Abstract

Forecasts at different time granularities are required in practice for addressing various business problems starting from short-term operational to medium-term tactical and to long-term strategic planning. These forecasting problems are usually treated independently by learning different ML models which results in forecasts that are not consistent with the temporal aggregation structure, leading to inefficient decision making. While prior work addressed this problem, this typically uses a post-hoc reconciliation strategy, which leads to sub-optimal results and cannot produce probabilistic forecasts. In this paper, we present a global model that produces coherent, probabilistic forecasts for different time granularities by learning joint embeddings for the different aggregation levels with graph neural networks and temporal reconciliation. Temporal reconciliation not only enables consistent decisions for business problems across different planning horizons but also improves the quality of forecasts at finer time granularities. A thorough empirical evaluation illustrates the benefits of the proposed method.

1 INTRODUCTION

Time series prediction, or forecasting, has many important applications ranging from retail demand forecasting (Mukherjee et al., 2018; Croston, 1972), to labor planning (Bohlike-Schneider et al., 2020), traffic flow (Laptev et al., 2017), electrical load forecasting (Hong et al., 2019) and cloud capacity planning (Petropoulos et al., 2020). Downstream decision makers need forecasts on different temporal aggregations (or time scales) depending on the nature of the decision. While strategic decisions require forecasts for the next quarters or months, short-term operational decisions require forecasts on hourly granularity or even lower. Tactical forecasts are somewhere in between along this spectrum. So, a single forecasting problem like demand forecasting in retail often has operational, tactical and strategical facets (Januschowski and Kolassa, 2019). In practice, these different aspects of the same forecasting problem are treated independently, often produced by different organizations and by teams with different skill sets. This leads to inconsistent results where the forecasts generated at finer time granularities do not add up to the aggregated forecasts, inevitably leading to inefficient decision making. One example is electrical demand forecasting where short-term forecasts ensure stable operation of the electric grid while long-term forecasts are important to plan and implement sufficient electricity supply. Both problems interact with each other in a non-trivial way. Having separate, independent forecasts for both problems introduces potential conflicts and inefficiencies.

The field of hierarchical forecasting explores how to deal with these potential inefficiencies gracefully (Ben Taieb et al., 2017; Wickramasuriya et al., 2015; Taieb et al., 2020; Athanasopoulos et al., 2009; Rangapuram et al., 2021). Most work in this area has considered handling cross-sectional hierarchies as induced by meta data (e.g., product hierarchies). Considerably less work is devoted to temporal hierarchies (Athanasopoulos et al., 2017; Theodosiou and Kourentzes, 2021). Building on recent work (Rangapuram et al., 2021), we present a novel method for probabilistic forecasting with temporal hierarchies. Our method obtains forecasts for a given univariate time series at different aggregation levels, enables information sharing between the aggregation levels, and incorporates the temporal aggregation structure into the overall model. We achieve this by
sharing information between aggregation levels via a graph neural network that leverages the temporal hierarchy and we ensure coherence by reconciling the samples with orthogonal projection. As a consequence of the end-to-end nature of our framework, the probabilistic forecasts generated at different time granularities are guaranteed to be consistent. We show in empirical evaluations that the proposed model reduces the forecasting error of noisy time series sampled at finer granularities. In summary, our contributions are as follows:

- We propose a unified global model that takes as input a univariate time series at the given base frequency, learns joint embeddings, and generates coherent, probabilistic forecasts for any required aggregation frequency.

- We provide empirical evidence that forecasts of (noisy) time series sampled at finer time granularities can be improved by simultaneously generating consistent forecasts for aggregated time series; this enables application of forecasting methods for finer frequencies, e.g., 1 minute data, which is usually considered too noisy for forecasting and related applications.

The rest of the paper is organized as follows. We first provide the necessary background on temporal hierarchies (Section 2), followed by reviewing related work in the general area of hierarchical forecasting as well as specific methods designed for temporal hierarchies (Section 3). We present our method in Section 4 and describe how our model uses a graph neural network to share information between aggregation levels. Our empirical evaluation is presented in Section 5. Section 6 concludes the paper. Societal impact is discussed in the Appendix.

2 BACKGROUND

2.1 Temporal Hierarchies

Consider a time series sampled at a given frequency, referred to as base frequency. We can aggregate non-overlapping, equally spaced values to arrive at a coarser frequency. For instance, if the base frequency is 15 minutes, we can aggregate the time series to half-hourly and hourly frequencies. The relationship between such temporally aggregated time series and the original time series can be described in terms of a temporal hierarchy as shown in Figure 1, similarly to a standard cross-sectional hierarchy notation (Hyndman et al. 2011).

In the classical cross-sectional hierarchy, each node in the tree corresponds to an individual item of the hierarchical time series dataset. However, in temporal hierarchy, each level of the tree shown in Figure 1 actually corresponds to a single time series sampled at a different aggregated frequency and nodes represent time points. The root level corresponds to the highest or coarsest aggregation level. The number of leaves of the temporal hierarchy is determined by the coarsest aggregation frequency. For example, in the case of 15-minute data, if the highest aggregation frequency is hourly, then the number of leaves is 4.

**Notation.** We first fix the notation to make the exposition clear. Given a time series at a base sampling frequency, a \( k \)-aggregated time series is constructed by summing up \( k \) successive non-overlapping values of the given time series. We assume that \( p \) such aggregated time series are constructed using aggregate multiples, given in the descending order, \( \{k_p, \ldots, k_2, k_1\} \), with \( k_1 = 1 \), to form a temporal hierarchy. Let \( m := k_p \) denote the highest aggregated multiple corresponding to the root of the hierarchy; note that \( m \) is the number of leaves of the hierarchy. In case of the temporal hierarchy given in Figure 1, \( p = 3 \) and the aggregate multiples are \( \{4, 2, 1\} \). We denote the time series at an aggregated level corresponding to the \( k^{th} \) aggregate multiple as \( y_{[k]} \), where \( k \in \{k_p, \ldots, k_2, k_1\} \); note the non-consecutive indexing for the level.

