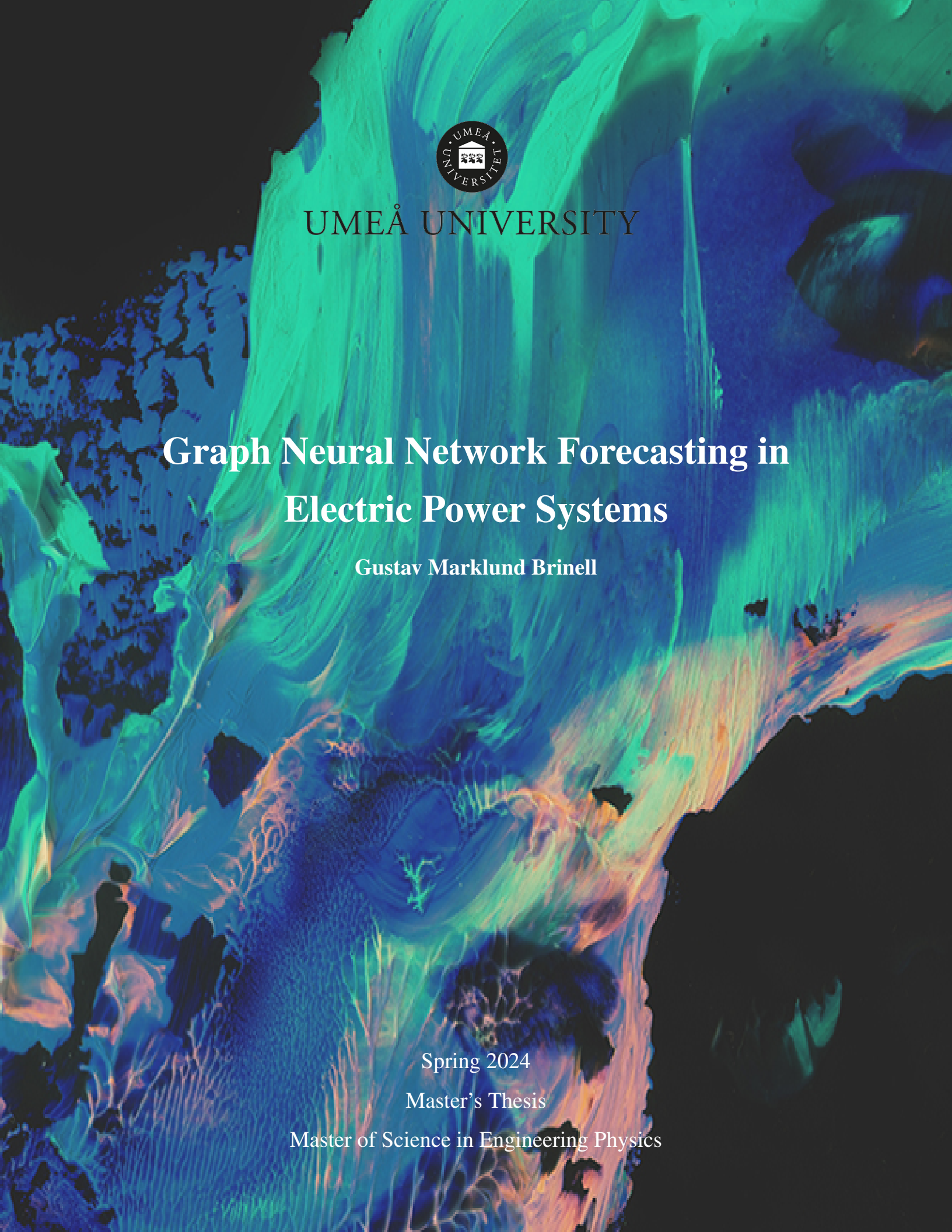


UMEÅ UNIVERSITY

Graph Neural Network Forecasting in Electric Power Systems

Gustav Marklund Brinell

Spring 2024

Master's Thesis

Master of Science in Engineering Physics

Graph Neural Network Forecasting in Electric Power Systems

Author:	Gustav Marklund Brinell	gustavmb@live.se
Supervisors:	Natallia Lundqvist	Svenska kraftnät
	Eva Krämer	Department of Physics
Examiner:	Jonas Westin	Department of Mathematics and Mathematical Statistics

Master's Thesis in Engineering Physics, 30 ECTS
Department of Physics
Umeå University

Abstract

This thesis explores the application of Graph Neural Networks (GNNs) for forecasting net-positions in the Nordic electricity market. Two GNN architectures, Gated Recurrent Unit Graph Convolutional Network (GRU-GCN) and Fourier Graph Neural Network (FGNN), were evaluated and compared to the existing forecasting model employed in the power grid. Results demonstrate that both GNN models achieve competitive performance, highlighting their potential for leveraging the graph structure inherent in power grids. However, regional variations in forecast uncertainty and the impact of data quality and disruptions necessitate further research. This thesis contributes to the understanding of GNNs in power grid forecasting and identifies future research directions, such as developing interpretable GNN models and incorporating additional data sources, to enhance the accuracy and reliability of power grid operations.

Sammanfattning

Denna avhandling undersöker tillämpningen av grafneurala nätverk (GNN) för att prognostisera nettopositioner inom den nordiska elmarknaden. Två GNN-arkitekturer, Gated Recurrent Unit Graph Convolutional Network (GRU-GCN) och Fourier Graph Neural Network (FGNN), utvärderades och jämfördes med den befintliga prognosmodellen i elnätet. Resultaten visar att båda GNN-modellerna uppnår konkurrenskraftig prestanda, vilket framhäver deras potential att utnyttja den grafiska struktur som finns i elnät. Regionala variationer i prognososäkerhet och påverkan av datakvalitet och störningar kräver dock ytterligare forskning. Denna avhandling bidrar till förståelsen av GNN inom elnätsprognostisering och identifierar framtida forskningsinriktningar, såsom att utveckla tolkningsbara GNN-modeller och inkorporera ytterligare datakällor, för att förbättra noggrannheten och tillförlitligheten i elnätsdriften.

Contents

Chapter 1

Introduction

Modern society is dependent on electricity which requires a stable power supply. With the green transition, the renewable power production has increased changing load patterns, which means that the condition for stable and safe operation of the system have changed. A consequence of this is that previous decision support systems for power system operation need to be supplemented with data-driven tools for forecasts of various kinds. These forecasts range from consumption in specific price ranges to power flows in individual components. The extensive data sets usually include technical information from the power system itself, as well as data from surrounding sources such as wind, solar radiation and temperature. To make these forecasting tools more efficient, it becomes essential to take advantage of prior knowledge of the underlying structures in the extensive data set. Within the electric power system, such a clear structure emerges in the form of stations and lines in the power grid. By using these underlying structures, one can potentially improve the precision and reliability of the forecasts.

Balancing electricity production and consumption is crucial for a stable power grid. Weather fluctuations affect both supply (e.g., wind, hydro) and demand (e.g., heating). Sweden is divided into four bidding areas where electricity is traded on a market. Prices in each area depend on local supply and demand, as well as limitations in transmitting power between regions. This system allows for efficient electricity flow from surplus areas in northern Sweden to areas with higher demand in the south [9]. Northern areas usually have a positive net-position (difference between electricity production and use) with excess electricity production, while southern areas have a negative net-position with higher consumption. The electricity market is designed according to the conditions in the physical power grid, i.e. how much can be transferred at a given time between the areas [4]. Therefore, an important task is to make forecasts in the electricity market to get an idea of the balance in the physical power grid.

1.1 Our approach

Traditionally, forecasts rely on unstructured tabular data that only considers temporal dependencies. This project explores the potential of using Graph Neural Networks (GNNs) for forecasting in the power grid. This approach captures both the temporal and spatial dependencies.

The goal of the project is to evaluate GNNs for power grid forecasting, where we will assess the potential and limitations of GNNs for forecasting net-positions in the Nordic electricity market. GNNs offer a unique advantage in their ability to handle the inherent network structure of the power grid, potentially leading to more accurate forecasts. We will develop GNN models and train them using real-world data to forecast hourly net-positions in each of the electricity areas. These areas are usually referred to as bidding zones, which we will now keep consistent throughout this thesis. The topological structure of these bidding zones can be seen in Figure 1, where Sweden's four bidding zones (SE1-SE4) are labeled in the figure.

1.2 Structure

We begin by establishing the fundamental theory of graph-structured data in Chapter 2. We then explore the essential principles of machine learning and neural networks, before diving into the specific theory of the GNNs utilized in this work. Chapter 3 transitions from theory to practice, presenting the results of numerical experiments where the proposed GNN models are compared and evaluated. Finally, Chapter 4 concludes the thesis by summarizing the findings, discussing potential limitations, and outlining possible avenues for future research.

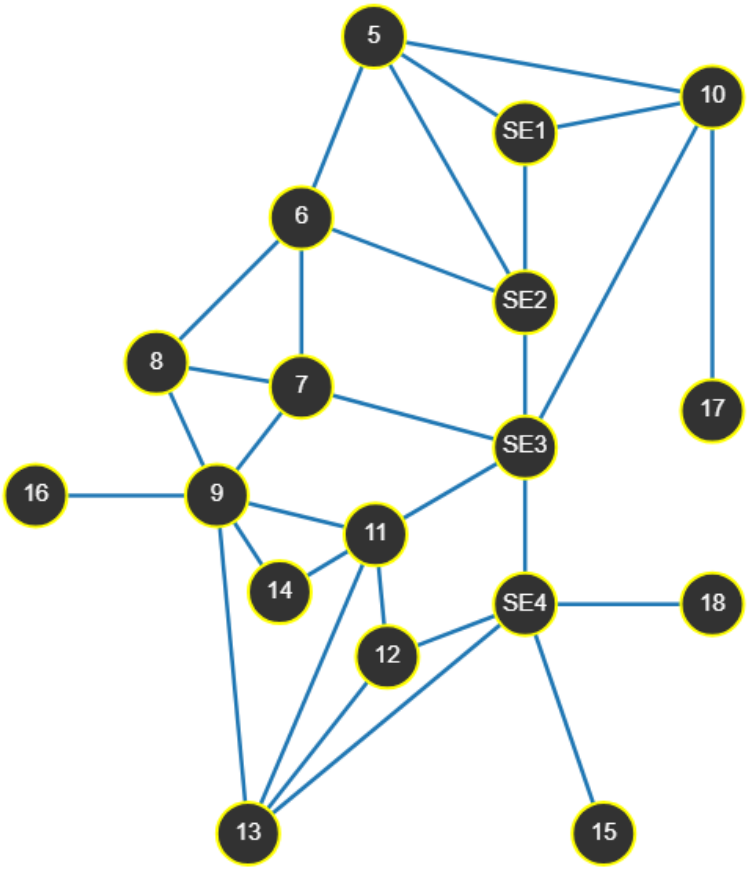


Figure 1: This figure depicts the Nordic bidding zones. Physical connections between zones are highlighted, forming a graphical structure. Each zone is represented by a node, and the connections between them represent edges. The following details the country affiliation of each zone: SE1 to SE4: Sweden; 5 to 9: Norway; 10: Finland; 11 and 14: Denmark; 12: Germany; 13: Netherlands; 15: Poland; 16: England; 17: Estonia; 18: Lithuania

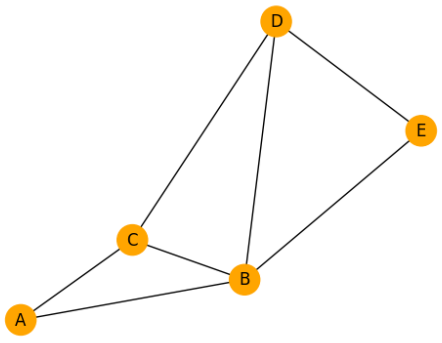
Chapter 2

Methodology and Theoretical Foundation

2.1 Definitions of Graph-Structured Data

The concept of graphs from graph theory is fundamental to understanding Graph Neural Networks. A graph in this context consists of two key elements: nodes (also called vertices) and edges. Nodes represent individual pieces of data, and edges represent the connections between them. We can represent a graph using the notation $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{G}$ is the entire graph, $\mathcal{V}$ is the set containing all the nodes, and $\mathcal{E}$ is the set containing all the edges [14].

