Efficient and Accurate Forecasting in Large-scale Settings

Olivier Sprangers

Efficient and Accurate Forecasting in Large-scale Settings

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties
ingestelde
commissie, in het openbaar te verdedigen in
de Agnietenkapel
op donderdag 5 september 2024, te 13:00 uur

door

Olivier Rudolf Sprangers

geboren te Alkmaar

The research was carried out at the Information Retrieval Lab at the University of Amsterdam. This research was (partially) funded by Ahold Delhaize,[1] through AIRLab.

Printed by Ridderprint, Alblasserdam

ISBN: 978-94-6483-952-4

[1]https://www.aholddelhaize.com/

Acknowledgements

Surprisingly, I found this chapter the hardest to start with. I have many people I'd like to thank, but thinking about whom I'd like to acknowledge led me to reconstruct my entire PhD-journey. This journey already started when I concluded my Master's degree back in 2012, after which it would have been a lot more convenient to embark on a PhD. But that's playing Captain Hindsight, and it's easy to predict the past knowing the future – if it wasn't hard to know what's up ahead, you wouldn't be reading this thesis!

So, I'll start this chapter by thanking those who have been the closest to me throughout this journey: Mariska, my parents, Maaike, Anne-Marie, Pieter, my family, my friends, and, of course, my annoying but sweet dog. I definitely couldn't have done this without the love and support from all of you; you – quite literally – enabled me to see clearly.

Next, it feels most natural that I should start with thanking my former Master thesis advisors, Robert and Gabriel. They always believed in me and suggested I should do a PhD. They definitely planted the seed for the years to come.

I'd like to thank my colleagues from my first job, who always believed in me even though I completely lost and destroyed myself somewhere along the job. A key thing I learned there – mostly from Richard – is simplicity. I used to have a knack for making huge, complex models. It's as if you need to go through a phase of complexity to realize that simpler solutions are often preferable over complex ones, but it's also harder to find this simple solution. I hope I applied that mindset in this thesis, and I hope I can continue that mindset for the years to come.

In my second job I had my first venture into large-scale forecasting, as I needed to predict payment rates of credit portfolios. I should especially thank Joost and Erwin for my time there. They taught me useful project management skills that I could apply throughout my PhD. However, in this job I also realized that maybe I should still undertake a PhD.

I switched jobs again and became responsible for the pricing team of a major Dutch e-commerce company. This further sparked the seed for undertaking the PhD, as I found it extremely annoying that I couldn't level with the data scientists at the lowest level of understanding. I'd like to thank Andres for the discussions we had. Even though I was sometimes annoyed at the time, these discussions did make me realize I should still really do the PhD. I should also thank Marijn for giving useful advice on doing a PhD and Ellen, for teaching me how to mentor someone (although it's always easy to mentor people who are that good).

And then I took the plunge! Thank you Maarten, for giving me this opportunity, your patience when working with me and always providing care when needed. Sebastian, thank you for helping me with all of my work and especially for being there for all my day-to-day quick questions. Both you and Maarten have been great mentors to me and hopefully we can work together again in the future. I'd like to thank my colleagues from the forecasting team at bol, who made me feel welcome and offered the support I needed to test my ideas within an industry setting. The partnership with bol even resulted in two publications together with bol colleagues – one with Wander and one with Barrie – so thank you both for that. Also a word of thanks to my fellow AIRLab colleagues – Mozdeh, Sami, Mariya, Ming, Arezoo, Shubha, Barrie and Stefan – it was always great to chat about work and other random stuff, and I'll miss the brainstorm sessions we had. Also, without the support of Petra and Ivana I don't think I could have been able to do my job, so thank you for your unwavering support of the research lab. Of course, my Master students also played an important role too. It turned out to be very fulfilling to see a student grow throughout her/his thesis project, and their enthusiasm for their projects often helped revitalize my own motivation for the PhD. I'm grateful for my PhD committee – Victoria, Paul, Tim, Evangelos and Ger – for ploughing through this thesis in order to assess it. I hope you enjoyed reading it and I hope we can work together in the future. Finally, a word of thanks to all the other great colleagues I worked with over the years. Even though some of you might not realize it, you helped me through this process.

Olivier
Amsterdam, April 2024

Contents

1 Introduction

You own an ice cream store. It's the end of the day, but you need to prepare the ice cream ingredients for tomorrow. How many ice creams will you sell tomorrow? This is a typical forecasting question: you are interested in predicting the future (the number of ice creams you will sell tomorrow), so that you can make better decisions today (what preparations you need to take today). To answer this forecasting question, a reasonable approach for you to take would be to look at the number of ice creams that were sold today as well as in the past, such as yesterday, or last week. In addition, you might want to use some information about tomorrow – you expect higher ice cream sales in weekends, as well as in summer versus winter. You might even add information from a good weather forecast for tomorrow you found online. Happy with the information you collected, you start building a model to forecast the number of ice creams that you will sell.

Whilst building your forecast model, you realize there is a big difference between the ice cream demand for different ice cream flavours. In addition, different ice cream flavours also typically sell in different sizes in your store. Finally, you realize that you can prepare the ingredients for the next seven days in advance. You also understand that the further into the future, the more uncertain you are of the ice cream demand. You now realize that you have to answer many forecasting questions rather than just a single one: for each ice cream flavour, for each ice cream size, for each of the next seven days, you want to know the number of ice creams you will sell, ideally with a notion of the uncertainty around each of these forecasts. Would you take the same approach to answer each of these forecasting questions?

This forecasting challenge is central to this thesis: how can we answer all these individual forecasting questions as efficiently and accurately as possible? It is not difficult to answer these forecasting questions *only* efficiently: for example, we could just use the same fixed constant prediction for every individual forecasting question. Highly efficient – we do not need a large quantity of resources to get our answer – but also probably highly inaccurate – our forecast will likely be far from the actual value. Conversely, it is also relatively easy to create *only* accurate forecasts: you hire a large team of smart forecasting practitioners who meticulously create a forecasting model for every individual forecasting question. Probably highly accurate, but also highly inefficient. Thus, the challenge is to solve these forecasting questions both efficiently *and* accurately.

Throughout this thesis, we visit a diverse set of forecasting problems, each in its own way suffering from inefficiencies when applying existing approaches to a large set of forecasting questions concurrently. We find many ways of improving efficiency whilst maintaining accuracy in these forecasting problems, and propose new directions for research into forecasting efficiency.

1.1 Thesis Scope and Research Questions

We give a brief overview of the scope of this thesis and the main research questions that will be answered.

The classical approach to forecasting is to learn a single model for each time series separately. As we saw in our ice cream example, this can quickly become time consuming when multiple forecasts are needed. Recently, a number of neural network architectures have been proposed [37, 76, 81] to address the disadvantages of classical forecasting methods. These neural architectures leverage the increasing availability of data, and enable the training of a single model that can produce forecasts for multiple time series over multiple time steps. The transformer [131] is one such neural architecture that has recently shown state-of-the-art performance on a set of real world forecasting datasets [80]. However, it requires a large number of parameters to achieve these state-of-the-art results compared to existing neural architecture-based probabilistic forecasting methods. The disadvantage of an increase in parameters is the associated higher energy cost of training and prediction, and higher storage cost. This naturally leads us to our first research question:

Research Question 1: *How can we efficiently generate probabilistic forecasts with neural networks for large-scale settings?*

To answer this question, we investigate neural network architectures that require fewer parameters whilst maintaining the same level of forecasting performance. In addition, we investigate how we can use the information available in our data more smartly in these neural architectures. Finally, we investigate how we can use a different loss function to improve forecasting performance and reduce training complexity.

Next, we turn to models of a different architecture, motivated by working with our industry partners Albert Heijn and bol, a grocery chain and e-commerce platform, respectively. Within the forecasting teams of these companies, neural networks are not often used for (probabilistic) forecasting. Instead, Gradient Boosting Machines (GBM) are the bread-and-butter for (probabilistic) forecasting for large-scale problems in these companies. Interestingly, existing implementations of Gradient Boosting Machines often require training multiple models for probabilistic forecasting (e.g., LightGBM [68] or xgboost [24] require a separate model for each quantile of the forecast), or require computing expensive second-order derivative statistics [NGBoost, 34]. This motivates our second research question:

Research Question 2: *How can we efficiently generate probabilistic forecasts with Gradient Boosting Machines (GBM) for large-scale settings?*

We answer this question by investigating how we can turn the deterministic predictions of GBM into stochastic predictions. We find that a small set of modifications to the main prediction equations of GBM allows us to create probabilistic predictions using a single model only.

Now that we have an idea on how to efficiently generate (probabilistic) forecasts for large-scale settings using two of the most commonly used large-scale forecasting methods (neural networks and GBM), we observe that in all of these methods, we typically only create forecasts for the lowest granularity of time series. However in industry, forecasts at the lowest granularity – often the individual product level – are required but we also need forecasts at higher granularities, for example at the category, department or regional level, as higher level forecasts are often needed in logistics and financial planning [21]. Second, forecasts at different time granularities are required, for example daily or weekly forecasts. It is common that separate forecast models are made for each separate (temporal) granularity, and as such these forecasts may not be coherent with each other. Hierarchical forecasting [61] and temporal hierarchical forecasting techniques [12, 101, 125] aim to solve the problem of creating forecasts that are coherent with respect to a pre-specified cross-sectional and/or temporal hierarchy of the underlying time series. Unfortunately, existing hierarchical forecasting methods scale poorly to settings with millions of time series, motivating our third research question:

Research Question 3: *How can we efficiently generate hierarchical forecasts for large-scale settings?*

We answer this question by investigating the use of a specialised loss function that directly optimizes both cross-sectional and temporal hierarchical structures. The benefit of performing hierarchical forecasting in an end-to-end manner by means of a specialised loss function is that we remove the need for a post-processing step, as is customary in existing hierarchical forecasting methods [12, 61, 134].

In our final research chapter, we turn to another forecasting problem often encountered at our industry partners: recommendations. We investigate state-of-the-art methods for session-based recommendation and surprisingly find that the most simple method gives the most accurate results. This motivates our final research question:

Research Question 4: *How can we efficiently generate session-based recommendations at the scale of bol?*

We answer this question by investigating how to efficiently implement Vector-Session kNN (VS-kNN) – one of the relatively simple methods that gives strong performance in offline tests – at the scale of bol.

This concludes the overview of the research questions that are answered in this thesis. Next, we turn to an overview of the main contributions of this thesis.

1.2 Main Contributions

We divide our contributions into three parts: theoretical, empirical, and software contributions.

Theoretical contributions

- We introduce the Bidirectional Temporal Convolutional Network, or BiTCN for short, a neural network that can achieve on-par or better forecasting results than competing methods whilst requiring an order of magnitude fewer parameters (Sprangers et al. [119], Chapter 2).
- We introduce Probabilistic Gradient Boosting Machines (PGBM), a method that can generate accurate probabilistic predictions using gradient boosting machines whilst using only a single model rather than multiple models (Sprangers et al. [118], Chapter 3).
- We design a sparse hierarchical loss function that enables direct end-to-end training of cross-sectional and temporal hierarchical forecasts in large-scale settings (Sprangers et al. [120], Chapter 4).
- We introduce Vector-Multiplication-Indexed-Session kNN, which we abbreviate to VMIS-kNN, an index-based variant of a state-of-the-art nearest neighbor algorithm for session-based recommendation, which scales to use cases with hundreds of millions of clicks to search through (Kersbergen et al. [71], Chapter 5).

Empirical contributions

- We empirically verify that we achieve state-of-the-art forecasting performance with BiTCN on four real-world datasets. We evaluate on point forecast error metrics (sMAPE, NRMSE) as well as on probabilistic forecast error metrics (quantile loss percentiles) (Sprangers et al. [119], Chapter 2).
- We show that BiTCN (1) requires an order of magnitude fewer parameters than the second-best scoring transformer-based method, and (2) employs a simpler architecture than WaveNet [130] – an autoregressive neural network based on a TCN – which placed

second in a Kaggle forecasting competition [69]. The result is that BiTCN can be trained using cheaper hardware because it requires less memory (approx. 2–4x less GPU memory, depending on batch size). Moreover, BiTCN also requires at least 20% less energy during training, thereby reducing costs to train the model (Sprangers et al. [119], Chapter 2).

- We demonstrate the benefits of choosing a Student's t(3)-distribution for the probabilistic forecasting setting, which enables us to eliminate a training hyperparameter compared to using a Gaussian distribution for probabilistic forecasting, and it results in a more stable training regime (Sprangers et al. [119], Chapter 2).
- We demonstrate state-of-the-art point performance and probabilistic performance of PGBM on a set of regression benchmarks (Sprangers et al. [118], Chapter 3).
- We show that PGBM's probabilistic performance can be optimized *after* training the model, which allows practitioners to choose different posterior distributions without needing to retrain the model (Sprangers et al. [118], Chapter 3).
- Our implementation of PGBM trains up to several orders of magnitude faster on larger datasets than competing methods, and our implementation allows the use of complex differentiable loss functions, where we observed up to 10% improvement in point forecasting performance and up to 300% improvement in probabilistic forecasting performance (Sprangers et al. [118], Chapter 3).
- We empirically demonstrate that our sparse hierarchical loss function can outperform existing hierarchical forecasting reconciliation methods by up to 10% (Sprangers et al. [120], Chapter 4).
- We show how our sparse hierarchical loss function scales to large-scale settings and demonstrate a reduction of both training and prediction time of up to an order of magnitude compared to the best hierarchical forecasting reconciliation methods (Sprangers et al. [120], Chapter 4).
- We present the results of an offline test of our method for the primary product demand forecasting model at bol, a European

e-commerce company with a catalogue of millions of unique products, demonstrating an improvement of 2% on RMSE and 10% on MAE as compared to the baseline forecasting system (Sprangers et al. [120], Chapter 4).

- We discuss design decisions and implementation details of our production recommender system *Serenade*, which applies stateful session-based recommendation with VMIS-kNN, and can handle more than 1,000 requests per second with a response latency of less than seven milliseconds in the 90th percentile (Kersbergen et al. [71], Chapter 5).

- To the best of our knowledge, we provide the first empirical evidence that the superior predictive performance of VMIS-kNN/VS-kNN from offline evaluations translates to superior performance in a real world e-commerce setting; we find Serenade to drastically increase a business-specific engagement metric by several percent, compared to the legacy system at bol (Kersbergen et al. [71], Chapter 5).

Software contributions

- We provide an open source implementation of BiTCN, as well as a suite of existing common neural network probabilistic forecasting methods, at `https://github.com/elephaint/pedpf`.

- We provide an open source implementation of PGBM, a Python pip-installable package that is fully compatible with Scikit-learn [97], at `https://github.com/elephaint/pgbm`.

- We provide an open source implementation of common hierarchical forecasting methods at `https://github.com/elephaint/hierts`.

- We make available under open license our implementation of *Serenade*, the session-based recommender system currently in production at bol, at `https://github.com/bolcom/serenade`.

1.3 Thesis Overview

In this section we shortly explain how this thesis is structured, and provide a guideline for how to read the thesis. In Chapters 2–5, we address each of the four research questions set out above. Each of these chapters can be read independently, as the required background information is contained in each chapter as well as the relevant related work. We conclude the thesis and provide avenues for future work in Chapter 6.

1.4 Origins

The chapters in this thesis are based on the following papers:

- **Chapter 2**
  O. Sprangers, S. Schelter, and M. de Rijke. Parameter-Efficient Deep Probabilistic Forecasting. *International Journal of Forecasting*, 39(1):332–345, Jan. 2023. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2021.11.011

  OS did conceptualization of this study, methodology, experiments, software and writing – original draft preparation, reviewing, and editing. SSC and MdR aided in reviewing and editing of the paper.

- **Chapter 3**
  O. Sprangers, S. Schelter, and M. de Rijke. Probabilistic Gradient Boosting Machines for Large-Scale Probabilistic Regression. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD ’21, pages 1510–1520, New York, NY, USA, Aug. 2021. Association for Computing Machinery. ISBN 978-1-4503-8332-5. doi: 10.1145/3447548.3467278

  OS did conceptualization of this study, methodology, experiments, software and writing – original draft preparation, reviewing, and editing. SSC and MdR aided in reviewing and editing of the paper.

- **Chapter 4**
  O. Sprangers, W. Wadman, S. Schelter, and M. de Rijke. Hierarchical Forecasting at Scale. *International Journal of Forecasting*, In Press, Mar. 2024. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2024.

02.006

This work was done while OS was part of the forecasting team at bol. OS did conceptualization of this study, methodology, experiments, software and writing – original draft preparation, reviewing, and editing. WW and SSC provided support for the experiments, and aided in reviewing and editing of the paper. MdR aided in reviewing and editing of the paper.

- **Chapter 5**
  B. Kersbergen, O. Sprangers, and S. Schelter. Serenade - Low-Latency Session-Based Recommendation in e-Commerce at Scale. In *Proceedings of the 2022 International Conference on Management of Data*, pages 150–159, Philadelphia, USA, June 2022. ACM. ISBN 978-1-4503-9249-5. doi: 10.1145/3514221.3517901

  BK did conceptualization of this study, implemented VMIS-kNN, executed the microbenchmarks, implemented *Serenade*, and executed the experiments at bol. OS designed VMIS-kNN, implemented VMIS-kNN and executed the microbenchmarks of VMIS-kNN. SSC did conceptualization of this study and implemented VMIS-kNN. All authors contributed to the writing.

Work on the thesis also benefited from insights and experiences gained through work on the following publications:

- S. Deng, O. Sprangers, M. Li, S. Schelter, and M. de Rijke. Domain Generalization in Time Series Forecasting. *ACM Transactions on Knowledge Discovery from Data*, Jan. 2024. ISSN 1556-4681. doi: 10.1145/3643035

- S. Schelter, S. Grafberger, S. Guha, O. Sprangers, B. Karlaš, and C. Zhang. Screening Native ML Pipelines with “ArgusEyes”. In *CIDR: Conference on Innovative Data Systems Research*, 2021

- O. Sprangers, R. Babuška, S. P. Nageshrao, and G. A. D. Lopes. Reinforcement Learning for Port-Hamiltonian Systems. *IEEE Transactions on Cybernetics*, 45(5):1017–1027, May 2015. ISSN 2168-2275. doi: 10.1109/TCYB.2014.2343194

2 Parameter Efficient Deep Probabilistic Forecasting

With the increasing availability of large volumes of data, a number of neural architectures have been proposed that can provide probabilistic forecasts for multiple time series using just a single model [37, 76, 80, 81]. In particular, transformer-based methods [80, 131] achieve state-of-the-art performance on real-world probabilistic forecasting benchmarks. However, these methods require a large number of parameters to be learned, which imposes high memory requirements on the computational resources for training such models. This leads us to our first research question:

> **Research Question 1:** *How can we efficiently generate probabilistic forecasts with neural networks for large-scale settings?*

To address this question, we introduce a novel Bidirectional Temporal Convolutional Network (BiTCN), which requires an order of magnitude fewer parameters than a common transformer-based approach. Our model combines two Temporal Convolutional Networks (TCNs): the first network encodes future covariates of the time series, whereas the second network encodes past observations and covariates. We jointly estimate the parameters of an output distribution via these two networks. Experiments on four real-world datasets show that our method performs on par with four state-of-the-art probabilistic forecasting methods, including a

This chapter was published as: O. Sprangers, S. Schelter, and M. de Rijke. Parameter-Efficient Deep Probabilistic Forecasting. *International Journal of Forecasting*, 39(1):332–345, Jan. 2023. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2021.11.011.

transformer-based approach and WaveNet, on two point metrics (sMAPE, NRMSE) as well as on a set of range metrics (quantile loss percentiles) in the majority of cases. Secondly, we demonstrate that our method requires significantly fewer parameters than transformer-based methods, which means the model can be trained faster with significantly lower memory requirements, which as a consequence reduces the infrastructure cost for deploying these models.

2.1 Introduction

The classical approach to forecasting is to learn a single model for each time series separately. However, creating such models often requires careful manual intervention from forecasting practitioners, which becomes impractical when the number of time series to be forecast is large (e.g., when the task is to forecast store demand for each product of a retail chain [21, 74, 124]). Moreover, decision makers often prefer probabilistic forecasts to point forecasts because they are interested in a quantification of the uncertainty of the forecast [21]. However, probabilistic forecasts may require a set of models for each time series, which further increases the disadvantages of using a single model for each time series. In this chapter, we address this problem of probabilistic forecasting for multiple time series.

Recently, a number of neural network architectures have been proposed [37, 76, 81] to address the disadvantages of classical forecasting methods. These neural architectures leverage the increasing availability of data in the aforementioned application domains, and enable the training of a single model that can produce forecasts for multiple time series over multiple time steps. The transformer [131] is one such neural architecture that has recently shown state-of-the-art performance on a set of real world forecasting datasets [80]. However, it requires a large number of parameters to achieve these state-of-the-art results compared to existing neural architecture-based probabilistic forecasting methods.

In this chapter, we demonstrate that it is possible to achieve on-par or better results than a transformer with an architecture that requires an order of magnitude fewer parameters. We achieve this by

(1) Smartly encoding available future information;

(2) Applying a simple temporal convolutional architecture; and

(3) Leveraging the Student's t(3)-distribution as a loss function to optimize parameters of our method.

Our BiTCN is based on two TCNs, as detailed in Section 2.3. The first network encodes future covariates of the time series, whereas the second network encodes past observations and covariates. This method allows us to preserve temporal information of sequence data, and is computationally more efficient by requiring fewer sequential operations than the commonly used bidirectional LSTM. The output of both our networks is used to estimate the parameters of a Student's t(3)-distribution of our forecast.

Next to introducing BiTCN, we contribute:

- We empirically verify that we achieve state-of-the-art forecasting performance with BiTCN on four real-world datasets (Sections 2.4 and 2.5). We evaluate on point forecast error metrics (sMAPE, NRMSE) as well as on probabilistic forecast error metrics (quantile loss percentiles).

- We show how BiTCN is designed (1) to require an order of magnitude fewer parameters than the second-best scoring transformer-based method, and (2) to employ a simpler architecture than WaveNet [130] – an autoregressive neural network based on a TCN – which placed second in a Kaggle forecasting competition [69]. The result is that our model can be trained using cheaper hardware because it requires less memory (approx. 2–4x less GPU memory, depending on batch size). Moreover, our method also requires at least 20% less energy during training, thereby reducing cost to train the model.

- We demonstrate the benefits of choosing a Student's t(3)-distribution for the probabilistic forecasting setting (Section 2.5.3), which enables us to eliminate a training hyperparameter compared to using a Gaussian distribution for probabilistic forecasting, and it results in a more stable training regime.

2.2 Related Work

Time series forecasting is a broad topic that is being studied in various scientific disciplines, such as econometrics, economics and machine learning. Traditional forecasting models such as ARIMA [22] and Exponential Smoothing [60] rely on considering the time series individually, and thus create separate models for each time series (typically referred to as local methods [93]). These local methods may be more difficult to apply in contemporary large-scale forecasting applications as maintaining individual models per time series may be impractical and these methods typically struggle to forecast unseen time series with little or no past observations. Moreover, Montero-Manso and Hyndman [93] have also demonstrated such local methods are typically outperformed by global methods (i.e. methods that create a single global model across all time series).

Recently, neural networks (in the form of sequence-to-sequence models) have been successfully applied to various autoregressive problems, such as speech generation with WaveNet [130] and probabilistic forecasting with DeepAR [107]. We refer to [89] for a more formal introduction to sequence-to-sequence modeling for time series, and to [93] for a study of the benefits of global models compared to local models for time series.

Recurrent neural networks RNNs, usually in the form of Long-Short Term Memory (LSTM) [53] modules, have been widely applied to the forecasting problem; Fischer and Krauss [37], Laptev et al. [76], Li et al. [81] provide recent examples of applications to financial markets, taxi services and traffic forecasting, respectively. Moreover, RNN models have been succesful at winning several forecasting competitions, such as the LSTM-based ES-RNN method that won the M4 forecasting competition [86], and the LSTM-based method that won the Kaggle Webtraffic competition.[1] The downside of employing RNN models in forecasting is that the recurrent nature of the model leads to slow training times, and long-term dependencies may not be properly captured. To improve the long-term memory of RNNs, attention mechanisms [14] have been combined with RNNs in [25, 75]. However, these methods still rely on the sequential training of RNN modules. Hewamalage et al. [51]

[1] `https://github.com/Arturus/kaggle-web-traffic`

provide a recent extensive study comparing RNN-based forecasting methods to classical approaches such as ARIMA and ETS, which concludes that RNN-based forecasting methods can be competitive to classical approaches in many scenarios.

Attention-only models The transformer [131] has been introduced in the domain of Natural Language Processing (NLP) to enable efficient parallel training of sequence-to-sequence problems. Recently, the transformer has been adopted to the problem of probabilistic forecasting by Li et al. [80], who employ a decoder-only model with convolutional sparse attention to reduce memory consumption of the base transformer and enhance its forecasting performance. This method currently achieves state-of-the-art results on the various public datasets commonly used in probabilistic forecasting papers.

Temporal convolutional networks TCNs employ stacked dilated Convolutional Neural Networks (CNNs) to overcome the limitations of RNNs [15]. The benefit of this architecture is that it allows for fast training like the transformer model, whilst having significantly lower memory consumption. This enables larger batch sizes during training which, in turn, speeds up training. Temporal convolutional networks have been applied to autoregressive problems in speech generation with Wavenet [130], and achieved the second place in a Kaggle sales forecasting competition [69]. A more recent method closely related to ours is introduced in [113], where a matrix factorization method is combined with a standard TCN to provide point forecasts.

Our method is different in that we do not employ matrix factorization, and study the problem of *probabilistic* forecasting instead of *point* forecasting. Probabilistic forecasting with TCNs has recently been conducted by Chen et al. [27]. Our work in this chapter differs from this work in that (1) our core temporal module is simpler as it uses fewer components, (2) our focus is on parameter efficient probabilistic forecasting, and (3) we introduce a forward-looking module to encode future covariates.

2.3 Methodology

We introduce the problem setting of probabilistic forecasting, and describe the core components of our method. We end the section by detailing the choice of our loss function.

Probabilistic forecasting A time series is a sequence of ordered measurements $\{y_t, y_{t+1}, \dots\}$, in which we assume the timestep t to be constant (e.g., a day, an hour). Denote the set of N time series as $\{\mathbf{y}_{i,1:t_0}\}_{i=1}^{N}$ and $\{\mathbf{a}_{i,1:t_0}\}_{i=1}^{N}$ a set of additional attributes. We are interested in modeling the conditional distribution

$$\begin{aligned} p(\mathbf{y}_{i,t_0:T}|\mathbf{y}_{i,1:t_0},&\mathbf{a}_{i,1:T};\mu_\theta,\sigma_\theta) \\ &= \prod_{t=t_0}^{T} p\left(\mathbf{y}_{i,t}|\mathbf{y}_{i,1:t-1},\mathbf{a}_{i,1:T};\mu_\theta,\sigma_\theta\right) \\ &= \prod_{t=t_0}^{T} p\left(\mathbf{y}_{i,t}|\mathbf{x}_{i,1:T};\mu_\theta,\sigma_\theta\right), \end{aligned} \tag{2.1}$$

where t_0 and T denote the start and end of the forecast, respectively, and $(\mu_\theta, \sigma_\theta)$ the location and scale parameters of a distribution parameterized by θ, which we learn with our model. We estimate separate output distribution parameters for each timestep in the forecast window. The input to our network consists of the concatenation of lagged target variables $\mathbf{y}_{lag}$ and additional attributes in the form of numerical covariates $\mathbf{a}_{cov}$ (e.g., a day-of-the-week indicator) and categorical covariates $\mathbf{a}_{cat}$ (e.g., a time series identifier). We denote this concatenation by $\mathbf{x}$.

Dilated convolutions We observe that future covariates on time series are usually available at the time of forecasting, such as item information, or time indicators (e.g., day-of-the-week or indicators for promotion days or holidays). We would like to encode all such knowledge of the future into a latent state on which the forecast of the current timestamp can be conditioned. To achieve this, we create a TCN consisting of dilated convolutions that look *forward* in time, instead of backward. More formally, the *forward* dilated convolution operation F on an element s of a sequential input $\mathbf{x} \in \mathbb{R}^n$ and a filter $f \in \mathbb{R}^k$ can be defined as [15]:

$$F(s) = \sum_{j=0}^{k-1} f(j) \cdot \mathbf{x}_{s+d \cdot j}, \tag{2.2}$$

with filter size k, dilation factor d, and $s + d \cdot j$ indicating the forward steps. For a backward (causal) dilated convolution, $s + d \cdot j$ in (2.2) becomes $s - d \cdot j$. We stack N layers with dilation 2^{i-1} for layer i to obtain a 'receptive field' for the TCN of size $1 + (k - 1)(2^N - 1)$. The receptive field is the effective sequence length the network can condition its forecast on. For example, if a forecasting problem requires taking into account sequences of a length up to 500 time steps, example valid options for the kernel size and number of layers N would be $(k = 3, N = 8)$, $(k = 7, N = 7)$ or $(k = 11, N = 6)$. It is necessary to add right (left) padding of size $(k - 1) \cdot 2^{i-1}$ for the forward (backward) convolution at each i^{th} layer to maintain a constant sequence length throughout the network. Many existing forecasting methods already incorporate future covariate information. However, our approach is novel due to the fact that:

(1) The temporal structure of the covariates is maintained vis-a-vis feed-forward networks to incorporate such information, and

(2) Dilated convolutions enable more efficient training than, e.g., a bidirectional LSTM, which requires more sequential operations during training (we refer to Section 2.5 for a discussion on the computational efficiency).

We considered an alternative design with an unmasked multi-head attention module [131] as a 'look-forward' layer. We decided against this design however, as preliminary experiments indicated relatively poor results in terms of parameter / performance balance (many additional parameters required for incremental accuracy gains) – see Section 2.5 for a discussion on the complexity of multi-head attention layers.

Temporal blocks Inspired by the work in [15, 130], we construct temporal blocks by stacking dilated convolutions. A single layer of our TCN is displayed in Figure 2.1. Each TCN layer in our network consists of a block that contains a dilated convolution, a GELU activation

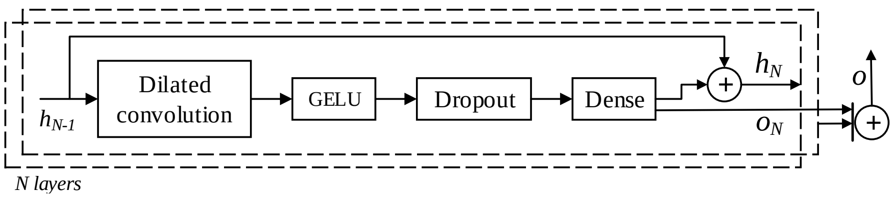


Figure 2.1: Temporal block.

[50], dropout and a dense layer. A Gaussian Error Linear Unit (GELU) multiplies an input x with the cumulative distribution function $\Phi(x)$ of the Gaussian distribution, i.e.

$$\begin{aligned} \mathrm{GELU}(x) &= xP(X \leq x) = x\Phi(x) \\ &\approx 0.5x\left(1 + \tanh\left[\sqrt{0.5\pi}\left(x + 0.044715x^3\right)\right]\right) \end{aligned} \tag{2.3}$$

and provides a 'softer' activation than the commonly used Rectified Linear Unit (ReLU), which generally improves performance [50].

Each layer produces a hidden state h_N and an output o_N; the latter is summed over all layers to provide the output of the network. Weight normalization [106] is applied to the dilated convolution and dense modules. Our temporal block is simpler than the ones used in WaveNet, as it does not require a *gated activation unit*, but instead relies solely on the GELU activation as non-linearity. In Section 2.5.1, we will show that this simplification has a positive effect on performance.

Our method employs two temporal blocks; the first block consists of N layers and encodes the past observations and covariates of time series with backward dilated convolutions, whereas the second block consists of $N + 1$ layers and encodes future covariates with forward dilated convolutions. The additional layer in the second block is required to enlarge the receptive field of the forward looking module, as the sequence length of the covariates and categorical inputs exceeds the dimension of the input and output sequence length in order to facilitate sufficient look-forward for later timesteps in the forecast. This is to ensure that later timesteps of the forecast also receive sufficient future covariate information to condition the forecast on. A 3-layer example of how these temporal blocks employ backward and forward dilated convolutions to condition the forecast is pictured in Figure 2.2.

Within the forward TCN, grouped convolutions are employed to reduce the number of parameters needed. The intuition behind this

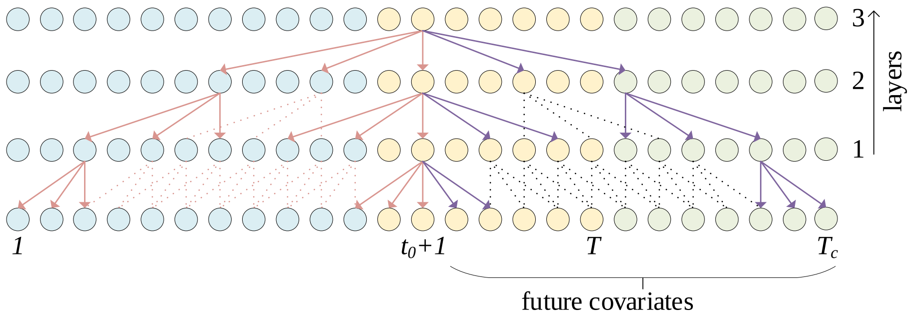


Figure 2.2: An illustration of how 3 stacked TCN layers enable conditioning the forecast at $t = t_0 + 1$ on both past and future information using forward and backward dilated convolutions with kernel size 3 and dilation 2^{i-1} for the i-th layer. The blue dots represent the input sequence, the yellow dots the output sequence and the green dots the additional future covariates on which the forecast can be conditioned. The red connections indicate the *backward* looking convolutions, and the purple connections the *forward* looking convolutions. For clarity purposes, some inner convolutional connections are shown with dashed lines.

choice is that the future covariates usually contain less information than the past covariates and lags, and thereby require fewer parameters for encoding. We employ dense layers with dropout before the temporal blocks to scale the input dimensions to the hidden dimension of the network. Finally, dense output layers provide the location μ and scale σ of the output distribution. These layers are activated with a softplus activation to ensure that their output remains positive (this activation may be removed in case the possibility of a negative output is desired) and a small number ϵ is added to ensure numerical stability of the scale output. The pseudocode of our resulting method, which we call Bidirectional Temporal Convolutional Network (BiTCN), is shown in Algorithm 2.1 and graphically depicted in Figure 2.3.

Output distribution Although BiTCN allows the use of many parametric (e.g., Gaussian) or non-parametric (e.g., Quantile) distribution functions, we use a Student's t-distribution with three degrees of freedom in our experiments as fixed parameterized distribution. The probability

Algorithm 2.1 BiTCN pseudocode

Input: $\mathbf{y}_{lag} \in \mathbb{R}^{T \times d_{batch} \times d_{input}}$, $\mathbf{a}_{cov} \in \mathbb{R}^{T_c \times d_{batch} \times d_{cov}}$, $\mathbf{a}_{cat} \in \mathbb{R}^{T_c \times d_{batch} \times d_{cat}}$
Output: μ, σ

```
1: $\mathbf{a}_{emb} = \texttt{Embedding}(\mathbf{a}_{cat})$
2: $\mathbf{x}_{lag} = \texttt{Concat}(\mathbf{y}_{lag}, \mathbf{a}_{cov}[:d_l], \mathbf{a}_{emb}[:d_l])$
3: $\mathbf{x}_{cov} = \texttt{Concat}(\mathbf{a}_{cov}, \mathbf{a}_{emb})$
4: $\mathbf{h}_{lag} = \texttt{Drop}(\texttt{Dense}(\mathbf{x}_{lag}))$
5: $\mathbf{h}_{cov} = \texttt{Drop}(\texttt{Dense}(\mathbf{x}_{cov}))$
6: $\mathbf{o}_{lag} = \texttt{Backward Temporal Block}(\mathbf{h}_{lag})$
7: $\mathbf{o}_{cov} = \texttt{Forward Temporal Block}(\mathbf{h}_{cov})$
8: $\mathbf{o} = \texttt{Concat}(\mathbf{o}_{cov}[:d_l], \mathbf{o}_{lag})$
9: $\mu = \texttt{SoftPlus}(\texttt{Dense}(\mathbf{o}))$
10: $\sigma = \texttt{SoftPlus}(\texttt{Dense}(\mathbf{o})) + \epsilon$
11: **return** $\mu, \sigma$
```

density function of this distribution for a variable $\mathbf{y}$ is given by:

$$f(\mathbf{y}) = \frac{2}{\pi\sqrt{3}\left(1 + \frac{\mathbf{y}^2}{3}\right)^2} \tag{2.4}$$

and we aim to minimize the log-likelihood loss

$$\mathcal{L}(\theta|\mathbf{y}) = \log\left(f\left(\frac{\mathbf{y} - \mu_\theta}{\sigma_\theta}\right)\right), \tag{2.5}$$

where μ and σ are the outputs of our network for a given time series at a certain timestep. There are several benefits of using this distribution. First, the Student's t-distribution can be seen as a fat-tailed version of the Gaussian distribution, which makes it a natural candidate for estimating real-valued quantities without prior information on the true distribution. The benefit of a fat-tailed distribution is two-fold:

(1) This allows our model to capture rare events better, and

(2) The log-likelihood loss is numerically stable around a large range of input values.

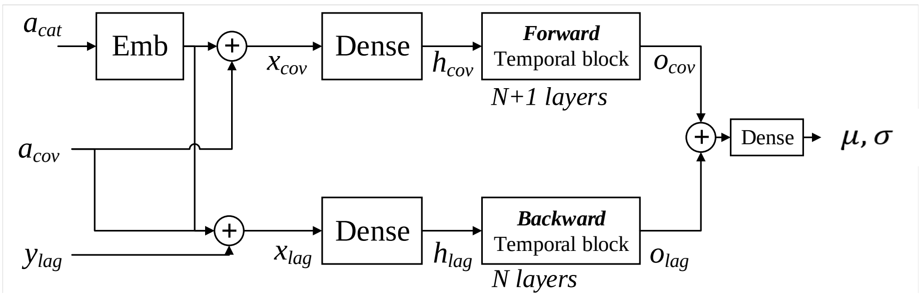


Figure 2.3: Overall network of BiTCN. First, in the top half, the categorical inputs a_{cat} are embedded using an embedding layer. The result is combined with the available covariates a_{cov} to result in the input X_{cov}, which includes all available (covariate) information about the past and future. This is then led through a linear layer and a temporal block that looks forward across all the future information. In the bottom half, the lagged target y_{lag} is combined with the available historical covariate and categorical information to result in the input X_{lag}. This is then led through a linear layer and an autoregressive temporal block (i.e. standard dilated convolutions). The outputs of both halfs are joined by adding them together at the right time stamp and a final linear layer provides the location and scale of our output distribution.

The disadvantage of fixing a distribution a priori is the imposition of a defined distribution on the output, which may not reflect the true underlying distribution properly. However, we observe this to be no issue in practice.

To facilitate a better comparison between architectures, our loss function is used for each method in our experimental section comparing forecasting accuracy across competing methods. We will demonstrate the benefits of choosing the Student's t(3)-distribution over the Gaussian distribution in Section 2.5.3.

2.4 Experimental Setup

Datasets We employ four real-world datasets to investigate the probabilistic forecasting performance of the BiTCN architecture.

- `Electricity` [129] Hourly electricity consumption for 370 clients over a 3 year period. The task is to forecast electricity consumption per client for the next 24 hours given the previous 7 days. The training/validation/test split is 80/10/10 based on date.

- `Traffic` [128] 15 months of hourly data describing the occupancy rate, between 0 and 1, of different car lanes of the San Francisco bay area freeways. We forecast occupancy rate per lane for the next 24 hours given the previous 7 days. The training/validation/test split is 80/10/10 based on date.
- `Favorita` [66] A dataset from Kaggle that contains 4 years of daily sales data for store-product combinations taken from a retail chain. The task is to forecast the log sales for all product-store combinations for the next 30 days based on the previous 90 days. The training data consists of data between 01-01-2013 and 02-09-2013. We validate on samples between the dates 03-09-2013 and 03-10-2013, and we test on the remaining data up to 31-03-2014. The target and lagged input variables are log-transformed.
- `WebTraffic` [67] 145k time series of daily Wikipedia pageviews in the period 2015-2017. The task is to forecast the number of pageviews for the next 30 days given the previous 90 days. We use a selection of 10k time series with the largest number of pageviews in the training period (which is up to 31-12-2016).

We provide more detailed descriptions of the datasets in Table 2.A.1 in 2.A.

The `Electricity` and `Traffic` datasets provide an indication of forecasting performance on relatively regular time series, with a small number of future covariates available to condition our forecast on (i.e., a few time-based features such as day of the week and hour of the day). `Favorita` and `WebTraffic` contain more irregular time series, and the first also contains a rich set of covariates. For each dataset, we add categorical covariates and numerical covariates consisting mainly of time series identifiers, time indicators (day-of-week, month) and other indicators (e.g., holiday). The time indicators are represented by two Fourier terms [58] to represent the periodic nature of, e.g., days or months (i.e., the first day of the week should be close to the last day of the week) as follows:

$$\text{covariate}_{\sin} = \sin\left(\text{covariate} \cdot \left(2 \cdot \frac{\pi}{\text{period}}\right)\right) \tag{2.6}$$

$$\text{covariate}_{\text{cos}} = \cos\left(\text{covariate} \cdot \left(2 \cdot \frac{\pi}{\text{period}}\right)\right). \tag{2.7}$$

The categorical and numerical covariates are assumed to be known a priori, i.e., we include these variables in the forward-look of BiTCN. For all datasets except `Favorita`, we employ the scaling mechanism from [107] for normalizing the outputs and lagged outputs to our network:

$$\overline{\mathbf{y}} = \frac{\mathbf{y}}{1 + \frac{1}{t_0}\sum_{i=0}^{t_0} \mathbf{y}_i}. \tag{2.8}$$

Again, we refer to Table 2.A.1 in the Appendix for further details on the datasets.

Baseline models We compare BiTCN against five state-of-the-art forecasting methods of different neural architectures:

- *DeepAR* [107]. An LSTM-based generic probabilistic forecasting framework.
- *ML-RNN* [133]. An encoder-decoder probabilistic forecasting framework that uses an LSTM to encode past observations and covariates and two MLP decoders to generate probabilistic forecasts. This method also uses future covariate information to condition its forecast on.
- *transformerConv* [80]. A transformer-based forecasting model augmented with causal convolutions, which has shown state-of-the-art results on the `Electricity` and `Traffic` dataset.
- *TCN* [15]. A standard TCN, as employed for probabilistic forecasting by [27].
- *WaveNet* [130]. which has been used to achieve the second place in a Kaggle sales forecasting competition [69].

In addition, we compare BiTCN against three traditional forecasting methods and one popular machine learning package:

- *Seasonal Naive*. The seasonal naive baseline, where we simply take the observation from the last period as our forecast. Since our datasets exhibit clear seasonality (either daily or weekly), this provides a sensible baseline.

