Timo Meiendresch

Recurrent Neural Networks for Time Series Forecasting

Capstone Project

Machine Learning Engineer Nanodegree (Udacity)

Cologne 2019

Contents

Abstract

The year is 2019 A.D. Quantitative research is entirely occupied by Machine Learning (ML). Well, not entirely... One small group of indomitable forecasters still holds out against the invaders. And life is not easy for the ML evangelists who garrison the fortified camps of quantitative research...

1 Introduction

Time series forecasting is an essential tool in business, economics, and many other disciplines. The ability to accurately predict future values given historic data is of huge importance to decision makers.

In recent years, **Machine Learning (ML)** methods had huge success in many areas of quantitative research. Prominent examples include classification tasks, image processing, or text analysis. ML took many fields by storm and it seems that many quantitative disciplines can benefit from giving in to the AI wave.

And yet, ML methods are rarely considered in time series forecasting. Research indicated that the performance of ML methods were not able to compete to traditional methods for a long time. Moreover, traditional statistical methods are often easier to implement, computationally cheap, and interpretable. Noteworthy examples include the **Autoregressive Integrated Moving-Average (ARIMA)** and the various **Exponential Smoothing (ES)** methods. Until recently, a widely shared notion was that complex methods for time series forecasting were universally not performing better than traditional ones (e.g. Hyndman, 2019). Among others, Makridakis et al. (2018) noted just recently that there is only very limited scientific evidence which suggests that artificial neural networks may be a useful tool for time series forecasting.

Paradigm Shift

But, advances in recent years start to challenge this perception. This gradual shift can be traced by a series of forecasting competitions.

One notable example is the "Web Traffic Time Series Forecasting" competition hosted by Kaggle (2017). The task was to forecast web traffic for 145,000 wikipedia articles over time. The winning solution by Arturus (2018) was based on a Long Short-term Memory (LSTM) recurrent neural network which originated in text analysis (Graves, 2013; Sutskever et al., 2014). This architecture is often referred to as sequence-to-sequence (seq2seq) model. Along with this seq2seq network architecture, the author incorporated time-specific features to capture the relative position of the observation in time. For example, features were used to indicate the day of the month, or month of the year to capture year-to-year or quarter-to-quarter seasonality. The approach of the model has strong resemblance to the more general implementation of the DeepAR algorithm which is covered in later chapters.

To even greater attention came the winner of the most recent version of the M4 competition (Makridakis et al., 2019). The M4 competition was the fourth round of a time series forecasting competition with the objective to identify the most accurate forecasting methods. These methods had to produce forecasts for a highly diverse dataset of 100,000 univariate time series from six domains and across six frequencies. Most accurate among 61 contributions was a method by Smyl (2019) which is a highly complex hybrid of **recurrent neural network (RNN)** and ES methods.

There is a fundamental difference between the competitions considering the data that were used. Whereas the web traffic series from the Kaggle competition represent a relatively homogeneous dataset, the M4 data are highly diverse and heterogeneous. The former has only one frequency (daily) and series from a very specific area (web traffic), the M4 data come from six rather crude domains (Micro, Macro, Industry, Finance, Demographic, Other) and six different frequencies (Yearly, Quarterly, Monthly, Weekly, Daily, Hourly). The results of the M4 competition showed the potential of recurrent networks even for such a highly diverse dataset, leading to a new surge of interest in the field.

Local vs. global methods

Another important aspect, emphasized by the M4 competition, is the transition from local to global methods in time series forecasting. Whereas traditional methods as ARIMA and ES are applied to individual series, ML-based methods are trained using the entirety of available series. These **local** methods estimate a number of parameters within a model space that is restricted by the respective structure of the model. Because these approaches rely on an explicit model structure with sparse parameters, they are also referred to as **model-based** (Wang et al., 2019). For this, each time series is modeled and trained independently of other series in the data set. Hence, the parameters of the individual model are independent of i) the total number of series in the dataset as well as ii) how similar other series in the dataset are. Both aspects may be important factors in improving accuracy by exploiting general patterns and dependencies.

Closely related is the emergence and availability of large sets of data, which poses a new kind of forecasting problem, namely to predict huge sets of time series (e.g. Salinas et al., 2017). Examples include product sales in online retailers, household energy consumption, or web traffic. In contrast to local models, **global** methods enable the use of **cross-series learning**, i.e. general patterns and dependencies can be picked up during the training process for which all available time series are used. This should lead to a general representation of the entire data set. In theory, RNN-based methods should be able to learn across individual series and extract general patterns of the data as well as learn dependency relationships between the individual series. The results of the forecasting competitions imply that this can actually improve forecast accuracy compared to local approaches.

There are various methods based on the idea of cross-series learning, in particular utilizing recurrent networks. The determinants of their performance is a relatively new question into which very little empirical research has so far been done. In theory, RNN should be able to work as a universal function approximator if it is only powerful enough. In practice however, the learnability of RNNs is limited. A typical problem is that they do not converge during training and loss doesn't decrease. This is why these networks are considered to be difficult to train. This is an important factor for practitioners as training a powerful network on such a huge amount of data is computationally expensive. Hence, in context of this project and RNN-based models, there are two main characteristics that should be considered in characterizing the underlying data basis, namely

Property 1. *The total number of individual time series in the underlying data set. How many time series are in the dataset as a whole?*

Property 2. *Similarity of the individual time series in the data set. How different are the time series in the dataset?*

Thus far, the models under consideration often neglect property 2 as they focus on rather homogeneous data even though the M4 competition showed that this kind of models could efficiently tackle highly diverse data. Moreover, these methods lack rigorous comparison with previous methods and known data sets for which benchmarks exist. This capstone project investigates the performance of three RNN-based methods on the diverse M4 data for which a wide variety of benchmarks exist. These methods are:

- **Deep Autoregressive Recurrent Neural Networks (DeepAR)** - Salinas et al. (2017)

- **Deep State Space Models (DeepState)** - Rangapuram et al. (2018)

- **Deep Factor Recurrent Neural Network Model (DF-RNN)** - Wang et al. (2019)

To evaluate these models I will apply them to the hourly, daily, weekly and monthly data of the M4 competition using the same measures of accuracy. Hence, the performance of the three algorithms can be directly compared to a variety of benchmarks that are available.

Note that these methods are designed and tested for large sets of related. In contrast, the M4 data consist of highly diverse time series from different domains and different frequencies. To the best of my knowledge, there is no thorough empirical investigation that applies the aforementioned algorithms to the M4 subsets. The primary contribution of this paper is to evaluate these algorithms on the well studied M4 data and compare the results to known benchmarks.

The paper proceeds as follows. In Section 2, I provide an overview of the project and introduce the performance measures. Section 3 presents the dataset and algorithms, whereas section 4 describes the implementation. And lastly, section 5 discusses the results.

2 Definition

2.1 Project Overview

In this paper, I compare three RNN-based time series forecasting algorithms with the methods of the well studied M4 competition. For this, I use the same data that were used during the competition and employ the same accuracy measures. The methods of the competition include state-of-the-art approaches as well as traditional methods.

