Universitatea de Stat din Moldova

Secţia Managementul Calităţii, Dezvoltare Curriculară şi Evaluare

VLADIMIR BABUSHKIN

DEVELOPMENT OF AUTOMATIC LEARNING ALGORITHMS FOR PRICE AND LOAD PREDICTION IN ELECTRICAL NETWORK

TEZELOR DE LICENŢĂ/MASTER

CHIŞINĂU, 2020

CZU

Aprobat de Consiliul Calităţii al Universităţii de Stat din Moldova

TABLE OF CONTENTS

LIST OF ABBREVIATIONS **4**

ABSTRACT **6**

INTRODUCTION **7**

IMPORTANCE OF THE STUDY *7*
RESEARCH METHODS *8*
NOVELTY OF THE STUDY *9*
KEYWORDS *10*

CHAPTER 1. LITERATURE REVIEW **11**

CHAPTER 2. THEORETICAL BACKGROUND **22**

PROBLEM DEFINITION *22*
EXPERT ALGORITHMS *23*
Autoregressive Integrated Moving Average (ARIMA) 23
k-means 25
K-medoids 26
Agglomerative clustering 27
Fuzzy C-means clustering 28
Self-organizing map (SOM) 29
PATTERN SEQUENCE-BASED FORECASTING ALGORITHM *31*
DEEP LEARNING NETWORKS FOR TIME SERIES PREDICTION *33*
Convolutional Neural Networks (CNNs) 37
Long-Short Term Memory Networks 38
PERFORMANCE METRICS *41*
Mean Absolute Percentage Error (MAPE) 41
Mean Relative Error (MRE) 42
Mean Absolute Error (MAE) 42
Root Mean Square Error (RMSE): 42
ELECTRICAL LOAD FORECASTING WITH LSTM AND CNN NETWORKS *43*
DEEP CNN FOR ELECTRICITY PRICE PREDICTION *44*

CHAPTER 3. RESULTS **47**

EVALUATING PERFORMANCE OF CONVENTIONAL STATISTICAL METHODS *47*
SEARCH FOR OPTIMAL PARAMETERS *48*
ANALYSIS OF CLUSTERING PATTERNS *52*
EVALUATING PERFORMANCE OF LOAD PREDICTION ALGORITHMS *59*
FORECASTING DURING PANDEMICS *67*
PREDICTING ELECTRICITY PRICE *70*
PREDICTING ELECTRICITY PRICE FROM LOAD *76*

CONCLUSION AND RECOMMENDATIONS **80**

BIBLIOGRAPHY **82**

APPENDIX 1 ..... **87**

APPENDIX 2 ..... **88**

LIST OF ABBREVIATIONS

AEMO – Australian Energy Market Operator

ANN – Artificial Neural Networks

API – Application Programming Interface

ARIMA – AutoRegressive Integrated Moving Average

BMU – Best Matching Unit

CNN – Convolutional Neural Network

CNTK – Microsoft Cognitive Toolkit

DBN – Deep Learning Believe Network

ES – Exact Sequence

EV – Electric Vehicle

FCRBM – Factored Conditional Restricted Boltzmann Machine

GAN – Generative Adversarial Networks

GARCH – Generalized Autoregressive Conditional Heteroskedasticity

IPP – Independent Power Producers

IQR – Interquartile Range

LBF – Label-Based Forecasting

LBMP – Location Based Marginal Pricing

LSTM – Long-Short Term Memory

MA – Moving Average

MAE – Mean Absolute Error

MAPE – Mean Absolute Percentage Error

MRE – Mean Relative Error

MTLF and LTLF – Medium-Term and Long-Term Load Forecasting

NYISO – New York Independent System Operator

OLS – Ordinary Least Squares

PSF – Pattern Sequence-based Forecasting algorithm

RBM – Restricted Boltzmann Machine

RBFNN – Radial Basis Function Neural Network

ReLU – Rectified Linear Unit

RMSE – Root Mean Square Error

SOM – Self- Organizing Maps

STLF – Short-Term Load Forecasting

SVR – Support Vector Regression

TSF – Time Series Forecasting

VSTLF – Very Short-Term Load Forecasting

ABSTRACT

Nowadays, the electricity market is becoming more diverse and deregulated due to the gradual introduction of smart grids. A smart grid analyzes information about electricity supply and demand to optimize the electric power production. This requires the knowledge of future electricity load that allow the Independent Power Producers to schedule the electric power generation to meet the peak demand and minimize the production costs. The IPPs can also implement the dynamic pricing to incentivize the end-users shifting the electricity consumption to off-peak hours. In other word, consumers, knowing the future electricity prices, can adjust their consumption profile to achieve an optimal utilization of electric power at the lowest cost. Other entities of the electricity market e.g. retailers and traders are also interested in electricity price prediction to optimize their financial operations. Thus, the accurate forecasting of price and load is essential for maintaining a stable interplay between demand and supply in the dynamic electricity market.

This work aims to investigate conventional technics for time series prediction and to compare them with a state-of-the-art deep learning networks. The later are capable of inferring hidden temporal dependences and trends as well as intrinsic factors, affecting the dynamicity of electricity load and price time series. We explore the performance of automatic learning approaches for predicting power balance in Moldova as well as electricity load and price for New York City and New South Wales. Next, we evaluated the performance of conventional methods and deep learning based approaches in the situation, when electricity consumption patterns change unexpectedly due to pandemics. Finally, we propose a novel deep learning approach for day-ahead electricity price forecasting from historical price/load data and the predicted day-ahead load value. The proposed system can be implemented in smart grid settings for online and offline electricity price prediction for forecasting horizons of different lengths, ranging from a single day to weeks and months.

INTRODUCTION

IMPORTANCE OF THE STUDY

Accurate forecasting of the electrical energy demand is crucial for electricity production, since electrical energy belongs to those commodities, that cannot be fully stored, and needs to be consumed as soon as it is produced. Inefficient production and distribution of electricity incurs additional costs. The accurate prediction of electrical energy demand allows not only to avoid these costs by scheduling the load production but also to manipulate with electricity price to achieve desired levels of consumption and maximize the profit. Moreover, the knowledge of expected demand allows more efficient use of natural resources for industrial production of electrical energy [1].

In the traditional electricity market framework electricity consumption was strongly aligned with the regular peak demand curve, and thus, was quite predictable. The power production was more resource consuming and expensive, since it ignored the immediate consumption feedback. Introduction of smart grids to the electricity market resulted in more efficient load management, allowing the supplier to set the price according to the consumption profile and making power production more cost-effective and environmentally friendly. It facilitates the adjustment of hourly consumption patterns of the end-users to minimize expenditures, thus lowering the electrical load in the grid at the peak hours. It also resulted in increased dynamics of the electricity market – the end-users became dependent on the price value at the particular time of the day. The knowledge of future prices is also important for traders, retailers and other entities of the market, since it allows them to optimize financial strategies. Therefore, implementation of price prediction mechanisms is essential in the deregulated market settings.

