# Probabilistic Demand Forecasting with Graph Neural Networks

Nikita Kozodoi[1], Elizaveta Zinovyeva[1], Simon Valentin[1,2], João Pereira[3], and Rodrigo Agundez[3]

[1] Amazon Web Services, Germany
{kozodoi,zinovyee}@amazon.com
[2] University of Edinburgh, United Kingdom
{s.valentin}@ed.ac.uk
[3] adidas, Amsterdam, The Netherlands
{joao.pereira2, rodrigo.agundez}@adidas.com

**Abstract.** Demand forecasting is a prominent business use case that allows retailers to optimize inventory planning, logistics, and core business decisions. One of the key challenges in demand forecasting is accounting for relationships and interactions between articles. Most modern forecasting approaches provide independent article-level predictions that do not consider the impact of related articles. Recent research has attempted addressing this challenge using Graph Neural Networks (GNNs) and showed promising results. This paper builds on previous research on GNNs and makes two contributions. First, we integrate a GNN encoder into a state-of-the-art DeepAR model. The combined model produces probabilistic forecasts, which are crucial for decision-making under uncertainty. Second, we propose to build graphs using article attribute similarity, which avoids reliance on a pre-defined graph structure. Experiments on three real-world datasets show that the proposed approach consistently outperforms non-graph benchmarks. We also show that our approach produces article embeddings that encode article similarity and demand dynamics and are useful for other downstream business tasks beyond forecasting.

**Keywords:** Demand forecasting · Graph neural networks · E-commerce

## 1 Introduction

Demand forecasting is a prominent business task faced by retailers. Predictions of future demand for articles sold by a retailer serve as inputs for many operational decisions, including inventory planning, logistics, and supply chain optimization [9]. The importance of demand forecasting is further emphasized by a rapid growth of the e-commerce retail sector, which is expected to account for more than \$6.51 trillion in 2023, taking up 22.3% of the global retail market [10].

E-commerce retailers frequently operate with thousands of articles sold simultaneously, and experience dynamic and volatile demand [7]. This poses challenges for accurate demand forecasting. One of the key difficulties is that the article's demand depends not only on its historical demand, but also on demand for similar articles, introduction and removal of competing articles, out-of-stock events, and other information. This emphasizes the importance of incorporating article relationships into a forecasting model in a principled manner.

Traditional forecasting techniques such as Autoregressive Integrated Moving Average (ARIMA) models [4] are univariate and consider individual time series in isolation. In contrast, modern techniques based on deep learning models such as DeepAR [19] or Temporal Fusion Transformer [13] are trained on multiple time series simultaneously, which allows learning the past behavior across similar series. However, such global models are typically unable to incorporate inter-series relationships during inference, failing to account for the impact of related articles on the article's of interest demand. Integrating time series relationships during both training and inference is feasible with multivariate models such as Vector Autoregressive (VAR) models [26]. At the same time, such models experience difficulties with scaling to high-dimensional e-commerce environments.

Recent research has suggested using graphs to account for the inter-series correlations. Representing the data as a graph and using Graph Neural Networks (GNNs) to extract patterns from graph-structured data allows integrating time series relationships directly into a prediction model [22]. Studies outside of the retail domain have shown promising results in forecasting road traffic or weather events [15,24,25]. At the same time, the literature on GNNs for demand forecasting is scarce. Recently, Gandhi et al. [7] proposed a GNN-based forecasting approach suited for a multiple-seller marketplace setting and demonstrated that GNNs have potential to improve the forecasting accuracy for cold-start articles.

This paper proposes a novel demand forecasting approach that builds on the framework of Gandhi et al. [7] and makes two contributions. First, we develop an end-to-end forecasting model architecture that combines two components: (i) GNN encoder that incorporates article relationships during training and inference; (ii) state-of-the-art DeepAR decoder for demand forecasting. In contrast to recurrent architectures commonly used in the literature, DeepAR produces probabilistic demand predictions, which are crucial for decision-making under uncertainty [19]. We denote the proposed forecasting model as GraphDeepAR.

Our second contribution is proposing a generic graph construction approach that does not require a pre-defined graph structure. We build graphs using pairwise article attribute similarity to define connections between articles, which allows leveraging article meta-data to model article relationships and augmenting it with domain knowledge. Unlike some of the previous approaches, the proposed solution is highly scalable. We test GraphDeepAR on three real-world datasets and show that it outperforms the standard DeepAR model.

## 2 Related Work

### 2.1 Leveraging Inter-Series Relationships

Forecasting refers to prediction of future events based on the previously observed data. One of the key challenges in forecasting is accounting for relationships between multiple time series [14]. Many of the modern forecasting techniques focus on a univariate setting, where the task is limited to modeling a single time series or a small number of individual unrelated series. In practice, one is often required to forecast a large number of related series at the same time (e.g., energy consumption across different households or consumer demand for multiple related articles) [19]. In such settings, demand for a given article depends not only on its historical demand, but also on the demand for substitute and complement articles, launch of new competing articles, and other related factors [7].

One way to account for such relationships during training is to use global models such as DeepAR [19]. Here, we refer to models that are trained simultaneously on all time series available in a dataset as global models. Such global models are able to learn the past behavior across similar sequences. However, during inference, they are still limited to univariate forecasts, which prohibits using the previous values of related time series to predict the target series.

