Normandie Université

INSA ROUEN NORMANDIE

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité Informatique

Préparée au sein de l' INSA Rouen Normandie

Probabilistic Exponential Smoothing for Explainable AI in the Supply Chain domain

Présentée et soutenue par

ANTONIO CIFONELLI

Thèse soutenue publiquement le 22 12, 2023
devant le jury composé de

M. Massih-Reza AMINI,	Professor at Université Grenoble, Alpes	Rapporteur
M.me Mireille BATTON-HUBERT,	Professor at École des Mines de Saint-Étienne	Rapporteur
M.me Samia AINOUZ,	Professor at INSA Rouen, Normandie	Examiner
M. Stéphane CANU,	Professor at INSA-Rouen, Normandie	Thesis director
M.me Sylvie LE HÉGARAT-MASCLE,	Professor at Paris-Saclay	Examiner
M. Joannes VERMOREL,	CEO of Lokad	Invited member

Thèse dirigée par STÉPHANE CANU (Laboratoire d'Informatique, du Traitement de l'Information et des Systèmes)

MIIS

litis

Résumé en Français

Le rôle clé que l'IA pourrait jouer dans l'amélioration des activités commerciales est connu depuis longtemps (au moins depuis 2017), mais le processus de pénétration de cette nouvelle technologie a rencontrzécertains freins au sein des entreprises, en particulier, les coûts de mise œuvre. Les entreprises restent attachées à leurs anciens systèmes en raison de l'énergie et de l'argent nécessaires pour les remplacer. En moyenne, 2.8 ans sont nécessaires depuis la sélection du fournisseur jusqu'au déploiement complet d'une nouvelle solution. Trois points fondamentaux doivent être pris en compte lors du développement d'un nouveau modèle. Le Désalignement des attentes, le besoin de compréhension et d'explications et les problèmes de performance et de fiabilité. Dans le cas de modèles traitant des données de la chaîne d'approvisionnement (supply chain), cinq questions spécifiques viennent s'ajouter aux précédentes:

- La gestion des incertitudes. La précision n'est pas tout. Les décideurs cherchent un moyen de minimiser le risque associé à chaque décision qu'ils doivent prendre en présence d'incertitude. Obtenir une prévision exacte est un rêve; obtenir une prévision assez précise et en calculer les limites est réaliste et judicieux.

- Le traitement des données entières et positives. La plupart des articles vendus dans le commerce de détail ne peuvent pas être vendus en sous-unités, par exemple, une boîte de conserve, une pièce de rechange ou un t-shirt. Cet aspect simple de la vente se traduit par une contrainte qui doit être satisfaite par le résultat de toute méthode ou modèle donné : le résultat doit être un entier positif.

- L'observabilité. La demande du client ne peut pas être mesurée directement, seules les ventes peuvent être enregistrées et servir de proxy pour la demande.

- La rareté et la parcimonie. Les ventes sont une quantité discontinue : un produit peut bien se vendre pendant une semaine, puis plus du tout la semaine suivante. En enregistrant les ventes par jour, une année entiére est condensée en seulement 365 (ou 366) points. De plus, une grande partie d'entre elles sera à zéro.

- L'optimisation juste-à-temps. La prévision est une fonction clé, mais elle n'est qu'un élément d'une chaîne de traitements soutenant la prise de décision. Le temps est une ressource précieuse qui ne peut pas être consacrée entièrement à une seule fonction. Le processus de décision et les adaptations associées doivent donc être effectuées dans un temps limité et d'une manière suffisamment flexible pour pouvoir être interrompu et relancé en cas de besoin afin d'incorporer des événements inattendus ou des ajustements nécessaires.

Cette thèse s'insère dans ce contexte et est le résultat du travail effectué au cœur de Lokad, une société parisienne de logiciels visant à combler le fossé entre la technologie et la chaîne d'approvisionnement. La recherche doctorale a été financée par Lokad en collaboration avec l'ANRT dans le cadre d'un contrat CIFRE. Le travail proposé a l'ambition d'être un bon compromis entre les nouvelles technologies et les attentes des entreprises, en abordant les divers aspects précédemment présentés.
Nous avons commencé à effectuer des prévisions en utilisant des méthodes de base - la famille des lissages exponentiels - qui sont faciles à mettre en œuvre et extrêmement rapides à exécuter. Étant largement utilisés dans l'industrie, elles ont déjà gagné la confiance des utilisateurs. De plus, elles sont faciles à comprendre et à expliquer à un public non averti. En exploitant des techniques plus avancées relevant du domaine de l'IA, certaines des limites des modèles utilisés peuvent être surmontées. L'apprentissage par transfert s'est avéré être une approche pertinente pour extrapoler des informations utiles dans le cas où le nombre de données disponibles était très limité. L'hypothèse gaussienne commune ne convenant pas aux données de vente discrètes, nous avons proposé d'utiliser un modèle associé à une loi de Poisson, une binomiale négative qui correspond mieux à la nature des phénomènes que nous cherchons à modéliser et à prévoir. Nous avons aussi proposé de traiter l'incertitude par la simulation. À travers des simulations de Monte Carlo, un certain nombre de scénarios sont générés, échantillonnés et modélisés par dans une distribution. À partir de cette dernière, des intervalles de confiance de taille dif-

férentes et adaptés peuvent être déduits. Sur des données réelles de l'entreprise, nous avons comparé notre approche avec les méthodes de l'état de l'art comme le modèle Deep AutoRegressive (DeepAR), le modèle Deep State Space (DeepSSMs) et le modèle Neural Basis Expansion Analysis (N-Beats). Nous en avons déduit un nouveau modèle conçu à partir de la méthode Holt-Winter. Ces modèles ont été mis en œuvre dans le "work flow" de l'entreprise Lokad.

Mots clés: Apprentissage par Transfert, Différentiation Automatique, Lissages Exponentiels, Méthod Holt-Winter, Programmation Différentiable, Prévision de la Demande, Prévision des Séries Temporelles, Prévision Probabiliste, Réseau Long Short-Term Memory (LSTM), Simulation de Monte Carlo.

Résumé en Anglais

The key role that AI could play in improving business operations has been known for a long time (at least since 2017), but the penetration process of this new technology has encountered certain obstacles within companies, in particular, implementation costs. Companies remain attached to their old systems because of the energy and money required to replace them. On average, it takes 2.8 years from supplier selection to full deployment of a new solution. There are three fundamental points to consider when developing a new model. Misalignment of expectations, the need for understanding and explanation, and performance and reliability issues. In the case of models dealing with supply chain data, there are five additionally specific issues:

- Managing uncertainty. Precision is not everything. Decision-makers are looking for a way to minimise the risk associated with each decision they have to make in the presence of uncertainty. Obtaining an exact forecast is a advantageous; obtaining a fairly accurate forecast and calculating its limits is realistic and appropriate.

- Handling integer and positive data. Most items sold in retail cannot be sold in subunits, for example, a can of food, a spare part or a t-shirt. This simple aspect of selling, results in a constraint that must be satisfied by the result of any given method or model: the result must be a positive integer.

- Observability. Customer demand cannot be measured directly, only sales can be recorded and used as a proxy for demand.

- Scarcity and parsimony. Sales are a discontinuous quantity: a product may sell well one week, then not at all the next. By recording sales by day, an entire year is condensed into just 365 (or 366) points. What's more, a large proportion of them will be zero.

- Just-in-time optimisation. Forecasting is a key function, but it is only one element in a chain of processes supporting decision-making. Time is a precious resource that cannot be devoted entirely to a single function. The decision-making process and

associated adaptations must therefore be carried out within a limited time frame, and in a sufficiently flexible manner to be able to be interrupted and restarted if necessary in order to incorporate unexpected events or necessary adjustments.

This thesis fits into this context and is the result of the work carried out at the heart of Lokad, a Paris-based software company aiming to bridge the gap between technology and the supply chain. The doctoral research was funded by Lokad in collaboration with the ANRT under a CIFRE contract. The proposed work aims to be a good compromise between new technologies and business expectations, addressing the various aspects presented above.
We have started forecasting using basic methods - the exponential smoothing family - which are easy to implement and extremely fast to run. As they are widely used in the industry, they have already won the confidence of users. What's more, they are easy to understand and explain to an unlettered audience. By exploiting more advanced AI techniques, some of the limitations of the models used can be overcome. Cross-learning proved to be a relevant approach for extrapolating useful information when the number of available data was very limited. Since the common Gaussian assumption is not suitable for discrete sales data, we proposed using a model associated with either a Poisson distribution or a Negative Binomial one, which better corresponds to the nature of the phenomena we are seeking to model and predict. We also proposed using simulation to deal with uncertainty. Using Monte Carlo simulations, a number of scenarios are generated, sampled and modelled using a distribution. From this distribution, confidence intervals of different and adapted sizes can be deduced. Using real company data, we compared our approach with state-of-the-art methods such as the Deep Auto-Regressive (DeepAR) model, the Deep State Space (DeepSSMs) model and the Neural Basis Expansion Analysis (N-Beats) model. We deduced a new model based on the Holt-Winter method. These models were implemented in Lokad's work flow.

Keywords: Automatic Differentiation, Cross-learning, Demand Forecasting, Differentiable Programming, Exponential Smoothing, Holt-Winter method, Long Short-Term Memory network (LSTM), Monte Carlo Simulation, Probabilistic Forecasting, Time series Forecasting.

Contents

Chapter 1

Introduction

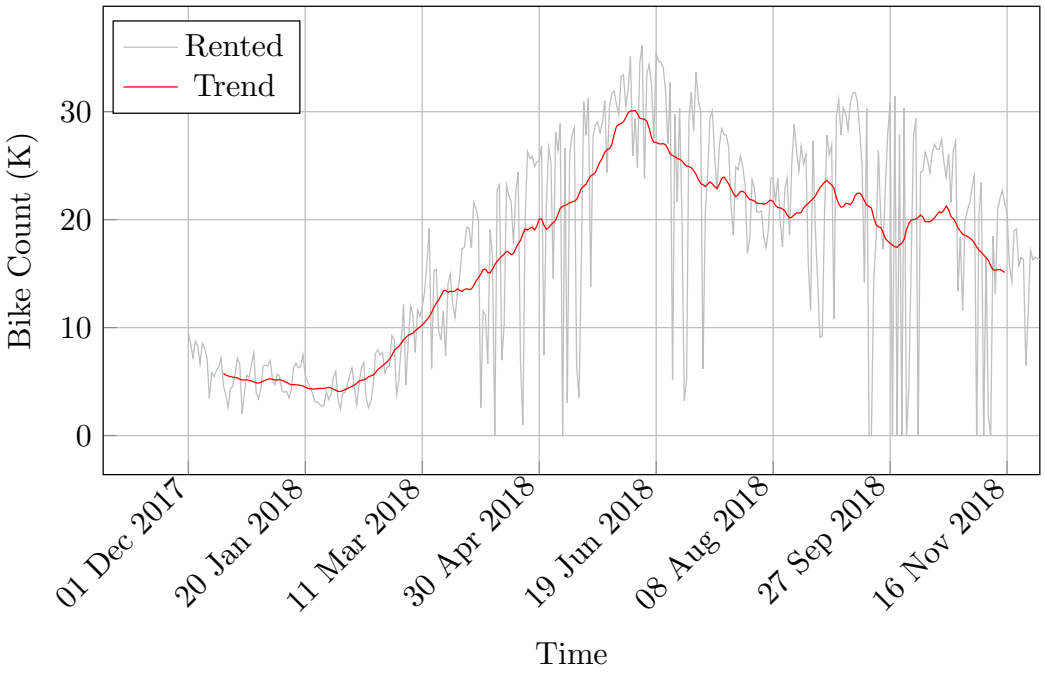
Contents

Time series are defined as a collection of data points for which the time order matter. They are a useful tool to capture, analyze and visualize disparate temporal phenomena, like the heartbeat recorded by our smart watch or fitness tracker over a day, the timeline on our social medias, the electricity demand of a household or the fluctuations of a bank account just to name a few. Virtually they are employed in each and every aspect of our life, a snapshot of our activities or events we are interested into. Sometimes - not to say always - the evolution of a time series is affected by numerous factors, which can either be time series themselves or not. For example our heartbeat can change in response to physical efforts or because we are asleep; over winter months the temperature goes down, we consume more energy to heat our houses and higher fluctuations in our bank accounts are appreciable. The interesting and central role played by time series had aroused curiosity of researchers and practitioners over decades, and still continues to do so.

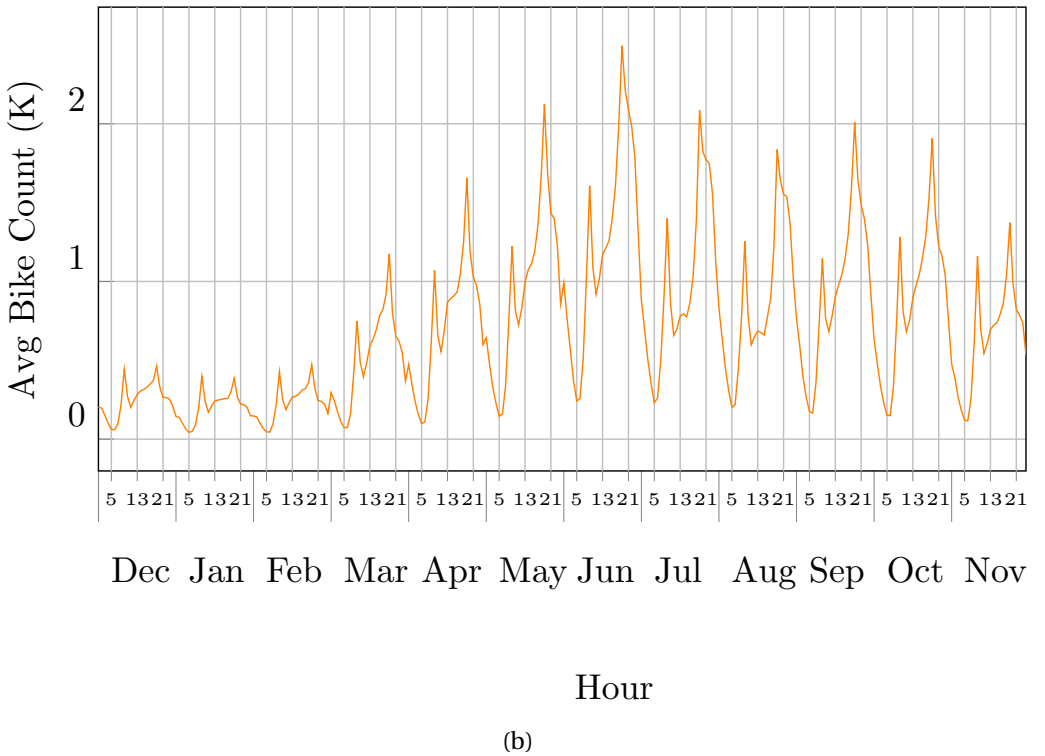
Since the very beginning of rigorous time series analysis, it was clear the possibility to extract valuable insights and or patterns from the data, which in turn could bring forward the time series evolution. By now it is common feeling to identify three foundation patterns within a time series: a monotonic upward or downward movement, called *trend*; one or more repeating patterns at dissimilar frequencies, these take the name of *seasonality*; a low pace

seasonal variation which is peculiar to time series spanning a wide time range - decades or event centuries - which is known as *cycle*. Occasionally these components are easily identified, more often their presence and the interactions with other constituents are elusive. Take the example of a rental bike service located in Seoul, presented in Fig. (1.1). Rental bikes, as well as electric scooters, are being introduced in many major cities - as one of the means to enhance sustainable mobility while diminish the number of circulating cars - to then extend to radial areas. The figure depicts in gray the daily demand - Fig. (1.1a) - which had been obtained aggregating the raw hourly demand. Looking at the chart an erratic day-to-day evolution is recognizable, with some drops to zero when the rental service was in maintenance. A closer inspection roughly spots a trend, being the red line in Fig. (1.1a) a crude initial approximation of it. It starts to climb as soon as the Spring approaches and people are more willing to pedal, reaches its peak in Summer and comes back in the winter months when the temperature's drop discourages cyclists. No seasonality seem manifest at this level of detail, but they are there. A bit more work is required to extrapolate the changes, for instance aggregating the data either by the hour for each week or by the hour for each month (as actually shown in Fig. (1.1b)). The seasonality emerged from the data and we can already derive some intuitions. The shape of the seasonal changes is nearly identical over the year with two clear peaks, one around 9 am and one roughly at 19 pm. These two peaks correspond to the start and the end of the work day. Finally it wanes overnight. The change in magnitude catches the eye, but we have to note that it is not amended by the trend influence.

Human activities at large are particularly rich in (hidden) patterns and the ability to automatically discover them with limited experience - or no experience at all - is crucial. The first methods appeared in the early 1900s and were all centered around smoothing techniques. *M*oving *A*verages (MA) and *E*xponential *S*moothing (ES) were among these first trials; they were essential and not suitable for time series with seasonality yet. In the 1970s *A*uto*r*egressive *M*oving *A*verage (ARMA) models, and their *I*ntegrated version (ARIMA), bridged the gap. Later all these methods and models had been gathered together under the universal *S*tate *S*pace Models (SSMs) class. Over decades the time series analysis toolbox had been filled with numerous data processing techniques and new models, last but not least Machine Learning models. We all got used to words like Machine Learning, Deep Learning and *A*rtificial *I*ntelligence (AI) over the last few years, and in the last months in particular thanks to the commercialization of services like Chat-GPT. Models belonging to the AI's sphere of influence proved to be effective in various fields, from vision and natural language processing to autonomous driving and drug discovery. However Exponential Smoothing and AR(I)MA represented and still represent the standard *de facto* in industry and business applications when it comes to time series analysis. The discrepancy has been



(a)



(b)

Figure 1.1: Seoul rental bike data. (1.1a) The raw daily bike count - obtained aggregating the raw hourly demand - is shown together with a rough estimation of its trend (red line). (1.1b) Aggregating the data by the hour for each month - for visualization sake - a seasonal profile is clearly visible; the trend effect has not been removed. Source data available at [Hol].

the subject up to this time for studies and reports, since the first years of the 2000s. The digitalisation process in fact guided to a steady and exponential data proliferation, multiplying not only the information sources but also the affecting factors. A technical shift within companies was expected, but not observed. The trend started to change only recently.

Business wise the contradiction can be explained with the cost implicated with a technical reorganization. Companies hang to older systems because of the time and money needed to replace them. As noted by McKinsey & Company in one of their 2022 reports - [McKd] - companies take 2.8 years on average from vendor selection to a complete roll-out of a new solution. Pharmaceutical companies can take up to six years to complete the process. The capital investment, as the time one, is dependent on the company's complexity; the pharmaceutical industry spends from €55 million to €110 million for a new solution, while less complex industries, like consumer-packaged goods, invest about €15 million. With such numbers involved, a failure is a pondered risk to minimize. Nevertheless sixty percent of the time the implementation of a new solution fails due to missed deadlines, over budget or disappointing outcomes. Finally, switching to a new system is not only a matter of technologies. Companies need to prepare the staff to use the new system, ensuring that they are engaged with it and the possible new working flow, while running in parallel the older one to not shutdown operations.

Back in the 2017 companies recognized the key role AI could play, but were unsure about what to expect and how it could fit within their business model. At the time only the 23 percent of the respondents to the MITSloan Management Review's survey [RKGR17] claim to had incorporated AI-related techniques in their processes; among adopters, only five percent of them asserted to use AI extensively. An additional 23 percent was using AI only in pilot projects, for a total of $\tilde{4}6$ percent of asked businesses being proactive. Finally a 22 percent of the participants were not using AI and didn't have a plan to incorporate it in their business. Customer-facing activities and service operations optimization in general had been recognized as one of the main business unit in which AI could make the difference. A couple o years after the trend is confirmed by the report wrote by McKinsey & Company [McKb]. Adoption has more than doubled since 2017, but plateaued between 50 and 60 percent for the past few years. We imagine that: a) at least a portion of the pilot projects detected in the 2017 reached the production phase; b) in a dynamic and healthy economic companies are established, grow and die, generating a turnover between adopters and skeptics about AI. Yet a great percentage of respondents are not using or are not interested at all in AI's capabilities. Looking at the top use cases the trend has remained quite stable too. With reference to the 2017, marketing and sales have came up beside service operations optimization, the latter being still at the first place of the list. Their asset position could only be strengthened by

the breakout of Generative AIs [McKc]. This should not be a surprise as these kind of business functions do not involve - most of the time - a human intervention and are supportive to other decision making units.

Makridakis et al. [MSA18b] questioned themselves about the disparity between AI in research and industry since 2000. Prof. Makridakis is the father of a series of competitions, named after him (3.1.5), which are focused on encouraging the adoption of new methods and models putting in contact practitioners and researchers. Echoing professor's words

> *[...] we hope that those in the field of AI and ML will accept the empirical findings and work to improve the forecasting accuracy of their methods.*
> *A problem with the academic ML forecasting literature is that the majority of published studies provide forecasts and claim satisfactory accuracies without comparing them with simple statistical methods or even naïve benchmarks.*

What Prof. Makridakis criticizes - and tries to exemplify in his competitions - is the lack of a fundamental question when it comes to ML research: is this feasible in practice? This is not a trivial question and can be articulated in 3 main sub-problems:

Expectation misalignment Accuracy is not everything when it comes to business judgement, each and every business function can require its own *K*ey *P*erformance *I*ndicators (KPIs) - which would in general diverge from the common metrics adopted in research - capturing a specific angle of the company. When KPIs are not aligned a belief over the goodness and value of a new solution is hardly constructed.

Need of explainable results Core business functions are not left completely automated. The model could provide and initial solution for the problem at hand, based on more or less advanced analytics; an expert of the field will then review and adapt the proposed solution to fit the company requirements or those of other correlated activities. The direct implication is two-fold: the answer proposed by the model needs to be correct, not only in the accuracy sense of the term but also from a business perspective; the internal model's process has to be accountable and explainable also to non-technical public. The latter statement is the definition of a "white-box" model. In contrast state-of-the-art AI models act as a "black-box". Information are fed to the model, which elaborates them and emits a result; how the information are distilled into the final answer cannot be clarified or need advanced techniques to retrace the inference process. Models like LIME [RSG16] and SHAP [LL17] had been developed toward this end; together with other models and methods make up the *Explainable-AI* branch. Yet their application requires a higher level of skill set and can be fooled, producing misleading interpretations, as illustrated by Slack et al. [SHJ+20].

Lack of trustworthiness This facet is the straight conclusion of the previous two instances. In our opinion it can also arise from a workforce transformation inertia. In the last 2022 report by McKinsey & Company [McKb] underlined a tech talent shortage. In response to hiring challenge the most popular strategy, among all respondents, is reskilling existing employees. Nearly half of the respondents are doing so. Experienced employees could own a set of preferred tools to be implemented in different scenarios; a clear advantage of the new solution over the set of preferences - or at least one of them - is a strong argument to ease the transition, differently the comfort of a well-known and frequently applied solution will be preferred.

1.1 AI & Supply Chain Management

In all the consulted reports [RKGR17, McKa, McKd, McKb, McKc] supply chain management is always ranked in the lowest positions among functions which can be improved by AI. Unequivocally there could be a revenue increase in improving this core function, but there is still friction against the modernization. Ninety percent of the supply-chain executives declared to expect an overhaul of the planning system in the next five years; more or less the same percentage claimed a similar objective five years before. As shown in Fig. 1.2 close to three-quarter of supply-chain functions rely on simplest method like spreadsheets. In addition more than half use SAP *A*dvanced *P*lanning and *O*ptimization (APO), an antiquated supply-chain planning application introduced back in the 1998 and which will be discounted in 2027.

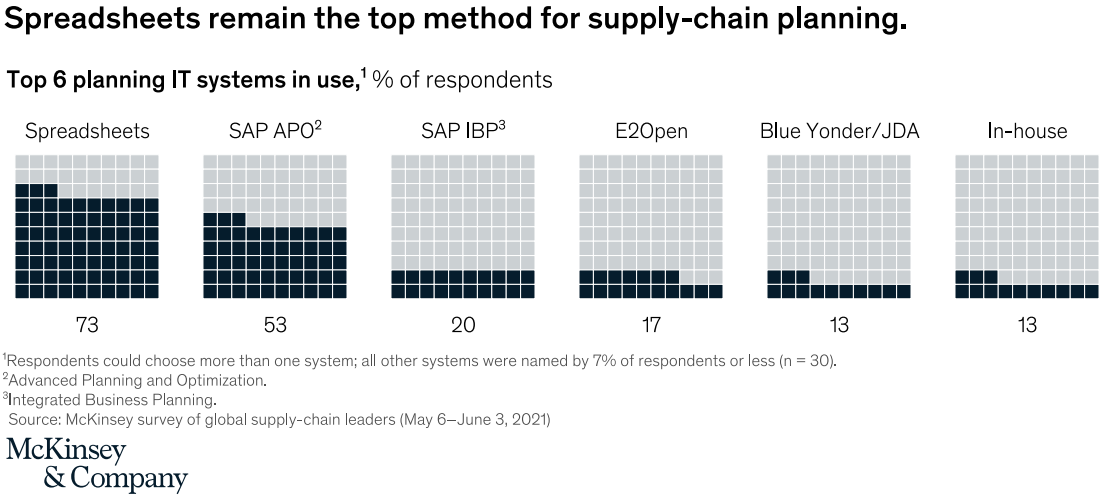


Figure 1.2: Courtesy of McKinsey & Company [McKd]. Spreadsheets are still the preferred method in industry to accomplish planning tasks.

What has already been discussed in the previous section is still sound - and even more pro-

nounced - for the supply-chain case. Additionally there are other concerns to take into account

Embrace uncertainty Again, accuracy is not everything. Decision makers seek a way to minimize the risk associated with each and every decision they have to make. Uncertainty is part of the undertaking. A simple yet concrete example is the supplier's lead time. On average a supplier will ship the order in time, no main disruptions will be encountered on the way and the client will receive its goods as expected. Lead times nevertheless are not deterministic, actually we have an array of possibilities and each will have a different impact on the business activities. Getting a forecast right is advantageous; getting a forecast accurate enough and showing its limitation is appropriate.

Dealing with integer and positive data In Fig. (1.3) 52 weeks of total sales and average price of a popular beer brand, 18-packs carton size, are shown. As it is not possible to exchange any given fraction of a pack, we would record only integer sales values. Someone could argue that we could pick a single can from a pack, that would represent a fraction of the pack ($1/18$). Even if logically it makes sense, from a business perspective it is incorrect. Inside the company's catalog each item is associated with an unambiguous identifier, the *S*tock *K*eeping *U*nit (SKU). Different goods, as an 18-pack and a single can, are linked to distinct SKUs. Hence the sale of a single can will be recorded for its SKU, should the can comes from either a shelf or an opened pack of 18 cans. This uncomplicated selling aspect translates to a constraint which has to be satisfied by the outcome of any given method or model.

Observability Sales are the focal point of Supply Chain Management and are the main quantity analysed to accomplish several tasks, including demand forecasting. Though they are not the quantity of interested, customer demand is. If company could record a customer demand they would prefer to do that (recall the case of rental bike in Fig. (1.1)). When customer demand is a non-observable quantity, sales are the second best option. Generally speaking sales act as a proxy to customer demand; in other words there is no mean by which we can record an unsatisfied demand.

Scarceness Look at a second example in Fig. (1.4). It illustrates the monthly sales of an auto spare part. Fifty one data points summarise 4 complete years (and a couple of months) and 27 of them - nearly the 53 percent - are zero.

Just-in-Time optimization Enterprises have different budgets and have to allocate a portion of it to each and every of their functions. Planning has a cost. Computational resources have a cost. Human resources have a cost. The time taken by a complex

model to train and generate results would raise the pile of expenses and could be put aside in favour of less complex - maybe even less accurate - systems which will deliver (comprehensible) results in less time. As a consequence of the field's uncertainty, the optimization step has to be as short as possible - data is already old the next working day - and as flexible as possible to be interrupted and re-launched to incorporate possible unexpected events or adjustments.

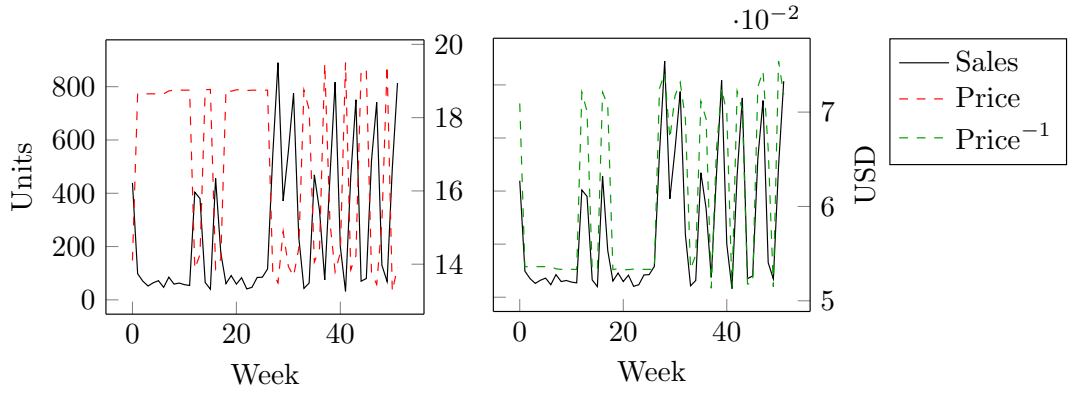


Figure 1.3: Real data showing 52 weeks of average-price and total-sales records of a popular beer brand 18-packs carton size (left). In this case it is easy to see a correlation between the price change and the units sold, emphasized in the right figure. Source data available at [Nau].

The divergence observed by Prof. Makridakis becomes obvious when inspecting the in-house research conducted by companies. Except for giant companies like Amazon, the rest of the businesses are consecrating their efforts to make simple models - most of the time already in use in production - scalable and flexible. For a more wide analysis of this phenomenon see Sec. 3.4.

This thesis is inserted in this context and is the result of the work done in the midst of Lokad, Parisian software company aiming at bridging technology and Supply Chain. The PhD research had been funded by Lokad in collaboration with the *A*ssociation *N*ationale *R*echerche *T*echnologie (ANRT) under a CIFRE contract. The proposed work has the ambition to be a good trade-off between new technologies and business expectations, addressing the various aspects previously introduced.

Trustworthiness and explainable results We started from the methods included in the Exponential Smoothing family. These methods are easy to implement and extremely fast in execution, even on common commercial hardware. Being widely used and accepted in the industry, they are a good candidate as starting point to promote the

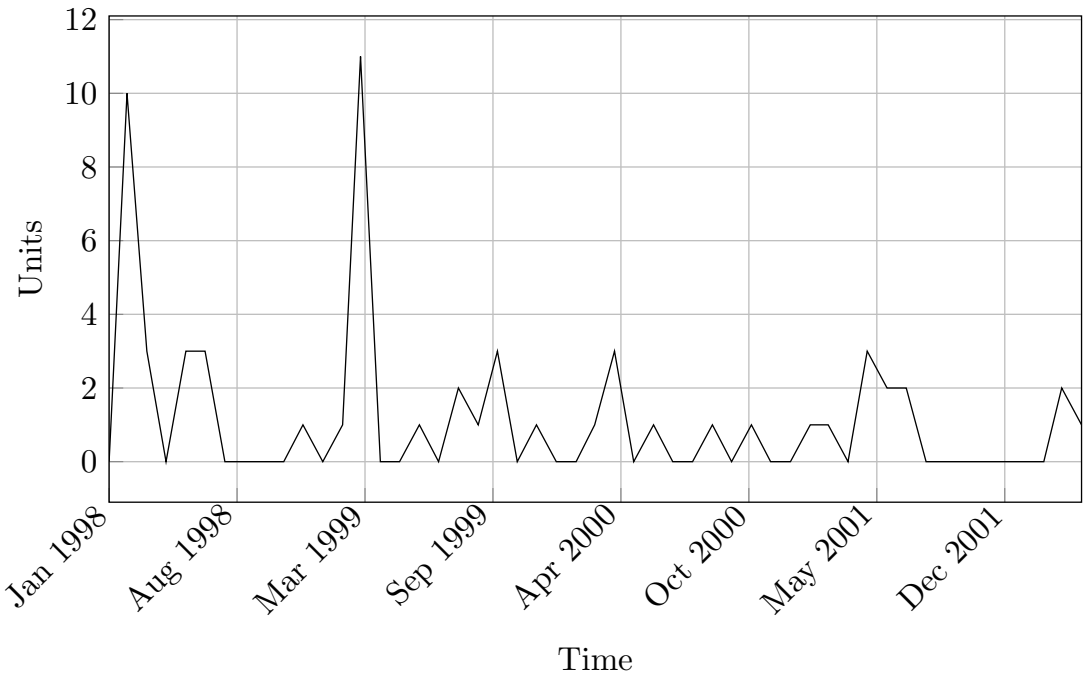


Figure 1.4: Sparse demand of the automotive spare part marked with *Id* 10055165 in the original dataset. The data has been extrapolated from the dataset *carparts* available at [HKOS]. The dataset is released as supplementary material of [HKOS08] and has been pre-processed as described in Snyder et al. [SOB12].

transition. Moreover they are easy to grasp and easily explainable. Nonetheless their limitations are well known, for instance either lack of seasonality or fixed seasonality, inability in shaping multiple seasonality.

Scarceness Scarcity plays against a stable and effective training process. Cross-learning can ease the difficulties and support pattern extrapolation even in a scarcity of series-wise data.

Integer and positive constraints The most common assumption done in practice is a Gaussian fit for the data, i.e. data follow a bell-shaped distribution around a mean (nearly all the time set at zero). For integer data - with a great percentage of zeroes - the assumption fells. The integer and positive constraints are addressed replacing the Gaussian assumption with either a Poisson or a Negative Binomial one.

Embrace uncertainty Simulating several scenarios, the impact of various factors can be approximated. The more scenarios are tested, the more we are prepared for the different outcomes (up to an irreducible degree of unexpected occurrences). Monte Carlo simulation adapts really well to these conditions. A number of scenarios are generated, sampled and summarised into a distribution. From the latter disparate confidence intervals can be derived.

We compare against both well known standard methods and *S*tate-*o*f-*t*he-*A*rt (SotA) models on a collection of datasets sampled from the literature, public forecasting competitions and in-house data lake.

1.2 Structure of the manuscript

The rest of the dissertation will be organised as follow.
Chapter 2 presents the literature starting from the most common methods and models - also known as *shallow* models - (Sec. 2.3) and some more modern architectures - Sec. 2.4 - like *N*eural *N*etworks (NNs).
Chapter 3 introduces demand time series and demand forecasting peculiarities. The history of Makridakis Competitions - Sec 3.1.5 - will be of use as an evolutionary timeline of time series forecasting, it will serve a double scope: give an idea of how the landscape of this core task has evolved over time; stress how arduous is for new models to penetrate the industry tech stack. The same chapter holds sections related to both the datasets included in the study and the up-to-date SotA designs. *Deep* Auto*R*egressive model (DeepAR) [SFGJ19], *D*eep *S*tate *S*pace *M*odels (DeepSSMs) [RSG+18] - both from Amazon - and the *N*eural *B*asis *E*xpansion *A*nalysis model (N-Beats) [ODPT21] are there presented. Finally a review

of the French industry researches highlights the exploration of the time series forecasting area, in contrast with the literature introduced previously.
Chapter 4 focuses on Differentiable Programming, the new paradigm shift which is interesting the Machine Learning community, and *A*utomatic *D*ifferentiation (AD) which had been the fuse for this change; Sec. 4.5 briefly discuss how this new paradigm had already been incorporated into the Lokad's work flow.
Chapter 5 introduces the new model conceived starting from Holt-Winter method.

1.3 Summary

The key role AI could play in enhancing business activities had been recognized since 2017 [RKGR17], yet the penetration process of this new technology encountered a certain degree of inertia within companies from different backgrounds. Costs are the main obstacles. Companies hang to older systems because of the time and money needed to replace them. On average 2.8 years are required from vendor selection to a complete roll-out of the new solution. The capital investment, au par of the time required, is dependent on the company's complexity; the magnitude ranges from a dozen of million to more than a hundred. With such numbers involved, a failure is a pondered risk to minimize. Nevertheless sixty percent of the time the implementation of a new solution fails due to missed deadlines, over budget or disappointing outcomes. Finally, switching to a new solution is not only a matter of technologies. Companies need to prepare the staff to use the new system, ensuring that they are engaged with it and the possible new working flow, while running in parallel the older one to not shutdown operations. The human component can not be taken out of the equation; each professional would rely on his or her expertise - and a set of preferred tools - to solve a problem. Three focal points should be taken into account when developing a new model:

Expectation misalignment Accuracy is not everything. Each and every business require different KPIs which would in general diverge from the common metrics adopted in research, capturing a specific angle of the company. When KPIs are not aligned a belief over the goodness and value of a new solution is hardly constructed.

Need for explainable results While the main research groups are focusing their activities on ever growing complex models, moving towards a "black-box" approach, businesses prefer to stand with "white-box" models which are more human friendly and transparent for the decision makers. To achieve a certain grade of transparency, multiple advanced techniques have to be applied.