2.2 Problem Statement

Main goal of this paper is to test the performance of the three algorithms applied to the M4 data. The algorithms are DeepAR, DeepState, and DF-RNN. Results will be compared to traditional time series methods and modern methods that were part of the competition. Within this setup, I can draw several conclusions about the performance of the three approaches applied to a heterogeneous dataset. While a number of studies focus on the applicability on homogeneous dataset, results of the M4 competition show that the underlying data need not necessarily come from a similar data generating process (DGP) in order to produce very accurate forecasts.

2.3 Metrics

I evaluate the performance of the methods using two measures that were also part of the M4 competition. Forecast measures are still a highly active area of research and there is no consensus on which one is considered to be the best one. Hence, two widely used **scale-independent errors** were used in the competition. Scale-independent means they are independent of the unit of the time series. Because the competition evaluated accuracy across highly diverse time series with different units, scale-independent measures are required. The two measures are the

- **symmetric mean absolute percentage error (sMAPE)**
- and the **mean absolute scaled error (MASE)**

The **sMAPE** belongs to the class of percentage errors which are unit-free. One drawback of the measure is that it is infinite (or undefined) if the target variable is zero for any t in the period of interest, and is highly skewed towards extreme values if the target variable is close to zero. The measure is defined as

$$sMAPE = \frac{1}{h} \sum_{t=1}^{h} \frac{2\,|Y_t - \hat{Y}_t|}{|Y_t| + |\hat{Y}_t|} \tag{1}$$

where Y_t is the post-sample value, $\hat{Y}_t$ the estimated forecast, and h the forecasting horizon.

Alternatively, Hyndman and Koehler (2006) propose the scaled error measure **MASE** when comparing forecast accuracy across series with different units. In this case, the absolute error is scaled using the **Mean Absolute Error (MAE)** from a simple forecast method (either naive or seasonal naive method). The method for seasonal data is defined as:

$$MASE = \frac{1}{h} \frac{\sum_{t=1}^{h} |Y_t - \hat{Y}_t|}{\frac{1}{n-m} \sum_{t=m+1}^{n} |Y_t - Y_{t-m}|}, \tag{2}$$

where m indicates the seasonality of the series. In case of non-seasonal data ($m = 1$), $Y_t - Y_{t-m}$ simplifies to $Y_t - Y_{t-1}$ being the naive forecast error. The naive forecast predicts the next value to be equal to the last value $\hat{Y}_{t+1} = Y_t$. This yields the non-seasonal version of the MASE as

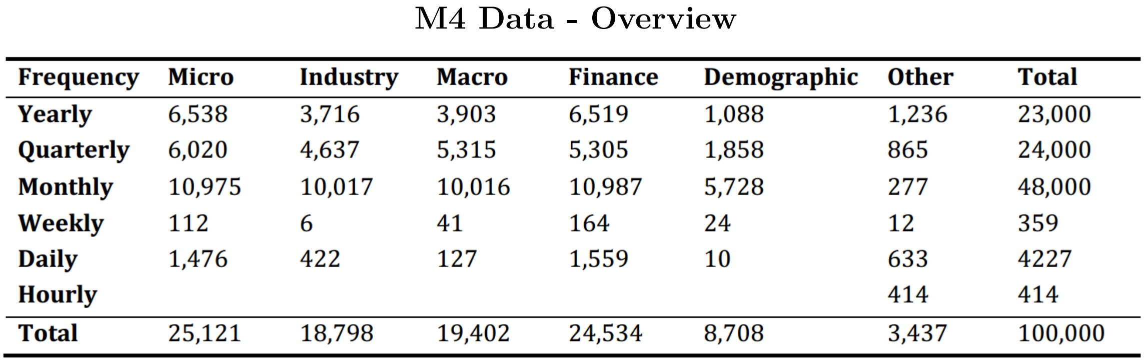
$$MASE = \frac{1}{h} \frac{\sum_{t=1}^{h} |Y_t - \hat{Y}_t|}{\frac{1}{n-m} \sum_{t=m+1}^{n} |Y_t - Y_{t-m}|} \tag{3}$$

In addition to measuring point forecast (PF) accuracy using MASE and sMAPE, prediction interval (PI) accuracy was included in the competition for the first time. For this, the **Mean Scaled Interval Score (MSIS)** has been used as accuracy measure. Note that there are other widely used measures for PF and PI accuracy, such as the normalized sum of quantile losses for quantiles $p \in \{0.5, 0.9\}$. These measures were used for example in the empirical studies of Rangapuram et al. (2018) and Wang et al. (2019). For reference, MSIS and quantile losses are therefore also included in this paper.

3 Analysis

3.1 Data Exploration

The M4 dataset consists of 100,000 univariate time series from six domains (Microeconomics, Industry, Macroeconomics, Finance, Demographic, Other) and six different frequencies (Yearly, Quarterly, Monthly, Weekly, Daily). A time series is defined by successive observations in time. Time intervals between observations are equal in this case (equidistant) which means that the time between two observations remains constant for each time series. As the data are from many different fields, units of the observations are different across series and unknown in this dataset. The data are a selected subset from a database of more than 900,000 time series and they are of different length. They exhibit common characteristics of time series, such as seasonal patterns, cyclic behavior, as well as trends that the respective model has to capture. Given the behavior of the series, the method has to extrapolate the series into the future. The number of future observations (forecast horizon) to be forecasted is given by A key challenge for the prediction method is the high diversity of the data. Some data exhibit cyclical behavior, up- or downward trends, or varying seasonal patterns. Accurately capturing this behavior to forecast future values is the goal of time series

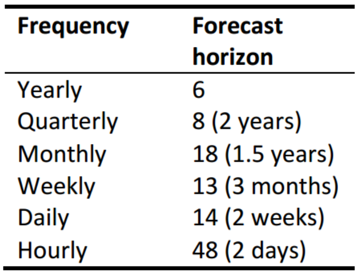


M4 Data - Overview

Frequency	Micro	Industry	Macro	Finance	Demographic	Other	Total
Yearly	6,538	3,716	3,903	6,519	1,088	1,236	23,000
Quarterly	6,020	4,637	5,315	5,305	1,858	865	24,000
Monthly	10,975	10,017	10,016	10,987	5,728	277	48,000
Weekly	112	6	41	164	24	12	359
Daily	1,476	422	127	1,559	10	633	4227
Hourly						414	414
Total	25,121	18,798	19,402	24,534	8,708	3,437	100,000

Table 1: M4 data by domain and frequency.

M4 Data - Forecast Horizons



Frequency	Forecast horizon
Yearly	6
Quarterly	8 (2 years)
Monthly	18 (1.5 years)
Weekly	13 (3 months)
Daily	14 (2 weeks)
Hourly	48 (2 days)

Table 2: Forecast horizons by frequency of the data.

forecasting.