To incorporate inter-series relationships during both training and inference, previous work suggested multivariate forecasting models such as VAR [26] or LSTNet [12]. Multivariate models use observations from all available time series as input and produce joint predictions of multiple time series [3]. Yet, such models are typically limited to a small number of time series and do not scale well to environments with thousands of related series. Other attempts to account for series relationships rely on lower-dimensional representations such as matrix factorization techniques that scale to larger volumes of data more favorably [21].

### 2.2 Forecasting with Graph Neural Networks

Recent developments in graph machine learning have inspired researchers to explore using graphs for modeling relationships between time series [22]. In graph machine learning, the input data is transformed into a graph, and the entities and their relationships are represented as graph nodes and edges containing a set of features. The literature has suggested GNN architectures such as Graph Convolutional Networks (GCN) to learn from graph-structured data [11].

Prior work on GNNs for forecasting has mainly focused on traffic forecasting, where a graph representation is constructed using locations of the traffic sensors. For instance, Yu et al. proposed the Spatio-Temporal GCN (ST-GCN) model that enables faster training [25]. More recently, Wu et al. developed the GraphWaveNet architecture that learns a self-adaptive adjacency matrix and was shown to outperform DCRNN and ST-GCN [24]. Other applications of GNN-based methods include forecasting frost incidence across multiple weather stations [15] and passenger demand prediction across city areas [1]. Going beyond applications with a clear pre-defined graph structure, Cao et al. developed a Spectral Temporal GNN (StemGNN) that uses self-attention to automatically learn latent correlations between related time series [5]. However, the application of StemGNN is limited to problems with a small number of time series (i.e., less than a thousand) due to the computational cost.

In the retail demand forecasting space, where one is faced with thousands of related time series, the literature on using GNNs remains rather limited.

Gandhi et al. suggested a graph-based model that combines GNN blocks with an LSTM layer [7]. The study has shown the superior performance of graph-based architectures compared to a non-graph baseline. At the same time, the approach proposed by Gandhi et al. is designed for a multiple-seller marketplace setting, where the graph structure is defined by seller-article relationships, and is limited to point-based predictions. This paper aims at bridging this gap and proposes a novel demand forecasting approach that extends the framework of Gandhi et al. and addresses some of the key limitations described above.

## 3 Methodology

### 3.1 Problem Formulation

We formalize the demand forecasting task as follows. Let $\mathcal{A} := \{X_i, Z_i, Y_i\}_{i=1}^N$ denote the time series and features of $N$ articles sold by a retailer, where each element $A_i \in \mathcal{A}$ is a tuple with three components. $Y_i = (y_i^t)_{t=1}^T$ is the individual time series of the $i-$th article up to the final time step $T$, such that $y_i^t \in \mathbb{R}^+$ represents the demand for article $i$ at some time step $t$. The articles are described with two sets of real-valued feature representations: $X_i \in \mathbb{R}^M$ is a vector with $M$ static features of article $i$, whereas $Z_i = (Z_i^t)_{t=1}^T$ with $Z_i^t \in \mathbb{R}^{L \times T}$ is a matrix with $L$ time-varying features of article $i$ at time $t$. The static features reflect article attributes such as size, color, and others. The time-varying features describe seasonal events such as week of the year, month, article promotions, and other attributes that are known in advance.

Our task is to predict demand for $N$ articles during the future $K$ time steps $(y_i^{T+1}, y_i^{T+2}, \dots, y_i^{T+K})_{i=1}^N$ given the historical demand $(y_i^1, y_i^2, \dots, y_i^T)_{i=1}^N$, the static features $(X_i)_{i=1}^N$ and the time-varying features $(Z_i^1, Z_i^2, \dots, Z_i^{T+K})_{i=1}^N$. The accuracy of the forecast is evaluated by comparing predicted demand values $(\hat{y}_i^t)_{i=1}^N$ to the actual values $(y_i^t)_{i=1}^N$ across all predicted articles and time steps.

### 3.2 Graph Construction

We address the demand forecasting task with a novel approach that splits into two components: (i) a spatial component, which includes the article graph and the GNN modules that extract knowledge from the graph-structured data; (ii) a temporal component, which is a sequence model that is able to learn from the temporal dynamics in the input time series and forecast future demand values.

The spatial component uses a graph, where each article is represented as a node, and edges between nodes reflect article relationships. That is, we construct a graph $\mathcal{G}^t = (\mathcal{N}^t, \mathcal{E})$ that describes articles at time $t$. The graph consists of the set of nodes $\mathcal{N}^t$ that includes time-varying node features of articles from $\mathcal{A}$ that are sold at time $t$. The set of edges $\mathcal{E}$ provides pairwise node connections and contains static edge features representing article relationships.

As reported in Section 2, graph edges are usually pre-defined by the nature of the problem. However, such knowledge is not available in a generic demand forecasting setting, which implies that relationships need to be identified by domain experts or inferred from data. This work uses attribute-based article similarity to identify article relationships automatically. We calculate pairwise cosine similarity scores between articles using static article features:

$$\text{similarity}\,(A_i, A_j) := \cos\,(X_i, X_j) = \frac{X_i \cdot X_j}{||X_i||||X_j||} \tag{1}$$

Note that calculating pairwise similarity values has exponential time complexity. At the same time, this calculation is only required once on the graph construction stage, and the graph is stored and reused for different model configurations. To optimize the time and memory usage of the calculation, we split the data in article chunks that can be run in parallel to construct the full similarity matrix.

