Erik Olsvik Dengerud

Global Models for Time Series Forecasting With Applications to Zero-shot Forecasting

Master's thesis in Applied Physics and Mathematics
Supervisor: Erlend Aune
February 2021

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Erik Olsvik Dengerud

Global Models for Time Series Forecasting With Applications to Zero-shot Forecasting

Master’s thesis in Applied Physics and Mathematics
Supervisor: Erlend Aune
February 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Abstract

Global time series models are fitted to a set of time series contrary to local models fitted to individual time series. A global approach can prevent overfitting for deep learning models by using fewer parameters per time series. As the number of time series gets increases, each time series contribute less to the model parameters. This can allow for successful zero-shot forecasting, which is forecasting using a model not fitted to the time series we wish to forecast.

This thesis uses global time series models to forecast the popular M4 dataset from the M4 forecasting competition. The M4 dataset has 100,000 time series and is a popular benchmark for general univariate time series forecasting. We also fit a global model on a larger source dataset from the Federal Reserve of Economic Data (FRED) and use this global model to zero-shot forecast the M4 dataset.

We highlight three of our findings. First, simple multilayer perceptrons (MLP) could have ranked 2nd in the M4 forecasting competition when trained as global models. This contrasts with the local MLP benchmark from the competition that ranked 57th out of 59 entries. Second, simple MLPs are also able to forecast the M4 dataset well in a zero-shot setting. One model per frequency ranks 10th, and one model for the full dataset ranks 14th. This is ahead of all of the statistical benchmarks that are fitted to the M4 dataset. Lastly, using one pretrained model from FRED per frequency, we can retrain only the last two layers to reach similar performance to the full model trained on the M4 dataset. One model for all frequencies struggles with the same task, likely because it is more prone to differences in the source and target dataset distributions.

The success of global models on the M4 dataset, both using the M4 dataset and in a zero-shot setting, is a step toward general global models. Future work can expand the analysis with other model types, larger and more general datasets, and more time series tasks besides forecasting the M4 dataset.

Sammendrag

Globale tidsrekkemodeller er tilpasset til et sett med tidsrekker. Dette er i motsetning til lokale tidsrekkemodeller som er tilpasset individuelle tidsrekker. En global tilnærming kan motvirke overtilpasning siden vi bruker færre parametere per tidsrekke. Når antall tidsrekker øker, vil påvirkningen til hver tidsrekke på modellens parametere bli mindre. Det kan føre til vellykket prediksjon av tidsrekker modellen ikke har blitt tilpasset. Dette kalles blind prediksjon.

Denne oppgaven bruker globale tidsrekkemodeller til å predikere det populære M4 datasettet fra M4 konkurransen. M4 datasettet har mer enn 100,000 tidsrekker og er en populær referanse for generelle, envariate tidsrekkemodeller. Vi tilpasser også globale tidsrekkemodeller på et større datasett fra Federal Reserve of Economic Data (FRED) og bruker denne globale modellen til blind prediksjon av M4 datasettet.

Vi vil trekke frem tre funn. For det første ville et enkelt flerlags perceptron (MLP) kunne kommet på andreplass i M4 konkurransen om den ble tilpasset som en global modell. Dette står i kontrast til den lokale MLP referansemodellen fra konkurransen som var rangert som 57 av 59 modeller. For det andre kan et MLP predikere M4 datasettet bra også ved blind prediksjon. En MLP per frekvens ville vært rangert som nr 10 og en modell for alle frekvensene ville vært nr 14. Dette er foran alle de klassiske statistiske referansemodellene beregnet på M4 datasettet. Til slutt, om vi har en modell per frekvens tilpasset FRED datasettet, så kan vi tilpasse kun de siste to lagene til M4 datasettet og oppnå tilsvarende presisjon som om vi hadde tilpasset hele modellen. Det er vanskelig for en modell som er tilpasset alle frekvensene på FRED datasettet å oppnå liknende resultater. En sannsynlig årsak er at denne modellen er mer utsatt for forskjeller i fordelingen til de to datasettene.

Suksessen til globale tidsrekkemodeller på M4 datasettet, både ved bruk av M4 og ved blind prediksjon, er et steg mot generelle globale tidsrekkemodeller. Videre arbeid kan utvide analysen med andre modelltyper, større og mer generelle datasett, og flere tidsrekkeproblemer i tillegg til prediksjon av M4 datasettet.

Preface

This thesis concludes my M.Sc degree in Applied Physics and Mathematics at NTNU. My years at NTNU have been very enjoyable, with both personal and academic growth. I am also very grateful for the opportunity to study abroad at the University of Colorado, Boulder.

First, I would like to thank my supervisor, Erlend Aune. Thank you for engaging in discussions and the development of ideas while allowing me to pursue a topic I find truly exciting.

Great teachers are not to be taken for granted, and I have been blessed with great mathematics teachers since middle school. Thank you for fueling my interest in mathematics and challenging my fellow students and me.

Most importantly, I want to thank my family for their unconditional support in life, sports, and academics. Thank you also to all my friends that I have met in Trondheim and Boulder. You are what made my time as a student great. A special mention goes to Bjørnulf, Henrik, Lars, Martin, Mikal, the boys on the CU ski team, and my roommates over the years. While our last year did not play out as we envisioned, I am sure we will have many memories to come.

Erik Olsvik Dengerud
Sjusjøen, Feb. 2021

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The amount of time series data is increasing as we measure and store more data. As we model this data, it is not always feasible to use separate models for each time series. We can instead take a more human approach by using the information we learn from other time series. Global time series models are models that learn from a large set of time series. They are fitted to a set of time series instead of individual time series. If a global model is robust enough, we might even use it to forecast incoming time series without using these to fit the model. This is called zero-shot forecasting.

In this thesis, we investigate global time series models and their applications to zero-shot forecasting. Global models can allow for more complex models without the same risk of overfitting as local models. Deep learning models can struggle with time series data because they require lots of data. Global models alleviate some of the challenges by including more time series in the dataset, requiring the model to learn more general characteristics.

The forecasting problem is the most well-known time series task. Forecasting is central for many businesses and their operations. In retail, it is essential to keep the right inventory to maximize profits and limit costs. Players in finance seek to know companies' future revenues, and electricity suppliers must forecast the expected load on the power grid. However, there are also other important time series tasks. Most noteworthy is perhaps time series classification. Cardiologists wish to detect electrocardiograms with heart anomalies (Chuah and Fu, 2007), and neurologists want to analyze brain activity to detect patients prone to Alzheimer's (Szenkovits et al., 2018).

The M4 forecasting competition in 2018 aimed to develop new techniques and methods for univariate forecasting problems focusing on business-relevant time series (Makridakis et al., 2020). After the competition, the M4 dataset has become a useful benchmark for evaluating univariate time series models. Part of the reason is the many precomputed benchmarks. These benchmarks make it easier to evaluate new methods and are the main reason we also use the dataset. Another important reason is that it contains time series from a range of domains and frequencies.

Global Models

Global models are similar to how we humans process data. If we want to forecast a company's quarterly revenue, it makes sense to look at other related companies to better estimate the trend and seasonality for the sector. While we could try to estimate characteristics based on the single company of interest, we are limited by relevant data. Data from the early 2000s is likely not as relevant as the data from the past few years. Our estimates for the trend and seasonality are likely to generalize better if we use our knowledge from all the other companies.

The benchmarks of the M4 competition are local models. Most of the entries to the competition are also local models. However, some of the top contestants utilize the knowledge of other similar time series. While these top contestants both have parts that use deep learning and machine learning, the idea about global models is also valid for classical statistical models.

Deep Learning

Deep learning has revolutionized areas like image analysis and language modeling. However, the M4 forecasting competition (Makridakis et al., 2020) concluded that classical statistical or combinations of machine learning and statistical outperformed pure deep learning approaches. Time series are sequential data, just as in language modeling. Based on the success of deep learning in other sequential data domains, the conclusion is somewhat surprising.

The two top entries in the competition used combinations of statistical and machine learning or deep learning (Smyl, 2020; Montero-Manso et al., 2020). Common for the two top entries is that the deep learning or machine learning part of the solution is learning across many time series. The current state-of-the-art method, N-BEATS, is taking this further by removing the statistical part and training a pure deep learning model for each frequency. By frequency, we refer to the type of time series, e.g., yearly, monthly, and daily. We will refer to models trained per frequency as almost global models. There is a distinction between global and almost global models, and the global modeling task is more challenging because it requires learning to forecast all frequencies using the same parameters.

Recent advances in language and image modeling show the advantage of pretrained models on general datasets for zero- and few-shot tasks (Brown et al., 2020; Chen et al., 2020). Oreshkin et al. (2020) show using N-BEATS trained on FRED and tested on M4 that pretraining can also be useful for time series. Two prominent use cases for pretrained models are zero-shot forecasting and transfer learning.

Thesis Structure

There are two main questions we want to research in this thesis. First, we want to see how well we can forecast the M4 dataset using global models. We have not yet seen successful applications of global models trained across frequencies on the M4 dataset in the literature.

Second, we want to see how additional data can be utilized. There are mainly two ways we can utilize this data. The first is in a zero-shot forecasting setting. Given a time series, we want to forecast it without fitting any parameters. This requires a pretrained model. The second setting is a transfer learning setting where we want to utilize the power of a larger model, but we are limited by available data. Retraining parts of the network will utilize the general characteristics of the time series while allowing the model to adapt to the ta rget dataset.

The remainder of the thesis is structured as follows,

- Chapter 2 introduce relevant background for time series modeling. It is structured into three parts. First, we introduce relevant models used for modeling time series. We divide this part into statistical and deep learning methods. Second, we motivate the use of global models by reviewing some statistical learning theory. Lastly, we perform a small study of N-BEATS, the current best-performing model on the M4 dataset. This is necessary to understand how deep learning methods can be successfully developed.

- Chapter 3 reviews the literature on time series forecasting. We also discuss relevant work from other sequential data domains such as language modeling.

- Chapter 4 discuss time series datasets for global models. We introduce the M4 and FRED datasets that we use in our experiments. In addition, we highlight some challenges using global models. We exemplify how global models can implicitly look at the test set, thus causing look-ahead bias.

- Chapter 5 explains the experimental setup and other details relevant to the experiments.

- Chapter 6 presents and discuss the results from our experiments. There are two parts to the experiments. First, we train and test global and almost global models on the M4 dataset. Second, we train models on the FRED dataset and test them on the M4 dataset without retraining, partly retraining, and full retraining.

- Chapter 7 summarize and conclude the thesis while also giving possible extensions for future work.

Chapter 2

Background for Time Series

This chapter gives an overview of relevant background time series modeling using global models. The aim is to provide the necessary background for the experiments in Chapter 6. We divide this chapter into three parts.

1. The first part is on time series modeling using both statistical and deep learning models. We highlight the relevance of the forecasting problem for time series modeling and introduce the relevant models. We also introduce standard techniques for training deep learning models.

2. Next, we introduce the statistical learning theory that motivates global models over local models. We show how models trained jointly on a more extensive set of time series, global, can have tighter generalization bounds than models fitted to individual time series. This section is the most theoretical of this thesis. We include it as motivation but do not apply any theory from this chapter directly.

3. Last, we perform a small study of N-BEATS (Oreshkin et al., 2019), the current state-of-the-art forecasting method measured by performance on the M4 dataset from the M4 forecasting competition. This study gives us the necessary background to design and train global models.

We include the small case study in the theoretical background chapter. An experimental part of a background chapter is somewhat unconventional. We reason that it is important to understand better the requirements for making deep learning methods work well in practice. The study will make the subsequent experiments easier to follow.

2.1 Time Series Modeling

This section focuses on time series modeling using deep learning methods. The models used in Chapter 6 are introduced in this section. We structure the section as follows.

- We first motivate the forecasting problem as a way to model time series. The likelihoods of the time series and the conditional time series, forecasts, are the same, and optimizing one is equivalent to optimizing the other.
- Next, we introduce a classical statistical approach, the SARIMA model.
- Last, we introduce deep learning models. This includes practices and techniques required to train deep learning models successfully.

For theory related to deep learning, we follow the book by Goodfellow et al. (2016). Theory related to statistical modeling of time series is from Brockwell et al. (1991).

2.1.1 Relevance of the Forecasting Problem

A time series is a set of random variables $\{\mathbf{X}_t\}$, where t is the time of observation. The observed time series is usually denoted using small letters, $\{\mathbf{x}_t\}$. We will only treat univariate time series and therefore write $\mathbf{X} = \{X_t\}$ for a time series and $\mathbf{x} = \{x_t\}$ for its observations. Our goal is to model the distribution of the time series,

$$\{X_t\} \sim p(\mathbf{X}) = p(X_1, X_2, \dots X_T), \tag{2.1}$$

where T is the length of the time series. By the chain rule of probability we have that for two random events A and B,

$$P(A \cup B) = P(B|A) \cdot P(A). \tag{2.2}$$

Applied recursively on the time series we get that the distribution can be written as a product of all the conditional distributions,

$$p(\mathbf{X}) = \prod_{t=1}^{T} p(X_t | X_{t-1}, X_{t-2}, \dots, X_1). \tag{2.3}$$

Equation 2.3 illustrated the relevance of the forecasting problem for time series modeling.

Equivalence of Likelihoods

We wish to approximate the distribution of the time series using a parameterized model. This can in our case either be a statistical model or a deep learning model. We write,

$$\mathbf{X} \sim p(\mathbf{X}|\boldsymbol{\theta}), \tag{2.4}$$

where $\boldsymbol{\theta}$ are the parameters of the model. We can define the likelihood function,

$$\mathcal{L}(\boldsymbol{\theta}|\mathbf{x}) = p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{t=1}^{T} p(x_t | x_{t-1}, \dots, x_1, \boldsymbol{\theta}), \tag{2.5}$$

and maximize this to find optimal parameters for our model. The logarithm is a monotonic function and we therefore define the negative log-likelihood which often is more tractable to minimize,

$$-\,l(\boldsymbol{\theta}|\mathbf{x}) = -\log \mathcal{L}(\boldsymbol{\theta}|\mathbf{x}) = -\log p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{t=1}^{T} -\log p(x_t|x_{t-1},\dots x_1,\boldsymbol{\theta}). \qquad (2.6)$$

The key point is that minimizing the negative log likelihood of the time series is the same as minimizing the sum of the conditional negative log likelihoods of the time series.

For global models we assume a set of observed time series $\mathcal{X} = \{\mathbf{x}^{(1)},\dots,\mathbf{x}^{(N)}\}$. The likelihood of the global model is,

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{X}) = \prod_{n=1}^{N} p(\mathbf{x}^{(n)}|\boldsymbol{\theta}) = \prod_{n=1}^{N}\prod_{t=1}^{T^{(n)}} p(x_t^{(n)}|x_{t-1}^{(n)},\dots,x_1^{(n)},\boldsymbol{\theta}), \qquad (2.7)$$

with negative log likelihood,

$$-l(\boldsymbol{\theta}|\mathcal{X}) = -\log \mathcal{L}(\boldsymbol{\theta}|\mathcal{X}) = -\log \prod_{n=1}^{N} p(\mathbf{x}^{(n)}|\boldsymbol{\theta}) \qquad (2.8)$$

$$= -\log \prod_{n=1}^{N}\prod_{t=1}^{T^{(n)}} p(x_t^{(n)}|x_{t-1}^{(n)},\dots,x_1^{(n)},\boldsymbol{\theta}) \qquad (2.9)$$

$$= \sum_{n=1}^{N}\sum_{t=1}^{T^{(n)}} -\log p(x_t^{(n)}|x_{t-1}^{(n)},\dots x_1^{(n)},\boldsymbol{\theta}). \qquad (2.10)$$

That is, optimizing the sum of the sum of the conditional likelihoods of each time series is equivalent to optimizing the likelihood of the full set. This shows the relevance of the forecasting problem to time series modeling.

2.1.2 Forecasting Using Statistical Models

By assuming that a time series comes from a statistical model, we can get nice properties and analyze and explain the time series. However, the assumptions we make are not necessarily true. Another characteristic of statistical models is that they require handcrafting and are less suited for automatic modeling of lots of time series. This section gives a short overview of some central concepts and models that will build towards a Seasonal-Auto-Regressive-Integrated-Moving-Average (SARIMA) model. The SARIMA model is a valid benchmark and has already been computed as one of the statistical benchmarks for the M4 competition. It is popular in many applications and made suitable for automatic time series modeling through the `auto.arima` function (Hyndman and Khandakar, 2008) in the **R** programming language.

We start out by defining what it means for a time series to be stationary. This is a property central to many models. It says that the distribution of equal lengths of the time series, but possibly at different starting indexes, are equal. When modeling a time series, transformations are used so that the transformed series is stationary.

Definition 2.1.1. *Stationary (Brockwell and Davis, 2016)*
$\{X_t\}$ is a strictly stationary time series if

$$(X_1, \ldots, X_n) \overset{d}{=} (X_{1+h}, \ldots, X_{n+h}), \tag{2.11}$$

for all integers h and $n \geq 1$.

Trend and Seasonality

Time series sometimes move in one direction for a longer amount of time. This is known as a trend and something that will defy the stationary property. We commonly transform the series by computing the differentiated series by one lag to work with change in growth. This is also known as integrating and the source of the integrated part of the SARIMA model. Further transformations can also be necessary to get a stationary time series.

Seasonality might still be present in the series and is the changes within a season or period. In time series with a clear cyclic nature like monthly or hourly data, the seasonality is often strong. It has to be removed to yield a stationary time series. We often use the same techniques as for the trend, but now with a lag equal to the seasonality. That is, for a monthly time series, we will differentiate with lag 12. The resulting time series after treating seasonality and trend can then be modeled using an appropriate model.

Autoregressive and Moving Average Processes

We assume that the signal in the time series can be expressed as a function of its own lags. This property is called autoregressive (AR). A time series $\{X_t\}$ is an $AR(p)$ process if it is stationary and,

$$\phi(B)X = Z_t, \tag{2.12}$$

for all t. Here, $Z_t \sim N(0, \sigma^2)$ and $\phi(B) = 1 - \phi_1 B - \cdots - \phi_p B^p$. B is the backshift operator such that $BX_t = X_{t-1}$. This means that the residuals after writing the time series as a function of its own lags is white noise.

Closely related are moving average (MA) processes. For MA processes the time series can be expressed as a function of lags of its own error terms. A time series is a an $MA(q)$ process if it is stationary and,

$$X_t = \theta(B)Z_t, \tag{2.13}$$

for all t. Here, $\theta(B) = 1 + \theta_1 B + \cdots + \theta_q B^q$.

SARIMA

The trend, seasonality and lag dependence can be expressed in the SARIMA model. A time series $\{X_t\}$ is a SARIMA$(p, d, q) \times (P, D, Q)$ process if it is stationary and for every t satisfies,

$$\phi(B)\Phi(B^s)(1 - B)^d(1 - B^s)^D(X_t) = \theta(B)\Theta(B^s)Z_t. \tag{2.14}$$

The time series is differenced by the transformation $(1 - B)^d$ and seasonally differenced by $(1 - B^s)^D$. Here, s is the seasonality, d is the non-seasonal order of difference, and

D the seasonal order of difference. $\Phi(B^s)$ and $\Theta(B^s)$ are the seasonal autoregressive and moving average terms of order P and Q.

The model can be extended to exogenous variables by writing it as,

$$\phi(B)\Phi(B^s)(1-B)^d(1-B^s)^D(X_t - \beta Y_t) = \theta(B)\Theta(B^s)Z_t, \tag{2.15}$$

where Y_t is the exogenous variable.

Naive Forecasters

Naive forecasters are simple models that give the last observation or last period as a forecast. We introduce naive forecasters because they are part of the scoring method in the M4 competition. All models are scaled according to how much better they forecast than a naive forecaster. The naive forecaster used is a Naive2 forecaster. This is like a simple naive forecaster, except it can be seasonally adjusted by applying a multiplicative decomposition. An autocorrelation test is used to decide whether to apply the seasonal adjustments(Makridakis et al., 2020).

2.1.3 Forecasting Using Neural Networks

Deep learning methods show state-of-the-art performance in many domains like image analysis and natural language processing. With N-BEATS (Oreshkin et al., 2019), the authors show that deep learning methods can outperform other methods in forecasting. The M4 competition organizers concluded that hybrid methods likely were the way forward (Makridakis et al., 2020). However, it now seems that pure deep learning methods are promising, given they are global or almost global.

One main argument for using deep learning models is to learn complex relations in the data. A factor for success is the availability of relevant data, as fewer assumptions about the model require more data to fit a good model. For time series, this poses a challenge as many time series are short. The risk of overfitting is high in such cases, and we turn to global models.

We differ between multi-step and one-step models. Multi-step models are trained to give a forecast of a given length. Such models require a separate model for each forecasting horizon. One-step models are trained to only forecast the next value. We can use a one-step model for all forecasting horizons by reapplying it to the already forecasted values.

This section gives an overview of how common deep learning architectures can be applied to time series. The overview includes a brief introduction to feedforward networks, recurrent networks, and convolutional networks. Both the optimization and regularizing of deep learning models are crucial to mitigate overfitting and train models that generalize well. We introduce some techniques that we use in our later experiments. Our theory about deep learning mostly follows Goodfellow et al. (2016) but includes recent research for some techniques.

Feedforward Networks

The most basic neural network architecture is a feedforward network. It has an input layer, hidden layers, and an output layer. Each layer has a number of neurons that take in weights

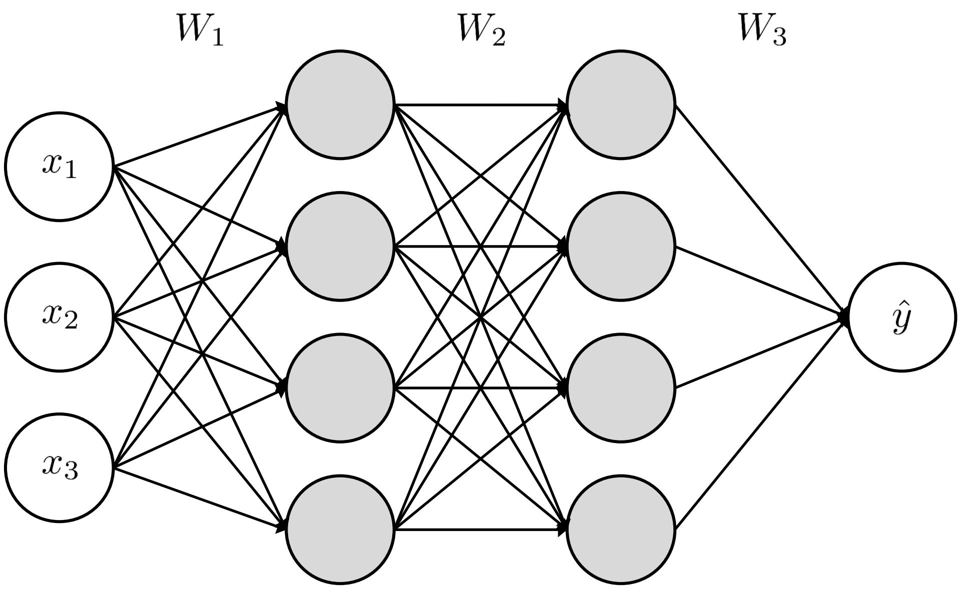


Figure 2.1: A simple fully connected feed forward network. The network has two hidden layers with 4 neurons.

from the previous layer's neurons together with a bias term and computes a function of these. We also refer to this as hidden units. When all neurons in one layer are connected to all neurons in the consecutive layer, we call it a fully connected network. Such a network is displayed in Figure 2.1. Nonlinear functions like $\max(0, x)$ are applied in the neurons to model more complex distributions. The output of the network is measured against a target. Through optimization of the weights in the network, we seek to decrease a loss function. More formally, we can write the output of the network as,

$$\hat{y} = \hat{f}(x) = g(b_n + (\mathbf{w}_n^T \cdot h(b_{n-1} + \mathbf{w}_{n-1}^T \cdot h(\cdots h(b_1 + \mathbf{w}_1^T \cdot \mathbf{x}) \cdots)))), \tag{2.16}$$

where h are the activation functions in the neurons, g is the output activation function and $\mathbf{w}_i$ and b_i are the weights and biases of layer i.

