

Spatio-Temporal Graph Neural Networks for Aggregate Load Forecasting

Simone Eandi

Andrea Cini, Slobodan Lukovic

Cesare Alippi

¹Università della Svizzera italiana ¹IDSIA, Università della Svizzera italiana ¹IDSIA, Università della Svizzera italiana
eandi.simo@outlook.it ²Politecnico di Milano

Abstract—Accurate forecasting of electricity demand is a core component of the modern electricity infrastructure. Several approaches exist that tackle this problem by exploiting modern deep learning tools. However, most previous works focus on predicting the total load as a univariate time series forecasting task, ignoring all fine-grained information captured by the smart meters distributed across the power grid. We introduce a methodology to account for this information in the graph neural network framework. In particular, we consider spatio-temporal graphs where each node is associated with the aggregate load of a cluster of smart meters, and a global graph-level attribute indicates the total load on the grid. We propose two novel spatio-temporal graph neural network models to process this representation and take advantage of both the finer-grained information and the relationships existing between the different clusters of meters. We compare these models on a widely used, openly available, benchmark against a competitive baseline which only accounts for the total load profile. Within these settings, we show that the proposed methodology improves forecasting accuracy.

Index Terms—Spatio-Temporal Graph Neural Network, Smart Grid, Electric Load Forecasting

I. INTRODUCTION

Efficient production, distribution, and management of electricity play a vital role in our modern society, especially considering the growing trend in global electricity demand and the strong correlation between electricity usage and economic growth [1], [2]. The ability to accurately forecast future electricity demand is a crucial component for many areas within the electric power industry: it allows companies to guarantee a more efficient distribution of electricity to end users, helps in devising more effective pricing strategies, and provides support for scheduling grid maintenance operations [3]–[5]. Most research efforts aim at devising methods for accurately predicting the total electric power consumption over a time horizon ranging from a few hours to a few days, a problem typically

referred to as Short-Term (total) Load Forecasting (STLF). In recent years, in particular, there has been considerable interest in using Deep Learning (DL) methods to tackle this task [6]–[8]. Modern smart grids have brought several innovations to traditional power grids, including the deployment of extensive smart metering infrastructures that gather fine-grained information about power grid usage [5]. Despite the availability of this large amount of high-resolution data, most DL methods for total load forecasting make predictions based only on the historical values of the total load, modeling the problem as a univariate time series forecasting task and ignoring all additional information coming from meters. Approaches that leverage this information do exist, but they mostly focus on making predictions at the level of individual meters or households [9]. Fig. 1 summarizes the different approaches that one might consider: while the aforementioned methods fall within the class of models shown in the first and second column, respectively, there is a third option. In fact, the class of models shown in the third column (f_3) exploits both global and fine-grained information to make predictions. We focus on architectures within this class.

In this paper, we exploit graph processing tools and modern Graph Neural Networks (GNNs) to take advantage of data captured by smart meters in combination with previous values of the total load to achieve accurate load forecasts. In fact, it is worth mentioning that in realistic settings the total aggregate load differs from the simple sum of loads relative to the single smart meters due to partial grid coverage and transportation losses. The aim of this work is, then, to propose novel GNN architectures tailored to aggregate load forecasting, as well as to investigate a methodology to learn from high-resolution data to assess if any improvements in total load forecasting are attainable. To this end, we propose to model input data as a Graph Time Series, where nodes are associated with time series coming from smart meters and a global graph-level attribute accounts for the total load time series. In this setting, the total load forecasting problem is formulated as the task of making predictions for future values of this global graph-level attribute. Within this framework, we propose two novel spatio-temporal graph neural networks (STGNNs), namely Global Graph WaveNet and Global Temporal Graph Attention Network, based on state-of-the-art methods and techniques that emerged in the STGNNs literature [10], [11]. In particular, both models account for global graph-level attributes – a

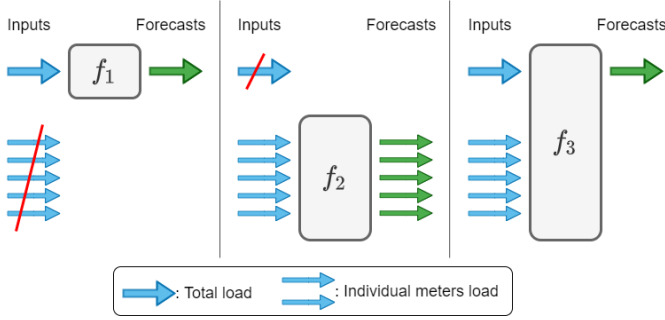


Fig. 1. Three load forecasting archetypes. Model f_1 predicts the total load as a single time series problem. Model f_2 uses the multiple time series from the smart meters to make predictions at the individual level. Model f_3 instead uses the loads of individual meters as additional input data to forecast the aggregate load.

novel contribution in these settings – and their architecture is formalized within the flexible Graph Network framework [12]. As a further novel investigation, we propose, instead of relying on the erratic load associated with every single meter, to adopt a cluster-based approach [13] where each node corresponds to a cluster-level aggregate. Besides the advantage of dealing with smoother signals instead of raw data, clustering allows scaling the method to settings characterized by a huge number of smart meters, which is the typical condition in modern smart grids. Finally, we evaluate the proposed solution on an openly available benchmark, where our approach is compared against a sensible baseline that models the problem of aggregate load forecasting within a univariate time series forecasting framework.

The paper is organized as follows. Section 2 covers related works, while Section 3 provides the necessary preliminaries and a formalization of the problem. Then, in Section 4, we present the proposed approach and discuss the design principles underlying the overall architecture. We show experimental results in Section 5, and, finally, draw our conclusions in Section 6.