3.2 Exploratory Visualization

As aforementioned, the data are highly diverse. They differ with respect to domain, frequency, and length. For each series there exists a training set as well as an extended test set. The training set consists of a varying number of successive observations in time. The test set includes the same observations but has the additional observations for the forecast horizon. A randomly chosen example series from the hourly M4 data is depicted below:

M4 Hourly - Series #338

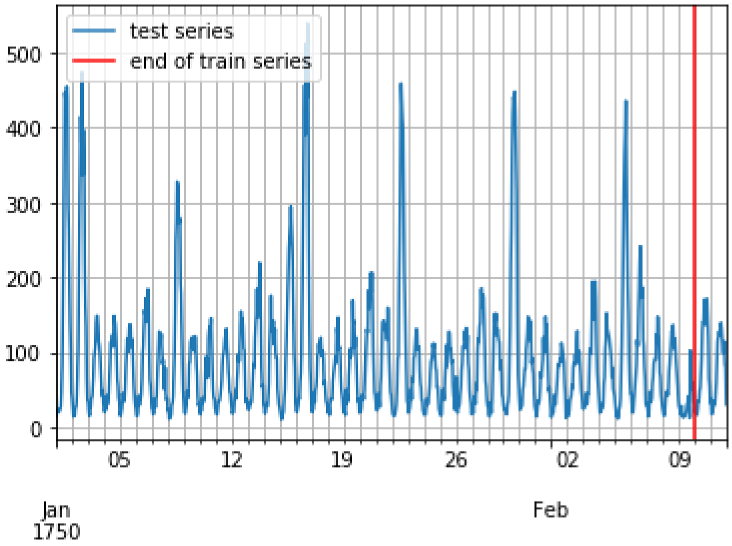


Figure 1: Hourly time series 338. The vertical red line indicates where the training data ends and the test set begins.

3.3 Algorithms and Techniques

The first of the three algorithms is the **Deep Autoregressive Recurrent Neural Network (DeepAR)** method, described in Salinas et al. (2017). It is the only built-in time series prediction algorithm in Amazon SageMaker for ML on cloud computing instances. DeepAR uses autoregressive features that are used as the inputs to a recurrent neural network. The RNN uses LSTM cells as default but can also be changed to GRU cells. Previous observations are utilized as inputs which makes this algorithm autoregressive. Examples of this features are day-of-month or week-of-year for weekly data. Which features are included depends on the frequency.

Rangapuram et al. (2018) introduced **Deep State Space Models (DeepState)**. DeepState combines state space models (SSM) with a recurrent neural network architecture that calculates the optimal parameters for the SSM. The SSM is applied locally to the individual time series. The parameters are determined by a jointly trained RNN using all available data. Hence, this algorithm combines a local method with a global one.

Furthermore, I included the **Deep Factor - Recurrent Neural Network (DF-RNN)** by Wang et al. (2019). This hybrid algorithm uses a local method specific to each time series in the dataset as well as a global RNN, which is trained jointly with the other time series. For now, the only available local method is using another RNN. Hence, the model combines a locally trained RNN with a globally trained RNN. In brief, key characteristics of the three algorithms :

- **DeepAR** - "Pure" global RNN that uses autoregressive features as inputs.
- **DeepState** - Globally trained RNN that determines the best parameters of the local SSM.
- **DF-RNN** - Hybrid of a local method that is applied to individual time series and a global RNN that is trained jointly.

The training process of the three algorithms uses a fixed window approach. During training, the algorithms extract a sequential number of observations (windows). Each window is a sliced version of the entire time series starting at different points of the series and is of length

$$l_{window} = l_{training} + l_{prediction},$$

where $l_{training}$ how many successive training observations are used (training range) and $l_{prediction}$ is the number of successive observations used for the evaluation (prediction range). The prediction range is has the length of the forecast horizon. The default is to use $l_{training} = l_{prediction}$. One window is then fed to the algorithms as one batch. In each epoch a number of batches are fed to the algorithm, where the number of batches per epoch can be varied as well as the number of total training epochs. Stochastic Gradient Descent (SGD) is used to adjust the parameters of the network during the training process. Once the training process is completed, the algorithm uses the parameters of the epoch with the smallest loss as final model. To avoid overfitting, dropout is used by default. Because random numbers are used in this setup, I have fixed them to make the results reproducible.

3.4 Benchmarks

In addition to providing a rich dataset across frequencies and domains, a key contribution of the M4 competition is the provision of a rich set of benchmark methods. Not only the metrics of the competition winners but also results for common time series methods, in particular ARIMA and ES, are provided in Makridakis et al. (2019) and its additional material. The following discussion will focus on the relative performance of the RNN-based methods with respect to:

1. Seasonally adjusted naive predictions (Naive2). This method uses a classical decomposition to deseasonalize the data, then predicts the next value to be equal to the last value and adds the seasonality to this forecast. In general, naive predictions for time series work surprisingly well.

2. Traditional methods (ARIMA and ES) and the combination benchmark (Comb)

3. Overall winning method of Smyl (2019), runner-up by Montero-Manso et al. (2019) as well as the winning method of the frequency under consideration.

4 Methodology

4.1 Data Preprocesing

The algorithms require the time series inputs in form of a **JSON Lines**-like format. In Python, this is essentially a dictionary of dictionaries, also referred to as a nested dictionary. Inputs must contain at least the following keys with their respective content:

- ‘`start`‘ - Timestamp with the format YYYY-MM-DD HH:MM:SS and the respective frequency of the data

- ‘target‘ - An array of floats or integers that represents the target series of interest.

An example input file takes the form

```
# get first entry in dataset.train
entry = list(dataset.train)[0]

# first entry
entry
>>> {'start': Timestamp('1750-01-01␣00:00:00', freq='H'),
  'target': array([605., 586., 586., ..., 521.], dtype=float32)
```

Additional keys are optional but often helpful in improving forecast accuracy. Here, the M4 competition does not provide additional covariates to the data. Because there were no dates supplied in the original data, an arbitrary start date is used (i.e. "1750-01-01 00:00:00").

4.2 Implementation

To obtain the results of the paper, I use the new **gluonts** API that is built on the deep learning framework **MXNet**. I proceed in a similar way for all three algorithms. First, I implement a function that takes in the following arguments:

- data - name of the built-in data in gluonts
- seed - a seed that fixes the random numbers. Thus, results can be reproduced using the seed from my results table.
- epochs - Number of epochs a model is trained
- batches - Number of batches epoch that are used per epoch.

```
def deepar(data="m4_weekly", seed=42, epochs=100, batches=50):

    dataset = get_dataset(data, regenerate=False)
   mx.random.seed(seed)
   np.random.seed(seed)
    ...

    return(output)
```

Applying this function returns a series of accuracy measures. Printing the results leads to

```
res = deepar(data="m4_weekly", seed=42, epochs=1, batches=50)
pprint(res)

>>> {'MASE': 5.54378962,
  'MSIS': 54.72282354,
  'epochs': 1,
  'sMAPE': 0.12889508,
  'seed': 42,
  'wQuantileLoss[0.5]': 0.09491686,
  'wQuantileLoss[0.9]': 0.07740071}
```

One challenge is the instability of the algorithms. Applying the same algrithm twice, keeping all inputs the same (i.e. epochs and batches), leads to different results. This is due to variability in the random numbers used for randomly selecting the series and respective windows. I use seeds to fix the random numbers and make the results reproducible. Note however, that the same seed leads to different random numbers whether a CPU or GPU is used. This issue is reported on github and will probably be fixed in future releases.

