Spatio-temporal Multi-graph Networks for Demand Forecasting in Online Marketplaces

Ankit Gandhi[1], Aakanksha[2,⋆], Sivaramakrishnan Kaveri[1], and Vineet Chaoji[1]

[1] Amazon – India Machine Learning, Bengaluru, India
{ganankit,kavers,vchaoji}@amazon.com
[2] Microsoft, Hyderabad, India, aakanksha@microsoft.com

Abstract. Demand forecasting is fundamental to successful inventory planning and optimisation of logistics costs for online marketplaces such as Amazon. Millions of products and thousands of sellers are competing against each other in an online marketplace. In this paper, we propose a framework to forecast demand for a product from a particular seller (referred as offer/seller-product demand in the paper). Inventory planning and placements based on these forecasts help sellers in lowering fulfilment costs, improving instock availability and increasing shorter delivery promises to the customers. Most of the recent forecasting approaches in the literature are one-dimensional, i.e, during prediction, the future forecast mainly depends on the offer i.e. its historical sales and features. These approaches don't consider the effect of other offers and hence, fail to capture the correlations across different sellers and products seen in situations like, (i) competition between sellers offering similar products, (ii) effect of a seller going out of stock for the product on competing seller, (iii) launch of new competing products/offers and (iv) cold start offers or offers with very limited historical sales data. In this paper, we propose a general demand forecasting framework for multivariate correlated time series. The proposed technique models the homogeneous and heterogeneous correlations between sellers and products across different time series using graph neural networks (GNN) and uses state-of-the-art forecasting models based upon LSTMs and TCNs for modelling individual time series. We have experimented with various GNN architectures such as GCNs, GraphSAGE and GATs for modelling the correlations. We applied the framework to forecast the future demand of products, sold on Amazon, for each seller and we show that it performs ∼16% better than state-of-the-art forecasting approaches.

Keywords: Demand forecasting in e-commerce · Time-series forecasting · Graph neural networks · Correlated Multivariate time series.

1 Introduction

Forecasting product demand for different sellers is important for e-commerce marketplaces for successful inventory planning and optimizing supply chain costs.

⋆ work was done as part of internship at Amazon

These forecasted demand recommendations are then used by sellers to stock inventory in their warehouses or fulfilment centres. In online marketplaces such as Amazon, there are hundreds of thousands of sellers offering millions of products. A single product can be offered by multiple sellers and a seller can sell multiple products. Demand for a particular offer not only depends on its historical sales but also on other factors such as competition with other sellers, other sellers offering same/similar product going out of stock, other sellers increasing or decreasing the price, launch of new competing products, etc. In order to accurately predict offer level demand, it is imperative to capture the correlations between different offers in the model.

In e-commerce, the demand is highly dynamic and often fluctuating because of holidays, deals, discounts, intermittent offer campaigns, competitor trends, etc. Recent works for e-commerce demand forecasting based on neural networks [21, 20, 25, 19] have shown that they significantly outperform traditional forecasting models such as ARIMA [4, 5] and exponential smoothing [13]. The traditional methods are univariate and forecast for each time series in isolation. Whereas in e-commerce, products are often related in terms of grouping, categories or sub-categories, and hence, their demand patterns are correlated. Neural networks take into account these correlations using dynamic historical attributes and static covariates to extract higher order features, and identify complex patterns within and across time series.

Even though these deep models are trained on all offers to capture these correlations, during prediction they only focus on using an offer's historical time series data to predict the future time series. However, for offer demand forecasting, looking into other time series may be useful during prediction time, for instance, it might be beneficial to look at (i) out of stock status for the same product from other sellers, (ii) launch of competing/similar products, (iii) price increase/decrease from other sellers offering same product, (iv) performance of competing seller going up/down suddenly (rating, shipping, reviews, etc.). In the past, authors in [15], proposed a combination of convolution and recurrent connections that takes multiple time series as input to the network, thus capturing some of the above scenarios during prediction. However, it doesn't scale beyond a few time series as the input layer size grows. In [22], authors propose a scalable network that can leverage both local and global patterns during training and prediction. They combine a global matrix factorization model over all time series regularized by a temporal convolution network with another temporal network to capture local properties of each time-series and associated covariates. In this paper, we propose a more systematic method of modelling the correlation between different entities across time series using GNNs.

Graphs are an extremely powerful tool to capture and represent interactions between entities in a seamless manner. The entities (sellers and products) can be represented as nodes of a graph and their direct correlation can be represented by edges (refer to Section 3.2 for graph construction). This results in a multimodal[3]

[3] Graph having different kinds of nodes (sellers, products)

and multi-relational[4] graph. In addition, nodes and edges can be represented using a set of historical features, characterizing their intrinsic properties. Recently, researchers have proposed various methods that are capable of learning from graph-structured data [6, 12, 14, 18, 27, 28, 30, 10, 23, 11]. All of these methods are based on Graph Convolutional Networks (GCNs) and its extensions. GCNs learn to aggregate feature information from local graph neighborhoods using neural networks. These methods have been shown to boost the performance of many graph-level tasks such as node classification, link prediction, graph classification, subgraph classification, etc.

