# Approaching sales forecasting using recurrent neural networks and transformers

Iván Vallés-Pérez[1], Emilio Soria-Olivas[1], Marcelino Martínez-Sober[1],
Antonio J. Serrano-López [*,1], Juan Gómez-Sanchís[1], Fernando Mateo[1]

*Escola Tècnica Superior d'Enginyeria, University of Valencia, Avenida de la Universitat s/n 46100 Burjassot, Valencia, Spain*

## ARTICLE INFO

## ABSTRACT

Accurate and fast demand forecast is one of the hot topics in supply chain for enabling the precise execution of the corresponding downstream processes (inbound and outbound planning, inventory placement, network planning, etc.). We develop three alternatives to tackle the problem of forecasting the customer sales at day/store/item level using deep learning techniques and the *Corporación Favorita* data set, published as part of a *Kaggle* competition. Our empirical results show how good performance can be achieved by using a simple *sequence to sequence* architecture with minimal data preprocessing effort. Additionally, we describe a training trick for making the model more time independent and hence improving generalization over time. The proposed solution achieves a RMSLE of around 0.54, which is competitive with other more specific solutions to the problem proposed in the *Kaggle* competition

## 1. Introduction

We have observed how the retail industry economic activity is moving online. In the last years, e-commerce companies are gaining more and more adepts every day. *E-Marketer* (Cramer-Flood, 2020; Lipsman, 2019) reported consistent year on year growths in number of sales of more than 13% in the last 5 years in the US. The percentage of total sales in the US owned by e-commerce companies increased from 8,9% to 14,5% in 2020 (Cramer-Flood, 2020; Lipsman, 2019).

The continuously increasing demand requires the online industries to constantly improve their supply chain systems. This process entails multiple challenges such as: optimal inventory placement (Graves & Willems, 2008), accurate network expansion (Badri et al., 2017), precise inbound and outbound planning (Kaipia, 2009), etc. One of the most important wires that enables all those improvements is the ability to accurately forecast the customer demand for different products, locations and times (Forslund & Jonsson, 2007).

This paper proposes several alternatives to solve the demand forecast problem using deep learning techniques (Goodfellow et al., 2016). The generalization power of these algorithms enables solving the problem using a single model for all the different locations and products time series, while other algorithms like the *ARIMA* family of models (Hyndman & Athanasopoulos, 2018) only can tackle one product–location time series at a time.

Two approaches are described in this work: a *sequence-to-sequence* (*seq2seq* hereafter) architecture with product and location conditioning and an adapted *transformer* architecture for time series forecasting. The code used in this work has been published in GitHub: https://github.com/ivallesp/cFavorita.

Section 1.1 describes the data set used in this study and the objective to be forecasted. The previous work done in the field of demand forecast and with the *Corporación Favorita* data is described in Section 1.2. Section 2 dives deep into the different forecasting methodologies used, together with the different tricks of each implementation. Finally, Sections 3 and 4 detail the results obtained and summarize the conclusions of this work.

### 1.1. Data

The *Corporación Favorita Grocery Sales* data set has been used to conduct this study (Corporación Favorita, 2018). *Corporación Favorita*, an Ecuadorian company owner of multiple supermarkets across Latin America, released this data set around 2017 as a *Kaggle* competition to challenge the community to forecast their sales. It contains daily sales records for 4400 unique items, in 54 different Ecuadorian stores from January 1st 2013 to August 15th 2017. Additional data provided along with the number of sales are described below.
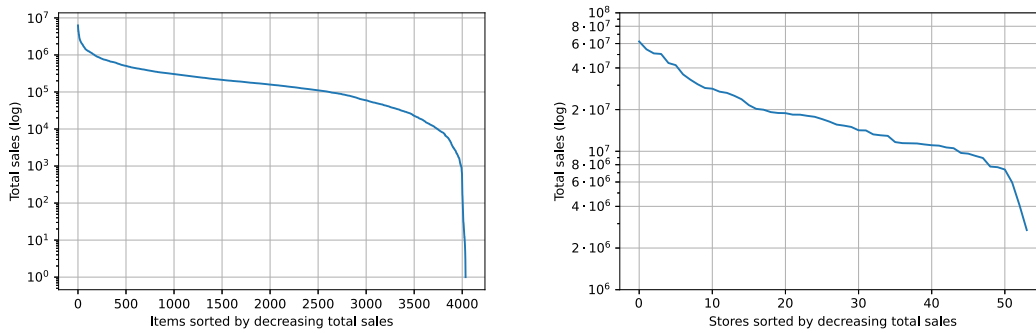
**Fig. 1.** Long tail distributions for items (left) and stores (right). Y axes represent total sells across all the history (around 5 years).
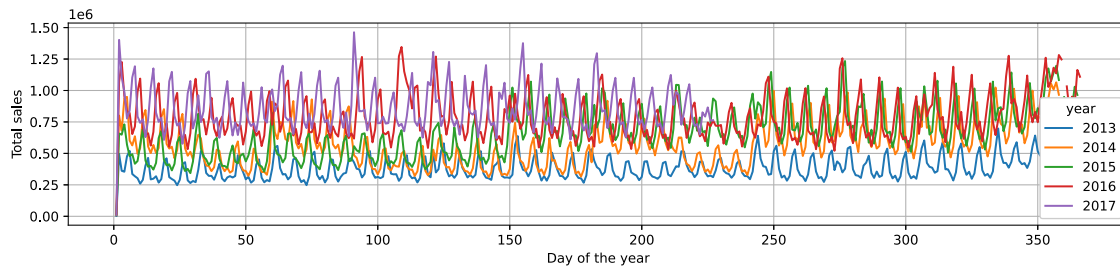


**Fig. 2.** Daily total sales for the 5 years included in the data.

- ID variables: date, store number and item number.
- Promotions: a binary variable indicating if a given item, in a given store at a given time was on promotion.
- Store information: location (city and state) and segment (type and cluster).
- Item information: item family, class, and a binary variable indicating if the item is perishable.
- Transactions: Number of total sales for each store at each date.
- Oil price: price of the oil on each date.
- Holidays and events: dates, locations and types of holidays.

As it is usual in the supply chain organizations, the distribution of sales across products is very uneven (see Fig. 1-left where the top 10% of the products bring the 44% of the sales). The same characteristic is observed in the case of the stores (see Fig. 1-right where the top 10 stores bring 40% of the sales). The sales for some products are very sparse: during the test period, the probability of having one or more sells for a given product is 47.6%, which in practice means that more than half of the days in the time series will have value of zero.