II. RELATED WORKS

There is a large literature studying the application of modern DL methods for time series forecasting (TSF) [14] in the context of STLF. These include Long Short-Term Memory networks (LSTM) [6], [15]–[17], Convolutional Neural Networks (CNNs) [7], [8], [18], [19], Sequence-to-Sequence (Seq2Seq) architectures and even attention-based neural networks [20]. Notably, Gasparin et al. [21] carry out a comprehensive study comparing several of these architectures on electric load datasets. Empirical results show that for the total load forecasting task, LSTMs achieve the best results, closely followed by Seq2Seq models and Temporal Convolutional Networks (TCNs). The study concludes that, given their effectiveness and the little attention that TCNs have received in STLF [7], [22], [23], they constitute a very promising research direction.

Concerning STLF at the smart meter level, most approaches have focused on forecasting the load profile of each meter

independently of the others, i.e., by modeling only temporal relationships within each time series and failing to consider relationships among them [24], [25], even if there is plenty of evidence that these relationships do matter [13], [26], [27]. Recently, however, STGNNs have been exploited to model spatio-temporal relationships within multivariate time series by looking at these data from the graph-processing perspective [10], [11], [28]–[30]. In particular, STGNNs have been used to forecast electric load of residential houses [9], [31] and to impute missing data in energy consumption patterns [30], [32].

Among methods related to our work, Cini et al. [13] propose a novel approach based on multitask learning for cluster-based aggregate forecasting, which is the task of predicting the aggregated load of a set of meters taking as inputs cluster-level aggregates of load profiles. As already mentioned, two notable benefits of cluster-based forecasting are: (i) scalability to large datasets with thousands of meters and (ii) reduced volatility of the time series. The model proposed in [13] can implicitly learn spatial relationships between clusters, however, it does not make use of graph processing tools in the forecasting architecture. To the best of our knowledge, our approach is the first attempt to exploit the effectiveness of graph neural networks in dealing with spatio-temporal data for aggregate load forecasting by leveraging a total load time series together with cluster-level load profiles. In this regard, we further stress that in realistic settings not all components that contribute to the total load are measured and, thus, taking into account only information coming from available meters is not enough.

III. PRELIMINARIES

A. Problem formulation

We model the multiple time series captured by the N smart meters in the grid and the one corresponding to the total load (measured upstream by the energy provider) as a Graph Time Series and represent it as a sequence $\mathcal{G}$ of graph snapshots G_t at discrete time intervals t , i.e., $\mathcal{G} = \{G_t\}_t$. Each graph G_t is defined as a triplet $G_t = (X_t, A_t, u_t)$, where $X_t \in \mathbb{R}^N$ is a node attribute vector, such that $X_{t,i}$ is the load measured by the i -th node (meter) at t -th time step. $A_t \in \mathbb{R}^{N \times N}$ is a weighted adjacency matrix defining relationships between each pair of nodes (i, j) . This matrix is assumed to be latent and not known a priori. Finally, the global attribute $u_t \in \mathbb{R}$ corresponds to the total load at time t . Given this formulation, the task of learning a predictor of the total load for the next H time steps, with H being the forecasting horizon, given the values of node and global attributes for the last T time steps can be framed as learning a model

$$\hat{y}_{t:t+H} = f(X_{t-T:t}, u_{t-T:t}; \Theta) \quad (1)$$

where Θ are the model parameters and $\hat{y}_{t:t+H}$ are the predictions for the ground truth values of the global attribute for the next H time steps, i.e., $y_{t:t+H} = u_{t:t+H}$. The shorthand notation $a_{b:c}$ is used to compactly refer to the sequence of

values $a_{b+1}, \dots, a_c$, that is, the lower extreme is excluded, while the upper one is included.

B. The Graph Network Framework

Next, we briefly review the Graph Network (GN) framework [12] as it serves both as an introduction to a wide range of Graph Neural Networks [33] and as a conceptual framework for the proposed models. A GN is formalized as a composition of several blocks. Each block is implemented by a set of differentiable functions mapping a set of input nodes, edges, and global attributes to their corresponding updated representations. In particular, the block is defined by six functions, three for updating node, edge and global attributes, ϕ^v , ϕ^e and ϕ^u respectively, one propagation function from nodes to edges, $\rho^{v \rightarrow e}$ and two additional propagation functions from nodes and edges to global attributes, $\rho^{v \rightarrow u}$ and $\rho^{e \rightarrow u}$. Many GNNs can be modeled as an instance of this general model. We refer to the original paper for more details. This framework can be extended to Spatio-Temporal GNNs by letting the inputs and outputs of the GN block have an additional temporal dimension (or a memory state) on which the ϕ and ρ functions can operate. In our context, the set of vertex attributes V corresponds to the node attributes $X_{t:T}$, while we do not consider edge attributes.

C. Clustering

As previously mentioned, time series associated with smart meters are not used directly but are first aggregated into clusters. We adopt the technique proposed in [13] to compute the cluster assignments: (1) we compute a $N \times N$ similarity matrix between meters, (2) build a k-nearest neighbor graph out of this matrix, and (3) perform spectral clustering on the graph to compute cluster assignments. In particular, the similarity between each pair of nodes i and j is computed as the *correntropy* [34] on the weekly load profiles of the nodes:

$$\text{sim}_{i,j} = \frac{1}{N_{\text{weeks}}} \sum_{w=1}^{N_{\text{weeks}}} \frac{1}{2\pi\sigma} e^{-\left(\frac{X_i^{(w)} - X_j^{(w)}}{2\sigma^2}\right)} \quad (2)$$

where $X_i^{(w)}$ are load measurements corresponding to w -th week for the i -th meter and σ is a hyperparameter controlling the width of the kernel.