We are using the terms feedforward network and MLP somewhat interchangeably in this thesis. MLPs are a subset of feedforward networks and are often characterized by having fully connected layers of the same size throughout the network. The N-BEATS model introduced later is an example of a feedforward network that is not an MLP.

Optimization is done by propagating the gradient of the loss function back through the network. Stochastic Gradient Descent (SGD) and Adam (Kingma and Ba, 2014) are two widely used algorithms used to optimize neural networks. The development of automatic differentiation frameworks like PyTorch (Paszke et al., 2019) and Tensorflow (Abadi et al., 2015) has facilitated the growth in deep learning by providing high-level abstractions, making it easier to build deep learning models.

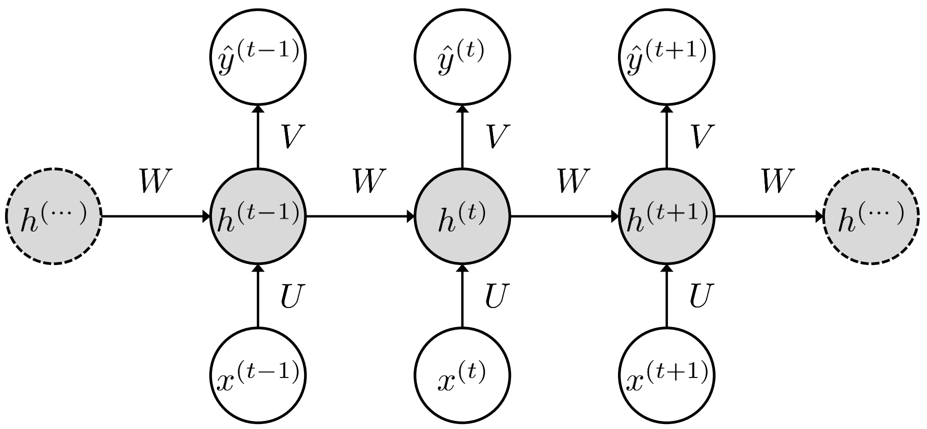


Figure 2.2: A recurrent neural network with one hidden layer (Goodfellow et al., 2016, p. 373).

Recurrent networks

Recurrent neural networks (RNN) are created to model sequences. The idea in recurrent architectures is to keep a hidden state that represents the memory of the network. Input, together with the hidden state, is used to calculate the output. We can write the network as,

$$\hat{y} = \hat{f}(\mathbf{h}_t, x_t; \boldsymbol{\theta}), \tag{2.17}$$

where h_t is the hidden state, x_t the input at time t, and θ the model parameters. Specific to recurrent networks is that the model shares parameters across every time step. Sharing allows for fewer parameters and possibly better generalization. A simple RNN like the one in Figure 2.2 shows how at each time step, the model takes the previous hidden state together with some input and updates the internal state, and produces an output. The network thus has a memory of all past time points.

The parameters in the network are the matrices $\mathbf{U}$, $\mathbf{W}$, and $\mathbf{V}$. These matrices connect the input to the hidden cell, the hidden cell to the output, and the hidden cell to the next hidden cell. The matrices are shared for all time steps, and activation functions are applied in the hidden cells and the output.

A desirable property of recurrent architectures is the infinite memory kept in the internal state. However, in practice, the network struggles to remember long into the past (Bai et al., 2018). Other architectures such as Long Short Term Memory (LSTM) networks and Gated Recurrent Units (GRU) have been proposed to mitigate this problem. Still, the practical memory can be far from the long-range dependencies we sometimes wish to model. Part of the challenge is due to vanishing and exploding gradients. As the gradient is computed for time steps farther away, the gradients might vanish or explode depending on the recurrence relation's eigenvalues (Goodfellow et al., 2016). Eigenvalues of $\mathbf{W}$ less than one will eventually vanish, and eigenvalues larger than one will explode. Small eigenvalues can make it challenging to capture long-term dependencies and training challenging.

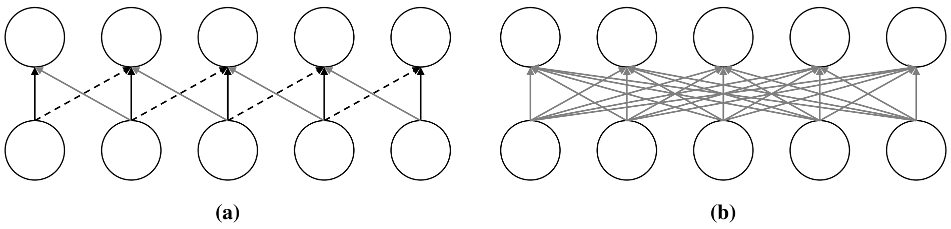


Figure 2.3: A convolutional layer, Figure 2.3a, as opposed to a fully connected layer, Figure 2.3b, with kernel size 3 (Goodfellow et al., 2016, p. 333). Arrow types in Figure 2.3a represent parameter sharing in the convolutional layer.

Convolutional Networks

Convolutional networks have been the backbone of image processing since winning the Imagenet competition in 2012 (Krizhevsky et al., 2012). Convolutional Neural Networks (CNNs) are used to model data on a grid, typically images on a 2D grid (3D counting the RGB channels). For univariate time series, we apply 1D convolutions. One main advantage for CNNs is that we can parallelize operations. Low computational complexity is an advantage as the size of the dataset grows. They are often easier to train than RNNs but lack the theoretical infinite memory. The length of the input window, therefore, determines the memory of the model.

A convolution is an operation on two functions x and w,

$$s(t) = (x \star w)(t), \tag{2.18}$$

where s is the feature map of the convolution, x is the input, and w is the kernel. In machine learning, x is a vector, and w is a set of trainable parameters. One convolutional layer consists of moving the kernel across the input vector, meaning that it has shared parameters. The distinction between a fully connected layer in a feedforward network and a convolutional layer is shown in Figure 2.3. A convolutional layer often uses several kernels that each produce a channel for the next layer by traversing the input. Multiple layers are stacked to get a deep network. In image analysis, kernels learn different features at different depths of the network. Shallow layers learn to detect low-level features like edges, and deeper layers often capture more dataset-specific properties like shapes. The shallow features can be useful across datasets, which is a reason for the success of transfer learning in image analysis (Goodfellow et al., 2016).

2.1.4 Training Deep Networks

Neural networks often require tuning of the training procedure to produce good results. The tuning can be network design, regularizing, and optimization-specific choices. This section is about training deep networks and the techniques we employ to increase model performance.

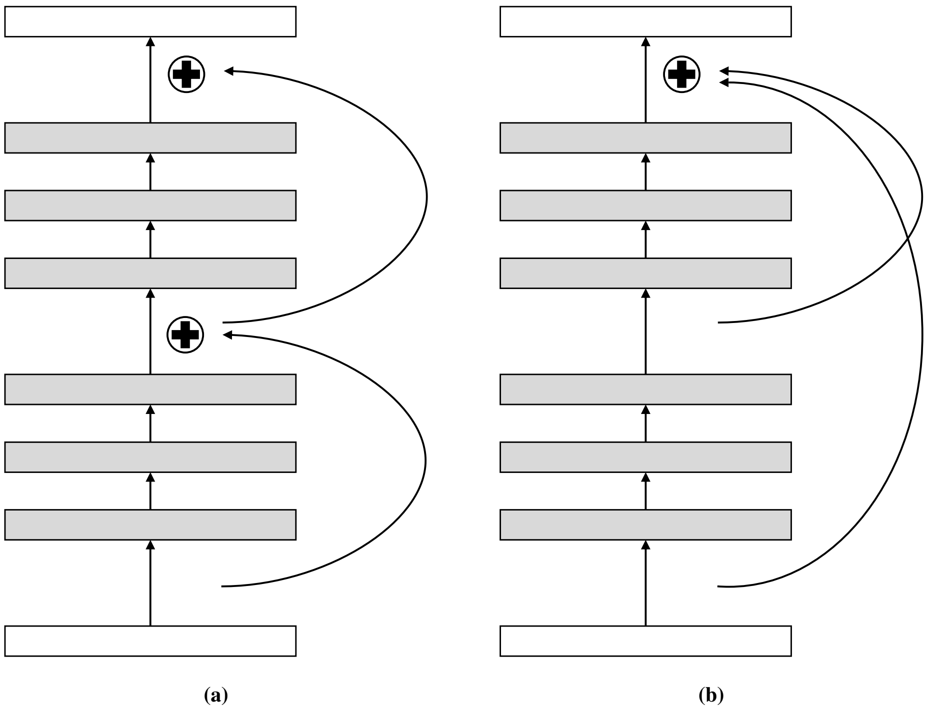


Figure 2.4: A network with 6 hidden layers showing residual and skip connections. Figure 2.4a show residual connections after every third layer. Figure 2.4b show skip connections to the output layer after every third hidden layer.

Residual and Skip Connections

It can be hard for the gradient to pass back through the network without vanishing. A common technique to help gradient flow is to add shortcuts in the network with residual or skip connections. Figure 2.4a shows a residual connection. The network has the option to use an identity mapping or go through the layers in the residual block. Figure 2.4b shows a skip connection. Skip connections are shortcuts from a hidden layer to the output. While residual connections allow the network to decide how important parts of the network are, skip connections encourage separate modeling of characteristics in different layers. Residual and skip connections have been used to train deep architectures for image classification (He et al., 2016).

Regularizing

As the network gets deeper, it also has more expressive power. Model complexity increases the risk of overfitting and, therefore, also the need for regularization. We will discuss dropout and stochastic weight averaging. The goal is that the model will generalize better despite the higher training error.

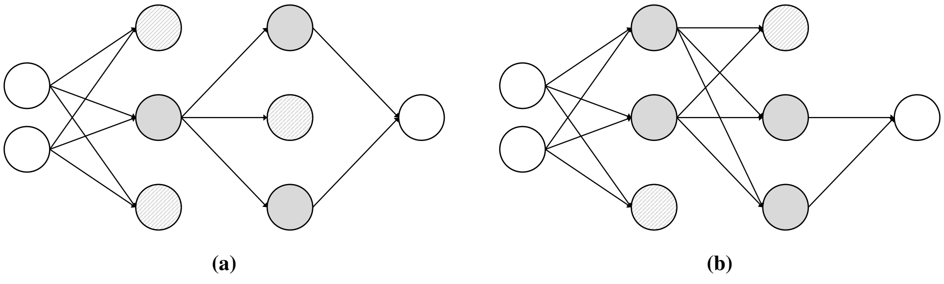

(a) (b)

Figure 2.5: Two subnetworks as a result of dropout. The outputs from the zeroed units does not affect the final output.

Perhaps the most popular regularization technique for neural networks is dropout (Hinton et al., 2012). At training time, each hidden unit is set to zero with a given probability. This enforces the network to find several paths through the network because some paths will be blocked. At test time, all hidden units are active. If we look at the different paths in the network as variants of the model, we can say that there is an ensemble effect at test time. Therefore, dropout often makes the network generalize better. Typical ranges of the probability of zeroing a hidden unit are in the range of 0.2 to 0.5. Figure 2.5 shows dropout in practice for two batches at training time. The zeroed units are different, but at test time, all will be active.

We can create an ensemble effect using stochastic weight averaging (SWA) (Izmailov et al., 2018). Instead of using an ensemble in model space, meaning several models from different random initializations, we create an ensemble in weight space. There are two main advantages of this. First, we can create an ensemble from one training run instead of training separate models with different initializations. Second, at test time, we have one model instead of an ensemble. This takes the test complexity from $\mathcal{O}(n)$ to $\mathcal{O}(1)$. SWA works by running stochastic gradient descent for a number of epochs after convergence. By reducing and then increasing the learning rate in this period, we will hit different parts of the area around the minima. Averaging the models' weights from the epochs with the low learning rate, we get an average of models in weight space. To see how this is different, we write the two types of ensembles. Let $f(\mathbf{w})$ be a neural network with weights $\mathbf{w}$. First, in model space, the ensemble is,

$$\bar{f} = \sum_{i=1}^{n} f(\mathbf{w}_i). \tag{2.19}$$

The SWA model in weight space is,

$$f_{\text{swa}} = f\left(\sum_{i=1}^{n} \mathbf{w}_i\right). \tag{2.20}$$

A full analysis of the method is given by Izmailov et al. (2018). The main idea is that SGD converges to minimas close to steep parts of the loss surface. SWA takes the average of models that are close to the steep parts of the loss surface. This results in a model that lies in the flat parts of this surface. Experiments show that models in flat parts of the region with low loss generalize better.

Learning Rates

The learning rate is also known as the step size from optimization. Tuning the learning rate can, in many cases, be crucial to finding good minimas. Adaptive learning rate schedules are important to explore the parameter space before converging to good minimas. We discuss two learning rate schedules, cosine annealing and reduce on a plateau.

Cosine annealing (Loshchilov and Hutter, 2016) decays the learning rate following a cosine function. It is common to reduce to a tenth or so of the original learning rate. The learning rate η at iteration i is specified by,

$$\eta_i = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{T_i}{T_{\max}}\pi\right)\right). \tag{2.21}$$

We can also reduce the learning rate based on the validation loss. This is called ”reduce on a plateau,” and we reduce the learning rate by a factor if the validation loss has not decreased in a set number of epochs. We continue to reduce the learning rate until we reach a predefined minimum value.

2.1.5 Transfer Learning

Transfer learning aims to utilize information acquired in one task to solve another related task better. Zero-shot forecasting is a relevant use case for pretrained global models. Transfer learning is likely just as relevant but more so in cases where we have some data and the time to train a model. Utilizing the time series features that a global pretrained model has learned across time series is a good starting point for other datasets.

There are mainly two ways pretrained models are used for transfer learning. First, they can provide a good initialization for networks and act as a regularizer keeping the model in a favorable part of the parameter space. The inner workings of how the initialization helps and which characteristics are reused are not very well understood (Goodfellow et al., 2016).

Second, we can use part of the pretrained model as a basis for a new model on a new dataset. The idea is that some features are common across datasets and tasks. In image analysis, these can be edges, shadows, and simple shapes. Utilizing these features makes it easier for a smaller network on another dataset to learn its task as it does not have to learn how to extract these features. Time series also have features like trend and seasonality that are often used for decomposition. Robust estimation of trend and seasonality is likely useful for time series models. However, some datasets might have limited data to estimate such features. Using related datasets to learn to model these features makes sense in these settings.

Data-scarce settings are likely where transfer learning and pretraining in general will benefit most. Goodfellow et al. (2016) argue that pretraining and transfer learning is most useful in settings where the initial representation is poor. This is part of the reason for the success of word embeddings. Word embeddings are much more expressive than one-hot encoded words. Most language modeling tasks utilize pretrained word embedding trained on large datasets.

There are both univariate and multivariate time series data. We are restricting ourselves to univariate data in this thesis. Multivariate time series data likely have towards a richer representation as covariates increase the representation space. Univariate time series data are less rich in comparison.

2.2 Global and Local Models

In this chapter, we give an introduction to learning theory to better understand global and local models. The chapter is structured as follows.

- We introduce concepts in statistical machine learning that we use to derive two key properties of global and local models.
- First, we show that global and local models can produce the same forecasts. That is, the range of global and local models is the same.
- Second, we derive bounds for the generalization error for local and global models. These bounds motivate the use of global models for automatic time series forecasting of groups of time series.

This chapter's results are from Montero-Manso and Hyndman (2020), but the results are standard theoretical machine learning bounds. We use Abu-Mostafa et al. (2012) for the learning theory parts of the chapter. It is worth noting that not all of the theory can be applied directly to our setting of using neural networks to approximate real-valued functions. For that, we require a more technical treatment of the complexity of functions. We will highlight where this is needed and point to relevant sources. However, the theory gives context to why global models are useful for time series modeling.

2.2.1 The Learning Problem

When analytic solutions are not available to a problem, we can approximate a solution by learning from data. Learning theory aims to formalize this learning process. We start by defining some key concepts that we build the theory around. The full learning problem is visualized in Figure 2.6, and we describe this setup in the remainder of this section.

We start with some data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_N, \mathbf{y}_N)\}$ coming from some underlying target function or process. For many of the theorems, we require that the data are independent and identically distributed (i.i.d). However, time series data are often not, and we discuss this in Section 2.2.5. To model the data, we first determine a class of functions that can describe the target function. This is called the hypothesis class, and is best illustrated by Example 1 and 2.

Definition 2.2.1 (Hypothesis class)**.** *A hypothesis class $\mathcal{H}$ is a set of functions. The class often consists of a type of functions or a set of types of functions.*

Example 1 (Linear models of one variable)**.** *A hypothesis class can be the class of linear models of one variable. We write this class as,*

$$\mathcal{H} = \{h = ax + b | a, b \in \mathbb{R}\}. \tag{2.22}$$

Example 2 (Feed forward neural networks)**.** *Another hypothesis class is the class of feed forward neural networks with 5 layers and 32 hidden units with ReLU activation functions. If $\boldsymbol{\theta}$ are the parameters of the feed forward network f, we can write the class* *as,*

$$\mathcal{H} = \{h = f(\boldsymbol{\theta}) | \boldsymbol{\theta} \in \mathbb{R}^n\}, \tag{2.23}$$

where n is the number of parameters of the network we specified.

An important quantity is the complexity of a hypothesis class. It can be defined as the number of different functions that can be fitted from the class.

Definition 2.2.2 (Complexity of a hypothesis class)**.** *The complexity, $|\mathcal{H}|$, of a hypothesis class $\mathcal{H}$, is the number of different functions that can be fitted from the class.*

However, in our two examples of hypothesis classes, the number of functions we can fit is infinite. More specifically, we are dealing with a large infinity as $\mathbb{R}$ is uncountable infinite. Had we restricted our a and b in Example 1 to take values in the set $\{1, 2, 3\}$ instead of $\mathbb{R}$, the complexity would have been nine, which is far from infinite.

The analysis we will do is a lot more technical once the hypothesis set gets infinite. It requires bounding the effective size of the hypothesis set by a growth function. The growth function will then be determined by the Pseudo dimension for real-valued functions or the more known Vapernik-Chervonenkis dimension for binary functions. Most of the interesting hypothesis classes we use are infinite. However, the analysis we do in the rest of the chapter compares global to local models. The ideas will translate to infinite hypothesis classes even if we restrict ourselves to finite classes. An introduction to VC and Pseudo dimensions together with the formalized bounds in terms of the size of the effective hypothesis class is given by Vidyasagar (2003). There are also other ways to measure complexity, like the Rademacher complexity, which works directly on real-valued functions, but which is distribution dependent. An introduction to Rademacher complexity is given by Shalev-Shwartz and Ben-David (2014).

Lastly, a learning algorithm chooses a hypothesis $h \in \mathcal{H}$ based on the data $\mathcal{D}$. The algorithm will choose the approximating function g. A learning algorithm can be back-propagation or quadratic programming and is chosen to work with the hypothesis class. Together they make the learning model.

Definition 2.2.3 (Learning algorithm)**.** *A learning algorithm is an algorithm that chooses a hypothesis from the hypothesis class $\mathcal{H}$ based on the data $\mathcal{D}$.*

2.2.2 Formalizing Global and Local Models

Global and local models can be formalized in the learning problem framework. This formalization is necessary to arrive at the bounds later in this chapter. In this subsection, we apply the learning problem framework to local and global models.

The models we approximate with our learning algorithms are forecasting functions. We define the functions to be autoregressive, meaning they only depend on past values.

Definition 2.2.4 (Forecasting function)**.** *A forecasting function is a real-valued function from the past of a time series to its future. We write,*

$$f : \mathbb{R}^T \to \mathbb{R}^H, \tag{2.24}$$

where T is the length of the input and H is the length of the forecast.

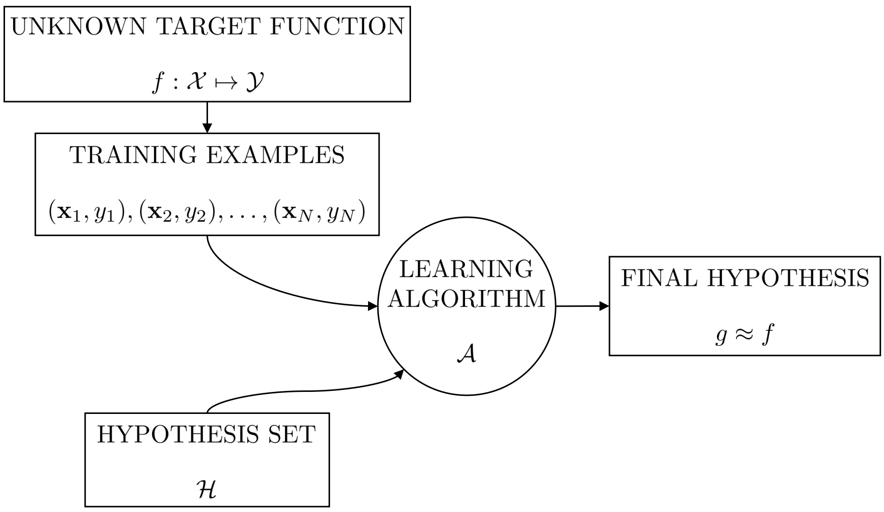


Figure 2.6: The basic learning algorithm (Abu-Mostafa et al., 2012). This is a somewhat simplified setup because we do not include probability. Probability can be included by considering an unknown target distribution $p(y|\mathbf{x})$ and an unknown input distribution $p(\mathbf{x})$. We limit ourselves to the basic setup for simplicity.

Forecasting functions can be local or global functions. They differ in that local models fit a separate function to each time series while global models fit a single function to all the time series. Let $\mathbb{S}$ be a set of univariate time series. We write $\mathbb{S} = \{S_1, \ldots, S_K\}$. A local model is defined by Definition 2.2.5. It is a learning algorithm that takes a time series in $\mathbb{R}^T$ and finds a function g.

Definition 2.2.5 (Local model). *A local time series model is a parameterized model where the parameters are fitted using a single time series. A local model is a learning algorithm s.t.*

$$A_L : \mathbb{R}^T \to g, \tag{2.25}$$

where g is a forecasting function, $g : \mathbb{R}^T \to \mathbb{R}^H$.

A global model is defined by Definition 2.2.6. Contrary to the local model, it takes the full set of univariate time series and finds one global function g.

Definition 2.2.6 (Global model). *A global time series model is a parameterized model where the parameters are fitted using a set of time series. A global model is a learning algorithm s.t.*

$$A_G : \mathbb{S} \to g, \tag{2.26}$$

where $\mathbb{S}$ is a set of univariate time series, and g is a forecasting function, $g : \mathbb{R}^T \to \mathbb{R}^H$.

An error metric or measure is used between the learning algorithm and the final hypothesis in Figure 2.6. A metric is required for the algorithm to find a hypothesis that approximates the unknown target function well. A critical question is how well the model will generalize outside of the training examples. We define the in sample and out of sample error that we will bound.

