Technische Universität Berlin

Chair of Database Systems and Information Management

Master's Thesis

Empirical Comparison of Forecasting methods

Leonardo Araneda Freccero
Degree Program: ICT Innovation
Matriculation Number: 406022

Reviewers
Prof. Dr. Volker Markl

Advisor(s)
Behrouz Derakhshan
Bonaventura Del Monte

Submission Date
27.07.2021

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 27.07.2021

. . . . . .

Leonardo,Araneda Freccero

Zusammenfassung

Tipps zum Schreiben dieses Abschnitts finden Sie unter [**?**]

Abstract

The abstract should be 1-2 paragraphs. It should include:

- a statement about the problem that was addressed in the thesis,
- a specification of the solution approach taken,
- a summary of the key findings.

For additional recommendations see [**?**].

Acknowledgments

For recommendations on writing your Acknowledgments see [**?**].

Contents

List of Figures . . . . . **viii**

List of Tables . . . . . **ix**

List of Abbreviations . . . . . **x**

List of Algorithms . . . . . **xi**

1 Introduction **1**
1.1 Motivation 2
1.2 Research Question 3
1.3 Contribution 3

2 Scientific Background **5**
2.1 Time series decomposition 5
2.2 Time Series Forecasting 6
2.3 Evaluating forecasting performance 8
2.3.1 Error Metrics 9
Absolute Error 9
Root Mean Squared Error - RMSE 11
Mean Absolute Percentage Error - MAPE 11
Mean Average Scaled Error - MASE 11
2.4 Deep Learning forecasting methods 12
2.4.1 Sources of randomness 14
2.5 Benchmarking 16
2.5.1 Reproducibility 17
2.6 Gluon-TS 19
2.6.1 Algorithms 19
CanonicalRNNEstimator 20
DeepAR 20
DeepFactorEstimator 20
DeepStateEstimator 21
GaussianProcessEstimator 21
GPVAREstimator and DeepVAREstimator 21

LSTNetEstimator . . . 21
NBEATSEstimator and NBEATSEnsembleEstimator . . . 21
Naive2Predictor . . . 22
NPTSPredictor . . . 22
ProphetPredictor . . . 22
RForecastPredictor . . . 23
SeasonalNaivePredictor . . . 23
MQCNNEstimator, MQRNNEstimator . . . 23
SimpleFeedForwardEstimator . . . 24
TransformerEstimator . . . 24
2.6.2 Datasets . . . 24
2.7 Runtool . . . 26
2.8 Hypothesis testing . . . 29

3 Empirical Comparison . . . 30

Bibliography . . . 31

List of Figures

1 Exchange rate of two currencies from 1990 to 2013. 5
2 Electricity consumed by households. 5
3 Point forecast 7
4 Probability forecast 7
5 An example train test split of a time series with six datapoints. The red circle is the datapoint which prediction will be compared to. . . 9
6 Example of a four fold cross validation dataset split. The red circles are the datapoints which predictions will be compaired to. Worth noting is that the train dataset here only has one timeseries of length two, i.e. one less than the shortest timeseries in the test dataset. This example assumes a prediction length for the dataset of one datapoint. 10
7 Equation for calculating the absolute error 11
8 Equation for calculating RMSE 11
9 Equation for calculating MAPE 11
10 Equation for calculating MASE 12
11 A simple neural network for forecasting based on the SimpleFeedForwardEstimator in section 2.6.1. 13
12 Python script using the config from figure 13 to create four experiments 27
13 Config file describing two algorithms and two datasets. $from inherits the values from another node. 28

List of Tables

1 Overview of six benchmarking suites for ML, the column marked TS depicts whether they support time series forecasting and the DS column is whether they offer custom datasets. A is shorthand for Accuracy and S for Speed. 16
2 Requirements for making ML workloads reproducible. 19
3 Datasets available in GluonTS. 25

List of Abbreviations

NN Neural Network

DNN Deep Neural Network

RNN Recurrent Neural Network

List of Algorithms

1 Introduction

Time series forecasting, the ability to predict future trends from past data, is an important tool in science, for businesses and governments. However, it can be a double edged sword if the predictions are incorrect. Predicting the future is hard and predictions are only as good as the data and learning capabilities of the forecasting model used. Recently the Covid-19 pandemic had disastrous effects on society. Time series forecasting was used to predict the spread of the virus so that governments, hospitals and companies could plan accordingly to minimize its effects. However, many of the forecasts were overestimating the spread of the virus which lead to both organizational and health issues for hospitals, personnel and patients. With better forecasting solutions which would be capable of generating probabilistic forecasts this issue could have been avoided [21].

Improving forecasting accuracy is an active area of research and for that, new forecasting methods are continuously proposed [40, 37, 33, 34, 39]. As more forecasting methods are developed, these need to be compared in an accurate and reproducible way. The currently most popular method for comparing forecasting methods is through evaluating the algorithm on a couple of reference datasets [20]. As a part of this process, the datasets often undergo some preprocessing before a model is trained on them. Similarly, the hyperparameters of the algorithm are often tuned to fit the dataset in question. These steps are considered good practice as it allows the algorithm to perform optimally. However, if the method used to process the dataset is unclear in any way or if configuration details such as which hyperparameters used are missing, reproducing results becomes hard [28].

Some forecasting methods exhibit a non-deterministic behaviour where each subsequent run of the algorithm won't necessarily produce the same output. This is especially the case for deep learning algorithms as they are heavily relying on random processes such as dropout or random initialization of weights [43]. This makes it hard to reproduce results from these algorithms. Additionally, in the field of forecasting, multiple different metrics are employed in order to quantify the error of forecasts. This is due to some metrics being better suited for specific use cases. Often only a few of these metrics are used in papers when presenting new forecasting methods which makes reproducing results even harder.

In other disciplines of machine learning benchmarking suites are common tools for performing automatic reproducible comparisons between different algorithms. Some examples of these are MLBench and MLPerf. In time series forecasting

however, no such benchmarking suites exist instead results from competitions such as the M competitions are held as the reference which future papers compare to.

Reproducible results imply that an algorithm needs to produce the same output no matter when someone wishes to reproduce them. It should not matter if it is shortly after the algorithm was presented or a long time thereafter. This introduces difficulties as some algorithms are continuously being improved upon by their makers. If any of these improvements would change the predictive power of the algorithm this would make it impossible to reproduce any results. These types of performance changes can be handled by automated tests which ensure that the accuracy of the algorithm remains. However, defining such tests is hard for non-deterministic output and therefore accuracy regressions can pass through undetected [22]. Changes in predictive performance can also stem from third party updates of dependencies. These things are hard to control and potential problems are hard to foresee ahead of time.

1.1 Motivation

Previous comparisons of forecasting algorithms have found subpar performance of machine learning based approaches both in terms of accuracy and resource consumption when compared to classical methods such as Arima, ETS and Theta. However these comparisons are often made unfairly as the deep learning models which were tested were rather simple neural networks with few bells and whistles. More advanced deep learning methods such as DeepAR, DeepState, WaveNet etc. have been developed since. Thus, there is a clear benefit of thoroughly evaluating these modern algorithms in a fair and accurate way.

Accurate and fair comparisons can however only be made if the algorithms being tested all have an equal opportunity to perform well. Ensuring equal opportunity allowing forecasting methods to perform optimally is however hard and implementing a strategy for doing this is required when doing large scale comparisons. Additionally, the choice of error metric to use when comparing forecasting models is important as each error metrics has their own benefits and drawbacks. Using a flawed error metric can invalidate a seemingly fair comparison as some metrics are unreliable for certain applications or datasets. Implementing a strategy for when to use and not use certain error metrics is required in order to allow for fair comparisons.

Reproducibility of results is a core tenet of the scientific method. Despite this, being able to reproduce results from forecasting algorithms is not always straightforward [28]. Much of the difficulties regarding reproducibility stem from two core parts. Firstly, code and datasets are often not public and secondly the non-deterministic behaviour of certain forecasting models causes different runs of the

models to produce different results.

This thesis will focus on how fair, accurate and reproducible comparisons of modern forecasting methods can be made. Particular focus will be put on how comparisons can be made in a reproducible way for non-deterministic forecasting models. As part of this thesis, a system is implemented which automates the tuning and benchmarking of forecasting models. This system is then used to perform a large scale comparison of several modern forecasting algorithms over multiple datasets.