As it can be noticed in Fig. 2, the total sales show clear year on year trends as well as a very remarkable weekly seasonality (see Fig. 3). There is also a peak of sales at the end of each year

The data set does not contain records for items in days when there were zero unit sales. It also lacks information about the available inventory. These two facts together make the forecast effort more complicated, given that when there are zero sales of a given product in a store at a given date, it can be either because there was not available inventory, or because there was inventory but not demand (or both of them at the same time).

Estimating the actual demand of a retailer is not a straightforward task (Deep & Salhi, 2019). In our case, the quantity being forecasted is the number of sales. That would have an important implication in the demand estimation: the number of sales represent the demand as long as there exists available inventory. Hence, the quantity estimated by the machine learning model will be the demand bounded by the inventory $min(demand(i, s, d), inventory(i, s, d))$ (where $i$, $s$ and $d$ are the inventory, the store and the date). There are techniques to correct the demand in these cases (e.g. Bell (2000)). However as the objective of this study is to build the forecast model, it is out of the scope of this paper to deal with that inconvenience.
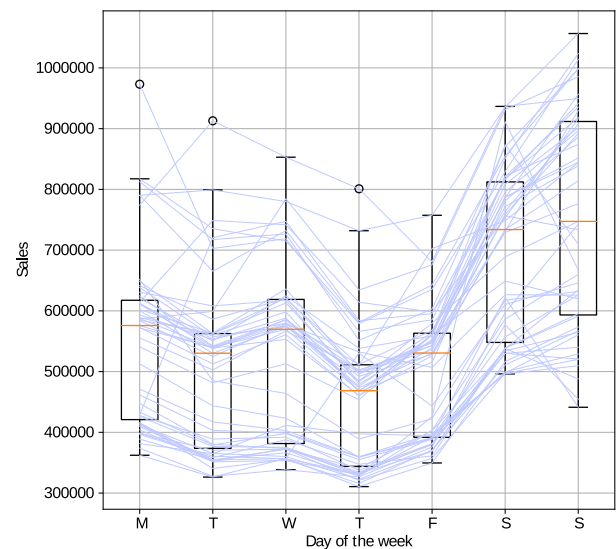


**Fig. 3.** Weekly pattern detail, including the 2014 sales.

### 1.2. Previous work

We have found several works applying deep learning techniques for demand and sales forecast in the supply chain environment. Kilimci et al. (2019) use several *multi-layer perceptrons* (MLP) to build a unified forecast and compare it with more classical techniques. The main disadvantage of this approach is that it requires heavy feature engineering work, as the MLPs are not suitable to deal with time series. In a supply chain problem, this is not practical given the huge quantity of data normally available. In the work by Talupula (2018), different deep learning architectures Convolutional Neural Networks (CNN), Long–Short Term Memory (LSTM) and Multi Layer Perceptron performance are compared over an outbound demand forecast task, using data from a retailer. Similarly, Helmini et al. (2019) compare a deep learning model based on LSTMs with peephole connections with more classical
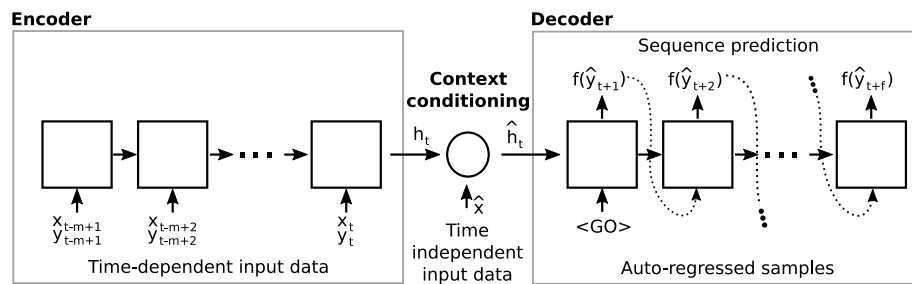
**Fig. 4.** *Seq2seq* architecture diagram. The left box shows the encoder, which takes the historical actual sales $y_i$ as input as well as other exogenous time-dependent features (i.e. oil price, holidays and events, transactions and promotions) denoted by $x_i$, and returns the hidden vector $h_t$ of the last recurrent module as output (aka *context vector*). The middle of the figure shows the context conditioning module, which is our variation proposal over the original sequence to sequence proposal. This module receives the *context vector* $h_t$ from the encoder module and concatenates it with static data $\hat{x}$ producing a conditioned context vector $\hat{h}_t = (h_t, \hat{x})$. Finally, on the right, the decoder module receives as initial state the conditioned context vector and provides the first recurrent cell with a *go symbol* (constant input indicating the input of the decoded sequence). The decoder generates the sequence prediction in an autoregressive way. $f$ is a stack of two fully-connected layers applied to each output of the decoder.
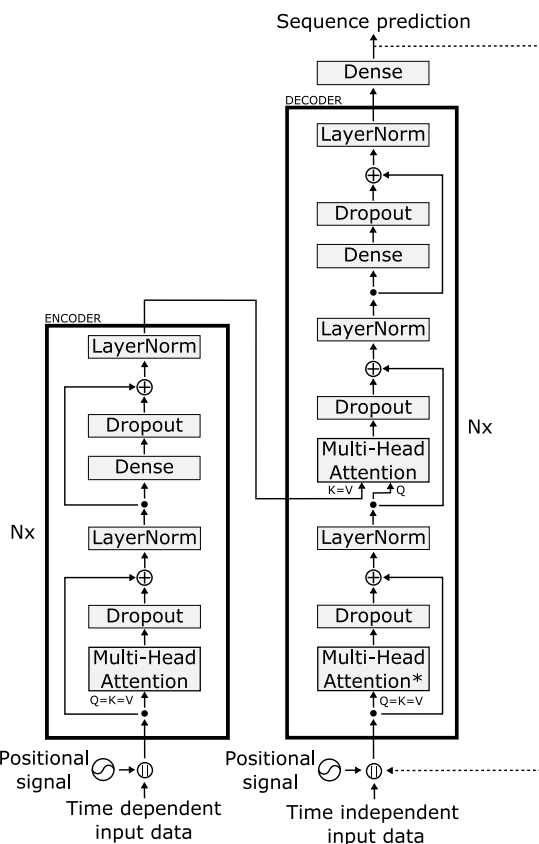


**Fig. 5.** Transformer architecture used to perform time series forecasting. In the diagram, the || symbol stands for the concatenation operation, and similarly as in Vaswani et al. (2017), $N_X$ represents the number of repetitions of the encoder and decoder blocks.

approaches using tree-based models. The difference with our approach is that we propose a set of flexible architectures capable to deal with multi-modal data more efficiently, while the models proposed in these works can only deal with time series. Furthermore, we use sequence-to-sequence modeling, which minimizes the error across all the predicted sequence, while the authors of the aforementioned papers use a next-step prediction auto-regressive model, which only optimizes the error of the next time step.