- *ETS* [54]. The exponential smoothing method, including Holt-Winters' seasonal and trend components.
- *Theta* [11]. The theta method, a univariate forecasting method that uses the local curvature of the time series.
- *LightGBM* [68]. A highly popular Gradient Boosting package on which the winning solution of the M5 forecasting competition [88] was based.

Training and optimization We implemented, trained and evaluated BiTCN and each neural network method using PyTorch [96] and our goal is available online on Github.[2] For DeepAR, we used the hyperparameters from [107], for ML-RNN from [133] and for transformerConv, we used the hyperparameters from [80]. For TCN, WaveNet and BiTCN we selected a kernel size and hidden size to establish a comparable parameter budget as DeepAR, as this is the state-of-the-art in terms of parameter budget compared against in this study. An overview of the key model hyperparameters is given in Table 2.1. Finally, we optimized learning rate and batch size for each dataset and method by performing a limited grid search using the following settings:

- Learning rate: $\{0.001, 0.0005, 0.0001\}$.
- Batch size: $\{128, 256, 512\}$, except for DeepAR and ML-RNN for which we used $\{64, 128, 256\}$.

We run each experiment for 100 epochs with an early stopping criterion of 5 epochs. We train, validate and test each method for 5 different random seeds for the neural network weight initialization. We optimize the parameters of each method using Adam [72]. The results of the limited grid search for each method and dataset are given in Figures 2.A.1–2.A.4 in the Appendix. For the forecasting performance evaluation, we report the evaluation metrics on the test set of the best performing {(learning rate, batch size)} combination according to Figures 2.A.1–2.A.4.

For the traditional forecasting methods, we use the `statsmodels` Python package and fit a model on each input sequence of our test set. For the probabilistic forecasts, we use the generated prediction intervals if the

[2]`https://github.com/elephaint/pedpf`

Table 2.1: Key model hyperparameters.

	DeepAR	ML-RNN	TransformerConv	TCN	WaveNet	BiTCN
state size	40	30	$d_{emb} + d_{cov} + d_{lag}$	20	20	12
layers	3	1	3	5	5	5
kernel size	n.a.	n.a.	9	9	9	9
heads	n.a.	n.a.	8	n.a.	n.a.	n.a.
dropout	0.1	n.a.	0.1	0.1	n.a.	0.1
receptive field	∞	∞	∞	249	249	497

method provides these. For LightGBM, we create 9 separate models for each quantile $0.1, 0.2, \ldots, 0.9$ using quantile regression and recursively apply each model to forecast each time step in our forecast. We use Optuna [7] to find the best hyperparameters for LightGBM, which we apply to all datasets.

Evaluation We employ a set of point accuracy metrics and range accuracy metrics to evaluate forecasting performance. For point accuracy, we use symmetric Mean Absolute Percentage Error (sMAPE) and Normalized Root Mean Squared Error (NRMSE) [8]:

$$sMAPE = \frac{1}{n} \sum_{t=t_0}^{n} \frac{2|\mathbf{y}_t - \hat{\mathbf{y}}_t|}{|\mathbf{y}_t| + |\hat{\mathbf{y}}_t|} \tag{2.9}$$

$$NRMSE = \frac{\sqrt{\frac{1}{n} \sum_{t=t_0}^{n} (\mathbf{y}_t - \hat{\mathbf{y}}_t)^2 \cdot \mathbb{1}_{\bar{\mathbf{y}}_t \neq 0}}}{\sum_{t=t_0}^{n} |\mathbf{y}_t| + \mathbb{1}_{\bar{\mathbf{y}}_t = 0}}, \tag{2.10}$$

where $n = T - t_0$ denotes the number of forecast steps and $\mathbb{1}_{\bar{\mathbf{y}}_t=0}$ is an indicator function to scale the metric when the observed target value equals zero. Note that the sMAPE ranges from $0 - 200\%$. The NRMSE is essentially the normal root mean squared error, but normalized to account for differences in values of each individual time series. Each forecasting metric has its advantages and disadvantages; we also considered using a popular one-step ahead metric such as the MASE [57], but considered this inappropriate for our task (which is multistep forecasting). For range accuracy, the normalized quantile loss function [107] is used for the quantiles $p = \{0.1, 0.5, 0.9\}$:

$$Q(\mathbf{y}_i, \hat{\mathbf{y}}_i, p) = 2 \cdot |(\mathbf{y} - \hat{\mathbf{y}}) \cdot (\mathbb{1}_{\mathbf{y} \leq \hat{\mathbf{y}}} - p)| \tag{2.11}$$

$$Q(\mathbf{y}, \hat{\mathbf{y}}, p) = \frac{\sum_i Q(\mathbf{y}_i, \hat{\mathbf{y}}_i, p)}{\sum_i \mathbf{y}_i}. \tag{2.12}$$

Finally, we show the mean quantile performance over the 9 quantiles in the range $p = \{0.1, 0.2, \ldots, 0.9\}$.

2.5 Results & Discussion

First, we demonstrate the forecasting performance of BiTCN on a set of real-world datasets to substantiate our claim that BiTCN achieves state-of-the-art forecasting performance with fewer parameters than competing methods (Section 2.5.1). Second, we will show the benefit of our architecture by studying the forecasting efficiency in terms of model complexity, training time and energy cost (Section 2.5.2). Finally, we study the impact of our design choices, such as employing the Student's t(3)-distribution (Section 2.5.3), the effect of our forward-looking module (Section 2.5.4) and how BiTCN's performance is impacted by its hyperparameters (Section 2.5.5).

2.5.1 Forecasting Effectiveness

We report the forecasting performance in Tables 2.2–2.3. Although similar model dimensions are used for each method across the experiments, parameter counts for each model may be different per experiment due to the size of the embedding dimension required to embed the categorical input vector $\mathbf{a}_{cat}$, which has a different dimension for each dataset.

We observe the following per dataset:

- `Electricity`: BiTCN performs on par with the best methods WaveNet and TransformerConv.
- `Traffic`: BiTCN performs on par with the best methods.
- `Favorita`: BiTCN outperforms or performs similar to competing methods on all metrics.
- `WebTraffic`: BiTCN performs similar to best-performing method TCN on mean quantile loss and in line with other methods on the other metrics.

On average, BiTCN ranks highest at an overall average ranking of 1.75. Given these observations, we conclude that BiTCN can achieve state-of-the-art results on a set of real world probabilistic forecasting tasks whilst using significantly fewer parameters than the second best performing method, TransformerConv (average ranking of 2.25).

Secondly, even though not a primary objective of our paper, we confirm the findings from [80, 107] that traditional methods such as Seasonal Naive, ETS and Theta are outperformed by neural network methods on the task of multistep probabilistic forecasting.

Finally, we find that LightGBM performs reasonable but not as well as expected. We attribute this to our experimental setup. First, to facilitate a like-for-like comparison to neural network-based methods, we use the same features for LightGBM as we provide to our neural networks. In practice, practitioners often spend a lot of time to engineer features that provide a better signal to a LightGBM model, which typically improves performance. Second, we apply the LightGBM model recursively, which means that any bias and variance that enters the model may be propagated to future timesteps. This issue is confirmed by results from the M5 competition, where most participants created separate LightGBM models for every timestep to avoid this bias/variance propagation. However, for our setting this would imply creating not only a separate model per quantile, but also per timestep. This would give LightGBM a somewhat unfair advantage compared to neural network based methods.

2.5.2 Forecasting Efficiency

Model complexity As can be seen from Tables 2.2–2.3, BiTCN requires almost an order of magnitude fewer parameters than the TransformerConv. This is an indication that our architecture is more efficient, and possibly a better choice for the task of probabilistic forecasting than existing neural architectures. To understand its computational complexity, we are mainly interested in the sizes of the following parameters (we study the impact of varying these parameters in Section 2.5.2):

- N, the number of layers of a neural network;
- k, the kernel size of a convolution;
- T, the sequence length used in the network;

Table 2.2: Forecasting results on various point and range accuracy metrics for the `Electricity` and `Traffic` dataset. For the neural network methods, we report mean metrics over 5 different seeds of parameter initializations per method, with standard deviation of the metric in brackets. Lower is better, bold indicates best method for the metric. Rank denotes the rank of the methods across the four metrics for each dataset (lower is better).

	No.	Point metrics		Range metrics		Rank
Dataset/Method	parameters	sMAPE	NRMSE	Q(0.5)	mQ	
`Electricity`						
Seasonal Naive	0	0.112	0.719	0.078	0.078	8
ETS	100k	0.201	1.755	0.152	0.136	11
Theta	20k	0.177	1.575	0.141	0.164	10
LightGBM	2M	0.089	0.679	0.065	0.077	5
DeepAR	45k	0.099 (0.0044)	0.671 (0.0142)	0.069 (0.0010)	0.057 (0.0009)	7
ML-RNN	2.9M	0.115 (0.0029)	0.829 (0.0555)	0.086 (0.0030)	0.068 (0.0024)	9
TCN	46k	0.103 (0.0017)	0.668 (0.0121)	0.068 (0.0011)	0.056 (0.0010)	6
WaveNet	48k	0.093 (0.0026)	**0.646 (0.0095)**	0.063 (0.0005)	0.052 (0.0005)	2
TransformerConv	415k	**0.083 (0.0014)**	0.658 (0.0237)	**0.062 (0.0012)**	0.052 (0.0010)	1
BiTCN	49k	0.089 (0.0009)	0.648 (0.0090)	0.063 (0.0003)	**0.052 (0.0003)**	2
`Traffic`						
Seasonal Naive	0	0.351	0.667	0.285	0.285	9
ETS	100k	0.483	0.693	0.371	0.348	10
Theta	20k	0.512	3.413	0.980	0.970	11
LightGBM	2M	0.128	0.358	0.111	0.114	4
DeepAR	57k	0.179 (0.0182)	0.408 (0.0274)	0.131 (0.0141)	0.111 (0.0126)	8
ML-RNN	2.4M	0.140 (0.0021)	0.387 (0.0026)	0.123 (0.0017)	0.102 (0.0013)	7
TCN	57k	0.150 (0.0128)	0.373 (0.0074)	0.117 (0.0027)	0.098 (0.0023)	6
WaveNet	60k	0.165 (0.0054)	0.357 (0.0028)	0.108 (0.0018)	0.091 (0.0015)	3
TransformerConv	372k	0.130 (0.0035)	**0.353 (0.0020)**	**0.105 (0.0020)**	**0.089 (0.0014)**	1
BiTCN	61k	**0.127 (0.0005)**	0.372 (0.0142)	0.108 (0.0005)	0.091 (0.0004)	2

Table 2.3: Forecasting results on various point and range accuracy metrics for the `Favorita` and `WebTraffic` dataset. For the neural network methods, we report mean metrics over 5 different seeds of parameter initializations per method, with standard deviation of the metric in brackets. Lower is better, bold indicates best method for the metric. Rank denotes the rank of the methods across the four metrics for each dataset (lower is better).

	No.	Point metrics		Range metrics		Rank
Dataset/Method	parameters	sMAPE	NRMSE	Q(0.5)	mQ	
`Favorita`						
Seasonal Naive	0	1.001	4.541	1.053	1.053	10
ETS	100k	1.024	4.297	1.021	0.842	8
Theta	20k	1.060	2.369	0.844	0.922	8
LightGBM	2M	0.879	1.562	0.489	0.396	5
DeepAR	61k	**0.574 (0.1823)**	1.680 (0.1040)	0.543 (0.0216)	0.422 (0.0150)	5
ML-RNN	1.9M	0.689 (0.0132)	1.406 (0.0483)	0.461 (0.0079)	0.362 (0.0064)	4
TCN	61k	0.612 (0.0989)	1.323 (0.0562)	0.440 (0.0210)	0.350 (0.0134)	3
WaveNet	66k	0.711 (0.0261)	1.623 (0.1773)	0.512 (0.0427)	0.405 (0.0319)	7
TransformerConv	210k	0.673 (0.0046)	1.319 (0.0602)	0.439 (0.0179)	**0.346 (0.0129)**	1
BiTCN	66k	0.674 (0.0015)	**1.317 (0.0179)**	**0.432 (0.0049)**	0.347 (0.0033)	1
`WebTraffic`						
Seasonal Naive	0	0.357	4.709	0.414	0.414	10
ETS	100k	0.311	4.096	0.350	0.359	8
Theta	20k	0.338	4.132	0.340	0.476	9
LightGBM	2M	0.260	4.022	0.273	1.002	6
DeepAR	238k	0.279 (0.0054)	4.003 (0.2526)	0.282 (0.0060)	0.244 (0.0050)	5
ML-RNN	2.4M	0.246 (0.0064)	4.027 (0.0269)	0.270 (0.0048)	0.234 (0.0051)	4
TCN	239k	**0.234 (0.0034)**	**3.718 (0.1692)**	**0.253 (0.0048)**	**0.231 (0.0046)**	1
WaveNet	241k	0.236 (0.0037)	3.953 (0.3083)	0.268 (0.0129)	0.244 (0.0107)	2
TransformerConv	630k	0.246 (0.0065)	4.191 (0.3368)	0.301 (0.0225)	0.273 (0.0198)	6
BiTCN	242k	0.254 (0.0062)	3.988 (0.2177)	0.267 (0.0057)	0.232 (0.0041)	2

- d_h, the hidden dimension of the network; and
- d_{out}, the number of output channels of a convolutional layer.

The computational complexity of each of our TCN layers can be computed by adding the computational complexity of respectively the convolution layer, the activation, and the dense output layer:

$$\begin{aligned} O(k \cdot T \cdot d_h \cdot d_{out} + T^2 + d_{out} \cdot d_h \cdot T^2) = \\ O(k \cdot T \cdot d_h^2 + d_h \cdot T^2), \end{aligned} \tag{2.13}$$

where in BiTCN, $d_{out} = 4 \cdot d_h$. Hence, the complexity for an N-layer network of our architecture is $O(N \cdot (k \cdot T \cdot d_h^2 + d_h \cdot T^2))$ with $O(1)$ sequential operations per layer. In comparison, an LSTM-based architecture such as DeepAR has a computational complexity of $O(N \cdot (T \cdot {d_h}^2))$, but requires $O(T)$ sequential operations, which significantly slows down training when sequence length increases.

Finally, transformer-based architectures have a computational complexity of $O(N \cdot (T \cdot {d_h}^2 + d_h \cdot T^2))$ and require $O(1)$ sequential operations [131], but through the use of dilated convolutions in the self-attention mechanism the computational complexity of the transformer architecture of [80] becomes $O(N \cdot (k \cdot T \cdot {d_h}^2 + d_h \cdot T^2))$, with k the kernel size of the dilated convolutions. Hence, the computational complexity of the TransformerConv architecture is equal to that of BiTCN, however in practice this leads to different outcomes. Even though BiTCN requires more layers to ensure a sufficient receptive field, it requires a much smaller hidden dimension throughout the network. In contrast, the TransformerConv network requires fewer layers but a higher hidden dimension in order to support sufficient model capacity for each of the attention heads.

Model complexity compared to non-neural methods We find that BiTCN requires signficantly fewer parameters than LightGBM. What causes this? Suppose we would like to have a probabilistic forecast for 28 days, for 9 quantiles. This requires at least 9 models in the GBT setting. In our setting, each GBT consists of approximately 2000 trees, and is trained with a maximum number of leaf nodes of 127. Hence, this yields 254k parameters. Then, we have 9 such models, yielding approximately 2M parameters. If one would opt for separate models for each forecast day, one would require $28 \cdot 2M = 64M$ parameters, which is orders of magnitude larger than comparable NN-based solutions. What causes this difference in model size? Ultimately, this is due to the fact that NN methods are better representational learning methods, or stated differently, these methods better learn to compress the data. In a NN-based solution, an existing set of parameters is modified during learning. GBTs however, work incrementally, and add parameters for each iteration. Hence, these models do not compress the data as much as the NN does, yielding models that have a higher complexity.

Training time One of the benefits of a model with fewer parameters is that *ceteris paribus* it can be trained faster. Therefore, we compare training times for each method in the left plane of Figure 2.4. For each method, the left side of Figure 2.4 shows the mean quantile loss on the test set as compared against the training time. The training times are normalized against a DeepAR baseline. We observe that BiTCN is the only algorithm that sits in the lower left corner – which implies smallest training time with the lowest quantile loss on the test set – for each of the datasets. Also, we observe that LightGBM's training time is comparable to those of NN methods, which is mostly due to its requirement to create separate models for each individual quantile in the forecast. Note that the latter finding is subject to hardware considerations, as the LightGBM models are trained on CPU whilst the neural models are trained on a GPU.

Energy cost Even though one method may train faster than the other, the energy cost for training may still exceed the energy cost of training other models, due to an increased resource consumption. Therefore, we analyze the consumed energy for training each model in the right half plane of Figure 2.4. For this experiment, we ran each method separately and sequentially for a single epoch, and measured the average GPU board power draw of a nVidia GTX 1080Ti using GPU-Z. During each experiment, we fixed the number of CPU threads available to the training process and there were no other processes running on the GPU used for measuring energy cost. The obtained result indicates the total average amount of energy in Joules that is required to compute a single epoch for each method. This energy consumption is then multiplied with the average number of epochs (across the 5 seeds) required to train each method to obtain the overall GPU energy cost. Finally, we normalize

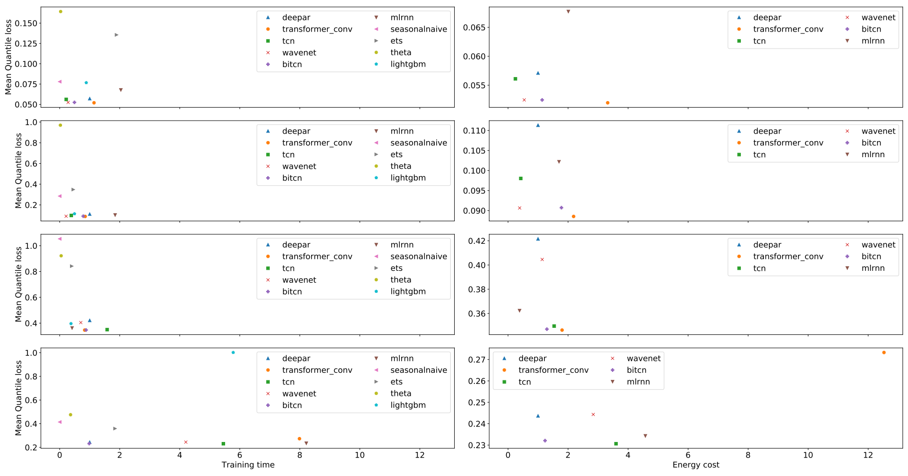


Figure 2.4: Running time (left) and energy cost (right) compared to mean quantile loss on the test set for respectively `Electricity` (top), `Traffic`, `Favorita` and `Webtraffic` (bottom) datasets, where DeepAR is the baseline.

this energy cost against a DeepAR baseline. BiTCN is the strongest performer across all the datasets, as it sits in the lower left corner on each graph. We now see a more clear distinction between TransformerConv and the other methods, and its performance is now less favourable, as it requires a higher energy consumption on every dataset compared to the other methods, and we see energy consumption differences of 20% up to an order of magnitude compared to BiTCN. In practice, this result is beneficical for practitioners who are interested in optimizing electricity and cooling costs, which commonly represent a significant portion of data center operating cost [122].

Memory usage The transformer's memory consumption scales quadratically with sequence length [80], as all activations of the multi-head attention layers need to be stored during a forward pass of the network to use during the backward pass. In contrast, for temporal convolutional architectures memory consumption predominantly scales with (1) the number of layers required to obtain a sufficient receptive field, and (2) the number of parameters of these layers. This enables training these models using less memory, which in turn enables the use of cheaper resources. In Table 2.4 we show the GPU memory consumption during training

Table 2.4: GPU memory consumption in Gigabytes during training of BiTCN compared to TransformerConv.

Batch size / Dataset	BiTCN	TransformerConv	Ratio (x)
128			
Electricity	1.54	3.43	2.23
Traffic	1.52	3.37	2.21
Favorita	1.34	2.15	1.60
WebTraffic	1.36	2.20	1.61
256			
Electricity	1.91	6.61	3.46
Traffic	1.90	6.55	3.45
Favorita	1.49	3.47	2.33
WebTraffic	1.51	3.46	2.30
512			
Electricity	2.62	10.20	3.89
Traffic	2.60	10.18	3.91
Favorita	1.77	5.95	3.36
WebTraffic	1.78	5.95	3.33

compared between BiTCN and TransformerConv across datasets. When comparing similar batch sizes, we observe a difference in memory consumption of 2–4x. In practice, this means that BiTCN models can be trained on much cheaper GPUs. For example, a TransformerConv model with batch size 512 requires a high-end video card such as the nVidia RTX 2080Ti (which is equipped with 11GB of GPU memory) to train the model on the `Electricity` and `Traffic` datasets, whereas a similar batch size BiTCN model would only require a low-end RTX 2060. The first card typically retails for over USD 1000, whereas the latter can be acquired starting from USD 300 on Amazon.

2.5.3 Effect of Student's t(3)-distribution

To illustrate the benefits of using a Student's t(3)-distribution for probabilistic forecasting, we re-run the experiments from Section 2.5.1 on the `Electricity` and `Traffic` datasets for BiTCN using a parameterized Gaussian output distribution. We keep the same training settings as before, however we note that the Gaussian distribution in our forecasting setting requires clipping gradients to achieve a stable training regime. This requires tuning yet another hyperparameter – the maximum gradient norm. Why is this clipping necessary? It is a direct consequence of

Table 2.5: Forecasting results of BiTCN on various point and range accuracy metrics comparing Student's t(3)-distribution with the Gaussian distribution as loss function. Lower is better, bold indicates best loss function for the method. We report mean metrics over 5 different seeds per method with standard deviation in brackets.

	Point metrics		Range metrics	
Dataset / Method	sMAPE	NRMSE	Q(0.5)	mQ
`Electricity`				
Gaussian	0.117 (0.0054)	0.713 (0.0280)	0.076 (0.0036)	0.062 (0.0023)
Student's t(3)	**0.089 (0.0009)**	**0.648 (0.0090)**	**0.063 (0.0003)**	**0.052 (0.0003)**
`Traffic`				
Gaussian	0.175 (0.0089)	0.404 (0.0763)	0.141 (0.0016)	0.115 (0.0012)
Student's t(3)	**0.127 (0.0005)**	**0.372 (0.0142)**	**0.108 (0.0005)**	**0.091 (0.0004)**

the 'thin-tailedness' of the probability density function of the Gaussian, which is illustrated in Figure 2.5. The thin tail of the Gaussian distribution causes the log-probability to exert numerical instability during training. In contrast, a fat-tailed distribution such as the Student's t(3)-distribution enables a stabler training regime. Aside from this practical disadvantage, a thin-tailed distribution is also expected to perform worse on forecasting for processes that do not follow a normal distribution in their output data. We confirm this expectation by observing the results in Table 2.5, where we compare the forecasting performance with a Gaussian as output distribution vis-a-vis the Student's t(3)-distribution. On both `Electricity` and `Traffic` datasets, we see performance differences of 5–15% when using a Gaussian output distribution instead of a Student's t(3)-distribution. For the `Traffic` dataset, the differences are generally larger, which is expected as this dataset is relatively skewed towards zero and hence benefits more from using an output distribution that has a heavier tail such as the Student's t(3)-distribution. Finally, we also observe lower variance in our test scores for the Student's t(3) distribution, which indicates a more stable training regime for this loss function. Our conclusion from this experiment is that the Student's t(3)-distribution improves forecasting performance whilst removing a hyperparameter from the optimization problem and enabling a more stable training regime.

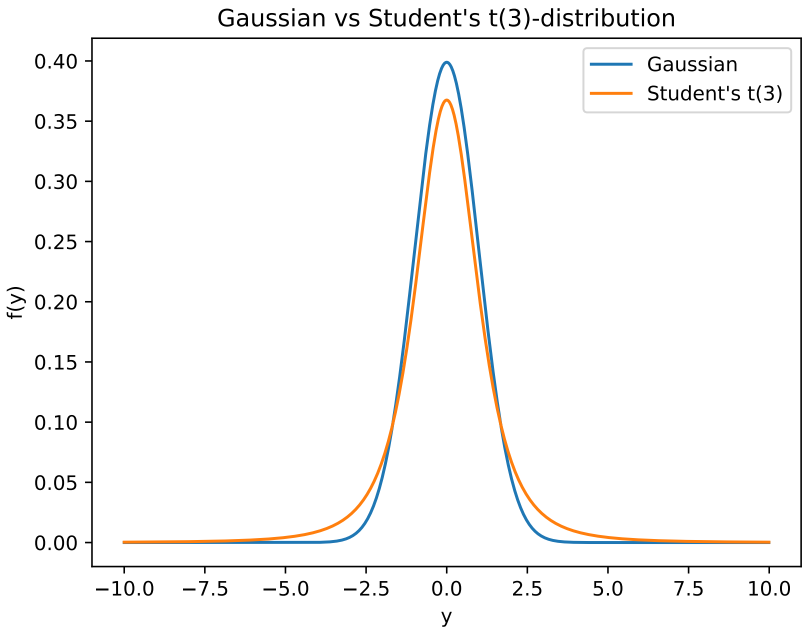


Figure 2.5: The probability density function of the normal distribution (Gaussian with (loc, scale) = (0,1)) compared to the Student's t(3)-distribution with (loc, scale) = (0,1). The normal distribution has a thin tail compared to the Student's t(3)-distribution.

2.5.4 Effect of Forward-looking Module

To study the effect of our forward-looking module, we re-run the experiments from Section 2.5.1 on every dataset where we disable the forward-looking module. We report the results in Table 2.6. We observe a positive impact of about 0–2% from the forward-module in all but one dataset, and especially in the `Traffic` and `Favorita` datasets. Especially for the `Favorita` dataset this is expected, as this dataset provides very informative future covariates (e.g., whether there is a holiday on a particular day). In contrast, the other datasets mostly contain covariates related to the day of the week or day of the month, which are less informative.

Table 2.6: Forecasting results on point and range accuracy metrics, ablating for our forward-looking module. We report mean metrics over 5 different seeds of parameter initializations per method, with standard deviation of the metric in brackets. Lower is better, bold indicates best method for the metric.

	No.	Point metrics		Range metrics	
Dataset/Method	parameters	sMAPE	NRMSE	Q(0.5)	mQ
`Electricity`					
BiTCN	49k	**0.089 (0.0009)**	0.648 (0.0090)	**0.063 (0.0003)**	**0.052 (0.0003)**
BiTCN (w/o forward)	40k	0.089 (0.0012)	**0.647 (0.0050)**	0.064 (0.0005)	0.053 (0.0004)
`Traffic`					
BiTCN	61k	**0.127 (0.0005)**	**0.372 (0.0142)**	**0.108 (0.0005)**	**0.091 (0.0004)**
BiTCN (w/o forward)	52k	0.128 (0.0007)	0.546 (0.3427)	0.112 (0.0062)	0.094 (0.0047)
`Favorita`					
BiTCN	66k	**0.674 (0.0015)**	**1.317 (0.0179)**	**0.432 (0.0049)**	**0.347 (0.0033)**
BiTCN (w/o forward)	58k	0.683 (0.0024)	1.390 (0.0306)	0.460 (0.0084)	0.366 (0.0058)
`WebTraffic`					
BiTCN	242k	0.254 (0.0062)	**3.988 (0.2177)**	0.267 (0.0057)	0.232 (0.0041)
BiTCN (w/o forward)	233k	**0.252 (0.0055)**	4.004 (0.1903)	**0.265 (0.0046)**	**0.232 (0.0032)**

2.5.5 Effect of Hyperparameters

Finally, we briefly study the impact of the choice of key hyperparameters of BiTCN. We reran a set of experiments for several choices of hyperparameters on the `Electricity` and `Traffic` dataset for which we display the results in Table 2.7. We highlight a number of interesting observations:

- Probabilistic forecasting performance of BiTCN seems relatively robust against a wide set of hyperparameter choices, as we commonly observe differences of 0–5% in mean quantile loss when varying hyperparameters, and BiTCN would rank as a top performer among the competing methods in Tables 2.2–2.3 for nearly all of the various hyperparameter settings.
- BiTCN's performance improves when the hidden dimension d_h (and thus the number of parameters) is increased. Conversely, performance also significantly degrades when the hidden dimension is reduced. However, an increased hidden size can result in both higher (`Electricity`) and lower (`Traffic`) training time and energy cost, which is due to the experiment requiring a greater (`Electricity`) and a fewer (`Traffic`) number of epochs when increasing the hidden dimension. Also, the increased running time and energy cost is still less than that of the TransformerConv (ref. Figure 2.4), further demonstrating that our architecture achieves the same performance but does so more efficiently.
- The dropout rate p_d seems very important, as excluding it by setting it to zero results in a large performance hit for both datasets.
- It seems beneficial to increase the kernel size k of the convolutions, as performance generally increases when k is higher.

2.6 Conclusion and Future Work

In this chapter, we set out to find more parameter efficient methods of probabilistic forecasting. We hypothesized that by (1) smartly leveraging future covariate information often available in real-world settings, (2) using a simple convolutional architecture, and (3) employing a Student's

Table 2.7: Sensitivity of sMAPE, mean quantile loss, training time and energy cost of BiTCN when varying key hyperparameters: the batch size b_s, the hidden size d_h, the kernel size k, the number of layers N and the dropout rate p_d. For each row, only the value listed is changed with respect to the base case.

	No. parameters	b_s	d_h	k	N	p_d	sMAPE	mQ	Training time	Energy cost
`Electricity`										
Base case	49k	512	12	9	5	0.1	0.089	0.052	1.00	1.00
		256					0.091 (2.4%)	0.054 (2.7%)	0.82	0.75
		128					0.091 (2.1%)	0.053 (1.2%)	0.94	0.69
	166k		24				0.083 (-6.0%)	0.051 (-1.9%)	1.54	1.82
	19k		6				0.096 (8.1%)	0.054 (2.4%)	0.77	0.68
	39k			3	7		0.095 (7.4%)	0.055 (4.6%)	0.48	0.46
	42k			5	6		0.090 (1.6%)	0.053 (1.5%)	0.76	0.73
	50k			7	6		0.089 (0.3%)	0.053 (0.9%)	0.78	0.76
	55k			11	5		0.087 (-2.3%)	0.053 (0.2%)	1.00	1.02
						0.0	0.101 (13.2%)	0.054 (2.1%)	0.62	0.60
						0.2	0.093 (5.3%)	0.054 (2.2%)	0.71	0.71
						0.3	0.094 (5.6%)	0.053 (1.8%)	1.14	1.09
`Traffic`										
Base case	61k	512	12	9	5	0.1	0.127	0.091	1.00	1.00
		256					0.128 (0.6%)	0.090 (-1.0%)	1.14	1.05
		128					0.126 (-0.6%)	0.090 (-1.2%)	1.56	1.14
	178k		24				0.126 (-1.2%)	0.088 (-2.7%)	0.88	1.02
	31k		6				0.138 (8.3%)	0.099 (9.2%)	0.95	0.79
	51k			3	7		0.128 (1.0%)	0.092 (1.2%)	1.34	1.25
	54k			5	6		0.127 (-0.2%)	0.09 (-0.8%)	1.34	1.24
	62k			7	6		0.126 (-1.2%)	0.089 (-2.1%)	1.22	1.23
	67k			11	5		0.126 (-0.5%)	0.093 (3.0%)	0.91	0.95
						0.0	0.213 (67.7%)	0.143 (57.9%)	0.53	0.55
						0.2	0.13 (1.9%)	0.093 (2.5%)	1.05	1.06
						0.3	0.134 (5.8%)	0.096 (6.2%)	0.78	0.77

t(3)-distribution, it is possible to achieve state-of-the-art probabilistic forecasting performance compared to existing transformer-based methods whilst requiring significantly fewer parameters. We find that our method BiTCN confirms these expectations, as we observe state-of-the-art forecasting effectiveness on a set of real-world benchmarks, even though BiTCN (i) uses an order of magnitude fewer parameters than the second-best transformer-based method, (ii) requires at least 20% less energy, and (iii) about a quarter of the amount of memory to train the model on a GPU.

We believe that these findings qualify BiTCN as a generic probabilistic forecasting method among practitioners, due to its simplicity and computational efficiency.

Even though we observed the benefit of encoding future information to condition the current forecast on, the effect was relatively limited and even absent in some scenarios.

For future work, we would therefore like to further investigate the benefit of this part of BiTCN by applying BiTCN in an industrial retail environment with thousands of products and stores, where a large set of historical data and future covariate information is available to condition a forecast on. An example of such a setting is the M5 forecasting dataset [88].

Secondly, we aim to investigate how our method scales to very long sequences (e.g., as in speech generation problems or high-frequency trading problems), where we expect to see more benefit of the forward-looking module. Finally, we intend to investigate creating a richer set of output distributions, in line with the recent work by Gasthaus et al. [42], which would further generalize our method by removing the choice of an output distribution.

In the next chapter, we turn to models of a different architecture, motivated by working with our industry partners Albert Heijn and bol, a grocery chain and e-commerce platform, respectively. Within the forecasting teams of these companies, neural networks are not often used for (probabilistic) forecasting. Instead, Gradient Boosting Machines (GBM) are the bread-and-butter for (probabilistic) forecasting for large-scale problems in these companies. Thus, we seek to investigate how we can improve forecasting efficiency of these methods.

Appendices

2.A Supplemental Materials

Table 2.A.1 contains detailed descriptions of the datasets used in the chapter.

Table 2.A.1: Dataset descriptions

		Electricity	Traffic	Favorita	WebTraffic
time series	#	370	963	170k	10k
time series description		customers	traffic lanes	item-store combinations	wikipedia pages
target		$\mathbb{R}^+$	$[0, 1]$	$\mathbb{R}^+$	$\mathbb{R}^+$
train samples	#	500k	500k	500k	500k
validation samples	#	7k	7k	10k	7k
test samples	#	7k	7k	10k	7k
time step	t	hour	hour	day	day
input sequence length	t_0	168	168	90	90
output sequence length	$T - t_0$	24	24	30	30
covariate sequence length	T_c	500	500	150	150
categorical covariates	#	1	1	2	1
embedding dimension	d_{emb}	20	20	[8, 3]	20
numerical covariates	d_{cov}	7	5	7	8
lagged inputs	d_{lag}	1	1	1	1
categorical covariate description		customer_id	lane_id	item_id	page_id
				store_id	
numerical covariates description		Month_sin	DayOfWeek_sin	Holiday	DayOfWeek_sin
		Month_cos	DayOfWeek_cos	On_promotion	DayOfWeek_cos
		DayOfWeek_sin	HourOfDay_sin	On_sale	DayOfMonth_sin
		DayOfWeek_cos	HourOfDay_cos	DayOfWeek_sin	DayOfMonth_cos
		HourOfDay_sin	Available	DayOfWeek_cos	Month_sin
		HourOfDay_cos		Month_sin	Month_cos
		Online		Month_cos	WeekOfYear_sin
					WeekOfYear_cos
lagged input description		target_lagged	target_lagged	target_lagged	target_lagged

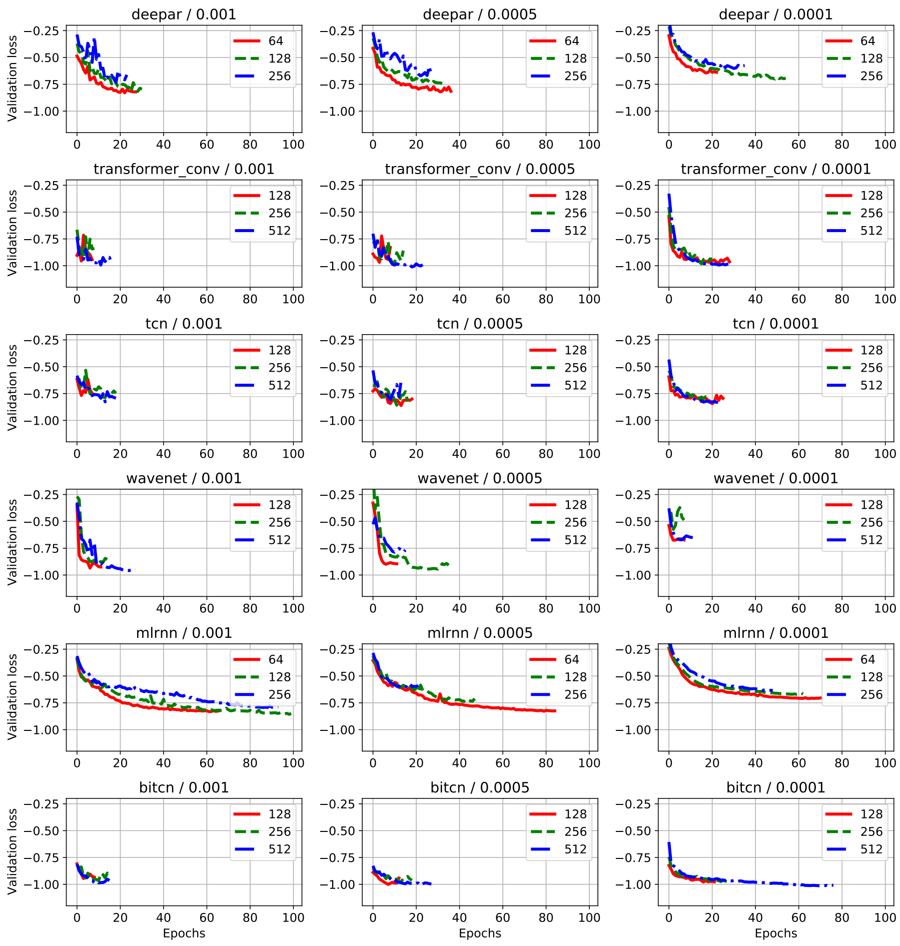


Figure 2.A.1: Training epochs versus validation loss for each method and learning rate for the `Electricity` dataset. The legends in the graphs denote the tested batch sizes.

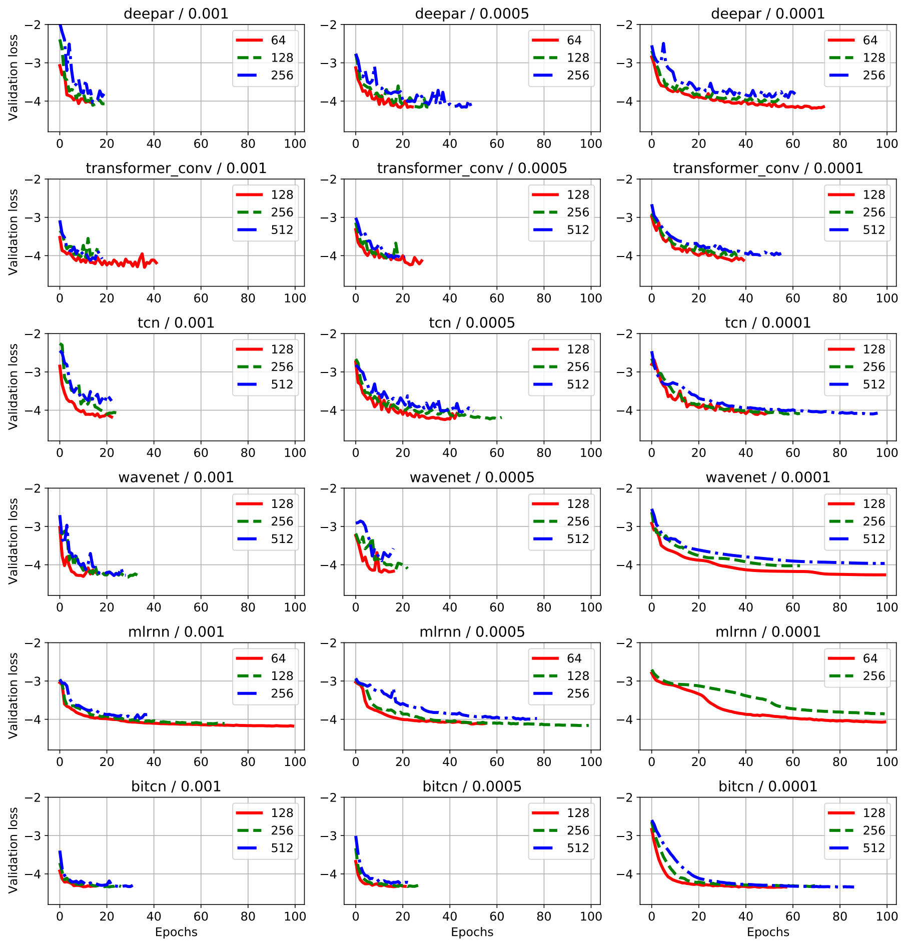


Figure 2.A.2: Training epochs versus validation loss for each method and learning rate for the `Traffic` dataset. The legends in the graphs denote the tested batch sizes. For ML-RNN, batch size 128 with learning rate 0.0001 resulted in repeated errors, therefore it is omitted in this graph.

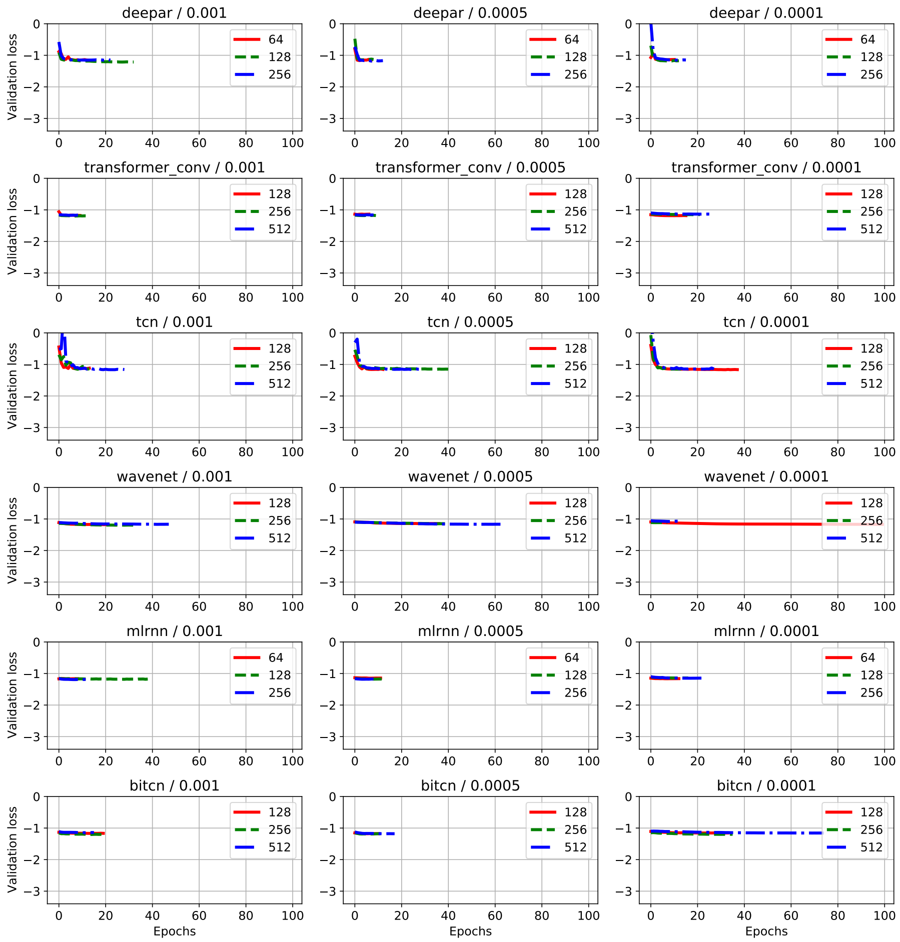


Figure 2.A.3: Training epochs versus validation loss for each method and learning rate for the `Favorita` dataset. The legends in the graphs denote the tested batch sizes.

