# Ensembles of Recurrent Neural Networks for Robust Time Series Forecasting

Sascha Krstanovic(✉) and Heiko Paulheim(✉)

Research Group Data and Web Science,
University of Mannheim, Mannheim, Germany
{sascha,heiko}@informatik.uni-mannheim.de

**Abstract.** Time series forecasting is a problem that is strongly dependent on the underlying process which generates the data sequence. Hence, finding good model fits often involves complex and time consuming tasks such as extensive data preprocessing, designing hybrid models, or heavy parameter optimization. Long Short-Term Memory (LSTM), a variant of recurrent neural networks (RNNs), provide state of the art forecasting performance without prior assumptions about the data distribution. LSTMs are, however, highly sensitive to the chosen network architecture and parameter selection, which makes it difficult to come up with a one-size-fits-all solution without sophisticated optimization and parameter tuning. To overcome these limitations, we propose an ensemble architecture that combines forecasts of a number of differently parameterized LSTMs to a robust final estimate which, on average, performs better than the majority of the individual LSTM base learners, and provides stable results across different datasets. The approach is easily parallelizable and we demonstrate its effectiveness on several real-world data sets.

**Keywords:** Time series · Ensemble · Meta-learning · Stacking · ARIMA · RNN · LSTM

## 1 Introduction

Tracking and logging information that is related to a timely dimension has been important in a variety of sectors such as energy, biology or meteorology. Using this data in order to estimate the future behavior is of high value since it has an immediate impact on decision making. Hence, time series forecasting is an established research field. State of the art solutions, among others, include recurrent neural networks, which have been shown to be very powerful for time series forecasting.

On the other hand, recurrent neural networks are not easy to configure for a given use case at hand. Configurations that work well for one setting can be sub-optimal for another problem. To account for that problem, we propose an approach which trains many recurrent neural networks with different parameter


M. Bramer and M. Petridis (Eds.): SGAI-AI 2017, LNAI 10630, pp. 34–46, 2017.
https://doi.org/10.1007/978-3-319-71078-5_3

settings and combine their forecasts using ensemble methods. With that approach, we can provide robust results and circumvent the problem of finding the optimal parameters for a given dataset.

The rest of this paper is structured as follows. Section 2 gives an introduction to important concepts of time series analysis and ensemble learning that are essential for the further sections. Section 2.2 introduces Long Short-Term Memory, a central algorithm used in this work. We propose a concrete time series ensemble architecture in Sect. 3 and validate its performance in the subsequent section, as well as discussing implications and limitations of the approach. We show that the stacked LSTM forecasts beat, on average, the majority of the base learners in terms of root mean squared error (RMSE). Finally, areas holding potential for further improvement are outlined in Sect. 5.

## 2 Background and Related Work

Although a time series can formally be straightforwardly defined as "a set of observations $y_t$, each one being recorded at a specific time $t$" [21], it has a number of important characteristics with implications for data sampling, model training, and ensemble architecture.

### 2.1 Properties of Time Series Data

Time series forecasting is a special case of sequence modeling. This implicitly means that the observed values correlate with their own past values. The degree of similarity of a series with a lagged version of itself is called autocorrelation. As a consequence, individual observations can not be considered independently of each other, which demands the right sampling strategies when training prediction models. Autocorrelation leads to a couple of special time series properties; first and foremost, stationarity. A series $Y$ is called stationary if its mean and variance stay constant over time, i.e., the statistical properties don't change [20]. Among other algorithms, autoregressive (AR) models theoretically assume stationarity.

Two further important properties are seasonality and trend. Seasonality means that there is some pattern in the series which repeats itself regularly. For example, the sales of ice cream in a year are higher in the summer months and decrease in the winter. Therefore, the seasonal period is fixed and known. We speak of a trend if a general change of direction in the series can be observed, for example if the average level of the series is steadily increasing over time. Identifying and extracting components like seasonality and trend is essential when dealing with state space model algorithms.

### 2.2 State of the Art Forecasting Algorithms

Very diverse application possibilities have been ensuring high interest in time series analysis and forecasting for decades. An extensive overview is given in [19]. One can arguably divide the majority of approaches to time series forecasting into two categories: autoregressive models for sequences generated by a linear process and artificial neural networks (ANNs) for nonlinear series.