1.2 Research Question

This thesis has as an overarching goal to compare modern forecasting models. This question is complex in itself and can be separated into three sub-questions:

1. *Accuracy*: How to perform accurate comparisons between forecasting models?
2. *Fairness*: How can one fairly compare forecasting models?
3. *Reproducibility*: How to make comparisons reproducible?

1.3 Contribution

The main contribution of this thesis is Crayon, an open source library for benchmarking deep learning models in a fair, accurate and reproducible way. Many deep learning based forecasting algorithms are non-deterministic in nature which makes reproducing results hard. Crayon solves this by ensuring that the distribution of error metrics is reproducible. With distributions of error metrics, quantifying how good the accuracy of an algorithm is becomes a problem. To handle this, Crayon implements a custom scoring method, the "crayon score" for ranking forecasting models against eachother.

This thesis also investigates the efficiency of using various hypothesis tests such as the t-test and the Kolmogorov-Smirnov test to make non-deterministic output from forecasting models reproducible. Additionally, these tests are evaluated on forecasts generated by several modern forecasting solutions on several datasets. The practical benefits of applying hypothesis testing to verify distributions of forecasts is showcased on several modern forecasting algorithms trained on several real world datasets. Further, as a practical example this thesis shows how reproducibility by hypothesis testing can protect modern forecasting solutions from suffering accuracy regressions.

An additional contribution of this thesis is the analysis of 21 datasets with the purpose of identifying a representable subset to use when benchmarking forecasting methods.

As the final contribution a large empirical comparison where the predictive performance of multiple modern forecasting algorithms is compared over four popular datasets.

2 Scientific Background

A time series is a set of data points with a clear ordering in time. Some examples are the exchange rate between two currencies over time or the electricity consumption of households, see figures 1 and 2 [6].

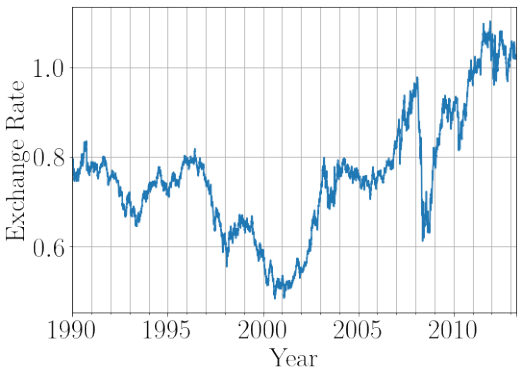

Figure 1: Exchange rate of two currencies from 1990 to 2013.

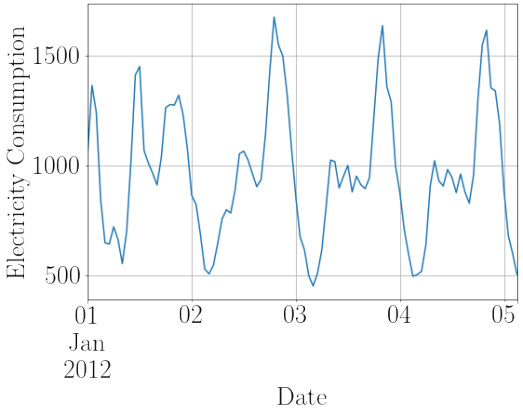

Figure 2: Electricity consumed by households.

Timeseries which increase or decrease in value over a longer period of time is said to have a upwards or downwards *trend*. If a timeseries has evenly distributed patterns such as the spikes in figure 2 the timeseries is said have a seasonal component with a frequency equal to the distance between the spikes. For the electricity timeseries this would be 24 as household electricity consumption follow the daily rythm of everyday life i.e. use more energy during the morning and evening with 24 hour intervals. If the timeseries exhibit spikes without a clear frequency, these are knowns as *cycles* [8]. The data which cannot be described by these features is known as the *remainder* [20]. Not all timeseries exhibit all of these patterns, for example, the timeseries in figure 2 does not display a trend.

2.1 Time series decomposition

Timeseries decomposition is the act of extracting features such as the seasonality and trend from timeseries data. This is often done in order to gain understanding

of the data [20]. While several methods of performing time series decomposition exists, one of the most powerful methods commonly used is STL decomposition [20]. After a timeseries has been split into its constituents one can quantify the strength of the trend and seasonality apparent in the data. This is done through compairing the variances of the seasonality S and trend T with that of the residual R. The strength of the trend F_t can thus be calculated as:

$$F_t = max(0, 1 - \frac{Var(R)}{Var(T + R)})$$

The strength of the seasonality F_s can be calculated as:

$$F_s = max(0, 1 - \frac{Var(R)}{Var(S + R)})$$

2.2 Time Series Forecasting

Time series forecasting is the art of predicting future trends of time series based on past observations. It is a key technology within many fields and is fundamental to the business decision process for many companies. Being able to interpret time-series in order to predict future trends makes it possible to plan for the future. For example power generation companies can adjust the amount of electricity which should be generated to match the expected demand and banks could make informed decisions about possible investment opportunities. There are many methods of generating such predictions, ranging from the use of simple heuristics to complex machine learning methods [20]. An overview of how to use deep learning to generate such forecasts is presented in section 2.4 and several state of the art forecasting models are presented in section 2.6.1.

Time series forecasting differs from common machine learning tasks such as image recognition or language processing in that predictions are not binary. More specifically, in time series forecasting a prediction is expected to be a close estimate of future values. Thus, quantifying the magnitude of the error from the ground truth is of importance in order to properly evaluate the accuracy of a forecasting model. This is commonly done by calculating an error metric, the choice of which varies depending on what is being forecasted and on preference of the forecasting practitioner. [20, 19, 49] In section 2.3.1 some such error metrics are presented.

Forecasts often comes in one of two forms; probabilistic forecasts or point forecasts. A point forecast is a forecast which consist of only one estimation of future values. This estimation could be either a single point in time or a series of points for several different timesteps. In figure 3 an example of a point forecast for several timesteps can be seen. Point forecasts has the disadvantage that they do not confer how accurate they are which may lead to prectitioners being overly confident

in the forecasts. This has historically caused several issues and more recently with covid-19 where over relying on point forecasts led to overly drastic measures for governments and hospitals [21]. Probabilistic forecasts such as the one in figure 4 includes this information by generating prediction intervals of the forecasts.

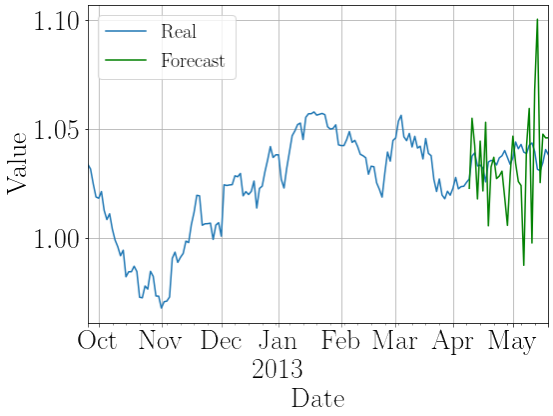

Figure 3: Point forecast

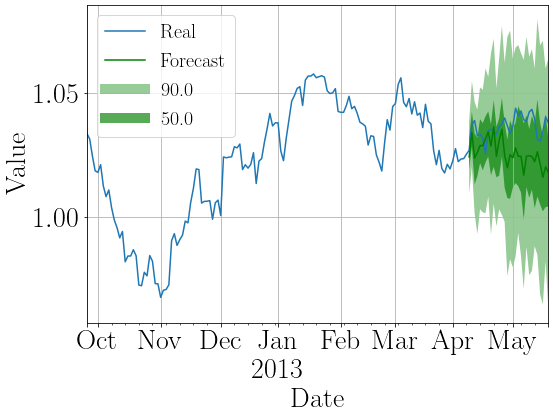

Figure 4: Probability forecast

Many methods exist for generating forecasts on timeseries data, two common groups of these are machine learning based methods and trivial methods. A trivial forecasting method would be, for example, to predict that the next value should be the same as the last recently observed value or the average of the last N observations. For certain timeseries, trivial methods perform competitively and as such these are often held as a baseline to which new forecasting methods are being compared too. Some more complex methods include those based on regression analysis such as linear regression or autoregressive models such as ARIMA [20]. Autoregressive models are a subset of regression models where the temporal ordering of the datapoints matter. I.e. autoregressive models use past observations to predict future values.

Lately, inspired by the successful use of neural networks in other domains, several neural network based approaches for time series forecasting have been introduced. Early implementations such as simple multi-layered perceptrons did not perform competitively with the more classical approaches such ARIMA. However modern deep learning models such as DeepAR [40] and MQCNN [48] have reported highly competitive accuracies.

