CHALMERS
UNIVERSITY OF TECHNOLOGY

AI Enabled Service Market Logistics

Forecasting Supplier Delivery Performance with Recurrent Neural Networks

Master's thesis in Engineering Mathematics and Computational Science

JOHAN RAMNE

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:NN

AI Enabled Service Market Logistics

Forecasting Supplier Delivery Performance
with Recurrent Neural Networks

JOHAN RAMNE

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2020

AI Enabled Service Market Logistics
Forecasting Supplier Delivery Performance with Recurrent Neural Networks
JOHAN RAMNE

Supervisors, Volvo Group: Iulian Carpatorea and Jesper Holst
Supervisor and Examiner, Chalmers: Annika Lang, Mathematical Sciences

Master's Thesis 2020:NN
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A Polar bear in a snowstorm.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

AI Enabled Service Market Logistics
Forecasting Supplier Delivery Performance with Recurrent Neural Networks
JOHAN RAMNE
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Uncertainty about upstream suppliers' ability to deliver ordered quantities on time is one the reasons that manufacturers and retailers need to keep safety stock in inventory. Through accurate prediction of suppliers' delivery performance the uncertainty can be quantified and used by material planners in their decision-making process. Representing the deliver performance of an individual supplier as a time series, the uncertainty can be predicted through probabilistic forecasting: estimation of the future probability distribution given past observations. This thesis presents two recurrent neural network models, using encoder-decoder architectures, for multi-step ahead probabilistic forecasting of the delivery performance of suppliers to Volvo Group Trucks Operations Service Market Logistics. The models are evaluated on mean quantile loss for a number of quantiles over a 14 week forecast range. One model, DeepAR, outperformed exponential smoothing models generated by the `forecast` package in R on four out of five quantiles.

Keywords: delivery performance, RNN, probabilistic forecasting, supply chain management, service market logistics, quantile recurrent neural network.

Acknowledgements

I would like to thank Jesper Holst and Iulian Carpatorea, together with the whole Advanced Analytics team, for their support, encouragement and feedback throughout the writing of this thesis. Your guidance helped me stay motivated and on course through these exceptional times. I also want to thank Annika Lang for all her help and guidance in putting this thesis together.

Johan Ramne, Gothenburg, May 2020

Contents

1

Introduction

1.1 Background

How inventory management drives a company's financial performance, ability to react to market signals, and customer satisfaction is well documented in the Supply Chain Management (SCM) literature (Timme, 2003; Jones and Riley, 1985; Gupta, Maranas, and McDonald, 2000). With the emergence in the 90s and 2000s of machine learning methods such as Support Vector Machines (Cortes and Vapnik, 1995) and Random Forests (Breiman, 2001) and the success of Convolutional Networks in image recognition tasks (Simonyan and Zisserman, 2015), Artificial Intelligence (AI) has become a topic of interest across academia and industry. Within SCM there is a growing interest in applying AI techniques, alongside other advanced analytics methods, to find better solutions to established problems in the field (Min, 2010; Carbonneau, Laframboise, and Vahidov, 2008), including inventory management (Bertsimas, Kallus, and Hussain, 2016).

Galbraith (1974) set the definition of uncertainty commonly used in SCM: "uncertainty [is] the difference between the amount of information necessary to perform a task and the information already possessed by the company". There are two main sources of uncertainty at any given point in a supply chain that affect inventory: the supplier ability to deliver the right quantity at the right time, and the customer demand downstream (Heydari, Kazemzadeh, and Chaharsooghi, 2009). This thesis will focus entirely on supply side uncertainty. In an early classification of supply chain uncertainty, T. Davis (1993) gave a number of reasons why suppliers can fail to deliver: bad weather delaying shipment, machine breakdowns or supply issues further up the chain. The list of possible issues grows exponentially with the complexity of the manufacturing process and as the supply network grows (Simangunsong, Hendry, and Stevenson, 2012).

The SCM literature on uncertainty management broadly falls into two categories: strategies for *coping* with uncertainty and strategies for *reducing* uncertainty (Simangunsong, Hendry, and Stevenson, 2012). Companies use a combination of these strategies to control uncertainty as cost-effectively as possible. Galbraith (1974) set out four fundamental methods for dealing with uncertainty in an organisation: creating slack resources, designing self-contained tasks, investing in vertical information systems and creating lateral relations. All four methods feature heavily in supply chain design and management. The most common manifestation of creating slack resources is to maintain a level of *safety stock* in inventory—additional stock beyond the expected demand (Rushton, Croucher, and Baker, 2014). A com-

mon example of vertically sharing information is for a company to share demand forecasts with their suppliers as guidance.

The two first methods decrease the need to process information in order to succeed, while the last two increase the capacity to process information (Galbraith, 1974). All four methods come with their own respective costs. Finding the right balance for each individual company is important and quantifying the different sources of uncertainty can help with making the right decision. T. Davis (1993) recommends quantifying a supplier's performance across several metrics, such as delivery performance:

$$\text{delivery performance} := \frac{\text{orders delivered on time with right quantity}}{\text{total number of orders}}, \tag{1.1}$$

which is the measure the models in Chapter 4 are trained to predict, as well as measure average tardiness and service rate and adjusting the safety stock as a coping strategy. Beyond these simple measures there has been a great deal of research into using advanced quantitative methods for modelling uncertainty in the supply chain (Peidro et al., 2009) with the goal of aiding in supply chain planning. This thesis' contribution to this field of research is the use of recurrent neural networks (RNNs) to predict the delivery performance of suppliers to Volvo Group Truck Operations' Service Market Logistics division with a forecast range of 14 weeks.

1.1.1 Company background

Volvo Group is a multinational company that manufactures and sells trucks under several brands—Volvo, Renault and Mack Trucks being the most important—as well as construction equipment and engines for marine and power-generation applications. Volvo Group Trucks Operations Service Market Logistics (SML) is responsible for supplying customers around the world with items from the large catalogue of spare parts necessary to keep Volvo Group's installed base functioning.

At the center of the SML supply chain in Europe lies the Central Distribution Center (CDC) in Gent, Belgium. All service parts ordered from suppliers are shipped to the CDC, which then takes orders from and delivers to Regional Distribution Centers (RDCs) and Support Distribution Centers (SDCs) around the continent. The supply chain has two primary functions: a *market mediation* function and a *physical* function (Fisher, 1997). The physical function of the supply chain covers the process of converting raw materials and components into finished goods as well as transportation and storage. The market mediation function matches supply to demand in terms of volume, variety and product requirements in an uncertain and dynamic environment (Herer, Tzur, and Yucesan, 2002).

These two functions are at the CDC roughly divided between three departments in SML. The Demand and Inventory Planning (DIP) team is responsible for market mediation together with the Continental Material Planning (CMP) department. DIP makes demand forecasts which the CMP department translates into orders from suppliers. With regards to the physical function the CDC is responsible for storage while transport is outsourced to third-parties.

In pursuit of reduced inventory costs, increased automation and improved customer service, SML has taken an active approach to using advanced analytics and AI in SCM. The Advanced Analytics (AdA) team within SML has been tasked with exploring how AI can be leveraged by the different functions in SML to achieve these three goals. They are also responsible for evaluating the business value of different solutions and implementing solutions as applications in production. The process projects in AdA follow, including to a large extent this thesis, is described in the method section and differs greatly from the conventional workflow at Volvo Group.

1.1.2 Service Market Logistics

The goal of the SML division is to maintain optimal uptime of the installed base with no unplanned stops and their responsibility is to supply dealers around the world with the parts necessary for keeping the fleet of sold Volvo Trucks rolling and Penta machines running. The standard sales agreement from Volvo binds them to make all required spare parts available for a sold truck for 15 years.

Supplying spare parts is a high-margin business of strategic interest to Volvo. The Volvo Truck brand is marketed toward the premium heavy-duty segment where customers require high quality service. Part of Volvo's value proposition to customers is easy access to any necessary spare part, no matter where in the world the truck is. A higher sale price compared to competitors can be justified if the total cost of ownership is lower due to better uptime and a longer lifespan.

Maintaining a spare part supply chain is, in certain aspects, more challenging than maintaining its production counterpart. The chief among these is that spare parts exhibit intermittent and unpredictable demand by nature—while many parts such as filters, brake-pads and bearings are replaced semi-regularly, most parts are not expected to be changed over a trucks lifespan (Cohen, N. Agrawal, and V. Agrawal, 2006). Supplying spare parts also differs from the production supply chain in distribution of consumed volume. In production, the consumption of any specific part will see a predictable decline to zero as the last model using that part is phased out of production. In the aftermarket the consumption distribution for any specific part generally has a long "fat tail"—low, intermittent demand often lasts for many years after the peak is passed. Cumulatively, the demand distribution follows a Pareto distribution, as demand rates vary significantly (Muckstadt, 2005).

1.1.3 Advanced Analytics

Advanced Analytics is a team within the SML organisation that is responsible for generating valuable insights with the use of analytical methods and creating value through their application. They explore possible improvements in all aspects of Volvo Group's service market apparatus with examples being automating repetitive tasks, improving forecasting or making data more readily available to all parts of the organisation. Most work is done in close collaboration with other teams and divisions and following their Exploratory Development framework, which lays out the process by which an idea becomes a finished product. The Exploratory Development is heavily inspired by the Agile Software movement (Beck et al., 2001), especially

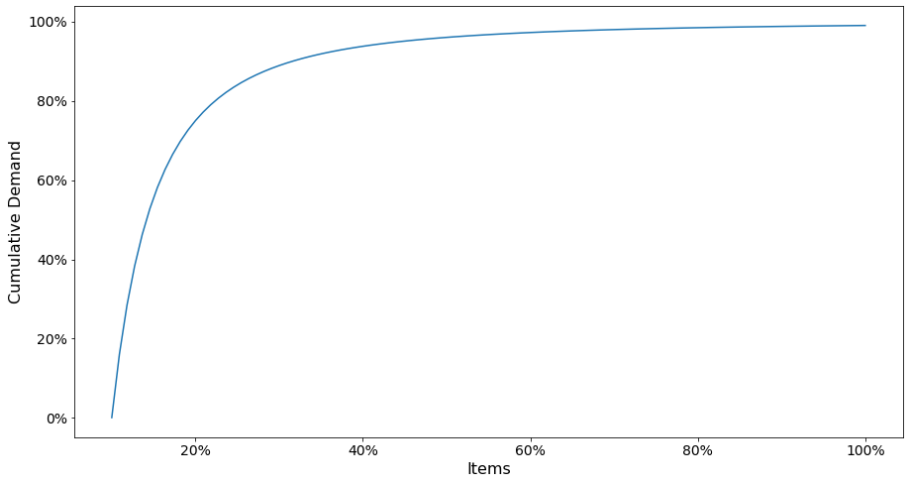

Figure 1.1: The cumulative demand by highest turnover items.

Scrum (Schwaber and Sutherland, 2017). The Exploratory Development framework is detailed in chapter 2 and the development of the supplier performance forecaster will as closely as possible follow it.

1.2 Problem formulation and research questions

In 2019 the SML organisation had a target of 95% delivery performance (DP) for the contracted suppliers, where DP is measured as the percentage of deliveries that are made available for transport on the ordered week. However, according to Huang and Bagheri (2019) the actual delivery performance was closer to 77%. These deviations are problematic because they can lead to stockouts at the CDC, which in turn can lead to backorders.

Backorders are costly in two aspects to the SML organisation and Volvo Group as a whole. First, if the spare part is critical to getting a vehicle or machine back in operation its unavailability will incur a running cost on the customer until it is solved. This cost is often significant and thus potential customers will take the availability and expediency of service into consideration when choosing between Volvo Group's brands and their competitors. SML will in many instances pay for expensive air freight to get a vehicle back on the road as quickly as possible when the needed spare part becomes available.

To guard against uncertainty, on both the supply and the demand side, a safety stock is kept for most items at the CDC. The recommendation of T. Davis (1993) to adjust safety stock of an item according to a set of performance metrics on its supplier makes the implicit assumption that historical data on delivery performance has predictive power. By making this assumption explicit, a forecasting model can be designed that specifies a belief about the supplier's delivery performance in the future. In the framework of Galbraith (1974) building this forecasting model increases the organisation's information processing capacity and could thus allow for adjustments of the slack resources—the safety stock.

The data available to build this forecast model is the delivery performance of all

suppliers to SML, on a week-by-week basis, over a multi-year period. Each supplier's delivery performance is its own time series and the problem lies in forecasting all of them separately and accurately. To be useful the forecast model's horizon must be longer than any given supplier's delivery lead time for the SML division to be able to be proactive. For simplicity the forecast horizon is set to 14 weeks for all time series.

The use of artificial neural networks (ANNs) has shown great promise for forecasting problems involving large numbers of time series (Smyl, 2020; Hewamalage, Bergmeir, and Bandara, 2019) and will be the foremost method investigated. With this in mind, we set the following research questions are:

RQ 1: *What forecasting model and method should be used to create multi-step ahead forecasts of supplier's delivery performance?*
RQ 2: *How can these forecasts be leveraged to create value?*
RQ 3: *How applicable is the AdA Exploration Development framework to developing neural network models?*

1.3 Scope and structure

This thesis is limited to examining the performance of the suppliers delivering spare parts for Volvo Trucks and Volvo Penta. Further, the investigation is limited to delivery performance towards the CDC in Ghent, Belgium, and does not look beyond the first tier of suppliers.

The rest of the thesis is structured as follows. Chapter 2 will present the Exploratory Development framework, developed by the AdA, and describe how its individual ideas and concepts connect with the broader literature on Agile software development. Chapter 3 gives a mathematical background on time series forecasting with a focus on forecasting with neural networks. Chapter 4 contains the description of two RNN forecasting models—MQRNN and DeepAR—along with their specific approaches, architectures and training schemes. Chapter 5 presents an evaluation of these models on supplier delivery performance data, a comparison with two baseline methods, business implications and conclusions.

2
Method

In order to answer *RQ 3* the thesis will, to the greatest extent possible, follow the AdA way of working. This will be done by adhering to the Exploratory Development framework and by participating in AdA's PulseLab, as described in the next section.

2.1 Agile methods within Advanced Analytics

The work within AdA is in many ways closer related to the scientific research than to software engineering, especially in the early stages of testing the viability of an idea. The issues encountered are similar to those described by Sletholt et al. (2011) in scientific computing development: the exploratory nature makes the elicitation and specification of requirements difficult. When this is the case the traditional waterfall-development model is generally unproductive.

Within the AdA team a way of working has been developed, based in large part on Agile methodologies, called the *Exploratory Development* framework. The framework is intended to promote creativity early in the development cycle, let unsuccessful ideas fail quickly, and scale out successful applications to the larger organization. The concept is illustrated in Figure 2.1. Campanelli and Parreiras (2015) aptly point out that software development methods are often defined under the assumption that they will work in any environment, which has not been the case in reality. Using out-of-the-box methodologies as described in a single book or prescribed by experts is often unsuccessful—organisations have their own distinct context that they are operating in and their own particular set of goals and challenges. Adoption is often piecemeal from the different flavors of Agile (e.g. Extreme Programming (XP), Scrum, Kanban, Lean) depending on needs and current processes (Kurapati, Manyam, and Petersen, 2012). The methods need to be chosen and tailored to fit the organisation to be beneficial.

The core values of the Agile Manifesto (Beck et al., 2001) working software, customer collaboration, and responding to change, and important principles such as self-organizing teams and continuous improvement, all align with the values at the heart of the AdA team. Working in the automotive industry the Toyota Production System (Liker, 2004) and its Western derivative Lean Manufacturing have also had an enormous influence on the design of processes and management at AdA and within Volvo Group at large. An example is the global set of processes called the Volvo Production System (VPS) designed to facilitate continuous improvement at all levels in the organisation (Volvo Group, 2017).

With that in mind it is clear that some of the ideas and methods within the

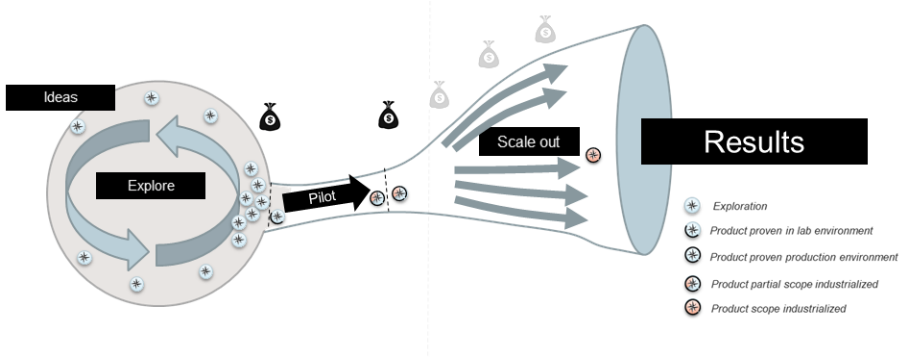

Figure 2.1: The AdA Exploration Development Framework.

Lean Software Development movement (M. Poppendieck and T. Poppendieck, 2003), which apply the industrial concepts of lean manufacturing to agile software development, have also influenced the Exploratory Development framework. Of particular note is that the AdA team has experimented with Kanban techniques (Ikonen et al., 2010) to visualize work.

The next section will introduce the Exploratory Development framework and attempt to link methods and principles to their origins in the agile literature. It is worth noting that, as the framework is built on the experience of the team members together with institutional knowledge and previous processes, these links will to certain extent be post-rationalizations as the original inspirations are unknown. The framework will be followed by a description of the Pulselab format, which is one way in which AdA practices agile development.

2.1.1 Exploratory Development

The Exploratory Development framework, illustrated in Figure 2.1 consists of three distinct, gated, stages: the *explore*, *pilot*, and *scale out* stages and is designed to drive ideas towards becoming products that generate business value for Volvo Group.

2.1.1.1 Explore stage

The explore stage is designed to be as conducive as possible to creative idea generation and problem solving. New ideas can come from within the AdA team or from an outside stakeholder in a different team within the SML organisation. An exploration is initiated by an *exploration lead*, based on an idea deemed worthy of looking into. Worthiness has in the past been judged on case-by-case basis taking into account priorities set higher up in the organisation together with the enthusiasm and confidence the exploration lead has for the idea.