The edges in $\mathcal{E}$ can be encoded into an adjacency matrix $\mathcal{A}$, describing the structure and connectivity in the graph. The adjacency matrix offers a compact way to represent a graph's connections using a square matrix. This matrix has dimensions $n \times n$, where n represents the number of nodes in the graph. Each cell (i, j) corresponds to the connection between node i and node j. A cell value of 1 signifies an edge exists between them, while a 0 indicates no connection. Notably, for graphs where edges have no direction (undirected graphs), the adjacency matrix is symmetrical [10]. An example of a graph and its corresponding adjacency matrix can be shown below in Figure 2.



(a) Graph

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

(b) Adjacency matrix

Figure 2: An undirected graph and its adjacency matrix

In addition to connections, each node in a graph can also hold its own information, called node features. These features can be numerical values representing properties of the data point associated with the node, categorical data indicating its type, or even text embeddings for textual data associated with the node. Node features capture the intrinsic characteristics of the data points represented by the nodes.

In the field of artificial intelligence, computers can be trained to learn and recognize patterns from data. These are called artificial neural networks, and they are powerful tools for various tasks. Traditional neural networks, like Convolutional Neural Networks, are a specific type of neural network that excels at analyzing data arranged in a fixed grid-like format, such as images [11]. However, they struggle to handle the flexible relationships between data points in graphs. This is where Graph Neural Networks come in. GNNs are specifically designed to process graphs. They leverage the graph structure encoded in the adjacency matrix and the node features to learn meaningful representations for both individual nodes and the entire graph itself. These learned representations are then used for various tasks on graphs, such as node classification, link prediction, and graph generation.

Before exploring Graph Neural Networks and their diverse architectures, we need basic understanding of machine learning and neural networks.

2.2 Machine Learning

This section covers the fundamentals of machine learning and is based on [6]. Machine learning is a field that focuses on developing algorithms and models that uses observed data to find patterns in the underlying structure that can be used to make predictions. A key distinction exists between classification and regression tasks. Classification algorithms predict discrete categories, like whether an email is spam or not. In contrast, regression algorithms predict continuous values, such as the price of a house. The goal of the machine learning model is to minimize a loss function, which measures the performance of the predicted target values versus the factual target values. The performance of the model is often assessed by the value of the loss function it achieves. Different tasks require different loss functions, but for regression, a few simple ones are Mean Squared Error (MSE) or Root Mean Squared Error (RMSE). Training the model involves optimizing this function through the machine learning algorithm's iterative adjustments of model parameters. These adjustable parameters are commonly known as trainable parameters. Machine learning algorithms aren't just about performing well on the training data they're given. The objective is to learn from that data and apply that knowledge to new situations. This means finding patterns in the training data that are likely to hold true even for data the model has never seen before. The model's ability to make accurate predictions on new inputs hinges on effective training. One common approach is to divide the data into three subsets. The first set, called the training set, is used to train the model. The model learns by finding a function that performs well on this data. In order to know how well this trained model perform on completely new data, we introduce the test set. This data is kept separate from the training set and used to evaluate the model's generalization ability, i.e. its performance on unseen data. Now, training a model often involves adjusting settings called hyperparameters. To find the best settings, we could just keep trying them out on the training set and see which one performs best. However, this would be misleading. The model might simply memorize the training data instead of learning true patterns. To avoid this bias, we introduce a third set: the validation set. This set is used to fine-tune the hyperparameters. We train the model with different hyperparameter settings on the training set, then evaluate its performance on the validation set. The hyperparameters that perform best on the validation set are then used with the final model evaluated on the completely unseen test set. The value of the loss function calculated for each respective set is also known as the training, validation, and test error.

There are two main challenges in machine learning that affect how well an algorithm performs: underfitting and overfitting. Underfitting happens when the model is too simple and can't capture the important patterns in the training data. As a result, the model performs poorly on both the training data (high training error) and unseen data (likely high test error). Overfitting occurs when the model becomes too complex and memorizes the specific details of the training data, including noise or irrelevant information. While the model might achieve a very low training error, it won't perform well on unseen data (large gap between training and test error). The ideal scenario is to find a balance between these two extremes. The model should be complex enough to learn the underlying patterns but not so complex that it memorizes the noise.

2.3 Neural Networks

In this section, based on [6], we present neural networks in its simplest form, as a Feed-Forward Neural Network. They are the foundation upon which more complex neural network architectures are built, and serve as a great introduction to how neural networks process information. In a Feed-Forward Neural Network, information travels in one direction only, forward from the input layer through hidden layers to the output layer. Each layer in the neural network is made up of individual neurons. They receive input from the previous layer, perform some computation, and then send their output to the next layer. The input layer receives the initial data, like the pixels of an image or numerical features. Hidden layers are used to help extract complex patterns from the data. One or more of these layers are used to process and transform the information from the previous layer. The output layer produces the final output of the network, like a classification (e.g., an object in an image) or a numerical value, as in the case of regression. During training, the network adjusts the connections between neurons in each layer. These connections are associated with weights that determine how strongly signals are transmitted. By adjusting the weights, the network learns to map the input data to the desired output.

If we consider the linear model, where it takes an input vector with d dimensions and transforms it into an output vector with m dimensions (the number of neurons in the layer), each hidden layer calculates

$$\mathbf{z} = \mathbf{W}^T \times \mathbf{x} + \mathbf{b},$$

where $\mathbf{z} \in \mathbb{R}^m$, $\mathbf{W} \in \mathbb{R}^{d \times m}$ is the weight matrix, $\mathbf{x} \in \mathbb{R}^d$ the input vector and $\mathbf{b} \in \mathbb{R}^m$ the bias

vector. It then passes the output through an activation function $\sigma(\mathbf{z})$ and then delivers the outcome to the output layer. The process is visualized in Figure 3. The activation function is used to introduce non-linearity and helps the network to learn complex patterns. The activation function is applied element-wise, meaning it's applied to each element of the vector $\mathbf{z}$ separately. Commonly used activation functions are the Rectified Linear Unit, defined as: ReLU(z) = $max(0, z)$, Tanh, and the Sigmoid function $\sigma(z) = \frac{1}{(1+e^{-z})}$. So the final output of the linear model will be

$$\mathbf{z} = \sigma(\mathbf{W}^T \times \mathbf{x} + \mathbf{b}). \tag{1}$$

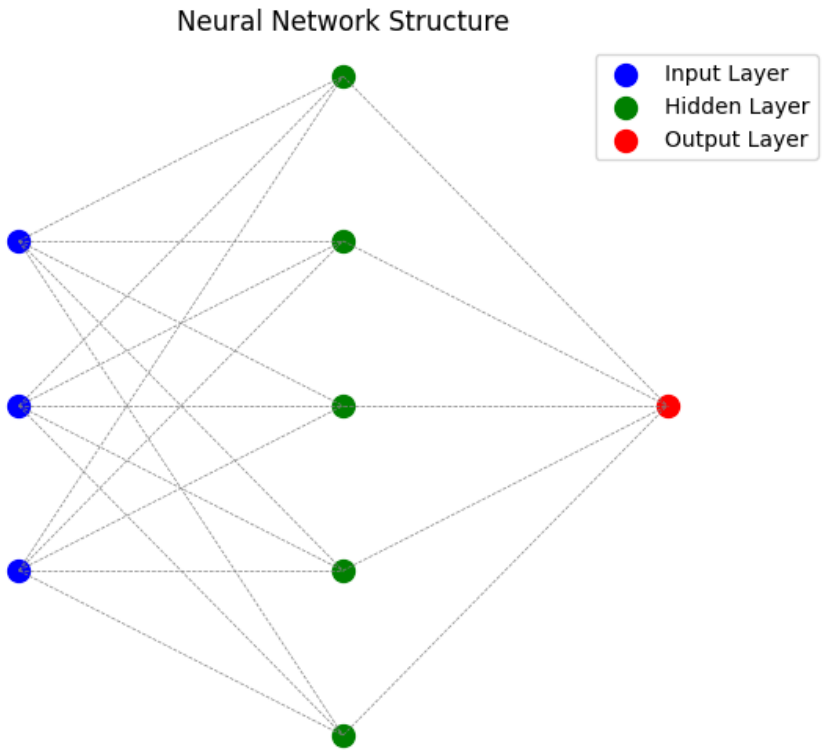


Figure 3: Structure of a Feed-Forward Neural Network, with a single output neuron.

2.4 Optimization

Different problems require different objective functions to be optimize during training. For minimization, functions like MSE or RMSE are commonly used to measures the difference between the network's predictions and the actual values. Backpropagation algorithms are used in order for the network to learn from its errors. It calculates the gradient of the objective function with respect to each weight and bias within the network. The gradient essentially provides information on the direction and magnitude in which changing a weight or bias will affect the error. After calculating the gradient using backpropagation, we can use it to optimize the parameters of the neural network. Gradient descent is an simple optimization algorithm that iteratively adjusts the network's weights and biases based on the calculated gradients. It follows the negative gradient, meaning it moves the weights in the direction that will most decrease the error. These gradients hold valuable information: a) Direction: They indicate whether increasing or decreasing a particular weight or bias will lead to a decrease in the error (loss). b) Magnitude: The magnitude of the gradient reflects how much of an adjustment should be made. Following these gradients, optimization algorithms like gradient descent iteratively update the network's parameters. This iterative process allows the network to learn from its errors and improve its performance on future predictions. However, basic gradient descent can be slow and prone to getting stuck in local minima. Hence different optimizers exists to alleviate this problem that are built upon gradient decent, such as Adam, RMSprop etc [13]. These optimizers incorporate additional factors like past gradients and momentum to navigate towards the minimum error more efficiently. A key consideration for optimization problems is the learning rate. This value controls the step size taken by the optimizer during weight updates. A very small learning rate can lead to slow convergence, while a large one can cause the optimizer to overshoot the minimum and oscillate around it.