Definition 2.2.7 (In sample error)**.** *The in sample error is the value of some error metric on the in sample data. Let $\mathcal{D}_{in}$ be the in sample data, $\hat{\mathcal{D}}_{in}$ some in sample prediction and $d(\cdot)$ the error metric. We write the in sample error as,*

$$E_{in} = d(\mathcal{D}_{in}, \hat{\mathcal{D}}_{in}). \tag{2.27}$$

Specifically, for a hypothesis $h \in \mathcal{H}$ and data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_N, \mathbf{y}_N)\}$, it is given by,

$$E_{in}(h) = \frac{1}{N} \sum_{i=1}^{N} d(h(\mathbf{x}_i), f(\mathbf{x}_i)), \tag{2.28}$$

with f being the unknown target function s.t. $f(\mathbf{x}_i) = \mathbf{y}_i$, and $d(\cdot, \cdot)$ being an error metric.

Definition 2.2.8 (Out of sample error)**.** *The out of sample error is the value of some error metric on the out of sample data. Let $\mathcal{D}_{out}$ be the out of sample data, $\hat{\mathcal{D}}_{out}$ some out of sample prediction and $d(\cdot)$ the error metric. We write the out of sample error as,*

$$E_{out} = d(\mathcal{D}_{out}, \hat{\mathcal{D}}_{out}). \tag{2.29}$$

However, we do not know the out of sample data and therefore define it as the expected error for new data given a hypothesis $h \in \mathcal{H}$,

$$\begin{aligned} E_{out}(h) &= \int_{\mathcal{X} \times \mathcal{Y}} d(h(\mathbf{x}), \mathbf{y}) p(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} && (2.30) \\ &= \mathbb{E}[d(h(\mathbf{x}), \mathbf{y})], && (2.31) \end{aligned}$$

where $p(\mathbf{x}, \mathbf{y})$ is the joint probability function of $\mathbf{x}$ and $\mathbf{y}$, and $f(\mathbf{x}) = \mathbf{y}$.

Typical error metrics are mean squared error, mean absolute scaled error, and symmetric mean absolute percentage error. The predictions $\hat{\mathcal{D}}$ come from the model f found by the learning algorithm.

Two Useful Theorems

We proceed with two useful theorems that we use for the lower bounds on the generalization error later in the chapter.

Theorem 2.2.1 (The union bound)**.** *Let $X_1, X_2, \dots$ be a set of events, then*

$$\mathbb{P}\Big(\bigcup_i X_i\Big) \leq \sum_i \mathbb{P}(X_i). \tag{2.32}$$

Proof. We refer to a proof by Casella and Berger (2002) but omit the actual proof as it does not add much to our analysis. □

Theorem 2.2.2 (Hoeffding's inequality)**.** *Let $X_1, \dots, X_N$ be independent random variables bounded on $[0,1]$. Then for any $\varepsilon > 0$,*

$$\mathbb{P}(|\bar{X} - E(\bar{X})| > \varepsilon) \leq 2e^{-2\varepsilon^2 N}, \tag{2.33}$$

with $\bar{X} = \frac{1}{N}\sum_{i=1}^{n} X_i$.

Proof. We refer to the proof by Hoeffding (1994). □

Montero-Manso and Hyndman (2020) discuss how the independence assumption in Hoeffding's inequality is violated by the dependencies in time series. Further, they argue that using an effective sample size N of a time series rather than the length T is a valid solution. The effective sample size is something that grows with T, but in general, it is less. The argument is that this is valid, or else it would not be feasible to learn from time series. While this may seem like an abstraction that magically fix this issue, we can also argue that it will be the same for global and local models and, therefore, not critical to the analysis.

2.2.3 Equivalence of Forecasts

This first result shows that local and global models can produce the same forecasts. This is the first main result from Montero-Manso and Hyndman (2020). In practice, it shows that neither global nor local models are better in terms of the forecasts they can produce. There will always be a global model that can produce the same forecasts that a set of local models can and vice versa.

Proposition 2.2.1 (Equivalence of local and global algorithms for time series forecasting)**.** *Let*

- *$\mathbb{A}_L$ be the set of all local learning algorithms,*
- *$\mathbb{A}_G$ be the set of all global learning algorithms,*
- *$\mathbb{F}_{L,\mathbb{S}} = \{[(A_L(X_i))(X_i)|X_i \in \mathbb{S}], A_L \in \mathbb{A}_L\}$ be the set of all possible local forecasts of $\mathbb{S}$,*
- *$\mathbb{F}_{G,\mathbb{S}} = \{[(A_G(X_i))(X_i)|X_i \in \mathbb{S}], A_G \in \mathbb{A}_G\}$ be the set of all possible local forecasts of $\mathbb{S}$.*

Then,

$$\mathbb{F}_{L,\mathbb{S}} = \mathbb{F}_{G,\mathbb{S}}. \tag{2.34}$$

Proof. We prove this proposition in two steps. First, $\mathbb{F}_{L,\mathbb{S}} \supseteq \mathbb{F}_{G,\mathbb{S}}$; Given a global model g from a learning algorithm $A_G(\mathbb{S})$, a local learning algorithm setting $f_i = g$, where f_i is the local model for time series $S_i \in \mathbb{S}$, will produce the same forecasts. Second, $\mathbb{F}_{L,\mathbb{S}} \subseteq \mathbb{F}_{G,\mathbb{S}}$; We fix the local models $\{f_1, \dots, f_K\}$ from the local learning algorithm A_L. The set of forecasts $\{f_i(S_i)\}$ is finite since $\mathbb{S}$ is finite with $|\{f_i(S_i)\}| \leq |\mathbb{S}| = K$. g is thus a function mapping from a domain of cardinality K to a codomain of cardinality less than K. Such functions exists and we can construct a global function covering the full codomain of $\{f_1, \dots, f_K\}$. □

2.2.4 Generalization Bounds

We now present a well-known result about the relation of the in sample to the out of sample error. The proof requires the union bound and Hoeffding's inequality. For Hoeffding's inequality we require that our error metric $d(\cdot, \cdot)$ is bounded on $[0, 1]$. Some error metrics like SMAPE are bounded on $[0, 200]$ and can therefore be applied after rescaling. We prove the bound in a setting with a set of time series $\mathbb{S}$ of size K. For simplicity, we also assume that all time series have the same effective size N.

Proposition 2.2.2 (Generalization error). *Let $|\mathcal{H}|$ be the complexity of the hypothesis class $\mathcal{H}$, and K, δ be fixed. Then*

$$E_{out} \leq E_{in} + \sqrt{\frac{\ln(|\mathcal{H}|) + \ln(\frac{2}{\delta})}{2K}}, \tag{2.35}$$

with probability at least $1 - \delta$. Here, K is the number of time series in the dataset.

Proof. For a hypothesis $h \in \mathcal{H}$, let $X_i = d(h(\mathbf{x}_i), f(\mathbf{x}_i))$. That is, each X_i is the error for a chosen data point i in the dataset. In total there are NK effective points. By Hoeffding's inequality, and the definitions of the in sample and out of sample error, we have that,

$$\mathbb{P}(|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon) = \mathbb{P}(|\bar{X} - \mathbb{E}[\bar{X}]| > \epsilon) \leq 2e^{-2\epsilon^2 NK}.$$

However, this does not yet say something about the approximate function g that we find by learning. We use the union bound to relate the h to g. Because g has to be one of the hypothesis in $\mathcal{H}$, we can use the following crude inequality,

$$\begin{aligned}\mathbb{P}(|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon) &\leq \mathbb{P}\Big(\bigcup_i |E_{\text{in}}(h_i) - E_{\text{out}}(h_i)| > \epsilon\Big) \\ &\leq \sum_i \mathbb{P}(|E_{\text{in}}(h_i) - E_{\text{out}}(h_i)| > \epsilon) \\ &\leq |\mathcal{H}| 2e^{-2\epsilon^2 NK},\end{aligned}$$

where the second inequality is the union bound and the last is Hoeffding's inequality.

We let $\delta = |\mathcal{H}|2e^{-2\epsilon^2 NK}$. Now, with probability at least $1 - \delta$,

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \epsilon.$$

Since ϵ is also in the substituted δ, we can manipulate the expression to get,

$$\begin{aligned}|\mathcal{H}|2e^{-2\epsilon^2 NK} &= \delta, \\ \ln(2) + \ln(|\mathcal{H}|) - 2\epsilon^2 NK &= \ln(\delta), \\ \epsilon &= \sqrt{\frac{\ln(|\mathcal{H}|) + \ln(\frac{2}{\delta})}{2NK}}.\end{aligned}$$

Substituting this for ϵ gives,

$$E_{\text{out}} \leq E_{\text{in}} + \sqrt{\frac{\ln(|\mathcal{H}|) + \ln(\frac{2}{\delta})}{2NK}},$$

as desired. □

The generalization bound now lets us compare the bounds for local and global models. We specify a model class $\mathcal{J}$ for local models and a class $\mathcal{H}$ for the global model. Both of these can be, e.g., feedforward neural networks or AR models. Our analysis will highlight how the complexity of the two relates as a function of the number of time series in $\mathbb{S}$.

For the full set of time series, $\mathbb{S}$, we have for local models,

$$E_{\text{out}} \leq E_{\text{in}} + \sqrt{\frac{\ln(\prod_{i=1}^{K} |\mathcal{J}|) + \ln(\frac{2}{\delta})}{2NK}}, \tag{2.36}$$

A local hypothesis for the full dataset $\mathbb{S}$ is to chose one hypothesis from $\mathcal{J}$ for each time series. The product $\prod_{i=1}^{K} |\mathcal{J}| = |\mathcal{J}|^K$ is therefore a result of that the size is $|\mathcal{J}|$ for each time series. Global models get the usual bound,

$$E_{\text{out}} \leq E_{\text{in}} + \sqrt{\frac{\ln(|\mathcal{H}|) + \ln(\frac{2}{\delta})}{2NK}}, \tag{2.37}$$

where $|\mathcal{H}|$ is the complexity of the global hypothesis class.

These two bounds of the out of sample error give us a way to relate local and global models. Local models will, in general, have lower in sample error as they are fitted to each time series individually. In determining the better generalization error, we will instead look at which model has the tightest bound. It is clear that as the number of time series grows, the local generalization bound outgrows the global bound. This is true even though we usually will use a larger hypothesis class for global models. With the equivalence of the forecasts, this observation is the primary motivation for using global models for automatic time series forecasting.

2.2.5 Theory in the Real World

So far in this chapter, we have focused on developing learning theory to compare the generalization of local and global models. Doing so required some assumptions and crude inequalities. This section will discuss how some of these choices relate to the real world.

Independence of Time Series in a Dataset

Applying Hoeffding's inequality to get the generalization bound required us to assume that the time series in our dataset are independent in addition to the observations in each time series being independent. The effective size of each time series is a way to overcome the independence assumptions on a per time series basis. However, there is also the assumption of independence across time series. Global models are generally applied to datasets of related time series like electricity consumption of clients (Sen et al., 2019). These types of datasets violate the independence assumption. In general, it is hard to avoid dependencies as long as you observe the real world. We discuss data for global models in Chapter 4.

Loose Bounds

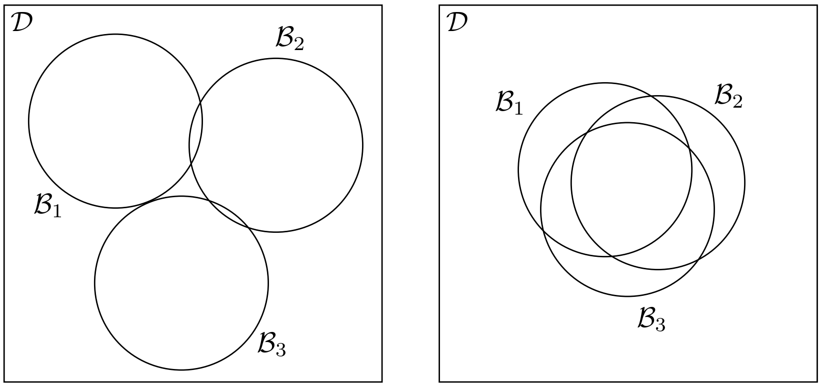
The bound for the generalization error is loose. We used the union bound and said that g had to be one of the hypotheses in the class in order to bound the generalization error of g.

This vastly over-estimates the probability, especially if the hypothesis class is large or the hypotheses in the class largely overlap. We best illustrate this using an example.

Example 3. *Let $\mathcal{H}$ be our hypothesis class containing 3 hypotheses. Rename the event $|E_{in}(h_i) - E_{out}(h_i)| > \epsilon$ to $\mathcal{B}_i$. The union bound states that,*

$$\mathbb{P}(\mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3) \leq \mathbb{P}(\mathcal{B}_1) + \mathbb{P}(\mathcal{B}_2) + \mathbb{P}(\mathcal{B}_3).$$

If the events $\mathcal{B}_i$ overlap, the union bound is loose because the right-hand side is the sum of the probabilities of the events. Figure 2.7 illustrate this fact. In Figure 2.6a we see the case where the events do not overlap much. The bound will be tight in this case. Figure 2.6b show the more realistic case with overlapping events. The bound is looser in this setting.



a: Tighter bound. **b:** Looser bound.

Figure 2.7: Two degrees of overlapping events, $\mathcal{B}_i$, in the probability space $\mathcal{D}$.

The question is now whether the hypotheses in the class overlap. We use a feedforward network, and the hypothesis in the class corresponds to different weights in the network. By changing the weights slowly, we can find an infinite number of infinitesimally different networks. They will differ only slightly in the outputs they produce, and the events $|E_{\text{in}}(h_i) - E_{\text{out}}(h_i)| > \epsilon$ and $|E_{\text{in}}(h_j) - E_{\text{out}}(h_j)| > \epsilon$ are likely similar on most datasets.

This touches on a more important aspect we fail to address. Our hypothesis class is infinite, but we assume a finite hypothesis class. However, the effective size of the hypothesis class is often far from infinite, as can be seen from the large overlap of many hypotheses. As described earlier, Vapnik-Chervonenkis (VC) theory can give an effective number on the size of the hypothesis class even when it is infinite. Using this theory, we can find a generalization bound as a function of the effective size of the class. This bound is known as the VC inequality and can also generalize to real-valued functions using the Pseudo dimension. A proof for the VC inequality can be found in Abu-Mostafa et al.

(2012). The inequality results in the generalization bound,

$$E_{\text{out}} \leq E_{\text{in}} + \sqrt{\frac{8}{N} \ln \frac{4m(2N)}{\delta}}, \tag{2.38}$$

for binary classification, where $m(N) \leq N^{d_{\text{vc}}} + 1$ with d_{vc} as the VC dimension of the model.

For practical applications, we introduce an empirical result presented in Abu-Mostafa et al. (2012). While the bounds are loose, they are found to be proportionally right in practical use. For practical applications, this says that a complex hypothesis class will indeed lead to large generalization error given insufficient data. This empirical result motivates global models even if the bounds are very loose, and we will never be close to them in practice. To see how loose the bounds become, you can use that the effective size of neural networks is in the order of the number of parameters (Vidyasagar, 2003).

2.3 Understanding N-BEATS

The purpose of this chapter is to understand the current state-of-the-art method on the M4 dataset, Neural Basis Expansion Analysis for Time Series (N-BEATS). To do so, we perform a series of ablation experiments to highlight the key drivers for the method's success. Currently, N-BEATS ranks comfortably with an overall score of 0.795 in front of the winning entry of the M4 competition that had an overall score of 0.821. We hypothesize that the success is mainly due to training a global model rather than the specific N-BEATS architecture.

The main findings of this chapter are as follows.

- A simple multi-step feedforward network for each frequency, with four layers and 512 hidden units in each layer, trained in the same pipeline as N-BEATS, reaches an overall score of 0.821. This is the same as the winning entry of the M4 competition.
- Ensembling weaker models is an important step to achieve state-of-the-art performance for N-BEATS. While the mean overall score of the ensemble members in a small ensemble is 0.896, the ensemble score is 0.803. This brings the method from a 17th place in the competition to 1st. The 0.795 score of N-BEATS is achieved by using a large ensemble. We use the smaller ensemble that gives a score of 0.803 in our experiments.
- A variety in input window lengths in the ensemble is more important than a variety in loss functions.
- A simple one-step feedforward network for each frequency, with 10 layers and 512 hidden units in each layer, has an overall score of 0.902. However, initializing the forecasts using a naive seasonal initialization instead of naive, results in a score of 0.812. This would have been sufficient to win the M4 competition.

Our experiments are done by modifying the original N-BEATS repository[1]. We start by introducing the N-BEATS architecture and highlighting the similarities to simple feedforward networks. Next, we do a simple study on the effect of the model's depth and width before looking into the effects of ensembling. Lastly, we remove parts until we are left with a simple feedforward architecture. This is used to compare multi-step to one-step methods. Importantly, we look at initializations of the forecasts of the two.

The M4 dataset is introduced in Chapter 4 together with a larger dataset and a discussion about data for global models. The scoring metric used to evaluate performance on the M4 dataset is introduced in Chapter 5.

2.3.1 Model Architecture

The N-BEATS architecture is very similar to an MLP, as seen in Figure 2.8a. The model consists of stacks, which again is built by blocks. The general N-BEATS model we use in these experiments is a multi-step model with 30 stacks with one block in each stack, as shown in Figure 2.8b. Each stack takes an input and gives a forecast and a backcast

[1]https://github.com/ElementAI/N-BEATS

as output. The forecast is skipped to the end, and the backcast is subtracted from the input before being sent to the next stack. Potentially this will allow the model to model distinct characteristics in each stack. For example, one can imagine that one block models seasonality, one the trend, and one the residuals' autoregressive features. The authors also propose an interpretable version of N-BEATS meant to enforce trend and seasonality decomposition. This is done by using trend and seasonality stacks in succession. The forecast of trend blocks is forced to be a polynomial of low degree. Seasonality stacks are enforced to produce periodic forecasts. The interpretable version of N-BEATS scores slightly worse but is interesting in its attempt to be more explainable. However, we restrict ourselves to the more general and better-performing version of N-BEATS.

One N-BEATS model is fitted to each frequency, but the same architecture is used for each frequency model. The M4 dataset has six frequencies, and as a result, one forecasting model of N-BEATS has six frequency models. The general model has 30 stacks with one block in each stack and four fully connected layers in each block. Such models have more than 100 fully connected layers, and the total number of parameters is large. Typical models, depending on the input window length, have about 25M parameters. Each model in the ensemble thus has about 150M parameters. As a comparison, the state-of-the-art model on ImageNet has 480M parameters[2]. ImageNet has more than 14M annotated images and is the benchmark for image classification. The M4 dataset has 100,000 time series that also are less dense in information than images. The apparent risk for overfitting is large. A full ensemble of N-BEATS has 180 models, with one submodel per frequency in each model, six different input window lengths, three types of loss functions, and as a result, more than 27B parameters since each model is also run ten times in the full ensemble. This is a lot and equivalent to 270k parameters per time series for the full ensemble and 1,500 per time series for each model in the ensemble.

Two steps prevent massive overfitting. First, the model uses backward residual connections that are subtracted from the input. Skip connections send each block's forecast to the final output, and this, together with the bottlenecks that occur after each block where the dimension is brought down to the input window length, can help restrict the model. The model's total complexity is less than for an MLP with 100 layers as these steps act as regularizers. Second, the learning rate schedule of the optimization is not aggressive. With a more aggressive schedule, we have a higher risk of overfitting. Instead of tuning the learning rate, the model is trained for more iterations. The parameters of the network and training pipeline are said to be chosen based on a validation set with the last horizon of each time series.

N-BEATS can be transformed into an MLP by using one stack, one block, and then whatever number of fully connected layers we choose. All residual and skip connections will then be irrelevant as they all go from the single block to the output. We will not be able to train as deep a network as N-BEATS without adding skip or residual connections. This is because the gradient can become too small as it is backpropagated through the network so that the weights will not change and that there is a larger chance of overfitting.

A key implementation detail when transforming the model into an MLP is the naive initialization of the forecast. This is not mentioned in the paper but implemented in the code. The consequence of the initialization is that the model is modeling the residuals

[2]https://paperswithcode.com/sota/image-classification-on-imagenet

from a naive forecast. Our experiments later find that the models with naive initialization struggle with frequencies that show strong seasonality. This observation leads us to propose a naive seasonal initialization.

Our experiments found that N-BEATS does not need this initialization to converge, but an MLP does. A reason can be that the MLP does not have skips that could act as such an initialization. N-BEATS have frequent skip connections and might therefore not be as dependent on the initialization.

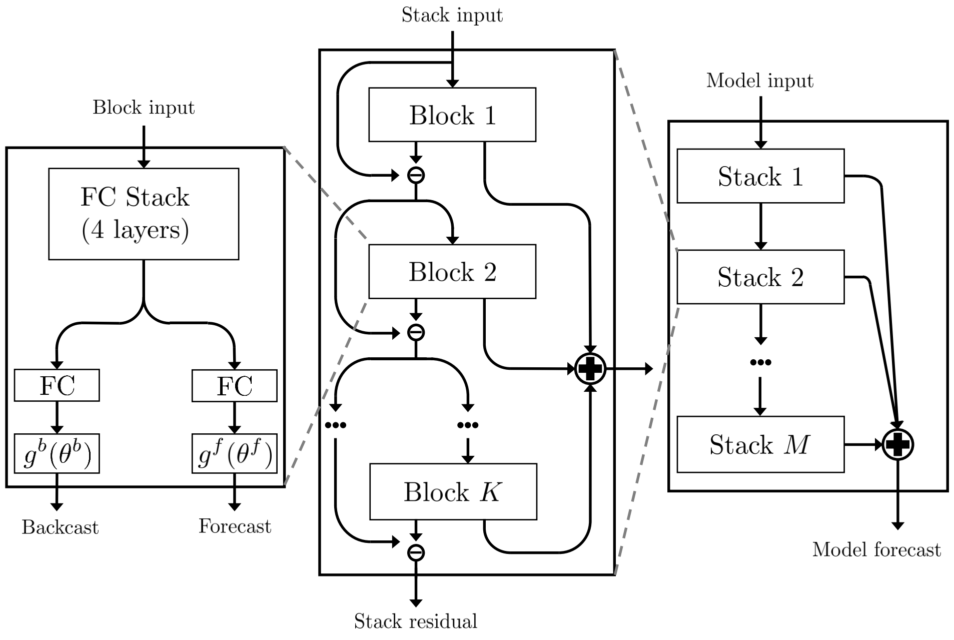
By default, N-BEATS is a multi-step model. However, by setting the output length to be one, we can create a one-step model. One-step models are needed to create universal time series encoders. These will also need scaling to be able to handle unseen time series and their scales. N-BEATS does not scale the input to the model. However, in a follow-up paper, the authors claim that scaling with each sample's max value gives similar scores (Oreshkin et al., 2020).