Forecasting models for forecasting can also be split into further sub families such as whether they are local or global models. Local models are trained on individual time series while global models are trained on all time series in the training set. Essentially for a dataset with N timeseries, if using local models, one will have N individually trained models. If the algorithm would be a global model then only a

single model would be used for all N timeseries in the dataset. Generally speaking, local models performs well for timeseries with much historical data. However for short or new timeseries, local models often perform poorly due to the inability to train the model on sufficiently large amounts of data points. This is known as the cold start problem. Since global models train on the entire dataset across all timeseries, the length of any individual timeseries has less of an impact. I.e. generally global models suffer less from the cold start issue as the global model can use data from other timeseries as a basis for its predictions [47].

So far, we have seen the time series forecasting problem as generating a forecast from a single timeseries for example predicting future temperatures based on previously recorded temperatures. Predicting using singular timeseries such as this is known as univariate forecasting. Another family of forecasting solutions leverage related timeseries when making predictions. For example temperatures may depend on cloud coverage or the hours of sunlight. Algorithm which leverage information from related timseries are known as multivariate algorithms. The benefits of multivariate forecasting is that trends or patterns can be identified using these related timeseries in order to improve the accuracy for the target variable. For example, the temperature is higher whenever there is a large amou8nt of sun hours and no clouds.

2.3 Evaluating forecasting performance

If a model generates forecasts it is important to be able to quantify how good or bad its predictions are. This is generally done via a process called backtesting. When backtesting, first the algorithm is trained on a dataset, thereafter, the trained algorithm generates predictions on a test dataset. The generated predictions from the algorithm are then compared to the ground truth and a suitable error metric quantifying how wrong the prediction was from the expected value is calculated.

In other machine learning domains such as image recognition, the datasets used for training the models are disjoint from the test datasets. However, in the domain of time series forecasting this is not the case. Generally the train set contains the same time series as in the test set except for the last couple of data points for each time series. The number of removed data points is commonly the number of timesteps that a model should predict [20]. In figure 5 a train test split is presented of a time series with six data points. In this example, a forecasting algorithm should predict one timesteps into the future, thus, the last datapoint would be removed and the train set would contain five data points while the test series would contain six.

A popular method of increasing the amount of test data is to make the test dataset contain multiple copies of each time series where each one is successively

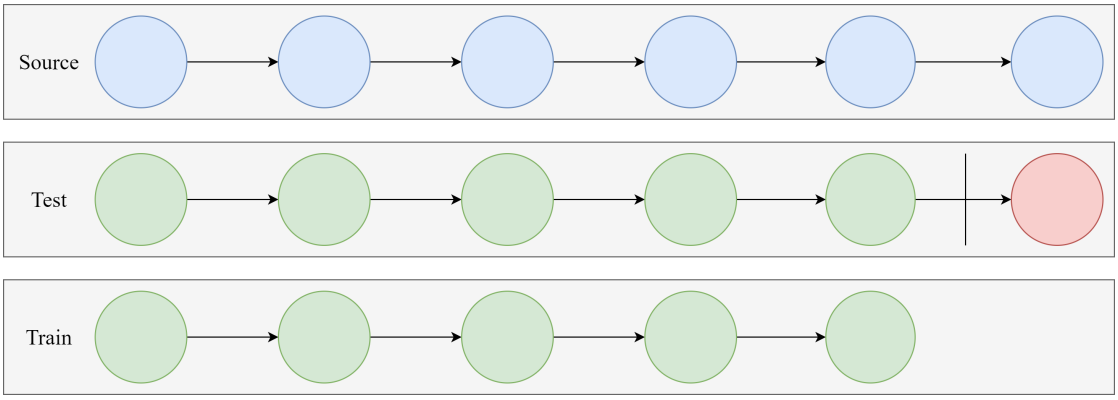

Figure 5: An example train test split of a time series with six datapoints. The red circle is the datapoint which prediction will be compared to.

shorter. The train dataset would contain the shortest of the truncated time series truncated one additional time. This process is called K-fold cross validation but is also known as forecasting on a rolling origin or time series cross validation. Performing a 3-fold cross validation on a dataset containing one time series of length six timesteps would result in a dataset containing three time series. An example of this is shown in figure 6. The advantage of the larger test dataset generated by this approach is that the performance of a model can be more accurately determined as it is excercised on more unseen data [20].

2.3.1 Error Metrics

When a forecasting model generates predictions on a timeseries, one needs to be able to quantify the error of the prediction from true observed values. In order to do this many different error metrics exists each with its own benefits and drawbacks. Often the choice of metric is dependent on data, and business application where the prediction will be used. In this section some of the error metrics which are commonly occurring in literature are presented. These metrics are calculated automatically when performing a backtest in GluonTS, see 2.6 for details about this process. Note that only a subset of the metrics available in GluonTS is presented here to form a basis for subsequent chapters.

For the remainder of this section a forecast is referred to as $\hat{Y}$ and the expected values are referred to as Y.

Absolute Error

The absolute error is calculated as the distance between the predicted value and the ground truth. An absolute error is positive and a value of 0 is optimal. The

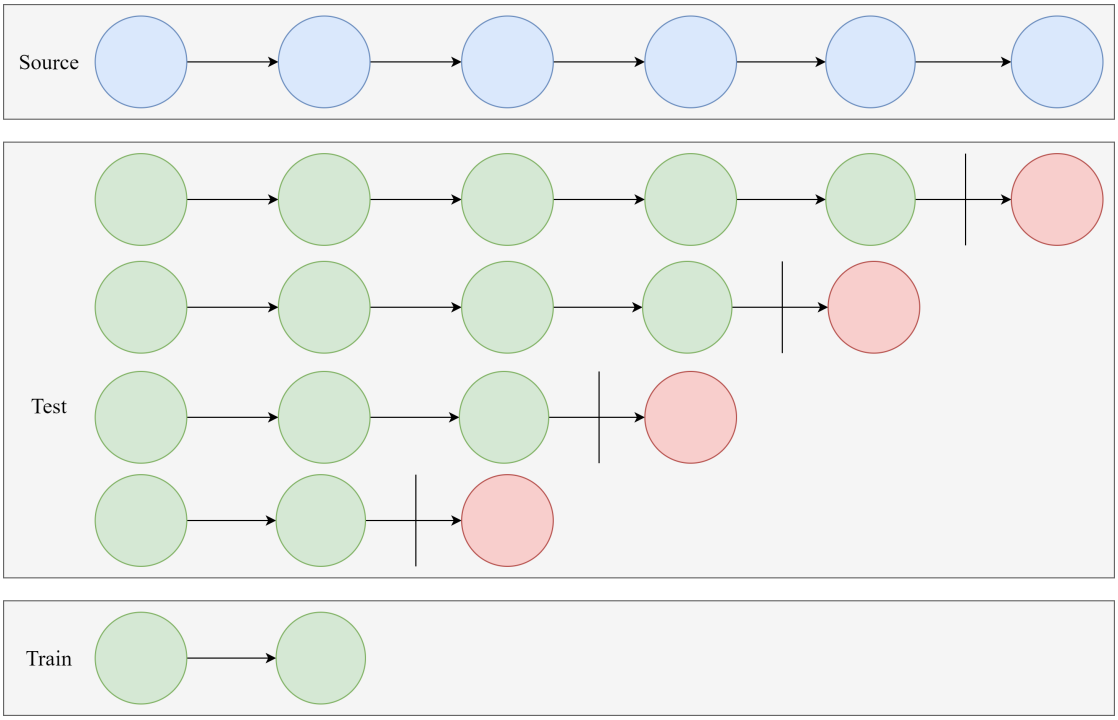

Figure 6: Example of a four fold cross validation dataset split. The red circles are the datapoints which predictions will be compaired to. Worth noting is that the train dataset here only has one timeseries of length two, i.e. one less than the shortest timeseries in the test dataset. This example assumes a prediction length for the dataset of one datapoint.

absolute error is an intuitively simple error, thus it can easily be understood and discussed when comparing algorithms.