However, we only found a few reports using this data set for benchmarking. In Kechyn et al. (2018), the authors used the *WaveNet* (van den Oord et al., 2016) architecture to build an autoregressive forecast. They achieved the 2nd position in the Kaggle competition

that published the *Corporación Favorita* data. The authors only show some charts summarizing their results, pointing to Root Mean Squared Weighted Logarithmic Errors (RMSWLE) of around 0.578. However, they do not give many details about the architecture used and the experimental framework. They also do not specify the period of time used to measure the errors. Calero and Caro (2018) used multiple classical techniques (historical average, ARIMA/SARIMA, Snaive and exponential smoothing) to forecast the daily sales. These techniques do not consider multiple sale points and products at the same time. They reach a minimum RMSWLE of 0.555. These solutions achieve competitive results. However, they require training one model per item and store (around 238,000 models), which is not a scalable approach for a production environment. The methods proposed in this paper consist of a single model that is used over all the time periods, items and points of sale.

Other studies using the same data set have been found: Curtin et al. (2020), Khamis et al. (2020), Kuleshov et al. (2018), Lim et al. (2019), Malik et al. (2019), Schleich et al. (2019), Shaikhha et al. (2020) and Wang et al. (2020). Although some of them may be interesting for the supply chain goals, they deviate from our demand/sales forecast goal.

## 2. Material and methods

The size of the data set chosen (around $4 \cdot 10^8$ samples) enables the use of deep learning models. Two different neural architectures have been designed: a *seq2seq* model and a *transformer* model.

### 2.1. Seq2seq

A *sequence-to-sequence* architecture (abbreviated as *seq2seq*) is a model that is trained to map an input sequence to an output sequence, without any length restriction in both sides (input and output sequences can be of different length) (Sutskever et al., 2014). This architecture contains two main blocks: one encoder and one decoder. The encoder consists of a recurrent neural network which processes the input sequence, one sample at a time, condensing all the relevant input sequence information into a fixed length *context vector*. This vector is usually the last hidden state of the encoder $\mathbf{h_t}$ (Uday Kamath & Whitaker, 2019). The decoder, also consisting of a recurrent neural network, generates the output sequence conditioned to the *context vector*. Both modules are trained together to minimize an error term over the output.

For the purpose of the current study, the original *seq2seq* architecture has been slightly modified to condition the *context vector* to a set of static (non time-dependent) features (see Fig. 4 for a graphical representation of the architecture). To achieve that, the original context vector $\mathbf{h_t}$ has been concatenated with the static features (item and store embeddings and related features) and then passed into a feed-forward
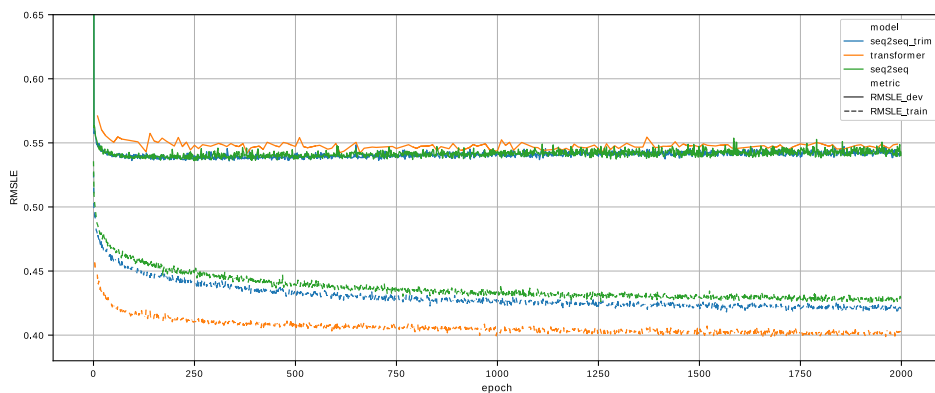
**Fig. 6.** Evolution of the train and validation (dev) error during the process of training, for all the models.
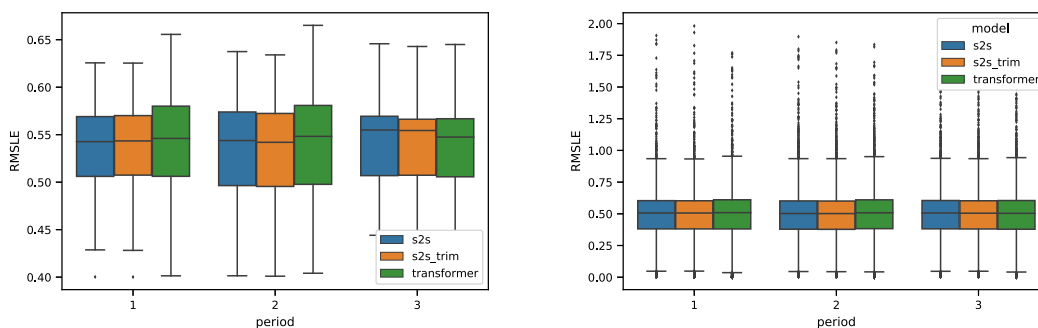


**Fig. 7.** Distribution of the error across stores (left) and across items (right) for every model and for the three different test periods used.

neural network with two hidden layers. The context conditioning module is an important part of the network because it allows the model to adapt the predictions to each product and store. The output of the feed-forward neural network has been used as the initial state of the decoder. The input of the first recurrent cell of the decoder is a special symbol that indicates the model that it is the first step of the output sequence. In this case, the special symbol is a vector containing all zeros.

The decoder module is a first order auto-regressive model, i.e. the predicted value for time step $t$ is used as input for the prediction of time step $t + 1$.

### 2.2. Transformer

*Transformer* architectures were firstly published in 2017 (Vaswani et al., 2017). This architecture removes the need to use recurrent neural networks by implementing attention and self-attention mechanisms (Bahdanau et al., 2015). Like *seq2seq* architectures, the transformers are able to map an input sequence to an output sequence, with potentially different lengths. Similarly, they also consist of two blocks: the encoder and the decoder.