Since each level of the hierarchy corresponds to a different time granularity, the time index \( t \) varies with each aggregation level. Hence, following the notation by Athanasopoulos et al. (2017), we define \( t \) as the observation index of the most aggregated series (i.e., root level), in order to use a common index for all levels. More precisely, observations at the \( k^{th} \) aggregation level are denoted by

\[
y_{[k]}_{M_k(t-1)+\delta_k}, \quad \delta_k \in \{1, 2, \ldots, M_k\},
\]

where \( M_k := \frac{m}{k} \). Here \( t-1 \) denotes a decrement of one step in the time granularity corresponding to the root level and \( \delta_k \) denotes increments at the \( k^{th} \) aggregation level. This common indexing is illustrated for the 15-minute temporal hierarchy in Figure 2.

Note that the rightmost leaf node and the root node correspond to the same time point; however their time indices are incremented differently. Whenever we speak of temporal hierarchy, we actually refer to sequences of time points corresponding to different aggregation frequencies. For example, in case of Figure 2, the temporal hierarchy refers to one time point at hourly level, namely, at-the-hour, two time points at 30-minute level, namely, half past and the...
An equivalent representation of the aggregation constraint where \((1)\) \cite{Rangapuram2021} is given by

\[
\text{Temporal hierarchical forecasting refers to the problem of forecasts to be coherent with the hierarchy induced at time } t.
\]

For ease of notation, we denote by \(y_t\) the vector of all observations in the hierarchy at time \(t\) ordered according to the pre-order of the hierarchical tree: \(y_t := [y_{k(t-1)+1}]_k \in \{k_p, \ldots, 2, 1\}\) and \(\delta_k \in \{1, 2, \ldots, M_k\}\). For 15-min base frequency with a 2-level hierarchy as depicted in Figure 2, \(y_t\) is a vector of seven values.

Following \cite{Athanasopoulos2017}, we call the hierarchy given in Figure 2 a 2-level hierarchy; i.e., the number of levels of the hierarchy, denoted by \(L\), is given by the height of the tree.

Similarly to the cross-sectional hierarchical time series, it is convenient to represent the temporal hierarchy via an aggregation or summation matrix. Recall that \(y_t\) denotes the vector of all observations in the hierarchy at time \(t\). Let \(b_t := [y_{m(t-1)+1}]_m\), \(\delta_t \in \{1, 2, \ldots, m\}\), denote the vector of all observations at the bottom level of the hierarchy at time \(t\). Then we have

\[
y_t = Sb_t,
\]

where \(S\) is the standard summation matrix used in the cross-sectional hierarchy \cite{Hyndman2011}.

In the following we denote by \(r\) the total number of aggregated nodes in the temporal hierarchy, \(m\) the number of leaf nodes and \(n\) the total number of nodes. We have \(n = r + m\).

An equivalent representation of the aggregation constraint \((1)\) \cite{Rangapuram2021} is given by

\[
C y_t = 0,
\]

where \(C := [I_r | - S_{\text{sum}}] \in \{0, 1\}^{r \times n}\), \(0\) is an \(r\)-vector of zeros, and \(I_r\) is the \(r \times r\) identity, \(S_{\text{sum}}\) is summation matrix.

### 2.2 Temporal Hierarchical Forecasting

Temporal hierarchical forecasting refers to the problem of producing forecasts simultaneously for all time granularities exploiting the hierarchical structure. An important requirement is for forecasts to be coherent with the hierarchy apart from being accurate. We follow \cite{Rangapuram2021} in defining the coherence of probabilistic forecasts obtained for different time granularities.

**Definition 2.1.** \cite{Rangapuram2021}. Let \(S \subseteq \mathbb{R}^n\) be a linear subspace defined as

\[
S := \{ y' y \in \text{null}(C) \}
\]

where \(C := [I_r | - S_{\text{sum}}] \in \{0, 1\}^{r \times n}\), and \(I_r\) is the \(r \times r\) identity. A point forecast \(\hat{y}_{T+h}\) is said to be coherent w.r.t. the corresponding hierarchy, iff \(\hat{y}_{T+h} \in S\). Similarly, a probabilistic forecast represented as samples \(\{\hat{y}_{T+h}\}\) is coherent iff each of its samples is.

**Thief.** We now describe Thief \cite{Athanasopoulos2017}, recent work that formalized the concept of temporal hierarchies for time series forecasting. Thief, a shorthand for Temporal hierarchical forecasting, is an approach to forecasting with temporal hierarchies that produces temporally reconciled forecasts. It follows a two-step procedure: first, it generates forecasts \(\hat{y}_{T+h}\) independently for all the required time granularities and second, it reconciles them to generate coherent forecasts \(\hat{y}_{T+h}\). Motivated by several reconciliation techniques from cross-sectional hierarchical forecasting, Thief considers several reconciliation approaches, which can be represented in the general form,

\[
\hat{y}_{T+h} = SP\hat{y}_{T+h},
\]

where \(S\) is the aggregation matrix and \(P \in \mathbb{R}^{n \times n}\) is a matrix that depends on the choice of the reconciliation technique. For instance, the Bottom-Up reconciliation returns forecasts of aggregated time series by aggregating the forecasts of the bottom level, and hence \(P = [0_{m \times r} | 1_{m \times m}]\). There are several choices possible for \(P\) resulting in different variants of the method \cite{Wickramasuriya2019}.