However intuitive the absolute error is, there is a a key issue with this error and that is that it is scale dependent. I.e. the greater the values of the dataset being predicted on, the larger the absolute error will be. This makes it impossible to compare the accuracy of an algorithm across different datasets unless they have the same scale. [20] Furthermore, scale dependent metrics such as these may not be suitable even for comparing between timeseries within a single dataset. Imagine a dataset such as the electricity dataset (see table 3) where each timeseries is the electricity consumption from a household. One household may contain many people and appliances while another may be a factory full of equipment with high power demands. These two timeseries would have two very different scales thus making it hard to compare the accuracy of the predictions.

$$AbsoluteError = \sum |Y - \hat{Y}|$$

Figure 7: Equation for calculating the absolute error

Root Mean Squared Error - RMSE

The root mean squared error is another scale dependent error such as the absolute error. RMSE is calculated by taking the square root of the mean of the squared error. The RMSE thus punishes larger errors more than smaller errors. The RMSE is more complicated to interpret than for example the Absolute Error due to the non-linear nature, despite this, RMSE is widely used in practise [20, 3].

$$RMSE = \sqrt{mean((Y - \hat{Y})^2)}$$

Figure 8: Equation for calculating RMSE

Mean Absolute Percentage Error - MAPE

Percentage based errors such as the Mean Average Percentage Error normalizes the errors between 0 and 1 thus MAPE can be compared across datasets with different scales. While this is beneficial, MAPE suffer from other issues. For example MAPE becomes infinite or undefined if the ground truth is 0 for any parts of the prediction. Further issues exists for example that it penalizes negative errors more than positive errors. These issues has lead to variations of MAPE to be created such as Symmetric MAPE [20, 3].

$$MAPE = mean(\frac{|Y - \hat{Y}|}{|Y|}))$$

Figure 9: Equation for calculating MAPE

Mean Average Scaled Error - MASE

MASE is a metric which takes the average error across all timeseries and scale is based on the error from a reference forecasting. An example a model which

MASE uses for reference would be the Naive2Estimator detailed in section 2.6.1. Scaling the forecasting error in this way causes the MASE metric to be scale independent which makes it suitable when comparing across datasets. A MASE of 1 corresponds to that the algorithm under test is equally good as the naive model. A MASE below 1 means that the current model performs better than the naive model while a MASE above 1 means that the model performs worse than the naive baseline model [20, 3].

In certain scenarios, for example if the reference model would result in a perfect prediction a division by zero would occur and MASE would tend towards infinity. This is a major drawback of the MASE metric which renders it unusable for certain datasets.

In equation 10 the denominator is the error of a seasonal naive model. Y_t means the value at time t and Y_{t-m} means the value of the timeseries at the previous period, i.e. if the expected seasonality is 24h Y_{t-m} would become the value 24 timesteps previously.

$$MASE = \frac{mean(|Y - \hat{Y}|)}{mean(|Y_t - Y_{t-m}]|)}$$

Figure 10: Equation for calculating MASE

2.4 Deep Learning forecasting methods

Neural Networks (NN) have in recent years been successfully applied in many different domains such as image recognition and natural language processing. However, within the field of forecasting NN based approaches has performed subpar compared to more traditional statistical methods [27, 28, 34, 36]. This relative inefficiency between NN based approaches and classical methods and the success of NN in other domains has resulted in much research being made in improving tools and NN based models for forecasting [6].

A simple neural network, similar in architecture to the network described in section 2.6.1 is presented in figure 11.

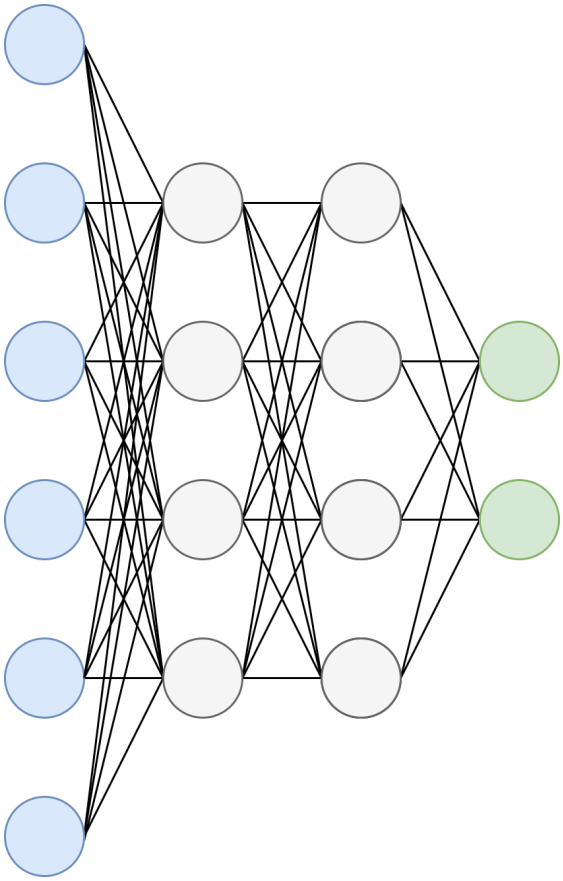

Figure 11: A simple neural network for forecasting based on the SimpleFeedForwardEstimator in section 2.6.1.

This very small network consists of four fully connected layers of different sizes with six input nodes, two fully connected hidden layers with four nodes each, and an output layer of two nodes. In this network, the input layer (marked in blue) has the same amount of nodes as the number of timepoints in the past which the network should make use of. This is will in the remainder of the thesis be called the *context length* of the network. The output nodes (in green) for this network produce a prediction of two timesteps. Thus this network has a *prediction length* of two timesteps. The context length and the desired prediction length are often exposed as hyperparameters for time series forecasting algorithms since they vary between datasets.

An issue with the simple network in figure 11 is that each timestep in the forecast is not influenced by the previous one. i.e. the prediction at timestep N does not depend on the prediction at timestep $N-1$. In real life scenarios, timeseries data often has a temporal dependency, for example, the temperature two days from now is affected by the temperature of tomorrow. This dependency is lost in simple neural network architectures such as this one. An alternative neural network architecture which can make use of temporal effects is known as a Recurrent Neural Network (RNN).

RNNs reuse the output of their nodes as inputs for the next iteration of the algorithm. This recurrent link causes previously seen values in the sequence to be represented in a hidden internal state. While this makes RNNs suited for use with sequential data such as time series, they become hard to train successfully as long term dependencies are lost [10]. One of the most successful techniques for use in RNNs to solve this is the Long Short-Term Memory (LSTM) node architecture. LSTM nodes enables RNNs to remember previously seen data for longer and forget or ignore parts of it [41].

Some issues with NN based approaches is that they require a large amount of data to be trained on properly. This is due to the fact that NNs are prone to overfit which makes them unable to generalize well to new data [43]. Despite this, NN has several advantages over classical approaches such as the capability to learn non-linear patterns from the data. Additionally, NN models are capable of cross learning between related time series without the need of manual feature extraction [42].

2.4.1 Sources of randomness

Neural networks heavily rely on random processess to function and thus randomnes is introduced at several points in a forecasting workflow. Some randomness can be introduced whenever an neural network is initialized as the weights and biases of a networks often are set randomly. Similarly, when training a neural network randomness can be introduced if optimization techniques such as dropout is used

as dropout randomly sets weights in the network to 0 at certain points in the training loop which causes the algorithm to be less prone of overfitting the data [43]. Even the method used for training neural networks, stochastic gradient descent introduces randomness.

Deep NNs are notoriously slow to train when compared to many other methods, thus lowering the time spent for training and inference is often done by parallelizing algorithms such that they run on several threads on a CPU or on hardware accelerators such as GPUs, FPGAs or ASICs. The execution speed of a forecasting model is thus highly dependent on what hardware the models are running on, however the hardware configuration also impacts the variance of the accuracy of ML models [50]. Parallelization of an algorithm often introduces a larger amount of data shuffling which impacts the rate of convergence of an algorithm. Parallelizing a forecasting model may make it unstable so that it cannot converge as will be shown in chapter 3. Due to this it is important that the hardware used when training and tuning forecasting models is presented in detail so that results can be properly reproduced [35].

A common method for reducing the randomness associated with ML models is to fix the seed used by the random number generator to a known number. While this removes randomness from for example random initializations or from backpropagation [9], other sources of randomness such as that introduced by the hardware or the OS remain. Furthermore, fixing the seed can be seen as an additional hyperparameter which needs to be tuned as it directly impacts the predictive performance of forecasting methods. While there are some major benefits of fixing the seed, there are also drawbacks. For example,luck becomes a factor in what accuracy can be expected for an algorithm. I.e. the accuracy of a model may seem highly competitive while if another seed would be used the accuracy of the model could change significantly [9]. In real life usecases such as when forecasting are used in companies or organizations, setting the seed may be beneficial or even necessary in order to optimize its performance for an individual task. In research however, setting the seed without reporting what it was can bias results and make them technically irreproducible [9, 35, 11].