The electrical load depends on other external factors, such as seasonal patterns, increasing with the drop in temperature. Traditional statistical, linear and non-linear models in most cases fail to deal with the high volatility in electricity price time series data. Artificial neural networks are known for learning complex non-linear dependences. Recently emerged Recurrent Neural Networks, in particular Long-Short Term Memory networks, are capable to extract long-term temporal dependences, thus providing a more suitable tool for electricity price and load prediction.

Despite the growing popularity of Deep Learning, the application of its methods to time series prediction is still in its nascent state and thus requires comprehensive and thorough analysis.

OBJECTIVES AND SCOPE OF THE THESIS

This work aims to explore the application of Deep Learning to electrical load and price forecasting and to compare performance of developed models with the conventional machine learning technics, in particular, clustering-based approaches to time series prediction. To achieve this goal we preprocessed electrical load and price data provided by New York Independent System Operator [2] for New York City district (N.Y.C.) and by Australian Energy Market Operator (AEMO) [3] for New South Wales. We focus on application of ARIMA statistical model, conventional PSF-based clustering algorithms and two Deep Learning models to obtain predictions for one year. Each model's performance has been evaluated in terms of Mean Average Error (MAE), Mean Absolute Percentage Error (MAPE), Mean Relative Error (MRE) and Root Mean Square Error (RMSE). The detailed analysis demonstrated the higher performance of LSTM-based approach for load predictions over other conventional PSF-based clustering models, considered in this work. We have also developed a novel CNN-based approach for day ahead electricity price prediction that takes as inputs the historical price and load data along with the predicted load to generate price forecast for the given day of interest. The proposed network can be deployed for both online and offline forecasting of electricity prices.

RESEARCH METHODS

Datasets:

- Monthly electricity balance data provided by "Moldelectrica" Transmission System Operator [4] for Moldova
- Electrical load and price data provided by New York Independent System Operator [2] for New York City district (N.Y.C.)
- Electrical load and price data provided by Australian Energy Market Operator [3] for New South Wales (N.S.W.)
- COVID-19 New York City data [5]
- COVID-19 New South Wales, Australia data [6]

Deep Learning tools:

Keras [7] is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. We use this framework to develop sequential CNN and LSTM models.

Machine Learning Tools:

Scikit-learn [8] is a free software machine learning library for Python.

K-medoids clustering package [9] within the scikit-learn-extra package. Scikit-learn-extra is a Python module for machine learning that extends scikit-learn. It includes algorithms that are useful but do not satisfy the scikit-learn inclusion criteria, for instance due to their novelty or lower citation number.

SciKit-Fuzzy [10] is a collection of fuzzy logic algorithms for SciPy Stack, Python.

MiniSom [11] a minimalistic and Numpy based implementation of the Self Organizing Maps (SOM)

NOVELTY OF THE STUDY

This study is novel to the extent that we employed Convolutional Neural Networks for electricity price prediction from historical price and load data and the predicted load for the day of interest. To our best knowledge, CNN-based mapping between price and load has not been addressed in the literature. An attempt to predict electricity price by mapping from price and load historical data was reported in [12]. However, the difference of our approach and the one in [12] is that we are using deep CNN instead of backpropagation network of two hidden layers. Our algorithm takes less parameters than the one proposed in [12] (5 vs 9). It is done intentionally to avoid overfitting and to allow the hidden layers of CNN to infer subtle dependences from the inputs.

We also considered the robustness of conventional methods and deep learning based approaches in the situation, when electricity consumption patterns change unexpectedly due to COVID-19 pandemics. To our best knowledge, there were no known research considering the impact of COVID-19 pandemics on the forecasting algorithms performance. The idea of testing the robustness of forecasting methods in the situations, totally out of human control, can potentially lead to a new line of research, aiming to investigate the performance of electricity load/price prediction technics in simulated situations/environments for developing more robust approaches.

Another novel contribution is an improvement to classical PSF by excluding days, containing outliers, above and below 1.5IQR of historical data, from prediction process. This improvement resulted in lower values of all four error metrics. We have also discovered that the same day as the day of interest a week/two weeks/three weeks, etc. ago is important for predicting the load. Finally, the analysis of clustering patterns showed that the clustering algorithms are capable of capturing the seasonal variations in electrical load time series.

KEYWORDS

Time series forecasting, Electrical load prediction, Electricity price prediction, Deep learning, Machine Learning, Long Short Term Memory Networks, Convolutional Neural Networks, K-means, K-medoids, Fuzzy C-Means, Hierarchical clustering, Self Organizing Maps.

CHAPTER 1. LITERATURE REVIEW

The most ubiquitous and popular way of electrical load forecasting is time series analysis. It heavily relies on thorough analysis of past observations to infer the patterns, that contain information about underlying processes. For example, it is logically to anticipate the periodic character of electrical time series, with the demand peaking at afternoon and evening hours. The other factors also have a very strong effect on the demand, for example, it can follow seasonal patterns, increasing with the drop in temperature.

Most of the known methods for electricity demand forecasting can be grouped based on the length of forecasting horizon (i.e. the period of time to be predicted) [13]:

- *Very Short-Term Load Forecasting* (*VSTLF*): for time intervals ranging from seconds or minutes to several hours. VSTLF methods are applicable for flow control and rely on most recent input variables like minutes and hours.

- *Short-Term Load Forecasting* (*STLF*): for a bit longer time intervals ranging from hours to weeks. STLF methods monitor the generation of electricity according to the demand and widely used in electrical market. The input variables for STLF methods are in the range of days.

- *Medium-Term and Long-Term Load Forecasting* (*MTLF* and *LTLF*): applicable for long intervals ranging from months to years. These methods are deployed for development of network assets and utilities. The range of input variables for MTLF and LTLF methods spans over weeks, months and even years.

The companies are more interested in hourly, daily and weekly forecasting horizons, since it allows them to plan electricity generation and sales.

Considering the number of values to predict we can differentiate between two main classes of models:

- models that forecast only one value (e.g. next day's total load or peak load)

- models that forecast multiple values, like aggregated load or load profile (shape) for the next day.

Based on the theoretical approach used in the model, we can distinguish between three main classes of models – linear, non-linear and models, combining both linear and non-linear techniques. Linear models are grouped by those that predict load peak only and load shape (profile) models. The later can be categorized into *time-of-day* and *dynamic* models.

The time-of-day models deal with load values $L(t)$ for each timepoint $t, t \in [1, T]$, in the given load timeseries, where T is the prediction period (e.g. $T = 24$ hours if we are dealing with daily load). In general, time-of-day models can be summarized by the formula:

$$L(t) = \sum_{i=1}^{N} w_i f_i(t) + v(t)$$

i.e. the load at timepoint $t, t \in [1, T]$ is approximated by the weighted sum of explicit time functions $f_i(t)$ and the term $v(t)$ is correction for model's error. Weights w_i can be estimated by linear regression. In some cases, replacing the functions $f_i(t)$ by eigenfunctions for the correlation function of the timeseries can lead to the better results [14]. Some time-of-day models, like *Ordinary Least Squares (OLS)* usually require longer historical data, e.g. 12 years for annual forecasting [15].