We indicate as $\mathcal{D} = \{x_1, \dots, x_N\}$ the original dataset with x_i being the time series recorded by the i -th smart meter. Then, we define partitions $\mathcal{S}_k$ as the set of meters IDs assigned to the k -th of K clusters produced by the aforementioned clustering algorithm. Finally, we obtain the new dataset of cluster-level aggregates $\mathcal{D} = \{x'_1, \dots, x'_K\}$ as

$$x'_k = \sum_{i \in \mathcal{S}_k} x_i. \quad (3)$$

As already mentioned, cluster-level aggregates are used to train the models and make predictions.

Note that while the clustering procedure has a cubic computational cost, it only has to be performed once.

D. Exogenous variables

We consider exogenous variables consisting of time of the day, day of the week, and whether the current day is a national holiday or not and collectively denote them as u_t^{exg} . Each node-level and graph-level time series is conditioned on these variables independently before any graph convolution. To do so, we use a trainable conditional block that takes the unconditioned time series $x_{t-T:t}$ (e.g., for one meter), where T indicates the window length, and the exogenous variables at the same time steps and outputs a C -dimensional embedding $z_{t-T:t}$. In particular, the conditional block is based on [22] and works as follows: two gated TCNs [10] process in parallel $u_{t-T:t}^{\text{exg}}$ and $x_{t-T:t}$ resulting in two C -dimensional intermediate representations for each time step, these are then further processed with 1×1 convolutional filters and finally added together resulting in the conditioned input time series $z_{t-T:t}$. Residual connections between input and (intermediate) output representations can be added to facilitate gradient flow and prevent temporal smoothing of features. The result of this sequence of operations is embedded node and global attributes $Z_{x,t-T:t}$ and $Z_{u,t-T:t}$, respectively, which are the actual inputs given to predictors. Note that the weights of the conditional blocks are learned end-to-end together with the predictor and that these weights are shared among nodes.

IV. PROPOSED ARCHITECTURES

Since the proposed models share some of their components, we start by describing those.

A. General Framework

Both the proposed spatio-temporal GNNs have an encoder-decoder architecture where the encoder first extracts a C -dimensional feature vector for both global attributes, $Z_u^{\text{enc}} \in \mathbb{R}^{T \times C}$, and nodes, $Z_x^{\text{enc}} \in \mathbb{R}^{T \times N \times C}$ which are then used by the decoder as inputs to make the predictions. In particular, a *global decoder* is responsible for forecasting the total load $y_{t:t+H}$ from Z_u^{enc} , while a *node decoder* outputs node-level predictions that are useful for defining auxiliary training objectives. The encoder is a stack of L identical layers, where each l -th layer can be framed as a GN block implementing the ϕ^v , $\rho^{v \rightarrow u}$ and ϕ^u functions, taking as input the residual node and global features extracted by the previous layer, $\tilde{Z}_x^{(l-1)}$ and $\tilde{Z}_u^{(l-1)}$ respectively, and computing the updated representations $Z_x^{(l)}$ and $Z_u^{(l)}$. These are added back with the layer inputs through residual connections and then fed in the next layers; we indicate the features after the residual connection as $\tilde{Z}_x^{(l)}$ and $\tilde{Z}_u^{(l)}$. The final representations, Z_x^{enc} and Z_u^{enc} , are instead obtained by WaveNet-style skip connections coming from each one of the previous layers (e.g., see Fig. 2).

The decoder is implemented by exploiting the popular softmax attention mechanism, akin to the one introduced by the Transformer architecture [20]. In particular, the H -dimensional intermediate features for the global decoder,

Z_u^{dec} are computed as a weighted sum of features Z_u^{enc} at each time step as:

$$Z_u^{dec} = W^{atn} Z_u^{enc} \quad (4)$$

$$W^{atn} = \text{Softmax} \left(\frac{QW^Q (KW^K)}{\sqrt{D_{atn}}} \right) \quad (5)$$

where $W^{atn} \in \mathbb{R}^{H \times T}$ are the attention weights (note that H refers to the forecasting horizon) and $W^Q \in \mathbb{R}^{d_{exg} \times d_{atn}}$ and $W^K \in \mathbb{R}^{d_{exg} \times d_{atn}}$ are trainable weights used to project Q and K , respectively. Queries and keys are given by the exogenous variables corresponding, respectively, to the future H time steps and to the previous T time steps:

$$Q_\tau = u_\tau^{exg} \quad \tau = t+1, \dots, t+H \quad (6)$$

$$K_\tau = u_\tau^{exg} \quad \tau = t-T+1, \dots, t \quad (7)$$

Finally, the total load forecasts $\hat{y}_{t:t+H}$ are computed by feeding Z_u^{dec} to a Feed-Forward Neural Network (FFNN), i.e., a nonlinear readout, which computes predictions for each time step independently by using shared parameters. The node decoder is implemented analogously, with its own encoder-decoder attention layer and FFNN processing each node independently.

The Global Graph WaveNet and Global Temporal Graph Attention Network, however, differ in how the ϕ^v , $\rho^{v \rightarrow u}$ and ϕ^u of each encoder layer is implemented, in the next subsections, we look at these modules.