One method of lowering the variance of machine learning methods is to use ensambles of forecasting methods as per the Bootstrap AGGregatING (Bagging) technique [12]. An ensamble is a set of multiple different ML models with slightly different configurations which are all trained in parallel.Bagging extends this such that a the input dataset is randomly sampled to create multiple independent datasets. Each model is then trained on one of these datasets and their forecasts are then combined via some voting or weighting scheme [12]. While bagging is often associated with decision trees they are also suitable for use with deep learning forecasting models such as the NBEATS Ensamble Estimator discussed

in section 2.6.1.

K-fold cross validation which was discussed in section 2.3 can also lower the variance of forecasting algorithms as it introduces multiple different train-test splits from a single dataset [12].

2.5 Benchmarking

Backtesting is a fundamental part of benchmarking, however benchmarking is also concerned with ensuring that different models can be fairly and reproducibly compared to each other [18]. In other domains of machine learning several benchmarking solutions exists, most notably MLPerf [31], MLBench [5], DLBS [45] and UCR [13]. Despite the success of benchmarks in other ML domains no established "go-to" benchmark exists in the domain of time series forecasting [18].

Attempts has been made to create such a benchmark, one such example is *Libra* [8] which uses custom datasets containing timeseries sampled from various well known datasets. The sampling methodology used in Libra was aimed to create datasets with a high amount of diversity between the timeseries. Due to this diversity, the datasets in Libra are not suitable for global forecasting models as they are highly unrelated. In a nutshell, Libra focuses on benchmarking the accuracy and time-to-result for univariate forecasts generated by local models.

Name	TS	Integration	DS	Focus	Info
Libra	X	R	X	A, S	Diverse datasets for univariate local models
UCR	X	CSV	X	A	Dataset repository with reference accuracies.
MLPerf		Docker, CLI		S	Benchmarking for hardware and ML frameworks
PMLB	X	Python, R	X	-	Dataset repository with 298 datasets for regression and classification.
MLBench		Kubernetes		S	For distributed ML frameworks
DLBS		Docker, Python		S	Benchmarking for hardware and ML frameworks

Table 1: Overview of six benchmarking suites for ML, the column marked TS depicts whether they support time series forecasting and the DS column is whether they offer custom datasets. A is shorthand for Accuracy and S for Speed.

Instead of using benchmarking suites when comparing forecasting models, new forecasting solutions are instead pitted against each other in forecasting competitions. The Makridakis Competitions organized by Makridakis Spyros et.al. are the most well known of these and has at the time of writing undergone five iterations; the M1 in 1983 [26], M2 in 1982 [25], M3 in 2000 [27], M4 in 2018 [28] and the M5 competition in 2020 [29]. These competitions have been highly impactful for the field of forecasting and has been the norm to which researchers compare too. However it was not until the M3 competition that NN based forecasters were included. These performed poorly compared to the non-NN algorithms which can partly be attributed to the limited size of the M3 dataset of 3003 timeseries [27]. This shortcoming of the M3 dataset lead to the creation of the M4 competition which had a similar but much larger dataset containing 100 000 timeseries [28]. Despite this, the findings of the M4 competition was inline with that of the M3 competition in that approaches purely using NN performed worse than even simple models. However, the M4 competition winner was a hybrid approach which combined RNNs with classical models. The latest competition, the M5, was focused on retail sales forecasting and found for the first time that ML based solutions, primarily Gradient Boosting Trees but also NN approaches such as DeepAR outperformed classical statistical methods [29].

In 2015 a thesis comparing forecasting solutions was published with the topic: *Benchmarking of Classical and Machine-Learning Algorithms (with special emphasis on Bagging and Boosting Approaches) for Time Series Forecasting* [36]. This thesis put the focus on performing a thorough dataset analysis and evaluating 14 algorithms on three real world datasets; Tourism, M3 and the NN5 [2]. Each algorithm was evaluated on multiple versions of each of these datasets with different preprocessing applied to them. In addition to evaluating the algorithms on the three real life datasets, a simulation study on artificial data was conducted. The purpose of this was to identify strengths and weaknesses of the algorithms for timeseries with simple characteristics such as only having trend or seasonality.

It is unclear whether time series cross validation was used as well as how many times each algorithm was executed. Further, as the title suggests there was little emphasis on deep learning methods and only a very simple neural network, similar to the one presented in figure 11 was used. This neural network performed worse than the naive approach for all datasets except for on NN5 [36].

2.5.1 Reproducibility

The ability to reproduce scientific results within the field of machine learning has been a hot topic in recent years with major conferences such as NeurIPS announcing dedicated reproducibility programs [35]. Certain fields of ML research such as within healthcare is reportedly in a *reproducibility crisis* [9, 32]. Surveys con-

ducted in 2018 reported that only 63% of 255 ML publications could be reproduced and only 4% of them could be reproduced without the original authors assistance [35]. Another survey in Nature sent out to more than 1500 scientists from various diciplines in 2016 reported that 50% researchers failed in reproducing even their own experiments [7]. Also Makridakis et.al., the organizers of the M-competitions, urged researchers to improve the reproducibility of their work. The following is a quote from their paper summarizing the M4 competition: *"We believe that there is an urgent need for the results of important ML studies claiming superior forecasting performance to be replicated/reproduced, and therefore call for such studies to make their data and forecasting algorithms publically available, which is not currently the case"* [28].

There are multiple methods of defining reproducibility. Pineau et.al. defined reproducibility as being able as *re-doing an experiment using the same data and same analytical tools* [35]. This definition aligns with that of Beam et.al. however, they point out that reproducibility is not concerned with the validity of the claims of a paper, only that practical results can be recreated [9]. McDermott et.al. further specifies the term reproducibility into three parts *technical*, *statistical* and *conceptual*. For technical reproducibility, the code and datasets used should be made publically available and in order to be statistically reproducible, the variance of the results should be published. To be conceptually reprodicble, it was important that multiple datasets should be used for the comparisons [32].

In addition to the requirement that code and datasets should be made available, Pineau et.al. suggested that standardized tools for supporting reproducibility should be developed and that any metrics used should be sufficiently specified. Furthermore it was argued that encapsulation tools such as Docker should be used as these would remove OS and dependency related issues [35]. This would also resolve issues pertaining to different versions of code being used [9]. The random seed which was discussed in section 2.4.1 is also an important value which should be reported as it can drastically affect the performance, especially for neural network based models [9, 11]. Increasing the use of statistical tests is another key aspect for improving the reproducibility [32]. In addition to this forecasting models should be executed multiple times with as much variation as possible between each run to further improve the accuracy and the statistical reproducibility of the results [11].

A summary of the requirements for achieving reproducibility is presented in table 2.

Type of reproducibility	Requirements
Technical	Code and datasets should be publically available. If random seed is set it should be reported. Specific versions of code should be detailed and assets such as Docker containers provided.
Statistical	Variance of results should be published and statistical tests should be used.
Conceptural	Multiple complimentary datasets should be used for the comparison.

Table 2: Requirements for making ML workloads reproducible.

2.6 Gluon-TS

Gluon-TS is an open source library which includes several deep learning algorithms, datasets and tools for building and evaluating deep learning forecasting methods [4, 6, 3]. In GluonTS, the *Estimator* class corresponds to the implementation of a forecasting method. An *Estimator* can be trained on a dataset in order to generate a *Predictor* object. This *Predictor* can then ingest timeseries in order to produce forecasts for them.

In order to make it easy to evaluate the performance of an algorithm, GluonTS offers the *backtest_metrics* function. This function trains an *Estimator* on a train dataset, the generated *Predictor* is then used to generate predictions on a test dataset. Each of the timeseries in the test dataset will have the final N datapoints removed prior to passing them to the *Predictor*. The *Predictor* then takes the remaining time series data and produces a prediction of the same length N. Thereafter, the previously truncated datapoints are compared with the generated forecast in order to calculate error of the prediction from the ground truth [3]. There are multiple ways to calculate this error some of these are presented in detail in section 2.3.1.

GluonTS also contains Dockerfiles which makes it easy to create docker images with installations of GluonTS inside. These images are compatible with AWS SageMaker which allows them to be executed both in the cloud as well as locally. These images executes the *backtest_metrics* function of GluonTS when run and can thus be used to generate error metrics for any *Estimator* class in GluonTS [3].

2.6.1 Algorithms

GluonTS offers several forecasting models which can be used to generate predictions on timeseries data. Most of these algorithms are presented below. Only

algorithms which were well documented in GluonTS or which I could explain by sifting through the source code are presented below.

