AMERICAN UNIVERSITY OF BEIRUT

MANY-TASK LEARNING FOR INDIVIDUALIZED CONSUMER POWER PREDICTIONS

by
MARC GEORGE DJANDJI

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Electrical and Computer Engineering
of the Maroun Semaan Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
Januray 2021

AMERICAN UNIVERSITY OF BEIRUT

MANY-TASK LEARNING FOR INDIVIDUALIZED CONSUMER POWER PREDICTIONS

by

MARC GEORGE DJANDJI

Approved by:

Dr. Hazem Hajj, Associate Professor — Advisor

Electrical and Computer Engineering

Dr. Rabih Jabr, Professor — Member of Committee

Electrical and Computer Engineering

Dr. Wassim El-Hajj, Associate Professor — Member of Committee

Computer Science

Dr. Khaled Shaban, Associate Professor — Member of Committee

Qatar University - Computer Science Departement

Date of thesis defense: January, 22, 2021

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name:	DJANDJI	MARC	GEORGE
	Last	First	Middle

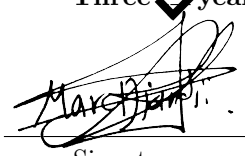
☒ Master's Thesis ○ Master's Project ○ Doctoral Dissertation

☐ I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

☒ I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes after: **One ___ year from the date of submission of my thesis, dissertation or project.**

Two ___ years from the date of submission of my thesis , dissertation or project.

Three ✓ years from the date of submission of my thesis , dissertation or project.



07/02/2021

Signature Date

This form is signed when submitting the thesis, dissertation, or project to the University Libraries

Acknowledgements

I would like to express my gratitude to my supervisor Professor Hazem Hajj for mentoring and accompanying me during the learning process of this master thesis.

I would like to dearly thank the committee members for their invaluable inputs throughout my thesis work.

I would like to thank my friends and colleagues at AUB MIND lab for their support and collaboration during my masters journey.

I would like to thank the Lord and my loved ones, who have supported me throughout the entire process, both by keeping me harmonious and helping me putting the pieces together. I will be forever grateful for your love.

This work was made possible by NPRP11S-1202-170052 grant from Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

An Abstract of the Thesis of

Marc George Djandji for Master of Engineering
Major: Electrical and Computer Engineering

Title: Many-task Learning For Individualized Consumer Power Predictions

Short Term Load Forecasting (STLF) aims at predicting power consumption in the hours, days, or weeks ahead. Accurate STLF is important for plant scheduling, financial planning, system security, short-term maintenance, short-term storage usage, and the application of demand response strategies, which aim at rewarding reduced power consumption at peak hours. State of the art work uses single task deep learning (STL) for its ability to model the uncertainties in the individualized load. However, the approach can be improved further by combining data from other smart meters. To advance STLF accuracy, this thesis explores the use of transfer learning. Two new transfer learning models are proposed: A hierarchical clustering with population STLF prediction models (HC-P) and a hierarchical clustering with deep multitask learning (HC-MTL). Each of the two hierarchical algorithms cluster similar smart meters into groups based on smart meters' data representation. The HC-P approach allows smart meters in each group to learn a shared feature representation. The HC-MTL approach uses hard parameter sharing (HPS) schemes for smart meters in the same group and soft parameter sharing (SPS) for smart meters that are unique and different from all other smart meters. The thesis's additional contributions include two studies for deeper insights into the limitations and opportunities of using transfer learning for STLF. The first study examines the effect of available data on STLF accuracy. The study shows that more smart meter training data helps in improving STLF accuracy, but up to a certain saturation point beyond which, limited performance gains can be obtained. This insight suggests that transfer learning will only make a difference for STLF of smart meters that do not have sufficient training data. To confirm the benefit of transfer learning, a second study examined the effect of getting more data by way of adding data from similar smart meters. Finally, to evaluate the proposed transfer learning STLF approaches HC-P and HC-MTL, experiments were conducted on a dataset consisting of 4225 residential smart-meters and 484 industrial smart meters. Sev-

eral models were implemented for comparison, including: the state of art STL model composed of 1D-Convolutional Neural Network (CNN) with Gated Recurrent Unit (CNN-GRU), a population prediction model without grouping, a hierarchical random grouping with population prediction models, Autoregressive Integrated Moving Average (ARIMA), and Seasonal ARIMA (SARIMA). The results showed that the HC-P worked best for residential smart meters' STLF and HC-MTL worked best for industrial STLF. In fact, compared to prior state-of-the-art, HC-P provided an accuracy improvement of 2% RMSE and MAE for residential smart meters. For industrial smart meters, HC-MTL provided an improvement of 2.78% and 4.97% in terms of RMSE and MAE, respectively

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Smart grids have modernized the traditional electrical grids by leveraging information technology advances, which allow the acquisition of the network components' data and it's use to maximize the efficiency and reliability of the grid. The Advanced Metering Infrastructure (AMI) technology was introduced to realize the smart grid concept. The AMI consists of smart meters, communication network components at every level of the infrastructure, Meter Data Management System (MDMS), and tools to integrate the collected data into proper software application platforms and interfaces [2]. The presence of smart meters in the advanced metering infrastructure (AMI) makes the application of Demand Side Response (DSR) easily applicable for domestic customers [3]. Demand Side Response (DSR) programs are applied by the utility to incentivize customers to reduce their consumption during peak hours with the goal of avoiding potential blackouts or additional energy generation. The application of DSR programs in the U.S. saved around 1.5 million MWh in 2019 [4] resulting in saving approximately 155850000 USD [5]. Various approaches for applying DSR can be found in the literature [6, 7]. However, one crucial requirement for the application of such approaches is the availability of accurate, individualized short-term load forecasts [8]. Furthermore, short-term load forecasting is also required for plant scheduling, fuel purchase plans, security capacity, short-term maintenance as well as short-term storage usage. Although a plethora of works can be found in the literature on aggregate load forecasting, the literature on individualized load forecasting is still developing. The problem of individualized short-term load forecasting (STLF) is challenging at fine temporal granularities, mainly due to the high volatility and uncertainty in the time-series sequences [9]. Benchmarks against prior state-of-the-art shallow neural network and statistical approaches were performed in previous work for individualized STLF [10]. Testing models included Autoregressive Integrated Moving Average (ARIMA), shallow feed-forward neural networks, and exponential smoothing state-space models. The models were tested against a persistence approach (e.g., previous day equals today), and it was found that these methods offer little to no improvements over a

naive persistence approach. Similar findings were reported in [11], where it was found that Support Vector Regression (SVR) and Multilayer Perceptron (MLP) did not outperform simple linear regression.

The main challenge with STLF is the modeling of the uncertainties in the load time-series (TS). Recent works have shown that Deep Learning models consisting of Recurrent Neural Network such as Gated Recurrent Unit (GRU) and Long-short term memory (LSTM) are well suited for learning the uncertainties in the individualized load TS. For example, [12] has shown that LSTM outperforms previous methods such as ARIMA, SVR, Conditional Restricted Boltzmann Machine (CRBM), and Factored Conditional Restricted Boltzmann Machine (FCRBM) for STLF. However, the LSTM suffers from degradation in performance as the length of the TS input increases or multiple modalities are included (i.e., weather TS) [13, 14]. Further improvements were achieved by [1], where it was shown that a Convolutional Neural Network (CNN) and LSTM combination outperforms LSTM, GRU, Bi-directional LSTM, and attention LSTM.

Although deep learning approaches tend to excel with the existence of large amounts of historical data, in the real world, however, individualized historical load TS data is limited, which results in degraded individualized model performance and overfitting [12]. Previous work in the literature addresses the accuracy challenge by proposing a transfer learning strategy that groups data from different smart meters and develops a single model per group in what is called a population model, which allows the model to leverage more data from group smart-meters and improve generalizability as was shown in [12]. Nevertheless, population models tend to overgeneralize by learning only a shared representation among the group smart-meters, which results in inaccurate individualized forecasts. Furthermore, the previous work suggests a random grouping of the smart meters which may not lead to learning a meaningful shared representation. Due to the limitations of STL and population models, a hybrid of both approaches emerged under what is called Multitask Learning (MTL). Multitask Learning enables the model to leverage data from multiple smart meters to learn the shared and task-specific representations resulting in better generalization and individualized accuracy. Recent literature in power prediction started proposing MTL as a method to address generalizability and accuracy. However, the proposed methods either extend statistical models such as Gaussian Process [15] or are unscalable, target larger prediction granularity, and long term load forecasting such as [16].

In this thesis we propose two new transfer learning models to improve STLF accuracy: A hierarchical clustering with population models (HC-P) and a hierarchical clustering with deep multitask learning (HC-MTL). Each of the two hierarchical algorithms clusters smart meters into groups based on the similarity in their load behavior. The HC-P approach allows smart meters in each group to learn a shared feature representation. Whereas the HC-MTL allows the learning of both a shared and smart meter specific feature representations. Furthermore,

the HC-MTL approach uses hard parameter sharing (HPS) schemes for smart meters in the same group and soft parameter sharing (SPS) for smart meters that are unique and different from all other smart meters. The thesis's additional contributions include two studies for deeper insights into the limitations and opportunities of using transfer learning for STLF. The first study examines the effect of available data on STLF accuracy. The study shows that more smart meter training data helps in improving STLF accuracy, but up to a certain saturation point beyond which, limited performance gains can be obtained. This insight suggests that transfer learning will only make a difference for STLF of smart meters that do not have sufficient training data. To confirm the benefit of transfer learning, a second study examined the effect of getting more data by way of adding data from similar smart meters. To evaluate the proposed transfer learning STLF approaches HC-P and HC-MTL, experiments were conducted on a dataset consisting of 4225 residential smart-meters and 484 industrial smart meters. Several models were implemented for comparison, including: the state of art STL model composed of 1D-Convolutional Neural Network (CNN) with Gated Recurrent Unit (CNN-GRU), a population prediction model without grouping, a hierarchical random grouping with population prediction models, Autoregressive Integrated Moving Average (ARIMA), and Seasonal ARIMA (SARIMA). The results showed that the HC-P worked best for residential smart meters' STLF and HC-MTL worked best for industrial STLF. In fact, compared to prior stateof-the-art, HC-P provided an accuracy improvement of 2% Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) for residential smart meters. For industrial smart meters, HC-MTL provided an improvement of 2.78% and 4.97% in terms of RMSE and MAE, respectively.

The contributions of this thesis can be summarized as follows:

1. Evaluation of Data Sufficiency for STL Smart Meter Data:
   (a) The study on 100 residential and industrial smart meters showed that adding more data from each smart meter indeed helps learning, but up to a certain saturation point beyond which, limited performance gains can be obtained.
   (b) We used a deep learning STL model for the experiment. The STL model, previously state of the art, consisted of a 1D Convolutional Neural Network and Gated Recurrent Unit (CNN-GRU) [1]
   (c) For the study data, the observation was more obvious for residential than industrial smart meters. But both showed the trend toward saturated improvement with more data.
2. Evaluation of STLF Learning from Similar Smart meters:
   (a) The study showed that for meters that needed more data, using data from similar smart meters improved performance.

(b) This observation suggests and supports claims that approaches other than STL, such as population and Multi-task Learning (MTL) methods, should improve performance for those meters.

(c) It is worth noting that this observation of benefiting from approaches other than STL is only applicable for smart meters that do not have sufficient data.

3. A New Hierarchical Clustering with Population models (HC-P) approach for residential STLF

   (a) The method groups smart meters into clusters based on the similarity in their behavior as quantified using the distance metric. The formed distance matrix is then used as input to an agglomerative hierarchical clustering algorithm to produce the clusters

4. A New Hierarchical Clustering with Deep Multitask Learning (HC-MTL) for industrial STLF

   (a) We propose an MTL Hard parameter sharing (HPS) extension for the smart meters that belong to the same cluster. The MTL-HPS modifies the CNN-GRU [1] model by keeping the CNN layers and adding a task-specific GRU and fully connected layer per task. For smart meters that were found to be unique and different from all other smart meters, as identified by the clustering, we propose an MTL with a soft-parameter sharing (SPS) scheme. The MTL-SPS consists of a separate CNN-GRU model per task. A regularization term is added to the loss function of each model, which constrains the weights of the layers to be similar by the degree of similarity between the two tasks. Therefore, enabling the tasks to learn the amount of information to share at each layer.

5. Study for Choice of Similarity Metric for Smart Meters

   (a) The study showed that the choice of similarity metric depends on the learning method.

   (b) The experiments showed that cosine similarity on embeddings, extracted within the deep learning MTL STLF model, worked best for HC-MTL. On the other hand, the Euclidean distance of raw smart meter data worked best for the HC-P STLF model.

6. Comparative Analysis of Performances for STLF Models

   (a) Compared Models:

i. **Base Models**: STL Deep Learning [1], Hierarchical Random Grouping with Population (HR-P) [12], ARIMA, SARIMA, Population model without grouping

ii. **New Models**: HC-P, HC-MTL

(b) Experiments were conducted on a dataset consisting of 4225 residential smart-meters and 484 industrial smart meters. The dataset contains measurements with half-hourly granularity starting from July 14, 2009, until December 31, 2010 (536 days)

(c) Tests supported with tests of confidence.

(d) Hierarchical Clustering performed best for residential smart meters. STL was next. Hierarchical Clustering was better by 2% in RMSE and MAE. MTL performed worse than STL by 0.6%.

(e) MTL performed best for industrial smart meters. STL was next. MTL was better by 2.78% in RMSE and 4.97% in MAE. Hierarchical Clustering performed worse than STL by 1.5% RMSE and 8% MAE.

The rest of the thesis is organized as follows: Chapter 2 provides an overview on the related work on single-task, population and multitask approaches for time-series data. In chapter 3, we investigate data sufficiency for STLF and the effects of adding data from similar and dissimilar smart meters. Chapter 4, discusses the smart meter similarity method and provides validation of the clustering approach. Chapter 5, contains the proposed models. Chapter 6 provides the results and a discussion of the conducted experiments. A conclusion of the work is presented in chapter 7

Chapter 2

Background & Related Work

2.1 Related Work

We first present an overview of the single task learning approaches for individual load forecasting using both Deep Learning and statistical techniques. Second, we present a review of population approaches. Third, we present a review of multi-task approaches using Deep Learning architectures and statistical techniques.

2.1.1 Single Task Learning (STL)

Single task learning models represent the traditional way in which any model is trained that is when the model is trained to minimize a single loss function [17]. Furthermore, STL models can be further divided into two categories based on the underlying model architecture: 1) Statistical and Feature-based models, 2) Deep Learning models.

2.1.1.1 Statistical & Feature-Based Models

Few works in the literature studied the problem of individualized STLF using statistical and feature-based approaches. [18] compared different previous state-of-the-art aggregate load forecasting approaches for the problem of individualized day ahead load forecasting. Tests were conducted for a population of 90 and 230 smart meters. Shallow feed forward neural network (FFNN), shallow wavelet neural network (WNN), ARIMA, AMRA and Fuzzy logic WNN, were compared and it was found that ARIMA modelling provided the best results in terms of Normalized Root Mean Squared Error (NRMSE). In [11], a comparison between Multilayer Perceptron (MLP), Support Vector Regression (SVR) and Linear Regression was presented for next-day individualized and aggregate hourly prediction. The input for individualized prediction was the energy consumption for the same hour of the previous seven days in addition to hour-of-day and day-of-week features, where it was found that linear regression outperformed MLP

Table 2.1: Summary of the previous works targeting load forecasting using Smart Meter data.

Paper	Features	Prediction Span	# of Smart Meters	Best Model
[18]	Last week's load	Day-ahead with hourly granularity	230	ARIMA
[11]	The energy consumption for same hour of the previous seven days, hour-of-day and day-of-week	Day-ahead with hourly granularity	782	linear regression
[19]	Previous 60 hours	Sixty hours ahead	1	LSTM Seq-to-Seq
[20]	Previous 12 half-hours	Half-hour ahead	69	LSTM
[21]	Previous 60 hours and calendar features	Sixty hours ahead	1	CNN
[1]	Previous 60 hours, hour-of-day, day-of-week, month-of-year	Sixty hours ahead	1	CNN-LSTM
[12]	Load for the previous 7 days with half-hourly granularity	Day-ahead with half-hourly granularity	920	LSTM
[22]	Load for the previous year, calendar features: time-of-day, day-of-year, day-of-week.	Half-year ahead with 3 hours granulatiy	6433	Multitask Output Kernel Learning

and SVR. In [10], a comparison between ARIMA, shallow FFNN, exponential smoothing models, and a naive persistence approach (i.e., previous day equals today) was performed. It was reported that the previous methods offer little to no improvements over the naive persistence approach. All these findings suggest that modelling short-term time-series data need more complex models that can accurately model the uncertainties in the load time-series data.