Dynamic models assume the relationship between the load value and the external factors, such as temperature or weather conditions. The most well-known dynamic models are *moving average (MA)* and *autoregressive integrated moving average (ARIMA)*. In general, ARIMA models have the following form:

$$L(t) = y_p(t) + y(t),$$

where $y_p(t)$ is correlated with the time of day and weather conditions and can be represented as (1), $y(t)$ is a residual term. In ARIMA model a temperature variable is transformed by non-linear function, to account for non-linearity of the demand forecasting problem.

In most cases, conventional statistical methods such as moving average (MA) and autoregressive integrated moving average (ARIMA) are unable to capture most of the non-linear relationships in

electricity load data [16]. Also, methods for automated time series analysis rely on the large amount of historical data for model training. However, the involvement of large amount of data into analysis and prediction renders the deployment of non-automated methods quite impractical, thus necessitating the emergence of non-traditional automated methods for time-series prediction [17].

In contrast to the traditional time-series forecasting approaches, non-linear models provide better representation of internal trends and impact of external factors on electricity load [18]. For instance, an interesting non-conventional approach to electrical load forecasting has been proposed by Martínez-Álvarez et al. in [17] and later extended in [19] to electricity price prediction. The method is based on the assumption that there exist a sequence of patterns in the series, preceding the target time series data, which is similar to the one around this target. The *Label-Based Forecasting* (LBF) algorithm, known as *Pattern Sequence-based Forecasting* (PSF) algorithm, first has been introduced in [17] and contained two steps: clustering and prediction. At the clustering step, the k-means clustering is applied to group similar chunks of time-series data, where the number of clusters is determined by majority votes of silhouette [20], Dunn [21] and Davies-Bouldin [22] index measures. In this case, the whole dataset is segmented with labels, thus substituting the real data with sequences of labels. These labels are used to forecast the values of the target time-point in prediction step. The LBF/PSF approach is advantageous in the sense that it is very simple and it reduces the dimensionality of the data, thus decreasing the processing time. The number of parameters is low resulting in easy and straightforward process of optimal parameters selection. The method also allows prediction of more than one samples, thus facilitating its deployment for online learning tasks [17].

Limitations of the PSF algorithm were addressed in [23] where authors pointed out to the ambiguity in using the majority votes of three metrics to determine the optimal cluster size. And indeed, as it can be noticed in [24] (Table 1), sometimes three indexes come up with totally contradictory values for optimal number of clusters. Also, the cluster pattern for a day of interest can be similar to the previous day, especially it is true for working days. The existence of patterns different from the previous working day may introduce additional noise at averaging stage, that results in different prediction [23]. The authors proposed an improvement to PSF by considering the frequency of clusters for a day of interest, the more frequently occurring cluster pattern was given higher weight value while averaging [23].

The variation of PSF algorithm has been proposed by Majidpour et al in [25] for calculating electricity consumption while charging the electrical car. Instead of averaging over all found sequences the modified version of PSF algorithm (MPSF) uses the latest found sequence to generate prediction. In case if there is no matching sequence is discovered, the MPSF algorithm considers the centroid of the largest cluster as a forecast value, instead of the previous day. Also, the algorithm starts with number of clusters equal to 10% of the total number of samples for parameters selection. The modified version of PSF algorithm demonstrated improved performance for the prediction of energy consumption in EV charging stations at the outlet level [25].

Several successful improvements were proposed for PSF algorithms. In [24] authors extended the PSF algorithm with other clustering methods, such as k-medoids, agglomerative clustering, fuzzy c-means and self-organizing maps. They also proposed an ensemble model, incorporating different PSF-based clustering algorithms, that implements iterative prediction. The performance of the ensemble exceeded one of each separate expert's and the model provided accurate and more reliable forecast for electricity load data, coming from different sources. Another improvement to PSF has been proposed in [26]. Instead of k-means algorithm authors deployed non-negative tensor factorization for energy supply and demand forecasting. The proposed two dimensional extension of PSF performs better than a conventional framework.

The PSF can be implemented not only for prediction but for outlier occurrences forecast, as it is demonstrated in [27]. In this work PSF employed to discover motifs – pattern sequences that precede the anomalous data. The data of interest, immediately following motifs, were treated separately. The discovery of outlier occurrence has been incorporated into the improved version of PSF. This approach demonstrated a high performance for discovering outliers in most of the electricity data.

A large group of studies representing the non-linear models are based on *Artificial Neural Networks (ANN)*, that became widely used since the second half of the 80's. The Artificial Neural Networks are capable of extracting significant features from the data, inferring non-linear relationships and learning more general concepts from experience. Depending on the neuron model, architecture and learning algorithm ANNs can be Hybrid (e.g. Counterpropagation), Supervised (Brain State in a Box, Fuzzy Cognitive Map, Perceptron, Radial Basis Function, Support Vector Machine), Unsupervised (Adaptive Resonance Theory, Self-Organizing Maps, Principal Component Analysis) and with Reinforcement (Award/Punishment Associative). In

comparison with conventional techniques, ANNs can detect non-linear features even using a small dataset for training, thus resulting in more robust and accurate forecast. Future models for load time series prediction will either incorporate ANNs along with conventional algorithms into hybrid models or will be solely based on ANNs [13].

Time-series can be modeled as combination of deterministic component and some noise. Minimizing the noise component will result in better prediction of the model. For this purpose, the wavelet transform is usually adopted as an efficient noise-reduction method. In [28] authors reinforce artificial neural network by wavelet denoising algorithm for forecasting load in Bulgarian power system grid. This approach utilizes wavelet decomposition to extract signals of different frequencies from historical load data. After reconstruction the signal contains less noise and preserves patterns of original data. In general, wavelet decomposition improves forecasting accuracy and considered an effective method for short-term forecasting.

Several works report implementation of Backpropagation Neural Networks for short-term forecasting of time-series data. For example, in [29] authors predicted hourly load and temperature data of North American and Slovakian networks. They also performed wavelet transform to decompose demand data into low and high frequency components and use these components as inputs to neural network, along with other features such as temperature, humidity and cloud cover.

Another interesting implementation of Artificial Neural Networks aimed to increase the precision of PSF algorithm for electricity load prediction has been adopted in [30]. The authors implemented backpropagation neural networks that use electricity demands for previous day, previous week and combined previous day with previous week to refine the prediction of the PSF algorithm. The new approach proved superior over the traditional PSF algorithm.

Despite the effectiveness of Artificial Neural Networks, for quite a long time they were not as popular as the kernels methods, such as support vector regression (SVR). The resurrection of Neural Networks occurred in 2006, with a seminal work by Geoffrey Hinton et al [31] who demonstrated that the Deep Learning Networks can perform better than conventional Machine Learning methods. It boosted the applications of deep learning methods in different machine learning tasks, including computer vision, natural languages processing and other domains of Engineering and Science. In comparison with traditional shallow learning, deep learning networks incorporate multiple hidden layers and deploy stochastic optimization to learn more complex non-

linear dependences [32]. Varying number of layers allow encoding different levels of abstraction which results in improved performance. As a recently emerged branch of machine learning, deep learning networks are applicable to broad spectrum of complex real-world learning problems. For instance, deep learning networks' implementations can be found in image classification, object detection and recognition, speech recognition, language translation, voice synthesis and many more.