The exploration lead takes responsibility for identifying the required competencies and driving the exploration forward. Exploration teams are composed of a cross-functional group of people, supply chain experts, data scientists, technology managers, who are able to make most decisions regarding the exploration at this stage without hierarchical bureaucracy. This follows the agile principle of empowering teams and avoiding spending time writing specifications upfront (Deemer et al.,

2010).

Each exploration has stated goals that are designed to be achievable within a short time frame, generally within one to four weeks. If the exploration is done as a part of a PulseLab, the time frame is set to four days. At this stage it is not strictly necessary that the exploration is justified by some future business value if it succeeds—examples of other stated goals include learning a new technology, expanding the team's knowledge base or mapping previously unexplored data generated somewhere in the organisation. That said, where possible the exploration lead should attempt to quantify potential business value. Depending on if the exploration has reached its goals within the given time frame the exploration lead, in conjunction with managers, product sponsor and developers, decides between iterating on the idea by running a new exploration, moving forward to the pilot stage with a potential product, or closing the exploration.

The explore stage has an iterative structure that follows the idea of *timeboxes* in agile development, championed prior to the Agile movement by Martin (1991) in the book Rapid Application Development. Iteration is a part of many agile methodologies but the structure of the explore stage is most akin to early Scrum's sprint concept (Schwaber, 1995)—within Pulselabs the duration of an exploration is explicitly called a sprint—in that the explorations are by nature somewhat unpredictable. Choosing to discontinue the development of ideas that do not show sufficient promise early, to minimize sunk costs and maximize the teams overall performance, is in line with the "Fail Fast" philosophy famously championed by Reiss (2011) in his book The Lean Startup.

Through early and direct involvement of product sponsors and stakeholders, the framework adheres to the agile values of "Customers collaboration over contract negotiation" and welcoming changes (Beck et al., 2001). Each exploration is documented through write-ups available to the whole team to to facilitate continuous improvement and amplifying learning, following lean software development principles (M. Poppendieck and T. Poppendieck, 2003).

2.1.1.2 Pilot and scale out stages

The pilot stage follows after one or more iterations of the explore stage. When the exploration lead together with the product sponsor deem the exploration of sufficient maturity and potential business value, it is labelled as a potential product and warrants more resources and more formal governance. Although agile principles still provide the foundation for the way of working, the governance is starting to resemble the traditional waterfall model. This transition is necessary to make project management compatible with the other parts of the organisation that are involved or impacted. The project now flows through discrete steps without moving backwards, see Figure 2.2.

The first step after the final exploration is to define a potential product. At this point scope and reasoning behind the product should be fully fleshed out, together with an estimate of the business value. To a certain extent the lean principle of "Decide as Late as Possible" (M. Poppendieck and T. Poppendieck, 2003) is still employed, but a majority of the design decisions will most likely have been made. If the project has met all its stated goals and criteria, and the project sponsor is fully

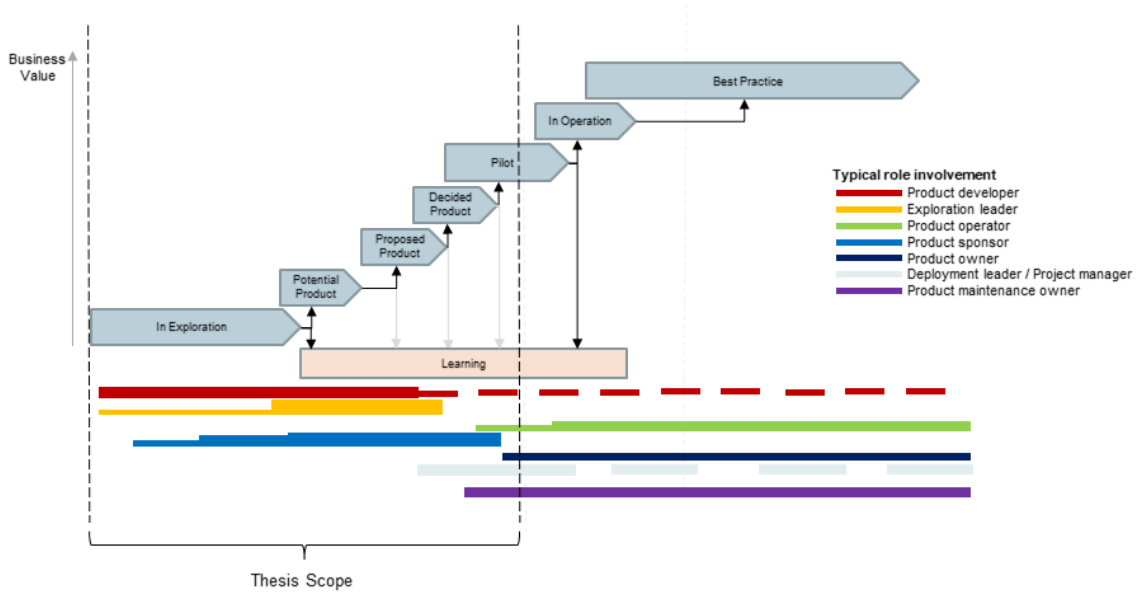

Figure 2.2: The steps of the pilot stage together with the different stakeholders level of involvement.

onboard, it is moved forward as a proposed product, otherwise it is archived.

In the next step, the now proposed product's features and requirements are solidified, while the system dependencies, requirements, impact on current IT and stakeholders are mapped. This step brings in other parts of the organisation, generally including Volvo Group IT, thus requires even stricter governance, and is run like a traditional IT project. When these things are in place, a decision is made to go forward and develop and test the product in a pilot environment.

An example of a product and pilot environment could be a warehouse inventory tool that is initially developed for the CDC in Ghent. The tool can be tested at this single warehouse and if the pilot project is successful, the product can be deployed into operational environments. Now, work transitions to applying or modifying the product to function in other parts of the organisation in order to maximize to business value and utility. In this example that would involve modifying the tool to work with the systems that are in place in other warehouses around the world.

2.1.2 PulseLab

Every four weeks the AdA team organizes *PulseLab*, a four-day Scrum-based event. At the initiation of the PulseLab anyone within Volvo Group Operations, not just members of the AdA team, can present an idea and concept that they wish to explore. When all prospective projects have been presented, participants choose freely between them based entirely on what they find interesting. For the next four days the teams meet for a Scrum-fashioned standup (Stray, Sjøberg, and Dybå, 2016) to report on progress and if necessary try and recruit new project members if they've discovered a new required competence. On the final day, groups present their results and conclusions as well as their thoughts on the next steps.

The PulseLab is intended to bring several benefits. Firstly, it is a tool for

outreach from AdA to the rest of the organisation. Many are interested in how machine learning and AI can be used in their particular field, or have ideas but lack the competencies to execute upon them. PulseLab is a platform to bring in novel ideas and simultaneously broadcast out how advanced analytics can be used. Secondly, while participants are allowed to choose freely between explorations, they are also encouraged to try new roles and tasks. Examples can be a line employee learning Python under the guidance of a data scientist, or an intern practicing project management. This amplifies learning and improves communication between the different roles outside of the PulseLab. Finally, PulseLabs are a way of striking while the iron is hot—a new idea will at most go five weeks from inception to exploration. Outside of the time allocated to PulseLabs, people have schedules that are full for months on end. In a traditional working scheme it may take months, or even years, before an idea is tested which dampens the enthusiasm, decreases innovation and leads to missed opportunities.

The PulseLab format has no easy to find analog in the Agile nomenclature, although it undoubtedly embraces Agile principles and values (Beck et al., 2001). To a certain extent it is similar to an in-house hackathon as described by Tucker (2014) with the short deadline and final presentations, but generally a hackathon is a competition with an overarching theme, problem or goal that the groups focus on. The PulseLab also takes the agile principle of self-organizing teams (Hoda, Noble, and Marshall, 2013), in which the team members manage their own workload, shift tasks and make collective decisions, to the next step in the form of *liquid teams*, where participants in the PulseLab can move between explorations based on need. The idea of liquid teams is possibly based on the concept of the "Liquid Workforce" touted by Accenture, a consultancy firm.

Recently, the overall success of PulseLabs has made AdA work to further incorporate agile ways of working in their week-to-week activities as well as bring in lessons learned. Standouts are the introduction of a weekly standup meeting, mirroring the daily standups in the PulseLab, and biweekly demo-sessions, analogous to the final presentation in the PulseLabs. The demo-sessions are especially interesting sessions as they will act as a mechanism promoting working software, one of the core values of the Agile movement (Beck et al., 2001).

2.1.3 Thesis work

The thesis project is explorative by design and there is a natural high risk of failure for any given approach to the stated problem. The work follows the Exploratory Development framework as closely as possible, with necessary adjustments to ensure that the produced thesis adheres to Chalmers specifications for academic rigour.

3

Mathematical background

3.1 Time series forecasting

Time series forecasting using statistical methods is actively researched field that has been an academic discipline for nearly 100 years although forecasting as an activity stretches far further back in history (Tsay, 2000; Yule, 1927). The following section will briefly introduce important traditional parametric forecasting methods. This will be followed by an introduction to forecasting competitions and their place in the further development of the field. With this framework in place we will rephrase forecasting as a machine learning problem and review how researchers are using cutting edge machine learning methods to get state-of-the-art results.

Time series forecasting is one of the central prediction tasks within data analytics with a wide range of applications (Brockwell and R. A. Davis, 2016). We will rely on the definition of *time series models* given by Brockwell and R. A. Davis (2016) and adapted by Lang and Petersson (2017):

Definition 3.1.1. A *time series* is real-valued sequence of observations $(\tilde{x}_t, t \in \mathbb{T})$ with respect to some index set $\mathbb{T} \subseteq \mathbb{R}$. A *time series model* for the observed data $(\tilde{x}_t, t \in \mathbb{T})$ is a specification of the joint distributions of a sequence of random variables $(x_t, t \in \mathbb{T})$ of which $(\tilde{x}_t, t \in \mathbb{T})$ postulates to be the realization.

Throughout this thesis the realization of the random variable x_t will be denoted $\tilde{x}_t$ and estimates will be denoted $\hat{x}_t$. Let Ψ_{t-1} be the information set available at t; generally Ψ_{t-1} is the σ field generated by by past observations of x (Tsay, 2000). A time series model can be written as

$$x_t = F(\Psi_{t-1}) + \epsilon_t \tag{3.1}$$

where $F : \Psi \to \mathbb{R}$ is a time series model and $\epsilon_t \sim \mathrm{WN}(0, \sigma_\epsilon)$ is a white noise process with $\sigma_\epsilon < \infty$; the ϵ_t is the *forecast error* at t. A classic example of a forecasting problem is predicting next month's demand for a consumer product, like tennis shoes or sports cars. But in many applications we are interested in forecasting over a range of equidistant steps into the future at once:

$$(x_t, x_{t+1}, \ldots, x_{t+K}) = F(\Psi_{t-1}) + \boldsymbol{\epsilon}_t, \tag{3.2}$$

where K is the forecasting horizon and $\boldsymbol{\epsilon}_t = [\epsilon_t, \ldots, \epsilon_{t+K}]$ is a vector of forecast errors. In an industrial setting having a sizeable forecast horizon can be important because of production lead time—adjusting to changes in demand takes time.

Another frequently encountered situation in time series forecasting is the need to predict a vector-valued process. This could be the (x, y, z)-coordinates of a satellite in motion, or the amount of time a person will spend on a set of N common activities in a day (e.g. brush their teeth, cook lunch etc.). Let $\boldsymbol{x} = (\boldsymbol{x}_t \in \mathbb{R}^{\mathbb{N}}, t \in \mathbb{T})$ denote a vector-valued time series and $\tilde{\boldsymbol{x}} = (\tilde{\boldsymbol{x}}_t \in \mathbb{R}^{\mathbb{N}}, t \in \mathbb{T})$ its realizations. The multivariate forecasting problem concerns specifying the model $F : \Psi \to \mathbb{R}^N$ where

$$\boldsymbol{x}_t = F(\Psi_{t-1}) + \boldsymbol{\epsilon}_t \tag{3.3}$$

with $\boldsymbol{\epsilon}_t = [\epsilon_{t,1}, \ldots, \epsilon_{t,N}]$. Problems (3.2) and (3.3) can be combined to produce a multivariate multi-step ahead problem of estimating the model $F : \Psi \to \mathbb{R}^{N \times K}$:

$$(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_{t+K}) = F(\Psi_{t-1}) + (\boldsymbol{\epsilon}_t, \boldsymbol{\epsilon}_{t+1}, \ldots, \boldsymbol{\epsilon}_{t+K}), \tag{3.4}$$

where $\boldsymbol{x}_{t+k} \in \mathbb{R}^N$ and $\boldsymbol{\epsilon}_{t+k} \in \mathbb{R}^N$ for $k \in \{0, \ldots, K\}$. Using a previous example, this could arise when trying to predict the amount of time a person spends on N common activities for each of the K coming days. Both models introduced in 4 will be of this type.

3.1.1 Multi-step ahead forecasting

Although much of the academic forecasting literature has focused on one-step ahead forecasting in the form of (3.1) and (3.3) (Box and Jenkins, 1970; Brockwell and R. A. Davis, 2016; Hyndman and Athanasopoulos, 2018) there is a rising interest in forecasting multiple steps ahead (Sorjamaa et al., 2007; Taieb, Sorjamaa, and Bontempi, 2010). This can be seen in the structure of benchmark forecasting competitions that have been held in the past decades such as the NN3 competition (Crone, Hibon, and Nikolopoulos, 2011) and the M Competitions (Makridakis, Spiliotis, and Assimakopoulos, 2020) focusing on multi-step ahead forecasts. There are two overarching approaches to multi-step forecasting, *iterative* and *direct* forecasts, with respective strengths and drawbacks.

3.1.1.1 Iterated method

Creating multi-step ahead forecasts through iteration is the oldest technique developed in the forecasting literature (Sorjamaa et al., 2007). The idea is relatively simple: iteratively use one-step ahead predictions as inputs into the same model, as proxies for the true observations, to get K predictions. In the univariate case

$$\hat{x}_{t+k} = \begin{cases} F(\Psi_{t-1}) & \text{for } k = 0 \\ F(\Psi_{t-1}; \hat{x}_t, \ldots, \hat{x}_{t+k-1}) & \text{for } k = 1, \ldots, K \end{cases} \tag{3.5}$$

The iterated method is intuitive and easy to implement but has the downside of being prone to bias (Marcellino, Stock, and Watson, 2006). Any misspecification in in the one-step model will be exasperated over time which can lead to large errors.

3.1.1.2 Direct method

In the direct method the time series model over the forecast range $\{0, \ldots, K\}$ is specified by K separate functions

$$\hat{x}_{t+k} = F_k(\Psi_{t-1}) \quad \text{for } k \in \{0, \ldots, K\}$$

where each function is built or trained separately from the others to minimize the error at its specified time-step (Taieb, Sorjamaa, and Bontempi, 2010).

The direct method avoids the issue of errors accumulating that exists in the iterated method. The downside is that there is no temporal link between the forecasted values, they are necessarily assumed conditionally independent, which can have consequences for the prediction accuracy. Model parameter estimation is less efficient in the direct method than in the iterated method and thus generally requires a large dataset to perform well (Marcellino, Stock, and Watson, 2006).

Multiple output method

One variant of the direct method, utilized by one of the models described in the 4, is the multiple output method. It muddies the distinction between the forecasting problem in (3.2) and the one in (3.3) by viewing $x_{t:t+K} := (x_t, x_{t+1}, \ldots, x_{t+K})$ as a $\mathbb{R}^K$ random vector to be forecast. In this method a model is trained to estimate this vector directly, which avoid the error propagation dilemma that the iterated method has but allows for a certain amount of interdependence between the forecasts. In general this method is implemented by some machine learning algorithm and requires a large training dataset to reduce the forecasting errors to a competitive level with the iterated and standard direct method.

3.2 Parametric forecasting methods

The classical parametric forecasting models are constructed around the assumption that the underlying time series is *weakly stationary*: the two first moments are time-invariant under translation. That is, the expectation $E[x_t] = \mu$ for some real constant μ and all t; and the lag covariance function $\gamma_l = \mathrm{Cov}(x_t, x_{t+k})$ is dependent only on k. Under the assumption of weak stationarity the autocorrelation function of a time series simply becomes $\rho_k = \gamma_k/\gamma_0$, which is particularly useful when modelling linear times series (Tsay, 2000). All following time series models have the implicit assumption that $\mu = 0$ for notational brevity—in practical applications where this is not the case a constant term is simply added.

3.2.1 Autoregressive and moving average models

The most intuitive model for forecasting a weakly stationary time series is the *autoregressive* model AR(p) which uses a linear combination of the previous p actuals to predict the value one step into the future

$$x_t = \sum_{i=1}^{p} \phi_i x_{t-i} + \epsilon_t \tag{3.6}$$

where $\phi_1, \ldots, \phi_p \in \mathbb{R}$ are the model weights and $\epsilon_t \sim \mathrm{WN}(0, \sigma_\epsilon)$. This was introduced by Yule (1927) together with *moving average* models MA(q), which model the future value as a function of the q random shocks

$$x_t = \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \tag{3.7}$$

with $\theta_1, \ldots, \theta_q \in \mathbb{R}$ and $\epsilon_{t-q}, \ldots, \epsilon_t \sim \mathrm{WN}(0, \sigma_\epsilon)$. Stationary stochastic processes often contain both an autoregressive component and a moving average component and are as such best modelled by combining the two into a *mixed autoregressive-moving average* model ARMA(p, q)

$$x_t = \epsilon_t + \sum_{i=1}^{p} \phi_i x_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \tag{3.8}$$

with $\phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q \in \mathbb{R}$ and $\epsilon_{t-q}, \ldots, \epsilon_t \sim \mathrm{WN}(0, \sigma_\epsilon)$.