2.1.1.2 Deep Learning models

Because the statistical modelling approaches are limited to capturing stationary information and modeling the linear relationships in the time-series data, researchers have shifted their attention to using data driven approaches that overcome the deficiencies of statistical modelling. For example, in [19], a Long Short-Term Memory (LSTM) was proposed for the task of predicting the individual consumer load for next 60 hours and the next minute given the consumption of the last 60 hours and the last minute, respectively. Furthermore, it was found that LSTM sequence to sequence model provided better results than the standard LSTM. In [20], the authors extended the experiments of running an LSTM model to include tests on 69 residential consumers for predicting the load for the next half hour given an hour, 3 hours and 6 hours of previous consumption. Moreover, it was found that the LSTM model outperforms k-nearest neighbors and Extreme Learning Machine models. They have confirmed that LSTMs can capture the subtle changes in the individual load data. However, the individual forecasts had an approximate 44% average mean absolute percentage error showing that the individual load forecasting remained a challenging problem. In [21], the use of Convolutional Neural Networks (CNN) for STLF of a single home was

explored. It was found that CNN's outperform feature-based models such as SVR and perform similarly to LSTMs. LSTMs were shown suffer from degradation in performance when multiple modalities are included (e.i., weather TS in addition to load TS) ([13, 14]), Kim et. al [1] suggested a CNN-LSTM combination for predicting the next 60 minutes given previous multimodel time-series data. It was shown that CNN-LSTM outperforms Linear Regression, Decision Tree, Random Forest, LSTM, GRU, Bi-directional LSTM, and attention LSTM.

In summary, the STL approaches proposed solutions that are limited to a single household or a small number of residential smart meters. Furthermore, for the previous deep learning approaches to outperform statistical and feature-based methods they require the availability of large amounts of historical data. In the real world, however, individualized historical load TS data is limited, which results in degraded individualized model performance and overfitting [12].

2.1.2 Population Models

A population approach refers to the idea of developing single model per group of smart meters or entire population of smart-meters in the grid. Recent work have shown that grouping allows the model to leverage data from customers of similar behavior, reaction to weather changes, and appliances to compensate for the lack of insufficient individualized historical data and results in better generalization. For example, in [12], 920 smart meters were split randomly into 10 groups of 92 and for each group a single LSTM model was developed and predictions for the next day with hourly granularity were targeted. The previous work have shown that grouping smart-meters to train an LSTM model provided improved performance in comparison to individualized (LSTM, SVR, RNN and ARIMA) models. Humeau et al. [11], proposed to group customers by clustering rather than randomly. It was found that extracting the average daily load profile feature and using it for clustering smart-meters using k-means, improves load forecasting if a population model was developed per cluster. They used an SVR and shallow MLP models. In [23], although authors targeted the problem of accurate aggregate load forecasting for the next day in hourly granularity, they confirmed that clustering smart meters using K-means and developing a model per cluster improved aggregate forecasting in comparison to developing a single model for all smart-meters.

Although population models improve deep learning model's generalizability, they tend to also make models overgeneralize by learning only a shared representation among the group smart-meters, which results in inaccurate individualized forecasts.

2.1.3 Multitask Task Learning (MTL)

Multi-task learning (MTL) is introduced as a learning paradigm that enables machine learning models to transfer the important learned information between related tasks in what is called inductive transfer of knowledge under the assumption that commonalities exist between the learned tasks. Furthermore, the main advantages of MTL is that it reduces the requirements for large amounts of labeled data, improves the performance of a task with less data by leveraging the common information from the related tasks with more data, and enables the model to be robust to missing observations for some tasks [24, 25].

2.1.3.1 Statistical Models

Recently, more works propose multitask learning approaches for load forecasting. For example, in the work of [26], a multitask Gaussian Process model was proposed for the short-term aggregate load prediction of 3 different cities. Although the proposed MTL model decreased the prediction error by a significant margin in comparison to other models, the model targets aggregate load prediction rather than individualized. In addition, only a small number of tasks where modelled, therefore, the scalability issues associated with modelling a large number of tasks were not addressed. In [22], the authors tried to address the scalability challenges of short-term load forecasting of (6433) households. The dataset contained meters categorized into (Residential, Small or Medium enterprise, and Others) and the authors proposed an MTL model for the residential and others categories and a separate model for the enterprise smart meters. Furthermore, a multitask low rank output kernel learning approach was applied in order to learn the similarities between the different smart meters within each category, where the complexity of the model was controlled by the size of the similarity matrix between the different meters within a group. Finally, the kernel-based multitask models were specifically optimized to capture the seasonal effects that are present in electricity load data outperforming other statistical single task methods such as kernel-based ridge regression. The main disadvantages of the proposed method in [22] is that it targets year ahead prediction with 3 hours granularity and does not scale to STLF with half-hourly granularity. Furthermore, the method is highly dependent on the kernel choice requiring domain knowledge and does not allow the learning of the shared and task-specific information.

In summary, the provided statistical MTL approaches proved to be superior to statistical single-task learning models. However, the proposed methods inherent the deficiencies of statistical approaches and cannot model the complex non-linear relationships within the short-term time-series data. Furthermore, they do not scale for half-hourly granularity predictions.

2.1.3.2 Deep Learning models

No previous work has attempted the extension of deep learning models with MTL for short-term load forecasting. Therefore, we will include in this section some of the previous Multitask deep learning works that were designed for sequence and time-series data modelling in general. Two main MTL design schemes can be found in the literature:

1. MTL with a Hard-parameter sharing scheme (MTL-HPS)
2. MTL with a Soft-parameter sharing scheme (MTL-SPS)

MTL with Hard-parameter Sharing (MTL-HPS) MTL using hard parameter sharing mainly involves having the model divided into two parts. The First part is a shared layer which gets trained by all the tasks' data to extract a general feature representation from all the tasks. The second part of the model involves using task-specific layers which get trained only by the task-specific data in order to capture the task-specific characteristics [24].

For example, In [27], an MTL Deep Learning model was proposed for solving multiple Natural Language Understanding (NLU) regression and classification tasks such as question answering, predicting the semantic similarity between different sentences and assessing the sentence grammatical plausibility. Furthermore, the proposed model implements MTL in a hard-parameter sharing scheme and provides state-of-the art results in comparison to previous methods. In [28], a hierarchical network composed of a shared feature extraction convolutional neural network (CNN) followed by separate task-specific Conditional Random Field (CRF) models was proposed to learn linguistic sequence tagging tasks together in a unified model. Their work showcases the effectiveness of using a shared CNN feature encoder followed by a task-specific sequential part to capture temporal dependencies. MTL-HPS models were also extended to spatio-temporal data modelling. For example, in [29], the authors targeted the problem of predicting the minimum, maximum and average mobile network traffic load in the next hour given the traffic data in the previous hour for 1503 locations within a city. Different MTL-HPS models were proposed where each location was considered to be a task. The different models were fed with a sequence of grids across time, where each grid-cell represented the mobile traffic-load in each location within the city and the sequence of grids represented how the mobile traffic is changing over time. It was found that among the three proposed multitask deep learning models (Long-Short Term Memory (LSTM) Model, 3D-Convolutional Neural Networks (3D-CNN) Model, and CNN-LSTM Model), the CNN-LSTM Model achieved the best results due to its ability to extract both spatial information (i.e., relationship between the adjacent locations) and temporal dependencies within the grid sequences. Other works have also leveraged the use of MTL-HPS

for time-series data modelling. In [30], pain estimation based on individual physiological and behavioral pain response was targeted and MTL augmentation of FFNN provided optimal performance in comparison to previous work. In [31], the problem of estimating the number of vehicles in a road or station during short-term time intervals (5-30 minutes) and found that MTL augmentation of deep learning models provided 5% over prior state-of-the-art methods.

MTL using hard-parameter sharing has the main advantage of enabling the model to learn a shared and task specific representation. Furthermore, MTL-HPS provides a scalable solution in terms of memory footprint as it requires the addition of a small number of parameters per task while having the same number of the shared part parameters. However, MTL-HPS requires the tasks to be highly related e.g., are drawn from similar distributions, and does not incorporate sharing partial information between the task specific layers in case some tasks exhibit some level of similarity on a very low level.

MTL with Soft-parameter Sharing (MTL-SPS) MTL with a soft-parameter sharing setting requires that each task has its own model with its separate parameters. Furthermore, the parameters of the different tasks' models get regularized in order to learn both what parameters should be shared between the different tasks and how much of each layer should be shared [17], therefore, enabling the model to extract how the different tasks interact with each other. For example, in [32], a generalization of hard-parameter sharing and block-sparse regularization approaches for deep learning was proposed in order to learn which layers and parameters should be shared, as well as at which layers the network has learned the best representations of the input sequences. Furthermore, the number of parameters of the proposed model increases linearly with the number of layers and quadratically with the number of tasks making it computationally prohibitive for a large number of tasks.

In spite of the fact that soft-parameter sharing schemes enable the model to learn exactly what to share, which improves the model's performance in case the tasks that are being learned are not strongly related, this learning paradigm requires learning additional parameters for each task and layer making the training process more complex and computationally expensive.

2.2 Background

In this section we provide a background obout STLF and provide the mathematical formulation of the problem using an STL model.

2.2.1 STLF Mathematical Formulation

Short Term Load Forecasting (STLF) is usually affected by the following factors [33]:

- Hour-of-day, on a specific day of week
- Weather information: Temperature, wind, humidity, and cloud cover.
- Utility hour-of-day pricing strategy
- Special events such as strikes, TV programs, or major political conventions.
- Random unknown factors

Traditionally for aggregate STLF, which refers to the STLF of the sum of the loads in the grid, these different factors are modeled separately, and the output of their models is then combined to form the total real load:

$$P_L(t) = B(t) + W(t) + S(t) + v(t) \tag{2.1}$$

Where: P_L: Total load at time t; W_t: Weather-sensitive load component at time t; B_t: Base (normal) load at time t; S_t: Load increment due to special events t; $v_(t)$: Random load component.

The time t is measured periodically, e.g., every hour, minute, 10 minutes etc., depending on the type of application. The historical data from the immediate past (i.e., last five weeks of hourly load and weather data) are used to identify the overall model in Eq. 2.1 and the following steps are performed:

- Smooth out the unexplained loads due to special events since they do not represent normal behavior
- Identify the parameters of the weather-sensitive part $W(t)$ and subtract it from P_L (t) to obtain an estimate of the base load $B(t)$
- $B(t)$ is then modeled using a sequential time-series model such as Auto Regressive Integrated Moving Average (ARIMA) model
- Combine the predictions from steps 2 and 3 to provide the predicted load.

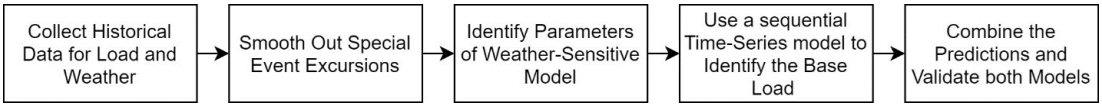


Figure 2.1: Steps Involved in Model Identification for Aggregate Short-Term Forecasting

For individualized STLF, which is STLF for each smart meter in the grid, the previous state-of-the-art approach in [1] proposed a single deep learning model that directly models all the previous components of the load to produce the prediction. Kim et al. [1] assumed that the relationship between the different predictive components and the predicted load is non-linear and thus can be modeled using a deep learning model. Furthermore, the weather and the special events' effects can be accounted for by adding weather and calendar features as input to the model, respectively, in addition to the historical load for the previous days.

2.2.2 Single Task Learning (STL) Mathematical Formulation

Kim et al. [1] addressed the problem of predicting the next 60 hours given historical data for the past 60 hours. They proposed a deep learning model consisting of a 1D-Convolutional Neural Network (CNN) for extracting shift and scale invariant features from the input signals which are then passed to a Long Short-Term Memory (LSTM) layer to extract sequential information from the features. Furthermore, the proposed method was applied to a dataset containing a single house in France. The dataset contains 7 different signals constituting the historical load for the house, which are 1) Global active power (GAP). 2) Global reactive power (GRP). 3) Voltage. 4) Global intensity (GI). 5) Three sub meter's signals in watts-per-hour. In addition, calendar features were added to the input such as day of month, day of week, month of year, and hour of day.

Suppose the dataset contains (N) total data-points for a smart meter (SM) (t), the training data contains n_t examples (x_i^t, y_i^t), where $i \in 1, 2, \ldots, n_t$. The input x_i^t is an input instance in $R^{Win \times C}$ for SM (t), where (Win) is the window length or prediction span (e.g., 60 hours) and C is the number of input feature-vectors. The input to the model x_i^t consists of the concatenation of multiple feature vectors consisting of:

1. Global active power $E_{\text{GAP}} \in R^{\text{Win}}$.
2. Global reactive power $E_{\text{GRP}} \in R^{\text{Win}}$
3. Voltage $V \in R^{\text{Win}}$
4. Global intensity $E_{\text{GI}} \in R^{\text{Win}}$
5. Three sub meter's signals in watts-per-hour $\text{E}_{S1},\ E_{S2},\ E_{S3} \in R^{\text{Win}}$.
6. Calendar numerical features for the input window:
   - Hour of day $H_r = \{0, 1, \ldots 23\} \in R^{\text{Win}}$
   - Day of week of day $W_k = \{1, 2, \ldots 7\} \in R^{Win \times d}$

- Month $M = \{1, 2, \ldots 12\} \in R^{Win \times d}$
- Day of month$M_d = \{1, 2, \ldots (28,\ 30,\ 31)\} \in R^{\text{Win}}$

$$x_i^t = [E_{GAP}, E_{GRP}, E_{GI}, E_{S1}, E_{S2}, E_{S3}, H_r, W_k, M, M_d] \tag{2.2}$$

As it is intended to predict the power consumption the task here is a regression problem.

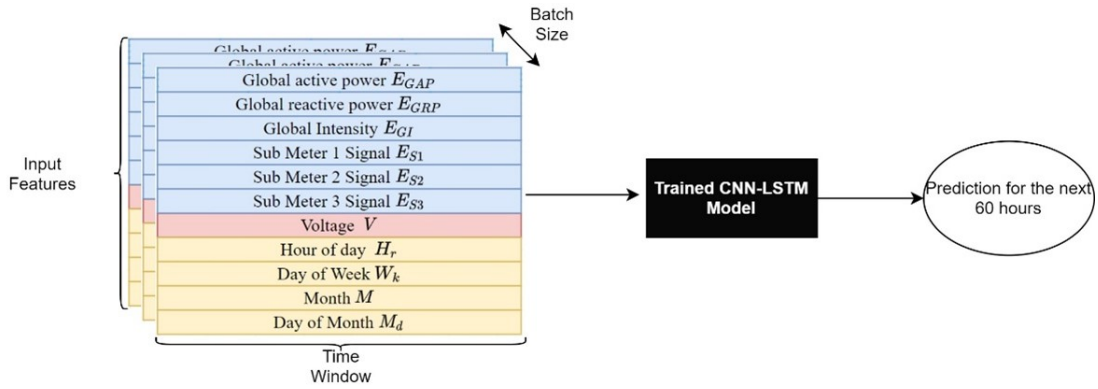


Figure 2.2: The input and output of the model proposed by Kim et al. [1]

The goal is to train a model to predict the power consumption for a smart meter $Y^t \in R^{\text{Win}}$ given the previous hours' load and calendar features $X^t \in R^{Win \times C}$. Let Γ denote the CNN-LSTM layers and $W_t \in R^{H \times Win}$ the parameters of the output fully connected layer, where H is the dimensionality of the output vector from the LSTM and Win is the output window size. The Mean Squared Error (MSE) denoted as L is the loss function that the model is trained to minimize:

$$L\left(f\left(x_i^t; \Gamma, W_t\right), y_i^t\right) = L\left(f\left(g\left(x_i^t; \Gamma\right); W_t\right), y_i^t\right) \tag{2.3}$$

$$p_i^t = \left((W_t)^\top g\left(x_i^t; \Gamma\right)\right) \tag{2.4}$$

$$L\left(f\left(g\left(x_i^t; \Gamma\right); W_t\right), y_i^t\right) = \frac{1}{n_t} \sum_{i=1}^{n_t} \left(p_i^t - y_i^t\right)^2 \tag{2.5}$$

Where $g(.) : R^{Win \times C} \rightarrow R^H$ represents a function applied by the CNN-LSTM feature extraction model-part on the input to obtain an embedding representation of the input. $f(.)$ is a function applied by the fully connected layers to obtain the prediction. p_i^t is the predicted value of power consumption for input example i and smart meter t. Eq. 2.5 shows the Mean Squared Error (MSE) loss that is calculated for the inputs of smart meter t.

The STL objective function can be given as follows:

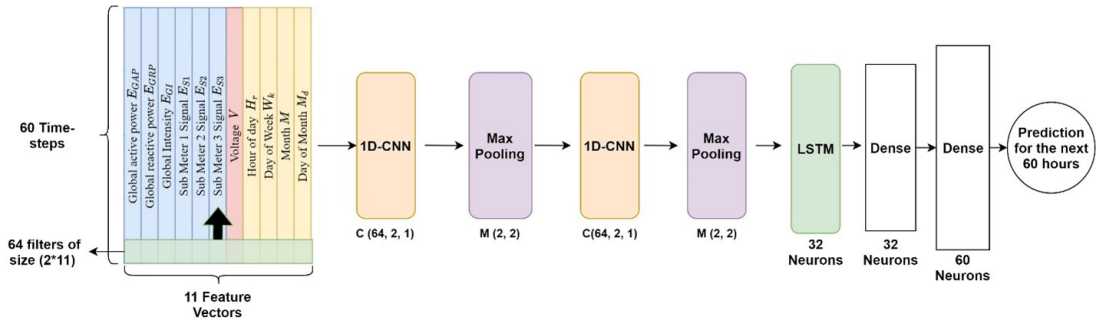


Figure 2.3: Prior State-of-the-art [1] Detailed Architecture

$$\min_{W,\Gamma} \frac{1}{n_t} \sum_{i=1}^{n_t} L\left(f\left(x_i^t; \Gamma, W_t\right), y_i^t\right) \tag{2.6}$$

Figure 2.3 shows the detailed prior state-of-the-art CNN-LSTM model [1]. The letter description are as follows: $C(F,\ K,\ S) : 'C'$ represents a convolution layer with $'F'$ filters, $'K'$ kernel size and $'S'$ strides. $M(K,\ S) : 'M'$ represents a Max Pooling layer with $'K'$ kernel size and $'S'$ strides. The model processes the data by first applying a series of 1D-convolutions. For example, the filter in the first convolution layer strides over the input signals and in each step the data and the filter, which are of size 2×11, are flattened to become a vector of dimensionality 22. Then the two vectors are convolved to produce a single feature. Since the first layer has 64 filters, each stride generates 64 features. The convolution process is then repeated until the filter reaches the last row of the valid data. The convolution procedure extracts both the relationship between the different features and the evolution of each feature over a small window of time. The Max Pooling layers then reduce the dimensionality of the extracted features while retaining only the important features. The cascade of the convolution and Max Pooling layers creates a hierarchy of abstract features that capture both the small temporal changes and the relationship between the different input features. The resulting feature matrix is then fed into the GRU layer to extract the temporal dependencies across the different features. The final embedding is then fed to fully connected layers to perform the inference.