To account for the variation between trials I apply the same algorithm using different seeds and keep all else fixed. For this, I use a simple loop that iterates over a range of seeds and applies the respective function repeatedly. My results show the median of three trials.

```
# Loop over a range of seeds
results = []

data="m4_monthly"
epochs=100
batches=50

if __name__ == "__main__":
    for i in range(42, 45):
        print("Seed:", i)
        res = deepar(data, seed=i, epochs, batches)
        pprint(res)
        results.append(res)
```

This experimental design makes the results reproducible by using seeds and meaningful by using several trials.

4.3 Refinement

One key issue is the diverse nature of the data. There are many adjustable hyperparameters and it seems unlikely that the default values are optimal for all of the four subsets that I use.

The number of series within the four subsets range from 359 (M4 weekly) to 48,000 series (M4 monthly). Moreover, the data stems from many different domains and areas. There is no apparent relation between the individual time series. Assuming aforementioned cross-learning by exploiting common patterns and dependencies seems to be difficult in this setting and with this kind of data.

Then again, these algorithms have many adjustable hyperparameters. Some of the seemingly important ones are:

- context length,
- prediction length of the forecasting horizon,
- epochs,
- number of batches per epoch,
- number of RNN layers,
- number of RNN cells for each layer,
- or dropout rate

to mention a few of them. Within the setup of the competition, prediction length is fixed but there are many other moving parts that may influence the prediction accuracy.

A common solution to this problem is to use hyperparameter optimization (HPO) techniques, such as grid search or random search. These HPO techniques essentially run variations of the same algorithm with different hyperparameters to determine the best ones. This is not a feasible option here as some of the models run for as many as thirteen hours per trial. Running different variations multiplies the already high computational costs. To keep computational costs to a bearable level, I start with the default values and adjust the number of epochs if necessary.

5 Results

This section describes the results per frequency, starting from the lowest (hourly) to the highest one (monthly). For every combination of algorithm, data, epochs, and batches, the median result of three trials is included in the table. Trials start with seed 42 because it is "the ultimate answer to life, the universe, and everything..." (Adams, 2005). Note that DeepState models are computationally very expensive. Hence, I limited experimentation with DeepState to an acceptable minimum as some models took more than 13 hours of CPU time per trial. Because this algorithm is more stable than the others in terms of results it is unlikely that this alters the conclusions.

5.1 Model Evaluation and Validation

M4 Hourly

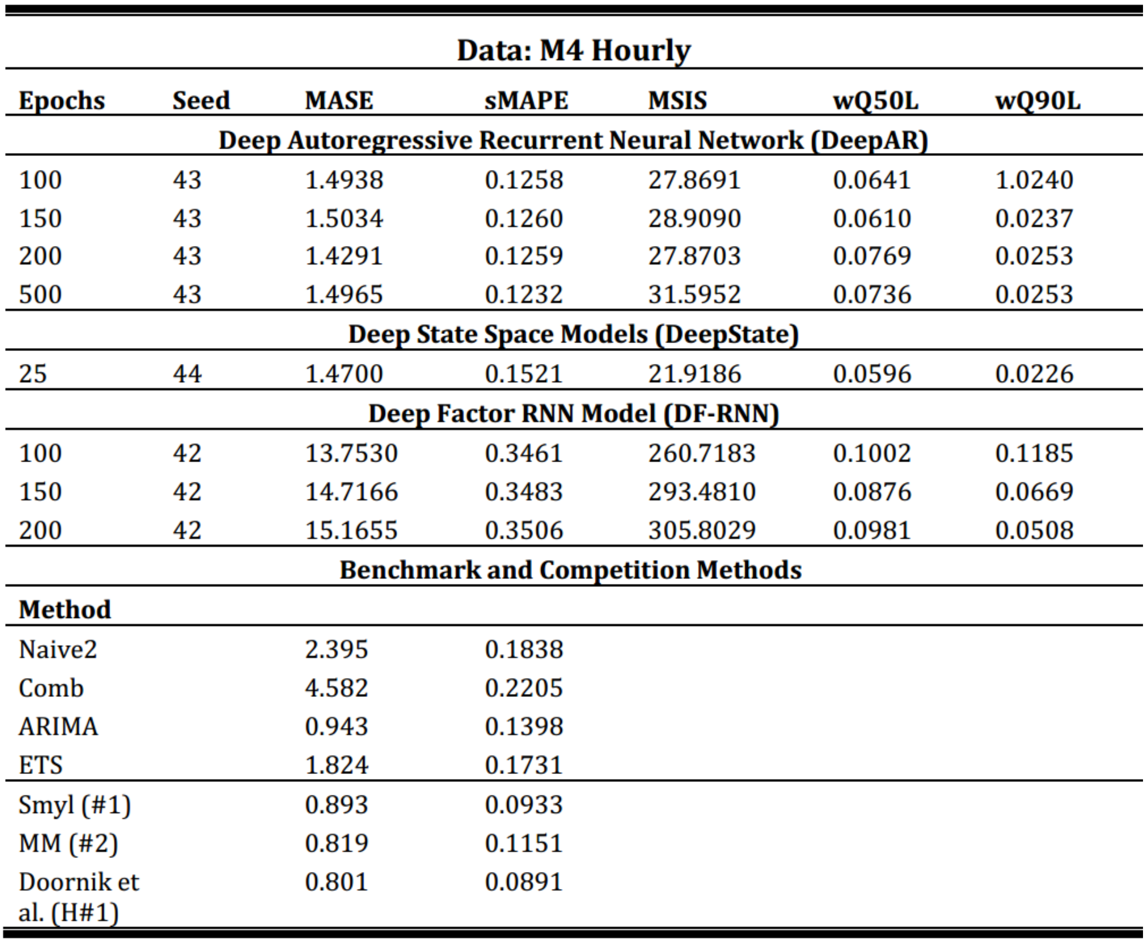
The hourly subset of the M4 data contains 414 time series. Given forecast horizon is 48, which corresponds to two days. Table 3 illustrates the performance of aforementioned algorithms, respective benchmarks, and winning methods applied to the hourly data of the M4 data.

It seems that DF-RNN is not able to capture the hourly subset at all. Comparing either MASE or sMAPE to the seasonally adjusted naive forecast (Naive2) shows how bad it works as it is way worse than the seasonally adjusted naive forecast. In addition, its performance is even worse than the standard naive forecast without taking seasonality into account. The naive forecast simply uses the previous value as future forecast (value not shown in the table).