B. Global Graph WaveNet

The l -th encoder layer of Global Graph WaveNet (GGWN) extracts features coming from input node residuals $\tilde{Z}_x^{(l-1)}$ akin to Graph WaveNet [10]. First, inputs are processed along the temporal dimension by a gated TCN [10], which extracts intermediate hidden representations $H_x^{(l)}$, then these are processed along the spatial axis by a graph convolutional filter [11] using a learned adjacency matrix $A^{(l)}$ defining the diffusion process between nodes and producing the updated node features $Z_x^{(l)}$. Analytically, this corresponds to:

$$H_x^{(l)} = \tanh \left(W_{T,1}^{(l)} *_{\mathcal{T}} \tilde{Z}_x^{(l-1)} \right) \odot \sigma \left(W_{T,2}^{(l)} *_{\mathcal{T}} \tilde{Z}_x^{(l-1)} \right) \quad (8)$$

$$Z_x^{(l)} = \sum_{k=1}^K A^{(l)k} H_x^{(l)} W_S^{(l)} \quad (9)$$

with $W_{\mathcal{T}}$ being the gated TCN weights, $*_{\mathcal{T}}$ the temporal convolution operator, σ the sigmoid function and W_S being the weights for the diffusion convolutional operator of order K . Notably, differently from Graph WaveNet, here no fixed adjacency matrix is used; more on this in the experimental section. This sequence of operations implements the ϕ^v module for GGWN, as shown in Fig. 2.

To exploit node features extracted in $Z_x^{(l)}$ to improve the global time series predictions, we implement the $\rho^{v \rightarrow u}$ module as a learnable global pooling layer, collapsing the spatial dimension to produce a global representation $H_u^{(l)} \in \mathbb{R}^{T \times C}$.

This is achieved through a weighted sum of $Z_x^{(l)} \in \mathbb{R}^{T \times N \times C}$ across the spatial dimension defined as:

$$H_u^{(l)} = \sum_{i=1}^N W_{gp,i}^{(l)} Z_{x,i}^{(l)} \quad (10)$$

$$W_{gp}^{(l)} = \text{Softmax} \left(W^{(l)} \right) \quad (11)$$

where the pooling weights $W^{(l)} \in \mathbb{R}^N$ are trainable parameters for the l -th layer, allowing each layer to learn different scores for the nodes when aggregating them.

Finally, the pooled node features are combined with the global features computed by the previous layer, $\tilde{Z}_u^{(l-1)}$, into the final global features $H_{u,x}^{(l)}$ by means of a simple concatenation over the channel dimension:

$$H_{u,x}^{(l)} = \tilde{Z}_u^{(l-1)} \parallel H_u^{(l)} \quad (12)$$

and then fed into a gated TCN which acts as ϕ^u module and computes updated global features $\tilde{Z}_u^{(l)}$.

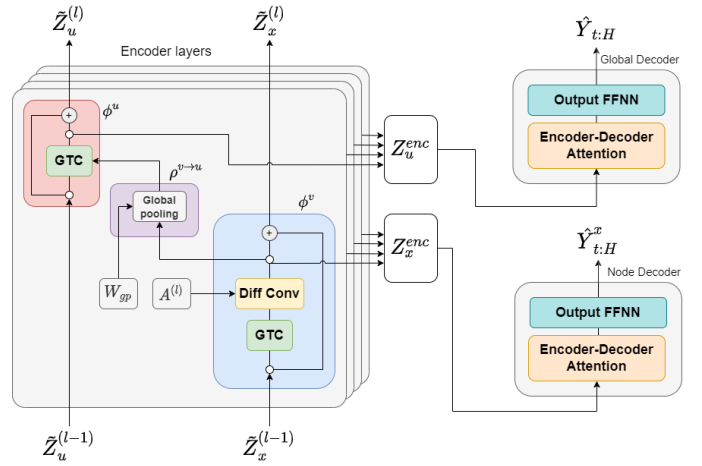


Fig. 2. The Global Graph WaveNet model architecture. Each encoder layer is a GN block where ϕ^v is implemented as a Graph WaveNet layer, $\rho^{v \rightarrow u}$ is a trainable global pooling operator and ϕ^u is a simple gated TCN (GTC).

C. Global Temporal Graph Attention Network

The architecture of the Global Temporal Graph Attention Network (GTGAN) differs from that of GGWN only for the ϕ^v and ϕ^u modules in the encoder layers, which in the case of GTGAN are implemented as a combination of multiple attention mechanisms. In particular, GTGAN computes messages exchanged in the propagation layer with a spatio-temporal attention mechanism, where a multilayer perceptron (MLP) computes attention scores by taking as input flattened subsequences. The overall architecture is shown in Fig. 3. The ϕ^v module learns input node representations along the temporal dimension by using a gated TCN analogously to GGWN. However, as already mentioned, it exploits a different mechanism to propagate information spatially among nodes based on message passing and attention. For the sake of clarity, this sequence of operations is described here at the node level.

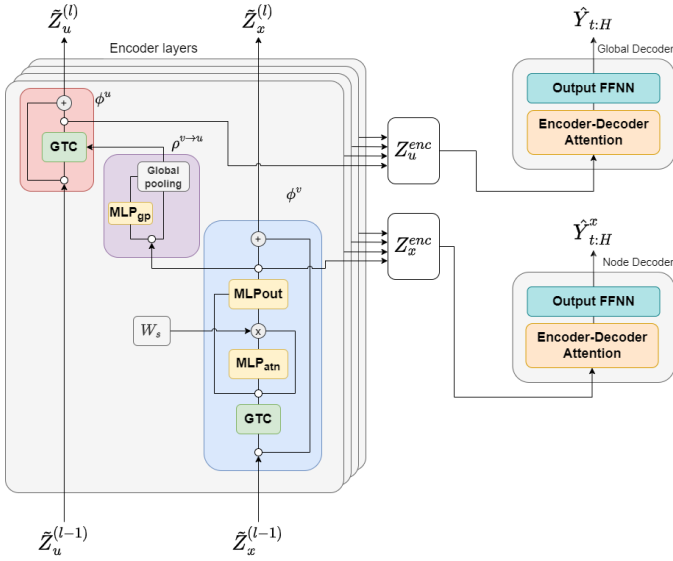


Fig. 3. The architecture of the Global Temporal Graph Attention Network. Node features are updated through a gated-TCN and a message-passing operation (ϕ^v), then pooled together using a weighted sum controlled by a self-attention mechanism ($\rho^{v \rightarrow u}$) and finally combined with the global attribute and fed to another gated-TCN extracting the new global features (ϕ^u).