We define an edge between two articles in $\mathcal{E}$ if their similarity exceeds a specified threshold, which serves as a graph construction hyperparameter. This approach allows us to incorporate domain expertise using the article features but also enables automatic identification of edges controlled by the choice of graph hyperparameters. All edges in the constructed graph are undirected.

The node features in $\mathcal{N}^t$ contain article information leveraged by the model to make demand forecasts. In our setting, an article can be described by the attribute-based features and the previous demand lags (i.e., demand across the previous timestamps). Since article attributes are static and can be directly inputted in the temporal component, we limit node features to $P$ demand lags: $\mathcal{N}_i^t = (y_i^{t-P+1}, y_i^{t-P+2}, \ldots, y_i^t)$. The number of lags serves as a hyperparameter. In addition, we calculate the node in-degree, which measures the number of neighbors of an article and serves as an additional node feature included in $\mathcal{N}_i^t$.

## 3.3 Model Architecture

The proposed demand forecasting approach leverages a deep neural network that combines a GNN encoder and a DeepAR decoder. Both components are trained end-to-end with a single loss function. Further, we refer to the end-to-end model as GraphDeepAR. The model architecture is depicted in Figure 1.

GraphDeepAR processes articles in mini-batches. For every batch presented to the model, we construct a graph with dynamic node features containing the article demand values over a certain time window, which spans over the $P$ consecutive time steps prior to the last timestamp in the mini-batch. The similarity-based graph edges remain static over the timestamps, whereas the node features are calculated based on the timestamp of the corresponding graph.

The graph $\mathcal{G}^t = (\mathcal{N}^t, \mathcal{E})$ serves as input for the two-layer GNN encoder. Each of the two layer blocks consists of a graph-based neighborhood pooling, activation function and dropout. The graph convolution layers represent article $i$ using aggregated node features of the neighboring articles $\mathcal{N}_{j\in I}^t$, where $\mathcal{I}$ is a set of articles connected with article $i$. This allows the encoder to gather information from similar articles and output article embeddings that encode information in its neighborhood. Formally, the output of the $k$-th GNN layer for article $i$ at

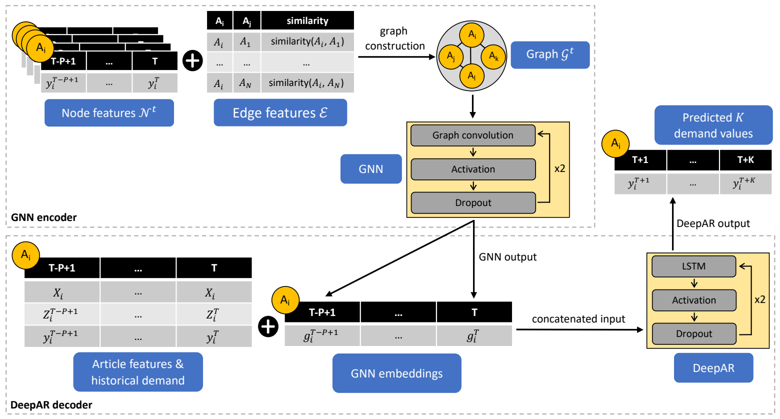

**Fig. 1.** GraphDeepAR Architecture. The figure uses a single article for illustration.

time $t$ is denoted as $h_k^t(i)$ and calculated as follows:

$$h^t_{k(i)} = \sigma\left(\frac{1}{|\mathcal{N}^t_j|}\sum_{j\in\mathcal{I}} W_k h^t_{(k-1)}(j)\right), \qquad (2)$$

where $W$ is a weight matrix to be learned from data, and $\sigma(\cdot)$ is a Leaky ReLU activation function [16]. GNN layers share parameters in $W$ across the timestamps, which implies that the encoder iteratively learns how to leverage node features irrespective of the time step. The embeddings generated by the last GNN layer are denoted as $G_i = (g_i^{t-P+1}, g_i^{t-P+2}\dots, g_i^t)$ with $G_i \in \mathbb{R}^{D\times P}$, where $D$ is a hyperparameter indicating the embedding dimensionality.

The temporal component represents a two-layer DeepAR forecasting model that produces demand predictions for each article. During training and inference, GNN embeddings $(G_i)_{i=1}^N$ encoding the data from the article neighborhood are concatenated with previous demand values $(y_i)_{i=1}^N$ that both cover the time window of length $P$, static article features $(X_i)_{i=1}^N$, and known time-varying features $(Z_i)_{i=1}^N$ for the previous $P$ and the following $K$ time steps. The concatenated matrix serves as input to the DeepAR decoder. The decoder outputs the mean $\mu$ and variance $s^2$ of a Students' $t$-distribution that is used to model a distribution over plausible future demand values. This choice is motivated by the heavier tails of the Student's $t$-distribution as compared to a Gaussian, which is more appropriate for typical demand dynamics in the retail domain. The demand predictions for the following $K$ time steps are then sampled from $t(\mu, s)$.