**Autoregressive Models**
For time series that are generated by a linear process, autoregressive models constitute a popular family of algorithms used for forecasting, in particular, the Box-Jenkins autoregressive integrated moving average (ARIMA) model [18] and its variants. It performs well especially if the assumption that the time series under study is generated from a linear process is met [7], but it is generally not able to capture nonlinear components of a series. The ARIMA model has several subclasses such as the simple autoregressive (AR), the moving average (MA) and the autoregressive moving average (ARMA). ARIMA generated forecasts are composed of a linear combination of most recent series values and their past random errors, cf. [18] for mathematical details.

**Artificial Neural Networks: Long Short-Term Memory**
Autoregressive models are usually not suited for nonlinear time series. In this case, ANNs are the better alternative since they are capable of modeling nonlinear relationships in the series. In fact, ANNs can approximate any continuous function arbitrarily well [4]. Recurrent neural networks (RNNs) are naturally suited for sequence modeling; we can think of them as forward networks with loops in them. [12] provides a detailed explanation of the various neural network architectures and their applications.

Although traditional RNNs can theoretically handle dependencies of a sequence even over a longer time interval, this is practically very challenging. The reason for this is the problem of vanishing or exploding gradients [13]. When training an RNN with hidden layers for a series with long-term dependencies, the model parameters are learned with the backpropagation through time and gradient descent algorithms. These gradient calculations imply extensive multiplications due to the chain rule, and this is where gradients tend to vanish (i.e., approach a value of zero) or explode. LSTM [1] overcomes the problem of unstable gradients. A coherent mathematical view on this is given in [15].

**Hybrid Approaches**
Since autoregressive models work well for linear series and ANNs suit nonlinear cases, it holds potential to use the two in combination. There have been several studies combining ARIMA and ANN models in a hybrid fashion [5,6,10,11]. In these approaches, an ARIMA model is used to model the linear component of the series, while an ANN captures the nonlinear patterns.

### 2.3 Approaches to Ensemble Learning

A comprehensive introduction to ensemble learning is given in [8]. Generally, there are different ways to combine a number of estimates to a final one. One popular approach known as *bagging* works by drawing $N$ random samples (with replacement) from a dataset of size $N$. This is repeated $m$ times such that $m$ datasets, each of size $N$, are collected. A model is then trained on each of these data sets and the results are averaged (in case of a nominal outcome, the majority vote is taken). The goal here is to reduce variance. A highly popular

and effective algorithm that incorporates bagging ideas is called Random Forest [16]. It extends bagging in the sense that also feature selection is randomized.

In the context of time series forecasting, bagging can not be applied in the defined manner since the values of a sequence are autocorrelated. Hence, random sampling of observations in order to train a forecasting model is not possible. Rather than that, it is necessary to develop reasonable sampling strategies instead of drawing random bootstrap samples.

*Boosting* constitutes another approach to ensembling. The core idea is that examples that were previously poorly estimated receive higher preference over well estimated examples. The objective is to increase the predictive strength of the model. Thinking of a reasonable sampling strategy in the context of sequence learning is essential for boosting. [17] combines a number of RNNs using a boosting approach for time series forecasting.

A more sophisticated ensemble approach is called *stacking*. In this case, a number of models are learned on various subsets of the data and a meta learner is then trained on the base models' forecasts. A meta-learner can theoretically be any model, e.g. Linear Regression or a Random Forest. The motivation is that the meta-learner successfully learns the optimal weights for combining the base learners, and, as a consequence, produces forecasts of higher quality compared to the individual base learners. Therefore, stacking aims at both reducing variance and increasing forecast quality.

## 3 An LSTM Ensemble Architecture

Finding optimal parameters of an RNN for a given dataset is generally a hard task, also for non-sequential tasks. Additionally, there exist no best parameters that are optimal for all data sets. As a consequence, an LSTM that was trained on a particular data set is very likely to perform poorly on an entirely different time series.

We overcome this problem by proposing an LSTM ensemble architecture. We show that the combination of multiple LSTMs enables time series forecasts that are more robust against variations of data compared to a single LSTM.