For each couple of i -th and j -th nodes, the l -th layer computes an attention weight vector $\alpha_{ij}^{(l)} \in \mathbb{R}^T$ by feeding concatenated node features through an MLP, denoted as MLP_{atn} :

$$C_{(i,j)}^{(l)} = H_{x,i}^{(l)} \| H_{x,j}^{(l)} \quad (13)$$

$$\alpha_{ij} = \sigma \left(MLP_{atn} \left(\text{t-flatten} \left(C_{(i,j)}^{(l)} \right) \right) \right), \quad (14)$$

where $H_{x,i}^{(l)} \in \mathbb{R}^{T \times C}$ are the temporal features extracted by the TCN for the i -th node, $C_{(i,j)}^{(l)}$ are the concatenated features of nodes i and j and σ is the sigmoid activation function. In particular, the temporal dimension of the concatenated features $C_{(i,j)}^{(l)}$ is flattened before feeding them to the MLP and recovered after the nonlinearity, that is, MLP_{atn} is a mapping $\mathbb{R}^{T \cdot 2 \cdot C} \rightarrow \mathbb{R}^T$. The aggregate message directed to node i , denoted as $M_i^{(l)}$, is computed as a weighted sum of the features of all nodes in the graph (self loops included) with weights determined by the $\alpha_{i,j}^{(l)}$ coefficients and an additional set of weights $w_{i,j}^{(l)} \in \mathbb{R}$ given by the node-wise *softmax* of a learned weighted adjacency matrix $A^{(l)}$ (which increases sparsity), such that:

$$M_i^{(l)} = \sum_{i,j} w_{i,j}^{(l)} \alpha_{i,j}^{(l)} \odot H_{x,i}^{(l)} \quad (15)$$

Note that the scalar $w_{i,j}^{(l)}$ captures static relationships between nodes, while the vector $\alpha_{i,j}^{(l)}$ can learn different relationships at different time steps. The idea of combining a static message weighting mechanism with a dynamical one was inspired by [35]. The representation for the i -th node is then updated

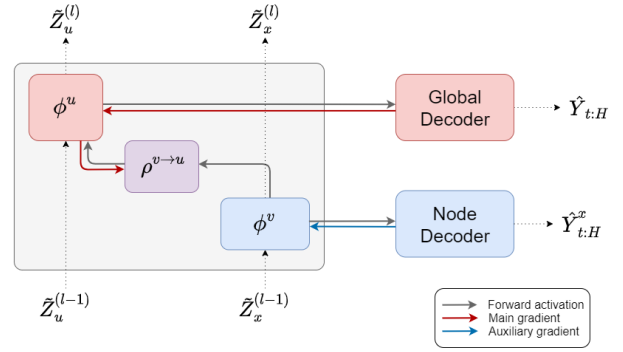


Fig. 4. Forward and backward activation paths for one encoder layer of the proposed models. The gradient back-propagated from ϕ^u is stopped before reaching ϕ^v . Gradients coming from successive layers are omitted for simplicity.

by another MLP, denoted as MLP_x :

$$Z_{x,i}^{(l)} = MLP_x \left(H_{x,i}^{(l)} \| M_i^{(l)} \right) \quad (16)$$

Regarding the $\rho^{v \rightarrow u}$ propagation function, global features extracted from nodes, $H_u^{(l)}$, are again obtained as a weighted sum of updated node features $Z_x^{(l)}$, however, in the case of GTGAN, weights are computed using an attention-based approach:

$$w_i^{(l)} = \sigma \left(MLP_{gp} \left(Z_{x,i}^{(l)} \right) \right) \quad (17)$$

$$H_u^{(l)} = \sum_{i=1}^N w_i^{(l)} Z_{x,i}^{(l)} \quad (18)$$

where $Z_{x,i}^{(l)}$ are the new node features for the i -th node, σ is the sigmoid function, $w_i^{(l)}$ are the graph pooling weights and $MLP_{gp} : \mathbb{R}^C \rightarrow \mathbb{R}$ is the MLP computing them. Similarly to GGWN, $H_x^{(l)}$ are concatenated with global features computed by the previous layer and fed into a gated TCN which extracts the new $Z_u^{(l)}$ representation.