DeepAR and DeepState show a significantly better performance. They deliver better forecasts than Naive2, Comb, and ETS. However, the traditional ARIMA method, that is based on modeling the autoregressive behavior of time series, outperforms both models with ease. Additionally, neither of the considered three algorithms comes close to the overall winning method by Smyl that also utilizes RNN. In addition, Montero-Manso (2019) and the best hourly method of Doornik and colleagues capture hourly data more precisely than the winning method.

The performance of DeepAR and DeepState is very similar in this setting but they cannot compete with the traditional ARIMA model nor current state-of-the-art methods.

M4 Hourly - Results



Data: M4 Hourly						
Epochs	**Seed**	**MASE**	**sMAPE**	**MSIS**	**wQ50L**	**wQ90L**
Deep Autoregressive Recurrent Neural Network (DeepAR)						
100	43	1.4938	0.1258	27.8691	0.0641	1.0240
150	43	1.5034	0.1260	28.9090	0.0610	0.0237
200	43	1.4291	0.1259	27.8703	0.0769	0.0253
500	43	1.4965	0.1232	31.5952	0.0736	0.0253
Deep State Space Models (DeepState)						
25	44	1.4700	0.1521	21.9186	0.0596	0.0226
Deep Factor RNN Model (DF-RNN)						
100	42	13.7530	0.3461	260.7183	0.1002	0.1185
150	42	14.7166	0.3483	293.4810	0.0876	0.0669
200	42	15.1655	0.3506	305.8029	0.0981	0.0508
Benchmark and Competition Methods						
Method						
Naive2		2.395	0.1838			
Comb		4.582	0.2205			
ARIMA		0.943	0.1398			
ETS		1.824	0.1731			
Smyl (#1)		0.893	0.0933			
MM (#2)		0.819	0.1151			
Doornik et al. (H#1)		0.801	0.0891			

Table 3: Results for hourly M4 data.

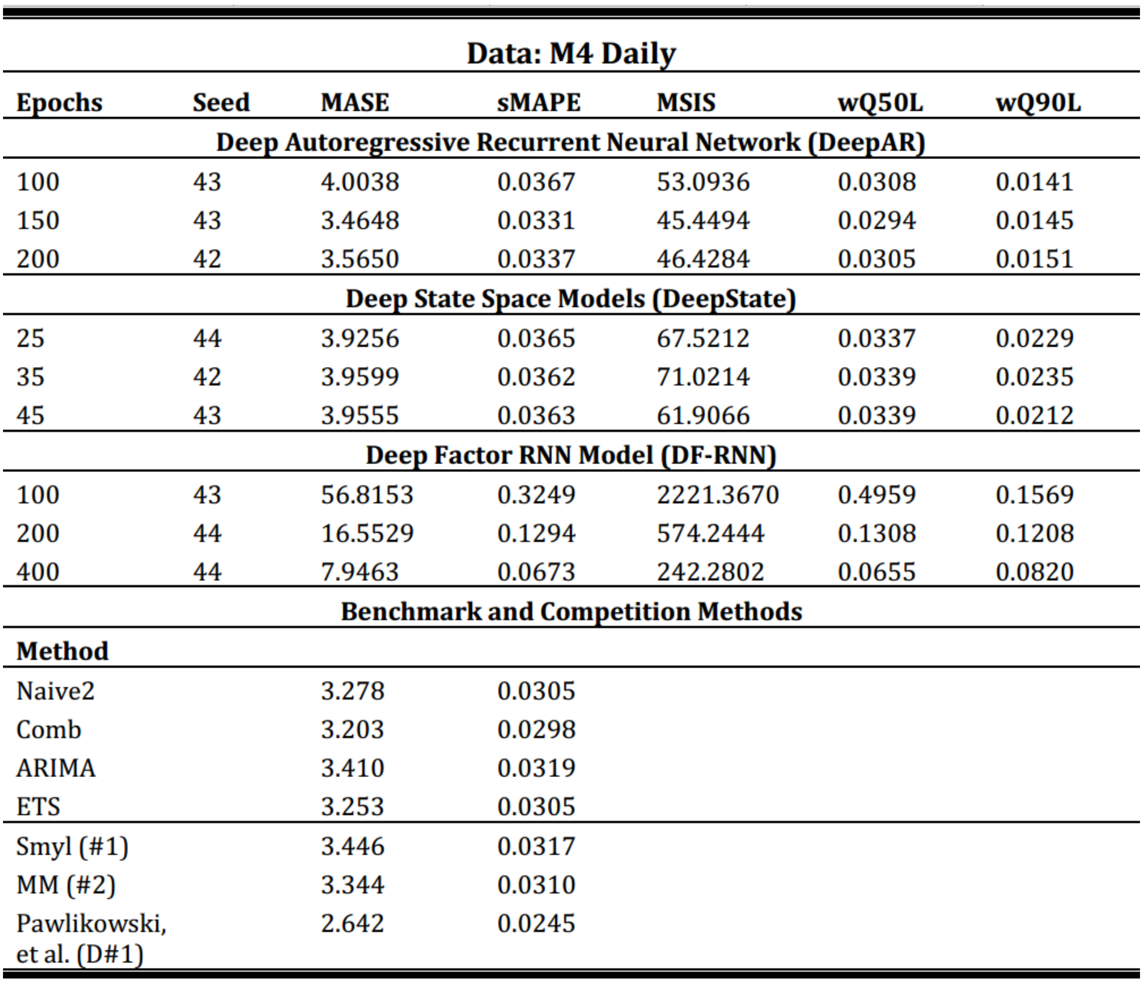
M4 Daily

Next, I will discuss the results of the daily subset that are shown in table 4. The subset consists of 4227 series and the task is to predict the next 14 observations, i.e. 2 weeks.

Again, the accuracy of the DF-RNN is nowhere near to being competitive. It first seems as the model converges very slowly for around 200-250 epochs. But after roughly 250 epochs the loss does not decrease further and the results are quite bad.

For DeepAR it seems as if more epochs are required to learn the mapping more accurately. Utilizing 200 epochs and 50 batches per epoch, the loss does not decrease any further after around epoch 100 to 200. DeepAR achieves a similar level of accuracy as ARIMA and the winning method of Smyl. Compared to Naive2, Comb, ETS, as well as the runner-up method of Montero-Manso it performs slightly worse. Note that the Naive2 method performs suprisingly well. It is conceivable that further hyperparameter tuning of DeepAR may lead to further performance improvements to a comparable level

M4 Daily - Results



Data: M4 Daily

Epochs	Seed	MASE	sMAPE	MSIS	wQ50L	wQ90L
Deep Autoregressive Recurrent Neural Network (DeepAR)						
100	43	4.0038	0.0367	53.0936	0.0308	0.0141
150	43	3.4648	0.0331	45.4494	0.0294	0.0145
200	42	3.5650	0.0337	46.4284	0.0305	0.0151
Deep State Space Models (DeepState)						
25	44	3.9256	0.0365	67.5212	0.0337	0.0229
35	42	3.9599	0.0362	71.0214	0.0339	0.0235
45	43	3.9555	0.0363	61.9066	0.0339	0.0212
Deep Factor RNN Model (DF-RNN)						
100	43	56.8153	0.3249	2221.3670	0.4959	0.1569
200	44	16.5529	0.1294	574.2444	0.1308	0.1208
400	44	7.9463	0.0673	242.2802	0.0655	0.0820
Benchmark and Competition Methods						
Method						
Naive2		3.278	0.0305			
Comb		3.203	0.0298			
ARIMA		3.410	0.0319			
ETS		3.253	0.0305			
Smyl (#1)		3.446	0.0317			
MM (#2)		3.344	0.0310			
Pawlikowski, et al. (D#1)		2.642	0.0245			

Table 4: Results for daily M4 data.

of these methods.