### 3 RELATED WORK

The field of hierarchical time series forecasting \cite{Hyndman2018} takes advantage of information present in time series that allows to define an aggregation hierarchy. Traditionally, hierarchical time series forecasting methods consists of two steps: first forecasts are generated at each aggregation level, and second one reconciles the computed forecasts so that forecasts from different aggregates are consistent, e.g. \cite{Hyndman2011, Wickramasuriya2015, BenTaieb2019}. Notable departures from this include \cite{Han2021} who rely on regularization or \cite{Abolghasemi2019} who propose to learn disaggregation proportions for parts of the hierarchy in a middle-out approach. The vast majority of methods for hierarchical time series forecasting is limited to point forecasts. Probabilistic forecasts are studied by \cite{BenTaieb2019} with a two-step framework. In contrast, we offer an end-to-end approach and do not make a Gaussianity
While most work in hierarchical forecasting considers cross-sectional aggregation hierarchies defined by meta-data such as product hierarchies or geographies (Hyndman and Athanasopoulos, 2018), temporal hierarchical forecasting has also been studied (Athanasopoulos et al., 2017). This extends the notion of hierarchical time series to the case where the aggregation is done on the temporal component, and hence one obtains forecasts of different temporal aggregations that are coherent. Ben Taieb (2017) introduces smooth and sparse adjustments to satisfy the aggregation constraints by solving a generalized lasso problem. Temporal aggregation of time series in is also studied in (Amemiya and Wu, 1972; Tiao, 1972; Lütkepohl, 1987). A number of surveys are available on the topic Silvestrini and Veredas (2008); Lütkepohl (2011); Clemen (1989). Another line of work takes the forecast from multiple temporal aggregation levels to produce an optimal final forecast using the insight (Hibon and Evgeniou, 2005) that the average forecast from different models provides a forecast quality similar to the best individual forecast, and hence reduces model uncertainty (Kourentzes et al., 2019). For instance, in (Kourentzes et al., 2014) MAPA is introduced and rather than combining forecasts from different temporal aggregation levels, the forecasted model parameters of each level are aggregated to generate a single forecast.

Further approaches in temporal hierarchical forecasting include (Nystrup et al., 2020) which proposes different estimates for autocorrelation, whereas Nystrup et al. (2020) provide an approach based on dimensionality reduction. Recently, Theodosiou and Kourentzes (2021) introduce DeepTHieF as an end-to-end extension of Thief where both forecasts and reconciliation are executed in a single deep learning model. However, contrary to Thief, DeepTHieF cannot guarantee temporally coherent forecasts. In contrast to Thief and DeepTHieF, which generate point-forecasts, our proposed approach generates probabilistic forecasts and further guarantees that the probabilistic forecasts are coherent for the temporal hierarchy. Chung et al. (2017) proposed a multiscale RNN approach for learning the hierarchy and the temporal relation in sequence data. However, this work aims to learn the hierarchy in general sequence tasks, while our method specifically focuses on producing coherent probabilistic forecasts with a fixed hierarchy given by the temporal aggregation structure.

4 END-TO-END FORECASTER FOR TEMPORAL HIERARCHIES

Rangapuram et al. (2021) provide an end-to-end model for forecasting time series with hierarchical structure given by item meta-data, which we refer to here as cross-sectional hierarchies. The main idea behind this approach is to combine the forecasting step and the reconciliation step in a single trainable model. They use an autoregressive neural network model (RNN) for the forecasting step and orthogonal projection for the reconciliation step. Since the reconciliation step is a part of the end-to-end training procedure, the underlying forecaster directly minimizes the loss on the final coherent forecasts. Consequently, this approach yields better results than methods that perform forecasting and reconciliation steps independently. However, this method is not directly applicable to temporal hierarchies for the following reasons:

- All time series in the item hierarchy have same time granularity and hence are processed by the same multivariate model (RNN). For temporal hierarchies, each time series is of different length, possesses different time dynamics and needs different seasonal features and therefore a different, independent RNN model for training.
- Because the embeddings are learned by independent RNNs, there is no information transfer among time series at different granularities. In the case of cross-sectional hierarchical model Rangapuram et al. (2021), the RNN embeddings that produce the final forecasts are learnt jointly.
- In the autoregressive setting, the item hierarchy model introduces a test-train time discrepancy when applied to temporal hierarchies; see Section 4.5.

Our model addresses these issues and brings further enhancement to the end-to-end approach by introducing a graph neural network layer for better information sharing across different granularities. This could also be directly applied to item hierarchies.

Figure 3 presents the overall architecture of the proposed end-to-end model for temporal hierarchies. For illustration, we use the temporal hierarchy given in Figure 6 for 15-min base frequency, as the running example. Our model consists of three main blocks:

- A univariate RNN model for each level of the temporal hierarchy: each RNN uses (batches of) time series at a given time granularity as input and outputs the embeddings needed for one-step ahead forecast at every time step.
- Graph Neural Network layer: this layer applies message passing on the embeddings obtained by individual
Temporal Aggregation

We use a standard RNN-based sequence-to-sequence model, with the corresponding aggregate multiples given by \( \Theta \) for generating forecasts for time points at that level. Given the embeddings \( \mathbf{H}_t \) for all time points in the hierarchy, joint learning is then facilitated by the graph convolutional network. These joint embeddings are then transformed to the parameters of \( \Theta_t \) of the predictive distribution. To generate coherent forecasts, we draw samples \( \hat{y}_t \) from this predictive distribution and reconcile them using orthogonal projection. The model is trained by minimizing the CRPS loss on coherent samples \( \hat{y}_t \). During prediction, we autoregressively unroll all RNNs to produce forecasts.

\[
\mathbf{H}_t := \Psi \left( \mathbf{X}_t, \mathbf{y}_{t-1}, \mathbf{H}_{t-1}; \{ \Phi_k \}_{k=1}^p \right)
\]

\[
\mathbf{H}_t^{(m+1)} = \sigma (A \mathbf{H}_t^{(m)} W)
\]

\[
\Theta_t = \text{NN}(\mathbf{H}_t^{(m)} W_0)
\]

\[
\hat{y}_t \sim \mathcal{P}(\cdot; \Theta_t)
\]

RNN models in order to promote cross-learning across different granularities. The output of this layer is then mapped to the parameters of the forecast distribution with a dense layer.