Width and Depth of the Model

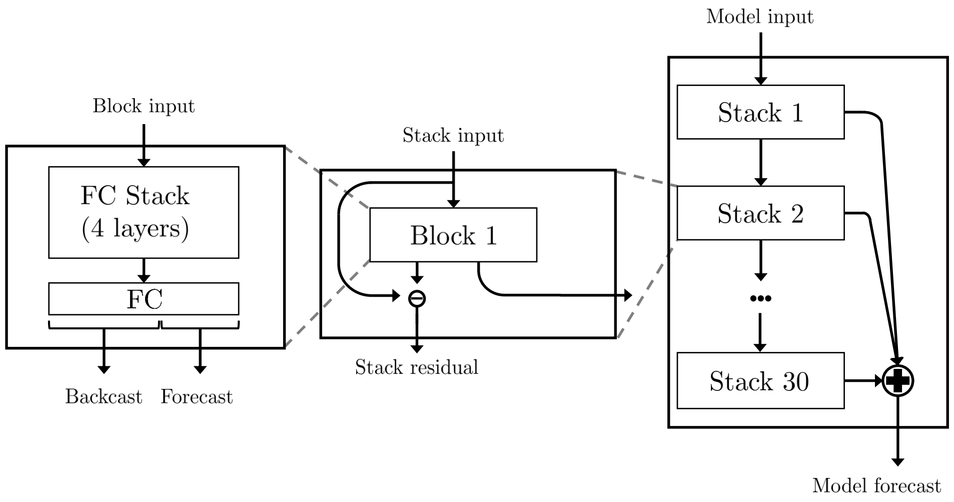
We perform a short experiment on the model's width and depth to get a better feel of the effect of the hyperparameters. First, we let the number of stacks range from 1 to 50 while keeping the width of 512 and the number of layers in the block, 4, constant. The results of this experiment can be seen in Figure 2.9a. Performance is increasing as the depth increases. However, the most interesting observation is that a model using one stack with one block with four layers has the same score as the winner of the M4 competition. This model is a simple multi-step MLP with 5 layers, also counting the output layer, and 512 hidden units in each layer.

Figure 2.9b shows the experiment where we vary the number of hidden units in each fully connected layer. Again, we see better performance with increased model complexity. The thinnest model also performs very well. We try using a small model with the thinnest and shallowest of both experiments, but this did not converge well at all and had a total overall score of 9.122.

One can ask why the MLP benchmark of the M4 competition performed poorly when we observe winning performance in this experiment. The benchmark MLP in the M4 competition had an overall score of 1.642 and ranked 57th out of 59 entries. There are three main differences between these two approaches. First, the benchmark was trained on each time series separately and only had six internal nodes. Restricting the number of nodes can have been a choice made to prevent overfitting. In this experiment, we train a larger model jointly on each frequency. Second, the benchmark is a one-step method, while N-BEATS is a multi-step method. Multi-step methods can sometimes act as a regularizer, but they are less flexible. In the M4 competition, all time series in the same frequency have the same forecasting horizon. This allows for less flexible multi-step methods. Lastly, there are several preprocessing steps in the benchmark. This includes detrending and deseasonalizing. Contrary, N-BEATS works on the raw time series. From these observations and the results of the experiments, it is clear that the MLP benchmark is not realistic. All benchmarks in the M4 competition are local models, and it would have been more informative to include global benchmarks. However, local models have dominated the literature before the competition, which likely explains why global models are omitted.

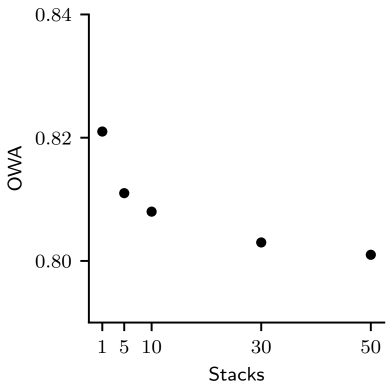


(a) The general N-BEATS architecture. For both the general and interpretable version.

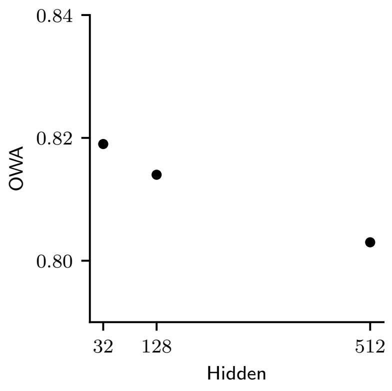


(b) The specific N-BEATS general architecture used in the paper.

Figure 2.8: The model is divided into M stacks that all give a stack forecast and a residual. The forecast are skipped to the output while the residual is forwarded to the next stack. Each stack has K blocks that give a block forecast and residual. The block forecast of each block are added while the residuals are subtracted from the input. A block has four fully connected (FC) layers and a backcast and forecast layer.



(a)



(b)

Figure 2.9: Overall score of N-BEATS by varying the number of stacks in Figure 2.9a, and the number of hidden units in Figure 2.9b. All other parameters are kept constant while varying the parameter in question.

2.3.2 Ensembling

Ensembling can be an essential technique to boost performance. The idea is that weaker models together will produce better predictions partly by reducing variance. We have already introduced the ensemble used in N-BEATS, but we will, in this section, explain the ensemble construction in more detail. Our experiments reveal important components in a well working ensemble.

One ensemble member of N-BEATS has six submodels, one for each frequency. It has a loss function and a lookback. The lookback determines how many multiples of the forecasting horizon the length of the input window should be. All submodels are thus equal except for the input window length and the output length, determined by the forecasting horizon. The loss functions used are Mean Absolute Scaled Error (MASE), Symmetric Mean Absolute Percentage Error (SMAPE), and Mean Absolute Percentage Error (MAPE). Lookback ranges from 2 through 7. An ensemble consists of models with all combinations of the loss functions and lookbacks. Also, a larger ensemble can be made by repeating each member several times with different initializations.

The full ensemble has 18 members if each configuration is run once and 180 members if they are run ten times as in the reported results. We will consider the small ensemble of 18 models to limit the computational cost. The added benefit of using a full model is a drop in overall score from 0.802 to 0.797. The large ensemble of 180 models has 1080 frequency models. Each frequency model has about 25M parameters. In total, there are more than 27B parameters. A visualization can be seen in Figure 2.10.

Ensemble Experiments

We want to better understand the drivers for performance in an ensemble and, therefore, look at combinations of the members. Figure 2.11 shows combinations of ensemble mem-

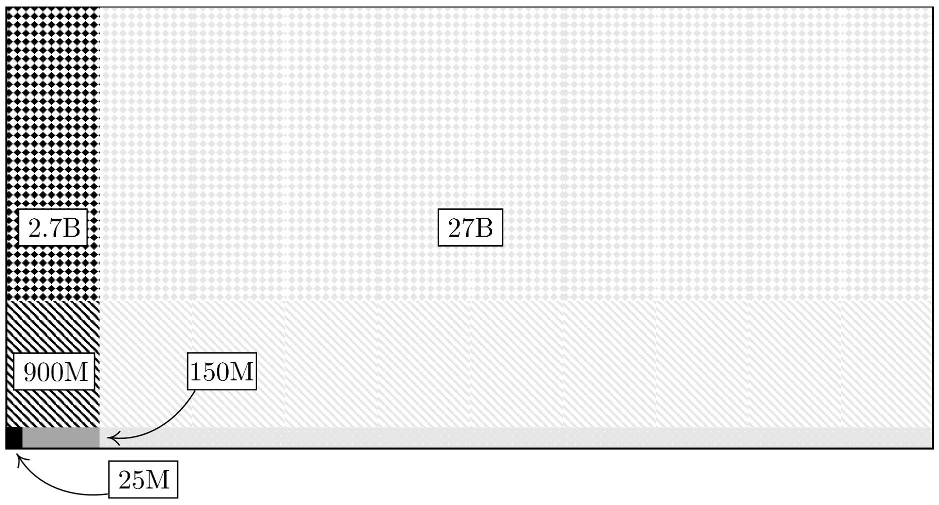


Figure 2.10: The full ensemble used in N-BEATS. The black rectangle with 25M parameters is on frequency model. The six frequencies together make up the grey model of 150M parameters. With six different lookback ranges, we get the 900M model. Using three loss functions we get the 2.7B model. Lastly, repeating these models 10 times using different random initialization creates the massive 27B ensemble.

bers from the small N-BEATS model (18 members). The leftmost scatters with labels lb (lookback) n indicate ensembles with the three loss functions for a given lookback. There does not seem to be a big difference between the ensembles. Next, the scatters labeled lb n-$n+1$ are ensembles of pairs of lookbacks and all the three loss functions. These show a difference in that the shorter lookbacks are more important than the longer lookbacks. We combine two lookbacks to get an ensemble of the same size as the rightmost scatters. These are ensembles of all lookbacks with a given loss function. The three loss functions all perform similarly and better than the ensembles of similar sizes with all loss functions but not all lookbacks. From this observation, a variety in input length is more important than a variety in loss functions.

Figure 2.12 shows a boxplot of the ensembles and the ensemble members (singles) that we train in this chapter. It clearly shows how ensembling is crucial to get competitive scores. The best single models show competitive scores, but they are still far from the ensembles' superior scores.

2.3.3 Multi-step and One-step Forecasting

One-step methods are vital as they are needed to build universal forecasting models. In this subsection, we turn N-BEATS into an MLP and compare multi and one-step models. We use a 10 layer MLP with 512 hidden units and the same ensemble members as before.

Table 2.1 show the results of the multi and one-step models. With a naive initialization, the one-step model struggles to pick up the seasonality in the monthly frequency. This is an important frequency because it stands for 48% of the dataset and thus 48% of the score.

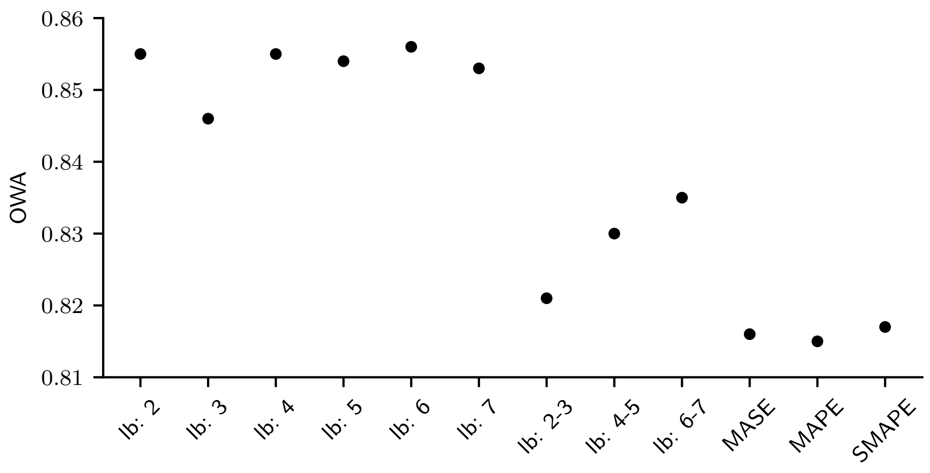


Figure 2.11: The total overall score for ensembles with different members. The points labeled lb n are ensembles with three members with the same lookback but different loss functions. Next, scatters labeled lb n-$n+1$ are ensembles where we have combined members with lookback n and $n+1$ and three different loss functions. These ensembles have six members. Lastly, the scatters labeled with loss functions are ensembles with all the possible lookbacks but only one type of loss function. These ensembles also have six members.

The score of the 10 layer multi-step MLP is slightly worse than for the 4 layer model we had in the depth experiment. This can be because of initialization as they are close but also indicate that small networks can also forecast well. Our one-step model has a significantly worse score. We try to mitigate this by using a naive seasonal initialization of the forecast. For the one-step model, we use the observation from p observations ago, where p is the period. For the multi-step model, we use the full last period as the initialization. Interestingly, we see that the initialization does not add anything to the multi-step model but significantly boosts the one-step model. It makes the one-step model better than the reference multi-step model without initialization.

There is one potential reason for the performance boost that we will discuss. As the initialization uses the observation from the last period as initialization, it will often use an actual observation from the training set. This is as opposed to using its own predicted value at the last point as initialization. One drawback with one-step models is that errors can accumulate. Using an actual value as initialization will help mitigate this. In many of the frequencies of the M4 competition, the forecasting horizon is short enough that the model will use real observations in many of its forecast initializations. If a real-world application requires forecasting many seasons into the future, we may see the effect diminish. However, using the last period as initialization will still alleviate some of the problems of modeling the seasonality.

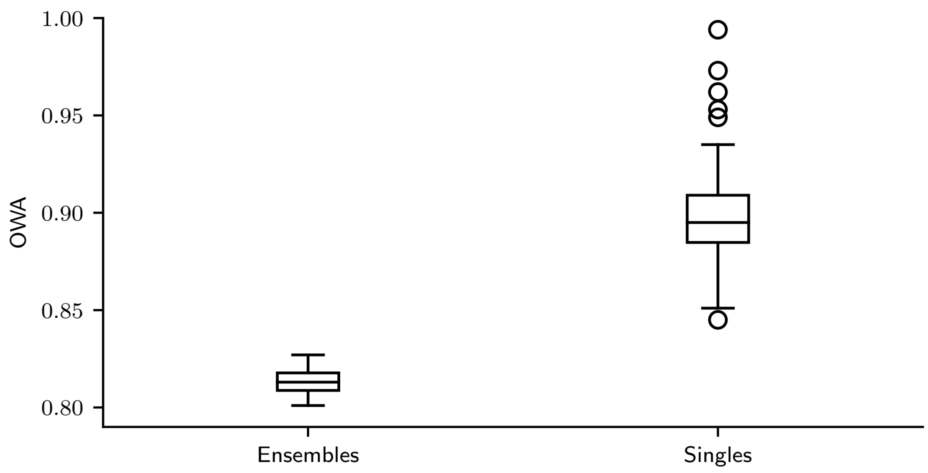


Figure 2.12: Total overall scores for the ensembles trained so far in this section and the score of the members of these ensembles. We have capped the y-axis at 1 for readability. There is one ensemble member above with a score of 1.44.

Table 2.1: Overall score for N-BEATS with one-step and multi-step forecasting. Snaive indicates naive seasonal initialization of the forecast.

	Yearly	Quarterly	Monthly	Other	OWA
Multi-Step: naive	0.775	0.828	0.870	0.884	0.827
One-Step: naive	0.761	0.828	1.091	0.874	0.902
Multi-Step: snaive	0.776	0.833	0.879	0.862	0.832
One-Step: snaive	0.759	0.839	0.844	0.871	0.812

2.3.4 Residual and Skip Connections

We wish to look at the importance of the residual and skip connections in N-BEATS. This is done by removing the skip connections, removing the residual connections, and, lastly, removing both. Table 2.2 shows the results from this experiment. It shows that either skip or residual connections are necessary, but not necessarily both. When there are no residual or skip connections, the model turns into a very deep MLP with bottlenecks after each block. Bottlenecks are layers that have fewer hidden layers than the layers before and after, therefore possibly limiting the expressive power through that layer thus acting as a bottleneck. Training deep MLPs is challenging, and one could perhaps expect an even worse result than what Table 2.2 shows.

Skip connections for MLPs are not as straightforward as for N-BEATS as the internal layers have different sizes than the output. A solution to this would be to add the skips to the layer before the output instead. This second last layer has the same size as the internal layers and can therefore be used. This is how we will apply skip and residual connections later.

Table 2.2: Overall score for N-BEATS with and without skip and residual connections.

	Yearly	Quarterly	Monthly	Other	OWA
Both	0.779	0.801	0.820	0.834	0.803
Skip	0.772	0.830	0.849	0.852	0.817
Res	0.775	0.810	0.841	0.842	0.812
None	0.972	1.032	1.093	1.297	1.040

Table 2.3: Overall score for N-BEATS with and without scaling of the inputs.

	Yearly	Quarterly	Monthly	Other	OWA
No-Scaling	0.779	0.801	0.820	0.834	0.803
Scaling	0.787	0.819	0.832	0.835	0.814

2.3.5 Scaling

The last part of this section is a brief look into the effect of scaling. We have previously discussed how this perhaps is a requirement for universal forecasting models as they will operate on scales not seen in training. We experiment using maxscaling by scaling each sample with its max value. In their follow-up paper (Oreshkin et al., 2020), the authors state that the model is comparable with and without scaling but do not support this by listing the scores. The results from the experiment are shown in Table 2.3. Scaling does increase the score, but whether it is significant is harder to determine. It is comparable to when we remove connections in the previous section.

Chapter 3

Literature Review

In this chapter, we give an overview of relevant time series literature. We focus on the univariate forecasting problem and start by reviewing the state of time series modeling in general. Second, we discuss current approaches to zero-shot forecasting. Next, we give a short review of pretraining where we also visit other domains. Lastly, we discuss where this thesis fits into the literature.

3.1 Time Series Modeling

Forecasting is an important business problem, and large industrial players are at the front of time series forecasting. Both Amazon and Facebook have developed solutions for automatic time series forecasting. DeepAR is a model that Amazon provides through its web services (Salinas et al., 2020). It is a recurrent architecture that is trained jointly over the time series in the dataset. This makes it a global model, and the use-case is meant to be forecasting sets of related time series. Facebook has developed Prophet aimed at business forecasting (Taylor and Letham, 2018). It is designed to be fast to fit and to produce automated forecasts that the user can tune. Prophet is a statistical model that is built to be explainable using the common trend and seasonality decomposition. It is a local model and therefore not as relevant in our analysis. However, it highlights the effort of large players in the time series analysis domain.

Statistical models like Prophet are important because they are easier to interpret. Parameter estimates on seasonality and trend effects make the model more explainable as we can better understand the contribution of well-known effects. Seasonal Auto-Regressive Moving Average (SARIMA) models, Exponential Smoothing (ES) models, and Generalized Auto Regressive Conditional Heteroskedasticity (GARCH) models are examples from popular statistical model families used in time series modeling. Brockwell et al. (1991) gives a comprehensive treatment of statistical time series models.

Recent years have seen an increase in the use of machine learning models as they allow for more complex models made possible by increased access to data and computing time. Recurrent models such as Recurrent Neural Networks (RNNs) and Long Short Term

Memory networks (LSTMs) have been widely used for sequence modeling tasks. One of the main challenges of recurrent architectures that the vanishing gradient problem can make it hard to model long-term dependencies. Bai et al. (2018) show that recurrent architectures have limited memory in practice and that it is better to use a finite memory model in some applications. They suggest a Temporal Convolutional Network (TCN) as a baseline for sequence modeling tasks. The architecture simplifies WaveNet (Oord et al., 2016) and uses exponentially dilated convolutions to get potentially a very long receptive field.

The M4 forecasting competition in 2018 aimed to find good and robust solutions relevant to business forecasting problems (Makridakis et al., 2020). The competition introduced the M4 dataset consisting of 100,000 time series from different domains and of several frequencies. It was the first of the M competitions where a deep learning method surpassed statistical methods. Smyl (2020) won the competition with a hybrid model using a parameterized Holt-Winter model and an RNN to model the residuals. Both the parameterized statistical model and the RNN were learnable instead of first fitting a purely statistical model and then modeling residuals with a deep learning model. The winning solution also overcomes limited data for deep learning models by fitting some parameters jointly within each frequency. The second-place solution by Montero-Manso et al. (2020) also used machine learning in combination with statistical models. They used an ensemble of statistical models where the ensemble weights are decided by a boosted tree model based on features of the time series in focus. The boosted tree can be seen as a meta-learning model that learns to optimize the model choice based on the task. The task is to forecast a time series given a set of extracted features from the time series.

Recent research has continued to use the M4 dataset as a benchmark dataset for general univariate time series forecasting methods. N-BEATS (Oreshkin et al., 2019) is currently the best performing model on the M4 dataset. It is a feedforward network with forwards and backward residual connections. While Smyl (2020) use a hybrid of a statistical and deep learning model, N-BEATS is a pure deep learning model. We refer to it as an almost global model in this thesis because it uses one model for each frequency. On each frequency, the model is trained jointly over the time series of that frequency. A large ensemble is used to reach state-of-the-art performance.

The use of global or almost models to overcome data scarcity in short time series is a trend in deep learning for time series. Using one model per frequency is an example of this trend. However, more common is the use of global models when the time series in a dataset are related. This can be for modeling road traffic, electricity loads, or web page data Sen et al. (2019). Some theoretical motivation for global models is shown by Montero-Manso and Hyndman (2020). Importantly, they show that global and local models can produce the same forecasts. Also, while local models can have a lower in-sample error, Montero-Manso and Hyndman (2020) show that global models have lower generalization errors than local models at the same model complexity. We use the number of parameters in a model as a measure for its complexity. As the number of time series to forecast grows, the total number of parameters for local models will grow faster than for global models. The theoretical results do not assume any dependence between the time series in the dataset. Through experiments on M4, the authors suggest that the lower generalization error is due to overfitting when using local models. Empirical results indicate that both statistical and deep learning models benefit from being fitted jointly over time series. For statistical models, it seems like components like seasonality are more robustly estimated. The theoretical and experimental results by Montero-Manso and Hyndman (2020) suggest that global models are applicable to sets of unrelated time series because they allow for more complex models while reducing the risk of overfitting.

3.2 Pretraining

Unsupervised pretraining aims to train a model on an unsupervised task to learn something useful for other downstream tasks. Reasons for this can be that the parameters are closer to a more stable part of the parameter space, thus leading to more stable and better optimization and that it acts as a regularizer in cases where the data for the downstream task is limited, and we risk overfitting. The forecasting task is always supervised because we can always create a forecasting task when we have a time series. Unsupervised pretraining is used in other time series tasks while we call it pretraining for the forecasting task.

Large language models such as the GPT models (Radford et al., 2018), (Radford et al., 2019) and (Brown et al., 2020), are unsupervised pretrained models that generalize well to other language modeling tasks than text generation. The GPT models are generative autoregressive models built with transformers (Vaswani et al., 2017). In essence, this means that they output a distribution over a vocabulary of words given the previous context. Transformers are models built by attention modules. They can be large because they are highly efficient to train. Generative autoregressive models have also been successful at modeling audio (Oord et al., 2016) and images (Van den Oord et al., 2016). The GPT type model from language modeling has also been applied to images by Chen et al. (2020). In addition to image generation, the authors show that the model in practice works like an encoder-decoder where the middle layers are useful representations of images for downstream tasks like classification.

Large pretrained models have not yet fully entered the time series domain. Serrà et al. (2018) train parts of a classifier jointly on the full UCR archive for classification reaching performance a little shy of the top methods. Sagheer and Kotb (2019) pretrain a recurrent network for multivariate time series from the UCI archive (Dua and Graff, 2017), and Laptev et al. (2018) perform a similar experiment using hourly electricity loads (Kwac et al., 2014) as both the source and target dataset. These works move towards universal time series models. However, none of these works thoroughly discuss general data from various domains, general model architectures, and competitive benchmarks.

3.3 Zero-shot Forecasting and Transfer Learning

Zero-shot forecasting is forecasting using a model that has not been fitted to the time series it is set to forecast. It is relevant in cases where you have very limited data, limited resources to fit a model, or limited time to train a model. As more and more data is collected from, e.g., sensors, it is desirable to have models that can work well without fitting each time series individually, and that can work well on unseen time series.

Oreshkin et al. (2020) use a diverse dataset and compare against competitive benchmarks. As part of the M4 competition, several standard benchmarks were calculated on the M4 dataset. Together with the entries in the competition, this gives a very competitive scoreboard for evaluating models. As part of their study, the authors also test DeepAR in a zero-shot setting. DeepAR is not competitive in this setting, neither to N-BEATS or the other benchmarks. Amazon uses DeepAR in their web services, and it has also scored well on homogenous datasets where the time series are all from the same domain. Whether it is the heterogeneous distribution of M4 or something else that prevents DeepAR from scoring well also on M4 is not discussed.