The first implementation of Deep Learning Networks for time-series forecasting is given in [33]. Authors deployed Deep Learning Believe Networks (DBN) for regression and prediction of time series data by combining the outputs from all DBNs with the help of support vector regression (SVR). The effectiveness of the proposed method was demonstrated on three independent datasets. In [34] authors deploy various simulations to show that recurrent networks can potentially outperform conventional nonlinear kernelized methods. They also explore how the performance of the network varies with its architecture (number of hidden layers and neurons per layer), demonstrating that additional complexity does not improve performance, and the optimal performance could be achieved with low number of layers. In another work, [35] authors developed a deep learning network with contrastive divergence for natural gas load forecasting. Other example of using Deep Believe Learning network with restricted Boltzmann machine (RBM) for time-series forecasting can be found in [36]. This novel approach outperformed traditional Multy-Layer Perceptron (MLP) neural network and statistical ARIMA model. One efficient application of Conditional Restricted Boltzmann Machine (CRBM) and Factored Conditional Restricted Boltzmann Machine (FCRBM) to an individual (single-meter) residential electricity consumption forecasting is reported in [37]. The method evaluated on benchmark dataset of individual customer electricity consumption data with temporal resolution of one minute. Authors come to conclusion that FCRBM outperforms ANN, Support Vector Machine (SVM), Recurrent Neural Networks (RNN) and CRBM [37].

In general, conventional ANN based methods are vulnerable to vanishing and exploding gradient problems that occur during the backpropagation stage in multilayer architectures. Also, traditional Artificial Neural Networks and conventional Machine Learning techniques, such as SVM, fail to capture sequential information in time-series data [16]. To overcome these issues, Recurrent Neural Networks (RNNs) were developed as a framework for capturing the non-stationary patterns in time-series as well as long-term dependences. RNNs are capable of sharing the parameters across different time steps and thus, require less training, resulting in more effective computations.

These features made RNNs popular for long and short-term forecasting of electricity load. For instance, in [38] RNN with a pooling layer has been proposed to forecast the household electricity load. It is able to mitigate the overfitting by pooling different customer's load profile into one input, thus diversifying the data. The proposed pooling-based deep RNN demonstrated higher performance in comparison with traditional technics such as ARIMA, SVR and conventional deep RNN. In [16] authors developed a hybrid RNN model, that incorporates one step-ahead concept. First RNN was trained on electricity load time series that later were passed to the hybrid RNN model to learn non-linear dependences. The proposed model demonstrated a superior performance compared to the known methods.

However, deep RNNs still suffer of vanishing and exploding gradient problems that have deteriorative effect on their performance. Long-Short Term Memory networks, were initially introduced by Sepp Hochreiter and Jurgen Schmidhuber [39] as variation of RNNs. Shortly, they became a popular tool for time-series prediction due to their ability to deal with the vanishing and exploding gradient problems. LSTM networks are successfully applied for time-series forecasting in a variety of domains. For example, in [40] authors successfully adapted a combination of Deep Believe Networks, AutoEncoder and LSTM for renewable energy power forecasting and demonstrated that Deep Learning methods perform better than ANNs. In [41] the additional features such as temperature, humidity and wing speed have been fed to LSTM network with electricity load time series data. The LSTM network trained on these data was used for forecasting electricity load both for short and long forecasting horizons, varying from 24 hours to 30 days. Authors show that the LSTM based forecast is more accurate than the prediction from conventional technics.

LSTM networks can be also scaled for the individual customer consumption (single-meter) forecasting. This type of forecasting is a very challenging task due to volatile nature of individual loads influenced by a variety of external factors, such as climate, performance of thermal systems, and residency patterns. Marino et al [42] were first to deploy LSTM networks to a single-meter load forecast. They used the same benchmark dataset as [37] and reported the similar performance as Factored Conditional Restricted Boltzmann Machine (FCRBM). Another work [43] deployed LSTM network for forecasting long-term electricity load time series. The authors deployed density based clustering to infer the inconsistent nature of residential electricity load profiles and discuss challenges associated with predicting load in residential areas. They also demonstrate that LSTM

networks are capable of capturing subtle patterns in individual load profiles which makes them superior in the majority of cases.

Another class of deep learning methods, *Convolutional Neural Networks* (CNNs), have been successfully used for object recognition due to their ability to learn hidden dependences in raw data. CNNs can also capture local trend features and scale-invariant features for closely interrelated neighboring data points, e.g. local trend patterns in nearby hours. These circumstances make CNNs widely applicable for the electricity load prediction [44]. An example of load forecasting with CNN can be found in [45], where the authors demonstrate the superiority of CNN-based approach by reporting the achievement of the smallest forecasting error among the known methods. In the proposed CNN architecture three consecutive convolutional layers were used for feature extraction. The in-between pooling layers decrease the dimension of feature map and extract the important features from the deep layers. The forecasting is done at the output layer, where the flattened values pass through fully connected structure between flatten and output layers to the inputs of sigmoid function. This algorithm allows to make prediction of electricity load for the next three days.

K. Amarasinghe et al [46] explored the application of CNNs for electricity load forecasting on individual building level. The historical loads were passed through multiple convolutional layers and fed to the fully connected layers to perform regression and produce a prediction. As in the similar work on LSTM [42] the benchmark dataset for a single-residential customer data were used to evaluate the performance of CNN-based approach against ANN, SVM, LSTM and FCRBM. The presented CNN demonstrated comparable results to the FCRBM [37] and Sequence to Sequence LSTM [42] on the same dataset. All the examples above demonstrate viability of the CNNs for accurate electricity load forecasting.

In the past the electricity suppliers were not market-oriented and the electricity price were set by governments. However, after the commercialization of the electricity supply and demand networks, the electricity price is determined by the newly emerged electricity market. In other words, the price is set according to the fluctuations in supply and demand in electricity market, where electricity is regarded as a commodity. Compared to the load, the electricity price is more volatile, i.e. it can significantly change over the given period of time. Moreover, the load data is relatively homogenous and demonstrate cyclic variations, while the price data lacks such homogeneity and its variations follow quite a transient cyclic pattern [12]. In these volatile settings

the knowledge of the electricity price beforehand is essential for decision making in electricity market. In fact, all entities of the market will benefit from the accurate price forecasts. The *Independent Power Producers* (IPPs) will be able adjust the generation of power according to the expected price, reducing the cost of production. Market players will devise strategies for selling and buying of the power in the market. The end-users will shift their most electricity consuming activities to the times where electricity prices are low, thus reducing the load in the grid during the peak hours. The problem of price forecasting relies on several factors such as correlations between power demand and supply, generation costs, market structure, influence of other market participants, etc. These factors are used to create models, reflecting causation between historical electricity prices and future prices. Knowing this causation allows to accurately predict electricity prices in the future [47].