CanonicalRNNEstimator

The CanonicalRNNEstimator is a bare bones RNN with a single layer of LSTM cells and is capable of producing probabilistic forecasts [3].

DeepAR

DeepAR is a RNN which produces probabilistic forecasts and was presented in *DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks* [40]. The model learns a global representation across all timeseries in a dataset and learns to identify seasonal behaviours in the data. DeepAR automatically adds certain meta information to the timeseries such as *day-of-the-week*, *hour-of-the-day*, *week-of-the-year* and *month-of-the-year*. Normally this type of meta information is manually added by the data scientist using the model, thus automating this procedure minimizes manual feature engineering. Another feature of DeepAR is the possibility to choose a likelihood distribution according to the data that it is to be trained on. Two different distributions are used by the original authors, the Gaussian Likelihood for real valued data and the negative binomial likelihood for postive count-data.

DeepFactorEstimator

The DeepFactorEstimator was presented in the paper *Deep Factors for Forecasting* in 2019 and consists of two main parts, a global and a local model [47]. The global model is a deep neural network (DNN) which is trained across all timeseries in order to capture complex non-linear patterns between the timeseries. The local model is a simpler model which is meant to capture local trends and patterns of individual timeseries. This hybrid architecture is thus able to leverage the DNN capability of learning complex patterns as well as having the computational efficiency of local models. Three different versions of the DeepFactorEstimator is presented in the original paper, the one implemented in GluonTS is the DF-RNN version. This version uses a second RNN to model the local timeseries. The DF-RNN presented in the paper performs better than Prophet and MQ-RNN on the Electricity, Taxi, Traffic and the Uber dataset both for short and long term forecasts. DeepAR performed worse than DF-RNN on average but was more accurate on the short term forecasts for the Uber dataset. The Uber dataset is not available as part of GluonTS however it is mentioned here for completeness, the other datasets are detailed in table 3.

DeepStateEstimator

The DeepStateEstimator combines linear state space models for individual timeseries with a jointly learned RNN which is taught how to set the parameters for the linear state space models. This has the benefit that the labour intensive tuning of multiple state space models is done automatically by the RNN without sacrificing performance. This approach was shown to perform better than DeepAR on the electricity dataset and on 4 out of 6 metrics on the traffic dataset. Furthermore, this approach outperformed ARIMA and ETS on all datasets [37].

GaussianProcessEstimator

The GaussianProcessEstimator is a local model where each timeseries in the dataset is modeled by a single gaussian process. The specific kernel used in the gaussian process can be specified as a parameter [4].

GPVAREstimator and DeepVAREstimator

The GPVAREstimator is a RNN model for handling multivariate timeseries which uses a gaussian copula technique for automatic data transformation and scaling. It utilizes a dimensionality reduction technique which minimizes the complexity of calculating a covariance matrix from $O(n^2)$ to $O(n)$. This allows much larger multivariate timeseries datasets to be used with it than what was previously possible. Additionally, the GPVAREstimator was compared against other multivariate forecasting algorithms, amongst them was a multivariate version of the DeepAR algorithm. This augmented version of DeepAR is implemented in GluonTS under the name DeepVAREstimator [39].

LSTNetEstimator

The LSTNetEstimator is a hybrid approach where long term patterns of the data is captured by a neural network architecture, these long term patterns are combined with a classical autoregressive model when generating predictions. The neural network architecture consists of four parts, a CNN, a RNN a novel RNN-skip layer and a fully connected layer. The RNN-skip layer makes up for the incapability of the LSTM cells in a RNN to remember very long term information by only updating the cells periodically [24].

NBEATSEstimator and NBEATSEnsembleEstimator

In the paper *N-BEATS: Neural basis expansion analysis for interpretable time series forecasting* [34] a univariate deep learning model for forecasting named N-

BEATS is presented. The authors of N-BEATS expressed that their goal with this algorithm was to disprove the notion that deep learning models had inferior performance compared to classical models. The authors reported a superior accuracy of N-BEATS model over all other models it was compared to.

The N-BEATS algorithm is an ensemble of 180 neural networks which were all trained to optimize for different metrics such as MASE, sMASE, MAPE and six different context lengths. Furthermore, bagging was used by running multiple runs of the algorithms with random initializations of the networks. A prediction of the N-BEATS model is the median value of the predictions of the algorithms in the ensemble.

Each model in the ensemble consists of multiple fully connected layers chained together to form blocks of networks. Each block is in turn chained together in order to form a stack. These stacks are also chained in order to form the final network.

GluonTS implements the N-BEATS model as the NBEATSEnsambleEstimator and the smaller algorithms in the ensemble as the NBEATSEstimator class. There are some differences between the implementation in the original paper and in GluonTS. Specifically, the training data is sampled differently in GluonTS than how it was in the source paper [4].

Naive2Predictor

the Naive2Predictor is a GluonTS implementation of the Naïve 2 forecasting method used as a reference in the m4 competition [28]. The Naive2Predictor predicts the future values to be that of the last datapoint in the timeseries adjusted by some seasonality. In GluonTS this seasonality can be deduced from the frequency of the data or by passing a custom seasonality via the *season_length* parameter [4].

NPTSPredictor

The NPTSPredictor is not a NN instead it predicts future values by sampling from previous data in the timeseries. The way that the samples are selected from the previous data can be modified via the hyperparameters. One can sample uniformly across all previous values in the timeseries or bias the sampling to more often sample from more recent datapoints depending on which kernel is used [4].

ProphetPredictor

Prophet is a nonlinear regression model developed at Facebook which frames the timeseries forecasting problem as a curve fitting exercise [20]. This is different

from many other forecasting models for timeseries which leverage the temporal aspect of timeseries in order to generate forecasts. Prophet was created with three goals in mind; it should be easy to use for people without much knowledge about timeseries methods, it should work for many different forecasting tasks which may have distinct features. Finally it should contain logic which makes it easy to identify how good the generated forecasts of Prophet are [44].

RForecastPredictor

The RForecastPredictor is a wrapper which allows a user of GluonTS to use the popular forecasting package *forecast* inside of GluonTS. The forecaster to use inside of the R package can be selected by passing the name of the method as a hyperparameter [4, 1].

SeasonalNaivePredictor

The SeasonalNaivePredictor is a naive model which predicts the future value to be the same as the value of the previous season. This is a very simple model however an example explains its functionality best. If I want to use the SeasonalNaivePredictor to forecast the average temperature of June. The SeasonalNaivePredictor would return the average temperature for last year in June. In GluonTS, if not enough data exists (i.e. no data for last year in June) the mean of all the data in the timeseries is returned [4, 20]

MQCNNEstimator, MQRNNEstimator

The GluonTS seq2seq package contains two forecasting algorithms, MQ-CNN and MQ-RNN. These are based on the MQ framework described in *A Multi-Horizon Quantile Recurrent Forecaster* [48]. The MQ framework is based on the sequence too sequence (Seq2Seq) architecture which consists of an encoder and a decoder network [16]. A Seq2Seq architecture encodes the training data into a hidden state which the decoder network then decompresses. Normally in Seq2Seq architectures, the RNN models tend to accumulate errors as the forecasts of an RNN for a timepoint t will be reused in order to generate a forecast for time $t+1$. By instead training the model to generate multiple point forecast for each timepoint in the horizon one wishes to forecast on, the errors tend to grow smaller. This is called *Direct Multi-Horizon Forecasting* and it is one of the changes introduced as part of the MQ framework. Further, the MQ framework allows for different encoders to used. Two of these are implemented in GluonTS, one with a CNN and one with a RNN. These are known as the the MQCNNEstimator and the MQRNNEstimator respectively [4].

SimpleFeedForwardEstimator

The SimpleFeedForwardEstimator in GluonTS is a traditional Multi Layer Perceptron (MLP) capable of generating probabilistic forecasts. The size of the network can be adjusted based on user parameters. Per default it has two densely connected layers containing 40 nodes in the input layer and 40 times the desired prediction length as the number of cells in the hidden layer.

The network has an additional layer which allows it to generate probability based predictions instead of only point predictions. This layer consists of a number of sublayers equal to the desired prediction length. Each of these layers consists of 40 nodes [3].