2.5 Uncertainty Metrics

In machine learning, error metrics, or preferably known as uncertainty metrics, play a crucial role in assessing the performance of prediction models. These metrics compare the predicted values with the actual values to measure the accuracy of the model. For classification tasks, accuracy is a common uncertainty metric. It measures the proportion of times the model correctly assigns data points to their respective categories. Regression models don't have pre-defined classes like

in classification, hence predicting the exact true value is often impractical. Therefore, using accuracy wouldn't be effective even if the models predictions are close to the real values. To evaluate regression models more effectively, we use alternative metrics that capture different aspects of the prediction uncertainty. MSE, RMSE and Mean Absolute Error (MAE) are some popular regression metrics

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2,$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2},$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|Y_i - \hat{Y}_i|.$$

MSE measures the average squared difference between the predicted and actual values. While sensitive to outliers, it penalizes larger errors more heavily. RMSE is the square root of MSE, providing the uncertainty in the same units as the original data. Hence, using RMSE can be more intuitive. MAE calculates the average of the absolute differences between predicted and actual values, and is less sensitive to outliers compared to MSE [2].

2.6 Graph Neural Networks

This section delves into the core theory behind the Graph Neural Networks used in this work. The data in this work consists of multivariate time series data, so we'll need specific GNNs that can efficiently handle this. We'll explore two prominent GNN architectures: Graph Convolutional Neural Networks (GCNs) and Fourier Graph Neural Networks (FGNNs). This work builds upon the core concepts introduced in ([16], [15], [8]). For a deeper understanding of these concepts, we refer the reader to the original sources cited in this section.

2.6.1 GRU-GCN

The GRU-GCN model consists of Graph Convolutional Networks and Gated Recurrent Units (GRUs). The model uses GCNs to capture the spatial relationships between the grid structured network. This is because electrical power flow in one bidding zone can be influenced by flow in neighboring zones. By analyzing the network structure, the GCN learns complex topological features that represent these spatial dependencies. The GRU model [3] are a type of Recurrent Neural Network (RNN) proficient at handling sequential data. RNNs process data sequentially, one step at a time. At each step, they take the current input and combine it with the information retained in their memory from previous steps. This allows them to learn how the current data point relates to the past. While RNNs are strong contenders for modeling sequential data like in power grid forecasts, their long-term forecasting abilities can suffer from the vanishing gradient problem, where the influence of past data diminishes as the sequence lengthens [1]. To overcome this limitation, variants like Long Short-Term Memory (LSTM) [7] and GRU were developed [5]. GRU shows a great potential in this area due to its simpler architecture and faster training process. In the GRU-GCN model, GRUs are employed to capture the temporal dynamics of of the data. This means they can learn how patterns change over time, considering factors like daily or hourly variations.

The GRU-GCN model leverages the strengths of both Graph Convolutional Networks and Gated Recurrent Units to capture the inherent properties of the electric power grid. The model takes a sequence of historical time series data as input, and a GCN layer processes this data, exploiting the network's topological structure to extract spatial features. These time series, now enriched with spatial information, are then fed into the GRU layer. The GRU captures the dynamic temporal

changes within the network through information flow between its internal units. This two-step approach allows the GRU-GCN model to effectively learn both the spatial and temporal characteristics of the data.

Lets consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, ..., v_N\}$ is the set of N nodes and $\mathcal{E}$ the set of edges. The $N \times N$ adjacency matrix representing the connections between the nodes in the network is defined as

$$A_{i,j} = \begin{cases} 1 & \text{if } (v_i, v_j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } 1 \leq i, j \leq N.$$

Let $X \in \mathbb{R}^{N \times P}$ be the feature matrix, where P represents the number of node attribute features, i.e. the length of the historical time series. We now define l simple graph convolution layers as

$$f_l(X, A) = \sigma_l(A f_{l-1}(X, A) W_l), \tag{2}$$

where $f_0(X, A)$ is the input feature matrix X, W_l the weight matrix at layer l and A the adjacency matrix. While simple graph convolution layers effectively combine information from neighboring nodes, they suffer from two shortcomings. Firstly, they don't consider a nodes own features, only taking neighbouring node features into account. Secondly, the varying number of connections (node degrees) leads to features with inconsistent scales, making comparisons between nodes difficult. To address these limitations, the adjacency matrix is modified. By adding self-loops, $\tilde{A} = A + I_N$, the model incorporates a nodes own information. Additionally, normalization ensures features have a comparable scale regardless of a node's degree. This essentially allows the model to take an average of information from neighboring nodes. This normalization is done as proposed in [8], with $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, $\tilde{D} = \sum_j \tilde{A}_{i,j}$, where $\tilde{D}$ is the degree matrix. Equation 2 is now transformed to

$$f_l(X, A) = \sigma_l(\hat{A} f_{l-1}(X, A) W_l). \tag{3}$$

Now we have the modeling for the spatial dependence, we move on to the temporal dependence. In the GRU model, h_{t-1} represents the hidden state at the previous time step $t - 1$, and x_t signifies the current data information. The model employs two key gates: the reset gate r_t and the update gate u_t. The reset gate controls the degree to which past information h_{t-1} is discarded, while the update gate regulates how much of this past information is incorporated into the current state.

This interplay between gates determines the cell memory c_t, which essentially captures the long-term dependencies within the data. Finally, the output state h_t is produced based on the precessed information, containing both the spatial and temporal dependence of the data [16]. By effectively combining past information h_{t-1} with current data x_t through the reset and update gates r_t , u_t , GRU can capture both short-term trends and historical patterns. This mechanism allows GRU to excel at modeling the crucial temporal dependencies present in power grid datasets, making it a valuable tool for accurate forecasting.

The GRU-GCN model can now be constructed as below. The function $f(X_t, A)$ denotes the graph convolution process as defined in Equation 3. Here, W and b denote the weights and biases used during training, respectively. The symbol * represents element-wise multiplication.

$$
\begin{aligned}
u_t &= \sigma(W_u f(X_t, A) + W_u h_{t-1} + b_u), \\
r_t &= \sigma(W_r f(X_t, A) + W_r h_{t-1} + b_r), \\
c_t &= \tanh(W_c f(X_t, A) + W_c(r_t * h_{t-1}) + b_c), \\
h_t &= u_t * h_{t-1} + (1 - u_t) * c_t.
\end{aligned}
$$

The graph convolution extracts the inherent spatial relationships within the data, effectively understanding how different data points are interconnected. Simultaneously, the GRU unit analyzes the temporal dependencies by incorporating past information into the current state. This combined approach allows the model to gain a comprehensive understanding of the data, considering both the static relationships and the evolving trends over time. The overview of the GRU-GCN architecture is shown in Figure 4.

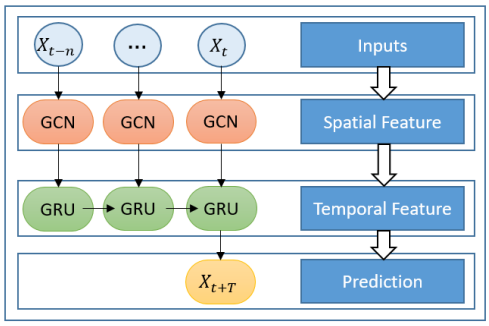


Figure 4: Architecture of the GRU-GCN model

2.6.2 Fourier Graph Neural Network

This section covers the theory about the FGNN used in this work. Now, instead of using a RNN to capture temporal dependence and a GNN for spatial dependence, the approach of FGNN is to capture the unified spatiotemporal dynamics of the data within a single framework. The concepts of FGNN is to define a hypervariate graph, where each node represents a single value from a time series regardless of variable or timestamp. This approach unifies the spatial and temporal aspects of the data into a single graph structure. Now the task becomes predictions on the hypervariate graph. The idea to efficiently do this is by leveraging what the authors call Fourier Graph Operators (FGOs) [15]. This operator is a replacement of classic graph operations, like convolutions, which performs matrix multiplications in Fourier space. In this way we both capture the inter-series dependencies, and reduce the complexity of the operations performed [15].

We first consider the hypervariate graph, $\mathcal{G}_t = (X_t^{\mathcal{G}}, A_t^{\mathcal{G}})$. We start with the input multivariate time series $X_t \in \mathbb{R}^{N \times T}$ with N variables observed at time t. We construct the hypervariate graph of NT nodes by regarding each element of X_t as one node of $\mathcal{G}_t$, resulting in the node features $X_t^{\mathcal{G}} \in \mathbb{R}^{NT \times 1}$. The adjacency matrix, $A_t^{\mathcal{G}} \in \mathbb{R}^{NT \times NT}$, is a binary matrix. The value at a position can indicate the presence or absence of an edge, the strength/weight of the connection, or even additional information about the relationship between the nodes. A value of 1 typically signifies a connection exists between nodes i and j, while 0 indicates no connection. In weighted graphs, the value represents the strength or importance of the connection. Higher values indicate stronger connections. In some cases, the matrix might contain additional information beyond just presence/absence, like the type of relationship or a distance measure. Even some datasets could be represented by a fully-connected graph when every node is connected to every other node. Through training, the network identifies the relevant connections within the graph structure. This allows the graph to encode three key types of dependencies: Intra-series temporal dependencies: This refers to the connections between nodes representing the same variable at different time steps, capturing how the values of a single variable evolve over time. Inter-series spatial dependencies: These connections exist between nodes representing different variables at the same time step, reflecting the relationships between variables at a specific point in time. Time-varying spatiotemporal dependencies: These connections capture the dynamic relationships between variables across different time steps. By considering all these connections within a single graph structure, the model can effectively learn the complex interplay between space and time within the data.

Now, we consider the Fourier Graph Operator $\mathcal{S} \in \mathbb{C}^{d\times d}$, which satisfies $\mathcal{F}(X)\mathcal{S}_{A,W} = \mathcal{F}(AXW)$, where $\mathcal{F}$ denotes Discrete Fourier Transform (DFT). This operation performs convolutions in Fourier space. An interesting aspect of this approach is that multiplying the transformed data $\mathcal{F}(X)$ with the FGO $\mathcal{S}$ in the Fourier space corresponds to performing a graph convolution operation in the original time domain [15]. This is significant because multiplications in the frequency domain have a computational complexity of $\mathcal{O}(n)$, where n is the number of elements. In contrast, performing the equivalent graph convolution directly in the time domain requires operations with a complexity of $\mathcal{O}(n^2)$. This difference in complexity motivates the development of a GNN by leveraging convolutions in Fourier space.