- Reconciliation: given the parameters of the forecast distribution, we sample from these (unreconciled) distributions and reconcile the samples via orthogonal projection (as in (Rangapuram et al., 2021)).

The output of the model is a sample-based forecast where each sample is coherent with respect to the temporal hierarchy. The overall model is trained by minimizing CRPS loss (Gasthaus et al., 2019) on these coherent samples.

4.1 Univariate Forecaster

We use a standard RNN-based sequence-to-sequence model, DeepAR (Salinas et al., 2020), as the univariate forecaster. Our choice is based on ease of implementation, but readily extends to other architectures including the sequence-to-sequence model family with the only assumption that the model transforms the input time series to an embedding. DeepAR is a nonlinear generalization of the classical autoregressive model and uses a recurrent neural network (RNN) to generate probabilistic predictions for the future values of the time series given its past values, known as lags.

Let us assume we are given \( p \) aggregated time series with the corresponding aggregate multiples given by \( \{k_p, \ldots, k_2, k_1\} \), with \( k_1 = 1 \) and \( k_p = m \), the number of leaves of the hierarchy. Then, we have \( p \) RNNs each processing batches of time series at one of the time granularities of the hierarchy. We unroll all \( p \) RNNs simultaneously by incrementing \( t \) by one step, where \( t \) refers to the time index of the most aggregated time series (see Section 2 for notation). This means the RNN for time series at bottom level is unrolled for \( m \) steps where as the RNN for the most aggregated time series is unrolled for only one step. More precisely, the RNN corresponding to the \( k^\text{th} \) aggregation level is unrolled for \( m/k \) steps. This is illustrated in Figure 3 for 15-min temporal hierarchy.

Each such unrolling at time \( t \) produces embeddings for all nodes (i.e., time points) in the corresponding temporal hierarchy. Recall that we denote by \( y_t \in \mathbb{R}^n \), the vector of all observations in the hierarchy at time \( t \): \( y_t := [y_{M_k(t-1)+\delta_k}, k \in \{k_p, \ldots, 2, 1\} \) and \( \delta_k \in \{1, 2, \ldots, M_k\} \). Using this notation, RNN embeddings for all nodes in the hierarchy can be expressed as

\[
\mathbf{H}_t := \Psi \left( \mathbf{X}_t, \mathbf{y}_{t-1}, \mathbf{H}_{t-1}; \{ \Phi_k \}_{k=1}^p \right)
\]

\[
\mathbf{H}_t \in \mathbb{R}^{n \times d} \text{ where } n \text{ is the total number of nodes in the hierarchy and } d \text{ is the embedding dimension. Here } \mathbf{X}_t \in \mathbb{R}^{n \times D} \text{ is the feature matrix specifying features for all nodes in the hierarchy, } y_{t-1} \in \mathbb{R}^n \text{ is the lag input, a vector of all observations in the temporal hierarchy at time } t-1 \text{ and } \Phi_k \text{ are the parameters of RNN at level } k.

A crucial difference to (Rangapuram et al., 2021) is that when unrolling RNNs at time \( t \), the lag input always comes...
from the observations in the temporal hierarchy corresponding to previous time step \( t - 1 \). None of the observations in the hierarchy at time \( t \) is used as a lag input when unrolling the RNNs at time step \( t \), even though some RNNs (e.g., bottom level) are unrolled for more than one step.

### 4.2 GNN Layer

The embeddings \( H_t \) produced by the RNNs are learned independently without considering the hierarchical structure of the data. Graph convolutional networks (GCN, for short) are a natural way to incorporate such hierarchical structure into the learning process. Given the embeddings \( H_t \in \mathbb{R}^{n \times d} \) at all nodes of the temporal hierarchy and a matrix \( A \in \mathbb{R}^{n \times n} \) representing the underlying tree structure, we learn the joint embeddings via a non-linear transformation:

\[
H_t^{\text{new}} = f(H_t, A).
\]

In our case \( f \) is a single neural network layer with learnable weights \( W \in \mathbb{R}^{d \times d} \) and a ReLU activation function and is given by

\[
f(H_t, A) = \sigma(AH_tW).
\]

Note that before applying the non-linear activation, the linearly transformed embeddings \( AH_tW \) are pre-multiplied by the matrix \( A \) representing the tree structure. This pre-multiplication, depending on the definition of \( A \), corresponds to exchanging embeddings at each node with the immediate neighbours. Since one would like to have the embeddings at any node to be propagated to every other node in the graph, this function is often applied repeatedly for several times. In our case, because of the tree structure, we only need to apply \( f \) for \( L \) times to achieve this effect, where \( L \) is the number of levels of the hierarchy.

Thus, starting with independent embeddings \( H_t^{(0)} := H_t \), we have

\[
H_t^{(l+1)} = \sigma(AH_t^{(l)}W), \quad l = 0, 1, \ldots, L - 1. \tag{5}
\]

**Matrix \( A \).** We define the matrix \( A \) in such a way that the embeddings at any node are propagated proportionately to every other node in the tree. This can be achieved efficiently by decomposing \( A \) into three components:

- \( A_{\text{acc.}} \): this is the standard adjacency matrix of the hierarchy tree where the undirected edges are replaced by directed edges pointing downwards; pre-multiplying any embedding \( H \in \mathbb{R}^{n \times d} \) by \( A_{\text{acc.}} \) amounts to replacing the embedding at every node by the average of its own embedding and the embeddings of its children. This means, the embeddings of leaves in this case are replaced by zeros.

- \( A_{\text{ret.}} \): this is the identity matrix that retains the embeddings at every node.

- \( A_{\text{dist.}} \): this is the adjacency matrix of the directed tree where the edges now point upwards; pre-multiplying any embedding \( H \in \mathbb{R}^{n \times d} \) by \( A_{\text{dist.}} \) amounts to replacing the embedding at every node by (the correct) fraction of the embedding of its parent. The fraction depends on the number of children of the parent and is the reason for normalizing this adjacency matrix. Again, this operation replaces the embeddings of the root node by zeros.