In this work, we propose a framework for performing demand forecasting in multivariate correlated time series data. We model the homogeneous and heterogeneous correlations between different sellers and products across time series at the time of training as well as prediction, and inject the learned representations into state-of-the-art neural network architectures [1, 25, 8] for demand forecasting. At each time step, we define a graph structure based on seller attributes, product attributes, offer demand, product similarity/substitute [24, 17], and obtain the seller and product representations. The edge structure in the graph vary over time based upon demand and other connections. Thus, an unrolled version of the network comprises of multiple graphs (referred to here as multi-graph networks) sharing the same parameters (refer to Figure 1). The sequence of representations from GNNs along with historical demand is then fed into sequential neural model such as LSTMs, TCNs, etc., to forecast the future demand. We experiment with different variations of GNN architectures such as GCNs [14], GraphSAGE [11], GAT [23], etc. by incorporating various node and edge features to learn the seller and product embeddings at each time step. While the LSTM/TCN modules make use of just the sequential information present in the data, our aim is to augment these modules with correlations across time series learnt using GNNs. We train the complete spatio-temporal multigraph network in an end-to-end fashion, where embeddings from the GNN layer are fed into the sequential model to make demand forecasts, and the loss is optimized over the entire network. Following are the main contributions of the paper – (i) a generic framework for handling competing/correlated time series in demand forecasting, (ii) use of GNNs for modelling the effect of sellers and products on each other in online marketplaces during training and prediction, (iii) the framework can be plugged into any state-of-the-art sequential model for better demand forecasts, (iv) extension of standard GNN architectures to heterogeneous graphs by leveraging edge features and (v) empirical evaluation of framework using real world marketplace data from Amazon against other forecasting techniques.

We evaluate the framework for forecasting demand of offers sold on Amazon marketplace on a dataset comprising of $21K$ sellers and $1.89MM$ products, and show that the proposed models have $\sim$16% lower mean absolute percentage error (MAPE) than state-of-the-art demand forecasting approaches used in e-commerce. For products that are sold by more than one seller, the improvement is $\sim$30%, and for cold and warm start offers (that has history of less than 3 months), the

[4] Graph having multiple types of edges between nodes (in-stock, product substitute)

improvement is ∼25%. The rest of the paper is organized as follows. Section 2 provides an overview of the extensive literature on forecasting models, especially for e-commerce and correlated time series. In Section 3, we present the proposed framework of spatio-temporal multi-graph networks for demand forecasting. In Section 4, we compare the performance of the proposed model and its variants with the state-of-the-art approaches for demand forecasting as well as some of the implementation details. We conclude with the final remarks in Section 5.

2 Prior Work

Time series forecasting is a key component in many industrial and business decision processes, and hence, a wide variety of different forecasting methods have been developed in the past. ARIMA models [4, 5] and state space models [13, 9] have been well established de-facto forecasting models in the industry. However, for retail demand forecasting they don't seem to work well as they cannot infer shared patterns from a dataset of similar time series. Deep neural network based forecasting solutions provide an alternative [21, 20, 25, 1, 22]. In this section, we mostly focus on recent deep learning approaches. Benidis et al. provide an excellent summary and review of various neural forecasting approaches in [3]. DeepAR [21] proposed an auto regressive recurrent neural network model on a large number of related time series to estimate the probability distribution of future demand. DeepState [20] models time series forecasting by marrying state space models with deep recurrent neural networks to learn complex patterns from data while retaining the interpretability. Wen et al. [25] proposed a multi-horizon quantile recurrent forecaster where the time series history is modelled using LSTMs, and an MLP is used to decode the input into multi horizon demand forecasts. LSTNet [15] uses a combination of CNN and RNN to extract short-term temporal patterns as well as correlations among variables. Chen et al. [8] proposed a probabilistic framework with temporal convolutional neural networks for forecasting based on dilated causal convolutions. DeepGLO [22] is a hybrid model that combines a global matrix factorization model regularized by a temporal convolution network and another temporal network to capture the local properties of time series and associated covariates. There have been methods to take into account correlation between time series like DeepGLO [22], LSTNet [15], etc., however, we provide a more systematic method of modelling the correlations between different entities in the time series using GNNs.

There have been few works in the past focusing specifically on retail demand forecasting. Mukherjee et al. [19] developed an MLP and LSTM based architecture for the eRetail company – Flipkart, and outputs the probability distribution of future demand as a mixture of Gaussians. Bandara et al. [2] built an LSTM based network for forecasting on real world dataset from Walmart. However, none of the previous works focus on demand forecasting for 'marketplaces', where multiple sellers are involved and explicitly model the correlations in their time-series.

There are also a few prior works that use GNNs for demand forecasting focusing mainly on the application of traffic forecasting. DCRNN [16] incorporates both