FGNN for multivariate time series forecasting can now be presented. Given the input data $X_t \in \mathbb{R}^{N\times T}$, we first construct a fully-connected hypervariate graph $\mathcal{G}_t = (X_t^{\mathcal{G}}, A_t^{\mathcal{G}})$, then project $X_t^{\mathcal{G}} \in \mathbb{R}^{NT\times 1}$ into node embeddings $\mathbf{X}_t^{\mathcal{G}} \in \mathbb{R}^{NT\times d}$ using the embedding matrix $E \in \mathbb{R}^{1\times d}$, where d is the embedding dimension.

To capture the spatiotemporal dependencies simultaneously, we feed embedded hypervariate graphs with $\mathbf{X}_t^{\mathcal{G}}$ to FGNN. We begin to perform DFT on each discrete spatio-temporal dimension of the embeddings $\mathbf{X}_t^{\mathcal{G}}$ and obtain the frequency output $\mathcal{X}_t^{\mathcal{G}} := \mathcal{F}(\mathbf{X}_t^{\mathcal{G}}) \in \mathbb{C}^{NT\times d}$. Next, we perform a recursive multiplication between $\mathcal{X}_t^{\mathcal{G}}$ and FGOs $\mathcal{S}_{0:k}$ in Fourier space, yielding the resulting representations $\mathcal{Y}_t^{\mathcal{G}}$ as follows:

$$\mathcal{Y}_t^{\mathcal{G}} = \sum_{k=0}^{K} \sigma(\mathcal{F}(\mathbf{X}_t^{\mathcal{G}}))\mathcal{S}_{0:k} + b_k),$$

where $\mathcal{S}_{0:k} = \prod_{i=0}^{K} \mathcal{S}_i$, K is the number of layers, $\mathcal{S}_k$ is the FGO in the k-th layer, $b_k \in \mathbb{C}^d$ the bias parameters and σ nonlinear activation function. Since $\mathcal{Y}_t^{\mathcal{G}}$ is in Fourier space, we transform it back to the time domain using Inverse Discrete Fourier Transform (IDFT) $\mathcal{F}^{-1}$, as $\mathbf{Y}_t^{\mathcal{G}} = \mathcal{F}^{-1}(\mathcal{Y}_t^{\mathcal{G}}) \in \mathbb{R}^{NT\times d}$. Then, the output from the FGNN is used in a Feed-Forward Neural Network (as in Equation 1) to obtain the final predictions $\hat{Y}_t = \text{FNN}(\mathcal{Y}_t^{\mathcal{G}})$. A schematic representation of the entire model can be illustrated in Figure 5.

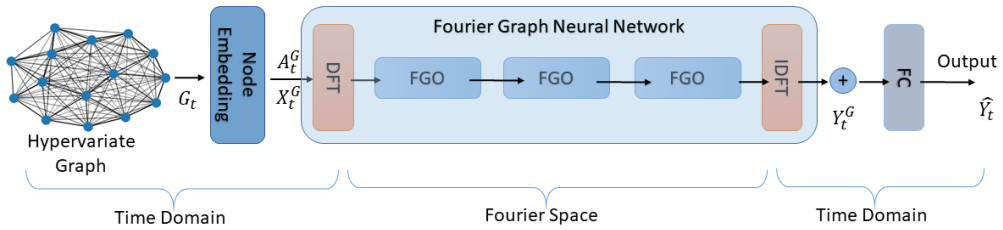


Figure 5: Architecture of the FGNN model. We take the hypervariate graph and embed the node features. We then feed those embeddings to FGNN and perform DFT. The network then performs a series of recursive multiplications and summations to capture relationships between nodes based on the graph structure. The output of the message passing is then transformed back into the time domain using the IDFT. Finally, the output is fed into fully-connected layers to generate predictions of our desired dimension

The overall time complexity of the FGNN is $\mathcal{O}(nd\log n + Knd^2)$, which includes the DFT, IDFT and the multiplications of FGOs $\mathcal{F}(X)\mathcal{S}$. This, compared to the time complexity of the same operations in the time domain, AXW, is $\mathcal{O}(n^2d + nd^2)$, makes FGNN more efficient [15].

Chapter 3

Numerical Experiments

So far, we have only discussed the theoretical and methodological foundations of this work. Now we'll introduce our dataset. In this chapter, we will go over a few experiments with the GRU-GCN and FGNN models, to evaluate how well they perform on our dataset. Since this work is done in collaboration with Svenska kraftnät (Svk), we will also compare these models to one currently used model at Svk that performs these kinds of forecasts. We will refer to that model as ModelX. Due to confidentiality, ModelX will remain unknown. The experiments in this section are conducted in Python using PyTorch 2.0 CUDA 11.7 [12].

The dataset contains the net-positions from all 18 bidding zones introduced in Figure 1. We give an overview over the net-positions of SE1-SE4 from 2018-2024 in Figure 6. Generally, northern Sweden (SE1, SE2) has the repeating pattern of a positive net-position, while southern Sweden (SE3, SE4) has a negative net-position. Furthermore, the net-position is measured in megawatts (MW).

Figure 6: Plot of Swedens net-positions

The dataset is a multivariate time series, comprising 54144 observations across 18 variables, with each variable representing a node. Further details about the data are shown in Table 1.

Table 1: Description of the dataset

Dataset	Time Span	#Nodes	#Edges	#Samples	Sampling Rate	Data Range	Median
Net-positions	01-01-2018/06-03-2024	18	30	54144	Hourly	-6853 MW ~7054 MW	-30 MW

3.1 Experiments GRU-GCN

We will begin to train our GRU-GCN model, and evaluate its performance on the validation set. We split the data into 70% training, 20% validation and 10% test. This means the training set includes the first 37901 observations for all 18 nodes, the validation set includes the next 10829 observations, and the test set consists of the final 5414 observations. We perform hyperparameter-tuning on the validation set, with the parameters as shown in Table 2. The batch size refers to the number of training examples utilized in one iteration, and it's a hyperparameter that determines the number of samples to work through before updating the model parameters during training. We train the model for 100 epochs, and prune trails with an early stopping strategy with the patience of 10 epochs. That means, trails that has not improved within 10 epochs are pruned. The objective is to minimize the RMSE, so lower RMSE indicates better performance. The result of the parameter-tuning can be shown in Figure 7.

Table 2: Parameters for tuning GRU-GCN

Parameter	**Value**
Hidden units	32, 48, 64
Batch size	32, 48, 64
Learning rate	0.0001, 0.001, 0.005
Input length	12, 24, 48

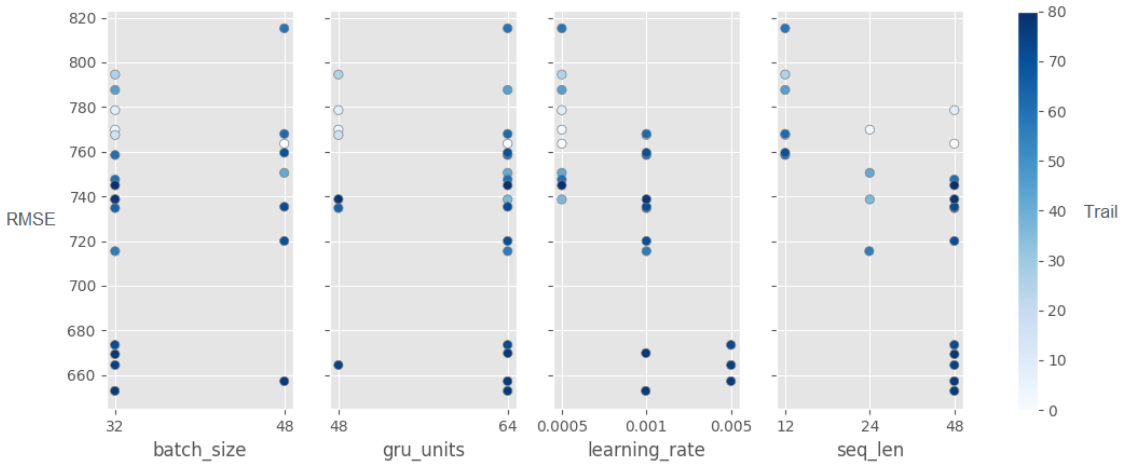
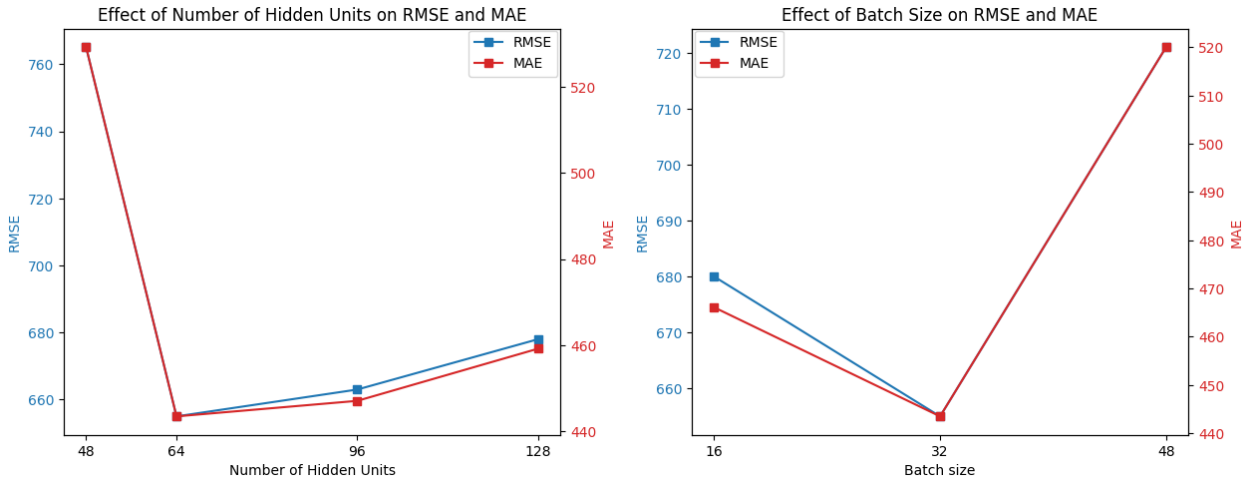


Figure 7: Plot of the tuned parameters. The objective value on the y-axis is the RMSE, and on the x-axis the tuneable parameters. Each trail is a combination of parameters, and the intensity in color to distinguish between different trials

The overall best configurations was with a batch size of 32, hidden units of 64, learning rate of 0.001 and input of 48. We can also see that this configuration was the best for each input length. It's worth noting that all trails where the batch size was equal or greater then the hidden units got all pruned. And a learning rate of 0.005 resulted in a lot of trails not completing, and increasing it even further introduces overfitting and very unstable trails. A lot of trails finished with hidden units of 64 and/or a batch size of 32, and since 64 and 32 are on the end tail of the spectrum, we therefore performed a follow up experiment with more/less units. We now test the best configuration for more hidden units, and less batch size to see which model performs best. As we can see in Figure 8, 64 hidden units and 32 batch size resulted in lowest RMSE and MAE.