## 4 Experimental Setup

### 4.1 Data

Table 1 overviews the datasets used in the paper. Each dataset provides demand across different articles sold by a retailer. The first two datasets, *retail* and *e-*

*commerce*, are publicly available on Kaggle[4]. The *adidas* dataset *(hidden for review)* is proprietary sales data provided by adidas. The number of articles is ranging between 629 and 80,838, which allows us to evaluate our approach across environments with different dimensionality.

**Table 1.** Datasets Summary

| Dataset | No. articles | No. weeks | No. features |
|---|---|---|---|
| Retail | 629 | 148 | 12 |
| E-commerce | 8,810 | 128 | 5 |
| adidas | 80,838 | 140 | 20 |

In each dataset, we aggregate weekly demand across all stores to produce the target time series and use between 5 and 20 features, including static features describing the article attributes (e.g., size and category), and three dynamic features describing the seasonal patterns (week of the year, day of the month, week number). Since the original *e-commerce* dataset includes many articles where demand is low and static, we limit the training set for this dataset to articles that have the standard deviation of the weekly demand greater than 1.

We split each dataset into three subsets along the time axis: training, validation, and test sets. The test set is used for model evaluation and covers the last 26 weeks. The validation set is used to tune model hyperparameters; the length of the validation set is fixed to 13 weeks preceding the test set. The training set covers all data available prior to the validation set. Depending on data availability, each article may be observed in all three data subsets.

### 4.2 Setup

To evaluate our approach, we compare the proposed GraphDeepAR model to a DeepAR benchmark. The two models are trained and evaluated on the same data splits and share the same hyperparameters. For example, for the datasets *retail* and *e-commerce*, both models use two LSTM layers with a hidden size of 128 and are trained to minimize the $t$-distribution loss with a symmetric loss penalty. For the *adidas* dataset, we use an asymmetric penalty to treat differently under- and over-prediction costs based on the business context. Appendix A reports hyperparameter values used on the two public datasets.

There are two differences between DeepAR and GraphDeepAR related to the addition of the graph component. First, GraphDeepAR includes a GNN encoder described in Section 3. For efficient training, the GNN encoder performs neighborhood sampling that limits the maximum number of article neighbors considered on each training pass. This is achieved by removing a random subset of edges for each node in the graph, similar to the sampling strategy introduced

[4] Source: https://www.kaggle.com/datasets/berkayalan/retail-sales-data, https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales/data

in GraphSAGE [8]. Second, while DeepAR benefits from randomized batching, GraphDeepAR uses synchronized batching, which implies that all demand sequences inside one mini-batch cover the same time window. Synchronized batching reduces the randomness in data shuffling but is required for GraphDeepAR to ensure that batch-level graphs have node features that are consistent in time.

The forecasting accuracy is evaluated using multiple metrics. On the public datasets, we use three established forecasting metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Weighted Mean Absolute Percentage Error (WMAPE), which uses actual demand as weights to assign a higher importance to top-selling articles. The *adidas* dataset provides the opportunity to directly measure the forecast quality from a business perspective based on the financial value of under- and over-prediction. Here, we calculate the monetary loss of two models and report the financial uplift from GraphDeepAR as a percentage improvement relative to the DeepAR baseline.

Both models are implemented using the PyTorch Forecasting library [2]. We also use the Deep Graph Library [23] to construct the graph and implement the GNN encoder. Training is performed in PyTorch Lightning [6]. All experiments are conducted on compute instances running on Amazon SageMaker.

## 5 Results

### 5.1 Graph Illustration

Figure 2 illustrates graphs constructed for the two public datasets. Each node represents an article color-coded with the article category. For illustration purposes, we do not display the self-connecting edges and limit the *e-commerce* graph depicted in the right panel to 1,000 top-selling articles.

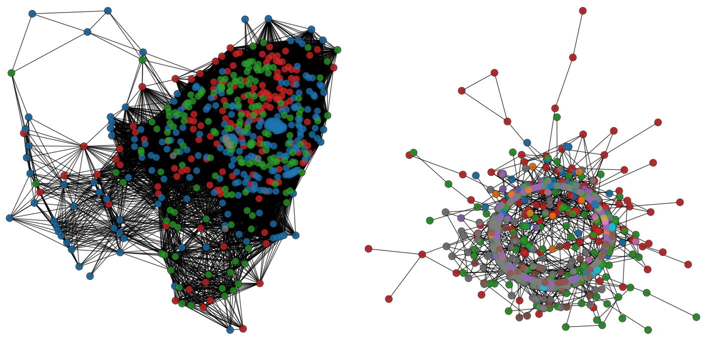

**Fig. 2.** Graph Illustration. Article graphs for *retail* (left) and *e-commerce* (right).

As described in Section 3, graph edges are based on article similarity. The *retail* dataset provides nine article features used to calculate pairwise cosine distance. The *e-commerce* data only includes two categorical features indicating the article hierarchy. To address this data scarcity, we encode article titles using a sentence transformer-based encoder with distilBERT [18,20]. The produced article embeddings of dimension 512 are used to calculate the cosine similarity.

As shown in Figure 2, the neighborhood size varies considerably across articles: both graphs contain clusters of densely connected articles as well as articles with no neighbors (42% in the *e-commerce* graph and 3% – in *retail*). Given the 0.95 similarity threshold, the average number of neighbors is 1.21 in the *e-commerce* graph and 198.13 in the *retail* graph (with a standard deviation of 114). This emphasizes the importance of selecting a suitable similarity threshold and using neighborhood sampling to facilitate fast training. Such variations in graph structure may affect forecasting accuracy differently for different articles, depending on the article's position in the graph and the number of neighbors.