With this, we define the structure of matrix \( A \) as

\[
A := (A_{\text{acc.}} + A_{\text{dist.}} + A_{\text{ret.}})/3.
\]

Pre-multiplying any embedding by \( A \) corresponds to updating the embedding at every node by the average of its own current embedding, sum of the embeddings of its children and a fraction of the embedding from its parent.

### 4.3 Sampling & Reconciliation

Given the joint embeddings \( H_t^{(L)} \in \mathbb{R}^{n \times d} \) according to Eq. (5), we would like to produce forecasts for all nodes in the hierarchy at time \( t \). Since we are interested in producing probabilistic forecasts, we transform the joint embeddings into parameters of the predictive distribution via a dense layer,

\[
\Theta_t = \text{NN}(H_t^{(L)}; W_\theta).
\]

For simplicity, we assume that the predictive distribution is Gaussian. In order to generate coherent forecasts, we follow the approach of Rangapuram et al. (2021) where we first produce a set of \( N \) Monte Carlo samples from the predictive distribution and reconcile each sample. As mentioned in Section 2, the space of all coherent target values is the null space of the matrix \( C \) (see Eq. (2)); hence one way to enforce coherence is by projecting samples onto the null space of \( C \). The projection step is essentially a matrix-vector multiplication and is differentiable w.r.t. the model parameters. The sampling step also does not pose any problem as far as the differentiability w.r.t. the model parameters is concerned, because of the re-parameterization trick, which is available for various parametric distributions (Figurnov et al. 2018; Jankowiak and Obermeyer 2018).

#### 4.3.1 Enforcing Non-negativity

It turns out that in some cases one needs to guarantee that the forecasts generated are non-negative. However, reconciling samples via projection, as mentioned above, might result in samples with negative values even if the unreconciled samples are non-negative. One way to address this issue is to treat the problem of enforcing both coherence and non-negativity as a single problem and then project the samples on to the intersection of the null space of the matrix \( C \) and the non-negative orthant. This way one guarantees that the projected samples are both coherent and non-negative.
Unlike the projection onto the null space, there is no analytical solution for the problem of projection onto the intersection of convex sets. However, one can use Dykstra’s method (Boyle and Dykstra, 1986), a variant of alternating projection method, that alternately projects on to each of the convex sets until convergence. In our case, this results in a simple iterative procedure where each step involves projection on to the null space of $C$ and the non-negative orthant, both of which have analytical solutions.

4.4 Training

To summarize, each unrolling of the RNNs first produces independent embeddings for all the nodes in the hierarchy at time $t$, which are combined via a graph neural network to obtain a joint embedding. This joint embedding is then transformed to coherent sample-based forecasts for all nodes in the hierarchy at time $t$. To train our model, we unroll the RNNs for all time steps where the observations are available. We define the loss directly on the coherent samples produced by our model, for which the continuous ranked probability score (CRPS) Matheson and Winkler (1976) is a natural candidate. It can be defined as the sum of the quantile losses evaluated at all possible quantile levels (Laio and Tamea, 2007). Let $\{\hat{y}^k_j\}$ denote $N$ coherent samples obtained for time point corresponding to node $j$ at the $k^{th}$ aggregation level and let $y^k_j$ be the corresponding true target. Since there is a finite number of unique quantiles in the forecast based on the empirical samples, the CRPS loss is given by

$$\text{CRPS} \left( y^k_j, \{\hat{y}^k_j\} \right) = \sum_{s_i \in \{\hat{y}^k_j\}} \Lambda_{\alpha_i}(y^k_j, s_i),$$

where $\alpha_i$ is the quantile level of the sample $s_i$ and $\Lambda_{\alpha_i}(q, z)$ is the quantile loss given by $\Lambda_{\alpha_i}(q, z) = (\alpha - I[z < q])(z - q)$.

4.5 Prediction

Since DeepAR is an auto-regressive model, it needs its own predictions as lag inputs if the forecast is required for more than one time step. In case of end-to-end approach, reconciled samples are only available after having predicted for $m$ consecutive time steps, where $m$ is the number of leaves of the temporal hierarchy. This impedes a direct application of the model proposed in Rangapuram et al. (2021) for cross-sectional hierarchies to the temporal setting, because during inference incoherent predictions would be used as lags whereas the model was trained using coherent lags (original target). This leads to a detrimental train-test set discrepancy. However, as mentioned in Section 4.1, by design, our method does not use any of the observations in the hierarchy at time $t$ as a lag input when unrolling the RNNs at time step $t$. This enables us to do a full forward pass of the model, including applying the GCN layer and performing reconciliation, before we feed the output obtained at time $t$ for the next time step.

5 EXPERIMENTS

We evaluate the proposed method empirically on public time series datasets where the time granularities range from 1-min to 1-day; see Table 3 in the supplement for a dataset summary. The Taxi-1min data for January 2021 contains pick-up and drop-off times of 1.3 million individual

1The raw taxi data is available at: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
Table 1: The CRPS loss (the lower, the better) for all datasets and models. For LogSparse, TimeGrad, DeepAR and COPDeepAR we average over 5 runs and report the mean and standard deviation. COPDeepAR has the lowest loss for most datasets and always does better than its base model DeepAR. In addition, the model variance is also reduced by introducing temporal hierarchies for most datasets. Some of the ARIMA variants failed to finish within 24 hours and are not reported.