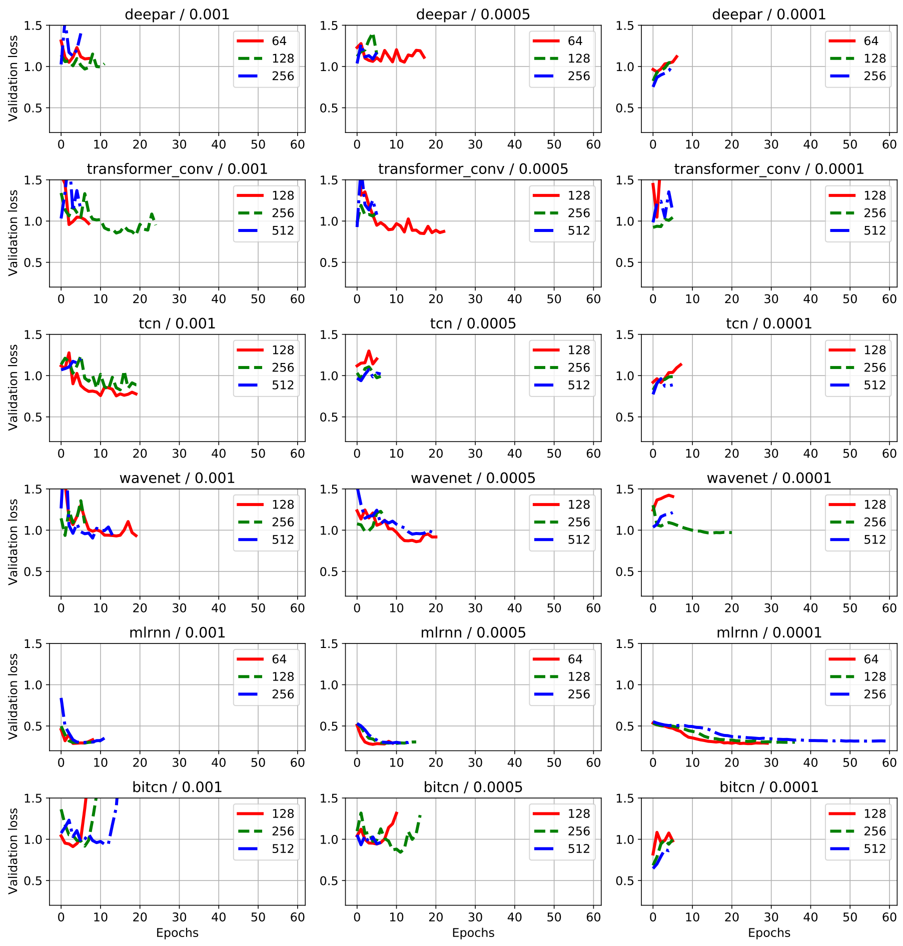


Figure 2.A.4: Training epochs versus validation loss for each method and learning rate for the `WebTraffic` dataset. The legends in the graphs denote the tested batch sizes.

3 Probabilistic Gradient Boosting Machines

Motivated by working with our industry partners Albert Heijn and bol, a grocery chain and e-commerce platform, respectively, we now turn to a different class of models: Gradient Boosting Machines (GBM). Within the forecasting teams of these companies, GBM are the most used model class for forecasting for large-scale time series problems. However, creating probabilistic predictions is difficult with existing GBM-based solutions: they either require training multiple models or they become too computationally expensive to be useful for large-scale settings. This motivates our second research question:

> **Research Question 2:** *How can we efficiently generate probabilistic forecasts with Gradient Boosting Machines (GBM) for large-scale settings?*

We propose *Probabilistic Gradient Boosting Machines* (PGBM), a method to create probabilistic predictions with a single ensemble of decision trees in a computationally efficient manner. PGBM approximates the leaf weights in a decision tree as a random variable, and approximates the mean and variance of each sample in a dataset via stochastic tree ensemble update equations. These learned moments allow us to subsequently sample from a specified distribution after training.

This chapter was published as: O. Sprangers, S. Schelter, and M. de Rijke. Probabilistic Gradient Boosting Machines for Large-Scale Probabilistic Regression. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pages 1510–1520, New York, NY, USA, Aug. 2021. Association for Computing Machinery. ISBN 978-1-4503-8332-5. doi: 10.1145/3447548.3467278.

We empirically demonstrate the advantages of PGBM compared to existing state-of-the-art methods: (i) PGBM enables probabilistic estimates without compromising on point performance in a single model, (ii) PGBM learns probabilistic estimates via a single model only (and without requiring multi-parameter boosting), and thereby offers a speedup of up to several orders of magnitude over existing state-of-the-art methods on large datasets, and (iii) PGBM achieves accurate probabilistic estimates in tasks with complex differentiable loss functions, such as hierarchical time series problems, where we observed up to 10% improvement in point forecasting performance and up to 300% improvement in probabilistic forecasting performance.

3.1 Introduction

Forecasting practioners are increasingly interested in probabilistic forecasts instead of point forecasts, in order to obtain a notion of uncertainty of the forecast [21]. Even though probabilistic forecasting techniques have been around for quite some time, applying these techniques in large-scale industrial settings often remains challenging. For example, retailers may require thousands of new forecasts for each product for each store. In this setting, traditional confidence interval techniques based on single model estimates are often computationally too expensive to execute on a daily basis [88]. Existing GBM-based methods for probabilistic forecasting often require training multiple models (e.g., LightGBM [68] or xgboost [24] require a separate model for each quantile of the forecast), or require computing expensive second-order derivative statistics [NG-Boost, 34]. Therefore, we aim to find a method that efficiently generates high-quality probabilistic forecasts with a single model using GBMs.

We address the challenge of large-scale probabilistic forecasting by proposing a novel, simple probabilistic forecasting method that leverages the popular GBM paradigm to provide accurate probabilistic forecasts in large-scale data settings (Section 3.4). We demonstrate that our approach achieves state-of-the-art point performance as well as probabilistic performance in forecasting tasks while only training a single ensemble of Gradient Boosted Decision Trees (GBDT). Our proposed method, *Probabilistic Gradient Boosting Machines* (PGBM), consists of two steps: 1. We treat the leaf weights in each tree as random variables that we approximate during training via the sample mean and sample variance of the samples in each leaf, and 2. We obtain an accurate estimate of the conditional mean and variance of our target for each sample by sequentially adding these random variables for each new tree. After training, we obtain a learned mean and variance for each sample, which can be used during prediction. Based on the learned mean and variance, we can specify a distribution from which to obtain our probabilistic forecast after training. Our PGBM is simple as its learning procedure is comparable to standard gradient boosting such as LightGBM [68] or xgboost [24] while it requires only few additional computation steps during training and prediction. However, contrary to these existing methods, our method only requires training a single ensemble of decision trees to obtain a model capable of providing probabilistic predictions.

We empirically demonstrate that PGBM offers state-of-the-art point and probabilistic regression performance on 11 datasets of various sizes (Section 3.5.1). Therefore, PGBM provides the advantages of existing state-of-the-art point prediction gradient boosting packages such as LightGBM or xgboost, as well as the advantage of a state-of-the-art probabilistic prediction package such as NGBoost. In addition, we show how to optimise PGBM's probabilistic estimates *after* training by varying a single hyperparameter and choosing different sets of posterior distributions. This offers the benefit of training a model only once, and optimizing for probabilistic performance thereafter. Neither existing standard gradient boosting packages or probabilistic packages such as NGBoost offer this. Furthermore, we demonstrate that our GPU-based implementation of PGBM trains an order of magnitude faster than existing state-of-the-art probabilistic gradient boosting methods on large datasets. Finally, we showcase the use of PGBM on the problem of probabilistic hierarchical time series forecasting, demonstrating that our implementation enables the optimization of complex differentiable loss functions without manually specifying an analytical gradient and hessian (in contrast to existing gradient boosting packages that rely on a priori specification of an analytical gradient and hessian).

In summary, we contribute:

- We introduce PGBM, a gradient boosting framework for probabilistic regression problems (Section 3.4);
- We demonstrate state-of-the-art point performance and probabilis-

tic performance of PGBM on a set of regression benchmarks (Section 3.5.1);

- We show that PGBM's probabilistic performance can be optimized *after* training the model, which allows practitioners to choose different posterior distributions without needing to retrain the model (Section 3.5.1);
- Our implementation of PGBM trains up to several orders of magnitude faster on larger datasets than competing methods (Section 3.5.1), and our implementation allows for the use of complex differentiable loss functions, where we observed up to 10% improvement in point forecasting performance and up to 300% improvement in probabilistic forecasting performance (Section 3.5.2).

3.2 Related Work

Traditional forecasting methods such as ARIMA [22] allow for probabilistic forecasts through specification of confidence intervals [58]. However, creating a confidence interval in these methods often requires assuming normality of the distribution of the target variable or its residuals. Generalized Additive Models for Shape, Scale and Location (GAMLSS) [104] is a framework that allows for a more flexible choice of distribution of the target variable in probabilistic regression problems. A disadvantage is that the model needs to be pre-specified, limiting flexibility of this method. Prophet [124] is a more recent example of generalized additive models applied to the probabilistic forecasting setting. However, Prophet has been shown to underperform other contemporary probabilistic forecasting methods [8, 113] and to have difficulties scaling to very large datasets [113]. Other Bayesian methods exhibiting similar issues include Bayesian Additive Regression Trees (BART) [28], which requires computationally expensive sampling techniques to obtain probabilistic predictions.

GBMs [38] are widely used for regression problems such as forecasting [24]. Popular GBM implementations such as LightGBM [68] or xgboost [24] have won various machine learning competitions [24]. The winning solution of the accuracy track of the recent M5 forecasting competition was based on a set of LightGBM models, and 4 out of the top-5 solutions used LightGBM models in their solutions [88]. However, GBMs are not naturally equipped to provide probabilistic forecasts as these models return point predictions, requiring multiple models when a practitioner desires a range of predictions. For example, the uncertainty track of the M5 forecasting competition required contestants to provide a set of quantiles for a hierarchical time series problem. The winning solution was based on 126 (!) separate LightGBM models, one for each requested quantile and time series aggregation level [88]. To address these limitations, NGBoost [34] allows for probabilistic regression with a single GBM by using the natural gradient to boost multiple parameters of a pre-specified distribution. Compared to NGBoost, our method PGBM does not require a natural gradient; it can achieve better or on-par predictive uncertainty estimates, without sacrificing performance on the point forecast of the same model as does NGBoost. Ben Taieb et al. [18] also propose boosted additive models for probabilistic forecasting. Our work is different in that we use GBMs with stochastic leaf weights to estimate the conditional mean and variance simultaneously for each estimator. Gouk et al. [44] present a method for incrementally constructing decision trees using stochastic gradient information. Our method is different in that (i) we focus on the general case of probabilistic regression instead of incremental online learning of trees and (ii) we obtain our stochastic estimates by approximating the ratio of the gradient and hessian.

Outside of the GBM context, decision trees have also been used for probabilistic regression problems in Quantile Regression Forests (QRF) [91]. However, this method requires storing all observations when computing leaf weights of a decision tree, which makes this method less suitable for large datasets. In addition, GBMs commonly outperform random forests on regression tasks, making the former a better choice when performance is a key consideration.

Contemporary large-scale probabilistic forecasting techniques often leverage the power of neural networks, such as DeepAR [107] or transformer-based models [80, 82]. However, in practice GBMs still seem the technique of choice — only one out of the top-5 solutions in the M5 uncertainty forecasting competition used a neural network method as its primary probabilistic forecasting tool.

In summary, we contribute the following on top of the related work discussed above: (i) PGBM is a single-parameter boosting method that achieves state-of-the-art point and probabilistic estimates using a single model, (ii) PGBM allows for choosing an output distribution after training, which means the probabilistic forecast can be optimized after training, (iii) our implementation allows training of larger datasets up to several orders of magnitude faster than the existing state-of-the-art, and (iv) our implementation allows for using complex differentiable loss functions, which removes the need to calculate an analytical gradient, thereby opening up a wider set of problems that can be effectively addressed.

3.3 Background

Gradient boosting optimizes a loss function by iteratively adding a set of weak learners into an ensemble [38]. Each new weak learner is added sequentially, such that this new learner reduces the aggregate error from the existing ensemble of weak learners. Typically, the weak learners are decision trees due to their strong empirical performance; and we also choose them as base learners in this chapter following the formalization of gradient boosting with decision trees due to Chen and Guestrin [24]. In gradient boosting, at each iteration k, we seek to construct a decision tree $f^{(k)}(\mathbf{x}_i)$ such that the update equation for our estimate for sample i reads:

$$\hat{y}_i^{(k)} = \hat{y}_i^{(k-1)} - \alpha f^{(k)}(\mathbf{x}_i), \tag{3.1}$$

in which α denotes the learning rate, typically chosen to be less than 1, such that only a tiny portion of each new base learner is added to the overall estimate at each iteration. We will derive a different set of update equations for PGBM in Section 3.4.3. To construct the decision tree $f^{(k)}$, we greedily split our training data based on its input features $\mathbf{x}$ into left (I_L) and right (I_R) nodes by maximizing the following *gain*:

$$G = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right], \tag{3.2}$$

where λ is a regularization parameter, $I = I_L \cup I_R$, and g_i, h_i are the gradient and hessian, respectively, with respect to $\hat{y}_i^{(k-1)}$ of some differentiable loss function that we aim to minimize, for example the mean-squared error loss in case of a regression problem.[1,2] When constructing the decision tree, Eq. (3.2) is evaluated at each node to find the best possible split gain G^* among all features in the input $\mathbf{x}$, and typically a split is made if the gain exceeds a certain threshold. If no split is made, the node becomes a *leaf* and the corresponding leaf weight w_j follows from:

$$w_j = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \tag{3.3}$$

in which $j \in \{0, 1, \dots, T\}$, with T the total number of leaves in our tree. We typically stop learning the tree if some pre-defined criterion is met, for example if no more splits with a positive split gain according to Eq. (3.2) can be made or the tree reaches a pre-defined fixed number of leaves. After training the tree, the output $f^{(k)}(\mathbf{x}_i)$ for a particular sample is then simply the leaf weight w_j, or:

$$\hat{y}_i^{(k)} = \hat{y}_i^{(k-1)} - \alpha w_j. \tag{3.4}$$

3.4 Probabilistic Gradient Boosting Machines (PGBM)

We introduce Probabilistic Gradient Boosting Machines (PGBM). We introduce our problem setting, the core components of PGBM, and end with an analysis and discussion of PGBM.

3.4.1 Probabilistic Forecasting

We are interested in the problem of probabilistic forecasting. More generally, we are interested in the problem of probabilistic regression, in which one aims to estimate a conditional probability distribution $P(y|\mathbf{x})$ of some target scalar variable y based on a set of inputs $\mathbf{x}$. In the case of probabilistic forecasting, $\mathbf{x}$ commonly includes lagged target variables

[1]Some researchers refer to this method as *Newton boosting* rather than *gradient boosting* [115], as it employs a second-order derivative.

[2]Unlike [24], we drop the regularization term γT. We have no need for this regularization parameter, as the number of leaves T is a hyperparameter that needs to be specified in our method. Note also that the parameter γ is in fact by default set to zero in the xgboost implementation of [24].

as well as additional covariates. We are interested in finding a model $f(\mathbf{x})$ that provides us with an estimate of the mean μ and variance σ^2 of a target distribution such that we can obtain a sample of an estimate $\hat{y}$ by sampling from a specified distribution D after training:

$$(\mu_{\hat{y}}, \sigma^2_{\hat{y}}) = f(\mathbf{x}) \tag{3.5}$$

$$\hat{y} \sim D(\mu_{\hat{y}}, \sigma^2_{\hat{y}}). \tag{3.6}$$

We construct our model $f(\mathbf{x})$ using gradient boosting. In order to find the estimate for the mean and variance of our target distribution, we next derive formulas for stochastic leaf weights (Eq. (3.3)) and new update equations (Eq. (3.4)).

3.4.2 Stochastic Leaf Weights

By creating stochastic leaf weights, we are able to learn a mean and variance of each leaf weight in each tree, thus enabling us to learn a mean and variance for each sample in our dataset. We assume that the gradient and hessian of our loss function are random variables with a mean (μ_g, μ_h) and finite variance (σ^2_g, σ^2_h) and covariance σ^2_{gh}, which we approximate separately in each tree for each instance set I_j using the sample mean, sample variance and sample covariance for the n_j samples in an instance set I_j:

$$\mu_g \approx \overline{g}_j = \frac{1}{n_j} \sum_{i \in I_j} g_i \tag{3.7}$$

$$\mu_h \approx \overline{h}_j = \frac{1}{n_j} \sum_{i \in I_j} h_i \tag{3.8}$$

$$\sigma^2_g \approx \overline{\sigma}^2_{g_j} = \frac{1}{n_j - 1} \sum_{i \in I_j} (g_i - \overline{g})^2 \tag{3.9}$$

$$\sigma^2_h \approx \overline{\sigma}^2_{h_j} = \frac{1}{n_j - 1} \sum_{i \in I_j} (h_i - \overline{h})^2 \tag{3.10}$$

$$\sigma^2_{gh} \approx \overline{\sigma}^2_{gh_j} = \frac{1}{n_j - 1} \sum_{i \in I_j} (g_i - \overline{g})(h_i - \overline{h}). \tag{3.11}$$

Note that the sample variance and covariance require the Bessel correction $n_j - 1$ in order to obtain an unbiased estimate of the true variance and covariance. Moreover, the Central Limit Theorem dictates that we obtain our true mean and variance if $n_j \to \infty$. However, for tiny datasets, n_j is typically a small number as each leaf may only contain a few samples and thereby using sample statistics might be inappropriate. In the Experiments section (Section 3.5), we will demonstrate that we are still able to provide accurate probabilistic estimates even in such cases. Next, we write Eq. (3.3) in terms of the sample mean of the gradient and hessian:

$$w_j = -\frac{\frac{1}{n_j}\sum_{i\in I_j} g_i}{\frac{1}{n_j}\sum_{i\in I_j} h_i + \frac{1}{n_j}\lambda} = -\frac{\overline{g}_j}{\overline{h}_j + \overline{\lambda}_j}. \tag{3.12}$$

Now, we can model the expectation and variance of the leaf weight w_j using the sample statistics as follows (dropping the subscript j on the right-hand side for readability):

$$\mu_j = E\left[\frac{\overline{g}}{(\overline{h}+\overline{\lambda})}\right] \approx \frac{\overline{g}}{(\overline{h}+\overline{\lambda})} - \frac{\overline{\sigma}^2_{gh}}{(\overline{h}+\overline{\lambda})^2} + \frac{\overline{g}\overline{\sigma}^2_{h}}{(\overline{h}+\overline{\lambda})^3} \tag{3.13}$$

$$\sigma_j^2 = V\left[\frac{\overline{g}}{(\overline{h}+\overline{\lambda})}\right] \approx \frac{\overline{\sigma}^2_{g}}{(\overline{h}+\overline{\lambda})^2} + \frac{\overline{g}^2\overline{\sigma}^2_{h}}{(\overline{h}+\overline{\lambda})^4} - 2\frac{\overline{g}\overline{\sigma}^2_{gh}}{(\overline{h}+\overline{\lambda})^3}. \tag{3.14}$$

We refer the reader to Appendix 3.A for the derivation of Eq. (3.13)–(3.14). Note that for most common loss functions, such as the mean-squared error, the two final terms of Eq. (3.13)–(3.14) are zero as the hessian h has no variation. When training a decision tree $f^{(k)}$, we store the obtained expectation and variance of each leaf of each tree and use these results to obtain our final estimate for the mean and variance using the update equations described in the next subsection.

3.4.3 Update Equations

Apart from the stochastic leaf weights, we require new update equations (Eq. (3.4)) in order to update the estimate for our mean and variance when adding a new tree at each iteration. These new update equations allow us to aggregate the stochastic weights over all the trees. For these equations, we make use of the following rules for the mean μ and variance σ^2 of some random variables (A, B) and a constant c:

$$\mu_{(A-cB)} = \mu_A - c \cdot \mu_B$$

$$\begin{aligned}\sigma^2_{cB} &= c^2\sigma^2_B \\ \sigma^2_{(A-cB)} &= \sigma^2_A + c^2\sigma^2_B - 2c \cdot \sigma^2_{(A,B)} \\ \sigma^2_{(A,B)} &= \rho_{(A,B)}\sigma_A\sigma_B,\end{aligned}$$

in which ρ denotes Pearson's correlation coefficient between the variables (A, B). Using these rules, we can modify Eq. (3.4) to arrive at the formulas for the expectation E and variance V of our estimate $\hat{y}_i^{(k)}$:

$$\mu_{\hat{y}_i^{(k)}} = E\left[\hat{y}_i^{(k)}\right] = \mu_{\hat{y}_i^{(k-1)}} - \alpha \cdot \mu_{j^{(k)}} \tag{3.15}$$

$$\sigma^2_{\hat{y}_i^{(k)}} = V\left[\hat{y}_i^{(k)}\right] = \sigma^2_{\hat{y}_i^{(k-1)}} + \alpha^2\sigma^2_{j^{(k)}} - 2\alpha\rho\sigma_{\hat{y}_i^{(k-1)}}\sigma_{j^{(k)}}, \tag{3.16}$$

where the hyperparameter ρ denotes the correlation coefficient between trees k and $k-1$. We provide further discussion around ρ in Section 3.4.5. Finally, the learned expectation and variance can be used for creating probabilistic predictions of new samples after training our model by sampling from a distribution parameterized by these learned quantities:

$$\hat{y}_i^{(k)} \sim D\left(\mu_{\hat{y}_i^{(k)}}, \sigma^2_{\hat{y}_i^{(k)}}\right). \tag{3.17}$$

We are now ready to fully present our method PGBM.

3.4.4 PGBM

Algorithm We provide a succint overview of the procedures for training and prediction with Probabilistic Gradient Boosting Machines (PGBM) in Algorithms 3.1 & 3.2.

***Training* (*Algorithm 3.1*)** In PGBM, gradient boosting is performed comparable to LightGBM [68] or xgboost [24], and PGBM employs global equal density histogram binning to bin continuous features into discrete bins in order to reduce the computational effort required to find the optimal split decision (Line 1). At the start of training, we initialize the estimate $\hat{\mathbf{y}}$, typically with the mean of the training set in a regression setting (Line 2). Then, gradient boosting is performed for a fixed number of iterations by first computing the gradient and hessian (Lines 4–5) of the training set and optionally choosing a subsample of the dataset (commonly referred to as bootstrapping, Line 6) on which to build the decision tree. The decision tree is then constructed up to a fixed number of leaves (Line 7) by first selecting the samples in the current node (Line 8), second by finding the best split for this node (Line 9), and third by splitting the current node or creating stochastic leaf weights if no split can be made (Line 10), for example, when the split does not result in a positive gain according to Eq. (3.2). After the tree construction has finished, predictions for the entire training set are generated (Line 11) and the overall estimate is updated (Line 12) and the process repeats for the next iteration.

Note that the learned variance is only used to create the probabilistic estimate in the prediction algorithm; it can also serve as a validation criterion during training (for example, by performing a prediction step on a validation set and deciding based on some probabilistic metric whether to continue training or not).

Prediction (Algorithm 3.2) During prediction, we initialize the estimate using the stored initial estimate of the training set (Line 1). Then, we make predictions on the dataset by iterating over all the trees (Line 2) using our new update equations (Line 3). Finally, we obtain our probabilistic estimate by sampling from a distribution parameterized by our learned mean and variance (Line 4).

Implementation We implement PGBM in PyTorch [96] and offer it as a Python package.[3] PyTorch offers (multi-)GPU acceleration by default, which allows us to scale PGBM to problems involving a large number of samples (we trained on datasets of over 10M samples) as we can distribute training across multiple GPUs. More importantly, our implementation allows for the use of the automated differentiation engine of PyTorch, such that we can employ arbitrary complex differentiable loss functions without requiring an analytical gradient and hessian. This is in stark contrast to existing popular packages such as LightGBM [68] or xgboost [24], where custom loss functions require the manual derivation of an analytical gradient and hessian. We provide an example of this benefit in Section 3.5.2. Note that PGBM can be made compatible with existing gradient boosting packages relatively easily too, as it only requires storing one additional sample statistic (the variance), changing the update equations according to Section 3.4.3 and choosing

[3] `https://github.com/elephaint/pgbm`

Algorithm 3.1 PGBM training algorithm

Input: Input dataset $\mathbf{X} \in \mathbb{R}^{n \times f}$ with n samples and f features, target output $\mathbf{y} \in \mathbb{R}^n$, differentiable loss function $l(\mathbf{y}, \hat{\mathbf{y}})$ and model `hyperparameters`.
Output:

```
 1: Bin features such that for each feature $|\mathbf{x}| \leq$ max_bins
 2: Set initial estimate $\hat{\mathbf{y}}$, e.g. to mean $\overline{\mathbf{y}}$ of target output
 3: for $k = 1$ to num_iterations do
 4:     Compute gradient $\mathbf{g}^{(k)} = \nabla_{\hat{\mathbf{y}}^{(k)}} l(\mathbf{y}, \hat{\mathbf{y}})$
 5:     Compute hessian $\mathbf{h}^{(k)} = \nabla^2_{\hat{\mathbf{y}}^{(k)}} l(\mathbf{y}, \hat{\mathbf{y}})$
 6:     Select subsample of input dataset as instance set $I_1$
 7:     for $j = 1$ to max_leaves do
 8:         Select instance set $I_j$ of $\mathbf{X}$, $\mathbf{g}^{(k)}$, $\mathbf{h}^{(k)}$
 9:         Find best split for all (features, bins) (Eq. (3.2))
10:         Create split if split criteria are met else create stochastic leaf
    weight (Eq. (3.7)–(3.14))
11:     Predict $\mathbf{X}$ to obtain $\mu_{\hat{\mathbf{y}}}^{(k)}$ (Eq. (3.15))
12:     Update estimate $\hat{\mathbf{y}} = \mu_{\hat{\mathbf{y}}}^{(k)}$
```

Algorithm 3.2 PGBM prediction algorithm

Input: Input dataset $\mathbf{X} \in \mathbb{R}^{n \times f}$ with n samples and f features, target distribution D and model `hyperparameters`.
Output:

```
1: Set initial estimate $\hat{\mathbf{y}}$ to mean $\overline{\mathbf{y}}$ of training dataset
2: for $k = 1$ to num_iterations do
3:     Predict $\mathbf{X}$ to obtain $(\mu_{\hat{\mathbf{y}}}^{(k)}, \sigma_{\hat{\mathbf{y}}}^{2(k)})$ (Eq. (3.15)–(3.16))
4: Draw n_samples $\hat{\mathbf{y}} \sim D\left(\mu_{\hat{\mathbf{y}}^{(k)}}, \sigma^2_{\hat{\mathbf{y}}^{(k)}}\right)$
```

a distribution D after training to sample from. We also offer PGBM using a fork of Scikit-learn's [97] *HistGradientBoostingRegressor*, such that it can be used as a drop-in replacement for pipelines employing *HistGradientBoostingRegressor*.

Furthermore, we implemented a custom CUDA kernel that integrates with PyTorch to calculate the optimal splitting decision (Eq. (3.2)), the most compute intensive part of PGBM. Our kernel leverages the parallel processing power of modern CUDA-capable GPUs, by parallelizing the split decision across the sample and feature dimension using parallel reductions. We demonstrate the effectiveness of our GPU training in Section 3.5.1.

3.4.5 Analysis & Discussion

Computational complexity Even though two parameters – a mean and variance – are learned in PGBM, the trees are constructed comparable to standard gradient boosting such as in [24, 68]. Therefore, the additional cost of our second parameter is negligible as only the sample statistics need to be calculated in the leaves. In contrast to NGBoost [34], PGBM also does not require calculation of a natural gradient, which involves the inversion of many small matrices. PGBM's runtime generally scales with the number of samples, the number of features and the number of bins used to bin the features, in accordance with existing GBM packages.

Higher-order moments and leaf sample quantiles We only consider the first two moments of a distribution (i.e., the mean and variance) to derive our stochastic leaf weights, which limits the output distribution D to distributions parameterized using location and scale parameters (i.e., our learned mean and variance). This is a limitation compared to, e.g., NGBoost [34]. We considered calculating higher order sample statistics such as the sample skewness (third moment) and sample kurtosis (fourth moment), however the disadvantage is that (i) there is no unbiased sample statistic for those measures, (ii) deriving approximations of the form of Eq. (3.13)–(3.14) becomes exceedingly complex, and (iii) higher-order sample statistics require more samples in order for the sample statistic to provide a reasonable estimate of the true statistic. Moreover, as we learn separate sample statistics for each leaf in each tree, we are still able to model complex distributions over the entire dataset using distributions parameterized only by location and scale parameters. Finally, one could also store the sample quantile information of each leaf and draw samples according to the stored quantile information. This would remove the need for specifying a particular distribution. While this seems an attractive option, calculating sample quantile information for each leaf is computationally difficult as it requires an expensive sorting operation, and storing a sufficient number of sample quantiles to reap the full benefits of this method requires storing at least 2–3x the number of leaf weights. In short, there is no real need to use higher order moments or leaf sample quantiles to provide accurate probabilistic estimates as we show in Section 3.5.1.

Output sampling PGBM allows one to sample from different output distributions *after* training, which allows practitioners to train their model by minimizing some point metric (e.g., RMSE) and after training try different distributions for optimizing the probabilistic forecast based on some validation metric. The key benefit is that this allows PGBM to achieve state-of-the-art point forecasting performance *as well as* accurate probabilistic forecasting performance *using the same model.* We will demonstrate this in Section 3.5.1. Note that practitioners can also optimize the probabilistic forecast by using a loss function that optimizes the probabilistic forecast.

Split decisions and tree dependence In PGBM, split decisions in the tree are not recomputed based on the stochasticity of the leaf weights, even though it could be argued that this would be appropriate when sampling from the trees. We intentionally avoid this as it is computationally expensive to recompute split decisions after training when sampling from the learned distribution. Secondly, by sequentially adding the mean and variance of each tree we omit the explicit covariance between tree k and trees $k-2, k-3, \ldots$, and only model the covariance between subsequent trees. We implicitly model both these effects using a single constant correlation hyperparameter ρ (Eq. (3.16)), which we further analyze in Section 3.5.1.

Hessian distribution The distribution of the hessian h should have a support of $[0, \infty)$ to avoid division by zero in Eq. (3.13)–(3.14), or equivalently, we require the sum of the hessians (plus regularization constant λ) of all samples in an instance set I_j of a leaf to be positive. For common convex loss functions such as the mean-squared error this is not an issue, however for non-convex loss functions this might pose a problem in rare cases where all hessians in an instance set add up to zero. In those cases, numerical issues can usually be avoided by requiring a decent (e.g., > 10) minimum number of samples in each leaf in a tree – this can be set as a hyperparameter in PGBM.

3.5 Experiments

First, we first demonstrate how PGBM can provide accurate point and probabilistic predictions on a set of common regression benchmark datasets from the UCI Machine Learning Repository (Section 3.5.1). We show how PGBM allows practitioners to optimize their probabilistic estimate after training, thereby removing the need to retrain a model under different posterior distribution assumptions. Next, we demonstrate the efficiency of our implementation of PGBM compared to existing gradient boosting methods. Finally, we demonstrate PGBM on the problem of forecasting for hierarchical time series, which requires optimizing a complex loss function for which deriving an analytical gradient is too complex (Section 3.5.2). Our experiments are run on open data, and our experimentation code is available online.[4]

3.5.1 UCI Regression Benchmarks

Task We perform probabilistic regression on a set of regression datasets. Our goal is to obtain the lowest probabilistic prediction score as well as the lowest point performance score.

Evaluation We evaluate the probabilistic performance of each method using the Continuously Ranked Probability Score (CRPS), which is a measure of discrepancy between the empirical cumulative distribution function of an observation y and the cumulative distribution F of a forecast $\hat{y}$ [139]:

$$C = \int [F(\hat{y}) - \mathbb{1}(\hat{y} \geq y)]^2 d\hat{y}, \tag{3.18}$$

[4]Repository at `https://github.com/elephaint/pgbm`

in which $\mathbb{1}$ denotes the indicator function. We compute the empirical CRPS based on 1,000 sample predictions generated by the trained models on the test set. We evaluate point performance using Root Mean Squared Error (RMSE): $\sqrt{\frac{1}{n}\sum_i^n(y_i - \hat{y}_i)^2}$. We present these metrics relative to the median of PGBM over all the folds tested for a dataset, and we refer the reader to Table 3.B.1 of Appendix 3.B for further details.

Protocol We follow the same protocol as Duan et al. [34], and create 20 random folds for each dataset except for `msd`, for which we only create one. For each of these folds, we keep 10% of the samples as test set. The remaining 90% is first split into an 80/20 validation/training set to find the number of training iterations that results in the lowest validation score. After validation, the full 90% training set is trained using the number of iterations found in the validation step. As output distribution for the probabilistic prediction we use a normal distribution, similar to Duan et al. [34].

Baseline models For probabilistic performance, we compare against NGBoost [34], which has recently been shown to outperform other comparable methods on the current set of benchmarks. We use the same settings for NGBoost as in [34]. For point performance, we also compare to LightGBM [68], one of the most popular and best-performing gradient boosting packages available. We configure LightGBM to have the same settings as PGBM.

PGBM For all datasets, we use the same hyperparameters for PGBM, except that we use a bagging fraction of 0.1 for `msd` in correspondence with Duan et al. [34]. Our training objective in PGBM is to minimize the mean-squared error (MSE). We refer the reader to Table 3.B.2 of Appendix 3.B for an overview of key hyperparameters of each method.

Results We provide the results of our first experiment in Figures 3.1–3.2 and observe the following:

- On probabilistic performance, PGBM outperforms NGBoost on average by approximately 15% as demonstrated by the relatively lower CRPS across all but one dataset (`msd`, where the difference is tiny). This is remarkable, as the training objective in PGBM is to minimize the MSE, rather than optimize the probabilistic forecast as does NGBoost.

- PGBM outperforms NGBoost on all datasets on point performance, and on average by almost 20%, which is in line with expectation as we explicitly set out to optimize the MSE as training objective in PGBM in this experiment. However, as becomes clear from this result, PGBM does not have to sacrifice point performance in order to provide state-of-the-art probabilistic estimates nonetheless. Compared to LightGBM, PGBM performs slightly worse on average (approx. 3%) on point performance. We suspect this is due to implementation specifics.

The main takeaway from this experiment is that even though we only optimized for a point metric (MSE) in PGBM, we were still able to achieve similar probabilistic performance compared to a method that explicitly optimizes for probabilistic performance.

Analysis: correlation hyperparameter We perform a brief analysis of the correlation hyperparameter ρ (Eq. (3.16)). This hyperparameter controls the dependence between variance estimates of subsequent trees in PGBM, and is critical for probabilistic performance. Figure 3.3a shows the CRPS evaluated on the validation set at different settings for ρ for each dataset for a single fold. We normalized the CRPS scores on the lowest CRPS for each dataset. Across all datasets, the CRPS seems to follow a parabolic shape and consequently there seems to be an optimal choice for ρ across different datasets: a value of 0.02–0.07 typically seems appropriate. Empirically, we found that an initial value of $\rho = \frac{\log_{10} n}{100}$, where n denotes the size of the training set generally works well and therefore we used that in our experiments as default value. Intuitively, the positiveness of the correlation between subsequent trees seems logical: if the leaf weight of a given tree shifts more positively (negatively) as a result of stochasticity, the residual on which the next tree will be constructed shifts in the same direction. Consequently, the leaf weights of this next tree will also shift in the same direction, thus exhibiting a positive correlation with the previous tree's leaf weights. Furthermore, larger datasets tend to cluster together in behavior, as can be seen from the curves for the `protein`, `msd`, `kin8nm`, `power` and

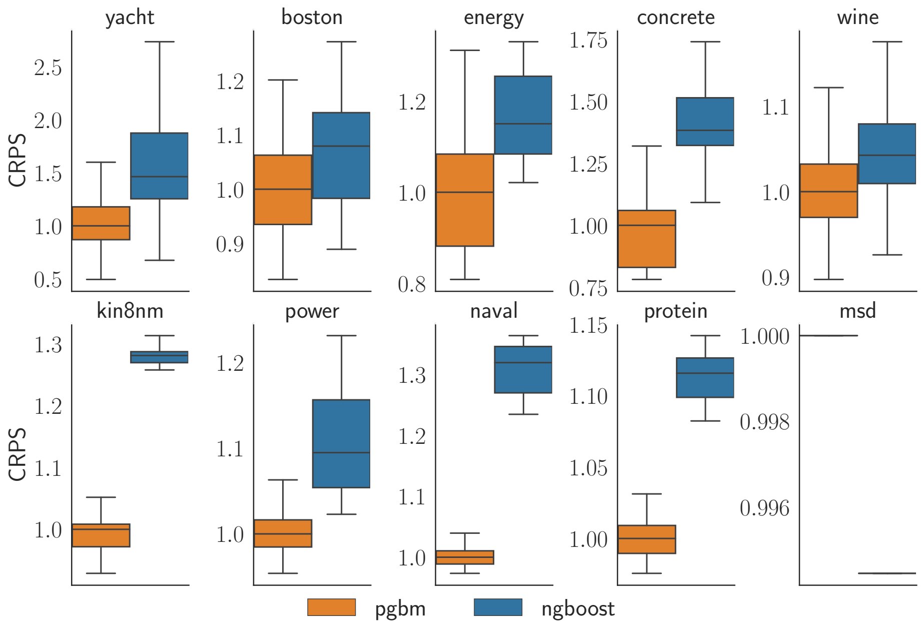


Figure 3.1: Results for probabilistic performance (CRPS) for each dataset for each method, with the smallest dataset `yacht` in the top left corner and the largest dataset `msd` in the lower right corner. Lower is better, and results have been indexed against the median test score of PGBM. PGBM outperforms NGBoost in probabilistic performance.

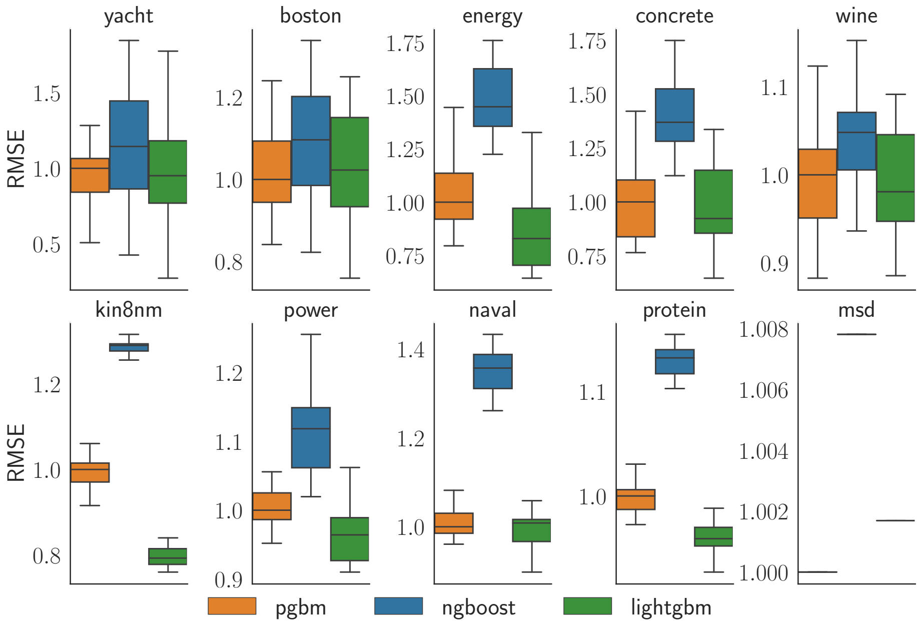
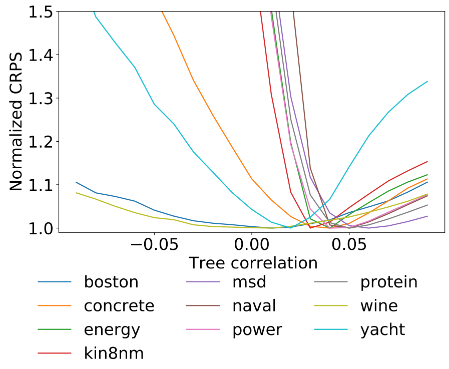
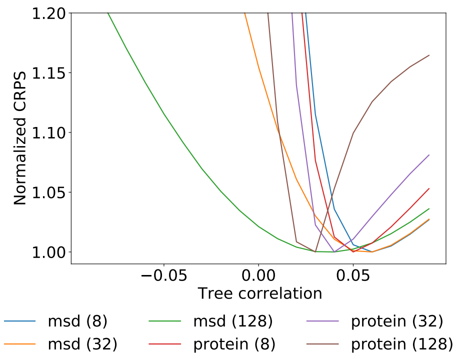


Figure 3.2: Results for point performance (RMSE) for each dataset for each method, with the smallest dataset `yacht` in the top left corner and the largest dataset `msd` in the lower right corner. Lower is better, and results have been indexed against the median test score of PGBM. PGBM outperforms NGBoost in point performance and performs on par with LightGBM.



(a)



(b)

Figure 3.3: Normalized CRPS on the validation set for different settings of tree correlation hyperparameter ρ, (a) for all datasets and (b) for `protein` and `msd` when trained using a maximum number of leaves per tree of $\{8, 32, 128\}$.

`naval` datasets. It seems that for larger datasets, typically larger settings for ρ are appropriate. Our hypothesis is that for trees containing many samples per leaf, the correlation between subsequent trees is higher, as more samples in the tree's leaves will generally imply that the model has not yet (over)fit to the training set and there is likely more information left in the residuals compared to the situation where there are few samples per leaf. This would explain the behavior observed in Figure 3.3a as we train each model with a maximum of 8 leaves per tree, resulting in more samples per leaf for larger datasets. We test this hypothesis by training the relatively larger `protein` and `msd` datasets using different settings for the maximum number of leaves, for which we show the results in Figure 3.3b. Confirming our hypothesis, we indeed observe the optimal correlation parameter decreasing when we train PGBM using a higher number of maximum leaves per tree (i.e., the minimum of the parabola shifts to the left).