(a) **(b)**

Figure 8: Effects on RMSE and MAE for different number of hidden units **(a)** and batch size **(b)**

Experimenting with different prediction lengths will also greatly impact the result. Forecasting a few hour ahead is easier then forecasting a few days ahead. In Figure 9 we can see how the RMSE and MAE is greatly affected by a longer prediction horizon.

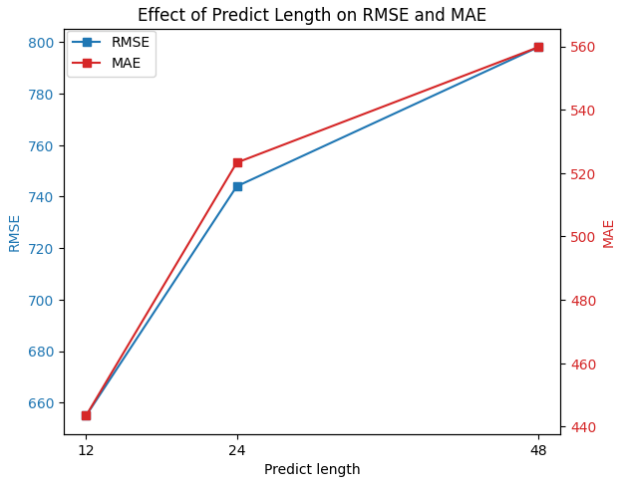


Figure 9: RMSE and MAE for different prediction lengths

We will now test our final model and forecast the net-positions with the best configuration and evaluate the results on the test set. This is done with the batch size of 32, hidden units of 64, learning rate of 0.001, input length of 48 hours and prediction length of 12 hours. This implies that the model uses 48 observations as input to predict the next 12, after which it moves the window until the entire test set is forecasted. To visualize the results, we plot the predicted versus the true net-position of SE3, as can be seen in Figure 10. We can see that the model seems to capture the overall trend in the data, but also over- and undershoots. To further investigate our assumption, we plot one weeks worth of data (see Figure 11) and see how the model performs. The model is successful in following the patterns, but do indeed have a tendency to overshoot and occasionally lag behind. Instead of visualizing each bidding zone at a time, we can plot the RMSE and MAE for all bidding zones and see how the forecast are affected by each zone. In Figure 12, we see that there are a few bidding zones that largely contributes to the increase in uncertainty in the model.

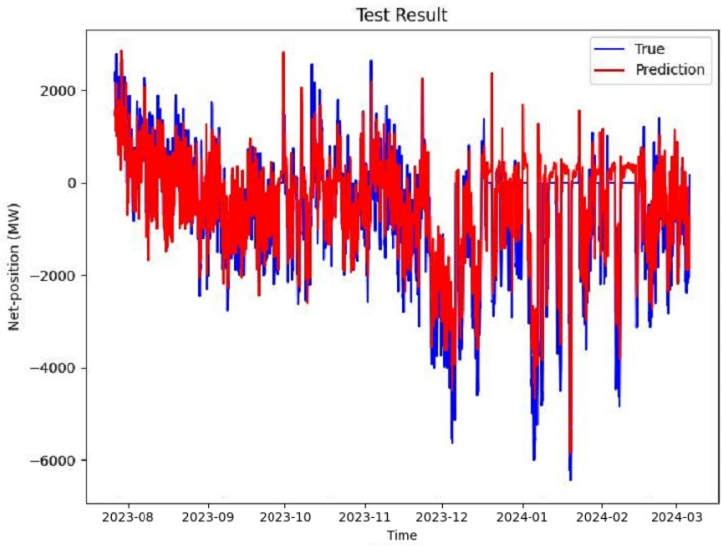


Figure 10: Plot of SE3 for all test data, with GRU-GCN as the forecasting model. Net-position versus time with one hour resolution

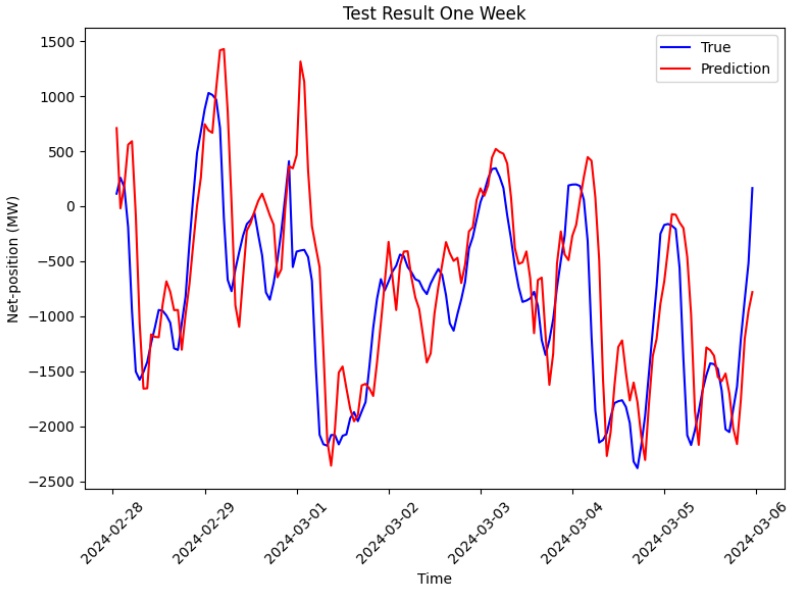


Figure 11: Plot of SE3 for the last week in the test data, with GRU-GCN as the forecasting model. Net-position versus time with one hour resolution

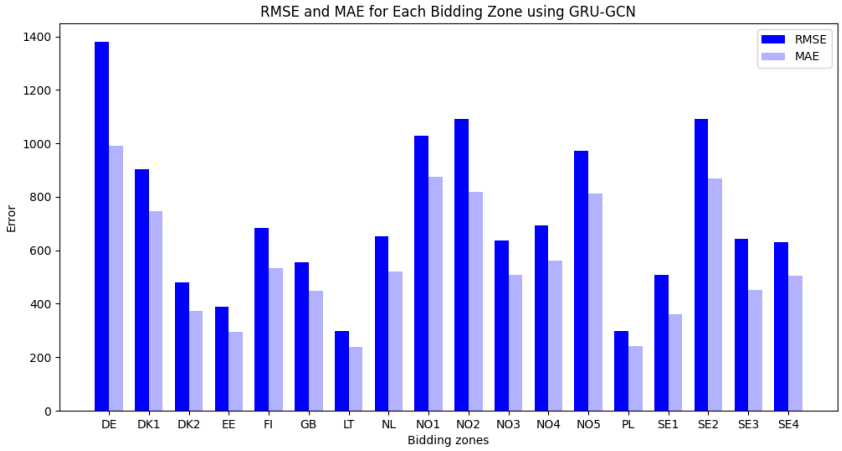


Figure 12: Plot of RMSE and MAE for all bidding zones using GRU-GCN

3.2 Experiments with FGNN

We have the same experimental setup as before, with 70% training, 20% validation and 10% test data. We also train the model for 100 epochs and perform hyperparameter-tuning on the validation set, with an early stopping strategy with the patience of 10 epochs. We tune the parameters as shown in Table 3. The result of the parameter-tuning can be visualized in Figure 13.

Table 3: Parameters for tuning FGNN

Parameter	Value
Hidden units	64, 96, 128
Batch size	16, 32, 64
Embedding size	64, 96, 128
Learning rate	0.00001, 0.0005, 0.0001
Input length	12, 24, 48

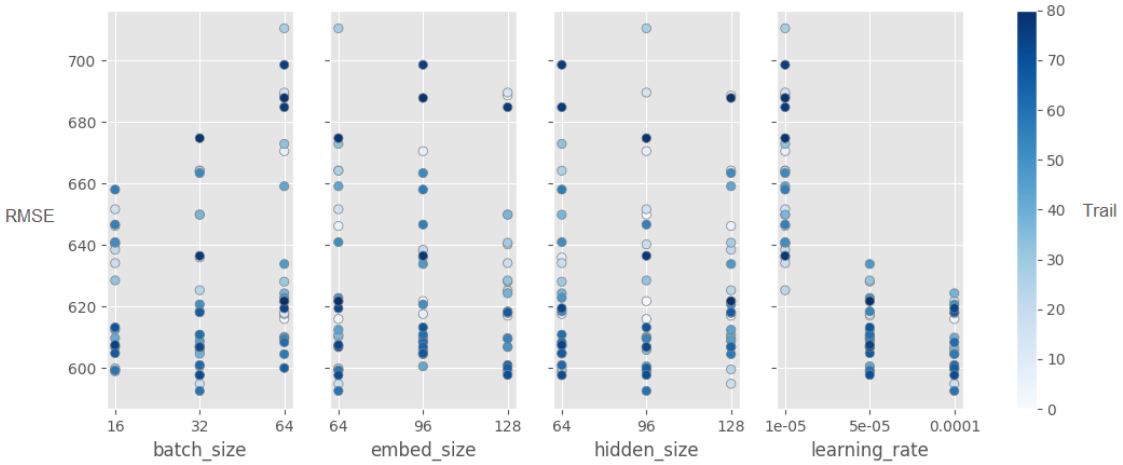


Figure 13: Plot of the tuned parameters. The objective value on the y-axis is the RMSE, and on the x-axis the tuneable parameters. Each trail is a combination of parameters, and the intensity in color to distinguish between different trials

The trail with the best configuration was with a batch size of 32, embedding size of 64, hidden size of 96 and learning rate of 0.0001. We also test for fewer embedding size, different input lengths and predict lengths, which can be shown in Figure 14.