In our experiments, we use the same architecture proposed in [1] with the main difference being in replacing the LSTM layer with a Gated Recurrent Unit (GRU) layer, because GRUs provide good performance while being more computational efficient than LSTMs [34].

Chapter 3

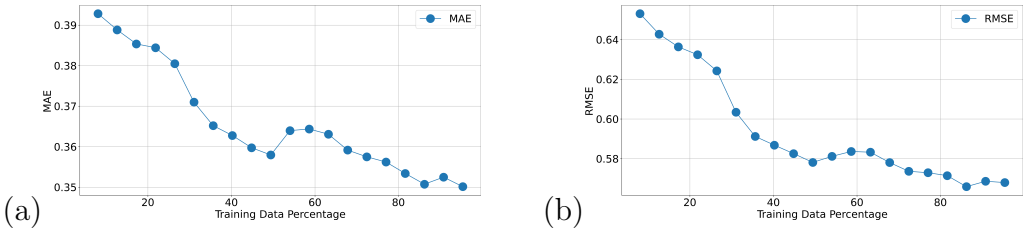
STLF Limitations & Opportunities

During our experiments we found that transfer learning approaches such as Multitask Learning and Population modelling did not initially help improve the models accuracy as was claimed in [12]. Therefore, in this section, we investigate the transfer learning possibility for smart meter data. We first explore the data sufficiency for STL models. We then explore the effect of adding data from smart meters of similar data and the effects of adding data from dissimilar smart meters.

3.1 Study for Evaluation of Data Sufficiency for STL Smart Meter Data

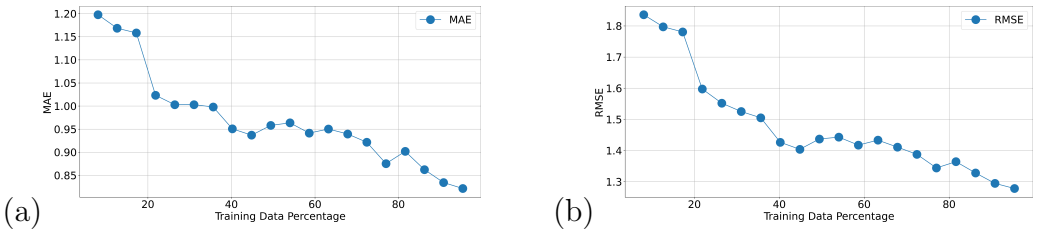
To validate the hypothesis of limited accuracy due to limited training data, we sample 50 smart meters from each type of smart meters in our dataset e.g. residential and industrial smart meters. We then reduce the training data size for each smart meter to 5% of its original size and then start increasing the training data size by 5%. For each training data size we fit an STL model per smart meter and collect the test set Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) then we average the RMSE and MAE over the 50 smart meters within each group as shown in Fig 3.1 3.2 .

From Figures 3.1 and 3.2 we can see that the average MAE and RMSE is decreasing as the training data size increases. The same behavior can be noticed for both residential and industrial smart meters. However, the performance gains, caused by the training data size, saturates at a certain training data size. In summary, we validate that indeed more training data results in improved performance but up to certain threshold after which performance gains are limited.



(a) (b)

Figure 3.1: The performance metrics averaged over 50 industrial smart meter as the training data size increases. a) The average MAE. b) The average RMSE.



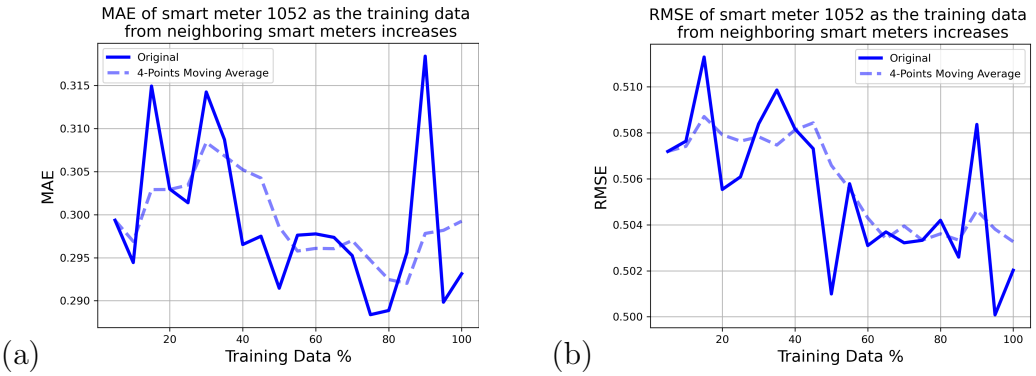
(a) (b)

Figure 3.2: The performance metrics averaged over 50 industrial smart meter as the training data size increases. a) The average MAE. b) The average RMSE.

3.2 Study for Evaluation of Transfer Learning from Similar Smart meters

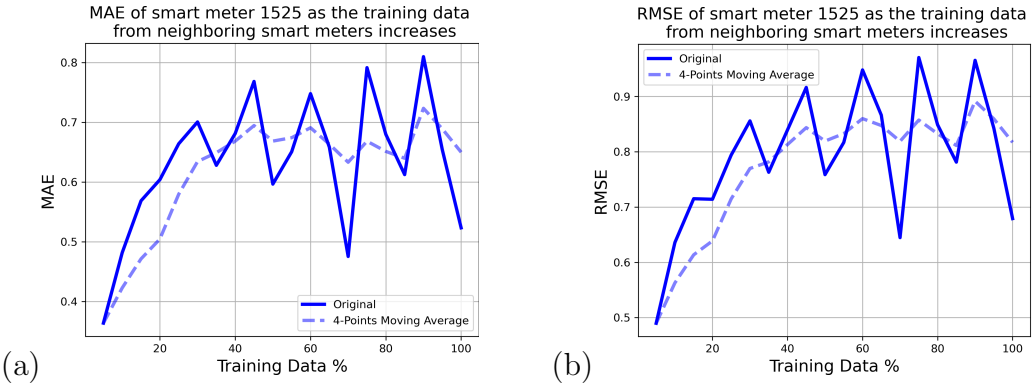
In this section we attempt to validate that adding more data from neighboring smart meters improves the accuracy of the smart meter with little training data. We build our experiment by first choosing 50 smart meters from each type of smart meters e.g., residential, and industrial smart meters. We then choose at random one smart meter from each group and reduce its training data size to 20% of its original size. We then identify the smart meters that are most similar to the one we have chosen earlier by first calculating the average Euclidean distance between its daily half-hourly load and the daily half-hourly load of all 50 smart meters within the same smart meter type. We use the resulting distance matrix as input to an agglomerative clustering algorithm to identify the cluster to which the smart meter belongs. After identifying the cluster for each smart meter, we reduce the training data size of the neighboring smart meters to 5% of its original value and then increase their training data size to 100% by steps of 5%. For each training data size, we fit a population model for and record the RMSE and MAE performance of the smart meter with the reduced training data size on the test set. Fig. 3.3-3.4 shows the RMSE and MAE for the residential smart meter with ID 1052 as the training data size of its neighbors increases. We can see that both the RMSE and MAE are decreasing as the training data size of the neighboring smart meters increases from 5% to a 100% of its size. The final recorded RMSE and MAE for smart meter 1052 is 0.50 and 0.293, respectively. We compare these

values to the metric values obtained by training a separate STL model for smart meter 1052 on its full training data as shown in Table 3.1. We can see that the performance improved by 13.8% and 8% in terms MAE and RMSE, respectively. We repeated the same experiment for an industrial smart meter with ID 1525 and obtained an improvement of 55% and 51% in terms of RMSE and MAE, respectively.



(a) (b)

Figure 3.3: The performance metrics for a residential smart meter as the training data size for its neighbors increases. a) The MAE. b) The RMSE.



(a) (b)

Figure 3.4: The performance metrics for an industrial smart meter as the training data size for its neighbors increases. a) The MAE. b) The RMSE.

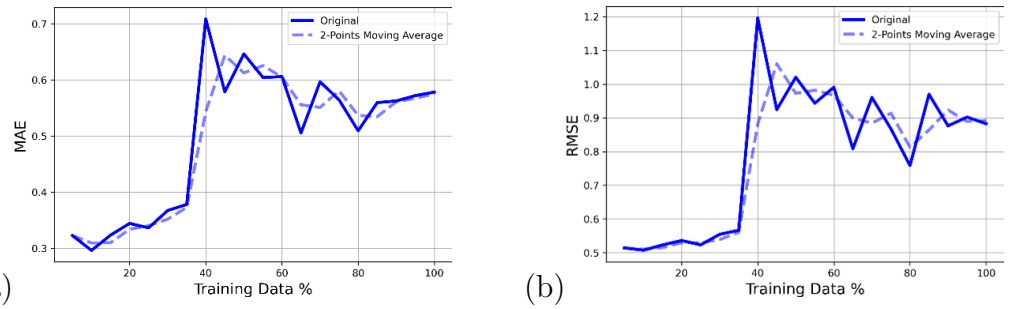
In summary, we can conclude that adding more data from similar smart meters improves the performance of the smart meter with little data in comparison to an STL model trained on the full-sized training data of the same smart meter.

Table 3.1: Comparison between the performance of the STL model when trained on the full smart meter's data and the population model when trained on a reduced size of the smart meter's data with the presence of similar smart meter's data.

	Population for Residential SM	**STL for Residential SM**	**Population for Industrial SM**	**STL for Industrial SM**
RMSE	**0.50**	0.63	**1.18**	1.40
MAE	**0.293**	0.34	**0.52**	0.67

3.3 Study for Evaluation of Transfer Learning from Dissimilar Smart meters

In this section we attempt to validate that adding more data from dissimilar smart meters degrades or limits the accuracy of the smart meter with little training data. We build our experiment similar to the previous section by first choosing 50 smart meters from each type of smart meters e.g., residential, and industrial smart meters. We then choose at random one smart meter from each group and reduce its training data size to 20% of its original size. We then identify the smart meters that are most dissimilar to the one we have chosen earlier by first calculating the average Euclidean distance between its daily half-hourly load and the daily half-hourly load of all 50 smart meters within the same smart meter type. We use the resulting distance matrix as input to an agglomerative clustering algorithm to identify the cluster that is most distant to the smart meter with little data. After identifying the cluster for each smart meter, we reduce the training data size of the neighboring smart meters to 5% of its original value and then increase their training data size to 100% by steps of 5%. For each training data size, we fit a population model and record the RMSE and MAE performance of the smart meter with the reduced training data size on the test set.



(a) (b)

Figure 3.5: The performance metrics for a residential smart meter as the training data size for its dissimilar neighbors increases. a) The RMSE. b) The MAE

Figures 3.5 shows the RMSE and MAE for the residential smart meter with

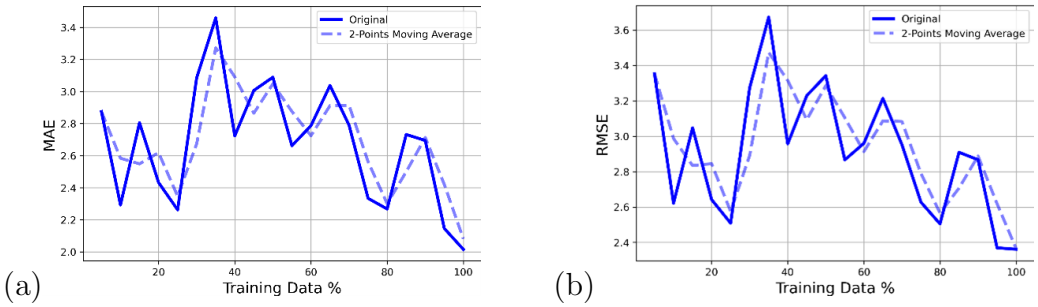


Figure 3.6: The performance metrics for an industrial smart meter as the training data size for its dissimilar neighbors increases. a) The RMSE. b) The MAE

Table 3.2: Comparison between the performance of the STL model when trained on the full smart meter's data and the population model when trained on a reduced size of the smart meter's data with the presence of dissimilar smart meter's data.

	Population for Residential SM	**STL for Residential SM**	**Population for Industrial SM**	**STL for Industrial SM**
RMSE	0.89	**0.63**	2.37	**1.40**
MAE	0.57	**0.34**	2.05	**0.67**

ID 1052 as the training data size of its dissimilar neighbors increases. We can see that both the RMSE and MAE are increasing as the training data size of the neighboring smart meters increases from 5% to a 100% of its size. The final recorded RMSE and MAE for smart meter 1052 is 0.89 and 0.57, respectively. We compare these values to the metric values obtained by training a separate STL model for smart meter 1052 on its full training data as shown in Table 3.2. We can see that the performance dropped by 41.26% and 67.64% in terms RMSE and MAE, respectively. We repeated the same experiment for an industrial smart meter with ID 1525 and obtained a degradation in performance of 69.28% and 205% in terms of RMSE and MAE, respectively.

In summary, we can conclude that adding more data from dissimilar smart meters degrades the performance of the smart meter with little data in comparison to an STL model trained on the full-sized training data of the same smart meter.

Chapter 4

Similarity Methods

4.1 Similarity Method

Recent work has shown that grouping smart meters and training a model for each group allows the model to leverage data from customers of similar behavior, reaction to weather changes, and appliances to compensate for the lack of insufficient individualized historical data resulting in better generalization. For example, in [12], 920 smart meters were split randomly into 10 groups of 92 and for each group a single LSTM model was developed and predictions for the next day with hourly granularity were targeted. The previous work has shown that grouping smart-meters to train an LSTM model provided improved performance in comparison to individualized (LSTM, SVR, RNN and ARIMA) models. Humeau et al. [11], proposed to group customers by clustering rather than randomly. It was found that extracting the average daily load profile feature and using it for clustering smart-meters using K-means, improves load forecasting if a population model was developed per cluster. They used an SVR and shallow MLP models. In [23], although authors targeted the problem of accurate aggregate load forecasting for the next day in hourly granularity, they confirmed that clustering smart meters using K-means and developing a model per cluster improved aggregate forecasting in comparison to developing a single model for all smart-meters. Given the findings in the literature, we hypothesize that clustering smart meters provides a better performance as oppose to random grouping as suggested in [12]. To validate our hypothesis, we first need to validate that there is a clustering tendency in our dataset. In other words, we need to validate that the dataset has a non-random structure thus clustering can lead to meaningful clusters. We then explain our choice of clustering algorithm. And finally, we compare between the performance of a population model trained on a cluster of smart meters and the performance of a population model trained on a group of smart meters picked at random.

4.1.1 Clustering Tendency

Clustering tendency analysis is used to validate whether a given dataset has a non-random structure, which may lead to the discovery of meaningful clusters. For example, if a dataset mostly consists of random structures, such as a set of uniformly distributed points, then clustering that dataset may return some clusters but those clusters will be random and are meaningless [35]. We perform the clustering tendency assessment by calculating the Hopkins Statistic, which is a spatial statistic that tests the spatial randomness of variables as distributed in a space [35]. The Hopkins Statistic informs us of how likely it is for a dataset to follow a uniform distribution in the data space by calculating an H value over the datapoints that are to be clustered. The null hypothesis is that the dataset is uniformly distributed and thus contains no meaningful clusters. If $H < 0.5$ then we reject the null hypothesis and assume that we have statistically significant clustering structure in our dataset. We calculate the H value over the four different representations of our dataset points:

1. **Daily Load:** After splitting the dataset into train, test, and validation set, we take the training data's daily load over 48 half-hours for each smart meter. And we use these vectors as input to the hypothesis test.

2. **Average Daily Load:** After splitting the dataset into train, test, and validation sets, we represent each smart meter by the average daily load over 48 half-hours similar to what was suggested in [9]. And we use these vectors as input to the hypothesis test.

3. **Predicted Daily Load Embedding:** After splitting the dataset into train, test, and validation sets, we train a separate STL model for each smart meter and collect the embedding representation for the prediction of each input example from the dense layer prior to one before the last layer. We include trained models' embeddings in our analysis, in addition, to the raw data in points 1 and 2, because we hypothesize that trained model embeddings would allow the measurement of distance in the deep learning feature space as oppose to Euclidean space, which might result in a more accurate quantification of the distance between the models of the different smart meters.