Analysis: posterior distribution One of the key benefits of PGBM is that it allows us to optimize the probabilistic estimate *after* training, as the choice of distribution D in Eq. (3.6) is independent from the training process. This offers the benefit of training to optimize a certain point metric such as RMSE, and choosing the distribution that best fits the learned mean and variance after training by validating a set of distribution choices on a validation set. To demonstrate this benefit, we repeated the experiments from our first experiment for a single fold. For each dataset, we evaluated the learned model on CRPS on the validation set for a set of common distributions and a range of tree correlations $\rho = \{0.00, 0.01, \dots, 0.09\}$. The optimal choice of distribution and tree correlation on the validation set was subsequently used for calculating the CRPS on the test set. We report the results in Table 3.1, where 'Base case' refers to the base case scenario from our first experiment, where we chose a Normal distribution and a tree correlation hyperparameter of $\rho = \frac{\log_{10} n}{100}$ across all datasets, and 'Optimal' refers to the result on the test set when choosing the distribution and tree correlation according to the lowest CRPS on the validation set. We see that for most datasets, the minimum CRPS on the validation set is similar across choices of distribution, which implies that it is more beneficial to optimize the tree correlation rather than the choice of output distribution for these datasets. On the test set, we see improved scores compared to the base case on all but the smallest dataset, thereby showcasing the benefit of optimizing the probabilistic forecast after training. We would advice practitioners to start with a generic distribution such as the normal or Student's t(3), and optimize the probabilistic estimate after training by testing for different tree correlations and distribution choices.

Analysis: training time Our implementation in PyTorch allows us to use GPU training by default, which allows us to significantly speed up training for larger datasets. We demonstrate this benefit in Table 3.2 where we compare training times for datasets of different size against a baseline of PGBM (we refer to Table 3.B.3 in the Appendix for the absolute timings). For this experiment, we also included the `higgs` dataset, which is a 10M sample UCI dataset commonly used to benchmark gradient boosting packages. For PGBM, we show results for training on GPU-only and CPU-only. NGBoost does not offer GPU training and runs on top of the default scikit-learn decision tree regressor. We ran our experiments on a 6-core machine with a nVidia RTX 2080Ti GPU. As can be seen, PGBM is up to several orders of magnitude faster than NGBoost as the dataset size increases. This demonstrates that PGBM and our implementation allow practitioners to solve probabilistic regression problems for datasets much larger than NGBoost.

Table 3.1: Results for probabilistic (CRPS) performance for each dataset when varying the posterior distribution and tree correlations on the validation set (left) and subsequently using the optimal choice of distribution and tree correlation on the test set (right). Best results on validation and test set are in bold. We report the minimum CRPS on the validation set across all the tree correlations tested per distribution.

	Validation set									Test set	
Dataset	Normal	Student's t(3)	Logistic	Laplace	LogNormal	Gumbel	Weibull	Poisson	NegBinomial	Base	Optimal
`yacht`	**0.37**	0.37	0.37	0.37	0.44	0.38	0.37	0.75	9.7	**0.18**	0.19
`boston`	1.41	1.40	1.39	**1.39**	1.4	1.42	1.39	1.58	12.27	2.08	**2.06**
`energy`	0.62	0.61	0.62	**0.61**	0.62	0.62	0.63	1.25	16.15	0.64	**0.64**
`concrete`	2.21	2.21	2.21	**2.19**	2.23	2.26	2.22	2.38	3.59	2.69	**2.64**
`wine`	**0.32**	0.33	0.32	0.32	0.32	0.32	0.32	0.59	0.64	0.35	**0.34**
`kin8nm`	0.08	0.08	**0.08**	0.08	1.04	0.08	0.08	0.26	0.26	0.08	**0.08**
`power`	1.86	1.86	**1.86**	1.86	1.86	1.87	1.88	5.22	454.34	1.81	**1.80**
`naval`	0.00	0.00	0.00	**0.00**	0.02	0.00	0.00	0.22	0.22	0.00	**0.00**
`protein`	2.18	2.18	2.18	2.18	45.3	2.17	2.54	2.15	**2.15**	2.14	**2.12**
`msd`	4.74	4.73	4.74	4.73	4.74	4.88	**4.69**	11.16	1982.19	4.78	**4.74**

Table 3.2: Training time for 2,000 iterations for 5 datasets of different size as a fraction of PGBM-gpu training time. Bold indicates probabilistic forecasting method with the lowest training time.

	Probabilistic forecast			Point forecast
Dataset	*PGBM-gpu*	*PGBM-cpu*	*NGBoost*	*LightGBM*
`wine` (n=1,599)	1.00	0.41	**0.20**	0.01
`naval` (n=12k)	**1.00**	1.62	1.05	0.02
`protein` (n=46k)	**1.00**	3.09	3.39	0.02
`msd` (n=515k)	**1.00**	26.85	48.83	0.09
`higgs` (n=10,5M)	**1.00**	101.76	147.78	0.43

In general, for smaller datasets, training on cpu offers the best timings for the two methods. We include the relative timings to LightGBM for reference in Table 3.2, which shows that PGBM even becomes competitive to LightGBM for the largest dataset (`higgs`). However, the timings for LightGBM represent the timings to train a *single* LightGBM model. If one is interested in obtaining a probabilistic forecast, the timings would be multiplied by the number of quantiles required. Hence, for a fine-grained probability distribution, the timings would be 5–10x higher for LightGBM, again demonstrating the effectiveness of our implementation for probabilistic forecasting.

3.5.2 Hierarchical Time Series

Task So far, our experimental results were obtained using the mean-squared error as objective function for which an analytical gradient and hessian can be easily derived. In this experiment, we apply PGBM to the problem of hierarchical time series forecasting, which is a problem where our loss function is rather complex, so that it becomes very tedious to manually calculate an analytical gradient and hessian for it:

$$L = \sum_{j}^{N} w_j (y_j - \hat{y}_j)^2, \tag{3.19}$$

where w_j is the weight of the j-th time series, and N is the number of time series. In hierarchical time series, we aggregate time series across multiple levels. For example, in the case of two time series and two levels, $N = 3$ and our loss for each series reads $L_1 = w_1(y_1 - \hat{y}_1)^2$, $L_2 = w_2(y_2 - \hat{y_2})^2$, $L_3 = w_3((y_1 + y_2) - (\hat{y}_1 + \hat{y}_2))^2$ with w_1, w_2, w_3 weights of each series, for example $0.25, 0.25, 0.5$. Hence, the gradient and hessian of L with respect to the first estimate $\hat{y}_1$ becomes $\frac{\partial L}{\partial \hat{y}_1} = -w_1(2y_1 - 2\hat{y}_1) - w_3(2y_1 + 2y_2 - 2\hat{y}_1 + 2\hat{y}_2)$ and $\frac{\partial^2 L}{\partial^2 \hat{y}_1} = 2w_1 + 2w_3$. It is clear that deriving this result analytically becomes increasingly complex when the number of levels and the number of time series increases, which necessitates the use of autodifferentiation packages such as PyTorch if we are interested in optimizing this objective.

Dataset We use a subset of the dataset from the M5 forecasting competition [88], in which participants were asked to provide hierarchical forecasts for Walmart retail store products. We use a single store and create forecasts for a single day. For each store, we are interested not only in accurately forecasting individual item sales, but also in optimizing the aggregate sales per day, aggregate sales per day per category and aggregate sales per day per department. Hence, we obtain four levels for our weighted loss function: (i) individual items, (ii) category aggregates per day, (iii) department aggregates per day, and (iv) total daily aggregates. For more details on the data and preprocessing we refer to Appendix 3.B.

Protocol We compare against a baseline of LightGBM, NGBoost and PGBM trained with the regular mean-squared error objective. All models are trained using the same hyperparameters. We validate on the last 28 days of the training dataset and pick the number of iterations resulting in the lowest item RMSE on this validation set. After validating, we retrain on the entire dataset for the number of optimal iterations and test on a held out test set of 28 days (with the first day starting the day after the last day in the validation set). We use the Normal distribution with a tree correlation of $\rho = \frac{\log_{10} n}{100}$ to generate our probabilistic forecasts for PGBM.

Results We evaluate our model on RMSE and CRPS for each aggregation and the results are displayed in Table 3.3. We observe that using the weighted MSE that incorporates our four levels of aggregation results in a similar point forecast score for individual items, but in a much better forecast for the aggregations – differences up to 10% compared to the

Table 3.3: Point (RMSE) and probabilistic (CRPS) forecasting performance for the M5 dataset across aggregations when using MSE or weighted MSE (only for PGBM) as training objective. Lower is better, best results are indicated in bold.

	RMSE				CRPS		
	PGBM		NGBoost	LightGBM	PGBM		NGBoost
objective	*MSE*	*wMSE*	*MSE*	*MSE*	*MSE*	*wMSE*	*MSE*
Individual[1]	**2.00**	**2.00**	2.01	**2.00**	**0.77**	0.93	0.78
Category[2]	67.9	**67.6**	77.4	70.0	72.6	**40.4**	82.0
Department[3]	108	**101**	129	111	160	**59**	184
Total[4]	213	**190**	276	225	472	**136**	560

1. $n = 85,372$. 2. $n = 196$. 3. $n = 84$. 4. $n = 28$.

second-best point performance of PGBM are observed. Secondly, we see that the gain using the weighted MSE increases at hierarchically higher aggregation levels such as 'total by date'. This is important, as this implies that we are able to generate item-level forecasts that are more consistent with higher-level aggregates. Finally, we observe that item-level CRPS is worse in the weighted MSE setting compared to the regular MSE setting, whereas our probabilistic estimate for higher aggregations improves up to 300% when using the weighted MSE. This can be expected: in the MSE setting, each individual item forecast 'does not consider' aggregates in the category or department, whereas in the weighted MSE setting, item forecasts are optimized to also consider the impact on the overall aggregations. All in all, this experiment demonstrates the benefit of our implementation: we can optimize over more complex loss functions, thereby enabling probabilistic forecasts of more complex problems such as hierarchical time series problems.

3.6 Conclusion

In this chapter we introduced PGBM, a method for probabilistic regression using gradient boosting machines. PGBM creates probabilistic estimates by using stochastic tree leaf weights based on sample statistics. By combining the learned weights for each subsequent tree, PGBM learns a mean and variance for samples in a dataset which can be used to sample from an arbitrary distribution of choice. We demonstrated that PGBM provides state-of-the-art probabilistic regression results across a range of small to large datasets. Benefits of PGBM compared to existing work are that (i) PGBM is a single-parameter boosting method that optimizes a point regression but achieves state-of-the-art probabilistic estimates using the same model, (ii) PGBM enables the choice of an output distribution after training, which means practitioners can optimize the choice of distribution without requiring retraining of the model, (iii) our implementation allows training of larger datasets up to several orders of magnitude faster than the existing state-of-the-art, and (iv) our implementation in PyTorch allows using complex differentiable loss functions which removes the need to calculate an analytical gradient as is common in existing gradient boosting packages. We demonstrated the latter benefit for the problem of hierarchical time series forecasting, where we observed up to 10% improvement in point performance and up to 300% improvement in probabilistic forecasting performance.

Limitations of PGBM are that it only learns the mean and variance in a tree, which limits the choice of output distribution. However, we observed no negative performance effects in our experiments thereof.

In the future, we intend to further work on the theoretical error bounds of PGBM. Under mild assumptions, sample statistics in each leaf of each tree appropriately represent the true statistics of the samples in each leaf provided a sufficient number of samples. However, we have yet to determine appropriate theoretical error bounds on the final estimated statistics when considering the simplifications we make during decision tree learning, such as the greedy approximate split finding, using a limited number of tree leaves, our approximation to the stochastic leaf weights, keeping decision points constant and treating the correlation between subsequent trees as a constant across samples and trees. Regarding the latter, we also expect that the probabilistic estimate can be further improved by using a better approximation to the tree correlations instead of our choice of keeping it fixed across trees and samples.

In the next chapter, we further dive into the problem of hierarchical forecasting, which we briefly touched upon in this chapter. We find that existing hierarchical forecasting techniques scale relatively poorly to large-scale problem settings, and investigate methods that overcome these limitations.

Appendices

3.A Derivation of Stochastic Leaf Weights

3.A.1 Expectation

We approximate the mean in each leaf by using a second-order Taylor approximation of the expectation of a function of the two random variables (g, h) around the point $\boldsymbol{a} = (\overline{g}, \overline{h})$:

$$\begin{aligned} E[f(g,h)] = E[f(\boldsymbol{a}) &+ f'_g(\boldsymbol{a})(g-\overline{g}) + f'_h(\boldsymbol{a})(h-\overline{h}) \\ &+ \frac{1}{2}f''_{gg}(\boldsymbol{a})(g-\overline{g})^2 + f''_{gh}(\boldsymbol{a})(g-\overline{g})(h-\overline{h}) \\ &+ \frac{1}{2}f''_{hh}(\boldsymbol{a})(h-\overline{h})^2 + H], \end{aligned} \tag{3.20}$$

with H denoting the higher-order terms that we drop for our estimate. Using the laws of expecations we then obtain:

$$\begin{aligned} E[f(g,h)] \approx\ & E[f(\boldsymbol{a})] + E[f'_g(\boldsymbol{a})(g-\overline{g})] + E[f'_h(\boldsymbol{a})(h-\overline{h})] \\ &+ E[\frac{1}{2}f''_{gg}(\boldsymbol{a})(g-\overline{g})^2] + E[f''_{gh}(\boldsymbol{a})(g-\overline{g})(h-\overline{h})] \\ &+ E[\frac{1}{2}f''_{hh}(\boldsymbol{a})(h-\overline{h})^2] \end{aligned} \tag{3.21}$$

$$\begin{aligned} =\ & E[f(\boldsymbol{a})] + f'_g(\boldsymbol{a})\underbrace{E[(g-\overline{g})]}_{0} + f'_h(\boldsymbol{a})\underbrace{E[(h-\overline{h})]}_{0} \\ &+ \frac{1}{2}f''_{gg}(\boldsymbol{a})\underbrace{E[(g-\overline{g})^2]}_{\sigma_g^2} + f''_{gh}(\boldsymbol{a})\underbrace{E[(g-\overline{g})(h-\overline{h})]}_{\sigma_{gh}^2} \\ &+ \frac{1}{2}f''_{hh}(\boldsymbol{a})\underbrace{E[(h-\overline{h})^2]}_{\sigma_g^2} \end{aligned} \tag{3.22}$$

$$\begin{aligned} =\ & E[f(\boldsymbol{a})] + \frac{1}{2}f''_{gg}(\boldsymbol{a})\sigma_g^2 + f''_{gh}(\boldsymbol{a})\sigma_{gh}^2 \\ &+ \frac{1}{2}f''_{hh}(\boldsymbol{a})\sigma_h^2, \end{aligned} \tag{3.23}$$

with σ^2_{gh} denoting the covariance of the gradient and the hessian. For a function $f(g,h) = \frac{g}{h}$, we have:

$$\begin{aligned} f'_g &= h^{-1} \\ f'_h &= -gh^{-2} \\ f''_{gg} &= 0 \\ f''_{gh} &= -h^{-2} \\ f''_{hh} &= 2gh^{-3}. \end{aligned}$$

Substituting and using $\boldsymbol{a} = (\overline{g}, \overline{h})$, $f(\boldsymbol{a}) = \frac{\overline{g}}{\overline{h}}$:

$$\begin{aligned} E[f(g,h)] &\approx E[f(\boldsymbol{a})] - h^{-2}(\boldsymbol{a})\sigma^2_{gh} + gh^{-3}(\boldsymbol{a})\sigma^2_h && (3.24) \\ &= \frac{\overline{g}}{\overline{h}} - \frac{\sigma^2_{gh}}{\overline{h}^2} + \frac{\overline{g}\sigma^2_h}{\overline{h}^3}. && (3.25) \end{aligned}$$

Finally, we can include the regularization constant $\overline{\lambda}$ to arrive at the final estimate of the expectation for the leaf weight w_j. This constant only affects the mean of the random variable h, therefore we can safely add it to the terms containing $\overline{h}$:

$$E\left[\frac{\overline{g}}{(\overline{h}+\overline{\lambda})}\right] \approx \frac{\overline{g}}{(\overline{h}+\overline{\lambda})} - \frac{\sigma^2_{gh}}{(\overline{h}+\overline{\lambda})^2} + \frac{\overline{g}\sigma^2_h}{(\overline{h}+\overline{\lambda})^3}. \tag{3.26}$$

Note that we can obtain the first-order Taylor approximation of the mean by dropping the last two terms of Eq. (3.26):

$$E\left[\frac{\overline{g}}{(\overline{h}+\overline{\lambda})}\right] \approx \frac{\overline{g}}{(\overline{h}+\overline{\lambda})}. \tag{3.27}$$

3.A.2 Variance

For the variance, we start with the definition of variance for a function $f(g,h)$:

$$V\left[f(g,h)\right] = E\left[(f(g,h) - E[f(g,h)])^2\right]. \tag{3.28}$$

We perform a first-order Taylor expansion of $f(g,h)$ around the point $\boldsymbol{a} = (\overline{g}, \overline{h})$ and we substitute the first-order approximation of the mean:

$$V\left[f(g,h)\right] \approx E\Big[\Big(f(\boldsymbol{a}) + f'_g(\boldsymbol{a})(g-\overline{g}) + f'_h(\boldsymbol{a})(h-\overline{h})$$

$$
\begin{aligned}
&- E[f(\boldsymbol{a})]\Big)^2\Big] && (3.29)\\
&= E[(f'_g(\boldsymbol{a})(g-\overline{g}) + f'_h(\boldsymbol{a})(h-\overline{h}))^2] && (3.30)\\
&= E[f'^2_g(\boldsymbol{a})(g-\overline{g})^2 + f'^2_h(\boldsymbol{a})(h-\overline{h})^2 \\
&\quad + 2f'_g(\boldsymbol{a})(g-\overline{g})f'_h(\boldsymbol{a})(h-\overline{h})] && (3.31)\\
&= f'^2_g(\boldsymbol{a})E[(g-\overline{g})^2] + f'^2_h(\boldsymbol{a})E[(h-\overline{h})^2] \\
&\quad + 2f'_g(\boldsymbol{a})f'_h(\boldsymbol{a})E[(g-\overline{g})(h-\overline{h})] && (3.32)\\
&= \overline{h}^{-2}\sigma_g^2 + \overline{g}^2\overline{h}^{-4}\sigma_h^2 - 2\overline{g}\overline{h}^{-3}\sigma_{gh}^2 && (3.33)\\
&= \frac{\sigma_g^2}{\overline{h}^2} + \frac{\overline{g}^2\sigma_h^2}{\overline{h}^4} - 2\frac{\overline{g}\sigma_{gh}^2}{\overline{h}^3}. && (3.34)
\end{aligned}
$$

Finally, including the regularization constant $\overline{\lambda}$ we obtain:

$$
V\left[\frac{\overline{g}}{(\overline{h}+\overline{\lambda})}\right] \approx \frac{\sigma_g^2}{(\overline{h}+\overline{\lambda})^2} + \frac{\overline{g}^2\sigma_h^2}{(\overline{h}+\overline{\lambda})^4} - 2\frac{\overline{g}\sigma_{gh}^2}{(\overline{h}+\overline{\lambda})^3}. \qquad (3.35)
$$

3.B Reproducibility

We report absolute scores and dataset statistics for the UCI benchmark in Table 3.B.1. An overview of the key hyperparameters for each method for both experiments is given in Table 3.B.2, and absolute timings for the timings of Table 3.2 in Table 3.B.3. An overview of the M5 dataset is given in Table 3.B.4. We refer to our code at `https://github.com/elephaint/pgbm` for further details, such as the features of the M5 dataset, which mainly comprise lagged target variables, time indicators (e.g., day-of-week), event indicators (e.g., holidays) and item indicators.

Table 3.B.1: Results for probabilistic (CRPS) and point (RMSE) performance for each dataset. We report mean metrics over all folds per method and indicate the standard deviation in brackets. Lower is better.

				CRPS		RMSE		
Dataset	folds	samples	features	PGBM	NGBoost	PGBM	NGBoost	LightGBM
`yacht`	20	308	6	0.22 (0.070)	0.32 (0.104)	0.63 (0.213)	0.75 (0.297)	0.64 (0.281)
`boston`	20	506	13	1.61 (0.201)	1.73 (0.236)	3.05 (0.507)	3.31 (0.661)	3.11 (0.675)
`energy`	20	768	8	0.21 (0.034)	0.25 (0.022)	0.35 (0.062)	0.49 (0.055)	0.29 (0.075)
`concrete`	20	1,030	8	2.06 (0.335)	2.95 (0.326)	3.97 (0.759)	5.50 (0.642)	3.80 (0.762)
`wine`	20	1,599	11	0.33 (0.034)	0.34 (0.024)	0.60 (0.054)	0.62 (0.043)	0.60 (0.050)
`kin8nm`	20	8,192	8	0.07 (0.002)	0.10 (0.002)	0.13 (0.005)	0.17 (0.003)	0.11 (0.003)
`power`	20	9,568	4	1.81 (0.053)	2.01 (0.120)	3.35 (0.153)	3.70 (0.222)	3.20 (0.140)
`naval`	20	11,934	14	0.00 (0.000)	0.00 (0.000)	0.00 (0.000)	0.00 (0.000)	0.00 (0.000)
`protein`	20	45,730	9	2.19 (0.030)	2.44 (0.038)	3.98 (0.056)	4.50 (0.059)	3.82 (0.058)
`msd`	1	515,345	90	4.78	4.75	9.09	9.16	9.11
`higgs`	1	10,500,000	28	0.253	0.238	0.418	0.419	0.414

Table 3.B.2: Key hyperparameters for the UCI benchmark and hierarchical time series experiment.

	UCI benchmark			Hierarchical time series		
	PGBM	NGBoost	LightGBM	PGBM	LightGBM	NGBoost
min_split_gain	0	0	0	0	0	0
min_data_in_leaf	1	1	1	1	1	1
max_bin	64	n.a.	64	1024	1024	n.a.
max_leaves	8	n.a.	8	64	64	64
max_depth	-1	3	-1	-1	-1	-1
learning_rate	0.1	0.01	0.1	0.1	0.1	0.1
n_estimators	2000	2000	2000	1000	1000	1000
feature_fraction	1.0	1.0	1.0	0.7	0.7	0.7
bagging_fraction	1.0	1.0	1.0	0.7	0.7	0.7
seed	1	1	1	1	1	1
lambda	1.0	n.a.	1.0	1.0	1.0	n.a.
early_stopping	n.a.	n.a.	n.a.	20	20	20

Table 3.B.3: Average time in seconds for running 2,000 iterations for each dataset on the UCI benchmark datasets. For `msd` and `higgs`, a bagging fraction of 0.1 was used.

	Probabilistic forecast			Point forecast
Dataset	*PGBM-gpu*	*PGBM-cpu*	*NGBoost*	*LightGBM*
`wine` (n=1,599)	100	41	20	1
`naval` (n=12k)	103	167	108	2
`protein` (n=46k)	115	355	389	2
`msd` (n=515k)	136	3,645	6,628	12
`higgs` (n=10,5M)	316	32,200	46,744	135

Table 3.B.4: M5 dataset description.

		`M5`
time series	#	3,049
time series description		item product sales
target		$\mathbb{R}^+$
train samples	#	2,415,359
validation samples	#	85,372
test samples	#	85,372
time step	t	day
features	#	48

4

Hierarchical Forecasting at Scale

Now that we have an idea on how to efficiently generate (probabilistic) forecasts for large-scale settings using two of the most commonly used large-scale forecasting methods (neural networks and Gradient Boosting Machines (GBM)), we observe that in all of these methods, we typically only create forecasts for the lowest granularity of time series, for example in the case of product demand forecasting. However, in e-commerce we often find ourselves in need of forecasts for both lower- and higher (temporal) granularities, such as product category demand forecasts. These granularities can be formalized through the use of a hierarchy. Hierarchical forecasting techniques allow for the creation of forecasts that are coherent with respect to a pre-specified hierarchy of the underlying time series. However, existing hierarchical forecasting techniques scale poorly when the number of time series increases, which limits their applicability at a scale of millions of products. This motivates our third research question:

> **Research Question 3:** *How can we efficiently generate hierarchical forecasts for large-scale settings?*

In this chapter, we propose to learn a coherent forecast for millions of products with a single bottom-level forecast model by using a loss function that directly optimizes the hierarchical product structure. We implement our loss function using sparse linear algebra, such that the number of operations in our loss function scales quadratically rather than cubically with the number of products and levels in the hierarchical

This chapter was published as: O. Sprangers, W. Wadman, S. Schelter, and M. de Rijke. Hierarchical Forecasting at Scale. *International Journal of Forecasting*, In Press, Mar. 2024. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2024.02.006.

structure. The benefit of our sparse hierarchical loss function is that it provides practitioners a method of producing bottom-level forecasts that are coherent to any chosen cross-sectional or temporal hierarchy. In addition, removing the need for a post-processing step as required in traditional hierarchical forecasting techniques reduces the computational cost of the prediction phase in the forecasting pipeline, as well as its deployment complexity.

In our tests on the public M5 dataset, our sparse hierarchical loss function performs up to 10% better as measured by RMSE and MAE compared to the baseline loss function. Next, we implement our sparse hierarchical loss function within an existing gradient boosting-based forecasting model at bol, a large European e-commerce platform. At bol, each day a forecast for the weekly demand of every product for the next twelve weeks is required. In this setting our sparse hierarchical loss resulted in an improved forecasting performance as measured by RMSE of about 2% at the product level, as compared to the the baseline model, and an improvement of about 10% at the product level as measured by MAE. Finally, we found an increase in forecasting performance of about 5–10% (both RMSE and MAE) when evaluating the forecasting performance across the cross-sectional hierarchies that we defined. These results demonstrate the usefulness of our sparse hierarchical loss applied to a production forecasting system at a major e-commerce platform.

4.1 Introduction

In e-commerce, we are often faced with two forecasting challenges. First, forecasts at the lowest granularity — often the individual product level — are required but we also need forecasts at higher granularities, for example at the category, department, or regional level, as higher level forecasts are often needed in logistics and financial planning. Second, forecasts at different time granularities are required, for example daily or weekly forecasts. It is common that separate forecast models are made for each separate (temporal) granularity, and as such these forecasts may not be coherent with each other. Hierarchical forecasting [61] and temporal hierarchical forecasting techniques [12, 101, 125] aim to solve the problem of creating forecasts that are coherent with respect to a pre-specified cross-sectional and/or temporal hierarchy of the underlying time series.

Challenges with existing cross-sectional and temporal hierarchical forecasting techniques Reconciliation methods adjust the forecasts for each level in the hierarchy by minimizing the errors at each forecast level. These methods are applied as a post-processing step that requires a matrix inversion that scales cubically with the number of products or product hierarchies [12, 61, 134]. In settings with millions of products such as in e-commerce, this becomes computationally expensive at prediction time. Neural network methods can optimize for the hierarchy in an end-to-end manner, however, these are either multivariate methods that scale poorly to millions of time series [100] or they can only optimize for the temporal hierarchy [101].

Sparse loss function In order to overcome these scaling issues, we design a sparse *hierarchical loss* (HL) function that directly optimizes both cross-sectional and temporal hierarchical structures. Our corresponding sparsity-aware implementation ensures that the number of operations in our loss function scales quadratically rather than cubically with the number of products and levels in the hierarchical structure, enabling computationally efficient training. The benefit of our sparse hierarchical loss function is that it provides practitioners a method of producing bottom-level forecasts that are coherent to any chosen cross-sectional and temporal hierarchy. In addition, removing the need for a post-processing step as used in traditional hierarchical forecasting techniques reduces the computational cost of the prediction phase in the forecasting pipeline. Furthermore, this also reduces the deployment complexity of the forecasting pipeline.

Evaluation We evaluate our sparse HL function on a gradient-boosted forecasting system on the public M5 dataset [88] and a proprietary dataset from our e-commerce partner. For the M5 dataset, we demonstrate that our implementation provides up to 10% better forecasting performance as measured by both RMSE and MAE compared with (i) reconciliation methods and (ii) baseline bottom-level forecasting methods that use a standard loss function. For the proprietary dataset, we present the results of an offline test on the product-level forecast system of bol, a European e-commerce company with a catalog of millions of unique products. We find that our sparse HL function improves the forecasting performance by about 2% on RMSE and 10% on MAE as compared to the baseline forecasting system. This demonstrates the usefulness of our sparse HL function in a large-scale setting.

Contributions In summary, we contribute:

1. We design a sparse hierarchical loss function that enables direct end-to-end training of cross-sectional and temporal hierarchical forecasts in large-scale settings in Section 4.4.

2. We empirically demonstrate that our sparse hierarchical loss function can outperform existing hierarchical forecasting reconciliation methods by up to 10% in Section 4.5.1. Contrary to most end-to-end hierarchical forecasting methods that leverage neural networks [100, 101], we use LightGBM [68] as our base forecasting model, a highly popular gradient boosting-based forecasting method that is widely used in industry [64] and was used by the majority of the top performing solutions in the M5 forecasting competition [88].

3. We show that our sparse hierarchical loss function scales to large-scale settings and demonstrate a reduction of both training and prediction time of up to an order of magnitude compared to the best hierarchical forecasting reconciliation methods (Section 4.5.1).

4. We present the results of an offline test of our method for the primary product demand forecasting model at bol, a European e-commerce company with a catalogue of millions of unique products, demonstrating an improvement of 2% on RMSE and 10% on MAE as compared to the baseline forecasting system, in Section 4.5.2.

4.2 Related Work

Forecasting for large-scale settings Contemporary large-scale forecasting applications require forecasting many time series concurrently [21]. In academia, there has been a surge in the use of neural network-based forecasting methods, which are methods that commonly learn a single forecast model that can produce forecasts for many time series. We refer the interested reader to the recent survey of Benidis et al. [20] for an overview of these methods. However, tree-based methods topped the M5 forecasting competition [88], which is believed to be due to the strong implementations available of these algorithms [64], such as the LightGBM [68] or XGBoost [24] packages. Our own experience within bol confirms this view: the ease of use, execution speed and strong default performance are key reasons a tree-based method is often the default choice when creating a new forecasting model.

Hierarchical forecasting Hierarchical forecasting [17, 19, 61, 62, 134] and temporal hierarchical forecasting techniques [12, 16, 101, 125] aim to solve the problem of creating forecasts that are coherent with respect to a pre-specified cross-sectional and/or temporal hierarchy of the underlying time series. We divide hierarchical forecasting methods into *Reconciliation methods* and *Other methods*.

Reconciliation methods. For a detailed overview of reconciliation methods, we refer the interested reader to the recent survey of Athanasopoulos et al. [13]. Reconciliation methods solve the hierarchical forecasting problem as a post-processing step by reconciling the forecasts to a pre-specified cross-sectional and/or temporal hierarchy [17, 19, 43, 61, 62, 95, 134]. Limitations of these approaches are (i) that they require a post-processing step, (ii) computing the reconciliation may be computationally expensive, as we show in Section 4.3.2, and (iii) approaches that are computationally less expensive tend to perform worse, as we show in Section 4.5. Recent work by Ben Taieb [16] and Ben Taieb and Koo [17] has improved forecasting performance of previous reconciliation approaches but at the expense of even higher computational costs, as we explain in Section 4.3.

Other methods. In [100, 101] neural network-based end-to-end hierarchical probabilistic forecasting method are proposed to solve the hierarchical forecasting problem. More recently and most closely related to our work, Han et al. [47] introduced SHARQ, a method that reconciles probabilistic hierarchical forecasts during training by employing a regularized loss function that aims to improve hierarchical consistency of bottom-up forecasts through regularization. However, the regularization does not strictly enforce the cross-sectional hierarchy in this method.

4.3 Background

To understand our problem setting and the issues we identify with existing hierarchical forecasting methods, we introduce the hierarchical forecasting problem and common methods of solving the hierarchical forecasting problem.

4.3.1 Problem Definition

Suppose we have n time series written as $\mathbf{y}_t \in \mathbb{R}^n$, where t denotes the time stamp. We are interested in finding h-step ahead estimates $\hat{\mathbf{y}}_h$ of the time series $\mathbf{y}_{T+h}$ using past values $\mathbf{y}_1, \ldots, \mathbf{y}_T$. In our hierarchical forecasting setting, we aim to create forecasts for many time series concurrently, whilst adhering to pre-specified hierarchical relationships that exist between the time series. This can be formalized as follows [13, 59]:

$$\tilde{\mathbf{y}}_h = SG\hat{\mathbf{y}}_h \,, \tag{4.1}$$

where $S \in \{0, 1\}^{n \times n_b}$ is a matrix that defines the hierarchical relationship between the n_b bottom-level time series and the $n_a = n - n_b$ aggregations, $G \in \mathbb{R}^{n_b \times n}$ is a matrix that encapsulates the contribution of each forecast to the final estimate, and $\tilde{\mathbf{y}}_h \in \mathbb{R}^n$ is the vector of forecasts adjusted for the hierarchy. We can use the matrix G to define various forecast contribution scenarios. Note that we can straightforwardly extend Eq. (4.1) to the setting of *temporal hierarchies* [12, 101] by considering forecasts of different time granularities in our vector of base forecasts $\hat{\mathbf{y}}_h$ and using an appropriate choice of S to aggregate series of a different time granularity. We will show how cross-sectional and temporal hierarchical forecasting can be combined in Section 4.4.

The optimal solution to the problem in Eq. (4.1) can be found using *Reconciliation methods* and *Other methods*.

Reconciliation methods *MinTShrink* [13, 134] and variants find the optimal G matrix by solving a minimization problem that has the following solution (ref. Theorem 1 of [134]):

$$G = (J - JWU(U^T WU)^{-1} U^T) \,, \tag{4.2}$$

in which S is partitioned as $S^T = [C^T \; I_{n_b}]$, $J = [0_{n_b \times n_a} \; I_{n_b}]$, $U^T = [I_{n_a} \; -C]$. In *MinTShrink*, W is estimated using the shrunk empirical covariance estimate of [109]. Simpler choices for W, such as the identity matrix, reduce the solution to the *Ordinary Least Squares (OLS)* solution of [61]. In *ERM*, Ben Taieb and Koo [17] note than *MinTShrink* and variants rely on the assumption of unbiasedness of the base forecasts. They relax this assumption by formulating the hierarchical reconciliation problem as an *Empirical Risk Minimization* problem, introducing the *ERM* method. In addition, they propose two regularized variants of *ERM* aimed at reducing forecast variance.

Other methods *Hier-E2E* [100] solves the problem of Eq. (4.1) by learning a neural network model that combines the forecasting and reconciliation step in a single model, resulting in an end-to-end solution removing the need for a post-processing step. Similarly, *COPDeepVAR* [101] is an end-to-end neural network method that enforces temporal hierarchies, however this is a univariate method that is not able to enforce structural hierarchies (i.e., cross-sectional hierarchies) simultaneously, and therefore not suited to our task. *SHARQ* [47] also moves the reconciliation step into the training phase and achieves reconciliation using a regularized loss function, where the regularization enforces the coherency. However, this method does not enforce absolute coherency to the hierarchy.

4.3.2 Scaling Issues of Hierarchical Forecasting Methods

Our main motivation for this chapter are the limitations of prior work for problem settings with many time series.

Scaling issues with reconciliation methods In reconciliation methods, we encounter the following issues when scaling to many time series:

- The reconciliation is performed as a *post-processing* step, and thus has to be performed as an additional step after generating the base forecasts. Even though G in Eq. (4.1) needs to be computed only once using Eq. (4.2), the reconciliation still needs to be performed after each base forecast is produced. Also, G ideally is sparse

[17], but no reconciliation method guarantees this, so computing Eq. (4.1) will generally be a dense matrix-vector product that scales with the number of time series.

- For *MinTShrink* [134], estimating W according to the method of [109] is computationally expensive, with a computational complexity of $O(Nn^2)$, where N denotes the number of training samples used to compute the shrunk covariance estimate. In addition, the shrunk covariance estimate of [109] is not guaranteed to give consistent results in high-dimensional settings [126], making it less applicable for problem settings with many time series. Finally, the estimate for W will generally be a dense matrix, so we cannot make use of efficient sparse algorithms to solve Eq. (4.2). However, even for simpler, sparse choices of W (such as the identity matrix of *OLS* [61]), we still need to invert a matrix of size $n_a \times n_a$ in order to solve Eq. (4.2), which becomes computationally costly for problems with many aggregations, which naturally arise in retail forecasting scenarios. For example, for the M5 retail forecasting competition [87], $n_a = 12,350$, even though there are only 3,049 unique products in this dataset.

- For *ERM* and its regularized variants [17], we need to either invert multiple dense matrices that scale quadratically with the number of time series, or we need to compute a Kronecker product that scales quadratically with the number of time series, followed by an expensive lasso search procedure. Improving the computational complexity of the *ERM* methods is also mentioned in [17] as an avenue for future work.

Scaling issues with other methods *Hier-E2E* [100] is a multivariate method, which means both input and output of the neural network scale with the number of time series. For neural networks, this significantly adds to the training cost and parameter cost as a large number of parameters are required to handle all the separate time series. This in turn requires GPUs with more memory to train these models, which increases cost to operate them.

4.4 Sparse Hierarchical Loss

In this section we present our main technical contribution, the sparse hierarchical loss. First, we show how cross-sectional and temporal hierarchical forecasting can be combined. Then, we introduce our loss function and demonstrate it via a toy example.

Combining cross-sectional and temporal hierarchical forecasting We are interested in finding forecasts that can be aggregated according to a pre-specified cross-sectional hierarchy $S^{cs} \in \{0,1\}^{n^{cs} \times n_b^{cs}}$ and temporal hierarchy $S^{te} \in \{0,1\}^{n^{te} \times n_b^{te}}$:

$$\tilde{\mathbf{y}}_h^{cs} = S^{cs} G^{cs} \hat{\mathbf{y}}_h^{cs} \;, \tag{4.3}$$

$$\tilde{\mathbf{y}}^{te} = S^{te} G^{te} \hat{\mathbf{y}}^{te} \;. \tag{4.4}$$

These equations can be interpreted as follows:

- In Eq. (4.3), we aggregate n_b^{cs} bottom-level time series from the same forecast h across a set of $n^{cs} = n_b^{cs} + n_a^{cs}$ cross-sectional aggregations.
- In Eq. (4.4), we aggregate each time series consisting of n_b^{te} time-steps across a set of $n^{te} = n_b^{te} + n_a^{te}$ temporal aggregations, hence we drop the subscript h.

We will only create bottom-level forecasts, thus $G^{cs} = [0_{n_b^{cs}} \times n_a^{cs} \; I_{n_b^{cs}}]$ and $G^{te} = [0_{n_b^{te}} \times n_a^{te} \; I_{n_b^{te}}]$, yielding:

$$\tilde{\mathbf{y}}_h^{cs} = S^{cs} \hat{\mathbf{y}}_h^{n_b^{cs}} \;, \tag{4.5}$$

$$\tilde{\mathbf{y}}^{te} = S^{te} \hat{\mathbf{y}}^{n_b^{te}} \;, \tag{4.6}$$

where $\hat{\mathbf{y}}_h^{n_b^{cs}}$ and $\hat{\mathbf{y}}^{n_b^{te}}$ denote the bottom-level base forecasts for the cross-sectional and temporal hierarchies, respectively. Considering only bottom-level forecasts has a number of benefits: (i) each forecast is coherent to any hierarchy by design, and (ii) we reduce the number of required forecasts from n to n_b, which can be a significant reduction (there is no need for a forecast for n_a aggregations in the hierarchy). We now construct a matrix of bottom-level base forecasts $\hat{\mathbf{Y}}^{n_b} \in \mathbb{R}^{n_b^{cs} \times n_b^{te}}$, in which the columns represent the forecasts of the bottom-level time series at a timestep h. This allows us to combine (4.5) and (4.6) as follows:

$$\tilde{\mathbf{Y}} = S^{cs}\hat{\mathbf{Y}}^{n_b}(S^{te})^{\intercal} \ , \tag{4.7}$$

in which $\tilde{\mathbf{Y}} \in \mathbb{R}^{n^{cs} \times n^{te}}$ represents the matrix of forecasts aggregated according to both cross-sectional and temporal hierarchies. Equivalently, we can aggregate our bottom-level ground truth values $\mathbf{Y}^{n_b} \in \mathbb{R}^{n_b^{cs} \times n_b^{te}}$:

$$\mathbf{Y} = S^{cs}\mathbf{Y}^{n_b}(S^{te})^{\intercal} \ . \tag{4.8}$$

Sparse hierarchical loss To find the best forecasts for the hierarchical forecasting problem (4.7), we try to find a forecasting model using gradient-based optimization of the following loss function:

$$L = \sum \left[\frac{1}{2} \left(\left(\mathbf{Y} - \tilde{\mathbf{Y}} \right) \odot \left(\mathbf{Y} - \tilde{\mathbf{Y}} \right) \right) \oslash \left(d^{cs} d^{te} \right) \right] \ , \tag{4.9}$$

in which $\sum$ denotes the sum over all $n^{cs} \ \times \ n^{te}$ elements of the matrix contained in the summation, $\odot$ denotes element-wise multiplication, $\oslash$ denotes element-wise division, and the vectors d^{cs} and d^{te} read:

$$d^{cs} = l^{cs} S^{cs} \mathbf{1}^{cs} \ , \tag{4.10}$$

$$d^{te} = \left(l^{te} S^{te} \mathbf{1}^{te} \right)^{\intercal} \ , \tag{4.11}$$

where $S^{cs}\mathbf{1}^{cs}$ and $S^{te}\mathbf{1}^{te}$ denote the row-sum of S^{cs} and S^{te}, respectively, and l^{cs} and l^{te} denote the number of levels in hierarchies S^{cs} and S^{te}, respectively. We will detail the necessity of the element-wise division of Eq. (4.9) by the matrix $(d^{cs}d^{te})$ later in this section. Note that Eq. (4.9) shares similarities with the *Weighted Root Mean Squared Error* from the M5 competition [88].