D. Training and gradient flow

In addition to the main training objective, we also introduce an auxiliary training objective for node-level representations learned by the node decoder. In particular, we train each node representation to predict the future *total* load. The reason behind this additional objective is to compensate for the large amount of input data that the proposed models receive during each forward pass by providing an additional learning signal. Overall, the training loss based on the mean absolute error and including L2 regularization becomes the following:

$$\mathcal{L}(\hat{Y}, \hat{Y}^x, Y) = \frac{1}{B} \sum_{\tau=1}^B \left(\left| \hat{Y}_\tau - y_\tau \right| + \gamma \left| \hat{Y}_\tau^x - Y_\tau^x \right| \right) + \gamma_{L2} \|\Theta\|_2 \quad (19)$$

$$Y_{\tau,i}^x = y_\tau$$

where B is the batch size, τ denotes sample indexes inside a batch, $\hat{Y}_\tau$ and $\hat{Y}_\tau^x$ are the global and node-level predictions

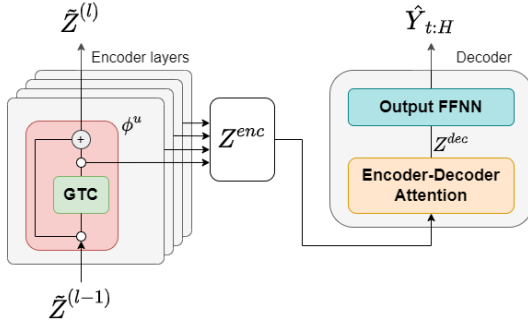


Fig. 5. The TCN baseline

made by the model, y_τ is the ground truth total load for the τ -th training sample, Θ are the model parameters, γ allows for different weights w.r.t. the main and auxiliary task, and γ_{L2} is the L2 regularization weight.

In preliminary experiments, both models showed to be prone to overfitting. We found that stopping the gradient flowing from the graph-level objective to the node representations is an empirically effective solution for this issue. In particular, we stop the gradient flow from ϕ^u to ϕ^v , as shown in Fig. 4. As a result, ϕ^v learns to extract features from nodes depending only on the auxiliary training objective and not on the graph-level task. The auxiliary task, however, is aligned with the total load forecasting problem by construction.

V. EXPERIMENTS AND DISCUSSION

A. Baseline and Dataset

To assess the performance of the proposed models, we compare them against a baseline consisting of a univariate forecasting model with an architecture similar to GGWN and GTGAN, but where each encoder layer implements only the ϕ^u module, ignores all nodes' data and has no node decoder. We refer to this model as the TCN baseline, Fig. 5. This effectively acts as a univariate time series predictor for the total load, and provides a baseline suitable to determine whether some gain in performance can be obtained by considering all the additional information captured by smart meters.

We evaluate the proposed models against the TCN baseline on the CER-E dataset [36], an openly available collection of electric load profiles from both residential households and Small and Medium Enterprises (SMEs) that are served by the Electric Ireland company and gathered as part of the Smart Metering Project sponsored by the Irish Commission for Energy Regulation (CER). The CER-E dataset consists of profiles for over 6000 smart meters: each profile contains the electricity demand in 30 minutes intervals, measured in kiloWatt-hour, from 14 July 2009 up to 31 December 2010. Previous works [30] have already shown how graph representations can capture useful relationships within this dataset. We build the total load time series by taking the sum of the load profiles of all smart meters; once the total load has been computed, we randomly remove a percentage

of profiles to simulate a setting with partial grid coverage (see next subsections).

B. Experimental setup

We ran all experiments on an Intel(R) Xeon(R) CPU and on Nvidia Titan GPUs. Apart from a few that are specific to the two spatio-temporal GNNs, hyperparameters are mostly shared across all 3 considered models. In particular, their values have been chosen based either on published results or by tuning shared hyperparameters starting from the baseline.

The historic time window T and horizon H are both set to 1 day, which corresponds to 48 time steps. The dataset is first split into 3 windows of equal length, each corresponding to roughly 6 months of data; then, each window is further split into training, validation, and test subsets using a 70/10/20% split. For the encoders, we use a stack of 8 layers, with temporal convolutions with kernel size equal to 6, 16 output channels and a dilation factor $d^{(l)}$ given by

$$d^{(l)} = d_{base}^{l\%d_{mod}} \quad l = 0, \dots, L-1 \quad (20)$$

with $d_{base} = 2$ and $d_{mod} = 3$ to prevent $d^{(l)}$ from becoming too large. With this configuration, features produced by the fourth layer already have a receptive field encompassing the whole input length. The decoder, instead, is configured to have an encoder-decoder attentional layer using 16-dimensional queries and keys to compute attention weights and a 3 layered output FFNN with 64 hidden units per layer. Models are trained with batches of 32 samples using the Adam optimizer for 500 epochs with a Cosine Annealing learning rate scheduler that gradually reduces the learning rate η from its initial value of 0.001 down to 0.0001 by the end of training. For all models, we use a weight decay factor of 0.0001.

We aggregate load profiles of meters at the cluster level and use those as nodes for GGWN and GTGAN: $K = 25$ clusters are built by using the previously described clustering technique with $\sigma = 1$ and $K_{nn} = 10$ neighbors per node (smart meter) are considered for creating the k -nearest-neighbors graph. In particular, only 66% percent of the available meters are used to build these aggregates, as previously discussed. A diffusion convolutional operator of order 1 is employed for the GGWN model, while 16-dimensional projections are used for both the global pooling and the node attentional layer of GTGAN.

During training, the model is evaluated on the validation set after each epoch. Once training terminates, we reload the best-performing weights and evaluate the model on the test set with respect to the following metrics: the Mean Absolute Error (MAE), the Mean Absolute Percentage Error (MAPE), the Mean Squared Error (MSE) and the average MAE of the three best-performing sets of weights on the validation set (MAE₃).

C. Main results

In Tab. I we reported the main experimental results comparing the two proposed models against the TCN baseline on the CER-E dataset with respect to the four performance metrics used. For each model are reported the average errors

TABLE I
MAIN EXPERIMENTAL RESULTS, EACH VALUE IS THE AVERAGE ACROSS
15 RUNS WITH THE CORRESPONDING STANDARD DEVIATION.

