input, hidden, output). Figure 3 depicts a design where a context-layer is used to save past inputs. This architecture is later referred to as multi-recurrent [28]. For all network types, the context-layer (hidden) possesses a self-recurrent connection \( \lambda \) with a constant weight. This weight determines the value of the context-neurons and defines the memory of the network. A high \( \lambda \) leads to a higher importance of further back context values in time than a smaller \( \lambda \). Thus, choosing proper values for \( \lambda \) facilitates controlling the memory of the network.

In order to use PRNN for sales forecasting, we consider a recursive approach combined with a time window [1]. That is, we train a network with defined parameters and architecture (neurons in different layers) to forecast one period into the future using the \( Rprop \) algorithm [23]. Each neuron in the input-layer receives a certain lagged realization of the sales time series \( y \). The number of input neurons \( d \) scales linear with the time lags. For example, when using sales data of the past five months as input to forecast sales in the sixth month, the input-layer contains five neurons. The output-layer has exactly one neuron, which provides the one-period-ahead forecast. Then, we shift the window of size \( d \) along the time series by one time step. After training the network using this approach, we can employ it to produce multi-step-ahead forecasts. With \( d \) neurons in the input layer, \( h \) horizons (steps ahead), \( y_N \) as the most recent monthly sale and \( \hat{y}_{N+h} \) as the predicted value \( h \) steps ahead, we obtain (3):

\[
\hat{y}_{N+h} = \begin{cases} 
  f(y_{N}, \ldots, y_{N-d+1}), h = 1 \\
  f(\hat{y}_{N+h-1}, \ldots, \hat{y}_{N+d}), h \in \{2, \ldots, d\} \\
  f(\hat{y}_{N+h-1}, \hat{y}_{N+h-d}), h \in \{d+1, \ldots, H\}
\end{cases}
\]
For $h$ greater than one (multi-steps), the next input set contains the output of the net from past time step(s).

Prior to using PRNN, we preprocess our sales time series, as is common practice in time series analysis [6]. As shown in (4), we assume that a sales time series $y$ consists of three different signals, and use standard techniques for time series decomposition [6] to isolate these signals.

$$y = f(trend, season, remainder) \quad (4)$$

The PRNN are trained with the seasonally adjusted series. We transform the adjusted series (trend and remainder) into the range of the function (logistic function) used by the neurons of the hidden-layer to process input signals. We achieve this by dividing all elements by their maximum. Note that seasonal effects are later added for the final error analysis.

2.2. Defining and Choosing the Parameters

For every network type, we need to set certain parameters. These are, i) the number of hidden neurons, ii) the value of the self-recurrent-link ($\lambda$), iii) the interval of the random initialization of the links, iv) the activation value of the context neurons, and v) the iterations of (Rprop).

To determine suitable parameter values, we form a parameter grid and fit networks with all possible parameter combinations on a given sales time series. Since the networks later forecast several steps ahead, the given series must be split into different records. The last 15 monthly sales of the series are used to compute the real forecast-errors. The remaining data is called #train, from which the #validate record is formed to serve as an internal validation record. Depending on the number of input-neurons $d$ the records are (5)

$$#train = series - 15$$
$$#validate = (#train - d) \times 0.15$$
$$#test = 15 \quad (5)$$

Using the PRNN technique, all elements of the records must follow the correct order in time. Then, we execute the following steps for every time series and network-type.

1. Decomposition. Removing the season
2. Scale data
3. Split data into #train, #validate, and #test
4. Use Rprop on #train without #validate
5. Use all networks to predict one step ahead along #validate
6. Select the parameter-combination with lowest error
7. Train selected net with Rprop using #train+#validate
8. Final check of the overall performance of the net
9. Forecast with selected net.
10. Compare forecasts with #test

3. Experimental Setup

To evaluate the effectiveness of PRNN for corporate decision support and sales forecasting in particular, we compare the empirical accuracy of PRNN forecasts to two established benchmark models: SARIMA [3] and exponential smoothing. The comparison includes six monthly time series of different length and shape. To decide on the specific time series model for exponential (trend, seasonal, trend-seasonal, etc.), we consider the approach of Hyndman et al. [12, 13], referred to as ETS in the following. As a benchmark for all models (PRNN, SARIMA and ETS) we use a seasonal naive model (N), where the forecasted value is equal to the last value 12 months ago. All models (SARIMA; ETS, PRNN and N) will forecast 1 to 15 steps ahead.