Lack of trustworthiness Straight conclusion of the previous two instances, it is also bounded to the human factor aforementioned.

If the model has to deal with supply chain data, the following points add to the previous ones:

Embrace uncertainty Again, accuracy is not everything. Decision makers seek a way to minimize the risk associated with each and every decision they have to make. Uncertainty is part of the undertaking. Getting a forecast right is advantageous; getting a forecast accurate enough and showing its limitation is appropriate.

Dealing with integer and positive data Most of the articles sold in retail can not be sold in sub-units, e.g. a can, a spare part or a t-shirt. This uncomplicated selling aspect translates to a constraint which has to be satisfied by the outcome of any given method or model: the result has to be a positive integer.

Observability Customer's demand can not be measured directly, only sales can be recorded and act as a proxy for the demand.

Scarceness Sales however are a discontinuous quantity: a product can sell well over a week and than not at all for the next month. Recording sales per day, an entire year is condensed in only 365 (366) data points. In addition, a great portion of them will be at zero.

Just-in-Time optimization Forecasting is a core function, but it belongs to a more general chain of sub-processes supporting decision making. Time is a valuable resource to pace and can not be consecrated completely to a single function. The training process therefore has to be limited in time and flexible enough to be interrupted and re-launched to incorporate unexpected events or mandatory adjustments.

This thesis is inserted in this context and is the result of the work done in the midst of Lokad, Parisian software company aiming at bridging technology and Supply Chain. The PhD research had been funded by Lokad in collaboration with the ANRT under a CIFRE contract. The proposed work has the ambition to be a good trade-off between new technologies and business expectations, addressing the various aspects previously introduced. We started from basic methods - the Exponential Smoothing family - which are easy to implement and extremely fast in execution. Being widely used in the industry, they already won practitioner's trust. Moreover they are easy to grasp by and easily explainable to a non-technical audience. Exploiting more advanced techniques belonging to the ML field, some of the limitations of the employed models can be overcame. Cross-learning had been proven a valid approach to extrapolate useful information even in a scarcity of series-wise data case. The common Gaussian assumption would not hold for sales data, either a Poisson one or a Negative Binomial expectation would fit better the situation. Finally uncertainty is dealt with

simulation. Through Monte Carlo simulations a number of scenarios are generated, sampled and summarised into a distribution. From the latter disparate confidence intervals can be derived.

Chapter 2

Time series analysis and forecasting

Contents

Framing the definition of time series in a mathematical way, we define a time series as a vector $\mathbf{z}$. Each vector is made by data points - or *observations*, or *dependent variable* - z_t, being t the time step at which each observation is gathered - $t = 1, 2, \ldots$ -, and their number define the *d*imension d of the vector such that $\mathbf{z} \in \mathbb{R}^d$. A finite collection of $i = 1, \ldots, N$ time

series constitute a dataset $\mathbb{D} = \left\{\mathbf{z}^i_{t_0:\mathrm{T}}\right\}_{i=1}^{\mathrm{N}}$. Two general tasks can be ran over a time series or a dataset.

2.1 Time series analysis

Sometimes called *descriptive modelling*, time series analysis focuses on understanding time series in order to develop models able to best describe the observed variables. It responds to the *why* behind a time series behaviour, decomposing it into constituent components and making assumption about the data distribution. It involves knowledge of the application field, mostly demanding an expert judgement. As a consequence it is prone to cognitive bias in the decomposition process or in the modelling choices. Given a time series $\mathbf{z}_{t_0=1:\mathrm{T}} = \{z_1, \ldots, z_t, \ldots, z_\mathrm{T}\} \in \mathbb{R}^d$, it can be checked with regards to several fundamental properties

Univariate & multivariate We say that $\mathbf{z}_{1:\mathrm{T}}$ is *univariate* if each and every observation z_t is a scalar, i.e. $z_t \in \mathbb{R}$. If instead each observation z_t is itself a vector made by k scalar values - hence we should write $\mathbf{z}_t$, $\mathbf{z}_t \in \mathbb{R}^k$ - $\mathbf{z}_{1:\mathrm{T}} \in \mathbb{R}^{d\times k}$ is said to be *multivariate* and can be expressed in a tabular or matrix form.

Regular & irregular If observations $\{z_1, \ldots, z_t, \ldots, z_\mathrm{T}\}$ are contiguous, i.e. sampled at a constant frequency, $\mathbf{z}_{1:\mathrm{T}}$ is said to be *regular*. On the contrary if the observations are collected erratically or the frequency changes over time, the time series is an *irregular* one.

Stationary & non-stationary $\mathbf{z}_{1:\mathrm{T}}$ can be qualified as *stationary* if its statistical properties - i.e. the moments of its generative distribution, such as the mean μ and the variance σ - don't change over time. A clear example is *white noise* ϵ_t, i.e. a time series whose samples are drawn from a Gaussian distributions. On the other hand $\mathbf{z}_{1:\mathrm{T}}$ is stated to be *non-stationary*. Stationarity is a type of dependence structure and a handy property, indeed if a sequence is stationary than plenty of results which hold for independent random variables also hold for the sequence.

Auto-correlation Common also with the name *serial correlation*, this property refers to the degree of linear correlation between two successive observations or, in general, between $\mathbf{z}_{1:\mathrm{T}}$ and its lagged version, either negative $\mathbf{z}_{1-\ell:\mathrm{T}-\ell}$ or positive
$\mathbf{z}_{1+\ell:\mathrm{T}+\ell}$ where ℓ is the number of lagging steps. Auto-correlation is defined in the range $[-1, 1]$ where -1 indicates a *perfectly negative* auto-correlation and 1 a *perfectly positive* one. A positive indicator reveals that an increment in the observation is linked to an increase

in the lagged quantity; in contrast a negative indicator implies that an increment in the observation is linked to a reduction in the lagged value.

The assessment of these basic characteristics can alleviate the cognitive bias effect leading to a well aware selection of the methods and models that better adapt to the problem faced. Nevertheless the advances in modelling techniques pushed increasingly towards a fusion between time series analysis and time series forecasting, see Sec. 2.2, almost erasing the boundaries and letting both of them to fall under the name of time series forecasting.

2.1.1 Structural time series

We used Fig. (1.1) to picture trend and seasonality ideas but we were silently introducing the notion of *s*tructural *t*ime *s*eries (STS), that we will recall in Sec. 2.3.4 talking about State Space Models. We owe the definition to Prof. A.C. Harvey who introduced it in his book *Forecasting, Structural Time Series Models and the Kalman Filter* [Har90] where he describes an alternative decomposition to represent a time series, not via the underlying data generation processes but rather through components that could highlight the features of the time series itself. The components have a comprehensible interpretation, are of interest in themselves and can be modelled independently if needed. Over the years it has became a sort of standard, applied through many different fields.
The traditional constituent of a STS are:

*T*rend (t): upward or downward movements over time;

*S*easonality (s): a repeating pattern at consistent time interavals;

*C*ycle (c): again an upward or downward variation, but appreciated over a longer time span than a usual trend, typically over decades or even more;

*E*rror (e): being also called *irregular*, *residual* or *remainder*, as the name suggests this component embraces everything that can not be explained by the other factors.

Two are the consolidated ways to formulate a model for a STS: an additive model

$$z_t = t_t + s_t + c_t + e_t \tag{2.1}$$

and a multiplicative one

$$z_t = t_t * s_t * c_t * e_t \tag{2.2}$$

with the possibility to treat the multiplicative case as an additive one by taking the logarithm of it. Alternative forms can also arise depending on how each and every component enters the equation [H+04].
Expecting the error term to follow a given distribution - e.g. a Gaussian with zero mean and unitary variance, very common choice in practice - a STS can be framed within the State Space Models scheme.

Seasonality

We would like for a moment to direct the attention over a seasonal property that we had found to pass nearly unnoticed. Either if we treat an additive seasonality or a multiplicative one, we expect it to be stationary and repeat itself from one period to the next. Abrupt changes, especially for additive seasonality, will most likely be incorporated either into the trend component, the error or both. To ensure that the seasonality is stationary we have to check that

$$\sum_{t=1}^{m_g} s_t^g = \begin{cases} 0 & \text{if additive} \\ m_g & \text{if multiplicative} \end{cases}$$

This basic property is clearly stated in [HA21] but appears to be disdained in a large portion of the literature reviewed.

2.1.2 Differencing

Structural Time Series are an example of non-stationary time series; as a matter of fact, the presence of either the trend, or the seasonality or both collide with the definition of stationarity. Differencing is a way to stabilize the mean of the time series, computing the differences between consecutive observations and possibly removing fluctuations. Taking the series of observations $\mathbf{z}^i_{1:\mathrm{T}}$, the first-order differencing is defined as

$$z'_t = z_t - z_{t-1}; \quad t = 2, \ldots, \mathrm{T}. \tag{2.3}$$

The new time series $\mathbf{z}'_{2:\mathrm{T}}$ has of course $\mathrm{T}-1$ elements, because it is not possible to compute the difference for z'_1. Moreover if $\mathbf{z}'_{2:\mathrm{T}}$ is now stationary, Eq. (2.3) suggests a model for the original time series. We can in fact rewrite the equality as $e_t = z_t - z_{t-1}$ and

$$z_t = z_{t-1} + e_t. \tag{2.4}$$

If $e_t = \epsilon_t$ the model is better known with the name of **random walk** model. The latter is at the foundation of many methods and models treating non-stationary time series, as we will

see further ahead.

Occasionally a second-order differencing can be performed over $\mathbf{z}'_{2:T_i}$ if it is not stationary. Informally the second-order differencing can be though as a second-order derivative at discrete-time, giving us a “velocity" indicator. More degree of differencing are of course possible, but in practice orders greater than two are rarely utilized.

Another type of differencing procedure is *seasonal differencing*; in a seasonal time series the correlation is stronger between z_t and the previous observation at $t - m$, where m is the *period* of the time series; e.g. $m = 12$ for monthly data. We rewrite Eq. (2.3) as

$$z'_t = z_t - z_{t-m}. \tag{2.5}$$

2.2 Time series forecasting

Time series forecasting is a kind of *regression* task. In contrast to *clustering* where one or multiple classes, picked from a finite set, are assigned to an observation, regression approximates the relationship between the input variables and a continuous dependent variable. It responds to the *how* a time series will evolve; once a model has been chosen, with or without having ran an analysis beforehand, we are interested in predicting future observations with the best margin of approximation. Before starting the *extrapolation* process - i.e. generate our predictions - we have to take a step backward and question ourselves about the forecasting problem we are aiming to solve

Qualitative & quantitative forecasting Qualitative forecasting is the perfect continuation of a time series analysis in the presence of an expert judgment. When prediction can not be backed up by data, because they are scarce or not present at all, the knowledge of an expert is a good starting point to guess future outcomes and, for some companies, the solely trusted way to proceed. The forecast of course will be strongly biased by the personal experience and beliefs of the specialist. Quantitative forecast conversely is a full data-driven process. Data-driven means that few or no assumptions are made a priori over the data, letting data shape the model's internal representation of the underlying generative process. Nevertheless these two modalities are often intermingled, having the expert to argue the outcome of a model in a *human-in-the-loop cycle*.

One-step-ahead & multi-step forecasting One-step-ahead forecast happens when we forecast only the next time step of a time series; when instead the prediction *horizon h* is

longer we talk about a multi-step forecast. Either for application and planning purpose or due to limitations in the preferred model, we have situations where we are interested in predicting only over a short-term period.

Aligned time series In a dataset defined as $\mathbb{D} = \left\{\mathbf{z}^i_{t_0:\mathrm{T}}\right\}_{i=1}^{\mathrm{N}}$, all the time series start at time t_0 and end at time T. The time series are therefore *aligned*. If instead either t_0, T or both are series-dependent - i.e $t_{0,i}$ and T_i - the time series are "misaligned". A technique to pass from a misaligned situation to an aligned one is padding, i.e. pushing a value either at one or both endpoints such that the total length is equal for all the time series. Zero is frequently used as padding value, but other common choices are the mean or the median of the time series. Padding however can have no meaning in some applications. Trimming the history could be also a solution; t_0 and T are chosen such that $\tilde{t}_0 = \max\left(\left\{t_{0,1}, t_{0,2}, \ldots, t_{0,\mathrm{N}}\right\}\right)$ and $\tilde{\mathrm{T}} = \min\left(\left\{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_\mathrm{N}\right\}\right)$. Depending on the problem, there could exist explicit rules whose application could result in a dataset alignment, filtering out observations or entire time series.
An always valid but double-edged alternative is to leave the dataset untouched, set $\tilde{t}_0 = \min\left(\left\{t_{0,1}, t_{0,2}, \ldots, t_{0,\mathrm{N}}\right\}\right)$ and $\tilde{\mathrm{T}} = \max\left(\left\{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_\mathrm{N}\right\}\right)$ and treat any observation $\tilde{t}_0 < z^i_t < t_{0,i}$ or $\mathrm{T}_i < z^i_t < \tilde{\mathrm{T}}$ as missing. This adds a data-overhead counting for the extra indexes to mark missing observations for each and every time series.
Those listed here are of course only example of possible approaches to deal with misalignment in a dataset.

2.2.1 Co-variates or predictors

To predict the future, the most popular and simple *time series models* use historical information embedded in the past values alone, therefore they will be able to extrapolate trend and seasonal patterns but be completely blind about factors that can affect the outcome.
Co-variates - or predictors, or *features* - $\mathbf{x}$ are additional pieces of information which can go together with observations to partially explain the data behaviour. They can be global or series-wise, time-varying or -invariant, known continuously or solely for a limited time span. Models employing these information are often referenced as *explanatory models*.
A third opportunity are the *dynamic regression models*: a combination of time series models and explanatory ones, where the prediction depends both on the past observed values and the external information. These models have been called by different names, among them we recall longitudinal models or transfer function models, also in relation with SSMs.
Either if we are using time series models, explanatory models, or a combination of them, the relationship between the dependent variables and the predictors will never be perfect. Therefore it is good practice to integrate in these models an error term that will accept any

change which can not be accommodated by the model itself. The error term has a fundamental relevance in the uncertainty appraisal, defining de facto the spread of the potential expected values.

2.2.2 Metrics and loss functions

In regard to the model's performance, most probably we had heard the word "accuracy" as a general term for both classification and regression tasks. However we can only compute a real accuracy for classification, where we can state how many classes had been predicted correctly against the total number of predictions. For regression tasks we better talk about metrics.
Metric functions - or simply *metrics* - are used to compute an error score which summarizes the predictive skills of a model. Depending on the application more than one metric can be used to assess the fitness of a model to the problem at hand. Often they are confused with *loss functions* since some of them - i.e. the differentiable ones - are used interchangeably as metrics or loss functions. Metrics are used to monitor and assess the performance of a model and can be either differentiable or not since, in general, they are not involved in the training process.

*M*ean *A*bsolute *E*rror (MAE)

The mean absolute error, for a single time series $\mathbf{z}_{t_0:\mathrm{T}_i}$ and related prediction $\hat{\mathbf{z}}_{t_0:\mathrm{T}_i}$, is defined as

$$\mathrm{MAE}\left(\mathbf{z}^i_{1:\mathrm{T}_i}, \hat{\mathbf{z}}^i_{1:\mathrm{T}_i}\right) = \frac{\left(\sum_{t=1}^{\mathrm{T}_i} \left| z_t - \hat{z}_t \right|\right)}{\mathrm{T}_i - 1} \tag{2.6}$$

The application of the absolute function avoids positive and negative errors to cancel each other. Positive and negative are equally weighted under this metric.
A cusp is present when $z_t - \hat{z}_t = 0$, hence the metric is not differentiable. Nevertheless it has been vastly employed in combination with a loss function - i.e. as a *regularization* term - due to its robustness to outliers. Minimization of the MAE will lead us to the median of the data.

*M*ean *S*quared *E*rror (MSE)

The mean squared error substitutes the absolute function with a square one, hampering again the deletion between positive and negative errors. For a single time series $\mathbf{z}_{t_0:\mathrm{T}_i}$ and related prediction $\hat{\mathbf{z}}_{t_0:\mathrm{T}_i}$ MSE is defined as

$$\mathrm{MSE}\left(\mathbf{z}^i_{1:\mathrm{T}_i}, \hat{\mathbf{z}}^i_{1:\mathrm{T}_i}\right) = \frac{\left(\sum_{t=1}^{\mathrm{T}_i}\left(z_t - \hat{z}_t\right)^2\right)}{\mathrm{T}_i - 1} \tag{2.7}$$

Compared to MAE, it is smooth, twice differentiable and penalizes big errors more than small ones. However, as for MAE, the error's direction doesn't play a role in how much the deviation is penalized. It is associated with an "interpretation complexity" since the computed score lives in the squared scale of the original input. Additionally it is more sensitive to outliers than MAE.

If the subtended predictive distribution is - or is assumed to be - symmetric, than choosing either MAE or MSE will lead to the same point forecast since the median and the mean of the data are equal. We will come back to this in Sec. 3.1.6.

*R*oot *M*ean *S*quared *E*rror (RMSE)

Together with MAE and MSE - of which it is the square root - RMSE is a scale-dependent metric commonly used either as loss function or comparison metric.

$$\mathrm{RMSE}\left(\mathbf{z}^i_{1:\mathrm{T}_i}, \hat{\mathbf{z}}^i_{1:\mathrm{T}_i}\right) = \sqrt{\frac{\sum_{t=1}^{\mathrm{T}_i}\left(z^i_t - \hat{z}^i_t\right)^2}{\mathrm{T}_i - 1}} \tag{2.8}$$

The same arguments exposed for the MSE are valid for the RMSE, except for the interpretation. Indeed taking the square root translates again the error back in the original input scale. For the latter property it has been found suitable for context where the error has a direct impact on real-life quantities, e.g. a money loss.

Mean absolute error, mean squared error and root mean squared error all belong to the family of scale dependent errors - i.e. those metrics producing a score over the same scale as the input data - and can not be applied to assess model's fitness across multiple time series.

2.2.3 Problem Statement

Let $\mathbb{D} = \left\{\mathbf{z}^i_{t_{0,i}:\mathrm{T}_i}\right\}_{i=1}^{\mathrm{N}}$ be a dataset of N time series, where $\mathbf{z}^i_{t_{0,i}:\mathrm{T}_i} = \left\{z^i_{t_{0,i}}, \ldots, z^i_t, \ldots, z^i_{\mathrm{T}_i}\right\}$ with $z^i_t \in \mathbb{R}$, and $\left\{\mathbf{x}^i_{t_{0,i}:\mathrm{T}_i+h}\right\}_{i=1}^{\mathrm{N}}$ a second set of associated co-variates. Let also define with $\mathscr{L}(z, \hat{z})$ the loss function of choice. The multi-step ahead time series forecasting can be stated as

$$\min \quad \sum_{\mathrm{N}} \sum_{t=\mathrm{T}_i+1}^{\mathrm{T}_i+h} \mathscr{L}\left(z_t^i, \hat{z}_t^i\right) \qquad \text{where}$$
$$\hat{z}^i_{\mathrm{T}_i+1:\mathrm{T}_i+h} = f\left(\mathbf{z}^i_{t_{0,i}:\mathrm{T}_i}, \mathbf{x}^i_{t_{0,i}:\mathrm{T}_i+h}, e_t\right), \tag{2.9}$$

where the quantities $\hat{\mathbf{z}}^i_{\mathrm{T}_i+1:\mathrm{T}_i+h}$ are to be estimated over the horizon h, given the past observations $\mathbf{z}^i_{t_{0,i}:\mathrm{T}_i}$ and the co-variates $\mathbf{x}^i_{t_{0,i}:\mathrm{T}_i+h}$. When $h = 1$ the multi-step ahead forecasting problem collapses to the one-step-ahead one. Multi-step and one-step-ahead forecasting are instances of *single-valued* forecasts, i.e. we forecast only a single number for each time step in the future. Another common name is *point forecast*

Point forecast

It is the most common forecasting exercise, bound tightly to the use of time series forecasting models. Consuming historical values, we need observations z^i_{t-1}, z^i_{t-2}, z^i_{t-3}, …, to estimate the next $\hat{z}^i_t$; to evaluate $\hat{z}^i_{t+1}$ we have to wait until z^i_t is known, and so on and so forth. We are therefore necessarily proceeding one step at the time; a multi-horizon forecasting - i.e. generate a forecast for multiple time steps in the future - is still possible, but likely it will be a broadcasting of the last forecast for which we have actual information. Model's fitness is usually assessed choosing one or more metrics, like those introduced in the previous section. Prediction intervals can be approximated imposing a Gaussian distribution over the error.

Probabilistic forecast

Probabilistic forecasting is an extension to the problem stated in Eq. (2.9); rather than emitting a single value, a probability is assigned to each element of a set of plausible values. The set of probabilities define the probability forecast. However, assigning a probability distribution to each time step could be unfeasible, hence in practice it is preferable to assign a probability distribution to the horizon of interest. Equation (2.9) is modified accordingly

$$\min \quad \sum_{\mathrm{N}} \sum_{t=\mathrm{T}_i+1}^{\mathrm{T}_i+h} \mathscr{L}(z^i_t, \hat{z}^i_t) \qquad \text{where}$$
$$\mathrm{P}\left(\hat{z}^i_{\mathrm{T}_i+1:\mathrm{T}_i+h} \middle| \mathbf{z}^i_{t_{0,i}:\mathrm{T}_i}, \mathbf{x}^i_{t_{0,i}:\mathrm{T}_i+h}\right), \tag{2.10}$$

The first advantage is that we can directly asses the goodness of our model inspecting the confidence intervals: the narrower a confidence interval is, the more the model has a higher

confidence over its prediction. Confidence intervals are easily built starting from the estimated probabilistic distribution via *quantiles*. A quantile function takes the probability of an event and tells us which is the proximate value of the process for which the probability is lower or equal to that we asked for. Mathematically

$$\mathrm{Q}(p) = \left\{ z : \mathrm{P}r\left(\mathrm{Z} \leq z\right) = p \right\} \tag{2.11}$$

Asking for the Q(0.5) we are requesting the median of the distribution (see also Sec. 3.1.6). Confidence intervals are the preferred strategy to keep uncertainty under control and drive policies. If we try to summarize the obtained probability distribution via a single number, mainly the expected value, we fall back to the point forecast case.

2.3 Shallow solutions

2.3.1 Naïve method

The naïve method is not intended to be used in practice, but mostly as a comparison reference. It is the optimal forecast when the data follow a random walk model and the forecast values $\hat{\mathbf{z}}_{t:t+h}$ are all set equal to the last observed z_t because of the impossibility to predict future movements

$$\begin{aligned} \hat{z}_{\mathrm{T}_i+1} &= z_{\mathrm{T}_i} \\ \hat{z}_{\mathrm{T}_i+2} &= z_{\mathrm{T}_i} \\ &\vdots \\ \hat{z}_{\mathrm{T}_i+h} &= z_{\mathrm{T}_i}. \end{aligned}$$

Seasonal naïve The same informal definition applies to the seasonal naïve method, with the only difference that the foreseen quantity refers to the last season rather than the previous time step

$$\begin{aligned}
\hat{z}_{\mathrm{T}_i+1} &= z_{\mathrm{T}_i+1-m(h_m^+ +1)} \\
\hat{z}_{\mathrm{T}_i+2} &= z_{\mathrm{T}_i+2-m(h_m^+ +1)} \\
&\vdots \\
\hat{z}_{\mathrm{T}_i+h} &= z_{\mathrm{T}_i+h-m(h_m^+ +1)}.
\end{aligned}$$

with $h_m^+ = (h-1) mod\ m$.

2.3.2 Exponential smoothing and its variants

*E*xponential *S*moothing (ES) methods originated in the 1950s and even though they were widely and rapidly adopted in business and industry, they received poor or no attention at all from the statistician due to lack of well-developed statistics foundation; successive works filled the void and a conspicuous number of variation to the method have been proposed. *S*imple - or *S*ingle - *E*xponential *S*moothing (SES) is the basic scheme within the family; it uses a *latent state* l_t - also called *level* - and is in general fitted to a singular time series. At time t the predicted value is found via

$$\begin{cases} l_t = \alpha z_{t-1} + (1-\alpha) l_{t-1}, \\ \hat{z}_t = l_t \end{cases} \tag{2.12}$$

where the parameter α is the *smoothing parameter* - or discount factor, or decay. The general rule of thumb to set the initial values of the scheme is to specify $l_1 = 0$ and $l_2 = z_1$. Another way consists in setting l_2 equal to the target of the dynamic process - if known or guessed - or averaging the first four or five observations.

In contrast with the simple *M*oving *A*verage (MA) where the past observations are associated with a fixed weight - thus the impact of each and every past data point on the current one is identical -, SES applies exponentially decaying weights. The decay rate is controlled by α. The *exponential* denomination derives exactly from the decay's nature of the weights assigned to older observations, as can be shown substituting recursively the smoothing equation for l_{t-1} into that of l_t:

$$
\begin{aligned}
l_1 &= 0, \\
l_2 &= z_1 \\
l_3 &= \alpha z_2 + (1-\alpha) l_2 = \alpha z_2 + (1-\alpha) z_1 \\
l_4 &= \alpha z_3 + (1-\alpha) l_3 = \alpha z_3 + (1-\alpha)\alpha z_2 + (1-\alpha)^2 z_1 \\
l_5 &= \alpha z_4 + (1-\alpha) l_4 = \alpha z_4 + (1-\alpha)\alpha z_3 + (1-\alpha)^2 \alpha z_2 + (1-\alpha)^3 z_1 \\
&\dots
\end{aligned}
$$

or more generally

$$
l_t = \alpha \sum_{i=1}^{t-2} (1-\alpha)^{i-1} z_{t-i} + (1-\alpha)^{t-2} l_2 \quad t \geq 2.
$$

From this unfolding we can see that the weights decrease geometrically and their sum is unity. It also helps us infer the impact of the smoothing factor: a high value for α means that the last observation receives higher attention than the history; on the other hand the more α is set to a small value the more the history of observations acquires importance. In the limiting case of $\alpha = 1$ we are producing a copy of the input time series, only shifted of one time step; if instead $\alpha = 0$ we are relying solely on the first estimation made. Any value of α in the range $[0,1]$ is therefore admissible, but practically speaking the smoothing parameter will frequently be chosen as multiple of 0.1 - like $[0.1, 0.2, 0.3, \dots, 0.9]$ - for convenience.

The title *simple* or *single* refers to the assumption that a single value, the level in this case, is enough to fully characterize the process generating the data.

When the available observations have been processed, the method doesn't have any further information to keep on doing the smoothing operation, hence the only information which can be forecast is the last computed level. Figure (2.1) reports three different curves obtained for three distinct values of the smoothing parameter α, together with the sparse demand already presented in Fig. (1.4). The first 39 data points illustrate the smoothing mechanism, beginning with the 40-th data point we assume that no more observations are available, hence we broadcast the last computed level over the remaining 12 time steps as forecast.

Holt [Hol04] extended the SES with the trend equation; Winters [Win60] built on top, adding seasonality and producing the so called *H*olt-*W*inter (HW) method. Equation (2.13) is an example of HW method with additive trend and multiplicative seasonality

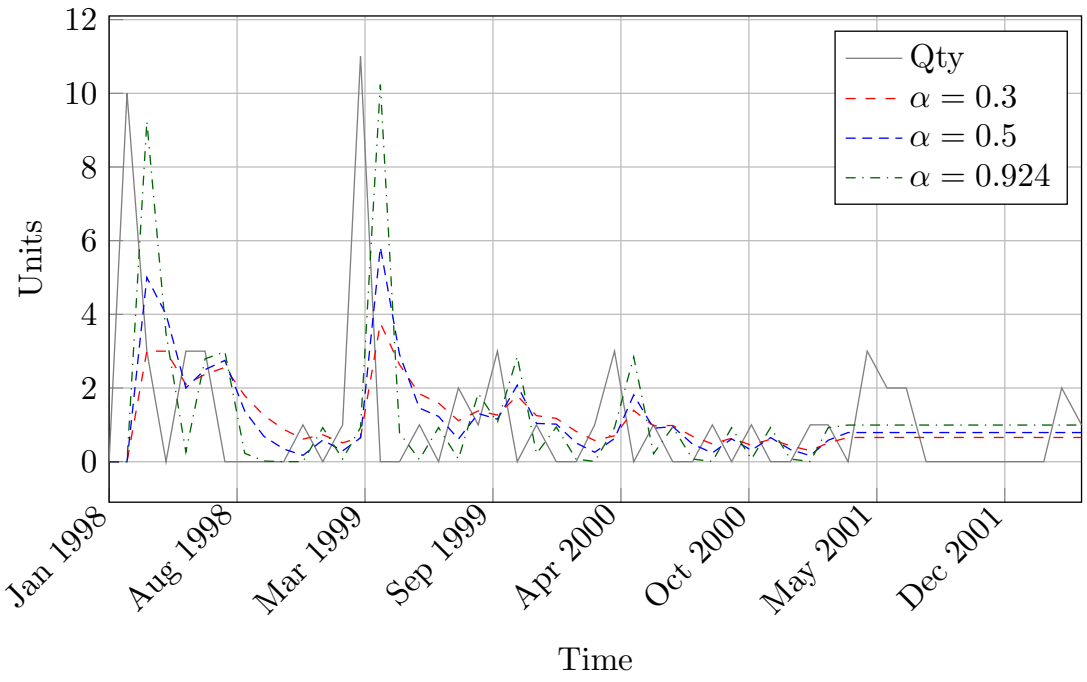


Figure 2.1: Sparse demand of the automotive spare part already presented in Fig. (1.4), together with the forecast obtained through distinct values of the smoothing parameter α. The first 39 months had been used to picture the smoothing procedure, while the last 12 highlight how the SES produces a flat forecast.

$$\begin{cases} l_t = \alpha \dfrac{z_{t-1}}{s_{t-m}} + (1-\alpha)\left(l_{t-1} + t_{t-1}\right), \\ t_t = \beta\left(l_t - l_{t-1}\right) + \left(1-\beta\right) t_{t-1}, \\ s_t = \gamma \dfrac{z_{t-1}}{l_{t-1} + t_{t-1}} + \left(1-\gamma\right) s_{t-m} \\ \hat{z}_{t+h|t} = \left(l_t + h t_t\right) s_{t+h-m}, \end{cases} \tag{2.13}$$

where β and γ are the smoothing factor associated to the trend and the seasonal pattern respectively, m is the seasonality period, s_t is the seasonal factor for the next period and $s_{t-m+h_m^+}$ is the seasonal factor from the last period. The first forecast value $\hat{z}_{t+h|t}$ is available from period $t = m+1m$.

The basic HW method accommodates for a single seasonality - being it daily, weekly or monthly - but modern time series can exhibit patterns at different granularity, e.g. *w*eek-*o*f-*y*ear, *m*onth-*o*f-*y*ear, etc.

Training & prediction The estimation of the parameter(s) of a method or a model is driven by an *objective function* - or *loss function*, frequently called just loss - selected by the practitioner. Different values of the parameter(s) are tested and the one giving the lowest loss - or the highest one, depending on the loss function nature - over the data is retained as the best estimation.

Turning to the smoothing parameters α, β and γ, in practice it is typical to constrain them in the range $[0,1]$, or even more, and test for few values by small increments; e.g. $[0.1, 0.2, \ldots, 0.9]$. The restriction imposed over the parameters is a mean to control the influence of distant past observations over the current forecast, an important aspect of the forecasting task that will come back over and over in the literature. Due to the restrained search space a naïve procedure can be put in place, iterating over all the possible parameters combination and testing for the loss function best outcome. Mean Squared Error - Sec. 2.2.2 - and RMSE - Sec. 2.2.2 - are two of the most used objective functions, not only for ES methods. When ES methods are turned into their state space form, they can rely on a well-founded statistical framework which enables the employment of the *likelihood function* - or just likelihood, i.e. a measure of the probability of the observations arising from the model, the most common one being the Gaussian likelihood - as a loss function and information criterions like the *A*kaike's *I*nformation *C*riterion (AIC) [Aka73] and the *B*ayesian *I*nformation *C*riterion (BIC) [Sch78]. The higher the likelihood, the higher the probability that the model is a good match for the data. Fitting the method to the data is a matter of rolling through the time steps until the whole *training* - or *conditioning* - *range* is covered.

Once an estimation of the parameters is available, the method can be run again over the

training range. The last smoothed value is emitted over the whole *testing range* - or *prediction range* or *horizon* - producing a flat outcome; for this reason predictions produced by a SES method are also referred to as “level". SES alone is not suitable for non-stationary data, i.e. those data exhibiting additional behaviours like trend and/or seasonality.

Variants Holt [Hol04] proposed a new method which adds to the level equation a second SES equation - hence a second smoothing parameter, commonly referenced with the greek letter β - to account for the trend component; for this reason the method is also called *Double E*xponential *S*moothing (DES). The input to the trend equation is the difference between two consecutive levels rather than raw observations; construction of stationary time series through differencing is a technique found many times in literature both for its implementation easiness and possible physical interpretation of the differenced quantities.
A couple of years after, Winters [Win60] reworked the Holt's proposal adding a third exponential smoothing formula - and the relative parameter γ - to deal with seasonality. The method was named Holt-Winter after the two main works that generated it, but it is also known as Triple Exponential Smoothing due to the number of equations involved.
Pegels [Peg69] was the first one to recognise two possible natures of trend and seasonality components: additive - if they impacted linearly the outcome - or multiplicative - if a non-linearity was introduced -. In 1985 Snyder [Sny85] noticed how ES could be considered as a particular case of *I*nnovation *S*tate *S*pace *M*odel (ISSM), see Sec. 2.3.4 - i.e. a model with a single source of noise -. The latter remark was a kind of turning point for the consequent works, aiming at applying the methods or proposing new variants, to the point that it is the foundation of several works in the recent years. Hyndman et al. [HKSG02, HA21] illustrate a complete taxonomy of the 15 available methods, generated by the combination of 5 different trends - none, additive, multiplicative, additive with drift, multiplicative with drift - and 3 seasonality - none, additive, multiplicative -.
In 2020 Smyl [Smy20] brought back to the forecasting scene ES methods, blending SES and Neural Networks in a hybrid method that won the M4 competition.

2.3.3 AR(I)MA and its variants

*A*uto*r*egressive (*I*ntegrated) *M*oving *A*verage models were firstly postulated by Yule [Yul27] and are based on the assumption that each and every time series arises as the realization of two stochastic processes: an autoregressive (AR) process, accounting for the linear dependency of the actual observation upon previous (lagged) observations via multiple regression - whose weights are identified by the vector $[\phi_1 \dots \phi_p]$ plus a constant -; a MA process modelling the error term as a smoothed linear combination of past errors - through the weights $[\theta_1 \dots \theta_q]$ plus a constant -. Although the original ARMA model had been intro-

duced by Whittle [Whi51] in 1951 as part of his PhD thesis, these models known their fortune in 1970 after Box and Jenkins book [BJ76] had been published (the same book in which the notorious Box-Jenkins transformation is outlined).
The established notation refers to the number of past observations to be employed in the AR process as p, while the time window over which the error related MA process is computed is identified by q. The extension of the models with the "integrated" portion marks the entrance in the computer era; the observations are replaced by differences between input values - either consecutive or not, e.g. seasonal differencing -, differences which can be performed several times and easily handled by algorithms. The number of times the differencing operator is applied - i.e. the degree of the operator - is known as d. The dependency introduced by the (I) process equips the models with a long-memory - i.e. also observations far away in time have non negligible effect -, specifically if $0 < d < 0.5$ (and the models are sometimes known as *F*ractional ARIMA, abbreviated in FARIMA [GJ80]). The parameters p, q, and d if necessary, are said to constitute the *order* of an AR(I)MA model and are used to quickly describe a model, i.e. ARMA(p,q) or ARIMA(p,q,d).

Training & prediction AR(I)MA models request the parameters p and q, and d if required, to be known beforehand. An initial guess for p and q, which instead produce unstable assumptions for d, can be presented by AIC and BIC methods. Once the order of the model is chosen, the search for the best combination of the parameters $[\phi_1 \dots \phi_p]$ and $[\theta_1 \dots \theta_q]$ involved in the AR and MA processes can be carried forward. The likelihood, again the most common one being the Gaussian, is the preferred way to train such models via *M*aximum *L*ikelihood *E*stimation (MLE). As for the ES methods, fitting the model is a matter of rolling over the training range applying the AR and MA processes while looking for the highest value - hence the highest probability - of the likelihood.
Once the observations in the conditioning range are consumed, the estimated parameters can be used to produce forecasts. Continuing rolling over the horizon, any missing information required by the processes are replaced: future unknown observations are interchanged with forecasts, future not evaluated errors are set to zero, error from the forecasts become past errors.