The attention mechanism can be described as shown in Eq. (1). Where $Q$, $K$, $V$ stand for *query*, *key* and *value*, respectively. These three pieces represent an analogy, introduced by Vaswani et al. (2017), of the information retrieval systems where a *query* is used in order to look for the matching *key* (or the most similar one) and retrieving its *value*. The attention mechanism working principle is similar to those systems. There are many possible differentiable similarity functions ($f$) that can be used (Uday Kamath & Whitaker, 2019). Vaswani et al. (2017) propose the Scaled Dot Product as similarity function, given that it is scaled so that different length sequences can be easily compared together. The Scaled Dot Product is defined in Eq. (2), where $d_K$ represents the length of the key vector $K$. We have adopted this version as it showed to work well in the initial transformer publication.

We applied a slight modification over the original transformer, removing the *softmax* operation of the output and only using the

categorical embeddings for the categorical inputs. This is necessary in our case because our task is a regression and not a classification.

$$C(Q, K, V) = \text{softmax}(f(\mathbf{Q}, \mathbf{K})) \cdot \mathbf{V} \quad (1)$$

$$f_{\text{SDP}}(\mathbf{Q}, \mathbf{K}) = \frac{(\mathbf{Q} \cdot \mathbf{K}^T)}{\sqrt{d_K}} \quad (2)$$

Following the information retrieval analogy and as illustrated in Fig. 5, there are two types of attention being used in this architecture.

- *encoder–encoder attention*: this is a form of self-attention that is used in the encoder module. In it, the *query*, the *key* and the *value* come from the same time series.
- *decoder–decoder masked attention*: this is also a form of self-attention with the particularity that the operation is forced to be causal, i.e. it only uses time steps from the past, the future ones are masked out. The *query*, *key* and *value* come from the same time series.
- *encoder–decoder attention*: this attention mechanism compares the decoder information with the encoder one, hence it is not self-attention. The *query* comes from the decoder while the *key* and *value* are taken from the encoder output.

To train the *transformer* architecture the *teacher forcing* technique (Goyal et al., 2016; Williams & Zipser, 1989) is used. It consists in feeding the decoder with the target sequence, right-shifted by one sample so that the training can be done in one single calculation per batch. This technique is commonly used in auto-regressive models to improve the speed of training, and showed good results in the literature (Vaswani et al., 2017). On inference, *teacher forcing* is no longer available (because the future time steps of the time series are unknown) so auto-regression is used to compute the next steps recursively (i.e. the predicted sample is fed back to the input in order to predict the next sample).
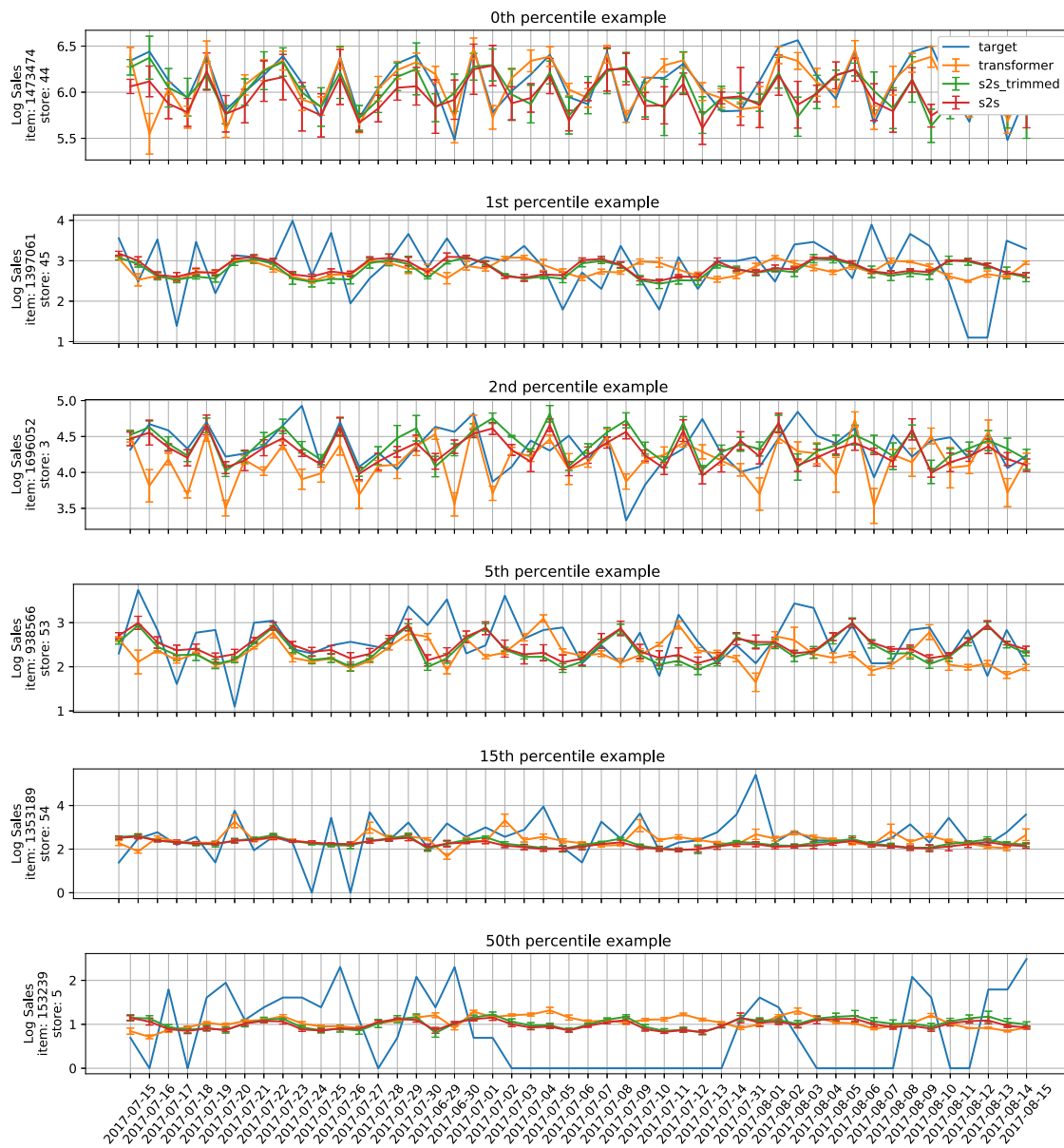
**Fig. 8.** Actual and forecasted sales in log space for six examples of store–item combinations representing the 0th (best prediction), 1st, 2nd, 5th 15th and 50th percentiles of RMSLE (relative to the target variable average) from top to bottom. The three test periods have been concatenated along the X axis. The error bars show the standard deviation across the 5 runs.