## 5.2 Predictive Performance

Table 2 presents empirical results on two public datasets. Apart from the overall forecasting accuracy across all articles, we calculate metrics across three groups: *cold starts, connected articles,* and *Top*-100 *articles* – the top-selling articles with the highest total demand. We define cold starts as articles with less than five historical demand values at the time of the forecast, and connected articles as articles that have edge connections to other articles in the graph. Examples of individual demand predictions for several articles are illustrated in Appendix B.

**Table 2.** Results: Retail and E-Commerce Datasets

| Dataset | Subset | Model | RMSE | MAE | WMAPE |
|---|---|---|---|---|---|
| Retail | All articles | DeepAR | 204.68 | 51.53 | 0.43 |
| | | GraphDeepAR | **196.13** | **50.35** | **0.42** |
| | Cold starts | DeepAR | 44.79 | 19.83 | 0.66 |
| | | GraphDeepAR | **41.78** | **18.84** | **0.63** |
| | Connected articles | DeepAR | 207.12 | 52.34 | 0.42 |
| | | GraphDeepAR | **198.46** | **51.12** | **0.41** |
| | Top-100 articles | DeepAR | 419.40 | 171.28 | 0.36 |
| | | GraphDeepAR | **401.27** | **164.10** | **0.35** |
| E-commerce | All articles | DeepAR | 30.36 | 3.39 | 0.67 |
| | | GraphDeepAR | **20.65** | **3.08** | **0.61** |
| | Cold starts | DeepAR | **8.66** | 2.62 | 0.79 |
| | | GraphDeepAR | 8.72 | **2.62** | **0.79** |
| | Connected articles | DeepAR | 31.40 | 3.59 | 0.69 |
| | | GraphDeepAR | **21.40** | **3.18** | **0.61** |
| | Top-100 articles | DeepAR | 164.68 | 42.78 | 0.98 |
| | | GraphDeepAR | **110.50** | **29.60** | **0.68** |

Comparing the overall results, we see that GraphDeepAR consistently outperforms DeepAR in all three evaluation metrics. The largest gains are observed for the RMSE metric, where GraphDeepAR outperforms DeepAR by 4.36%

on *retail* and by 31.98% on *e-commerce* data. The bigger improvement on *e-commerce* can be explained by a substantially larger sample size of the dataset.

Performance gains vary across the article groups. On the *retail* dataset, the largest RMSE uplift of 6.72% is observed for cold starts, which suggests that leveraging historical demand for similar articles is particularly useful if there are no or too few previous demand values for a given article. Interestingly, we do not observe improvements for cold start articles on *e-commerce* data. This can be explained by the fact that cold starts in this dataset are rare (2% of the considered articles), have 30% less neighbors compared to carry-over articles, and tend to have a lower and more stable weekly demand with a standard deviation of 2.29. This is also reflected in smaller absolute values of MAE and RMSE achieved by both forecasting models for cold start articles on the *e-commerce* dataset in comparison to other article groups.

Considering further article groups, GraphDeepAR consistently outperforms DeepAR on connected articles on both datasets. This result is in line with our expectations, since GraphDeepAR aggregates historical demand values over the article neighborhood, which helps to improve the forecast performance for well-connected articles. Articles with neighbors constitute 97% articles on *retail* data and 58% articles on *e-commerce* data, which corresponds to a large share of the sold articles and emphasizes the importance of the observed gains.

The lack of financial data for the two public datasets makes it challenging to translate the observed gains into monetary values. As a step in this direction, we evaluate models on 100 top-selling articles, which are likely to be stronger drivers of the retailer's profit compared to the rarely-sold articles. The results indicate that GraphDeepAR consistently improves the forecasting performance for the top-selling articles, achieving a 4.32% RMSE uplift on *retail* data and a 32.90% uplift on the *e-commerce* dataset.

**Table 3.** Results: adids Dataset

| Dataset | Subset | Financial uplift |
|---|---|---|
| adidas | Test set 1 | 5.42% |
| | Test set 2 | 4.05% |
| | Test set 3 | 1.66% |
| | Test set 4 | 0.52% |
| | Test set 5 | 0.33% |
| | Test set 6 | 0.32% |
| | Average | 2.05% |

The *adidas* dataset includes information needed to quantify the retailer's financial efficiency. Table 3 reports the financial uplift from GraphDeepAR across six article subsets. GraphDeepAR consistently outperforms DeepAR on all article splits, achieving the average financial uplift of 2.05%. This implies that performance gains from our approach are reflected not only in statistical metrics, but also in a tangible financial value improvement.

### 5.3 Runtime

Introducing a GNN encoder increases the computational complexity of the model. Table 4 compares the running time of DeepAR and GraphDeepAR across the three datasets. We provide the training and inference times and calculate the total running time difference as a percentage increase relative to DeepAR. Model training and inference is performed using the same Amazon SageMaker instance.