Variants The success of AR(I)MA models is easily grasped looking at [DGH06, Tab. 1, Sec. 3] which presents a long list of examples of real applications, numerous variants have been presented over the years among which we recall: the already mentioned FARIMA; ARARMA [Par82], which achieved the best *M*ean *A*bsolute *P*ercentage *E*rror (MAPE)[1] in

[1] $\text{MAPE}\left(\mathbf{z}_{1:\mathrm{T}}, \hat{\mathbf{z}}_{1:\mathrm{T}}\right) = \frac{1}{\mathrm{T}-1} \sum_{t=1}^{\mathrm{T}} \left| \frac{(z_t - \hat{z}_t)}{z_t} \right|$

the first M competition [MAC+82]; *V*ector ARMA (VARMA) [Que57] and its “integrated" version VARIMA, a multivariate generalization of the univariate AR(I)MA; ARMAX i.e. an ARMA model accepting e*X*ogenous factors.

2.3.4 State Space Models

With the name State Space Models we refer to a family of generative models describing the relationship between one or more variables, referred to as *state*, and the observations. SSMs are completely defined by two equations

> *transition* - or state equation -: defines the dynamic behaviour of the model, i.e how the model evolves over time from a state to the next, possibly under the effect of an action;
>
> *observation* - or measurement equation -: represents the relationship between the observation and the state;

to each of these equations a source of error - or noise - is added and in the case a singular source of noise is defined the models are called ISSMs, as already specified. These models were well known in the engineering community since 1960 when Kalman [Kal60] presented his avant-garde work, introducing the idea of the SSMs but more importantly having demonstrated a recursive algorithm - known nowadays as Kalman’s filter - to produce a better prediction while dealing with uncertainty.
To see statisticians make use of these models we have to wait till the 1980s; Harrison et al. [HS76] introduced a particular class of SSMs named *D*ynamic *L*inear *M*odels (DMMs) where both the state and the measurement equations are linear operators. Later in 1984, Harvey [Har84, Har90] exploited SSMs and Kalman’s filter to present his *S*tructural *T*ime *S*eries (STS) idea in which more than one state equation - one for each elemental component, level trend and seasonality, of the time series - are there, the standard measurement equation plus possible exogenous factors. One year later, as already mentioned before, Snyder considered the relationship between ES methods and SSMs but he was just the first one to make the link. Over the years other methods and models received a re-treatment falling under the SSMs umbrella, e.g. AR(I)MA and ARMAX.

Training & prediction With SSMs we enter the domain of stochastic models in which *a priori* assumptions are required. The most common ones, which holds for several real applications and ease the calculation of the Kalman’s filter, are the linearity of the system and a Gaussian distribution laid upon the error terms. Trying to outline the fitting procedure without being too technical for the sake of conciseness, two different steps can be summarised:

prediction: applying the state equation to the a-priori state and its error, the current state is generated as well as its related noise;

update: incorporating the observation through the measurement equations the current state and error are updated, generating the
a-posteriori state and noise - i.e. the input for the next iteration -.

The forecasting step can be executed seemingly in the same way as ES methods and AR(I)MA models, but in contrast SSMs are actually producing an entire forecasting distribution instead of point forecasts. Once the conditioning range has been exhausted, estimations of the state and the noise are available; the iteration producing new states and noise continues over the horizon but this time, without new measurements to correct for, the noise - hence the uncertainty - starts to raise as we move further into the testing range. This effect - sometimes called *shotgun effect* in the Supply Chain context, due to its visual appearance - is desirable in virtue of the intuitive insight it gives: the more the model explores a foreign region, the more the uncertainty grows, the more also extreme scenarios become probable.

Variations Unfortunately the Gaussian error assumption is the main reason why practitioners from different fields restrain themselves from using SSMs. Modelling count data - i.e. integer quantities - the Gaussian assumption doesn't hold anymore - e.g. the forecast could end foreseeing negative values - and other types of distributions are required to reflect the data characteristics, for example the Poisson distribution or the Negative Binomial one. In this case the computation of the Kalman's filter becomes problematic and more complex schema are employed, namely the Extended Kalman's Filter [Sor85] - the system is approximated with a dual linear version around the state and the computation is carried over - and the Unscented Kalman's Filter [JU97] - for highly non-linear systems, uses a set of sampled points around the mean and construct a surrogate of the true distribution -. Other filtering techniques can be exploited - e.g. Particle Filters [DM97, LC98] - but they require advanced knowledge, a proper implementation as well as computational time and cost.
SSMs are not limited to the forecasting task but are extensively adopted for filtering and smoothing works in various context, e.g. sensors fusion.
The idea of STS has been brought back by Facebook - now Meta - in the past years with the project known with the name *Prophet* [TL18]. For a full analysis of SSMs, their variants and the relationship with other existing methods and models Harvey [Har90] and Koopman et Durbin [KD01] are key references.

The works so far reviewed, even though different in nature, can all be categorised as white-box models. They presume a certain degree of linearity in the relationship between the

dependent variable and co-variates; linearity which can be exploited by an expert user in order to make the right assumption and use the right tools. These methods are children of their time. Pattern recognition and modeling expertise was highly valued back in the 1980s, maybe more than they are esteemed today. Personal computers were a reality - the average size shrank from room to desk - but could only be sold in limited quantities - mainly to technicians of the field - and were far away from being widespread as they are nowadays. The linear relationship idea instead is something more complex to deal with and deeply-rooted in human beings: the *linear bias*. As clearly exposed by de Langhe et al. [dLPR17] decades of research in cognitive psychology have shown how human mind struggles to grasp non-linear relationships, human beings are more keen on straight lines and clean geometry rather than complexity. Linear reasoning serves well in many situations but in many others relying on gut feelings can lead to modelling errors and poor decision making.
The potential to automatically discover non-linear relationships just processing the data at hand is appealing and is driving attention on Neural Networks, however it implies approaching a black-box proposal which could not fit well in a human-centered decision making processes.

2.4 Neural Architectures

Neural Networks - a.k.a. *A*rtificial NNs (ANNs) or Simulated NNs, Fig. (2.3) - are a subset of ML and the heart of Deep Learning algorithms. Their name and structure are inspired by the human brain, since they are designed to mimic the way in which biological neurons interact each other. They consist of a (fully) connected graph composed of different layers, each of which consist of a collection of nodes also called artificial neurons. Two layers are always recognizable in any NN architecture: the input layer - the entry point of the network - and the output layer, where the predictions are collected. The number of nodes in the input layer accommodate the number of independent variables in the model, similarly the number of nodes in the output layer adjusts to the number of expected output variables. The number of hidden layers present in the architecture can vary from one to many - architecture with more than one hidden layer will be classified as *Deep*, Fig. (2.4) - depending on the assigned task. Each neuron in the i-th hidden layer receives from the layer $(i-1)$-th - either the input layer or another hidden one - one or more inputs x_j and assign a weight w_i^j to each of them. The inputs are combined together and passed through a function f also known as *activation function*. The result is passed to the next step in the network.
The main idea behind NNs can be dated back to 1943 when McCulloch et al. [MP43] made the first hypothesis on how the human brain could produce complex patterns through the neuron interactions and linked neurons with a binary threshold to the Boolean logic. Fif-

teen years after Rosenblatt [Ros58] built on top of McCulloch et al. work, adding weights to the equation and presenting the oldest known neural network, the *Perceptron* - Fig. (2.2) - i.e. a single artificial neuron. Fast-forward to 1989, Lecun et al. [LBD+89] applied the backpropagation algorithm [RHW85] to successfully train a neural network to recognize hand-written zip code digits provided by the U.S. Postal Service. While transitioning from a "passing fad" to a valid modelling alternative, NNs started to compete with established methods: they could have been computationally demanding but offered better results using the same data. A first utilization of ANNs for time series forecasting is dated back to 1996 by Czernichow et al. [CPI+96] who applied it to electricity load data. Nineteen ninety-nine marked a significant evolutionary step for NNs; the growing computer's processing capabilities and the commercialization of the first *G*raphics *P*rocessing *U*nits (GPUs), drastically reduced NNs' computational burden and paved the way for the forthcoming works. Over the last few decades NNs prevailed in various fields like Computer Vision, Natural Language Processing, autonomous vehicles and games with an abundant collection of architectures and works published.

Supervised & unsupervised learning Before seeing in the coming paragraph how a NN is trained, a mention about *Supervised* and *Unsupervised* learning is mandatory. In the corresponding paragraphs for each of the beforehand introduced shallow solutions, we always passed the current observation either because it was stored for later use or since it was involved in the computation of the current step. This is an example of supervised learning: we fed to the model both the co-variates and the observations; the model used the co-variates to build its own representation of the phenomenon and tested its hypothesis against the observations. In this context observations take also the name of *labels* - or *targets* -, as a consequence supervised learning is said to work on *labelled* data. As it will be explained in the next paragraph, the error will be "pushed" through the NN to gradually update its parameters.
Neural Networks are however used also in cases where no labels are available. For instance discover groups of similar examples within the data - i.e. *clustering* - or project the data from an high-dimensional space to a lower-dimensional space - *dimensionality reduction* -, with the smallest information loss possible. We refer to this case with the name of unsupervised learning, as a distinction with the former case.

Training & prediction The objective is always the same: find by some mean the best combination of parameters producing the best outcome for the loss function of choice. The difference now relies in the number of parameters involved in the search; while ES methods, ARIMA(p,q,d) models and SSMs had just a bunch of parameters to be trained, NNs present instead a number of parameters which grows directly with the number of nodes the network

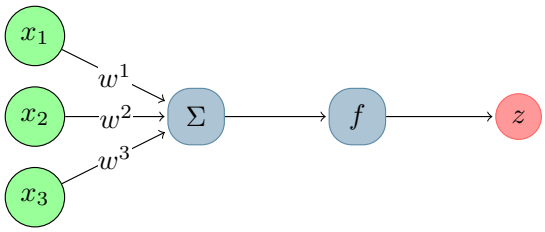


Figure 2.2: A *Perceptron*: the simplest form of NN composed by a single artificial neuron. Presented by Ronsenblatt [Ros58].

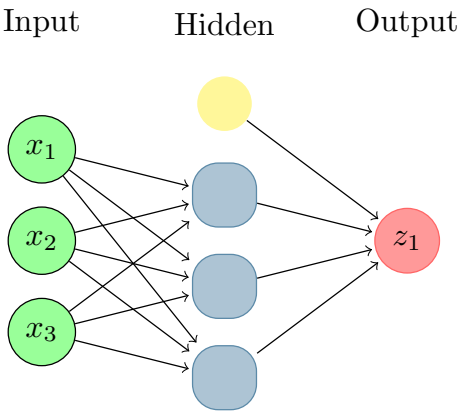


Figure 2.3: ANN with one hidden layer - plus a bias (the yellow node) - taking 3 inputs and returning one value.

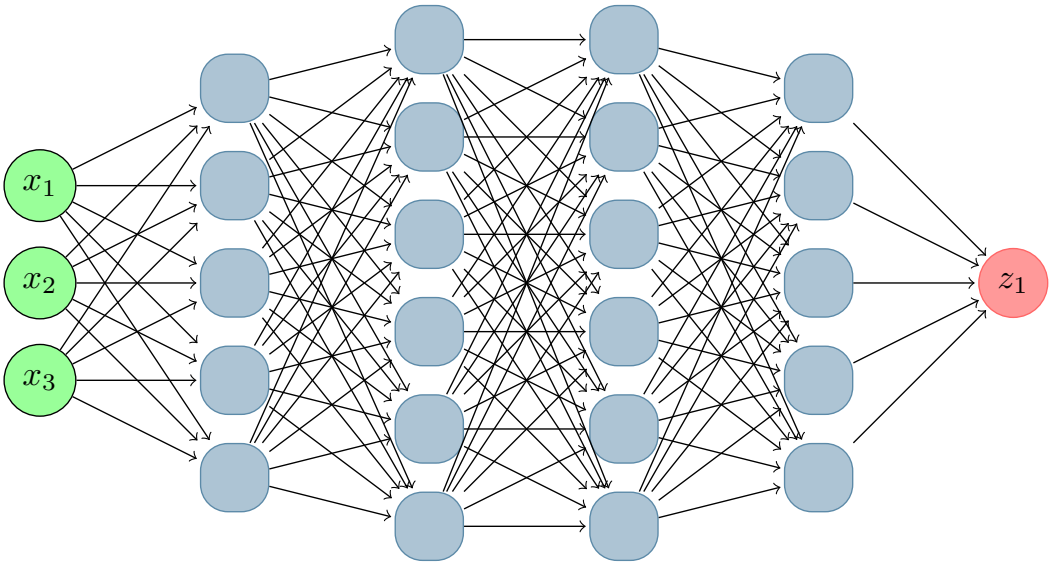


Figure 2.4: Deep NN with 4 hidden layers - no bias - taking 3 inputs and returning one value.

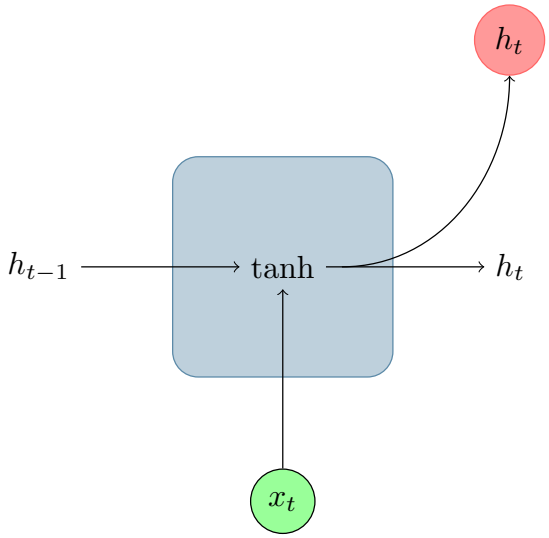


Figure 2.5: Basic Vanilla RNN layer. At each time step t the layer accepts the current input x_t and blends it with the past information in h_{t-1}. The new information is stored in state h_t which is passed to the next layer. tanh and σ - standing for the sigmoid function - are two of the most common activation functions employed.

is made of and their incoming edges. When the number of weights is too large, searching in a “brute force" fashion as described so far is of course inconceivable, this is the reason why backpropagation was such an effective way to train NNs.

Data flow into the input layer, get filtered by the network and an estimate is presented at the output layer. The result is harvested, the loss function is computed and the error committed is quantified. Backpropagation allows the error to flow back from the output layer till the input layer, updating the network weights on the fly. At the next iteration over the data - known as *epoch* - the network emits a new output and the process continues until a threshold on the error or on the maximum number of epochs is reached. This procedure has great advantages in searching for a good combination of a huge collection of parameters as well as different pitfalls, one of which is known as the *vanishing/exploding gradient problem*. Imaging to stack the hidden layers one on top of the other - starting with that immediately after the input layer and finishing with the one right before the output layer - as soon as the network gets wider the lowest layers could either not receive any error information coming from the upper layers, ending not updating their weights - in vanishing case - or ending in instability - in the exploding case -. An early solution was a layer-by-layer pre-training step but other solutions emerged over the years either as standalone or influencing network design.

Prediction is accomplished feeding new unseen data to the network and observing the result emitted at the output layer.

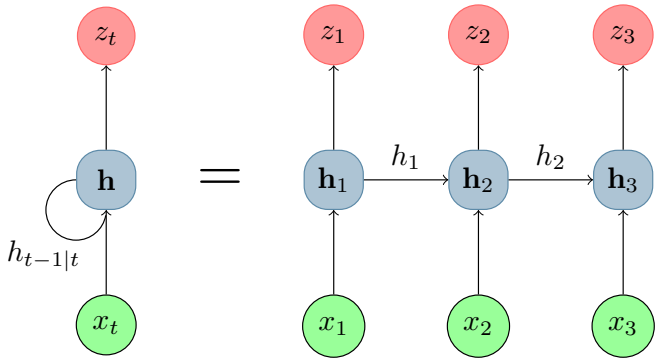


Figure 2.6: A RNN is most of the time depicted in its rolled form, where the loop connection is visible. However it is useful to take a look at its unroll version to better grasp how the information retained in the state h passes from a layer to its successor.

Variants The basic architecture presented in this section is also called *Feedforward* NNs (FNNs) as: each node is connected to each and every one node in the layers immediately before and after; data flow in a forward way, from the input layer to the output one. Many adaptations have been proposed in literature for any kind of task tackled, some of them are out of the scope of the current manuscript, e.g. Convolutional NNs, hence only those relevant to the subsequent presentation are here briefly described.

*R*ecurrent NNs (RNNs) - sometimes also named Vanilla RNNs - are derived from FNNs and it is said that they possess a “memory" since they keep an internal *state h* - i.e. a representation of the context seen so far by the network - which is propagated through time via a loop connection. Using this “memory" this class of networks can exhibit temporal dynamic behaviour, reason for which they are extensively used for problems involving sequences or time series data. The basic RNN layer is depicted in Fig. (2.5). At each time step t the RNN accepts an input x_t, processes it together with its state h_{t-1} - and emits a new state h updated with the information carried by x_t. The state h_t becomes an input for the network at time $t+1$ together with x_{t+1}. How much of the information stored in x_t contributes to the update of h_{t-1} is determined by inner activation function, the most popular one being the hyperbolic tangent tanh. RNNs are also the first example of parameter sharing across different layers: as can be seen in Fig. (2.6) we can unroll the loop and think the RNN as multiple copies of the same network, each passing a message to a successor; since it is always the same network repeating over and over, weights are also retained while iterating.

Exploding and vanishing gradient problems are still an issue for this kind of networks, but is solved in general reducing the number of recurrent layers. These problems are also connected to the *long-term dependency problem* affecting RNNs [BSF94]: if the relavant information for the current forecast are not stored in the recent past but in a far away observation, RNNs fail in handling the connection.

It is common to use interchangeably the term RNNs to identify either the vanilla architecture or the family of recurrent networks embodying the vanilla and its adaptations.

Introduced by Hochreiter et Schmidhuber [HS97] in 1997 as a solution to the vanishing gradient problem, *L*ong *S*hort-*T*erm *M*emory networks (LSTMs) are themselves a variation of RNNs. Solving the vanishing gradient problem, the authors indirectly address also the long-term dependencies issues aforementioned. The main contribution is the extension of the vanilla RNN with a “gating" mechanism plus and additional context c_t also called *cell state*. The main concept behind the introduction of the latter state is that c_t is responsible for capturing the long-term dependencies - being updated less frequently with respect to h_t and running along the whole chain - while h_t will continue to focus on the local information regarding the recent past.

In order to control the information flow and decide how much of it is worthy of passing through and how much of it can be discarded, three different gates have been added: a “forget" gate, an “input" gate and an “output" one. The fist gate encountered is the forget gate - σ_f - which decides how much of the current information - coming from the combination of h_{t-1} and x_t - will be discarded and removed from the cell state. Next is the input gate - σ_i and $\tanh_i$ - determines which part of the information is going to be updated and produce cell state candidates that update the current c_{t-1} to output the current c_t. Finally also a new state h_t is produced based on both the input and a filtered version of the new cell state. The basic LSTM layer is depicted in Fig. (2.7).

This architecture is far more complex than the Vanilla RNN adding more weights and computational time to the table. LSTMs themselves are subject to variations that introduced more recurrent architectures addressing the computational problem or introducing a new hypothesis about how memory works in biological neurons.

This is by far the most used architecture in time series forecasting, being challenged only by the introduction of the - still immature - *Transformers* [VSP+17] which are essentially based on the concept of “attention" [BCB15].

Since RNNs in general and LSTMs in particular are well suited for sequence data, they have found application in the context of Natural Language Processing and Neural Machine Translation, especially arranged in the architecture better known as *Encoder - Decoder*. As sketched in Fig. (2.8) this particular network is composed by two sub-networks, each one in general made of several RNNs connected together. The first sub-network is called *Encoder* and accepts the sequence in input; the second is named *Decoder* and accepts the final hidden state of the Encoder in order to produce the output sequence. It is not unusual to find works where the Encoder and the Decoder networks are played by the same RNNs.

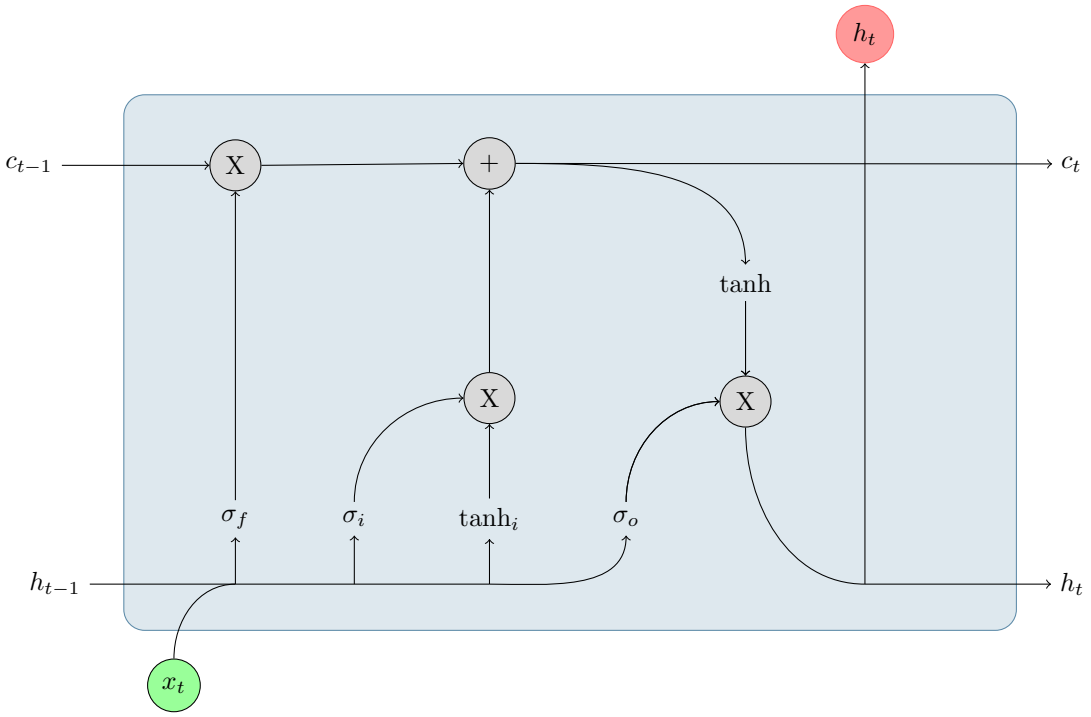


Figure 2.7: Basic LSTM layer.

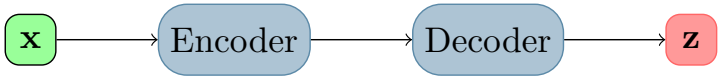


Figure 2.8: Encoder-Decoder architecture. Each *coder sub-networks is made of one or more RNNs composed by one or more RNN layers.

Even if nowadays NNs - and ML techniques in general - are well accepted in practice they are still not recognized as an established forecasting technique by practitioners, as noted by Hippert et al. [HPS01] previously and Smyl [Smy20] later, with a significant gap between the field progress and the industrial adoption. This gap can be possibly be rooted in the business needs for model transparency - which facilitates the explanation -, training efficiency - especially strict and restrained delivery time -, trustworthiness and ease of implementation. ARIMA, Exponential Smoothing and other standard methods and models were the solutions at hand to answer these requests, being easy to implement, understandable by everyone and requiring less data. The Makridakis Competitions - Sec. 3.1.5 - are a meeting spot for these two, sometimes clashing, realities and highlight an evolutionary timeline of forecasting methods.
For a more in depth review of time series forecasting methods and models, covering the period 1985-2005, refer to De Gooijer et al. [DGH06] and Hippert et al. [HPS01].

2.5 Summary

Time series analysis (Sec. 2.1) and time series forecasting (Sec. 2.2) are two sides of the same coin. Time series analysis tries to answer to the *why* behind a time series behaviour, decomposing it into components and making assumption about the data distribution. It demands knowledge of the application field, without discarding an expert judgement. As a consequence it is prone to cognitive bias in the decomposition process or in the modelling choices. Several properties are of interest to accomplish this task:

Univariate & multivariate If each and every observation of a time series is a scalar, we say that it is univariate; in contrast if each and every observation groups several values, we are dealing with a multivariate time series.

Regular & irregular If the frequency at which observations are recorded is constant, then the corresponding time series is regular; irregular otherwise.

Stationary & non-stationary A time series can be qualified as stationary if its statistical properties - i.e. the moments of its generative distribution, such as mean and variance - don't change over time. A clear example is a white noise, i.e. a time series sampled from a Gaussian distribution. Differently the time series is noted to be a non-stationary one. Stationarity is a type of dependence structure and a handy property, indeed if a sequence is stationary than plenty of results which hold for independent random variables also hold for the sequence.

Once all the principal characteristics of the data had been shown, the insights can be used to select a model and continue with time series forecasting (*how* a time series will behave). Forecasting a single value, e.g. the mean of the time series, takes the name of point forecast. If instead we are interested not only in the quantity itself but also how much it can change, we should prefer a probabilistic forecast task. Either if we are accomplishing a point forecast or a probabilistic one, we can project the time series into the future just for a single step (one step ahead) or for a fixed number of steps (multi-step ahead). Section 2.3 reviewed all the classical methods and models to accomplish the forecasting exercise. Classical but not antiquated since most of them - if not even all - remain undisturbed in the companies' working flow. Section 2.3.2 and Sec. 2.3.4 are of particular interest for following chapters.
Established systems inevitably expect knowledge of the application domain to exploit a priori information. Neural Networks - and AI models and methods at large - are appealing for their ability to autonomously discover patterns in the data.
Nonetheless they represent a standard shift, from white-box models to black-box ones. This change could or could not be appropriate for distinct decision making processes, where transparency is a worthwhile quality. Moreover the great majority of neural architectures don't directly handle the probabilistic forecast problem - for which adaptations are required -, being relegated to point forecasts tasks. Sequences are treated with specific neural architectures, called Recurrent Neural Networks, which retain a representation of the context seen so far - summarised into a state - and propagate it through time. Recurrent Neural Networks come with several designs - some not treated in this context - but LSTM is by far the most used architecture deign in practice. Attempts to merge standard methods and NNs are not lacking, a top example is the ES-RNN [Smy20] architecture conceived by Smyl during the M4 competition.
For a more in depth review of time series forecasting methods and models, covering the period 1985-2005, refer to De Gooijer et al. [DGH06] and Hippert et al. [HPS01].

Chapter 3

Demand forecasting

Contents

3.1 Introduction

Economics and Supply Chain fields have been studied for decades and a copious literature exists. The review will be organised as follow: we will present some of the most used and well-known methods and models in industry, followed by more modern architectures including some *S*tate-*o*f-*t*he-*A*rt (SotA) design like DeepAR [SFGJ19] and Deep State Space Models [RSG+18], both from Amazon. We will also review the history of time series forecasting through the various M Competitions and the respective winners which give us an idea of how the landscape of this key component in many industrial and business decision processes is evolving.

3.1.1 Hierarchical and cross-sectional

Time series can often be naturally disaggregated by distinctive attributes of interest or by a geographical division. Taking again the hypermarket example we can start sorting the time series by membership department - e.g. food -, continuing with the SKU's category - e.g. vegetable, fruit, etc. - and the SKU's type - e.g. salad, apple, etc. - ending with the SKU itself, see Fig. 3.1. Naming the hierarchy levels starting with ℓ_0 for the top level - or the leftmost one in the figure - and ending with ℓ_n for the bottom one - or the rightmost one in the figure -, the peculiarity of hierarchical time series lies in the dependency between successive levels. The top level ℓ_0 is the aggregation of the time series at level ℓ_1, which in turn gathers the values of time series at level ℓ_2 and so on and so forth, until we reach the last level.

The direct implication is that a forecast should maintain the same property; forecasting time series at level ℓ_j, we would like to add up them to those at levels ℓ_{j-1}, then aggregate at ℓ_{j-2}, etc. while climbing the hierarchy. In other words we are asking for a *coherent* forecast that has to be consistent with the aggregation structure.

An hypermarket in general is part of a chain with stores scattered over a region, a state or a country. Each store will inherit the same hierarchy from the central management; the same SKU is now sold in different places and we could be interested in understanding how a specific product behaves not only in a single store, but across the chain and for a specific country. In this case we talk about cross-sectional time series.

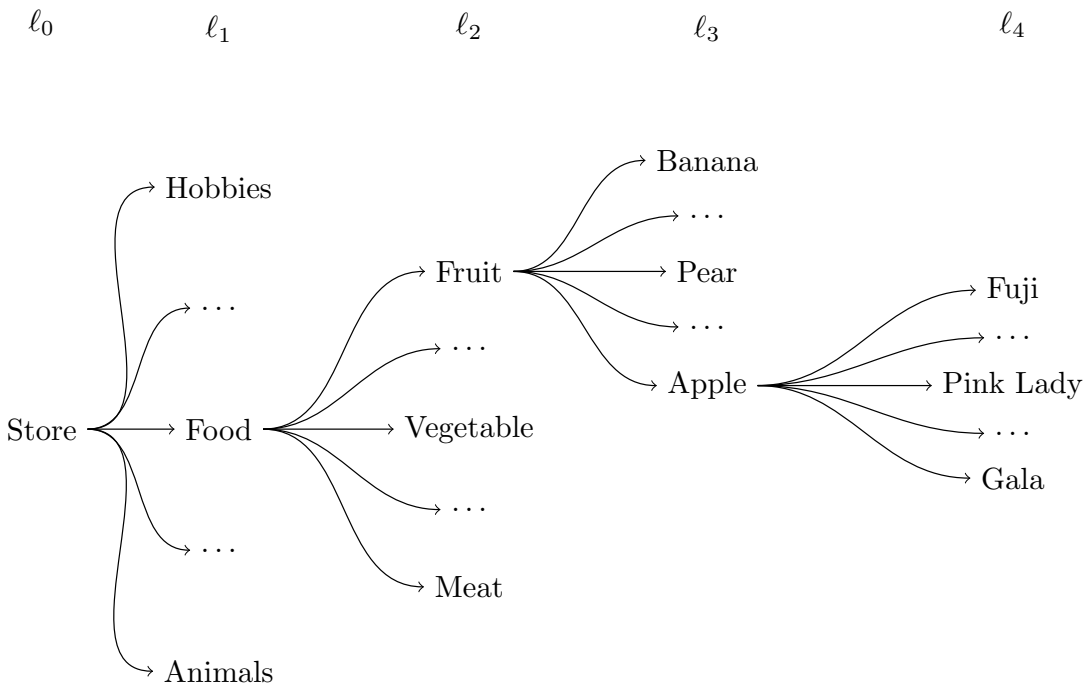


Figure 3.1: Example of hierarchical time series within a hypermarket. Each level in the hierarchy can be treated either as a singular time series or as the result of the aggregation at the lower level.

3.1.2 Count time series

Fields such as Economics, Epidemiology, Finance, Supply Chain, etc. intrinsically generate non-Gaussian time series comprised of non-negative integers. These time series are called *count time series*.

Count time series often expose a non-negative autocorrelation and could be *over-dispersed*, i.e. their variance is greater than their mean. Depending on the field and/or the granularity of the sampling process, count time series can be characterized by a high zero values tally. Various distributions to handle count data are recommended in the literature, but two are the most established one: the *Poisson* distribution - $P(z;\lambda)$ - and the *Negative Binomial* one - $NB\big(z;r,p\big)$; both of them accounting for a positive probability at zero.

Poisson

Is the most basic discrete probability distribution employed to model count data. It expresses the probability of a given number of events occurring over a fixed time interval. The occurrence of an event does not affect the probability of a second event to happen, implying an independence between events. The Poisson distribution is characterized by a single parameter λ - the mean of the distribution, also called *rate* - and the following *p*robability

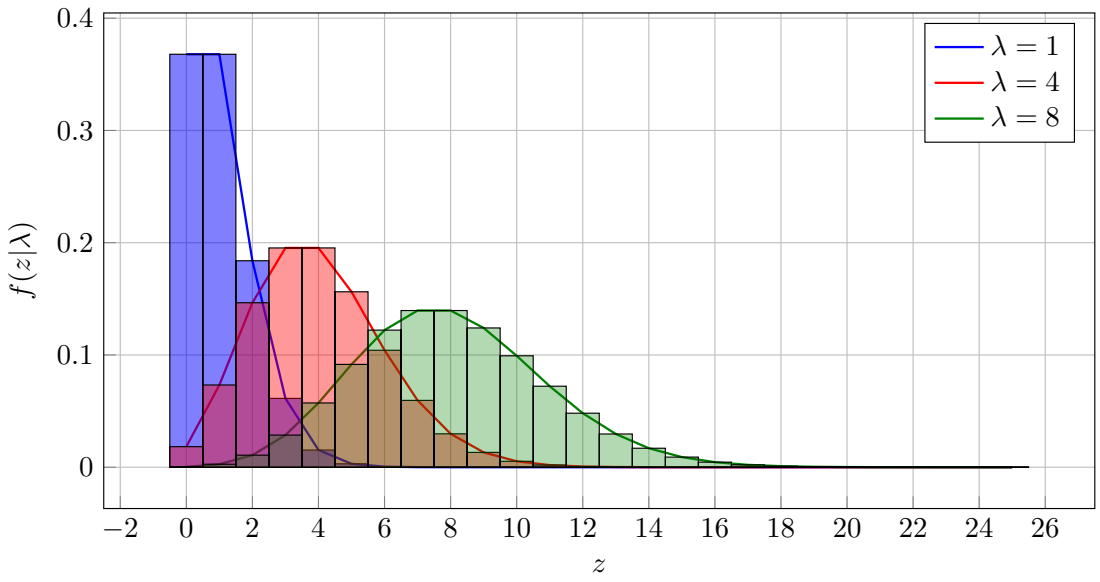


Figure 3.2: Examples of three Poisson distributions with the different λ means.

*m*ass *f*unction (pmf) [1]

$$f(z|\lambda) = \frac{\lambda^z e^{-\lambda}}{z!}\,. \tag{3.1}$$

The Poisson distribution is said to be *equi-dispersed*, i.e. the variance of the data σ^2 equals the mean of the data λ ($\sigma^2 = \lambda$). Some examples for different values of λ are presented in Fig. 3.2.

Negative Binomial

*N*egative *B*inomial (NB) is again a discrete distribution - even though extensions to the continuous domain exist in literature - and a generalization of the Poisson one, made for the over-dispersion case.
The distribution is completely defined by two parameters: r, the number of failures, and p, the success rate. The distribution pmf is defined as follows

$$f\left(z|r,p\right) = \frac{\Gamma\left(z+r\right)}{z!\Gamma\left(r\right)}\left(1-p\right)^z p^r \tag{3.2}$$

where $\Gamma(\cdot)$ is the Gamma function. In the limit $p \to 0$, the NB degenerates to a Poisson

[1] The function describing the probability of an observed value z under a distribution is called: probability mass function if the distribution is discrete - denoting the probability that a discrete random variable will take on a particular value -; *p*robability *d*ensity *f*unction (pdf) if the distribution is continuous - giving the probability that a continuous random variable will lie between a specified interval -.

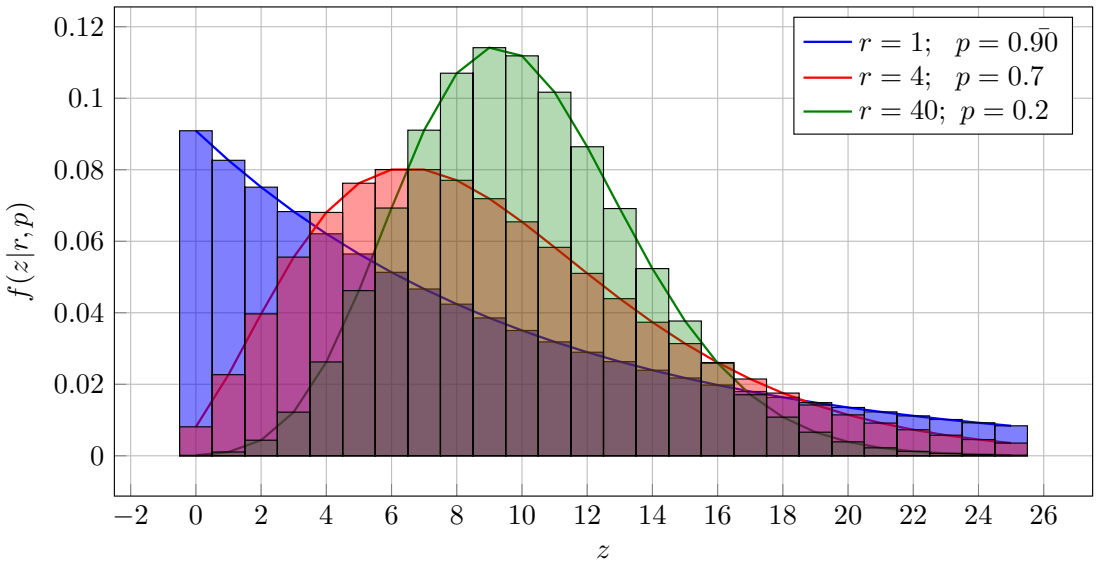


Figure 3.3: Examples of three Negative Binomial distributions with the same mean μ (μ = 10) obtained for different values of the characterizing parameters r and p.

distribution. For a given pair of (r, p) the mean and variance of the NB are defined as

$$\mu = \frac{pr}{(1-p)}; \quad \sigma^2 = \mu + \frac{\mu^2}{r} = \frac{pr}{(1-p)^2}.$$

Figure 3.3 depicts three different NB distributions with the same mean μ = 10, but realizing it via different parameter pairs, resulting in distinct variances and probability mass shapes. Depending on the interpretation of the parameter r, being it either the number of failures or the number of successes, alternative formulations are available. Sometimes it is beneficial to substitute r with its reciprocal $\delta = 1/r$ which is also known as the *shape* parameter. Snyder et al. [SOB12] proposed a definition characterized by the mean μ - which can also vary over time - and η such that

$$p = \frac{\eta}{1+\eta}; \quad r = \frac{(1-p)\,\mu}{p} = \frac{1}{1+\eta} \cdot \mu \cdot \frac{1+\eta}{\eta} = \frac{\mu}{\eta};$$

$$f(z|\mu,\eta) = \frac{\Gamma(z+\mu/\eta)}{z!\,\Gamma(\mu/\eta)} \left(\frac{1}{1+\eta}\right)^z \left(\frac{\eta}{1+\eta}\right)^{\mu/\eta}. \tag{3.3}$$

Again Snyder et al. [SOB12] conducted an in-depth comparison of different statistical methods grounded on Poisson, NB and Hurdle shifted Poisson, reaching the conclusion that NB worked better in representing intermittent demand data. The work also presented a NB auto-regressive method which had been the SotA in demand forecasting tasks and had

been used as competing baseline in [SFGJ19]. Having been proven to be effective in our experiments, the notation in Eq. ((3.3)) has been adopted as the preferred NB formulation throughout the present research work.