### 3.1 LSTM Base Learners and Diversity Generation

The models that are included in an ensemble are called base learners. In this work, we choose a number of differently constructed LSTMs as base learners. It is trivial to see that creating an ensemble of models is only reasonable if the included models sufficiently differ from one other. Otherwise, a single model would yield results of similar quality. In other words, the generated model forecast estimates should all differ significantly from one another. In our approach, diversity is introduced in two ways:

1. When designing the architecture of an LSTM, one crucial decision is the length of the training sequences that are fed into the network. We train one

LSTM for each user-specified length of the input sequences. Since the input sequence length directly affects the complexity of the learning problem, we change the sizes of the hidden layers accordingly. The applied rule is that the number of nodes in the two hidden layers is equal to the sequence length, respectively. For evaluation, we choose $S = \{50, 55, 60, 65, 70\}$ as sequence lengths under consideration.
2. Generally, LSTM expressibility is sensitive to parameter selection and much time-consuming tuning is required. We overcome this by training a number of LSTMs with different values for four parameters: dropout rate, learning rate, number of hidden layers, and number of nodes in the hidden layers. For each of these parameters, a set $\Delta$ of selected values is evaluated. For each of these parameters, we end up with $|S| \cdot |\Delta|$ LSTMs as base learners.

In order to measure the diversity and quality of the base learner forecasts, we compare the average pairwise Pearson correlation coefficients $\rho$ as well as the mean RMSE of the individual sequence forecasts.

**Training LSTMs on Temporal Data**
In order to train an LSTM model for a sequence forecasting problem, it is necessary to split the training data into a number of sequences whose size depends on the input sequence length as well as the forecasting horizon. Given $l$ past time steps that are used in order to forecast the next $k$ values of the series, a sequence $Y$ must be split into sequences of length $k + l$. These sequences are in turn split into two parts, where the first one represents the LSTM input sequence and the second one the target variable. Formally, the original training data $Y_{train} = [y_1, y_2, ..., y_T]$ of the standardized sequence $Y = [y_1, y_2, ..., y_N], N > T$ is firstly cut into

$$\begin{gathered}
[y_1, ..., y_l, y_{l+1}, ..., y_{l+k}] \\
[y_2, ..., y_{l+1}, y_{l+2}, ..., y_{l+k+1}] \\
[y_3, ..., y_{l+2}, y_{l+3}, ..., y_{l+k+2}] \\
\vdots \\
[y_{T-l-k}, ..., y_{T-k-1}, y_{T-k}, ..., y_T]
\end{gathered}$$

Finally, these sequences are split into LSTM input sequences (left) and LSTM target sequences (right):

$$\begin{array}{ll}
[y_1, ..., y_l] & [y_{l+1}, ..., y_{l+k}] \\
[y_2, ..., y_{l+1}] & [y_{l+2}, ..., y_{l+k+1}] \\
[y_3, ..., y_{l+2}] & [y_{l+3}, ..., y_{l+k+2}] \\
\multicolumn{2}{c}{\vdots} \\
[y_{T-l-k}, ..., y_{T-k-1}] & [y_{T-k}, ..., y_T]
\end{array}$$

The training data is now in a suitable shape for training an LSTM. The same procedure is applied to the holdout data in order to compute the models' forecast estimates.

### 3.2 Meta-Learning with Autocorrelation

After the individual LSTMs are trained, the key question is how to combine their individual forecast estimates. We use two approaches to combining:

1. Mean forecast: For each step in the forecasting horizon, take the mean of the base learners' forecasts for each future point.
2. Stacking: First, 70% of the holdout data $Y_{holdout}$ is used to generate the base learners' forecasts. In order to achieve this, the data is prepared as explained in Sect. 3.1. Since the true values of the forecasts are available, the forecasts (i.e., features of the meta-learners) are interpreted as the explanatory variables of the meta-learner and the true values are the target variable. We apply (1) Ridge Regression, (2) the Random Forest algorithm and (3) the xgboost algorithm as meta-learners, such that both linear relationships as well as non-linear ones can be modeled.
   **Ridge Regression** can be interpreted as linear least squares with L2 regularization of the coefficients. It is particularly effective if the number of predictors is high and if multicollinearity between the features is high. It is, however, a linear model, therefore suited for the case where the relationship between input features and target is linear.
   **Random Forest** constructs an ensemble of $m$ decision trees, where each tree is trained on a bootstrap sample of the original training data. In addition to this, different random subsets of features are considered at each tree split. The trees usually remain unpruned such that they have high variance. Ultimately, individual tree predictions are averaged to a final estimate. This way, random forests can model non-linear relationships.
   **Extreme Gradient Tree Boosting (xgboost)** combines trees in a boosting manner and currently provides state of the art performance amongst several prediction challenges.

Independent of the combiner, all approaches are evaluated on the exact same data, i.e., the latter 30% of $Y_{holdout}$, in order to ensure result comparability.

### 3.3 Constructing the Ensemble

Concisely, the combined forecasts estimates for a univariate, continuous series $Y = \{y_1, y_2, ..., y_N\}$ are generated as follows:

1. Split $Y$ into $Y_{train}$ (85%) and $Y_{holdout}$ (15%).
2. Standardize training and test data:
   $Y_{train} = \frac{Y_{train} - \bar{y}_{train}}{sd_{train}}$, $Y_{holdout} = \frac{Y_{holdout} - \bar{y}_{train}}{sd_{train}}$, where $\bar{y}_{train} = \frac{1}{T}\sum_{i=1}^{T} y_i$ and $sd_{train} = \sqrt{\frac{1}{T}\sum_{i=1}^{T}(y_i - \bar{y}_{train})^2}$. This step is essential when training neural networks due to gradient descent. $\bar{y}_{train}$ and $sd_{train}$ are used to standardize both $Y_{train}$ and $Y_{holdout}$ since $\bar{y}_{holdout}$ and $sd_{holdout}$ are unknown in a real-word scenario.

3. Split the standardized holdout data $Y_{holdout}$ into $Y_{metatrain}$ (first 70% of $Y_{holdout}$) and $Y_{test}$ (last 30% of $Y_{holdout}$) data. $Y_{metatrain}$ is used to generate the training data for the meta-learners, and $Y_{test}$ is unseen data that will be used for the final model evaluations.
4. Train $|S|\cdot|\Delta|$ LSTMs on the training data $Y_{train}$ with given ensemble parameters $S = \{seqlen_1, seqlen_2, ...\}$ and $\Delta = \{\delta_1, \delta_2, ...\}$ as elaborated in Sect. 3.1.
5. Compute the individual LSTM forecasts on all sequences of the $Y_{metatrain}$ holdout data.
6. Train the meta-learners (Ridge Regression and Random Forest), where the individual LSTM forecasts serve as input features. The target variable is given by the actual values of the sequence forecasts.
7. Determine the sequence forecasts on the $Y_{test}$ holdout data. Do this for the individual LSTMs as well as the stacking models. Further, calculate a mean forecast which, for each forecasted future point, takes the average of the LSTMs' individual forecasts for that point.
8. Transform all forecasts back to the original scale, i.e. $FC = FC \cdot sd_{train} + \bar{y}_{train}$ for each forecast vector $FC$.

Since the LSTMs in step 4 are independent of each other, they can be trained in parallel.

## 4 Experimental Analysis

We test the performance of the approach by applying it to four datasets of different size, shape and origin. The experimental analysis shows that the ensemble of LSTMs gives robust results across different data sets. Even more impressive is that stacking outperforms all other models, both base LSTM learners and baselines, in any considered case.