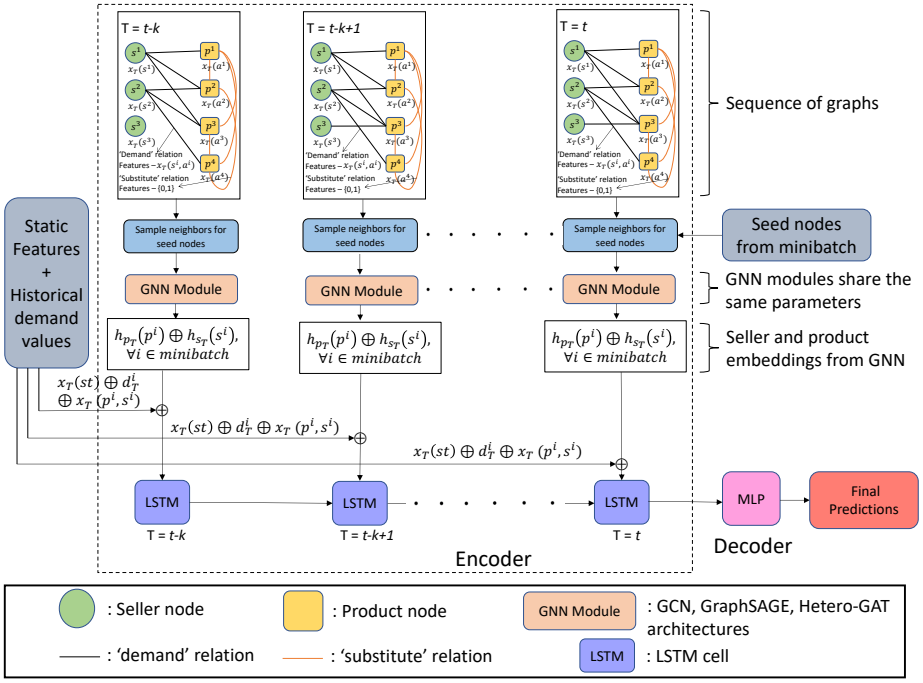

Fig. 1. Training architecture of the demand forecasting model for online marketplaces. At each time-step, a graph is defined between seller and product nodes that models the correlation between them using 'demand' and 'substitute' relations. From these graphs, product and seller embeddings are learned using GNN layer. The embeddings from GNN are then concatenated with static features and demand value for that time-step, and fed into the sequential model (in this case, LSTM) to forecast offer-level demand.

spatial and temporal dependency in the traffic flow using diffusion convolution and RNNs for traffic forecasting. ST-GCN [29] is a deep learning framework that integrates graph convolution and gated temporal convolution through spatio-temporal convolutional blocks for traffic forecasting. GraphWaveNet [26] captures spatio-temporal dependencies by combining graph convolution with dilated casual convolution. StemGNN [7] is a spectral temporal GNN that captures inter series correlations and temporal dependencies jointly in the spectral domain for demand forecasting. To the best of our knowledge, this is the first work, that predicts offer demand for online marketplaces by explicitly modelling the correlations between different sellers and products in the time series, and accounting for their effects on each other.

3 Proposed Method

This section describes the problem formulation and the technical details of the network architecture employed to solve the problem of demand forecasting in

online marketplaces. We start with the problem formulation, and by describing the general graph structure for modelling the correlations between time series containing different sellers and products. In Section 3.3, we describe various GNN architectures to produce node (seller and product) representations and how they have been adapted for our problem. Finally, we describe some of the sequential models that have been considered for the experimentation for extracting the temporal patterns in the historical time series data. Figure 1 represents the overall architecture of the proposed method.

3.1 Problem Formulation

Let $\mathcal{S}$ denote the set of sellers and $\mathcal{P}$ denote the set of products in the marketplace. Given a set of N time series, each series comprising of a seller, a product and historical demand, $< s^i, p^i, [y^i_{t-k,t}] >_{i=1}^{N}$, where $y^i_{t-k,t} = [d^i_{t-k}, d^i_{t-k+1}, \dots d^i_t]$, d^i_t denotes the demand for a seller-product at time t, k represents the length of the time series, $s^i \in \mathcal{S}$ and $p^i \in \mathcal{P}$ denotes the seller and product respectively for i^{th} time series, and N is the number of time series. Our goal is to predict $[\hat{y}^i_{t+1,t+K}]_{i=1}^{N}$, where $\hat{y}^i_{t+1,t+K} = [\hat{d}^i_{t+1}, \hat{d}^i_{t+2}, \dots \hat{d}^i_{t+K}]$, the demand for future K time steps. Let x^i_t be the feature vector for i^{th} time series at time t. We can break down this feature vector into four different components, $x^i_t = [x_t(p^i), x_t(s^i), x_t(p^i, s^i), x_t(st)]$–
(i) $x_t(p^i)$ denotes the features specific to product only such as product brand, product category, total product sales in trailing months, total product gross merchandise sales (GMS) in trailing months, product shipping charges, number of product views/clicks in trailing months, etc.,
(ii) $x_t(s^i)$ denotes the features specific to seller only such as seller rating, seller performance metrics, seller reviews, total GMS in trailing months for the seller, total sales by the seller in trailing months, total views of all the products offered by a seller in trailing months, etc.,
(iii) $x_t(p^i, s^i)$ denotes the features dependent on both seller and product of the time series such as total views of p^i offered by s^i in trailing months, total sales of p^i offered by s^i in trailing months, total GMS of p^i offered by s^i in trailing months, whether product belongs to seller's main category/subcategory, out of stock history of p^i from s^i, etc., and
(iv) $x_t(st)$ denotes static features independent of the seller as well as product such as the number of days to the nearest holidays that have significant impact on the future demand, bank related offers and cashbacks on the platform, etc.

We formulate the demand forecasting for an offer as a regression problem where we have to predict the future demand ($\hat{y}^i_{t+1,t+K}$) given the historical demand ($y^i_{t-k,t}$), and the time series features ($x^i_{t-k:t}$).

3.2 Graph Construction

We represent correlation between different sellers and products across time series using graphs. Sellers and products are represented as nodes, and their interactions are represented as edges in the graph. For instance, products can be correlated to

each other in terms of their similarity or substitutability; sellers can be correlated to each other if they are selling similar products, their shipping channels are same (self-fulfilled vs marketplace-fulfilled), primary category of the products offered by them is same, etc. The goal is to generate accurate embeddings or representations of the sellers and products, in order to use them effectively to predict their demand for a specified period in the future. To learn these embeddings for time t, in this work, we construct a graph $\mathcal{G}_t = ([\mathcal{P}, \mathcal{S}], \mathcal{E}_t)$ consisting of nodes in two disjoint sets, namely, $\mathcal{S}$ (representing sellers) and $\mathcal{P}$ (representing the products they offer), and the set of edges $\mathcal{E}_t$ with following connections to capture their correlatedness:
(i) Demand edge: It is defined between seller s^i and product p^i, $1 \leq i \leq N$, at time t, if there exists a demand for product p^i from seller s^i at time t. This edge models the dynamic connections in the graphs.
(ii) Substitute edge: It is defined between two products, p^i and p^j, representing their similarity or substitutability. This edge models the static connections in the graphs and is not dependent on time.