Model	MAE	MAPE(%)	MSE(²)	MAE ₃
TCN	0.165±0.008	4.618±0.291	0.069±0.006	0.166±0.008
GGWN	0.158±0.004	4.361±0.134	0.065±0.003	0.160±0.004
GTGAN	0.160±0.005	4.393±0.167	0.066±0.003	0.161±0.005

TABLE II
ADDITIONAL EXPERIMENTS FOR THE GGWN MODEL.

Model	MAE	MAPE(%)	MSE(²)	MAE ₃
TCN	0.165±0.008	4.618±0.291	0.069±0.006	0.166±0.008
GGWN	0.158±0.004	4.361±0.134	0.065±0.003	0.160±0.004
GGWNadj	0.177±0.017	4.961±0.519	0.076±0.012	0.177±0.017
GGWNnograph	0.172±0.010	4.873±0.382	0.072±0.005	0.171±0.011
GGWNnoaux	0.170±0.012	4.709±0.432	0.075±0.008	0.170±0.011
GGWNnodes	0.179±0.010	5.028±0.316	0.076±0.006	0.180±0.009

across fifteen runs performed with different seeds and the corresponding standard deviations. For all metrics we used target values in MegaWatt-hour (MWh), unless specified otherwise. By looking at Tab. I, we can observe that both the GGWN and GTGAN spatio-temporal models show an improvement with respect to the TCN baseline, with error metrics that are between 3% and 7% lower than the corresponding values for the baseline. Overall, these results suggest that spatio-temporal GNNs can learn to exploit latent relationships among cluster-level aggregates from smart meters to improve the aggregate load forecasting accuracy with respect to models that only use the history of the total load.

D. Additional experiments and ablations

We carried out additional experiments and ablation studies to investigate the impact of the various components of GGWN and GTGAN. For these experiments, we used only 5 independent runs to estimate performance.

Results for additional experiments with GGWN are reported in Tab. II. GGWNadj denotes a modified version of GGWN with a prior adjacency matrix A , obtained by taking the k-nearest-neighbors w.r.t. the correntropy between nodes. GGWNnograph is an ablation where no diffusion convolution is present, and node features are only processed temporally. One could argue that only the information propagation from each node to the global time series is useful for our task, while diffusion among the nodes themselves has little impact. However, the results suggest that propagating information graph wise is actually beneficial. GGWNnoaux and GGWNnodes are two variation of the auxiliary training objective. The first refers to a GGWN model where only the main training objective is used, while, in the second, node representations are trained to predict node-level loads. In both these cases, the gradient is allowed to flow from ϕ^u to ϕ^v , as doing otherwise would not

TABLE III
A COMPARISON OVER THE NUMBER OF CLUSTERS USED FOR GGWN.

Clusters	MAE	MAPE(%)	MSE(²)	MAE ₃
5	0.161±0.007	4.365±0.159	0.068±0.006	0.163±0.005
25	0.158±0.004	4.361±0.134	0.065±0.003	0.160±0.004
50	0.168±0.009	4.662±0.329	0.070±0.006	0.168±0.010

TABLE IV
GTGAN ABLATIONS REGARDING THE AGGREGATION OF INCOMING
MESSAGES INSIDE THE ϕ^v MODULE.

Model	MAE	MAPE(%)	MSE(²)	MAE ₃
TCN	0.165±0.008	4.618±0.291	0.069±0.006	0.166±0.008
GTGAN	0.160±0.005	4.393±0.167	0.066±0.003	0.161±0.005
GTGANstat	0.162±0.004	4.466±0.154	0.066±0.003	0.161±0.004
GTGAndyn	0.166±0.011	4.620±0.377	0.069±0.008	0.166±0.010

make sense. Results suggest that using an additional objective is indeed beneficial and that our proposed auxiliary task is superior to the more straightforward alternative.

A comparison of performance while varying the number of considered clusters (nodes) is presented in Tab. III. From one side, a lower number of clusters corresponds to smoother signals, from the other, a higher number of clusters allows access to finer grained information. Results show that a balanced solution yields the best results.

Finally, we carry out an ablation study to investigate whether learning both static and dynamic spatial relationships between nodes is actually helpful. To this end, two versions of GTGAN are compared against the proposed architecture: GTGAndyn uses no trainable adjacency matrix to aggregate the incoming messages from other nodes, thus combining them based only on the dynamical attention coefficients, while GTGANstat, conversely, uses only static weights. From results reported in Tab. IV, we can see that removing either one of these two components is likely to result in lower performance.

VI. CONCLUSIONS

In this work, we proposed a method for capturing sparse spatio-temporal relationships between load profiles of smart meters and the total load with the objective of improving the accuracy of STLF. Results suggest that the additional relational information is indeed helpful for accurate predictions, and that spatio-temporal GNNs are an effective way to learn and exploit such relationships. Indeed, a core challenge when following the approach proposed here is extracting this relational information from the raw data while ignoring noise and redundancies in the inputs. Developing more advanced techniques for doing so could allow using finer-grained resolutions and further improve performance. In this regard, an interesting direction for future work would request to investigate end-to-end techniques for aggregating data from smart meters

instead of relying on pre-computed clusters. Finally, it is worth mentioning the proposed architectures were designed to be quite general, therefore it could be interesting to see if they can bring improvements in tasks different from total load forecasting.