### 4.1 Setup

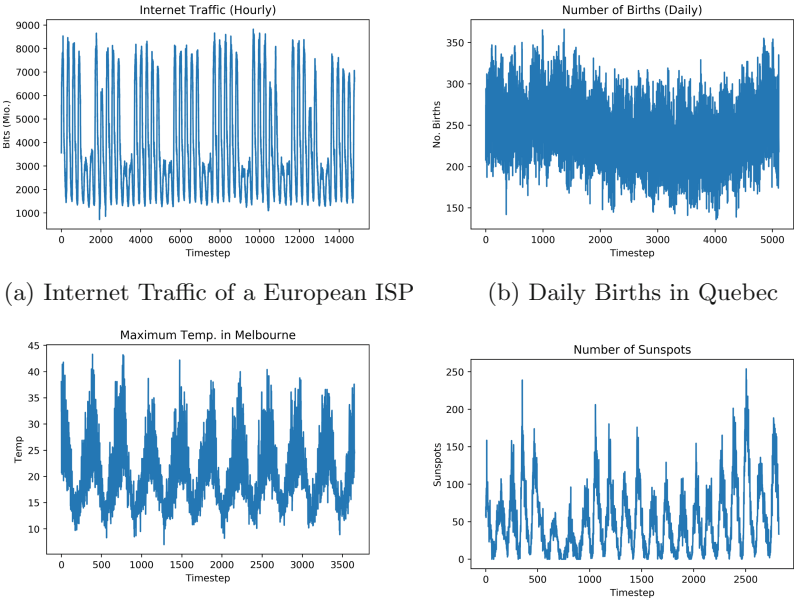
Figure 1 depicts the four datasets in their original shape, Table 1 describes their basic properties.

The algorithm specified in Sect. 3.3 is applied to each of the datasets. The following forecasting approaches are evaluated and compared:

- LSTM base models
- LSTM ensemble variants: mean forecast, stacking forecast via Ridge Regression (RR), Random Forest (RF) and xgboost (XGB)
- Simple moving average, predicting a constant value which is the mean of the input sequence
- Simple exponential smoothing. Here, the $i$-th forecasted value is given by

$$y_{t+i} = \alpha y_t + \alpha(1-\alpha)y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + ... + \alpha(1-\alpha)^{49} y_{t-49} \quad (1)$$

- ARIMA




(a) Internet Traffic of a European ISP

(b) Daily Births in Quebec

(c) Maximum Temperature in Melbourne

(d) Number of Sunspots

**Fig. 1.** Four time series in scope for the experimental analysis

– xgboost. Out-of-the-box xgboost is not capable of sequence forecasting. In order to account for this, we generate an additional variable which encodes forecasting step $1, 2, \ldots, 50$ of each example. Therewith, the feature matrix $X$ and target variable $y$ for the xgboost algorithm are

$$X = \begin{pmatrix} y_{t-49}(s_1) & y_{t-48}(s_1) & \ldots & y_t(s_1) & 1 \\ y_{t-49}(s_1) & y_{t-48}(s_1) & \ldots & y_t(s_1) & 2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{t-49}(s_1) & y_{t-48}(s_1) & \ldots & y_t(s_1) & 50 \\ y_{t-49}(s_2) & y_{t-48}(s_2) & \ldots & y_t(s_2) & 1 \\ y_{t-49}(s_2) & y_{t-48}(s_2) & \ldots & y_t(s_2) & 2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{t-49}(s_2) & y_{t-48}(s_2) & \ldots & y_t(s_2) & 50 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{t-49}(s_N) & y_{t-48}(s_N) & \ldots & y_t(s_N) & 1 \\ y_{t-49}(s_N) & y_{t-48}(s_N) & \ldots & y_t(s_N) & 2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{t-49}(s_N) & y_{t-48}(s_N) & \ldots & y_t(s_N) & 50 \end{pmatrix}, y = \begin{pmatrix} y_{t+1}(s_1) \\ y_{t+2}(s_1) \\ \vdots \\ y_{t+50}(s_1) \\ y_{t+1}(s_2) \\ y_{t+2}(s_2) \\ \vdots \\ y_{t+50}(s_2) \\ \vdots \\ y_{t+1}(s_N) \\ y_{t+2}(s_N) \\ \vdots \\ y_{t+50}(s_N) \end{pmatrix} \tag{2}$$

**Table 1.** Data description, number of examples $N$, mean $\mu$, and standard deviation $\sigma$

| Data | N | $\mu$ | $\sigma$ |
|---|---|---|---|
| Births in Quebec [26] | 5,113 | 250.8 | 41.9 |
| Internet Traffic (Mio. bits) [25] | 14,772 | 3,811 | 2,161 |
| Maximum Temperature in Melbourne[a] | 3,650 | 20.0 | 6.1 |
| Number of Sunspots[a] | 2,820 | 51.3 | 43.4 |

[a]http://datamarket.com, accessed July 7 2017.

where $y_i(s_j)$ is the $i$-th value of input sequence $s_j$ for $i \leq t$. In case of $i > t$, $y_i$ should be interpreted as the $i$-th actual value of the respective sequence forecast.