3.2.2 ARIMA

Autoregressive integrated moving average (ARIMA) models were popularized for modelling nonstationary stochastic processes by Box and Jenkins (1970) together with the Box–Jenkins method, a systematic approach to times series identification, estimation and verification (Gooijer and Hyndman, 2006). Let B be the backward shift operator $Bx_t = x_{t-1}$. An ARIMA(p, q, d) model can be written as

$$w_t = (1 - B)^d x_t,$$
$$F(\Psi_{t-1}) = \sum_{i=1}^{p} \phi_i w_{t-i} - \sum_{j=1}^{q} \theta_j \epsilon_{t-j}$$

where $\phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q \in \mathbb{R}$ and, $p,\ d,\ q \in \mathbb{N}$ are the model parameters (Tsay, 2000).

3.2.3 Exponential Smoothing

Simple exponential smoothing is an old method for smoothing out the time series with an exponential window. It was first suggested as a method for demand prediction by Brown (1956) and gives estimates of x_t as

$$\hat{x}_t = \alpha \tilde{x}_{t-1} + (1 - \alpha)\hat{x}_{t-1}$$

where $\tilde{x}_{t-1}$ is the previous observation of x_t and $\alpha \in \mathbb{R}$ is the smoothing coefficient. This is in fact an ARIMA(0, 1, 1) model (Brockwell and R. A. Davis, 2016). Exponential smoothing has been popular in business and industry since its inception because of its simplicity and flexibility (Gooijer and Hyndman, 2006). The simple exponential smoothing method was first extended by Holt (1957) and then Winters (1960), to include trend, into what is today known as the Holt–Winters or Double

Exponential model. It includes a second coefficient β that smooths the trend such that the estimate of x_t is given by

$$\hat{x}_t = \alpha\tilde{x}_{t-1} + (1-\alpha)(\hat{x}_{t-1} + b_{t-1})$$
$$b_{t-1} = \beta(\hat{x}_{t-1} - \hat{x}_{t-2}) + (1-\beta)b_{t-2}$$

with $\alpha, \beta \in \mathbb{R}$.

3.3 Non-parametric forecasting methods

In the past decade a large share of academic research in forecasting has been dedicated the use of non-parametric methods, especially neural networks (Parmezan and Souza, 2019). The advances in natural language processing (NLP), image recognition and language translation together with the continued successes of open-source development have given researchers a slew of new tools to develop forecasting models in such as Tensorflow (Abadi et al., 2015), PyTorch (Paszke et al., 2019) and GluonTS (Alexandrov et al., 2019).

3.3.1 Global and local models

So far, the examples given have been focused on forecasting a single time series, such as demand for tennis shoes or a persons daily activities. In many real-world applications forecast needs to be made for a set of items with corresponding time series: the individual demand for all items sold by a sporting goods outlet; the default risk on a set of housing loans; or sale of individual titles by bookstores across a region.

In the traditional way of forecasting each time series is dealt with individually by creating a *local* model for each item (Sen, Yu, and Dhillon, 2019). This is powerful because the model can be tailored to individual characteristics of the underlying data using a standardized methodology, such as the popular Box–Jenkins method (Box and Jenkins, 1970). A second advantage of this approach is that the complexity of these models can generally be kept quite low. Manually building a model, through the Box–Jenkins method or other popular frameworks, is a time-consuming activity that does not scale well when the number of time series to forecast increases.

Modern tools for automatically generating, tuning and evaluating local models have vastly improved this process; two of the most popular such tools are the `forecast` package in R (Hyndman and Khandakar, 2008) and Facebook Prophet (S. J. Taylor and Letham, 2017). But with the growth in available data over the past decades (Gantz and Reinsel, 2012) interest in methods for efficiently forecasting millions of correlated time series is rising and at that scale even these automation tools have issues.

A very different approach when dealing with a large number of similar time series, that has gained popularity over past few years, is to train a single *global* model for generating predictions (Sen, Yu, and Dhillon, 2019). This has a few major advantages: first, a global model can learn dependencies between time series both

as simple correlations (e.g. sales of tennis balls and tennis shoes may correlate) and as lagged dependencies (e.g. high late-season sales of summer clothes could indicate worse sales for the autumn collections). Second, global models often perform better for time series with few observations, compared to creating a local model on little data. This is has important application in domains such as sales forecasting where it can be used to accurately forecast demand for newly introduced products.

3.3.2 Ensemble methods

Ensemble methods using decision trees have gained popularity in a wide range of tasks in recent years (Marsland, 2015). Ensemble methods are built out of the premise that by combining a set of individual *weak predictors* $\{f_1, f_2, \ldots, f_N\}$ to a strong predictor F through some aggregation operator (Moraga, 2017). Using ensemble methods for time series prediction has seen some success in forecasting tasks (Galicia et al., 2019), including at Volvo Group.

3.3.3 Neural networks

Using artificial neural networks (ANNs) for forecasting problems dates back to 1964 (M. J. C. Hu and Root, 1964) but really started to gain traction in the 1990's (Zhang, Patuwo, and M. Y. Hu, 1998). The mathematical foundation for training neural networks, backpropagation, was formulated in the the 1980's by Rumelhart, Hinton, and Williams (1986) and Werbos (1988) with interest in ANNs in recent years largely been driven by the increasing availability of high-performance computing environments (Dean et al., 2012; Raina, Madhavan, and Ng, 2009).

A neural network is a directed graph of nodes, usually organised in layers. Each node maps an input vector $\boldsymbol{x}$ and a weight vector $\boldsymbol{w}$ to a scalar *activation* a, which can then be used as part of the input to the next layer. The most popular maps are the sigmoid function σ or the hyperbolic tangent tanh

$$\sigma(\boldsymbol{w}^T\boldsymbol{x} + b) := \frac{1}{1 + e^{-\boldsymbol{w}^T\boldsymbol{x}-b}}, \tag{3.9}$$

$$\tanh(\boldsymbol{w}^T\boldsymbol{x} + b) := \frac{e^{\boldsymbol{w}^T\boldsymbol{x}+b} - e^{-\boldsymbol{w}^T\boldsymbol{x}-b}}{e^{\boldsymbol{w}^T\boldsymbol{x}+b} + e^{-\boldsymbol{w}^T\boldsymbol{x}-b}} \tag{3.10}$$

where b is a variable bias.

Following the notation of Nielsen (2015) let w^l_{jk} denote the weight for the connection from the k-th node in the $(l-1)$-th layer to the j-th node in the l-th layer. Then the activation of node j in layer l is given by

$$a^l_j = f^l_j\Big(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\Big)$$

where f^l_j generally is the σ or tanh function. Let n_l be the number of nodes in layer l, $f^l(\boldsymbol{z}) := (f^l_1(z_1), \ldots, f^l_{n_l}(z_{n_l}))^T$ for $\boldsymbol{z} \in \mathbb{R}^{n_l}$,

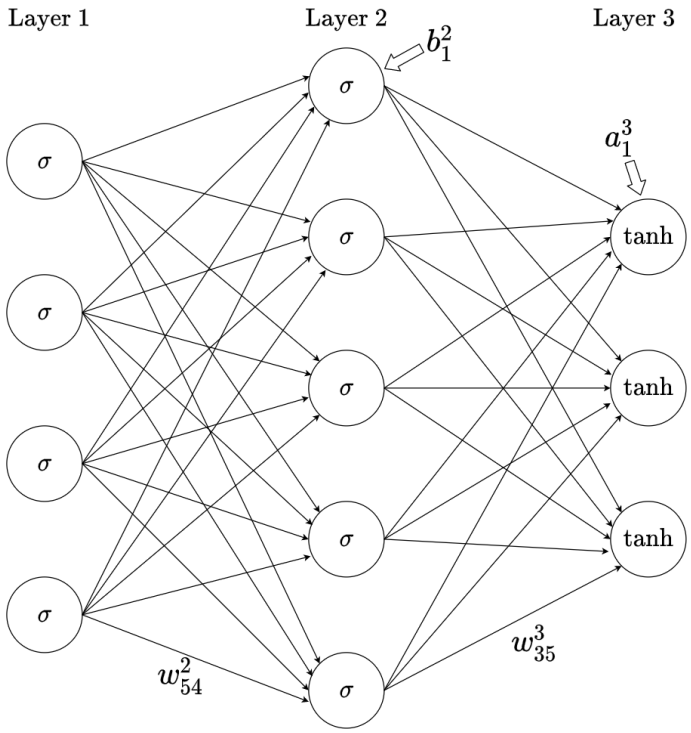

Figure 3.1: An example of a feedforward network . The first two layers are sets of sigmoid functions while the final layer has tanh as activation function.

$$\begin{aligned} \boldsymbol{W}^l &:= [w_{jk}^l]_{j,k=1,\dots,n_l}, \\ \boldsymbol{b}^l &:= (b_1^l, \dots, b_{n_l}^l)^T \text{ and} \\ \boldsymbol{a}^l &:= (a_1^l, \dots, a_{n_l}^l)^T \end{aligned}$$

where $\boldsymbol{W}^l$ is called the weight matrix, $\boldsymbol{b}^l$ the bias vector and $\boldsymbol{a}^l$ the activation vector at layer l. The activation vector is then given by $\boldsymbol{a}^l = f^l(\boldsymbol{W}^l\boldsymbol{a}^{l-1} + \boldsymbol{b}^l)$. A part of network showing this calculation is featured in Figure 3.1.

In a supervised setting, a neural network learns by processing an input and comparing the networks output with the truth label. The comparison is done with a *loss* function $\mathcal{L}$. The gradient of the loss function with regard to each weight at each node is then calculated through backpropagation and the weights are updated according to their gradients and a line search method. One common example of a loss function is the mean square error (MSE) loss

$$\mathcal{L}_{\text{MSE}} := \frac{\sum_{i=1}^{N}(\tilde{x}_i - \hat{x}_i)^2}{N} \tag{3.11}$$

when estimating target variables on the real line, where $\tilde{x}_i$ are the realizations—truth labels—of the random variable x, $\hat{x}_i$ are estimates generated by the model and N is the number of samples. A common loss function when working with probabilities and estimating distribution parameters θ is negative log likelihood (ℓ)

$$\mathcal{L}_\ell := -\sum_{i=1}^{N} \log(P(x = \tilde{x}_i|\hat{\theta}_i)), \tag{3.12}$$

where the probability density function depends on $\hat{\theta}_i$, which is the output by the network. To make this loss function more tangible, imagine a neural network training to differ between a number of different animals in an image, i.e. if the animal pictured is a cat, rooster, cow etc. The input to the network are the pixels of the image and the output are the parameters $\hat{\theta}$ of a multinomial distribution: say 10% chance it's a cow, 60% chance it is a cat and 30% chance it is a rooster. This loss function then rewards assigning a high probability to the right label.

Backpropagation

The goal of backpropagation is to adjust the weights at each node to minimize the loss function $\mathcal{L}$ using stochastic gradient descent (Rumelhart, Hinton, and Williams, 1986). To do this we need to calculate $\partial\mathcal{L}_{\tilde{x}}/\partial w_{jk}^l$ and $\partial\mathcal{L}_{\tilde{x}}/\partial b_j^l$ for all individual weights and biases and for each training example $\tilde{x}$ (Nielsen, 2015). The loss is required to be obtainable as an average of the individual samples and to be a deterministic function of the output of the neural network (Nielsen, 2015); we see that both $\mathcal{L}_{\text{MSE}}$ and $\mathcal{L}_\ell$ satisfy these requirements. To simplify the notation let $\boldsymbol{z}^l := \boldsymbol{W}^l\boldsymbol{a}^{l-1} + \boldsymbol{b}^l$ be the weighted input to layer l so that $\boldsymbol{a}^l = f^l(\boldsymbol{z}^l)$. Let

$$\delta_j^l := \frac{\partial\mathcal{L}}{\partial z_j^l} \tag{3.13}$$

be the loss of node j in layer l and $\boldsymbol{\delta}^l := (\delta_1^l, \delta_2^l, \dots)^T$. Then, the loss at the output layer L is

$$\boldsymbol{\delta}^L = \nabla_a\mathcal{L} \odot (f^L)'(\boldsymbol{z}^L), \tag{3.14}$$

where $\nabla_a\mathcal{L}$ is the vector of partial derivatives $\partial\mathcal{L}/\partial a_j^L$, $\odot$ is the Hadamard product (elementwise multiplication) and $(f^L)'(\boldsymbol{z}^L)$ is a vector of the derivative of f^L evaluated at each z_j^L. The loss at layer l is computed recursively by

$$\boldsymbol{\delta}^l = ((\boldsymbol{W}^{l+1})^T\boldsymbol{\delta}^{l+1}) \odot (f^l)'(\boldsymbol{z}^l). \tag{3.15}$$

With this the partial derivative of the loss with respect to any bias is

$$\frac{\partial\mathcal{L}}{\partial b_j^l} = \delta_j^l. \tag{3.16}$$

and the partial derivative with regard to any weight w_{jk}^l is calculated by

$$\frac{\partial\mathcal{L}}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l. \tag{3.17}$$

With these gradients in hand we can update the weights and biases in accordance with the chosen line search method. Both models in 4 are trained using the stochastic optimization algorithm Adam (Kingma and Ba, 2015).

3.3.4 Recurrent neural networks

The network shown in Figure 3.1 is a *feedforward* network, where the activation of each layer $l-1$ is fed forward into layer l. This works well when the data samples are independent observations, but in a time series $(x_t, t \in \mathbb{T})$ there is often a strong correlation within the series which is not captured by the feedforward architecture. A Recurrent Neural Network (RNN) is a type of network architecture designed to have *associative memory* where important information from previous inputs of the sample sequence is retained as each new input is fed into the network. RNNs have been a natural choice for forecasting problems for decades (Gent and Sheppard, 1992; Hewamalage, Bergmeir, and Bandara, 2019). Let $(x_t, t \in \mathbb{T})$ be a times series to be forecast with observations $(\tilde{x}_t, t \in \mathbb{T})$. An RNN iteratively reads $\tilde{x}_1, \tilde{x}_2, \ldots$ into a non-linear activation function f that outputs a vector $\boldsymbol{h}_t$

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \tilde{x}_t) \tag{3.18}$$

called the *hidden state* that encodes important information from previous observations (Cho et al., 2014). The length of $\boldsymbol{h}_t$ is held constant and is called the context length. The function f can be a simple tanh function in some instances, but generally a more complex structure is necessary.

RNNs have seen much use and development for tasks such as language translation and NLP both in academia and in the tech industry (Cho et al., 2014; Sutskever, Vinyals, and Le, 2014). The most commonly used RNN architectures are the Long Short-term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and the Gated Recurrent Unit (GRU) (Cho et al., 2014) networks. Recently, RNN-based models have been achieving state-of-the-art results—Smyl (2020) won the M4 competition with a hybrid exponential smoothing and RNN model. The M5 competition, which is currently ongoing on Kaggle, looks likely to introduce further innovation of this sort.

3.3.4.1 Long short-term memory

The long short-term memory (LSTM) cell is a complex structure and a powerful choice for the function f in (3.18) first proposed by Hochreiter and Schmidhuber (1997). A network of LSTM cells can be said to have two parallel tracks—one for the *cell state* and one for the hidden state—along which information from previous cells flows. A single LSTM cell is illustrated in Figure 3.2. The internal structure of the LSTM cell is a series of *gates* that regulate how new information is incorporated into the network's "memory". In Figure 3.2 circles denote a network layer with the given activation function, squares denote a pointwise operation and at each fork along the black paths the state is copied. Each network layer has its own set of weights which are tuned when the model is trained.

First the new observation $\tilde{x}_t$ is concatenated, denoted by the operator $\uplus$, with the hidden state

$$\boldsymbol{u}_t := \boldsymbol{h}_{t-1} \uplus \tilde{x}_t = \begin{pmatrix} \boldsymbol{h}_{t-1} \\ \tilde{x}_t \end{pmatrix}$$

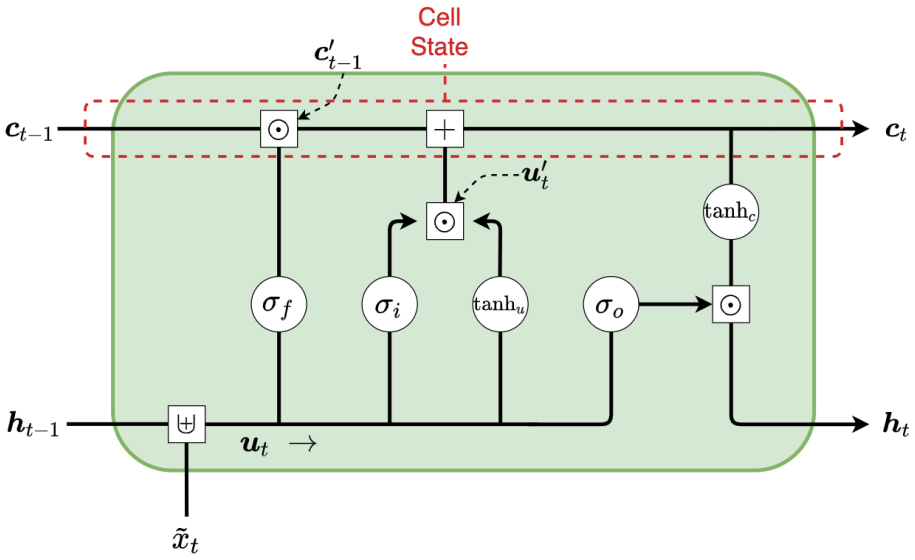

Figure 3.2: The LSTM cell

from the previous cell; let d be the length of $\boldsymbol{u}_t$. Note that this operation works both when the input is a scalar value and when it is a vector.

The first sigmoid layer σ_f is called the *forget gate*, see Figure 3.3, and is used to gradually reset the cell state (Gers, Schmidhuber, and Cummins, 1999). The $(0,1)^d$ range of the sigmoid output is multiplied elementwise with $\boldsymbol{c}_{t-1}$ to "forget" (σ values close to zero) or keep (σ values close to one) each value in the cell state $\boldsymbol{c}_{t-1}$ and in this way remove out-of-date information. Let

$$\boldsymbol{c}'_{t-1} := \boldsymbol{c}_{t-1} \odot \sigma(\boldsymbol{W}_f \boldsymbol{u}_t + \boldsymbol{b}_f)$$

denote the adjusted $\boldsymbol{c}_{t-1}$; $\boldsymbol{W}_f$ is the first sigmoid layer's trained weight matrix and $\boldsymbol{b}_f$ are the learned biases.