<table>
<thead>
<tr>
<th>DATASET LEVEL</th>
<th>TAXI-1MIN</th>
<th>TAXI-5MIN</th>
<th>ELEC-15MIN</th>
<th>SOLAR-1H</th>
<th>TRAFFIC-1H</th>
<th>EXCHANGE-RATE-1D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taxi-1min</td>
<td>0.235 ± 0.008</td>
<td>0.307 ± 0.014</td>
<td>0.345 ± 0.003</td>
<td>0.353 ± 0.000</td>
<td>0.335 ± 0.000</td>
<td>0.335 ± 0.003</td>
</tr>
<tr>
<td>Taxi-5min</td>
<td>0.263 ± 0.007</td>
<td>0.330 ± 0.013</td>
<td>0.348 ± 0.002</td>
<td>0.354 ± 0.000</td>
<td>0.355 ± 0.000</td>
<td>0.355 ± 0.003</td>
</tr>
<tr>
<td>ELEC-15MIN</td>
<td>0.109 ± 0.005</td>
<td>0.114 ± 0.004</td>
<td>0.120 ± 0.003</td>
<td>0.121 ± 0.003</td>
<td>0.121 ± 0.003</td>
<td>0.121 ± 0.003</td>
</tr>
<tr>
<td>SOLAR-1H</td>
<td>0.345 ± 0.008</td>
<td>0.353 ± 0.003</td>
<td>0.365 ± 0.002</td>
<td>0.370 ± 0.002</td>
<td>0.371 ± 0.002</td>
<td>0.371 ± 0.003</td>
</tr>
<tr>
<td>TRAFFIC-1H</td>
<td>0.121 ± 0.003</td>
<td>0.121 ± 0.003</td>
<td>0.121 ± 0.003</td>
<td>0.121 ± 0.003</td>
<td>0.121 ± 0.003</td>
<td>0.121 ± 0.003</td>
</tr>
<tr>
<td>EXCHANGE-RATE-1D</td>
<td>0.011 ± 0.003</td>
<td>0.011 ± 0.003</td>
<td>0.011 ± 0.003</td>
<td>0.011 ± 0.003</td>
<td>0.011 ± 0.003</td>
<td>0.011 ± 0.003</td>
</tr>
</tbody>
</table>

Table 2: CRPS loss (the lower, the better) for all aggregation levels.

To evaluate the accuracy of our forecasting models we use the continuous ranked probability score (CRPS) (Matheson and Winkler, 1976). We use a discrete version of the CRPS loss Eq. (9) implemented in GluonTS Alexandrov et al., 2019, where the loss is computed over a finite quantile set and is normalized by the sum of the absolute values of the observations. We use the quantile range from 0.05 to 0.95 in steps of 0.05.

We compare against the following categories of models:

- state-of-the-art univariate local models that do not incorporate temporal hierarchies; these include ETS (Hyndman et al., 2008), ARIMA (Box and Jenkins, 1968), Theta (Assimakopoulos and Nikolopoulos, 2000).
We train and evaluate stochastic models five times and evaluate the models in two parts: in the first half, we do not incorporate temporal hierarchies; these include DeepAR (Salinas et al., 2020), an RNN based univariate autoregressive model, TimeGrad (Rasul et al., 2021), an RNN based multivariate autoregressive de-noising diffusion model, LogSparse (Li et al., 2019), a memory-efficient transformer model suitable for time series at finer granularities, and models that explicitly incorporate temporal hierarchies; Thief (Athanasopoulos et al., 2017) and DeepTHief (Theodosiou and Kourentzes, 2021). For Thief, we consider various combinations of base model and reconciliation strategies as separate models.

Our model uses DeepAR as the base forecasting model and is referred to here as COPDeepAR (shorthand for COherent Probabilistic DeepAR). We implement our model in GluonTS, an open-source forecasting library (Alexandrov et al., 2019) and also use the GluonTS implementations of DeepAR and LogSparse in our experiments. For TimeGrad, we use the implementation available in PyTorchTS (Rasul, 2021). For other competing methods, including ETS, ARIMA, Theta and Thief variants, we use the open source implementations (Hyndman et al., 2008; Athanasopoulos et al., 2017). For DeepTHief we use the implementation provided by one of the authors of Theodosiou and Kourentzes (2021). Note that our model has the same base model hyper-parameters as DeepAR. We use default hyper-parameter values for running all models. For Solar-1h, which is a non-negative dataset with a lot of zero values, we enforce non-negativity of forecasts along with temporal coherence via Dykstra’s projection method as described in Section 3.3.1. Since Dykstra’s method is iterative and more expensive (in contrast to the single-step projection on to null space needed to guarantee coherence), we divide the training in two parts: in the first half, we do not enforce coherence or non-negativity and instead use the negative log-likelihood loss directly on the parameters \( \Theta_t \) of the predictive (Gaussian) distribution; then in the second part of the training we switch to enforcing both coherence and non-negativity by generating samples from \( \Theta_t \) and projecting them via Dykstra’s method. We run all of our experiments on Amazon SageMaker (Liberty et al., 2020). We train and evaluate stochastic models five times and evaluate the CRPS with 100 samples. We report the mean and standard deviation over these five runs. For deterministic models, we report the result of a single run. We supply code as part of the Appendix.

Table 1 provides the results of all models. The best performing model per dataset is highlighted in bold. COPDeepAR outperforms all the compared models in five out of the six datasets. We find that COPDeepAR consistently outperforms its base model DeepAR. COPDeepAR also reduces the variance of DeepAR in all cases. Example forecasts plotted in Figure 4 further show that our model is able to capture the true target by narrower prediction intervals compared to DeepAR; see also Figure 5 for calibration plots. Moreover, by being probabilistic in nature, in comparison to the Thief-methods, COPDeepAR is also able to provide uncertainty estimates to their forecasts, an important requirement in business applications.

Table 2 shows the accuracy at different aggregations levels of the temporal hierarchy. For better readability, we choose the best performing method from all the variants of Thief and report its performance. Column LEVEL indicates the aggregated frequency. Coarser granularities result in lower error, which might be due to noise cancellation through aggregation. Again, COPDeepAR outperforms all Thief variants in five of the six datasets considered.

An additional qualitative experiment and an ablation study that analyze the effectiveness of the GNN layer are presented in the Appendix.