These graphs are constructed for each time step of the historical data, i.e., there are as many graphs as the number of time steps $(\mathcal{G}_{t-k}, \mathcal{G}_{t-k+1}, \ldots, \mathcal{G}_t)$. In addition to the graph structure, we utilise x_t^i (defined in Section 3.1) as the input features of the nodes and edges in the graph $\mathcal{G}_t$. The seller node i in the graph $\mathcal{G}_t$ is initialized with seller specific feature – $x_t(s^i)$, the product node i is initialized with product specific feature – $x_t(p^i)$, demand edges (if exist) are initialized with seller-product features – $x_t(p^i, s^i)$, whereas the substitute edges are just binary connections with no features. Hence, we efficiently utilise the seller and product characteristics in conjunction with the graphical information and edge features to produce high-quality embeddings. These embeddings are then fed into a time series model for the generation of accurate forecasts. Figure 1 represents the sequence of graphs constructed for modelling the correlation between sellers and products over time.

3.3 Graph Neural Networks

In this section, we present the details of various GNN architectures and their adaptations that we have employed to generate representations for seller and product nodes in our graphs. The basic unit for all the architectures is GCN, which uses localized convolutional modules that capture information from the node's neighborhood. Following subsections describe all the architectures in detail. We empirically evaluate each of these methods in Section 4 on a real-world online marketplace dataset. We create graph for each timestep t separately, hence, dropping the subscript t from the notation in this section.

Graph Convolutional Networks: Using GCNs, we capture the homogeneous correlations in the graph. In this case, we construct a graph with only one type of node and edge. Since, we have two types of nodes in the graph – sellers and products, a 2-layer MLP is used for the input features to map them to the same dimension before GCN layer, so that the nodes can be treated homogeneously.

Also, we consider only the demand relation in this homogeneous graph. The idea behind GCNs is that it learns how to transform and propagate information across the graph, captured by node feature vectors [14]. To generate embedding for a node, it uses the localized convolutional module that captures information from the node's neighborhood. In GCNs, we stack multiple such convolutional modules to capture information about the graph topology. Initial node features are provided as input to the GCN and then, the node embeddings are computed by applying the series of convolutional modules.

Let $h^l(i)$ denote the embedding of i^{th} node at l^{th} layer from graph $\mathcal{G}$. Then, $h^l(i)$ can be computed as follows -

$$h^l(i) = \sigma\left(\frac{1}{\mid \mathcal{N}(i) \mid} \sum_{u \in \mathcal{N}(i)} W^l h^{l-1}(u)\right) \tag{1}$$

where, $\mathcal{N}(i)$ denotes the neighborhood of node i, and W^l is the learnable weight for layer l that is shared across nodes. This technique is referred as Homo-GCN in the paper. Edge features $(x_t(p^i, s^i))$ are not used in this formulation, only a binary relation is considered.

GraphSAGE (GS): A seller can offer thousands of products. And if the seller has demand for all the products, then it is very inefficient to take the full size of a seller's neighborhood for learning its representation as is done in GCN. In GS [11], a sampling mechanism is adopted to obtain a fixed number of neighbors for each node, which makes it suitable for representation learning in such large graphs. This technique also captures only the homogeneous correlations. It performs graph convolutions and obtains the node embeddings in the following manner –

$$h^l(i) = \sigma\left(W^{'l} h^{l-1}(i) + \frac{1}{\mid S_{\mathcal{N}(i)} \mid} \sum_{u \in S_{\mathcal{N}(i)}} W^l h^{l-1}(u)\right) \tag{2}$$

where, $S_{\mathcal{N}(i)}$ is a random sample of the i^{th} node neighbors, and W^l & $W^{'l}$ are the learnable weights shared across all the nodes. Note that we perform aggregation on neighborhood information using the mean function. However, any alternate function can be employed for this operation. For GS formulation also, we ignore the edge features, and map the seller and product features using a 2-layer MLP network to the same dimension, to treat the nodes homogeneously. We refer to this method as Homo-GS.

Heterogeneous GraphSAGE with Edge Features: The graphs built in Section 3.2 to capture the correlatedness are inherently heterogeneous in nature (attributing to the presence of two node types – 'sellers' and 'products', and two relation types – 'demand' and 'substitute'). Homo-GCN and Homo-GS methods considered above treat the neighborhood of seller and product nodes in a similar manner. In this formulation, we extend Homo-GS to heterogeneous graphs where a separate convolutional module for each relation type is learned. Let us denote

the hidden representation of i^{th} seller node at layer l by $h_s^l(i)$ and i^{th} product node at layer l by $h_p^l(i)$. Then, we obtain the seller representations using following –

$$h_s^l(i) = \sigma(W_s^{'l} h_s^{l-1}(i) + \underbrace{\frac{1}{\mid S_{\mathcal{N}_s(i)} \mid} \sum_{u \in S_{\mathcal{N}_s(i)}} W_p^l [h_p^{l-1}(u) \overbrace{\oplus x(u,i)]}^{\text{edge features}}}_{\text{aggregation under relation demand}}) \tag{3}$$

where, $S_{\mathcal{N}_s(i)}$ is a random sample of the i^{th} seller node neighbors under relation 'demand' (neighbors for seller nodes would be product nodes), W_p^l is the learnable weight matrix under relation demand and $W_s^{'l}$ is the learnable self weight matrix for seller nodes. Before performing the aggregation over product nodes, we also concatenate (denoted by $\oplus$) the edge features $x(u,i)$ with the product embeddings.