Transfer learning can be used in settings where we have limited data but still want to fit the model to the limited data. Relying on characteristics learned from another dataset, the model can often be trained with less risk of overfitting by allowing fewer parameters to be fitted to the target dataset. Fawaz et al. (2018) perform a thorough experiment on the datasets in the UCR time series classification archive (Dau et al., 2019). By training models on one dataset and then changing the last layer of the model before retraining to the target classification task, they find the models to generalize better. However, it is dependent on the similarity between the source and target dataset. This indicates that it is important that the distributions of the source and target datasets are similar.

3.4 Relation to Current Literature

This thesis is inspired by the recent advances in language modeling using pretrained models and the work by Montero-Manso and Hyndman (2020) discussing global and local models. This led to the discovery of the paper by Oreshkin et al. (2020) and the zero-shot experiments using FRED as a source dataset for zero-shot forecasting of the M4 dataset. Removing the skip and residual connections from N-BEATS yields an MLP. In this thesis, we want to test the abilities of this simple form of neural networks to look into whether the good performance of N-BEATS is due to the model architecture or global properties. In addition, we want to expand on the analysis using a pretrained model from FRED by looking into the effect of the size of the source dataset and its transfer learning abilities.

The main distinction from previous work is accurately forecasting a general dataset using a truly global model. In addition, we discuss the properties of the data for global models. Lastly, global models, together with other techniques, can reduce the number of parameters we need in a full ensemble. The full N-BEATS ensemble is perhaps too large for practical applications. Sustaining accurate forecasts while reducing the number of parameters in a full solution is useful.

Chapter 4

Data for Global Models

Global models differ from local models by their shared parameter set. Because of this, global models use information across time series in the dataset. The shared information, represented by the parameters, can introduce information leakage through time when the last forecasting horizon of each time series is used as the test set. This effect is called look-ahead bias. Local models will not be affected by the potential information leakage because they treat each time series independently. In this chapter, we look into the possible effects and requirements of datasets for global models. The main findings in this chapter are:

- Global models can ”cheat” by using temporal information in the dataset to peak at the test set. We give both a constructed and real-world example and show that this property can be present in the M4 dataset.
- A possible solution is to use the standard train test split by date. While this mitigates the problem at test time, it still poses two main challenges. First, we are potentially leaving out more of the available data. Second, splitting by date will also induce bias in which date we choose.

We first introduce known large open-source time series databases. Industrial players like Amazon (Salinas et al., 2020) and Facebook (Taylor and Letham, 2018) have their own massive private datasets of real-world data. These consist of count or event data of their sales and user actions. The datasets we consider are, on the other hand, continuous and origin in economic data or observation of systems. Next, we discuss look-ahead bias for global models and give an artificial example. Lastly, we look at possible solutions to mitigate the look-ahead bias and how this will affect the datasets and remaining challenges.

4.1 Time Series Databases

Pretraining requires lots of data from different domains and frequencies. The largest open-source database of general time series data is the Federal Reserve Economic Data (FRED)

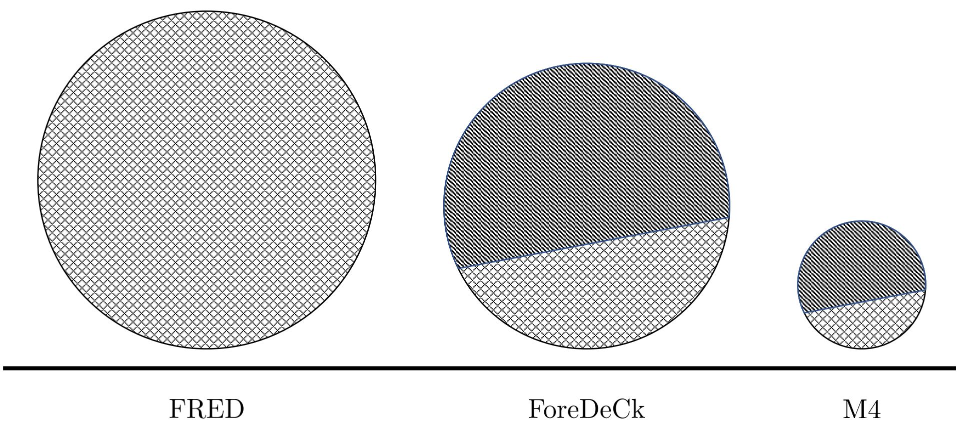


Figure 4.1: The relative sizes of the databases discussed in this chapter. The FRED dataset has more than 700,000 time series, ForeDeCk more than 490,000, and M4 has 100,000 time series.

(Federal Reserve Bank of St. Louis, 2020). It contains time series ranging from flour production at mills in the 1800s to GDP per capita in South Sudan, the world's youngest country. FRED is the basis of other time series databases like ForeDeCk, which contains data from FRED in addition to other sources. A subset of ForeDeCk was used in the M4 forecasting competition and is now a standard benchmark for automatic time series forecasting. We give each dataset a short introduction to better understand their usage in the literature, how they connect, and why they allow for look-ahead bias when using global models. Figure 4.1 illustrates the sizes of the datasets we discuss and how ForeDeCk contains data from FRED, and how M4 is a subset of ForeDeCk.

4.1.1 FRED

The distribution of the time series of FRED into frequencies is shown in Table 4.1. Yearly data is heavily overrepresented. We would ultimately want the dataset to be a general dataset that can be used for pretraining. Our view of a general distribution over time series is the M4 dataset, which is our downstream task. However, the M4 dataset has had comments on it not containing enough noisy time series with shorter time intervals (Fry and Brundage, 2019). The FRED dataset is even less biased towards these shorter frequencies than the M4 dataset. Economic data is often yearly, quarterly, and monthly, as shown in the distribution in Table 4.1. Applications in economics are likely the ones that will have the highest potential of pretrained models on FRED.

While FRED is not representative of a general time series distribution, it is likely relevant for pretraining tasks. One of the main arguments for pretrained models is to reduce overfitting in cases with limited data. Yearly, quarterly, and monthly time series are likely to be present in these types of settings. In these cases, FRED is likely a valid dataset to use for pretraining.

Table 4.1: The distribution over frequencies for the FRED dataset.

Freq	Yearly	Quarterly	Monthly	Weekly	Daily	Total
Num	197,920	51,913	100,808	1,728	43	352,412
Pct	56.2%	14.7%	28.6%	0.5%	0.0%	100.0%

Sourcing the FRED Dataset

We source the dataset using the FRED API together with a Python wrapper (Williams, 2015). The Federal Bank of St. Louis gives access to the FRED API through an online form. Sourcing the data is challenging for two reasons. First, there is no complete list of the time series in the database nor an overview of the database structure. Second, the API has a restriction on the number of calls a user can make per minute.

Each time series has a link to its parent and children in a hierarchical structure. The id of a time series is required in order to download it. We retrieved all ids by traversing the unknown structure using the references to parents and children. Next, we downloaded each time series using the fetched ids. Downloading the time series takes 5-8 days because of the restriction on the number of calls. Beware of moving too close to the limit, or you will be in time out and receive bad karma. We found the limit to be somewhere in the range of 110-120 calls per minute. Time outs also decrease the call limit for some time after the user is allowed access again.

Preprocesing FRED

A time series object from FRED contains lots of metadata together with the observations. In this project, we are only interested in the observations and, therefore, store each observation's value and date. However, the metadata are highly interesting for other projects.

We further preprocess the data by enforcing a minimal length of each time series. The minimal lengths are chosen based on the minimal lengths of the M4 dataset. The minimal values are 19 for yearly, 24 for quarterly, 60 for monthly, 93 for weakly, and 107 for daily.

Next, we try to save any series with missing values. We split series with missing values into subseries. If the subseries pass the minimal length condition, they can be used. This step avoids throwing out series that are long but only contain a few missing values. Of the 686,623 time series we download, 24.5% contain missing values.

Some of the time series are constant or only have a small finite set of values. While this, in theory, should be learnable for the models, we found that this, in some cases, just introduced noise hiding the signal. We, therefore, remove any constant series, series with less than five distinct values. There is also a considerable amount of time series that contain almost only zeros. We discard time series with more than 50% zero values for the same reason. Lastly, we remove time series with negative values to be closer to M4, which does not have any negative values. These steps result in the distribution over frequencies in Table 4.1.

Table 4.2: The distribution into frequencies and domains in the M4 dataset (Makridakis et al., 2020)

Freq	Micro	Industry	Macro	Finance	Dmgr	Other	Total	Pct
Yrly	6,538	3,716	3,903	6,519	1,088	1,236	23,000	23.0%
Qrtrly	6,020	4,637	5,315	5,305	1,858	865	24,000	24.0%
Mthly	10,975	10,017	10,016	10,987	5,728	277	48,000	48.0%
Wkly	112	6	41	164	24	12	359	0.4%
Dly	1,476	422	127	1,559	10	633	4,227	4.3%
Hrly	0	0	0	0	0	414	414	0.4%
Total	25,121	18,798	19,402	24,534	8,708	3,437	100,000	100%

4.1.2 ForeDeCk

The ForeDeCk database is a collection of time series from a range of sources, including FRED. It was proposed as an open-access database that could be used for benchmarking methods by Spiliotis et al. (2017). Unfortunately, it is currently unavailable due to downtime on the server and no technicians allowed on site due to covid. It would have been the most relevant dataset to use for pretraining as the M4 dataset is a direct subset, and the distribution should be very similar.

4.1.3 The M4 Dataset

chosen from a business perspective

A random subset of ForeDeCk was used in the M4 forecasting competition (Makridakis et al., 2020). It has time series from 6 domains and six frequencies. Table 4.2 shows the distributions of the 100,000 time series in domains and frequencies. The yearly, quarterly, and monthly frequencies are dominating the dataset. Fry and Brundage (2019) suggest that future M competitions include more data of the high-frequency data that are more common in practice and time series with a very short history.

Table 4.2 show that the micro, industry, macro, and finance dominate and are comparable in size, while the demographic and other categories are smaller. The distribution for each frequency over the domains is not the same. Specifically, we see that the hourly time series only come from the other domain.

The M4 dataset is now often used as a benchmark. As M4 inherits from FRED through ForeDeCk, there might be duplicates. Oreshkin et al. (2020) compares the time series in their version of FRED and M4 to look for duplicates. They report three yearly, 34 quarterly, and 195 monthly time series in M4 originating from FRED. This is low compared to the dataset's size and can likely be disregarded without affecting the results much.

4.2 Temporal Bias in General Time Series Databases

When creating general time series datasets for time series forecasting, current practice is to consider the last horizon of each time series as the test set as in the M4 competition (Makridakis et al., 2020). Local models have often been the study of previous forecasting competitions. Proper treatment of the challenges with the dataset has therefore not been discussed. Future competitions should take care in creating the dataset and look into the effects of temporal bias.

We illustrate the look-ahead bias by an example. Consider a dataset consisting of three time series, $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$, where $\mathbf{x}_i = \{x_{i,1}, \ldots, x_{1,t_i}\}$. Now, assume the time series in the dataset are measuring related processes. In other words, the time series are correlated. We now define an interval $T = [T_0, T_1]$, with $|T| > |\mathbf{x}_i|$ and let the time series be correlated across this period. Next, let the time series $\mathbf{x}_i$ begin at different indexes. Lastly, construct the test set by using the last n observations as the test set and the remaining as the train set.

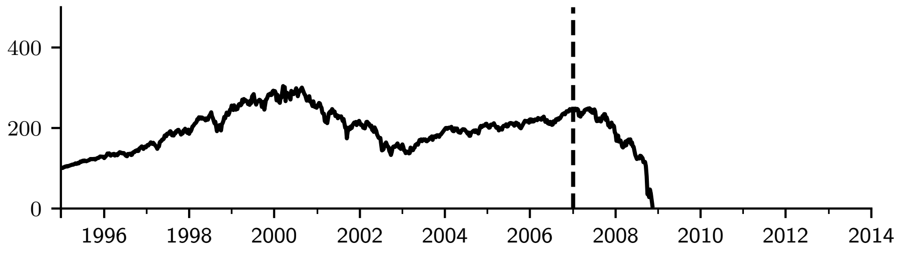
Parts of the training data are now correlated with the test data. This allows a global model to learn patterns from the time series ahead in time that will resemble patterns that will occur at test time in the time series that are back in time. Local models can not exploit the dataset in the same way, thus favoring global models. The parameters in the global model are affected by events ahead in time in other time series. However, the local model parameters are just an effect of prior observations in the same time series.

The effect of the look-ahead bias is not necessarily large, but this remains to be tested. In Example 4 we create an artificial dataset with financial data to further exemplify the issue and how severe it might be.

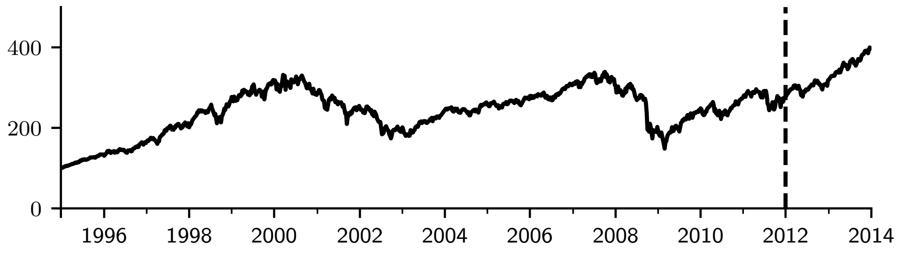
Example 4. *Let there be two listed companies in the dataset, ABC and XYZ. They were both founded in 1995 and survived the Dot-com bubble*[a] *in their early years. However, in the 2008 financial crisis, company ABC did not make it through. XYZ survived albeit harsh times and was bought up by a large tech company in 2014. Our data provider's data are weekly stock prices, and as the careful statisticians we are, we want to use the last 100 weeks of each company as a test set. Our two time series with their train and test split can be seen in Figure 4.2.*

Now, imagine that you are a model that is given the task of forecasting weekly stock prices. By nature, the prices are very hard to predict and debated to be a random walk. The local models you are competing against are expected to perform as well as a random number generator. However, as a global model, you can pick up the signal present in ABC that a recession is coming seven years after the last one. This is, of course, not a true signal, but one created by chance. As you forecast ABC's weekly prices from 2006 to 2008, you recognize the Dot-com bubble and therefore anticipate a price drop. Therefore, your forecast of ABC will at least be in the right direction on the test set. For XYZ, you are also a random number generator as you do not recognize a signal.

When evaluating the models, you will score very well on the ABC part of the test set and, therefore, well overall. Confident that we have found the holy grail[b] *we send you out in the market to trade live only to see your losses accumulate until you end up like company ABC.*



a: ABC



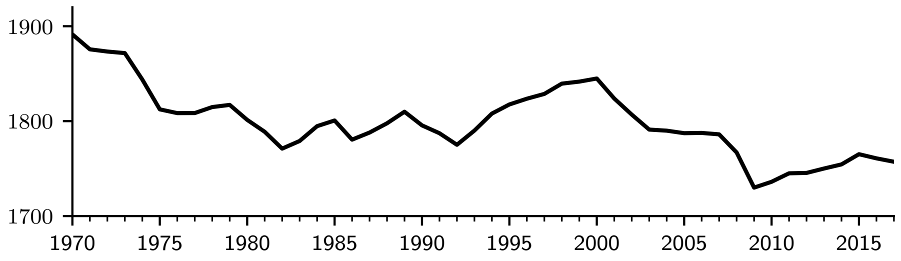
b: XYZ

Figure 4.2: Weekly stock prices for companies ABC and XYZ. The train-test split is marked with a horizontal bar.

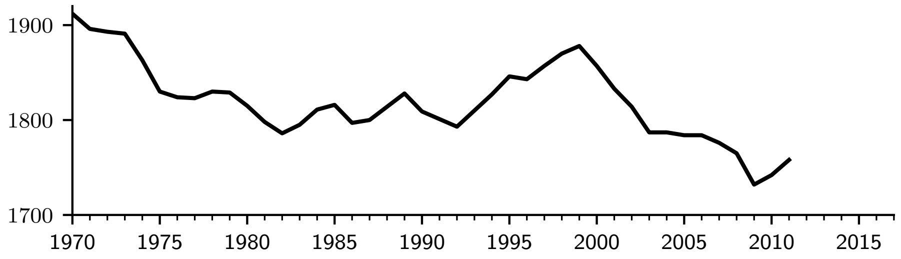
[a]https://en.wikipedia.org/wiki/Dot-com_bubble
[b]https://en.wikipedia.org/wiki/Holy_grail_distribution

A reasonable question is the extent of such related time series in the databases. By looking for related time series where one or more have been discontinued, we can start searching the databases. Example 5 shows an occasion from the FRED database. We found this example by a naive search after discontinued time series and something related to it. As ForeDeCk inherits from FRED and M4 is a subset of ForeDeCk, there might be time series that introduce temporal bias.

Example 5. *An example of potential candidates for implicit look-ahead bias are "Average Annual Hours Worked by Persons Engaged for the United States" in Figure 4.2a and "Average Annual Hours Worked per Employed Person in the United States (Discontinued)" in Figure 4.2b. The time series in Figure 4.2a has been cut before 1970, but Figure 4.2a has true start in 1970. These two time series are closely related, and the correlation can be seen from Figure 4.3. This illustrates that there are time series in the FRED dataset with the properties we have described earlier. To find this example, we did a relatively naive search after discontinued time series and chose time series suggested by the FRED website as a related series.*



a: Average Annual Hours Worked by Persons Engaged for United States.



b: Average Annual Hours Worked per Employed Person in the United States (Discontinued).

Figure 4.3: Two series from the FRED dataset that measure similar processes, but where one has been discontinued. In this case it is likely that the series was discontinued because the underlying process was already appropriately covered by the other.

4.3 Mitigating Temporal Bias Using Train-Test Split by Date

A possible solution to the problem is to do a train-test split by date rather than time. If we have metadata about the date of the observations in the time series, such a solution is feasible. Doing this will remove the possibility for information to flow from the test set. A challenge with this approach is that it can make the test set require a range of different forecast lengths. Multi-step approaches are not as suited for such a scenario as they will require one model per forecast length. We have earlier said that one-step models are needed for true global models, and these will not have issues with different forecast lengths.

We will introduce a bias towards newer time series. However, this is not necessarily bad as that is the type of series of interest in most applications. Forecasting is about the future, and this bias will perhaps make the test set more relevant to our applications.

Another approach is disregarding all time series that have been discontinued. The example we found in FRED was a result of looking for discontinued series. For a time dataset to be prone to look ahead bias from global models, it has to include discontinued series. Removing these will therefore remove the possibilities for look-ahead bias. However, there are two drawbacks to this approach. First, we are removing possibly large parts of the dataset. FRED contains historical economic data in addition to the data that is still being tracked. Second, and possibly more severe, is that we introduce survivorship bias to the dataset. This type of bias is towards things that have survived the ravages of time. It is, therefore, not the same distribution as the set we might want to forecast. Our test set can include companies that will not get through the next financial crisis and production from factories that will be shut down. Removing discontinued series will not be a satisfactory solution.

A test for the extent of the possible look-ahead bias for global models will be a good solution. That could help punish global models for exploiting the caveat. We do not have any specific solution for this problem but give an approach that could help enlighten the subject. The M4 dataset has metadata with dates. Splitting the dataset at a range of dates can give an idea of how important different periods in time are. As the M4 contains economic data, periods where the economy has changed can be significant. We do not perform these experiments in this project.

Chapter 5

Experimental Setup

In this chapter, we describe the setup used in our experiments. This includes the specific models used, training procedure, evaluation setup, and software and hardware used. The full implementation of the experiments is available in a GitHub repository[1].

We describe the setup for the following two experiments:

- Forecasting the M4 dataset using global and almost global MLPs. This is similar to the M4 competition, where we find the best model using a validation set.

- Zero-shot forecasting of the M4 dataset using pretrained models from the FRED dataset. This experiment also includes experiments with transfer learning.

The first three subsections of the chapter describe a setup that is similar for both experiments. Each experiment has its own section after that, where the details for that experiment are specified.

In some cases, we rely on implementations by other researchers. The code is then refactored for our experiments and pipeline. We are comparing our method to N-BEATS in the two forecasting experiments. The discussion of our experimental setup, therefore, aims to highlight key differences. Knowing this will better enable us to discuss which observations are the results of methodology and which are of optimization choices.

5.1 Software and Hardware

We use Python (Rossum, 1995) for the implementation of all experiments. Deep learning models are implemented using PyTorch (Paszke et al., 2019) and NumPy (Harris et al., 2020). We use a wrapper for the FRED API written in Python to access the database (Williams, 2015) and Pandas (pandas development team, 2020) for processing.

[1] https://github.com/erikdengerud/GPTime-Series

For comparison with N-BEATS, we rely on the authors' implementation and modify the code to fit our experiments and pipeline. N-Beats is available in a GitHub repository by the authors[2].

We trained all models using the NTNU IDUN computing cluster (Själander et al., 2019). The cluster has more than 70 nodes and 90 GPGPUs. Each node contains two Intel Xeon cores, at least 128 GB of main memory, and is connected to an Infiniband network. Half of the nodes are equipped with two or more Nvidia Tesla P100 or V100 GPGPUs. Idun's storage is provided by two storage arrays and a Lustre parallel distributed file system.

5.2 Training Procedure

In this section, we describe how the models are trained. We use the same procedure for all experiments unless otherwise specified. Properties of M4 and FRED are discussed in Chapter 4 and theory supporting the training procedure is explained in Chapter 2.

5.2.1 Datasets

We use the M4 and FRED datasets introduced in Chapter 4. Importantly, we create validation sets using the last forecasting horizon of the FRED dataset and the last horizon of the train part of the M4 dataset. The validation sets are used to find the hyperparameters of the models. Trial and error are necessary to get the models to learn at the beginning of training deep learning models. We use the validation sets for this trial and error.

Final models on the M4 dataset are trained using the full training part of the dataset. The test part of the M4 dataset is one additional horizon after the end of the train part of the dataset.

In the zero-shot experiments, we do not use the validation part of the M4 dataset. Instead, we use the validation part of FRED to get a pretrained model and then test this directly at the M4 test set.

5.2.2 Sampling of Time Series During Training

We sample training samples according to Algorithm 1. There is one sample from each time series in each epoch. The training samples in each epoch are different because we sample cut points randomly from the time series. Effectively, the dataset's total size is the number of time series multiplied by the number of valid cut points for each time series. For more stable and relevant training, we sample only from the end of the time series. Typically we sample from the last N horizons where the horizon is the forecasting horizons in the M4 competition.

The sampling algorithm we use is very similar to the one used by the N-BEATS authors. The main difference is that they sample for multi-step methods, which require a mask for the labels as well in order to prevent padded zeros from affecting the loss. The padding of zeros occurs when they sample too close to the endpoint such that the length of the output of the multi-step method is longer than the remainder to the endpoint.

[2]https://github.com/ElementAI/N-BEATS

Algorithm 1: Sampling algorithm

```
Result: One epoch of training samples, labels, and binary masks.
require a $c_{\text{frequency}}$ for each frequency, determining how close to the end of the
  training set we sample from.;
require the input window of the network.;
for each time series in the dataset do
    Sample a cut point in the last $c_{\text{frequency}}$ points of the time series. ;
    Use the cut point as a label and the observations before it as the sample. ;
    if the length of the sample is greater than the input window then
        Cut from the beginning of the sample so that it has the same length as the
          input window.
    else
        Zero pad the beginning of the sample so that it has the same length as the
          input window.
    end
    Create a binary mask of the length of the input window signaling whether a
      value is an observation, 1, or a padded zero, 0.;
end
```

5.2.3 Scaling

In general, deep learning models require appropriate scaling to converge properly. N-BEATS does not use scaling but reports similar performance scaling by the max value of the training samples. We experiment using the same scaling and compare the effects. To build true global pretrained models, it is necessary to scale in order to handle unseen data.