TransformerEstimator

The transformer is a Seq2Seq model which replaces the classical RNN encoder and decoders in a Seq2Seq model with more easily parallelizable NN components. A RNN requires data to be passed sequentially in order to learn dependencies between datapoints. This introduces a bottleneck and makes RNNs harder to train efficiently on modern hardware accelerators such as GPUs. The Transformer architecture presented in *Attention is All you Need* [46] alliviates this by leveraging parallelizable architectures such as feed forward networks as well as making heavy use of attention. Attention means that the architecture automatically can identify which parts of the input data is most relevant to the value we are trying to predict[46]. I.e. for the next value we are predicting, the most relevant previous values may be any or all of the previous n observations. In GluonTS the Transformer is implemented under the name TransformerEstimator [4].

2.6.2 Datasets

GluonTS offers 17 datasets ready to be used for training and evaluation, in table 3 an overview of these are presented. Of the available datasets 7 of them were first used in the M3, M4 and the M5 competitions [28, 27, 29]. The remaining datasets has been used in multiple research papers with various amounts of preprocessing applied to them [34, 24, 37, 48, 47, 16].

Name	Freq	Description
Electricity	Hourly	Hourly electricity consumption of 370 clients sampled between 2011 - 2014. The original dataset on UCL was sampled each 15 minutes whilst the dataset in GluonTS had the data resampled into hourly series [4, 39].
Exchange Rate	B	Daily currency exchange rates of eight countries; Australia, Britain, Canada, China, Japan, New Zealand, Singapore and Switzerland between 1990 and 2016 [24].
Traffic	H	This dataset contains the hourly occupancy rate of 963 car lanes of the San Francisco Bay are freeways [3].
Solar Energy	10 Min	The solar power production records in the year of 2006, sampled every 10 minutes from 137 solar energy plants in Alabama [24].
Electricity NIPS	H	The Electricity dataset with additional processing [39].
Exchange Rate NIPS	B	The Exchange Rate dataset with additional processing [39].
Solar Energy NIPS	H	The Solar Energy dataset with additional processing [39].
Traffic NIPS	H	The Traffic dataset with additional processing [39].
Wiki Rolling NIPS	D	The Wiki dataset contains the amount of daily views for 2000 pages on Wikipedia [39].
Taxi	30 Min	Number of taxi rides taken on 1214 locations in New York city every 30 minutes in the month of January 2015. The test set is sampled on January 2016 [39].
M4 Hourly	H	Hourly timeseries used in the M4 competition randomly sampled from the ForeDeCk database [28].
M4 Daily	D	Daily timeseries used in the M4 competition randomly sampled from the ForeDeCk database [28].
M4 Weekly	W	Weekly timeseries used in the M4 competition randomly sampled from the ForeDeCk database [28].
M4 Monthly	M	Monthly timeseries used in the M4 competition randomly sampled from the ForeDeCk database [28].
M4 Quarterly	3M	Quarterly timeseries used in the M4 competition randomly sampled from the ForeDeCk database [28].
M4 Yearly	Y	Yearly timeseries used in the M4 competition randomly sampled from the ForeDeCk database [28].
M5 Dataset	D	Daily Walmart sales for 3049 products across 10 stores [3, 29].

Table 3: Datasets available in GluonTS.

2.7 Runtool

As part of an internship I had at Amazon Web Services (AWS) I created a Python toolkit for creating and executing large scale machine learning experiments. This tool was named *the runtool* and is open sourced as part of the GluonTS sister repository gluon-ts-tools [15]. Originally the runtool was meant to be a core part of this thesis, however the runtool diverged from the thesis and is now represented here as a third party package.

The runtool works in three parts, first assets such as algorithms and datasets are defined in a YAML config file. This file is then loaded by the runtool from within a python script. Thereafter experiments can be generated via mathematical operators such as + and *. Addition groups algorithms or datasets together into sets and multiplication generates an experiment of the cartesian product of the two sets being multiplied. Finally the runtool starts the generated experiments as training jobs in SageMaker. SageMaker is a cloud service enabling machine learning workloads such as model training and inference to be run on dedicated hardware.

Executing a few training jobs is not especially hard via a simple Python script as there are many powerful libraries available such as GluonTS. However scheduling and dispatching hundreds or thousands of different training jobs in parallel can quickly become complex. This is further complicated when multiple algorithms with different hyperparameter configurations should be evaluated on multiple different datasets. The runtool simplifies these things via so called $ operators within the config file. These operators enable, for example, inheritance between algorithms or datasets and dynamic updates of values in the config based on what the current experiment is.

In order to execute training jobs on SageMaker via the runtool there are certain requirements:

- The ML model needs to be in a SageMaker compliant docker container.
- The dataset has to be in an AWS S3 bucket.
- An IAM role granting the runtool access to the dataset and the docker image.

A key benefit of the runtool is its use of config files. Through these, it is possible to rerun any experiment with the exact same configuration as long as one has access to the docker container, the dataset and the config file. Due to that the dependencies are built into the docker image any experiment is fully rerunnable with minimum configuration.

The part of the Runtool which is responsible for starting training jobs is called the jobs dispatcher. While the builtin job dispatcher starts training jobs on SageMaker the runtool is built to make it easy to implement multiple backends. Thus

it is possible to extend the runtool such that training jobs can be executed on a local machine instead of on SageMaker.

In figure 13 a simple config file is displayed which defines two different algorithms and two datasets. This config is then used in the script presented in figure 12 to create four experiments where both algorithm are executed on both datasets.

```
import boto3
import runtool

# load config file
config = runtool.load_config("config.yml")

# create an experiment
my_experiment = (
    config.myalgo + config.anotherAlgo
    ) * (config.electricity + config.traffic)

# initialize runtool
tool = runtool.Client(
    role="arn:aws:iam::012345678901:role/my_role",
    bucket="my_bucket",
    session=boto3.Session(),
)

# dispatch the jobs
tool.run(my_experiment)  # blocking call
```

Figure 12: Python script using the config from figure 13 to create four experiments

```
electricity:
    meta:
        freq: 1H
        prediction_length: 24
    path:
        test: file:///path/to/dataset1/train.json
        train: file:///path/to/dataset1/test.json
traffic:
    meta:
        freq: 1H
        prediction_length: 24
    path:
        test: file:///path/to/dataset2/train.json
        train: file:///path/to/dataset2/test.json

base_algo:
    hyperparameters:
        epochs: 10
    instance: local
    metrics:
        MASE: 'MASE\): (\d+\.\d+)'
        abs_error: 'abs_error\): (\d+\.\d+)'
algo1:
    $from: base_algo
    image: image_with_algo1

algo2:
    $from: base_algo
    image: image_with_algo2
```

Figure 13: Config file describing two algorithms and two datasets. $from inherits the values from another node.

2.8 Hypothesis testing

Distributions occur in many areas within nature and science and being able to validate whether different samples come from the same underlying distribution is a long studied problem. This is known as hypothesis testing and several methods have been developed and one can group these tests into two major families, parametric and non-parametric tests [23]. Parametric tests are suitable whenever the type of the underlying distribution is known, otherwise non-parametric tests are more suitable.

The Student's T-test is a commonly used hypothesis test which compares the mean values of two distributions and only works if the samples are independent, normal and they have equal variance [23]. The requirement that the samples should have equal variance can make the T-test unsuitable in practice. However, an alternative to the Student's T-test is the Welch's T-test which allows for different variance between the two samples being tested. The Welch's test has however been shown to be unreliable if the two samples being tested have a considerable difference in size and variance [38].

A well known non-parametric test is the Diebold-Mariano (DM) two sample test which was designed to determine whether two samples of forecasts were statistically different [14]. Hassani et.al. showed that the DM test is outperformed by another non-parametric test, the Komogorov-Smirnov two sample test [17].

The Kolmogorov-Smirnov (KS) two sample test is a distance test based on the empirical cumulative distribution function (CDF) of the samples [30]. In essence, it tests if the distance between the two CDFs is too large, if they are to far apart, the test fails.