## 2.3. Random max time step trick

At training time and with the aim of improving generalization over different periods of time, each *minibatch* has been constructed so that the maximum time step (the most recent one) is drawn randomly from the time line. This trick allows the algorithm to learn a model that generalizes over different periods of time, preventing it to overfit to a single time span.

## 3. Experimental setup and results

### 3.1. Experimental setup

The data set provided has intentionally been minimally pre-processed as one of the goals of the current study is to provide a simple and flexible solution to the demand forecast problem. The most important transformation consisted of filling the zero sales records, as the data set was provided without them. The numerical input

variables have been normalized by centering and scaling them while the categorical variables have been turned into *one hot encoding* representations. The target variable has been normalized using a logarithmic transformation, as suggested by the authors of the data set in the *Kaggle* competition (Corporación Favorita, 2018). The ID variables corresponding to the store and the item have been used as an input to an embedding lookup layer to give the model the opportunity to learn store or item related information.

The model has been trained using daily data from January 1st 2013 to May 27th 2017, to produce daily forecasts of the next 16 days.[2] Data from June 13th 2017 to June 28th 2017 have been used for validation purposes and the next 3 16-days time spans (June 29th to July 14th, July 15th to July 30th and July 31st to August 15th, referred

---

[2] This is not an arbitrary decision, we chose 16 days because that was the requirement in the *Kaggle* competition. That choice would make sense for bimonthly forecast publications, as it would be applicable for months with 28, 29, 30 and 31 days.

**Fig. 9.** Actual and forecasted sales in linear space for six examples of store–item combinations representing the 0th (best prediction), 1st, 2nd, 5th 15th and 50th percentiles of RMSLE (relative to the target variable average) from top to bottom. The three test periods have been concatenated along the X axis. The error bars show the standard deviation across the 5 runs.

subsequently as period 1, 2 and 3 respectively) have been used to test the performance of the algorithms.

The *random max time step* has been constrained not to lay before October 29th 2013, to assure that the model has at least 300 days of history to learn from.

In the *seq2seq* model all the history (from January 1st 2013) has been used as input. In the *transformer* model, given the quadratic computational complexity dependence on the length of the input sequence, the history had to be shortened to 200 days. In the spirit of fair comparison, an alternative seq2seq version (referred subsequently as *seq2seq trimmed*) has been trained using the 200 most recent time steps in every *minibatch*. To facilitate the interpretation of the results, two baselines have been included: *random* and *average*. The first one consists of measuring the accuracy of a naive prediction built by randomly permuting the target variable. The second one consists of predicting the average of the target variable for all the instances.

### 3.2. Results and discussion

The results have been measured using the *Root Mean Squared Logarithmic Error* (*RMSLE*, defined in Eq. (3)), *Root Mean Squared Weighted Logarithmic Error* (*RMSWLE*, defined in Eq. (4), where the perishable items are given a weight of 1.25, and 1.0 to the rest), and *Mean Absolute Logarithmic Error* (*MALE*, defined in Eq. (5)).

$$RMSLE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\log(\hat{y}_i + 1) - \log(y_i + 1)\right)^2} \tag{3}$$

$$RMSWLE = \sqrt{\frac{\sum_{i=1}^{n} w_i \left(\log(\hat{y}_i + 1) - \log(y_i + 1)\right)^2}{\sum_{i=1}^{n} w_i}} \tag{4}$$

$$MALE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left|\log(\hat{y}_i + 1) - \log(y_i + 1)\right|} \tag{5}$$

**Table 1**

Results of the models trained for three different time spans. All the models have been trained five times to reduce the effect of different random initialization. The errors are represented as mean ± standard deviation, across the five runs. We have highlighted the rows corresponding with the models that achieved the lowest RMSLE. At the bottom of the table we have included two benchmark metrics extracted from the previous works, although the authors do not clearly specify the period of time used to measure the results, they should be used only as a reference.

| Period | Model | RMSLE | RMSWLE | MALE |
|---|---|---|---|---|
| 1 | **Seq2seq** | **0.5380 ± 0.0016** | **0.5376 ± 0.0016** | **0.3450 ± 0.0024** |
| | Seq2seq trimmed | 0.5381 ± 0.0008 | 0.5377 ± 0.0008 | 0.3442 ± 0.0008 |
| | Transformer | 0.5439 ± 0.0024 | 0.5436 ± 0.0023 | 0.3386 ± 0.001 |
| | Baseline: random | 1.474 ± 0.0003 | 1.4795 ± 0.0003 | 1.0691 ± 0.0002 |
| | Baseline: average | 1.0422 | 1.05 | 0.8744 |
| 2 | Seq2seq | 0.5431 ± 0.0014 | 0.5421 ± 0.0013 | 0.3475 ± 0.0012 |
| | **Seq2seq trimmed** | **0.5413 ± 0.0019** | **0.5403 ± 0.0018** | **0.3444 ± 0.0012** |
| | Transformer | 0.5495 ± 0.0021 | 0.5486 ± 0.0021 | 0.3415 ± 0.0012 |
| | Baseline: random | 1.4649 ± 0.0002 | 1.4702 ± 0.0002 | 1.0577 ± 0.0003 |
| | Baseline: average | 1.0358 | 1.0433 | 0.8655 |
| 3 | Seq2seq | 0.544 ± 0.0021 | 0.5431 ± 0.0021 | 0.3502 ± 0.0028 |
| | Seq2seq trimmed | 0.5423 ± 0.0015 | 0.5414 ± 0.0016 | 0.3481 ± 0.0017 |
| | **Transformer** | **0.5414 ± 0.0015** | **0.5407 ± 0.0014** | **0.3366 ± 0.0012** |
| | Baseline: random | 1.4555 ± 0.0002 | 1.4606 ± 0.0002 | 1.0517 ± 0.0002 |
| | Baseline: average | 1.029 | 1.0363 | 0.8616 |
| Benchmark | Kechyn et al. (2018) | – | 0.578 | – |
| Benchmark | Calero and Caro (2018) | – | 0.555 | – |