We can derive the gradient and the second-order derivative of (4.9) with respect to the bottom-level forecasts $\hat{\mathbf{Y}}^{n_b}$ (ref. 4.A for the full derivation):

$$\frac{\partial L}{\partial \tilde{\mathbf{Y}}} = \left(\tilde{\mathbf{Y}} - \mathbf{Y} \right) \oslash \left(d^{cs} d^{te} \right) \ , \tag{4.12}$$

$$\frac{\partial L}{\partial \hat{\mathbf{Y}}^{n_b}} = (S^{cs})^{\intercal} \left(\frac{\partial L}{\partial \tilde{\mathbf{Y}}} \right) S^{te} \ , \tag{4.13}$$

$$\frac{\partial^2 L}{\partial \left(\hat{\mathbf{Y}}^{n_b} \right)^2} = (S^{cs})^{\intercal} \left(\mathbf{1} \oslash \left(d^{cs} d^{te} \right) \right) S^{te} \ . \tag{4.14}$$

Analysis The best possible forecast is achieved when the loss (4.9) is minimized, or equivalently when the gradient (4.12) is zero:

$$\begin{aligned}\frac{\partial L}{\partial \tilde{\mathbf{Y}}} &= \left(\tilde{\mathbf{Y}} - \mathbf{Y}\right) \oslash \left(d^{cs} d^{te}\right) \ , \\ &= \left(S^{cs}\hat{\mathbf{Y}}^{n_b}(S^{te})^\intercal - S^{cs}\mathbf{Y}^{n_b}(S^{te})^\intercal\right) \oslash \left(d^{cs} d^{te}\right) \ , \\ &= \left(S^{cs}\left(\hat{\mathbf{Y}}^{n_b} - \mathbf{Y}^{n_b}\right)(S^{te})^\intercal\right) \oslash \left(d^{cs} d^{te}\right) \ ,\end{aligned}$$

which becomes zero when $\hat{\mathbf{Y}}^{n_b} = \mathbf{Y}^{n_b}$. Thus, the best forecast model is found when each bottom-level forecast equals the ground truth. This is equivalent to the standard (i.e., non-hierarchical) squared error loss often used in forecasting problems. We argue that our hierarchical loss gradient can be seen as a *smoothed* gradient compared to the standard squared error loss gradient (i.e., $\hat{\mathbf{Y}}^{n_b} - \mathbf{Y}^{n_b}$). For example, consider the canonical case where we have two bottom-level time series ($n_b^{cs} = 2$) consisting of two timesteps ($n_b^{te} = 2$). Furthermore, suppose we have a single cross-sectional aggregation (the sum of the two time series, thus $n_a^{cs} = 1$ and $n^{cs} = n_a^{cs} + n_b^{cs} = 3$), and a single temporal aggregation (the sum of the two timesteps, thus $n_a^{te} = 1$ and $n^{te} = n_a^{te} + n_b^{te} = 3$). Finally, there are two levels in our cross-sectional hierarchy and in our temporal hierarchy, thus $l^{cs} = 2$ and $l^{te} = 2$. The standard squared error loss gradient for this problem is:

$$\begin{bmatrix} \frac{\partial L}{\partial \hat{\mathbf{y}}_{0,0}} & \frac{\partial L}{\partial \hat{\mathbf{y}}_{0,1}} \\ \frac{\partial L}{\partial \hat{\mathbf{y}}_{1,1}} & \frac{\partial L}{\partial \hat{\mathbf{y}}_{1,1}} \end{bmatrix} = \begin{bmatrix} e_{0,0} & e_{0,1} \\ e_{1,0} & e_{1,1} \end{bmatrix} \ , \tag{4.15}$$

in which $e_{i,j}$ denotes the bottom-level forecast error ($\hat{y}_{i,j} - y_{i,j}$) of the i-th bottom-level timeseries and j-th timestep, respectively. For our hierarchical loss, Eq. (4.7) reads:

$$\tilde{\mathbf{Y}} = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}}_{S^{cs}} \underbrace{\begin{bmatrix} \hat{\mathbf{y}}_{0,0} & \hat{\mathbf{y}}_{0,1} \\ \hat{\mathbf{y}}_{1,0} & \hat{\mathbf{y}}_{1,1} \end{bmatrix}}_{\hat{\mathbf{Y}}^{n_b}} \underbrace{\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}}_{(S^{te})^\intercal} , \tag{4.16}$$

and the gradient of the loss with respect to the bottom level time series Eq. (4.13) reads (ref. 4.B for the full derivation):

$$\begin{bmatrix} \frac{\partial L}{\partial \hat{\mathbf{y}}_{0,0}} & \frac{\partial L}{\partial \hat{\mathbf{y}}_{0,1}} \\ \frac{\partial L}{\partial \hat{\mathbf{y}}_{1,1}} & \frac{\partial L}{\partial \hat{\mathbf{y}}_{1,1}} \end{bmatrix} = \begin{bmatrix} \frac{9}{16}e_{0,0} + \frac{3}{16}e_{1,0} + \frac{3}{16}e_{0,1} + \frac{1}{16}e_{1,1} & \frac{9}{16}e_{0,1} + \frac{3}{16}e_{0,1} + \frac{3}{16}e_{1,1} + \frac{1}{16}e_{1,0} \\ \frac{9}{16}e_{1,0} + \frac{3}{16}e_{0,0} + \frac{3}{16}e_{1,1} + \frac{1}{16}e_{0,1} & \frac{9}{16}e_{1,1} + \frac{3}{16}e_{1,0} + \frac{3}{16}e_{0,1} + \frac{1}{16}e_{0,0} \end{bmatrix} . \tag{4.17}$$

When we compare this result to the standard squared error loss gradient Eq. (4.15), we find that we *smooth* the bottom-level gradient by adding to it portions of the gradients of all cross-sectional and temporal aggregations the bottom-level series belongs to. This derivation also shows the motivation of adding the denominator matrix $(d^{cs}d^{te})$ to the loss function (4.9): it is neccessary to scale the aggregation gradients by the number of elements in the aggregation, otherwise the magnitude of the gradient grows with the number of time series and the number of levels in the hierarchy, which we found to be undesirable when trying to facilitate stable learning. Thus, we add (portions of) the average gradient of the aggregations to the bottom-level gradient.

Sparsity S^{cs} and S^{te} are highly sparse. For example, S^{cs} has at most $n_b^{cs}l^{cs}$ non-zero elements: the number of bottom-level time series multiplied by the number of aggregations in the hierarchy. Hence, the overall sparsity of S^{cs} is given by $1 - \frac{n_b^{cs}l^{cs}}{n^{cs}n_b^{cs}}$. For the M5 dataset [87], $n_b^{cs} = 3,049$, $l^{cs} = 12$, $n^{cs} = 42,840$, corresponding to a sparsity of 99.97%. Next, the matrix of bottom-level ground truth values $\mathbf{Y}^{n_b}$ in (4.8) may be sparse too, for example in the case of products that are not on sale for every timestep n_b^{te} in the dataset. All these sources of sparsity motivate the use of sparse linear algebra when computing Eqs. (4.9)–(4.14).

Implementation We implement the hierachical loss Eq. (4.9), the bottom-level gradient Eqs. (4.12)–(4.13) and second-order derivative Eq. (4.14) in Python using the sparse library from *SciPy* [132]. Note that Eqs. (4.12)–(4.13) can be rearranged:

$$\frac{\partial L}{\partial \hat{\mathbf{Y}}} = \frac{(S^{cs})^{\intercal}}{d^{cs}} \left(\tilde{\mathbf{Y}} - \mathbf{Y}\right) \frac{S^{te}}{d^{te}} , \tag{4.18}$$

such that the parts before and after the brackets can be precomputed as they do not depend on the forecast values $\tilde{\mathbf{Y}}$, avoiding a costly division operation inside a training iteration. Also note that the second-order derivative Eq. (4.14) does not depend on the forecast values $\tilde{\mathbf{Y}}$, so it can be precomputed as well. Our implementation, including the code to reproduce the experiments on public data from Section 4.5, is available on GitHub.[1]

4.5 Experiments

In this section we empirically verify the usefulness of our sparse hierarchical loss. First, we evaluate forecasting accuracy using a set of experiments on the public M5 dataset [87]. Then, we evaluate our sparse hierarchical loss in an offline experiment on a proprietary dataset from our e-commerce partner.

4.5.1 Public Datasets

Task & dataset Our task is to forecast product demand. We use the M5 dataset [87] for our offline, public dataset experiments. The M5 dataset contains product-level sales from Walmart for 3,049 products across 10 stores in the USA. Furthermore, the dataset contains 12 cross-sectional product aggregations (e.g., department, region), which allow us to test hierarchical forecasting performance. We preprocess the dataset resulting in a set of features as described in Appendix 4.C. We forecast 28 days into the future.

Baseline models For our baseline forecasting model, we primarily use LightGBM [68], trained to predict one-day ahead. We subsequently recursively generate predictions for 28 days. Tree-based models dominated the M5 forecasting competition due to their strong performance and ease of use [64, 88]. Moreover, our e-commerce partner's primary product forecasting is a LightGBM-based model, so we expect results from offline experiments on public datasets to transfer to our proprietary setting when using the same base forecasting model. We compare the performance of our LightGBM models against traditional statistical methods ARIMA

[1]`https://github.com/elephaint/hfas/`

[22], ETS [60], Theta [11], SeasonalNaive [59], Naive [59] and Croston [30]. We note that deep learning-based approaches are becoming more prevalent in e-commerce [74], especially with the rise of the transformer-architecture in forecasting models [80, 82]. We consider this for future work, and did not consider this for our study as (i) the cloud cost to operate these models is 10x higher for our e-commerce partner as compared to a tree-based model, and (ii) none of the neural network-based methods are able to scale to the size of our e-commerce partner, as explained in Section 4.3.2.

Experimental setup We consider the following scenarios to test our hierarchical sparse loss function against baseline forecasting systems:

1. `Bottom-up`. We train a single global model on only the bottom-level time series. Subsequently, the bottom-level forecasts are aggregated to obtain the aggregated (reconciled) forecasts.

2. `Separate aggregations`. We train separate models for every aggregation in the hierarchy, resulting in 12 models for the entire M5 dataset.

3. `Global`. We train a single global model on all time series in the dataset, including all the aggregations.

For the first scenario in our experiments (`Bottom-up`), we vary both the *objective* (i.e., the loss function that is optimized by LightGBM) and the *evaluation metric* (i.e., the loss function that governs early-stopping during hyperparameter optimization). For the *objective*, we consider the *squared error loss* (SL), the *Tweedie loss* (TL) and our *sparse hierarchical loss* (HL). The Tweedie loss is a loss function that assumes that the time series follow a distribution somewhere in between a Poisson and a Gamma distribution, which is useful in zero-inflated settings such as retail demand forecasting. It is a loss function that was favored by contestants in the M5 forecasting competition [64], and it is the loss also used in the primary forecasting system of our e-commerce partner.

For the latter two scenarios, we will obtain non-coherent forecasts. Thus, these methods require a reconciliation post-processing step to reconcile the forecasts to the hierarchy. We employ the following cross-sectional reconciliation methods:

- **Base**. No reconciliation is performed.
- **OLS**. Ordinary Least Squares (OLS) [61], where W in Eq. (4.2) is the identity matrix.
- **WLS-struct**. and **WLS-var**. Weighted Least Squares (WLS) [134], where W in Eq. (4.2) is a diagonal matrix containing the sum of the rows of S (*WLS-struct*) or the in-sample forecast errors (*WLS-var*), respectively.
- **MinT-shrink**. *Trace Minimization* [134], where W in Eq. (4.2) is the shrunk covariance matrix of in-sample forecast errors. We also experimented with using the non-shrunk covariance matrix of the in-sample forecast errors (*MinT-cov*), but this produced erroneous/high variance results, which we attribute to precisely the motivation to shrink the covariance matrix in *MinT-shrink*: to reduce the variance when the number of time series considered becomes very large.
- **ERM**. The *Empirical Risk Minimization* (ERM) method [17]. Due to computational issues explained in Section 4.3.2, we were not able to apply the regularized ERM variants to our experiments, but only the unregularized variant.

We optimize the hyperparameters of each of the LightGBM models by Bayesian hyperparameter optimization using Optuna [7]. The settings for the hyperparameter optimization can be found in Appendix 4.D. Each model is trained for 10 different random seeds, and our results are based on the mean and standard deviation of those 10 rollouts. For the traditional statistical methods we use Nixtla's StatsForecast [41], which includes automatic optimization of the hyperparameters of the statistical methods.

Evaluation We evaluate our results for every aggregation in the hierarchy using the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) [59]. In the results section, we present the RMSE / MAE relative to the `Bottom-up` scenario using the squared-loss objective with the squared-loss metric. For absolute values and standard deviation of the results, see 4.E.

Results – LightGBM as baseline model For our first experiment, we only consider cross-sectional hierarchies (i.e., $S^{te} = I_{n_b^{te}}$). We present our results on relative RMSE using LightGBM as baseline model in Table 4.4 and conclude the following:

- The best performing method is the `Bottom-up`-scenario combined with our sparse hierarchical loss as objective, outperforming the baseline by 0–20% across aggregations. This holds for both settings in which we use our sparse hierarchical loss.
- Even when we only use our sparse hierarchical loss as an evaluation metric during training whilst optimizing the standard squared loss (the SL/HL scenario), we already see a small improvement of $\pm 5\%$ across aggregations.
- Even though the Tweedie loss improves over the baseline loss, our sparse hierarchical loss function still outperforms it by $\pm 5\%$ across aggregations.
- From the reconciliation methods, *MinT-shrink* and *WLS-var* perform best in the `Separate aggregations`-scenario, although the performance delta across aggregations is still $\pm$5-30% as compared to the best (our) method.

For relative MAE, we present our results in Table 4.5. We find that overall, our sparse hierachical loss still performs best by $\pm 5\%$ compared to other loss functions and scenarios. However, the results are more nuanced: we find that *MinT-shrink* in the `Separate aggregations`-scenario performs strong as well. In addition, we also find that the Tweedie loss (TL) performs relatively well. This finding corroborates the usefullness of the TL in intermittent demand settings, such as retail, where zero demand is often observed.

Next, we compare our findings against the forecasting results when employing different baseline models in Table 4.1. For brevity, we only show the metrics for all time series combined (incl. aggregations). In addition, we only show a single reconciliation method for the other baseline models, as we found little difference in results when employing different reconciliation methods. We then find that in terms of RMSE, our sparse hierachical loss outperforms the other baseline models by at least 50%, and in terms of MAE by at least 10%. This verifies that on

Table 4.1: Forecasting results for all time series (incl. aggregations) on the M5 dataset, using different baseline models. We show absolute and relative RMSE and MAE. Lower is better, and bold indicates the best performing method.

		RMSE		MAE	
Model	Reconciliation	Abs.	Rel.	Abs.	Rel.
LightGBM (SL/SL)	None	*22.39*	*1.00*	*2.20*	*1.00*
LightGBM (HL/HL)	None	**19.54**	**0.87**	**2.10**	**0.95**
LightGBM (HL/SL)	None	**19.59**	**0.88**	**2.10**	**0.95**
ARIMA	MinT-shrink	39.88	1.78	2.43	1.10
ETS	MinT-shrink	36.48	1.63	2.35	1.07
Theta	MinT-shrink	36.66	1.64	2.39	1.08
Croston	None	39.40	1.76	2.76	1.25
Naive	None	74.91	3.35	3.95	1.80
Seasonal Naive	None	39.40	1.76	2.76	1.25

this dataset and with this type of problem, using a more complex model such as LightGBM greatly improves forecasting performance, as was also shown in the M5 forecasting competition [88].

Analysis: impact of hierarchy We investigate the impact of the choice of hierarchy.

Temporal hierarchies. As we noted before, we only used cross-sectional aggregations in our first experiments. We now also include temporal aggregations by aggregating our bottom-level time series across years, weeks and months. We ablate for every setting and show the results in Table 4.2. Interestingly, we find that using temporal hierarchies jointly with cross-sectional hierarchies reduces forecasting performance by ±35% (RMSE) and ±17% (MAE). This setting is even worse than only using temporal hierarchies, which performs worse than using only cross-sectional hierarchies by ±26% (RMSE) and ±12% (MAE). We further analyze these results by studying the RMSE across the forecast days in Table 4.3. As noted before, we forecast 28 days ahead, and each forecast is created by recursively applying the one-step ahead LightGBM model. We find that as we forecast further into the future, the setting with only using cross-sectional aggregations starts to perform better by up to

Table 4.2: Forecasting results for all time series (incl. aggregations) on the M5 dataset, ablating for the use of cross-sectional and temporal hierarchies. We show absolute and relative RMSE and MAE, with the standard deviation in brackets. Lower is better, and bold indicates the best performing method. Note that when not using cross-sectional nor temporal aggregations, the hierarchical loss is equal to the standard squared error loss.

Hierarchies		RMSE		MAE	
Cross-sectional	Temporal	Abs.	Rel.	Abs.	Rel.
No	No	*22.39* (0.16)	*1.00*	*2.20* (0.01)	*1.00*
Yes	No	**19.54** (0.38)	**0.87**	**2.10** (0.01)	**0.95**
Yes	Yes	29.81 (1.52)	1.33	2.47 (0.04)	1.12
No	Yes	26.65 (0.32)	1.19	2.36 (0.01)	1.07
Random	No	25.62 (4.75)	1.14	2.31 (0.15)	1.05

20% as compared to the baseline where we do not use any aggregations. Again, the setting where we employ temporal hierarchies too shows relatively bad performance across all forecast day buckets.

Random hierarchies. In hierarchical forecasting problems, the aggregation matrices S^{cs} and S^{te} are commonly fixed a priori and considered constant during training and prediction. As we are performing the reconciliation in an end-to-end fashion during training, we can modify these matrices at every iteration. This allows us to understand the robustness of our solution to possible misspecification of the hierarchy, and more generally, to what extent the choice of the hierarchy has an effect on forecasting performance. We perform an experiment by randomly sampling an S^{cs}-matrix at every iteration of the LightGBM training process. At every iteration, we sample uniformly at random (i) a number of levels for the cross-sectional hierarchy and (ii) a number of maximum categories for the level and construct a random S^{cs}-matrix to be used in the gradient, Eq. (4.13), and hessian, Eq. (4.14). We validate and test on the 'true' S^{cs}-matrix. We present the results in Table 4.2 and Table 4.3, under 'Random'. We find that performance deteriorates by about 30% as compared to the baseline, and we also find that our results in this experiment show higher variance compared to the baseline methods. Thus, misspecification of the hierarchy can severely deteriorate forecasting performance.

Table 4.3: Forecasting results for all time series (incl. aggregations) on the M5 dataset, ablating for the use of cross-sectional and temporal hierarchies. We show relative RMSE for several forecasting day buckets of the forecast. Lower is better, and bold indicates the best performing method.

Hierarchies		Forecast day			
Cross-sectional	Temporal	1–7	8–14	15–21	22–28
No	No	*1.00*	*1.00*	*1.00*	*1.00*
Yes	No	**0.98**	**0.99**	**0.81**	**0.80**
Yes	Yes	1.75	1.50	0.96	1.66
No	Yes	1.37	1.24	0.97	1.41
Random	No	1.25	1.16	1.00	1.32

Analysis: time complexity We investigate the computational time complexity required to perform training and prediction for each scenario and present the results in Table 4.6. The training and prediction time complexity is indicated by how the training time and prediction time, respectively, scales with respect to the default LightGBM training and prediction time complexity. We first investigate the case where we only consider cross-sectional hierarchies. This case is indicated by 'HL' in Table 4.6. First, we note that adding our hierarchical loss objective adds a component to the time complexity that scales with $(n_b^{cs})^3$, as we need to compute (4.12). However, our sparse implementation of the hierarchical loss reduces this component from $(n_b^{cs})^3$ to $(n_b^{cs})^2\, l^{cs}$, effectively reducing the scaling from cubic to quadratic in the number of bottom-level time series, as l^{cs} is generally small. In the reconciliation scenarios, we always need to compute a matrix inversion to solve Eq. (4.2) that scales cubically with the number of cross-sectional aggregations n_a^{cs} or with the total number of time series n^{cs}. The first is not problematic as generally $n_a^{cs} \ll n_b^{cs}$ in large-scale settings, but methods with this time complexity consequently trade in performance, as we observed in Table 4.4. To empirically verify the differences in asymptotic time complexity, we recorded the training and prediction time for each scenario. We show timings for training and prediction for a single store of the M5 dataset (4M training samples) and for the entire M5 dataset (52M training samples), to provide an indication of scaling when the problem size increases by an

Table 4.4: Forecasting results for all stores on the M5 dataset, using LightGBM as baseline model. We report relative RMSE as compared to the baseline (shown in italic). Lower is better, and bold indicates best method for the aggregation, taking into account standard deviation of the best method across the 10 seeds. For absolute values and standard deviation of the results, see Appendix 4.E. The Bottom-up scenario using the HL loss commonly outperforms all other scenarios.

						Store			Product		State				
Scenario/Objective	Metric	Reconciliation	Product	Department	Category	Department	Category	Total	Store	State	Department	Category	Total	Total	All series
`Bottom-up`															
SL	*SL*	*None*	***1.00***	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*
SL	HL	None	**1.00**	0.97	0.95	0.98	0.97	0.98	0.99	1.00	0.98	0.96	0.99	1.00	0.98
HL	HL	None	**1.00**	**0.88**	**0.80**	**0.93**	**0.89**	**0.93**	**0.97**	**0.99**	**0.89**	**0.84**	**0.88**	**0.87**	**0.87**
HL	SL	None	**1.00**	**0.88**	**0.81**	**0.94**	**0.90**	**0.94**	**0.96**	**0.98**	**0.89**	**0.84**	**0.88**	**0.87**	**0.88**
TL	HL	None	**1.00**	0.95	0.96	0.99	1.00	1.00	0.99	1.00	0.97	0.98	0.98	0.96	0.97
TL	SL	None	**1.00**	0.94	0.93	1.00	1.00	1.00	0.99	1.00	0.97	0.98	0.98	0.93	0.96
TL	TL	None	1.17	2.72	2.81	1.76	1.83	1.73	1.52	1.33	2.15	2.18	2.08	2.71	2.38
`Sep. agg.`															
SL	SL	Base	**1.00**	1.44	1.29	1.19	1.14	1.14	1.01	0.99	1.23	1.34	1.27	1.60	1.35
SL	SL	OLS	**1.00**	1.39	1.41	1.10	1.06	1.07	1.00	1.00	1.20	1.19	1.23	1.50	1.30
SL	SL	WLS-struct	**1.00**	1.26	1.37	1.03	1.05	1.02	0.99	0.99	1.11	1.16	1.16	1.39	1.23
SL	SL	WLS-var	**1.00**	1.12	1.23	0.99	1.02	0.99	0.99	0.99	1.03	1.09	1.07	1.22	1.12
SL	SL	MinT-shrink	**1.00**	1.15	1.27	0.97	0.99	0.97	1.00	1.00	1.03	1.09	1.09	1.30	1.15
SL	SL	ERM	1.22	1.25	1.29	1.07	1.03	1.07	1.17	1.22	1.17	1.14	1.22	1.49	1.26
`Global`															
SL	SL	Base	1.02	1.33	1.45	1.09	1.10	1.10	1.03	1.03	1.25	1.27	1.81	1.57	1.46
SL	SL	OLS	1.01	1.32	1.39	1.07	1.09	1.16	1.02	1.02	1.20	1.25	1.38	1.49	1.34
SL	SL	WLS-struct	1.01	1.38	1.54	1.08	1.13	1.11	1.03	1.02	1.19	1.28	1.27	1.55	1.36
SL	SL	WLS-var	1.01	1.51	1.70	1.18	1.27	1.22	1.03	1.02	1.31	1.43	1.38	1.66	1.48
SL	SL	MinT-shrink	1.03	1.26	1.41	1.05	1.10	1.15	1.06	1.05	1.11	1.17	1.24	1.54	1.30
SL	SL	ERM	1.21	1.59	1.69	1.26	1.28	1.34	1.20	1.23	1.45	1.49	1.61	1.80	1.59

Table 4.5: Forecasting results for all stores on the M5 dataset, using LightGBM as baseline model. We report relative MAE as compared to the baseline (shown in italic). Lower is better, and bold indicates best method for the aggregation, taking into account standard deviation of the best method across the 10 seeds. For absolute values and standard deviation of the results, see Appendix 4.E.

						Store			Product		State				
Scenario/Objective	Metric	Reconciliation	Product	Department	Category	Department	Category	Total	Store	State	Department	Category	Total	Total	All series
`Bottom-up`															
SL	*SL*	*None*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*
SL	HL	None	1.00	0.98	0.96	0.98	0.98	0.99	1.00	1.00	0.97	0.97	1.00	1.02	0.99
HL	HL	None	0.99	**0.81**	**0.78**	**0.90**	**0.89**	0.94	0.98	0.99	**0.85**	**0.83**	**0.89**	**0.88**	**0.95**
HL	SL	None	0.99	**0.81**	**0.77**	**0.90**	**0.90**	0.94	0.98	0.99	**0.85**	**0.83**	**0.89**	**0.87**	**0.95**
TL	HL	None	0.98	**0.83**	0.82	0.93	0.94	0.96	0.97	0.98	0.88	0.89	0.92	**0.88**	**0.95**
TL	SL	None	0.99	0.85	0.84	0.95	0.95	0.98	0.98	0.99	0.90	0.91	0.96	**0.90**	**0.96**
TL	TL	None	1.02	2.06	2.39	1.47	1.61	1.65	1.14	1.07	1.69	1.88	1.97	2.86	1.29
`Sep. agg.`															
SL	SL	Base	1.00	1.11	1.06	1.02	1.10	1.08	0.96	0.97	1.03	1.15	1.20	1.55	1.02
SL	SL	OLS	**0.96**	1.08	1.16	1.01	0.99	1.00	0.96	0.97	1.00	1.01	1.12	1.49	0.99
SL	SL	WLS-struct	0.97	0.99	1.13	0.92	0.95	0.95	**0.96**	**0.97**	0.93	0.98	1.04	1.42	0.98
SL	SL	WLS-var	0.98	0.91	1.00	0.91	0.92	**0.92**	0.96	0.97	0.90	0.93	0.95	1.16	**0.96**
SL	SL	MinT-shrink	0.97	0.93	1.05	**0.90**	**0.90**	**0.91**	0.96	0.97	0.89	0.92	0.98	1.30	**0.96**
SL	SL	ERM	1.18	1.17	1.21	1.09	1.06	1.07	1.19	1.22	1.11	1.12	1.19	1.57	1.18
`Global`															
SL	SL	Base	1.04	1.04	1.17	1.00	1.02	1.05	0.99	0.99	1.04	1.09	1.62	1.58	1.05
SL	SL	OLS	0.98	1.09	1.17	1.05	1.07	1.13	0.99	1.00	1.09	1.13	1.27	1.50	1.03
SL	SL	WLS-struct	0.99	1.09	1.26	0.98	1.01	1.04	1.00	0.99	1.00	1.07	1.15	1.57	1.02
SL	SL	WLS-var	0.99	1.24	1.40	1.09	1.13	1.13	1.01	1.00	1.14	1.21	1.25	1.69	1.06
SL	SL	MinT-shrink	0.99	1.13	1.24	1.06	1.08	1.15	1.02	1.01	1.05	1.07	1.15	1.60	1.04
SL	SL	ERM	1.17	1.49	1.62	1.30	1.34	1.37	1.19	1.20	1.39	1.49	1.58	1.96	1.27

order of magnitude. First, we note that using our sparse implementation of the HL reduces training time by a factor of $3\times$ when training for all stores. Second, our sparse HL has a prediction time similar to the baseline (SL).

Next, we find that the training time of our sparse hierarchical loss is two orders of magnitude faster than reconciliation methods in the `Separate aggregations`-scenario. This is mainly due to the many individual models that need to be trained in this scenario and thus shows a clear benefit of having just a single model. We observe an order of magnitude difference in prediction time when comparing the sparse hierarchical loss to the `Separate aggregations`-scenario when predictinf all stores. Again, this shows a clear benefit of having just a single model for this forecasting task.

For the `Global`-scenario, we see that reconciliation methods require a smaller training time when training for all stores (about twice less), however that scenario also did not give strong forecasting performance as we established in Table 4.4. Also, the prediction time using our sparse HL is an order of magnitude lower. As ML costs in production systems mainly consist of prediction costs, having a lower prediction time is beneficial.[2]

Finally, we also show the time complexity of using both cross-sectional and temporal hierarchies jointly, as indicated by '$HL+$' in Table 4.6. Adding temporal hierarchies adds another matrix multiplication that scales with the number of timesteps to the complexity. In our experiments, we find that adding temporal hierarchies results in a twice higher training time when training for all stores and a 50% higher prediction time when predicting for all stores. We view it as potential future work to investigate how to perform this end-to-end learning of both cross-sectional and temporal hierarchies even more efficiently.

To conclude, we showed that our sparse HL incurs some additional training overhead but no additional prediction overhead as compared to the base case SL, whereas it does not require the additional reconciliation step that reconciliation methods require.

[2]For example, Google designed its first TPU for inference: `https://techcrunch.com/2017/05/17/google-announces-second-generation-of-tensor-processing-unit-chips`.

Table 4.6: Computational time complexity and observed timings in seconds for all scenarios. The complexity is indicated by how respectively the training time and prediction time scales with respect to the default LightGBM training/prediction time L, where n_b^{te} denotes the number of timesteps per time series, n_b^{cs} denotes the number of bottom-level time series in the hierarchy, $n_b^{cs_l}$ the number of time series in each level in the hierarchy and l^{cs} the number of levels in the cross-sectional hierarchy, n^{cs} (n^{te}) the total number of cross-sectional (temporal) aggregations, and $n_a = n - n_b$.

			Complexity		Training time (s)		Prediction time (s)	
Scenario/Obj.	Metric	Reconciliation	Training	Prediction	1 store	All stores	1 store	All stores
`Bottom-up`								
SL	SL	None	$O\left(L\left(n_b^{te} n_b^{cs}\right)\right)$	$O\left(L\left(n_b^{te} n_b^{cs}\right) + n_b^{te}\left(n_b^{cs}\right)^3\right)$	8	173	1.1	11
HL (dense)	HL (dense)	None	$O\left(L\left(n_b^{te} n_b^{cs} + n_b^{te}\left(n_b^{cs}\right)^3\right)\right)$	$O\left(L\left(n^{te} n^{cs}\right) + n_b^{te}\left(n_b^{cs}\right)^3\right)$	14	1,185	1.1	10
HL (sparse)	HL (sparse)	None	$O\left(L\left(n_b^{te} n_b^{cs} + n_b^{te}\left(n_b^{cs}\right)^2 l^{cs}\right)\right)$	$O\left(L\left(n^{te} n^{cs}\right) + n_b^{te}\left(n_b^{cs}\right)^2 l^{cs}\right)$	12	318	0.1	11
HL+ (sparse)	HL+ (sparse)	None	$O\left(L\left(n_b^{te} n_b^{cs} + n_b^{te}\left(n_b^{cs}\right)^2 l^{cs} n_b^{cs}\left(n_b^{te}\right)^2 l^{te}\right)\right)$	$O\left(L\left(n^{te} n^{cs}\right) + n_b^{te}\left(n_b^{cs}\right)^2 l^{cs} n_b^{cs}\left(n_b^{te}\right)^2 l^{te}\right)$		723		15
`Sep. agg.`								
SL	SL	Base		$O\left(l^{cs} \cdot L\left(n_b^{cs_l} n_b^{te_l}\right)\right)$			4.4	103
SL	SL	OLS		$O\left(l^{cs} \cdot T\left(n_b^{cs_l} n_b^{te_l}\right) + \left(n_a^{cs}\right)^3\right)$			4.5	149
SL	SL	WLS-struct	$O\left(l^{cs} \cdot L\left(n_b^{cs_l} n_b^{te_l}\right)\right)$	$O\left(l^{cs} \cdot T\left(n_b^{cs_l} n_b^{te_l}\right) + \left(n_a^{cs}\right)^3\right)$	11	36,018	4.5	151
SL	SL	WLS-var		$O\left(l^{cs} \cdot T\left(n_b^{cs_l} n_b^{te_l}\right) + \left(n_a^{cs}\right)^3\right)$			4.5	151
SL	SL	MinT-shrink		$O\left(l^{cs} \cdot T\left(n_b^{cs_l} n_b^{te_l}\right) + \left(n^{cs}\right)^3\right)$			5.8	305
SL	SL	ERM		$O\left(l^{cs} \cdot T\left(n_b^{cs_l} n_b^{te_l}\right) + \left(n^{cs}\right)^3\right)$			6.0	239
`Global`								
SL	SL	Base		$O\left(T\left(n^{te} n^{cs}\right)\right)$			2.4	71
SL	SL	OLS		$O\left(T\left(n^{te} n^{cs}\right) + \left(n_a^{cs}\right)^3\right)$			2.5	118
SL	SL	WLS-struct	$O\left(L\left(n^{te} n^{cs}\right)\right)$	$O\left(T\left(n^{te} n^{cs}\right) + \left(n_a^{cs}\right)^3\right)$	4	173	2.5	120
SL	SL	WLS-var		$O\left(T\left(n^{te} n^{cs}\right) + \left(n_a^{cs}\right)^3\right)$			2.5	120
SL	SL	MinT-shrink		$O\left(T\left(n^{te} n^{cs}\right) + \left(n^{cs}\right)^3\right)$			4.2	274
SL	SL	ERM		$O\left(T\left(n^{te} n^{cs}\right) + \left(n^{cs}\right)^3\right)$			4.0	207

Table 4.7: Comparison of dataset characteristics between the M5 dataset and the proprietary dataset. We split the weekly demand by weekly demand buckets, and show the percentage of samples and percentage of demand for each bucket.

Weekly	% samples		% demand	
demand	M5	Proprietary	M5	Proprietary
0	40.73%	34.15%	0%	0%
1	7.92%	19.23%	1.02%	3.99%
2–10	33.28%	37.31%	20.93%	31.6%
11-100	17.21%	8.89%	57.38%	46.01%
101–500	0.84%	0.39%	18.59%	14.27%
501+	0.02%	0.02%	2.08%	4.13%
Total	100.00%	100.00%	100.00%	100.00%

4.5.2 Proprietary Datasets

At our e-commerce partner bol, a LightGBM-based forecasting model is used as the primary product forecasting model. The model is used to forecast weekly product demand for 12 weeks. Every day, 12 separate models are trained, each tasked to forecast demand for a single week for every product. The model is used to forecast the majority of the products on sale at any moment, which are approximately 5 million unique items. We investigate the use of our sparse hierarchical loss function as a drop-in replacement for the existing Tweedie loss that is used within the company.

Dataset The offline dataset consists of 36M training samples from the period January 2017 to the end of June 2021. We test on 55M samples from the period July 2021 to January 2022. We show statistics of the proprietary dataset as compared to the M5 dataset in Table 4.7, in which we split the weekly demand of both datasets according to weekly demand buckets used by our e-commerce partner. In Table 4.7, we find that the M5 dataset and our proprietary dataset share demand characteristics in terms of sparsity (i.e., zero demand), which is 41% for M5 and 34% for our proprietary dataset, respectively. In general, we find that the two datasets share sufficient weekly demand density characteristics to warrant using our sparse HL on our proprietary dataset. The proprietary dataset contains 19 proprietary features, which are similar to those used in the M5 dataset (ref. Table 4.C.1), and consist of (i) product categorical features, (ii) weekly demand (target) lagged features, and (iii) seasonality features.

Experimental setup The baseline model for every weekly forecast model is a LightGBM model with a *Tweedie loss* (TL). We replace the TL with our HL and investigate forecasting performance on the test set. We apply log-scaling to the target values. For the HL, we use the proprietary aggregations *product_group* and *seasonality_group*, each containing respectively ±70 and ±6,000 unique values. We have $n_b^{cs} = \pm 5\text{M}$ bottom-level time series and $n_a^{cs} = \pm 6,070$ aggregated time series across $l^{cs} = 4$ levels: *product* (bottom-level), *product_group*, *seasonality_group* and *total*.

Results On average, we find that our sparse HL outperforms the existing TL model by about 1–2% on RMSE and ±10% on MAE. We further investigate the performance by investigating how the RMSE and MAE vary across the 12 forecasting horizons and weekly demand buckets, and present the results in Figures 4.1–4.2. We find that our sparse HL performs best on both RMSE and MAE on the lower weekly demand buckets (up to 100 products sold per week), where it outperforms the TL averaged over all the forecasting horizons. The TL is clearly better for higher weekly demand buckets, commonly outperforming the HL and SL by up to 5%. Next, we investigate forecasting performance across the cross-sectional hierarchies that we defined. We show the results in Figure 4.3. We find that for most forecasting horizons, the HL and SL outperform the TL, with an average outperformance of the HL over the TL of ±10% at the product level, ±5% at product group level and ±4–7% at seasonality group level. Hence, we are able to confirm some of the results we found in the M5 experiment, although the baseline SL performed quite strong in this experiment as well. We believe this is due to the M5 experiment having much more hierarchical levels (12 as compared to the 4 we used for our proprietary dataset experiment), since the HL is equal to the SL in the case of no hierarchies, and with fewer hierarchies the HL thus becomes closer to the SL. Hence, we believe our HL is most useful in settings with both many timeseries as well as many hierarchies.

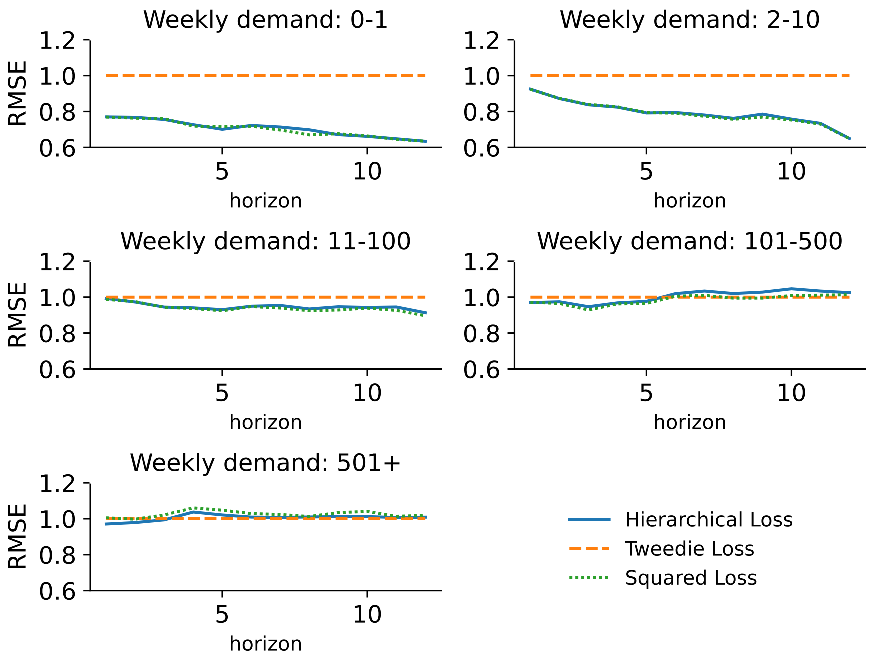


Figure 4.1: Forecasting results for the primary product forecasting model at our e-commerce partner bol. We show RMSE by weekly demand bucket relative to the Tweedie loss baseline for each forecasting horizon (week). The Hierarchical loss outperforms the Tweedie loss on smaller weekly demand buckets.

To conclude, we find that this experiment demonstrates the usefulness of our sparse HL applied to a production forecasting system at a major e-commerce platform.

4.6 Conclusion

In this chapter, we introduced a sparse hierarchical loss function to perform hierarchical forecasting in large-scale settings. We demonstrated that we are able to outperform existing hierarchical forecasting methods both in terms of performance as measured by RMSE and MAE by up to 10% as well as in terms of computational time required to perform the end-to-end hierarchical forecasting in large-scale settings, reducing prediction time as compared to the best hierarchical forecasting recon-

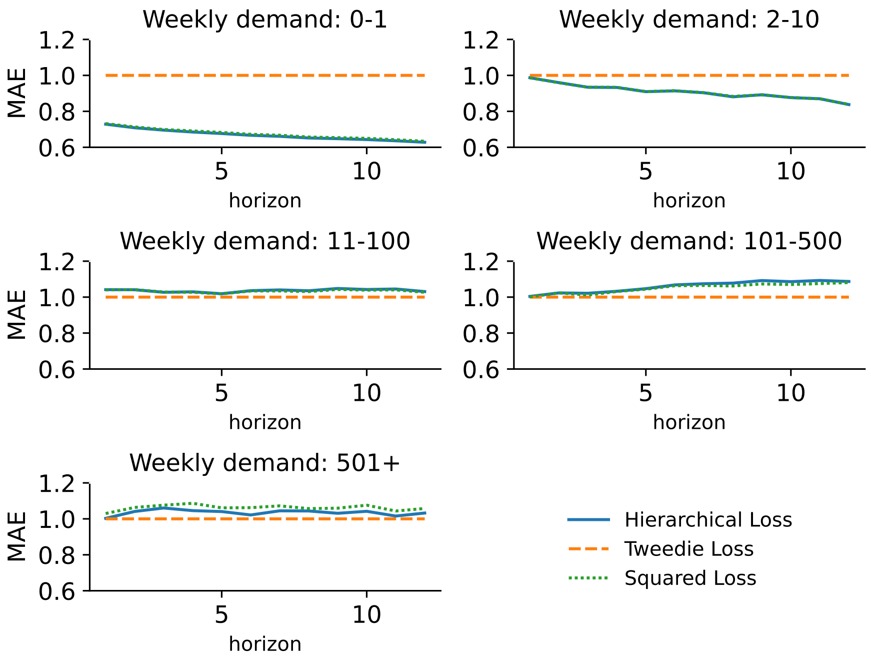


Figure 4.2: Forecasting results for the primary product forecasting model at our e-commerce partner bol. We show MAE by weekly demand bucket relative to the Tweedie loss baseline for each forecasting horizon (week). The Hierarchical loss outperforms the Tweedie loss on smaller weekly demand buckets.

ciliation method by an order of magnitude. We empirically verified our sparse hierarchical loss in an offline test for bol, where we confirmed the results from our offline test on the public M5 dataset.

In addition to our main contributions, one of our main learnings has been that we could not find a benefit of having multiple models for separate aggregations in the hierarchy, as the bottom-up scenario we employed consistently outperformed other scenarios. Secondly, we did not find a benefit of training a model whilst adhering to both cross-sectional and temporal hierarchies jointly.

Limitations of our work in this chapter are that we did not consider the probabilistic forecasting setting, where reconciled forecasts are required across an entire forecast distribution.

For future work, we aim to extend our work to the setting of proba-

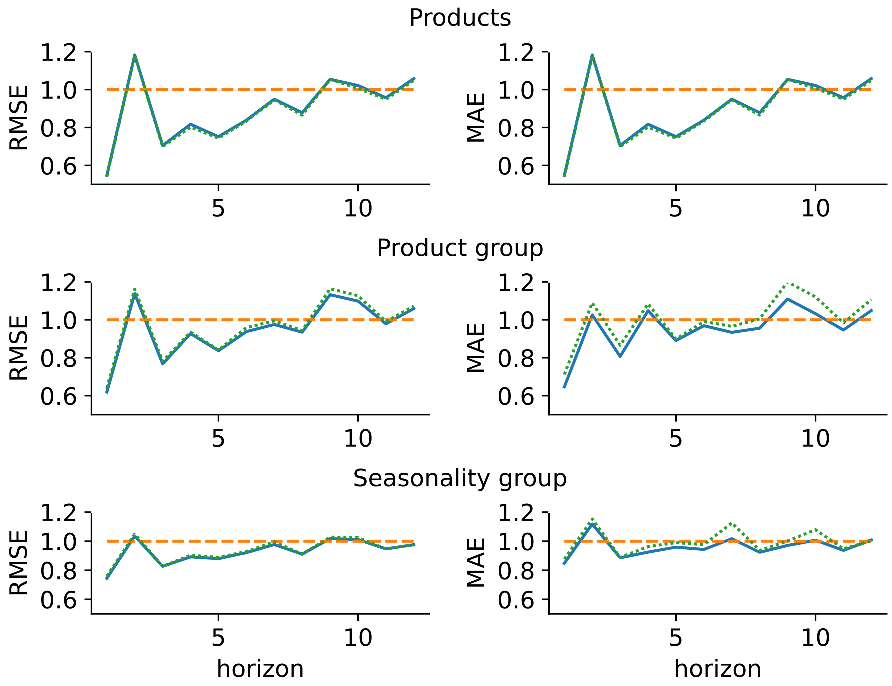


Figure 4.3: Forecasting results for the primary product forecasting model at our e-commerce partner bol. We show RMSE (left column of figures) and MAE (right column of figures) by aggregation level relative to the Tweedie loss baseline for each forecasting horizon (week). The Hierarchical loss commonly outperforms the Tweedie loss on every aggregation level.

bilistic forecasting by combining our sparse hierarchical loss with existing probabilistic forecasting frameworks from Chapter 3 or from other frameworks [48, 121]. In addition, we seek to further investigate solutions for efficiently combining cross-sectional and temporal hierarchies.