3 Empirical Comparison

Bibliography

[1] forecast package in r, https://pkg.robjhyndman.com/forecast/

[2] Forecasting competition for artificial neural networks and computational intelligence, http://www.neural-forecasting-competition.com/NN5/

[3] GluonTS - github repository, https://github.com/awslabs/gluon-ts

[4] GluonTS - probabilistic time series modeling, https://ts.gluon.ai/

[5] mlbench: Distributed machine learning benchmark, https://mlbench.readthedocs.io/en/latest/

[6] Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., Maddix, D.C., Rangapuram, S., Salinas, D., Schulz, J., Stella, L., Türkmen, A.C., Wang, Y.: GluonTS: Probabilistic time series models in python http://arxiv.org/abs/1906.05264

[7] Baker, M.: 1,500 scientists lift the lid on reproducibility. Nature News 533(7604), 452 (2016)

[8] Bauer, A., Züfle, M., Eismann, S., Grohmann, J., Herbst, N., Kounev, S.: Libra: A benchmark for time series forecasting methods. In: Proceedings of the ACM/SPEC International Conference on Performance Engineering. pp. 189–200 (2021)

[9] Beam, A.L., Manrai, A.K., Ghassemi, M.: Challenges to the reproducibility of machine learning models in health care. Jama 323(4), 305–306 (2020)

[10] Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks 5(2), 157–166 (1994)

[11] Bouthillier, X., Delaunay, P., Bronzi, M., Trofimov, A., Nichyporuk, B., Szeto, J., Mohammadi Sepahvand, N., Raff, E., Madan, K., Voleti, V., et al.: Accounting for variance in machine learning benchmarks. Proceedings of Machine Learning and Systems 3 (2021)

[12] Bühlmann, P., Yu, B.: Analyzing bagging. The annals of Statistics 30(4), 927–961 (2002)

[13] Dau, H.A., Bagnall, A., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Keogh, E.: The ucr time series archive. IEEE/CAA Journal of Automatica Sinica 6(6), 1293–1305 (2019)

[14] Diebold, F.X.: Comparing predictive accuracy, twenty years later: A personal perspective on the use and abuse of diebold–mariano tests. Journal of Business & Economic Statistics 33(1), 1–1 (2015)

[15] Freccero, L.A.: The runtool, https://github.com/awslabs/gluon-ts-tools/tree/main/runtool

[16] Graves, A.: Generating sequences with recurrent neural networks. CoRR abs/1308.0850 (2013), http://arxiv.org/abs/1308.0850

[17] Hassani, H., Silva, E.S.: A kolmogorov-smirnov based test for comparing the predictive accuracy of two sets of forecasts. Econometrics 3(3), 590–609 (2015)

[18] Huang, X., Fox, G.C., Serebryakov, S., Mohan, A., Morkisz, P., Dutta, D.: Benchmarking deep learning for time series: Challenges and directions. In: 2019 IEEE International Conference on Big Data (Big Data). pp. 5679–5682. IEEE, https://ieeexplore.ieee.org/document/9005496/

[19] Hyndman, R.J.: Measuring forecast accuracy p. 9

[20] Hyndman, R.J., Athanasopoulos, G.: Forecasting: principles and practice. OTexts, 3nd edition edn., https://otexts.com/fpp3/

[21] Ioannidis, J.P., Cripps, S., Tanner, M.A.: Forecasting for covid-19 has failed. International Journal of Forecasting (2020), https://www.sciencedirect.com/science/article/pii/S0169207020301199

[22] github user: jsirol: Deepar + negativebinomialoutput performance degradation after upgrading from v0.4.2 to 0.5.0, https://github.com/awslabs/gluon-ts/issues/906

[23] Kim, T.K.: T test as a parametric statistic. Korean journal of anesthesiology 68(6), 540 (2015)

[24] Lai, G., Chang, W.C., Yang, Y., Liu, H.: Modeling long- and short-term temporal patterns with deep neural networks http://arxiv.org/abs/1703.07015

[25] Makridakis, S., Chatfield, C., Hibon, M., Lawrence, M., Mills, T., Ord, K., Simmons, L.F.: The m2-competition: A real-time judgmentally based forecasting study. International Journal of forecasting 9(1), 5–22 (1993)

[26] Makridakis, S., Hibon, M., Lusk, E., Belhadjali, M.: Confidence intervals: An empirical investigation of the series in the m-competition. International Journal of Forecasting 3(3-4), 489–508 (1987)

[27] Makridakis, S., Hibon, M.: The m3-competition: results, conclusions and implications. International Journal of Forecasting 16(4), 451–476 (2000), https://www.sciencedirect.com/science/article/pii/S0169207000000571, the M3- Competition

[28] Makridakis, S., Spiliotis, E., Assimakopoulos, V.: The m4 competition: 100,000 time series and 61 forecasting methods 36(1), 54–74, https://linkinghub.elsevier.com/retrieve/pii/S0169207019301128

[29] Makridakis, S., Spiliotis, E., Assimakopoulos, V., Chen, Z., Gaba, A., Tsetlin, I., Winkler, R.: The m5 uncertainty competition: Results, findings and conclusions (11 2020)

[30] Massey Jr, F.J.: The kolmogorov-smirnov test for goodness of fit. Journal of the American statistical Association 46(253), 68–78 (1951)

[31] Mattson, P., Cheng, C., Coleman, C., Diamos, G., Micikevicius, P., Patterson, D., Tang, H., Wei, G.Y., Bailis, P., Bittorf, V., Brooks, D., Chen, D., Dutta, D., Gupta, U., Hazelwood, K., Hock, A., Huang, X., Ike, A., Jia, B., Kang, D., Kanter, D., Kumar, N., Liao, J., Ma, G., Narayanan, D., Oguntebi, T., Pekhimenko, G., Pentecost, L., Reddi, V.J., Robie, T., John, T.S., Tabaru, T., Wu, C.J., Xu, L., Yamazaki, M., Young, C., Zaharia, M.: MLPerf training benchmark http://arxiv.org/abs/1910.01500

[32] McDermott, M., Wang, S., Marinsek, N., Ranganath, R., Ghassemi, M., Foschini, L.: Reproducibility in machine learning for health. arXiv preprint arXiv:1907.01463 (2019)

[33] Oord, A.v.d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: WaveNet: A generative model for raw audio http://arxiv.org/abs/1609.03499

[34] Oreshkin, B.N., Carpov, D., Chapados, N., Bengio, Y.: N-BEATS: Neural basis expansion analysis for interpretable time series forecasting http://arxiv.org/abs/1905.10437

[35] Pineau, J., Vincent-Lamarre, P., Sinha, K., Larivière, V., Beygelzimer, A., d'Alché Buc, F., Fox, E., Larochelle, H.: Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). arXiv preprint arXiv:2003.12206 (2020)

[36] Pritzsche, U.: Benchmarking of classical and machine-learning algorithms (with special emphasis on bagging and boosting approaches) for time series forecasting, https://pdfs.semanticscholar.org/719f/e0e9823ee42630f1c663c102074d86354f67.pdf

[37] Rangapuram, S.S., Seeger, M.W., Gasthaus, J., Stella, L., Wang, Y., Januschowski, T.: Deep state space models for time series forecasting. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 7785–7794. Curran Associates, Inc., http://papers.nips.cc/paper/8004-deep-state-space-models-for-time-series-forecasting.pdf

[38] Sakai, T.: Two sample t-tests for ir evaluation: Student or welch? In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval. p. 1045–1048. SIGIR '16, Association for Computing Machinery, New York, NY, USA (2016), https://doi.org/10.1145/2911451.2914684

[39] Salinas, D., Bohlke-Schneider, M., Callot, L., Medico, R., Gasthaus, J.: High-dimensional multivariate forecasting with low-rank gaussian copula processes http://arxiv.org/abs/1910.03002

[40] Salinas, D., Flunkert, V., Gasthaus, J.: DeepAR: Probabilistic forecasting with autoregressive recurrent networks http://arxiv.org/abs/1704.04110

[41] Sherstinsky, A.: Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network 404, 132306, http://arxiv.org/abs/1808.03314

[42] Smyl, S.: A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting 36(1), 75–85, https://linkinghub.elsevier.com/retrieve/pii/S0169207019301153

[43] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting 15(1), 1929–1958, publisher: JMLR.org

[44] Taylor, S.J., Letham, B.: Forecasting at scale, https://peerj.com/preprints/3190v2

[45] Vassilieva, N., Serebryakov, S.: Deep learning benchmarking suite, https://hewlettpackard.github.io/dlcookbook-dlbs

[46] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need p. 11, https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[47] Wang, Y., Smola, A., Maddix, D.C., Gasthaus, J., Foster, D., Januschowski, T.: Deep factors for forecasting http://arxiv.org/abs/1905.12417

[48] Wen, R., Torkkola, K., Narayanaswamy, B., Madeka, D.: A multi-horizon quantile recurrent forecaster http://arxiv.org/abs/1711.11053

[49] Willmott, C., Matsuura, K.: Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance 30, 79–82, http://www.int-res.com/abstracts/cr/v30/n1/p79-82/

[50] Zhuang, D., Zhang, X., Song, S.L., Hooker, S.: Randomness in neural network training: Characterizing the impact of tooling. arXiv preprint arXiv:2106.11872 (2021)