However, there is one outlier method by Pawlikowski et al. (2019) whose method performed disproportionately well on the hourly subset of the data. It is not simply better than all other hourly methods, it actually crushes them by a huge margin. Among the overall top ten methods, the method that achieves the second best MASE on the hourly data achieves an MASE of 3.194. Hence, Pawlikowski's MASE of 2.642 constitutes a relative performance improvement of staggering 17.28%. For comparison, the MASE range of the top ten M4 methods, excluding Pawlikowski, is within 3.194 and 3.446.

For the method of Pawlikowski et al. (2019) data are deseasonalized before applying various statistical models, such as ARIMA, simple exponential smoothing, Naive2, Theta method, etc. This yields a series of predictions which are then reseasonalized to restore seasonality. The final forecast is a weighted average of these individual predictions.

But, the performance of this model is significantly boosted using in-sample forecasting which may explain the performance edge of the model. This means the authors exploit patterns in the data by searching explicitly for time series with strong correlations. They basically search for time series y that looks similar to x but which is longer. Then, these data points of the longer series y are used to predict future observations of x. In this way, observations are incorporated in the prediction model that actually should not be available. From the perspective of series x, the utilized observations come from the future. This in-sample forecasting is possible because the time series come from different points in time and have different end dates. This can be thought of as having two time series of stock prices, for example Apple (A) and Netflix (N). Series A ends on January 31, whereas series N ends on February 14 of the same year. Your task is now to forecast series A for the time period from February 1 to February 14. Stocks in general exhibit strong correlations and it is a good idea to simply predict stock A to perform in the same way as stock N has performed during the given time period. Unfortunately, future values are usually not available to rely on during prediction which signifies a major problem of this method.

It should also be noted that 37% of the daily time series originate in the finance domain which are generally considered to be difficult to predict. Using in-sample forecasting vs. out-of-sample forecasting may explain the performance of Pawlikowski's model. Moreover, the comparatively good performance of the Naive2 method may also be an indicator of the presence of series that are difficult to forecast. In the absence of exploitable patterns in the data, predicting the next value to be equal to the previous one is not a bad approach.

Taking this into account, the (almost) untuned version of DeepAR may be closer to the top performing methods than previously thought. It is conceivable that further tuning improves the performance to a competitive level. Further research may be useful to compare DeepAR on other data without the possibility of in-sample forecasting and excluding finance data.

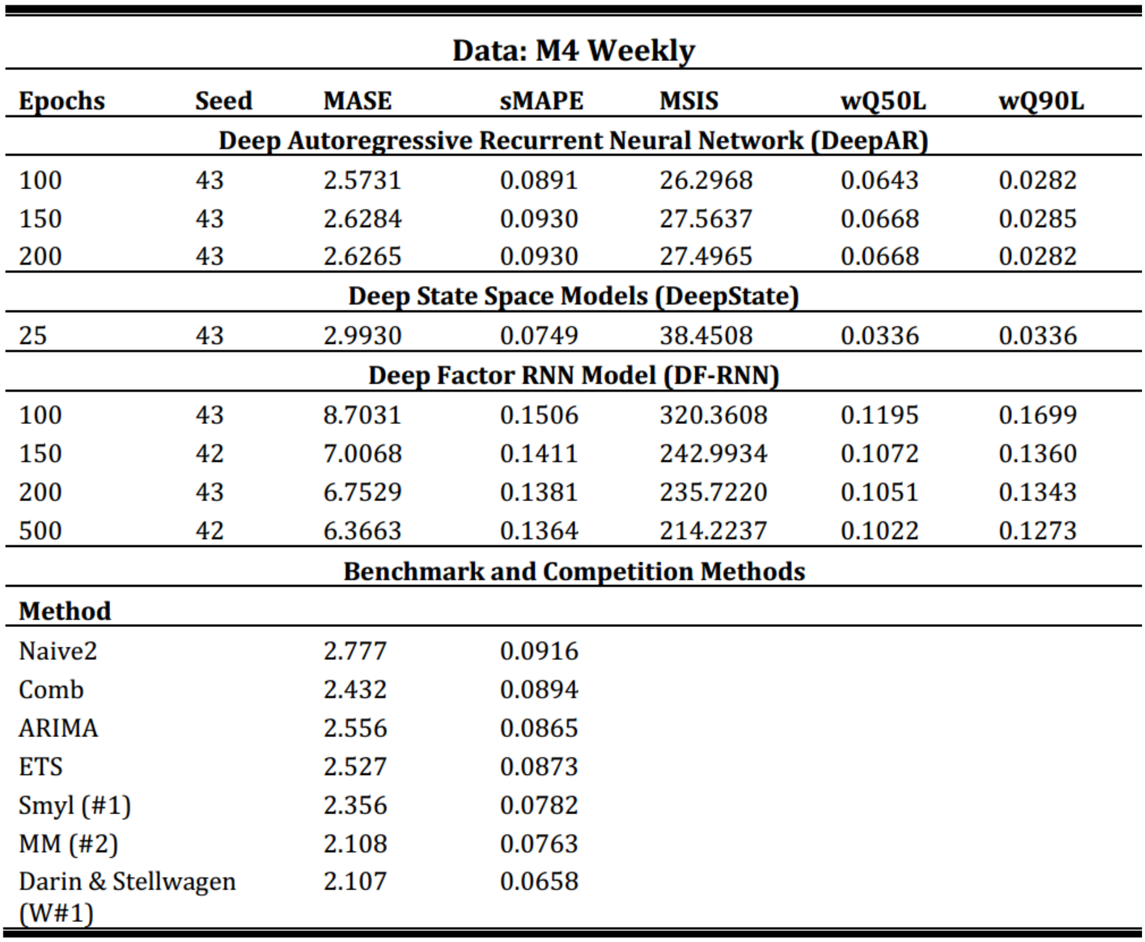
The presence of overlapping as well as finance data may distort the conclusions drawn from comparing the methods and may also be a reason that the rather simple Naive2 method performs better than all complex RNN methods, including the winning solution by Smyl.

DeepState performs worse than all other methods, except DeepFactor. In particular, it is not able to replicate the success of the automatic exponential smoothing (ETS) method. This is noteworthy because DeepState uses a complex RNN to fit the best SSM, whereas ETS uses a simpler approach for this.