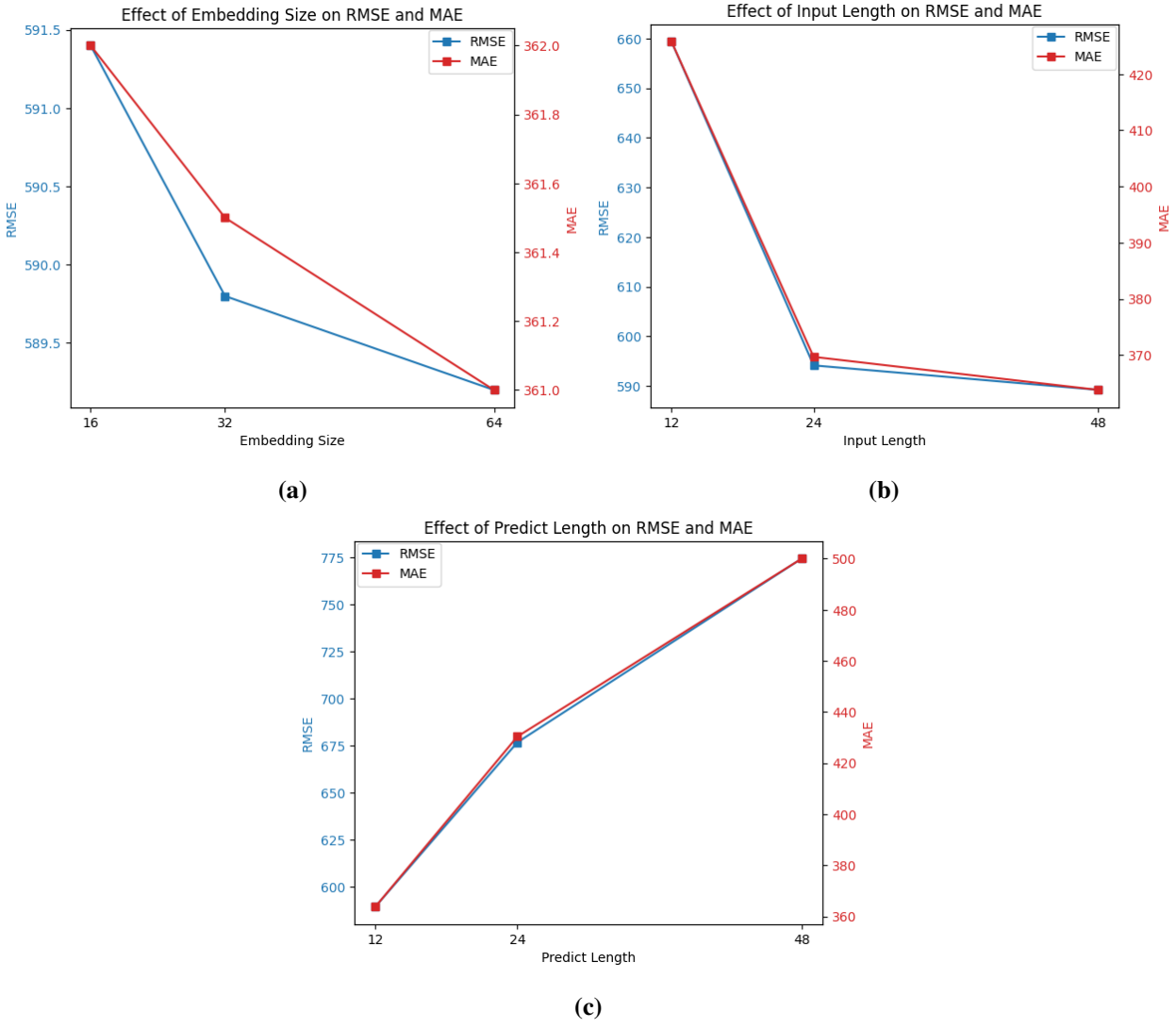


Figure 14: Effects on RMSE and MAE for different embedding size **(a)**, input length **(b)**, and prediction length **(c)**

We will now proceed to assess the performance of our finalized model using the best configuration. This is done with the batch size of 32, embedding size of 64, hidden size of 96, learning rate of 0.0001, input length of 48 hours and prediction length of 12 hours. For comparison purposes, we replicate the previous visualizations. Figure 15 depicts the predicted net-position against the true SE3 values. Furthermore, Figure 16 illustrates the model's performance on one week's worth of data. Both visualizations suggest the model effectively captures the underlying trend of the data, demonstrating good agreement with ground truth data.

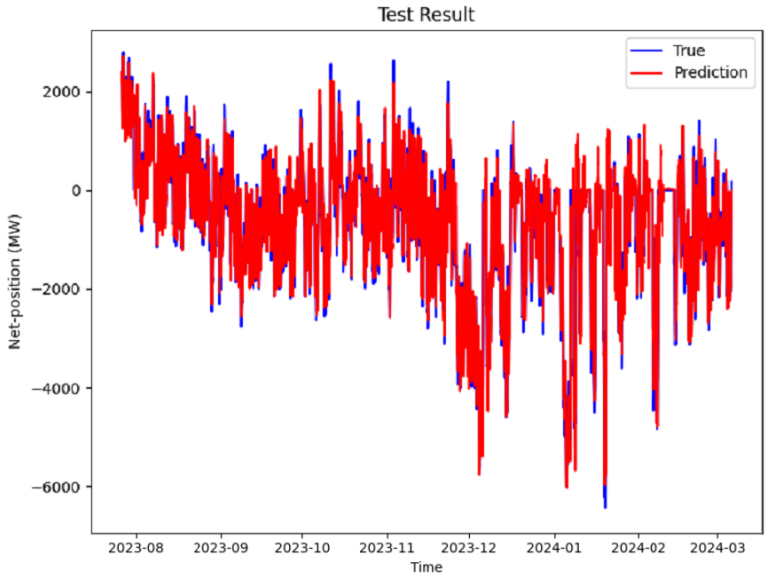


Figure 15: Plot of SE3 for all test data, with FGNN as the forecasting model. Net-position versus time with one hour resolution

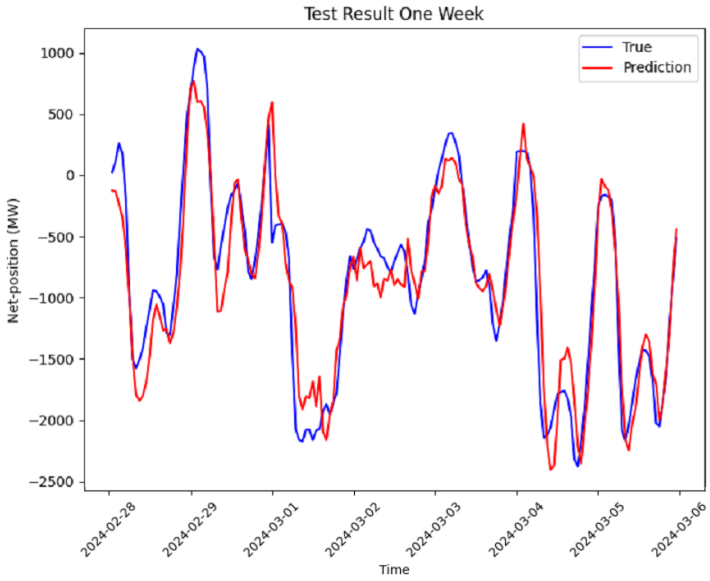


Figure 16: Plot of SE3 for the last week in the test data, with FGNN as the forecasting model. Net-position versus time with one hour resolution

We also visualize the RMSE and MAE across all zones simultaneously in Figure 17. As depicted in Figure 17, a few bidding zones appear to contribute significantly to the increased model uncertainty. Consistent with Figure 12, we see the same bidding zones contributing to an increased uncertainty. While still significant, their influence is noticeably lower compared to the GRU-GCN model.

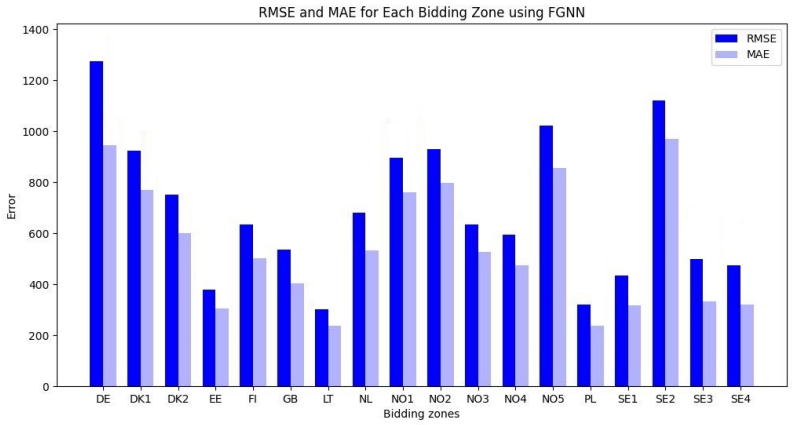


Figure 17: Plot of RMSE and MAE for all bidding zones using FGNN

3.3 ModelX

We now compare the performance of our proposed models with ModelX, the currently used model at Svk. ModelX is evaluated on the same underlying data as GRU-GCN and FGNN. As in the previous evaluations, we visualize the predicted net-position of SE3 for both the entire test duration (see Figure 18) and a one-week time-frame (see Figure 19). A key observation from Figure 18 is that the model occasionally exhibits significant overshoots. However, when observing shorter time-frames, Figure 19 demonstrates that the model retains the ability to capture underlying patterns in the data.

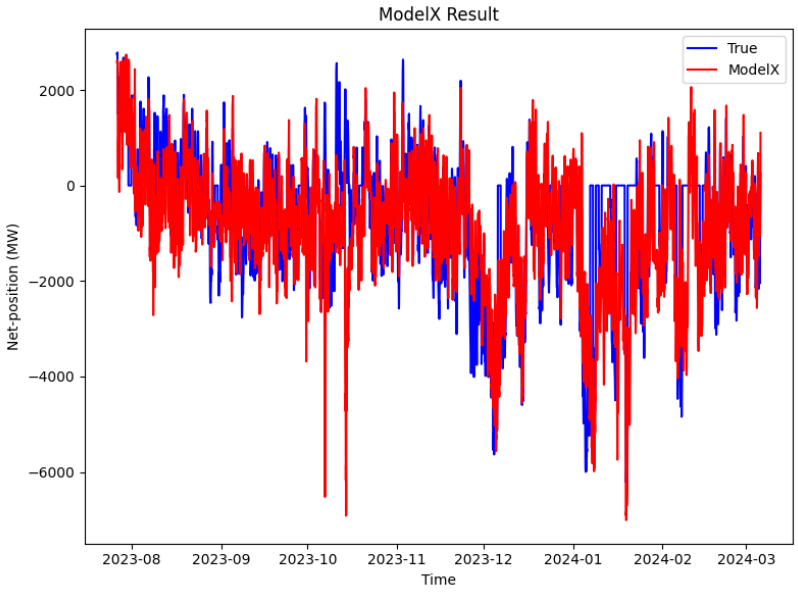


Figure 18: Plot of SE3 for all test data, with ModelX as the forecasting model. Net-position versus time with one hour resolution

To further investigate the source of uncertainty, Figure 20 identifies the bidding zones that significantly contribute to the model's overall uncertainty. Consistent with prior observations, a few number of bidding zones appear to be the primary drivers. It's noteworthy that, as evidenced in Figure 20, the severe overshoots exhibited by ModelX significantly inflate its overall performance. Interestingly, a comparison of the remaining bidding zones suggests that ModelX may outperform, or at least compete with both the GRU-GCN and the FGNN models in these specific zones.