Under-dispersion Negative Binomial adjusts for Poisson over-dispersion but is not suitable if the time series is *under-dispersed*, i.e. the variance of the data is smaller than the mean.

When dealing with under-dispersion the first suggested approach is to model the data with a Binomial distribution. The Binomial distribution is defined by n - the number of trials -, p - the success probability - and $q = (1-p)$ parameters. However when we move from a Poisson distribution to a Binomial one, we are restraining the support for our data from the entire domain of natural numbers including zero - $\mathbb{Z}_0$ - to the support $\{0,\dots,n\}$. This restriction could have either no effect or be impracticable, depending on the problem under examination.

*G*eneralized *P*oisson (GP) distribution [Fam93, Fam97] could be another candidate to focus on the under-dispersion problem. In the GP definition, the Poisson distribution is extended with a dispersion parameter ϕ: when $\phi > 0$, the distribution address the over-dispersion case; for $\phi = 0$ the distribution is equi-dispersed, i.e. it degenerates to a Poisson distribution; if instead $^{-2}/\lambda < \phi < 0$, the variance is under-dispersed and the distribution accounts for it. Generalized Poisson is not extensively used in practice, as far as we know, because of the deviates generation process which count five different algorithms - that can lead to significant computation overhead if not properly implemented - to be employed for distinct parameter compositions [Fam97].

Likelihood, Log-likelihood & Negative Log-Likelihood

Equation (3.1) and Eq. (3.2) provide us a way to calculate the probability of observing a particular set of outcomes by making suitable assumptions about the underlying stochastic process. Generalizing, we could say that the probability of observing z under a set of distribution's parameters θ - e.g. $\theta_{Poisson} = (\lambda)$ - is given by $f(z|\theta)$. We say that z is conditioned on θ. When modeling real life stochastic processes, often parameters θ are not know and have to be estimated. The optimization process should refine θ such that the probability $f(\cdot)$ of observing z given the parameters are maximized. Therefore we are swapping the role of z and θ since the latter are unknown. The new function $\mathscr{L}(\theta|z)$ is called *likelihood* and the associated optimization process is known as *M*aximum *L*ikelihood *E*stimation (MLE).

Provided a set of *i*ndependent *i*dentically *d*istributed (i.i.d.) observations - or a time series - **Z**, the likelihood is written as

$$\begin{aligned}\mathscr{L}\left(\theta|\mathbf{Z}\right) &= f\left(z_1|\theta\right)\cdot f\left(z_2|\theta\right)\cdot\cdots\cdot f\left(z_{\mathrm{T}}|\theta\right)\\ &= \prod_{j=1}^{\mathrm{T}} f\left(z_j|\theta\right).\end{aligned} \tag{3.4}$$

To maximize this function we would take its derivative and solve for equality to zero. However the long chain of multiplications involved will result in an excessive long expression. Since we are interested only in the parameters and not in the function itself, it is handy to work with the *log-likelihood* $\ell\left(\theta|\mathbf{Z}\right)$ function

$$\begin{aligned}\ell\left(\theta|\mathbf{Z}\right) = \ln\left(\mathscr{L}\left(\theta|\mathbf{Z}\right)\right) &= \ln\left(\prod_{j=1}^{\mathrm{T}} f\left(z_j|\theta\right)\right)\\ &= \sum_{j=1}^{\mathrm{T}} \ln\left(f\left(z_j|\theta\right)\right).\end{aligned} \tag{3.5}$$

Since the logarithm is a monotonic function, the maximum is preserved.
Gradient descent methods, either stochastic or not, are regularly used - and implemented - to minimize a (error) function instead of maximizing it. We can surely transpose the log-likelihood maximization problem into its dual, taking the *negative log-likelihood* and searching for its minimum. The log-likelihood functions for the two distributions introduced in the previous section are shown in Eq. (3.6) and Eq. (3.7) respectively.

$$\ell_{\mathrm{P}oisson}\left(\lambda|\mathbf{Z}\right) = -\mathrm{T}\lambda + \ln\left(\lambda\right)\sum_{j=1}^{\mathrm{T}} z_j - \sum_{j=1}^{\mathrm{T}} \ln\left(z_j!\right) \tag{3.6}$$

$$\begin{aligned}\ell_{\mathrm{NB}}\left(r,p|\mathbf{Z}\right) &= -\mathrm{T}\ln\Gamma\left(r\right) + \mathrm{T}r\ln\left(p\right)\\ &\quad + \sum_{j=1}^{\mathrm{T}} \ln\Gamma\left(z_j + r\right) - \ln\left(z_j!\right) + z_j\ln\left(1-p\right)\end{aligned} \tag{3.7}$$

3.1.3 Erratic, Lumpy, Smooth & Intermittent series

In 2005 Syntetos et al. [SBC05] tried to schematize a mechanical procedure to identify demand patterns. Expertise was driving the demand categorization, especially in many inventory control practices, with expert arbitrarily sorting the demand patterns to then proceed with model selection and parameter optimization. The authors thought this process could have been automatised. They conceived a classification purely based on two factors for any given time series $\mathbf{z}_{1:\mathrm{T}}$: the *a*verage inter-*d*emand *i*ntervals - i.e. average distance between two consecutive positive demands -

$$\mathrm{ADI} = \frac{Total\ number\ of\ periods}{Number\ of\ positive\ demand\ buckets} = \frac{\mathrm{T}}{\sum_{i=1}^{\mathrm{T}} \mathrm{I}_{\neq 0}(z_i)} \tag{3.8}$$

and the squared coefficient of variation

$$\mathrm{CV}^2 = \frac{\sigma^2}{\mu^2}. \tag{3.9}$$

The indicator function $\mathrm{I}_{\neq 0}(\cdot)$ is defined as

$$\mathrm{I}_{\neq 0}(x) = \begin{cases} 1, & x \neq 0 \\ 0, & otherwise \end{cases}$$

while σ and μ are the sample standard deviation and the sample mean, respectively, of the non-zero values of $\mathbf{z}_{1:\mathrm{T}}$.

They also proposed a new forecasting method and derived cut-off values for the two parameters, comparing the MSE of their method against the error committed by the Croston [Cro72] model. The goal was to have fixed thresholds to confidently suggest their method or the Croston one for forecasting. The derived cut-off values together with the four classes arising from the authors' proposition are shown in Fig. 3.4.

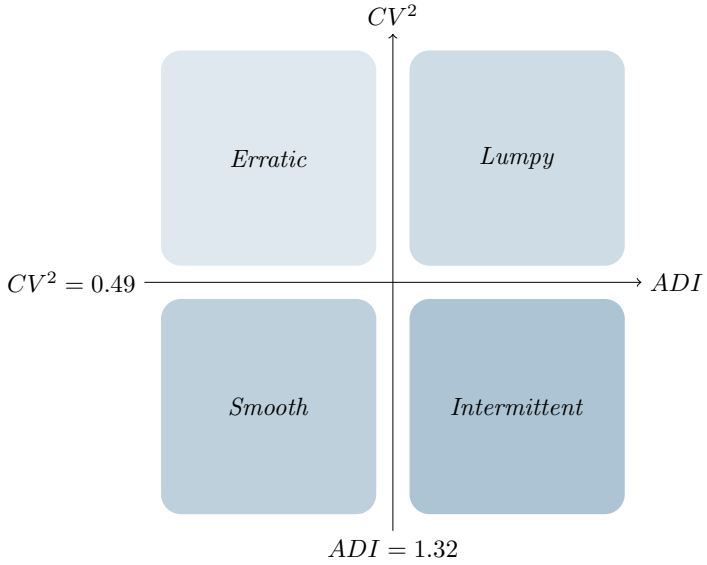


Figure 3.4: Demand patterns categorization as shown in Syntetos et al. The cutoff values for ADI and CV^2 had been derived by comparison between the MSE achieved by the authors' forecast and the one obtained by the Croston's method.

This categorization however is also interesting for an additional split, regarding the expected

variability either in time or in quantity or both - Fig. 3.5 -:

Erratic Occurs for ADI < 1.32 and $CV^2 \geq 0.49$ and translates to a regularity in time but a great variability in the observed values. For instance a weekly selling product for which the demand goes from 7 to 700 from one week to the next.

Lumpy This is the most delicate pattern to forecast. Presenting a wildly varying positive demand at irregular time steps, the forecasting accuracy is fated to degrade. This is one of the circumstances in which managing uncertainty plays a crucial role. In terms of categorization parameters, a demand is categorized as lumpy for ADI ≥ 1.32 and $CV^2 \geq 0.49$.

Smooth The best case possible, which happens for ADI < 1.32 and $CV^2 < 0.49$. The demand time series is rich, with no irregularities.

Intermittent The demanded quantity does not undergo through massive changes, however the appearance in time is irregular. A time series falls in this category if its ADI ≥ 1.32 and $CV^2 < 0.49$.

The year after, Kostenko et al. [KH06] reviewed the Syntetos et al. proposition and updated the cut-off values to ADI = $4/3 = 1.\bar{3}$ and $CV^2 = 0.5$.

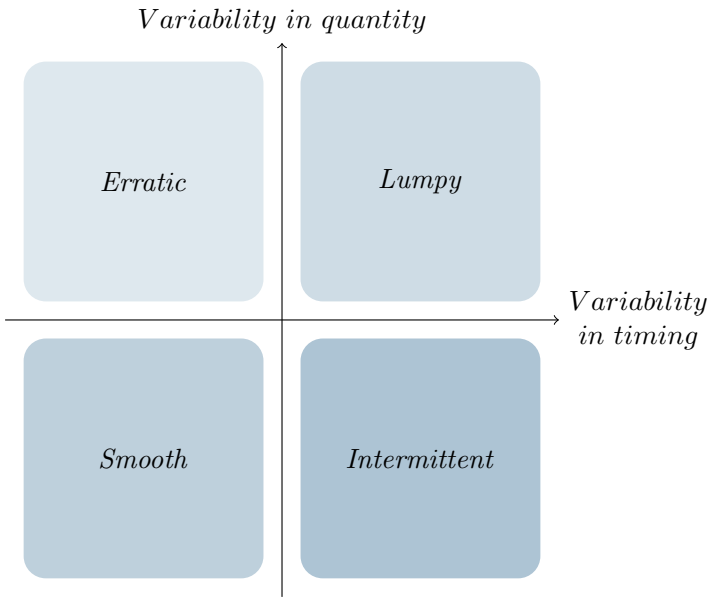


Figure 3.5: Demand patterns categorization and their implication on the expected variability in time and quantity.

3.1.4 Bullwhip effect

Also known as *whipsaw effect* or *whiplash*, it is a symptom of coordination failure in traditional and modern Supply Chain. It refers to the amplification of small fluctuations - positive or negative - over the chain, starting at retailer level up to supplier stage. It was firstly observed by *P*rocter & *G*amble's (P&G) executives, analysing demand data for baby's diapers: even though the demand placed at retailers was quite stable over time, the orders placed by retailers to distributors were fairly variable; the orders placed by distributors to P&G itself were higher than those they had received, finally the demand placed by P&G to its own suppliers was even more volatile. The effect has been linked by researchers to an irrational behaviour by managers overreacting to a certain event [Ste89] - e.g. small shortage on one week - ordering more than needed. Over the years other factors added to the list, including demand information misuse and forecast demand inaccuracies.

Even though ordering more than needed is a problem which affects inventory costs and service levels, ordering less than needed can be even more detrimental for businesses. Suppose to strictly endorse the forecasting suggestion; forecasting a lower demand leads to ordering a lower quantity to cover the future needs. However the actual demand stands above that foreseen, the available stocks can not cope with it and we enter a stock-out situation. Stock-outs censor the potential demand - that most probably will be place to competitors by customers - and at the next iteration the forecasting engine could evaluate a even lower demand, starting an inward spiral which could lead to great income loss if not recognized.

3.1.5 Makridakis competitions

Makridakis Competitions have been started by Prof. Spyros Makridakis in 1982, in response to some hostility and charge of incompetence emerged in the occasion of a precedent competition. Five years before, in 1979, Makridakis et al. [MHM79] presented to the Royal Statistical Society a study involving 111 time series and more than 20 forecasting methods. At the time forecasting "competitions" were carried out by single research group or individuals comparing several forecasting methods, since it was not feasible to conduct large-scale forecasting competitions as we are used to do today. The results presented, driven by data, caused quite a stir as in contrast with the mindset of the time. The common belief was that a single model - e.g. ARIMA, Box-Jenkins, etc. - could describe the data and the work of a forecaster was to uncover it. Therefore in the first Makridakis Competition - abbreviated in M Competitions - anyone was invited to submit a solution while using a common dataset made of 1001 time series. The intent was to evaluate and compare the accuracy of different forecasting methods and models, removing the possible forecaster's incompetence excuse. Over the years these competitions gain attention from both Academia and practitioners pro-

viding to the attendants a growing number of time series, harvested from different fields and companies, and greeting an increasing set of different methods and models.

The first competition [MAC+82] counted 1001 time series taken from demography, industry and economics, and ranged in length between 9 and 132 observations. All the data were either non-seasonal - e.g. annual - quarterly or monthly. More than 24 submissions, between methods and models, had been investigated, nine of which were variations - e.g. to include seasonal effects - of another one. A common example is the ARARMA [Par82] model. The fundamental conclusions drawn after the competition's end were: combinations of various models and methods outperformed, on average, the individual methods being combined and did well compared to others; the ranking of the methods varied according to the accuracy metric being used; the length of the forecasting horizon had a direct impact on the methods' performance; statistically sophisticated or complex models - like those belonging to the AR(I)MA family - didn't perform systematically better than simpler methods or models. The latter statement confirmed the findings of the previous challenge and got got criticised again for not being enough aligned with real business scenarios. Remarkably, the best performing method was a "DSES" a method employing a classical multiplicative decomposition followed by a simple exponential smoothing to forecast the seasonal adjusted data together with a naïve method to forecast the seasonal component.

To address such criticisms the organisers cooperated with firms to gather real data. The M2 Competition [MCH+93] counted only 29 time series - but with a much richer context information - and ran in real-time and gave to the practitioners the chance to manually incorporate information and insights in a post-hoc way. Nevertheless this additional degree of freedom didn't change the conclusions of the first competition and the best performing forecast method was a SES with damping.

The third competition in the series [MH00], taking place in 1998, saw the number of time series jumping to 3003. Data had been collected from a wider selection of fields - e.g. finance and demographic - with lengths ranging from 14 to 126 observations. Again time series were either non-seasonal - e.g. annual -, quarterly or monthly. Practitioners from the emerging (for the time) Neural Networks field had been asked to partecipate, but the call didn't receive a swift response. The competitions upheld the findings from the previous two challenges even though ARIMA and Box-Jenkins models could be listed in the top performing entries, in contrast with their results in the preceding competiitons. Another top performer in the competition was the commercial forecasting software package ForecastPro, which most probably used a state space approximation to select between a simple exponential smoothing and an ARIMA model, based on the BIC calculation. The winner of the competition was the so called *Theta model*.

Theta model [AN00] The basic idea was to double differentiate the input time series to create a surrogate one. The local curvature of the surrogate time series is modified by a “Theta-parameter" - simply θ -; different values of θ correspond to a different deflated or delated surrogate time series that are named “Theta-lines". The Theta-lines are extrapolated independently and then recombined to produce the forecast. The authors used a linear combination, but different combinations can also be exploited. In 2003 Hyndman et Billah [HB03] demonstrated how this method is equivalent to a SES with drift, where the drift parameter can be initialized as half the slope of a linear regression fitted to the data.

The M4 Competition [MSA18a], after almost 20 years from its predecessor, the second to last competition in the sequence started. Having to cover so many years and relative advances, both in data availability and models at hand, the competition had to scale with reference to the previous ones. The dataset was made of 100’000 time series, coming from the usual fields - like finance, demographic, etc. - but weekly daily and hourly data were included along with annual quarterly and monthly ones. In contrast with the four previous challenges, the partecipants had been invited to submit prediction intervals as well as point forecast. Furthermore the solution should had been open-sourced - for reproducibility purposes - and posted on GitHub [2]. The number for entries based on Machine Learning techniques was stunning, out of 64 methods tested in total, 49 could be classified as belonging to the ML realm.

ESS-RNN [Smy20] Winner of the M4 competition with a solid margin, it was submitted by Slawek Smyl - right then working full time on time series forecasting at Uber - and it was illustrative of an “hybrid" approach; in particular it was the mixture of a ES method with a RNN. Specifically the author used a multiplicative Holt-Winter model where the equations for level and seasonality where SES formula, while the LSTM was delegated to the identification of a non-linear trend.

The findings of the first three competitions were partially overturned by the M4. Combinations of methods/models were still a good way to improve the forecasting accuracy but there was not anymore a clear advantage of simple methods/models over more complex ones. Even if some of the NN methods or pure ML approaches submitted were not able to beat the benchmarks, the solution proposed by Smyl highlighted the potential of hybrid methods.

[2] GitHub: https://github.com

3.1.6 Metrics

The metrics presented in Sec. 2.2.2 could virtually be employed also in the demand forecast case, but we must pay attention. As Hyndman [H+06] highlighted, for example, some of the traditional metrics are not tailored to demand data - especially for intermittent demand - since they can give infinite or undefined values.
If the subtended predictive distribution is - or is assumed to be - symmetric, than choosing either MAE or RMSE will lead to the same point forecast since the median and the mean of the data are equal.

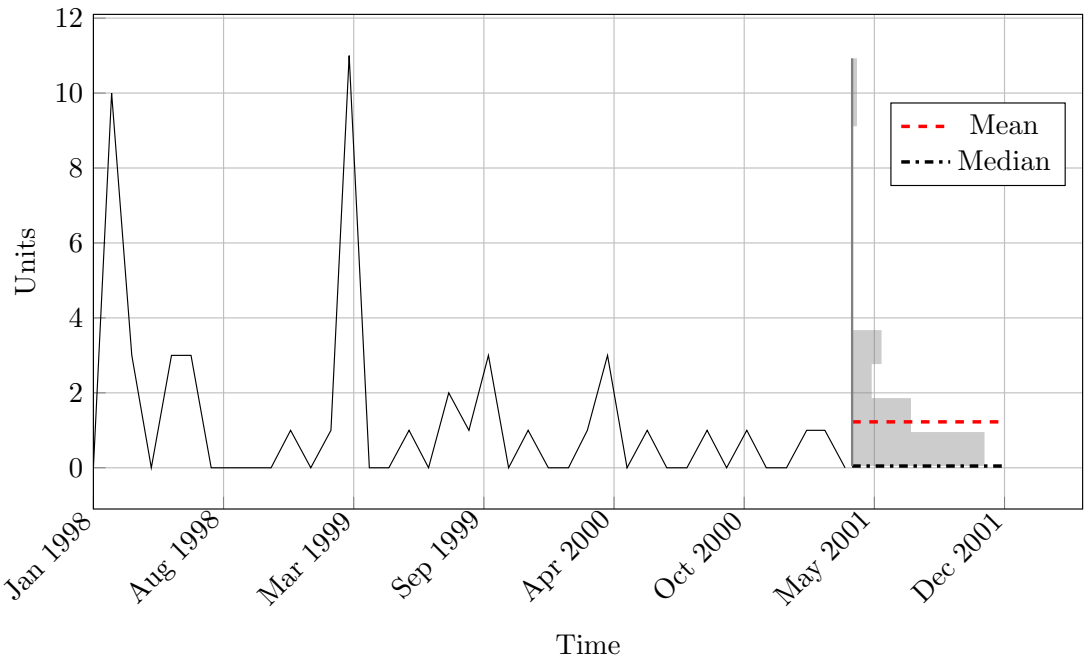


Figure 3.6: Plot of the first 39 months of the low-level sparse demand presented in Fig. 1.4, together with the relative histogram. Employing RMSE or MAE will lead the optimization process to converge towards the *Mean* line and the*Median* line respectively. The forecast generated by a model optimized via MAE using data resembling the displayed one, will therefore be inconsistent with a real business scenario resulting always in zero.

However if the predictive distribution is asymmetric, the RMSE and the MAE will lead the optimization towards two distinct targets. Taking the low-level intermittent demand presented in Fig. 1.4 as an example and plotting the relative histogram in Fig. 3.6, the RMSE will lead the optimization towards $\hat{\mu} = 1.3$ - which is in the neighbourhood of true mean $\mu = 1$ - while MAE will point to the median $\tilde{\mu} = 0$. Having a forecast biased towards zero has no concrete business utility and brought to discard the use of MAE in practice for this kind of tasks.

Scale-independent metrics are available, for example those belonging to the percentage error family - e.g. the MAPE (Sec. 2.3.3) - but these latter make sense only in those contexts where divisions and ratios have a meaning. Percentage errors have the disadvantage to be both infinite or undefined for any observation equal to zero. To cite an instance MAPE had been used during the M3 competition but only including strictly positive data in the competition dataset. Additionally they have an extremely skewed distribution when the observation is close to zero. Finally percentage errors put a heavier penalty on positive errors than negative ones, circumstance which could lead to the bullwhip effect (Sec. 3.1.4). The latter observation preceded the introduction of "symmetric" errors like the *s*ymmetric *MAPE* (sMAPE) [3]. Nevertheless other disadvantages typical of the MAPE remain.

A possible alternative are scaled-errors metrics in which the error realized by a forecasting method is related to that of a benchmark one. An example is the *M*ean *A*bsolute *S*caled *E*rror (MASE), proposed by Hyndman et al. [HK06] and used in the occasion of the M4 competition. Usually the benchmark method is the Naïve one introduced in Sec. 2.3.1.

Mean Absolute Scaled Error (MASE)

The idea is to scale the error based on the *in-sample* MAE from the naïve forecast (see Sec. 2.3.1). For a single time series $\mathbf{z}_{t_0:\mathrm{T}_i}$ and related prediction $\hat{\mathbf{z}}_{t_0:\mathrm{T}_i}$, MASE is defined as
All the metrics discussed so far could be applied to the probabilistic forecasting problem only if we restrict our selves to either the mean or the median of the process. However, the need to extrapolate uncertainty information is what guide users' attention to this kind of problem. We need another set of metrics.

Pinball Loss

Well known in the context of Quantile Regression - where it is also referenced as *Quantile Loss* -, it is an asymmetrical loss function defined as

$$\mathrm{L}_\rho\left(z,\hat{z}\right)=\begin{cases}\rho\left(z-\hat{z}\right) & z\geq\hat{z}\\ \left(1-\rho\right)\left(\hat{z}-z\right) & z<\hat{z}\end{cases}\qquad \rho\in(0,1)$$

or compactly

$$\mathrm{L}_\rho\left(z,\hat{z}\right)=\mathrm{I}_{z\geq\hat{z}}\rho|z-\hat{z}|+\mathrm{I}_{z<\hat{z}}\left(1-\rho\right)|z-\hat{z}|$$

[3] $sMAPE\left(\mathbf{z}_{1:\mathrm{T}},\hat{\mathbf{z}}_{1:\mathrm{T}}\right)=\frac{200}{\mathrm{T}-1}\sum_{t=1}^{\mathrm{T}}\frac{|z_t-\hat{z}_t|}{(|z_t|+|\hat{z}_t|)}$

where ρ is the target quantile and $I_*(\cdot)$ is an indicator function telling us if the prediction is either equal or greater than the observation or the other way round. Generalizing the metric to a vector case

$$L_\rho\left(\mathbf{z}^i_{1:T_i}, \hat{\mathbf{z}}^i_{1:T_i}\right) = \sum_{\substack{t=1;\\ I_{z^i_t \geq \hat{z}^i_t}}}^{T_i} \rho|z^i_t - \hat{z}^i_t| + \sum_{\substack{t=1;\\ I_{z^i_t < \hat{z}^i_t}}}^{T_i} (1-\rho)|z^i_t - \hat{z}^i_t| \tag{3.10}$$

Playing with the ρ value, it is possible to assign different penalties to the under-forecasting and the over-forecasting cases, property really appreciated for the reasons explained in the paragraph about Bullwhip effect.
When the target quantile is $\rho = 0.5$, the Pinball Loss is said to be “equal" to the MAE; indeed Eq. (3.10) reduces to

$$\begin{aligned} L_{0.5}\left(\mathbf{z}^i_{1:T_i}, \hat{\mathbf{z}}^i_{1:T_i}\right) &= \frac{1}{2}\sum_{\substack{t=1;\\ I_{z^i_t \geq \hat{z}^i_t}}}^{T_i} |z^i_t - \hat{z}^i_t| + \frac{1}{2}\sum_{\substack{t=1;\\ I_{z^i_t < \hat{z}^i_t}}}^{T_i} |z^i_t - \hat{z}^i_t| \\ &= \frac{1}{2}\sum_{t}^{T_i} |z^i_t - \hat{z}^i_t| \end{aligned}$$

which differs from MAE only by a constant factor, ineffective in terms of optimization.

ρ-risk

It is a modified version of the Pinball Loss, introduced by Seeger et.al [SSF16] and proposed again in [SFGJ19, RSG+18]. Over the time range $\{1,\dots,t,\dots,T_i\}$ we compute the quantity $Z^i = \sum \mathbf{z}^i_{1:T_i} = \sum_{t=1}^{T_i} z^i_t$, i.e. the sum of the observations. For any given quantile $\rho \in (0,1)$ the corresponding sum is defined as $\hat{Z}^{\rho,i} = \sum \mathbf{z}^{\rho,i}_{1:T_i} = \sum_{t=1}^{T_i} z^{\rho,i}_t$. The predicted values $\mathbf{z}^{\rho,i}_{1:T_i}$ can be acquired through quantile regression, Monte Carlo simulation or any other sampling method. With Z^i and $\hat{Z}^{\rho,i}$ so estimated, we can compute $L_\rho(Z^i, \hat{Z}^i)$, following the definition given in Eq. (3.10). Finally we will use the latter score to measure the ρ-*risk*

$$QL_\rho = \frac{L_\rho\left(Z^i, \hat{Z}^i\right)}{Z^i}. \tag{3.11}$$

The definition of this metric can sound redundant once the pinball loss had been introduced. Nonetheless, as pointed out by Salinas et al. [SFGJ19], it is needed to scale the pinball loss when the magnitudes of the time series involved differ widely, i.e. the time series are either erratic or lumpy. Dividing by the sum of the observations the error is kept in the same range. We could say that the relationship between the pinball loss and the ρ-risk is the same as that between MAPE and the *W*eighted *M*ean *A*bsolute *P*ercentage *E*rror (wMAPE). The

weighted MAPE had been devised to overcome the infinity or undefined issues of its ancestor. Being computed over the whole time series rather than single data point, the metric can go to infinity only if the whole time series is zero.

$$w\text{MAPE}\left(\mathbf{z}^i_{1:T}, \hat{\mathbf{z}}^i_{1:T}\right) = \sum_{t=1}^{T} \frac{\left|z^i_t - \hat{z}^i_t\right|}{z^i_t} \tag{3.12}$$

If observations are only positive values we can drop the absolute operator, landing to a definition that can be mapped to that of the ρ-risk.

3.2 Datasets

3.2.1 The Part dataset

Presented in [HKOS08] the dataset comprises 2674 aligned monthly series of slow moving parts supplied by a US car company. Covering a period of 51 months - a bit more than four years, from January 1998 to March 2002 - the majority of the time series, 89% of them specifically, are over-dispersed with an average dispersion ratio of 2.3. Only 2059 time series possess a complete history - i.e. no missing data - with an average gap between positive demands of 2.9 months. To avoid any computational problem with time series containing too few strictly positive observations, the number of time series had been culled from 2059 to 1046 keeping only those time series with

- at least 10 months with positive demand;
- at least some positive demands in the first 15 and in the last 15 months.

The culling process had been described in Snyder et al. [SOB12]. Figure 3.7 proposes again the data shown in Fig. 1.4 - which had been sampled from this dataset - together with the relative histogram and the aggregated demand.
A clear descendent trend is visible in the aggregated data but, as is already evident in the same figure, it is not always discernible if and how such a trend operates at the single SKU level.
Following Syntetos et al. categorization the parts dataset can be summarised as displayed in Tab. 3.1.

The dataset had been used by Snyder [SOB12] - as a benchmark for all the methods tested in the work - and by Salinas et al. [SFGJ19] and Rangapuram et al. [RSG+18] to compare against Snyder's work. To be aligned with Rangapuram et al. [RSG+18], the cut-off value to

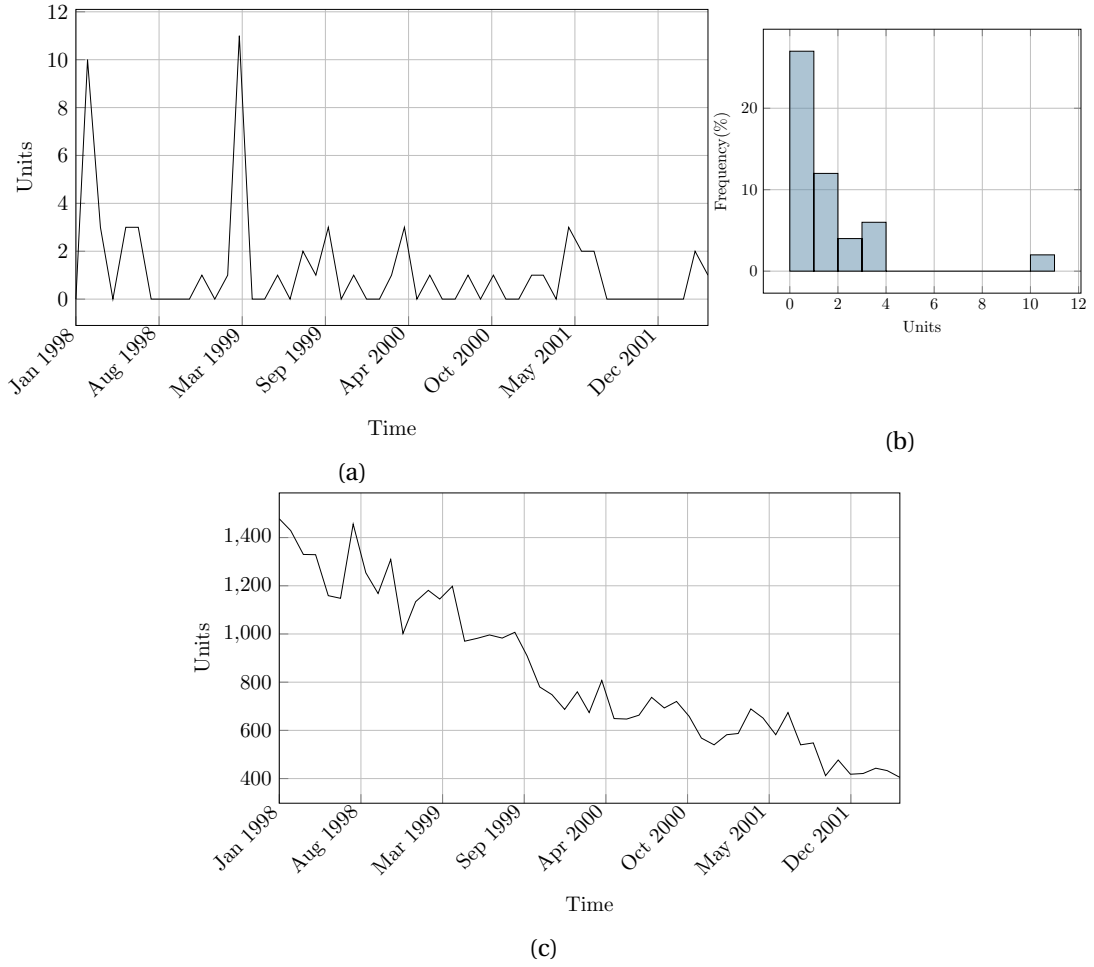


Figure 3.7: (a) Time series, already presented in Fig. 1.4, identified with #10055165 in the original dataset. The relative histogram is shown in (b). (c) The aggregated demand over the whole dataset.

Table 3.1: Parts' categorization per Syntetos et al.

Erratic	Lumpy	Smooth	Intermittent
$\leq 0.2\%$	$\approx 20\%$	$\leq 0.1\%$	$\approx 80\%$

split the dataset into training and testing set had been appointed at $T_i = 39$. The training set contains the first 39 months while the last year is used as prediction range, i.e. $T_{i+h} = 51$ with $h = 12$. The original dataset is released under a GPL-3 license and is available in its raw format at [HKOS].

3.2.2 The M4 competition dataset

Released for the fourth edition of the M Competitions, it is very heterogeneous dataset. One hundred thousand time series, representative of six different domains, had been randomly selected from the *ForeDeCk*[4] database. Differently from previous competitions, hourly daily and weekly time series had been included along with monthly quarterly and annual ones. The summary is shown in Tab. 3.2.

For each frequency a minimum number of observations had been ensured, specifically: 700 for hourly, 93 for daily, 80 for weekly, 42 for monthly, 16 for quarterly and 13 for yearly series. In addition, to guarantee error-free metric calculations, the organizers scaled each and every time series to prevent negative observations and values lower than 10. The scaling was performed adding a constant to the series so that their minimum value was equal to $c = 10$, precisely. The addition of a constant c to data with a shifting purpose is a functional artifice, used as habitually in competition or research contexts as avoided in enterprise' practices. The main reason is the possibility to either hide or erase some features of the data, for instance a shift in the original values could elicit a variation of the beneath distribution - relocating the peak previously on zero - and modify the probabilities of the values within the range $[0, c]$.

Individually the different frequencies had been provided as standalone datasets - already subdivided into training and testing - and could be treated accordingly, requesting a specific horizon h. The time series had been anonymized and any information that could lead to their identification had been omitted, including the starting dates of the series (which did not become available to the participants until the M4 had ended). The dataset is available in its entirety at the competition repository [Mak18].

In contrast with the *parts* dataset, the T_i value can vary from series to series as well as T_{i+h}. To be aligned with Rangapuram et al. [RSG+18] and Oreshkin et al. [ODPT21], we target only the *hourly* dataset for the current work. The subset is composed by 414 time series, all belonging to the domain *Other*, with a length ranging from 700 to 960 data points. The request horizon was $h = 48$ hours. Table 3.3 reports the hourly dataset categorization following Syntetos et al. While the percentages of lumpy and intermittent time series don't change too much, there is a difference of exactly 8.22 percent between those for erratic and smooth

[4] No reference had been found for the dataset.

Table 3.2: M4 dataset composition: the number of time series divided by their frequency and domain.

	Domain						
Freq./h	Demogr.	Fin.	Ind.	Macro	Micro	Oth.	Total
Hourly/48	0	0	0	0	0	414	414
Daily/14	10	1'559	422	127	1'476	633	4'227
Weekly/13	24	164	6	41	112	12	359
Monthly/18	5'728	1'0987	10'017	10'016	10'975	277	48'000
Quarterly/8	1'858	5'305	4'637	5'315	6'020	865	24'000
Yearly/6	1'088	6'519	3'716	3'903	6'538	1'236	23'000
Total	8'708	24'534	18'798	19'402	25'121	3'437	100'000

data. We can imagine that the scaling process moved some erratic and lumpy time series - changing their CV^2 and consequently their variability in quantity (Fig. 3.4 and Fig. 3.5) - to the smooth and intermittent basket respectively.

Table 3.3: M4 Hourly categorization, per frequency, along Syntetos et al. "In Competition" points to the categorization ran on the time series as given. "Re-scaled" refers to the time series purged from the added constant $c = 10$. The last line in the table reports the categorization as found in Oreshkin et al. [ODPT21]

	Erratic	Lumpy	Smooth	Intermittent
In Competition	6.76%	10.63%	52.42%	30.19%
Re-scaled	14.98%	10.87%	44.20%	29.95%
Oreshkin et al.	17%	--	83%	--

About sixty percent of the M4 time series are characterized by a rich history - compared to those in the parts dataset - emphasized by a low variation in the time steps, i.e. a lower average inter-demand interval.