In our final research chapter, we turn to another forecasting problem often encountered at our industry partners: recommendations. We investigate state-of-the-art methods for session-based recommendation and surprisingly find that the most simple method gives the most accurate results.

Appendices

4.A Derivation of Gradient and Second-order derivative

We have the following loss function (Eq. (4.9)):

$$L = \sum \left[\frac{1}{2} \left(\left(\mathbf{Y} - \tilde{\mathbf{Y}} \right) \odot \left(\mathbf{Y} - \tilde{\mathbf{Y}} \right) \right) \oslash \left(d^{cs} d^{te} \right) \right] . \tag{4.19}$$

First, note that L is a scalar, as we sum over all $n^{cs} \times n^{te}$ elements of the matrix contained in the summation. We can expand this summation as follows:

$$L = \frac{\left(\frac{1}{2} \left(\mathbf{Y}_{0,0} - \tilde{\mathbf{Y}}_{0,0} \right)^2 \right)}{\left(d^{cs} d^{te} \right)_{0,0}} + \cdots + \frac{\left(\frac{1}{2} \left(\mathbf{Y}_{n^{cs},n^{te}} - \tilde{\mathbf{Y}}_{n^{cs},n^{te}} \right)^2 \right)}{\left(d^{cs} d^{te} \right)_{n^{cs},n^{te}}} , \tag{4.20}$$

with $\mathbf{Y}_{i,j}$, $\tilde{\mathbf{Y}}_{i,j}$ and $\left(d^{cs} d^{te} \right)_{i,j}$ denoting the i, j-th entry of the matrices $\mathbf{Y}$, $\tilde{\mathbf{Y}}$ and $\left(d^{cs} d^{te} \right)$, respectively. The derivative of L with respect to the first element $\tilde{\mathbf{Y}}_{0,0}$ is the standard squared error loss gradient divided by $\left(d^{cs} d^{te} \right)_{1,1}$, as all other terms in the summation cancel out:

$$\frac{\partial L}{\partial \tilde{\mathbf{Y}}_{0,0}} = \frac{\left(\tilde{\mathbf{Y}}_{0,0} - \mathbf{Y}_{0,0} \right)}{\left(d^{cs} d^{te} \right)_{0,0}} . \tag{4.21}$$

The second derivative of L with respect to the first element $\tilde{\mathbf{Y}}_{0,0}$ then reads:

$$\frac{\partial^2 L}{\partial \left(\tilde{\mathbf{Y}}_{0,0} \right)^2} = \frac{1}{\left(d^{cs} d^{te} \right)_{0,0}} . \tag{4.22}$$

We can then straightforwardly generalize the first and second derivative to the matrix case, such that the first and second derivative of L with respect to the matrix $\tilde{\mathbf{Y}}$ read:

$$\frac{\partial L}{\partial \tilde{\mathbf{Y}}} = \left(\left(\tilde{\mathbf{Y}} - \mathbf{Y} \right) \oslash \left(d^{cs} d^{te} \right) \right) , \tag{4.23}$$

$$\frac{\partial^2 L}{\partial\left(\tilde{\mathbf{Y}}\right)^2} = \left(\mathbf{1} \oslash \left(d^{cs} d^{te}\right)\right) \ , \tag{4.24}$$

with $\mathbf{1}$ denoting a matrix of ones of the same size as $\tilde{\mathbf{Y}}$. As L is a scalar and $\tilde{\mathbf{Y}} \in \mathbb{R}^{n^{cs} \times n^{te}}$, the derivative $\frac{\partial L}{\partial \tilde{\mathbf{Y}}}$ is a matrix of the same size as $\tilde{\mathbf{Y}}$. Similarly, $\frac{\partial^2 L}{\partial\left(\tilde{\mathbf{Y}}\right)^2}$ is a matrix of the same size as $\tilde{\mathbf{Y}}$, as we only consider the second-order derivatives with respect to the same component in $\tilde{\mathbf{Y}}$. To get to the gradient of L with respect to the bottom-level forecasts $\hat{\mathbf{Y}}^{n_b}$, we need to collect every gradient component in $\frac{\partial L}{\partial \tilde{\mathbf{Y}}}$ that contains a bottom-level forecast element. We can achieve this by multiplying the derivatives by our summing matrices S^{cs} and S^{te}:

$$\underbrace{\frac{\partial L}{\partial \hat{\mathbf{Y}}^{n_b}}}_{\mathbb{R}^{n_b^{cs} \times n_b^{te}}} = \underbrace{\left(S^{cs}\right)^{\intercal}}_{\mathbb{R}^{n_b^{cs} \times n^{cs}}} \underbrace{\left(\frac{\partial L}{\partial \tilde{\mathbf{Y}}}\right)}_{\mathbb{R}^{n^{cs} \times n^{te}}} \underbrace{S^{te}}_{\mathbb{R}^{n^{te} \times n_b^{te}}} \ , \tag{4.25}$$

$$\underbrace{\frac{\partial^2 L}{\partial\left(\hat{\mathbf{Y}}^{n_b}\right)^2}}_{\mathbb{R}^{n_b^{cs} \times n_b^{te}}} = \underbrace{\left(S^{cs}\right)^{\intercal}}_{\mathbb{R}^{n_b^{cs} \times n^{cs}}} \underbrace{\left(\frac{\partial^2 L}{\partial\left(\tilde{\mathbf{Y}}\right)^2}\right)}_{\mathbb{R}^{n^{cs} \times n^{te}}} \underbrace{S^{te}}_{\mathbb{R}^{n^{te} \times n_b^{te}}} \ . \tag{4.26}$$

4.B Derivation of Gradient of Toy Example

We have the following hierarchical forecasting problem:

$$\tilde{\mathbf{Y}} = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}}_{S^{cs}} \underbrace{\begin{bmatrix} \hat{\mathbf{y}}_{0,0} & \hat{\mathbf{y}}_{0,1} \\ \hat{\mathbf{y}}_{1,0} & \hat{\mathbf{y}}_{1,1} \end{bmatrix}}_{\hat{\mathbf{Y}}^{n_b}} \underbrace{\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}}_{(S^{te})^{\intercal}} ,$$

which results in the following gradient:

$$\begin{aligned} \begin{bmatrix} \frac{\partial L}{\partial \hat{\mathbf{y}}_{0,0}} & \frac{\partial L}{\partial \hat{\mathbf{y}}_{0,1}} \\ \frac{\partial L}{\partial \hat{\mathbf{y}}_{1,1}} & \frac{\partial L}{\partial \hat{\mathbf{y}}_{1,1}} \end{bmatrix} &= \left(S^{cs}\right)^{\intercal} \left[\left(S^{cs}\left(\hat{\mathbf{Y}}^{n_b} - \mathbf{Y}^{n_b}\right)\left(S^{te}\right)^{\intercal}\right) \oslash \left(d^{cs} d^{te}\right)\right] S^{te} \ , \\ &= \left(S^{cs}\right)^{\intercal} \left[\left(S^{cs}\left(\hat{\mathbf{Y}}^{n_b} - \mathbf{Y}^{n_b}\right)\left(S^{te}\right)^{\intercal}\right) \right. \\ &\qquad \left. \oslash \left(\left(l^{cs} S^{cs} \mathbf{1}^{cs}\right)\left(l^{te} S^{te} \mathbf{1}^{te}\right)^{\intercal}\right)\right] S^{te} \ , \end{aligned}$$

$$
\begin{aligned}
&= \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \left[\left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} \hat{\mathbf{y}}_{0,0} & \hat{\mathbf{y}}_{0,1} \\ \hat{\mathbf{y}}_{1,0} & \hat{\mathbf{y}}_{1,1} \end{bmatrix} - \begin{bmatrix} \mathbf{y}_{0,0} & \mathbf{y}_{0,1} \\ \mathbf{y}_{1,0} & \mathbf{y}_{1,1} \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \right) \right. \\
&\qquad \left. \oslash \left(\left(2 \begin{bmatrix} 2 & 1 & 1 \end{bmatrix}^{\mathsf{T}} \right) \left(2 \begin{bmatrix} 2 & 1 & 1 \end{bmatrix} \right) \right) \right] \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} , \\
&= \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \left[\left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \overbrace{\hat{\mathbf{y}}_{0,0} - \mathbf{y}_{0,0}}^{e_{0,0}} & \overbrace{\hat{\mathbf{y}}_{0,1} - \mathbf{y}_{0,1}}^{e_{0,1}} \\ \underbrace{\hat{\mathbf{y}}_{1,0} - \mathbf{y}_{1,0}}_{e_{1,0}} & \underbrace{\hat{\mathbf{y}}_{1,1} - \mathbf{y}_{1,1}}_{e_{1,1}} \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \right) \right. \\
&\qquad \left. \oslash \begin{bmatrix} 16 & 8 & 8 \\ 8 & 4 & 4 \\ 8 & 4 & 4 \end{bmatrix} \right] \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} , \\
&= \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{16}(e_{0,0} + e_{1,0} + e_{0,1} + e_{1,1}) & \frac{1}{8}(e_{0,0} + e_{1,0}) & \frac{1}{8}(e_{0,1} + e_{1,1}) \\ \frac{1}{8}(e_{0,0} + e_{0,1}) & \frac{1}{4}e_{0,0} & \frac{1}{4}e_{0,1} \\ \frac{1}{8}(e_{1,0} + e_{1,1}) & \frac{1}{4}e_{1,0} & \frac{1}{4}e_{1,1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} , \\
&= \left[\begin{matrix} \frac{1}{16}(e_{0,0} + e_{1,0} + e_{0,1} + e_{1,1}) + \frac{1}{8}(e_{0,0} + e_{1,0}) + \frac{1}{8}(e_{0,0} + e_{0,1}) + \frac{1}{4}e_{0,0} \\ \frac{1}{16}(e_{0,0} + e_{1,0} + e_{0,1} + e_{1,1}) + \frac{1}{8}(e_{0,0} + e_{1,0}) + \frac{1}{8}(e_{1,0} + e_{1,1}) + \frac{1}{4}e_{1,0} \end{matrix} \right. \\
&\qquad \left. \begin{matrix} \frac{1}{16}(e_{0,0} + e_{1,0} + e_{0,1} + e_{1,1}) + \frac{1}{8}(e_{0,1} + e_{1,1}) + \frac{1}{8}(e_{0,0} + e_{0,1}) + \frac{1}{4}e_{0,1} \\ \frac{1}{16}(e_{0,0} + e_{1,0} + e_{0,1} + e_{1,1}) + \frac{1}{8}(e_{0,1} + e_{1,1}) + \frac{1}{8}(e_{1,0} + e_{1,1}) + \frac{1}{4}e_{1,1} \end{matrix} \right] , \\
&= \begin{bmatrix} \frac{9}{16}e_{0,0} + \frac{3}{16}e_{1,0} + \frac{3}{16}e_{0,1} + \frac{1}{16}e_{1,1} & \frac{9}{16}e_{0,1} + \frac{3}{16}e_{0,1} + \frac{3}{16}e_{1,1} + \frac{1}{16}e_{1,0} \\ \frac{9}{16}e_{1,0} + \frac{3}{16}e_{0,0} + \frac{3}{16}e_{1,1} + \frac{1}{16}e_{0,1} & \frac{9}{16}e_{1,1} + \frac{3}{16}e_{1,0} + \frac{3}{16}e_{0,1} + \frac{1}{16}e_{0,0} \end{bmatrix} .
\end{aligned}
$$

4.C M5 Dataset

For each of the scenarios of our experiments in Section 4.5, we construct a set of features for the LightGBM model as given in Table 4.C.1. To facilitate the most 'fair' comparison across methods, each model has the same features, and for the time series aggregations in the hierarchy we construct the features taken over the aggregation. We refer the reader to [87] for a detailed description of the dataset.

Table 4.C.1: Features used for the M5 dataset in our experiments.

Feature	Description
`Aggregation`	Aggregation level in the hierarchy
`Value`	Identifier of time series of this aggregation
`sales_lag1-7`	Lagged sales (target) (7 features)
`sales_lag28`	Sales 28 days ago
`sales_lag56`	Sales 56 days ago
`sales_lag364`	Sales last year
`sales_lag1_mavg7`	Moving average of sales last 7 days
`sales_lag1_mavg28`	Moving average of sales last 28 days
`sales_lag1_mavg56`	Moving average of sales last 56 days
`dayofweek`	Day of the week
`dayofmonth`	Day of the month
`weekofyear`	Week of year
`monthofyear`	Month of year
`sell_price_avg`	Sell price (average if aggregation)
`sell_price_change`	Day-to-day change in sell price
`weeks_on_sale_avg`	Weeks on sale
`snap_CA`	State indicator for California
`snap_TX`	State indicator for Texas
`snap_WI`	State indicator for Wyoming
`event_type_1_enc`	Encoded events
`event_type_2_enc`	Encoded events

4.D M5 Model Training & Optimization

We optimize the hyperparameters of our LightGBM models using Optuna [7], using the settings found in Table 4.D.1. The validation is performed on a rolling-forward basis for 3 validation sets, where we use three years of data to predict the next 28 days ahead. After the hyperparameter optimization procedure, we use the average number of iterations at which the lowest validation loss was achieved across the 3 validation sets as the number of estimators to use in our final model. The final model uses the last three years of data preceding the first day in the test set.

4.E Experiments

Table 4.D.1: Key hyperparameters used in our experiments. The parameters with a search range included are optimized in a hyperparameter search.

Parameter	Description	Default value	Search range
n_estimators	Number of trees in each model	2000	Lowest validation loss
n_trials	Number of optimization trials to run	100	
learning_rate	Learning rate	0.05	
n_validation_sets	Number of validation sets	3	
n_days_test	Number of days in validation and test sets	28	
max_levels_random	Max. number of levels when using a random hierarchy	2	
max_categories_per_random_level	Max. categories per level in the random hierarchy	1000	
hier_freq	Frequency of performing the randomized hierarchical aggregation	1	uniform(1, 10)
lambda_l1	L1-regularization	0	log_uniform(10^{-8}, 10^{1})
lambda_l2	L2-regularization	0	log_uniform(10^{-8}, 10^{1})
num_leaves	Max. number of leaves per tree	31	uniform(2^{3}, 2^{10})
feature_fraction	Fraction of features to use to build a tree	1.0	uniform(0.4, 1.0)
bagging_fraction	Fraction of training samples to use to build a tree	1.0	uniform(0.4, 1.0)
bagging_freq	Frequency at which to create a new bagging batch	1.0	uniform(1, 7)
min_child_samples	Minimum number of samples per leaf	20	log_uniform(5, 5000)

Table 4.E.1: Forecasting results for all stores on the M5 dataset. We report mean RMSE scores across 10 random seeds with standard deviation in brackets. Lower is better.

						Store			Product		State				
Scen./Obj.	Metric	Reconciliation	Product	Department	Category	Department	Category	Total	Store	State	Department	Category	Total	Total	All series
Bottom-up															
SL	SL	None	2.11 (0.00)	580 (3.95)	1096 (13.71)	110 (0.60)	207 (1.21)	416 (2.07)	8.8 (0.03)	4.26 (0.01)	263 (1.30)	510 (3.81)	997 (8.44)	2086 (38.53)	22.4 (0.16)
SL	HL	None	2.11 (0.00)	564 (5.58)	1039 (14.64)	107 (0.74)	200 (1.69)	408 (2.37)	8.76 (0.06)	4.25 (0.01)	256 (1.30)	490 (3.93)	983 (5.93)	2079 (33.46)	21.8 (0.15)
HL	HL	None	2.11 (0.00)	511 (8.93)	882 (20.54)	102 (0.90)	185 (1.48)	388 (4.21)	8.5 (0.02)	4.21 (0.01)	233 (2.42)	427 (4.78)	878 (17.17)	1819 (78.42)	19.5 (0.38)
HL	SL	None	2.11 (0.00)	511 (9.47)	886 (24.67)	103 (1.08)	187 (2.12)	390 (3.80)	8.46 (0.04)	4.19 (0.01)	234 (3.15)	430 (7.18)	881 (15.27)	1813 (64.6)	19.6 (0.36)
TL	HL	None	2.11 (0.00)	549 (9.21)	1055 (22.68)	109 (0.58)	207 (1.27)	415 (2.58)	8.73 (0.03)	4.25 (0.01)	256 (2.09)	502 (5.04)	978 (13.8)	2007 (63.57)	21.8 (0.34)
TL	SL	None	2.11 (0.00)	544 (7.66)	1017 (25.23)	110 (0.40)	208 (1.20)	416 (2.41)	8.68 (0.04)	4.25 (0.01)	256 (1.55)	497 (5.23)	972 (7.67)	1943 (41.21)	21.5 (0.25)
TL	TL	None	2.46 (0.00)	1579 (9.71)	3075 (20.5)	193 (0.87)	379 (1.84)	717 (5.17)	13.41 (0.05)	5.68 (0.02)	566 (3.11)	1111 (6.42)	2074 (18.68)	5646 (60.40)	53.3 (0.39)
Sep. agg.															
SL	SL	Base	2.11 (0.00)	835 (23.47)	1417 (69.78)	130 (1.86)	236 (5.25)	474 (8.71)	8.88 (0.02)	4.23 (0.01)	322 (11.48)	682 (19.02)	1269 (47.11)	3339 (86.79)	30.1 (0.21)
SL	SL	OLS	2.11 (0.00)	804 (14.22)	1548 (22.24)	120 (1.49)	220 (1.95)	443 (2.37)	8.77 (0.02)	4.25 (0.01)	315 (4.15)	605 (7.98)	1222 (9.69)	3125 (24.25)	29.1 (0.22)
SL	SL	WLS-struct	2.10 (0.00)	731 (8.55)	1505 (16.09)	113 (0.58)	217 (1.13)	426 (2.17)	8.71 (0.02)	4.23 (0.01)	291 (2.38)	591 (4.82)	1153 (7.53)	2909 (23.52)	27.6 (0.22)
SL	SL	WLS-var	2.11 (0.00)	647 (7.23)	1345 (18.95)	109 (0.51)	211 (1.43)	411 (2.06)	8.72 (0.03)	4.24 (0.01)	272 (2.06)	555 (5.58)	1068 (8.77)	2541 (31.06)	25.1 (0.25)
SL	SL	MinT-shrink	2.11 (0.00)	667 (18.15)	1397 (44.32)	106 (1.15)	206 (3.14)	405 (5.45)	8.76 (0.02)	4.24 (0.01)	272 (4.92)	558 (12.59)	1091 (22.52)	2706 (80.28)	25.8 (0.63)
SL	SL	ERM	2.58 (0.01)	728 (48.01)	1411 (118.36)	117 (3.66)	213 (8.73)	443 (20.5)	10.30 (0.10)	5.21 (0.04)	307 (14.61)	583 (33.15)	1215 (75.3)	3101 (258.29)	28.1 (1.83)
Global															
SL	SL	Base	2.16 (0.00)	771 (54.94)	1587 (256.06)	120 (2.11)	227 (8.81)	458 (18.45)	9.09 (0.05)	4.4 (0.01)	329 (18.51)	650 (31.09)	1807 (1203.62)	3273 (338.71)	32.8 (6.60)
SL	SL	OLS	2.13 (0.01)	764 (58.91)	1524 (181.38)	117 (10.32)	227 (27.24)	484 (109.14)	8.99 (0.05)	4.34 (0.03)	316 (42.99)	637 (107.63)	1371 (417.84)	3114 (417.38)	29.9 (2.52)
SL	SL	WLS-struct	2.13 (0.00)	800 (45.37)	1686 (103.86)	118 (2.52)	235 (5.96)	463 (11.90)	9.09 (0.03)	4.35 (0.01)	314 (10.72)	651 (24.97)	1264 (55.94)	3231 (279.96)	30.4 (1.68)
SL	SL	WLS-var	2.14 (0.01)	876 (43.62)	1867 (106.77)	130 (2.31)	263 (7.10)	508 (18.51)	9.10 (0.05)	4.35 (0.01)	345 (11.71)	727 (30.30)	1374 (80.66)	3463 (283.05)	33.2 (1.84)
SL	SL	MinT-shrink	2.18 (0.02)	730 (50.73)	1545 (155.65)	115 (2.42)	228 (5.97)	478 (19.61)	9.29 (0.16)	4.46 (0.06)	293 (12.06)	599 (33.27)	1235 (89.41)	3211 (432.02)	29.1 (2.33)
SL	SL	ERM	2.56 (0.14)	923 (113.39)	1857 (271.23)	138 (16.52)	265 (34.54)	558 (105.87)	10.52 (0.62)	5.23 (0.34)	382 (56.45)	762 (118.54)	1607 (371.34)	3752 (669.36)	35.7 (3.74)

Table 4.E.2: Forecasting results for all stores on the M5 dataset. We report mean MAE scores across 10 random seeds with standard deviation in brackets. Lower is better.

						Store			Product		State				
Scen./Obj.	Metric	Reconciliation	Product	Department	Category	Department	Category	Total	Store	State	Department	Category	Total	Total	All series
`Bottom-up`															
SL	SL	None	1.07 (0.00)	420 (5.51)	772 (12.94)	70 (0.43)	131 (0.84)	312 (2.27)	4.41 (0.01)	2.2 (0.00)	181 (1.45)	343 (2.57)	812 (7.46)	1628 (24.81)	2.2 (0.01)
SL	HL	None	1.07 (0.00)	410 (5.82)	743 (12.96)	68 (0.37)	128 (0.75)	309 (1.79)	4.39 (0.01)	2.19 (0.00)	177 (1.29)	332 (2.25)	808 (5.34)	1653 (40.22)	2.2 (0.01)
HL	HL	None	1.06 (0.00)	342 (7.54)	598 (18.24)	63 (0.58)	117 (1.36)	293 (4.26)	4.33 (0.01)	2.18 (0.00)	154 (2.01)	283 (4.61)	724 (14.31)	1427 (81.18)	2.1 (0.01)
HL	SL	None	1.06 (0.00)	340 (6.2)	597 (18.44)	63 (0.56)	117 (1.19)	294 (3.06)	4.33 (0.01)	2.18 (0.00)	154 (1.91)	284 (4.49)	725 (13.35)	1417 (57.53)	2.1 (0.01)
TL	HL	None	1.05 (0.00)	350 (4.5)	634 (10.36)	65 (0.17)	122 (0.49)	299 (1.69)	4.27 (0.01)	2.15 (0.00)	160 (0.69)	305 (1.95)	747 (9.07)	1425 (50.6)	2.1 (0.00)
TL	SL	None	1.06 (0.00)	359 (4.18)	647 (10.08)	66 (0.25)	124 (0.52)	305 (1.54)	4.3 (0.01)	2.17 (0.00)	164 (1.27)	311 (2.84)	778 (4.19)	1457 (28.24)	2.1 (0.00)
TL	TL	None	1.09 (0.00)	866 (4.9)	1843 (11.88)	102 (0.44)	210 (1.1)	515 (4.95)	5.03 (0.01)	2.35 (0.00)	306 (1.48)	645 (3.75)	1601 (17.55)	4647 (59.66)	2.8 (0.01)
`Sep. agg.`															
SL	SL	Base	1.07 (0.00)	467 (13.86)	819 (36.81)	71 (0.62)	144 (2.96)	336 (9.36)	4.23 (0.01)	2.13 (0.00)	187 (4.24)	393 (11.11)	971 (50.11)	2523 (105.3)	2.2 (0.00)
SL	SL	OLS	1.03 (0.00)	455 (8.08)	895 (10.88)	70 (0.73)	129 (0.87)	311 (1.76)	4.24 (0.01)	2.13 (0.00)	181 (1.81)	345 (2.84)	910 (9.07)	2430 (45.28)	2.2 (0.01)
SL	SL	WLS-struct	1.03 (0.00)	417 (2.56)	876 (8.99)	64 (0.28)	124 (0.66)	298 (1.64)	4.22 (0.00)	2.12 (0.00)	168 (1)	334 (2.27)	842 (6.11)	2315 (22.33)	2.1 (0.00)
SL	SL	WLS-var	1.04 (0.00)	383 (3.33)	771 (10.62)	63 (0.25)	120 (0.66)	286 (1.48)	4.24 (0.00)	2.13 (0.00)	162 (1)	318 (2.75)	770 (6.87)	1887 (32.1)	2.1 (0.00)
SL	SL	MinT-shrink	1.04 (0.00)	393 (10.24)	811 (27.92)	62 (0.56)	118 (1.64)	283 (4.33)	4.23 (0.01)	2.13 (0.00)	162 (2.44)	316 (7.33)	796 (18.35)	2116 (81.24)	2.1 (0.01)
SL	SL	ERM	1.26 (0.00)	490 (28.29)	935 (91.26)	76 (1.99)	139 (5.8)	335 (17.11)	5.26 (0.03)	2.67 (0.01)	201 (7.9)	383 (22.22)	966 (63.05)	2559 (296.35)	2.6 (0.04)
`Global`															
SL	SL	Base	1.12 (0.01)	436 (22.31)	900 (156.94)	70 (1.01)	133 (3.84)	326 (15.88)	4.37 (0.01)	2.18 (0.01)	188 (8.04)	373 (13.48)	1314 (611.36)	2569 (320.87)	2.3 (0.04)
SL	SL	OLS	1.05 (0.01)	456 (32.4)	906 (97.58)	73 (11.48)	139 (28.62)	352 (93.72)	4.38 (0.03)	2.19 (0.04)	197 (41.58)	388 (103.17)	1033 (336.46)	2441 (389.89)	2.3 (0.13)
SL	SL	WLS-struct	1.05 (0.00)	456 (17.42)	970 (53.68)	68 (0.83)	133 (2.1)	323 (7.31)	4.4 (0.01)	2.18 (0.01)	181 (3.3)	368 (10.47)	937 (40.46)	2552 (258.94)	2.2 (0.02)
SL	SL	WLS-var	1.06 (0.00)	522 (18.96)	1082 (59.2)	76 (0.8)	148 (3.13)	353 (13.72)	4.43 (0.02)	2.19 (0.01)	206 (4.75)	415 (14.39)	1017 (62.93)	2744 (260.65)	2.3 (0.02)
SL	SL	MinT-shrink	1.06 (0.01)	474 (24.59)	960 (92.69)	74 (1.56)	141 (3.13)	358 (11.28)	4.49 (0.06)	2.22 (0.02)	190 (7)	367 (22.7)	935 (87.38)	2605 (424.61)	2.3 (0.03)
SL	SL	ERM	1.25 (0.06)	625 (60.87)	1252 (148.15)	90 (9.25)	175 (24.13)	426 (85.57)	5.26 (0.29)	2.65 (0.14)	252 (34.4)	512 (89.23)	1285 (310.6)	3190 (636.58)	2.8 (0.19)

5 Serenade

In this chapter, we turn to another forecasting problem often encountered at our industry partners: recommendations. Session-based recommendation predicts the next item with which a user will interact, given a sequence of her past interactions with other items. This machine learning problem targets a core scenario in e-commerce platforms, which aim to recommend interesting items to buy to users browsing the site. Session-based recommenders are difficult to scale due to their exponentially large input space of potential sessions. This impedes offline precomputation of the recommendations, and implies the necessity to maintain state during the online computation of next-item recommendations. This motivates our final research question:

Research Question 4: *How can we efficiently generate session-based recommendations at the scale of bol?*

We propose Vector-Multiplication-Indexed-Session kNN (VMIS-kNN), an adaptation of a state-of-the-art nearest neighbor approach to session-based recommendation, which leverages a prebuilt index to compute next-item recommendations with low latency in scenarios with hundreds of millions of clicks to search through. Based on this approach, we design and implement the scalable session-based recommender system *Serenade*, which is in production usage at bol.
We evaluate the predictive performance of VMIS-kNN, and show that Serenade can answer a thousand recommendation requests per second

This chapter was published as: B. Kersbergen, O. Sprangers, and S. Schelter. Serenade - Low-Latency Session-Based Recommendation in e-Commerce at Scale. In *Proceedings of the 2022 International Conference on Management of Data*, pages 150–159, Philadelphia, USA, June 2022. ACM. ISBN 978-1-4503-9249-5. doi: 10.1145/3514221.3517901.

with a 90th percentile latency of less than seven milliseconds in scenarios with millions of items to recommend. Furthermore, we present results from a three week long online A/B test with up to 600 requests per second for 6.5 million distinct items on more than 45 million user sessions from our e-commerce platform. To the best of our knowledge, we provide the first empirical evidence that the superior predictive performance of nearest neighbor approaches to session-based recommendation in offline evaluations translates to superior performance in a real world e-commerce setting.

5.1 Introduction

Session-based recommendation targets a core scenario in e-commerce and online browsing. Given a sequence of interactions of a visitor with a selection of items, we want to recommend to the user the next item(s) of interest to interact with [78, 83, 84, 102]. This machine learning problem is crucial for e-commerce platforms [70].

Challenges in scaling session-based recommendation Scaling session-based recommender systems is a difficult undertaking, because the input space (sequences of item interactions) for the recommender system is exponentially large (of size $|\mathbf{I}|^n$ for all possible sessions of length n from a set of items I), which renders it impractical to precompute recommendations offline and serve them from a data store. This is in stark contrast to classical collaborative-filtering based recommendations [73, 108], which are relatively static as they rely on long-term user behavior [110]. Instead, session-based recommenders have to maintain state in order to react to online changes in the evolving user sessions, and compute next item recommendations with low latency [10, 70] in real-time.

Recent research indicates that nearest neighbor methods provide state-of-the-art performance for session-based recommendation, and even outperform complex neural network-based approaches in offline evaluations [70, 84]. It is however unclear whether this superior offline performance also translates to increased user engagement in real-world recommender systems. Furthermore, it is unclear whether the academic nearest neighbor approaches scale to industrial use cases, where they have to efficiently search through hundreds of millions of historical clicks while adhering to strict service-level-agreements for response latency. This scalability challenge is further complicated by the fact that the applied session similarity functions do not constitute a metric space (e.g., due to lack of symmetry), which renders common approximate nearest neighbor search techniques inapplicable.

VMIS-kNN In order to tackle the scalability challenge, we present Vector-Multiplication-Indexed-Session kNN (VMIS-kNN) in Section 5.4, an adaption of the state-of-the-art session-based recommendation algorithm VS-kNN [84]. VMIS-kNN leverages a prebuilt index to compute next-item recommendations in milliseconds for scenarios with hundreds of millions of clicks in historical sessions to search through. Our approach can be viewed as the joint execution of a join between evolving and historical sessions on matching items and two aggregations to compute the similarities. During this joint execution, we minimise intermediate results, control the memory usage and prune the search space with early stopping. As a consequence, VMIS-kNN drastically outperforms VS-kNN in terms of latency and scalability (Section 5.6.2), while still providing the desired prediction quality advantages over neural network-based approaches (Section 5.6.1).

Serenade Finally, we present the design and implementation of our scalable session-based recommender system *Serenade*, which employs VMIS-kNN, and can serve a thousand recommendation requests per second with a 90th percentile latency of less than seven milliseconds in scenarios with millions of items to recommend. Our system runs in the Google Cloud-based infrastructure of bol, a large European e-commerce platform, and is in production usage. We discuss design decisions of Serenade, such as stateful recommendation servers, which colocate the evolving user sessions together with update and recommendation requests (Section 5.5.1). Additionally, we describe implementation and deployment details (Section 5.5.2), as well as insights into the remarkably low operational costs for our system (Section 5.7).

Offline and online evaluation We conduct an extensive evaluation to validate the predictive performance and low latency of VMIS-kNN in

Section 5.6.1. For the Serenade system, we present results from a load test with more than 1,000 requests per second, and the outcome of a three week long online A/B test of our system on the live e-commerce platform in Section 5.6.2. Our system is available under an open license.[1]

Contributions In summary, we contribute:

- We present VMIS-kNN, an index-based variant of a state-of-the-art nearest neighbor algorithm to session-based recommendation, which scales to use cases with hundreds of millions of clicks to search through (Section 5.4).
- We discuss design decisions and implementation details of our production recommender system Serenade, which applies stateful session-based recommendation with VMIS-kNN, and can handle more than 1,000 requests per second with a response latency of less than seven milliseconds in the 90th percentile (Section 5.5).
- To the best of our knowledge, we provide the first empirical evidence that the superior predictive performance of VMIS-kNN/VS-kNN from offline evaluations translates to superior performance in a real world e-commerce setting; we find Serenade to drastically increase a business-specific engagement metric by several percent, compared to our legacy system (Section 5.6.2).

5.2 Related Work

Research on recommender systems [26, 29, 39, 46, 49, 85, 98, 103, 116, 137, 140–142] is a growing field, with a close connection to industry use cases [127, 135, 136], as illustrated by the famous "Netflix Prize" competition [73]. Translating academic progress into deployable solutions has proven to be very difficult though [70], exemplified by the fact that the winning solution of the Netflix prize never went into production [9]. Nearest neighbor-based recommendations, which are the focus of our work, are a classical approach to recommendation mining [23, 77, 108, 110, 111], and are widely deployed in industry [31, 32, 35, 56, 94]. Despite their popularity, these approaches are

[1]`https://github.com/bolcom/serenade`

typically outperformed by matrix factorisation- and deep learning-based methods in offline evaluations on classical collaborative filtering problems [73].

However, recent research indicates that nearest neighbor-based approaches provide state-of-the-art performance and outperform neural networks in sequence-based recommendation tasks. An example for such a task is session-based recommendation, which is the focus of our work, where recent studies [63, 70, 84] indicate that nearest neighbor-based methods outperform previously proposed neural networks [52, 78, 83, 138]. Similar results have been obtained for the more general sequence-based recommendation task of next basket recommendation (where the set of items in a future shopping basket has to be predicted). Here, the nearest neighbor-based state-of-the-art approach TIFU-kNN [55] and simple popularity-based approaches [79] outperform neural networks as well.

5.3 Background

We introduce session-based recommendation and the Vector-Session-kNN method. Given an evolving session (a sequence of interactions with a set of items $\mathbf{I}$) at time t, the goal of session-based recommendation is to accurately predict the next item that the user will interact with at time $t+1$.

Vector-Session-kNN Vector-Session kNN (VS-kNN) [84] is a state-of-the-art nearest neighbor based approach to session-based recommendation, which outperforms current deep learning approaches for this task. In VS-kNN, we have a set of historical sessions $\mathbf{H} \in \{0,1\}^{|\mathbf{I}|}$ represented as binary vectors in item space, and an evolving user session $\mathbf{s}^{(t)} \in \{0,1\}^{|\mathbf{I}|}$ at time t, as well as a function $\omega(\mathbf{s})$ which replaces the non-zero entries of s with integers denoting the insertion order of the items in $\mathbf{s}^{(t)}$. Algorithm 5.1 describes how VS-kNN computes its recommendations for an evolving session $\mathbf{s}^{(t)}$. First a recency-based sample $\overline{\mathbf{H}}_s$ of size m is taken from all historical sessions $\mathbf{H}_s$ that share at least one item with the evolving session (Lines 5–6). Next, we compute the k closest sessions $\mathbf{N}_s$ from $\overline{\mathbf{H}}_s$ according to the similarity $\pi(\omega(\mathbf{s}^{(t)}))^\top \mathbf{h}$ (Line 7), which applies an element-wise decay function π to the entries denoting

Algorithm 5.1 Vector-Session-kNN.

```
1: function VS-KNN($\mathbf{s}^{(t)}, \mathbf{H}, \pi, \lambda, m, k$)
2:     Input: Evolving session $\mathbf{s}^{(t)}$, set of historical sessions $\mathbf{H}$, decay function $\pi$,
3:     match weight function $\lambda$, sample size $m$, number of neighbors $k$.
4:     Output: Scored list of recommended next items $\mathbf{d}$.
5:     $\mathbf{H}_s \leftarrow$ historical sessions that share at least one item with s
6:     $\overline{\mathbf{H}}_s \leftarrow$ recency-based sample of size $m$ from $\mathbf{H}_s$
7:     $\mathbf{N}_s \leftarrow k$ closest sessions $\mathbf{h} \in \overline{\mathbf{H}}_s$ according to similarity $\pi(\omega(\mathbf{s}^{(t)}))^\top \mathbf{h}$
8:     for each item $i$ occuring in the sessions $\mathbf{N}_s$ do
9:         $d_i \leftarrow \sum_{\mathbf{n} \in \mathbf{N}_s} \mathbb{1}_n(i) \cdot \frac{1}{|\mathbf{s}^{(t)}|} \cdot \lambda(\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{n})) \cdot \pi(\omega(\mathbf{s}^{(t)}))^\top \mathbf{n} \cdot (1 + \log \frac{|H|}{h_i})$
       return item scores $\mathbf{d}$
```

the insertion order in the evolving session. All items occurring in these neighboring sessions are finally scored (Lines 8–9) by summing their similarities (the previously computed decayed dot product) weighted by a non-linear function λ applied to the position $\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{n})$ of the most recent shared item between the evolving session $\mathbf{s}^{(t)}$ and the neighbor session $\mathbf{n}$. The session similarity contribution is additionally weighted by a factor of one over the session length, and by a factor of one plus the "inverse document frequency" $\log \frac{|H|}{h_i}$ of the item, where h_i denotes the number of historical sessions containing item i (a common technique from information retrieval to de-emphasise highly frequent items). Note that the indicator function $\mathbb{1}_n(i)$ is 1 if item i occurs in the historical session $\mathbf{n}$ and 0 otherwise.

Toy example We provide a toy example for the session similarity and match weighting computation executed by VS-kNN. Assume that we have an evolving session $\mathbf{s}^{(t)} = [0\ 1\ 1\ 0\ 1]$ representing interactions with the three items `[1,2,4]` and a historical session $\mathbf{h} = [0\ 0\ 1\ 0\ 1]$ representing interaction with the items `[2,4]`. The function ω gives us the chronological insertion order for the evolving session, e.g., $\omega(\mathbf{s}^{(t)}) = [0\ 1\ 2\ 0\ 3]$ of the items in $\mathbf{s}^{(t)}$, starting from the first item (item 1 with insertion time 1) to the most recent item (item 4 with insertion time

3). The insertion order is used to weight matches between the items of the evolving session and the historical session, and the weights are determined by the decay function π, which is a hyperparameter of VS-kNN. A common choice for π is to divide the insertion time by the session length, e.g., $\pi(\omega(\mathbf{s}^{(t)})_i) = \omega(\mathbf{s}^{(t)})_i \,/\, ||\mathbf{s}^{(t)}||_1$. The similarity is finally determined by computing the decayed dot product $\pi(\omega(\mathbf{s}^{(t)}))^\top \mathbf{h}$ between the evolving session $\mathbf{s}^{(t)}$ and historical session $\mathbf{h}$ as the sum of the decayed weights for the intersection of the sessions (the shared items), e.g., $\pi(\omega(\mathbf{s}^{(t)}))^\top \mathbf{h} = [0\ \frac{1}{3}\ \frac{2}{3}\ 0\ \frac{3}{3}]^\top\, [0\ 0\ 1\ 0\ 1] = \frac{2}{3} + \frac{3}{3} = \frac{5}{3}$.

After finishing the session similarity computation, VS-kNN computes item scores from the similarities (Lines 8–9). The score for an item is the weighted sum of similarities with $\mathbf{s}^{(t)}$ from the k closest historical sessions $\mathbf{n} \in \mathbf{N}_s$ in which the item occurs. The weights for this sum are computed by the matching function λ, which is applied to the insertion time $\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{n})$ of the most recent shared item between $\mathbf{s}^{(t)}$ and $\mathbf{n}$. The default choice for λ in VS-kNN is $1 - (0.1 \cdot (\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{n}))$ for insertion times less than 10 and zero otherwise. For our toy example, the contribution of the matching function for $\mathbf{h}$ looks as follows: $\lambda(\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{h})) = \lambda(\max(\,[0\ 1\ 2\ 0\ 3]\, \odot\, [0\ 0\ 1\ 0\ 1]\,)) = \lambda(\max(\,[0\ 0\ 2\ 0\ 3]\,)) = \lambda(3) = 0.7$.

5.4 Vector-Multiplication-Indexed-Session kNN (VMIS-kNN)

In the following, we present our scalable, index-based adaption of VS-kNN: Vector-Multiplication-Indexed-Session kNN (VMIS-kNN).

VMIS-kNN operates on an index structure $(\mathbf{M}, \mathbf{t})$, which we build from a large dataset of historical sessions. We create a hash index $\mathbf{M}$ from an item i to an array $\mathbf{m}_i$ of the m most recent historical sessions in which the item occurs. Note that m is a hyperparameter of VMIS-kNN, which denotes the size of the recency-based sample from which session similarity candidates are taken. Each array $\mathbf{m}_i$ of session identifiers for an item i is stored in descending timestamp order of the sessions (i.e., the most recent historical session h that contained the item i is the first entry in the vector $\mathbf{m}_i$). The key benefit of this data structure is to allow us amortised constant-time access to the m most recent sessions containing an item.

Furthermore, we maintain an array $\mathbf{t}$ where an entry t_h denotes the integer timestamp for a historical session h. This again provides constant time random access during the online computation of the session similarity score across all the items in an evolving session, as we use consecutive integer identifiers for historical sessions. Algorithm 5.2 describes the individual steps and data structures that VMIS-kNN leverages for efficient session-based recommendation based on our index data structure.

Index-based session similarity At the heart of VMIS-kNN is the efficient computation of the neighbor sessions $\mathbf{N}_s$ for an evolving session $\mathbf{s}^{(t)}$ using our previously introduced index structure $(\mathbf{M}, \mathbf{t})$ in the function $\mathrm{neighbor\_sessions\_from\_index}$ in Line 8.

We first initialize a set of temporary hashmaps and heaps (Line 11) which serve as buffers for intermediate results during the computation. Next, VMIS-kNN starts the *item intersection loop*, which iterates over the items in an evolving session $\mathbf{s}^{(t)}$ in reverse order (Line 12). Our approach processes an evolving session $\mathbf{s}^{(t)}$ in inverse insertion order, such that the most recent (and therefore most important) items of an evolving session are visited first. We then add the item identifier i to the temporary hashset $\mathbf{d}$, such that duplicate items in the evolving session can be skipped (Lines 13–14). Next, we look up the item in our inverted index $\mathbf{M}$ to obtain the vector $\mathbf{m}_i$ containing up to m historical session identifiers (Line 15). We then compute the decay score π_i based on the item's position in the evolving session (Line 16).