6 CONCLUSION

We presented an end-to-end model for simultaneously forecasting at different time granularities of a given temporal hierarchy by learning joint embeddings with a graph neural network while respecting the aggregation structure. We demonstrated that this strategy reduces the forecasting error of noisy time series sampled at finer frequencies. Two limitations of our method are that the computational cost of the reconciliation grows with the number of samples and that we require parametrizable distributions. These could be overcome by considering other reconciliation methods and non-parametric distributions. Our method could also be extended to joint cross-sectional and temporal hierarchies.

References


A Potential Negative Societal Impact

Our method itself is generic and can be applied to any forecasting problem with temporal hierarchies. One of our contributions is that our method improves the forecast on noisy time series sampled at fine granularities. One particular instance of these fine granularity time series could be data of individuals (for example, user data). Therefore, our method could be used to make decisions on individuals because these decisions are in line with strategic considerations at higher aggregation levels. These decisions could lead to negative consequences for individuals that are not grounded because our method does not consider causality.

B Comments on Matrix $A$ used in Graph Neural Network

Here, we would like to illustrate the new variant of the adjacency matrix introduced in our paper using an example temporal hierarchy shown in Figure 6. Recall that we defined the structure of the matrix $A$ for the underlying GNN layer as

$$ A := (A_{\text{acc.}} + A_{\text{dist.}} + A_{\text{ret.}}) / 3, $$

where $A_{\text{acc.}}$ is the standard adjacency matrix corresponding to the hierarchy tree where the undirected edges are replaced by directed edges pointing downwards, while $A_{\text{dist.}}$ is the normalized adjacency matrix but for the directed tree where the edges are pointing upwards. And $A_{\text{ret.}}$ is the identity matrix.

For the temporal hierarchy given in Figure 6 we have

$$ A_{\text{acc.}} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A_{\text{dist.}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \end{pmatrix}. $$

We want to highlight the connection between these matrices and the typical top-down, bottom-up reconciliation strategies used in the hierarchical literature. In the standard top-down approach, only the values of the root are distributed to every other node in the tree, whereas in our case, every level distributes its own values to the upper level as well as to the lower level. The examples matrices shown above are quite different from the matrices used for top-down and bottom-up reconciliation strategies (see Chapter 10 in Hyndman and Athanasopoulos (2018)). Essentially, multiplying by $A$ corresponds to doing top-down, bottom-up and middle-out (at every middle level) reconciliations simultaneously.

C Dataset Summary & Experiment Details

The summary of the datasets is given in Table 3. Note that we do evaluation on rolling-predictions; i.e., at a time, each model produces forecasts for $\tau$ time steps given data until the beginning of forecast horizon and then the data is rolled over to obtain forecasts for the next $\tau$ time steps. This is repeated for $k$ times; this number is given under the column No. Rolls in Table 3. In case of local models (i.e., non-deep learning based models), we retrain the model for every roll, whereas for deep learning models DeepTHieF, LogSparse, TimeGrad, DeepAR and COPDeepAR we use a single model that is trained using the data until the beginning of forecast start time corresponding to the first roll.

Choice of temporal hierarchy. We chose the temporal hierarchy with the following heuristics depending on the granularity of the base time series: (i) the coarsest aggregation level (i.e., root of the temporal hierarchy) should still preserve the seasonal patterns present in the base time series; e.g., not aggregating hourly time series to daily aggregation if there is a clear day vs night patterns in the hourly data, (ii) there is relative continuity between the successive aggregation levels;

![Figure 6: Example of temporal hierarchy for 15min frequency, aggregated to 30min and 1h frequency.](image-url)
Figure 7: First two PCA components of the time series embeddings at different aggregation levels of one forward pass of COPDeepAR. The different aggregation levels are color coded. Diamonds denote the embeddings of a single time series to illustrate that our findings also hold for a single time series example. Left: Before the GNN layer, the embedding are clustered, which suggests that the embeddings are dissimilar to each other. Right: After the GNN layer, the embeddings at different aggregation levels do not form clusters. This suggests that the GNN layer facilitates information sharing between the aggregation levels.

e.g., not aggregating 1-min time series directly to daily aggregation without any intermediate aggregations. Using these heuristics, we aggregated Taxi-1min and Taxi-5min and Elec-15min datasets to 30-min & hourly aggregations. These aggregations can be specified by aggregation multiples as depicted in Table 3 under the column HIERARCHY. For the Solar-1H dataset, we chose 8 as the aggregation multiple since eight hours constitute the night interval where the observation values are zeros. Aggregating it to any coarser level will cancel the night vs day pattern present in this dataset. We kept the same aggregation multiple for the hourly Traffic-1H dataset as well. Finally ExchangeRate-1D is a daily dataset and we aggregated it to the weekly level. Note that the base frequency of ExchangeRate-1D is "Business day" and hence the corresponding aggregation multiple is 5, as five business days constitute one week. In general, the choice of temporal hierarchy is application-dependent and in the absence of such a choice, experimenting with different hierarchies might be beneficial. Automatically selecting the temporal hierarchy based on the data and systematic analysis of its impact on forecast accuracy could be studied in future work.