Product representations are computed by aggregation under two relations – demand and substitute, using the following equation –

$$h_p^l(i) = \sigma \left(W_p^{'l} h_p^{l-1}(i) + \underbrace{\frac{1}{\mid S_{\mathcal{N}_p(i)} \mid} \sum_{u \in S_{\mathcal{N}_p(i)}} W_s^l [h_s^{l-1}(u) \oplus x(i,u)]}_{\text{aggregation under relation demand}} + \underbrace{\frac{1}{\mid S_{\mathcal{N}_p''(i)} \mid} \sum_{u \in S_{\mathcal{N}_p''(i)}} W_p^{''l} h_p^{l-1}(u)}_{\text{aggregation under relation substitute}} \right) \tag{4}$$

where, $S_{\mathcal{N}_p(i)}$ is a random sample of the i^{th} product node neighbors under relation 'demand' (neighbors of product nodes would be seller nodes), W_s^l is the learnable weight matrix under relation demand, $W_p^{'l}$ is the learnable self weight matrix for product nodes, $S_{\mathcal{N}_p''(i)}$ is a random sample of the i^{th} product node neighbors under relation 'substitute' (neighbors for product nodes would be product nodes), and $W_p^{''l}$ is the learnable weight matrix under relation 'substitute'. Here also, we concatenate the 'demand' edge features as explained above. This architecture is referred as Hetero-GS-Demand-Substitute.

Heterogeneous Graph Attention Networks with Edge Features: In all the above architectures, we compute hidden representations of seller and product nodes by assigning equal importance to all their neighboring nodes. In the context of demand forecasting, we may want to assign higher weight to more similar products or to products that have higher demand in the neighborhood while learning the representation, as that will help in capturing the correlation between nodes in a better way and accurately predicting the future demand. GAT [23]

networks allow us to compute the hidden representation of each node in the graph by attending over its neighbors, following a self-attention strategy. As opposed to GCN or GS, GAT allows for implicitly assigning different importances to neighboring nodes. We compute the seller embeddings by extending Hetero-GS-Demand-Substitute architecture using attention weights –

$$h_s^l(i) = \sigma\left(W_s^{'l} h_s^{l-1}(i) + \frac{1}{\mid S_{\mathcal{N}_s(i)} \mid} \sum_{u \in S_{\mathcal{N}_s(i)}} \underline{\alpha_{iu}^l} W_p^l [h_p^{l-1}(u) \oplus x(u,i)]\right) \tag{5}$$

where α_{iu}^l denotes the attention weights for l^{th} layer and is computed as follows –

$$e_{iu}^l = LeakyReLU(a^{l^T}.[h_p^{l-1}(u) \oplus h_s^{l-1}(i) \oplus x(u,i)]), \qquad \alpha_{iu}^l = \frac{\exp(e_{iu}^l)}{\sum_{v \in S_{\mathcal{N}_s(i)}} \exp(e_{iv}^l)} \tag{6}$$

where a^l is the shared learnable linear transformation applied to every node. Attention scores are computed by utilizing source node embeddings, destination node embeddings and the edge features between source and destination as shown in Equation 6.

Likewise, by modifying the Equation 4, we obtain the product representations as follows –

$$\begin{aligned} h_p^l(i) = \sigma\Bigg(W_p^{'l} h_p^{l-1}(i) &+ \frac{1}{\mid S_{\mathcal{N}_p(i)} \mid} \sum_{u \in S_{\mathcal{N}_p(i)}} \underline{\alpha_{iu}^{'l}} W_s^l [h_s^{l-1}(u) \oplus x(i,u)] \\ &+ \frac{1}{\mid S_{\mathcal{N}_p''(i)} \mid} \sum_{u \in S_{\mathcal{N}_p''(i)}} \underline{\alpha_p^{''l}} W_p^{''l} h_p^{l-1}(u)\Bigg) \end{aligned} \tag{7}$$

where $\alpha_{iu}^{'l}$ and $\alpha_p^{''l}$ denote the attention weights and are computed in a similar manner as shown in Equation 6. This architecture is referred as Hetero-GAT-Demand-Substitute in the experimental section.

3.4 Sequential Model

The seller and product representations are obtained from the above GNN module for each timestep of the time series data. Given the tuple$< s^i, p^i, y_{t-k,t}^i >$ (refer to Section 3.1), the sequence of i^{th} product and i^{th} seller representations – $(h_{p_{t-k}}(i), h_{p_{t-k+1}}(i), \ldots, h_{p_t}(i))$ and $(h_{s_{t-k}}(i), h_{s_{t-k+1}}(i), \ldots, h_{s_t}(i))$ is obtained from the GNN module using graphs $\mathcal{G}_{t-k}, \mathcal{G}_{t-k+1}, \ldots, \mathcal{G}_t$ respectively. These embeddings are aggregated with the demand, static features and the available seller-product features, which are then fed into the sequential model for predicting the future demand. The final input to the sequential model is given by –

$$\begin{aligned} \big(&h_{p_{t-k}}(i) \oplus h_{s_{t-k}}(i) \oplus x_{t-k}(p^i, s^i) \oplus x_{t-k}(st) \oplus d_{t-k}^i, \\ &\ldots, h_{p_t}(i) \oplus h_{s_t}(i) \oplus x_t(p^i, s^i) \oplus x_t(st) \oplus d_t^i\big)_{i=1}^N \end{aligned} \tag{8}$$