Now, we start a loop over each historical session j in $\mathbf{m}_i$ (Line 17). If we have already encountered this historical session for a different item, we add the current decay score π_i to the session score r_j (Line 19). However, if the historical session is not yet part of our temporary similarity score hashmap $\mathbf{r}$, we first obtain the timestamp t_j of the historical session (Line 21). If our temporary similarity score hashmap $\mathbf{r}$ contains less than m items, we insert the session identifier j and session similarity score r_j as (key, value)-pair into $\mathbf{r}$, and we insert the session identifier j and session timestamp t_j as (key, value)-pair into a min-heap $\mathbf{b}_t$ (Lines 22–25). If our temporary similarity score hashmap $\mathbf{r}$ already contains m sessions, we need to investigate whether to remove the oldest session. Therefore, we first retrieve the oldest session and corresponding timestamp from the heap $\mathbf{b}_t$ (Line 27).

If the current historical session j is more recent than the oldest session, we need to remove the oldest session from our temporary similarity score hashmap $\mathbf{r}$ and heap $\mathbf{b}_t$, and update both with the values from the current historical session j (Lines 28–32). Finally, we extract the top-k scored sessions from the max-heap $\mathbf{N}_s$ in the *top-k similarity loop* and return them (Line 34).

VMIS-kNN computes the final item scores by using the pre-computed session similarity r_n for a neighboring historical session $\mathbf{n}$. We however simplify the item scoring function from Line 9 of Algorithm 5.1 in two ways: (i) we remove the constant factor $1/|\mathbf{s}^{(t)}|$ applied to each similarity (which does not change the neighbor ranking), and (ii) we use a weight of $\log \frac{|H|}{h_i}$ instead of $(1 + \log \frac{|H|}{h_i})$ for the similarities, which gives us better results in offline evaluations on held-out data.

A particular advantage of VMIS-kNN is its support for early stopping, which allows us to skip certain historical sessions during the similarity computation: we can immediately break the session for-loop if our current historical session j is older than the eldest session l in our heap $\mathbf{b}_t$ as $\mathbf{m}_i$ is already sorted in descending timestamp order, and will not contain more recent sessions in later positions (Line 33).

Time complexity The time complexity of the online computation of our similarity score is dominated by the linear time required to execute the three for-loops (Lines 12, 17 and 34) and the logarithmic time required to modify the heaps $\mathbf{b}_t$ (Lines 25, 32) and $\mathbf{N}_s$ (Lines 35, 38, 39), yielding a theoretical time complexity of $O(|\mathbf{s}^{(t)}| \cdot m \cdot \log_2 m + m \cdot \log_2 k) = O(|\mathbf{s}^{(t)}| \cdot m \cdot \log_2 m)$ as $k \leq m$. Thus, the time complexity only depends on: (i) the number of items in the evolving session $\mathbf{s}^{(t)}$, which we cap at a maximum value, and (ii) the number m denoting how many recent historical sessions to consider. Hence, the time complexity of our implementation is (theoretically) independent of the number of historical sessions $|\mathbf{H}|$ and the number of unique items $|\mathbf{I}|$ in our dataset. As a micro-optimisation, we leverage octonary heaps [2] instead of binary heaps, which have more children per node, and therefore provide better performance for workloads like ours with frequent insertions.

Space complexity The space complexity of the index for VMIS-kNN is dominated by the storage cost $O(|\mathbf{I}| \cdot m)$ for the hashmap holding the inverted index $\mathbf{M}$, which maps unique item indices to the m most recent historical sessions containing the item.

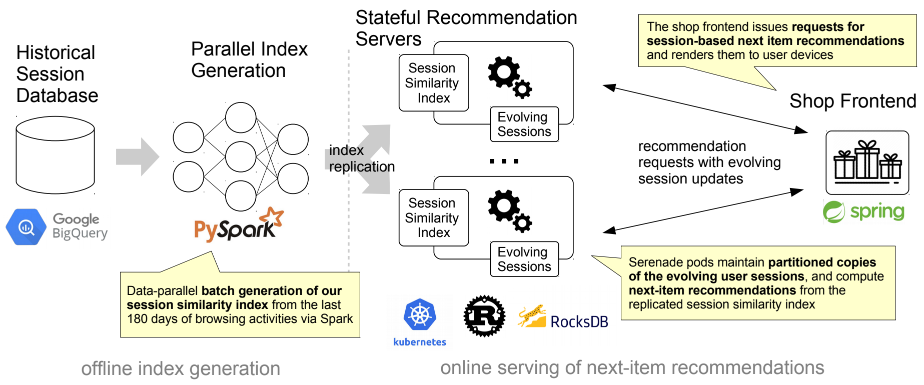

Historical Session Database
Google BigQuery
Parallel Index Generation
PySpark
Data-parallel batch generation of our session similarity index from the last 180 days of browsing activities via Spark
index replication
Stateful Recommendation Servers
Session Similarity Index
Evolving Sessions
Session Similarity Index
Evolving Sessions
kubernetes
RocksDB
The shop frontend issues requests for session-based next item recommendations and renders them to user devices
Shop Frontend
recommendation requests with evolving session updates
spring
Serenade pods maintain partitioned copies of the evolving user sessions, and compute next-item recommendations from the replicated session similarity index
offline index generation
online serving of next-item recommendations

Figure 5.1: High level architecture of the Serenade recommendation system. The offline component (left) generates a session similarity index [1] from several billion historical click events via a parallel Spark job in regular intervals. The online serving machines (right) maintain state about the evolving user sessions [2], and leverage the session similarity index to compute next item recommendations with VMIS-kNN in response to recommendation requests from the shopping frontend [3].

From a classical query processing perspective, VMIS-kNN conducts two aggregations (identifying the m most recent sessions with an item match, and computing their similarities) on the result of a join between the items of the evolving session $\mathrm{s}^{(t)}$ and the historical sessions $\mathbf{H}$. The efficiency of VMIS-kNN derives from the fact that we jointly execute the join and subsequent aggregations in Algorithm 5.2, while only maintaining intermediate results of a size proportional to the final outputs (instead of first materialising the potentially large complete join result before running the aggregations).

5.5 Serenade

We present the design and implementation of our scalable recommender system *Serenade*, which leverages VMIS-kNN (Section 5.4) and provides recommendations on the product detail pages of bol.

5.5.1 Design Considerations

At the core of the design of our production system are two questions: (i) how to maintain the session similarity index over time, and (ii) how to efficiently serve next-item recommendations with low latency?

Index maintenance We execute the index computation in an offline manner once per day with a data-parallel implementation of the relational operations required for the index generation. This batch job is easy to schedule and scale; note that Serenade will thus only see sessions for new items on the platform with a delay of one day. This "cold-start" issue is no problem in practice however, because our e-commerce platform has a separate, specialised system for presenting new and trending items to users.

Low latency serving of next-item recommendations The biggest challenge in our system is to serve session-based recommendations with a low latency for a catalog containing millions of items (our business constraint is to respond in 50 ms or less for at least 90% of all requests). As discussed in Section 5.1, we cannot precompute the recommendations due to the exponentially large input space of potential sessions, and we cannot apply approximate nearest neighor search techniques because our similarity function is not a metric. As a consequence, our recommendation servers have to be stateful, by maintaining copies of the evolving sessions, to be able to compute recommendations online on request. We decide to replicate our session index to all recommendation servers, and colocate the session storage with the update and recommendation requests, so that we only have to use machine-local reads and writes for maintaining sessions and computing recommendations. Note that similar techniques are often used to accelerate joins [36].

5.5.2 Implementation

The high-level architecture of Serenade (derived from our design decisions in Section 5.5.1) is illustrated in Figure 5.1. Serenade consists of two components: The offline component (shown in the left part of the figure) builds the session index from click data and is implemented as an Apache Spark pipeline. The online component (shown in the right part of the figure) computes and serves session-based recommendations with VMIS-kNN, and is implemented as a REST application. Note that Serenade builds upon existing Google Cloud infrastructure rented by bol.

Offline index generation The index generation [1] from historical click data is implemented as a parallel dataflow computation in Apache Spark using Spark MLLib pipeline steps [92] as abstraction, and executed in regular intervals (typically once per day) in Google Dataproc. It uses historical click data from the last 180 days of our platform (stored in Google BigQuery) as its input, which amounts to roughly 2.3 billion user-item interactions. The output of the Spark job is a compressed representation of our index, stored in the distributed filesystem in the Apache Avro format. The index data is later on ingested by Serenade's serving component, where it requires around 13 gigabytes of memory.

Online serving of next item recommendations The serving component of Serenade is responsible for computing next item recommendations with VMIS-kNN in response to session updates. We implement this serving component in Rust, as a web application based on the Actix [1] framework. The shopping frontend contacts our Serenade servers whenever a user generates new item interactions in their session (e.g., by visiting a product detail page). The Serenade servers update the state of the evolving user session [2], and respond to the shopping frontend with a list of 21 recommended next items for the user (the number of items required by the UI in the frontend) based on a VMIS-kNNprediction [3]. The VMIS-kNN predictions leverage the previously computed offline index. We additionally apply business rules to the recommendations to remove unavailable products and to filter for adult products.

Colocation of evolving sessions and session updates As discussed in Section 5.5.1, we need to colocate the evolving sessions with the recommendation requests and session updates to be able to compute up-to-date recommendations with low latency. We maintain the evolving sessions in a local key-value store (RocksDB [4]) directly on the serving machines, to avoid additional network reads and writes. For colocation, we have to partition both the evolving sessions and the recommendation requests (which also contain the session updates) over the serving machines, based on their session identifier. In order to guarantee that all the update/recommendation requests for a particular session are always handled by the same machine, we configure request routing via “sticky sessions” provided by Kubernetes’ session affinity functionality [3]. The communication with RocksDB turns out to be extremely fast; in a microbenchmark with 10 million operations for our workload, we found the 99th percentile of the read latency to be 5 microseconds, and the 99th percentile of the write latency to be 18 microseconds. This colocation approach provides a big latency improvement over network reads and writes to a distributed key-value store like BigTable, where the response latency for lookups is already 15ms on the 99.5 percentile in our experience.

Discussion Our colocation approach can be viewed as a trade-off between reducing the response latency and guaranteeing fault tolerance for the session data, as the session data could be temporarily lost in cases of machines failures or elastic scaling of the machine pool. However, this turns out to be no problem in practice for several reasons: (i) our service proved to be very stable, we encountered no issues in a long A/B test running for several weeks (details will be described in Section 5.6.2), where no elastic scaling was required, as a small set of cheap machines with a low number of cores could reliably handle hundreds of request per second, and (ii) the sessions are very short-lived anyways, we only leverage the most recent interactions for recommendations (which also have the highest impact on the session similarities), their loss would not have a drastic impact, as the recommender would quickly collect new interactions, and (iii) the sessions are additionally tracked by other parts of our e-commerce platform for analytics. It is not the task of the recommendation system to store them permanently, on the contrary, we configure RocksDB to remove the data for a session after 30 minutes of inactivity.

Deployment We deploy our recommendation servers via a Docker image managed by Kubernetes. The image is created by our continuous integration infrastructure, and we leverage a multi-stage build. In the first stage, we download all dependencies and compile our Rust application (which results in a large image with a size of several gigabytes); in the second stage, we reduce the size of this image by only retaining the compiled application and the runtime dependencies. The image for

Serenade is then pushed into a Docker repository. The application is deployed to a Google Kubernetes Engine cluster, alongside with load balancing pods (`istio sidecars` [5]) which provide us with the session affinity routing required for colocating the evolving user sessions and recommendation requests on our machines.

Depersonalisation We are required to provide non-personalised recommendations for users who do not give consent to leverage their session history for personalisation. This is comparatively easy to implement with VMIS-kNN: we create a non-personalised variant which only leverages the currently displayed item on the product detail page for recommendation. This depersonalisation can be applied in real-time (e.g., when a user revokes their consent to personalisation), as each request from the shop frontend includes a binary flag denoting the status of the user consent.

5.6 Experimental Evaluation

In the following, we first evaluate the prediction quality and index design of VMIS-kNN in Section 5.6.1, and subsequently evaluate the scalability and business performance of Serenade in offline experiments and an online A/B test (Section 5.6.2). We provide the code for our experiments on Github.[2]

Datasets We leverage a combination of public and proprietary click datasets from e-commerce for our offline experiments. We experiment with the publicly available [84] datasets *retailrocket* (an e-commerce dataset from the company "Retail Rocket") and *rsc15* (a dataset used in the 2015 ACM RecSys Challenge), which are commonly used in comparative studies on session-based recommendation [84]. In addition, we create the non-public datasets *ecom-1m*, *ecom-60m*, *ecom-90m* and *ecom-180m* by sampling data from our e-commerce platform with increasing numbers of clicks. The statistics of these datasets are shown in Table 5.1. Each dataset consists of tuples denoting the `session_id`, `item_id` and `timestamp` of a click event on the platform.

Our proprietary dataset *ecom-180m* is more than six times larger than the largest publicly available dataset *rsc15*. We additionally show

[2]`https://github.com/bolcom/serenade-experiments-sigmod`

Table 5.1: Public and proprietary datasets for evaluation.

	retailr	*rsc15*	*ecom-1m*	*ecom-60m*	*ecom-90m*	*ecom-180m*
clicks	86,635	31,708,461	1,152,438	67,017,367	89,883,761	189,317,506
sessions	23,318	7,981,581	214,490	10,679,757	13,799,762	28,824,487
items	21,276	37,483	110,988	1,760,602	2,263,670	3,305,412
days	10	181	30	29	91	91
public?	yes	yes	no	no	no	no
clicks per session						
p25	2	2	2	2	2	2
p50	2	3	4	4	4	4
p75	4	4	6	7	7	7
p99	19	19	28	36	38	39

statistics of the distribution of clicks per session in the form of its 25th, 50th, 75th and 99th percentile. We find that the majority of sessions on e-commerce platforms is very short (e.g., the median number of clicks per session is less than five) and that these statistics are very similar across all six datasets. In the tail, the sessions from our platform are about twice as long though compared to the public datasets (e.g., the 99th percentile is around 38 clicks in our data and 19 clicks in the public datasets).

5.6.1 VMIS-kNN

State-of-the-Art Prediction Quality

Before evaluating systems-related aspects, we run a sanity check experiment for the predictive performance of VMIS-kNN. We aim to confirm that VMIS-kNN also outperforms current neural-network based approaches in the task of session-based recommendation in e-commerce (as recently shown for VS-kNN [70, 84]).

Experimental setup We replicate the setup from [70, 84], and compare the predictive performance of VMIS-kNN against three recent neural network-based approaches to session-based recommendation (GRU4Rec [52], NARM [78] and STAMP [83]) on various clickstream datasets sampled from our e-commerce platform. We create five versions of the *ecom-1m* dataset by sampling a million clicks from certain months in the past as historical sessions, and measure the prediction quality of the top 20 recommended items for each session of the subsequent day.

We optimise the hyperparameters of each approach on samples of the training data, and report the average for each metric over all our evaluation datasets. We report the metric values averaged over all five versions of *ecom-1m*.

Results and discussion We first investigate the Mean Average Precision (MAP@20), Precision (Prec@20) and Recall (R@20), which denote to what extent an approach correctly predicts the next items in a session. VMIS-kNN outperforms the neural approaches in all of these metrics. Its MAP@20 is .0268 compared to .0251 for the best performing neural approach (GRU4Rec); the Prec@20 of VMIS-kNN is .0722 compared to .0680 for the best performing neural approach (NARM in this case); and VMIS-kNN's R@20 is .378 compared to .359 for the best performing neural approach GRU4Rec. We additionally look at the Mean Reciprocal Rank (MRR@20), which puts a stronger weight on the immediate next item in a session. Again, VMIS-kNN outperforms all neural-based approaches with an MRR@20 of .286 compared to .255 for the best performing neural method (GRU4Rec in this case).

In summary, we confirm that the findings from recent studies on the state-of-the-art performance of VS-kNN also hold for VMIS-kNN on our proprietary data. It is an open question, why neural networks do not outperform conceptually simpler methods in sequential recommendation. There is recent evidence that neural networks have difficulties capturing item frequency information [55], and that many researchers do not adequately compare their proposed neural methods against simple baselines [79, 84].

Sensitivity to Hyperparameter Choices

Next, we investigate the sensitivity of VMIS-kNN to its hyperparameters: the number of neighbors k and the number of most recent sessions per item m.

Experimental setup We run an exhaustive grid search over 55 combinations of the hyperparameters (the k most similar sessions out of the m most recent sessions) for our four large datasets *ecom-60m*, *ecom-90m*, *ecom-180m* and *rsc15*, where we use the last day as held-out test set.

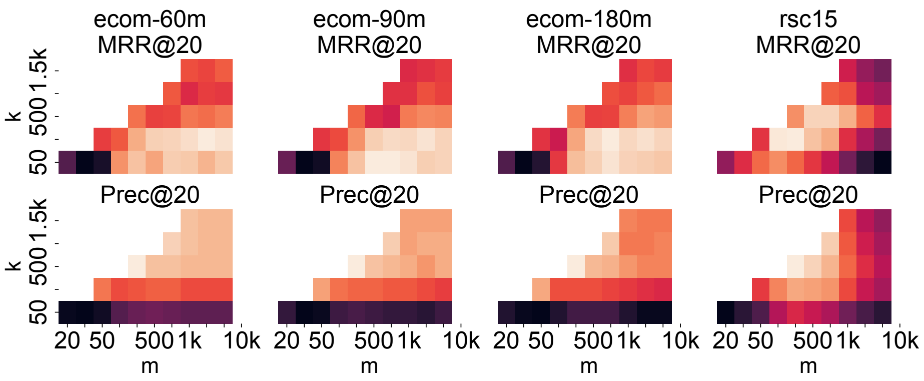


Figure 5.2: Sensitivity of MRR@20 and Prec@20 to the hyperparameters k (the number of neighbors) and m (the number of most recent sessions per item) in our proprietary datasets.

Results and discussion Figure 5.2 illustrates the results of the grid search for MRR@20 and Prec@20 on our datasets with a heatmap where lighter colors indicate better metric values. We observe a unimodal distribution of the resulting metric values for each dataset and metric. The results differ (i) based on dataset, e.g., all samples from our proprietary data show similar outcomes, while the distribution for *rsc15* is very different, and (ii) based on metric, e.g., hyperparameters that work well for MRR (which focuses on the position of the first correctly predicted relevant item) do not necessary provide the best performance for Precision (which considers all correctly predicted relevant items). Our results indicate that VMIS-kNN is easy to tune via offline grid search for a given dataset and target metric.

Index Design

Next, we run a microbenchmark comparing VMIS-kNN vs VS-kNN to validate the performance of our index-based similarity computation.

Experimental setup We experiment with our index and similarity computation (refered to as *VMIS-kNN*) from Section 5.4 and compare it against two baseline implementations: (i) *VS-kNN* – a baseline implementation that mimics VS-kNN's similarity computation by holding the historical data in hashmaps, and first identifying the m most recent sessions with at least one shared item before computing the similarities,

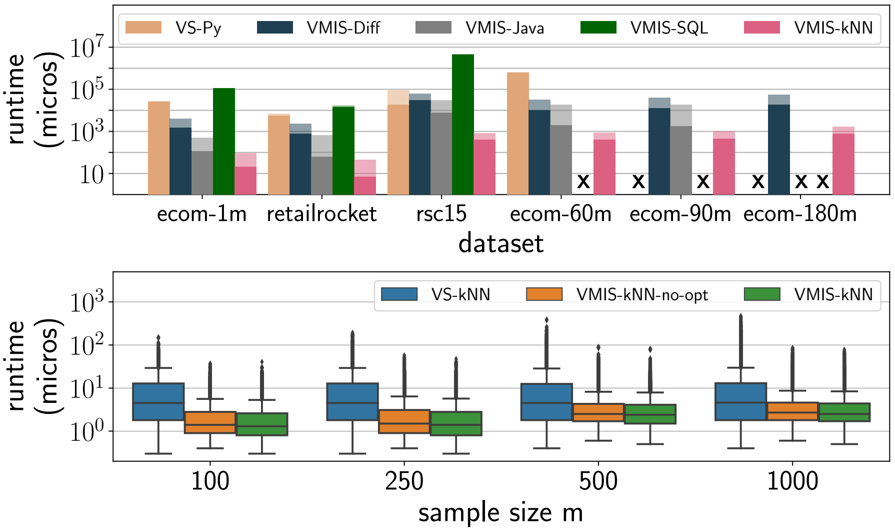


Figure 5.3: (Top) Median and 90th percentile of the computation time per session in microseconds (log-scale) for different VMIS-kNN implementations; (Bottom) Microbenchmark runtimes in microseconds (log-scale) for VMIS-kNN vs. VS-kNN on the *ecom-1m* dataset with k = 100.

and (ii) *VMIS-kNN-no-opt*, a basic variant of VMIS-kNN, which does not contain several optimisations such as early stopping or using octonary heaps instead of binary heaps.

We conduct a micro-benchmark on the *ecom-1m* dataset. We ask each variant to compute the k closest sessions for the sessions from the test set, and we randomly pick the number of items (e.g., the session length) for each session to include in the computation. We repeat this experiment ten times for various values of m (the number of most recent sessions to consider) with six threads, and measure the execution times for k = 100 (trying other values of k did not significantly change the results). We implement all algorithms in Rust 1.54 and run the comparison on a machine with an i9-10900KF CPU @ 3.7GHz with ten cores and 64GB of RAM, running Windows 10 21H1.

Results and discussion The bottom plot in Figure 5.3 shows the resulting runtimes in microseconds for each of our variants. The results are consistent across all values of m: We find that both *VMIS-kNN* and *VMIS-kNN-no-opt* drastically outperform the *VS-kNN* baseline by a factor of three to five. We attribute this observation to the optimised access patterns in the index of VMIS-kNN, which allows us to avoid costly set intersection operations, and the minimisation of intermediate results with our heap data structures (Section 5.4). We furthermore observe that *VMIS-kNN* consistently outperforms *VMIS-kNN-no-opt* by 6% to 12% which validates our micro optimisations such as early stopping and leveraging octonary heaps instead of binary heaps.

5.6.2 Serenade

Next, we evaluate our Serenade system. We validate our implementation choices (Section 5.6.2), run a load test for the system in Section 5.6.2, and finally present the results from a three week long A/B test on the live platform (Section 5.6.2).

Validation of Implementation Choices

We present an offline experiment which focuses on the performance of our index-based VMIS-kNN approach. We compare our Rust-based implementation against implementations in other programming languages and computational engines to validate our design choice of a custom implementation in Rust. Note that we provide the source code for the alternative implementations in our experiment repository as well.

Experimental setup We compare our Rust-based VMIS-kNN implementation against four other implementations:

- **VS-Py** – a Python-based implementation of the original VS-kNN approach, based on the reference code [6] from the original VS-kNN paper; we expect this variant to be non-competitive as it is a mere research implementation;
- **VMIS-Diff** – an implementation of VMIS-kNN in *Differential Dataflow* [90], which computes the recommendations incrementally via joins and aggregations; this variant allows us to evaluate the benefits of an incremental similarity computation for growing sessions;

- **VMIS-Java** – an implementation of VMIS-kNN in Java, which stores the historical session data in Java hashmaps; the purpose of this variant is to evaluate the effects of not having full control over the memory management during the similarity computation (and instead relying on a garbage collector);
- **VMIS-SQL** – an implementation of VMIS-kNN in SQL, which leverages the embeddable analytical database engine *DuckDB* [99] in version 0.2.2; the purpose of this variant is to evaluate whether a custom implementation of the approach is necessary; we note that we found it very difficult to express the similarity computation in plain SQL, as it required several deeply nested subqueries;

We ensure through evaluations on held-out data that all variants are correctly implemented and provide equal predictive performance. We expose the historical session data from our public and proprietary datasets to each of these baselines. Next, we ask each implementation to sequentially compute next-item recommendations with a single thread for the growing evolving sessions in the test set of each dataset, and measure the prediction time in microseconds. We run each implementation on a `n1-highmem-8` instance in the Google cloud with 50 gigabytes of RAM, and use $m = 5000$ and $k = 100$ as hyperparameter settings.

Results and discussion The top plot of Figure 5.3 illustrates the resulting runtimes from our experiment for the different datasets and baseline implementations. Note that we plot the median and 90th percentile (p90) of these runtimes on a logarithmic scale in a single bar, where the lighter top part denotes the 90th percentile runtime. Our VMIS-kNN implementation consistently outperforms all the baselines both in terms of median and p90 runtime; it is more than two orders of magnitude faster than the Python reference implementation, and more than one order of magnitude faster than the differential dataflow implementation. The second-best implementation is the Java baseline, which is still outperformed by an order of magnitude for the 90th percentile runtime on all datasets except the small *ecom-1m* dataset. When we look at the results for larger datasets, we additionally observe that several baselines start to encounter memory issues (even though they can use 50 gigabytes of RAM), and fail to complete the computation. This happens for the Python implementation (which relies on pandas dataframes internally), for the SQL implementation as well as for the Java variant. Note that our Serenade implementation provides a p90 runtime of at most 1.7 milliseconds on all datasets. We attribute this to the fact that our implementation allows us to carefully control memory allocation and to avoid the materialisation of large intermediate results (such as the complete set of item matches with the historical sessions). We observe that the differential implementation always manages to compute results; however, the incremental computation does not pay off runtime-wise, because differential dataflow has to index all intermediate results due to its support for updates in response to input data changes (which is not required in our use case). Finally, we find the SQL implementation to be non-competitive and to not scale to large datasets, which we attribute to the large intermediates from the nested subqueries, and which confirms that a custom implementation of VMIS-kNN is more suitable to scale to large datasets.

Offline Load Test

We finally run a load test in our staging environment to validate that Serenade is able to handle peak production workloads.

Experimental setup We leverage a setup that resembles our production environment: Serenade's index is built from the last 180 days of browsing activities, covering 6.5 million distinct items. We deploy Serenade on two Kubernetes pods, running on shared core `n1-standard-16` instances in the Google Cloud, where each pod gets provisioned with three cores from an Intel Xeon CPU @ 2.00GHz and 16 GB of RAM.

We generate a simulated load of more than 1,000 requests per second by replaying historical traffic via a load generator application for several hours. We measure the response latency of Serenade as well as the core usage on the machines.

Results and discussion Figure 5.4 plots the resulting response latency and core usage for our load test. We find that Serenade gracefully handles the load of more than 1,000 requests per second, and responds within less than 7 milliseconds in 90 percent of the cases (`p90`) and in less than 15 milliseconds in 99.5% of the cases (`p99.5`). Each instance uses only one of the three provisioned cores for most of the time. We base our production experiments in the following on the outcomes of this load test.

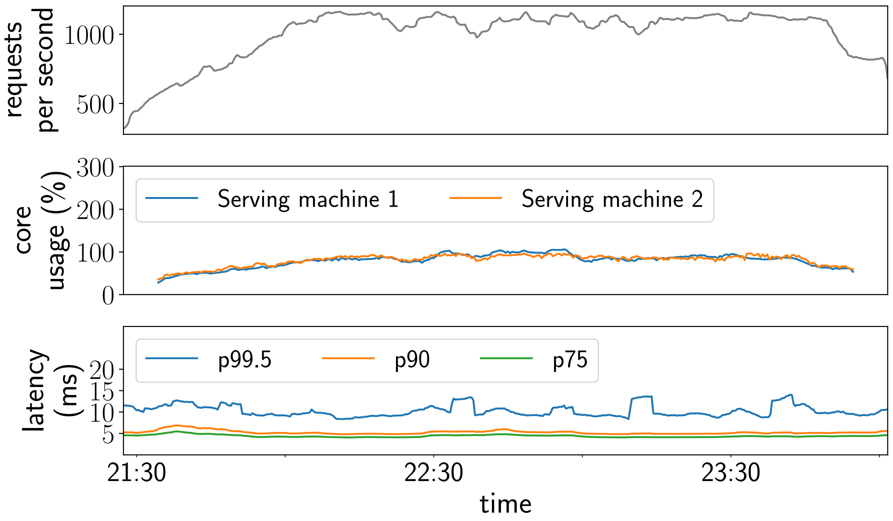


Figure 5.4: Requests per second, core usage in percent and response latency during a load test with more than 1,000 requests per second. Serenade handles about 500 requests per second per core with a 90th percentile latency of less than seven milliseconds.

Online Evaluation in an A/B Test

We present results from a three week long online A/B test on our e-commerce platform, where we compared two variants of Serenade against our existing legacy recommendation system (refered to as *legacy*), which applies a variant of classic item-to-item collaborative filtering [108].

Experimental setup We show Serenade's recommendations on the product detail page of our e-commerce platform, in a slot titled 'other customers also viewed'. We evaluate two different variants of Serenade: the first variant *serenade-hist* leverages the last two items from each evolving session to compute predictions, while the second variant *serenade-recent* only leverages the most recent item. We set the hyperparameters of VS-kNN to $m = 500$ and $k = 500$, which provide a reasonable trade-off between prediction quality in offline experiments and index size. We run the test for 21 days, in which more than 45 million randomly assigned user sessions were subjected to the recommendations from our variants. We ensure that both the legacy system and Serenade consume the same

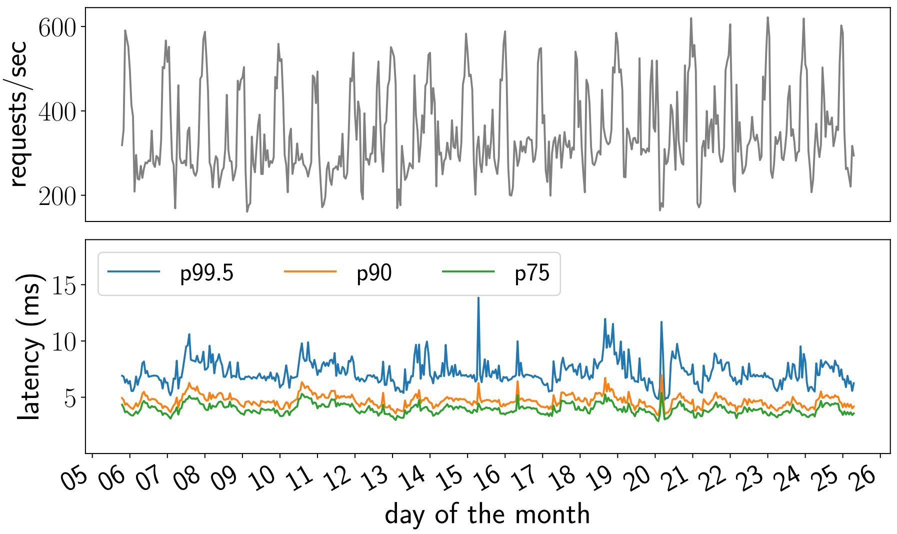


Figure 5.5: Requests per second and response latency per hour during our three week long A/B test on the live platform. Serenade responds within less than seven milliseconds in 90% of the cases, even for peak times with more than 600 requests per second.

click data as input at the same time (once per night). Serenade builds its index from the last 180 days of data; after filtering, its daily training data consists of around 111 million sessions with 582 million distinct user-item interactions and contains 6.5M distinct items. We measure the request load to the recommendation system, the response latency (as experienced from the shop frontend) and several business-specific engagement metrics.

Results and discussion We discuss the systems- and business-specific outcomes of our A/B test.

Response latency Our most important systems-related metric is the response latency. Our recommendation systems have to adhere to a strict SLA of responding in less than 50 milliseconds, otherwise requests would be discarded. Recent research also indicates that fast response times help with the acceptance of recommendations in general [70]. Our system architecture and implementation decisions (Section 5.5) are tailored to allow for low latency responses of our system. This is confirmed by the experimental results illustrated in Figure 5.5, which plots different percentiles of the response latency distribution over the three weeks of our A/B test, and shows the load of the system (in terms of the number of requests per second) for comparison. The request load varies between 200 and 600 requests per second over the day. We find that Serenade's response latencies are very low, the 90th percentile is consistenly around 5 milliseconds, and even the 99.5th percentile is below 10 milliseconds in the majority of cases. This confirms that Serenade exhibits a consistently fast, and stable low-latency response behavior.

CPU usage and operational cost We deploy Serenade analogously to the setup from the load test in Section 5.6.2: We leverage two Kubernetes pods, running on shared core `n1-standard-16` instances in the Google Cloud, where each pod gets provisioned with cores of an Intel Xeon CPU @ 2.00GHz and 16 GB of RAM. Even with such low resources, Serenade is able to gracefully handle the request workload. We reconfirm the findings from our load test (Section 5.6.2), as Serenade only exhibits a core usage of less than 36% (less than one core) in cases with over 500 requests per second. We also observe a well-behaved linear scaling (with a gentle slope) of the core usage with the number of requests per second.

Customer engagement Systems-related metrics are important for successfully operating a recommender system, however in the end the recommender system has to perform well in business-related metrics to be valuable for an e-commerce platform. As VMIS-kNN outperforms other approaches in offline evaluations (Section 5.6.1), we are interested to determine how this behavior translates to customer engagement in our A/B test. For that, we measure a conversion-related business metric for the engagement with recommendations on the product detail page.

We find that our session-based recommenders drastically increase this engagement metric for the slot on the product detail page. *Serenade-hist* exhibits a 2.85% increase in the business metric (compared to *legacy*), and *serenade-recent* even shows an increase of 5.72% (both findings are statistically significant). When we control for the overall impact on a site-wide level however, we find that *serenade-recent* exhibits a cannibalising behavior, as it drives down the engagement of other slots on the product detail page (e.g., the ‘often bought together’ slot). We do not observe this effect for *serenade-hist* though, rendering it the preferred variant.

Summary We find that Serenade easily handles the load of up to 600 requests per second during our A/B test and consistently generates its recommendations with very low response latency (less than seven milliseconds in the 90th percentile). We furthermore find that the session-based recommendations produced by VMIS-kNN significantly increase customer engagement compared to classical item-to-item recommendations (as produced by our *legacy* system). We would like to highlight that, to the best of our knowledge, we are the first to provide empirical evidence that the superior offline performance of VS-kNN/VMIS-kNN also translates to superior performance in terms of business metrics in a live, real recommender system. This is often not the case for academic recommendation approaches, the winning solution of the highly popularised Netflix prize, for example, never went into production [9].

5.7 Learnings & Conclusion

In this chapter we presented our nearest neighbor approach VMIS-kNN as well as the design and implementation of our scalable session-based recommender system *Serenade*. We conducted an extensive offline evaluation of VMIS-kNN and Serenade to validate our design decisions, and detailed results on the latency, throughput and predictive performance of our recommender system from an online A/B test with up to 600 requests per second for 6.5 million distinct items on more than 45 million user sessions on bol’s e-commerce platform.

In addition to the contributions listed in Section 5.1, we would like to highlight Serenade’s low operational cost: We run two instances with three cores each in the Google cloud (provisioned on shared core `n1-standard-16` instances) for the serving pods, and require 40 minutes on 75 machines of type `n1-highmem-8` for creating the index with Spark every day, which results in a total operational cost of less than 30 euros per day for Serenade. As discussed in Section 5.6.2, Serenade only leverages one of the three cores on each instance, and we only provision the other cores to be prepared for peak loads, e.g., during denial-of-service attacks.

This low cost becomes especially attractive when we compare it with the high cost to train deep learning models. As an example, a neural learning-to-rank model on our platform incurs at least an order of magnitude more cost to be operated on a daily basis, and additionally requires GPU machines for training, which are often a contested resource in the cloud.

In future work, we intend to explore whether we can run our similarity computations on a compressed version of the index, and whether we can incrementally maintain the index with a system such as Differential Dataflow [90].

This concludes the final research chapter of this thesis. In the next chapter, we summarize our findings and provide avenues for future work.

Algorithm 5.2 Vector-Multiplication-Indexed-Session kNN (VMIS-kNN)

```
 1: function VMIS-KNN($\mathbf{s}^{(t)}, (\mathbf{M}, \mathbf{t}), \pi, \lambda, m, k$)
 2:     Input: Evolving session $\mathbf{s}^{(t)}$, session similarity index $(\mathbf{M}, \mathbf{t})$, decay function $\pi$,
 3:        sample size $m$, match weight function $\lambda$, number of neighbors $k$.
 4:     Output: Scored list of recommended next items $\mathbf{d}$.
 5:     $(\mathbf{N}_s, \mathbf{r}) \leftarrow$ neighbor_sessions_from_index($\mathbf{s}^{(t)}, (\mathbf{M}, \mathbf{t}), \pi, m, k$)
 6:     for each item $i$ occuring in the sessions $\mathbf{N}_s$ do
 7:         $d_i \leftarrow \sum_{\mathbf{n} \in \mathbf{N}_s} \mathbb{1}_n(i) \cdot \lambda(\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{n})) \cdot r_n \cdot \log \frac{|H|}{h_i}$
        return item scores $\mathbf{d}$

 8: function NEIGHBOR_SESSIONS_FROM_INDEX($\mathbf{s}^{(t)}, (\mathbf{M}, \mathbf{t}), \pi, m, k$)
 9:     initialize hashmap $\mathbf{r}$ for temporary similarity scores, min-heap $\mathbf{b}_t$ of cap-
10:     acity $m$ for the most recent similar historical sessions, hashset $\mathbf{d}$ for already
11:     processed items, max-heap $\mathbf{N}_s$ of capacity $k$ for closest sessions
12:     for item $i \in \mathbf{s}^{(t)}$ in reverse insertion order do          ▷ Item intersection loop
13:         if $i \notin \mathbf{d}$ then
14:             insert $i$ into $\mathbf{d}$
15:             $\mathbf{m}_i \leftarrow$ most recent sessions for item $i$ from inverted index $\mathbf{M}$
16:             $\pi_i \leftarrow$ decay weight $\pi(\omega(\mathbf{s}^{(t)}))_i$ of item $i$ in session $\mathbf{s}^{(t)}$
17:             for session $j \in \mathbf{m}_i$ do
18:                 if $j \in$ keys($\mathbf{r}$) then
19:                     $r_j \leftarrow r_j + \pi_i$
20:                 else
21:                     $t_j \leftarrow$ timestamp of session $j$ fetched from index $\mathbf{t}$
22:                     if $|\mathbf{r}| < m$ then
23:                         $r_j \leftarrow \pi_i$
24:                         insert $(j, r_j)$ into $\mathbf{r}$
25:                         insert $(j, t_j)$ into $\mathbf{b}_t$
26:                     else
27:                         $(l, t_l) \leftarrow$ current heap root of $\mathbf{b}_t$
28:                         if $t_j > t_l$ then
29:                             $r_j \leftarrow \pi_i$
30:                             remove $(l, r_l)$ from $\mathbf{r}$
31:                             insert $(j, r_j)$ into $\mathbf{r}$
32:                             update heap root of $\mathbf{b}_t$ with $(j, t_j)$
33:                         else break

34:     for $(j, r_j) \in \mathbf{r}$ do                                    ▷ Top-k similarity loop
35:         if $|\mathbf{N}_s| < k$ then insert $(j, r_j)$ into $\mathbf{N}_s$
36:         else
37:             $(n, r_n) \leftarrow$ current heap root of $\mathbf{N}_s$
38:             if $r_j > r_n$ then update heap root of $\mathbf{N}_s$ with $(j, r_j)$
39:             else if $r_j = r_n$ and $t_j > t_n$ then update heap root of $\mathbf{N}_s$ with $(j, r_j)$
40:     return $\mathbf{N}_s$
```

6 Conclusion

In this thesis we investigated the forecasting problem for large-scale settings: how can we efficiently and accurately generate forecasts when we need to generate many of them? We have shown (i) how we can improve the efficiency and accuracy of different model classes for point- and probabilistic forecasting in Chapters 2–3, (ii) how we can efficiently leverage cross-sectional and temporal information of time series in Chapter 4 and (iii) how we can modify, implement and productionize a state-of-the-art academic recommender system in Chapter 5.

In this chapter, we first look back at the research questions from Chapter 1 and summarize our findings to these questions in Section 6.1. Finally, we conclude with our view on future work in Section 6.2.

6.1 Summary of Findings

Research Question 1: *How can we efficiently generate probabilistic forecasts with neural networks for large-scale settings?*

In Chapter 2 we introduced Bidirectional Temporal Convolutional Network (BiTCN), a neural network that can be used for probabilistic forecasting for large-scale settings that requires an order of magnitude fewer parameters than a common transformer-based approach. Experiments on four real-world datasets showed that BiTCN performs on par with four state-of-the-art probabilistic forecasting methods, including a transformer-based approach and WaveNet, on two point metrics (sMAPE, NRMSE) as well as on a set of range metrics (quantile loss percentiles) in the majority of cases. Secondly, we demonstrated that our method requires significantly fewer parameters than transformer-based methods, which means the model can be trained faster with significantly lower memory requirements, which as a consequence reduces the infrastructure cost for deploying these models.

Research Question 2: *How can we efficiently generate probabilistic forecasts with Gradient Boosting Machines (GBM) for large-scale settings?*

Motivated by the use of GBM at our industry partnerss, we proposed *Probabilistic Gradient Boosting Machines* (PGBM) in Chapter 3, a method to create probabilistic predictions with a single ensemble of decision trees in a computationally efficient manner. We empirically demonstrated the advantages of PGBM compared to existing state-of-the-art methods: (i) PGBM enables probabilistic estimates without compromising on point performance in a single model, (ii) PGBM learns probabilistic estimates via a single model only (and without requiring multi-parameter boosting), and thereby offers a speedup of up to several orders of magnitude over existing state-of-the-art methods on large datasets, and (iii) PGBM achieves accurate probabilistic estimates in tasks with complex differentiable loss functions, such as hierarchical time series problems, where we observed up to 10% improvement in point forecasting performance and up to 300% improvement in probabilistic forecasting performance. We thus found that we can efficiently generate accurate probabilistic forecasts using another common class of machine learning models (GBM), whilst retaining accuracy.

Research Question 3: *How can we efficiently generate hierarchical forecasts for large-scale settings?*

In Chapter 4, we further investigated the problem of hierarchical forecasting, which we briefly touched upon in Chapter 3. We found that existing hierarchical forecasting techniques scale relatively poorly to large-scale problem settings, and investigated methods that overcome these limitations. We proposed to learn a coherent forecast for millions of products with a single bottom-level forecast model by using a loss function that directly optimizes the hierarchical product structure. By removing the need for a post-processing step as required in traditional hierarchical forecasting techniques, we reduced the computational cost of the prediction phase in the forecasting pipeline, as well as its deployment complexity.

In our tests on the public M5 dataset, our sparse hierarchical loss function performed up to 10% better as measured by RMSE and MAE compared to the baseline loss function. Next, we implemented our sparse hierarchical loss function within an existing gradient boosting-based forecasting model at bol. In this setting our sparse hierarchical loss resulted in an improved forecasting performance as measured by RMSE of about 2% at the product level, as compared to the baseline model, and an improvement of about 10% at the product level as measured by MAE. Finally, we found an increase in forecasting performance of about 5–10% (both RMSE and MAE) when evaluating the forecasting performance across the cross-sectional hierarchies that we defined. These results demonstrated the usefulness of our sparse hierarchical loss applied to a production forecasting system at a major e-commerce platform.