Since electricity is not a storable commodity, the balance between power supply and demand can be achieved only by a stable power system. Therefore, in most cases electricity price is highly volatile and easily affected by the factors such as such as sudden changes in weather conditions, transmission problems in the power system, outages of large power plants, participants' bidding strategies, transmission congestion, fuel prices, cost of unit operations to name a few. All these factors introduce additional complexities and challenges to the price forecasting task [47], [48].

Similarly to the electricity load prediction, there exist traditional statistical methods and recent machine-learning based approaches to electricity price forecasting. An example of statistical model based on *Autoregressive Integrated Moving Average* (ARIMA) combined with wavelet transformation is presented in [49]. The day-ahead forecast for the next 24 hours was obtained by applying inverse wavelet transform. The model has been tested on mainland Spain and California markets demonstrating quite reliable results. In [50] ARIMA has been also combined with the *Generalized Autoregressive Conditional Heteroskedasticity* (GARCH) and the wavelet transform was applied to the original time series to decompose them into two parts, that can be predicted separately. To obtain the final forecast the inverse wavelet transform was applied to the prediction of both subseries. The performance of the proposed approach has been evaluated on the Spanish and PJM electricity markets and the results were compared with other forecasting methods.

The problem with conventional statistical frameworks is they perform well on the linear data however, when it comes to non-linear time series, they demonstrate a significant decline in prediction accuracy. To deal with the extreme non-linearity of price time series data, Voronin et

al [51] developed an iterative model, consisting of modules for normal price and price outliers (spikes) prediction. The module for a normal price prediction is a mixture of wavelet transform, linear ARIMA and nonlinear neural network models. The spikes were handled by compound classifiers. The combined forecasts from both normal price prediction and spike prediction modules result into the overall price prediction. The combined model outperformed other known methods for price prediction.

The ability of Artificial Neural Networks to learn non-linear dependences in time-series data was exploited by several authors to create successful models for electricity price prediction. In [52] *Radial Basis Function Neural Network* (RBFNN) is applied for prediction of short term electricity load and price. The model took a day type (e.g. Sunday, Monday) as an optional parameter, and generally demonstrated better performance if the day type is supplied. Another work [12] relies on Artificial Neural Networks to extract the complex dependences between the electricity price, load and other factors. The model takes as inputs several features such as day of the week, time slot of the day, forecasted demand, change in demand, price for one day ago, one week ago, two weeks ago, three weeks ago and a month ago. The forecast horizon can vary from hours to days and weeks. The model proved to be efficient for normal days but demonstrated poor performance for days with spikes.

In [47] authors propose an electricity price forecasting system based on the combination of *Convolutional Neural Network* (CNN) and the *Long Short Term Memory* (LSTM). The CNN network consists of two convolutional 1D layers, with batch normalization after the second convolutional layer. The forecasting is performed by LSTM unit with ReLU activation function. The hybrid system accepts past 24 hours price data as its input and produces a forecast for the next hour. In comparison with the traditional machine learning approaches, the presented hybrid model demonstrates the best forecasting accuracy.

Application of deep learning framework for electricity price prediction has also been explored in [53] where authors consider four different deep learning models and compare their prediction accuracy. The deep learning approaches usually outperform conventional methods. Authors also demonstrate that machine learning approaches perform better than traditional statistical models. The conventional approaches fail to achieve high forecast accuracies for price time series data, particularly the moving average-based models. Also, the hybrid models are not superior over their

simpler components. Thus, the preference should be given to simple non-linear models, e.g. deep learning networks with few neurons in each layer.

In general, deep learning methods demonstrate better performance on electricity price datasets due to their ability of learning complex non-linear dependences and patterns. Considering all the examples above we can make the following observations:

- Long-Short Term Memory networks [39] are better suited for dealing with non-linear sequences and thus, provide better accuracy for electricity price prediction

- A network with more than one layers is capable of modeling the same dependence as the single-layered network, but with the less number of neurons, that simplifies the architecture of the deep-learning network. It also improves generalization of network [53].

Therefore, deep learning approaches represent a very promising technology for electricity price prediction.

CHAPTER 2. THEORETICAL BACKGROUND

PROBLEM DEFINITION

Formally, times series can be represented as sequence of vectors, depending on time t: $X(t), t = 0,1, \ldots$. Components of the vector $X(t)$ are observable variables, such as temperature, speed of wind, stock price, electricity load, etc. Actually, $X(t)$ is a continuous function, however, in practice, we are dealing with discrete time intervals. $X(t)$ defined on such discrete time intervals are called *time sequences* or *time series*. In the case of continuous variable domains, such as temperature, the values are *sampled* at points through the chosen time interval (e.g. measuring the temperature once in an hour). The number of measured points at the given time interval is known as *sampling frequency*. It is an important parameter, since different sampling frequencies may affect the characteristics of the time series.

If vector $X(t)$ consists of one component only, the time series are called *univariate*. In the case of multicomponent vectors $X(t)$ we deal with *multivariate* time series. Analysis of multivariate time series focus on identifying dependences between observable components of vector $X(t)$, for example, how the temperature affects the electricity price and utilizing these patters to predict one of the component.

Time series forecasting, i.e. predicting the future dynamics of the variable of interest, is the most popular technics in real life applications. Most of the problems involve prediction of future values of vector $X(t)$, e.g. predicting electrical energy demand to estimate the network load and schedule electricity production. Formally, the time series forecasting problem can be defined as follows [54]: given time series $X(t), t = 0,1, \ldots$, find a function $f: \mathbb{R}^{k \times n+l} \to \mathbb{R}^k$, where $k = dim\big(X(t)\big)$, e.g. $k = 24$ if we are interested in predicting hourly load. n is the number of separate time series vectors to train, e.g. number of days to train the model. The ultimate goal is to obtain an estimate $\hat{X}(t + d)$ of the vector X, where d is *prediction lag* (usually, $d = 1$), given the values of $X(t)$ up to time t and also additional time-dependent exogenous features $\pi_i, i = 1, \ldots, l$:

$$\hat{X}(t + d) = f(X(t), X(t - 1), \ldots, X(0), \pi_1, \ldots, \pi_l)$$

In this work exogenous features $\pi_i, i = 1, \ldots, l$ that can contain for example, the dynamics of population density in the district of interest, are neglected.

In this work we aim to investigate the forecasting accuracy of traditional methods and the deep learning approaches. From traditional technics we selected the classic ARIMA model and a PSF-style algorithm for load forecasting with the following clustering methods:

- k-means
- k-medoids
- agglomerative clustering
- fuzzy c-means clustering
- self organizing maps

We compare the conventional machine learning methods of short – term load forecast with two powerful deep learning network approaches – Convolutional Neural Networks (CNN) and Long-Short Term Memory Networks (LSTM). The convolutional neural network (CNN) can extract the local trend and capture similar patterns, and the long short-term memory (LSTM) are able to learn the relationship in time steps [55].