Automatic modeling of ETS and SARIMA can be realized through the R-Software [22] and the package forecast [13] by using the functions ets() and auto.arima(), respectively. We obtain the signals (trend, season and remainder) by using the stl() function from the base package [22]. The networks are trained using the package RSNNS [2]. Recall that we deseasonalize all our sales time series prior to applying the PRNN [29]. We initialize the weights between the neurons randomly in the interval [-1; 1]. Their value will be changed during training. We vary $\lambda$ in the interval [0.1, 0.2…, 1]. We set the value of $\eta$ to 1. We initialize the activation value of the context-neurons with 0.1 at first, adapting this setting when the training process was not satisfying. We chose the number of neurons in the hidden-layer from the interval [1; …; 20] for Jordan and Elman Networks, and from the interval [1; …; 10] for the multi-recurrent-Networks. The Hierarchical Elman Networks have three hidden-layers with neurons from the interval [1; …; 10].

For the experiment, we use six different monthly sales series (a-f). This results in training and analyzing approximately 1200 Elman, 1200 Jordan, 600 multi-recurrent and 60000 Hierarchical Elman Networks. Table 1 shows the number of observations in the different time series partitions per time series.
Table 1. Records of the time series a–f

<table>
<thead>
<tr>
<th></th>
<th>#train (#validate)</th>
<th>#test</th>
<th>#total</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>346 (49)</td>
<td>15</td>
<td>361</td>
</tr>
<tr>
<td>b</td>
<td>161 (22)</td>
<td>15</td>
<td>176</td>
</tr>
<tr>
<td>c</td>
<td>150 (21)</td>
<td>15</td>
<td>165</td>
</tr>
<tr>
<td>d</td>
<td>93 (11)</td>
<td>15</td>
<td>108</td>
</tr>
<tr>
<td>e</td>
<td>81 (10)</td>
<td>15</td>
<td>96</td>
</tr>
<tr>
<td>f</td>
<td>62 (8)</td>
<td>15</td>
<td>77</td>
</tr>
</tbody>
</table>

We train the PRNN on the #train records at first with 2000 Iterations using Rprop with given parameters. After training is completed, we compare the generalization capabilities of the trained networks on the #validate records. We perform one-step-ahead forecasts using the #validate records and the trained networks. Afterwards we compare the symmetrical mean percentage errors (smape) (6), whereby \(y_t\) represents the real value of the record and \(\hat{y}_t\) the corresponding forecast, of the different parameter-combinations on the #validate records of the network types.

\[
\text{smape} = \frac{1}{n} \sum_{t=1}^{n} \frac{|y_t - \hat{y}_t|}{y_t + \hat{y}_t} \cdot 200
\]  

We then chose the parameter combination with minimal error. Afterwards we merge the #train and #validate records to form a single record. We use this record to train the chosen parameter combinations of the network types. To ensure a fair comparison of different forecasting methods, we use the same record to parameterize SARIMA and ETS. The #test records are not used for any optimization by any model. The final errors are computed by comparing the \(h\)-steps forecast with the actual value of the #test records.

3.1. Results of the model selection stage

The error during the selection process results after the training was finished. The candidate networks perform one-step-ahead forecasts along the validation records. They therefore operate on unknown data, but get the real value of past data as input for the next forecasts. Since this step aims at deciding on network parameters, we chose only one network per network type for subsequent comparisons with the benchmark models. Table 2 depicts the number of neurons in the different layers of the chosen networks.

Table 2. Number of neurons in the layers per network type and series

<table>
<thead>
<tr>
<th></th>
<th>in</th>
<th>out</th>
<th>HE</th>
<th>E</th>
<th>J</th>
<th>MR</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>20</td>
<td>1</td>
<td>5-2-4</td>
<td>1</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>20</td>
<td>1</td>
<td>2-1-5</td>
<td>14</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>10</td>
<td>1</td>
<td>7-2-3</td>
<td>19</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>20</td>
<td>1</td>
<td>4-3-2</td>
<td>19</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>20</td>
<td>1</td>
<td>2-4-4</td>
<td>12</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>f</td>
<td>10</td>
<td>1</td>
<td>4-1-4</td>
<td>14</td>
<td>20</td>
<td>1</td>
</tr>
</tbody>
</table>

HE (Hierarchical Elman Networks) shows a variation of combinations in the three different context-layers. The context-layer of the Elman Networks (E) contain, apart from the one chosen at series a, at least 12 neurons. The Jordan Networks (J) also contains a larger number of neurons in this layer. A network with only one neuron in the context-layer was chosen at series c. The multi-recurrent Networks (MR) tend to contain lesser neurons in the context-layer. Table 3 gives an overview of the chosen parameters \(\lambda\) (self-recurrent links) and \(\psi\) (initialization of the context units), as well as the iterations of Rprop.