The second sigmoid layer σ_i and the $\tanh_u$ layer together constitute the *input gate*. The $\tanh_u$ layer squashes the entries of $\boldsymbol{u}_t$ into the $(-1,1)^d$ range to ensure that the values of the cell state do not explode over time. The sigmoid layer decides which information to introduce to the cell state through elementwise multiplication. Denote the output of the input gate by $\boldsymbol{u}'_t$; it is then given by

$$\boldsymbol{u}'_t := \sigma(\boldsymbol{W}_i \boldsymbol{u}_t + \boldsymbol{b}_i) \odot \tanh(\boldsymbol{W}_u \boldsymbol{u}_t + \boldsymbol{b}_u)$$

where $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ are the weights and biases of the input gate sigmoid layer while $\boldsymbol{W}_u$ and $\boldsymbol{b}_u$ are the weights and biases for the tanh layer. This is then added to $\boldsymbol{c}'_{t-1}$ to generate $\boldsymbol{c}_t$

$$\boldsymbol{c}_t = \boldsymbol{c}'_{t-1} + \boldsymbol{u}'_t. \tag{3.19}$$

The final construction in the LSTM cell is the *output gate*, which calculates $\boldsymbol{h}_t$. The cell state $\boldsymbol{c}_t$ is passed through the $\tanh_c$ layer to regulate the vector's magnitude and $\boldsymbol{u}_t$ is passed to the third sigmoid layer σ_o to determine what to keep and what to forget from the cell state. The full expression for $\boldsymbol{h}_t$ is thus

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_c \boldsymbol{c}_t + \boldsymbol{b}_c) \odot \sigma(\boldsymbol{W}_o \boldsymbol{u}_t + \boldsymbol{b}_o), \tag{3.20}$$

where $\boldsymbol{W}_c$, $\boldsymbol{W}_o$, $\boldsymbol{b}_c$ and $\boldsymbol{b}_o$ are the weights and biases of $\tanh_c$ and σ_o, respectively.

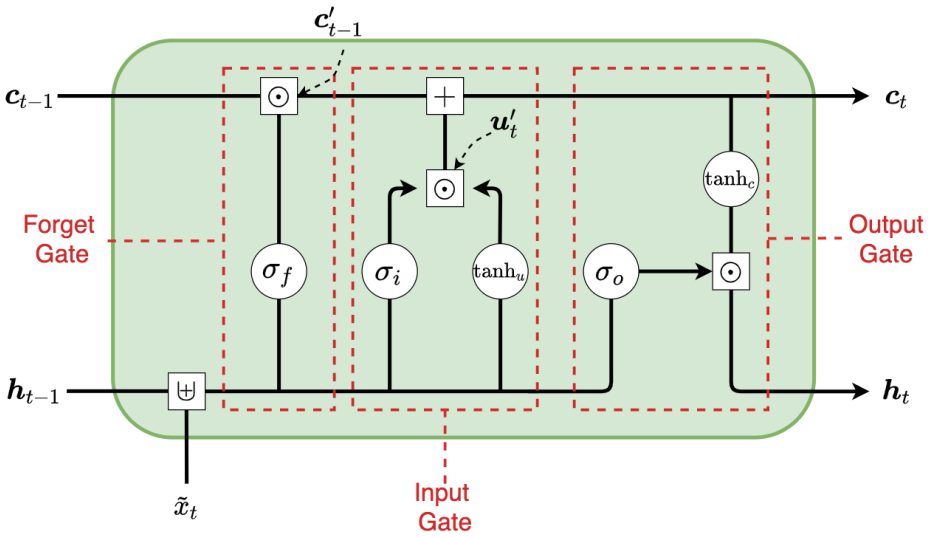

Figure 3.3: The three gates that make up an LSTM cell.

3.3.4.2 Gated recurrent units

The Gated Recurrent unit was introduced by Cho et al. (2014), together with the encoder-decoder architecture, as a simpler alternative to the LSTM cell without a cell state. See Figure 3.4 for a graphical depiction. The GRU consists of two gates: a *reset gate* and an *update gate*.

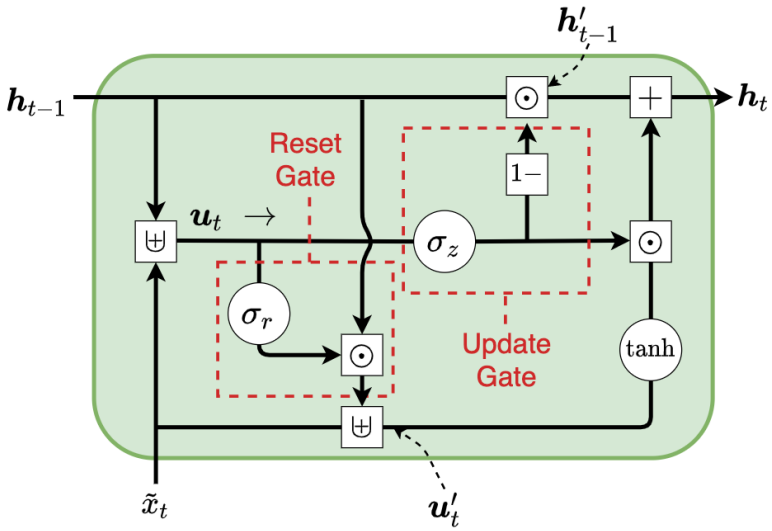

Figure 3.4: The input and reset gate in the GRU cell.

Just as with the LSTM cell a concatenation $\boldsymbol{u}_t := (\boldsymbol{h}_{t-1}, \tilde{x}_t)^T$ is first fed into two different sigmoid layers the output of which determines how much of $\boldsymbol{h}_{t-1}$ is forgotten. The first sigmoid layer σ_r is multiplied elementwise with $\boldsymbol{h}_{t-1}$ and contatenated with $\tilde{x}_t$ such that

$$\boldsymbol{u}'_t := \boldsymbol{h}_{t-1} \odot \sigma_r(\boldsymbol{W}_r \boldsymbol{u}_t + \boldsymbol{b}_r)$$

where $\boldsymbol{W}_r$ is the learned weight matrix for the layer and $\boldsymbol{b}_r$ is a vector of learned biases. The vector $\boldsymbol{u}'_t$ is then fed through a tanh layer and multiplied elementwise with the output from the second sigmoid layer σ_z.

The output of the σ_z is also copied, inverted and multiplied elementwise with $\boldsymbol{h}_{t-1}$ to generate

$$\boldsymbol{h}'_{t-1} := \boldsymbol{h}_{t-1}(\boldsymbol{1} - \sigma(\boldsymbol{W}_z \boldsymbol{u}_t + \boldsymbol{b}_z)),$$

where $\boldsymbol{W}_z$ and $\boldsymbol{b}_z$ are the layer's weights and biases. This is then combined with $\boldsymbol{u}'_t$ to calculate $\boldsymbol{h}_t$ by

$$\boldsymbol{h}_t = \boldsymbol{h}'_{t-1} + (\tanh(\boldsymbol{W}_\phi \boldsymbol{u}'_t + \boldsymbol{b}_\phi) \odot \sigma_z(\boldsymbol{W}_z \boldsymbol{u}_t + \boldsymbol{b}_z)), \tag{3.21}$$

where $\boldsymbol{W}_\phi$ and $\boldsymbol{b}_\phi$ are the learned weights and biases of the tanh layer.

4
Models

This chapter will lay out the design of and training scheme for two probabilistic forecasting models: a simplified multi-horizon quantile recurrent neural network (MQRNN) model, based on the original work by Wen et al. (2017), and an autoregressive recurrent neural network (DeepAR) model based on Salinas et al. (2019). Both are built upon the encoder-decoder framework (Cho et al., 2014) using RNNs to model the conditional distributions

$$P(\mathbf{y}_{t+1:t+K,j}|\mathbf{y}_{:t,j}, \mathbf{x}_{:t+K,j}) \tag{4.1}$$

where $\mathbf{y}_{t+1:t+K,j} := \{y_{t+1}, \ldots, y_{t+K}\}_j$ and $\mathbf{y}_{:t,j} := \{y_0, \ldots, y_t\}_j$; $j \in \mathcal{S}$ are indices for the individual time series; $\mathbf{x}_{:t,j}$ are covariates and K is the number of steps to forecast. The covariates are assumed to be known not only to the current time t but all the way to $t + K$ and can consist of such things as holidays, day-of-the-month or other similar features. For brevity let $\mathbf{z}_{t,j} := \{\mathbf{y}_{:t,j}, \mathbf{x}_{:t+K,j}\}$.

The ultimate goal is to, for all $k \in \mathcal{K} := \{1, \ldots, K\}$ and $j \in \mathcal{S}$, estimate the τ-th *quantile* $Q_{y_{t+k,j}|\mathbf{z}_{t,j}}(\tau)$

$$Q_{y_{t+k,j}|\mathbf{z}_{t,j}}(\tau) := \inf\{y \in \mathbb{R} : F_{y_{t+k,j}|\mathbf{z}_{t,j}}(y) \geq \tau\}, \quad 0 < \tau < 1$$

for a set of chosen quantiles $\mathcal{Q}$. $F_{y_{t+k,j}|\mathbf{z}_{t,j}}$ is the conditional cumulative distribution function of $y_{t+k,j}$. The two models have differing methods for obtaining the conditional quantile estimates: the MQRNN model directly estimates them through *quantile regression*, while the DeepAR model uses maximum likelihood to fit distribution parameters and draws sample paths over the forecast range. They are both implemented using the GluonTS framework (Alexandrov et al., 2019) and trained using the training schemes that the original authors used.

4.1 Multi-Horizon Quantile Recurrent Forecaster

4.1.1 Quantile regression

When looking at predicting supplier's delivery performance for the coming weeks, it can be argued the predicted percentage is of less importance compared to how *confident* we are in that prediction. Quantile regression, a type of regression analysis that has less strict assumptions than linear regression, is a technique for estimating the conditional median (or any other chosen quantile) of a target variable introduced by Koenker and Basset (1978). It differs from traditional linear regression in that

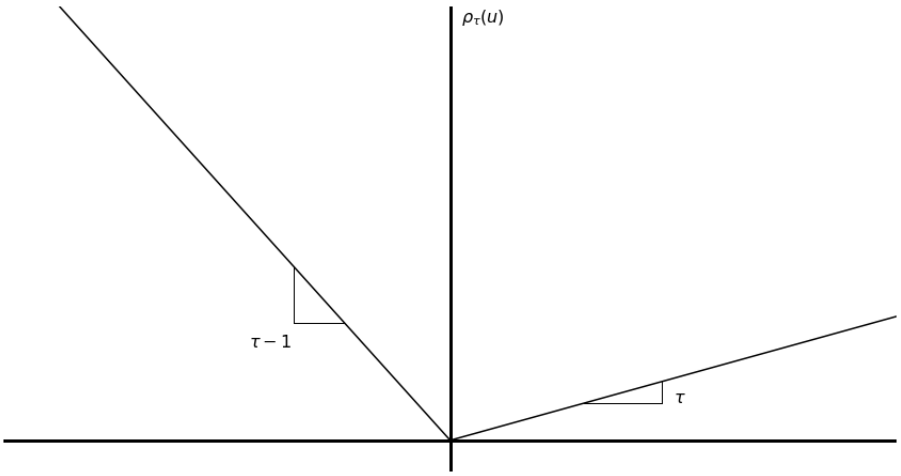

Figure 4.1: Tilted absolute value function

linear regression estimates the conditional mean of the target variable, and also in the underlying minimization problem.

Quantile regression is based on the observation that the unconditional τ-th sample quantile can be expressed as the solution to the minimization problem

$$\min_{\xi \in \mathbb{R}} \sum_{t \in \{t:\tilde{y}_t \geq \xi\}} \tau |\tilde{y}_t - \xi| + \sum_{t \in \{t:\tilde{y}_t < \xi\}} (1-\tau)|\tilde{y}_t - \xi| = \min_{\xi \in \mathbb{R}} \sum_t \rho_\tau(\tilde{y}_t - \xi), \tag{4.2}$$

where $(\tilde{y}_t,\ t = 1, ..., T)$ are random samples on the random variable y and $\rho_\tau(\cdot)$ is the *tilted absolute value* function (4.3) seen in Figure 4.1 (Koenker and Hallock, 2001) given by

$$\rho_\tau(u) := \tau \max(u, 0) + (1-\tau)\max(-u, 0). \tag{4.3}$$

The quantile regression minimization problem is analogous to minimizing $\sum_t (\tilde{y}_t - \mu)^2$ over $\mu \in \mathbb{R}$ to obtain an estimate of $E(y)$ (Koenker and Basset, 1978).

The classical way of estimating the quantile function of a random variable y conditional on a random vector of covariates $\boldsymbol{x}$, $Q_{y|\boldsymbol{x}}(\tau)$, has been to replace ξ in equation (4.2) by a linear parametric function $\xi(\tilde{\boldsymbol{x}}_t, \beta)$

$$\min_{\beta \in \mathbb{R}^P} \sum_t \rho_\tau(\tilde{y}_t - \xi(\tilde{\boldsymbol{x}}_t, \beta)) \tag{4.4}$$

where $\beta \in \mathbb{R}^P$ and $\tilde{\boldsymbol{x}}_t$ are realizations of $\boldsymbol{x}$ with the resulting minimization problem being solved through linear programming (Koenker and Hallock, 2001).

4.1.2 Quantile regression with neural networks

Techniques for nonlinear quantile regression were developed during the 1990's with Koenker and Park (1996) applying the interior point method to (4.4) with nonlinear $\xi(\mathbf{x}_t, \beta)$. White (1992) laid the mathematical foundation for using neural networks

for quantile regression by proving the consistency of the conditional quantile estimator given by a feedforward network with (4.3) as loss function. The neural network formulation of the quantile regression problem for a quantile τ is

$$\min_{\mathbf{w}} \sum_{t} \rho_\tau(y_t - f(\mathbf{x}_t, \mathbf{w})), \tag{4.5}$$

where f is a neural network with weights $\mathbf{w}$ and the minimization is done through backpropagation. The first full implementation of quantile regression using neural networks was by J. W. Taylor (2000) on stock market data using a single feedforward network; the model was later built into a package for R by Cannon (2011).

The MQRNN model (Wen et al., 2017) takes inspiration in the success that RNNs have had in point forecasting problems (Hewamalage, Bergmeir, and Bandara, 2019) and applies the structure to quantile regression. It extends the loss function of (4.5) to sum over the set of chosen quantiles to the forecast horizon to make the model minimize total loss. As before we let $\mathcal{K} := \{1, 2, \ldots, K\}$ be the forecast range and $\mathcal{Q}$ be the set of quantiles we are interested in estimating. The MQRNN model's output is a $|\mathcal{K}| \times |\mathcal{Q}|$ matrix $\hat{\mathbf{Y}}_t = [\hat{y}^{(\tau)}_{t+k}]_{k\in\mathcal{K},\tau\in\mathcal{Q}}$ and it is optimized by the loss function $\mathcal{L}_j$

$$\mathcal{L}_j = \sum_{t\in\mathcal{T}} \sum_{\tau\in\mathcal{Q}} \sum_{k\in\mathcal{K}} \rho_\tau(y_{t+K} - \hat{y}^{(\tau)}_{t+k})_j \tag{4.6}$$

for $j \in \mathcal{S}$, where $\mathcal{T}$ is the set of starting points for forecasts in the training phase, called the *forecast creation times* (FCTs).

4.1.3 Model architecture

The MQRNN model has two different modules: the encoder function m and the decoder function Λ. The encoder learns to transform variable length sequences, in our application time series of delivery performance for different manufacturers, to a fixed-length vector representation $\mathbf{h}_t$, for an FCT t, called the *hidden state*; the fixed length $|\mathbf{h_t}|$ is called the *context length*. This is done by iteratively feeding the input sequence into the encoder

$$\begin{aligned} \mathbf{h}_t &= m(\mathbf{h}_{t-1}, y_{t-1}, \mathbf{w}^{(e)}), \\ \mathbf{h}_0 &:= \mathbf{0}. \end{aligned}$$

The encoder itself is a network of GRU cells with weights $\mathbf{w}^{(e)}$. The decoder module, with model weights $\mathbf{w}^{(d)}$, takes as input the hidden state and outputs a matrix $\hat{\mathbf{Y}}_t \in \mathbb{R}^{|\mathcal{K}|\times|\mathcal{Q}|}$

$$\hat{\mathbf{Y}}_t = [\hat{y}^{(\tau)}_{t+k}]_{k\in\mathcal{K},\tau\in\mathcal{Q}} = \Lambda(\mathbf{h}_t, \mathbf{w}^{(d)})$$

that forecasts each quantile of interest at each time-step in the forecast range. A partially unrolled illustration of the model is given in Figure 4.2.

In the original MQRNN architecture the decoder consisted of two feedforward network branches. The first is a *global* feedforward network that synthesises $\mathbf{h}_t$ with

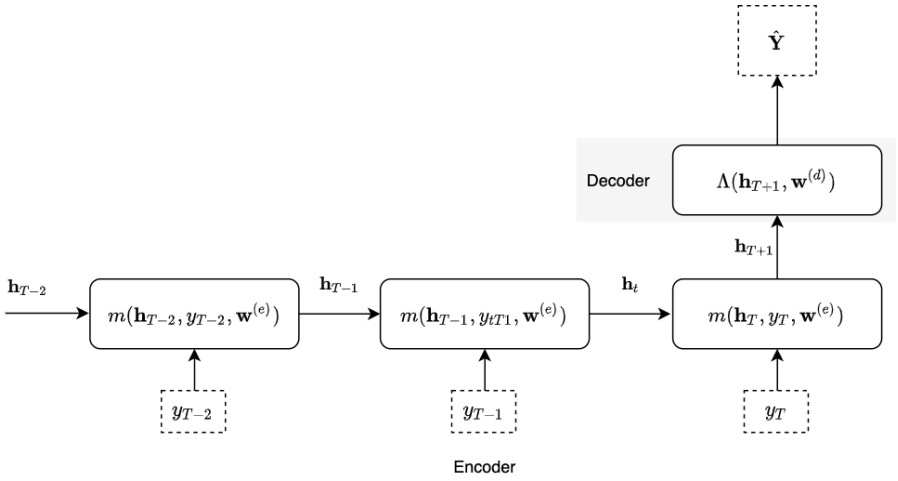

Figure 4.2: The MQRNN model in the prediction phase. The time series is fed into encoder iteratively up to the most recent observation y_T. The hidden state $\mathbf{h}_{T+1}$ is input into the decoder, which outputs the forecast matrix $\hat{\mathbf{Y}}_T$.

known future information, such as upcoming product promotions or holidays. It then feeds its output into *local* feedforward networks, for each step in the forecast horizon, which output the vector $(\hat{y}_{t+k}^{(\tau)})_{\tau \in \mathcal{Q}}$, the quantile estimates at their time-step k.