3.3 State-of-the-Art

While in Sec. 2.3 we already discussed the standard methods, the corresponding section regarding the neural architectures didn't introduce any specific model, preferring the introduction of some of the essential architectures. Those architecture are the foundation for the state-of-the-art models introduced in the following. Three main works had been chosen as benchmark for this study: DeepAR [SFGJ19],
DeepSSM [RSG+18] and N-Beats [ODPT21]. The latter is considered to be the SotA model

thanks to its performance on the M4 dataset and the claimed interpretability. The first two instead were the previous SotA models, both coming from Amazon Lab's groups. They are interesting because, contrary to N-Beats, they are commercially available and hence virtually usable in a production environment. Moreover, while N-Beats had been presented as a general solution for time series forecasting, DeepAR and DeepSSM persist specifically on the demand forecasting task (even though they can be used for time series forecasting at large) with a probabilistic perspective.

3.3.1 Deep auto-regressive recurrent networks

Long Short-Term Memory Networks (Fig. 2.7) had been designed to handle local temporal data and are extensively used in architectures for time series forecasting models. In general they are trained at single time series level and are asked to produce predictions directly. Their usage in the DeepAR architecture is a bit more peculiar.
The authors aim at creating a global model, trained over related time series to handle quantities' variability. Additionally, being interested into the data' statistical properties, they set a probabilistic forecasting problem. Therefore LSTMs are requested to output a latent representation of the distribution's parameters, rather than a prediction. The architecture is assembled into an Encoder-Decoder network where the same LSTM plays both roles. The LSTM, arranged in a many-to-many style, is fed with multiple time series at each step. The latent representation is then passed through a fixed number of dense layers - as numerous as the target distribution's parameters - to be mapped to the chosen distribution's parameters.

Training The left side of Fig. 3.8 illustrates the training process. Given the i-th series and a conditioning range $[1,\dots,t,\dots,t_0-1]$, at each time step t the network is fed with: a) the previous network state $\mathbf{h}^i_{t-1}$; b) the target value at the previous time step z^i_{t-1}; c) the (optional) co-variate x^i_t. Transforming these inputs, the latent representation θ^i_t is produced. How the latent representation is then mapped to distribution's parameters depend on the hypothesis made:

> *Gaussian (Normal) distribution* ordinary choice for real-valued data, especially in conjunction with state space models. The researchers had chosen the conventional Gaussian parameterisation, i.e. the distribution is fully described by its mean μ and standard deviation σ, such that $\theta = (\mu, \sigma)$.

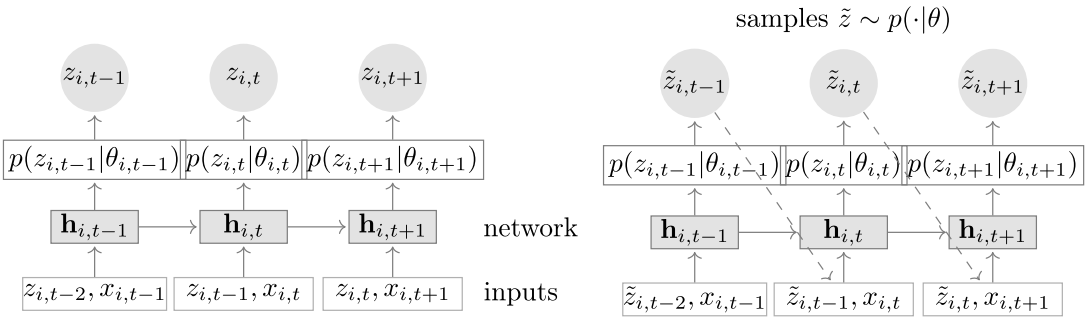


Figure 3.8: Summary of the DeepAR model as reported in [SFGJ19]. (Left) Training: at each time step t, the network is fed with the co-variates $x_{i,t}$, the target value at the previous time step $z_{i,t-1}$, and the previous network output $\mathbf{h}_{i,t-1}$. The network output $\mathbf{h}_{i,t} = h\left(\mathbf{h}_{i,t-1}, z_{i,t-1}, x_{i,t}, \Theta\right)$ is then used to compute the parameters $\theta_{i,t} = \theta\left(\mathbf{h}_{i,t}, \Theta\right)$ of the likelihood $p(\cdot \mid \cdot)$. (Right) Prediction: starting from $t \geq t_0$ a sample $\hat{z}_{i,t} \sim p\left(\cdot \mid \theta_{i,t}\right)$ is drawn and fed back for the next point until the end of the prediction range $t = t_0 + \mathrm{T}$, generating one sample trace. Repeating this prediction process yields many traces that represent the joint predicted distribution.

$$\mu = \mathbf{w}_\mu^{\mathrm{T}} \mathbf{h}_{i,t} + b_\mu \tag{3.13}$$

$$\sigma = \log\left[1 + \exp\left(\mathbf{w}_\sigma^{\mathrm{T}} \mathbf{h}_{i,t} + b_\sigma\right)\right] \tag{3.14}$$

The mean is given by an affine function of the network output - Eq. (3.13) -, at the same time the standard deviation is obtained by applying an affine transformation followed by a softplus activation in order to ensure $\sigma > 0$ - Eq. (3.14) -

Negative Binomial distribution already outlined in Sec. 3.1.2, it is a standard choice for positive count data. The distribution had been parameterised via its mean μ and shape parameters δ - i.e. $\theta == (\mu, \delta)$ -

$$\mu = \log\left[1 + \exp\left(\mathbf{w}_\mu^{\mathrm{T}} \mathbf{h}_{i,t} + b_\mu\right)\right] \tag{3.15}$$

$$\delta = \log\left[1 + \exp\left(\mathbf{w}_\delta^{\mathrm{T}} \mathbf{h}_{i,t} + b_\delta\right)\right] \tag{3.16}$$

Both the mean and the shape parameters are given by an affine transformation followed by a softplus activation of the network output - Eq. (3.15) and Eq. (3.16) respectively -, to ensure that μ and δ are greater than zero.

Distribution's parameter θ and those belonging to the LSTM - namely $\mathbf{w}_*$ and b_* - are gath-

ered together into the parameters set Θ and learnt maximizing the log-likelihood of the selected distribution. We already introduced the Negative Binomial's log-likelihood in Eq. (3.7), in the following the Gaussian log-likelihood is presented together with its pdf (Sec. 3.1.2) :

$$f\left(\mu,\sigma|z\right) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2\right] \tag{3.17}$$

$$\ell_{Gaussian} = -\mathrm{T}\ln\left(\sigma\right) - \frac{\mathrm{T}}{2}\ln\left(2\pi\right) - \sum_{j=1}^{\mathrm{T}}\frac{\left(z_j-\mu\right)^2}{2\sigma^2}. \tag{3.18}$$

Quoting the authors it is worth mentioning that other distributions can also be used readily, e.g. a beta likelihood for data in the unit interval, a Bernoulli likelihood for binary data, or mixtures in order to handle complex marginal distributions, as long as samples from the distribution can be obtained cheaply, and the log-likelihood and its gradients with respect to the parameters can be evaluated.

Prediction Figure 3.8, right side, demonstrates the prediction procedure and it is quite straightforward. To output a prediction $\hat{z}_t^i$ over the horizon $\{t_0,\ldots,t,\ldots,\mathrm{T}\}$, the network makes use of the learnt distribution's parameter, drawing a sample $\hat{z}_t^i$ ($\tilde{z}_{i,t}$ in the figure) from the estimated prediction. The same sample is fed back to the network for the next step and the process repeats untile the prediction range has been exhausted.
Repeating the prediction a fixed number of time, yields to the generation of several traces which approximate the predicted distribution. From this set of traces it is possible to extract quantiles of interest, the median - $\rho = 0.5$ - and $\rho = 0.9$ specifically (also called 0.5-risk and 0.9-risk in the paper).

3.3.2 Deep state space models

About one year after the first publication on arXiv [5] of the DeepAR paper, another reasearch group affiliated with the Amazon Research Lab presented DeepSSMs
[RSG+18] to the 32nd Conference on *Neur*al *I*nformation *P*rocessing *S*ystems
(NeurIPS 2018), at Montréal. The central objective is the same that guided the DeepAR group: construct a global model which can learn jointly from several time series. The steps undertaken to achieve the goal diverges. The DeepSSM architecture is a bit more complicated than that presented with DeepAR, so let's proceed step by step.

The main global dispatcher role is always entrusted to a LSTM. It is parameterised via the set

[5] arXiv: https://arxiv.org/

Φ which are jointly optimized over all the data, as it was also the case for DeepAR. As done previously we identify with $\mathbf{h}_t^i$ the network state and with $\mathbf{h}_t^i = h\left(\mathbf{h}_{t-1}^i, x_t^i, \Phi\right)$ its output, with $\mathbf{x}_{1:t}^i$ series-wise co-variates up to time t.

At single $\mathbf{z}^i$ time series level, a linear state space model is applied. As briefly described in Sec. 2.3.4 two steps are required while training a SSM: a transition step - or state equation - and an observation one - also called measurement equation -. A linear innovation SSM can be defined as

$$\begin{cases} \mathbf{l}_t = \mathbf{F}_t \mathbf{l}_{t-1} + \mathbf{g}_t^i \epsilon & \text{(transition)} \\ \\ y_t^i = \mathbf{a}_t^{i,\mathrm{T}} \mathbf{l}_{t-1}^i + b_t^i \\ z_t^i = y_t^i + \sigma_t^i \epsilon & \text{(observation)} \end{cases} \tag{3.19}$$

$\mathbf{l}_t^i \in \mathbb{R}^{\mathrm{L}}$ identifies the L-dimensional state of the model and $\epsilon \sim \mathcal{N}(0,1)$ is the Gaussian error model which will dictate its likelihood. The initial state $\mathbf{l}_0^i \sim \mathcal{N}\left(\mu_0^i, \Sigma_0^i\right) = \mathcal{N}\left(\mu_0^i, \mathrm{diag}\left(\sigma_0^{i,2}\right)\right)$ is assumed to follow an isotropic Gaussian distribution. The transition equation is described by the *transition matrix* $\mathbf{F}_t^i$ and a random *innovation* $\mathbf{g}_t^i \epsilon$. Similarly the observation equation is defined via the *observation matrix* $\mathbf{a}_t^i$ - even though in this case it is just a vector -, the bias b_t^i and the noise $\sigma_t^i \epsilon$. The transition's noise part is called innovation due to its time dependency, in fact it could have been written as $\mathcal{N}\left(0, \mathbf{g}_t^i\right)$. The same is true for the observation's noise, i.e. $\mathcal{N}\left(0, \sigma_t^i\right)$. All together, matrices, bias and random noises form the linear SSM's parameters group $\Theta_t^i = \left(\mu_0^i, \Sigma_0^i, \mathbf{F}_t^i, \mathbf{a}_t^i, b_t^i, \mathbf{g}_t^i, \sigma_t^i\right)$. Nevertheless this collection of parameters is not the learning target, they have to be deduced from the global shared parameters Φ instead. The extrapolation is supervised by the mapping function Ψ such that, at each time step t

$$\Theta_t^i = \Psi\left(\mathbf{x}_{1:t}^i, \Phi\right) \tag{3.20}$$

The introduction of the linear model, its set of parameters and the mapping function $\Psi(\cdot)$ marks the difference between the current architecture and the DeepAR's one. If in the latter each and every time series received the same collection of parameters - i.e. was governed by the same stochastic process (the same distribution) - in the DeepSSM's conception time series are treated individually, deducing the corresponding linear SSM's parameters from those shared by the global model, i.e. Φ, via $\Psi(\cdot)$. In other words in the former the LSTM will directly output the distribution parameters, in the latter it will return proxies to a sub-model parameters.
The mapping function $\Psi(\cdot)$ itself is a collection of affine mappings, followed by element-

wise transformations restricting the parameters to the appropriate range. Specifically, once the $\mathbf{h}_t^i$ is available, each SSM parameter $\theta_t^i \in \Theta_t^i$ goes under an affine transformation

$$\tilde{\theta}_t^i = \mathbf{w}_\theta^{\mathrm{T}} \mathbf{h}_t^i + b_\theta$$

to then be fed to different constraints, depending on the target parameter domain:

$\theta_t^i \in \mathbb{R}$, e.g. b_t, undergoes no transformation;

$\theta_t^i > 0$ goes through a softplus, i.e.

$$\theta_t^i = \log\left[1 + \exp\left(\tilde{\theta}_t^i\right)\right] ;$$

$\theta_t^i \in [a, b]$ experiences a scaled and shifted sigmoid, i.e.

$$\theta_t^i = (b-a)\,\frac{1}{1+\exp\left(-\tilde{\theta}_t^i\right)} .$$

This is not really dissimilar to Eq. (3.13) – (3.16).

Training Consistently with DeepAR, the log-likelihood is maximized to learn the parameters Φ too. Contrary to DeepAR, it is not so straightforward. The likelihood now has a dependecy from the linear SSM's state, which has to be marginalized out. Precisely, given a dataset $\mathbb{D} = \left\{\mathbf{z}_{1:\mathrm{T}_i}^i\right\}_{i=1}^{\mathrm{N}}$

$$\begin{aligned}\ell\left(\Phi|\mathbb{D}\right) &= \sum_{i=1}^{\mathrm{N}} \ln\left[f\left(\mathbf{z}_{1:\mathrm{T}_i}^i|\mathbf{x}_{1:\mathrm{T}_i}^i, \Phi\right)\right] \\ &= \sum_{i=1}^{\mathrm{N}} \ln\left[f_{\mathrm{SSM}}\left(\mathbf{z}_{1:\mathrm{T}_i}^i|\Theta_{1:\mathrm{T}_i}^i\right)\right] \end{aligned} \tag{3.21}$$

where $f_{\mathrm{SSM}}(\cdot)$ - $p_{\mathrm{SS}}(\cdot)$ in Fig. 3.9a - is the per linear SSM marginal likelihood

$$\begin{aligned} f_{\mathrm{SSM}}\left(\mathbf{z}_{1:\mathrm{T}_i}^i|\Theta_{1:\mathrm{T}_i}^i\right) &= f\left(z_1^i|\Theta_1^i\right)\prod_{t=2}^{\mathrm{T}_i} f\left(z_t^i|z_{1:t-1}^i, \Theta_{1:t}^i\right) \\ &= \int f\left(\mathbf{l}_0\right)\left[\prod_{t=1}^{\mathrm{T}_i} f\left(z_t^i|\mathbf{l}_t^i\right) f\left(\mathbf{l}_t^i|\mathbf{l}_{t-1}^i\right)\right] d\mathbf{l}_{0:\mathrm{T}_i}. \end{aligned}$$

This is required since the state is not deterministic and hence its distribution - i.e. all the possible values - has to be taken into account. Since the authors had set all the distribution as Gaussian, the marginalisation is "easily" computed via the Kalman filtering algorithm. In

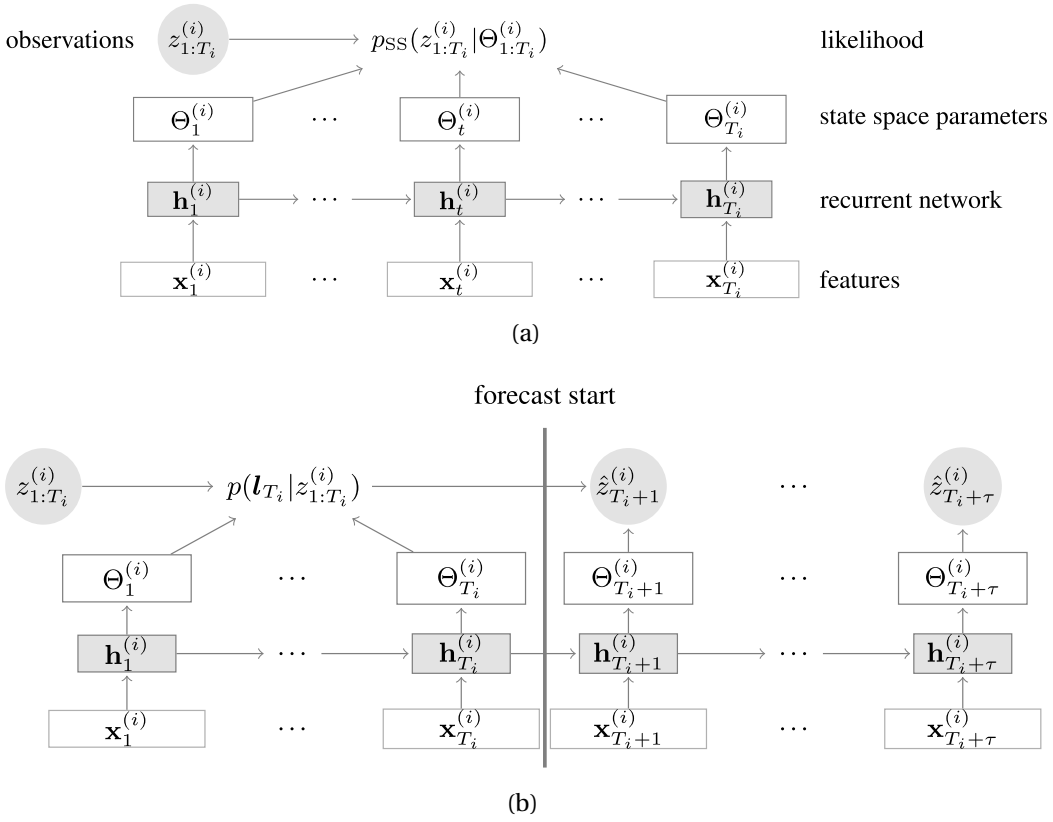


Figure 3.9: Summary of the Deep State Space Models architecture as reported in [RSG+18]. (a) Training: the network is fed with the co-variates $x_t^{(i)}$ and the previous network output $\mathbf{h}_{t-1}^{(i)}$ at each time step t in the training range $1,2,\ldots,\mathrm{T}_i$. The network output $\mathbf{h}_t^{(i)} = h\left(\mathbf{h}_{t-1}^{(i)}, x_t^{(i)}, \Phi\right)$ is then used to compute the parameters of the state space model $\Theta_t^{(i)}$ after mapping it to the corresponding ranges of the parameters. Given the time series observations $z_{1:\mathrm{T}_i}^{(i)}$, the likelihood of the state space parameters $\theta_{1:\mathrm{T}_i}^{(i)}$ (which are functions of the shared network parameters Φ) is computed. (b) Prediction: given the posterior of the latent state, prediction samples are generated by recursively applying the model equations where the state space parameters for the prediction range $\Theta_{\mathrm{T}_{i+1}:\mathrm{T}_{i+\tau}}^{(i)}$ are obtained by unrolling the RNN in the prediction range.

the original paper the authors point out that other choices than Gaussian are possible, but that will require the insertion of other neural networks - like *V*ariational *A*uto-*E*ncoders (VAEs) - to efficiently handle the marginalisation which will be too cumbersome otherwise. Except for this precaution regarding the log-likelihood, the training proceed as already described in the previous section. For each time step t the network is fed with the previous network state $\mathbf{h}_{t-1}^i$ and the co-variate x_t^i. The network output is mapped via $\Psi(\cdot)$ to the state space model parameters, which are then used to compute the marginalized likelihood.

Prediction Prediction is carried out by means of Monte Carlo simulation. Figure 3.9b depicts the procedure. The unrolling of the LSTM over the prediction range is alike that described for the DeepAR architecture, with the only difference that the sample $\hat{z}_t^i$ is not drew from a distribution but rather returned by the underneath state space model. Quantiles of interest - the median and $\rho = 0.9$ precisely (also called $p50$ and $p90$ in the paper) - are again derived to have a measure of the prediction uncertainty.

3.3.3 Neural basis expansion analysis

Among the three SotA architectures presented in this chapter this is the only one which: can be defined as a purely Deep Learning design; nonetheless it claims to be interpretable. Navigating the network from a single block outward we find: the l-th block, the s-th stack it belongs to, the collection of stacks forming the whole network. Figure 3.10 schematizes a possible network's arrangement.

The l-th block is made of a *F*ully *C*onnected (FC) network, with a fixed number l_{FC} of layers, which outputs two sets of parameters θ_l^b and θ_l^f:

$$
\begin{aligned}
\mathbf{h}_l^1 &= \text{FC}_l^1\left(\mathbf{z}_l\right) = \text{R}e\text{LU}\left(\mathbf{W}_l^{1,\text{T}}\mathbf{z}_l + \mathbf{b}_l^1\right);\\
\mathbf{h}_l^2 &= \text{FC}_l^2\left(\mathbf{h}_l^1\right);\\
&\vdots\\
\mathbf{h}_l^m &= \text{FC}_l^m\left(\mathbf{h}_l^{m-1}\right);\\
\theta_l^b &= \text{LINEAR}_l^b\left(\mathbf{h}_l^m\right); \qquad (3.22)\\
\theta_l^f &= \text{LINEAR}_l^f\left(\mathbf{h}_l^m\right). \qquad (3.23)
\end{aligned}
$$

Parameters θ_l^* are then fed to two basis expansion functions g_l^b and g_l^f.
In the architecture's generic arrangement, g_l^b and g_l^f are set to be a linear projection of the

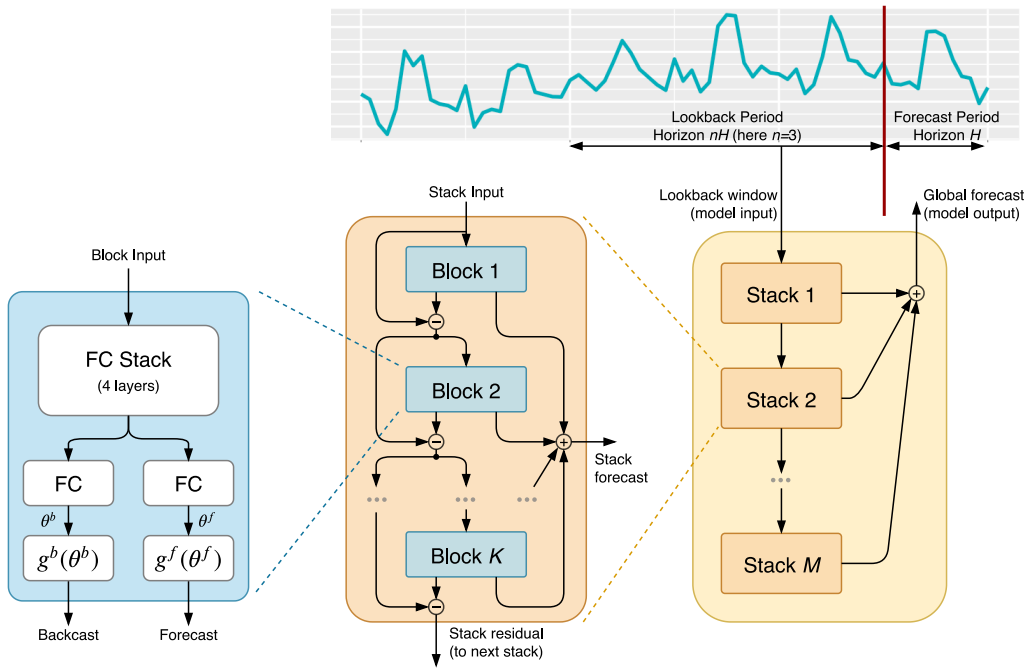


Figure 3.10: Proposed N-Beats architecture, as reported in [ODPT21]. Fundamental blocks are Fully Connected networks with ReLU non linear activation functions. Each block predicts the basis expansion coefficient both forward θ^f (forecast) and backward θ^b (backcast). Blocks are organized into stacks by means of double residual connections. Stacks can be assembled together.

previous layer output, i.e.

$$\hat{\mathbf{y}}_l^b = g_l^b\left(\theta_l^b\right) = \mathbf{W}_l^b\theta_l^b + \mathbf{b}_l^b \tag{3.24}$$

$$\hat{\mathbf{y}}_l^f = g_l^f\left(\theta_l^f\right) = \mathbf{W}_l^f\theta_l^f + \mathbf{b}_l^f \tag{3.25}$$

The output is twofold: a forward (*forecast*) signal $\hat{\mathbf{y}}_l^f$ and backward (*backcast*) portion $\hat{\mathbf{y}}_l^b$. For the very first block - i.e $l = 1$ - $\mathbf{z}_l = \mathbf{z}_t$. For the following blocks - $l = 2,\ldots,m$ - in the cascade, $\mathbf{z}_l = \mathbf{z}_{l-1} - \hat{\mathbf{y}}_{l-1}^b$. This is the first of two residual branches running through the network. Residual connection had been firstly introduced by He et al. [HZRS16]. The intuition was that it would have been easier to optimize the residual mapping than to optimize the original, unreferenced mapping. As an extreme instance, if an identity mapping was optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers. Huang et al. [HLVDMW17] extended the principle introducing extra connections from one stack to additional related stacks. The authors of the current architecture re-used the same principles. Each block removes from its input the portion of the signal which it can approximate ($\hat{\mathbf{y}}_{l-1}^b$), saving some workload to its successors.
Blocks within the same stack can share the same g_s^b and g_s^f.

The next level - middle in the Fig. 3.10 - is dedicated to stacks. Requesting an interpretable model, the shared basis expansion functions g_s^* can be structured to meet *a priori* choices:

Trend block Usual trend models are linear, logistic and polynomials. The last-mentioned option had been chosen, constraining $g_{s,l}^b$ and $g_{s,l}^f$ to be a polynomial of degree p and function of a (moving) window H

$$\hat{\mathbf{y}}_{s,l}^{\text{trend},*} = \sum_{i=0}^{p} t^i\theta_{s,l}^* = \mathbf{T}\theta_{s,l}^* \tag{3.26}$$

where $\mathbf{t} = [0,1,\ldots,\mathrm{H}-2,\mathrm{H}-1]^{\mathrm{T}}$ is the column time steps vector and
$\mathbf{T} = \left[\mathbf{1},\mathbf{t},\mathbf{t}^2,\ldots,\mathbf{t}^p\right]$ is relative matrix of powers.

Seasonality block A Fourier Series shapes the seasonal's basis functions

$$\hat{\mathbf{y}}_{s,l}^{\text{seas},*} = \sum_{i=0}^{\lfloor \mathrm{H}/2-1\rfloor} \cos\left(2\pi i\, t\right)\theta_{s,l,i}^* + \sin\left(2\pi i\, t\right)\theta_{s,l,i+\lfloor \mathrm{H}/2\rfloor}^* = \mathbf{S}\theta_{s,l}^* \tag{3.27}$$

where $\mathbf{S} = \left[\mathbf{1},\cos\left(2\pi\mathbf{t}\right),\ldots,\cos\left(2\pi\lfloor \mathrm{H}/2-1\rfloor\,\mathbf{t}\right),\sin\left(2\pi\mathbf{t}\right),\ldots,\sin\left(2\pi\lfloor \mathrm{H}/2-1\rfloor\,\mathbf{t}\right)\right]$ is the waves forming matrix and $\mathbf{t}$ is the time vector already outlined in the trend block summary.

The single stack output mimics the block's one, with backcast and forecast signals:

$$\begin{aligned} \hat{\mathbf{y}}_s^b &= \hat{\mathbf{y}}_{s,m}^b; \\ \hat{\mathbf{y}}_s^f &= \sum_{l=1}^{m} \hat{\mathbf{y}}_{s,l}^f; \end{aligned} \tag{3.28}$$

In an interpretable architecture context only two stacks are instantiated, a trend stack followed by a seasonal one.

In the end - outmost level in the reference figure -, the model output $\hat{\mathbf{y}}$ is obtained summing together all the stack-level predictions $\hat{\mathbf{y}}_s^f$.

Training & Prediction There are few facts to report about the training process of this architecture. To improve overall accuracy the researchers preferred an ensemble of models over more popular alternatives, like dropout or L2-norm penalty. The models ensemble is constructed starting from different windows of the input sequence - named *lookback windows* - and one or more losses. The window's lengths are dependent on the forecasting horizon H. Six windows, and six models accordingly, had been used primarily in the work's presentation $\mathrm{L_H} = [2\mathrm{H}, 3\mathrm{H}, \dots 7\mathrm{H}]$. If more than one loss is required, each model is trained on each and every loss requested. The work had been centered mainly around MAPE, SMAPE and MASE metrics. As a consequence, if all the lookbacks are trained over all the losses a total of 18 models will be generated. Additionally the training is repeated a fixed number of times performing a bagging procedure, procedure which includes models with different random initialization. Forecasts from the ensemble are aggregating via median function. To cite an instance a total of 180 models had been employed to achieve SotA results over the M4 dataset (6 lookbacks ·3 metrics ·10 repetitions). The set of trainable parameters Θ is made of all the weight matrices $\mathbf{W}_*^*$ and bias vectors $\mathbf{b}_*^*$ of the various blocks and stacks, varying with the kind of architecture requested. The intermediate forecasts covering the horizon H, as well as the final one, is always available either at the final block in the chain or at stack level, depending on the architecture's kind.

To compare with probabilistic models like DeepAR and DeepSSM the authors had chosen the Normalized Deviation metric (ND)

$$\mathrm{ND}\left(\mathbf{z}_{1:\mathrm{T}}^i, \hat{\mathbf{z}}_{1:\mathrm{T}}^i\right) = \sum_{t=1}^{\mathrm{T}} \frac{\left|z_t^i - \hat{z}_t^i\right|}{\left|z_t^i\right|} \tag{3.29}$$

which is equivalent to the 0.5-risk ($p50$).

3.4 Research overview

As opposed to the state of the art just now introduced, we are going to review some of enterprises' efforts into demand forecasting research. As we already stated the adoption pace of AI models at large has increased overall across different industries, but Supply Chain management - and demand forecasting as a consequence - is one of the business assets with the lowest AI acceptance. Different enterprises are committed to different models depending on their field and the nature of their data. Hopefully it should be clearer at this point why. As a case of study we focus on some French companies, active in various economic fields. *É*lectricité *d*e *F*rance (EDF) is a mandatory mention in this brief review. *Load forecasting* is crucial for the planning and operation of electric utilities. At national level the models used in production are already achieving good performances; *G*radient *B*oosting *M*odels (GBMs), based on tree-learning algorithms, are still the preferred choice. Nonetheless the introduction of smart-meters to support a sustainable energy development and the flexibility market moved the challenge to the demand forecasting at single household and grid level. The latter would be inserted in a chain of correlated models to then back up operations. The direct implication is that any chosen model at this level has to be: as adaptable as possible - there are no two equals households -; complete training and prediction under the hour - the canonical forecasting horizon at this level ranges from one hour to one week -; able to provide consumption insights which will be translated into feedbacks for the customer. In 2019 Gérossier [Ger19] proposed, in his PhD work carried out within the SENSIBLE [6] framework, a short-term forecasting model based on the *A*lternating *D*irection *M*ethod of *M*ultipliers (ADMM) [BPC+11]. Proposed by Boyd et al. in 2011, the ADMM algorithm is a decomposition-coordination procedure which solves a complex convex optimization splitting the problem into smaller pieces which are easier to handle. Accordingly the forecast made at sub-problem level will be coherent with that made at the upper level, and so on. Gérossier used it to navigate from predictions at household level to that at grid level, as opposed to a global approximation or a bottom-up reconstruction. Load forecasting is without any doubt one of the most studied field, especially in this time of transition. Nevertheless even more up-to-date works [7] are revolving around hybrid solutions between classical and AI models, state space models - in which some of the hybrid architectures could potentially be categorized - and ensembles of (weak) models.

The most recent work found, strictly related to demand forecasting - made available online

[6] *S*torage *EN*abled *S*usta*I*nable energy for *B*ui*L*dings and communiti*E*: https://www.projectsensible.eu

[7] Search conducted among the publications presented on HAL: https://hal.science/search/index/?q=demand+forecasting&rows=30&sort=producedDate_-tdate+desc&docType_s=THESE+OR+UNDEFINED

in 2021 - is by Klibi et al. [KBDOAEA21]. They were forecasting sales for a cosmetic retailer, using basket data to construct exogenous factors and ARIMAX to perform the forecast. A year before Huard et al. [HGS20] used an ensemble of exponential smoothing and Holt's linear trend methods to solve Cdiscount's hierarchical demand forecasting problem. The ensemble had been optimized via *robust online aggregation*, otherwise called *prediction with expert advice*, to achieve coherency along the hierarchy. The Cdiscount's dataset greatly matched the peculiarities outlined in Sec. 3.1, recording daily sales of $\sim 620'749$ products. The researchers aggregated the sales by week and yield a forecast up to 6 weeks ahead. Back in 2012, Rostami Tabar [RT13] proposed a temporal aggregation algorithm to better handle forecast coherency and based the lowest level forecasts on ARIMA models.

The review of the available works designed in the industrial research is far from being exhaustive, despite that we can see a clear trend in it. The forecasting models have to be as lightweight and explainable as possible, preferably providing probabilities insights on future outcomes. Consolidating the concept, the forecasting task is a core task but it is not the only one. It is deeply bounded to other processes which form a complex chain of decision making mechanisms, and decision makers at different levels have to be able to understand, investigate and possibly quickly modify the outcome of such models.

3.5 At Lokad

3.5.1 Envision

Envision [8] is the Domain-Specific Language (DSL) engineered by Lokad. The language is specifically dedicated to the predictive optimization of supply chains. Unlike many scripting languages, Envision focuses on delivering a high-degree of correctness by design, which means capturing as many issues as possible at compile time - the moment when the script is compiled - rather than runtime - the moment when the script is run -. Capturing issues at compile time is preferable because whenever the amount of processed data is sizable, a runtime issue can take a long time (several minutes) to manifest itself causing productivity and production reliability problems. While the Lokad platform includes many features beyond Envision, the bulk of Lokad's capabilities are delivered through this DSL. Forecasting utilities is one of these capabilities. Numeric calculations, flat files - commonly extracted from clients' systems -, machine learning and probabilistic forecasting are all supported by Envision. Like SQL, Envision adopts array programming, processing whole columns at once. The language is space sensitive, much like Python. In order to create a supply chain

[8] Technical documentation: https://docs.lokad.com/

Listing 3.1: Envision: π approximation via Monte Carlo simulation

```
montecarlo 1000 with // approximate π value
  x = random.uniform(-1, 1)
  y = random.uniform(-1, 1)
  inCircle = x^2 + y^2 < 1
  sample approxPi = avg(if inCircle then 4 else 0)
show scalar "Approximation" with approxPi // 3.22
```

optimization app with Lokad, the Data Scientist - also named Supply Chain Scientist within the company - is expected to write a script in Envision code. Results are presented through Dashboards.

Although Envision is under a commercial license, the script in List. 3.1 can be ran accessing the Lokad's playground [9] which exposes a feature-limited version of Envision.

3.5.2 The forecasting engine evolution

Envision had been designed to evolve. Over 5 years of operations, the language underwent more than one hundred incremental rewriting. Rewriting ensure that the company's clients benefit from the latest version of Envision without having to manually revise their scripts. This allowed the forecasting engine to evolve at a brilliant pace, passing through different stages, as is clear in Fig. 3.11. The temporal scale is obviously not rigorous, but gives a good impression of when each stage has taken place. A significant amount of time and efforts had been devoted to support the continuous fast paced progression of the forecasting engine through the current work lifespan. We will briefly describe some of the critical points in the following.

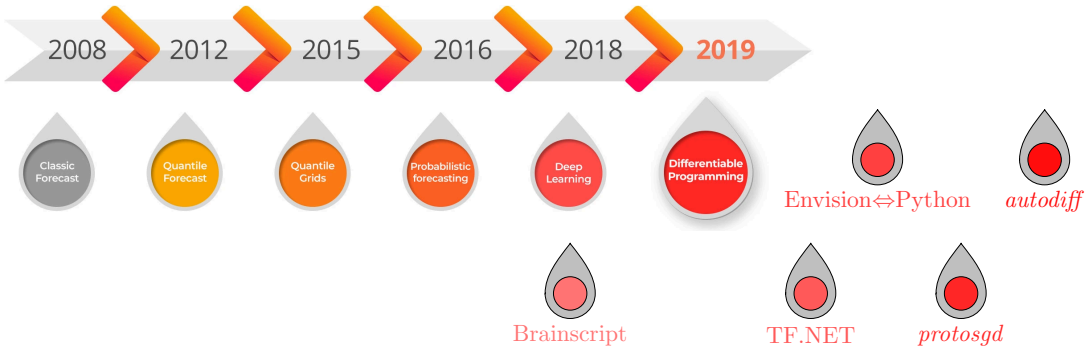


Figure 3.11: Lokad's forecasting engine evolution.

[9] Playground: https://try.lokad.com

Brainscript It was the model description language - hence a kind of domain specific language - introduced by Microsoft together with its *Cog**n**itive Tool**k**it* (CNTK) - no longer actively developed, last major realead in August 2022 -, an open-source toolkit for commercial-grade distributed deep learning. It described neural networks as a series of computational steps via a static directed graph. The network was trained through *s*tochastic *g*radient *d*escent (SGD). CNTK could be included as a library in Python, C#, or C++ programs, or used as a standalone.
Working with a network defined in Brainscript and running it via CNTK was a cumbersome operation. If initially it was on par with TensorFlow feature-wise, it was not able to keep the development progression of the competition and started to fall behind. To cite an instance, to make use of the ADAM optimizer an in-house version of CNTK had to be developed and deployed. Additionally some Deep Learning architectures - e.g. *V*ariational *A*uto *E*ncoders (VAEs) - were impossible to code due to missing features, e.g. random operations.