**Table 2**

Results of the models trained with and without the *random max time step trick*. All the models have been trained five times to reduce the effect of different random initialization. The errors are represented as mean ± standard deviation, across the five runs. We have highlighted the rows corresponding with the models that achieved the lowest RMSLE.

| Period | Trick/No trick | RMSLE | RMSWLE | MALE |
|---|---|---|---|---|
| 1 | Trick | **0.5381 ± 0.0008** | **0.5377 ± 0.0008** | **0.3442 ± 0.0008** |
| | No trick | 0.6077 ± 0.0055 | 0.6073 ± 0.0054 | 0.4037 ± 0.0171 |
| 2 | Trick | **0.5413 ± 0.0019** | **0.5403 ± 0.0018** | **0.3444 ± 0.0012** |
| | No trick | 0.5895 ± 0.0042 | 0.5886 ± 0.0042 | 0.3892 ± 0.0216 |
| 3 | Trick | **0.5423 ± 0.0015** | **0.5414 ± 0.0016** | **0.3481 ± 0.0017** |
| | No trick | 0.5929 ± 0.0127 | 0.5922 ± 0.0125 | 0.3938 ± 0.0318 |

**Table 3**

Results of the *Seq2seq* models trained with different input sequence lengths. All the models have been trained five times to reduce the effect of different random initialization. The errors are represented as mean ± standard deviation, across the five runs. We have highlighted the rows corresponding with the models that achieved the lowest RMSLE.

| Period | Sequence length | RMSLE | RMSWLE | MALE |
|---|---|---|---|---|
| 1 | Full | 0.5380 ± 0.0016 | 0.5376 ± 0.0016 | 0.3450 ± 0.0024 |
| | 200 | 0.5381 ± 0.0008 | 0.5377 ± 0.0008 | 0.3442 ± 0.0008 |
| | **75** | **0.5372 ± 0.0016** | **0.5369 ± 0.0016** | **0.3458 ± 0.0034** |
| | 10 | 0.5452 ± 0.0008[a] | 0.5447 ± 0.0009[a] | 0.3478 ± 0.0017 |
| | 1 | 0.5812 ± 0.0042[a] | 0.5807 ± 0.0042[a] | 0.3795 ± 0.0029[a] |
| | 0 | 0.8411 ± 0.0016[a] | 0.8453 ± 0.0015[a] | 0.5408 ± 0.0028[a] |
| 2 | Full | 0.5431 ± 0.0014 | 0.5421 ± 0.0013 | 0.3475 ± 0.0012 |
| | 200 | 0.5413 ± 0.0019 | 0.5403 ± 0.0018 | 0.3444 ± 0.0012[a] |
| | **75** | **0.5400 ± 0.0010[a]** | **0.5392 ± 0.001[a]** | **0.3458 ± 0.0009** |
| | 10 | 0.5510 ± 0.0049[a] | 0.5501 ± 0.0048[a] | 0.3509 ± 0.0038 |
| | 1 | 0.6162 ± 0.0035[a] | 0.6156 ± 0.0037[a] | 0.4011 ± 0.0032[a] |
| | 0 | 0.8426 ± 0.0016[a] | 0.8461 ± 0.0015[a] | 0.5388 ± 0.0027[a] |
| 3 | Full | 0.5440 ± 0.0021 | 0.5431 ± 0.0021 | 0.3502 ± 0.0028 |
| | 200 | 0.5423 ± 0.0015 | 0.5414 ± 0.0016 | 0.3481 ± 0.001 |
| | **75** | **0.5418 ± 0.0026** | **0.5411 ± 0.0025** | **0.3499 ± 0.0047** |
| | 10 | 0.5560 ± 0.0051[a] | 0.5548 ± 0.0051[a] | 0.3562 ± 0.0049 |
| | 1 | 0.6360 ± 0.0058[a] | 0.6352 ± 0.0062[a] | 0.4163 ± 0.0041[a] |
| | 0 | 0.8387 ± 0.0017[a] | 0.8419 ± 0.0016[a] | 0.5373 ± 0.0027[a] |

[a]Added to the experiments where the metric is significantly different than the full sequence error (as per a two-tail T-test with $\alpha = 0.05$).

dependence on the sequence length in the *transformer* architecture; in *seq2seq* it is linear. The simplest model (*seq2seq trimmed*) was the fastest of the three alternatives, with no noticeable decrease in performance, either in the general picture or in the daily figures.

### 3.3. Ablation study

In this subsection we analyze the effect of two core pieces of our proposed architecture: the *random max time step trick* and the length of the input sequences in the *Seq2seq trimmed* model.

#### 3.3.1. Random max time step trick

We retrained the *Seq2seq trimmed* model without the *random max time step trick*. Table 2 summarizes the errors obtained at the best iteration of each model, averaged across five repeated runs. From the results, we conclude that when the *random max time step trick* is used the model achieves significantly superior performance. This is due to the fact that randomizing the max time step helps the model to capture behaviors of the target signal at different times. Fig. 11, shows the training curves when the trick is used and when it is not used, suggesting that the trick may also act as a regularization technique, as the model is much less prone to overfitting when the trick is used.

#### 3.3.2. Input sequences length

As we showed in the Table 1, the *Seq2seq* model can be further simplified by trimming the length of the input sequences. In this subsection we study the minimum length of the input sequence without significant performance degradation. For that, we retrained the *Seq2seq* model with different input sequence lengths to determine where is the optimum. The Table 3 shows the results of the model with the full sequences, and with sequences trimmed to 200, 75, 10, 1 and 0 time steps (where 0 time steps means not using any input sequence information at all, only static features). The results show that we can reduce the sequence lengths to up to 75 time steps without losing performance (and even slightly improving the generalization). Further reductions to 10 and 1 time steps start showing performance degradation. When not using input sequences (length = 0) the performance degrades notably, as compared to using only one time step. We hypothesize that this happens because the model needs some reference level of number of sales per product to produce accurate forecasts.

In the previous equations, $\hat{y}$ represents the predicted sales, $y$ represents the actual sales and $N$ is the total number of samples. The logarithmic component of the error metrics was introduced because different products at different shops have arbitrarily different demand levels. The usage of the logarithm normalizes the unit sales distribution and makes the whole problem easier to measure. We always used the natural logarithm in this study. The *RMSWLE* error metric has been introduced for easier comparison and benchmarking with future studies.