Table 3. Self-recurrent weights, initialization of the context-neurons and iterations of Rprop per network type and series

<table>
<thead>
<tr>
<th></th>
<th>HE</th>
<th>E</th>
<th>J</th>
<th>MR</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3;1;2</td>
<td>10;1;2</td>
<td>7;1;2</td>
<td>10;0.1;1.5</td>
</tr>
<tr>
<td>b</td>
<td>3;5;2</td>
<td>5;1;2</td>
<td>8;1;2</td>
<td>6;4;0.2</td>
</tr>
<tr>
<td>c</td>
<td>3;2;1</td>
<td>7;1;1.65</td>
<td>2;1;2</td>
<td>4;1;0.1</td>
</tr>
<tr>
<td>d</td>
<td>7;1;1</td>
<td>5;1;1</td>
<td>1;1;1</td>
<td>4;1;0.05</td>
</tr>
<tr>
<td>e</td>
<td>3;1;2</td>
<td>9;1;2.5</td>
<td>5;1;0.3</td>
<td>7;1;0.35</td>
</tr>
<tr>
<td>f</td>
<td>1;1;2</td>
<td>4;1;0.7</td>
<td>1;1;0.35</td>
<td>2;1;1.05</td>
</tr>
</tbody>
</table>

(\(\lambda; \psi\); Iterations) in (0.1:0.1;1000)

3.2. Results on the test records

The overall results across all time series and forecasting horizons are shown in Table 4. Here, we use smape for every model and time-series and divide it by six to compare the overall performance. The ets() function (ETS) gives the best overall forecasts (lowest mean smape). HE achieves the second rank followed by MR and E. SARIMA (SA) is ranked fifth, followed by J. The naïve forecasts (N) perform worst (last rank).
Table 4. Comparison of errors by the different models over all series

<table>
<thead>
<tr>
<th>Error (mean smape)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETS</td>
</tr>
<tr>
<td>HE</td>
</tr>
<tr>
<td>MR</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>SA</td>
</tr>
<tr>
<td>J</td>
</tr>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

The errors (mean smape) of the networks by time series a-f are shown in Table 5. SA has the lowest error on series a. On series b, HE outperforms the other networks and the statistical models. The best forecasts for series c and d are given by ETS and E, respectively. Once again, ETS outperforms all other models on series e and f.

Table 5. Comparison of errors by the models for the series a-f

<table>
<thead>
<tr>
<th>HE</th>
<th>E</th>
<th>J</th>
<th>MR</th>
<th>SA</th>
<th>ETS</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>8.3</td>
<td>8.3</td>
<td>8.1</td>
<td>8.3</td>
<td>7.7</td>
<td>7.7</td>
</tr>
<tr>
<td>b</td>
<td>5.7</td>
<td>7.3</td>
<td>10.9</td>
<td>6.8</td>
<td>10.8</td>
<td>7.2</td>
</tr>
<tr>
<td>c</td>
<td>9.5</td>
<td>9.7</td>
<td>9.9</td>
<td>9.9</td>
<td>8.9</td>
<td>7.7</td>
</tr>
<tr>
<td>d</td>
<td>10.3</td>
<td>7.6</td>
<td>8.6</td>
<td>11.2</td>
<td>10.8</td>
<td>9.1</td>
</tr>
<tr>
<td>e</td>
<td>12.9</td>
<td>11.4</td>
<td>20.1</td>
<td>12.6</td>
<td>16.0</td>
<td>9.8</td>
</tr>
<tr>
<td>f</td>
<td>18.2</td>
<td>24.5</td>
<td>23.3</td>
<td>18.2</td>
<td>15.9</td>
<td>11.2</td>
</tr>
</tbody>
</table>

ETS shows very good forecasts on all series, but the PRNN partly got better forecasts than the statistical models (SA and ETS) on series b and d. The errors seem to relate to the length of the time series. It is useful to look at a plot (Figure 4), where the summands of the error (smape) is plotted against the length (here we took the double natural logarithm) of the time series.