Figure 19: Plot of SE3 for the last week in the test data, with ModelX as the forecasting model. Net-position versus time with one hour resolution

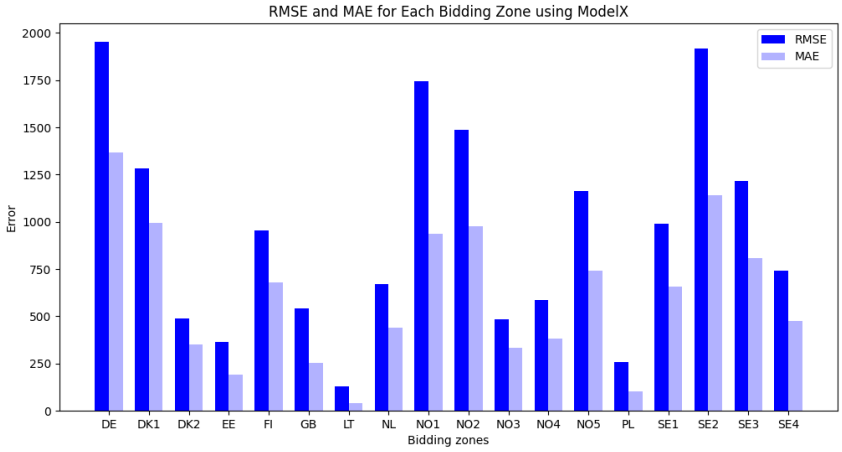


Figure 20: Plot of RMSE and MAE for all bidding zones using ModelX

Figure 21 presents a comparative visualization of all three models alongside the true values. This facilitates a side-by-side evaluation of each models performance across different bidding zones at a given time.

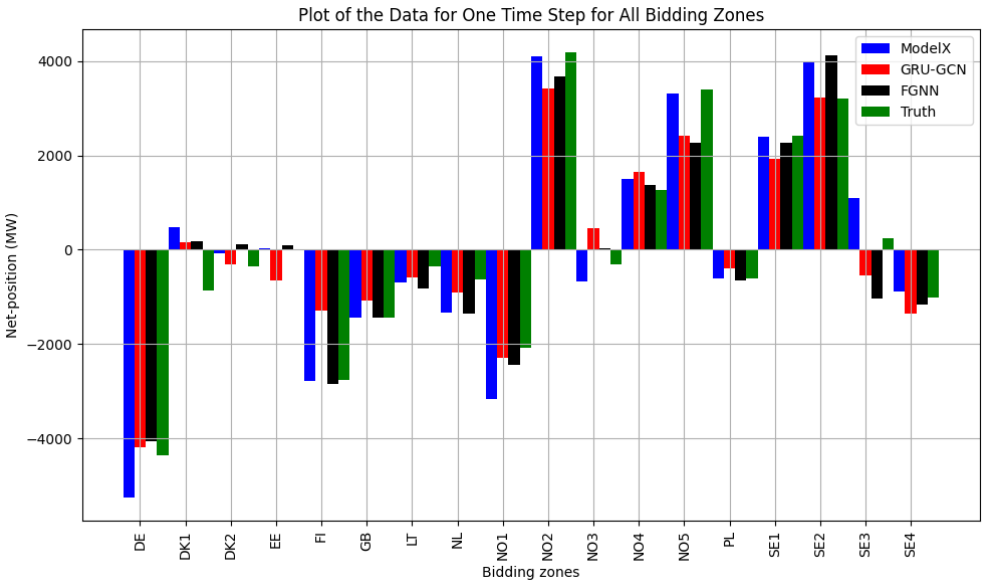


Figure 21: Comparison between the models for the final time step

One thing that might be a big contributor to the increased uncertainty in the models is the impurity of the dataset. A few bidding zones had abnormally large amount of zeros as value or missing data, compared to the other bidding zones. Missing data points are particularly common in time series data due to sensor malfunctions, outages, or irregular recording intervals. It's important to note that the reasons for missing data are complex and might involve a combination of these factors. Understanding the cause of missing data is crucial for choosing the most appropriate technique to handle it and avoid introducing bias into the model. To investigate this, we kept the noisy data as it was, only filling the missing values with zeros (as a added perturbation), indicating no exchange flow between neighboring zones and zone of interest. A few bidding zones had very many zeros in the training data, and very few or none in validation and test data, and a few bidding zones had very few or none in the training data, but vastly more in the validation or test data. Having a net-position of zero can also indicate that there is a balance in the system. However,

observing Figure 22, which included a lot of zeros and had a larger test metric, we can see drastic jumps, going from -5000 MW at one instance and to zero the next. This is not possible in the physical grid. As observed, missing values can greatly impact the accuracy and performance of the models, especially when the length of the perturbation exceeds the input length of the model. First of all, missing values limit the information available to the model, hindering its ability to learn underlying patterns and trends. Second, depending on the missing data pattern (random vs. systematic), the model might learn biased relationships, leading to inaccurate predictions. Several techniques can be employed to address these missing values in time series, for example, simple imputation. This involves replacing missing values with a basic statistic like mean, median, or the previous observation. For a more comprehensive comparison, imputing missing values with the previous non-zero value might be worth exploring. This approach could potentially improve the performance of all models, particularly for capturing these disruptive events. The choice of this imputation technique is based on its simplicity while preserving patterns and trends in the data. Consider a time series that oscillates evenly around a certain value, with the mean close to that value. If there's a missing value at a peak, imputing it using the mean would create a jump, disrupting the oscillating pattern. To avoid this, imputing with the previous non-zero value ensures a smoother transition and preserves the underlying trend.

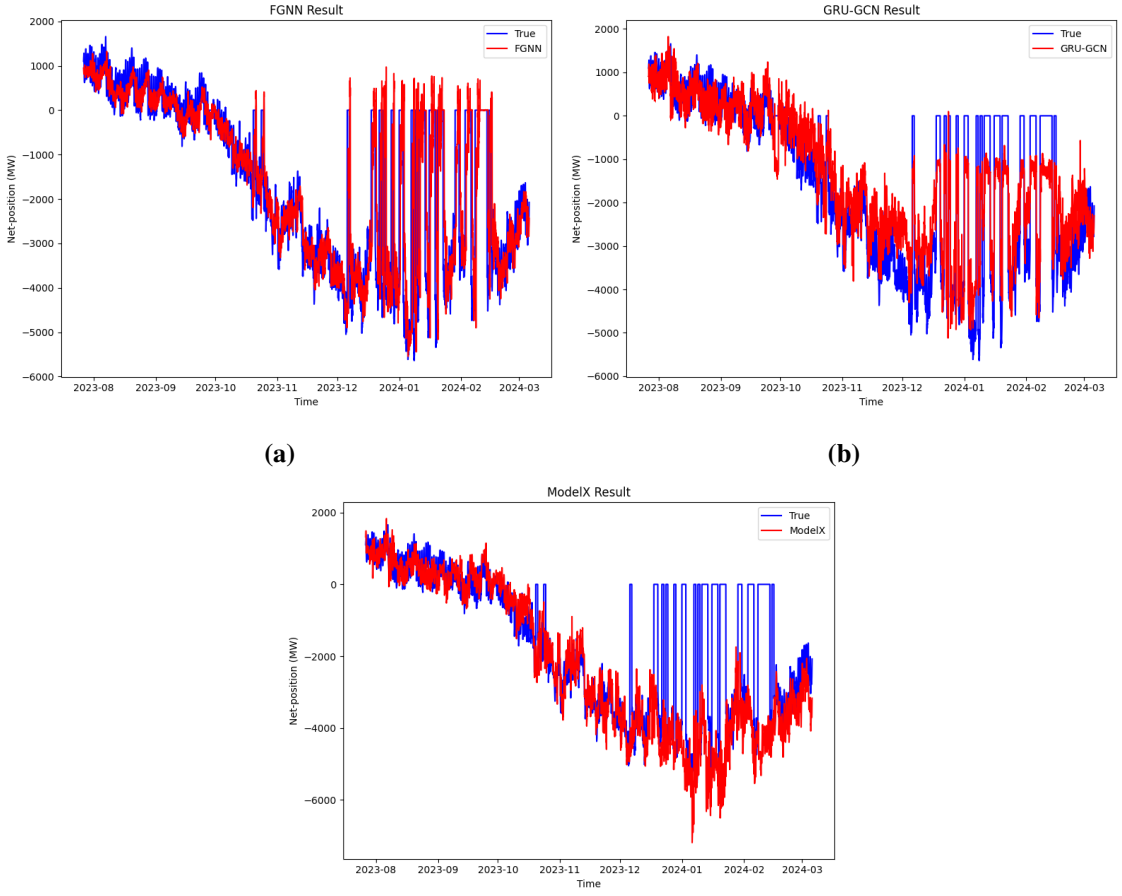


Figure 22: Plot of the net-positions of NO1 over time. **(a)** Using the forecasting model FGNN, **(b)** using GRU-GCN, and **(c)** using ModelX

To further explore the impact of missing data imputation on model performance, we conducted a follow-up experiment. We maintain the same setup as before for the models with a prediction length of 12 hours, but now we have replaced all missing values with the previous non-zero observation (illustrated in Figure 23). As anticipated, all models exhibited performance improvements. However, the models (FGNN and GRU-GCN) that previously captured some level of disruption in the data benefited less significantly from this imputation compared to ModelX. ModelX however, which previously disregarded these disruptions, benefited considerably from this imputation technique. Its performance improved notably, and a more comprehensive comparison between the models uncertainty can now be shown in Figure 24, where we visualize the RMSE and MAE for each bidding zone in the test set.