TensorFlow.NET[10] Began as an independent .NET Standard project it is now supported by Microsoft itself and accepted within the ML.NET infrastracture. The goal is to deploy bindings for Google's TensorFlow in C# and F# for developing, training and deploying Machine Learning models. The main consideration being that Python code, and especially Machine Learning frameworks like TensorFlow, resort to C calls for high-demanding performance portion of the code. Even though the project sounds promising, the problem is a dual rate development. The community behind TensorFlow.NET can not keep the Google's pace and not infrequently when a feature has been properly bound, it has to be reworked to math a change in the new release of the original code. The development delay was so large at some point that, to allow a proper experimentation of the solution, we contributed to several binding implementations; from simple numerical and logical operations to the gradient of the batch matrix multiplication.

Envision calls Python It was more an experiment run in collaboration with a single client. Data coming from the client's system would be digested by Envision before calling Python externally. In this way any in between software proxy would be avoided. This required a complex pipeline to be set to ensure data correctness. The experiment was ran in parallel with the development and debugging of *protosgd* and could be disposed after the latter's deployment.

protosgd Ancestor of *autodiff*, it was the embryo of differentiable programming efforts at Lokad. It was a Lisp-like DSL within the DSL. It required a dedicated compiler Please

[10] TensorFlow.NET repository: https://github.com/SciSharp/TensorFlow.NET

refer to Sec. 4.5 for more details.

autodiff Current forecasting engine completely integrated within Envision. Please refer to Sec. 4.5 for more details.

The temporal scale depicted in the figure, together with the different evolutionary steps highlighted, should be of help in judging how much effort and time is required to a company for select, test and integrate - and maybe fail and repeat - a new technological solution.

3.6 Summary

Demand time series are characterized by several oddities. First of all they are made only by positive integer observations, i.e. the recorded quantities - the sales - are always greater or equal to zero and non fractional. Due to this property they are also called count time series (Sec. 3.1.2). Sales are only a proxy for the real value we are interested into, the demand. Count time series are generated at large by processes which are intrinsically non-Gaussian, and that are better approximated via discrete distributions like Poisson or Negative Binomial. Depending on the application domain, it is not unusual to find demand time series aggregated by distinctive attributes and organized in a hierarchical manner (Fig. 3.1). The hierarchy in general reflects some more practical aspect of the business. A supermarket, for example, would arrange its products into departments. Within each departments we would find different product types. Among a type we would pick up the one specific product which is aligned with our preferences. The supermarket itself could just be a store in a more global retail chain.
Forecasting demand time series is not an easy task. Being filled with zeroes when no sales are noted, they present a variability in both time - inter-demand intervals, i.e. the time passed between two positive observations - and quantity - i.e. the observation's magnitude can vastly vary -. A measure of both variability is used to categorise the time series profile into four major classes - Sec. 3.1.3 -: erratic, lumpy, smooth, intermittent. Lumpy time series are the worst-case scenario to forecast, being characterized by a double high variability. On this kind of data, especially intermittent demand, we must pay attention in selecting the correct metrics for training and evaluation: a) some of the traditional metrics indeed are not tailored to demand data, since they can give infinite or undefined values; if the predictive distribution is asymmetric - e.g. the Negative Binomial - the metric of choice could bias the forecast towards zero, leaving us with no concrete business utility of the developed model. A training approach based on the (log-) likelihood and Maximum Likelihood Estimation is preferred in this context.
Section 3.2.1 and Sec. 3.2.2 introduced the parts dataset and the M4 dataset respectively. The

former had been shared by a US car company selling spare components and made of only lumpy and intermittent time series. The latter had been presented during the fourth edition of the Makridakis Competitions and it is more heterogeneous. Both of them are made of univariate, count time series. These two datasets had been chosen as common field to benchmark against the three State of the Art models: DeepAR [SFGJ19] and DeepSSM [RSG+18] - both from Amazon, commercially available and potentially employable in a production environment - and N-Beats [ODPT21]. The last one is considered to be the SotA model thanks to its great performance on the M4 dataset.
In the end we briefly reviewed the research efforts carried over by French companies and how they are not really aligned with the literature found.

Chapter 4

Automatic Differentiation & Differentiable Programming

Contents

Although the terms *A*utomatic *D*ifferentiation (AD) and Machine Learning had been pulled together only recently, the AD is a small but well established field already popular in various scientific areas. It is a family of powerful techniques for automatic numerical computation of functions' derivatives. One of its main modes - the *Reverse Mode* - is a generalization of the more popular backpropagation algorithm extensively used to train Neural Networks.

4.1 Introduction

Derivatives, both in the form of gradients or Hessians, are pervasive in the Machine Learning field and represent the drive for learning algorithms. Derivatives can be computed in different ways, the most naïve one being manually working out the derivatives and code them. Lets take the example presented in [GW08]

$$f(\theta_1,\theta_2) = \left[\sin\left(\frac{\theta_1}{\theta_2}\right) + \frac{\theta_1}{\theta_2} - e^{\theta_2}\right]\left(\frac{\theta_1}{\theta_2} - e^{\theta_2}\right) \tag{4.1}$$

and imagine that we are interested its derivative $\nabla f(\theta_1,\theta_2) = \left(\frac{\partial f}{\partial \theta_1}, \frac{\partial f}{\partial \theta_2}\right)$. Manually deriving the function we reach

$$\begin{aligned}
\nabla f(\theta_1,\theta_2) = \left(\frac{\partial f}{\partial \theta_1}, \frac{\partial f}{\partial \theta_2}\right) = \\
&\Bigg(\\
&\quad \left[\frac{1}{\theta_2}\cos\left(\frac{\theta_1}{\theta_2}\right) + \frac{1}{\theta_2}\right]\left(\frac{\theta_1}{\theta_2} - e^{\theta_2}\right) + \frac{1}{\theta_2}\left[\sin\left(\frac{\theta_1}{\theta_2}\right) + \frac{\theta_1}{\theta_2} - e^{\theta_2}\right], \\
&\quad \left[-\frac{\theta_1}{\theta_2^2}\cos\left(\frac{\theta_1}{\theta_2}\right) - \frac{\theta_1}{\theta_2^2} - e^{\theta_2}\right]\left(\frac{\theta_1}{\theta_2} - e^{\theta_2}\right) + \\
&\qquad \left[\sin\left(\frac{\theta_1}{\theta_2}\right) + \frac{\theta_1}{\theta_2} - e^{\theta_2}\right]\left(-\frac{\theta_1}{\theta_2^2} - e^{\theta_2}\right) \\
&\Bigg),
\end{aligned} \tag{4.2}$$

where no simplification had been carried out. Manual differentiation is error prone and requires a non-negligible amount of time. A way to automatise the process is *symbolic differentiation.*

Representing the function $f(\theta_1,\theta_2)$ symbolically, a symbolic differentiation engine will parse it and recurrently apply transformations of derivation rules [BPRS18] to get a symbolic expression for $\nabla f(\theta_1,\theta_2)$. For sake of presentation we show only the symbolic representation for $\frac{\partial f(\theta_1,\theta_2)}{\partial \theta_2}$

$$\begin{aligned}
\frac{\partial f(\theta_1,\theta_2)}{\partial \theta_2} = &\left[-(\theta_1/(\theta_2 * \theta_2)) * cos(\theta_1/\theta_2) - (\theta_1/(\theta_2 * \theta_2)) - e^{\theta_2}\right]\left[(\theta_1/\theta_2) - e^{\theta_2}\right] + \\
&\left[\sin(\theta_1/\theta_2) + (\theta_1/\theta_2) - e^{\theta_2}\right]\left[-(\theta_1/(\theta_2 * \theta_2)) - e^{\theta_2}\right]
\end{aligned}$$

which is exact, correct but not easy to digest. The formula can grow exponentially, depend-

ing on the original function itself, the order of the derivative requested and the number n of dimensions involved; this problem is known as *expression swell*. The latter directly affects also the technique's computational complexity, which grows as the representation expands. The formulae so far derived could be simplified, but the process can be tricky and time consuming.

Most of the time however we are interested in the numerical value of a derivative rather than its mathematical expression. *Numerical differentiation* computes derivative's numerical values through finite differences, evaluating the function at some point of interest

$$\begin{aligned}\nabla f(\theta_1,\theta_2)_{|_{\bar{\theta}_1,\bar{\theta}_2}} &= \left(\frac{\partial f}{\partial \theta_1}, \frac{\partial f}{\partial \theta_2}\right), \\ &\approx \left(\frac{f(\bar{\theta}_1+h)-f(\bar{\theta}_1)}{h}, \frac{f(\bar{\theta}_2+h)-f(\bar{\theta}_2)}{h}\right).\end{aligned}$$

The approach is not free of downsides. Computationally it scales poorly, requiring $\mathcal{O}(n)$ to compute a n-dimensional gradient; practically it is easy to code but plagued by numerical errors directly related to the choice of h: for small h a *round-off* error dominates; for large h it is the *truncation* error which becomes dominant. As a matter of fact this technique is used, in the majority of the cases, as a validation step in the implementation of more complex techniques.

Both symbolic and numerical differentiation need to work with *closed form* expressions; control flows with unknown number of iterations, conditional statements, etc. are not good candidates to be differentiated by these means. Automatic Differentiation can compute derivatives without the downfalls of the aforementioned techniques. The main idea is to reduce calculations into elementary steps. Each step belongs to a finite set of operations for which the derivatives are well known [Ver00, GW08, BPRS18]; we can include in this set the arithmetic and binary arithmetic operations, sign switch, trigonometric functions, exponential and logarithm and so on. Griewank et al. [GW08] introduced a *three-part notation* to schematize a function decomposition into atomic operations.

An arbitrary function $f : \mathbb{R}^n \to \mathbb{R}^m$ can be decomposed into atomic operations represented by intermediate variables v_i such that

- variables $v_{i-n} = \theta_i$, $i = 1 \dots n$ for the input variables;
- variables v_j, $j = 1 \dots l$ for the working intermediate variables;
- variables $y_{m-k} = v_{l-k}$, $k = m-1, \dots, 0$ for the output variables.

The numbering system is arbitrary as long as it is consistent. Table 4.1 shows the decomposition steps for the example in Eq. (4.1); Figure 4.1 presents the same record in a graphical way, highlighting dependency relations between intermediate variables.

Table 4.1: Evaluation trace for the function presented in Eq. (4.1).

Decomposed	**Original**
v_{-1}	θ_1
v_0	θ_2
$v_1 = \frac{v_{-1}}{v_0}$	$= \frac{\theta_1}{\theta_2}$
$v_2 = \sin(v_1)$	$= \sin\left(\frac{\theta_1}{\theta_2}\right)$
$v_3 = e^{v_0}$	$= e^{\theta_2}$
$v_4 = v_1 - v_3$	$= \frac{\theta_1}{\theta_2} - e^{\theta_2}$
$v_5 = v_2 + v_4$	$= \sin\left(\frac{\theta_1}{\theta_2}\right) + \frac{\theta_1}{\theta_2} - e^{\theta_2}$
$v_6 = v_5 * v_4$	$= \left[\sin\left(\frac{\theta_1}{\theta_2}\right) + \frac{\theta_1}{\theta_2} - e^{\theta_2}\right]\left(\frac{\theta_1}{\theta_2} - e^{\theta_2}\right)$
$y_1 = v_6$	

Respectively the steps recorded in Tab. 4.1 and the graph in Fig. 4.1 are called *evaluation trace* - or *Wengert's list* [Wen64] - and *computational graph* [Bau74]. Computational graph should not be a new term, as neural networks are usually visualized in this way. Assigning values to θ_1 and θ_2 we can compute the value of y_1 following the evaluation trace
The evaluation trace shown is not unique, we could have defined v_2 as $v_2 = e^{v_0}$ - and accordingly all the other steps - without changing the outcome. Contrary to the derivation in the symbolic case, sub-expressions - hence intermediate variables - are reused avoiding the expression swelling problem. Loops, branching, recursion and conditional statements, as well as procedure calls, can be modelled via evaluation traces and the computational graph going beyond the closed form constraint.
Evaluation traces are the AD's core and we will see in the following how they are exploited in the two main AD modes: *Forward* and *Reverse* mode.

Table 4.2: Evaluation trace for the function presented in Eq. (4.1).

Decomposed	**Original**	**Value**	
v_{-1}	θ_1	1.5000	
v_0	θ_2	0.5000	
$v_1 = \frac{v_{-1}}{v_0}$	$= \frac{\theta_1}{\theta_2}$	$\frac{1.5000}{0.5000}$	$= 3.000$
$v_2 = \sin(v_1)$	$= \sin\left(\frac{\theta_1}{\theta_2}\right)$	$\sin(3.0000)$	$= 0.1411$
$v_3 = e^{v_0}$	$= e^{\theta_2}$	$e^{0.5000}$	$= 1.6487$
$v_4 = v_1 - v_3$	$= \frac{\theta_1}{\theta_2} - e^{\theta_2}$	$3.000 - 1.6487$	$= 1.3513$
$v_5 = v_2 + v_4$	$= \sin\left(\frac{\theta_1}{\theta_2}\right) + \frac{\theta_1}{\theta_2} - e^{\theta_2}$	$0.1411 + 1.3513$	$= 1.4924$
$v_6 = v_5 * v_4$	$= \left[\sin\left(\frac{\theta_1}{\theta_2}\right) + \frac{\theta_1}{\theta_2} - e^{\theta_2}\right]\left(\frac{\theta_1}{\theta_2} - e^{\theta_2}\right)$	$1.4924 * 1.3513$	$= 2.0167$
$y_1 = v_6$			$= 2.0167$

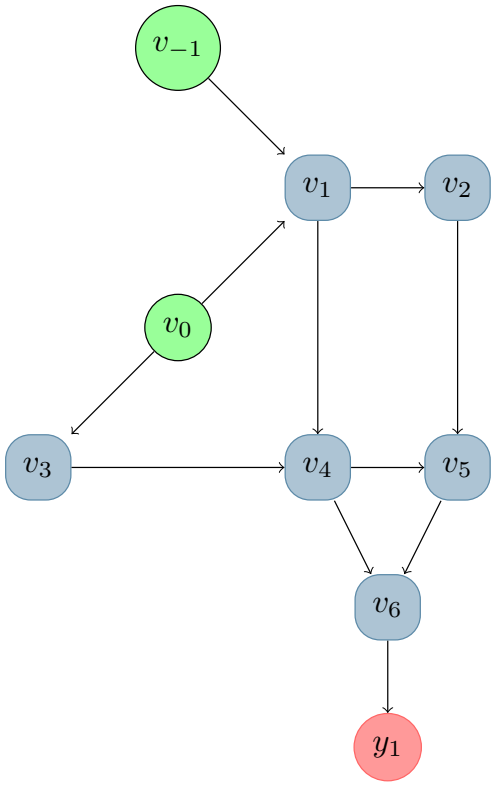


Figure 4.1: Computational Graph.

4.2 Forward Mode

The forward mode, also known as *tangent mode*, is conceptually the most simple modality. Considering the evaluation trace in Tab. 4.1 and the computational graph in Fig. 4.1, we augment each line and each node with the derivative of the intermediate variable v_j

$$\dot{v}_j = \frac{\partial v_j}{\partial \theta_i}.$$

Applying the chain rule to each elementary operation in the evaluation trace we obtain a tangent (derivative) trace as shown in Tab. 4.3. The forward mode is a natural generalization of the Jacobian's evaluation with n variables $\theta_{i=1\ldots n}$ and m output variables $y_{k=1\ldots m}$. The derivative $\partial f/\partial \theta_i$ with reference to θ_i is evaluating plugging the actual value of θ_i and setting $\dot{\theta}_i = 1$, leaving the rest at zero; the full Jacobian is then evaluated in n runs over the tangent trace.

Table 4.3: Forward mode evaluating the derivative for the function in Eq. (4.1).

Decomposed	**Derivative (Tangent)**
v_{-1}	$\dot{v}_{-1} = \dot{\theta}_1$
v_0	$\dot{v}_0 = \dot{\theta}_2$
$v_1 = v_{-1}/v_0$	$\dot{v}_2 = \dot{v}_1 \cos(v_1)$
$v_3 = e^{v_0}$	$\dot{v}_3 = \dot{v}_0 e^{v_0} = \dot{v}_0 v_3$
$v_4 = v_1 - v_3$	$\dot{v}_4 = \dot{v}_1 - \dot{v}_3$
$v_5 = v_2 + v_4$	$\dot{v}_5 = \dot{v}_2 + \dot{v}_4$
$v_6 = v_5 * v_4$	$\dot{v}_6 = v_4 \dot{v}_5 + \dot{v}_4 v_5$
$y_1 = v_6$	$\dot{y} = \dot{v}_6$

Forward mode is straightforward and efficient for functions $f : \mathbb{R} \to \mathbb{R}^m$, i.e. when $n = 1$; in this case indeed the full Jacobian can be computed in a single pass. The opposite is true for $f : \mathbb{R}^n \to \mathbb{R}$, i.e. when $m = 1$. In the latter case the Jacobian is defined as

$$\nabla f = \left(\frac{\partial f}{\partial \theta_1}, \ldots, \frac{\partial f}{\theta_n} \right)$$

but we still need n runs to compute it. In general the forward mode is preferred for cases where $n \ll m$.

4.2.1 Dual Numbers

A convenient way to link forward mode, automatic differentiation and derivatives is through *Dual Numbers*. Dual numbers had been introduced as an extension to the Hamilton's vec-

tor [Cli71] to consider rotation around any line in the tri-dimensional space. Clifford called this extension rotors and the sum of such rotors a motor. He also needed to introduce a way to go back and forth between rotors and vectors, he so introduction $\omega^2 = 0$. Formally dual numbers follow the component-wise addition rule

$$(a + b\omega) + (c + d\omega) = (a + c) + (b + d)\,\omega$$

and have a multiplication similar to that of complex numbers

$$\begin{aligned}(a + b\omega) * (c + d\omega) &= ac + (ad + bc)\,\omega + bd\omega^2 \\ &= ac + (ad + bc)\,\omega.\end{aligned}$$

The link between differentiation and dual numbers resides in the Taylor series expansion. Any polynomial with real coefficients

$$f(\theta) = c_0 + c_1\theta + c_2\theta^2 + \cdots + c_n\theta^n$$

can be revised as a function of dual numbers

$$\begin{aligned}f(a + b\omega) &= c_0 + c_1(a + b\omega) + c_2(a + b\omega)^2 + \cdots + c_n(a + b\omega)^n \\ &= c_0 + c_1 a + c_2 a^2 + \cdots + c_n a^n + c_1 b\omega + 2c_2 ab\omega + \cdots + c_n a^{n-1} b\omega \\ &= f(a) + b\frac{\mathrm{d}f}{\mathrm{d}a}\omega.\end{aligned}$$

Generalizing for any real function, the corresponding Taylor series expansion at θ_0

$$f(\theta) = f(\theta_0) + f'(\theta_0)(\theta - \theta_0) + \cdots + \frac{f^{(n)}(\theta_0)}{n!}(\theta - \theta_0)^n + \mathcal{O}\left((\theta - \theta_0)^{n+1}\right)$$

can be extended with dual numbers to

$$f(a + b\omega) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)\, b^n \omega^n}{n!}(\theta - \theta_0)^n = f(a) + bf'(a)\,\omega.$$

Setting $b = 1$ and recalling that $\omega^2 = 0$, we can evaluate the expansion at $\theta = a$ as $f(a + \omega) = f(a) + f'(a)\,\omega$. Under this formulation the addition of functions

$$f(a + \omega) + g(a + \omega) = f(a) + g(a) + (f'(a) + g'(a))\omega,$$

their multiplication

$$f(a+\omega)\,g(a+\omega) = f(a)\,g(a) + (f'(a)\,g(a) + f(a)\,g'(a))\omega,$$

as well as the chain rule

$$f\left(g(a+\omega)\right) = f\left(g(a) + g'(a)\,\omega\right) = f\left(g(a)\right) + f'\left(g(a)\right)g'(a)\,\omega$$

are satisfied. The higher-dimensional case where the gradient ∇f is needed is a direct extension of the one-dimensional case where each components is associated with an ω quantity.

4.3 Reverse Mode

If the forward mode is also known as tangent mode, its counter reverse mode is often called *cotangent linear* or *adjoint* mode. Contrary to the forward mode which propagates the derivatives from the input to the output, the reverse mode evaluates derivatives backward. It augments the evaluation trace and the computational graph with an adjoint

$$\bar{v}_j = \frac{\partial y_k}{\partial v_j}.$$

of the intermediate variable v_j. The calculation is broken in two phases: a first pass is run forward populating the trace with intermediate v_j's values and recording the dependencies of the computational graph; the second pass propagates adjoints $\bar{v}_j$ backwards, from outputs to inputs, to compute derivatives.

Table 4.4 lists the adjoint evaluation trace. Some of the adjoints are listed more than once, in incremental steps. The incremental steps are just a convenient way to break the adjoint's contribution and align it with the line which generated it. Taking v_4 as an example and looking up Fig. 4.1, we see that v_4 affects the output through v_5 and v_6, hence its contribution to the change in y_1 is given by

$$\frac{\partial y_1}{\partial v_4} = \frac{\partial y_1}{\partial v_5}\frac{\partial v_5}{\partial v_4} + \frac{\partial y_1}{\partial v_6}\frac{\partial v_6}{\partial v_4}$$

or

$$\bar{v}_4 = \bar{v}_6\frac{\partial v_6}{\partial v_4}$$
$$\bar{v}_4 = \bar{v}_4 + \bar{v}_5\frac{\partial v_5}{\partial v_4}$$

Table 4.4: Forward mode evaluating the derivative for the function in Eq. (4.1).

Decomposed	**Derivative (Adjoint)**
v_{-1}	$\bar{\theta}_1 = \bar{v}_{-1}$
v_0	$\bar{\theta}_2 = \bar{v}_0$
$v_1 = v_{-1}/v_0$	$\bar{v}_{-1} = \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$
	$\bar{v}_0 = \bar{v}_0 + \bar{v}_1 \frac{\partial v_1}{\partial v_0}$
$v_2 = \sin(v_1)$	$\bar{v}_1 = \bar{v}_1 + \frac{\partial v_2}{\partial v_1}$
$v_3 = e^{v_0}$	$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0}$
$v_4 = v_1 - v_3$	$\bar{v}_1 = \bar{v}_4 \frac{\partial v_4}{\partial v_1}$
	$\bar{v}_3 = \bar{v}_4 \frac{\partial v_4}{\partial v_3}$
$v_5 = v_2 + v_4$	$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2}$
	$\bar{v}_4 = \bar{v}_4 + \bar{v}_5 \frac{\partial v_5}{\partial v_4}$
$v_6 = v_5 * v_4$	$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4}$
	$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5}$
$y_1 = v_6$	$\bar{v}_6 = \bar{y}$

in incremental steps.

Reverse mode is significantly less costly to evaluate than the forward mode and performs better when $n \gg m$; taking again the edge case $f : \mathbb{R}^n \to \mathbb{R},\ m = 1$, the derivative can be computed with a single reverse application in place of the n operations required by the forward mode. The latter describes the derivation of a scalar-valued function with respect to a large number of parameters, which is most known in the machine learning community with the name of backpropagation. Reverse mode is indeed a generalization of backpropagation. The cost for the efficient derivative evaluation is a higher memory's waste, which can grow proportionally - in the worst case - to the number of operations. How to improve storage requirements is still an active research area [DH06, SP18].

4.4 Automatic Differentiation

A naïve implementation of automatic differentiation could result in prohibitively slow code and excessively use of memory. Both actual implementation and memory management need to be carefully thought to reach a good trade-off between the two. Over the years several research groups active in the field lead to the introduction of different approaches for both subjects. A comparison of libraries and packages demonstrating each approach is difficult and mostly empirical, mostly due to the tight relation between an approach and the language in which it is implemented.

4.4.1 On the computational subject

Three are the principal approaches implementation-wise [vMBBL18]:

Domain Specific Language (DSL) A sub-language is created specifically to support automatic differentiation. Any function meant to be automatically differentiated has to be written in the DSL and there should be a one-to-one mapping between the DSL and the host language, i.e. the DSL should support all the operations available in the original code. A manual rewriting step is obviously required with the translation being as fast as the DSL matches the host language. The more the DSL moves away from the original, the more the translation process will take time and be keen to errors. The computational graph outlined by the DSL is statically defined and can potentially be optimized ahead of time.

Operator Overloading (OO) is based on the host language's capability to: extend variables with a new type containing additional derivative information; overload functions and operators to use these new types. This can be an elegant and powerful solution, the main advantage being implementation easiness. The principal drawback is an additional interpretation overhead: the adjoint program is dynamically constructed at each execution - i.e. we deal with a dynamic computational graph - and requires an embedded interpreter, the last can interfere with debugging and performance analysis. Moreover since the tracing operation happens at runtime, operator overloading incurs overhead at each function call which can be particularly problematic if the primitive operation is faster to evaluate than the tracing operation. Finally operator overloading doesn't allow ahead-of-time optimization since the approach is not aware of the entire computational graph.

Source Transformation (ST) constructs a new source code for the adjoint program, starting from the original program, in the same host language. Then the original and the adjoint programs are interpreted and executed together. For this approach to be

effective the code has to be available in its entirety. As in the DSL case but contrary to the OO one, the graph is statically defined; the path to be taken is chosen at runtime depending on the input.

It is not uncommon to found solutions that mix two of the previous approaches, e.g. DSL and ST.

4.4.2 On the memory management

Intermediate variables created over the tracing process need to be stored for later use. Three are the leading memory schemes at the moment:

Tape and Retaping A global stack - called *tape* - is created to place the intermediate variables. In the ST approach the tape is an actual stack, whereas in the OO case it is a program trace which stores the executed primitives besides intermediate variables. The primal program writes to the tape over the forward pass, while the adjoint program reads from the tape during the backward pass. A downside of tape-based implementation is the tape's construction at runtime, which complicates the optimization process. Moreover it can represents a memory bottleneck, since each AD sweep will have its own tape associated. An intuitive example is the Jacobian's computation already discussed in Sec. 4.2; n transitions will be performed to compute the partial derivative, consequently n tapes will be created. In this case a technique termed *re-taping*[Mar19] can be employed: rather than reconstructing the computational graph at each sweep, information about the structure and intermediate variables are stored to be reused in the successive passages. Of course re-taping can be effective when the computational graph does not change from evaluation to evaluation, i.e. if no *true* conditional statements are involved in the target function.

Checkpointing had been developed as a trade-off between computing performance and memory usage, in particular it can mitigate memory usage peaks [RLG98, DH06, Mar19]. The underlying idea had been presented by Griewank [Gri92] who observed how the reverse mode does not need to record the entire computation. He suggested how the reverse mode could be implemented following a divide and conquer style: the computation flow is subdivided into sections, at appropriate *checkpoints*; the last section - from the second to last to the last checkpoint - is recorded and the corresponding derivatives are taken; finally the computation is resumed from the third to last to the most recent checkpoint. An immediate question is where the checkpoints should be placed. As argued by Hascoet et al [HP13] there is no optimal placement;

if the program can be split into sequential computational steps the *binomial partitioning* scheme presented by Griewank [Gri92]. Checkpointing is also related to AD parallelism; if we can split the program into independent sections, it means that we can evaluate each section separately.

Region Based Memory Allocating and freeing memory one intermediate variable at the time can be excessively costly. Region Based Memory management [GA01, Gay06] focuses on regions - stacks - which can be dedicated to a derivative calculation: intermediate variables are allocated on a custom stack element-wise; when the derivative had been evaluated, the entire stack is destroyed and the associated memory is freed.

4.4.3 In Machine Learning frameworks

Automatic Differentiation started to appear in Machines Learning frameworks in 2005, with TensorFlow and Pytorch trying to get the lead of the charge. Early TensorFlow opted for a DSL in which the user could expressly define the computational graph. Each variable and function should had been marked expressly and the AD would perform the derivation on the static graph so defined, as shown in List. 4.1.

Listing 4.1: Computational graph for ReLU3's evaluation, defined in the TensorFlow's DSL.

```
import tensorflow as tf

x = tf.Variable(3.14)

with tf.GradientTape() as tape:
    out = tf.Condition(
        x > 0,
        lambda: tf.math.pow(x, 3),
        lambda: 0
    )

grd = tape.gradient(out, x).numpy()
```

Requiring the user to personally specify the graph however was harmful for flexibility. Control flow constructs would be limited to those that could be defined statically, for example a `tf.Condition(<condition>, <then_branch>, <else_branch>)` function statement is different from a conditional `if ... then ... else` code because the first is semantically equivalent to calling both branches and then select the actual branch based on the input. TensorFlow had a domain-specific tensor-oriented compiler named *Accelerated Linear Algebra* (XLA) to produce an *intermediate representation* (IR) and simplify the computational graph. With TensorFlow Eager, a certain degree of dynamic control flows support was added. A function called in eager mode will be immediately executed; loops and conditional

Listing 4.2: Computational graph for ReLU3's evaluation, defined in Pytorch.

```
import torch

# Before it was torch.Variable()
x = torch.tensor([3.14], requires_grad = True)
relu3 = torch.where(x > 0, torch.pow(x, 3), 0)

relu3.backward()
```

statements will be unrolled and the resulting sub-graph returned as soon as everything had been made static. This of course added an additional computation overhead for three main reasons: long and complex loops will put the execution in idle until fully evaluated; the corresponding sub-graph could be a one-shot evaluation, being discarded immediately and never saw again, depending on the inputs; XLA optimizations could not be applied anymore because complete information on the computational graph are lost. Eager mode also failed the community's performance expectations, pushing researchers and practitioners towards alternative solutions.

Pytorch, on the other hand, preferred an OO approach, providing some level of control flows since the very beginning via ad-hoc constructs. It came first in unrolling loops and conditional statements but backed up the computational overhead with a huge catalog of heavily optimized functions which hid the additional burden [1] when optimization involved huge models, as in the Deep Learning case. With smaller optimization problems, the raised computation time would be appreciable again.
Comparing List. 4.1 and List. 4.2 highlights another difference in the computational graph construction between the two packages: TensorFlow had a *define-and-run* construction - first we build the graph, then we run the computation - while Pytorch pointed at a *define-by-run* composition in which the data coming into the model would select the outcome.

Successive solutions focused efforts on expanding the domain of supported functions. Applying operator overloading and enclosing the *numpy* package [Oli07], *Autograd* [MDA15] was a first independent attempt to widen automatic differentiation support to Python, and consequently to third-party libraries e.g. *scipy* [JOP01]. The package got so much a foothold in the machine learning community that Autograd became a synonymous of automatic differentiation for many. Being a wraparound numpy, Autograd's performance were deeply bounded to those of the original scientific computing package; furthermore the developing team scheduled but never delivered support for GPUs, which set for constant downfall of

[1] Where do the 2000+ PyTorch operators come from?: More than you wanted to know

Listing 4.3: Computational graph for ReLU3's evaluation, defined in Pytorch.

```
using Zygote

# An alternative and valid solution
# would be
# relu3(x) = ifelse(x > 0, pow(x, 3), 0);
relu3(x) = x > 0 ? pow(x, 3) : 0;

x = 3.14;

Zygote.pullback(relu3, x);
```

usages. Part of the Autograd's developing team moved to Google and in the 2018 launched JAX: following the same philosophy behind Autograd, JAX put Autograd and XLA near each other. JAX too was essentially a wrapper around numpy, but added another level of abstraction introducing an extra IR, on which AD is performed, which is then lowered to the XLA. As in the TensorFlow case, dynamism was introduced via ad-hoc function calls. JAX, with its functional soul, expected to work on *pure* functions - i.e. functions that don't do side effects on their inputs - and this imposed some restrictions on the code which can be automatically differentiated, e.g. array mutation through indexing could not be supported.

All the solutions so far outlined operated AD on an IR which can't be optimized, or optimized up to a certain degree but before the compiler can run its own optimizations. Recently projects like *Zygote.jl* [Inn19] - for Julia - and *Enzyme* [MC20] - a cross-language solution - committed to work on an IR which could be directly digested by an underlying compiler. Zygote used a different strategy to trace the variables and construct the computational graph applying *Static Single Assignment* (SSA) [CFR+91], a generalization of the Wengert's list in which each and every variable is assigned exactly once. Zygote extended the Julia's dynamic semantics with syntactic AD transformations; the IR is then be fed to the LLVM compiler. Potentially Zygote has the ability to support true control flow constructs - i.e. not limited to some ad-hoc function calls - as well as array mutation; however the developers team brought some constraints to the table, e.g. not supporting array mutation, when it came to release the package, in order to ease the implementation. Enzyme on the other hand was conceived as a cross-language, cross-platform plugin for the LLVM compiling infrastructure itself. Operating directly on the LLVM IR, the LLVM's optimizations can be applied straightforward to the adjoint program. That being said, Enzyme can be applied to any LLVM compiled language such as C++, Rust and Swift. Both Zygote and Enzyme allow a definition of the model in plain Julia, as shown in List. 4.3 and List. 4.4, without recurring to any DSL or additional IR.

Listing 4.4: Computational graph for ReLU3's evaluation, defined in Pytorch.

```
using Enzyme

# An alternative and valid solution
# would be
# relu3(x) = ifelse(x > 0, pow(x, 3), 0);
relu3(x) = x > 0 ? pow(x, 3) : 0;

x = 3.14;
xshadow = 1;

# Reverse: specifies reverse mode AD
# Active: take derivative w.r.t. this
# variable
Enzyme.autodiff(Reverse, relu3, Active(x));
```

4.4.4 Differentiable Programming

With the term *Differentiable Programming* we address a novel programming paradigm in which models are seen as a concatenation of parameterized differentiable blocks. Each block can be defined in a recursive way; it can either be an atomic operation - e.g. a mathematical operator - or a concatenation of blocks. The term had been coined by Prof. Yann Lecun[2], back in 2018, to describe a super-set of the Deep Learning field in which programs, constructed via differentiable blocks and relative parameters, could be optimized using well known gradient-based optimization techniques. Prof. Lecun noted how popular machine learning frameworks were working to help users in defining neural networks in procedural data-dependent way, adding support for loops and conditional controls - as we outlined in the previous section - and how neural networks were evolving towards a more dynamic functional version of themselves. From this point of view, NNs match the definition of algorithms and, as we move AD systems from an abstraction level to an intrinsic feature of the language - or even deeper of the compiler - we can also imagine a shift in designing specialized chips for AI.

[2]https://www.facebook.com/yann.lecun/posts/10155003011462143

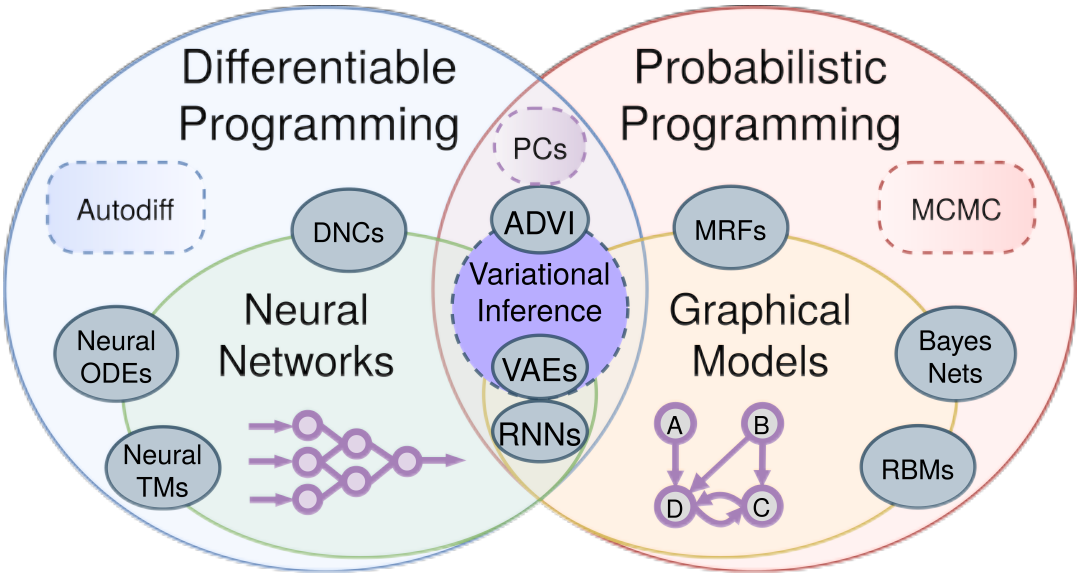


Figure 4.2: Differentiable programming includes neural networks, but more broadly, arbitrary programs which use gradient-based optimization to approximate a loss function. *Probabilistic programming* [CGH+17, GGS19, THT+14] is a generalization of probabilistic graphical models which uses Monte Carlo methods to approximate a density function. Source: [Con21].

4.5 At Lokad

Listing 4.5: Envision: ReLU3 implemenation.