For training and forecasting with LSTMs, the keras implementation [27] is used. xgboost is applied for extreme gradient boosting. A functional implementation of the entire experimental setup is available on GitHub[1].

### 4.2 Results

All trained models are evaluated on the same test set and performance is measured in terms of RMSE. The chosen forecasting horizon is 50, i.e., the models are trained and tested to estimate the next 50 values of given input sequences. The average performance across all test sequences in the respective test set is shown in Tables 2 and 3. The first column indicates the diversity-generating parameter. For our experiments, we evaluate dropout values $\{0.1, 0.2, 0.3, 0.4, 0.5\}$, a number of hidden layers in $\{2, 3, 4, 5\}$, the number of nodes in the input and hidden layers varies between the length of the input sequence, half of the length, and quarter of the length. Learning rate is set to values $\{0.01, 0.001, 0.0001, 0.00001\}$.

As default values, we choose RMSProp [23] as optimizer, the learning rate is set to 0.001, the loss function is the mean squared error (MSE), batch size is 32 and training is performed over 15 epochs per LSTM. One LSTM input layer and two LSTM hidden layers are used, whose number of nodes is equal to the current sequence input length. Further, a dropout [22] of 0.3 is added to the LSTM layers in order to prevent model overfitting.

The second column represents the metric under consideration. We compare the model performance is terms of RMSE. Results are transformed back to their original scale prior to computing the RMSE for better interpretability. The cases where an ensemble beats all other tested models are marked in bold and the best performing combiner algorithm is stated in parentheses (RF: Random Forest, RR: Ridge Regression, XGB: xgboost). Additionally, we provide the average pairwise Pearson correlation $\rho$ between the forecasts of the base LSTMs. The more the model forecasts differ from one another, the higher the potential improvement gained by an ensemble. The key observations are:

[1] https://github.com/saschakrs/TSensemble, accessed July 7 2017.

**Table 2.** Result summary for "Births" and "Traffic" datasets

| Varied parameter | Performance measure | Dataset | |
|---|---|---|---|
| | | Births | Traffic |
| - | Simple mean RMSE | 42.19 | 1380.53 |
| - | Exp. smoothing RMSE | 49.36 | 1389.90 |
| - | ARIMA RMSE | 38.13 | 1224.83 |
| - | xgboost RMSE | 40.55 | 1033.65 |
| Dropout | $\rho$ between base LSTM forecasts | 0.94 | 0.91 |
| Dropout | Avg. base LSTM RMSE | 27.76 | 991.41 |
| Dropout | Best base LSTM RMSE | 25.23 | 826.42 |
| Dropout | Best ensemble RMSE | 25.45 (XGB) | **652.42 (RF)** |
| #Hidden L | $\rho$ between base LSTM forecasts | 0.95 | 0.91 |
| #Hidden L | Avg. base LSTM RMSE | 27.77 | 726.20 |
| #Hidden L | Best base LSTM RMSE | 25.31 | 811.35 |
| #Hidden L | Best ensemble RMSE | 25.62 (XGB) | **656.97 (RF)** |
| #Nodes | $\rho$ between base LSTM forecasts | 0.95 | 0.91 |
| #Nodes | Avg. base LSTM RMSE | 28.55 | 944.92 |
| #Nodes | Best base LSTM RMSE | 25.65 | 826.42 |
| #Nodes | Best ensemble RMSE | **25.57 (XGB)** | **630.85 (RF)** |
| Learning rate | $\rho$ between base LSTM forecasts | 0.56 | 0.80 |
| Learning rate | Avg. base LSTM RMSE | 42.85 | 1578.64 |
| Learning rate | Best base LSTM RMSE | 25.65 | 826.41 |
| Learning rate | Best ensemble RMSE | **25.37 (RR)** | **667.26 (RF)** |