4. **Average Predicted Daily Load embedding:** We generate the embeddings for the smart meters same as in 3. And then represent each smart meter by the average embedding vector e.g. We will have one embedding (vector) per smart meter.

Table 4.1 shows the H value for each group of smart meters and for each smart meter representation choice. We can see from the table that the H value for all the smart meter representation choices and all the smart meter groups

Table 4.1: The Hopkins statistic value (H-value) for each group of smart meters and each smart meter representation choice

SM Representation \SM Type	Daily Load	Average Daily Load	Predicted Daily Load Embedding	Average Predicted Daily Load Embedding
Residential	0.054	0.050	0.014	0.261
Industrial	0.041	0.087	0.008	0.295

is significantly less than 0.5. Therefore, we can reject the null hypothesis and validate that the dataset is not uniformly distributed thus containing meaningful clusters.

4.1.2 Clustering Algorithm

One of the popular algorithm choices for clustering smart meter data is K-means, as was suggested in [11, 23]. K-means is a simple clustering algorithm to identify spherically shaped clusters requiring the number of clusters as a main hyperparameter [10]. The number of clusters can be specified by three different approaches:

1. **Domain knowledge:** Given some knowledge of the dataset and the problem provide a guess of the possible number of clusters.

2. **Heuristics:** such as the assumption that the number of clusters is $\sqrt{\frac{n}{2}}$ where n is the number of data points or the "elbow" method, which is based on observing the reduction in the sum of within cluster variance as the number of clusters increase and choosing the k value at the which the sum of within cluster variance forms an elbow

3. **Data Driven**: Suggests varying the number of clusters and for each choice fit the a model per cluster and collect the average performance on the validation set. Then choose the number of clusters that provides the best results as was suggested in [11, 23].

In our case, we have no valid assumption about the possible number of clusters that might be available in the dataset eliminating the possibility of using a heuristic or domain knowledge. Furthermore, the data driven approach used in [8-9] can be feasible only for the simple models the authors suggested such as Linear Regression, Support Vector Machine (SVM), and Multilayer Perceptron. However, for our deep learning models, performing the same steps would be impractical in the real world and very computationally expensive. In addition, using K-means restricts the analysis to using only the Euclidean distance to measure the distance between the examples, which can only allow the clustering algorithm to identify spherically shaped clusters [35].

Given the limitations of K-means in terms of requiring the number of clusters and the use of Euclidean distance. We suggest the use of an agglomerative hierarchical clustering algorithm that would allow us to use different distance metrics and alleviates the need for specifying the number of clusters.

4.1.2.1 Hierarchical Clustering

We explore the use of an agglomerative approach as opposed to a divisive approach for Hierarchical Clustering because a divisive approach faces the challenge of partitioning the large cluster into several smaller ones e.g., there are $2^{n-1}-1$ possibilities to partition a set of n objects into two exclusive subsets. Given that the exploration of all divisions' possibilities becomes computationally prohibitive as n grows, divisive approaches usually resort to heuristics that may result in inaccurate clustering [35].

To implement the agglomerative clustering, we need to find the best value for four different parameters:

- **Choice of Smart Meter Representation:** It is required to have a representation for each smart meter upon which we measure the distance between the different smart meters. As mentioned in section 4.1.1, there are four approaches to represent a smart meter, which are the daily load, average daily load, predicted daily load embedding, average predicted daily load embedding. In our analysis we avoid the use of an average representative load pattern because an average pattern might be skewed by outliers.

- **Distance Metric:** Different smart meter representations require the use of different distance metrics. For example, we can use the Euclidean distance to quantify the distance between the daily load representations, but for quantifying the distance between the different embeddings we use the cosine similarity or the Maximum Mean Discrepancy (MMD). The following are the different distance metrics:

$$\mathrm{Euc}\left(x^{t_1}, x^{t_2}\right) = \sqrt{\sum_{i=1}^{n}\left(x_i^{t_1} - x_i^{t_2}\right)^2} \tag{4.1}$$

$$\mathrm{MMD}\left(E^{t_1}, E^{t_2}\right) = \sum_{i=1}^{N}\frac{K\left(v_i^{t_1}, v_i^{t_1}\right)}{N^2} - 2\sum_{i=1}^{N}\frac{K\left(v_i^{t_1}, v_i^{t_2}\right)}{N^2} + \sum_{i=1}^{N}\frac{K\left(v_i^{t_2}, v_i^{t_2}\right)}{N^2} \tag{4.2}$$

$$K\left(v_i, v_j\right) = \sum_{n}\omega_n \exp\left\{-\frac{1}{2\sigma_{\mathrm{n}}}||\mathrm{v_i} - \mathrm{v_j}||^2\right\} \tag{4.3}$$

$$\text{Cos}\left(v^{t_1}, v^{t_2}\right) = \frac{\sum_{i=1}^{L} v_i^{t_1} \cdot v_i^{t_2}}{\sqrt{\sum_{i=1}^{L}\left(v_i^{t_1}\right)^2}\sqrt{\sum_{i=1}^{L}\left(v_i^{t_2}\right)^2}} \tag{4.4}$$

Where x^t: Daily loads vector for an example from task t; E^t : Task t embeddings set $E^t = \{v_1^t, v_2^t, \ldots, v_N^t\}$; v_i^t : Embedding for example i of task t; N : Number of examples; L: Embedding length; n: Daily load vector length; $K(.,.)$: Is a positive semi-definite kernel function; σ_n is the standard deviation; ω_n is the weight for n^{th} kernel. We use a weighted linear combination of RBF kernels that have different standard deviations as suggested in [11] to ensure that the MMD is sufficiently high when the distributions of the two smart meters are not similar. We explore the Maximum Mean Discrepancy (MMD) distance since it measures the distance between the mean of two datasets' distribution given the feature vectors representing the examples of these two datasets [36, 37]. The distance calculation steps between two smart meters are as follows:

1. Align the examples in time such that we are comparing same calendar days between the two smart meters. This is done to ensure that the distance captures the difference in behavior between the two smart meters
2. For the Cosine and Euclidean distances, we calculate the distance between each two examples for the smart meters and then take the average over all the examples' distances to obtain the average distance across the examples.
3. For the MMD, we take the resulting value directly as the distance.

- **Linkage Method:** The linkage method is the method used to calculate the distance between each two clusters. The resulting distance is then used by the agglomerative clustering algorithm to merge two clusters. There are four main linkage methods:

  1. **Single Linkage:** Merge clusters based on the minimum distance between their points.

  $$\text{dist}_{\min}\left(C_A, C_B\right) = \min_{p \in C_A, p' \in C_B} \{|p - p'|\} \tag{4.5}$$

  2. **Complete Linkage:** Merge clusters based on the maximum distance between their points.

  $$\text{dist}_{\max}\left(C_A, C_B\right) = \max_{p \in C_A, p' \in C_B} \{|p - p'|\} \tag{4.6}$$

3. **Average Linkage:** Merge clusters based on the average distance between their points.

$$\text{dist}_{avg}\left(C_A, C_B\right) = \frac{1}{n_i, n_j} \sum_{p \in C_A, p' \in C_B} |p - p'| \tag{4.7}$$

4. **Ward Linkage:** Merge two clusters such that the variance within the new cluster is minimized.

$$\begin{aligned} \text{dist}_{\text{ward}}\left(C_A, C_B\right) = & \sum_{i \in C_A \cup C_B} \left\|\vec{x}_i - \vec{m}_{C_A \cup C_B}\right\|^2 \\ & - \sum_{i \in C_A} \left\|\vec{x}_i - \vec{m}_{C_A}\right\|^2 \\ & - \sum_{i \in C_B} \left\|\vec{x}_i - \vec{m}_{C_B}\right\|^2 \end{aligned} \tag{4.8}$$

Where $\left|p - p^{'}\right|$: is the distance between two objects or points, p and $p^{'}$; n_i is the number of objects in cluster C; m_A: Center of cluster A.

The single and complete linkages tend to be sensitive to noisy data and outliers, whereas the average linkage provides a compromise between the two extremes overcoming the outlier sensitivity issue [35]. The ward linkage calculates the sum of squared intercluster variances across all clusters and then chooses to merge clusters that would minimize the loss [38]. We explore all four linkage methods in our search for the best clustering parameters.

- **Dendrogram Cutting Threshold:** Given the merge history of the linkage method, a dendrogram representing the hierarchical distance between the different clusters is built. We then need to specify a distance threshold below which we consider each line to be a cluster. We search over 5 values for the threshold $\in \{10\%, 20\%, 30\%, 40\%, 50\%\}$ and for each percentage we multiply the threshold value by the maximum distance in the dendrogram to obtain the actual threshold value.

4.1.3 Clustering Validation

To find the best combination of clustering parameters while being computationally efficient, we sample 50 smart meters from each type of smart meters in our dataset e.g., residential, and industrial smart meters. We split the data of each smart meter into train, validation, and test sets. Then for each type of smart meters, we iteratively perform the clustering on the training set using a combination of parameters. Moreover, we fit a population model and collect the model

performance on the validation set. The clustering parameter combination that achieves the least MAE and RMSE is chosen as the combination for that type of smart meters. The clustering parameters' search space includes the following sets:

- Smart Meter Representation $\in \{daily\ load,\ predicted\ daily\ load\ embedding\}$
- Distance Metric $\in \{Euclidean,\ Cosine,\ MMD\}$
- Linkage $\in \{Single,\ Complete,\ Ward,\ Average\}$
- Threshold $\in \{10\%,\ 20\%,\ 30\%,\ 40\%,\ 50\%\}$

Table 4.2 shows the clustering parameter combination that provided the best validation performance. To validate that the clustering improves the performance, we fit a population model per cluster of smart meters and compare its performance against fitting a separate population model per group of 10 smart meters that were randomly chosen to be on the same group and also compare against the performance of an STL model.

Table 4.2: The clustering parameters that provided the best RMSE and MAE for each group of 50 residential and industrial smart meters

Parameter Choice	**Residential**	**Industrial**
Smart Meter Representation	daily load	daily load
Distance Metric	Euclidean	Euclidean
Linkage	Ward	Average
Threshold	30%	30%

Tables 4.3 and 4.4 show that indeed the performance of the population model with clustering outperforms the STL model's performance by 2.15% and 3.4% in terms of RMSE and MAE for residential smart meters, respectively. And by 8.57% and 6.88% in terms of RMSE and MAE for industrial smart meters, respectively. We can also observe that the performance of the population approach with clustering provides more improvement over STL in comparison to random grouping.

Figs. 4.1 and 4.2 show the predicted vs. true value for the STL model, and the population model with and without clustering. From the figures we can see that training a population model with clustering provides better predictions that follows closely the actual load.

Figs. 4.3 and 4.4 show the dendrogram and distance matrix for 50 residential smart meters, respectively. The distance matrix is sorted according to the dendrogram to reflect the presence of the clusters, which could be seen by dark blue boxes along the diagonal. For example, we can see that we have three main clusters and a smart meter that is very dissimilar to all 49 remaining smart meters.

Table 4.3: The average RMSE and MAE over 50 residential smart meters for a population model with clustering, a population model with random grouping, and an STL model

Metric	STL	Population with clustering	Population with random grouping
RMSE	0.571	**0.562**	0.558
MAE	0.353	**0.341**	0.355
RMSE improvement % over STL	-	**2.15**	1.50
MAE improvement % over STL	-	**3.40**	-0.715

Table 4.4: The average RMSE and MAE over 50 industrial smart meters for a population model with clustering, a population model with random grouping, and an STL model

Metric	STL	Population with clustering	Population with random grouping
RMSE	1.342	**1.227**	1.281
MAE	0.868	**0.808**	0.852
RMSE improvement % over STL	-	**8.57**	4.54
MAE improvement % over STL	-	**6.88**	1.876

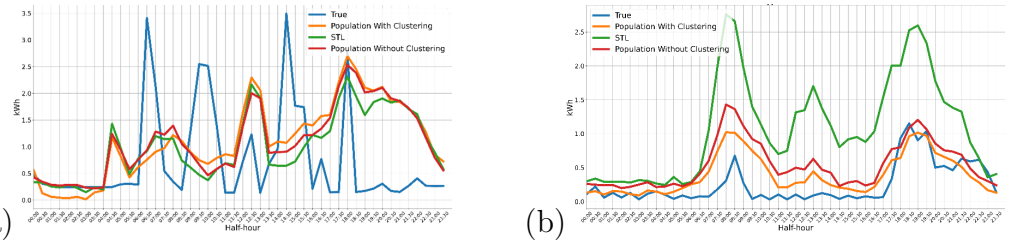


Figure 4.1: Predicted vs. True half-hourly load for two residential smart meters. a) Smart meter 1073. b) Smart meter 1243

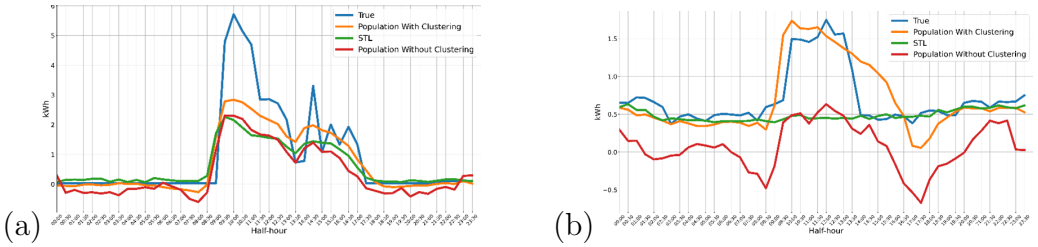


Figure 4.2: Predicted vs. True half-hourly load for two industrial smart meters. a) Smart meter 1356. b) Smart meter 1391

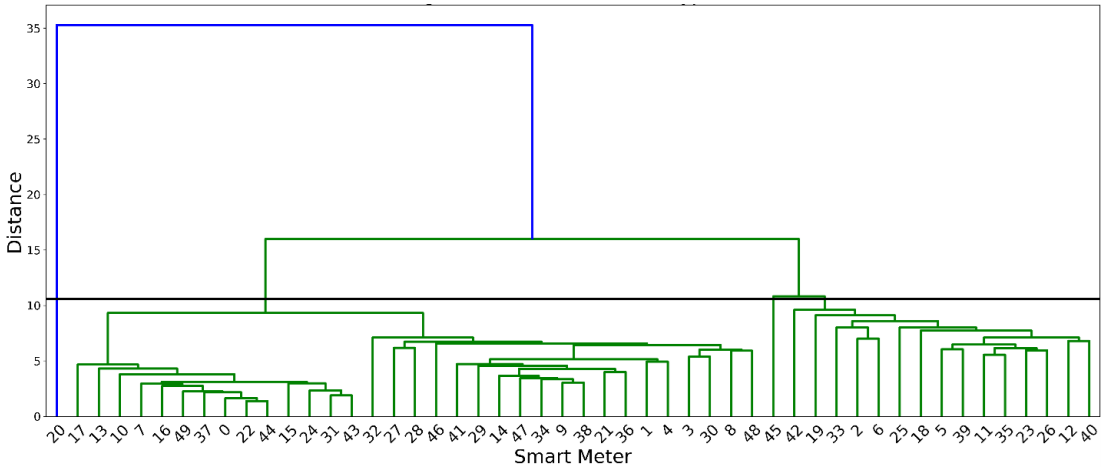


Figure 4.3: The dendrogram for the 50 residential smart meters with the black horizontal line representing the threshold below which we consider each line to be a cluster.

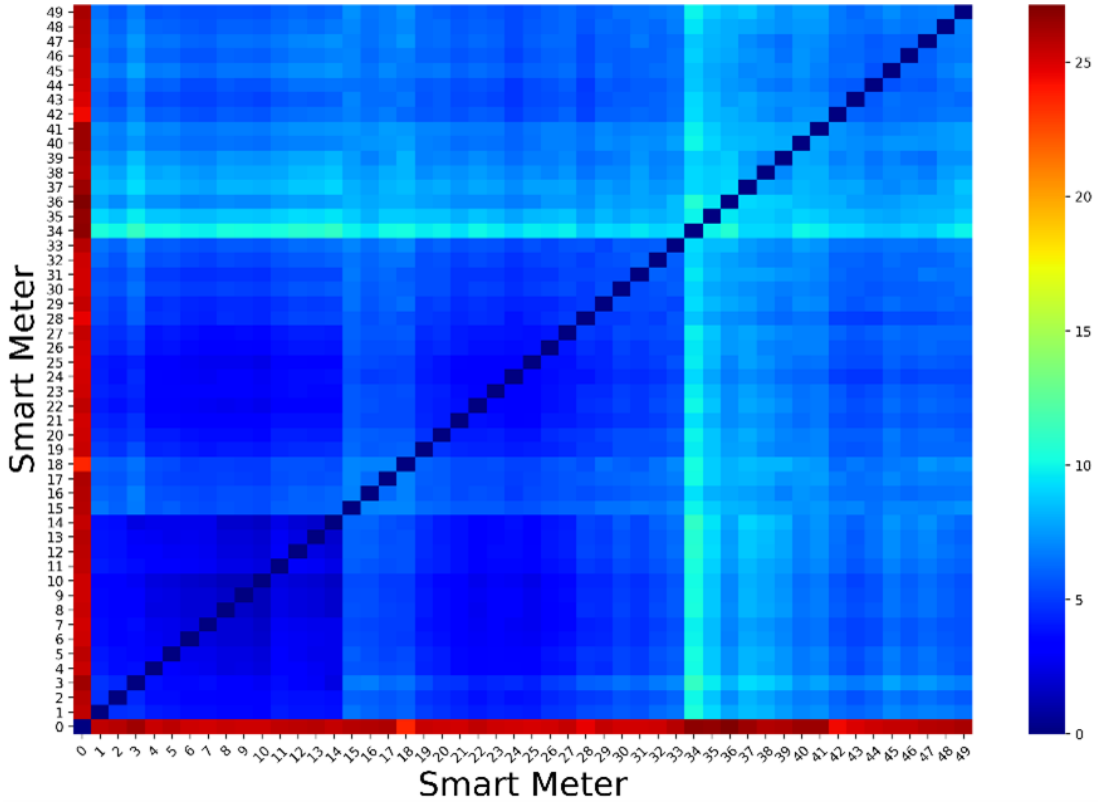


Figure 4.4: The distance matrix for the 50 residential smart meters. The colder the smaller the distance.