M4 Weekly

Among the M4 subsets by frequency, the M4 weekly subset is the smallest one with 359 individual time series. The forecast range is 13 which is roughly equivalent to three months or one quarter. As before, DF-RNN does not fit the data accurately even after increasing the number of training epochs to 500. Because training loss does not decrease further after around 250 epochs and due to the fact that gluonts selects the model with the minimum training loss, another increase in training epochs is unlikely to lead to improve the model's accuracy.

M4 Weekly - Results



Data: M4 Weekly						
Epochs	**Seed**	**MASE**	**sMAPE**	**MSIS**	**wQ50L**	**wQ90L**
Deep Autoregressive Recurrent Neural Network (DeepAR)						
100	43	2.5731	0.0891	26.2968	0.0643	0.0282
150	43	2.6284	0.0930	27.5637	0.0668	0.0285
200	43	2.6265	0.0930	27.4965	0.0668	0.0282
Deep State Space Models (DeepState)						
25	43	2.9930	0.0749	38.4508	0.0336	0.0336
Deep Factor RNN Model (DF-RNN)						
100	43	8.7031	0.1506	320.3608	0.1195	0.1699
150	42	7.0068	0.1411	242.9934	0.1072	0.1360
200	43	6.7529	0.1381	235.7220	0.1051	0.1343
500	42	6.3663	0.1364	214.2237	0.1022	0.1273
Benchmark and Competition Methods						
Method						
Naive2		2.777	0.0916			
Comb		2.432	0.0894			
ARIMA		2.556	0.0865			
ETS		2.527	0.0873			
Smyl (#1)		2.356	0.0782			
MM (#2)		2.108	0.0763			
Darin & Stellwagen (W#1)		2.107	0.0658			

Table 5: Results for weekly M4 data.

Applying DeepAR on the weekly subset produces forecasts that are more accurate than the Naive2 method and comparable to the traditional methods as well as the combination benchmark. Comparing the performance to the overall winner of Smyl, DeepAR produces acceptable results. Further improvements by using model tuning or additional features are conceivable. However, the leading methods of Darin and

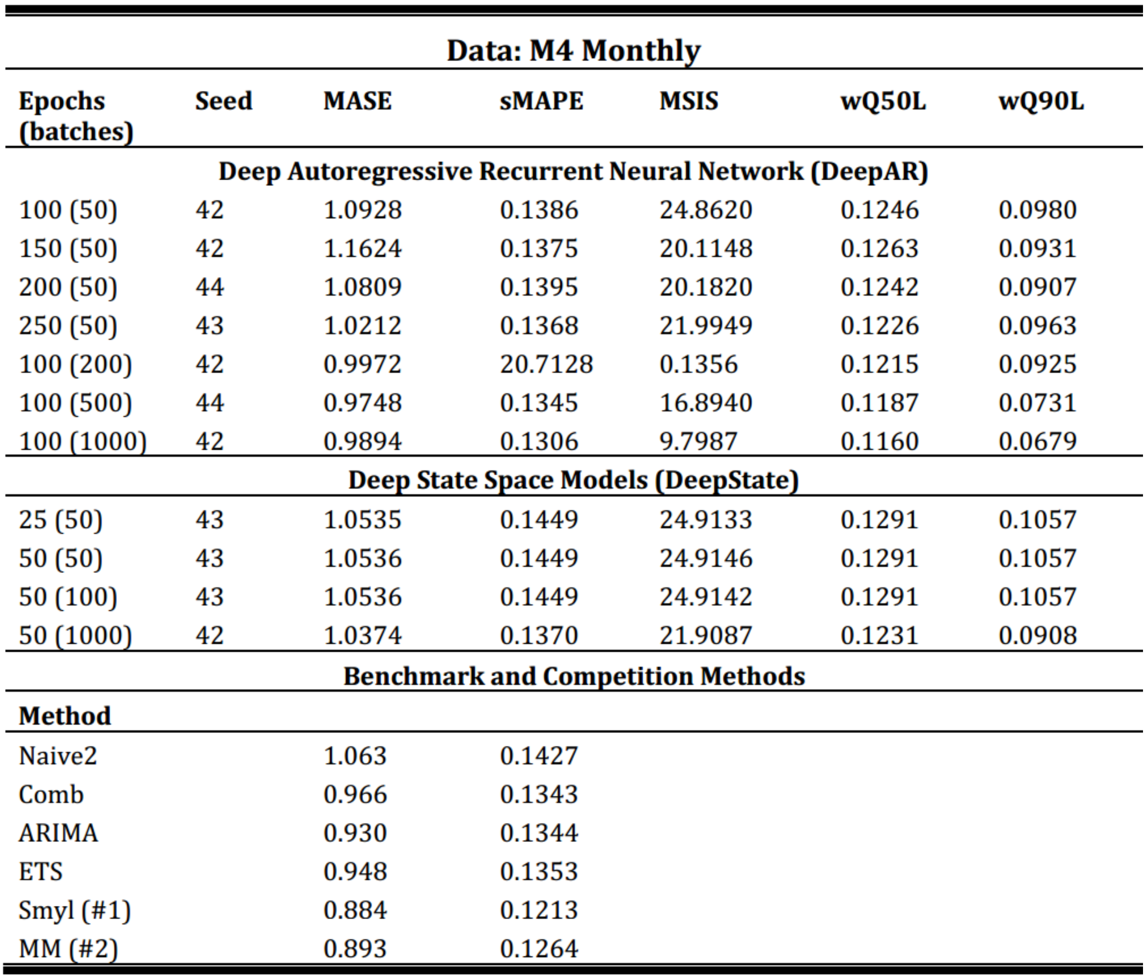
Stellwagen as well as runner-up of Montero-Manso are more accurate relying on local methods.

The performance of DeepState is again slightly worse than DeepAR but also worse than the Naive2 benchmark. As before, it is quite surprising that the DeepState model does not produce results of similar quality than ETS, which also uses SSM.

M4 Monthly

The last subset includes the monthly data of the M4 competition. It contains 48,000 time series and is the largest subset of all frequencies. DF-RNN failed again to produce accurate results using default values as well as experimenting with number of epochs, and batches per epoch (not shown in the table).