Autoregressive Integrated Moving Average (ARIMA)

The Autoregressive Integrated Moving Average (ARIMA) model is a combination of Autoregressive (AR) [56] and Moving Average (MA) [56] models for linear time series forecasting. The Autoregressive (AR) model represents the dependence of current observation on previous measures with added noise (random error). According to AR model, the current/future value $\hat{x}_t$ is expressed a linear combination of p past observations (known as order of the model) and the random error (noise), i.e.

$$\hat{x}_t = c + \sum_{i=1}^{p} \varphi_i x_{t-i} + \varepsilon_t,$$

Where $x_{t-1}, \dots, x_{t-p}$ are the actual values at time $t-i, i = 1, \dots, p$, ε_t is a random error at time t, $\varphi_i,\ i = 1, \dots, p$ are model parameters and $c = const$ [57]. Usually, the constant term is omitted. The parameters are determined from Yule-Walker equations [58] .

The MA model describes how an observation depends upon the past (residual) errors and current white noise term. Conceptually, it is a linear regression of the current/future observation $\hat{x}_t$ over

the random errors of one or more prior observations. The random errors are normally distributed [57]:

$$\hat{x}_t = \mu + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t,$$

where μ is the mean of the series, $\theta_j, j = 1, \dots, q$ are model parameters and q is the order of the model [57].

ARMA model is popular for predicting stationary time series and formulated as a combination of Autoregressive (AR) and moving average (MA) models [57]:

$$\hat{x}_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i x_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j},$$

However, it is usually expressed in lag operator notation, where lag (backshift) is defined as $Lx_t = x_{t-1}$. The lag operator can be raised to powers, i.e. $L^2 x_t = x_{t-2}$. This property allows us to form polynomials of lag operator (lag polynomials):

$$\varphi(L) = \varphi_0 + \varphi_1 L + \varphi_2 L^2 + \cdots + \varphi_p L^p,$$

$$\varphi(L)x_t = \varphi_0 x_t + \varphi_1 x_{t-1} + \varphi_2 x_{t-2} + \cdots + \varphi_p x_{t-p}.$$

Thus in terms of lag operator, the AR,MA and ARMA models can be expressed as [57]:

AR model:

$$\varepsilon_t = \varphi(L)x_t,$$

MA model:

$$x_t = \theta(L)\varepsilon_t,$$

ARMA model:

$$\varphi(L)x_t = \theta(L)\varepsilon_t$$

In practice, time series demonstrate non-stationary behavior, i.e. they can contain trends and seasonal patterns, which can't be captured by ARMA models. A new ARIMA model has been proposed as a generalization of ARMA model for prediction of non-stationary time series. To convert non-stationary time series into stationary, the ARIMA model applies finite differencing of raw observations [57]:

$$\varphi(L)(1-L)^d x_t = \theta(L)\varepsilon_t,$$

p, d and q are integers greater than or equal to zero and refer to the order of the autoregressive, integrated, and moving average parts of the model respectively. More specifically, p is known as *lag order* and defines the number of observations in the model. d – the *degree of differencing*, is the number of times the raw observations are subtracted. q is the order of moving average (size of moving average window). The parameter d controls the level of differencing, for ARMA $d = 0$. If $d = 0$ and $q = 0$ we have a case of AR model and with $d = 0$ and $p = 0$ – the case of MA model.

k-means

The goal of k-means clustering is to minimize an objective function J, which represents the cumulative distance between the cluster center and the set C_j of N_j data points belonging to a particular cluster k. In other words, k-means attempts to partition N observations into k separate subsets C_j containing N_j data points. K-means algorithm is an example of unsupervised learning, since it does not require the labeled data. It is also a greedy algorithm which performance depends on the choice of the parameter k and the appropriate selection of the initial cluster centers [59].

The objective function J can be specified as [24]:

$$J = \sum_{k=1}^{K} \sum_{n \in C_j} w_{i,n} d(x_n, \mu_k)$$

where μ_k is the cluster centroid for points in C_k and $d(x_n, \mu_k)$ is a distance metrics between a data point x_n and the cluster center μ_k. The most popular metrics for distance is Euclidean distance.

In general, k-means algorithm includes the following steps:

1. Initialize parameters of the algorithm by specifying the number of clusters k
2. Shuffle the data to eliminate any ordering
3. Randomly select k separate data points as initial cluster centroids.
4. For every data point x_n and for every cluster k compute the distance $d(x_n, \mu_{\mathrm{k}})$ between this data point and the cluster center μ_k
5. Based on this distance assign each data point to the closest cluster (centroid).
6. Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.
7. Keep iterating until no change occurs with centroids, i.e. assignment of data points to clusters is fixed.

The k-means algorithm is easy to implement and it is computationally faster than other known methods (e.g. hierarchical clustering). However, the initial seed and the order of data can affect the clustering (that's why we need step 2). Also normalizing dataset can impact the results significantly.

K-medoids

The k-medoids algorithm [60] is similar to the k- means algorithm in the sense, that it relies on medoids rather than centroids for representation of cluster center. In contrast to centroid, which is an average of the data points in the cluster C_j, a medoid is an instance of the data in this cluster. To determine medoids the algorithm looks foe a datapoint i within the cluster that minimizes $\sum_{j \in C_j} d(i,j)$, where C_j is the cluster containing object i and $d(i,j)$ is the distance between objects i and j.

The k-medoids algorithm can be described as follows [60]:

- Initialize parameters of the algorithm by specifying the number of clusters k
- Shuffle the data to eliminate any ordering
- Randomly select k separate data points as initial cluster medoids.
- Assign each data point to the cluster associated with the closet medoid.
- Recalculate the positions of the k medoids.
- Keep iterating until no change occurs with medoids, i.e. assignment of data points to clusters is fixed.

There are several advantages of using existing objects as the centers of the clusters:

- As a representative of a data, medoid serves to describe the cluster better than an abstract centroid
- K-means recalculates all the distances between datapoints and centroids, since the new centroids are created on every iteration. In the case of k-medoids, there is no need to recalculate distances between data points and algorithm can obtain them from distance matrix, speeding the convergence of the k-medoids algorithm in a fixed number of steps.
- K-medoids is less sensitive to outliers than other partitioning algorithms.

However, the k-medoids algorithm can converge to different partitions for different runs on the same dataset because the first k medoids are chosen randomly.

Agglomerative clustering

Agglomerative (Hierarchical) Clustering initializes each data point as an individual cluster, iteratively fusing similar points into corresponding clusters. There is no need to provide a pre-specified number of clusters [61]. The only parameter that requires to prevent merging all the clusters into one is the cut-off distance or the optimal number of clusters. It can be determined from dendrogram – a tree representing relationship between similar clusters. One axis of dendrogram represents data points and clusters, the other – distance or dissimilarity between clusters. An intuitive way of determining the cut-off distance is to select a distance corresponding to the maximum dissimilarity between clusters (where the branches are longer. Several computational methods are also available such as the Dynamic Cut Method [62] that process the topology of the tree.