Fig. 6 shows how the errors evolve in every epoch. Fig. 7 decomposes the error at store and item level, in order to show in detail how the errors vary along these dimensions. Finally, Figs. 8 and 9 show examples of actual and forecasted time series in log and linear scales, respectively.

From Fig. 7, we observe that the performance is very similar across models. A numerical comparison of the errors obtained for every model is presented in Table 1. Additionally, a deeper daily analysis is provided in Fig. 10. From these figures, we conclude that the error is not distributed randomly across products, stores and time.

From Table 1, we can conclude that the three models perform similarly. However, the daily figures show that the transformer error has more variability around the second and fourth day of forecast. This may be due to the fact that the model has been trained using *teacher forcing* (Goyal et al., 2016; Williams & Zipser, 1989), and at inference time, an auto-regressive strategy has been used to compute the forecasted sales. This may cause distribution shifts that impact the quality of the forecast. Besides, the *seq2seq* models were much faster at training and inference time. This is due to the quadratic complexity
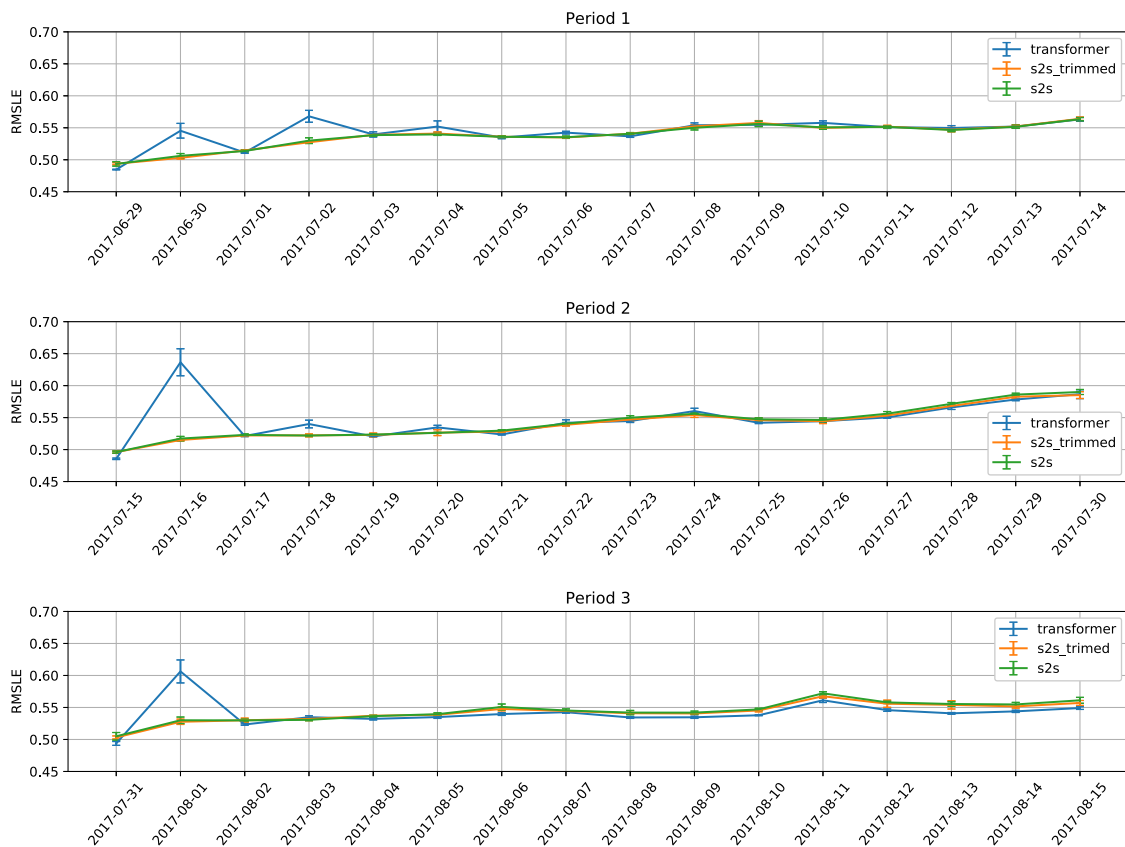
**Fig. 10.** Daily *RMSLE* for the three test periods (in chronological order, from top to bottom). The error bars shown in the figures represent the standard deviation of the three runs. Despite the error spikes in the 2nd day of the forecast, an ANOVA test shows non-significant differences between the average performance of the three models (with the following p-values: 0.0947, 0.1823 and 0.6181 for periods 1, 2 and 3, respectively).
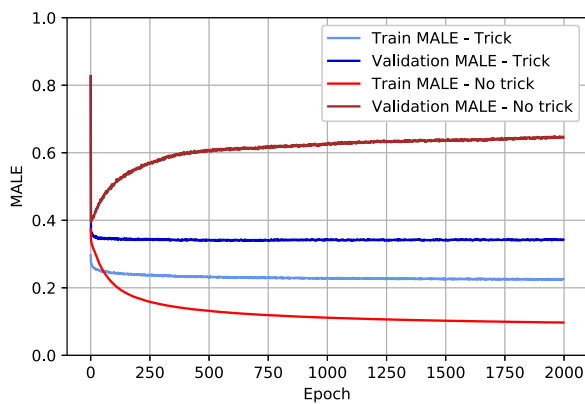


**Fig. 11.** Training and validation MALE curves of the *Seq2seq trimmed* when using the *random max time step trick* and when disabling it.

## 4. Conclusions

Along this work, we have proposed a *seq2seq* and a transformer architecture capable to solving the problem of sales forecasting for the *Corporación Favorita* problem. We have also provided a trick (that we named *random max time step trick*) that allowed to train the model to adapt to different time steps, not requiring to retrain the model every time a prediction is needed. We have empirically proved that it is possible to build a forecast for different products, at different points of sale and at different points in time using a single model. Our *seq2seq trimmed* model achieved the best performance at the lowest theoretical

computational cost. For that reason, we recommend its usage for this type of use cases.

Deeper and more complex models must be tested further in order to try to improve performance by allowing more non-linear representations. In a real case, feature engineering may also be useful in order to help finding better representations. Finally, more sophisticated normalization methods for the target variable might be useful to deal with different magnitudes and sparsity.

### Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

### References

Badri, H., Ghomi, S., & Hejazi, T. H. (2017). Supply chain network design: A value-based approach. *Transportation Research*, 1–17. http://dx.doi.org/10.1016/j.tre.2017.06.012.

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *ICLR'15, 3rd international conference on learning representations*.