M4 Monthly - Results



Data: M4 Monthly

Epochs (batches)	Seed	MASE	sMAPE	MSIS	wQ50L	wQ90L
Deep Autoregressive Recurrent Neural Network (DeepAR)						
100 (50)	42	1.0928	0.1386	24.8620	0.1246	0.0980
150 (50)	42	1.1624	0.1375	20.1148	0.1263	0.0931
200 (50)	44	1.0809	0.1395	20.1820	0.1242	0.0907
250 (50)	43	1.0212	0.1368	21.9949	0.1226	0.0963
100 (200)	42	0.9972	20.7128	0.1356	0.1215	0.0925
100 (500)	44	0.9748	0.1345	16.8940	0.1187	0.0731
100 (1000)	42	0.9894	0.1306	9.7987	0.1160	0.0679
Deep State Space Models (DeepState)						
25 (50)	43	1.0535	0.1449	24.9133	0.1291	0.1057
50 (50)	43	1.0536	0.1449	24.9146	0.1291	0.1057
50 (100)	43	1.0536	0.1449	24.9142	0.1291	0.1057
50 (1000)	42	1.0374	0.1370	21.9087	0.1231	0.0908
Benchmark and Competition Methods						
Method						
Naive2		1.063	0.1427			
Comb		0.966	0.1343			
ARIMA		0.930	0.1344			
ETS		0.948	0.1353			
Smyl (#1)		0.884	0.1213			
MM (#2)		0.893	0.1264			

Table 6: Results for monthly M4 data.

The DeepAR algorithm produces forecasts that are comparable to the Naive2 and Comb benchmark. Using 100 epochs and 1000 batches per epoch leads to acceptable results but are not an improvement over traditional methods such as ARIMA or ETS. Compared to the overall winning method of Smyl, which was also the most precise method on the monthly subset, there are still considerable performance differences. Because the dataset is really big, a more powerful network (i.e. additional layers) may improve accuracy even further. It should be noted that there are some convergence issues with the algorithm for smaller batches. It may be the case that the dataset is too large or too diverse to be used with only 50 training examples per epoch. As a result the training loss did not decrease continuously for batch sizes below 200.

Another familiar finding are the results for the DeepState algorithm. Forecast results are slightly worse than DeepAR and slightly better than the Naive2 benchmark. Furthermore, the performance quality is inferior to traditional methods as well as the competition winners.

5.2 Conclusion

In the course of this project I applied three RNN-based algorithms (DeepAR, DeepState, and DF-RNN) to subsets of the M4 dataset and compared their performance to the competition methods. While I was able to answer how the algorithms performed on the M4 subsets, some questions remain and even more questions were raised.

1) A first implication is that DF-RNN has some major flaws of unknown origin. This makes it unusable for now until they are solved. Note that the authors used a different set of measures and datasets which made a direct comparison difficult. Applying the method to the M4 data with known benchmarks reveals that there are some serious issues with this approach or its implementation. Further research why this method does not deliver the desired results is necessary.

2) The untuned DeepAR algorithm yields accurate predictions that are comparable to traditional methods for some frequencies but not as good as the state-of-the art methods of the competition. It is conceivable that the use of additional features (covariates) or hyperparameter optimization can improve its performance to a competitive level. Unlike most of the competition methods, such as the winning method by Smyl, it can be used "out-of-the-box" by practitioners using either gluonts or AWS SageMaker's built-in version. This increases the value of DeepAR tremendously. Also, the performance may be limited due to the diverse nature of the M4 data. Applying the method to a homogeneous dataset may improve its performance beyond local methods. Comparing its performance on a set of highly related data to the M4 methods may be a rewarding topic for future research.

3) DeepState produces forecasts that are significantly worse than DeepAR and the benchmarks of the competition. However, it seems that DeepState captures at least

some of the broad patterns of the data. Additional research into why this algorithm does not fully capture the underlying process may help in solving its issues. Another interesting question is why this global approach does not lead to comparable results as the local ETS method. Both methods use SSM but the selection process of DeepState is global, whereas ETS uses a local statistical one.

4) A shared problem for the algorithms may be the suboptimal choice of hyperparameters. For now, training neural networks seems more like an art than a science. Often, the training process is dominated by trial and error. In the case of RNNs, this seems particularly true as some networks simply won't converge without obvious reasons. The choice of optimal values seems generally difficult. Possible future research into how this process can be automated without running several variations of this model may help to save computational costs, time, and improve accuracy.

5) Finally, the M4 data are highly diverse with respect to their underlying data-generating process. They are taken from various domains and across frequencies. The heterogeneous nature may have affected the performance of the algorithms in a negative way. Using a homogeneous dataset for the analysis of the RNN algorithms may yield completely different results and rankings than the M4 competition. In theory, this should improve the learnability of the underlying process. Comparing the global RNN methods with the M4 methods (local and global) can lead to interesting insights into the performance differences between local and global methods. It is likely that global methods outperform local methods only in settings with a lot of similar series. More empirical studies into how many series are required and to what degree the series should be related are necessary. In order to do this it may also be helpful to quantify the degree of similarity between time series in a dataset.

6) Whereas Smyl's method was able to outperform other methods even on such a highly diverse dataset, the introduced RNN algorithms did not. Note that the implementation of Smyl's approach is highly and customized to the M4 data. It cannot be easily reproduced or applied to other datasets for now. Why no other modern RNN algorithm could replicate his results remains a key question. It seems that for now traditional forecasting methods remain important tools in the toolkit of time series forecasters as no out-of-the-box RNN method seems to be able to replace them. If or in which particular area or forecasting they will fit in remains an open question.

6 References

[1] Adams, Douglas. "The Hitchhiker's Guide to the Galaxy." Del Rey Books. 2005

[2] Arturus (2018). "How it works." Retrieved from https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

[3] Graves, Alex. "Generating sequences with recurrent neural networks." *arXiv preprint arXiv:1308.0850 (2013).*

[4] Hyndman, Rob J. "A brief history of forecasting competitions." *International Journal of Forecasting (2019).*

[5] Hyndman, Rob J., and Anne B. Koehler. "Another look at measures of forecast accuracy." *International journal of forecasting 22.4 (2006): 679-688.*

[6] Kaggle (2019). "Web Traffic Time Series Forecasting." Retrieved from https://www.kaggle.com/c/web-traffic-time-series-forecasting

[7] Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos. "Statistical and Machine Learning forecasting methods: Concerns and ways forward." *PloS one 13.3 (2018): e0194889.*

[8] Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos. "The M4 competition: 100,000 time series and 61 forecasting methods." *International Journal of Forecasting (2019).*

[9] Montero-Manso, Pablo, et al. "FFORMA: Feature-based forecast model averaging." *International Journal of Forecasting (2019).*

[10] Pawlikowski, Maciej, and Agata Chorowska. "Weighted ensemble of statistical models." *International Journal of Forecasting (2019).*

[11] Rangapuram, Syama Sundar, et al. "Deep state space models for time series forecasting." *Advances in Neural Information Processing Systems.* 2018.

[12] Salinas, David, Valentin Flunkert, and Jan Gasthaus. "DeepAR: Probabilistic forecasting with autoregressive recurrent networks." *arXiv preprint arXiv:1704.04110 (2017).*

[13] Smyl, Slawek. "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting." *International Journal of Forecasting (2019).*

[14] Spiliotis, Evangelos, et al. "Are forecasting competitions data representative of the reality?." *International Journal of Forecasting (2019).*

[15] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." *Advances in neural information processing systems. 2014.*

[16] Wang, Yuyang, et al. "Deep Factors for Forecasting." *arXiv preprint arXiv:1905.12417 (2019).*