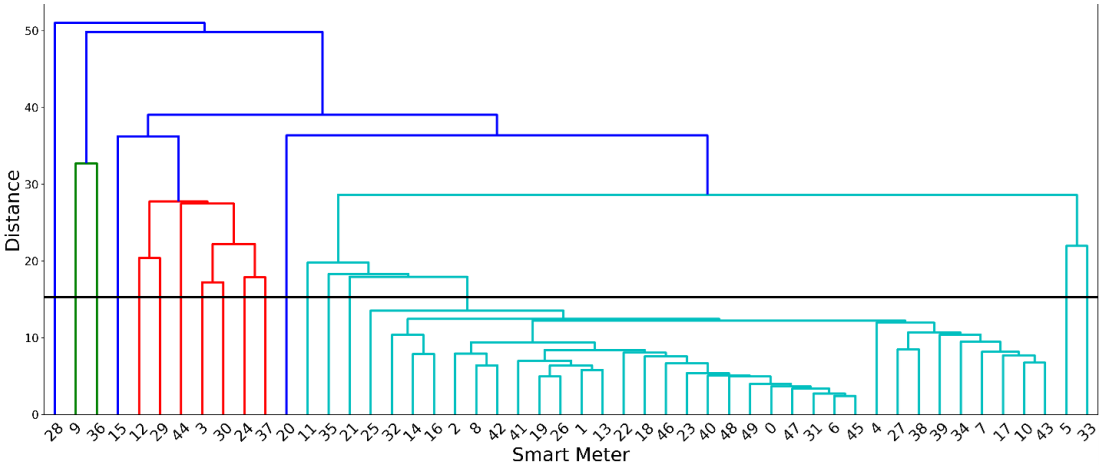


Figure 4.5: The dendrogram for the 50 industrial smart meters with the black horizontal line representing the threshold below which we consider each line to be a cluster.

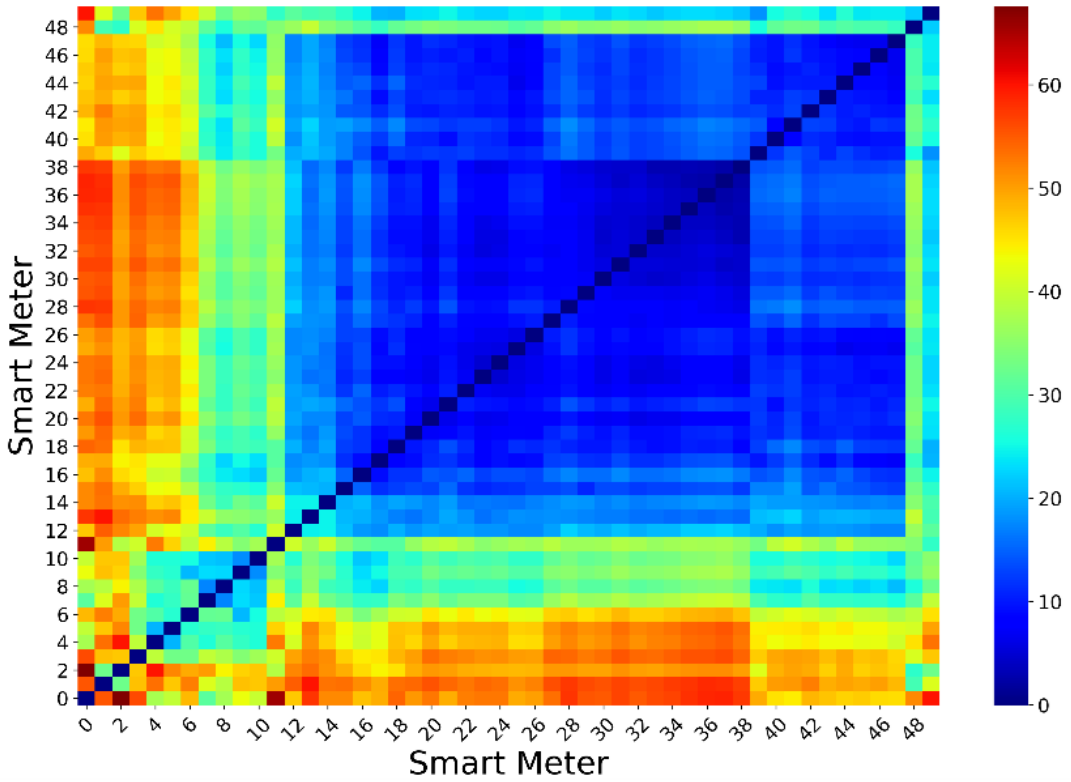


Figure 4.6: The distance matrix for the 50 industrial smart meters. The colder the smaller the distance.

Chapter 5

Methodology

5.1 Multitask Learning (MTL)

In this section we discuss the different MTL schemes that will be used in our proposed method.

5.1.1 Multitask Hard Parameter Sharing (MTL-HPS)

5.1.1.1 Model Architecture

The Multitask Hard Parameter Sharing (MTL-HPS) extends the STL CNN-GRU model by splitting the model into a shared part and a task specific part as shown in Fig. 5.2:

- **Shared Part:** This part gets trained by all the tasks' data. In other words, the parameters of the shared parts get updated by the gradients of all the tasks during backpropagation. The CNN layers represent the shared part.
- **Task Specific Part:** This part gets trained by the task-specific gradients during backpropagation, therefore, extracting task-specific representations. The GRU and dense layers represent the task-specific part.

In our MTL-HPS model we would have as many task-specific layers as there are tasks but only one shared part per group model. Furthermore, we consider each task to correspond to one smart-meter.

The MTL-HPS model is trained by a gradient descent algorithm with alternate batch training. In other words, at each iteration, a batch of examples for task t is forward propagated through the shared CNN layers and task t's GRU and dense layers to obtain a prediction. Then the loss for the examples is calculated and the weights of task t's GRU and dense layers and the shared CNN layers' weights are modified according to the calculated gradients during backpropagation. The previous process is repeated iteratively while at each time

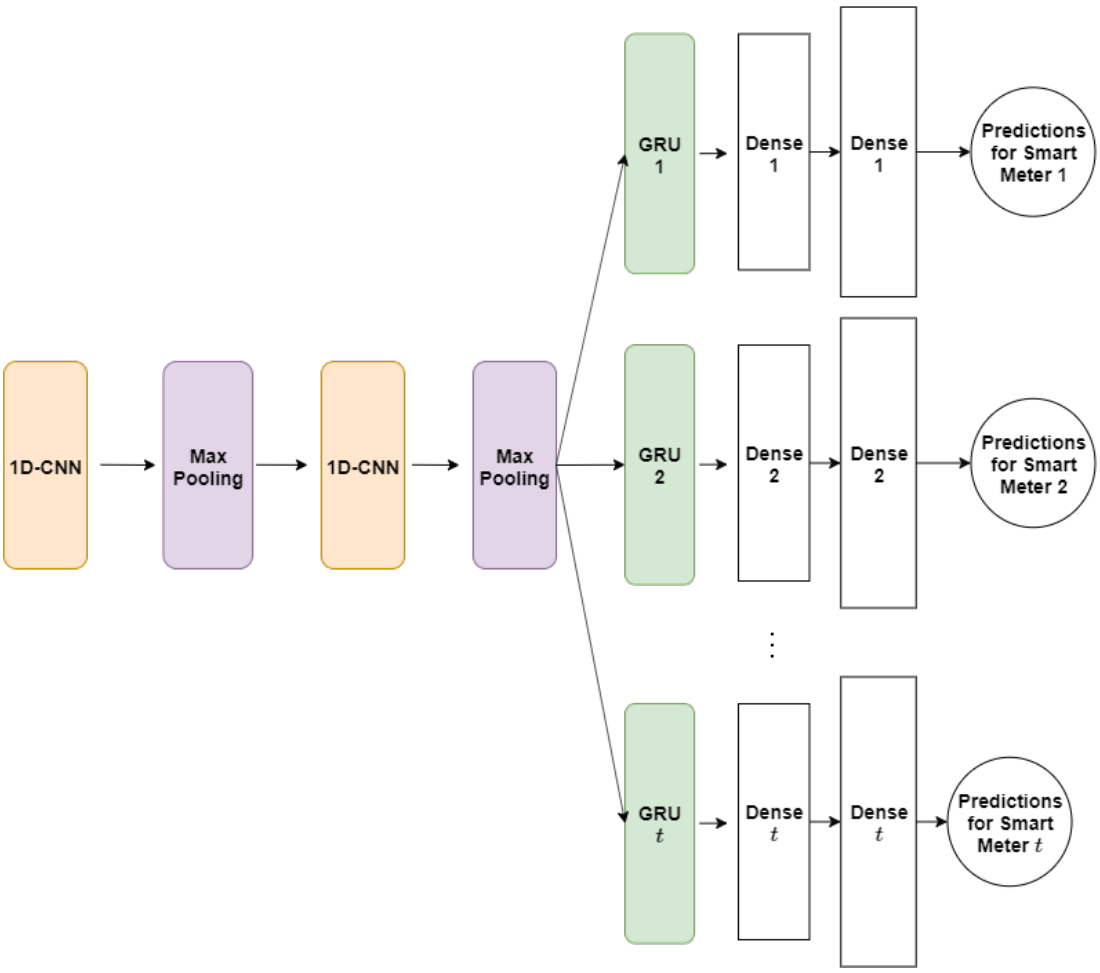


Figure 5.1: MTL Hard Parameter Sharing Architecture

taking a batch from a different task and modifying the weights of the shared and task-specific parts according to the loss's value.

5.1.1.2 Mathematical Formulation

Suppose we have (T) Smart Meters (SMs) in the dataset with (N) total datapoints. For an SM $t \in \{1, 2, 3, \ldots T\}$, the MTL-HPS formulation modifies on the Single Task formulation by: First, making Γ refer to the shared CNN Layers, Second, introducing Φ_t as the task-specific GRU layer, and $\mathrm{W}_t \in R^{H \times Win}$ the parameters of the output fully connected task-specific layer, where H is the dimensionality of the output vector from the GRU and Win is the output window size. The Mean Squared Error (MSE) denoted as L is the loss function that the model is trained to minimize:

$$L\left(f\left(x_i^t; \Gamma, W_t\right), y_i^t\right) = L\left(f\left(g\left(x_i^t; \Gamma\right); \Phi_t + W_t\right), y_i^t\right) \tag{5.1}$$

$$p_i^t = \left(\left(\Phi_t + W_t \right)^\top g\left(x_i^t; \Gamma \right) \right) \tag{5.2}$$

$$L\left(f\left(g\left(x_i^t; \Gamma \right); \Phi_t + W_t \right), y_i^t \right) = \sum_{t=1}^{T} \frac{1}{n_t} \sum_{i=1}^{n_t} \left(p_i^t - y_i^t \right)^2 \tag{5.3}$$

Where $g\left(.\right) : R^{Win \times C} \rightarrow R^{h_1 \times h_2}$ represents a function applied by the CNN feature extraction model-part on the input to obtain a shared embedding representation of the input of shape $(h_1 \times h_2)$. $f\left(.\right) : R^{h_1 \times h_2} \rightarrow R^{\mathrm{Win}}$ is a function applied by the task specific GRU and fully connected layers to obtain the prediction. p_i^t is the predicted value of power consumption for input point i and smart meter t. Eq. 5.3 shows the MSE loss that is calculated for the inputs of smart meter t.

The objective function is as follows:

$$\min_{W,\Phi,\Gamma} \sum_{t=1}^{T} \frac{1}{n_t} \sum_{i=1}^{n_t} L\left(f\left(X_i^t, \Gamma, \Phi_t, W_t, y_i^t \right) \right) \tag{5.4}$$

Although MTL-HPS has the main advantage of providing the most regularization for the learned tasks and is computationally efficient [32], it requires the tasks to be highly related e.g. are drawn from similar distributions. Otherwise, negative transfer of knowledge occurs resulting in degraded performance for the tasks.

5.1.2 Multitask Learning with Soft Parameter Sharing (MTL-SPS)

5.1.2.1 Model Architecture

MTL with Soft Parameter Sharing requires that each task has its own model with its separate parameters. Furthermore, the parameters of the different tasks' models get regularized to learn both what parameters should be shared between the different tasks and how much of each layer should be shared [32]. Therefore, enabling the model to extract how the different tasks interact with each other while reducing the effect of negative transfer of knowledge, since only the useful information between the different tasks is transferred. However, this approach tends to be more memory expensive than the MTL-HPS approach since each task requires a separate model.

5.1.2.2 Mathematical Formulation

We use the MTL-SPS formulation proposed in [25] to extend the STL objective in Eq. 2.6 into an MTL-SPS formulation. The MTL-SPS is achieved by adding

a regularization term to the objective function of each single task model. The regularization term constrains the weights of each layer of the single task model to be similar by the degree of similarity between the two tasks:

$$\begin{gathered}\min_{W,\Gamma\Omega} \frac{1}{n_t} \sum_{i=1}^{n_t} L\left(f\left(x_i^t; \Gamma_t, W_t\right), y_i^t\right) + \lambda \sum_{l=1}^{L} \operatorname{tr}\left(W_l \Omega_l^{-1} W_l^t\right) \\ \text{s.t. } \Omega_l \geqslant 0; \operatorname{tr}\left(\Omega_l\right) = 1\end{gathered} \tag{5.5}$$

Where W_l is the weights matrix for layer l of all the tasks $W_l \in R^{d \times T}$, T is the number of tasks and d is the dimension of the flattened layer's weights; Ω_l: is the similarity matrix between the layer l weights of all tasks $\Omega_l \in R^{T \times T}$; $tr(.)$ is the trace of a matrix.

First, Ω_l is initialized to the identity matrix normalized by the number of tasks $\left(\frac{1}{T} I_T\right)$. Second, during each training batch Ω_l is frozen and the loss is calculated, and after modifying the weights of each single task model by backpropagation, Ω_l is updated using the following equation as was found in [39]:

$$\Omega_l = \frac{\left(W_l^T W_l\right)^{1/2}}{\operatorname{tr}\left(\left(W_l^T W_l\right)^{\frac{1}{2}}\right)} \tag{5.6}$$

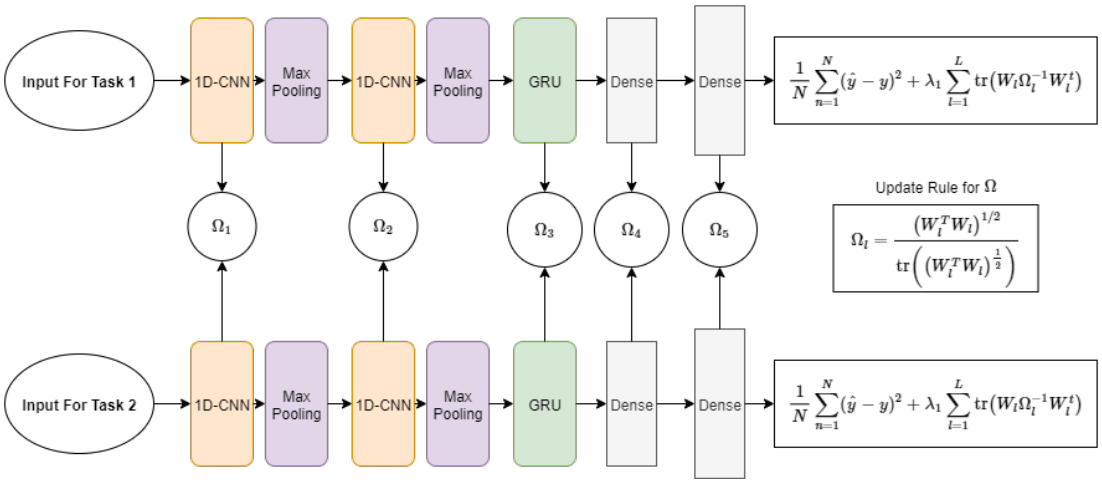


Figure 5.2: MTL Soft-Parameter Sharing Architecture

5.1.3 Proposed Hierarchical Clustering with MTL (HC-MTL)

We propose a hierarchical clustering with deep multitask learning (MTL) approach for industrial smart meters since it enables the CNN-GRU model to learn

the shared and smart-meter specific representation resulting in improved performance in terms of accuracy and generalizability. Our hierarchical MTL approach consists of two main steps:

1. Agglomerative Hierarchical Clustering
2. Cluster-specific MTL model

5.1.3.1 Agglomerative Hierarchical Clustering

As mentioned in section 4.1.2 we propose the use of an agglomerative hierarchical clustering approach since it does not require specifying the number of clusters beforehand and allows more flexibility in quantifying the distance between the different smart meters. We choose to search for the optimal clustering parameters by sampling 50 smart meter from each group in our dataset e.g., residential and industrial smart meters. For each group we perform the clustering according to the given set of parameters and then fit an MTL-HPS model per cluster of smart meters and record the validation set MAE and RMSE. We then choose the set of parameters that would achieve the least MAE and RMSE. The clustering parameters that achieved the best validation results are shown in Table 5.1.