This module consists primarily of a sequential network to capture the temporal characteristics of the historical demand data. We employ an encoder-decoder based Seq2Seq model for this purpose which has shown to outperform other models for e-commerce forecasting [2, 19, 25]. The sequential model we experimented with resembles the one used by Wen et al. [25], where a vanilla LSTM is used to encode the history into hidden state and an MLP is used as a decoder for predicting the future demand. Furthermore, MLP is used instead of LSTM as a decoder to avoid feeding predictions recursively as surrogate of ground truth, since it leads to error accumulation. Figure 1 represents the complete architecture of the model pictorially.

We also employed TCN [1] architecture for modelling the above sequential data. In the past, TCNs have been shown to outperform recurrent architectures across a broad range of sequence modelling tasks, and we observed the same for our task. TCNs perform dilated causal convolutions, and a key characteristic is that the output at time t is only convolved with the elements that occurred before t so that there is no information leakage. Some of the key advantages of TCNs are that they are easily parallelisable because of convolutional architecture, require less memory as compared to recurrent architectures and have flexible receptive field making them easy to adapt to different domains.

Note that in this work, we conducted our experiments using mainly LSTMs and TCNs. However, the GNN module can easily extend to any other suitable and relevant network for sequence modelling.

4 Experimental Results

This section outlines our experiments using different GNN architectures for demand forecasting in online marketplaces. We validate the proposed framework on a real-world dataset from Amazon for forecasting product demand for different sellers.

Dataset details: We use a random sample of the demand data (weekly sales of products from sellers) on Amazon from January 2016 to July 2018 for evaluating our models. For training the models, we use the demand data from January 2016 to July 2017 (∼1.5 years). For validation and testing, we use the demand data from August 2017 to Jan 2018 (6 months) and Feb 2018 to July 2018 (6 months) respectively to perform out of window evaluation. Table 1 shows the statistics on number of sellers and number of products on the random sample of demand data used for experimentation. We organize the time-series at weekly level, i.e., a single time-step in our models corresponds to demand for 1 week. We use the historical demand data for the last 2 years (104 weeks) for a seller and a product, to predict the future demand for next 4 weeks. Therefore, the length of our time series is $k = 104$ and we predict the demand for future $K = 4$ weeks in all our experiments.

Metrics: We mainly use mean absolute percentage error (MAPE) to compare our models. We compute the MAPE metric over a span of four weeks after skipping the week of forecast creation date. MAPE is defined as the average of

Table 1. Statistics on the train, validation and test dataset used for the experimentation

Dataset	No. of time series	No. of sellers	No. of products
Train	6,411,000	21,020	1,889,908
Validation	2,136,002	22,476	1,801,478
Test	2,154,694	23,700	1,812,206

absolute percentage errors over all time series in the test set, i.e.,
$MAPE = \frac{1}{M}\left(\sum_i L(\sum_{j=2}^{5} d^i_{t+j}, \sum_{k=2}^{5} \hat{d}^i_{t+j})\right)$ where, M = #time series in test set
$L(d, \hat{d}) = 1$, if $d = 0$ and $\hat{d} \neq 0$, $\quad L(d, \hat{d}) = 0$, if $d = 0$ and $\hat{d} = 0$,
$L(d, \hat{d}) = \frac{||d - \hat{d}||}{d}$, otherwise
Loss function: We train our multi-graph network in an end-to-end supervised manner using the quantile loss at the 50th percentile, also known as the P50 loss. A quantile is the value below which a certain fraction of samples in the distribution fall. The quantile loss can be defined as:
$Loss = \sum_i \sum_{j=2}^{5} q \times \max(0, d^i_{t+j} - \hat{d}^i_{t+j}) + (1 - q) \times \max(0, \hat{d}^i_{t+j} - d^i_{t+j})$
where q is the required quantile (between 0 and 1). Here, q = 0.5.

4.1 Implementation Details

We use DGL and PyTorch for implementing all our architectures. We train our networks in a minibatch fashion. In each minibatch, we take 1048 (= *batch size*) time series, identify the unique sellers and products in them (seed nodes), perform sampling in all the graphs to identify the neighbors for these seed nodes, compute embeddings for the seed nodes from each of the sampled graphs, feed it into a sequential network for generating predictions, and finally back-propagate the loss to train the network. A 2-layer GNN network (16 hidden units) followed by a 2-layer LSTM network (16 hidden units) is used in all the variants. For decoder, an MLP with 2-layers, having 512 hidden units each is used. All the variants converge after 8-10 epochs of training with *learning rate = 0.003*. We finetune these hyparameters by optimizing the performance on the validation set. For TCNs, we use an 8-layer network with kernel size as 7. For training multi-graph networks, we make use of a multi-GPU approach using Torch's DistributedDataParallel library. The main computational bottleneck while training such a huge network is the sampling mechanism. And, the sampling has to be done for all the graphs at each minibatch, based on the sellers and products in that minibatch. Presently, sampling in DGL is implemented in CPU and consumes a large amount of time. For example, in an p3.8x AWS instance, LSTM takes 30 minutes/epoch for training whereas Homo-GCN and Homo-GS take 1.5 hours/epoch and 3 hours/epoch respectively for training. The feature dimension of our input features is as follows – $x_t(s^i) = 26$, $x_t(p^i) = 16$, $x_t(p^i, s^i) = 83$, and $x_t(st) = 8$.