– In 81% of all cases, an LSTM stacking model outperforms all other approaches. In the other cases, there is only one LSTM model (respectively) that slightly outperforms the stacked LSTMs.
– Although the ensemble architecture is identical for all data sets, there is no single best meta-learner for all data sets.
– Model diversity is essential: $\rho$ is correlated to the best ensemble RMSE by more than 70%, i.e., a low $\rho$ between forecasts tends to increase ensemble performance. This becomes visible especially in the context of the Sunspots data, where the stacked LSTMs outperform their base learners by more than 50% RMSE. Hence, combining many comparably weak LSTM predictors results in a greater performance win than the combination of a few good learners.

**Table 3.** Result summary for "Melbourne" and "Sunspots" datasets

| Varied parameter | Performance measure | Dataset | |
|---|---|---|---|
| | | Melbourne | Sunspots |
| - | Simple mean RMSE | 7.44 | 74.88 |
| - | Exp. smoothing RMSE | 7.47 | 47.88 |
| - | ARIMA RMSE | 7.41 | 54.50 |
| - | xgboost RMSE | 5.90 | 47.09 |
| Dropout | $\rho$ between base LSTM forecasts | 0.76 | 0.40 |
| Dropout | Avg. base LSTM RMSE | 6.94 | 79.39 |
| Dropout | Best base LSTM RMSE | 6.51 | 70.82 |
| Dropout | Best ensemble RMSE | **6.10 (RR)** | **33.74 (RR)** |
| #Hidden L | $\rho$ between base LSTM forecasts | 0.69 | 0.39 |
| #Hidden L | Avg. base LSTM RMSE | 6.90 | 79.90 |
| #Hidden L | Best base LSTM RMSE | 6.70 | 67.53 |
| #Hidden L | Best ensemble RMSE | **6.11 (RR)** | **31.69 (XGB)** |
| #Nodes | $\rho$ between base LSTM forecasts | 0.75 | 0.51 |
| #Nodes | Avg. base LSTM RMSE | 7.01 | 81.58 |
| #Nodes | Best base LSTM RMSE | 6.69 | 67.53 |
| #Nodes | Best ensemble RMSE | **6.13 (RR)** | **32.91 (RR)** |
| Learning rate | $\rho$ between base LSTM forecasts | 0.59 | 0.59 |
| Learning rate | Avg. base LSTM RMSE | 7.36 | 85.42 |
| Learning rate | Best base LSTM RMSE | 5.91 | 67.53 |
| Learning rate | Best ensemble RMSE | 5.97 (RR) | **31.35 (XGB)** |

– For all ensembles, it holds that its forecasts are significantly different from all baseline estimates. This result is based on the paired t-test for significance.[2]
– Out of the four investigated LSTM parameters, varying the learning rate leads to greatest diversity generation. The reason for this is that the learning rate has a strong effect on the local minimum that is reached. Varying the values for dropout, hidden layers and nodes tends to generate forecasts with higher correlation and less diversity.

## 5 Future Work and Conclusion

The experiments suggest that the LSTM ensemble forecast is indeed a robust estimator for multi-step ahead time series forecasts. Although there exist single models that perform better in terms of RMSE, the proposed ensemble approach

[2] Note that even if for smaller datasets, like the Sunspot dataset, the test set is fairly small, this shows that the results are still significant.

enables users to achieve solid forecasts without the need to focus on heavy parameter optimization. An interesting observation is that the outstanding performance of the ensemble forecast is valid across multiple datasets from entirely different domains. There remains, however, significant potential to further improve some aspects of the algorithm, especially with regard to the fundamental design of the ensemble.

The proposed LSTM ensemble architecture opens the door to lots of further potential. First and foremost, the meta-learner of the stacking model could be improved in two ways. One is to generate more features describing the dynamics of the series, especially that part immediately preceding the forecasting horizon. Additionally, the meta-learners' parameters could be tuned more heavily, or it could be replaced by an entirely different meta-learning algorithm.

Another area of improvement lies in the design of the ensemble itself. The selection of values for sequence lengths $S$ and LSTM parameters $\Delta$ could further influence the final result, especially if some domain specific knowledge regarding the series is available.