Table 5.1: The clustering parameters that provided the best average RMSE and MAE validation set results over a group of 50 residential and industrial smart meters for the MTL-HPS model.

Metric	**Residential**	**Industrial**
Smart Meter Representation	Predicted daily load embedding	Predicted daily load embedding
Distance Metric	Cosine	Cosine
Linkage	Ward	Ward
Threshold	30%	30%

5.1.3.2 Cluster-specific MTL model

We explore two formulations to implement an MTL model per cluster, namely MTL with hard parameter sharing and MTL with soft parameter sharing.

5.1.3.2.1 Multitask with Hard Parameter Sharing (MTL-HPS)

As shown in section 5.1.1, this formulation divides the model into a shared part that gets optimized by all the tasks' gradients and a task-specific part that gets optimized by the tasks' specific gradients. It has the main advantage of providing the most regularization for the learned tasks [32]. However, it requires the tasks to be highly related e.g., are drawn from similar distributions otherwise

negative transfer of knowledge occurs resulting in degraded performance on the tasks [32]. Furthermore, MTL-HPS does not incorporate sharing partial information between the shared tasks' layers.

Since the MTL-HPS provides the best performance when the trained tasks are related, we use this model only for tasks that are clustered within the same cluster to ensure that no negative transfer of knowledge occurs.

5.1.3.2.2 Multitask with Soft Parameter Sharing (MTL-SPS)

MTL in a soft parameter sharing setting requires that each task has its own model with its separate parameters as shown in section 5.1.2. This formulation allows partial information transfer between the different task's layers and avoids negative transfer between the tasks in case unrelated tasks were trained jointly. However, this formulation is more computationally expensive during the training as it requires the extraction of a similarity matrix between the weights of two tasks' layers and updating the similarity matrix for each layer at each training batch.

We propose the use of this formulation for the smart meters that were identified not to belong to any cluster where we group each 20 smart meters and train them jointly in a separate MTL-SPS model. We propose the MTL-SPS formulation for the least similar smart meters because this formulation is capable of learning the amount of information to share between the different smart meters' models and at what layer the sharing should occur. Therefore, we hypothesize that this formulation could allow the model to improve the generalization by allowing the transfer of useful information only between the unrelated tasks.

5.2 Hierarchical Clustering with Population models (HC-P)

We propose a hierarchical clustering with population model per group (HC-P) for residential smart meters. The hierarchical clustering groups smart meters into groups of similar load behavior, which enables the population model to leverage existing correlations between the different smart meter loads to improve performance. The HC-P approach consists of two main steps:

1. Agglomerative Hierarchical Clustering
2. Cluster-specific population model

5.2.1 Agglomerative Hierarchical Clustering

As mentioned in section 4.1.2 we propose the use of an agglomerative hierarchical clustering approach since it does not require specifying the number of clusters

beforehand and allows more flexibility in quantifying the distance between the different smart meters. We choose the parameters in Table 4.2 for the residential and industrial smart meters as mentioned in section 4.1.3 .

5.2.2 Cluster-specific Population Model

A population model refers to developing a single model per group of smart meters or entire population of smart meters in the grid. We use the formulation proposed in [12] to extend the STL CNN-GRU model to a population model. The data for the population model is prepared by first assigning a one-hot encoded ID for each smart meter within the group of smart meters. During the training, we feed a batch of examples belonging to a certain smart meter within the group. After the batch is passed through the CNN-GRU layers for feature extraction, the resulting embedding is concatenated with the smart meter's ID. This new extended embedding is then fed to the fully connected layers to perform the prediction and the procedure is repeated for all the smart meters and examples. Fig. 5.3 illustrates the population model architecture and the mechanism to feed the data.

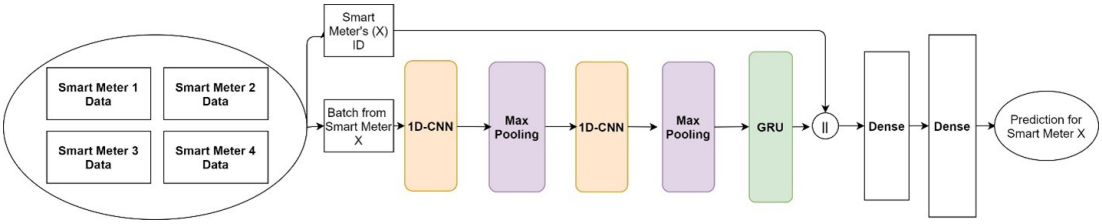


Figure 5.3: Population Model Architecture and Data insertion Procedure

Chapter 6

Experiments & Results

In this work, we target the problem of predicting the day ahead individualized energy consumption with half-hourly granularity. The model prediction is done in a rolling window manner, where after the prediction is conducted and the day passes, the input to the model is shifted one day ahead to include the new readings. The prediction method's output is useful in energy management system applications, such as optimal demand-side response.

6.1 Experimental Setup

Our models were implemented using Google's deep learning python library TensorFlow. All experiments were run on a PC equipped with an NVIDIA GTX 1080 GPU, an Intel Core i7-7700 (3.60 GHz) CPU and 32 GB of RAM.

6.2 Dataset

6.2.1 Dataset Description

For our experiments, we chose the dataset provided by the Irish Commission for Energy Regulation (ICER) [40]. The dataset contains electric energy measurements from 6435 smart meters with half-hourly granularity starting from July 14, 2009, until December 31, 2010 (536 days). We preprocess the data by first identifying and removing faulty meters, which are meters that have more than 50 measurements per day for multiple days. We also account for Daylight Saving Time (DST) changes by removing measurements that exceed 48 half-hours for days on which DST starts and forward filling the half-hours on the days on which there are only 46 measurements (DST ends) as was suggested in [19]. We Extract categorical calendar features (day-of-week, day-of-month, hour-of-day, month-of-year). The dataset also has the smart meters categorized into three groups as

shown in Table 8. In our experiments, we include only the residential and industrial smart meters that have measurements from 14/7/2009 to 31/12/2010, which reduces the number of smart meters to 4708.

Table 6.1: Number of smart meters per smart meter category after preprocessing

Smart Meter Type	Number of Smart Meters
Residential	4224
Industrial	484
Other	1641

6.2.2 Data Exploration

In this section we explore the characteristics of residential and industrial load.

6.2.2.1 Industrial Smart Meters

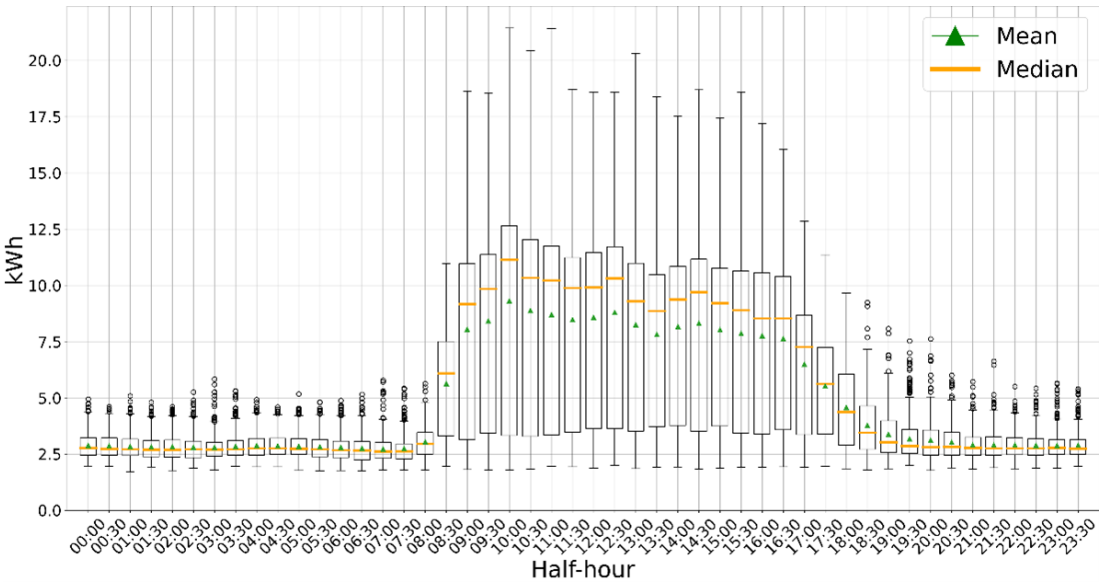


Figure 6.1: Hourly boxplot for an industrial smart meter

The individual industrial load can be characterized by regularity or even periodicity in comparison to individual residential loads. That is since a typical business have regular working days and hours. Fig. 6.1 shows that the load outside the working hours of smart meter 1028 is distributed within tight boxes indicating consumption regularity, while during working hours the boxplots seem to be more stretched depending on the workload but show little to no outliers. Figures 6.2 (a)-(d) show the load of industrial smart meters across the same day

(Tuesday) over two weeks (August 3rd 2010 and July 27th 2010) whose IDs are 1056, 3088, 5132, 7195, respectively. The figures show that periodicity exists for the same day across the weeks for industrial smart meters making the prediction task easier than residential load. On the other hand, there is still some industrial smart meters whose consumption is irregular as shown in Fig. 6.2 (d).

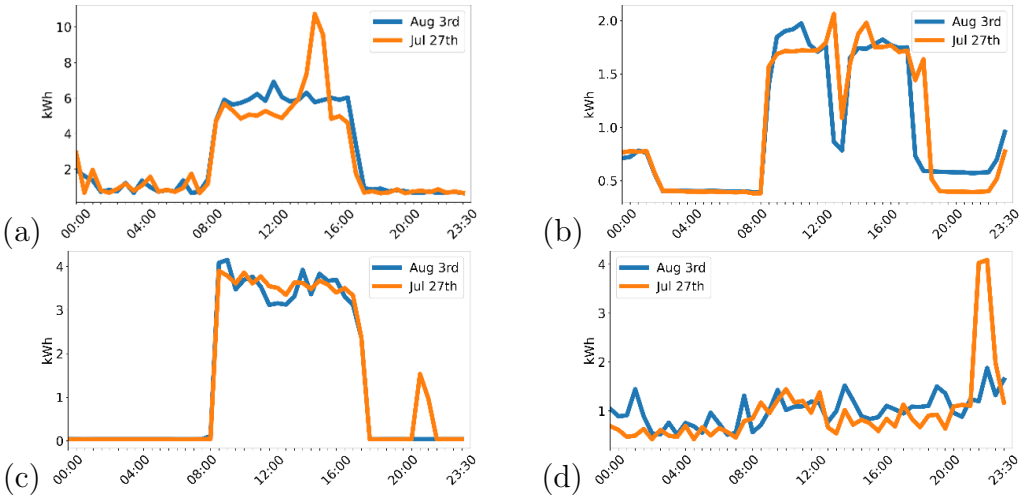


Figure 6.2: The load of a Tuesday across two weeks for industrial smart meters

6.2.2.2 Residential Smart Meters

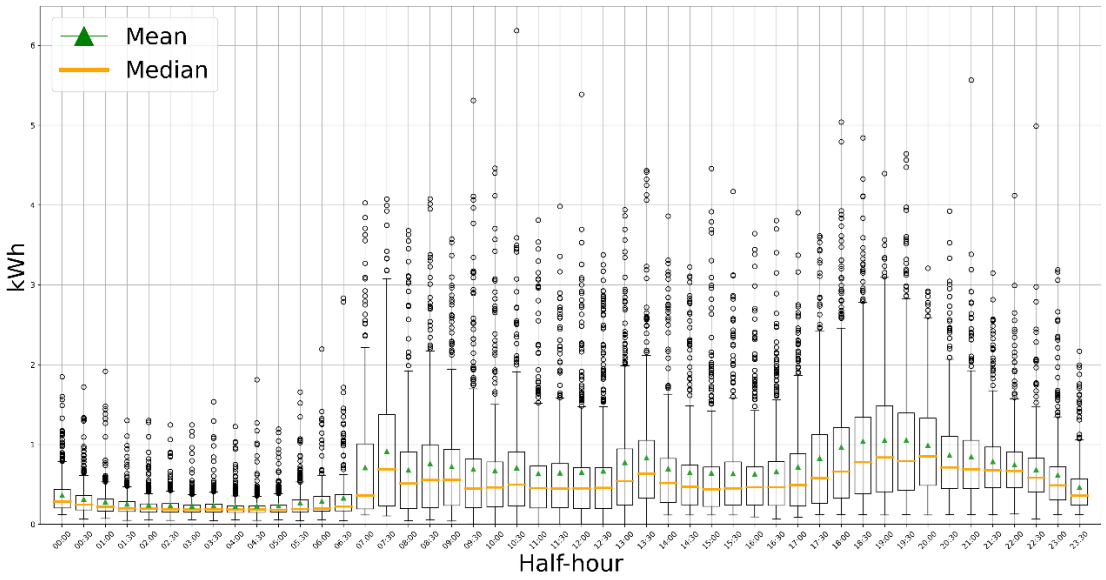


Figure 6.3: Hourly boxplot for a residential smart meter

Residential load at the individual level is characterized by high irregularity and heavy dependence on user behavior. Fig. 6.3 shows that during sleep hours the consumption tends to follow a tight range of values. However, over the day's hour, we can see that the load values have more outliers than the industrial load boxplot in Fig. 6.3 reflecting the irregularity in the residential load. Fig. 6.4 (a)-(d) shows residential load patterns for meters whose IDs are 1065, 1178, 1067, 1058, respectively. Fig. 6.4 (a) shows high peaks around 8:00-11:00, 16:00-20:00pm, 21:30-22:30 which can be due to breakfast and dinner preparations and an evening bath. Although the load pattern is relatively similar amongst the different days, differences in behavior still exist, which can also be seen in Fig. 6.4 (b). In addition, the residential load patterns can vary across different households or even the same households at different days. Fig. 6.4 (c) shows significant differences in the load pattern for the same day across two weeks. Fig. 6.4 (d) shows an overall low electricity consumption, which can be due to some appliances such as a fridge while the house residents are absent. In summary, individual residential loads are highly irregular and dependent on the user's lifestyle. Therefore, forecasting such pattern accurately at the individual level is very challenging given only historical load data.

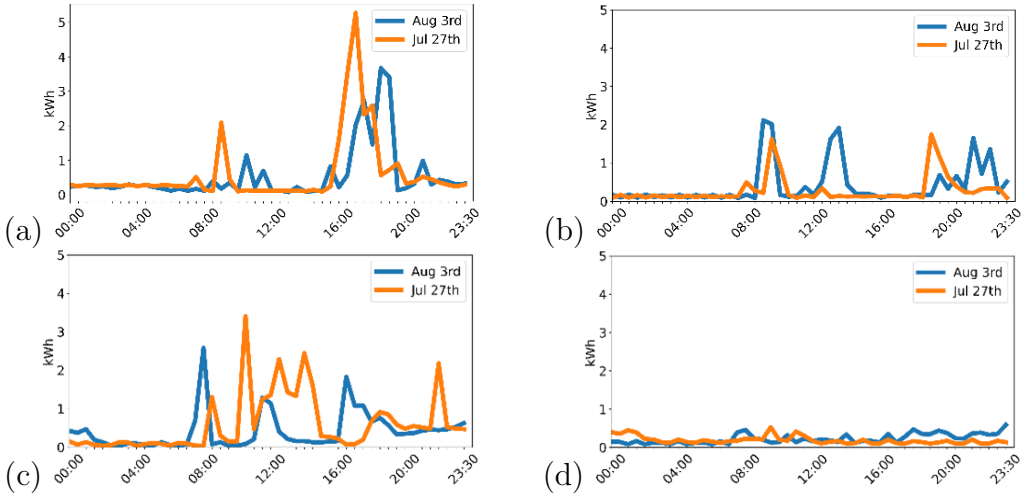


Figure 6.4: The load of a Tuesday across two weeks for residential smart meters

6.3 Model Input Span Selection

To find the optimal input span, we first adopted the architecture proposed in [1] while replacing the LSTM with a GRU as shown in Figure 2.3. The initial STL model consisted of the following hyperparameters:

- Each 1D CNN layer has 64 filters of size [2*1] and a stride of 1 with ReLU activation functions

- Each max-pooling filter has a size of [2*1] with a stride of 2
- The GRU layer has 64 cells with the default gate activations as in [2]
- The first dense layer has 32 neurons with no activation functions
- The last dense layer has 48 neurons with no activation functions

The network was trained for 75 epochs with a batch size of 32 and optimized using Adam optimizer [41] with a starting learning rate of 0.001. Given the previous model, we conducted an experiment to find the optimal input span. Where we varied the input span among the following choices:

- Previous [4, 5, 6] same days i.e., to predict a Monday the input is the previous X Mondays loads
- Previous [1, 2, 3] weeks

For each choice, the following calendar features were chosen:

- Hour of the day [1->48]
- Day of the week [1->7]
- The month of the year [1->12]
- Day of the month [1-> (31,30,28)]

The data was split sequentially into 70% of the days as training set, and 30% as test set. Then the last 20% of the training set days were used for validation. The dataset was normalized into values between 0 and 1 for each input feature using a Minmax scaler:

$$x_{\text{normalized}} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{6.1}$$

Where x is the feature vector. Note that the minimum and the maximum feature values were recorded on the training set and then were used to normalize the validation and test sets. For each input choice our single task model was trained and tested for all smart meters and the average individualized Root Mean Square Error (RMSE) and Mean-Absolute Error (MAE) were recorded as can be seen in Figure 6.5. We can see from the Figure 6.5 that the average RMSE and MAE values for the different input spans fluctuate around the same values. Because increasing the input size beyond 4 days increases complexity while not providing any significant improvements, we chose the input span of our models to be 4 days.