The keyword *autodiff* directly marks a function as differentiable. The ADSL language will automatically take care of the derivative if the function is used in an *autodiff* block

```
def autodiff pure relu3(x : number) with
  return if x > 0 then x^3 else 0
```

Differentiable programming and automatic differentiation represent the sixth generation of the company's forecasting engine. The first attempt, named *protosgd*, was a DSL within Envision. The sub-DSL was a Lisp-like language equipped with a standalone compiler to lower the representation, following a ST schema. For the memory management a tape-based implementation, with a re-taping option, had been preferred over the others. The successor of this prototype, and actual solution used in production, had been called *autodiff* and it has been the main topic for another PhD thesis conducted at Lokad.

Despite its name which recalls the Python package, it is more akin to most of the Zygote's and Enzyme's basic principles. The model can be specified completely in plain Envision, the SSA rule is employed over computational graph's construction and it is perfectly integrated into the language's compiler. Contrary to Zygote, array mutation is allowed. Dissecting En-

Listing 4.6: Envision: simple linear regression with autodiff.

```
table T = with
  [| as X, as Y |]
  [| 1, 3.1 |]
  [| 2, 3.9 |]
  [| 3, 5.1 |]
  [| 4, 6.0 |]

autodiff T with // automatic differentiation
  params a auto // 1st parameter
  params b auto // 2nd parameter
  return (a * T.X + b - T.Y)^2 // loss for the stochatic gradient descent

show scalar "Learned" a1c1 with "y ~ \{a} x + \{b}"
```

vision, autodiff can be seen similar to Zygote. Envision indeed comprises circa ten different languages, each with its own IR, all developed in-house; the one dedicated to autodiff takes the name of *ADSL* [3] [Pes21]. Hence we have a chain of source transformation before reaching the final and executable one, ADSL being one of the many. Taking Envision as a whole autodiff is alike Enzyme, being its dedicated sub-language perfectly embedded into the compiler and subject to the latter optimization rules.

4.6 Summary

The chapter presented Automatic Differentiation reviewing its main modes, Forward (Sec. 4.2) and Reverse (Sec. 4.3), and all the relevant associated definitions.
Successively the major solution for both the computational requirement and the memory management had been presented respectively in Sec. 4.4.1 and Sec. 4.4.2 together with how these solutions are reflecting into mainstream AD frameworks like Pytorch and Tensorflow (Sec. 4.4.3). The emerging Differentiable Programming paradigm, offspring of AD and Machine Learning, had been briefly summarised in Sec. 4.4.4
Finally in Sec. 4.5 we had shortly outlined how automatic differentiation and differentiable programming had been embedded into Lokad's workflow.

[3] ADSL repository

Chapter 5

Probabilistic exponential smoothing for demand forecasting

Contents

5.1 Introduction

The work presented in this thesis is focused on the univariate time series, probabilistic forecasting problem. The model had been derived from the well known Holt–Winter model but,

contrary to its ancestor, it can accommodate multiple seasonality as well as different interaction modalities between its components. As discussed in Sec. 5.3.2, seasonality can be shared among related time series and learnt jointly. The most commonly used seasonality, or combinations of them, are listed in Tab. 5.1.

Table 5.1: Seasonal granularity: for each frequency we can initialize from one to several frequency-dependent seasonal profiles (marked with * in the table). Abbreviations stand for: *H*our-*o*f-the-*D*ay; *D*ay-*o*f-the-*W*eek; *D*ay-*o*f-the-*Y*ear; *W*eek-*o*f-the-*Y*ear; *M*onth-*o*f-the-*Y*ear; *Q*uarter-*o*f-the-*Y*ear

	Granularity					
Frequency	HoD	DoW	DoY	WoY	MoY	QoY
Hourly	*	*				
Daily		*	*	*		
Weekly				*	*	
Monthly					*	
Quarterly						*

An interesting property of the model presented is the direct mapping between business requirements and model's parameters. We will discuss this aspect further in Sec. 5.3. The architectural design allows to externally pass the model's parameters, allowing for a later inspection and modification (if needed). This embrace the Differentiable Programming paradigm and the company's DSL approach.

Section 5.2 is going to dissect the popular Holt–Winter model to derive an analogy between the current work and the LSTM cell. This parallelism will help the presentation of the various fundamental operations which thrust our model. Section 5.4 shows the accuracy metrics over the two datasets already presented, namely parts and the M4 Hourly.

5.2 An LSTM analogy

Exponential Smoothing family's equations (Sec. 2.3.2) are recurrent, i.e. auto-regressive, by design. We report in the following the (slightly modified) Holt-Winter method with additive trend and multiplicative seasonality

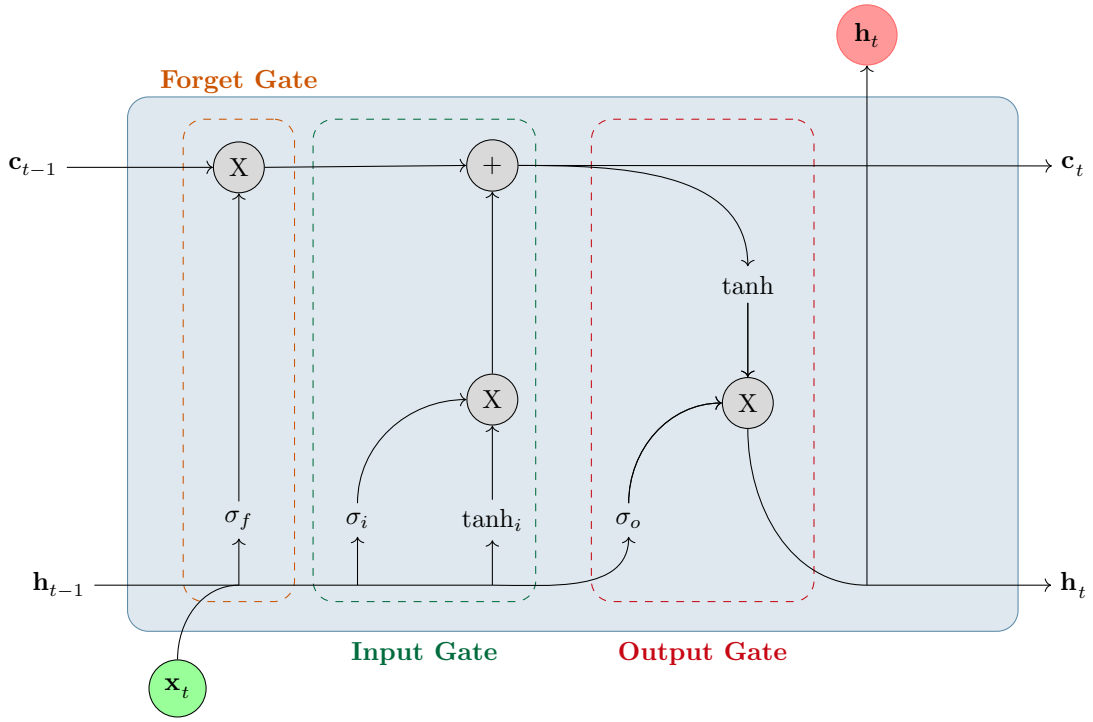


Figure 5.1: Basic LSTM cell as already presented in Fig. 2.7. The three gates - namely forget , input and output gate - are highlighted.

$$\begin{cases} l_t = \alpha \dfrac{z_{t-1}}{s_{t-m}} + (1-\alpha)\left(l_{t-1} + t_{t-1}\right), \\ t_t = \beta\left(l_t - l_{t-1}\right) + \left(1-\beta\right) t_{t-1}, \\ s_t = \gamma \dfrac{z_{t-1}}{l_{t-1} + t_{t-1}} + \left(1-\gamma\right) s_{t-m} \\ \hat{z}_{t+\mathrm{H}|t} = \left[l_t + (\mathrm{H}+1)\, t_t\right] s_{t+\mathrm{H}-m}, \end{cases} \tag{5.1}$$

with: α, β and γ smoothing factors associated to level l_*, trend t_* and seasonality s_* respectively; m the seasonality period and $\mathrm{H} = 0, 1, \ldots$ the forecasting horizon. The method had been chosen for convenience, since it presents all the useful operations to support the coming explanation.

5.2.1 Context & state vectors

Searching for similarities between the neural architecture and the method, the previous time step values stand out. They are directly related to their successors and updated at each time step. We can therefore collect them into a single vector which can be called state vector, and labelled accordingly as

$$h_{t-1} = \left[\tilde{z}_{t-1}, l_{t-1}, t_{t-1}\right],$$

with the addition of $\tilde{z}_{t-1}$ value which had been introduced to keep track of the input values z_t as the system evolves over time.
Despite the indexes also s_{t-m} is the immediate predecessor of s_t, however it has to be treated differently. Imagine to have a monthly seasonality with $m = 12$. The seasonal factor s_{13} would be a function of s_1, $s_{14} = \omega_m(\cdot, s_2), \ldots, s_{24} = \omega_m(\cdot, s_{12}), \ldots$ and so on and so forth in a circular manner. Consequently seasonal values computed at time t have to be stored for later use at time $t+m$ and we need a way to stock them. Without loosing generality we can set H = 0 - i.e. we ask for the actual value's prediction, given the past state - since, for any given H, the Holt-Winter method will carry on the prediction step by step - as an LSTM would do - but without updating internal components. The seasonal information are indeed projected over a longer time span than those enclosed in the state, in a similar fashion of a context vector. We can therefore imagine that our context vector is given by

$$\mathbf{c}_{t-1} = \left[s_1, s_2, \ldots, s_m\right].$$

Generalizing we could write for $t = (m+1), (m+2), \ldots, s_{[t-(m\lfloor t/m \rfloor)]} = \omega_m(\cdot, \mathbf{c}_{t-1})$. The $\omega_m(\cdot)$ operator acts like an indexing function, selecting the right seasonal factor to be used. The index however is time-dependent, but there is no such information involved at the moment. We proceed introducing the exogenous variable x_t to provide temporal data. The exogenous variable could be either an integer or a calendar information, e.g. a week or a date. Therefore $\omega_m(\cdot)$ can be written as

$$\omega_m(x_t, \mathbf{c}) = \begin{cases} \mathbf{e}^{\mathrm{T}}_{(x_t \mod m)}\mathbf{c} & \text{if} \quad x_t \in \mathrm{Z} \\ \mathbf{e}^{\mathrm{T}}_m \mathbf{c} & \text{if} \quad x_t \in \mathrm{Z} \wedge (x_t \mod m) = 0 \\ \omega_m(toIndex(x_t), \mathbf{c}) & \text{otherwise} \end{cases} \tag{5.2}$$

where $\mathbf{e}_j \in \mathbb{R}^m$ is the base vector with the j-th element - $j = 1, \ldots, m$ - set to 1, and *toIndex*$(\cdot)$ is any function that maps a calendar information to an index, e.g. *dayOfWeek*$(\cdot)$. Similarly we can write the inverse indexing function

$$\omega_m^{-1}(x_t, s) = \begin{cases} \mathbf{e}_{(x_t \mod m)} s & \text{if} \quad x_t \in \mathrm{Z} \\ \mathbf{e}_m s & \text{if} \quad x_t \in \mathrm{Z} \wedge (x_t \mod m) = 0 \\ \omega_m^{-1}(toIndex(x_t), s) & \text{otherwise} \end{cases} \tag{5.3}$$

The system of equations 5.1 can consequently be rewritten as

5.2.2 Operators

Inspecting the Holt-Winter model we can recognize two main operations.

Scaling Either additive or multiplicative this operator is responsible for the extraction of the seasonal component from the input values, such that the output one is purged by seasonal effects. In Eq. (5.1) we have three of such operations, one for the level, one for the seasonality and one for the output value. When also the trend is multiplicative, we would have one additional operation.
Applying inversely the scaling operation, seasonal effects previously removed are restored. This is the case for $\hat{z}_{t+\mathrm{H}|t}$ for example.

$$S(z,s) = \frac{z}{s}, \tag{5.4}$$

$$S^{-1}(z,s) = z \cdot s, \tag{5.5}$$

where z and s are a generic input and a generic seasonal factor respectively.

Smoothing The three structural components are governed by an exponential smoothing process accepting different inputs, but sharing the same structure

$$\begin{aligned} \langle new\_value \rangle = \langle smoothing\_factor \rangle \quad &\cdot \quad \langle input \rangle + \\ \left(1 - \langle smoothing\_factor \rangle\right) \quad &\cdot \quad \langle previous\_value \rangle . \end{aligned}$$

Formally

$$ES(\alpha, z, v_{-1}) = \alpha z + (1-\alpha)\, v_{-1}. \tag{5.6}$$

where again α, z and v had been used to identify generic function's inputs. The number of smoothing operators is directly related to the number of structural components.

Using the so far introduced operations and vectors, Eq. (5.1) becomes

$$\tilde{z}_t = z_t \tag{5.7}$$

$$l_t = ES\big(\alpha, \quad S_\alpha\left(\tilde{z}_{t-1}, \omega_m\left(x_t, \mathbf{c}_{t-1}\right)\right), \quad l_{t-1}\big) \tag{5.8}$$

$$b_t = ES\big(\beta, \left(l_t - l_{t-1}\right), b_{t-1}\big) \tag{5.9}$$

$$\hat{z}_t = S^{-1}\left[\left(l_t + t_t\right), \omega_m\left(x_t, \mathbf{c}_{t-1}\right)\right] \tag{5.10}$$

$$\mathbf{c}_t = \mathbf{c}_{t-1} \oplus \omega_m^{-1}\left[x_t, \quad ES\Big(\gamma, \quad S_\gamma\left(\tilde{z}_{t-1}, \left(l_{t-1} + t_{t-1}\right)\right), \quad \omega_m\left(x_t, \mathbf{c}_{t-1}\right)\Big)\right] \tag{5.11}$$

where $\oplus$ is the element-wise addition.

Three gates are distinguished inside the LSTM cell in Fig. 5.1: a forget gate, an input gate and an output gate. Each gate is composed by one or more operations to fulfill its task. The LSTM's input gate filters the combined information from both the current input $\mathbf{x}_t$ and the state $\mathbf{h}_{t-1}$ to then update the current context vector $\mathbf{c}_{t-1}$, producing the new $\mathbf{c}_t$. Looking at the above set of equations, Eq. (5.11) proposes a similar behaviour. Indeed the context's value selected via $\omega_m(\cdot)$ is updated through a smoothing operation, driven by the factor γ, dependent on the previous state's values.

Equations (5.7)–(5.10), grouped together, act similarly to an output gate. The latter's scope is to establish how much the current cell should influence the output. Taking information coming from both the state vector and the context vector, it updates the internal representation of the model which is finally passed to the next time step. Simultaneously, the current prediction is emitted.

The last one is the forget state. Its job is to decide if the context vector should be retained for the current iteration or flushed. No operations could be related, in our opinion, to a potential forget gate except if, in the current setting, we think about Eq. (5.11) as a blending between a forget and input gate.

The same reflections hereby exposed can be applied to any method belonging to the ES family. To cite an instance, a SES can be derived: initializing t_1 either at zero or one - depending if the trend is additive or multiplicative - and setting $\beta = 0$ (to always propagate the initial value); initializing all the factors s_t either at zero or one - depending again if we are dealing with an additive or multiplicative seasonality - and setting $\gamma = 0$ (to always propagate the initial value). Figure 5.2 depicts the "cell" characterized thus far.

The analogy helped the explanation of the various component contained in our model and track the data flow. However, we restrained ourselves from calling it a proper cell due to how we treat and pass the model parameters. Without lowering into implementation details, to instantiate a common RNN cell we will specify some hyperparameters, like the dimension of the weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$. Once the cell had been initialized, its parameters are held within the cell itself. Our approach follows more a functional design, passing ex-

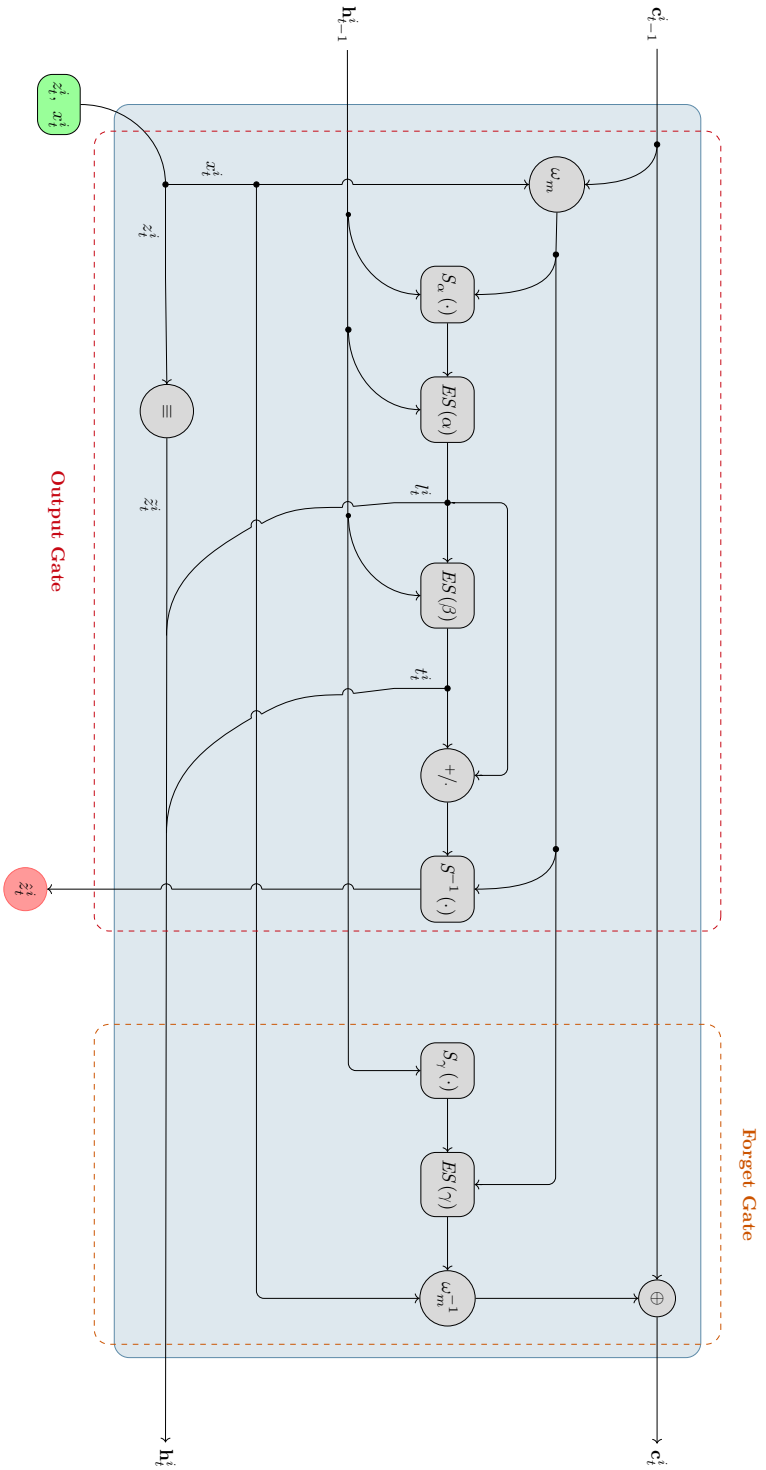


Figure 5.2: Intersecting edges represent a concatenation or any gathering function of the input. The ● specify a "copy" of the input, e.g. when the input is fed to multiple branches, or a "split". The ≡ is the identity operator. The ⊕ is an element-wise update operation of the context vector. The fictional gates are also highlighted. The fictional gates are also highlighted.

ternally all the parameters. This is aligned with the Differentiable Programming paradigm and the company's DSL approach. Parameters inspection and modification are also easier to accomplish in this context.

5.3 Model

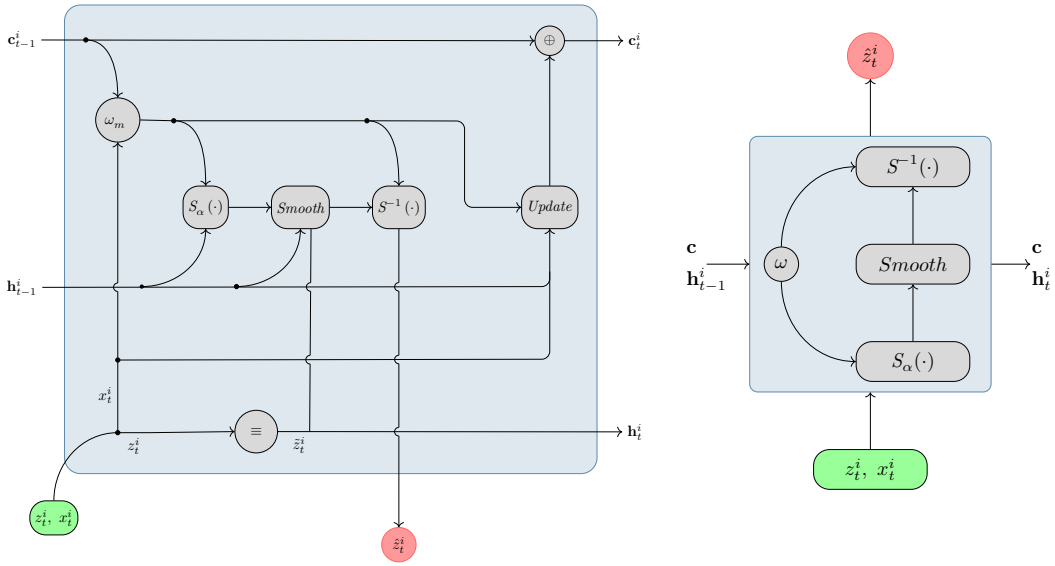


Figure 5.3: (Left) Compressed view of the fictional cell shown in Fig. 5.2. The two smoothing operations for level and trend respectively had been unified under the *Smooth* block. Seasonal related operations had been gathered under the *Update* block. (Right) When the seasonality is shared the context is no more updated at single time series level, hence the *Update* block is factored out.

The fictional cell outlined in Fig. 5.2 is equivalent to a single time series model. A more compact visualization is shown in Fig. 5.3 (Left), where the level and trend smoothing processes had been unified under the *Smooth* block, while the ensemble of seasonal related operations had been collapsed into the *Update* block. Given the dataset $\mathbb{D} = \left\{\mathbf{z}^i_{t_0:\mathrm{T}}\right\}_{i=1}^{\mathrm{N}}$ the i-th cell will be associated with the i-th time series as well as its set of parameters $\Theta^i = \{\mathbf{A}^i, \mathbf{G}^i\}$. The latter will be made of: smoothing factors $\mathbf{A}^i = \{\alpha^i, \beta^i, \gamma^i\}$; seasonal factors

$$\begin{aligned}\mathbf{G}^i &= \left\{ s_1^{(i,g_1)}, \ldots, s_{m_{g_1}}^{(i,g_1)}, s_1^{(i,g_2)}, s_{m_{g_2}}^{(i,g_2)}, \ldots, s_{m_{g_{n_g}}}^{\left(i,g_{n_g}\right)} \right\} \\ &= \left\{ \mathbf{s}^{i,g_1}_{1:m_{g_1}}, \mathbf{s}^{i,g_2}_{1:m_{g_2}}, \ldots, \mathbf{s}^{i,g_1}_{1:m_{g_1}} \right\};\end{aligned}$$

or any intersection of them, depending on the structure of the model - e.g. SES, DES, Holt–

Winter - and the number of seasonality n_g required. For instance, a model with no trend requesting a Week-of-Year seasonality in addition to an Hour-of-Day one will receive a set Θ^i

$$\Theta^i = \left\{\alpha^i, \gamma^i, \mathbf{s}_{1:24}^{(i,\mathrm{HoD})}, \mathbf{s}_{1:52}^{(i,\mathrm{WoY})}\right\}.$$

The aggregation of all Θ^i, make the whole set Θ of our model parameters. Exogenous variables $\mathbf{x}_{t_0:\mathrm{T}}^i$ are supposed to be known over the whole i-th time series' history. They can be item-dependent, time-dependent or both. In general any co-variate $\mathbf{x}^i$ which does not vary with time can be made time-dependent, repeating it along the time dimension. This distinction has more a practical importance rather than theoretical one, for instance when the co-variate represent a categorization of the i-th item. Both DeepAR [SFGJ19] and DeepSSM [RSG+18] incorporate temporal and categorical information into the exogenous variables in a similar manner.

5.3.1 Parameters, encoding & initialization

As already discussed in Sec. 3.3.2 for DeepSSM [RSG+18], constraints are cast upon the sets $\mathbf{A}^i$ and $\mathbf{G}^i$.

Smoothing factors Historically any smoothing factor in the ES framework has been expected to lie in the range $[0,1]$. In Hyndman et al. [HA21] different boundaries are reported for β and γ, specifically: $0 \leq \beta^* \leq 1$, $\beta = \alpha\beta^*$; $0 \leq \gamma \leq (1-\alpha)$. These boundaries ease the computation of the exponential smoothing methods in the state space form, but we found that the ordinary ranges work well in practice. For each generic smoothing factor $\alpha^i \in \mathbf{A}$, we introduced the quantity $\tilde{\alpha}^i$ such that:

$$\alpha^i = (ub - lb)\frac{1}{1+\exp\left(-\tilde{\alpha}^i\right)} + lb.$$

The *l*ower *b*ound lb and *u*pper *b*ound ub are set to 0.05 and 0.95, respectively.

No specific mechanisms had been put in place for the initialization of this class of parameters. Empirically $\alpha^i = 0.5$, $\beta^i = 0.3$ and $\gamma^i = 0.1$ worked well as initial guess. At large, any value ≤ 0.5 is acceptable in order to have a certain dynamic within the model, without pressuring it to use information coming from the observation rather than its previous condition. The latter represents also the business-wise value of this family of parameters, discerning if the past values or the current one describe better the time series behaviour.

Seasonal factors In theory, the seasonality is expected to be a stationary repeating pattern. Empirically we expect an additive seasonality to sum up to zero, or sum up to the period m if multiplicative. To control the magnitude of the patterns and their shape, we again employed some proxies. For any seasonality $\tilde{\mathbf{s}}^{i,g}_{1:m_g} \in \mathbf{G}$ we have:

Additive

$$s^{i,g}_t = \tilde{s}^{i,g}_t - \bar{s}^{i,g}, \quad t = 1, \dots, m_g;$$

with $\bar{s}^{i,g} = 1/m_g \sum_{t=1}^{m_g} s^{i,g}_t$ average of the seasonal factors.

If there is enough data available, seasonal factors can be initialized averaging all the observations for a given time instant within the period. For example, having 4 full years of a monthly time series - $m_g = 12$ -

$$\begin{aligned} s^{\text{MoY}}_1 &= \frac{1}{4}\left(z_1 + z_{13} + z_{25} + z_{37}\right), \\ s^{\text{MoY}}_2 &= \frac{1}{4}\left(z_2 + z_{14} + z_{26} + z_{38}\right), \\ &\dots \\ s^{\text{MoY}}_{12} &= \frac{1}{4}\left(z_{12} + z_{24} + z_{36} + z_{48}\right). \end{aligned}$$

Such a method can be cumbersome for huge datasets. To initialize the factors we used a sinusoidal function with a wavelength proportional to the period m_g. Technically, any harmonic function which can sum up to zero over $\left[1, m_g\right]$ can be used to initialize the parameter, e.g. a sawtooth function.

Multiplicative

$$s^{i,g}_t = \log\left[1 + \exp\left(\frac{\tilde{s}^{i,g}_t m_g}{\bar{s}^{i,g}}\right)\right], \quad t = 1, \dots, m_g;$$

where $\bar{s}^{i,g} = \sum_{t=1}^{m_g} s^{i,g}_t$ sum of the seasonal factors. In this case the initialization is quite straightforward, with each and every factor taking the value $1/m_g$.

Factors normalization is not mandatory but we found that it helped when dealing with intermittent data. Such constraints help the training process into shaping the seasonality profile. We also found that additive factors initialized with a sinusoidal hardly exited - or completely failed to exit - such regime, thus we preferred a sawtooth function.

5.3.2 Multiple seasonality

Contrary to the standard ES methods, from which the model is derived, it has room to accommodate multiple seasonality.

For sake of clarity we limited the function ω_m and the context vector $\mathbf{c}_*$ to the single period m. However the respective definition can be generalized to a multiple seasonality scenario. Assume $\mathbf{s}^{i,g} = \left\{s_1^{(i,g)}, s_2^{(i,g)}, \ldots, s_{m_g}^{(i,g)}\right\}$ the seasonal factors vector belonging to seasonality g - e.g. hourly, weekly, monthly - with period m_g, e.g. 24, 7, 12. The context vector $\mathbf{c}_*$ could either become a context matrix $\mathbf{C}^i_{[*,(\mathrm{M}_g \times n_g)]} = \left[\mathbf{s}^{(i,g_1)\mathrm{T}}, \mathbf{s}^{(i,g_2)\mathrm{T}}, \ldots, \mathbf{s}^{\left(i,g_{n_g}\right)\mathrm{T}}\right]$, with $\mathrm{M}_g = \max\left\{m_{g_1}, \ldots, m_{g_{n_g}}\right\}$, or an extended version

$$\mathbf{c}^i_* = \left\{s_1^{(i,g_1)}, \ldots, s_{m_{g_1}}^{(i,g_1)}, s_1^{(i,g_2)}, \ldots, s_{m_{g_2}}^{(i,g_2)} \ldots, s_{m_{n_g}}^{\left(i,g_{n_g}\right)}\right\}.$$

Accordingly $\omega_{m_{g_*}}$ is going to select either a column of $\mathbf{C}^i_*$ or the starting offset within $\mathbf{c}^i_*$, before applying the indexing. Consequently the number of forget (input) gates in our cell will linearly increase with the number of seasonality.

Comparing with the SotA models, we can not exclude that the latter are able to model multiple seasonality. Even so the architecture design does not allow for a discrimination of the single seasonality. Both DeepAR [SFGJ19] and DeepSSM [RSG+18] delegate the approximation of trend and seasonality - and complex components at large - to the global LSTM. Accessing the recurrent network weights is not enough to extract the desired information. N-Beats [ODPT21], on the other hand, claims interpretability and make no use of temporal data. It is true that the interpretable architecture can output a seasonality and a trend component, nonetheless there seem to be no way to isolate the different seasonality - if present - from the Fourier series. In our case instead each and every seasonality is directly mapped to a sub set of the model's parameters and possesses a real business significance.

5.3.3 Shared seasonality

Having shown how multiple seasonality can be contained within the context array, we can also imagine to pass the same collection of seasonal factors to related time series, actively learning them jointly. Either if the context vector had been transformed into a context matrix or an extended version of itself, we can drop the i-th index to specify shared factors

$$\mathbf{C}_{*,(\mathrm{M}_g \times n_g)} = \left[\mathbf{s}^{(g_1)\mathrm{T}}, \mathbf{s}^{(g_2)\mathrm{T}}, \ldots, \mathbf{s}^{\left(g_{n_g}\right)\mathrm{T}}\right],$$

$$\mathbf{c}_* = \left\{s_1^{(g_1)}, \ldots, s_{m_{g_1}}^{(g_1)}, s_1^{(g_2)}, \ldots, s_{m_{g_2}}^{g_2} \ldots, s_{m_{g_{n_g}}}^{\left(g_{n_g}\right)}\right\}.$$

If the i-th cell expect a shared seasonality, the corresponding forget/input gate for the seasonal component can be factored out - Fig. 5.3 (Right) -, leaving only to the training process the responsibility to shape the seasonality. This distinction is critical. Dealing with seasonality at time series level, data is scarce and the smoothing process support the training one in outlining seasonality. Conversely, when we are learning the seasonality jointly, the amount of data available is increased and we can trust more the training operation.

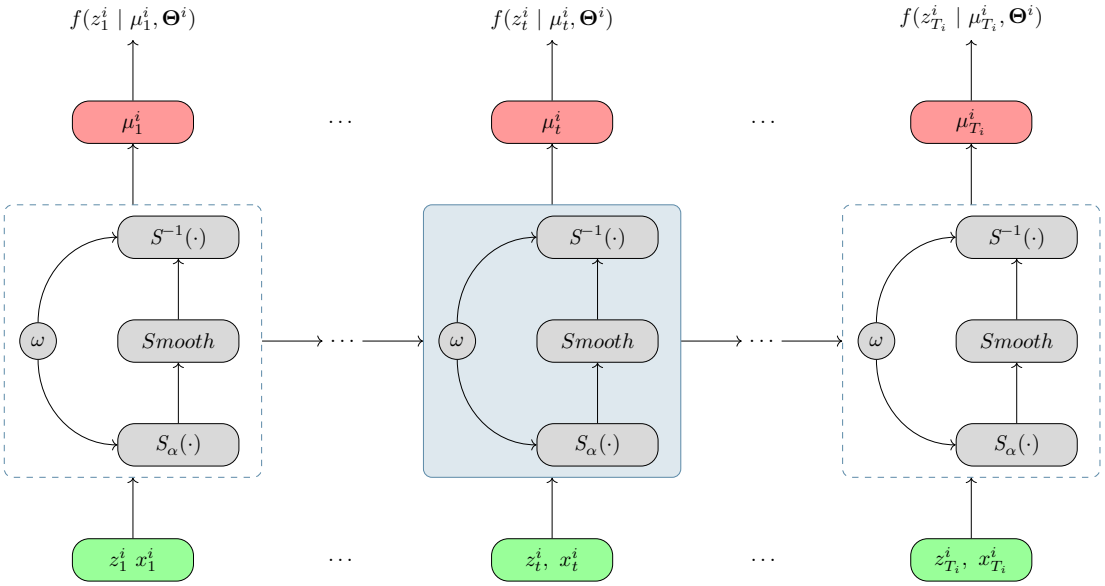


Figure 5.4: Summary of the model. Observations, covariates and seasonal factors are passed to the scaling operator $s(\cdot)$ to produce seasonal-free time series. The latter are passed to the smoothing block ses_t^i. Seasonal effects are applied back by mean of $s^{-1}(\cdot)$. A sharing seasonality scenario is depicted for visualization clearness.

5.3.4 Likelihood model

For the current work we always used the Negative Binomial distribution - please refer to Sec. 3.1.2 - and the corresponding likelihood and log-likelihood. We also employed the distribution parameterization proposed by Snyder et al. [SOB12] based on the mean μ and the paremeter η. For the i-th time series we have

$$p^i = \frac{\eta^i}{1+\eta^i};$$
$$r_t^i = \frac{\left(1-p^i\right)\mu_t^i}{p^i} = \frac{1}{1+\eta^i}\cdot\mu_t^i\cdot\frac{1+\eta^i}{\eta^i} = \frac{\mu_t^i}{\eta^i};$$

$$f\left(z_t^i|\mu_t^i,\eta^i\right) = \frac{\Gamma\left(z_t^i+\mu_t^i/\eta^i\right)}{z_t^i!\Gamma\left(\mu_t^i/\eta^i\right)}\left(\frac{1}{1+\eta^i}\right)^{z_t^i}\left(\frac{\eta^i}{1+\eta^i}\right)^{\mu_t^i/\eta^i}$$

$$\ell_{\mathrm{NB}}\left(\mathbf{r}_{1:\mathrm{T}_i}^i, p^i|\mathbf{z}_{1:\mathrm{T}_i}^i\right) = \sum_{t=1}^{\mathrm{T}_i} + \ln\Gamma\left(z_t^i+\frac{\mu_t^i}{\eta^i}\right) - \ln\Gamma\left(\frac{\mu_t^i}{\eta^i}\right) - \ln\left(z_t^i!\right)$$
$$+ z_t^i\ln\left(\frac{1}{1+\eta^i}\right) + \frac{\mu_t^i}{\eta^i}\ln\left(\frac{\eta^i}{1+\eta^i}\right) \tag{5.12}$$

The prediction $\hat{z}_t^i$ will serve as time-dependent mean μ_t^i, while η^i is attached to the set Θ^i. The relationship introduced by Snyder forces the Negative Binomial's r parameter to become time-dependent, as it is obvious in the previous equations. We have already seen the same practice when discussing DeepAR and DeepSSM, except that in those cases the time-dependent parameters were emitted from the LSTM. As it was true for the two SotA models, our approach is not limited to the application of the Negative Binomial. Any distribution can be adapted, as long as it is possible to derive log-likelihood's gradients with respect to the parameters and samples from the distribution can be obtained easily.