4.2 Comparison with baseline

We perform the exhaustive evaluation of various GNN architectures proposed in the Section 3.3. The methods under contention are –

1. **LSTM:** This is our baseline method. This architecture resembles the one proposed by Wen et al. [25], where a vanilla LSTM is used to encode all history into hidden state and an MLP is used as a decoder for predicting the future demand. At each time-step, we concatenate the features x_t^i with the demand value d_t^i and provide it as an input to the LSTM.
2. **Homo-GCN:** In this architecture, GCN is applied to homogeneous graph without considering the edge features in the graph.
3. **Homo-GS:** In this architecture, GS is applied to homogeneous graph without considering the edge features in the graph.
4. **Homo-GAT:** This is an extension of Homo-GCN or Homo-GS to GAT networks. For this method also, we convert the graph into a homogeneous graph, and ignore the edge related features in the graph.
5. **Homo-GS-Demand**: This architecture is an extension of Homo-GS. Along with the input node features, we also add the demand relation features in the homogeneous graph.
6. **Homo-GAT-Demand**: This architecture is an extension of Homo-GAT. In the homogeneous graph, demand relation features are added.
7. **Hetero-GS-Demand-Substitute**: This architecture is proposed in Section 3.3. It includes both demand and substitute relations in the graph along with their features.
8. **Hetero-GAT-Demand-Substitute**: This architecture is also proposed in Section 3.3. It includes both demand and substitute relations in the graph along with their features.

Table 2 summarizes the performance of above models relative to an LSTM model based baseline. As it can be seen, the best performing GNN model results in $\sim$16% improvement in MAPE as compared to LSTM model. Hence, there is a merit in modelling correlations in the times series using GNNs. The GNN module allows us to capture the homogeneous and heterogeneous correlations residing in the time series data of online marketplaces in a principled manner. In all the experiments, on expected lines, we see that GAT performs significantly better than the GCN and GS variants, and GS performs better than the GCN. Also, adding the features for demand relation in the graph improves the performance by $\sim$3.7% and $\sim$4.7% for Homo-GS and Homo-GAT respectively. Finally, moving to a heterogeneous setup with multiple kinds of relation (demand and substitute), further improves the performance by $\sim$1.5% and yields the best model.

As discussed in Section 3.4, we have also experimented with TCNs as the sequential model. We plugged the GNN modules into TCN network and trained the complete network in an end-to-end fashion. With Homo-GS, we observed that the MAPE improves by 5.43% and with Homo-GAT, the MAPE improves by 6.65% relative to an TCN baseline.

Table 2. Improvement in MAPE metric for different variants of GNN architectures relative to an LSTM model based baseline. We also report improvement in MAPE for two special cases – (i) when a product is being sold by more than one seller (multi-seller products), (ii) cold/warm start offers, and show that the performance improvement is even more significant in these cases.

Method	MAPE (all offers)	MAPE (multi-seller products)	MAPE (cold start offers)
Homo-GCN	4.08%	4.91%	10.20%
Homo-GS	9.33%	12.60%	17.41%
Homo-GAT	10.13%	14.78%	17.54%
Homo-GS-Demand	13.05%	20.51%	20.72%
Homo-GAT-Demand	14.82%	25.48%	21.51%
Hetero-GS-Demand-Substitute	14.34%	27.43%	23.56%
Hetero-GAT-Demand-Substitute	16.30%	29.43%	24.43%

4.3 Demand Forecasting for multi-seller products & cold start offers

The idea behind using GNNs with sequential model is to model the homogeneous and heterogeneous correlations in the multivariate time series data. Intuitively, the correlation is high when a single product is being offered by multiple sellers on the platform due to competition, being out of stock, price increase/decrease by other sellers, etc. In order to validate this, we filter out the time series from the test set that contain products being offered by more than one seller. There are 505,485 such time series in the test data. We evaluate the MAPE on this set explicitly and show that the best GNN model performs 29.34% better than the LSTM model (refer to Table 2). This improvement is much higher than the improvement on the full test set (16.30%), thereby, highlighting the fact that the selected subset of the data has more correlations and the proposed framework is able to capture them across different time series using GNNs.

Another scenario that can greatly benefit with the modelling of correlations is the problem of forecasting demand for cold-start or warm-start offers. The cold/warm-start offers do not have enough history to predict their future demand accurately. This happens quite often in online marketplaces when a seller launches a new product or a new seller starts offering any of the existing products on the platform. In such cases, the proposed framework can be leveraged to derive their demand from other correlated time series in the data. To empirically validate this, we filter out the time series from the test data that contain offers which have less than 3 months (12 weeks) of history. On this set, the best performing GNN model improves MAPE by ∼24.43% which is again much higher than the improvement on the overall test set. Figure 2 shows the actual forecast values using different methods for a cold-start offer and a multi-seller product.