Due to an implementation issue in GluonTS the encoder built into the package for the MQRNN model can only take a one-dimensional sequence, making it impossible to include covariates of the target variable in the model. This does not change the encoder architecture, but the lack of covariates does warrant decreasing the complexity of the decoder. In our simplified MQRNN model the decoder consists of a single feedforward network containing three layers with the number of nodes in each matching the context length.

4.1.4 Training scheme

The training scheme designed by Wen et al. (2017) is one of the most interesting aspects of the model. Previously, the standard method for training a forecasting model has been to choose a set of FCTs $\mathcal{T}$, either randomly or following some logical rules, and for each FCT let the preceding observations be the input sequence and the K following observations be the target vector. With the inclusion of covariates, both static and time-dependent, creating the training dataset requires considerable data augmentation.

The MQRNN model avoids this data augmentation by instead augmenting the model during the training phase in what they call a *forking-sequences* training scheme. Conceptually, in the training phase the decoder is duplicated into $T-1$ copies with identical weights, where T is the length of the time series. The output of the encoder for each iteration $\mathbf{h}_{t_i}$ is forked with one copy of the vector being the input into the next iteration while the other is fed into the decoder. This is illustrated in Figure 4.3. The output from each of these decoders is a forecast $\hat{\mathbf{Y}}_{t_i}$ of the future time series K steps forward relative to t_i. In this way each point in the time

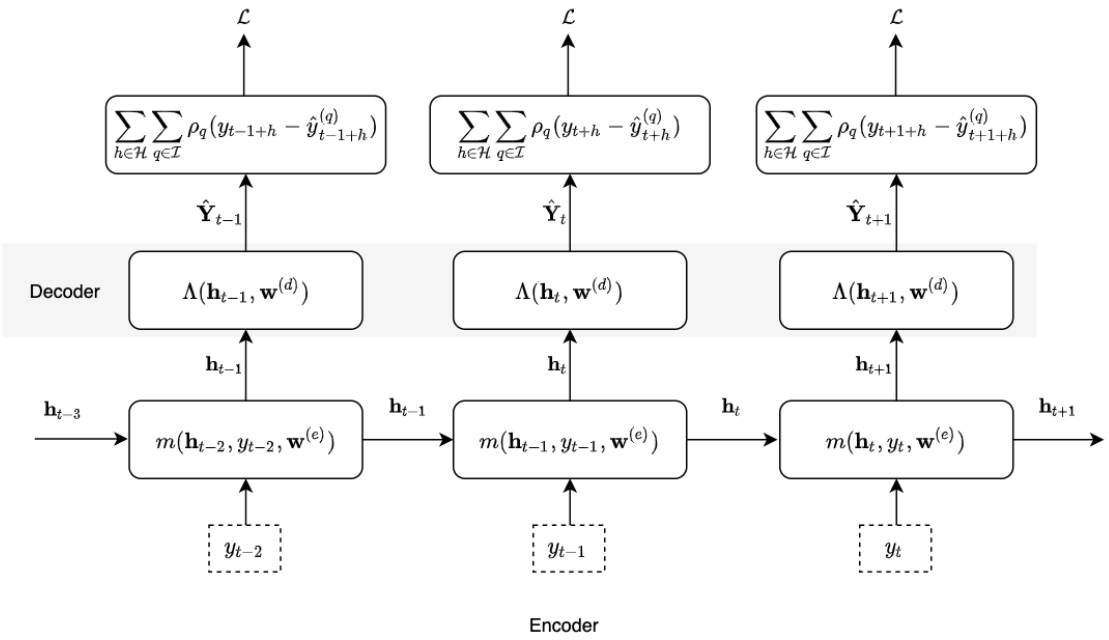

Figure 4.3: The MQRNN model in the training phase. The time series is iteratively input into the encoder. At each step the hidden state $\mathbf{h}_{t_i}$ is forked, feeding into both the encoder and into a decoder to produce a forecast matrix at each time step.

series becomes an FCT without data augmentation. The forecasts are compared to their corresponding target in the loss function (4.6) and summed up before the errors are propagated backwards through the network.

4.2 DeepAR

The DeepAR model (Salinas et al., 2019), in contrast to the MQRNN model, does not directly estimate the conditional quantiles. Instead it follows closer to the original design of the seq2seq model (Graves, 2013) and models (4.1) as the product of K likelihood factors

$$f_{\mathbf{w}}(\mathbf{y}_{t+1:t+K,j}|\mathbf{z}_{t,j}) = \prod_{k\in\mathcal{K}} f_{\mathbf{w}}(y_{t+k,j}|\mathbf{z}_{t,j}) = \prod_{k\in\mathcal{K}} p(y_{t+k,j}|\theta(\mathbf{h}_{t+k,j}, \mathbf{w}^{(d)})) \tag{4.7}$$

parametrized by the *hidden state* $\mathbf{h}_{t,j}$

$$\mathbf{h}_{t,j} := m(\mathbf{h}_{t-1,j}, y_{t-1,j}, \mathbf{x}_{t,j}, \mathbf{w}^{(e)}) \tag{4.8}$$

which is the output of the encoder network m. It consists of LSTM cells parametrized by the model encoder weights $\mathbf{w}^{(e)}$. The likelihood $p(y_{t+k,j}|\theta(\mathbf{h}_{t+k,j}, \mathbf{w}^{(d)}))$ is a fixed distribution, the "noise model" in the words of Salinas et al. (2019), chosen by the modeller to fit the characteristics of the data. The parameters of the likelihood function are given by the decoder network $\theta(\mathbf{h}_{t,j}, \mathbf{w}^{(d)})$ with $\mathbf{w}^{(d)}$ being the decoder network's weights.

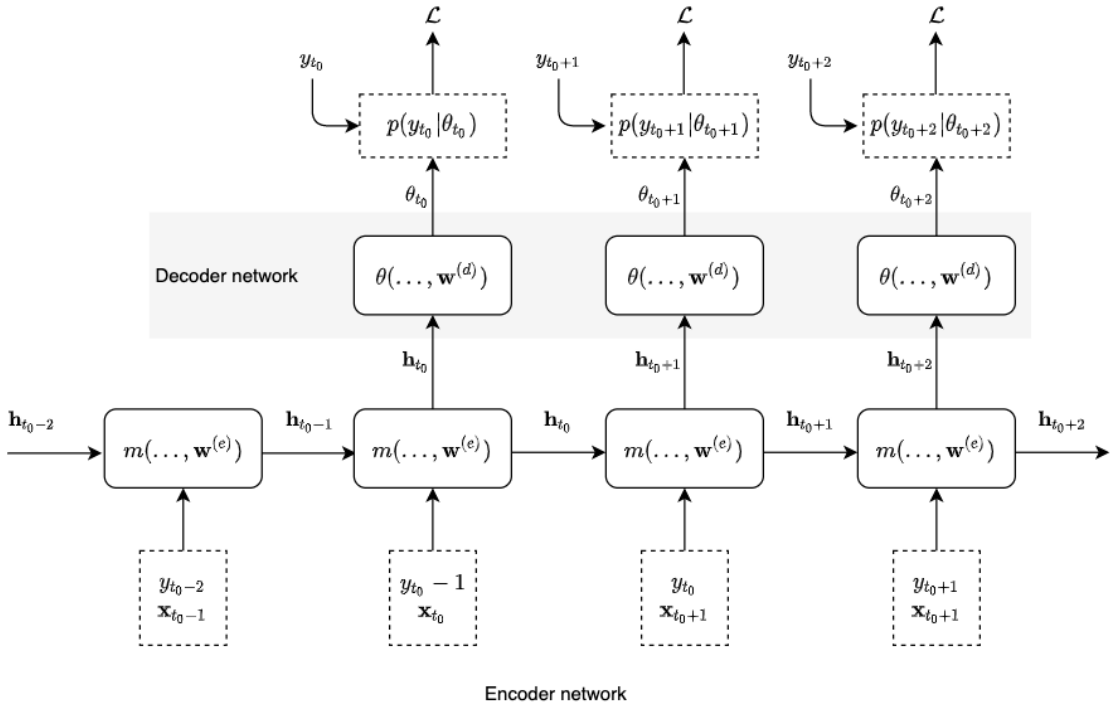

Figure 4.4: The DeepAR model in its training phase. The hidden state $\mathbf{h}_{t_0}$ is calculated iteratively through equation (4.8). From t_0 to $t_0 + K$ the hidden state $\mathbf{h}_{t_0+K}$ is forked with one copy going to the decoder network and the other being the recurrent input to the encoder. The decoder network outputs distribution parameters for the likelihood function, which is calculated as a term in the loss function.

In the training phase the likelihoods $p(y_{t+k,j}|\theta(\mathbf{h}_{t+k,j}, \mathbf{w}^{(d)}))$, $j \in \mathcal{S},\ k \in \mathcal{K}$ are computed from the outputs $\theta(\mathbf{h}_{t+k,j}, \mathbf{w}^{(d)})$ of the complete encoder-decoder network. The loss function $\mathcal{L}$ is the total negative log-likelihood

$$\mathcal{L} = \sum_{j \in S} \sum_{k \in \mathcal{K}} -\log p(y_{t_0+k,j}|\theta(\mathbf{h}_{t_0+k,j})), \tag{4.9}$$

where t_0 is the FCT. Gradients with respect to $\mathbf{w}^{(e)}$ and $\mathbf{w}^{(d)}$ are calculated and the weights are updated through backpropagation. The model in the training phase is illustrated in Figure 4.4.

In the prediction phase, the encoding up to the FCT t_0 is done in the same way as during the training phase, but instead of calculating the likelihood a random sample $\hat{y}_t \sim p(\cdot|\theta(\mathbf{h}_t, \mathbf{w}^{(d)})$ is drawn to replace $y_{t,j}$ in equation (4.8) such that

$$\hat{\mathbf{h}}_{t,j} = m(\hat{\mathbf{h}}_{t-1,j}, \hat{y}_{t-1,j}, \mathbf{x}_{t,j}, \mathbf{w}^{(e)})$$

with $\hat{\mathbf{h}}_{t_0-1,j} := \mathbf{h}_{t_0-1,j}$ and $\hat{y}_{t_0-1,j} := y_{t_0-1,j}$ for all $j \in \mathcal{S}$. By running this process multiple times sample paths can be generated. The empirical distribution of these sample paths can then be used to estimate the conditional quantiles of $y_{t_0+h,j}$ over the forecast range.

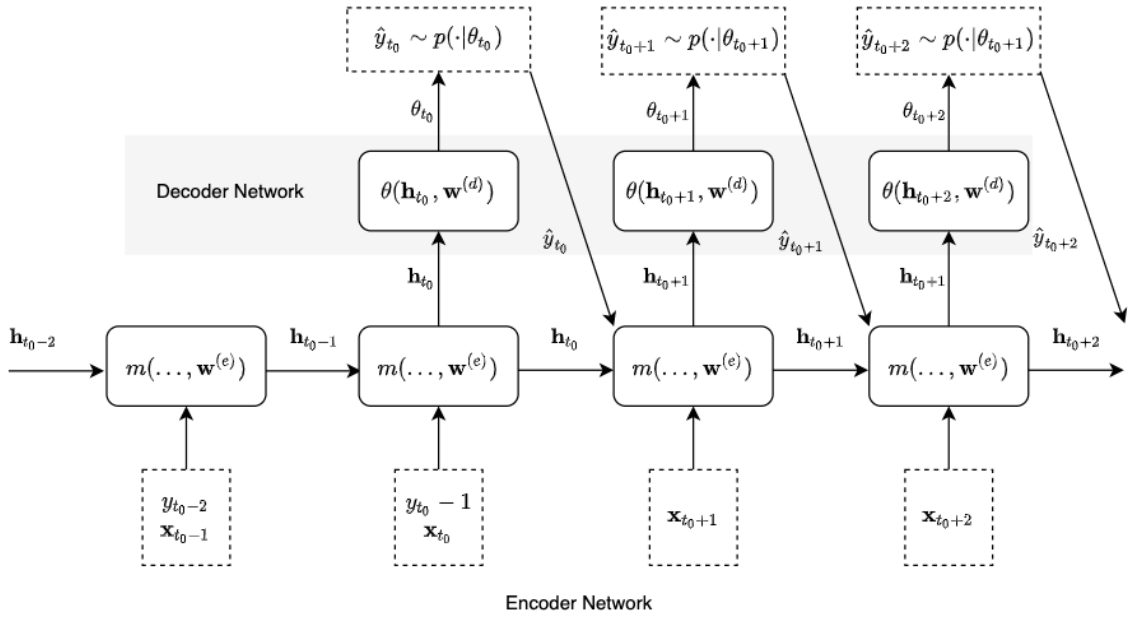

Figure 4.5: The DeepAR model in the prediction phase. The decoder network outputs distribution parameters and a random sample over this distribution is drawn. The random sample is fed into the encoder to output the next hidden state.

4.2.1 Training scheme

The training scheme used by Salinas et al. (2019) is more rudimentary than the one designed by Wen et al. (2017) for the MQRNN model. FCTs are chosen randomly in such a way that the expected number of FCTs per time series is uniform over all time series, so as to not over-represent the longer ones. Within a time series of length T the probability of a time point being an FCT is uniform for $t \in [1, T - H]$ and zero for $t > T - H$.

4.3 Data

The target time series are weekly measures of supplier delivery performance (DP). DP is defined as the percentage of order lines correctly filled by the supplier that week. An order line is counted as incorrectly filled (KO'd) if it does not arrive in the correct week or if the quantity is incorrect. To visualize this think of a single supplier of oil filters that has accepted orders for ten separate products in a given week. For each product there is an ordered quantity and if the delivered quantity does not match this or if it does not arrive, it counts as incorrectly delivered. If eight out of ten orders are correctly filled the supplier gets a DP score of 0.8 for that week. The measure does not take into account how "wrong" a delivery is, i.e. a shipment of 49 units when 50 were ordered is counted in the same way as a fully missed shipment.

The dataset contains 2436 time series with 408 929 observations in total, some examples of which are shown in Figure 4.6. For weeks when a supplier does not have any deliveries to fill, the latest DP value is carried forward.

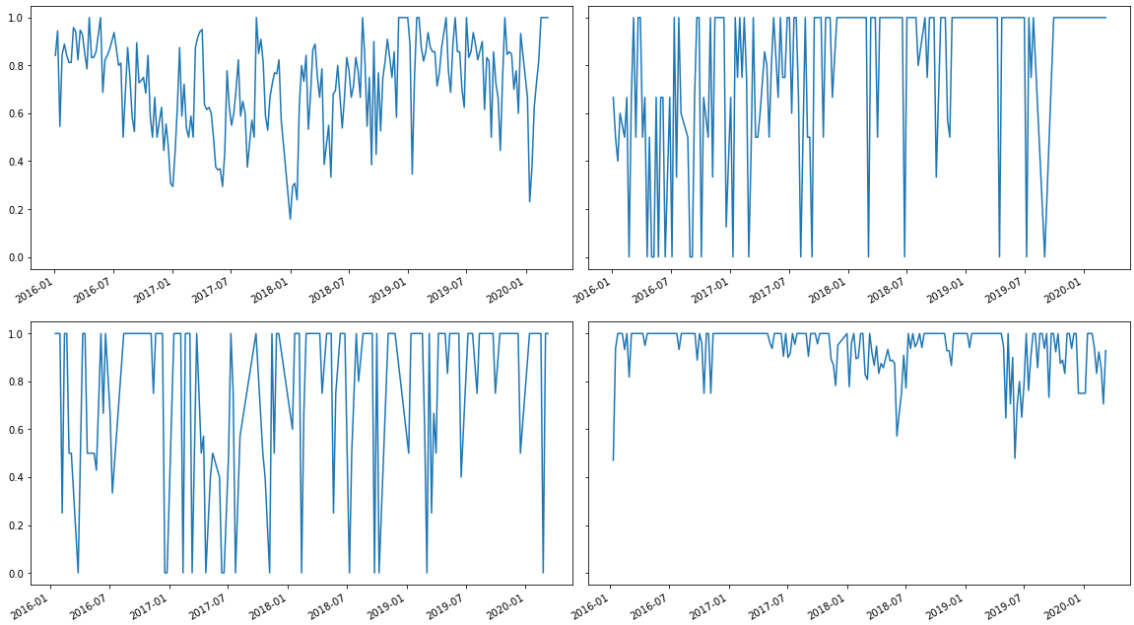

Figure 4.6: Examples of the delivery precision metric for a few different suppliers.

Static covariates, such as average lead time and the mean demand for the suppliers items, are included in the DeepAR model. Further, "age" of an observation—distance from first observation—together with the week of the year is included as time-dependent covariates.

5
Results

Three RNN models—MQRNN, DeepAR-feats and DeepAR-pure—have been trained to predict the future delivery performance of 2436 different suppliers to SML. In this chapter the predictions are evaluated against standard automatic forecasting tools and their business value is discussed.