**Table 4.** Running Time Differences

| Dataset | Model | Training time | Inference time | Total difference |
|---|---|---|---|---|
| Retail | DeepAR | 10.80 min | 0.14 min | 160.96% |
| | GraphDeepAR | 28.33 min | 0.22 min | |
| E-commerce | DeepAR | 90.28 min | 3.26 min | 154.64% |
| | GraphDeepAR | 234.73 min | 3.46 min | |
| adidas | DeepAR | 55.92 min | 20.69 min | 120.28% |
| | GraphDeepAR | 139.80 min | 28.96 min | |

As expected, GraphDeepAR exhibits a longer running time. The largest differences are observed during training, where GraphDeepAR is around 159% slower due to the need to backpropagate gradients through additional GNN layers. Interestingly, inference with GraphDeepAR is only around 34% slower, indicating that leveraging graph-structured data on the prediction stage requires less additional resources. Note that the absolute training times of GraphDeepAR are not prohibiting from regularly retraining the forecasting model on a weekly or monthly basis to dynamically update the demand projections. This implies that in practice, using GraphDeepAR will incur substantially higher costs only if the forecasting model has to be retrained more frequently.

Compared to DeepAR, GraphDeepAR requires an additional data processing step that involves pairwise article similarity calculation and graph construction. In our experiments, the time required to build and export the article graph varied between 5 and 52 min across the three datasets. Note that this calculation only needs to be done once and can be reused for different model configurations as long as no new articles are introduced by a retailer.

### 5.4 Article Embeddings

GraphDeepAR can be used to produce time-varying article embeddings. The embeddings $(G_i)_{i=1}^N$ outputted by the GNN encoder are learned during training in an end-to-end fashion and leverage data contained in node and edge features to produce article representations useful for demand forecasts. The resulting embeddings can be used in further downstream tasks, such as pricing problem.

Figure 3 illustrates GraphDeepAR's embeddings for a subset of articles from the *retail* dataset for two time steps. Embeddings are mapped onto a two-dimensional space using UMAP [17] and are depicted as a scatter plot in the

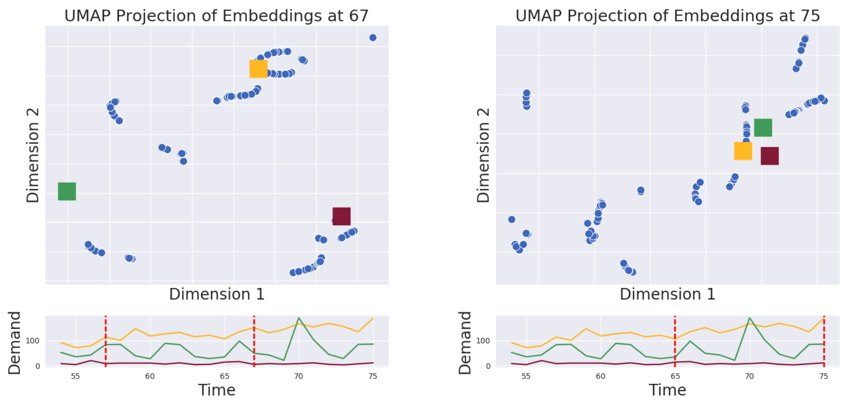

**Fig. 3.** GNN article embeddings for *retail* for week 67 (left) and 75 (right).

upper part of the figure. We highlight three neighboring articles with square markers. The lower part of the figure shows demand for the highlighted articles and a box indicating the time window covered by the embedding.

As shown in Figure 3, in week 67 (left panel), demand correlation between the neighboring articles is negative. Here, despite showing a high attribute-based similarity, the articles are located far away from each other in the 2D projection of the embedding space. At the same time, in week 75 (right panel), where articles' demand goes in the same direction, the articles move closer to each other. This illustrates that GNN embeddings integrate two types of information: initial article features used for graph construction and demand dynamics of the neighboring articles. This makes the embeddings useful for other prediction tasks, by serving as a proxy to indicate substitute and complement articles, under appropriate conditions.

## 6 Conclusion

This paper proposes a novel demand forecasting approach that leverages Graph Neural Networks (GNNs) to account for article relationships during training and inference. We introduce an end-to-end GraphDeepAR model that provides probabilistic demand predictions and avoids reliance on a pre-defined graph structure for graph construction. The approach is tested on three real-world datasets. The results indicate that GraphDeepAR outperforms a non-graph benchmark in statistical metrics and financial value, demonstrating the added value of integrating the article graph. We also show that GraphDeepAR produces dynamic GNN embeddings that can represent articles in different downstream prediction tasks.

Empirical comparisons provide guidelines for decision-makers, revealing domains in which retailers are more likely to benefit from a graph-based approach. Across the two public datasets, we observe higher gains on a larger dataset, which indicates that GraphDeepAR has higher potential given a large number of related articles. We also observe consistent accuracy gains on articles that have neighbors in the graph. Constructing a well-connected graph is, therefore, an important requirement, since GraphDeepAR requires edge connections to leverage patterns observed for related articles. The good performance of GraphDeepAR on top-selling articles indicates that our approach is able to improve forecasting accuracy on the popular articles that drive the retailer's revenue.

The accuracy gains come at a cost of longer running times. On average, GraphDeepAR is 159% slower during the training stage and 34% slower during inference. This indicates that our approach incurs higher costs if the forecasting model has to be retrained frequently, whereas producing predictions with GraphDeepAR requires a reduced number of additional resources.