<table>
<thead>
<tr>
<th>DATASET</th>
<th>NO. TIME SERIES</th>
<th>HIERARCHY</th>
<th>τ</th>
<th>NO. ROLLS</th>
<th>FREQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taxi-1min</td>
<td>185</td>
<td>[60, 30, 1]</td>
<td>180</td>
<td>8</td>
<td>1-MIN</td>
</tr>
<tr>
<td>Taxi-5min</td>
<td>185</td>
<td>[12, 6, 1]</td>
<td>144</td>
<td>2</td>
<td>5-MIN</td>
</tr>
<tr>
<td>Elec-15min</td>
<td>319</td>
<td>[4, 2, 1]</td>
<td>96</td>
<td>1</td>
<td>15-MIN</td>
</tr>
<tr>
<td>Solar-1H</td>
<td>137</td>
<td>[8, 1]</td>
<td>24</td>
<td>7</td>
<td>1-HOUR</td>
</tr>
<tr>
<td>Traffic-1H</td>
<td>862</td>
<td>[8, 1]</td>
<td>24</td>
<td>7</td>
<td>1-HOUR</td>
</tr>
<tr>
<td>ExchangeRate-1D</td>
<td>8</td>
<td>[5, 1]</td>
<td>30</td>
<td>5</td>
<td>1-BUSINESS DAY</td>
</tr>
</tbody>
</table>

Table 3: Summary of the datasets used in this paper. Here τ refers to the prediction length and the number of rolls refer to the number of evaluation windows (where each window is of length τ). The hierarchy refers to the multiples used for temporal aggregation. Note that for the ExchangeRate-1D dataset, the aggregation multiple is 5, since five business days constitute one week.

## D Additional Experiments

### D.1 Qualitative Evaluation

We analyzed the embeddings obtained by our model before and after applying the GNN layer on the Taxi-5min dataset. We took the embeddings of all time series in a batch at all nodes of the hierarchy corresponding to all sample-paths (32 x 100 samples x 15 nodes) and applied PCA to it. The first two principal components are visualized in Figure 7 (both before & after the GNN layer) where the embeddings are coloured by the corresponding time granularity; we also highlighted the embeddings of the first time series in the batch by a different marker (diamond). The embeddings before the GNN layer clearly reveal the underlying cluster structure separating the three levels of the hierarchy, which shows that these
embeddings at different aggregation levels are highly dissimilar (left-side plot). The GNN layer makes them more similar (right-side plot) where the embeddings at different levels lie on top of each other, removing the cluster structure. This also holds for a single time series (exemplary shown for one time series by diamonds) and shows that the GNN indeed facilitates the sharing of information. Figure 4 in the main paper depicts that this qualitatively leads to improved forecasts because longer-running seasonality can be picked up more effectively.

### D.2 Ablation Studies

In this section, we present ablations to study the effect of the proposed graph neural network layer in our model and the new variant of the adjacency matrix $A$, which we introduced for message passing within the GNN layer. Recall that the GNN layer in our model takes as input the embeddings $H^{(0)}_t$ from the individual RNNs and successively applies the following transformations $L$ times, where the $L$ is the number of levels of the hierarchy:

$$H^{(l+1)}_t = \sigma(AH^{(l)}_t W), \quad l = 0, 1, \ldots, L - 1.$$ 

Here $W$ denotes the learnable weights of a neural network layer. We evaluate the proposed method COPDeepAR with the following variants:

- **WithoutGNN**: In this variant, we completely remove the GNN layer from our model architecture. That is $H^{(L)}_t = H^{(0)}_t$.
- **GNN-Std-Adj**: In this variant, we keep the GNN layer but use the standard adjacency matrix, denoted $A_{\text{std}}$, in place of the proposed matrix for $A$. Note that multiplication with the standard adjacency matrix simply adds up the features vectors of all neighbours but not the node itself thus ignoring the own embeddings. It is standard practice (Kipf and Welling 2017) to add an identity matrix to the adjacency matrix (equivalent of having self loops in the graph) and hence we use the same:

$$A = A_{\text{std}} + I,$$

where $I$ is the identity matrix. Note that we also tested variants where we do not add the identity matrix and the corresponding results were worse than those obtained with the identity matrix.
- **GNN-Norm-Adj**: Since $A_{\text{std}}$ is normalized, the scale of the features are changed when multiplied with $A$. Hence, often in practice, the normalized adjacency matrix is used instead:

$$A = D^{-1}(A_{\text{std}} + I),$$

where $D$ is the diagonal degree matrix corresponding to the adjacency matrix $A_{\text{std}} + I$.
- **GNN-Symm-Norm-Adj**: Since above normalization yields an asymmetric matrix, some works like Kipf and Welling (2017) consider a symmetric version:

$$A = D^{-1/2}(A_{\text{std}} + I)D^{-1/2},$$

where $D$ is again the diagonal degree matrix corresponding to the adjacency matrix $A_{\text{std}} + I$.
- **GNN-Without-MLP**: In the final variant, we keep the proposed matrix for $A$ (Eq. (7)) but remove the learnable neural network layer from the GNN. That is, the embeddings are transformed via non-learnable, linear transformation:

$$H^{(l+1)}_t = AH^{(l)}_t, \quad l = 0, 1, \ldots, L - 1.$$ 

Comparing WithoutGNN versus the rest, we see that the use of the GNN layer helps in three out of the six datasets and achieves the same (Traffic-1H) or on-par results (Elec-15min) in the two of the remaining three datasets; in the case of ExchangeRate-1D, the magnitude of the error is already low (0.01 or 1%) and the use of GNN makes it slightly worse 0.011 or 1.1%. The improvement is more prominent in the case of Taxi-5min where the CRPS is reduced by 9% with the help of GNN layer. Since these improvements outweigh the (small) deterioration of results in case of other (and maybe easier) datasets, we suggest in general to tune the models by treating this (enable/disable GNN layer) as a hyper-parameter. Moreover, using the proposed variant of the adjacency matrix for $A$ (Eq. 7) helps achieve better or on-par results than using any variant of the adjacency matrix used in practice. This difference is again significant for Taxi-5min dataset where the proposed matrix $A$ yields 4% better results than any of the standard (normalized and unnormalized) adjacency matrices. Finally, as seen from the result obtained for GNN-Without-MLP versus COPDeepAR, using a dense layer in GNN, on top of using the proposed matrix for $A$, further helps in achieving better results.
Table 4: CRPS loss for ablation variants of COPDeepAR. Here we evaluate the effect of the GNN layer as well as the individual components within the GNN layer; more specifically, the choice of the adjacency matrix for the GNN layer and the use of the dense layer within the GNN.