Lastly, configuring the individual LSTMs may increase the general quality of the base learners. This can be achieved by tuning the LSTM parameters. It must be ensured, however, that the diversity between these models remains sufficiently large.

## References

1. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
2. Tsukamoto, K., Mitsuishi, Y., and Sassano, M.: Learning with multiple stacking for named entity recognition. In: Proceedings of the 6th Conference on Natural Language Learning, vol. 20, pp. 1–4. Association for Computational Linguistics (2002)
3. Lai, K.K., Yu, L., Wang, S., Wei, H.: A novel nonlinear neural network ensemble model for financial time series forecasting. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3991, pp. 790–793. Springer, Heidelberg (2006). https://doi.org/10.1007/11758501_106
4. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989). Elsevier, Amsterdam
5. Zhang, G.P.: Time series forecasting using a hybrid ARIMA and neural network model. Neurocomputing **50**, 159–175 (2003). Elsevier, Amsterdam
6. Adhikari, R., Agrawal, R.K.: A linear hybrid methodology for improving accuracy of time series forecasting. Neural Comput. Appl. **25**(2), 269–281 (2014). Springer, London, UK
7. Adhikari, R.: A neural network based linear ensemble framework for time series forecasting. Neurocomputing **157**, 231–242 (2015). Elsevier, Amsterdam
8. Armstrong, J.S.: Combining forecasts. In: Armstrong, J.S. (ed.) Principles of Forecasting. ISOR, pp. 417–439. Springer, Boston (2001). https://doi.org/10.1007/978-0-306-47630-3_19
9. Babu, C.N., Reddy, B.E.: A moving-average filter based hybrid ARIMA-ANN model for forecasting time series data. Appl. Soft Comput. **23**, 27–38 (2014). Elsevier, Amsterdam

10. Wang, L., Zou, H., Su, J., Li, L., Chaudhry, S.: An ARIMA-ANN hybrid model for time series forecasting. Syst. Res. Behav. Sci. **30**(3), 244–259 (2013)
11. Aladag, C.H., Egrioglu, E., Kadilar, C.: Forecasting nonlinear time series with a hybrid methodology. Appl. Math. Lett. **22**(9), 1467–1470 (2009)
12. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
13. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994)
14. Malhotra, P., Vig, L., Shroff, G., Agarwal, P.: Long short term memory networks for anomaly detection in time series. In: Proceedings of the 23rd European Symposium on Artificial Neural Networks. Computational Intelligence and Machine Learning, pp. 89–94. Presses universitaires de Louvain (2015)
15. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: Proceedings of the 30th International Conference on Machine Learning, ICML 2013, vol. 28, pp. 1310–1318 (2013)
16. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
17. Assaad, M., Boné, R., Cardot, H.: A new boosting algorithm for improved time-series forecasting with recurrent neural networks. Inf. Fusion **9**(1), 41–55 (2008)
18. Durbin, J., Koopman, S.J.: Time Series Analysis by State Space Methods, vol. 38. Oxford University Press, Oxford (2012)
19. Hamilton, J.D.: Time Series Analysis, vol. 2. Princeton University Press, Princeton (1994)
20. Shumway, R.H., Stoffer, D.S.: Time Series Analysis and Its Applications: with R Examples. Springer Science & Business Media, Heidelberg (2010)
21. Brockwell, P.J., Davis, R.A.: Introduction to Time Series and Forecasting, 2nd edn. Springer, New York (2010)
22. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
23. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural Netw. Mach. Learn. **4**(2), 26–31 (2012)
24. Lichman, M.: UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine, CA (2013). http://archive.ics.uci.edu/ml
25. Cortez, P., Rio, M., Rocha, M., Sousa, P.: Multi-scale Internet traffic forecasting using neural networks and time series methods. Expert Syst. **29**(2), 143–155 (2012)
26. Hipel, K.W., McLeod, A.I.: Time Series Modelling of Water Resources and Environmental Systems, vol. 45. Elsevier, Amsterdam (1994)
27. Chollet, F.: Keras (2015). https://github.com/fchollet/keras