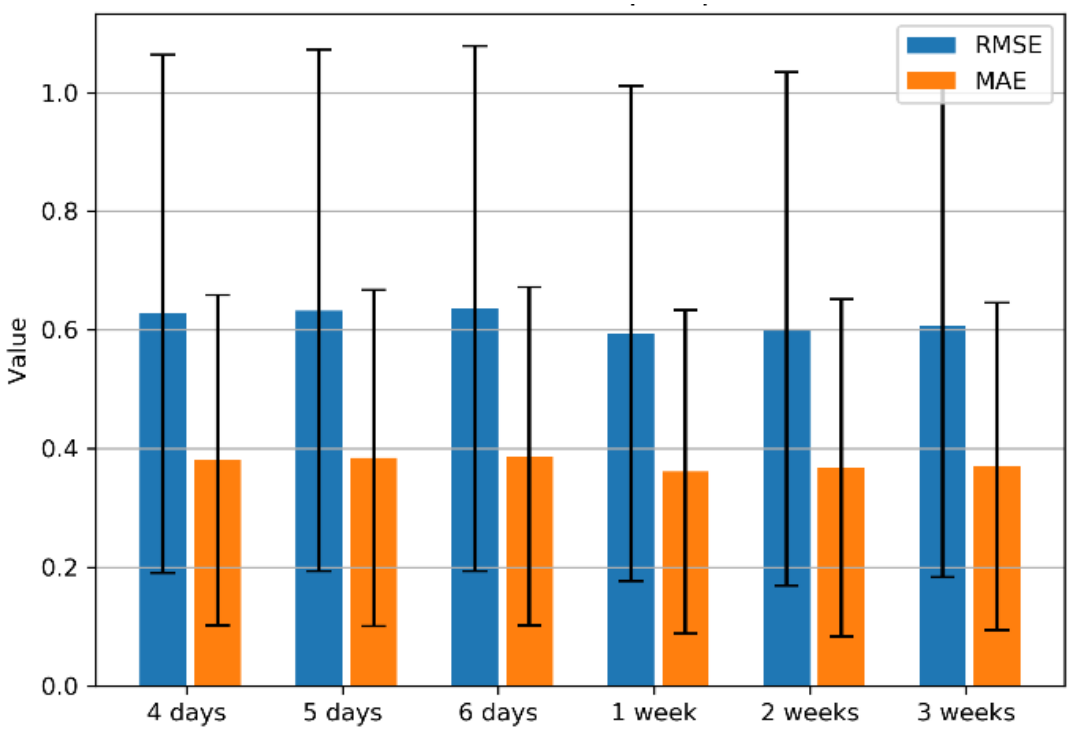


Figure 6.5: The average individualized MAE and RMSE value for the different

6.4 Tuning Parameters For The STL Model

To find the best model parameters for our dataset, 10 smart meters were chosen at random from each smart-meter category. For each of these 20 smart meters, a CNN-GRU model with the input span of 4 days was developed. Sixteen model architectures were explored for all 20 smart meters and the mean individualized RMSE and MAE were recorded for the validation set. Different combinations of the following parameters were explored:

- CNN layers $\in [1, 2, 3]$
- CNN kernel number $\in [32, 64, 128]$
- GRU layers $\in [1, 2]$
- GRU number of neurons $\in [32, 64, 128]$
- Number of Neurons for the 1^{st} Dense layer $\in$ [32, 64]

We chose the combination that provided the best average performance in terms of RMSE and MAE on the validation set while being the least computationally expensive. The following are the model parameters: 2 CNN layers, 1

GRU layer, the 1st CNN has 32 filters, 2nd CNN has 16 filters, the GRU has 32 cells and the 1st dense layer has 32 neurons.

After finding the best architecture hyperparameters, the number of epochs, learning rate and batch size were chosen by grid search over the following ranges of values:

- batch size ∈ [32, 64, 128]
- epochs ∈ [20, 30, 40, 50, 60, 70, 80]
- learning rate ∈ [0.0001, 0.0005, 0.001, 0.01, 0.02, 0.1]

The combination that achieved the best validation set average RMSE and MAE across all 20 smart meters was chosen as the model's training hyperparameters, which is: 60 epochs, 0.001 learning rate, batch size of 64

6.5 Tuning Parameters For The MTL Model

Fifty smart meters were chosen from each group of smart meters and for each group we tune the MTL model's hyperparameters along with the clustering hyperparameters and then pick the hyperparameters that would achieve the best performance on the validation set. We tune for the batch size, number of epochs, learning rate for both MTL-HPS and MTL-SPS approaches. For the MTL-HPS, we also search for the point at which the model layers are split into shared and task specific layers. For the MTL-SPS, we search for the value of the regularization parameter λ. The following are the ranges of searched parameters that were found by grid search:

- Batch size ∈ [16, 32, 64, 128]
- Epochs ∈ [20, 30, 40, 50, 60, 70, 80]
- Learning rate ∈ [0.0001, 0.0005, 0.001, 0.01, 0.02, 0.1]
- MTL-HPS model split-point ∈ :
    1. Splitting after the first Max Pooling layer
    2. Splitting after the 2nd Max Pooling layer
    3. Splitting after the GRU layer
- MTL-SPS λ ∈ [0.00001, 0.0001, 0.001, 0.01]

The combination that achieved the best validation set average RMSE and MAE across both groups of 50 smart meters was chosen as the model's training hyperparameters, which is: 60 epochs, 0.001 learning rate, batch size of 16, splitting after the GRU layer, and λ of value 0.0001

6.6 Tuning Parameters For The Population Model

Fifty smart meters were chosen from each group of smart meters and for each group we tune the population model's hyperparameters along with the clustering hyperparameters and then pick the hyperparameters that would achieve the best performance on the validation set. We tune for the batch size, number of epochs, learning rate. The following are the ranges of searched parameters that were found by grid search:

- Batch size $\in$ [16, 32, 64, 128]
- Epochs $\in$ [20, 30, 40, 50, 60, 70, 80]
- Learning rate $\in$ [0.0001, 0.0005, 0.001, 0.01, 0.02, 0.1]

The combination that achieved the best validation set average RMSE and MAE across both groups of 50 smart meters was chosen as the model's training hyperparameters, which is: 60 epochs, 0.001 learning rate, batch size of 64.

6.7 Tuning Parameters For The Baseline Statistical Models

We also compare against Auto Regressive Integrated Moving Average (ARIMA) and Seasonal ARIMA (SARIMA). We choose the input to the ARIMA and SARIMA model to be the previous 4 same days' load concatenated from old to new to form a vector of length ($n = 192$) similar to [18]. The ARIMA and SARIMA parameters for each test-set example were estimated using a grid-search over the following parameters:

- Order of the autoregressive model: $p \in [1, 2, \ldots, 40]$
- Order of the moving average model: $q \in [1, 2, \ldots, 40]$
- Order of Differencing (d) is increased by increments of one such that the stationarity is met according to the Augmented Dickey–Fuller test
- Order of the autoregressive portion of the seasonal model: $p_{\text{seasonal}} \in [1, 2, \ldots, 40]$
- Order of the moving average portion of the seasonal model: $q_{\text{seasonal}} \in [1, 2, \ldots, 40]$
- Order of Differencing (d_{seasonal}) increased by one such that the stationarity is met according to the Augmented Dickey–Fuller test

We perform a grid search over the previous parameter values per test-set example. The parameters that achieve the best Akaike Information Criterion (AIC) per example are chosen to fit the ARIMA/SARIMA model and produce a prediction for the next 48 half-hours.

6.8 Comparative Evaluation of Models

In this section we compare our proposed HC-MTL and HC-P models to the previous approahces includeing: state-of-the art STL model proposed in [1], hierarchical random clustering with population model as proposed in [12], a population model without grouping e.g., one model per all smart meters, baseline statistical models such as Auto Regressive Integrated Moving Average (ARIMA) and Seasonal ARIMA (SARIMA). The different approaches are assessed using the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE) as shown in equations 6.2 and 6.3, respectively.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{6.2}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{6.3}$$

Where y_i is the actual value and $\hat{y}_i$ is the forecasted value. The RMSE penalizes large errors, thus large values of RMSE indicate large errors. Whereas MAE measures the error on average. Ideally, we would like our approach to avoid large errors (RMSE) while providing low errors on average (MAE).

Table 6.2: The RMSE and MAE average over all residential smart meters for the different methods

Metric	STL [2]	HR-P [3]	HC-P	Population-All	HC-MTL	ARIMA	SARIMA
RMSE	0.535 ± 0.11	0.529 ± 0.106	**0.523 ± 0.108**	0.532 ± 0.105	0.538 ± 0.11	0.647 ± 0.65	0.654 ± 0.72
MAE	0.328 ± 0.075	0.329 ±0.074	**0.320 ±0.075**	0.326 ± 0.074	0.330 ± 0.076	0.403 ± 0.10	0.407 ± 0.11
RMSE improvement Vs. STL	0	1.12 %	**2.24%**	0.56%	-0.56%	-31.72	-33.711
MAE improvement Vs. STL	0	-0.30%	**2.43%**	0.60%	-0.60%	-22.86%	-24.08%

We can see from Table 6.2 that the proposed Hierarchical MTL method provides similar performance in terms of both metrics in comparison to an STL approach for residential smart meters. Whereas the HR-P provides minimal improvement over the STL model in terms of RMSE but provides no improvement in terms of MAE. Although random grouping improves the generalization of population model resulting in reduced RMSE, it does not provide any improvement on the average error (MAE). The HC-P, on the other hand, improved the population model's performance in terms of both RMSE and MAE by 2%.

Table 6.3: The RMSE and MAE average over all industrial smart meters for the different methods

Metric	STL [2]	HR-P [3]	HC-P	Population-All	HC-MTL	ARIMA	SARIMA
RMSE	1.184± 0.29	1.259± 0.26	1.202± 0.26	1.25 ± 0.26	**1.151± 0.26**	2.48 ± 3.61	2.52±3.61
MAE	0.724±0.19	0.817±0.17	0.782± 0.17	0.812 ± 0.17	**0.688±0.16**	1.293±0.46	1.30±0.45
RMSE improvement Vs. STL	0	-6.33%	-1.52%	-5.57%	**2.78%**	-109.45%	-113.26%
MAE improvement Vs. STL	0	-12.84%	-8.01%	-12.15%	**4.97%**	-78.59%	-79.55%

From Table 6.3, we can see that our proposed Hierarchical MTL approach provides an improvement of 2.78% in terms of RMSE and 5% in terms of MAE for industrial smart meters. Although the HC-P leveraged the clustering to improve the performance of population model's performance over random grouping, both methods tend to significantly provides worse performance than the STL model.

We also perform the Welch t-test [42] between each pair of models, for both the RMSE and MAE of all smart meters. Fig. 26 (a) and (b) show the p-values for each pair of models for both RMSE and MAE, respectively. By using a threshold of $\alpha = 10^{-2}$, most of the models are statistically different from each other.

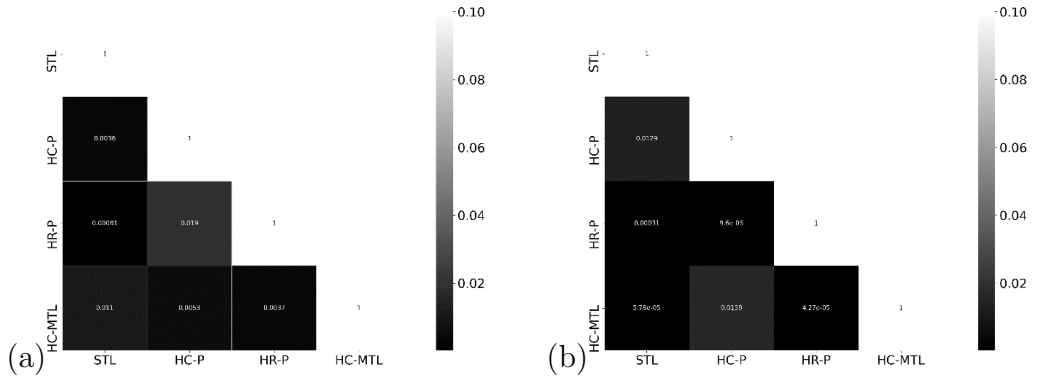


Figure 6.6: P-values of the Welch t-test between the overall metrics of all methods. a) The p-values for the MAE. b) The p-values for the RMSE

Figures 6.7, 6.8 and 6.9 show the prediction of the different methods for random days of three different residential smart meters. All figures show that the HR-P model with clustering tends to perform best in predicting the base load. In addition, all methods tend to miss predicting the sudden peaks in the load or mispredict it with a small shift in time.

From figure 6.9 we can see the amount of sudden peaks that characterize the residential behavior. Figures 6.10 and 6.11, show the predictions of the different methods for random days of two different industrial smart meters. Both figures demonstrate the effectiveness of the HC-MTL approach in predicting the daily load pattern. In addition, having the consumption occurring during the working hours demonstrates the regularity in the industrial load.

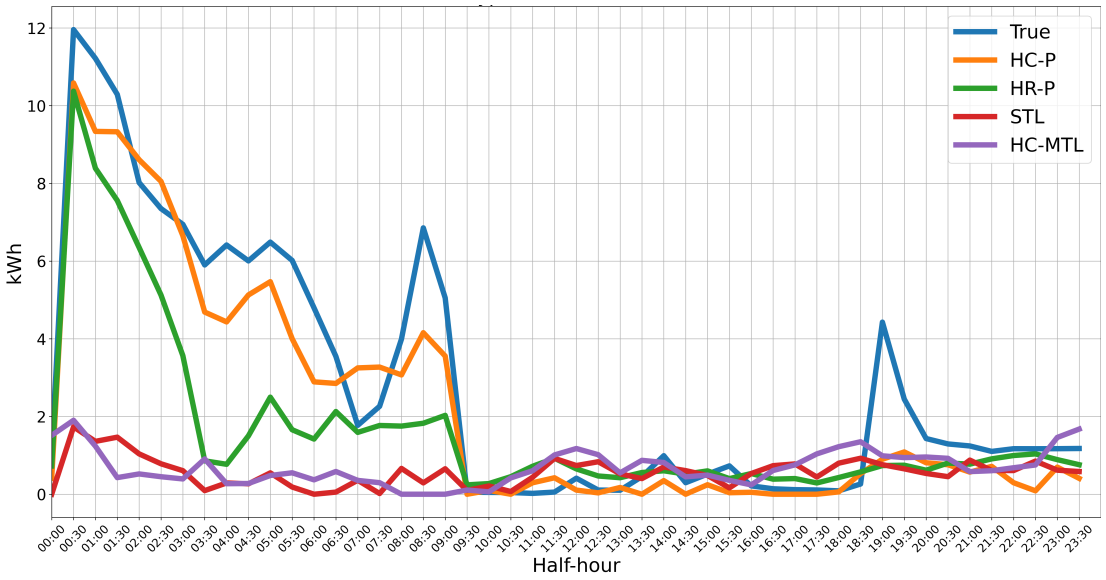


Figure 6.7: The prediction of the different methods for residential smart meter 1035 on 15/10/2010

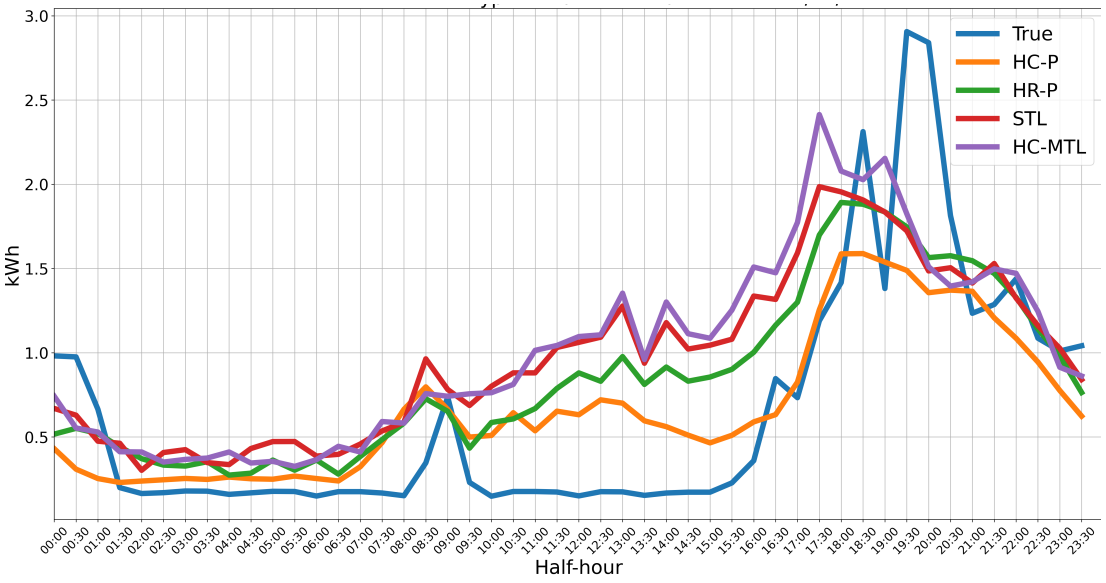


Figure 6.8: The prediction of the different methods for residential smart meter 1044 on 19/11/2010