The zero-shot forecasting task differs in that there might be time series at a vastly different scale than in the source and target datasets. Oreshkin et al. (2020) find scaling crucial for the zero-shot task as the model suffers from catastrophic forgetting without. We adopt the same scaling in zero-shot experiments. All scaling in our experiments is done using maxscaling on the sample that is drawn.

5.2.4 Modeling Residuals

A key technique for both statistical and deep learning models is to work with residuals after some transformation, as these are often more convenient. Our investigation into N-BEATS revealed that modeling deviations from a naive forecast are easier than modeling the next value directly. We use the same initialization of the forecast in our experiments.

From our experiments on N-BEATS, we also discovered that a naive seasonal initialization of the forecast was beneficial for frequencies with strong seasonal components like monthly series. This initialization can also be useful for true global models since it alleviates some of the problems with modeling seasonality. We experiment using both naive and seasonal naive initialization of the forecast.

5.2.5 Ensembling

Our investigation into N-BEATS revealed that it is an ensemble of several weaker models. Most of the top methods in the M4 competition also use ensembles or combinations of methods. Most prominent is perhaps the second place solution that uses a boosted tree model as a meta learner to determine the weights of an ensemble of models based on statistics from the time series it forecasts. It is reasonable that we also utilize an ensemble to be comparable.

Traditional ensembles are not necessarily the way forward for global pretrained models. The ensemble in N-BEATS consists of 180 models that have around 6×25M parameters each, resulting in more than 2.7B parameters for the full method. There is also overhead in the forecasting as there are 180 models that need to forecast. We do not use the larger ensemble of N-BEATS with 27B parameters.

Stochastic weight averaging (SWA) can create an ensemble of models from a single training run. Instead of rerunning the training from different random seeds, which is done to increase the ensemble size from 18 to 180 in N-BEATS, we can use SWA to get an ensemble of models in weight space rather than output space. This will drastically reduce the compute recovered to get a comparable ensemble. Ensemble members originating from SWA or different random seeds are by construction different from the ensemble members that differ in loss function or input window length. It is only the randomness in the initialization and training procedure that differs the random seed ensemble members. Ensemble members with different loss functions or input windows change the function we optimize or the model structure. Unfortunately, we do not apply SWA in our experiments as the Pytorch we used in our experiments did not have SWA implemented, and we did not find time to use a newer version. Initial experiments on another computing resource indicated that it could give similar performance-enhancing as using models with different random seeds.

5.2.6 Learning Rates

The learning rate is perhaps the hyperparameter that is most closely related to the performance of deep learning models. A too large hyperparameter can overshoot minimas, while a too small learning rate can get stuck in local minimas. Therefore, an adaptive learning rate is often used to first find the area of the wight space of a minima with a larger learning rate before reducing the learning rate to converge to the desired minima.

N-BEATS uses a simple, adapted learning rate schedule. When investigating N-BEATS, we found that this scheme was not aggressive enough. Therefore, we experiment using the learning schedules described in Section 2.1.4; multiplicative, cosine annealing and reducing on plateaus.

5.2.7 Dropout

We use dropout to reduce the risk of overfitting. Dropout is also, in a sense, a way of ensembling. By zeroing random nodes at training time, we force the model to learn different paths through the network. When all nodes are active at test time, we are ensembling all the paths to create a stronger model that generalizes better. Dropout is not used in

N-BEATS but generally results in better generalization. We apply a dropout of 0.2 in our experiments.

5.2.8 Gradient Clipping

Deep learning sometimes suffers from the gradient vanishing or exploding. Gradient clipping mitigates exploding gradients by clipping them if they surpass a value, 1.0 in our case. While gradient vanishing and explosion are mostly a challenge in recurrent architectures, we apply it as a safeguarding measure.

5.2.9 Training Time

Early stopping can be used to determine an appropriate time to stop training. Using a separate validation set, we can monitor the loss on the validation set to ensure we stop training before we overfit. The validation set will be different in our three experiments. In our M4 experiment, we use the last horizon of the training set as this best mimics the task. We divide FRED into a test and validation set on a time series level for our zero-shot experiments. This means that the models see different time series at train and test time and mimics testing on M4.

5.3 Model Specification

Our model in all experiments is a fully connected MLP with residual and skip connections and is displayed in Figure 5.1. This is a simple architecture that allows us to test the global properties of deep learning models for time series. We will use global models and almost global models. The almost global models are models trained for each frequency. The model parameters are the input window length, number of hidden units in each layer, number of layers, forecast initialization, dropout, and how often to apply skip and residual connections. In addition, some parameters affect the sampling in training. These are the loss function, learning rate schedule, weight initialization, and scaling.

5.3.1 One-step and Multi-step Models

We restrict our experiments to one-step methods. The primary motivation for this is that it is more general and also necessary for global models. Our analysis is aimed towards true global models, and we prioritize analysis required for global models. Experiments with N-BEATS also show that one-step models can reach comparable accuracy given seasonal initialization.

5.4 Forecasting M4

The first experiment mimics competing in the M4 competition with a global model. We train a global model on M4 to see how it would perform in the competition and against

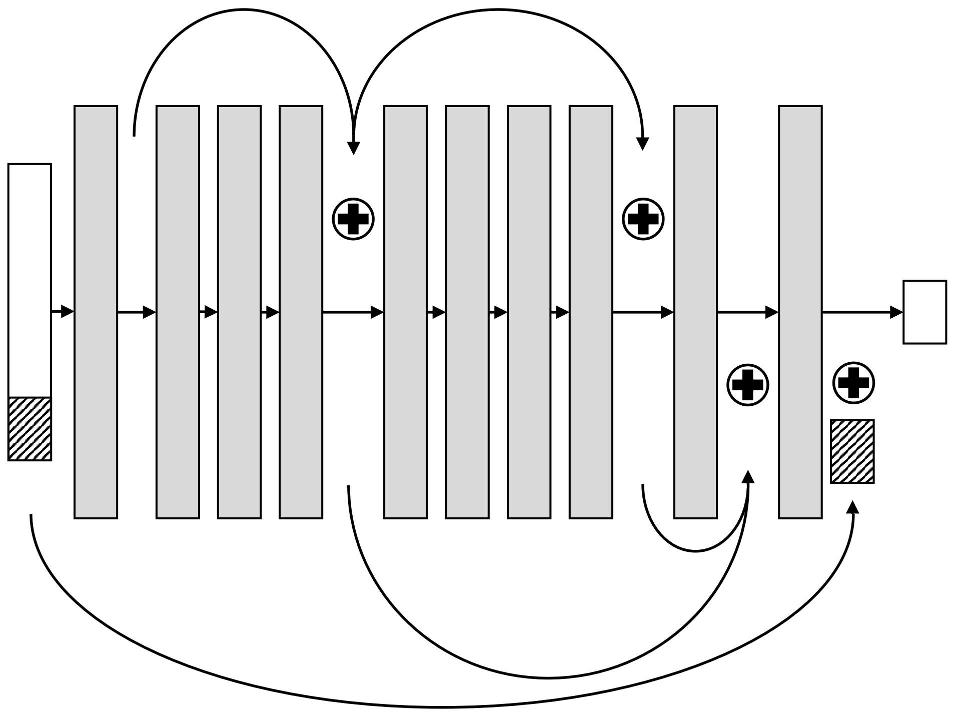


Figure 5.1: The baseline model used in the experiments. There are ten layers with 512 hidden units in each. The last observation in the input, indicated by the hatched block, is used to initialize the forecast. Residual and skip connections are added from every fourth layer. The first residual connection is initialized after the first layer. Due to inattentive implementation, the first block has only three layers. Subsequent blocks all have four layers. Finally, the last layer uses the sum of the skip connections to forecasts the residual of the naive forecast.

N-BEATS. Our main hypothesis is that we can achieve comparable results using a single MLP and that it is the global part of N-BEATS that puts it in front of other entries.

We will define a baseline model that we can then alter in the layer and width direction to find the best model. It is not feasible to test all configurations of width and depth. Our approach resembles looking in the direction of the partial derivative in that we keep all parameters constant except the one we are experimenting with. There might be a model outside of the models we test with better performance than the one we arrive at, but our approach is more feasible.

We choose a baseline model that has 10 layers with 512 hidden units and ReLU activation functions. We use skip and residual connections every fourth layer. To regularize the network, we use dropout with a probability of 0.2. The data is not scaled, but the forecast is initialized using a naive forecast. We train the network using a batch size of 1024 and the Adam optimizer. We use a validation set with the last horizon of the training set of each frequency to evaluate models. The validation set is also used for early stopping and reducing the learning rate. Global models are trained for at most 200 epochs with early stopping set to 30 epochs. The learning rate starts at 0.001 and is reduced by a factor of 0.1 if the validation error has not decreased for 10 epochs. The learning rate is decreased until it reaches $1E-8$. Almost global models are trained for at most approximately $200C$, where C is a scaling factor determined by $\frac{N}{\max(1024, N_f)} \times 200$, where N_f is the number of time series of the frequency. Early stopping tenacity and the patience for adjusting the learning rate are scaled using the same factor. We found it necessary to increase the number of epochs for the frequencies with fewer time series to get competitive results. This is likely due to fewer parameter updates in each epoch, which is perhaps a better measure of how long we train.

We use a restricted ensemble to reduce the overall number of models to fit. Based on N-BEATS analysis in Chapter 2.3 we use only one loss function in the ensemble. We choose SMAPE as it is faster to compute than MASE and closer related to the overall score than MAPE. Local models have lookbacks 2 through 7. Global models do not have a template we can follow, and we have to find our own. The frequency models with the shortest input window lengths are the yearly models, ranging from 12 to 42. The longest are the hourly models ranging from 96 to 336. Most of the frequency models in the almost global ensemble have input windows significantly shorter than the longest hourly model. We will therefore focus the input window lengths of our ensemble members in the range 20-200. For the baseline global model, we choose input windows $[30, 60, 90, 120, 150, 180]$

5.5 Zero-Shot Forecasting

We use FRED as a source dataset to train a global model that we later use for zero-shot forecasting on the M4 dataset. Models on FRED are trained using a validation set to determine how long we should train the model. We use the same models we find to be best in the M4 experiments. The larger FRED dataset could potentially allow for more complex models without overfitting. We, therefore, test three model sizes for the global model in the zero-shot experiments. However, we skip the same analysis for the almost global model due to the many submodels needed. To keep the total number of models reasonable, we only use ensemble members with SMAPE loss.

In the transfer learning part, we perform three experiments,

1. Retraining the output layer and the last layer while freezing the rest of the network.
2. A two-step retraining. The first step is as in the first experiment. The second part is to retrain the full model on the new dataset after the last layers have been trained on the new dataset.
3. Using the pretrained weights as initialization for a new network.

In these experiments, we use the pretrained models from the zero-shot experiments. We use the models that are trained on the full FRED dataset. For the almost global models, we skip the hourly segment as it is not present in FRED. We use the same training schedule as before but reduce the number of epochs by half when finetuning the last layer and a quarter when finetuning the second step of the two-step finetuning. When we retrain the last layers, these are randomly initialized.

Frequency	Seasonality
Yearly	1
Quarterly	4
Monthly	12
Weekly	1
Daily	1
Hourly	24

Table 5.1: Seasonalities used in MASE in the M4 competition.

5.6 Performance Metrics

For the forecasting experiments we use the same performance metrics as in the M4 competition. These are the Mean Absolute Scaled Error and Symmetric Mean Absolute Percentage Error,

$$\text{MASE} = \frac{1}{h} \frac{\sum_{t=n+1}^{n+h} |y_t - \hat{y}_t|}{\frac{1}{n-m} \sum_{t=m+1}^{n} |y_t - y_{t-m}|}, \tag{5.1}$$

$$\text{SMAPE} = \frac{2}{h} \sum_{t=n+1}^{n+h} \frac{|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|} \times 100\%. \tag{5.2}$$

Here, h is the forecasting horizon, n is the number of observations in the train set, and m is the seasonality of the time series. MASE is scaled intuitively against a the error of a naive seasonal forecast on the train set. The period for the frequencies are defined in Table 5.1. We can also see that SMAPE is bounded on $[0, 200]$.

The overall score in the M4 competition is a weighted average of MASE and SMAPE scaled by how well a naive seasonal forecast adjusted for trend (NAIVE2) would perform. We therefore also use the Overall Weighted Average score (OWA),

$$\text{OWA} = \frac{1}{2} \frac{\text{MASE}}{\text{MASE}_{\text{NAIVE2}}} + \frac{1}{2} \frac{\text{SMAPE}}{\text{SMAPE}_{\text{NAIVE2}}}. \tag{5.3}$$

The OWA score is easier to understand because it is a comparing score. Naive forecasters are easy to understand and an important benchmark. Working with the OWA score gives quick feedback because of the intuitive scaling. It is easier to relate to a model that we know how works than an error metric that can vary across frequencies and datasets.

Chapter 6

Experiments and Discussion

In this chapter, we perform two experiments. First, we train global and almost global MLPs on the M4 dataset. Second, we look into zero-shot and transfer learning abilities of global and almost global models trained on the FRED dataset. The theory supporting these experiments can be found in Chapter 2, a discussion about the datasets used can be found in Chapter 4, and the experimental setup can be found in Chapter 5.

6.1 Global Models in the M4 Competition

In this section, we test how global models could have performed in the M4 competition. Using only the training data, we emulate the process of finding the best architecture and training procedure. Our final models are then compared to other methods using the test set. The main findings in this section are,

- A global MLP can be trained to achieve a total overall score of 0.833, which would have placed 2nd in the M4 forecasting competition. Encoding the frequency as a one-hot encoded vector and scaling the data is central in making the global model perform well.
- An almost global model is better than the global model. We get a total overall score of 0.826, which is not very far ahead of the global model.
- Both the global and almost global models are not performing as well as N-BEATS. However, looking into the models' ensembles, we find that all the global and almost global models we train in this section have better-performing ensemble members than N-BEATS.
- Using a global model, we can drastically reduce the number of parameters compared to N-BEATS while still having a performance at a level of the best-performing entries in the M4 competition.

We start by finding parameters for the width and depth of the models. For the global model, we also find the ensemble members we use in the rest of the experiments. Second, we look into how we can best train the models. We tune the learning rate and incorporate frequency information in the models. Next, we report final values on the test set of the global and almost global models. Lastly, we look into the models' ensemble members and compare them to the N-BEATS ensemble members.

6.1.1 Model Architecture

Our first experiments are related to model architecture. The number of layers and the number of hidden units is not given, and the best choice varies from dataset to dataset. We perform these experiments on the almost global and the global models.

One weakness with our approach is that we determine when to stop based on the validation set that we are also testing. This can make the models look better than they will at test time. Another serious flaw is that we use the naive2 scores from the test set to scale the MASE and SMAPE scores. This will introduce look-ahead bias. We realized this later in the project where it was not feasible to rerun the experiments. However, it is not unreasonable to assume that the relative performance of the naive2 forecaster on the different frequencies would be similar on the validation and test sets. If this is true, we will only be scaling wrong, which is not that serious. The issue is serious if the naive2 forecaster performs much better on, e.g., monthly data on the test set than on the validation set. We will then choose models that prioritize doing well on monthly data even though we should not have that information based on the validation set.

6.1.2 Global Architecture

Table 6.1 show the overall score for each frequency and in total. The model with 1024 hidden units is the best performer. Interestingly, 256 units also perform well, and this is a model with a lot fewer parameters. We see the most variety in performance in the hourly segment. The hourly segment is small and is a minor contributor to the total score. However, when the overall score for the hourly segments gets very large, it will be a factor. It is worth noting that all models struggle to forecast the hourly series accurately. As the same model is used for all time series, the poor performance likely is because the hourly series are different from the rest. Since the proportion of hourly series is so small, it will make sense for the network to learn how to predict the other frequencies.

The performance as we vary the depth of the network is seen in Table 6.2. While the model with 5 layers performs best in most segments, it is the model with 10 layers that has the best total score. Again it is the hourly segment that varies the most and is hard to forecast. The hourly segment is what prevents the 5 layer model from being the best model. Interestingly, the model with 50 layers is the only one that is close to forecast hourly data. However, poor performance on all other frequencies prevents the model from being competitive.

We continue with a model with 10 layers and 1024 hidden units in each layer based on our experiments. However, we are not far from using a model with 5 layers and 256 hidden units. The first model is almost 40 times larger than the second measured in the number of parameters. If we believe we are risking overfitting, we should shift to the smaller model.

Table 6.1: Overall scores of the global model on the validation set as a function of the number of hidden units in each layer. We use the baseline model with 10 layers in the experiment and only change the number of hidden units.

Hidden units	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
64	0.863	0.999	0.912	0.931	1.095	29.071	1.067
128	0.862	1.001	0.916	0.933	1.091	22.344	1.033
256	0.859	0.976	0.874	0.910	1.105	12.467	0.962
512	**0.858**	0.967	0.864	0.911	1.099	14.199	0.964
1024	0.860	0.961	**0.859**	**0.909**	**1.080**	**13.310**	**0.956**
2048	0.864	**0.957**	0.868	0.912	1.081	20.973	1.000

Table 6.2: Overall scores of the global model on the validation set as a function of the number of layers. We use the baseline model with 512 hidden units in the experiment and only change the number of layers. The model with 10 layers is equivalent to the model with 512 hidden units in Table 6.1, but with different random initializations.

Layers	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
1	1.047	1.047	1.077	1.095	1.168	15.852	1.146
5	0.861	**0.952**	**0.855**	**0.894**	1.085	24.879	1.014
10	**0.856**	0.964	0.865	0.911	1.087	14.854	**0.966**
15	0.857	0.974	0.869	0.919	**1.082**	19.822	0.996
20	0.861	0.994	0.895	0.920	1.089	11.601	0.969
30	0.871	1.059	1.008	0.984	1.090	7.877	1.005
50	1.236	1.087	1.041	1.022	1.050	**3.304**	1.136

Based on the analysis in Chapter 2.2, the smaller model will have a tighter generalization bound.

Input Window Lengths for Global Models

We do not have a template for building ensembles for global models like we do for almost global models. Figure 6.1 show the total overall score for a global model as a function of input window length using the baseline model. The input windows were chosen from the range where most of the almost global models live. The models with the shortest input windows struggle to show consistent performance. Once the input window goes above 70, the performance is stable. According to Goodfellow et al. (2016), an ensemble should have a diverse set of members that make uncorrelated errors. Therefore, we will choose members with input windows distributed in the full range from 70 to 300 rather than finding the best performers in the range. That will hopefully introduce more diversity to the ensemble. We omit the shortest input windows as it seems they are a lot less accurate. The ensemble members we will use in the rest of the experiments will have input window lengths $[70, 110, 150, 190, 230, 270]$.

Table 6.3 show the two candidate models discussed as a result of the width and depth experiments on the new ensemble together with the baseline model. The largest model is the best in terms of the total overall score and the one we will use going forward. It

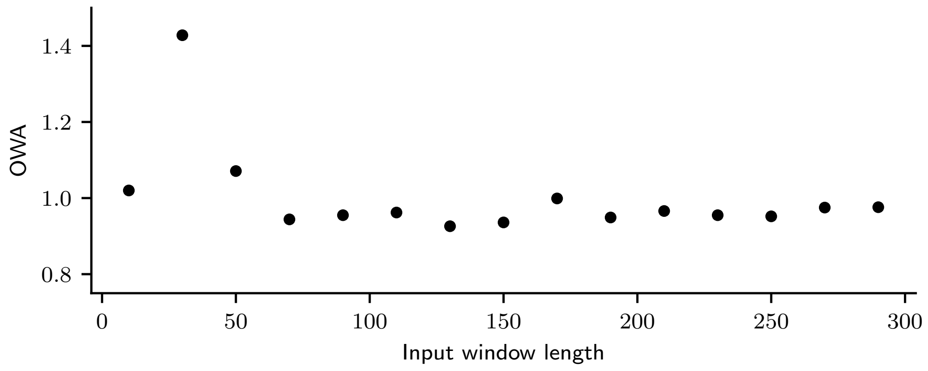


Figure 6.1: Total overall score of candidate ensemble members as a function of input window length for the baseline global model.

Table 6.3: Overall score of the global model on the validation set for candidate architectures. The model with 10 layers with 512 hidden units is the baseline model, but now with the new ensemble members. W stands for width and D for depth in the first column.

W - D	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
256 - 5	**0.858**	**0.966**	0.878	**0.890**	1.094	19.885	0.998
512 - 10	**0.858**	0.977	0.880	0.918	1.100	14.473	0.973
1024 - 10	0.860	**0.966**	**0.868**	0.928	**1.086**	**11.215**	**0.950**

emerges as the best model because it models the monthly segment better. The monthly segment is almost half of the dataset and, therefore, the most important.

It is worth noting that choosing the best models for an ensemble may not be the best possible ensemble. The best ensemble will have well-performing members that are uncorrelated. This can motivate picking worse performing models if they are uncorrelated to the rest of the ensemble. We do not look into the correlation of the members in our experiment, but note that the baseline model with our first ensemble in Table 6.2 and 6.1 perform better than the baseline model with the new ensemble in Table 6.6.

6.1.3 Almost Global Architecture

Table 6.4 shows the almost global model's overall score as a function of the number of hidden units in each layer. There is less variability in the total score than what we saw for the global model. As each frequency is given its own model, there is room to model that specific frequency without compromising with modeling other frequencies. Again it is the model with 1024 hidden units that has the best total overall score. It beats the model with 512 units by modeling the monthly segment slightly better. With wider layers, the model is better able to model the shorter frequencies. Table 6.4 show that the weekly and hourly data score is monotonically decreasing as the width increases.

The overall scores as we vary the depth of the network are shown in Table 6.5. Models

Table 6.4: Overall score for the almost global model on the validation set as a function of the number of hidden units in each layer.

Hidden units	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
64	0.868	0.939	0.855	0.911	**1.053**	0.771	0.887
128	0.861	0.934	0.844	0.856	1.055	0.764	0.880
256	0.860	0.928	0.836	0.840	1.055	0.734	0.875
512	**0.852**	**0.921**	0.830	0.824	1.058	0.724	0.869
1024	0.853	0.922	**0.827**	0.810	1.059	0.718	**0.868**
2048	0.857	0.924	0.829	**0.785**	1.060	**0.662**	0.870

Table 6.5: Overall score of the almost global model on the validation set as a function of the number of layers.

Layers	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
1	1.010	0.980	0.944	0.950	1.055	0.821	0.985
5	0.858	**0.919**	**0.830**	**0.775**	1.060	**0.628**	**0.870**
10	**0.853**	0.923	0.831	0.813	1.057	0.713	**0.870**
15	0.856	0.927	0.834	0.831	1.057	0.775	0.873
20	0.862	0.933	0.836	0.882	1.055	0.867	0.877
30	0.882	0.942	0.857	0.900	1.051	2.911	0.905
50	0.983	1.071	1.041	1.016	**1.048**	3.397	1.036

with between 5 and 20 layers all perform well. The models with 5 and 10 layers are the best performers. In the global model depth experiment, we saw that the hourly score decreased as the network got deeper. We observe the opposite for the almost global models where the hourly segment performs worse as we increase the depth.

Again, we are left with a small subset of possible best models that vary a lot in the total number of parameters. Table 6.6 show the overall scores of the models on the validation set. It is the largest model that performs the best also here. The models are all very similar in performance, but it is again better modeling of the monthly segment that puts the largest model ahead.