Bell, P. C. (2000). Forecasting demand variation when there are stockouts. *Journal of the Operational Research Society*, *51*(3), 358–363, URL: http://www.jstor.org/stable/254094.

Calero, A. S. M., & Caro, J. M. B. (2018). *Corporación favorita grocery sales forecasting* (Master thesis), Universidad Internacional de Andalucía, URL: https://dspace.unia.es/handle/10334/3921.

Corporación Favorita, K. (2018). Corporación favorita grocery sales forecasting data set. Available in the following link: https://www.kaggle.com/c/favorita-grocery-sales-forecasting/data.

Cramer-Flood, E. (2020). Global ecommerce 2020: Ecommerce decelerates amid global retail contraction but remains a bright spot. *E-Marketer*.

Curtin, R. R., Moseley, B., Ngo, H. Q., Nguyen, X., Olteanu, D., & Schleich, M. (2020). Rk-means: Fast clustering for relational data. In S. Chiappa, & R. Calandra (Eds.), *Proceedings of machine learning research*: vol. 108, *The 23rd international conference on artificial intelligence and statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]* (pp. 2742–2752). Proceedings of Machine Learning Research, URL: http://proceedings.mlr.press/v108/curtin20a.html.

Deep, K., & Salhi, M. J. S. (2019). *Logistics, supply chain and financial predictive analytics*. http://dx.doi.org/10.1007/978-981-13-0872-7.

Forslund, H., & Jonsson, P. (2007). The impact of forecast quality on supply chain performance. *International Journal of Operations & Production Management*, *27*, 90–107. http://dx.doi.org/10.1108/01443570710714556.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.

Goyal, A., Lamb, A., Zhang, Y., Zhang, S., Courville, A., & Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks. In *NIPS'16, Proceedings of the 30th international conference on neural information processing systems* (pp. 4608–4616).

Graves, S., & Willems, S. (2008). Strategic inventory placement in supply chains: Non-stationary demand. *Manufacturing & Service Operations Management*, *10*, 278–287. http://dx.doi.org/10.1287/msom.1070.0175.

Helmini, S., Jihan, N., Jayasinghe, M., & Perera, S. (2019). Sales forecasting using multivariate long short term memory network models. *PeerJ Preprints*, *7*, e27712v1. http://dx.doi.org/10.7287/peerj.preprints.27712v1.

Hyndman, R., & Athanasopoulos, G. (2018). *Forecasting: Principles and practice* (2nd ed.). Australia: OTexts.

Kaipia, R. (2009). Coordinating material and information flows with supply chain planning. *The International Journal of Logistics Management*, *20*, 144–162. http://dx.doi.org/10.1108/09574090910954882.

Kechyn, G., Yu, L., Zang, Y., & Kechyn, S. (2018). Sales forecasting using WaveNet within the framework of the Kaggle competition. arXiv:1803.04037.

Khamis, M. A., Ngo, H. Q., Nguyen, X., Olteanu, D., & Schleich, M. (2020). Learning models over relational data using sparse tensors and functional dependencies. *ACM Transactions on Database Systems*, *45*(2), http://dx.doi.org/10.1145/3375661.

Kilimci, Z. H., Akyuz, A. O., Uysal, M., Akyokus, S., Uysal, M. O., Bulbul, B. A., & Ekmis, M. A. (2019). An improved demand forecasting model using deep learning approach and proposed decision integration strategy for supply chain. *Complexity*, *2019*, 1–15. http://dx.doi.org/10.1155/2019/9067367.

Kuleshov, V., Fenner, N., & Ermon, S. (2018). Accurate uncertainties for deep learning using calibrated regression. In J. Dy, & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (pp. 2796–2804).

Lim, B., Arik, S. O., Loeff, N., & Pfister, T. (2019). Temporal fusion transformers for interpretable multi-horizon time series forecasting. CoRR, abs/1912.09363. URL: http://arxiv.org/abs/1912.09363. arXiv:1912.09363.

Lipsman, A. (2019). Global ecommerce 2019: Ecommerce continues strong gains amid global economic uncertainty. *E-Marketer*.

Malik, A., Kuleshov, V., Song, J., Nemer, D., Seymour, H., & Ermon, S. (2019). Calibrated model-based deep reinforcement learning. In K. Chaudhuri, & R. Salakhutdinov (Eds.), *Proceedings of machine learning research*: vol. 97, *Proceedings of the 36th international conference on machine learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA* (pp. 4314–4323). Proceedings of Machine Learning Research, URL: http://proceedings.mlr.press/v97/malik19a.html.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., & Kavukcuoglu, K. (2016). WaveNet: A generative model for raw audio. In *The 9th ISCA speech synthesis workshop, Sunnyvale, CA, USA, 13-15 September 2016* (p. 125). ISCA, URL: http://www.isca-speech.org/archive/SSW_2016/abstracts/ssw9_DS-4_van_den_Oord.html.

Schleich, M., Olteanu, D., Abo Khamis, M., Ngo, H. Q., & Nguyen, X. (2019). A layered aggregate engine for analytics workloads. In *SIGMOD '19, Proceedings of the 2019 international conference on management of data* (pp. 1642–1659). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3299869.3324961.

Shaikhha, A., Schleich, M., Ghita, A., & Olteanu, D. (2020). Multi-layer optimizations for end-to-end data analytics. In *CGO 2020, Proceedings of the 18th ACM/IEEE international symposium on code generation and optimization* (pp. 145–157). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3368826.3377923.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems 27* (pp. 3104–3112). Curran Associates, Inc..

Talupula, A. (2018). *Demand forecasting of outbound logistics using machine learning* (Master thesis), 371 79 Karlskrona, Sweden: Faculty of Computing, Blekinge Institute of Technology, URL: https://www.diva-portal.org/smash/get/diva2:1367098/FULLTEXT02.

Uday Kamath, J. L., & Whitaker, J. (2019). *Deep learning for NLP and speech recognition* (1st ed.). Springer Publishing Company, Inc., http://dx.doi.org/10.1007/978-3-030-14596-5.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, u., & Polosukhin, I. (2017). Attention is all you need. In *NIPS'17, Proceedings of the 31st international conference on neural information processing systems* (pp. 6000–6010).

Wang, J., Cevik, M., & Bodur, M. (2020). On the impact of deep learning-based time-series forecasts on multistage stochastic programming policies. arXiv e-prints, arXiv:2009.00665. arXiv:2009.00665.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*, 270–280.