The proposed approach introduces additional hyperparameters into DeepAR, including graph construction (e.g., similarity measure and cutoff) and GNN encoder (e.g., embedding sizes and neighborhood sampling ratios). The future research could perform ablation studies to investigate the impact of important hyperparameters on the model performance and training time.

Using the GNN encoder implies that demand predictions are affected by all neighboring articles, which makes GraphDeepAR useful for simulating certain scenarios. One promising direction would be to investigate how forecasts are affected by introducing or removing articles, which requires reconstructing the graph at the inference time. Another idea concerns exploring alternative ways of modeling article relationships. For example, historical demand correlation could serve as a proxy to build connections if article attribute data is not available.

## References

1. Bai, L., Yao, L., Kanhere, S.S., Wang, X., Sheng, Q.Z.: Stg2seq: Spatial-temporal graph to sequence model for multi-step passenger demand forecasting (5 2019), http://arxiv.org/abs/1905.10069
2. Beitner, J., The PyTorch Forecasting team: PyTorch Forecasting (2022), https://github.com/jdb78/pytorch-forecasting
3. Benidis, K., Rangapuram, S.S., Flunkert, V., Wang, Y., Maddix, D., Turkmen, C., Gasthaus, J., Bohlke-Schneider, M., Salinas, D., Stella, L., et al.: Deep learning for time series forecasting: Tutorial and literature survey. ACM Computing Surveys **55**(6), 1–36 (2022)
4. Box, G.E.P., Cox, D.R.: An analysis of transformations. Journal of the Royal Statistical Society: Series B (Methodological) **26** (1964)
5. Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., Huang, C., Tong, Y., Xu, B., Bai, J., Tong, J., Zhang, Q.: Spectral temporal graph neural network for multivariate time-series forecasting. Advances in Neural Information Processing Systems **33**, 17766–17778 (2020)
6. Falcon, W., The PyTorch Lightning team: PyTorch Lightning (2022), https://github.com/Lightning-AI/lightning
7. Gandhi, A., Aakanksha, Kaveri, S., Chaoji, V.: Spatio-temporal multi-graph networks for demand forecasting in online marketplaces. vol. 12978 LNAI (2021)
8. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. Advances in neural information processing systems **30** (2017)

9. Huber, J., Stuckenschmidt, H.: Daily retail demand forecasting using machine learning with emphasis on calendric special days. International Journal of Forecasting **36**(4), 1420–1438 (2020)
10. Keenan, M.: Global ecommerce explained: Stats and trends to watch (2022), https://www.shopify.com/enterprise/global-ecommerce-statistics#:~:text=According%20to%20research%20completed%20by,22.3%25%20of%20total%20retail%20sales. Last accessed 15 Mar 2023
11. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings (9 2016)
12. Lai, G., Chang, W.C., Yang, Y., Liu, H.: Modeling long- and short-term temporal patterns with deep neural networks. 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2018 pp. 95–104 (6 2018)
13. Lim, B., Arık, S., Loeff, N., Pfister, T.: Temporal fusion transformers for interpretable multi-horizon time series forecasting. International Journal of Forecasting **37**, 1748–1764 (10 2021)
14. Lim, B., Zohren, S.: Time-series forecasting with deep learning: a survey. Philosophical Transactions of the Royal Society A **379** (4 2021)
15. Lira, H., Martí, L., Sanchez-Pi, N.: A graph neural network with spatio-temporal attention for multi-sources time series data: An application to frost forecast. Sensors **22** (2022)
16. Maas, A.L., Hannun, A.Y., Ng, A.Y., et al.: Rectifier nonlinearities improve neural network acoustic models. In: Proc. icml. vol. 30, p. 3. Atlanta, Georgia, USA (2013)
17. McInnes, L., Healy, J., Melville, J.: Umap: Uniform manifold approximation and projection for dimension reduction. arXiv preprint arXiv:1802.03426 (2018)
18. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics (11 2019)
19. Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T.: Deepar: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting **36**, 1181–1191 (7 2020)
20. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108 (2019)
21. Sen, R., Yu, H.F., Dhillon, I.S.: Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. Advances in Neural Information Processing Systems **32** (2019)
22. Veličković, P.: Everything is connected: Graph neural networks. Current Opinion in Structural Biology **79**, 102538 (2023)
23. Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., Zhang, Z.: Deep graph library: A graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315 (2019)
24. Wu, Z., Pan, S., Long, G., Jiang, J., Zhang, C.: Graph wavenet for deep spatial-temporal graph modeling. IJCAI International Joint Conference on Artificial Intelligence **2019-August**, 1907–1913 (5 2019)
25. Yu, B., Yin, H., Zhu, Z.: Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. IJCAI International Joint Conference on Artificial Intelligence **2018-July**, 3634–3640 (9 2017)
26. Zivot, E., Wan, J.: Vector autoregressive models for multivariate time series. Modeling Financial Time Series with S-PLUS® pp. 385–429 (10 2006)

## Supplementary Material

### Appendix A: Meta-Parameters

Table 5 provides the used meta-parameter values of DeepAR and GraphDeepAR on *retail* and *e-commerce* datasets. We split meta-parameters into three groups: sequential model, GNN encoder, and the training procedure. Note that the GNN encoder meta-parameters are only relevant for the GraphDeepAR model.