Table 6.6: Overall score of the almost global model on the validation set for candidate architectures. The model with 10 layers with 512 hidden units is the baseline model. W and D in the first columns indicate the width and depth of the model.

W - D	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
512 -5	0.858	**0.919**	0.830	**0.775**	1.060	0.628	0.870
512 - 10	**0.852**	0.921	0.830	0.824	**1.058**	0.724	0.869
1024 - 5	0.861	0.920	0.829	0.780	**1.058**	**0.615**	0.871
1024 - 10	0.853	0.922	**0.827**	0.810	1.059	0.718	**0.868**

Table 6.7: Overall score of the global model with 10 layers and 1024 hidden units on the validation set for different learning rate schedules.

Schedule	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
Plateau	**0.860**	0.966	0.868	0.928	1.086	**11.215**	**0.950**
3-step	0.944	1.363	2.667	3.292	5.782	11.931	1.861
10-step	0.861	**0.955**	**0.862**	0.918	**1.073**	20.741	0.995
Cosine	**0.860**	0.968	0.891	**0.909**	1.093	149.540	1.675

6.1.4 Learning Rate

The learning schedule we use affects the final score of a model to a large extent. N-BEATS uses a conservative learning schedule, perhaps to prevent overfitting. We experiment with different levels of aggression when tuning the learning rate. First, we use the conservative N-BEATS schedule, which reduces the learning rate by 0.5 three times. Such learning schedules are called stepwise learning schedules. Second, we use a stepwise schedule that is more aggressive. This schedule reduces the learning rate by a factor of 0.5 ten times. Reducing the learning rate ten times will yield similar learning rates as when we reduce on plateaus, only at different periods of training. When we test on the test set, it will be easier to implement the stepwise schedule. Lastly, we use a cosine annealing schedule introduced in Chapter 5.

Table 6.7 show the total overall scores for the three learning rate schedules on the validation set together with the previously used schedule. None of the schedules are better than reducing on a plateau. Still, the aggressive stepwise schedule outperforms plateau in most segments except yearly, where it is close, and hourly where it is catastrophic. The same goes for the cosine annealing schedule, where the hourly segment's performance is even worse. None of the segments show competitive scores with the three step schedule. This suggests that an aggressive scheme is important.

The learning rate schedules on the almost global model can be seen in Table 6.8. The ten step schedule is better than the plateau schedule for this model. We saw that the ten step schedule was competitive on all segments except the hourly on the global model. When the model has a separate network for each frequency, it can easily model the hourly segment's characteristics.

N-BEATS uses the three step schedule and gets very good results. One reason can be that the deep network requires regularizing in the form of a higher learning rate. Another can be that they train the network for longer. Training the network for many epochs can possibly work without risking overfitting if we keep the learning rate high. However, N-BEATS can perhaps also benefit from a more aggressive learning rate schedule.

6.1.5 Initialization of the Forecast

In Section 2.3 we saw that initialization of the forecast was an important step to make the almost global one-step MLP work well. We also improved on the performance by using a naive seasonal initialization. Naive seasonal initialization can also be done for the global model. This could possibly help model the hourly segment, which seems to be hard for

Table 6.8: Overall score of the almost global model on the validation set for different learning rate schedules. We use the model with 10 layers and 1024 hidden units.

Schedule	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
Plateau	0.853	**0.922**	0.827	0.810	**1.059**	0.718	0.868
3-step	1.070	0.997	1.462	3.212	2.012	1.999	1.225
10-step	**0.849**	0.926	**0.825**	**0.784**	1.087	**0.580**	**0.866**
Cosine	1.081	0.936	0.829	1.615	2.033	0.644	1.007

Table 6.9: Overall scores of the global model on the validation set for naive, seasonal naive and no initialization. We use models with 10 layers and 1024 hidden units. All models are trained with the plateau schedule.

Initialization	Yearly	Qrtrly	Monthly	Weekly	Daily	Hourly	Global
None	67.291	349.858	2E5	1.6E4	3.5E4	2.0E14	1.0E12
Naive	**0.860**	0.966	**0.868**	**0.928**	1.086	11.215	**0.950**
S-naive	1.486	1.530	1.113	46.818	58.984	**10.393**	5.078

global models. So far, the global models struggle with the segments that typically show strong seasonality.

Table 6.9 shows the results from using a naive seasonal initialization in the global model. Unfortunately, the naive seasonal initialization of the forecast drastically reduces the performance of the model. One bright spot is that the model is better able to model the hourly segment. However, compared to the hourly score of the almost global models we have tested, the score is not good at all. All segments struggle to learn anything, with the daily segment coming in last. As we also saw in Section 2.3, using no initialization prevents the network from learning.

The local models show improvement in all segments except the monthly as seen in Table 6.10. However, the improvement in most segments is minor, resulting in a total overall score that is very similar. The naive and seasonal naive models can be useful as part of an ensemble as they are different but show similar performance. Chances are they are less correlated than, e.g., models with different random seeds. We will include both in our final ensemble for almost global models.

In Chapter 2.3 we saw a considerable improvement using the naive seasonal initialization. Why we do not see the same improvement in this experiment is not clear. We are getting better results with naive initialization than in Chapter 2.3. One possibility can be that we have already trained the network in a better and more stable fashion, and we are therefore not seeing a better result. Another likely reason is differences in the training procedure.

6.1.6 Encoding Frequency

The global model was not able to learn anything when we used the seasonal initialization. We will use the frequency information by encoding the frequency as a one-hot vector and concatenate this to the input. This will allow the model to change its behavior base on

Table 6.10: Overall scores of the almost global model on the validation set for naive, seasonal naive and no initialization. We use models with 10 layers and 1024 hidden units. All models are trained with the plateau schedule.

Initialization	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
None	71.269	378.527	5.9E3	3.2E3	8.3E3	8.5E10	4.4E8
Naive	0.853	0.922	**0.827**	0.810	1.059	0.718	**0.868**
S-naive	**0.851**	**0.921**	0.830	**0.793**	**1.057**	**0.550**	**0.867**

Table 6.11: Overall score of the global model on the validation set with and without encoding the frequency. All models are trained with the plateau schedule. The baseline model is the same run as in Table 6.7

Model	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
Baseline	**0.860**	0.966	**0.868**	0.928	**1.086**	11.215	0.950
Encoding	**0.860**	**0.962**	0.869	**0.922**	1.088	**6.911**	**0.927**

the frequency. Our model is a fully connected feedforward network, which means that all input variables are connected to each hidden unit in the first layer. As long as the one-hot vector is concatenated to the input vector in the same way, it does not matter where it is concatenated to the input.

Table 6.11 show the model with and without encoding the frequency. The total overall score improves, mainly due to better modeling of the hourly segment. As we have seen so far, it is the hourly segment the global model has trouble modeling. In all other segments, the baseline model and the model with encoded frequencies are comparable. In our final model, we will encode the frequencies.

In the context of global pretrained models, we can use the same encoding. If the frequency is unknown at test time, we have a one-hot encoding for unknown data. Typically we reserve the zero-vector for unknown series as that will zero out all weight coming from the part of the input signaling frequencies.

We could have encoded the specific domain of the time series as well. However, this would have been unfair to the almost global models and made the model very specific to the dataset.

6.1.7 Residual and Skip Connections

Lastly, we will check whether the residual and skip connections are useful for the network. Our current model is not that deep, and it is possible that the residual and skip connections are not necessary. We use the model with ten layers and 1024 hidden units with residual and skip connections and compare this to the same network without residual and skip connections. The global model in this experiment does not encode the frequencies, and the almost global model does not use naive seasonal initialization of the forecast.

Table 6.12 shows the results when removing the residual and skip connections for the global model. Unfortunately for our experimental setup, but fortunately for model simplicity, the model without residual and skip connections is the best. It is better in all segments, and the overall score is comparable to when we encode the frequencies.

Table 6.12: Overall score of the global model on the validation set with and without residual and skip connections. All models are trained with the plateau schedule. The baseline model is the same run as in Table 6.7. The None model is the model without any skip or residual connections.

Model	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
Baseline	0.860	0.966	0.868	0.928	1.086	11.215	0.950
None	**0.858**	**0.957**	**0.866**	**0.916**	**1.077**	**7.591**	**0.928**

Table 6.13: Overall score of the almost global model on the validation set with and without residual and skip connections. All models are trained with the plateau schedule. The baseline model is the same run as in Table 6.8. The None model is the model without any skip or residual connections.

Model	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
Baseline	**0.853**	**0.922**	**0.827**	**0.810**	1.059	0.718	**0.868**
None	0.859	0.924	0.835	0.854	**1.056**	**0.717**	0.874

We observe the opposite effect for the almost global model, as shown in Table 6.13. The model using residual and skip connections is better in most segments and comparable when it is not better.

The difference we observe in global and almost global models is very interesting but hard to explain. There is a possibility that the different origins in that the almost global model only has to focus on one frequency, but that the global model has to understand all frequencies. The more complex task can perhaps benefit from less intervention in the network.

Residual and skip connections only add shortcuts to the network, and it is hard to explain why it should harm the network. As that was the basis for the experimental design, we started with a model using residual and skip connections. This baseline model was the one we found the best depth and width for. Redoing all those experiments will not be sustainable, and we will therefore not redo them. Instead, we will assume that the behavior is similar and that the model is close to what we would have ended up doing all experiments without residual and skip connections.

As a final selection step, we will confirm that encoding the frequencies is also beneficial when removing the residual and skip connections. Table 6.14 show the baseline model, the baseline with encoding, the model without residual and skip connections, and the model without residual and skip connections with encoding. The last model has the best total overall score, mainly because it models the hourly segment better. The two models without residual and skip connections are very similar, as are the two with. It seems like the encoding of the frequencies mainly helps distinguish the hourly time series from the rest to be modeled better.

6.1.8 Full Ensembles on the Test Set

We will use the same three loss functions as in N-BEATS, MASE, SMAPE, and MAPE for the full ensembles. The almost global model will have lookbacks 2 through 7 and use both naive and seasonal naive initialization. The global model will have input window lengths $[70, 110, 150, 190, 230, 270]$. We will use residual and skip connections in the al-

Table 6.14: Overall score of the global model on the validation set with and without residual and skip connections and encoding of frequencies. All models are trained with the plateau schedule. The baseline model is the same run as in Table 6.8, and the encoding model the same run as in Table 6.11.

Model	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
Baseline	0.860	0.966	0.868	0.928	1.086	11.215	0.950
Encoding	0.860	0.962	0.869	0.922	1.088	6.911	0.927
None	**0.858**	**0.957**	**0.866**	0.916	**1.077**	7.591	0.928
None + enc	0.859	0.958	0.867	**0.911**	1.078	**5.754**	**0.919**

most global model, but not in the global model. The global model will use the encoding of the frequencies. All models will be trained using the ten step learning schedule introduced. We have seen that this schedule behaved similarly to the plateau schedule for the almost global model. Even though the schedule was not as good as the plateau schedule for the global model, we will use it as it gives similar results in most segments, is easier to implement, and we do not know how the learning schedules behave for other loss functions. Lastly, we report performance using scaled and non-scaled inputs. N-BEATS is better without scaling, but scaling is often necessary for transfer learning and is therefore included.

Creating an ensemble requires models that perform well and are uncorrelated. We have seen that, e.g., the global model was perhaps as good using the original members as our selected members even though they overall had a worse average performance. The naive and seasonal naive initialization are likely good complementary members in an ensemble as they are different but comparable in performance. Using both in the ensemble will lead to a larger ensemble than the N-BEATS model we compare against and the global model.

Table 6.15 shows the global model results on the test set with and without scaling. First, the scaled model score better than the model without. We do not see the same improvement for the almost global model. This observation suggests that scaling allows for better learning across frequencies. If that is that information is better shared across frequencies, or something else is harder to determine. Scaling removes information about the time series, but for some machine learning models, it is necessary.

Second, by looking at individual ensemble member scores, we observe that the ensemble members with shorter input window lengths score better. We also observed in our experiment finding good ensemble members that the original ensemble scored better than the one we picked from our analysis. Contrary to the experiment with input window lengths in Figure 6.1, we observe that the performance is best for the shortest window. We therefore include an ensemble using shorter input windows, $[20, 30, 40, 50, 60, 70]$. We refer to this as the alternative ensemble (AE). As seen in Table 6.15, the model using scaled inputs and the shorter ensemble scores better. The score would have been a second place in the M4 competition. To be complete, the shorter ensemble choice should be based on the validation set, and we acknowledge this weakness.

Table 6.16 shows the overall scores on the test set for the almost global model with and without scaling. The model performs better without scaling, but the two are comparable. The score would have been a second place in the M4 competition, close to the first place

Table 6.15: Overall scores of the global model on the test set with and without scaling, the scaled alternative ensemble (AE), and the scaled alternative ensemble with less aggressive learning rate tuning (-LR).

Model	Yearly	Qrtrly	Monthly	Weekly	Daily	Hourly	Global
Base	0.805	0.875	0.891	0.935	0.953	2.622	0.866
Scale	0.785	0.862	0.907	0.931	0.937	**1.368**	0.855
Scale+AE	0.777	**0.849**	**0.864**	**0.934**	**0.934**	1.586	**0.833**
Scale+AE-LR	**0.773**	0.863	0.933	0.917	0.944	2.193	0.861

solution. We use a larger ensemble than in our other experiments as we also use models with naive seasonal initialization. With the smaller ensemble using only naive or only seasonal naive, the unscaled model's scores would have been 0.831 and 0.828.

The results in Table 6.16 are far from the strong 0.812 score of the one-step almost global MLP with seasonal initialization we saw in Chapter 2.3. We would have expected these two models to score about the same. There are some possible differences we will highlight next.

First, we did not see the same benefit from using the naive seasonal initialization in this chapter as we did in Chapter 2.3. Perhaps this is an indication that there is something different in the models. The model without seasonal initialization in Chapter 2.3 did not score very well. The model using seasonal initialization was comparable to the one with naive in this chapter.

Second, there might be differences in the training procedure. Our algorithm is based on showing each time series once every epoch. N-BEATS sample random batches and do not treat epochs. The cut points can also be different when sampling during training. The authors tune a hyperparameter L_h for each frequency used to decide how close to the end of the training set the algorithm should sample. We called this cut point parameter c_f and set it to the last horizon. This choice can result in differences in the training samples the models see.

How long and how aggressive the models are trained also differ. We found a more aggressive learning rate schedule to be beneficial. We used early stop to decide how long we should train the models. When training the final model, we used a set number of epochs for all models as we saw that it was similar to the early stopping schedule. This is similar to what is done when training N-BEATS, but the duration we train for might differ. This, together with a different learning schedule, can also add to the gap in performance.

There are two useful experiments using N-BEATS that we have not done. First, we could have trained N-BEATS using a more aggressive learning rate schedule. The learning rate showed to be an important part of training global models successfully. Second, we could have tried to make N-BEATS into a truly global model. However, this would have required a more extensive refactoring of the code from the N-BEATS paper and was therefore not included in this thesis.

Table 6.16: Overall scores of the almost global model on the test set with and without scaling.

Model	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly	Global
Base	0.787	**0.834**	**0.853**	**0.729**	**0.977**	0.462	**0.826**
Scale	**0.779**	0.835	0.863	0.734	**0.977**	**0.443**	0.827

A Look Into the Ensembles

The total overall scores of the ensemble members of the models discussed in this chapter are shown in Figure 6.2. We find that all the ensemble members for our best global and almost global model have a better total overall score than all the members of the N-BEATS ensemble. This is a curious find as both N-BEATS ensembles, and the MLP trained using the N-BEATS training procedure have a better total overall score than the models in this chapter. The difference of the ensemble members is highly significant and had the ensembles had the same properties, we would have expected the ensemble with better members to score better. N-BEATS is a qualitatively different model and can potentially produce ensembles with different properties than the almost global model. However, the MLP derived from N-BEATS has members with the same input windows and loss functions as the almost global model. Why this would produce an ensemble with qualitatively different properties is unknown.

Since the MLP trained in the N-BEATS pipeline performs well, we suspect that there are differences in training procedures that cause better ensembles. Table 6.15 also show a model where we use the same learning rate schedule as in N-BEATS. It is meant to test whether models that are not trained aggressively will be more diverse and better in an ensemble. We only did this experiment with the global model but saw no signs of that being the case. Our best guess is that the differences are due to some other part of the training procedure.

Another possibility is that there are properties of the N-BEATS model creating better ensemble candidates. This is a viable explanation if we have made some error somewhere in our experiment, which is hard to safeguard against entirely.

6.1.9 Favorable Model Sizes With Global Models

An area where the models in this chapter are better is in terms of the total number of parameters. The global model has about 8.5M parameters per ensemble member. In total, that yields around 150M parameters for a full ensemble using six input lengths and three loss functions. From our discussion of the N-BEATS ensemble, this is close to one ensemble member of N-BEATS. A comparable ensemble in N-BEATS has 18 times more parameters than the ensemble of our global model. Reductions of this size can be important in applications. Also, the best ensemble member of the global model is better than an ensemble member of N-BEATS. The best single ensemble members from the global model are also close in performance to the ensemble performance. However, in terms of the total overall performance, N-BEATS is still preferable.

Work towards smaller models and ensembles is important because it will make the model more usable in applications. It is not feasible for any applications to use the large ensemble from N-BEATS. Storing the weights of one ensemble member takes up around

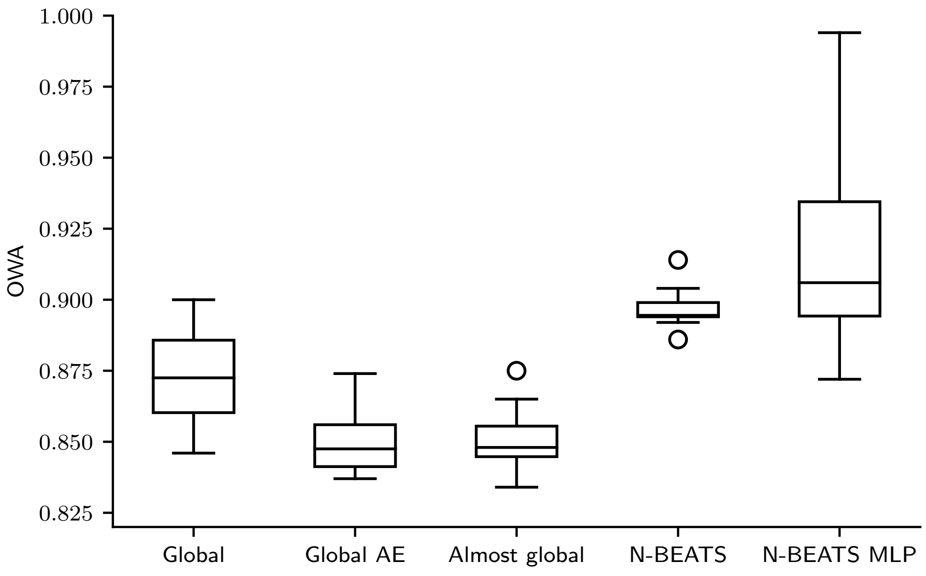


Figure 6.2: A boxplot showing the OWA score ensemble members for some of the models we discuss. The Global model is the global model with the original ensemble, and global ae is the global model with the shorter, alternative ensemble. Almost global is the almost global model from this chapter, while N-BEATS MLP is the similar almost global MLP trained in the N-BEATS pipeline. Lastly, N-BEATS is the N-BEATS ensemble. We have capped the plot at 1.000, but the N-BEATS MLP has one outlier above which is excluded because of this choice.

0.6GB. The small ensemble takes up more than 10GB. Besides storing the weights, forecasting is also more involved with that many models.

6.2 Zero-shot Forecasting, Extended Datasets, and Transfer Learning

In this section, we utilize the FRED dataset. We aim to understand how more data affects the models. This section has three experiments that are different despite being very similar. All experiments are evaluated using the M4 dataset. We leave the setting of the M4 competition in that we no longer evaluate models on the validation set but directly on the test set. Our reason for this approach is that we are no longer selecting the best model for the competition but rather trying to understand the different properties of the models.

We start by looking at the zero-shot properties of a model trained on the FRED dataset used to forecast the M4 dataset. Second, we look at the effects of using an extended dataset. Using the data from the FRED dataset, we expand the size of our training set, but we do not single out periods of training where we only focus on FRED or only focus on M4. This is aimed to look at how more data will affect the model. Lastly, we look at the transfer learning properties of our models. We pretrain models on the FRED dataset and finetune the model on the M4 dataset. Transfer learning is very successful in image analysis, and finetuning is a relevant use-case for pretrained models.

The main findings in this section are,

- Pretrained global and almost global MLPs on FRED can forecast the M4 dataset well. Both the global and almost global models score about the same and would have placed 14th and 10th in the M4 competition without being fitted to the M4 dataset. Noteworthy is also that they both place in front of the benchmarks in the competition.

- It is better to use only the target M4 dataset for training than to add additional data from FRED. The effect is most evident for global models indicating that they are more affected by changes in the distribution over frequencies.

- Retraining only the last layers of the almost global MLP gives similar performance to training a full model. The global model does not see the benefits of transfer learning, again indicating that it is sensitive to changes in the distribution.

6.2.1 Pretrained Models for Zero-shot Forecasting

In this experiment, we look at the performance of models trained on proportions of FRED tested on the M4 dataset. For the global model, we also look at three models of different complexities. We do not do this for the almost global models due to the computational complexity of training that many models.

The pretrained models are trained on the FRED dataset using the same training procedure as before. For the proportions of the dataset, we make sure to select proportions on the frequency level. We do not perform the experiment several times to get some idea about the variability from the random selection of proportions. We make sure that each larger proportion contains the smaller ones.

Figure 6.3 show the total overall scores for proportions of the FRED dataset. The models using 0.1 of FRED have about 35,000 time series in their dataset. For the global model, we test three models of increasing complexity. The small model has 5 layers and 512 hidden units, the medium is the model we have used earlier with 10 layers and 1024 hidden units, and the large model has 10 layers with 2048 hidden units. We do not test model sizes for the almost global model as it would take lots of extra compute. Training two proportion models of the almost global model is the same as training all the proportion models of the three model sizes for the global model in terms of computing.

The results of the global model can be seen in Figure 6.3a. With a limited dataset available, it is only the smaller model that is able to converge. Part of this can also be that the larger models are not trained for long enough, but larger models also have a higher risk of overfitting on smaller datasets, which can also be a cause. Both the experiments using N-BEATS and our experiments using global models on M4 showed that the performance of the small model in this experiment is competitive. When the dataset increases, we see that the three models are similar in performance. The larger model also seems to benefit more from increased data, even though the differences between the three can not be said to be significant.

Figure 6.3b show the results for the almost global model. The model performs better as we increase the size of the dataset until we use the full dataset. This is strange, but the difference in score for the models is small and can therefore be randomness as we only do the experiments once. It indicates that the performance flattens once we use about half the dataset, which is around 175,000 time series.

The global and almost global models are similar in their total overall scores, with the almost global model coming in slightly in front. We can give the scores some context by using the results from the M4 competition. The best pretrained global model is the largest global model trained on the full FRED dataset. This model has a total overall score of 0.886. In the M4 competition, this would have been a 14th place out of 59 entries, including benchmarks. It beats all the benchmarks, both statistical and machine learning. The benchmarks include the Theta method (Assimakopoulos and Nikolopoulos, 2000) that won the M3 competition and also popular models like ARIMA[1] and exponential smoothing. These benchmarks all lie in the range from 0.897 to 0.908. These results are from not looking at the time series of the M4 dataset at all. This is a clear indicator that there are characteristics across time series that are better approximated using many time series than individual ones.