The Agglomerative (Hierarchical) Clustering algorithm starts with calculating a distance matrix M containing all pairwise distances between n points of the given dataset $D = (x_1, x_2, \dots, x_n)$. Then it proceeds with the following steps:

1. Select two closest points $x_i, x_j \in D$, i.e. $i, j = \underset{i,j:\, i \neq j}{argmin}\, d(x_i, x_j)$
2. Agglomerate these two points into a new point (cluster) c. This new point can be obtained by calculating the mean of x_i, x_j or using some other method
3. Replace x_i, x_j in D with this new point c and recalculate the distance matrix M
4. Keep iterating recursively until D left with a single data point

Hierarchical clustering creates a hierarchical structure of the data which is way more informative and allows to determine the number of clusters more accurately. The disadvantage of the method is that it can converge to unrealistic dendrograms, that does not reflect the existing correlations between the data points. It is sensitive to outliers and the initialization of clusters at the first step. Also, calculating the distance matrix M requires ample computational resources and is not feasible with large datasets.

Fuzzy C-means clustering

Fuzzy logic approach can be utilized to cluster multidimensional data by assigning each data point a membership value, varying between 0 and 100%, in each cluster center, which reflects the proximity of the given data instance to the particular cluster center [63]. Thus, a piece of data belongs to more than one clusters at a time, thus demonstrating better results while clustering the overlapping datasets. It makes Fuzzy C-means clustering more flexible and powerful in comparison with the conventional thresholding approaches.

Given the dataset $D = (x_1, x_2, \dots, x_n)$ of N points and set $V = \{v_1, v_2, \dots, v_c\}$, the Fuzzy C-means algorithm consists of the following steps:

1. Specify randomly c cluster centers.
2. Initialize the fuzzy membership matrix $U = \left(u_{i,j}\right)_{n\times c}$, where $u_{i,j}$ is a fuzzy membership of i^{th} data point to j^{th} cluster:

$$u_{i,j} = \frac{1}{\sum_{k=1}^{c}\left(\frac{d_{ij}}{d_{ik}}\right)^{\left(\frac{2}{m-1}\right)}}$$

   where n is the number of data points, d_{ij} is the distance of i^{th} data point to j^{th} cluster, $m \geq 1$ is the fuzziness index.

3. Compute the fuzzy centers v_j by formula: $v_j = \frac{\sum_{i=1}^{n} x_i (u_{ij})^m}{\sum_{i=1}^{n} (u_{ij})^m}$, for every $j = 1,2, \dots, c$
4. Keep repeating steps 2 and 3 to minimize the objective function:

$$J(U,V) = \sum_{i=1}^{n}\sum_{j=1}^{c}\left(u_{ij}\right)^m\left(x_i, v_j\right)$$

   or until the termination threshold β is reached, $\beta > d\left(U^{(k+1)}, U^{(k)}\right)$ and $\beta \in [0,1]$.

Self-organizing map (SOM)

A self-organizing map is an artificial neural network for unsupervised learning, able to reduce the dimensionality of the data by mapping it into two dimensional space in the meantime, preserving the topological properties (relationships) of the data (input space), but not the actual distances [64]. In other words, nodes in the network's output space reflect statistical features of the input space (data).

Utilizing a process of learning by observation, the self-organizing map is able to infer underlying hidden patterns and dependences in the data [65]. For this purpose, the SOM employs a competitive learning policy, i.e. making the output nodes to compete among themselves for better representation of distinct concepts learned from the data [66]. The quality of representation is estimated by discriminant function, e.g. a best matching unit (BMU), that compares the input vector with the weighted vector of every output node so that the node with connection weights closest to the original input instance gets selected. The neighboring nodes in output space encode similar patters in the input data. Therefore, the winner node creates a neighborhood of the nodes with common features (cooperating nodes). As a result of competitive training, the winning node and its' surrounding get modified: the winner becomes a center of the local group of cooperating nodes, which contribute in the process of weight adjustment. In other words, weights of winner node and its neighbors get updated to maximize their discriminant functions, thus adapting the relevant nodes to the input data. These nodes are capable of inducing mutual activation and it makes them more responsive to the similar features in the input data.

SOM is a completely connected network in the sense that every node in input space is linked to every node in output layer [67]. Nodes in input layer do not participate in the training of the network. They directly transmit the information from the input nodes to the nodes of the output layer, so that each value of the input data is associated with one input node. Each node is initialized by the vector of weights $w = [w_1, w_2, \dots, w_n]^{\mathrm{T}}$, where n is the dimensionality of the input space. The data can be directly transferred to the output layer, as it is done in Kohonen maps [68] or pass through several hidden layers (more general cases of SOM). A two-dimensional grid structure is adopted to facilitate dimensionality reduction from n- dimensional to a two-dimensional plane. The neurons of the output layer are located at the nodes of a two-dimensional network of a rectangular or hexagonal structure. Note that the Euclidean metrics is easier to define on hexagonal rather than rectangular structure. The network is sensitive to the metrics choice since the degree

of interaction between nodes depends on the distance between them and the number of nodes defines the map's ability to generalize.

The SOM clustering algorithm consists of the following steps:

1. *Initialization* involves setting the following network configuration: number of nodes in the network, learning speed η and the initial learning radius R (the number of winner node's neighbors). Specifying a right set of parameters depends on compromise between the degree of detalization and the training speed. A situation may arise when the number of neurons exceeds the number of objects in the training data sample. In this case, the learning outcome depends on the choice of the initial radius, which affects the network's ability to generalize. Large number of nodes will result in longer training. Also, during initialization, certain values are assigned to the weights of the nodes. It is possible to initialize all nodes with small random numbers, or with randomly selected feature values from the training set, or initialize with the linearly ordered vectors in a linear subspace between the two main eigenvectors of the original data set.

2. *Training.* It starts with activating network by activation vector X_n, randomly selected from the dataset. On this stage of each node competes with the other nodes to become more similar to the training data. At the end of this process, known as vector quantization, the winner node is the one closest to X_n. If we denote the weight vector of winner node c as w_c, than the distance metrics characterizing the proximity of node c to activation vector X_n is $\|X_n - w_c\| = min_i \|X_n - w_i\|$, where $i = 1, \dots, n$. After choosing the winner node, a consolidation of other nodes within the learning radius of R takes place. At this stage of training the weights of winner node and its neighbors move towards the vector X_n. The new weights for the iteration $t + 1$ are

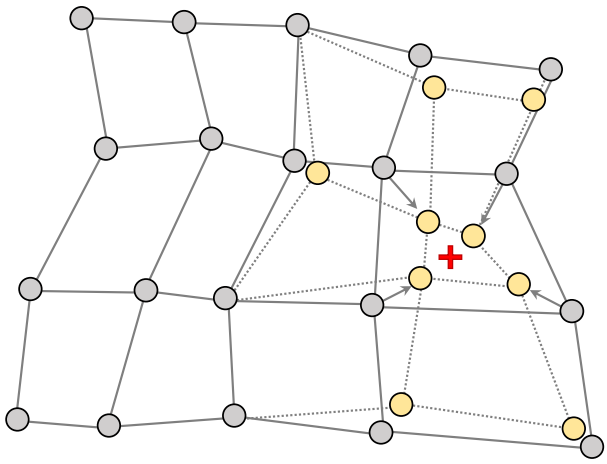


Fig. 1. Tuning weights of winner node and its neighbors. Position of activation vector marked with cross, maps' nets before tuning are grey, after tuning are yellow. Network after tuning denoted by dashed lines.

initilized as $w_i^{t+1} = w_i^t + \eta \| X_n^t - w_i^t \|$, where η is learning speed and w_i^t – are weights from previous iteration (see Fig. 1).