5.1 Model evaluation

Judging the accuracy of a probabilistic forecast is slightly more difficult and less intuitive than evaluating a point forecast. Generally, when evaluating a point forecast, a measure of the distance between the forecast and the observed value such as absolute deviation $|\tilde{y}_t - \hat{y}_t|$ or mean square error $(\tilde{y}_t - \hat{y}_t)^2$ is used, where a smaller distance indicates a better forecast. Evaluating a forecast of the distribution of the target variable is less clear because there is only one sample from that particular distribution to judge it by. If the observed value is above the forecast 0.9-quantile for example, it can either mean the forecast is inaccurate, or that the 10% probability event happened.

One way to evaluate the accuracy of a number of probabilistic forecasts is by using the tilted absolute value function given in equation (4.3). It weighs the distance between the observed value and the quantile estimate differently depending on which quantile is estimated and if the observed value is greater or smaller. Let ρ_τ be the tilted absolute value function given in equation (4.3) and denote $\rho_\tau(y_t - \hat{y}_t^{(\tau)})$ as the τ-quantile loss of $\hat{y}_t^{(\tau)}$ on the random variable y_t. The sample mean τ-quantile loss $\bar{L}_\tau$ is then given by

$$\bar{L}_\tau = |\mathcal{S}|^{-1}|\mathcal{K}|^{-1}\sum_{j\in\mathcal{S}}\sum_{k\in\mathcal{K}}\rho_\tau(\tilde{y}_{t_0+k,j} - \hat{y}^{(\tau)}_{t_0+k,j}). \tag{5.1}$$

The basis for this measure is that $E(\rho_\tau(y_t - \hat{y}_t^{(\tau)}))$ is minimized when $\hat{y}_t^{(\tau)}$ equals the τ-quantile of y_t (Koenker and Hallock, 2001). Further, the median estimate, $\rho_{0.5}(\tilde{y}_t - \hat{y}_t^{(0.5)}) = 0.5|\tilde{y}_t - \hat{y}_t^{(0.5)}|$, is half the absolute deviation.

All the forecasting models are evaluated over $\tau \in \mathcal{Q} = \{0.1, 0.25, 0.5, 0.75, 0.9\}$; $\mathcal{S} := \{1, \ldots, 2436\}$ is the index set for the delivery performance time series and $\mathcal{K} := \{0, \ldots, 13\}$ is the forecast range. The MQRNN model gives upfront estimates of $[\hat{y}^{(\tau)}_{t+k}]_{k\in\mathcal{K},\tau\in\mathcal{Q}}$ for each $j \in \mathcal{S}$ while the DeepAR model outputs n sample paths and as such, to obtain $\hat{y}^{(\tau)}_{t+k}$, the sample observations for each $t_0 + k$ are sorted and the empirical τ-quantile is taken as the estimate.

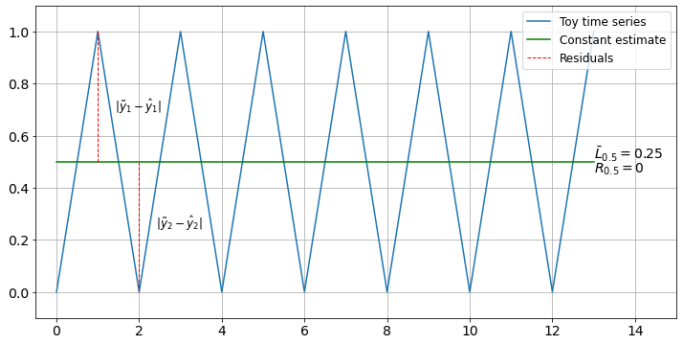

Figure 5.1: The τ-risk and mean quantile loss at $\tau = 0.5$ for a toy time series and a constant mean estimate.

A secondary measure, a variant of τ-risk introduced by Seeger, Salinas, and Flunkert (2016) (originally denoted ρ-risk), will additionally be used to evaluate the models. The sample τ-risk R_τ is given by

$$R_\tau = 2|\mathcal{S}|^{-1} \sum_{j \in \mathcal{S}} \rho_\tau(Z_j - \hat{Z}_j^{(\tau)}), \tag{5.2}$$

where $Z_j = \sum_{k \in K} \tilde{y}_{t_0+k,j}$. For the MQRNN model we set $\hat{Z}_j^{(\tau)} = \sum_{k \in \mathcal{K}} \hat{y}_{t+k,j}^{(\tau)}$ for all $j \in \mathcal{S}$. For the DeepAR model we first sum the individual sample paths and take the empirical τ-th quantile of them as the estimate.

The τ-risk is the measure preferred by Salinas et al. (2019) because it lends itself well to evaluating a model constructed around multiple sample paths. But evaluating over the summations Z_j, $j \in \mathcal{S}$, does not punish a sequence of estimates for missing peaks and nadirs to the same extent that mean quantile loss does. Figure 5.1 shows this for a toy time series that oscillates between zero and one at each step. A constant estimate $\hat{y}_t = 0.5$ for all t completely misses this periodicity but has $R_{0.5} = 0$; the mean quantile loss $\bar{L}_{0.5} = 0.25$ more accurately reflects the estimate's accuracy. For this reason focus lies on mean quantile loss and τ-risk is only include as a secondary measure.

The DeepAR model has been tested using just lagged observations of the target variable (DeepAR-pure) and using the two static covariates, mean forecast demand and mean lead time, together with the temporal covariates week-of-month, week-of-year, "age" of the observation and if a holiday falls into that week (DeepAR-feats). The models are compared to two automatic forecasting packages: Facebook Prophet (S. J. Taylor and Letham, 2017), and simple exponential smoothing (ETS) with the `forecast` package in R (Hyndman and Khandakar, 2008). All model's forecast accuracy is measured over the same 14 observations at the tail of each time series.

Overall the delivery performance shows high variability, with a mean intra-time series standard deviation $\bar{\sigma} = 0.2753$. To put this into perspective, the standard deviation of a uniform random variable with the same range as the delivery precision, $[0, 1]$, is 0.2887. For some of the time series, such as those shown in Figure 5.2 both models make similar predictions, while in others they make very different forecasts, as in Figure 5.3.

In table 5.1 the mean quantile loss and τ-risk at each quantile is shown for all five

Mean Quantile Loss					
	$\bar{L}_{0.1}$	$\bar{L}_{0.25}$	$\bar{L}_{0.5}$	$\bar{L}_{0.75}$	$\bar{L}_{0.9}$
MQRNN	0.0924	0.0980	0.0938	0.0818	0.0659
DeepAR-pure	0.0827	0.0880	**0.0817**	**0.0667**	0.0541
DeepAR-feats	0.0806	**0.0865**	0.0837	0.0672	**0.0538**
ETS	**0.0657**	0.0993	0.0828	0.0925	0.0602
Prophet	0.2101	0.2532	0.2675	0.2414	0.1980
τ-risk					
	$R_{0.1}$	$R_{0.25}$	$R_{0.5}$	$R_{0.75}$	$R_{0.9}$
MQRNN	1.7798	2.1463	2.0525	**1.7479**	**1.2405**
DeepAR-pure	2.1158	2.3287	2.4910	2.4690	2.4464
DeepAR-feats	2.1121	2.3383	2.5134	2.5293	2.5050
ETS	**1.4147**	**2.1113**	**1.8732**	2.2120	1.5089
Prophet	3.6020	4.5440	4.5292	4.4266	3.4673

Table 5.1: Accuracy metrics for the DeepAR-pure, DeepAR-feats, MQRNN as well as for two reference methods.

models—in both measures, lower scores indicate better forecast accuracy. At $\tau = 0.1$ the automatically fit ETS models performed far better than any other model on the mean quantile loss measure. At all other quantiles, the collection of ETS models was outperformed by at least one RNN model, with DeepAR-feats performing best $\bar{L}_{0.25}$ and DeepAR-pure on $\bar{L}_{0.5}$ and $\bar{L}_{0.75}$. DeepAR-pure scoring better than DeepAR-feats in two quantiles suggests that the included static covariates had limited predictive power. The collection of Facebook Prophet models performed the worst across the board. On the τ-risk measure the MQRNN model outperformed both DeepAR models in all quantiles, but overall the collection of ETS models had the lowest score in three out of five categories. Its comparatively high performance in the τ-risk measure may be due to the effect shown in Figure 5.1.

In Figure 5.4 we see that the MQRNN has learned a non-linear growth in uncertainty over the forecast range from the data, which the DeepAR model does no exhibit at all. This is due to imperfect hyperparameter tuning, as in the original paper (Salinas et al., 2019) growth in uncertainty over time is shown.

5.2 Business implications

The models developed in this thesis project showcases two concepts that can be of use to Volvo Group SML. The power of these concepts lays in their ability to be applied to a large domain of problems, and not just forecasting delivery precision.

The main concept in this thesis is the use of a neural network model to forecast a large number of time series. Using a global model that can learn interdependencies between time series could simplify a number of forecasting tasks done in the SML organisation today. One especially important strength of this approach is that a model that has learned these interdependencies can be used to create forecasts for new time series, where no previous observations exist. This is a domain in which

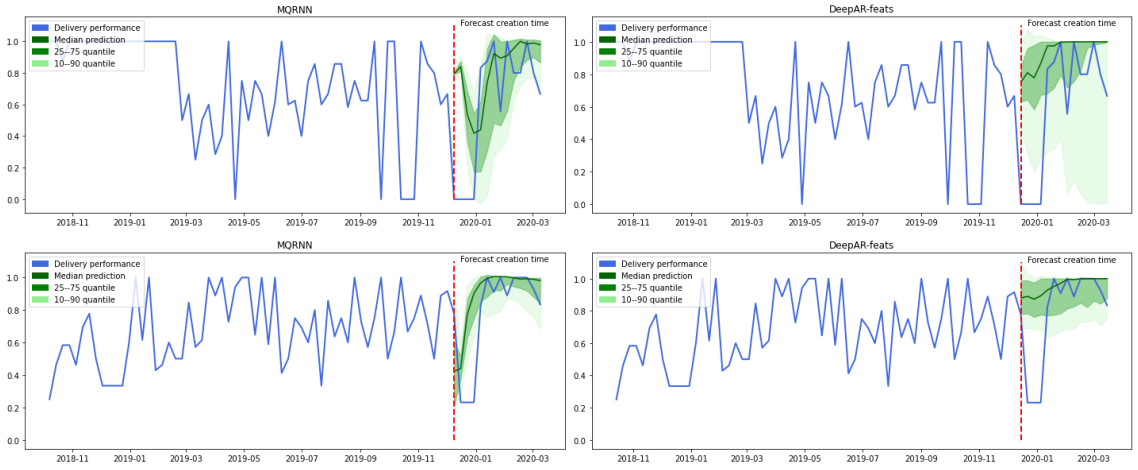

Figure 5.2: Side-by-side comparison of the forecasts from MQRNN and DeepAR-feats.

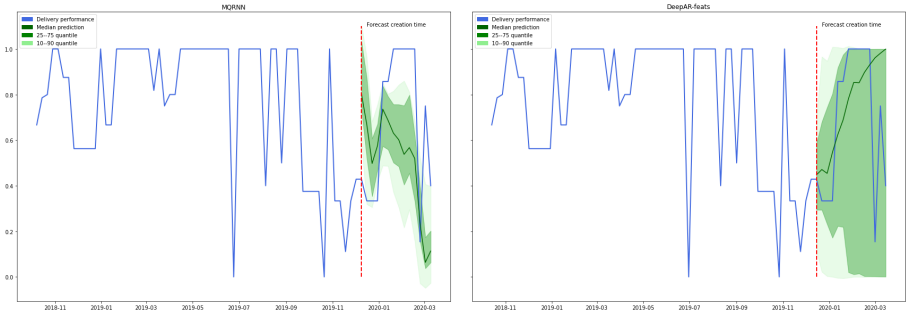

Figure 5.3: A time series where the MQRNN and the DeepAR-feats model's forecasts are dissimilar.

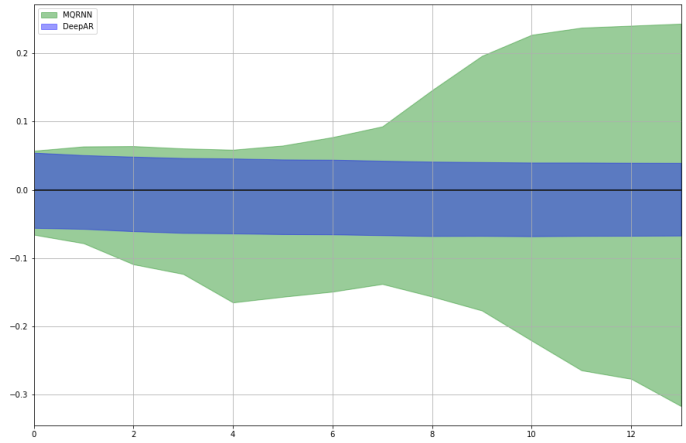

Figure 5.4: Mean uncertainty over the forecast range. Lower limit given by $|\mathcal{S}|^{-1} \sum_{s \in \mathcal{S}} \hat{y}_{t_0+k}^{(0.1)} - \hat{y}_{t_0+k}^{(0.5)}$; upper by $|\mathcal{S}|^{-1} \sum_{s \in \mathcal{S}} \hat{y}_{t_0+k}^{(0.9)} - \hat{y}_{t_0+k}^{(0.5)}$ for each $k \in \mathcal{K}$.

Mean Quantile Loss					
	$\bar{L}_{0.1}$	$\bar{L}_{0.25}$	$\bar{L}_{0.5}$	$\bar{L}_{0.75}$	$\bar{L}_{0.9}$
MQRNN	0.0603	0.0852	0.0895	0.0704	0.0453
DeepAR-feats	0.0782	0.0859	0.0832	0.0692	0.0562

Table 5.2: Model performance when training and testing models solely on pre-2020 data.

traditional forecasting methods struggle. The use of neural networks for forecasting in the automotive industry is not new—both the MQRNN model and the DeepAR model have been tested on public data on demand for automotive spare parts (Wen et al., 2017; Salinas et al., 2019); within Volvo Group SML some experimentation with feedforward networks has been done previously. These models in particular have useful applications within SML because they are built to handle *intermittency*, which is a typical feature in, for example, spare part demand.

The second outstanding concept in this thesis is the use of probabilistic forecasting. By modelling the full distribution of the target variable at each forecast step, a more complete set of information can be conveyed. This is useful both when this information is being used by a human analyst or when it is used by an AI system to make decisions. Thinking in terms of uncertainty instead of point forecasts is particularly advantageous when it comes to risk mitigation and is something we do naturally in our everyday lives. One of the most common probabilistic forecasts that we encounter is the weather forecast: "70% chance of rain" implies carrying an umbrella is a good idea as it is a relatively innocuous way of mitigating that risk. Probabilistic forecasting of the delivery performance of each supplier can give material planners a similar overview—a high chance of low delivery performance implies that adding buffer time or contacting the supplier may be good actions to take.

The external validity of these forecasting models may be significantly impacted by the Covid-19 pandemic. Covid-19 has had a devastating short-term effect on supply chains around the world (Choi, Rogers, and Vakil, 2020), the full extent of which is a fast-growing area of research—several leading operations and logistics journals have put out calls for papers on the impact of the virus (Journal of Operations Management, 2020; Production and Operations Management, 2020; International Journal of Production Research, 2020). The long-term ramifications of Covid-19 on supply chain design in terms of cost and resiliency are likely to be considerable (Kilpatrick and Barter, 2020; Schatteman, Woodhouse, and Terino, 2020), though uncertainty is high at this early stage.

The models are trained exclusively on historical data from before the pandemic, but as the test data is the last 14 weeks of available observations for each time series, a large portion of the predictions are over the period February–April 2020 in which it is likely that Covid-19 had an effect on delivery performance. Excluding data from 2020, retraining the models and testing them on the last 14 weeks of observations in each time series gives better mean quantile losses across the board, as shown in Table 5.2.

5.3 Exploratory Development for building AI

The Exploratory Development framework is designed to facilitate exploration with high uncertainty and allows for iteration through ideas before moving forward with the best one. In this way it worked quite well. This thesis pivoted through several iterations before landing in the use neural networks to predict delivery performance. Each pivot worked to clarify the scope and adjust the project to inputs from the AdA team and other stakeholders.

One of those iterations was done as a part of a PulseLab. The most valuable result of this was that the Fluid Teams dynamic, which opened the exploration to everyone that felt they could contribute, brought in constructive criticism of the approach and helped with the project's structure. One drawback however with the short format and split focus, where participants can be on several explorations, was that it felt difficult to collaborate on writing code because of the substantial upfront effort it takes.

An aspect of the framework that facilitated speed and decision-making in this thesis project was the availability of write-ups from previous explorations. Having this material to learn from helped with generating ideas in the early stage, discovering what data are available and guidance in the use of specific technologies. A pleasant side-effect of building a common and open knowledge base is that onboarding into the team becomes easier.

The explore phase works well with modern tools for developing neural network models such as GluonTS or Pytorch, as the modular design of these tools is ideal for quick prototyping. By abstracting away finicky implementation details and standardizing data structures they allow for quick testing of ideas. By embracing the use of cloud computing the AdA team has made sure that the necessary infrastructure is in place to efficiently train models.

One of the difficulties with developing AI applications at Volvo Group SML with the Exploratory Development framework lies in collaborating with other teams. By the Volvo Group definition of AI it needs to make decisions in the real world, and thus goes beyond being an analytics tool or recommendation engine. This makes the need for integrating existing stakeholders and current owners of the given decision even more prevalent.

A final sidenote is that the framework has held up as a methodology in the face of vastly changing circumstances. Working in a way that welcomes change has made it easier to adapt the thesis project to the changes that have been instituted by Volvo Group during the projects runtime.

5.4 Conclusion

Both neural network models performed roughly on par with the much simpler local ETS models for predicting delivery precision, but far better than the Prophet model. The DeepAR model shows the most promise for forecasting delivery performance based on the the mean quantile loss measure, although the MQRNN model displays more realistic growth in the uncertainty over time. The approach of training

global models for probabilistic forecasting has a lot of promise in other applications within SML, with examples being demand forecasting and backorder prediction. The DeepAR model shows promise for forecasting intermittent time series, which is a prominent feature in many forecasting situations that are encountered at SML.