5.3.5 Training

Model parameters Θ are learnt maximizing the sum of log-likelihood functions in Eq. (5.12), i.e. $\Theta^\star = \arg\max_{\Theta} \mathscr{L}(\Theta)$

$$\mathscr{L}\left(\Theta\right) = \sum_{i=1}^{\mathrm{N}} \ell_{\mathrm{NB}}\left(\Theta^i|\mathbf{z}_{1:\mathrm{T}_i}^i\right). \tag{5.13}$$

Proceeding as depicted in Fig. 5.4 the observations $\mathbf{z}_{1:\mathrm{T}_i}^i$ are fed to the cell and forwarded to the next step. The co-variates $\mathbf{x}_{1:\mathrm{T}_i}^i$ helps in the seasonal factors selection, the latter are then applied to the $\tilde{\mathbf{z}}_{1:\mathrm{T}_i}^i$ to obtain

$$\bar{\mathbf{z}}_{1:\mathrm{T}_i}^i = S_\alpha(\tilde{\mathbf{z}}_{1:\mathrm{T}_i}^i, \mathbf{s}_{1:m_g}^i);$$

a version of the original series purged by seasonal effects. The majority of the work is done

by the smoothing operations. The latter output level and trend information, which are stored into the new state $\mathbf{h}^i_{1:T_i}$. Level and trend are combined and re-scaled, resulting in the prediction $\hat{\mathbf{z}}^i_{1:T_i}$ which also serves as the time-varying mean

$$\mu^i_{1:T_i} = S^{-1}(\mathbf{l}^i_{1:T_i}, \mathbf{t}^i_{1:T_i}, \mathbf{s}^i_{1:m_g});$$

for the stochastic process describing the current time series. If there is a local seasonality, the context will be updated by a smoothing operation. Otherwise the training process will learn them jointly and average them over all the time series forming the dataset. From mean $\mu^i_{1:T_i}$ and the set Θ^i, distribution's parameters are derived and used to compute the likelihood $f\left(\mathbf{z}^i_{1:T_i} \mid \mu^i_{1:T_i}, \Theta^i\right)$.

5.3.6 Prediction

Once all the observations have been consumed and the parameters Θ have been learnt, we can continue estimating the forecast distribution in Eq. (2.10) via Monte Carlo simulation. For each time series we start from the $\mu^i_{T_i}$, generate a new sample

$$\hat{z}^i_{T_i} \sim \mathrm{NB}(\mu^i_t, \Theta^i)$$

and pass it through the same scaling, smoothing, re-scaling steps seen during training to obtain the mean $\mu^i_{T_{i+1}}$ for the successive step. The process - depicted in Fig. 5.5 - is repeated over the whole prediction horizon, for all the time series, until K trajectories are generated for each of them. The higher K is, the more precise would be the distribution approximation. From the trajectory collection, we extract the median quantile ($\rho = 0.5$) to serve as point forecast of the model. From the same collection, other quantiles could be derived for the interval prediction.

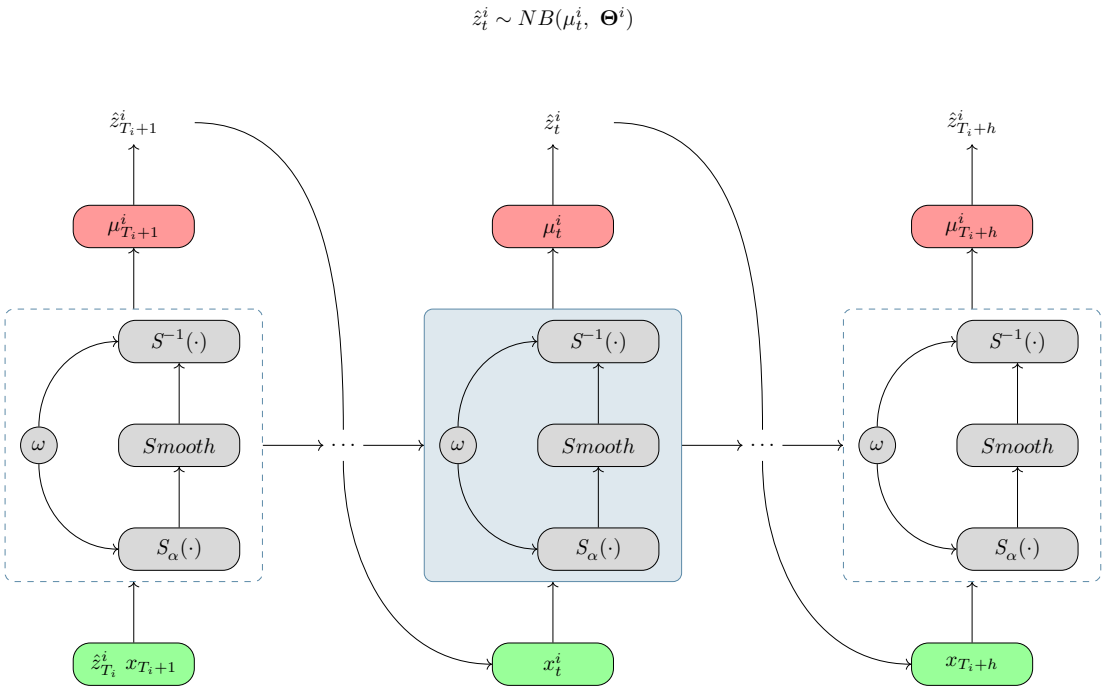


Figure 5.5: Illustration of the sampling process. For each time step $t \in [T_{i+1}, \ldots, T_{i+h}]$ a new sample $\hat{z}_t^i$ is drawn from the distribution characterized by μ_t^i and θ^i; the smoothing process is carried over, producing a new mean μ_{t+1}^i which is further used to generate the next input, and so on until the whole time range is covered. The process is repeated multiple times to better approximate the true conditional distribution.

5.4 Results

We ran our experiments locally, on commercial laptops with no peculiar specifications as shown in Tab. 5.2. Even though the platforms were equipped with GPUs, we restrained the experimentation to models running on CPUs. The ongoing climate change crisis and the energy transition, together with policies required to face the issues, are pushing researchers to question the impact of training AI models and the amount of resources they need to run. Manufacturers are pressured to either asses their energy efficiency [1] and low carbon footprint or design new chips to accomplish it. Conversely AI SotA models are producing promising results in various fields, escalating resource requests at an exponential rate [XZF+21]. The majority of the demand is related to Graphic Processing Units (GPUs). The latter have a great power consumption compared to Central Processing Units (CPUs), but it is commonly justified by the gain in performances, which reduces the time spent to complete a job. Nevertheless most of the tests, in our humble opinion, appear to not take into account all possible CPU scenarios - or are vague about them - running tests in the conventional sequential mode. An additional degree of awareness is indeed required to take advantage of CPU parallelism, either data or thread or both. Opposite to GPUs which exploit data and thread parallelism by design, supported by dedicated kernels. The choice between GPUs and CPUs is not relegated only to energy consumption and performances, it affects the business expenses too. Computational resources are offered at different costs, depending if the customer either opts for an on-demand plan, or reserves a whole instance or chooses a spot instance. The average price for GPU instances ranges roughly from the 13$ per hour for an on-demand plan to the 2$ per hour when an instance is reserved. The price drops between 5 and ten times for the single CPUs. Thus being efficient on CPUs, while not loosing accuracy, can cut the energy consumption, emission and expenses. To cite an instance this is what we observed for the DeepAR model tested on the parts dataset. The original paper used p2.xlarge AWS EC2 instance with 4 CPUs and 1 NVIDIA Tesla K80 GPU and claimed a 5 minutes running time on this machine. Properly setting the environment, we replicated both the accuracy and the running time on the commercial machines listed in the table.

In the coming sections, to assess the model accuracies, we used the ρ-risk metric (Sec. 3.1.6). The median risk, 0.5-risk, is equivalent to the $p50$ value reported in the DeepSSM research paper and to the ND noted in the N-Beats work. The DeepAR and DeepSSM open-source implementations are part of the GluonTS [ABBS+20] repository [2], the corresponding com-

[1] What's Up? Watts Down – More Science, Less Energy: https://blogs.nvidia.com/blog/2023/05/21/gpu-energy-efficiency-nersc/

[2] GluonTS repository: https://github.com/awslabs/gluonts/

Table 5.2: Benchmark hardware specifitions.

	Laptop 1	Laptop 2
CPU	i7-10750H	i7-10850H
RAM (GB)	16	32
GPU	NVIDIA GeForce RTX 2060	NVIDIA Quadro P620

mercial version is accessible through Amazon SageMaker [3]. The original N-Beats code is available on the corresponding repository [4], while the one including the parts dataset and used for the current work is available on a public fork [5]. All the models had been trained using the ADAM optimizer.

To develop our model outside the company technological stack, we had chosen Julia. Julia is a fair new programming language - the first release dates back to 2012 - which aims at providing a single environment productive enough for prototyping, efficient enough for deploying performance-intensive applications. It is a flexible dynamic language performance comparable to traditional statically-typed languages. Both flexibility and performance are coming from *j*ust-*i*n-*t*ime (JIT) compilation (and optionally ahead-of-time compilation), implemented using LLVM. It is multi-paradigm, combining features of imperative, functional, and object-oriented programming. Two choices are available in Julia to achieve automatic differentiation: Zygote.jl [Inn19] and Enzyme [MC20]. Zygote is an open-source library which extends Julia's dynamic semantics with syntactic AD transformations. Contrary, Enzyme is a cross-language solution which works on the LLVM intermediate representation. It could be virtually applied to all LLVM compiled languages, like C++ and Rust. Thus the code produced using Julia in combination with Enzyme can exploit all the optimization at disposal of the underlying compiler. Both Zygote and Enzyme impose no restrictions to the class of functions which can be differentiated and introduce no sub-languages, allowing to specify the model in plain Julia. That being said we have chosen Enzyme for our work. Finally, thanks to Julia's distributed features, we were able to accomplish without (so many) troubles data parallelism on CPUs. A repository [6] for this thesis work is publicly available.

Parts dataset

For the DeepAR and DeepSSM architectures we used the same hyperparameters found in the respective works, and reported in table Tab. 5.3 for convenience. Settings for the N-

[3] Amazon SageMaker website: https://aws.amazon.com/it/sagemaker/
[4] N-Beats repository: https://github.com/ServiceNow/N-BEATS
[5] N-Beats fork (with parts dataset experiments): https://github.com/acifonelli/N-BEATS
[6] Probabilistic Exponential Smoothing repository: https://github.com/acifonelli/ES-AD

Beats architectures had been found via Grid Search, since the dataset was not included in the original paper. Only hyperparameters for the best performing global (N-Beats-G) and interpretable (N-Beats-I) models are reported in Tab. 5.3. As already mentioned while introducing the dataset, the data represent monthly aligned time seris. For each and every time series, we had to forecat the last year of data, i.e. H = 12.

Table 5.3: Parts: State-of-the-Art models hyperparameters.

		DeepAR	DeepState	N-Beats-G	N-Beats-I
	Batch Size	64		512	
	Learning Rate	1e-3		1e-3	
	Epochs	80	100	–	–
DeepAR	#Cells	40			
	#Layers	3			
	Embedding Dim.	1046	1		
	Encorder Length	12			
	Decoder Length	12			
DeepSSM	Latent Dimension	12			
N-Beats	Repetition	1			
	Losses	MAPE / MASE / SMAPE			
	Lookback	[H, 2H, 3H, 4H]			
	FC Size	512			
	FC Layers	3			
	#Blocks	30			
	Seasonal FC Size				512
	Seasonal FC Layers				3
	Seasonal #Blocks				3
	Trend FC Size				256
	Trend FC Layers				3
	Trend #Blocks				3
	Polynomial Degree				3

All the Fully Connected quantities refer to the first FC sub-element in Fig. 3.10, the one preceding the backcast/forecast fork. *Repetition* tells how many times the training is repeated, for all the losses specified, over all the lookback requested. In our experiments we trained a total of 12 models (3 metrics · 4 lookbacks · 1 repetition). This condition, i.e. when only one repetition is demanded, is also called *small ensemble* by the authors. The final accuracy is computed on the ensemble average. Both DeepAR and N-Beats used early stopping over training. The forget bias for the LSTM in the DeepAR work had been set at 1. No further

information are available for the DeepSSM model.
We used a $5e-3$ learning rate and trained on the full dataset for 481 epochs. Contrary to the previous models, we didn't use any hyperparameter. In the worst case scenario, i.e. time series-wise seasonality, we will use: 3'138 smoothing factors ($3 \cdot 1'046$), 1'046 θ, 12'552 seasonal factors ($12 \cdot 1046$) to model the monthly seasonality. The number of parameters will therefore sum to 16'736. However we don't need all of them. The intermittent data do not present any sign of trend, hence we can get rid of the $[\beta][][i]$ factors. Modelling seasonality at singular time series level with a scarcity of data will be of no use and will not align us with the other models. Hence we learn jointly a single seasonal monthly profile. The latter remark alone removes the γ^i factors and reduces the parameters toll to 1'058 parameters.

Table 5.4 and Tab. 5.5 show the model performances and the training time on the benchmark platforms respectively. Our approach is able to achieve State-of-the-Art performances while being the fastest, among the tested models, to train.

Table 5.4: Parts: Accuracy metrics over the horizon H = 12. The lower, the better. **Bold** - best performing model. Red - second best performing.

	Rangapuram et al. [RSG+18]						
	ets	auto.arima	DeepAR	DeepState	N-Beats-G	N-Beats-I	PES
$p50$	1.639	1.6444	1.273	1.470	1.19	1.18	**1.087**
$p90$	1.0086	1.0664	1.086	0.935	—	—	**0.432**

Table 5.5: Parts: Average training time on the benchmark machines in Tab. 5.2.

	DeepAR	DeepState	PES	N-Beats-G	N-Beats-I
Time	~ 5 min.		< 1 min.	~ 1 min. 30 sec.	

The N-Beats architecture can not take advantage of its pure Deep Learning design in this context. Notably, the architecture is still able to perform better than the other two SotA models. Even if by a negligible amount, the N-Beats-I performs better than its general counterpart. We could imagine that the a priori knowledge imposed on the basis functions by the Fourier series and the polynomial, helped the model coping with the data scarcity.

M4 Hourly

Even though the dataset was part of the DeepSSM research paper, no hyperparameters were reported. We had derived them from the GluonTS implementation, reworking them

if needed to replicate the original results. For the N-Beats architecture, we report them as found in the related research. The time series included in the dataset are not aligned - i.e. T_i can vary - and each of them have to receive a forecast for the last 48, hence H = 48, hours of their history.

Table 5.6: M4: State-of-the-Art models hyperparameters.

		DeepAR	DeepState	N-Beats-G	N-Beats-I
	Batch Size	50		512	
	Learning Rate	1e-3			
	Epochs	100		5000	
	#Cells	40			
	#Layers	2			
	Embedding Dim.	–	–		
DeepAR	Encorder Length	48			
	Decoder Length	48			
DeepSSM	Latent Dimension	31			
N-Beats	Repetition	10			
	Lookback	[2H, 3H, 4H, 5H, 6H, 7H]			
	FC Size	512			
	FC Layers	4			
	#Blocks	30			
	Seasonal FC Size				2048
	Seasonal FC Layers				4
	Seasonal #Blocks				3
	Trend FC Size				256
	Trend FC Layers				4
	Trend #Blocks				3
	Polynomial Degree				2

We used the same learning rate and trained on the full dataset for 1040 epochs. Opposite to the parts dataset, we left the trend in place - as the majority of the time series have a rich history - and asked for two seasonality, Hour-of-Day and Day-of-Week, which we learnt jointly. The number of parameters involved in the sum to 859 parameters: 414 α, 414 β, 24 hourly seasonal factors, 7 day of week seasonal factors.

Table 5.7 and Tab. 5.8 are dedicated to report model performances and training time each. The time reported for the N-Beats architecture refers to the small ensemble mentioned in the introduction. For this experiment the small ensemble is composed by 18 models (3 met-

rics · 6 lookbacks · 1 repetition). The final accuracy is computed on the ensemble average. Nonetheless to faithfully reproduce the paper results we should have used 10 repetitions, for a total of 180 models. In the latter case a bit more than *12 days* are required to train the network.

Table 5.7: M4 Hourly: Accuracy metrics over the horizon H = 48. The lower, the better. **Bold** - best performing model. Red - second best performing.

	Rangapuram et al. [RSG+18], Oreshkin et al. [ODPT21]				
	DeepAR	DeepState	N-Beats-G	N-Beats-I	PES
$p50$	0.090	0.044	**0.023**	0.027	0.19
$p90$	0.030	**0.026**	—	—	0.097

Table 5.8: M4: Average training time on the benchmark machines in Tab. 5.2.

	DeepAR	DeepState	PES	N-Beats-G	N-Beats-I
Time	~ 4 hours 45 min.		~ 45 min.	~ 1 day 5 hours	

Our approach is remarkably fast over the training process compared to all the other models. However it is inferior in performance, between 4 and 8 times, with reference to the best performing architecture. The loss in performance is rooted to the trend component. Most of the time series exposing a trend have also more than one changing points, i.e. points where the direction and slope of the trend changes. The smoothing process introduces a latency in the response, thus a quick rise in the trend will be diluted over the time. We can appreciate as deep architectures benefit from the rich history present in the dataset. The best performing model is the N-Beats one, but contrary to the parts dataset experiment the general architecture has an advantage over the interpretable one.

Chapter 6

Conclusions

In Chapter 1 we talked about the role of AI in the Supply Chain management activities and how the adoption of new technologies encounter a certain degree of inertia within companies from different backgrounds. Costs are the main obstacles. Companies hang to older systems because of the time and money needed to replace them. On average 2.8 years are required from vendor selection to a complete roll-out of the new solution. Time is the second. Sixty percent of the time the implementation of a new solution fails due to missed deadlines, over budget or disappointing outcomes. Finally, switching to a new solution is not only a matter of technologies. Companies need to prepare the staff to use the new system, ensuring that they are engaged with it and the possible new working flow, while running in parallel the older one to not shutdown operations. The human component can not be taken out of the equation; each professional would rely on his or her expertise - and a set of preferred tools - to solve a problem. Three focal points had been highlighted:

- misalignment with the business perfomances;
- need for explainable results;
- lack of trustworthiness in the new solution.

If the previous three points are true at large, dealing with supply chain data is even more demanding:

- embrace uncertainty;
- dealing with integer and positive data;
- lack of observability of the real business target - customer's demand can not be measured directly -;

- data scarcity;
- just-in-time optimization to incorporate unexpected events or mandatory adjustments.

In Chapter 2, the fundamental concepts of time series analysis and time series forecasting had been introduced. Time series analysis tries to apprehend from the data the most important properties of the series. Frequently the task demands knowledge of the application field, thus it could be prone to cognitive bias. Nonetheless it is preparatory for the time series forecasting job, helping in the model selection. Forecasting a single value, e.g. the mean of the time series, takes the name of point forecast. If instead we are interested not only in the quantity itself but also how much it can change, we should prefer a probabilistic forecast task. Classical methods and models, as well as Neural Networks, had been reviewed. Classical but not antiquated since most of them - if not even all - remain undisturbed in the companies' working flow. They follow a white-box approach, i.e. the model "reasoning" and outcome are transparent to the user. Contrarily Neural Networks, and AI models at large, obey to a black-box paradigm. Sequences are treated with specific neural architectures, called Recurrent Neural Networks, which retain a representation of the context seen so far - summarised into a state - and propagate it through time. Recurrent Neural Networks come with several designs - some not treated in this context - but LSTM is by far the most used architecture in practice. Attempts to merge standard methods and AI models are not lacking in the literature.
Demand forecasting and the relative literature is the focus of Chapter 3. Demand time series are characterized by several oddities like being made only by positive integer observations or being arranged into a hierarchy. Sales, the quantity effectively recorded in the time series, are only a proxy for the real value we are interested into, the demand. Unfortunately, demand is a not observable quantity (we can not measure a not satisfied demand). Probabilistic time series forecasting, together with a learning process based on the Maximum Likelihood Estimation, is a preferable choice in this context. The latest State-of-the-Art models had been here introduced and discussed.
Chapter 4 reviewed Automatic Differentiation, a more functional - in the computer science mean - approach to model developing, and its main mode called Reverse, a generalization of the most known backpropagation. Automatic Differentiation is thrusting Differentiable Programming, a paradigm shift which treats, at large, a general puece of code as a sequence of differentiable operations. In this perspective any function, concatenation of functions or algorithms in general can be seen as a model.
With a functional design in mind and attention to practitioners' needs, we introduced our new model in Chapter 5. The Probabilistic Exponential Smoothing model is tailored to the

univariate time series solving a probabilistic forecasting problem. It is grounded on the Exponential Smoothing family, a well known, vastly studied and widely accepted suit of methods which are still one of the standard solutions in the industry. Contrary to its ancestor our model has room for multiple seasonality and can levarage machine learning techniques, like cross-learning, to cope with limitations. Although it is not still able to compete with SotA models in cases where a rich history is accessible, it sets state-of-the-art results on real sparse and intermittent business data. Additionally it is transparent for the practitioner since each and every parameter, from the smoothing factors to the seasonal profiles, have a clear and immediate translation into business values. The functional design allow the parameters inspection and modification to test the same model under different scenarios. Thanks to its probabilistic nature, it can easily cope with the integer constraint and propose confidence intervals to guide decision under uncertain situations. There are of course limitations to the approach. To cite an instance, in case of asymmetric data the cross-learning process, smoothing the extrema, can impact the forecasting at the distribution's tail. Moreover, the smoothing process at trend level seems to not deal well with unexpected rise in the signal, introducing a latency in the model responde and negatively impacting the outcome.

6.1 Future perspectives

A common design involving LSTMs - and RNNs at large - is a stacked architecture. Multiple instances of the LSTM are randomly initialized and composed together. Stacking helps the generalization process and avoid biases in the forecast. In the current setting we could think of stacking as a way to construct a small ensemble, time series wise. However how to aggregate the outcome of such a stack is not clear. Dealing with only a point forecast we can aggregate the outcome of a stacked architecture summing all the responses and taking the average. In the context of probabilistic forecast each and every outcome represents a distribution. For most of the count distribution used in this work there is no properties similar to the sum of Gaussians, thus we should leverage a form of convolution. The latter however is a very complex and cumbersome operation which will swell both training and prediction timing.

Trend seems to be the Achille's heel of the model. The introduction of different trend models alongside the smoothing one, like a piece-wise linear trend, could help the model in achieving better results.

As it is, Probabilistic Exponential Smoothing can not be used immediately to forecast a new time series with no history. However a possible workaround would be to assign to the new series smoothing values α^i, β^i - and γ^i optionally, depending on the context - averaged over those of related time series.

Another interesting enhancement would be the promotion of the α^i parameters from scalar to vector. The latter could be beneficial to further capture dependencies between successive time steps and variations over the time series period.
Despite the future work still required by the solution, we think that it answers most of the critical points raised by the Supply Chain management and can be a valid candidate to solve a wide range of real business problems.

Acknowledgements

Professionally I would like to thank Joannes Vermorel, CEO of Lokad. His vision on Machine Learning and how it should be coupled with the business needs paved the way for this thesis opportunity. I would like to express my gratitude to Prof. Canu, my thesis advisor, for the guidance through this journey. I am also grateful to Prof. Cecilia Zanni-Merk and Prof. Ahlame Douzal, not only for their insightful technical feedbacks but also for their kindness and humanity in the gloomy moments over these years. To all of them I am thankful for the patience.
I would express my appreciation to Gaëtan Delétoille, Vincent Berthoux and Paul Peseux for their contribution with technical insights, bad jokes and some beers.
My appreciation also goes to the Lokad's administrative team, especially to Estelle Dewost, for their hard work. Bureaucracy is not something which France made easy.
At large I would like to acknowledge all the people that in some way, technically or not, contributed directly to this work or indirectly creating a pleasant environment to work.

On the personal side there is only one person I am grateful to and grateful for, Annalisa. My wife. In French *épouse* means "wife"; in English the term *espouse* is synonym of "support", "help". You are my *é(s)pouse*. You are my rock.

Bibliography

[ABBS+20] Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C. Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, Lorenzo Stella, Ali Caner Türkmen, and Yuyang Wang. GluonTS: Probabilistic and Neural Time Series Modeling in Python. *Journal of Machine Learning Research*, 21(116):1–6, 2020.

[Aka73] H Akaike. Information theory and an extension of the maximum likelihood principle. In *2nd Inter. Symp. on Information Theory*, pages 267–281. Akademiai Kiado, 1973.

[AN00] Vassilis Assimakopoulos and K. Nikolopoulos. The theta model: A decomposition approach to forecasting. *International Journal of Forecasting*, 16:521–530, 10 2000.

[Bau74] F. L. Bauer. Computational graphs and rounding error. *SIAM Journal on Numerical Analysis*, 11(1):87–96, 1974.

[BCB15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[BJ76] GEP Box and GM Jenkins. *Time series analysis forecasting and control-Rev*. San Francisco, Calif.(USA) Holden-Day, 1976.

[BPC+11] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[BPRS18] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[CFR+91] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. K Zadeck. Efficiently computing static single assignment form and the control dependence graph. Technical report, Brown University, 1991.

[CGH+17] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 76(1):1–32, 2017.

[Cli71] Clifford. Preliminary sketch of biquaternions. *Proceedings of the London Mathematical Society*, s1-4(1):381–395, 1871.

[Con21] Breandan Considine. Programming tools for intelligent systems, 2021.

[CPI+96] T Czernichow, A Piras, K Imhof, P Caire, Y Jaccard, B Dorizzi, and A Germond. Short term electrical load forecasting with artificial neural networks. *Engineering Intelligent Systems for Electrical Engineering and Communications*, 4(ARTICLE):85–99, 1996.

[Cro72] J. D. Croston. Forecasting and stock control for intermittent demands. *Operational Research Quarterly (1970-1977)*, 23(3):289–303, 1972.

[DGH06] Jan G De Gooijer and Rob J Hyndman. 25 years of time series forecasting. *International journal of forecasting*, 22(3):443–473, 2006.

[DH06] Benjamin Dauvergne and Laurent Hascoët. The data-flow equations of checkpointing in reverse automatic differentiation. In *Computational Science – ICCS 2006*, pages 566–573. Springer Berlin Heidelberg, 2006.

[dLPR17] Bart de Langhe, Stefano Puntoni, and Larrick Richard. Linear thinking in a nonlinear world. *Harvard Business Review*, 95(5–6):130–139, 2017.

[DM97] Pierre Del Moral. Nonlinear filtering: Interacting particle resolution. *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, 325(6):653–658, 1997.

[Fam93] Felix Famoye. Restricted generalized poisson regression model. *Communications in Statistics - Theory and Methods*, 22(5):1335–1354, 1993.

[Fam97] Felix Famoye. Generalized poisson random variate generation. *American Journal of Mathematical and Management Sciences*, 17(3-4):219–237, 1997.

[GA01] David Gay and Alex Aiken. Language support for regions. *SIGPLAN Not.*, 36(5):70–80, may 2001.

[Gay06] David M. Gay. Semiautomatic differentiation for efficient gradient computations. In *Automatic Differentiation: Applications, Theory, and Implementations*, pages 147–158. Springer Berlin Heidelberg, 2006.

[Ger19] Alexis Gerossier. *Short-term forecasting of electricity demand of smart homes and distribution grids*. Theses, Université Paris sciences et lettres, May 2019.

[GGS19] Maria I. Gorinova, Andrew D. Gordon, and Charles Sutton. Probabilistic programming with densities in slicstan: Efficient, flexible, and deterministic. *Proceedings of the ACM on Programming Languages*, 3:35:1–35:30, January 2019.

[GJ80] Clive WJ Granger and Roselyne Joyeux. An introduction to long-memory time series models and fractional differencing. *Journal of time series analysis*, 1(1):15–29, 1980.

[Gri92] Andreas Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1(1):35–54, 1992.

[GW08] Andreas Griewank and Andrea Walther. *Evaluating Derivatives*. Society for Industrial and Applied Mathematics, second edition, 2008.

[H+04] Rob J Hyndman et al. The interaction between trend and seasonality. *International Journal of Forecasting*, 20(4):561–563, 2004.

[H+06] Rob J Hyndman et al. Another look at forecast-accuracy metrics for intermittent demand. *Foresight: The International Journal of Applied Forecasting*, 4(4):43–46, 2006.

[HA21] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts: Melbourne, Australia., 2021. Accessed on 2022-03-09.

[Har84] Andrew C Harvey. A unified view of statistical forecasting procedures. *Journal of forecasting*, 3(3):245–275, 1984.

[Har90] Andrew C Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, 1990.

[HB03] Rob J Hyndman and Baki Billah. Unmasking the theta method. *International Journal of Forecasting*, 19(2):287–290, 2003.

[HGS20] Malo Huard, Rémy Garnier, and Gilles Stoltz. Hierarchical robust aggregation of sales forecasts at aggregated levels in e-commerce, based on exponential smoothing and Holt's linear trend method. working paper or preprint, June 2020.

[HK06] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

[HKOS] Rob Hyndman, Anne B Koehler, J Keith Ord, and Ralph D Snyder. Forecasting with exponential smoothing: the state space approach. https://robjhyndman.com/expsmooth/. Website with supplementary materials for the homonymous book [Online].

[HKOS08] Rob Hyndman, Anne B Koehler, J Keith Ord, and Ralph D Snyder. *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media, 2008.

[HKSG02] Rob J Hyndman, Anne B Koehler, Ralph D Snyder, and Simone Grose. A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of forecasting*, 18(3):439–454, 2002.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[Hol] South Korea Public Holidays. Seoul bike sharing demand data set. https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand. hosted by UCI Machine Learning Repository [Online].

[Hol04] Charles C Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting*, 20(1):5–10, 2004.

[HP13] Laurent Hascoet and Valérie Pascual. The tapenade automatic differentiation tool: Principles, model, and specification. *ACM Trans. Math. Softw.*, 39(3), may 2013.

[HPS01] Henrique Steinherz Hippert, Carlos Eduardo Pedreira, and Reinaldo Castro Souza. Neural networks for short-term load forecasting: A review and evaluation. *IEEE Transactions on power systems*, 16(1):44–55, 2001.

[HS76] P Jeffrey Harrison and Colin F Stevens. Bayesian forecasting. *Journal of the Royal Statistical Society: Series B (Methodological)*, 38(3):205–228, 1976.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[Inn19] Michael Innes. Don't unroll adjoint: Differentiating ssa-form programs, 2019.

[JOP01] Eric Jones, Travis Oliphant, and Pearu Peterson. Scipy: Open source scientific tools for python. *SciPy, Austin, USA. URL http://www. scipy. org*, 01 2001.

[JU97] Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997.

[Kal60] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[KBDOAEA21] Walid Klibi, Mohamed Zied Babai, Yves Ducq, and Haytham Omar Abd El Akher. Basket data-driven approach for omnichannel demand forecasting. working paper or preprint, April 2021.

[KD01] SJ Koopman and J Durbin. *Time Series Analysis by State Space Methods*. Number 019-961199 in Oxford Statiscal Science Series. Oxford University press, 2001.

[KH06] A V Kostenko and R J Hyndman. A note on the categorization of demand patterns. *Journal of the Operational Research Society*, 57(10):1256–1257, 2006.

[LBD+89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied

to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[LC98] Jun S Liu and Rong Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American statistical association*, 93(443):1032–1044, 1998.

[LL17] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[MAC+82] S. Makridakis, A. Andersen, R. Carbone, R. Fildes, M. Hibon, R. Lewandowski, J. Newton, E. Parzen, and R. Winkler. The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, 1(2):111–153, 1982.

[Mak18] Spyros Makridakis. M4 forecasting competition. https://github.com/Mcompetitions/M4-methods, 2018. [Online].

[Mar19] Charles C. Margossian. A review of automatic differentiation and its efficient implementation. *WIREs Data Mining and Knowledge Discovery*, 9(4):e1305, 2019.

[MC20] William Moses and Valentin Churavy. Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12472–12485. Curran Associates, Inc., 2020.

[MCH+93] Spyros Makridakis, Chris Chatfield, MichÃ¨le Hibon, Michael Lawrence, Terence Mills, Keith Ord, and LeRoy F. Simmons. The m2-competition: A real-time judgmentally based forecasting study. *International Journal of Forecasting*, 9(1):5–22, 1993.

[McKa] McKinsey & Company. Global survey - the state of ai in 2020. https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/global-survey-the-state-of-ai-in-2020. McKinsey Analytics [Online].

[McKb] McKinsey & Company. The state of ai in 2022 - and a half decade in review. https://www.

mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2022-and-a-half-decade-in-review. McKinsey Analytics [Online].

[McKc] McKinsey & Company. The state of ai in 2023: Generative ai's breakout year. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year. McKinsey Analytics [Online].

[McKd] McKinsey & Company. To improve your supply chain, modernize your supply-chain it. https://www.mckinsey.com/capabilities/operations/our-insights/to-improve-your-supply-chain-modernize-your-supply-chain-it. McKinsey Analytics [Online].

[MDA15] Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML workshop*, volume 238, 2015.

[MH00] Spyros Makridakis and MichÃ¨le Hibon. The m3-competition: results, conclusions and implications. *International Journal of Forecasting*, 16(4):451–476, 2000.

[MHM79] Spyros Makridakis, Michele Hibon, and Claus Moser. Accuracy of forecasting: An empirical investigation. *Journal of the Royal Statistical Society. Series A (General)*, 142(2):97–145, 1979.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[MSA18a] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4):802–808, 2018.

[MSA18b] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, 13(3):e0194889, 2018.

[Nau] Robert Nau. Regression example - weekly beer sales. https://people.duke.edu/~rnau/Regression_example--weekly_beer_sales.xlsx. Statistical forecasting: notes on regression and time series analysis [Online].

[ODPT21] Boris N Oreshkin, Grzegorz Dudek, Paweł Pełka, and Ekaterina Turkina. N-beats neural network for mid-term electricity load forecasting. *Applied Energy*, 293:116918, 2021.

[Oli07] Travis E. Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.

[Par82] Emanuel Parzen. Ararma models for time series analysis and forecasting. *Journal of Forecasting*, 1(1):67–82, 1982.

[Peg69] C Carl Pegels. Exponential forecasting: Some new variations. *Management Science*, pages 311–315, 1969.

[Pes21] Paul Peseux. Differentiating relational queries. In *VLDB 2021 PhD Workshop*, 2021.

[Que57] Maurice Henry Quenouille. The analysis of multiple time-series. Technical report, London: Griffin, 1957.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[RKGR17] Sam Ransbotham, David Kiron, Philipp Gerbert, and Martin Reeves. Reshaping business with artificial intelligence: Closing the gap between ambition and action. *MIT Sloan Management Review*, 59(1), 2017.

[RLG98] Juan Mario Restrepo, Gary K. Leaf, and Andreas Griewank. Circumventing storage limitations in variational data assimilation studies. *SIAM Journal on Scientific Computing*, 19(5):1586–1605, 1998.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[RSG+18] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. *Advances in neural information processing systems*, 31:7785–7794, 2018.

[RT13] Bahman Rostami Tabar. *ARIMA demand forecasting by aggregation.* Theses, Université Sciences et Technologies - Bordeaux I, December 2013.

[SBC05] A A Syntetos, J E Boylan, and J D Croston. On the categorization of demand patterns. *Journal of the Operational Research Society*, 56(5):495–503, 2005.

[Sch78] Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, pages 461–464, 1978.

[SFGJ19] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 2019.

[SHJ+20] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 180–186, 2020.

[Smy20] Slawek Smyl. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1):75–85, 2020.

[Sny85] RD Snyder. Recursive estimation of dynamic linear models. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 272–276, 1985.

[SOB12] Ralph D. Snyder, J. Keith Ord, and Adrian Beaumont. Forecasting the intermittent demand for slow-moving inventories: A modelling approach. *International Journal of Forecasting*, 28(2):485–496, 2012.

[Sor85] Harols W Sorenson. *Kalman Filtering: Theory and Application.* IEEE Press, 1985.

[SP18] Jeffrey Mark Siskind and Barak A. Pearlmutter. Divide-and-conquer checkpointing for arbitrary programs with no user annotation. *Optimization Methods and Software*, 33(4-6):1288–1330, 2018.

[SSF16] Matthias W Seeger, David Salinas, and Valentin Flunkert. Bayesian intermittent demand forecasting for large inventories. *Advances in Neural Information Processing Systems*, 29, 2016.

[Ste89] John D Sterman. Modeling managerial behavior: Misperceptions of feedback in a dynamic decision making experiment. *Management science*, 35(3):321–339, 1989.

[THT+14] Jean-Baptiste Tristan, Daniel Huang, Joseph Tassarotti, Adam C Pocock, Stephen Green, and Guy L Steele. Augur: Data-parallel probabilistic modeling. *Advances in Neural Information Processing Systems*, 27, 2014.

[TL18] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[Ver00] Arun Verma. An introduction to automatic differentiation. *Current Science*, 78(7):804–807, 2000.

[vMBBL18] Bart van Merrienboer, Olivier Breuleux, Arnaud Bergeron, and Pascal Lamblin. Automatic differentiation in ml: Where we are and where we should be going. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[Wen64] R. E. Wengert. A simple automatic derivative evaluation program. *Commun. ACM*, 7(8):463–464, aug 1964.

[Whi51] Peter Whittle. *Hypothesis Testing in Time Series Analysis*. PhD thesis, Uppsala University, 1951.

[Win60] Peter R Winters. Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3):324–342, 1960.

[XZF+21] Jingjing Xu, Wangchunshu Zhou, Zhiyi Fu, Hao Zhou, and Lei Li. A survey on green deep learning. *CoRR*, abs/2111.05193, 2021.

[Yul27] George Udny Yule. On a method of investigating periodicities disturbed series, with special reference to wolfer's sunspot numbers. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 226(636-646):267–298, 1927.

List of Figures

List of Tables