Research Question 4: *How can we efficiently generate session-based recommendations at the scale of bol?*

In Chapter 5, we turned to another forecasting problem often encountered at our industry partners: recommendations. We investigated state-of-the-art methods for session-based recommendation and surprisingly found that the most simple method gave the most accurate results. We proposed Vector-Multiplication-Indexed-Session kNN (VMIS-kNN), an adaptation of a state-of-the-art nearest neighbor approach to session-based recommendation, which leverages a prebuilt index to compute next-item recommendations with low latency in scenarios with hundreds of millions of clicks to search through. Based on this approach, we designed and implemented the scalable session-based recommender system *Serenade*, which is in production usage at *bol*.

We evaluated the predictive performance of VMIS-kNN, and showed that Serenade can answer a thousand recommendation requests per second with a 90th percentile latency of less than seven milliseconds in scenarios with millions of items to recommend. Furthermore, we presented results from a three week long online A/B test with up to 600 requests per second for 6.5 million distinct items on more than 45 million user sessions from our e-commerce platform. To the best of our knowledge, we provided the first empirical evidence that the superior predictive performance of nearest neighbor approaches to session-based recommendation in offline evaluations translates to superior performance in a real world e-commerce setting.

6.2 Future Work

Each of the questions we answered throughout this thesis leads to new questions, and thus, potential for future work. In this section we briefly discuss the possibilities for future work.

Probabilistic forecasting. First, in the area of large-scale probabilistic forecasting, we demonstrated how to learn the parameters of a fixed distribution using a neural network (Chapter 2) and using GBM (Chapter 3). An exciting opportunity for future work is to relax the constraint of choosing a distribution a priori as in Chapter 2 or a distribution that is limited to the (location, scale)-family of distributions as in Chapter 3. There are several works that have tried to solve this problem, for example for neural networks by implicit quantile regression [45] or by directly learning the empirical quantile function of the data [42], and for GBM by ensembling approaches [48] or by using conformal prediction [105]. However, these methods have their limitations too, and generating distribution-free probabilistic forecasts efficiently at large-scale remains a difficult issue to tackle. Second, the increasing success of Large Language Models (LLMs) has motivated its application to the time series domain, and early work on generalized time series models is starting to surface [40]. This presents an interesting efficiency tradeoff: the cost to train such a large model may be huge due to the required amount of training data and sheer model size, but if it can be subsequently applied to many use cases it can be very resource efficient as we can ‘amortize’ the cost of training across a large set of use cases.

Structured forecasting. In the area of hierarchical forecasting that we discussed in Chapter 4, we see potential for work that can leverage both temporal and structural information more efficiently than existing methods – even though this was our main aim in Chapter 4. Next, we see that forecasting practitioners are increasingly interested in understanding the causal relationships between inputs and outputs in a forecasting model. For example, what is the causal effect of changing a price on the demand of the product that a retailer sells? Traditional causal inference techniques are commonly hard to apply to the time series domain due to the temporal nature of the data, whereas traditional forecasting methods commonly 'ignore' the causal treatment/outcome paradigm by simply treating every input as a statistical covariate, rather than a causal treatment effect. Hence, further bringing together these fields might improve both causal understanding of the underlying process that is forecast as well as improve forecasting performance.

Decision making. As we started this thesis with, a forecast is typically a tool to improve decision making. We did not discuss this part in this thesis, but it would be a natural extension for future work to extend beyond forecasting into decision making. For example, a probabilistic forecasting model can be used as a simulator to generate possible process paths, which can in turn be optimized, for example using Reinforcement Learning (RL) [123] approaches. Typical practioner questions that such approaches could answer are 'what price should I set my product at to maximize my profit' or 'what product do I need to put on the shelves'?

Recommender systems. In the area of (session-based) recommender systems we see a disconnect between academic research and industry practice. In Chapter 5 we found that simple methods commonly outperform complex, neural network-based methods on our task even though these neural network techniques have been highly succesful in other application areas. Hence, it seems that there is still an opportunity to investigate better representation learning methods for (session-based) recommender systems. Secondly, in Chapter 5 we found that complex methods can be hard to apply in practice due to their high running time. Reducing the running time of (session-based) recommender systems is an exciting avenue for future work, for example by using approximation techniques and/or leveraging accelators [65, 114].

Bibliography

[1] Actix Web. https://actix.rs, 2021. (Cited on page 126.)

[2] D-ary heap. https://docs.rs/dary_heap/0.3.0/dary_heap/, 2021. (Cited on page 123.)

[3] Kubernetes networking services. https://kubernetes.io/docs/concepts/services-networking/service/, 2021. (Cited on page 127.)

[4] RocksDB. https://rocksdb.org, 2021. (Cited on page 126.)

[5] Istio sidecars. https://istio.io/latest/docs/reference/config/networking/sidecar/, 2021. (Cited on page 128.)

[6] VS-kNN reference implementation. https://github.com/rn5l/session-rec/blob/master/algorithms/knn/vsknn.py, 2021. (Cited on page 133.)

[7] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 2623–2631, New York, NY, USA, July 2019. Association for Computing Machinery. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330701. (Cited on pages 25, 93, and 110.)

[8] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. Rangapuram, D. Salinas, J. Schulz, L. Stella, A. C. Türkmen, and Y. Wang. GluonTS: Probabilistic and Neural Time Series Modeling in Python. *Journal of Machine Learning Research*, 21(116):1–6, 2020. ISSN 1533-7928. (Cited on pages 25 and 50.)

[9] X. Amatriain. Building Industrial-scale Real-World Recommender Systems. *RecSys*, pages 7–8, 2012. (Cited on pages 118 and 139.)

[10] I. Arapakis, X. Bai, and B. B. Cambazoglu. Impact of Response Latency on User Behavior in Web Search. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 103–112, 2014. (Cited on page 116.)

[11] V. Assimakopoulos and K. Nikolopoulos. The Theta Model: A Decomposition Approach to Forecasting. *International Journal of Forecasting*, 16(4):521–530, Oct. 2000. ISSN 0169-2070. doi: 10.1016/S0169-2070(00)00066-2. (Cited on pages 24 and 92.)

[12] G. Athanasopoulos, R. J. Hyndman, N. Kourentzes, and F. Petropoulos. Forecasting with Temporal Hierarchies. *European Journal of Operational Research*, 262(1):60–74, Oct. 2017. ISSN 0377-2217. doi: 10.1016/j.ejor.2017.02.046. (Cited on pages 4, 80, 81, 83, and 84.)

[13] G. Athanasopoulos, R. J. Hyndman, N. Kourentzes, and A. Panagiotelis. Forecast Reconciliation: A Review. *International Journal of Forecasting*, In Press, 2024. (Cited on pages 83 and 84.)

[14] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly

Learning to Align and Translate. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. (Cited on page 14.)

[15] S. Bai, J. Z. Kolter, and V. Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv:1803.01271 [cs]*, Apr. 2018. (Cited on pages 15, 17, and 23.)

[16] S. Ben Taieb. Sparse and Smooth Adjustments for Coherent Forecasts in Temporal Aggregation of Time Series. In *Proceedings of the Time Series Workshop at NIPS 2016*, pages 16–26. PMLR, Feb. 2017. (Cited on page 83.)

[17] S. Ben Taieb and B. Koo. Regularized Regression for Hierarchical Forecasting Without Unbiasedness Conditions. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1337–1347, Anchorage AK USA, July 2019. ACM. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330976. (Cited on pages 83, 85, 86, and 93.)

[18] S. Ben Taieb, R. Huser, R. J. Hyndman, and M. G. Genton. Probabilistic Time Series Forecasting with Boosted Additive Models: An Application to Smart Meter Data. Technical report, Monash University, Department of Econometrics and Business Statistics, 2015. (Cited on page 51.)

[19] S. Ben Taieb, J. W. Taylor, and R. J. Hyndman. Coherent Probabilistic Forecasts for Hierarchical Time Series. In *International Conference on Machine Learning*, pages 3348–3357, July 2017. (Cited on page 83.)

[20] K. Benidis, S. S. Rangapuram, V. Flunkert, Y. Wang, D. Maddix, C. Turkmen, J. Gasthaus, M. Bohlke-Schneider, D. Salinas, L. Stella, F.-X. Aubet, L. Callot, and T. Januschowski. Deep Learning for Time Series Forecasting: Tutorial and Literature Survey. *ACM Computing Surveys*, 55(6):1–36, July 2023. ISSN 0360-0300, 1557-7341. doi: 10.1145/3533382. (Cited on page 83.)

[21] J.-H. Böse, V. Flunkert, J. Gasthaus, T. Januschowski, D. Lange, D. Salinas, S. Schelter, M. Seeger, and Y. Wang. Probabilistic Demand Forecasting at Scale. *Proceedings of the VLDB Endowment*, 10(12):1694–1705, Aug. 2017. ISSN 21508097. doi: 10.14778/3137765.3137775. (Cited on pages 4, 12, 48, and 82.)

[22] G. E. P. Box and D. A. Pierce. Distribution of Residual Autocorrelations in Autoregressive-Integrated Moving Average Time Series Models. *Journal of the American Statistical Association*, 65(332):1509–1526, 1970. ISSN 0162-1459. doi: 10.2307/2284333. (Cited on pages 14, 50, and 92.)

[23] B. Chandramouli, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. StreamRec: A Real-Time Recommender System. *SIGMOD*, pages 1243–1246, 2011. (Cited on page 118.)

[24] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, San Francisco California USA, Aug. 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. (Cited on pages 3, 48, 49, 50, 52, 53, 56, 57, 59, and 83.)

[25] T. Chen, H. Yin, H. Chen, L. Wu, H. Wang, X. Zhou, and X. Li. TADA: Trend Alignment with Dual-Attention Multi-task Recurrent Neural Networks for Sales

Prediction. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 49–58, Singapore, Nov. 2018. IEEE. ISBN 978-1-5386-9159-5. doi: 10.1109/ICDM.2018.00020. (Cited on page 14.)

[26] T. Chen, H. Yin, H. Chen, R. Yan, Q. V. H. Nguyen, and X. Li. Air: Attentional Intention-aware Recommender Systems. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 304–315. IEEE, 2019. (Cited on page 118.)

[27] Y. Chen, Y. Kang, Y. Chen, and Z. Wang. Probabilistic Forecasting with Temporal Convolutional Neural Network. *Neurocomputing*, 399:491–501, July 2020. ISSN 0925-2312. doi: 10.1016/j.neucom.2020.03.011. (Cited on pages 15 and 23.)

[28] H. A. Chipman, E. I. George, and R. E. McCulloch. BART: Bayesian Additive Regression Trees. *The Annals of Applied Statistics*, 4(1):266–298, Mar. 2010. ISSN 1932-6157. doi: 10.1214/09-AOAS285. (Cited on page 50.)

[29] K.-J. Cho, Y.-C. Lee, K. Han, J. Choi, and S.-W. Kim. No, That's Not My Feedback: TV Show Recommendation Using Watchable Interval. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 316–327. IEEE, 2019. (Cited on page 118.)

[30] J. D. Croston. Forecasting and Stock Control for Intermittent Demands. *Operational Research Quarterly (1970-1977)*, 23(3):289–303, 1972. ISSN 0030-3623. doi: 10.2307/3007885. (Cited on page 92.)

[31] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google News Personalization: Scalable Online Collaborative Filtering. *WWW*, pages 271–280, 2007. (Cited on page 118.)

[32] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath. The YouTube Video Recommendation System. *RecSys*, pages 293–296, 2010. (Cited on page 118.)

[33] S. Deng, O. Sprangers, M. Li, S. Schelter, and M. de Rijke. Domain Generalization in Time Series Forecasting. *ACM Transactions on Knowledge Discovery from Data*, Jan. 2024. ISSN 1556-4681. doi: 10.1145/3643035.

[34] T. Duan, A. Avati, D. Ding, K. K. Thai, S. Basu, A. Ng, and A. Schuler. NGBoost: Natural Gradient Boosting for Probabilistic Prediction. *ICML*, 2020. (Cited on pages 3, 48, 51, 59, and 62.)

[35] T. Dunning and E. Friedman. *Practical Machine Learning: Innovations in Recommendation*. O'Reilly Media, Inc., 2014. (Cited on page 118.)

[36] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson. CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop. *Proceedings of the VLDB Endowment*, 4(9):575–585, 2011. (Cited on page 125.)

[37] T. Fischer and C. Krauss. Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions. *European Journal of Operational Research*, 270(2):654–669, Oct. 2018. ISSN 0377-2217. doi: 10.1016/j.ejor.2017.11.054. (Cited on pages 2, 11, 12, and 14.)

[38] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189–1232, 2001. ISSN 0090-5364. (Cited on pages 50 and 52.)

[39] C. Gao, X. He, D. Gan, X. Chen, F. Feng, Y. Li, T.-S. Chua, and D. Jin. Neural

Multi-Task Recommendation from Multi-Behavior Data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1554–1557. IEEE, 2019. (Cited on page 118.)

[40] A. Garza and M. Mergenthaler-Canseco. TimeGPT-1. *arXiv:2310.03589 [cs, stat]*, Oct. 2023. (Cited on page 146.)

[41] F. Garza, M. Mergenthaler Canseco, C. Challú, and K. G. Olivares. StatsForecast: Lightning fast forecasting with statistical and econometric models. In *PyCon*, Salt Lake City, USA, 2022. (Cited on page 93.)

[42] J. Gasthaus, K. Benidis, Y. Wang, S. S. Rangapuram, D. Salinas, V. Flunkert, and T. Januschowski. Probabilistic Forecasting with Spline Quantile Function RNNs. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1901–1910, Apr. 2019. (Cited on pages 39 and 146.)

[43] D. Girolimetto and T. Di Fonzo. Point and Probabilistic Forecast Reconciliation for General Linearly Constrained Multiple Time Series. *Statistical Methods & Applications*, In Press, May 2023. doi: 10.1007/s10260-023-00738-6. (Cited on page 83.)

[44] H. Gouk, B. Pfahringer, and E. Frank. Stochastic Gradient Trees. In *Asian Conference on Machine Learning*, pages 1094–1109. PMLR, Oct. 2019. (Cited on page 51.)

[45] A. Gouttes, K. Rasul, M. Koren, J. Stephan, and T. Naghibi. Probabilistic Time Series Forecasting with Implicit Quantile Networks. In *Proceedings of the Time Series Workshop at ICML 2021*, volume 139. PMLR, July 2021. (Cited on page 146.)

[46] L. Guo, H. Yin, Q. Wang, B. Cui, Z. Huang, and L. Cui. Group Recommendation with Latent Voting Mechanism. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 121–132. IEEE, 2020. (Cited on page 118.)

[47] X. Han, S. Dasgupta, and J. Ghosh. Simultaneously Reconciled Quantile Forecasting of Hierarchically Related Time Series. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 190–198. PMLR, Mar. 2021. (Cited on pages 83 and 85.)

[48] H. Hasson, B. Wang, T. Januschowski, and J. Gasthaus. Probabilistic Forecasting: A Level-Set Approach. In *Advances in Neural Information Processing Systems*, volume 34, pages 6404–6416. Curran Associates, Inc., 2021. (Cited on pages 106 and 146.)

[49] J. He, J. Qi, and K. Ramamohanarao. A Joint Context-Aware Embedding for Trip Recommendations. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 292–303. IEEE, 2019. (Cited on page 118.)

[50] D. Hendrycks and K. Gimpel. Gaussian Error Linear Units (GELUs). *arXiv:1606.08415 [cs]*, Nov. 2018. (Cited on page 18.)

[51] H. Hewamalage, C. Bergmeir, and K. Bandara. Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427, Jan. 2021. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2020.06.008. (Cited on page 14.)

[52] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-Based Recommendations with Recurrent Neural Networks. In *4th International Conference*

on Learning Representations (ICLR 2016), Conference Track Proceedings, San Juan, Puerto Rico, 2016. (Cited on pages 119 and 129.)

[53] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, Nov. 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. (Cited on page 14.)

[54] C. C. Holt. Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages. *International Journal of Forecasting*, 20(1):5–10, Jan. 2004. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2003.09.015. (Cited on page 24.)

[55] H. Hu, X. He, J. Gao, and Z.-L. Zhang. Modeling Personalized Item Frequency Information for Next-Basket Recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1071–1080, 2020. (Cited on pages 119 and 130.)

[56] Y. Huang, B. Cui, W. Zhang, J. Jiang, and Y. Xu. TencentRec: Real-time Stream Recommendation in Practice. *SIGMOD*, pages 227–238, 2015. (Cited on page 118.)

[57] R. J. Hyndman. Another look at Forecast-Accuracy Metrics for Intermittent Demand. *Foresight: The International Journal of Applied Forecasting*, page 4, June 2006. (Cited on page 25.)

[58] R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2018. (Cited on pages 22 and 50.)

[59] R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice (3rd Ed)*. OTexts: Melbourne, Australia, 2021. (Cited on pages 84, 92, and 93.)

[60] R. J. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder. *Forecasting with Exponential Smoothing: The State Space Approach*. Springer Science & Business Media, June 2008. ISBN 978-3-540-71918-2. (Cited on pages 14 and 92.)

[61] R. J. Hyndman, R. A. Ahmed, G. Athanasopoulos, and H. L. Shang. Optimal Combination Forecasts for Hierarchical Time Series. *Computational Statistics & Data Analysis*, 55(9):2579–2589, Sept. 2011. ISSN 0167-9473. doi: 10.1016/j.csda.2011.03.006. (Cited on pages 4, 80, 81, 83, 85, 86, and 93.)

[62] R. J. Hyndman, A. J. Lee, and E. Wang. Fast Computation of Reconciled Forecasts for Hierarchical and Grouped Time Series. *Computational Statistics & Data Analysis*, 97:16–32, May 2016. ISSN 0167-9473. doi: 10.1016/j.csda.2015.11.007. (Cited on page 83.)

[63] D. Jannach and M. Ludewig. When Recurrent Neural Networks Meet the Neighborhood for Session-Based Recommendation. *RecSys*, pages 306–310, 2017. (Cited on page 119.)

[64] T. Januschowski, Y. Wang, K. Torkkola, T. Erkkilä, H. Hasson, and J. Gasthaus. Forecasting with Trees. *International Journal of Forecasting*, 38(4):1473–1481, Oct. 2022. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2021.10.004. (Cited on pages 82, 83, 91, and 92.)

[65] J. Johnson, M. Douze, and H. Jégou. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, July 2021. ISSN 2332-7790. doi: 10.1109/TBDATA.2019.2921572. (Cited on page 147.)

[66] Kaggle. Corporación Favorita Grocery Sales Forecasting. https://kaggle.com/c/favorita-grocery-sales-forecasting, 2017. (Cited on

page 22.)

[67] Kaggle. Web Traffic Time Series Forecasting. https://kaggle.com/c/web-traffic-time-series-forecasting, 2017. (Cited on page 22.)

[68] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017. (Cited on pages 3, 24, 48, 49, 50, 56, 57, 59, 62, 82, 83, and 91.)

[69] G. Kechyn, L. Yu, Y. Zang, and S. Kechyn. Sales Forecasting Using WaveNet within the Framework of the Kaggle Competition. *arXiv:1803.04037 [cs]*, Mar. 2018. (Cited on pages 6, 13, 15, and 23.)

[70] B. Kersbergen and S. Schelter. Learnings from a Retail Recommendation System on Billions of Interactions at Bol.com. *ICDE*, 2021. (Cited on pages 116, 118, 119, 129, and 137.)

[71] B. Kersbergen, O. Sprangers, and S. Schelter. Serenade - Low-Latency Session-Based Recommendation in e-Commerce at Scale. In *Proceedings of the 2022 International Conference on Management of Data*, pages 150–159, Philadelphia, USA, June 2022. ACM. ISBN 978-1-4503-9249-5. doi: 10.1145/3514221.3517901. (Cited on pages 5 and 7.)

[72] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference on Learning Representations*, Dec. 2014. (Cited on page 24.)

[73] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8), 2009. (Cited on pages 116, 118, and 119.)

[74] M. Kunz, S. Birr, M. Raslan, L. Ma, and T. Januschowski. Deep Learning Based Forecasting: A Case Study from the Online Fashion Industry. In *Forecasting with Artificial Intelligence: Theory and Applications*, Palgrave Advances in the Economics of Innovation and Technology, pages 279–311. Springer Nature Switzerland, Cham, 2023. ISBN 978-3-031-35879-1. doi: 10.1007/978-3-031-35879-1_11. (Cited on pages 12 and 92.)

[75] G. Lai, W.-C. Chang, Y. Yang, and H. Liu. Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, pages 95–104, Ann Arbor, MI, USA, June 2018. ISBN 978-1-4503-5657-2. doi: 10.1145/3209978.3210006. (Cited on page 14.)

[76] N. Laptev, J. Yosinski, E. L. Li, and S. Smyl. Time-series Extreme Event Forecasting with Neural Networks at Uber. In *ICML 2017 Time Series Workshop*, 2017. (Cited on pages 2, 11, 12, and 14.)

[77] J. J. Levandoski, M. Sarwat, M. F. Mokbel, and M. D. Ekstrand. RecStore: An Extensible and Adaptive Framework for Online Recommender Queries inside the Database Engine. *EDBT*, pages 86–96, 2012. (Cited on page 118.)

[78] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma. Neural Attentive Session-Based Recommendation. *CIKM*, pages 1419–1428, 2017. (Cited on pages 116, 119, and 129.)

[79] M. Li, S. Jullien, M. Ariannezhad, and M. de Rijke. A Next Basket Recommendation Reality Check. *ACM Transactions on Information Systems*, 41(4):

116:1–116:29, Apr. 2023. ISSN 1046-8188. doi: 10.1145/3587153. (Cited on pages 119 and 130.)

[80] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting. In *Advances in Neural Information Processing Systems 32*, pages 5244–5254. Curran Associates, Inc., 2019. (Cited on pages 2, 11, 12, 15, 23, 24, 27, 30, 32, 51, and 92.)

[81] Y. Li, R. Yu, C. Shahabi, and Y. Liu. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. (Cited on pages 2, 11, 12, and 14.)

[82] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister. Temporal Fusion Transformers for Interpretable Multi-Horizon Time Series Forecasting. *International Journal of Forecasting*, 37(4):1748–1764, Oct. 2021. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2021.03.012. (Cited on pages 51 and 92.)

[83] Q. Liu, Y. Zeng, R. Mokhosi, and H. Zhang. STAMP: Short-Term Attention/Memory Priority Model for Session-Based Recommendation. *KDD : proceedings. International Conference on Knowledge Discovery & Data Mining*, pages 1831–1839, 2018. (Cited on pages 116, 119, and 129.)

[84] M. Ludewig, N. Mauro, S. Latifi, and D. Jannach. Performance Comparison of Neural and Non-Neural Approaches to Session-Based Recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 462–466, 2019. (Cited on pages 116, 117, 119, 128, 129, and 130.)

[85] S. Madisetty. Event Recommendation Using Social Media. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 2106–2110. IEEE, 2019. (Cited on page 118.)

[86] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The M4 Competition: 100,000 Time Series and 61 Forecasting Methods. *International Journal of Forecasting*, 36(1):54–74, Jan. 2020. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2019.04.014. (Cited on page 14.)

[87] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The M5 Competition: Background, Organization, and Implementation. *International Journal of Forecasting*, Sept. 2021. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2021.07.007. (Cited on pages 86, 90, 91, and 109.)

[88] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. M5 Accuracy Competition: Results, Findings, and Conclusions. *International Journal of Forecasting*, 38(4): 1346–1364, Oct. 2022. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2021.11.013. (Cited on pages 24, 39, 48, 51, 70, 81, 82, 83, 88, 91, and 95.)

[89] Z. Mariet and V. Kuznetsov. Foundations of Sequence-to-Sequence Modeling for Time Series. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 408–417, Apr. 2019. (Cited on page 14.)

[90] F. McSherry, D. G. Murray, R. Isaacs, and M. Isard. Differential Dataflow. In *CIDR*, 2013. (Cited on pages 133 and 140.)

[91] N. Meinshausen. Quantile Regression Forests. *Journal of Machine Learning*

Research, 7(35):983–999, 2006. ISSN 1533-7928. (Cited on page 51.)

[92] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, DB. Tsai, M. Amde, S. Owen, et al. Mllib: Machine Learning in Apache Spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016. (Cited on page 126.)

[93] P. Montero-Manso and R. J. Hyndman. Principles and Algorithms for Forecasting Groups of Time Series: Locality and Globality. *International Journal of Forecasting*, June 2021. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2021.03.004. (Cited on page 14.)

[94] S. Owen. *Mahout in Action*, volume 10. Manning Shelter Island, NY, 2012. (Cited on page 118.)

[95] A. Panagiotelis, G. Athanasopoulos, P. Gamakumara, and R. J. Hyndman. Forecast Reconciliation: A Geometric View with New Insights on Bias Correction. *International Journal of Forecasting*, 37(1):343–359, Jan. 2021. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2020.06.004. (Cited on page 83.)

[96] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. (Cited on pages 24 and 57.)

[97] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. ISSN 1533-7928. (Cited on pages 7 and 59.)

[98] A. Pfadler, H. Zhao, J. Wang, L. Wang, P. Huang, and D. L. Lee. Billion-Scale Recommendation with Heterogeneous Side Information at Taobao. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 1667–1676. IEEE, 2020. doi: 10.1109/ICDE48307.2020.00148. (Cited on page 118.)

[99] M. Raasveldt and H. Mühleisen. DuckDB: An Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1981–1984, 2019. (Cited on page 134.)

[100] S. S. Rangapuram, L. D. Werner, K. Benidis, P. Mercado, J. Gasthaus, and T. Januschowski. End-to-End Learning of Coherent Probabilistic Forecasts for Hierarchical Time Series. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8832–8843. PMLR, July 2021. (Cited on pages 81, 82, 83, 85, and 86.)

[101] S. S. Rangapuram, S. Kapoor, R. S. Nirwan, P. Mercado, T. Januschowski, Y. Wang, and M. Bohlke-Schneider. Coherent Probabilistic Forecasting of Temporal Hierarchies. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, pages 9362–9376. PMLR, Apr. 2023. (Cited on pages 4, 80, 81, 82, 83, 84, and 85.)

[102] P. Ren, Z. Chen, J. Li, Z. Ren, J. Ma, and M. de Rijke. RepeatNet: A Repeat

Aware Neural Recommendation Machine for Session-Based Recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4806–4813, 2019. (Cited on page 116.)

[103] F. Ricci, L. Rokach, and B. Shapira. Introduction to Recommender Systems. In *Recommender Systems Handbook*, pages 1–35. Springer, 2011. (Cited on page 118.)

[104] R. A. Rigby and D. M. Stasinopoulos. Generalized Additive Models for Location, Scale and Shape. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 54(3):507–554, 2005. ISSN 1467-9876. doi: 10.1111/j.1467-9876.2005.00510.x. (Cited on page 50.)

[105] Y. Romano, E. Patterson, and E. Candes. Conformalized Quantile Regression. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. (Cited on page 146.)

[106] T. Salimans and D. P. Kingma. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. In *Advances in Neural Information Processing Systems 29*, pages 901–909. Curran Associates, Inc., 2016. (Cited on page 18.)

[107] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *International Journal of Forecasting*, Oct. 2019. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2019.07.001. (Cited on pages 14, 23, 24, 25, 27, and 51.)

[108] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-Based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, pages 285–295, 2001. (Cited on pages 116, 118, and 136.)

[109] J. Schäfer and K. Strimmer. A Shrinkage Approach to Large-Scale Covariance Matrix Estimation and Implications for Functional Genomics. *Statistical Applications in Genetics and Molecular Biology*, 4(1), Jan. 2005. ISSN 1544-6115, 2194-6302. doi: 10.2202/1544-6115.1175. (Cited on pages 85 and 86.)

[110] S. Schelter, C. Boden, and V. Markl. Scalable Similarity-Based Neighborhood Methods with MapReduce. *RecSys*, pages 163–170, 2012. (Cited on pages 116 and 118.)

[111] S. Schelter, U. Celebi, and T. Dunning. Efficient Incremental Cooccurrence Analysis for Item-Based Collaborative Filtering. In *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*, pages 61–72, 2019. (Cited on page 118.)

[112] S. Schelter, S. Grafberger, S. Guha, O. Sprangers, B. Karlaš, and C. Zhang. Screening Native ML Pipelines with "ArgusEyes". In *CIDR: Conference on Innovative Data Systems Research*, 2021.

[113] R. Sen, H.-F. Yu, and I. S. Dhillon. Think Globally, Act Locally: A Deep Neural Network Approach to High-Dimensional Time Series Forecasting. In *Advances in Neural Information Processing Systems 32*, pages 4838–4847. Curran Associates, Inc., 2019. (Cited on pages 15 and 50.)

[114] A. Shanbhag, H. Pirk, and S. Madden. Efficient Top-K Query Processing on Massively Parallel Hardware. In *Proceedings of the 2018 International*

Conference on Management of Data, SIGMOD ’18, pages 1557–1570, New York, NY, USA, May 2018. Association for Computing Machinery. ISBN 978-1-4503-4703-7. doi: 10.1145/3183713.3183735. (Cited on page 147.)

[115] F. Sigrist. Gradient and Newton Boosting for Classification and Regression. *Expert Systems with Applications*, Oct. 2020. ISSN 0957-4174. doi: 10.1016/j.eswa.2020.114080. (Cited on page 53.)

[116] J. Song, Z. Li, Z. Hu, Y. Wu, Z. Li, J. Li, and J. Gao. PoisonRec: An Adaptive Data Poisoning Framework for Attacking Black-Box Recommender Systems. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 157–168. IEEE, 2020. (Cited on page 118.)

[117] O. Sprangers, R. Babuška, S. P. Nageshrao, and G. A. D. Lopes. Reinforcement Learning for Port-Hamiltonian Systems. *IEEE Transactions on Cybernetics*, 45(5):1017–1027, May 2015. ISSN 2168-2275. doi: 10.1109/TCYB.2014.2343194.

[118] O. Sprangers, S. Schelter, and M. de Rijke. Probabilistic Gradient Boosting Machines for Large-Scale Probabilistic Regression. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD ’21, pages 1510–1520, New York, NY, USA, Aug. 2021. Association for Computing Machinery. ISBN 978-1-4503-8332-5. doi: 10.1145/3447548.3467278. (Cited on pages 5 and 6.)

[119] O. Sprangers, S. Schelter, and M. de Rijke. Parameter-Efficient Deep Probabilistic Forecasting. *International Journal of Forecasting*, 39(1):332–345, Jan. 2023. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2021.11.011. (Cited on pages 5 and 6.)

[120] O. Sprangers, W. Wadman, S. Schelter, and M. de Rijke. Hierarchical Forecasting at Scale. *International Journal of Forecasting*, In Press, Mar. 2024. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2024.02.006. (Cited on pages 5, 6, and 7.)

[121] K. Stankeviciute, A. M. Alaa, and M. van der Schaar. Conformal Time-series Forecasting. In *Advances in Neural Information Processing Systems*, volume 34, pages 6216–6228. Curran Associates, Inc., 2021. (Cited on page 106.)

[122] E. Strubell, A. Ganesh, and A. McCallum. Energy and Policy Considerations for Modern Deep Learning Research. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(09):13693–13696, Apr. 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i09.7123. (Cited on page 32.)

[123] R. S. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts, nachdruck edition, 2014. ISBN 978-0-262-19398-6. (Cited on page 147.)

[124] S. J. Taylor and B. Letham. Forecasting at Scale. *The American Statistician*, 72 (1):37–45, Jan. 2018. ISSN 0003-1305. doi: 10.1080/00031305.2017.1380080. (Cited on pages 12 and 50.)

[125] F. Theodosiou and N. Kourentzes. Forecasting with Deep Temporal Hierarchies, Sept. 2021. (Cited on pages 4, 80, and 83.)

[126] A. Touloumis. Nonparametric Stein-type Shrinkage Covariance Matrix Estimators in High-Dimensional Settings. *Computational Statistics & Data Analysis*, 83:251–261, Mar. 2015. ISSN 01679473. doi: 10.1016/j.csda.2014.10.018.

(Cited on page 86.)

[127] M. Tsagkias, T. H. King, S. Kallumadi, V. Murdock, and M. de Rijke. Challenges and Research Opportunities in Ecommerce Search and Recommendations. In *ACM SIGIR Forum*, volume 54, pages 1–23. ACM New York, NY, USA, 2021. (Cited on page 118.)

[128] UCI. PEMS-SF Data Set. https://archive.ics.uci.edu/ml/datasets/PEMS-SF, 2009. (Cited on page 22.)

[129] UCI. Electricity LoadDiagrams20112014 Data Set. https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014, 2014. (Cited on page 21.)

[130] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. In *The 9th ISCA Speech Synthesis Workshop, Sunnyvale, CA, USA, 13-15 September 2016*, page 125. ISCA, 2016. (Cited on pages 5, 13, 14, 15, 17, and 23.)

[131] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. (Cited on pages 2, 11, 12, 15, 17, and 30.)

[132] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. Van Der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. Van Mulbregt, SciPy 1.0 Contributors, A. Vijaykumar, A. P. Bardelli, A. Rothberg, A. Hilboll, A. Kloeckner, A. Scopatz, A. Lee, A. Rokem, C. N. Woods, C. Fulton, C. Masson, C. Häggström, C. Fitzgerald, D. A. Nicholson, D. R. Hagen, D. V. Pasechnik, E. Olivetti, E. Martin, E. Wieser, F. Silva, F. Lenders, F. Wilhelm, G. Young, G. A. Price, G.-L. Ingold, G. E. Allen, G. R. Lee, H. Audren, I. Probst, J. P. Dietrich, J. Silterra, J. T. Webber, J. Slavič, J. Nothman, J. Buchner, J. Kulick, J. L. Schönberger, J. V. De Miranda Cardoso, J. Reimer, J. Harrington, J. L. C. Rodríguez, J. Nunez-Iglesias, J. Kuczynski, K. Tritz, M. Thoma, M. Newville, M. Kümmerer, M. Bolingbroke, M. Tartre, M. Pak, N. J. Smith, N. Nowaczyk, N. Shebanov, O. Pavlyk, P. A. Brodtkorb, P. Lee, R. T. McGibbon, R. Feldbauer, S. Lewis, S. Tygier, S. Sievert, S. Vigna, S. Peterson, S. More, T. Pudlik, T. Oshima, T. J. Pingel, T. P. Robitaille, T. Spura, T. R. Jones, T. Cera, T. Leslie, T. Zito, T. Krauss, U. Upadhyay, Y. O. Halchenko, and Y. Vázquez-Baeza. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, Mar. 2020. ISSN 1548-7091, 1548-7105. doi: 10.1038/s41592-019-0686-2. (Cited on page 90.)

[133] R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka. A Multi-Horizon Quantile Recurrent Forecaster. *31st Conference on Neural Information Processing Systems (NIPS2017), Time Series Workshop. Long Beach, CA, USA.*, June 2018. (Cited on pages 23 and 24.)

[134] S. L. Wickramasuriya, G. Athanasopoulos, and R. J. Hyndman. Optimal Forecast Reconciliation for Hierarchical and Grouped Time Series Through Trace Minimization. *Journal of the American Statistical Association*, 114(526):804–819, Apr. 2019. ISSN 0162-1459, 1537-274X. doi: 10.1080/01621459.2018.1448825. (Cited on pages 4, 81, 83, 84, 86, and 93.)

[135] C.-M. Wong, F. Feng, W. Zhang, C.-M. Vong, H. Chen, Y. Zhang, P. He, H. Chen, K. Zhao, and H. Chen. Improving Conversational Recommendation System by Pretraining on Billions Scale of Knowledge Graph. *ICDE*, 2021. (Cited on page 118.)

[136] X. Xie, F. Sun, X. Yang, Z. Yang, J. Gao, W. Ou, and B. Cui. Explore User Neighborhood for Real-time E-commerce Recommendation. *ICDE*, 2021. (Cited on page 118.)

[137] H. Yin, Q. Wang, K. Zheng, Z. Li, J. Yang, and X. Zhou. Social Influence-Based Group Representation Learning for Group Recommendation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 566–577. IEEE, 2019. (Cited on page 118.)

[138] F. Yuan, A. Karatzoglou, I. Arapakis, J. M. Jose, and X. He. A Simple Convolutional Generative Network for Next Item Recommendation. *WSDM*, pages 582–590, 2019. (Cited on page 119.)

[139] M. Zamo and P. Naveau. Estimation of the Continuous Ranked Probability Score with Limited Information and Applications to Ensemble Weather Forecasts. *Mathematical Geosciences*, 50(2):209–234, Feb. 2018. ISSN 1874-8953. doi: 10.1007/s11004-017-9709-7. (Cited on page 61.)

[140] Y. Zheng, C. Gao, X. He, Y. Li, and D. Jin. Price-Aware Recommendation with Graph Convolutional Networks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 133–144. IEEE, 2020. (Cited on page 118.)

[141] X. Zhou, D. Qin, X. Lu, L. Chen, and Y. Zhang. Online Social Media Recommendation over Streams. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 938–949. IEEE, 2019.

[142] Z. Zolaktaf, R. Babanezhad, and R. Pottinger. A Generic Top-n Recommendation Framework for Trading-off Accuracy, Novelty, and Coverage. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 149–160. IEEE, 2018. (Cited on page 118.)

Summary

In this thesis we investigate the forecasting problem for large-scale settings: how can we efficiently and accurately generate forecasts when we need to generate many of them?

Motivated by the increasing availability of large volumes of data and the ever increasing popularity of neural network models we investigate how we can improve the efficiency and accuracy of neural network models for point- and probabilistic forecasting in Chapter 2. We find that we can achieve better forecasting accuracy whilst reducing resource consumption – leading to reduced operational costs – by designing a neural network that requires an order of magnitude fewer parameters as compared to existing neural network probabilistic forecasting models. However, we also see that outside academia, there is a different class of models that is commonly used to solve the point- and probabilistic forecasting problem at a larger scale.

Thus, in Chapter 3, we investigate a similar question, but for a different class of models: how can we improve the efficiency and accuracy of Gradient Boosting Machines (GBM) models for point- and probabilistic forecasting? We propose *Probabilistic Gradient Boosting Machines* (PGBM), a method to create probabilistic predictions with a single ensemble of decision trees in a computationally efficient manner. We empirically demonstrate the advantages of PGBM compared to existing comparable state-of-the-art methods and find that PGBM can produce probabilistic estimates without compromising on point performance, using a single model only, thereby greatly improving forecasting efficiency as compared to existing methods.

In Chapter 4 we investigate the problem of hierarchical forecasting, which is the forecasting problem where time series need to adhere to a cross-sectional or temporal hierarchy, for example product groupings at a grocery store. We find that existing hierarchical forecasting techniques scale relatively poorly to large-scale problem settings and propose to learn a coherent forecast for millions of time series with a single bottom-level forecast model by using a loss function that directly optimizes

the hierarchical structure. This reduces the computational cost of the prediction phase in the forecasting pipeline, as well as its deployment complexity, whilst maintaining forecasting accuracy. We demonstrate the benefit in an offline test at *bol*, and show forecast improvements of up to 10%.

Finally, in Chapter 5, we study another forecasting problem often encountered in industry: session-based recommendation. We investigate state-of-the-art methods for session-based recommendation and find that the most simple method gives the most accurate results. We propose Vector-Multiplication-Indexed-Session kNN (VMIS-kNN), an adaptation of a state-of-the-art nearest neighbor approach to session-based recommendation, which leverages a prebuilt index to compute next-item recommendations with low latency in scenarios with hundreds of millions of clicks to search through. Based on this approach, we design and implement a scalable session-based recommender system *Serenade*, which is in production usage at *bol*.

Samenvatting

In dit proefschrift onderzoeken we het voorspellingsprobleem voor grootschalige omgevingen: hoe kunnen we efficiënt en nauwkeurig voorspellingen maken als we er veel moeten genereren?

Gemotiveerd door de toenemende beschikbaarheid van grote hoeveelheden data en de steeds toenemende populariteit van neurale netwerkmodellen onderzoeken we in Hoofdstuk 2 hoe we de efficiëntie en nauwkeurigheid van neurale netwerkmodellen voor punt- en probabilistische voorspellingen kunnen verbeteren. We stellen vast dat we een betere voorspellingsnauwkeurigheid kunnen bereiken en tegelijkertijd het energie- en geheugenverbruik kunnen verminderen – wat leidt tot lagere operationele kosten – door een neuraal netwerk te ontwerpen dat een orde van grootte minder parameters vereist in vergelijking met bestaande probabilistische voorspellingsmodellen die gebruik maken van neurale netwerken. We zien echter ook dat er buiten de academische wereld een andere klasse modellen bestaat die gewoonlijk wordt gebruikt om het punt- en probabilistische voorspellingsprobleem op grotere schaal op te lossen.

Daarom onderzoeken we in Hoofdstuk 3 een soortgelijke vraag, maar voor een andere klasse modellen: hoe kunnen we de efficiëntie en nauwkeurigheid van Gradient Boosting Machines (GBM)-modellen voor punt- en probabilistische voorspellingen verbeteren? We introduceren *Probabilistic Gradient Boosting Machines* (PGBM), een methode om probabilistische voorspellingen te maken met een enkel ensemble van beslissingsbomen op een computationeel efficiënte manier. We demonstreren empirisch de voordelen van PGBM vergeleken met bestaande vergelijkbare methoden en ontdekken dat PGBM probabilistische schattingen kan produceren zonder concessies te doen aan de puntprestaties, gebruikmakend van slechts een enkel model, waardoor de voorspellingsefficiëntie aanzienlijk wordt verbeterd in vergelijking met bestaande methoden.

In Hoofdstuk 4 onderzoeken we het probleem van hiërarchische voorspellingen, het voorspellingsprobleem waarbij tijdreeksen gebonden zijn aan een (temporele) hiërarchie, zoals bijvoorbeeld productgroeperingen in een supermarkt. We constateren dat bestaande hiërarchische voorspellingstechnieken relatief slecht kunnen worden geschaald naar grootschalige probleemsituaties en stellen voor om een hiërarchisch coherente voorspelling voor miljoenen tijdreeksen te leren met een enkel voorspellingsmodel op het laagste niveau door gebruik te maken van een verliesfunctie die de hiërarchische structuur direct optimaliseert. Dit vermindert de rekenkosten van de voorspellingsfase in de voorspellingspijplijn, evenals de complexiteit ervan, terwijl de voorspellingsnauwkeurigheid behouden blijft. We demonstreren het voordeel in een offline test bij *bol* en laten voorspelde verbeteringen zien tot wel 10%.

Tenslotte bestuderen we in Hoofdstuk 5 een ander voorspellingsprobleem dat vaak voorkomt in de praktijk: op online-sessies gebaseerde aanbevelingen. We onderzoeken de modernste methoden voor sessiegebaseerde aanbevelingen en komen tot de conclusie dat de meest eenvoudige methode de meest nauwkeurige resultaten oplevert. We introduceren Vector-Multiplication-Indexed-Session kNN (VMIS-kNN), een aanpassing van een bestaande *kNN*-methode voor sessiegebaseerde aanbevelingen. VMIS-kNN maakt gebruik van een vooraf gebouwde index om aanbevelingen voor het volgende item te berekenen met lage latentie in scenario's met honderden miljoenen *clicks* om doorheen te zoeken. Op basis van deze aanpak ontwerpen en implementeren we een schaalbaar, op sessies gebaseerd aanbevelingssysteem *Serenade*, dat in productie is bij *bol*.