Further research should be done on using the the MQRNN model with covariates, both static and temporal. The model is actively being developed by researchers at AWS, which maintains the GluonTS platform, and will in the future allow for easy inclusion of covariates to the target variable. The next step for forecasting delivery precision with the DeepAR models is to include more temporal covariates such as holidays specific to each supplier's country of operation, number of different products they serve Volvo Group with on a regular basis, and the value of the products purchased each week.

Bibliography

Abadi, Martın et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Tech. rep. Google Research.

Alexandrov, Alexander et al. (2019). *GluonTS: Probabilistic Time Series Models in Python*. Tech. rep. Amazon Web Services.

Beck, Kent et al. (2001). *Manifesto for Agile Software Development*. URL: `http://www.agilemanifesto.org/` (visited on 01/22/2020).

Bertsimas, Dimitris, Nathan Kallus, and Amjad Hussain (2016). "Inventory Management in the Era of Big Data". In: *Productions and Operations Management* 25 (12), pp. 2002–2013.

Box, G.E.B. and G.M. Jenkins (1970). *Time Series Analysis: Forecasting and Control*. Holden-Day Series in Time Series Analysis. Holden Day.

Breiman, Leo (2001). "Random forests". In: *Machine Learning* 45, pp. 5–32.

Brockwell, Peter J. and Richard A. Davis (2016). *Introduction to Time Series and Forecasting*. 3rd ed. Springer.

Brown, Robert G. (1956). *Exponential Smoothing for Predicting Demand*. Tech. rep. Cambridge, Massachusetts: Arthur D. Little, Inc.

Campanelli, Amadeu Silveira and Fernando Silva Parreiras (2015). "Agile methods tailoring — A systematic literature review". In: *The Journal of Systems and Software* 110, pp. 85–100.

Cannon, Alex J. (2011). "Quantile regression neural networks: Implementation in R and application to precipitation downscaling". In: *Computers & Geosciences* 37, pp. 1277–1284.

Carbonneau, Real, Kevin Laframboise, and Rustam Vahidov (2008). "Application of machine learning techniques of supply chain demand forecasting". In: *European Journal of Operational Research* 184, pp. 1140–1154.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics.

Choi, Thomas Y., Dale Rogers, and Bindiya Vakil (2020). "Coronavirus Is a Wake-Up Call for Supply Chain Management". In: *Harvard Business Review* 3. URL: `https://hbr.org/2020/03/coronavirus-is-a-wake-up-call-for-supply-chain-management`.

Cohen, Morris A., Narendra Agrawal, and Vipul Agrawal (2006). "Winning in the Aftermarket". In: *Harvard Business Review* (5).

Cortes, Corinna and Vladimir Vapnik (1995). "Support-vector networks". In: *Machine Learning* 20, pp. 273–297.

Crone, Sven F., Michele Hibon, and Konstantinos Nikolopoulos (2011). "Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction". In: *International Journal of Forecasting* 27, pp. 635–660.

Davis, Tom (1993). "Effective Supply Chain Management". In: *MIT Sloan Management Review* 7.

Dean, Jeffrey et al. (2012). "Large Scale Distributed Deep Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1223–1231. URL: http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf.

Deemer, Pete et al. (2010). *The Scrum Primer*. InfoQ.

Fisher, Marshall L. (1997). "What Is the Right Supply Chain for Your Product?" In: *Harvard Business Review* 75 (2), pp. 105–116.

Galbraith, Jay R. (1974). "Organization Design: An Information Processing View". In: *Interfaces* 4.3, pp. 28–36.

Galicia, A. et al. (2019). "Multi-step forecasting for big data time series based on ensemble learning". In: *Knowledge-Based Systems* 163, pp. 830–841.

Gantz, John and David Reinsel (2012). *The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East*. Tech. rep. EMC Corporation. URL: https://www.speicherguide.de/download/dokus/IDC-Digital-Universe-Studie-iView-11.12.pdf.

Gent, C. R. and C. P. Sheppard (1992). "Predicting time series by a fully connected neural network trained by back propagation". In: *Computing & Control Engineering Journal* 3 (3).

Gers, Felix A., Jurgen Schmidhuber, and Fred Cummins (Jan. 1999). *Learning to Forget: Continual Prediction with LSTM*. Tech. rep. Lugano, Switzerland: IDSIA.

Gooijer, Jan G. De and Rob J. Hyndman (2006). "25 years of time series forecasting". In: *International Journal of Forecasting* 22, pp. 443–473.

Graves, Alex (2013). "Generating Sequences With Recurrent Neural Networks". In: *CoRR* abs/1308.0850. arXiv: 1308.0850. URL: http://arxiv.org/abs/1308.0850.

Gupta, Anshuman, Costas D. Maranas, and Conor M. McDonald (2000). "Mid-term supply chain planning under demand uncertainty: customer demand satisfaction and inventory management". In: *Computers and Chemical Engineering* 24, pp. 2613–2621.

Herer, Yale T., Michal Tzur, and Enver Yucesan (2002). "Transshipments: An emerging inventory recourse to achieve supply chain leagility". In: *International Journal of Production Economics* 80, pp. 201–212.

Hewamalage, Hansika, Christoph Bergmeir, and Kasun Bandara (Sept. 2019). "Recurrent Neural Networks for Times Series Forecasting: Current Status and Future Directions". In:

Heydari, Jafar, Reza Baradaran Kazemzadeh, and S. Kamal Chaharsooghi (2009). "A study of lead time variation impact on supply chain performance". In: *International Journal of Manufacturing Technology* 40, pp. 1206–1215.

Hochreiter, Sepp and Jurgen Schmidhuber (1997). "Long short-term memory". In: *Neural Computation* 9.8, pp. 1735–1780.

Hoda, Rashina, James Noble, and Stuart Marshall (Mar. 2013). "Self-Organizing Roles on Agile Software Development Teams". In: *IEEE Transactions on Software Engineering* 39.3, pp. 422–444.

Holt, Charles C. (1957). "Forecasting seasonals and trends by exponentially weighted moving averages". In: *repor to the Office of Naval Research* 52.

Hu, M. J. C. and Halbert E. Root (1964). "An Adaptive Data Processing System for Weather Forecasting". In: *Journal of Applied Meteorology* 3.5, pp. 513–523.

Huang, Meng and Masood Bagheri (2019). "Predicting Deviation in Supplier Lead Time and Truck Arrival Time Using Machine Learning". MA thesis. Chalmers University of Technology.

Hyndman, Rob J. and George Athanasopoulos (2018). *Forecasting: Principles and Practice*. 2nd ed. Melbourne, Australia: OTexts.

Hyndman, Rob J. and Yeasmin Khandakar (2008). "Automatic Time Series Forecasting: The forecast Package for R". In: *Journal of Statistical Software* 27 (3). URL: https://www.jstatsoft.org/index.php/jss/article/view/v027i03/v27i03.pdf.

Exploring the Sources of Waste in Kanban Software Development Projects (Sept. 2010), pp. 376–381.

International Journal of Production Research (Apr. 2020). *Special Issue: Viability of Supply Networks and Ecosystems: Lessons Learned From COVID-19 Outbreak*. URL: https://techjournals.wixsite.com/techjournals/viability-supply-networks-ecosystem.

Jones, Thomas C. and Daniel W. Riley (1985). "Using Inventory for Competitive Advantage Through Supply Chain Management". In: *International Journal of Physical Distribution & Materials Management* 15.5.

Journal of Operations Management (Apr. 2020). *Call for Papers: The Effects of COVID-19 on Global Supply Chains: Responsiveness, Resilience, and Restoration (3Rs)*. URL: https://onlinelibrary.wiley.com/journal/18731317.

Kilpatrick, Jim and Lee Barter (Apr. 2020). *COVID-19: Managing supply chain risk and disruption*. Deloitte. URL: https://www2.deloitte.com/global/en/pages/risk/articles/covid-19-managing-supply-chain-risk-and-disruption.html.

Kingma, Diederik P. and Jimmy Lei Ba (2015). "Adam: A Method for Stochastic Optimization". In: *ICLR*. URL: https://arxiv.org/abs/1412.6980.

Koenker, Roger and Gilbert Basset (1978). "Regression Quantiles". In: *Econometrica* 46.1, pp. 33–50.

Koenker, Roger and Kevin F. Hallock (2001). "Quantile Regression". In: *Journal of Economic Perspectives* 15.4, pp. 143–156.

Koenker, Roger and Beum J. Park (1996). "An interior point algorithm for nonlinear quantile regression". In: *Journal of Econometrics* 71, pp. 265–283.

Kurapati, Narendra, Venkata Sarath Chandra Manyam, and Kai Petersen (2012). "Agile Software Development Practice Adoption Survey". In: *Agile Processes in Software Engineering and Extreme Programming*. Ed. by Claes Wohlin. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 16–30.

Lang, Annika and Andreas Petersson (2017). *Financial Time Series*. Chalmers University of Technology.

Liker, Jeffrey K. (2004). *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer*. McGraw-Hill.

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos (2020). "The M4 Competition: 100 000 time series and 61 forecasting methods". In: *International Journal of Forecasting* 36, pp. 54–74.

Marcellino, Massimiliano, James H. Stock, and Mark W. Watson (2006). "A comparision of direct and iterated multistep AR methods for forecasting macroeconomic time series". In: *Journal of Econometrics* 135, pp. 499–526.

Marsland, Stephen (2015). *Machine Learning: An Algorithmic Perspective*. 2nd ed. Boca Raton, USA: CRC Press.

Martin, James (1991). *Rapid Application Development*. Macmillan Publishing Company.

Min, Hokey (Feb. 2010). "Artificial intelligence in supply chain management: theory and applications". In: *International Journal of Logistics: Research and Applications* 13.1, pp. 13–39.

Moraga, Claudio (2017). *Claudio Moraga: A Passion for Multi-Valued Logic and Soft Computing*. Ed. by Rudolf Seising and Hector Allende. Cham, Switzerland: Springer.

Muckstadt, John A. (2005). *Analysis and Algorithms for Service Part Supply Chains*. Springer.

Nielsen, Michael (2015). *Neural Networks and Deep Learning*. Determination Press. URL: http://neuralnetworksanddeeplearning.com.

Parmezan, Antonio Rafael Sabino and Vinicius M.A. Souza (2019). "Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model". In: *Information Sciences* 484, pp. 302–337.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Vancouver, Canada: Neural Information Processing Systems.

Peidro, David et al. (2009). "Quantitative models for supply chain planning under uncertainty: a review". In: *The International Journal of Advanced Manufacturing Technology volume* 43, pp. 400–420.

Poppendieck, Mary and Tom Poppendieck (May 2003). "Lean Software Development: An Agile Toolkit". In:

Production and Operations Management (Apr. 2020). *Production and Operations Management Call for Papers Special Issue: Managing Pandemics: A POM Perspective*. URL: https://www.poms.org/journal/announcements/.

Raina, Rajat, Anand Madhavan, and Andrew Y. Ng (2009). "Large-Scale Deep Unsupervised Learning Using Graphics Processors". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, pp. 873–880. ISBN: 9781605585161. DOI: 10.1145/1553374.1553486. URL: https://doi.org/10.1145/1553374.1553486.

Reiss, Eric (2011). *The Lean Startup*. Random House.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323.9, pp. 533–536.

Rushton, Alan, Phil Croucher, and Peter Baker (2014). *The Handbook of Logistics and Distribution Management*. 5th ed. London, UK: Kogan Page.

Salinas, David et al. (2019). "DeepAR: Probabilistic forecasting with autoregressive recurrent networks". In: *International Journal of Forecasting*. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2019.07.001. URL: http://www.sciencedirect.com/science/article/pii/S0169207019301888.

Schatteman, Olaf, Drew Woodhouse, and Joe Terino (Apr. 2020). *Supply Chain Lessons from Covid-19: Time to Refocus on Resilience*. Bain & Company. URL: https://www.bain.com/insights/supply-chain-lessons-from-covid-19/.

SCRUM Development Process (1995), pp. 117–134.

Schwaber, Ken and Jeff Sutherland (Nov. 2017). *The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game*. scrumguides.org.

Seeger, Matthias W, David Salinas, and Valentin Flunkert (2016). "Bayesian Intermittent Demand Forecasting for Large Inventories". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., pp. 4646–4654. URL: http://papers.nips.cc/paper/6313-bayesian-intermittent-demand-forecasting-for-large-inventories.pdf.

Sen, Rajat, Hsiang-Fu Yu, and Inderjit Dhillon (2019). "Think Globally, Act Locally: A Deep Neural Network Approach to High-Dimensional Time Series Forecasting". In: *Advances in Neural Information Processing Systems 32*. Vancouver, Canada: Neural Information Processing Systems Foundation.

Simangunsong, Eliot, Linda Hendry, and Mark Stevenson (Aug. 2012). "Supply Chain Uncertainty: A Review and Theoretical Foundation for Future Research". In: *International Journal of Production Research* 50, pp. 4493–4523. DOI: 10.1080/00207543.2011.613864.

Simonyan, Karen and Andrew Zisserman (2015). "Very deep convolutional networks for large-scale image recognition". In: *Conference Paper at International Conference on Learning Representations*.

Sletholt, Magnus Thorstein et al. (2011). "A Literature Review of Agile Practices and Their Effects in Scientific Software Development". In: *4th international workshop on Software engineering for computational science and engineering (SECSE '11)*. Association for Computing Machinery (ACM).

Smyl, Slawek (2020). "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting". In: *International Journal of Forecasting* 36, pp. 75–85.

Sorjamaa, Antti et al. (2007). "Methodology for long-term prediction of time series". In: *Neurocomputing* 70, pp. 2861–2869.

Stray, Viktoria, Dag I.K. Sjøberg, and Tore Dybå (2016). "The daily stand-up meeting: A grounded theory study". In: *The Journal of Systems and Software* 114, pp. 101–124.

Sutskever, Ilya, Oriol Vinyals, and Quoc Le (Sept. 2014). "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems* 4.

Taieb, Souhaib Ben, Antti Sorjamaa, and Gianluca Bontempi (2010). "Multiple-output modeling for multi-step-ahead time series forecasting". In: *Neurocomputing* 73, pp. 1950–1957.

Taylor, James W. (2000). "A Quantile Regression Neural Network Approach to Estimating the Conditional Density of Multiperiod Returns". In: *Journal of Forecasting* 19, pp. 299–311.

Taylor, Sean J. and Benjamin Letham (2017). "Forecasting at Scale". In: *The American Statistician* 72 (1), pp. 37–45. URL: https://doi.org/10.1080/00031305.2017.1380080.

Timme, Stephen G. (July 2003). "The real cost of holding inventory". In: *Supply Chain Management Review* 7.4, pp. 30–38.

Tsay, Ruey S. (June 2000). "Time Series and Forecasting: Brief History and Future Research". In: *Journal of the American Statistical Association* 95.450, pp. 638–643.

Tucker, Jennifer (2014). "Human-Centered System Development". In: *Computing Handbook Third Edition: Information Systems and Information Technology*. Ed. by Heikki Topi and Allen Tucker. CRC Press. Chap. 29.

Volvo Group (2017). "Your Ideas Matter: creating a culture of continuous improvement". In: *Volvo Group Magazine* 2.

Wen, Ruofeng et al. (2017). "A Multi-Horizon Quantile Recurrent Forecaster". In: *31st Conference on Neural Information Processing Systems (NIPS 2017), Time Series Workshop*. Neural Information Processing Systems Foundation.

Werbos, Paul J. (1988). "Generalization of Backpropagation with Application to a Recurrent Gas Market Model". In: *Neural Networks* 1, pp. 339–356.

White, Halbert (1992). "Nonparametric Estimation of Conditional Quantiles Using Neural Networks". In: *Computing Science and Statistics*. Ed. by Connie Page and Raoul LePage. Springer New York, pp. 190–199.

Winters, Peter R. (1960). "Forecasting Sales by Exponentially Weighted Moving Averages". In: *Management Science* 6.3, pp. 324–342.

Yule, Udny (1927). "On a Method of Investigating Periodicities in Disturbed Series, with Special Reference to Wolfer's Sunspot Numbers". In: *Philosophical Transactions of the Royal Society London*. A 226, pp. 267–297.

Zhang, Guoqiang, B. Eddy Patuwo, and Michael Y. Hu (1998). "Forecasting with artificial neural networks: The state of the art". In: *International Journal of Forecasting* 14, pp. 35–62.