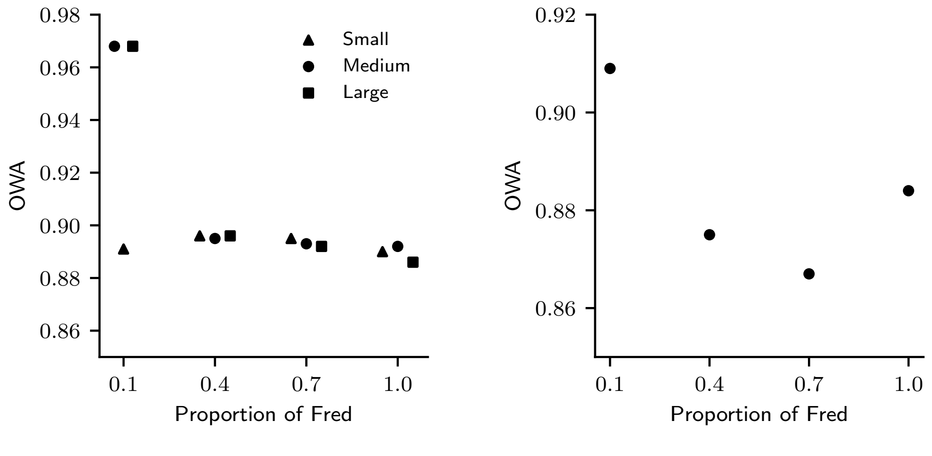
The almost global model is slightly better with a score of 0.867 using 70% of the FRED. This is a 10th place in the M4 competition. We leave the hourly segment out of calculating the overall score for the pretrained almost global model as hourly series are not present in FRED. Oreshkin et al. (2020) overcomes this by considering upsampling the monthly data by adding midpoints by interpolation to get a frequency of 24. Such techniques can be used, but it is more reasonable to add some hourly time series to the dataset. We have seen that the almost global models we have tested have been able to forecast the hourly segment well despite the limited number of time series in this segment (414). Given the low number of time series required to get good performance in this segment, it seems like a better choice to source hourly data and add this to the dataset.

Compared to the zero-shot results of Oreshkin et al. (2020), the models in this section are behind. Oreshkin et al. (2020) do not report total overall scores, but rather the SMAPE score. The zero-shot SMAPE score of N-BEATS trained on FRED and tested on M4 is 11.70, compared to 11.14 for N-BEATS trained on M4. The global model goes from 11.66 to 12.36, and the almost global goes from 11.61 to 12.10. DeepAR is also included as a benchmark by Oreshkin et al. (2020), however not in a zero-shot fashion. The SMAPE score of DeepAR trained and tested on the M4 dataset is 12.25. An almost global model can therefore forecast the M4 dataset in a zero-shot fashion than the also almost global DeepAR model. The zero-shot global model is also close in performance. While these are very interesting results, it can also signify that DeepAR is better suited at homogeneous datasets.

6.2.2 Extended Dataset

Our next experiment is to use the full M4 dataset but add extra data from FRED. An increased dataset can allow for more complex models without overfitting and better generalization. We use the same models and setup as in the zero-shot experiment, and in

[1]The authors use the `auto.arima` function in `R` on each time series for the ARIMA benchmark.



(a) Global **(b)** Almost Global

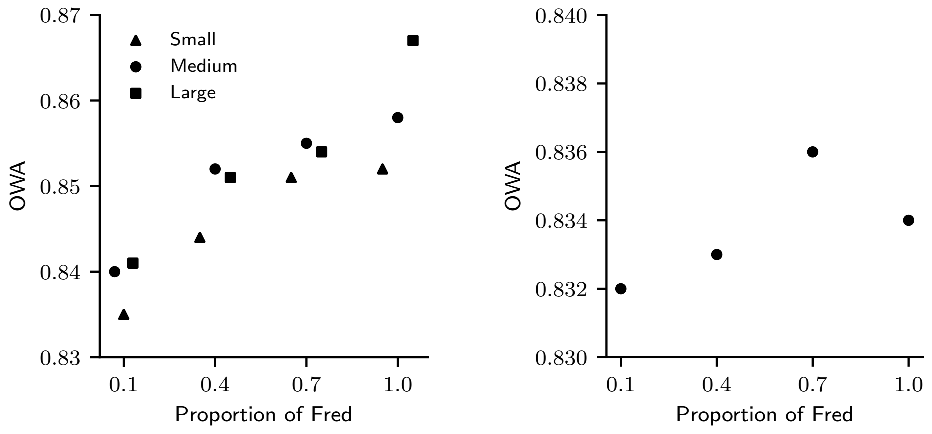
Figure 6.3: Zero-shot overall scores for global and almost global models. The models are trained on proportions of FRED and tested on the M4 dataset. There are three global models. Small has 5 layers with 512 hidden units, medium has 10 layers with 1024 hidden units, and large has 10 layers with 2048 hidden units. All models are trained using SMAPE loss. Global models have an ensemble of input lengths $[20, 30, 40, 50, 60, 70]$ and almost global models have lookbacks $[2, 3, 4, 5, 6, 7]$. The scatter points for the global models have been slightly shifted on the x-axis for better readability. We test models with proportions 0.1, 0.4, 0.7, and 1.0.

addition, use a smaller and larger global model.

Figure 6.4 show the results for the global and almost global models when we extend the M4 dataset using proportions of FRED. The global models in Figure 6.4a all perform worse when we add more data from FRED. The smallest model is best for all proportions. This is interesting and can indicate that the smaller model is less sensitive to changes in the distribution.

The results for the almost global model is shown in Figure 6.4b. While we see an increase also here, the scale of the increase is very small and not necessarily significant. This observation further indicates that the added data is not what decreases the performance but rather the change in the distribution of the data. This is less noticeable for almost global models as they will not be affected by changes in distribution over frequencies, only changes within each frequency.

Both the global and almost global model results indicate that we do not get better generalization by adding data from FRED. This indicates that the data in the M4 dataset is sufficient to generalize well when using global and almost global models. Therefore, the task of modeling the M4 dataset is likely not a task that will benefit from pretrained models or additional data. However, it would have been very interesting to see how these experiments would have been replacing FRED with ForeDeCk.



(a) Global **(b)** Almost global

Figure 6.4: Overall scores for global and almost global models trained on M4 and proportions of FRED. The models and setup is as in Figure 6.3.

6.2.3 Transfer Learning

Another way a pretrained model is useful is in transfer learning scenarios. If we have limited available data, we might not be able to train the desired model without a high risk of overfitting. Instead of training a full model, we instead use the pretrained model but retrain some of the last layers on the limited dataset. In that way, we utilize the hopefully fundamental characteristics from the pretrained model and let the model find out how to best utilize these characteristics on the new dataset.

Retraining the Last Layers

Retraining the last layers is motivated by the idea that the first layers extract features from the time series. If these features are somewhat general, they can be useful for a model on another dataset.

Table 6.17 shows the overall scores of retraining the last layers of the global and almost global models in the second row group. The almost global model reaches performance close to the model trained on the full dataset. The model also performs better in the yearly segment than the full model. We have skipped the yearly segment for the almost global model. Dropping this also for the full model would have moved the two even closer. The finetuned model's performance indicates that the earlier parts of the pretrained network also extract useful features on the target dataset. It also indicates that the forecasting is done in the last layers.

The story is different for the global model. Finetuning the last two layers improves the performance over the pretrained model, but it is far from the model trained on the target dataset. We are not observing similar behavior as for the almost global model. We also observed differences in behavior when training the models on the M4 dataset.

For example, naive seasonal initialization broke the global model. The observation that retraining the final layers is insufficient to get comparable performance for global models indicates that they work differently from the almost global models. Most relevant is that it indicates that global models do not necessarily work in a two step process of extracting characteristics and then using these to forecast.

We have earlier discussed how the differences in the distribution of FRED and M4 can affect global models. This experiment further strengthens the argument that global models also require the source dataset to be similar in the joint distribution over frequencies.

Settings with limited data are where we expect retraining the last layers of a model to be most useful. In situations with limited data, we can still benefit from a larger model while restricting the number of parameters we fit the target dataset. This allows us to mitigate the risk of overfitting better. Large neural networks are often not feasible either due to the computational complexity or limited data. In our experiment, we are not restricted by computational complexity and likely not limited by limited data. Utilizing FRED to get a larger dataset could allow for a larger model than possible only using the M4 dataset. However, it seems like the M4 dataset has enough data to allow for neural networks to train well.

A better simulation of global models' transfer learning properties would have been to simulate scarce data situations. For example, we could sample small subsets of M4 and then treat them as target datasets. This will be a more realistic scenario, and in fact, also feasible from a comparison side of view. The M4 organizers have a repository with all forecasts for all entries in the competition. We could therefore compare the transfer learning approach to all local benchmarks and entries. We leave this as an exciting direction for future work.

Two-step Retraining

The second experiment extends the first by also allowing retraining the full network after retraining the last layers. We emphasize that this is different from retraining the last layers and the full network at once.

Table 6.17 shows the two step finetuning results for the global and almost global models in the third row group. The almost global model does not show any improvement over just retraining the final layers. The global model, on the other hand, shows better performance. We saw that just retraining the final layers did not work as well for the global as for the almost global models. When we let the model adjust all its weights, it can better adjust to the target dataset.

Since only retraining the last layers was not successful for global models, but the two step retraining was, we suspect that the two step is closer to initializing the network with the weights of the pretrained model. It does seem like the global models disregard the first part of the two step training procedure and exploit the second part.

Initializing the Weights with the Pretrained Model

The last of the transfer learning experiments is to use the pretrained model as an initialization for the models instead of random initialization. Potentially this will make sure the weights are in the right region of a minima by inducing the bias from the FRED dataset.

Table 6.17: Overall scores of the transfer learning experiments with the global (G) and almost global (AG) models pretrained on the full FRED dataset. The pretrained model and the model trained on the target dataset are also included for comparison. Note that the target models are with an ensemble using three loss functions instead of one as the source and transfer learning models.

Model	Yearly	Qrtrly	Monthly	Weekly	Daily	Hourly	Global
Source G	0.803	0.875	0.941	0.953	0.992	4.296	0.892
Source AG	0.797	0.865	0.897	0.942	1.661	-	0.884
Finetune G	0.783	0.873	0.904	0.968	1.010	4.828	0.875
Finetune AG	0.769	0.854	0.867	0.853	0.971	-	0.829
Two-step G	0.784	0.862	0.874	0.919	0.947	2.017	0.845
Two-step AG	0.772	0.844	0.868	0.756	0.971	-	0.829
Init G	0.780	0.860	0.873	0.929	0.951	1.467	0.840
Init AG	0.772	0.843	0.867	0.752	0.973	-	0.828
Target G	0.777	0.849	0.864	0.934	0.934	1.586	0.833
Target AG	0.779	0.835	0.863	0.734	0.977	0.443	0.827

Table 6.17 shows the results from using the pretrained model as initialization and then training the models using the same procedure as we used in the experiments on the M4 dataset. The results are in the fourth row group. We see that both the global and almost global models are close to the models trained solely on M4 with random weight initialization.

When we were retraining the last layers, we discussed that the M4 dataset could be large enough for models to perform well without additional help. It does seem that we see the same effect in this experiment. The bias introduced by the pretrained weights is towards a slightly different distribution than M4. If we can approximate the M4 distribution well enough using the existing data, the bias from FRED will move the model slightly away from the better distribution. The case is different in scenarios where the data is limited, and we will not be able to approximate the distribution without overfitting if we use deep learning models. This hypothesis can be tested in the same ways as we discussed in the experiment of retraining the last layers.

Chapter 7

Conclusion and Outlook

This chapter summarizes the work in this thesis. We review and discuss the main contributions and propose possible extensions for future work.

The first part of the thesis gave required background for time series modeling and highlighted the distinction between global and local models. We chose to use MLPs for our experiments as this is the simplest deep learning architecture. The main goal was to look into whether global models' theoretical properties would also show in experiments. A simple and general model was therefore preferred.

We chose the M4 dataset as our target dataset. This is the most used dataset for evaluating the performance of general univariate forecasting models. We also sourced FRED, an even larger dataset, to look into the effects of pretraining, zero-shot forecasting, and additional data as a means to make models generalize better. Our discussion of these datasets also emphasized the potential temporal bias that can occur in datasets for global models. The two datasets were the basis for our two experiments.

First, we showed that both global and almost global MLPs could be trained to score as the top entries in the M4 competition. We emulated how one could have approached the competition using global models. Deep learning benchmarks scored among the last in the official M4 competition. We have shown how this is not because they are unsuited for time series forecasting, but rather that the added model complexity requires global or almost global models. However, we have also seen that applying deep learning models is not straightforward and that it takes care to make the models work well.

The global and almost global models we have trained are still behind the current state-of-the-art model, N-BEATS. N-BEATS is an almost global model but more specialized than our simple MLPs. Our initial hypothesis was that N-BEATS could be seen as a deep MLP. This view is slightly changed as we could not train simple MLPs to a score close to N-BEATS. However, in our small study of N-BEATS, we were able to train an MLP with naive seasonal initialization using the N-BEATS training procedure to be close in performance to N-BEATS. This is an indication that it is possible. Interestingly, we saw that it is likely not because each model is any better, but that the ensemble members complement each other.

An important consequence of that the single models we have trained score better is that we can drastically reduce the number of parameters. A full ensemble of N-BEATS is massive and can be reduced by a factor of 3000 using a global MLP while still scoring a 2$^{\text{nd}}$ place in the M4 competition. A global model with 10 layers and 1024 hidden units is about 80 parameters per time series. We saw that even smaller models with 5 layers and 512 hidden units scored close to our top models, resulting in less than 10 parameters per time series.

Second, we used FRED to pretrain models and evaluate them on the M4 dataset. Again, we used simple and general models. The pretrained models still scored well compared to the entries in the M4 competition, but worse than the models trained solely on the M4 dataset. Our results are also behind what Oreshkin et al. (2020) score by training N-BEATS on FRED and evaluating on M4.

Adding data from FRED on top of M4 did not increase the performance of the models either. However, there was a difference in that the global model was much more negatively affected by mixing FRED and M4. We discussed how this could be because global models are more prone to differences in the distribution over frequencies of the two datasets.

We showed that finetuning a pretrained network to a target dataset by retraining only the last layers can result in performance similar to training the full model for almost global models. Global models did again seem to behave differently as retraining only the last layers did not result in a similar performance increase.

Pretraining and finetuning are most useful in situations with limited available data. Our experiments suggest that the M4 dataset has enough data for us to approximate the distribution well. There are, therefore, other datasets or tasks where we can likely better evaluate the usefulness of the availability of additional data.

7.1 Outlook

Our experiments have revealed some exciting extensions for future work. There is also work supported by the theory in this thesis that we have not pursued in this thesis.

First, we should expand our experiments on M4 to include convolutional, recurrent, and attention-based architectures. This will give more insights into the validity of the global arguments across model types. It can also potentially reveal which types of architectures are most promising moving towards general time series encoders.

Second, as the ForeDeCk dataset becomes available, we can test if pretraining and finetuning global models is feasible when the source and target distribution are very close. ForeDeCk will also allow us to test if M4 can benefit from more data if the data is from a superset.

Third, we discussed how there are more relevant tasks for pretraining and finetuning than forecasting M4. This line of experiments will give insights into the effect of pretraining on large and general datasets and finetuning on a small target dataset.

These three experiments are all on the path towards universal time series encoders. More has proven to be more for language models and image analysis. A larger and more diverse dataset is needed to make the models more general. So is also a wider range of datasets and tasks we test on. We have mentioned that there have been comments on the M4 competition having too many yearly, quarterly and monthly time series with a business focus. Important types of time series like, e.g., medical data are not present, and the data is in general very clean and preprocessed. The broader set of tasks can be pure forecasting challenges like M4 and tasks with limited data, intermittent data, and a more diverse range of frequencies and domains. Our discussion in Chapter 4 highlighting the possibility of temporal bias in global models is relevant in this context of larger and more diverse datasets. The extent of information leakage from test sets is relevant and can form a direction of work by itself. We suggested an empirical line of work with the potential experiment of doing train-test splits by date, but a more thorough theoretical analysis is also interesting.

Global models also have some applications that we have not tested in this thesis. Classification is a very relevant task and is supported that optimizing for the conditional likelihoods is the same as optimizing for the full likelihood. This approach is also supported by the success it has had in image analysis. A good general global model can also be used for imputing missing values. In the case of imputing, it makes sense to use the context of the missing value using a bidirectional model. Multivariate problems are also relevant despite using a univariate model. A global univariate model can act as a prior on each covariate and leave it up to a specialized network to model dependencies. Lastly, we believe one of the most useful applications of global pretrained models is in more data-scarce settings. Retraining the last layers can then adapt the model to specific datasets while still utilizing the powers of the pretrained model.

As discussed, several possible directions will move closer to general global models. If such models are good enough to be useful in many applications remain to be seen. In this thesis, we have shown that it is possible to train global models close to the performance of almost global models and that both are comparable to top forecasting methods. This is an important step and can motivate future work in this direction.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL: https://www.tensorflow.org/. software available from tensorflow.org.

Abu-Mostafa, Y.S., Magdon-Ismail, M., Lin, H.T., 2012. Learning from data. volume 4. AMLBook New York, NY, USA:.

Assimakopoulos, V., Nikolopoulos, K., 2000. The theta model: a decomposition approach to forecasting. International journal of forecasting 16, 521–530.

Bai, S., Kolter, J.Z., Koltun, V., 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271 .

Brockwell, P.J., Davis, R.A., 2016. Introduction to time series and forecasting. springer.

Brockwell, P.J., Davis, R.A., Fienberg, S.E., 1991. Time series: theory and methods: theory and methods. Springer Science & Business Media.

Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al., 2020. Language models are few-shot learners. arXiv preprint arXiv:2005.14165 .

Casella, G., Berger, R.L., 2002. Statistical inference. volume 2. Duxbury Pacific Grove, CA.

Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Dhariwal, P., Luan, D., Sutskever, I., 2020. Generative pretraining from pixels, in: Proceedings of the 37th International Conference on Machine Learning, pp. 10456–10468.

Chuah, M.C., Fu, F., 2007. Ecg anomaly detection via time series analysis, in: International Symposium on Parallel and Distributed Processing and Applications, Springer. pp. 123–135.

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

Dua, D., Graff, C., 2017. UCI machine learning repository. URL: http://archive.ics.uci.edu/ml.

Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A., 2018. Transfer learning for time series classification, in: 2018 IEEE international conference on big data (Big Data), IEEE. pp. 1367–1376.

Federal Reserve Bank of St. Louis, 2020. Fred economic data. Data retrieved from Federal Reserve Economic Data database, https://fred.stlouisfed.org/.

Fry, C., Brundage, M., 2019. The m4 forecasting competition-a practitioner's view .

Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y., 2016. Deep learning. volume 1. MIT press Cambridge.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del R'ıo, J.F., Wiebe, M., Peterson, P., G'erard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., ravis E. Oliphant, 2020. Array programming with NumPy. Nature 585, 357–362. URL: https://doi.org/10.1038/s41586-020-2649-2, doi:`10.1038/s41586-020-2649-2`.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R., 2012. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 .

Hoeffding, W., 1994. Probability inequalities for sums of bounded random variables, in: The Collected Works of Wassily Hoeffding. Springer, pp. 409–426.

Hyndman, R.J., Khandakar, Y., 2008. Automatic time series forecasting: the forecast package for R. Journal of Statistical Software 26, 1–22. URL: https://www.jstatsoft.org/article/view/v027i03.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., Wilson, A.G., 2018. Averaging weights leads to wider optima and better generalization. arXiv preprint arXiv:1803.05407 .

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.

Kwac, J., Flora, J., Rajagopal, R., 2014. Household energy consumption segmentation using hourly data. IEEE Transactions on Smart Grid 5, 420–430.

Laptev, N., Yu, J., Rajagopal, R., 2018. Reconstruction and regression loss for time-series transfer learning, in: Proc. SIGKDD MiLeTS.

Loshchilov, I., Hutter, F., 2016. Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983 .

Makridakis, S., Spiliotis, E., Assimakopoulos, V., 2020. The m4 competition: 100,000 time series and 61 forecasting methods. International Journal of Forecasting 36, 54–74.

Montero-Manso, P., Athanasopoulos, G., Hyndman, R.J., Talagala, T.S., 2020. Fforma: Feature-based forecast model averaging. International Journal of Forecasting 36, 86–92.

Montero-Manso, P., Hyndman, R.J., 2020. Principles and algorithms for forecasting groups of time series: Locality and globality. arXiv preprint arXiv:2008.00444 .

Van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al., 2016. Conditional image generation with pixelcnn decoders, in: Advances in neural information processing systems, pp. 4790–4798.

Oord, A.v.d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K., 2016. Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499 .

Oreshkin, B.N., Carpov, D., Chapados, N., Bengio, Y., 2019. N-beats: Neural basis expansion analysis for interpretable time series forecasting. arXiv preprint arXiv:1905.10437 .

Oreshkin, B.N., Carpov, D., Chapados, N., Bengio, Y., 2020. Meta-learning framework with applications to zero-shot time-series forecasting. arXiv preprint arXiv:2002.02887 .

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. Pytorch: An imperative style, high-performance deep learning library, in: Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., 2018. Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., 2019. Language models are unsupervised multitask learners. OpenAI blog 1, 9.

Rossum, G., 1995. Python reference manual .

Sagheer, A., Kotb, M., 2019. Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems. Scientific reports 9, 1–16.

Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T., 2020. Deepar: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting 36, 1181–1191.

Sen, R., Yu, H.F., Dhillon, I., 2019. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. arXiv preprint arXiv:1905.03806 .

Serrà, J., Pascual, S., Karatzoglou, A., 2018. Towards a universal neural network encoder for time series., in: CCIA, pp. 120–129.

Shalev-Shwartz, S., Ben-David, S., 2014. Understanding machine learning: From theory to algorithms. Cambridge university press.

Själander, M., Jahre, M., Tufte, G., Reissmann, N., 2019. EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure. arXiv:1912.05848.

Smyl, S., 2020. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. International Journal of Forecasting 36, 75–85.

Spiliotis, E., Patikos, A., Assimakopoulos, V., Kouloumos, A., 2017. Data as a service: Providing new datasets to the forecasting community for time series analysis. URL: https://forecasters.org/wp-content/uploads/gravity_forms/7-c6dd08fee7f0065037affb5b74fec20a/2017/07/ISF2017_presentation_Spiliotis.pdf.

Szenkovits, A., Meszlenyi, R., Buza, K., Gaskó, N., Lung, R., Suciu, M., 2018. Feature Selection with a Genetic Algorithm for Classification of Brain Imaging Data. pp. 185–202. doi:10.1007/978-3-319-67588-6_10.

Taylor, S.J., Letham, B., 2018. Forecasting at scale. The American Statistician 72, 37–45.

pandas development team, T., 2020. pandas-dev/pandas: Pandas. URL: https://doi.org/10.5281/zenodo.3509134, doi:10.5281/zenodo.3509134.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need, in: Advances in neural information processing systems, pp. 5998–6008.

Vidyasagar, M., 2003. Vapnik-chervonenkis, pseudo-and fat-shattering dimensions, in: Learning and Generalisation. Springer, pp. 115–147.

Williams, Z., 2015. Python wrapper of the st. louis federal reserve bank's fred api web service for retrieving economic data. https://github.com/zachwill/fred.