REFERENCES

- [1] S. Abraham, "National transmission grid study," tech. rep., USDOE Office of the Secretary of Energy, Washington, DC (United States), 2003.
- [2] P. J. Burke, D. I. Stern, and S. B. Bruns, "The impact of electricity on economic development: A macroeconomic perspective," *International Review of Environmental and Resource Economics*, vol. 12, no. 1, pp. 85–127, 2018.
- [3] I. k. Nti, A. Adekoya, O. Nyarko-Boateng, and M. Teimah, "Electricity load forecasting: a systematic review," *Journal of Electrical Systems and Information Technology*, vol. 7, 09 2020.
- [4] H. Farhangi, "The path of the smart grid," *IEEE Power and Energy Magazine*, vol. 8, no. 1, pp. 18–28, 2010.
- [5] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart grid — the new and improved power grid: A survey," *IEEE Communications Surveys Tutorials*, vol. 14, no. 4, pp. 944–980, 2012.
- [6] J. Zheng, C. Xu, Z. Zhang, and X. Li, "Electric load forecasting in smart grids using long-short-term-memory based recurrent neural network," in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6, IEEE, 2017.
- [7] P. Lara-Benítez, M. Carranza-García, J. M. Luna-Romera, and J. C. Riquelme, "Temporal convolutional networks applied to energy-related time series forecasting," *applied sciences*, vol. 10, no. 7, p. 2322, 2020.
- [8] C. Tian, J. Ma, C. Zhang, and P. Zhan, "A deep neural network model for short-term load forecast based on long short-term memory network and convolutional neural network," *Energies*, vol. 11, no. 12, p. 3493, 2018.
- [9] W. Lin, D. Wu, and B. Boulet, "Spatial-temporal residential short-term load forecasting via graph neural networks," *IEEE Transactions on Smart Grid*, vol. 12, no. 6, pp. 5373–5384, 2021.
- [10] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph wavenet for deep spatial-temporal graph modeling," in *The 28th International Joint Conference on Artificial Intelligence (IJCAI)*, International Joint Conferences on Artificial Intelligence Organization, 2019.
- [11] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *International Conference on Learning Representations*, 2018.
- [12] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *arXiv*, 2018.
- [13] A. Cini, S. Lukovic, and C. Alippi, "Cluster-based aggregate load forecasting with deep neural networks," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2020.
- [14] K. Benidis, S. S. Rangapuram, V. Flunkert, B. Wang, D. Maddix, C. Turkmen, J. Gasthaus, M. Bohlke-Schneider, D. Salinas, L. Stella, *et al.*, "Neural forecasting: Introduction and literature overview," *arXiv preprint arXiv:2004.10240*, 2020.
- [15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [16] Y. Cheng, C. Xu, D. Mashima, V. L. Thing, and Y. Wu, "Powerlstm: power demand forecasting using long short-term memory neural network," in *International Conference on Advanced Data Mining and Applications*, pp. 727–740, Springer, 2017.
- [17] J. Bedi and D. Toshniwal, "Empirical mode decomposition based deep learning for electricity demand forecasting," *IEEE access*, vol. 6, pp. 49144–49156, 2018.
- [18] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.
- [19] Y. Wang, J. Chen, X. Chen, X. Zeng, Y. Kong, S. Sun, Y. Guo, and Y. Liu, "Short-term load forecasting for industrial customers based on tcn-lightgbm," *IEEE Transactions on Power Systems*, vol. 36, no. 3, pp. 1984–1997, 2020.
- [20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.
- [21] A. Gasparin, S. Lukovic, and C. Alippi, "Deep learning for time series forecasting: The electric load case," *CAAI Transactions on Intelligence Technology*, 2019.
- [22] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional time series forecasting with convolutional neural networks," *arXiv preprint arXiv:1703.04691*, 2017.
- [23] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.
- [24] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on lstm recurrent neural network," *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 841–851, 2017.
- [25] W. Kong, Z. Y. Dong, D. J. Hill, F. Luo, and Y. Xu, "Short-term residential load forecasting based on resident behaviour learning," *IEEE Transactions on Power Systems*, vol. 33, no. 1, pp. 1087–1088, 2017.
- [26] A. Tascikaraoglu, "Evaluation of spatio-temporal forecasting methods in various smart city applications," *Renewable and Sustainable Energy Reviews*, vol. 82, pp. 424–435, 2018.
- [27] A. Tascikaraoglu and B. M. Sanandaji, "Short-term residential electric load forecasting: A compressive spatio-temporal approach," *Energy and Buildings*, vol. 111, pp. 380–392, 2016.
- [28] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *International Conference on Neural Information Processing*, pp. 362–373, Springer, 2018.
- [29] C. Zheng, X. Fan, C. Wang, and J. Qi, "Gman: A graph multi-attention network for traffic prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 1234–1241, 2020.
- [30] A. Cini, I. Marisca, and C. Alippi, "Filling the gaps: Multivariate time series imputation by graph neural networks," in *International Conference on Learning Representations*, 2022.
- [31] W. Lin and D. Wu, "Residential electric load forecasting via attentive transfer of graph neural networks," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21 (Z.-H. Zhou, ed.)*, pp. 2716–2722, International Joint Conferences on Artificial Intelligence Organization, 8 2021.
- [32] S. R. Kuppannagari, Y. Fu, C. M. Chueng, and V. K. Prasanna, "Spatio-temporal missing data imputation for smart power grids," in *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, (New York, NY, USA), Association for Computing Machinery, 2021.
- [33] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.
- [34] W. Liu, P. P. Pokharel, and J. C. Principe, "Correntropy: Properties and applications in non-gaussian signal processing," *IEEE Transactions on signal processing*, vol. 55, no. 11, pp. 5286–5298, 2007.
- [35] M. Xu, W. Dai, C. Liu, X. Gao, W. Lin, G.-J. Qi, and H. Xiong, "Spatial-temporal transformer networks for traffic flow forecasting," *arXiv preprint arXiv:2001.02908*, 2020.
- [36] Commission for Energy Regulation, "Cer smart metering project - electricity customer behaviour trial, 2009-2010 [dataset]," 2012.