A

Appendix

A.1 Data augmentation

```
"""
data_augmentation.py
"""
import pandas as pd
import numpy as np
import pickle
from gluonts.dataset.field_names import FieldName

import holidays
from datetime import date



# Auxiliary functions
def str_strip(series: pd.Series) -> pd.Series:
    return series.astype(str).str.strip()

def yearweek_to_date(series: pd.Series) -> pd.Series:
    """Casts a column of yyyyww entries to dates using the monday of each week"""
    return pd.to_datetime(series.astype(str) + '1', format='%G%V%u')

# Read the data
def parse_dp(base_path: str) -> pd.DataFrame:
    """
    Parses the excel spreadsheets at 'base_path' into a useable
    dataframe containing time series of delivery performance for
    all suppliers.
    """
    append = "_MM_last.xlsx"

    df_lst = []
    df_lst.append(
        pd.read_excel(base_path + "2016" + append,
                      sheet_name="database",
                      usecols=np.arange(7))
    )
    for yr in ['2017', '2018', '2019', '2020']:
        df_lst.append(
            pd.read_excel(base_path + yr + append,
```

```
                                sheet_name="database",
                                skiprows=1,
                                usecols=np.arange(7))
            )

    df = (pd.concat(df_lst, axis=0)
            .dropna())

    # strip blank spaces and cast to str
    df.loc[:, 'Parma'] = str_strip(df['Parma'])
    df.loc[:, 'Supplier Name'] = str_strip(df['Supplier Name'])
    df.loc[:, 'Brand'] = str_strip(df['Brand'])
    df.loc[:, 'Parma'] = df['Parma'].str.replace(".0", "", regex=
    False)

    # cast to datetime entries W-Mon
    df['WEEK'] = yearweek_to_date(df['WEEK'].astype(int))

    df = df.sort_values(by=['Parma', 'WEEK'])

    # group across Penta and VT and calculate overall DP
    df = (df.groupby(['WEEK', 'Parma'])
            [['Order Lines Nb', 'KO']]
            .sum()
            .reset_index())
    df['DP'] = 1 - (df['KO'] / df['Order Lines Nb'])

    return df


def parse_lt_fc(path_to_file: str) -> pd.DataFrame:
    """
    Parses spreadsheet of average leadtime and forecast demand
    per supplier.
    """

    df = pd.read_excel(path_to_file, sheet_name='Database')
    # strip blank spaces and cast to str
    for col in df.dtypes[df.dtypes == 'object'].index:
        df[col] = str_strip(df[col])

    # fix a '.' and ',' issue
    df['FC Total (per month)'] = (df['FC Total (per month)']
                                  .str.replace(',', '.', regex=
    False)
                                  .astype('float64'))

    return (df.groupby('Main Supplier No')
              [["LT (Working days)", "FC Total (per month)"]]
              .mean())


def holidays_belgium() -> pd.DataFrame:
    """
    Create a time series counting the number of holiday days
    in Belgium in a given week.
```

```
94      """
95      belg_holidays = holidays.Belgium()
96      holiday_dates = []
97      for year in range(2013, 2021):
98          holiday_dates += belg_holidays[date(year, 1, 1):date(year
    +1, 1, 1)]
99
100     idx = pd.date_range(start='2013-01-01', end='2020-12-31', freq=
    'D')
101     data = {'holidays_during_week': 0, 'holidays_during_next_week':
     0}
102     ts_holidays = pd.DataFrame(index=idx, data=data)
103
104     ts_holidays.loc[holiday_dates, 'holidays_during_week'] = 1.0
105     ts_holidays = ts_holidays.resample('W-Mon', convention='end').
    sum()
106     ts_holidays['holidays_during_next_week'] = ts_holidays['
    holidays_during_week'].shift(-1)
107     ts_holidays = ts_holidays.iloc[:-1]
108
109     return ts_holidays
110
111
112 def create_data_folder(path_to_folder) -> None:
113     """
114     Creates or fills a folder at 'path_to_folder' with .parquet
    files
115     of the augmented data.
116     """
117     import os
118
119     path_to_dp = "V:\\DP"
120     path_to_lt_fc = "V:\\Average_LT_per_supplier.xlsx"
121
122     try:
123         os.makedirs(path_to_folder, exist_ok=True)
124     except OSError:
125         print ("Creation of the directory %s failed" %
    path_to_folder)
126         return
127     else:
128         print ("Successfully created the directory %s" %
    path_to_folder)
129
130     parse_dp(path_to_dp).to_parquet(path_to_folder + '\\dp.parquet'
    )
131     parse_lt_fc(path_to_lt_fc).to_parquet(path_to_folder + '\\lt_fc
    .parquet')
132     holidays_belgium().to_parquet(path_to_folder + '\\
    holidays_belgium.parquet')
133
134
135 def gluonts_data_format(
136         path_to_data_folder: str = None,
137         num_test_obs: int = 14
138     ) -> None:
```

```
    """
    Creates training and test datasets that can be input into
    gluonts models and pickles them in 'path_to_data_folder'.
    """
    FREQ = "W-Mon"
    if path_to_data_folder is None:
        path_to_data_folder = 'masters_thesis_data'
        create_data_folder(path_to_data_folder)

    dp = pd.read_parquet(path_to_data_folder + '\\dp.parquet')
    lt_fc = pd.read_parquet(path_to_data_folder + '\\lt_fc.parquet')
    holidays_ts = pd.read_parquet(path_to_data_folder + '\\holidays_belgium.parquet')

    lt_fc.index.name = 'Parma'
    MEAN_LT, MEAN_FC = lt_fc.mean()

    datasets = {
        'deepar_train': [],
        'deepar_test' : [],
        'mqrnn_train' : [],
        'mqrnn_test'  : []
    }

    for key in dp['Parma'].unique():
        try:
            mean_lt, mean_fc = lt_fc.loc[key]
        except:
            mean_lt, mean_fc = MEAN_LT, MEAN_FC
        subframe = dp[dp['Parma'] == key]
        ts = (subframe[['WEEK', 'DP']].set_index('WEEK')
                                      .resample(FREQ)
                                      .ffill()
              )
        if len(ts) >= num_test_obs:
            train_entry = {
                FieldName.TARGET: (ts.values
                                     .flatten()
                                     [:-num_test_obs]
                                   ),
                FieldName.START: ts.index[0],
                FieldName.ITEM_ID: key,
                FieldName.FEAT_STATIC_REAL: [mean_lt, mean_fc],
                FieldName.FEAT_DYNAMIC_REAL:
                    (holidays_ts.loc[ts.index.values]
                                .values
                                .transpose()
                                [:, :-num_test_obs]
                    )
            }
            test_entry = {
                FieldName.TARGET: (ts.values
                                     .flatten()
                                   ),
                FieldName.START: ts.index[0],
```

```
193                 FieldName.ITEM_ID: key,
194                 FieldName.FEAT_STATIC_REAL: [mean_lt, mean_fc],
195                 FieldName.FEAT_DYNAMIC_REAL:
196                     (holidays_ts.loc[ts.index.values]
197                                 .values
198                                 .transpose()
199                     )
200             }
201 
202             # mqrnn model cannot currently handle covariates
203             mqrnn_train_entry = train_entry.copy()
204             mqrnn_train_entry.pop(FieldName.FEAT_STATIC_REAL)
205             mqrnn_train_entry.pop(FieldName.FEAT_DYNAMIC_REAL)
206             mqrnn_test_entry = test_entry.copy()
207             mqrnn_test_entry.pop(FieldName.FEAT_STATIC_REAL)
208             mqrnn_test_entry.pop(FieldName.FEAT_DYNAMIC_REAL)
209 
210             datasets['deepar_train'].append(train_entry)
211             datasets['deepar_test'].append(test_entry)
212             datasets['mqrnn_train'].append(mqrnn_train_entry)
213             datasets['mqrnn_test'].append(mqrnn_test_entry)
214 
215     for name, lst in datasets.items():
216         with open(path_to_data_folder + '\\' + name + '.pickle', '
    wb') as h:
217             pickle.dump(lst, h, protocol=pickle.HIGHEST_PROTOCOL)
218 
219 if __name__ == '__main__':
220     gluonts_data_format('testing_data_function')
```

A.2 Model training

```
"""
model_training.py
"""
import mxnet as mx
from mxnet import gluon
from gluonts.model.deepar import DeepAREstimator
from gluonts.model.seq2seq import MQRNNEstimator
from gluonts.trainer import Trainer

from gluonts.dataset.field_names import FieldName
from gluonts.transform import (
    AddAgeFeature,
    AddObservedValuesIndicator,
    AddTimeFeatures,
    AsNumpyArray,
    Chain,
    ExpectedNumInstanceSampler,
    InstanceSplitter,
    RemoveFields,
    SetField,
    Transformation,
    VstackFeatures,
)

import pickle
from pathlib import Path
import os

path_to_data_folder = 'testing_data_function'
NUM_TEST_OBS = 14
FREQ = "W-Mon"

# read the data
with open(path_to_data_folder + '\\deepar_train.pickle', 'rb') as h
    :
    deepar_train = pickle.load(h)

with open(path_to_data_folder + '\\mqrnn_train.pickle', 'rb') as h:
    mqrnn_train = pickle.load(h)

# DeepAR model
class ModifiedEstimator(DeepAREstimator):

    def __init__(
        self,
        prediction_length,
        freq, scaling,
        use_feat_static_real,
        use_feat_dynamic_real,
        trainer
    ):
        super().__init__(
            prediction_length=prediction_length,
```

```
53              freq=freq,
54              scaling=scaling,
55              use_feat_static_real=use_feat_static_real,
56              use_feat_dynamic_real=use_feat_dynamic_real,
57              trainer=trainer)
58
59      def create_transformation(self):
60          remove_field_names = [FieldName.FEAT_DYNAMIC_CAT]
61          if not self.use_feat_static_real:
62              remove_field_names.append(FieldName.FEAT_STATIC_REAL)
63          if not self.use_feat_dynamic_real:
64              remove_field_names.append(FieldName.FEAT_DYNAMIC_REAL)
65
66          return Chain(
67              [RemoveFields(field_names=remove_field_names)]
68              + (
69                  [SetField(output_field=FieldName.FEAT_STATIC_CAT,
    value=[0.0])]
70                  if not self.use_feat_static_cat
71                  else []
72              )
73              + (
74                  [
75                      SetField(
76                          output_field=FieldName.FEAT_STATIC_REAL,
    value=[0.0]
77                      )
78                  ]
79                  if not self.use_feat_static_real
80                  else []
81              )
82              + [
83                  AsNumpyArray(
84                      field=FieldName.FEAT_STATIC_CAT,
85                      expected_ndim=1,
86                      dtype=self.dtype,
87                  ),
88                  AsNumpyArray(
89                      field=FieldName.FEAT_STATIC_REAL,
90                      expected_ndim=1,
91                      dtype=self.dtype,
92                  ),
93                  AsNumpyArray(
94                      field=FieldName.TARGET,
95                      # in the following line, we add 1 for the time
    dimension
96                      expected_ndim=1 + len(self.distr_output.
    event_shape),
97                      dtype=self.dtype,
98                  ),
99                  AddObservedValuesIndicator(
100                     target_field=FieldName.TARGET,
101                     output_field=FieldName.OBSERVED_VALUES,
102                     dummy_value=self.distr_output.value_in_support,
103                     dtype=self.dtype,
104                 ),
```

```
                AddTimeFeatures(
                    start_field=FieldName.START,
                    target_field=FieldName.TARGET,
                    output_field=FieldName.FEAT_TIME,
                    time_features=self.time_features,
                    pred_length=self.prediction_length,
                ),
                AddAgeFeature(
                    target_field=FieldName.TARGET,
                    output_field=FieldName.FEAT_AGE,
                    pred_length=self.prediction_length,
                    log_scale=True,
                    dtype=self.dtype,
                ),
                VstackFeatures(
                    output_field=FieldName.FEAT_TIME,
                    input_fields=[FieldName.FEAT_TIME, FieldName.FEAT_AGE]
                    + (
                        [FieldName.FEAT_DYNAMIC_REAL]
                        if self.use_feat_dynamic_real
                        else []
                    ),
                ),
                InstanceSplitter(
                    target_field=FieldName.TARGET,
                    is_pad_field=FieldName.IS_PAD,
                    start_field=FieldName.START,
                    forecast_start_field=FieldName.FORECAST_START,
                    train_sampler=ExpectedNumInstanceSampler(num_instances=10),
                    past_length=self.history_length,
                    future_length=self.prediction_length,
                    time_series_fields=[
                        FieldName.FEAT_TIME,
                        FieldName.OBSERVED_VALUES,
                    ],
                    dummy_value=self.distr_output.value_in_support,
                ),
            ]
        )

# the three estimators
mqrnn_estimator = MQRNNEstimator(
    prediction_length=NUM_TEST_OBS,
    freq=FREQ,
    quantiles=[0.1, 0.25, 0.5, 0.75, 0.9],
    mlp_hidden_dimension_seq=[14, 14],
    trainer=Trainer(ctx="cpu",
                    epochs=300,
                    learning_rate=1e-3,
                    num_batches_per_epoch=1000
                    )
)

deeparfeats_estimator = ModifiedEstimator(
```

```
    prediction_length=NUM_TEST_OBS,
    freq='W',
    scaling=False,
    use_feat_static_real=True,
    use_feat_dynamic_real=True,
    trainer=Trainer(ctx="cpu",
                    epochs=300,
                    learning_rate=1e-3,
                    num_batches_per_epoch=1000)
)

deeparpure_estimator = ModifiedEstimator(
    prediction_length=NUM_TEST_OBS,
    freq='W',
    scaling=False,
    use_feat_static_real=False,
    use_feat_dynamic_real=False,
    trainer=Trainer(ctx="cpu",
                    epochs=300,
                    learning_rate=1e-3,
                    num_batches_per_epoch=1000)
)

# train the models
mqrnn_predictor = mqrnn_estimator.train(mqrnn_train)
deeparpure_predictor = deeparpure_estimator.train(deepar_train)
deeparfeats_predictor = deeparfeats_estimator.train(deepar_train)

# serialize the models
os.makedirs(path_to_data_folder + '\\deeparpure', exist_ok=True)
deeparpure_predictor.serialize(Path(path_to_data_folder + '\\
    deeparpure\\'))

os.makedirs(path_to_data_folder + '\\deeparfeats', exist_ok=True)
deeparfeats_predictor.serialize(Path(path_to_data_folder + '\\
    deeparfeats\\'))

os.makedirs(path_to_data_folder + '\\mqrnn', exist_ok=True)
mqrnn_predictor.serialize(Path(path_to_data_folder + '\\mqrnn\\'))
```

A.3 Model evaluation

```
"""
model_evaluation.py
"""
from pathlib import Path
import pickle
import numpy as np
import pandas as pd

import mxnet as mx
from mxnet import gluon
from gluonts.evaluation.backtest import make_evaluation_predictions
from gluonts.model.forecast import SampleForecast, QuantileForecast
from gluonts.model.predictor import Predictor

path_to_data_folder = 'testing_data_function'

# Metrics
def quantile_loss(quantile_forecast, target, tau):
    return np.sum(
        np.abs(
            (quantile_forecast - target)
            * ((target <= quantile_forecast) - tau)
        )
    )


def tau_risk(fc_lst, ts_lst, tau):
    """
    Follows exactly the approach of Seeger, Salinas and Flunkert
    (2016)
    """
    ql = 0
    for ts, fc in zip(ts_lst, fc_lst):
        # Extract the target values
        pred_target = np.atleast_1d(
            np.squeeze(ts.loc[fc.index])).transpose()
        pred_target = np.array(pred_target)
        pred_target = np.ma.masked_invalid(pred_target)

        Z = pred_target.sum()

        if type(fc) == QuantileForecast:
            # MQRNN
            # get sum of estimated quantiles
            Z_hat = fc.quantile(tau).sum()
        elif type(fc) == SampleForecast:
            # DeepAR
            sorted_sample = np.sort(fc.samples.sum(axis=1))
            sample_idx = int(np.round(len(sorted_sample)-1)*tau)
            Z_hat = sorted_sample[sample_idx]

        ql += 2*quantile_loss(Z, Z_hat, tau)
    return ql / len(ts_lst)
```

```
53
54
55 def mean_quantile_loss(fc_lst, ts_lst, tau):
56     ql = 0
57     for ts, fc in zip(ts_lst, fc_lst):
58         # Extract the target values
59         pred_target = np.atleast_1d(
60             np.squeeze(ts.loc[fc.index])).transpose()
61         pred_target = np.array(pred_target)
62         pred_target = np.ma.masked_invalid(pred_target)
63
64         quantile_fc = fc.quantile(tau)
65         ql += (quantile_loss(pred_target, quantile_fc, tau) / len(
    pred_target))
66     return ql / len(ts_lst)
67
68
69 # Data
70 with open(path_to_data_folder + '\\deepar_test.pickle', 'rb') as h:
71     deepar_test = pickle.load(h)
72
73 with open(path_to_data_folder + '\\mqrnn_test.pickle', 'rb') as h:
74     mqrnn_test = pickle.load(h)
75
76 # Models
77 deeparpure_predictor = Predictor.deserialize(Path(
    path_to_data_folder + '\\deeparpure\\'))
78 deeparfeats_predictor = Predictor.deserialize(Path(
    path_to_data_folder + '\\deeparfeats\\'))
79 mqrnn_predictor = Predictor.deserialize(Path(path_to_data_folder +
    '\\mqrnn\\'))
80
81 # Forecasts
82 mqrnn_fc, mqrnn_ts = make_evaluation_predictions(
83     dataset=mqrnn_test,
84     predictor=mqrnn_predictor,
85     num_samples=None)
86 deeparpure_fc, deeparpure_ts = make_evaluation_predictions(
87     dataset=deepar_test,
88     predictor=deeparpure_predictor,
89     num_samples=100)
90 deeparfeats_fc, deeparfeats_ts = make_evaluation_predictions(
91     dataset=deepar_test,
92     predictor=deeparfeats_predictor,
93     num_samples=100)
94
95 mqrnn_fc = list(mqrnn_fc)
96 mqrnn_ts = list(mqrnn_ts)
97
98 deeparpure_fc = list(deeparpure_fc)
99 deeparpure_ts = list(deeparpure_ts)
100
101 deeparfeats_fc = list(deeparfeats_fc)
102 deeparfeats_ts = list(deeparfeats_ts)
103
104 # Evaluation
```

```
quantiles = [0.1, 0.25, 0.5, 0.75, 0.9]
idx = ['MQRNN', 'DeepAR-pure', 'DeepAR-feats']
ql_results = pd.DataFrame(index=idx, columns=quantiles)
risk_results = ql_results.copy()
for q in quantiles:
    ql_results.loc['MQRNN', q] = mean_quantile_loss(mqrnn_fc,
                                                    mqrnn_ts,
                                                    q)
    ql_results.loc['DeepAR-pure', q] = mean_quantile_loss(
    deeparpure_fc,

    deeparpure_ts,
                                                          q)
    ql_results.loc['DeepAR-feats', q] = mean_quantile_loss(
    deeparfeats_fc,

    deeparfeats_ts,
                                                           q)

    risk_results.loc['MQRNN', q] = tau_risk(mqrnn_fc,
                                            mqrnn_ts,
                                            q)
    risk_results.loc['DeepAR-pure', q] = tau_risk(deeparpure_fc,
                                                  deeparpure_ts,
                                                  q)
    risk_results.loc['DeepAR-feats', q] = tau_risk(deeparfeats_fc,
                                                   deeparfeats_ts,
                                                   q)


print(ql_results)
print(risk_results)

# parquet cannot deal with integer column names
col_names = [str(i) for i in quantiles]
ql_results.columns = col_names
risk_results.columns = col_names

ql_results.to_parquet(path_to_data_folder + '\\mean_quantile_loss.
    parquet')
risk_results.to_parquet(path_to_data_folder + '\\tau-risk.parquet')
```