5 Conclusion

In this work, we propose a generic framework for handling competing/correlated time series in demand forecasting. We evaluated the framework for the task of

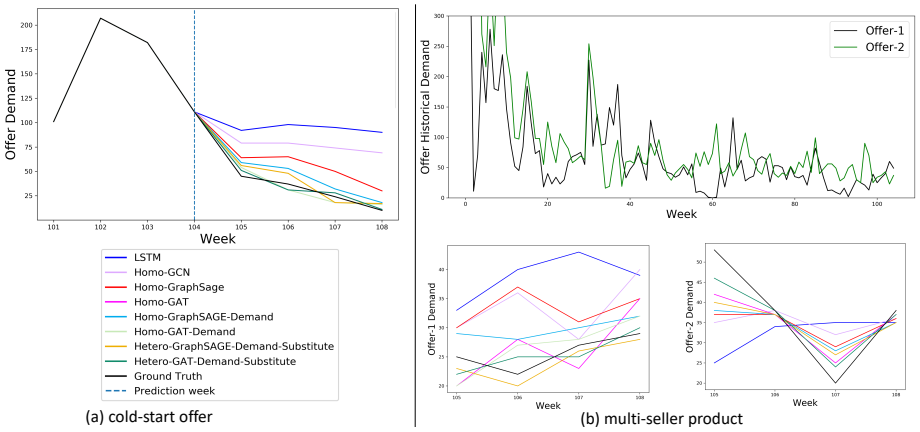

Fig. 2. Future forecast for (a) a cold-start offer and (b) a multi-seller product (from 2 sellers) from our dataset using different techniques. Note that the cold start offer has historical demand for 3 weeks only (101, 102 and 103) whereas for multi-seller offer, there are 2 sellers offering the same product, and history is available for both the offers for 104 weeks. The task is to predict the demand for future 4 weeks. As the LSTM architecture looks only at the offer sales and features to forecast the demand, it performs much worse than the GNN based techniques. GNN techniques leverage other correlated time series also for predicting the future demand.

demand forecasting in online marketplaces. We capture the correlation between sellers and products using different variants of GNN, and show that it can be plugged into any sequential model for demand prediction. The proposed technique improves the MAPE on a real-world marketplace data by ∼16% by capturing the homogeneous and heterogeneous correlations across multivariate time series using GNNs. We also extended various standard GNN architectures to utilise edge features as well for updating the node embeddings.

References

1. Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling (2018)
2. Bandara, K., et. al.: Sales demand forecast in e-commerce using a long short-term memory neural network methodology. In: Neural Information Processing (2019)
3. Benidis, K., Rangapuram, S.S., Flunkert, V., et. al.: Neural forecasting: Introduction and literature overview (2020)
4. Box, G.E.P., Cox, D.R.: An analysis of transformations. Journal of the Royal Statistical Society: Series B (Methodological) (1964)
5. Box, G.E.P., Jenkins, G.M., Reinsel, G.C., Ljung, G.M.: Time series analysis: Forecasting and control. John Wiley and Sons (2015)
6. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: going beyond euclidean data. CoRR (2016)

7. Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., et. al.: Spectral temporal graph neural network for multivariate time-series forecasting. In: NeurIPS (2020)
8. Chen, Y., Kang, Y., Chen, Y., Wang, Z.: Probabilistic forecasting with temporal convolutional neural network (2020)
9. Durbin, J., Koopman, S.J.: Time series analysis by state space methods: Second edition. Oxford University Press (2012)
10. Ghorbani, M., Baghshah, M.S., Rabiee, H.R.: Multi-layered graph embedding with graph convolutional networks. CoRR (2018)
11. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. CoRR (2017)
12. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. CoRR (2017)
13. Hyndman, R., Koehler, A.B., Ord, J.K., Snyder, R.D.: Forecasting with exponential smoothing: The state space approach. Springer Series in Statistics (2008)
14. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. CoRR (2016)
15. Lai, G., Chang, W., Yang, Y., Liu, H.: Modeling long- and short-term temporal patterns with deep neural networks. CoRR (2017)
16. Li, Y., Yu, R., Shahabi, C., Liu, Y.: Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In: ICLR (2018)
17. McAuley, J., Pandey, R., Leskovec, J.: Inferring networks of substitutable and complementary products. KDD (2015)
18. Monti, F., Bronstein, M.M., Bresson, X.: Geometric matrix completion with recurrent multi-graph neural networks. CoRR (2017)
19. Mukherjee, S., Shankar, D., Ghosh, A., et. al.: ARMDN: associative and recurrent mixture density networks for eretail demand forecasting. CoRR (2018)
20. Rangapuram, S.S., Seeger, M.W., Gasthaus, J., Stella, L., Wang, Y., Januschowski, T.: Deep state space models for time series forecasting. In: NeurIPS (2018)
21. Salinas, D., Flunkert, V., Gasthaus, J.: Deepar: Probabilistic forecasting with autoregressive recurrent networks (2019)
22. Sen, R., Yu, H.F., Dhillon, I.S.: Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. In: NeurIPS (2019)
23. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. ICLR (2018)
24. Wang, Z., Jiang, Z., Ren, Z., et. al.: A path-constrained framework for discriminating substitutable and complementary products in e-commerce. WSDM (2018)
25. Wen, R., Torkkola, K., Narayanaswamy, B., Madeka, D.: A multi-horizon quantile recurrent forecaster (2018)
26. Wu, Z., Pan, S., Long, G., Jiang, J., Zhang, C.: Graph wavenet for deep spatial-temporal graph modeling. In: IJCAI-19 (2019)
27. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. CoRR (2018)
28. You, J., Ying, R., Ren, X., Hamilton, W.L., Leskovec, J.: Graphrnn: A deep generative model for graphs. CoRR (2018)
29. Yu, B., Yin, H., Zhu, Z.: Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In: IJCAI (2018)
30. Zitnik, M., Agrawal, M., Leskovec, J.: Modeling polypharmacy side effects with graph convolutional networks. CoRR (2018)