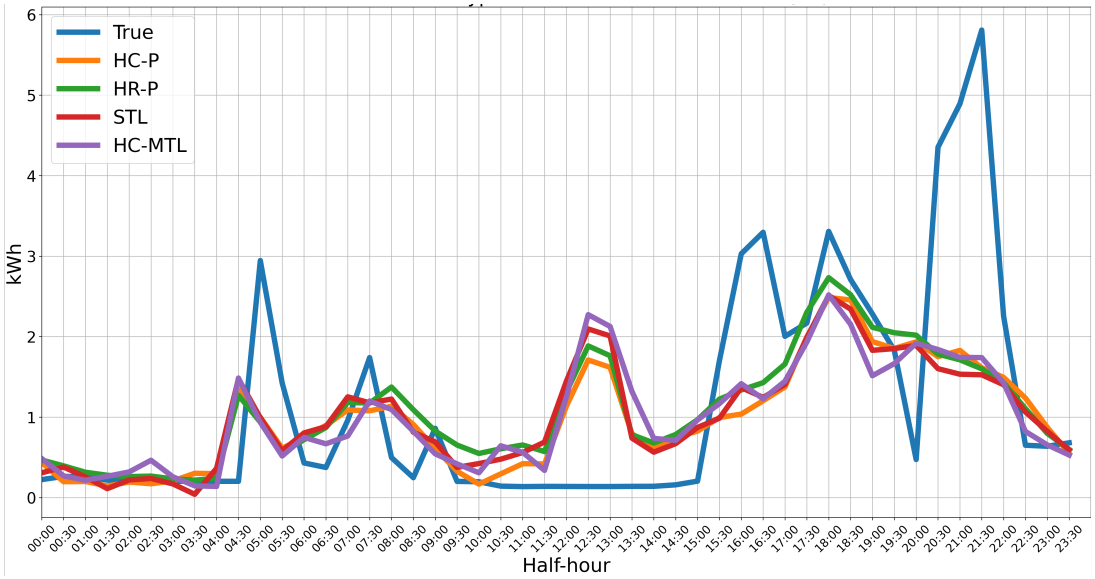


Figure 6.9: The prediction of the different methods for residential smart meter 1073 on 18/10/2010

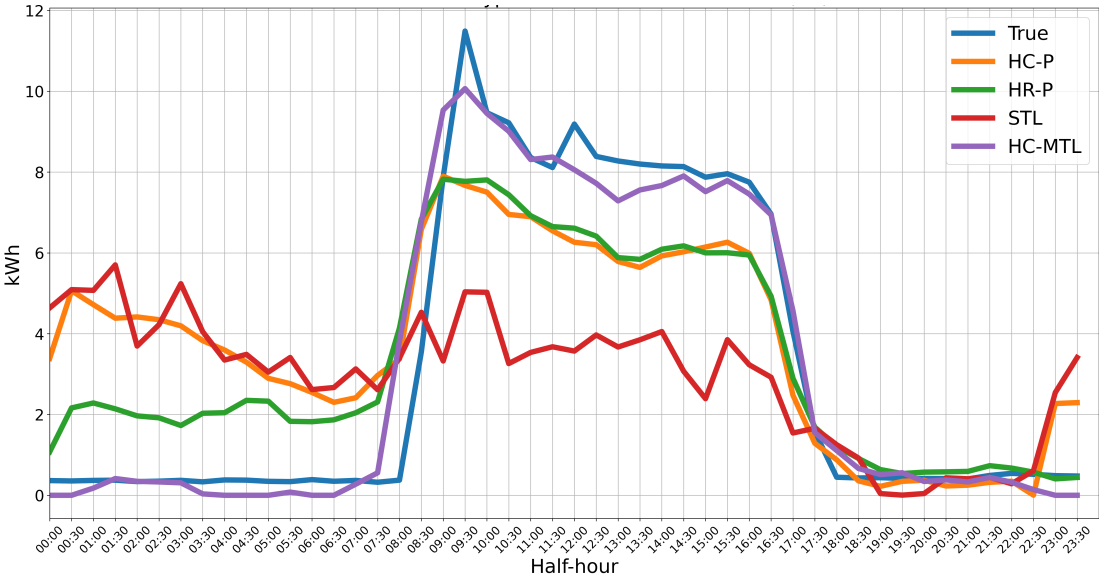


Figure 6.10: The prediction of the different methods for industrial smart meter 1146 on 30/07/2010

6.9 Discussion

6.9.1 STLF Results for Industrial and Residential Smart Meters

While examining the different models' performances on the residential smart meters, we can see that limited improvements are obtained from clustering ap-

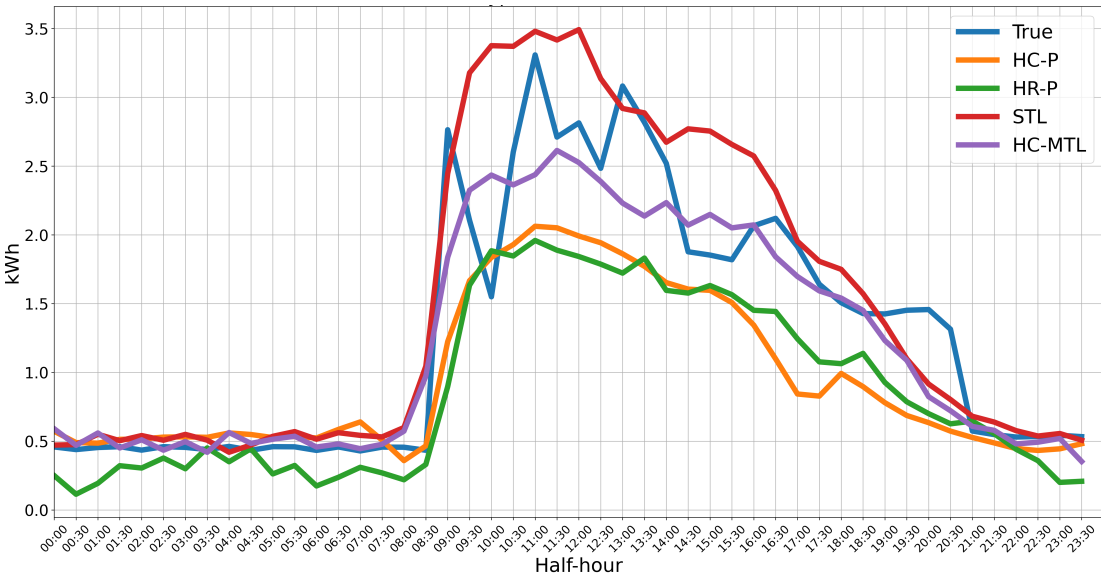


Figure 6.11: The prediction of the different methods for industrial smart meter 1181 on 26/07/2010

proaches. This is expected since the residential loads are highly irregular and dependent on the person's life-style. Furthermore, the residential behavior at a half hourly granularity shows more peaks as seen in Fig 6.9. and the time at which these peaks occur can drastically change from one week to another as shown in Fig 6.4 (a-c) making the forecasting process very challenging since a small change in the user behavior can cause the models to mispredict the peak by one time-step. MTL provides the best performance when a good balance between shared and task-specific information is present between the learned tasks. Otherwise negative transfer of knowledge occurs resulting in no improvement or even degraded performance [3].

Industrial loads, on the other hand, share more information in their behavior namely in working hours, days, but have unique patterns regarding the appliances they use. Therefore, the MTL approach was capable of capturing both shared and specific aspects resulting in improved model performance as seen in Table 6.3, whereas HC-P was incapable of learning the individualized load patterns.

6.9.2 Model Complexity and Real-Time Feasibility

Given that the STLF model is designed to be used in real life, it is required that it delivers predictions or be retrained if need be within 24 hours. We therefore report in Table 6.4 the sequential training time for the different methods on an Nvidia GTX 1080 GPU.

We see from Table 6.4 that the training time for the STL and HC-P is close whereas the HC-MTL approach requires the most training time and is 3x more

Table 6.4: The Training Time in Hours for the different approaches

	STL	**HC-P**	**HC-MTL**
Training Time in Hours	5.68	8.08	18.5

than the STL training time. Although the proposed approaches require more training time in comparison to STL, they both are within the 24 hours limit, making them applicable in real life. It is also important to note that the per smart meter or cluster of smart meters model, is being run sequentially on one GPU, in practice however, it is possible to re-train these models on multiple GPUs in parallel to further reduce the training time to meet the business demand.

Chapter 7

Conclusion

Accurate short-term load forecasting has always been a fundamental requirement for the application of demand response policies. Prior State-of-the-art deep learning approaches' accuracy can be improved by combining data from similar smart meters. In this thesis, we provide two studies exploring the data sufficiency and transfer learning capabilities of prior work. Our data sufficiency study validated that adding more training data from the same smart meters improved the accuracy up to a certain saturation point after which limited performance gains were obtained. The transfer learning feasibility study showed that adding more data from similar smart meters improves the performance for the smart meters with insufficient training data. Both experiments have shown that industrial smart meters benefit more from additional training data in comparison to residential smart meters. We then propose two novel transfer learning approaches for improving STLF accuracy:

- Hierarchical Clustering with Population models (HC-P): Based on numerical testing, we found HC-P to be most suitable for residential smart meters.

- Hierarchical Clustering with deep Multitask Learning (HC-MTL): Based on numerical testing, we found HC-MTL to be most suitable for industrial smart meters.

Since residential loads are user-specific and more irregular, limited performance gains were obtained from jointly training them. Industrial loads, on the other hand, are more regular and share similar general behavior while retaining individualized differences related to the appliances they use. The industrial load's characteristics allow for more benefit from transfer learning using HC-MTL. The HC-P approach improves the RMSE and MAE over prior state of the art by 2% on average for 4224 residential smart meters. The HC-MTL provides an average improvement of 2.78% and 4.97% in terms of RMSE and MAE, respectively, over 484 industrial smart meters.

Although our work provides insights and a proof of the benefit of transfer learning for STLF accuracy using deep learning models. Few questions remain unexplored in this work including:

- How to determine smart meters with insufficient training data? In this work we showed that smart meters with insufficient training data tend to benefit from more data from similar smart meters. However, no metric or method was provided to detect such smart meters.
- How to detect the training data size beyond which limited performance gains are obtained per smart meter? In this work we discovered that adding more data from the same smart meter improves the performance up to a certain saturation point beyond which no significant improvements are gained. However, we don't provide a metric or method to quantify the training data sufficiency for a smart meter.
- What is the best clustering strategy for accurate STLF? In this work we found that hierarchical clustering with an appropriate distance metric provides improvements for STLF. However, we did not compare our proposed clustering to other possible clustering algorithms such as: K-means or DB Scan.

The previous questions remain as possible directions to explore for future work.

Bibliography

[1] T.-Y. Kim and S.-B. Cho, "Predicting residential energy consumption using cnn-lstm neural networks," *Energy*, vol. 182, pp. 72–81, 2019.

[2] R. R. Mohassel, A. Fung, F. Mohammadi, and K. Raahemifar, "A survey on advanced metering infrastructure," *International Journal of Electrical Power & Energy Systems*, vol. 63, pp. 473–484, 2014.

[3] P. Warren, "A review of demand-side management policy in the uk," *Renewable and Sustainable Energy Reviews*, vol. 29, pp. 941–951, 2014.

[4] U. E. I. Administration, "U.s. energy information administration - eia - independent statistics and analysis," Oct 2020.

[5]

[6] A. Garulli, S. Paoletti, and A. Vicino, "Models and techniques for electric load forecasting in the presence of demand response," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 3, pp. 1087–1097, 2014.

[7] P. Samadi, H. Mohsenian-Rad, R. Schober, and V. W. Wong, "Advanced demand side management for the future smart grid using mechanism design," *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1170–1180, 2012.

[8] S. Aman, M. Frincu, C. Chelmis, M. Noor, Y. Simmhan, and V. K. Prasanna, "Prediction models for dynamic demand response: Requirements, challenges, and insights," in *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 338–343, IEEE, 2015.

[9] M. Chaouch, "Clustering-based improvement of nonparametric functional time series forecasting: Application to intra-day household-level load curves," *IEEE Transactions on Smart Grid*, vol. 5, no. 1, pp. 411–419, 2013.

[10] A. Veit, C. Goebel, R. Tidke, C. Doblander, and H.-A. Jacobsen, "Household electricity demand forecasting: benchmarking state-of-the-art methods," in *Proceedings of the 5th international conference on Future energy systems*, pp. 233–234, 2014.

[11] S. Humeau, T. K. Wijaya, M. Vasirani, and K. Aberer, "Electricity load forecasting for residential customers: Exploiting aggregation and correlation between households," in *2013 Sustainable Internet and ICT for Sustainability (SustainIT)*, pp. 1–6, IEEE, 2013.

[12] H. Shi, M. Xu, and R. Li, "Deep learning for household load forecasting—a novel pooling deep rnn," *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 5271–5280, 2017.

[13] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long-and short-term temporal patterns with deep neural networks," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 95–104, 2018.

[14] L. Han, Y. Peng, Y. Li, B. Yong, Q. Zhou, and L. Shu, "Enhanced deep networks for short-term and medium-term load forecasting," *IEEE Access*, vol. 7, pp. 4045–4055, 2018.

[15] M. Gilanifar, H. Wang, L. M. K. Sriram, E. E. Ozguven, and R. Arghandeh, "Multi-task bayesian spatiotemporal gaussian processes for short-term load forecasting," *IEEE Transactions on Industrial Electronics*, 2019.

[16] J.-B. Fiot and F. Dinuzzo, "Electricity demand forecasting by multi-task learning," *IEEE Transactions on Smart Grid*, vol. 9, no. 2, pp. 544–551, 2016.

[17] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[18] A. Marinescu, C. Harris, I. Dusparic, S. Clarke, and V. Cahill, "Residential electrical demand forecasting in very small scale: An evaluation of forecasting methods," in *2013 2nd International Workshop on Software Engineering Challenges for the Smart Grid (SE4SG)*, pp. 25–32, IEEE, 2013.

[19] D. L. Marino, K. Amarasinghe, and M. Manic, "Building energy load forecasting using deep neural networks," in *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pp. 7046–7051, IEEE, 2016.

[20] 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*, 2017.

[21] K. Amarasinghe, D. L. Marino, and M. Manic, "Deep neural networks for energy load forecasting," in *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, pp. 1483–1488, IEEE, 2017.

[22] J.-B. Fiot and F. Dinuzzo, "Electricity demand forecasting by multi-task learning," *IEEE Transactions on Smart Grid*, vol. 9, no. 2, pp. 544–551, 2018.

[23] F. L. Quilumba, W.-J. Lee, H. Huang, D. Y. Wang, and R. L. Szabados, "Using smart meter data to improve the accuracy of intraday load forecasting considering customer behavior similarities," *IEEE Transactions on Smart Grid*, vol. 6, no. 2, pp. 911–918, 2014.

[24] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[25] M. Qiu, P. Zhao, K. Zhang, J. Huang, X. Shi, X. Wang, and W. Chu, "A short-term rainfall prediction model using multi-task convolutional neural networks," in *2017 IEEE International Conference on Data Mining (ICDM)*, pp. 395–404, IEEE, 2017.

[26] Y. Zhang, G. Luo, and F. Pu, "Power load forecasting based on multi-task gaussian process," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 3651–3656, 2014.

[27] X. Liu, P. He, W. Chen, and J. Gao, "Multi-task deep neural networks for natural language understanding," *arXiv preprint arXiv:1901.11504*, 2019.

[28] Z. Yang, R. Salakhutdinov, and W. W. Cohen, "Transfer learning for sequence tagging with hierarchical recurrent networks," *arXiv preprint arXiv:1703.06345*, 2017.

[29] C.-W. Huang, C.-T. Chiang, and Q. Li, "A study of deep learning networks on mobile traffic forecasting," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1–6, IEEE, 2017.

[30] D. Lopez-Martinez, O. Rudovic, and R. Picard, "Physiological and behavioral profiling for nociceptive pain estimation using personalized multitask learning," *arXiv preprint arXiv:1711.04036*, 2017.

[31] W. Huang, G. Song, H. Hong, and K. Xie, "Deep architecture for traffic flow prediction: deep belief networks with multitask learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 5, pp. 2191–2201, 2014.

[32] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard, "Learning what to share between loosely related tasks," *arXiv preprint arXiv:1705.08142*, 2017.

[33] A. S. Debs, *Modern power systems control and operation*. Springer Science & Business Media, 2012.

[34] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[35] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[36] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain separation networks," *arXiv preprint arXiv:1608.06019*, 2016.

[37] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton, "Demystifying mmd gans," *arXiv preprint arXiv:1801.01401*, 2018.

[38] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, "Hierarchical clustering," *Cluster analysis*, vol. 5, pp. 71–110, 2011.

[39] Y. Zhang and D.-Y. Yeung, "A convex formulation for learning task relationships in multi-task learning," *arXiv preprint arXiv:1203.3536*, 2012.

[40] "Home, irish social science data archive."

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[42] B. L. Welch, "The generalization ofstudent's' problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1/2, pp. 28–35, 1947.