Figure 4 suggests that the highest errors occur at the shorter time series for all network types, which leads to the conclusion that the length of the time series has an influence on the goodness of forecasts using the above described technique.

Figure 5 illustrates the multi-step forecasts of the PRNN and the test records of series f without added seasonal factors. The first peak at h=3 is anticipated by all networks. The Jordan Network provides very accurate forecasts up to h=5. The following forecasts were too low, which results in a high test-error. HE forecasted an ascending trend for the remaining horizons, but leveling to low and incorrectly descending after the 13th step. MR and E show similar predictions, with E subject to a higher level. From the 12th step, both methods incorrectly predict a deceasing trend.

Figure 6 depicts the multi-step forecasts of the PRNN and the test records of series f without added seasonal factors. The PRNN could obviously learn the underlying structural relationships of the time series to some extent. The Jordan and Elman Networks show good forecasts. The last two months are still forecasted far too low by all networks, except Jordan Networks.
Both plots (Figure 5 and Figure 6) illustrate that all network types forecasted the structure of the test records to some extent. We provide the errors (mean smape) on the test records without adding the seasonal values, thus demonstrating the real forecasted errors of the network types. We provide these errors for every network-type and series in Table 6. This allows us to re-check the forecasting capabilities of the trained networks. HE and E have almost the same values (9.89, resp. 9.90). MR has a mean smape of 10.53; J ranks last with a mean smape of 11.43.

Table 6. Comparison of errors by the network types for the series a-f before the season values was added

<table>
<thead>
<tr>
<th></th>
<th>HE</th>
<th>E</th>
<th>J</th>
<th>MR</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>9.62</td>
<td>7.46</td>
<td>8.04</td>
<td>11.19</td>
</tr>
<tr>
<td>b</td>
<td>7.71</td>
<td>7.77</td>
<td>7.54</td>
<td>7.76</td>
</tr>
<tr>
<td>c</td>
<td>5.74</td>
<td>7.57</td>
<td>10.77</td>
<td>7.03</td>
</tr>
<tr>
<td>d</td>
<td>9.02</td>
<td>9.21</td>
<td>9.27</td>
<td>9.45</td>
</tr>
<tr>
<td>e</td>
<td>11.51</td>
<td>9.35</td>
<td>15.93</td>
<td>11.34</td>
</tr>
<tr>
<td>f</td>
<td>15.75</td>
<td>18.02</td>
<td>17.01</td>
<td>16.42</td>
</tr>
<tr>
<td>mean smape</td>
<td>9.89</td>
<td>9.90</td>
<td>11.43</td>
<td>10.53</td>
</tr>
</tbody>
</table>

We compare the forecast-errors (smape) of the PRNN, SARIMA, ETS and naïve for different horizons in Figure 7, again taking the mean over all time series.

The PRNN perform relatively well on smaller horizons with SARIMA and ETS performing better on later horizons. These models seem to be more stable for longer horizons than the trained networks (see Figure 7).

4. Conclusions

We examined the task of sales forecasting with different statistical models and PRNN. For this purpose, six different monthly sales time series were used to monitor the forecasts of the PRNN and to compare them to forecasts of statistical models. We show that PRNN can outperform statistical models on certain series. However, a general superiority of PRNN was not detected. The influence of experimental factors such as the length of the time series and the network type was also examined. The results indicate better forecasts for longer series containing more than 100 monthly observations. We also observed that Hierarchical Elman Networks perform better than other types of PRNN considered in the study. Finally, we examined ways of finding suitable parameters, which can be used to forecast monthly sales multi-steps ahead. The results evidence the difficulties in parameter selection and how they can be overcome using our grid-search approach. In business applications, the proposed PRNN-based forecasting approach can be implemented using the described software in such a way that the only input needed is the time series. The forecasts are then generated fully automatically. The training and validation process runs as a background task and the planner receives the desired forecasts for various horizons, which can be compared to currently preferred forecasting models.

In summary, our study illustrates the potential of PRNN for sales forecasting and provides evidence that they are a viable alternative to other forecasting methods. However, future research is needed to replicate our results on a large set of sales time series and, more generally, time series related to operational planning.
6. References