**Table 5.** Model Meta-Parameters

| Dataset | Component | Meta-parameter | DeepAR | GraphDeepAR |
|---|---|---|---|---|
| retail | Sequential model | No. layers | 2 | 2 |
| | | Hidden size | [128, 128] | [128, 128] |
| | | Cell type | LSTM | LSTM |
| | | Dropout | 0.2 | 0.2 |
| | | Context length | 10 | 10 |
| | GNN encoder | No. layers | – | 2 |
| | | Hidden size | – | [16, 8] |
| | | Cell type | – | GCN |
| | | Dropout | – | 0.2 |
| | | Similarity cutoff | – | 0.95 |
| | | Max no. neighbors | – | 10 |
| | | Context length | – | 10 |
| | Training procedure | Max no. epochs | 50 | 50 |
| | | Early stopping | 5 | 5 |
| | | Learning rate | $5 \times 10^{-3}$ | $5 \times 10^{-3}$ |
| | | Optimizer | Ranger | Ranger |
| | | Loss function | t-distribution | t-distribution |
| | | Batch sampler | Random | Synchronized |
| e-commerce | Sequential model | No. layers | 2 | 2 |
| | | Hidden size | [128, 128] | [128, 128] |
| | | Cell type | LSTM | LSTM |
| | | Dropout | 0.2 | 0.2 |
| | | Context length | 10 | 10 |
| | GNN encoder | No. layers | – | 2 |
| | | Hidden size | – | [16, 8] |
| | | Cell type | – | GCN |
| | | Dropout | – | 0.2 |
| | | Similarity cutoff | – | 0.95 |
| | | Max no. neighbors | – | 5 |
| | | Context length | – | 10 |
| | Training procedure | Max no. epochs | 50 | 50 |
| | | Early stopping | 5 | 5 |
| | | Learning rate | $1 \times 10^{-2}$ | $1 \times 10^{-2}$ |
| | | Optimizer | Ranger | Ranger |
| | | Loss function | t-distribution | t-distribution |
| | | Batch sampler | Random | Synchronized |

## Appendix B: GraphDeepAR Predictions

Figure 4 depicts several individual demand predictions produced by the DeepAR and GraphDeepAR models on the *retail* dataset. As illustrated on the figure, GraphDeepAR is able to provide better forecasts for some of the articles.

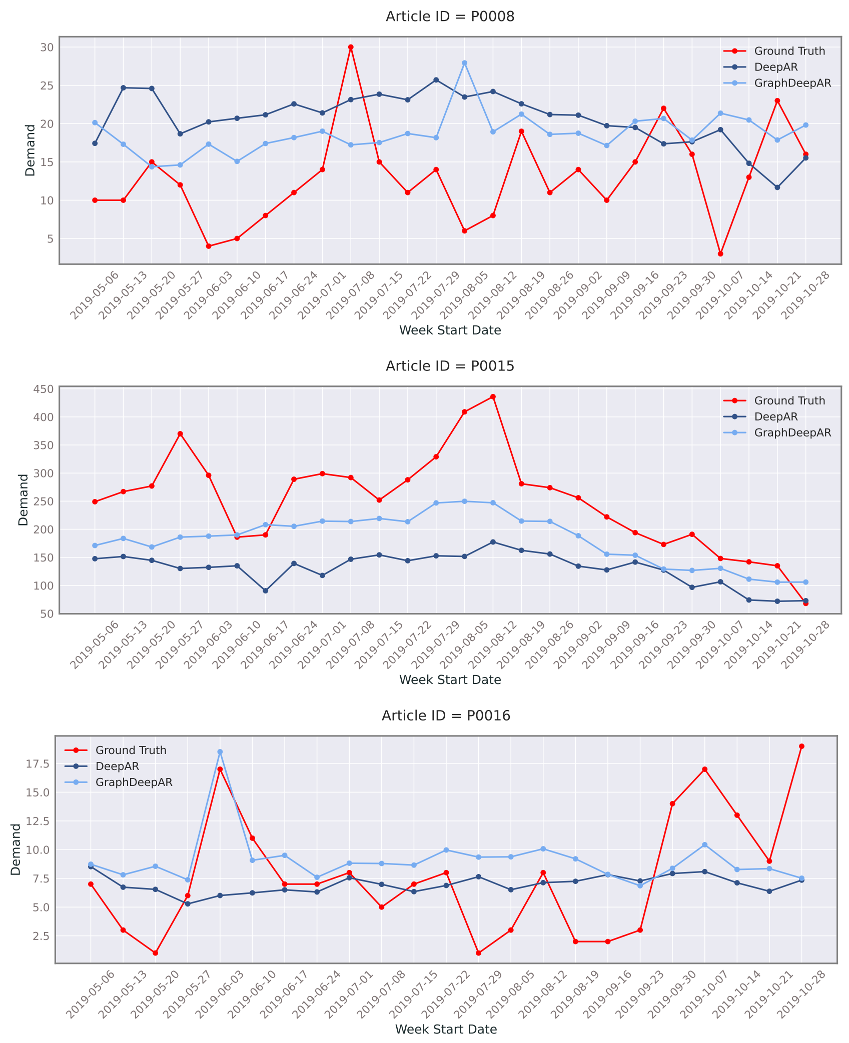

**Fig. 4.** Example predictions of DeepAR and GraphDeepAR on the *Retail* dataset.