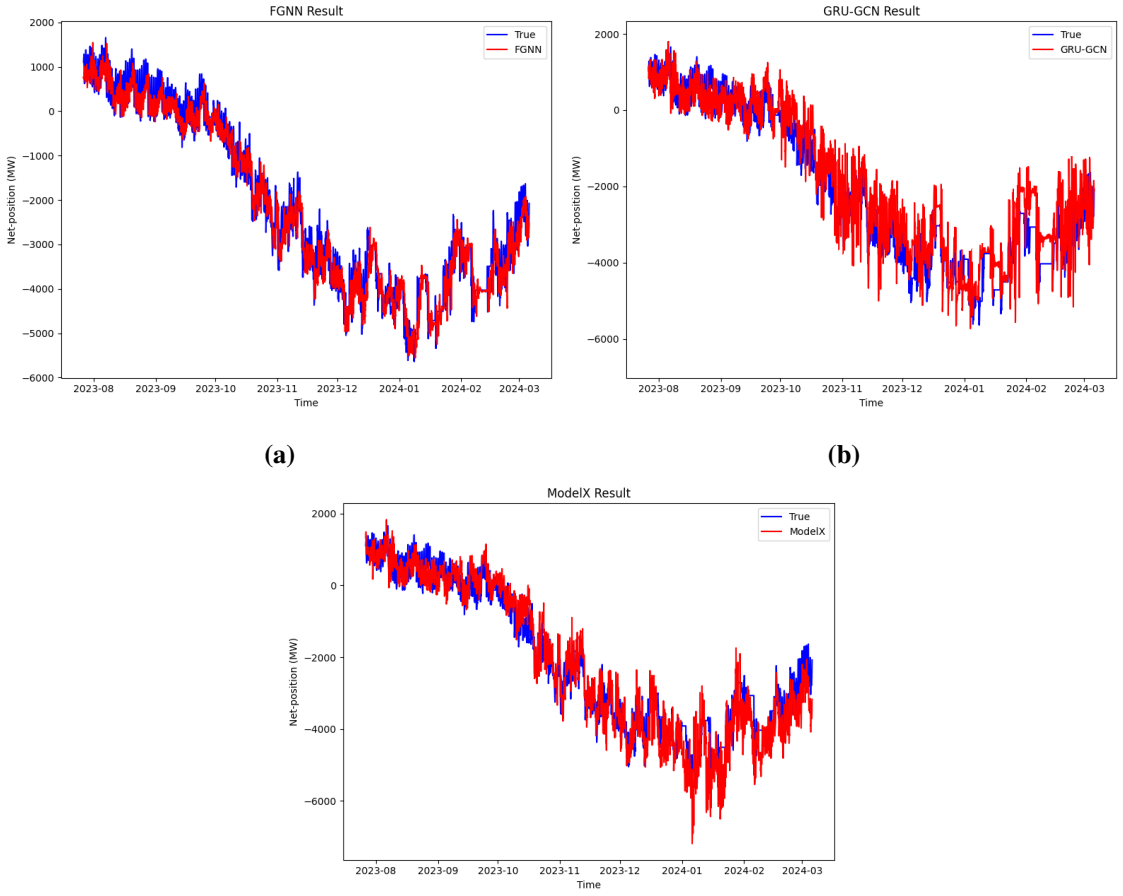
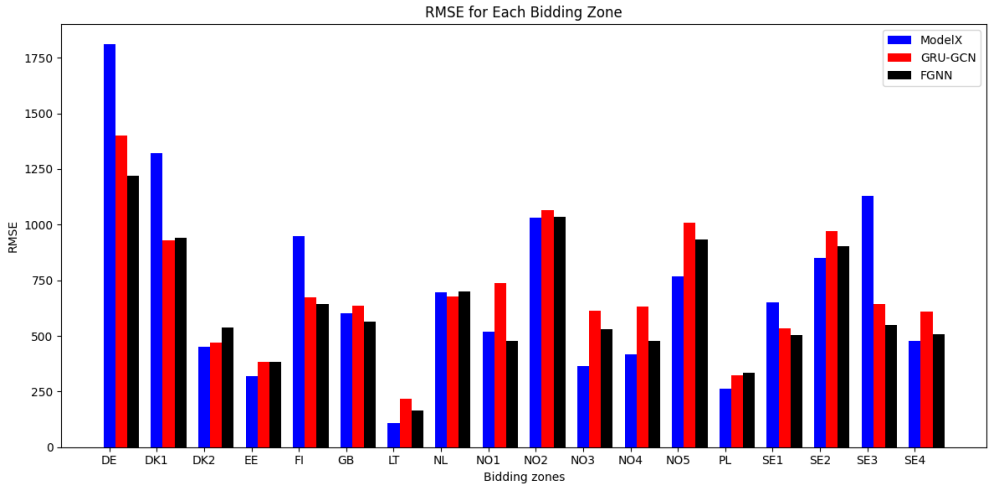
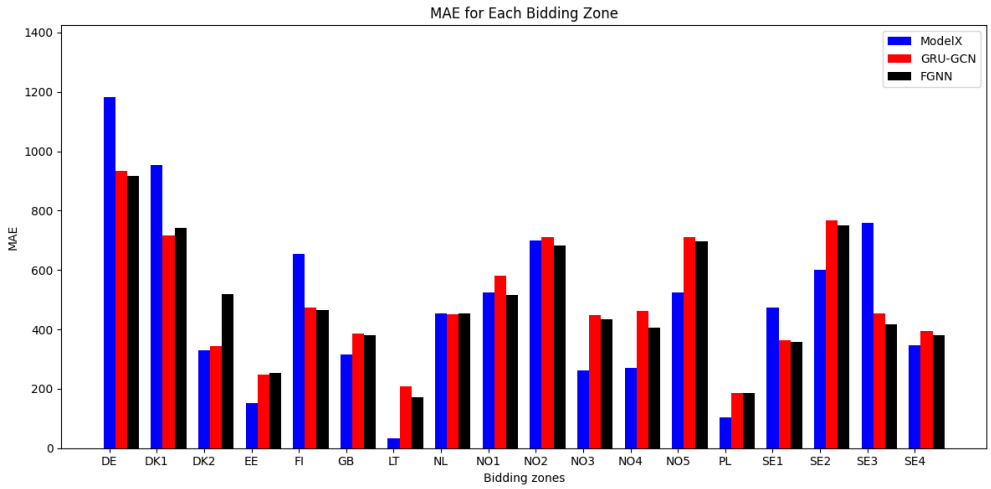


Figure 23: Plot of the net-positions of NO1 over time, with the modified data. **(a)** Using the forecasting model FGNN, **(b)** using GRU-GCN, and **(c)** using ModelX



(a)



(b)

Figure 24: Plot of RMSE **(a)** and MAE **(b)** for the entire test period, using the three models

The RMSE and MAE for Sweden's four bidding zones and the overall performance can be shown in Table 4 and Table 5, respectively.

Table 4: RMSE and MAE for Sweden's bidding zones (BZ)

	GRU-GCN		**FGNN**		**ModelX**	
BZ	**RMSE**	**MAE**	**RMSE**	**MAE**	**RMSE**	**MAE**
SE1	510.2	384.5	501.4	382.9	633.1	505.7
SE2	975.5	790.6	896.8	785.8	862.1	602.4
SE3	624.0	423.3	515.5	402.2	1119.7	788.7
SE4	613.6	396.8	501.8	392.9	495.4	381.5
SE	680.8	498.8	603.9	490.8	777.6	569.6

Table 5: Overall RMSE and MAE

Model	**RMSE**	**MAE**
GRU-GCN	716.6	526.6
FGNN	676.1	502.5
ModelX	748.2	516.8

Both the GRU-GCN and FGNN models achieved competitive performance on the test set. The overall average performance obtained with the GRU-GCN model was a RMSE of 716.6, and a MAE of 526.2. Similarly, the FGNN achieved an RMSE of 676.1 and an MAE of 502.5. ModelX achieved an RMSE of 748.2 and an MAE of 516.8. We suspect that the occasional significant overshoots exhibited by all models might be attributed to data impurities. The presence of frequent disruptions within the underlying dataset creates challenges for all models in perfectly predicting these events. Even with added imputations, we can still see in overall that it's the same bidding zones contributing to an increased uncertainty.

Chapter 4

Discussion & Conclusion

This thesis contributes to the field of power grid forecasting by demonstrating the potential of Graph Neural Networks (GNNs). Two GNN architectures, GRU-GCN and FGNN, were successfully applied to the Nordic electricity market, achieving competitive performance against the existing ModelX. The results indicate that GNNs can effectively leverage the inherent graph structure of the power grid, leading to improved accuracy in net-position forecasts.

The high uncertainty observed in specific bidding zones (DE, DK1, NO2, NO5, SE2) and the variations between models in other zones (DE, DK1, DK2, FI, NO3, NO4, NO5, SE1, SE3) underscore the complex nature of forecasting in these regions. The lower uncertainty in zones like EE, LT, and PL suggests potential regional disparities in data quality or underlying market dynamics.

Despite the promising results, challenges remain. The occasional significant overshoots by all models point towards the need for improved data preprocessing and handling of disruptions. Additionally, the black-box nature of GNNs necessitates further research into interpretability to understand the factors influencing their predictions.

In conclusion, this thesis demonstrates the viability of Graph Neural Networks for power grid forecasting. Both GRU-GCN and FGNN demonstrate competitive performance, suggesting that GNNs are a promising tool for improving power grid forecasting accuracy. However, regional variations in uncertainty and model performance across bidding zones highlight the need for region-specific considerations. Additionally, occasional overshoots by all models emphasize the importance of robust data preprocessing and models capable of handling real-world disruptions. Future research directions include exploring interpretable GNNs, incorporating additional features or data sources, and investigating alternative GNN architectures. These advancements could further enhance GNNs performance and applicability in power grid forecasting.

References

[1] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[2] Alexei Botchkarev. "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology". In: *arXiv preprint arXiv:1809.03006* (2018).

[3] Kyunghyun Cho et al. "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259* (2014).

[4] Lindholm. K. Energiföretagen. *Energiföretagen förklarar: Elområden och elpriset*. (2023). URL: https://www.energiforetagen.se/pressrum/nyheter/2022/augusti/energiforetagen-forklarar-elomraden-och-elpriset/. (accessed: 31.01.2024).

[5] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM". In: *Neural computation* 12.10 (2000), pp. 2451–2471.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. (accessed: 28.04.2024).

[7] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[8] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[9] Svenska kraftnät. *Om elmarknaden*. (2023). URL: https://www.svk.se/om-kraftsystemet/om-elmarknaden/. (accessed: 31.01.2024).

[10] Maxime Labonne. *Hands-On Graph Neural Networks Using Python: Practical techniques and architectures for building powerful graph and deep learning apps with PyTorch*. Packt Publishing Ltd, 2023.

[11] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[12] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).

[13] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[14] Feng Xia et al. “Graph learning: A survey”. In: *IEEE Transactions on Artificial Intelligence* 2.2 (2021), pp. 109–127.

[15] Kun Yi et al. “FourierGNN: Rethinking multivariate time series forecasting from a pure graph perspective”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[16] Ling Zhao et al. “T-gcn: A temporal graph convolutional network for traffic prediction”. In: *IEEE transactions on intelligent transportation systems* 21.9 (2019), pp. 3848–3858.