3. At the *correction* stage learning radius R and learning speed η are tuned to achieve optimal performance. First, large values of the radius and learning speed are taken, to ensure the location of network node corresponds to the data in the training set. Then the radius and learning speed are reduced to achieve more fine tuning.

In trained SOM the relationship between nodes from output layer and patters in input data are determined. Therefore, all the patterns from inputs can be mapped in the output layer. In this work, to cluster the data with SOM we set the number of nodes in output layer equal to the number of clusters, and train the network so that each node becomes a centroid of a cluster, thus identifying the clusters with the neurons of the network. Despite the need to specify the number of clusters and network structure (node topology), the self-organizing maps have several advantages over conventional dimensionality reduction technics, such as principal component analysis and multidimensional scaling. Firstly, they can deal with any data, i.e. there is no requirement for data instances to be independent. The self-organizing maps also do not make any assumption about data distribution [66] [69]. Next, they are able to solve high-complexity problems and does not require much efforts in implementation [70]. And finally, self-organizing maps are more efficient to deal with outliers, noise, missing values [71], they can process instances of very low dimensionality [66] and work well with big data [72]. The ability of self-organizing maps to reduce dimensionality of the data while preserving relationships between data points makes them a great tool to apply for the prediction of non-linear time series [73].

PATTERN SEQUENCE-BASED FORECASTING ALGORITHM

The Pattern Sequence-Based Forecasting (PSF) algorithm [17] (known also as Label Based Forecasting Algorithm (LBF)) algorithm is robust to noise due to operations with integer substitutes of the real valued time series. The schematic representation of the algorithm is given on Fig. 2. Electricity load for each day i is stored in 1×24 vectors $E_i = \langle e_1, e_2, \dots, e_{24} \rangle$, where e_i are load values for each hour. After applying clustering method to all of these N vectors, we get the set of labels $L = \langle L_1, L_2, \dots, L_N \rangle$. Let's denote $S_W^d = \langle L_{i-W+1}, L_{i-W+2}, \dots, L_{d-1}, L_d \rangle$ the subsequence of labels of the loads of the W consecutive days, from day d backward. The size of

the window is determined in [17] and usually around 5 days. The algorithm goes through the previous days, labeled by clustering algorithm, to look for an exact sequence of W labels, preceding the day of interest (one we are going to predict). In other words, we need to find the set ES defined by equation:

$$ES = \{\text{set of indexes } j \text{ such that } S_W^j = S_W^d\}$$

In case of not finding any subsequence in data base equal to S_W^d, the procedure searches the subsequences of labels which are exactly equals to S_{W-1}^d. I other words, we decrease the length of the window composed of the subsequence of labels.

The 24 hourly loads of day $d + 1$ are predicted by averaged the loads of the days succeeding those in ES. That is,

$$E_{d+1} = \frac{1}{\text{size}(ES)} \sum_{j \in ES} E_{j+1},$$

where size(ES) is the number of elements in set ES.

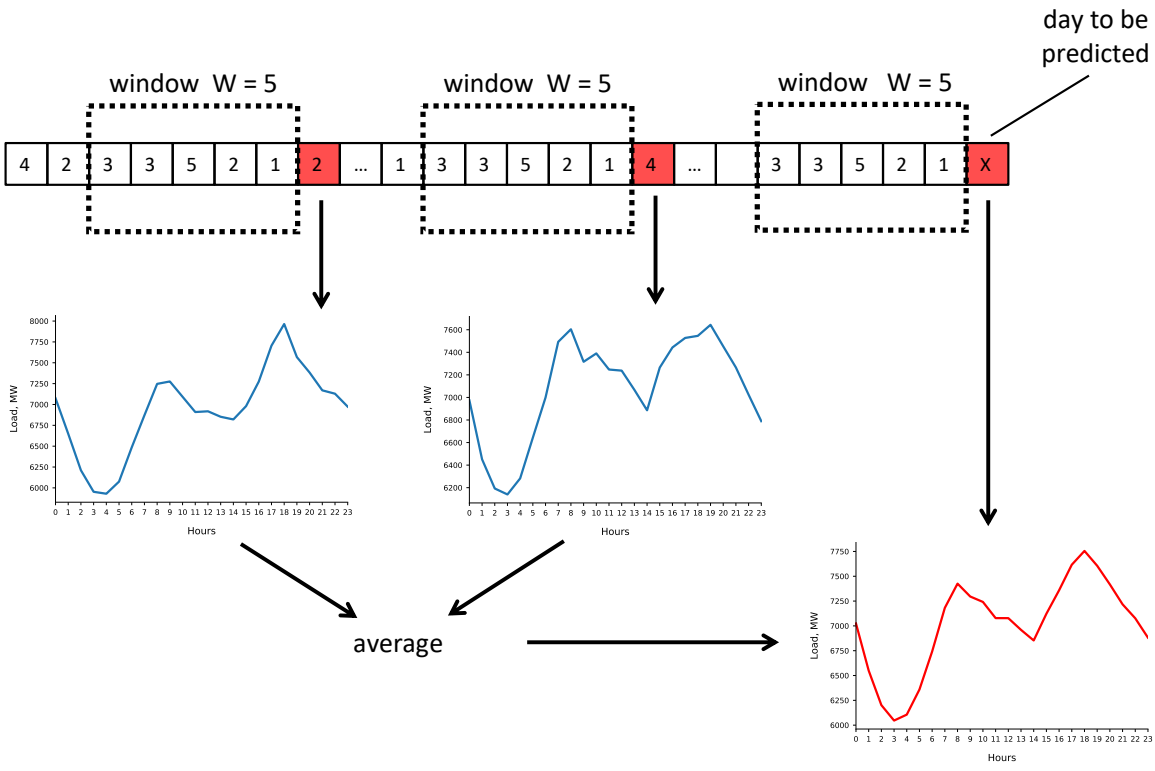


Fig. 2. Pattern Sequence-based Forecasting algorithm (see Fig. 3 [17]).

Deep Learning is a subfield of Machine Learning that utilizes multiple layers for successive inference of higher level features from the input data [74]. The higher the layers the more meaningful get the learned representations and concepts. The "deep" in term deep learning refers to the number of successive layers of representations that contribute to a model (so called *depth of the model*). Deep learning model can contain hundreds of successive layers. The models using a few layers are referred as *shallow learning* models. In other words, a deep learning model is a multistage structure that distills information, passing through successive filters, to make it clean and suitable for concepts extraction and representation learning [75].

The layered representations are learned with *neural networks* consisting of layers, stacked on top of each other. A weight vector, associated with the layer, parametrizes the transformation implemented by the layer. Weight vectors are called *parameters* of the layer and assigned randomly at the initial stage of the training. To correctly map inputs to their targets the network needs to calculate the parameters for each layer. In order to determine network's performance the output must be compared to the target value. The *loss function* evaluates the difference between the target value and output by computing the distance score. This distance score represents how well the network performed on this particular instance and used as a feedback to adjust the values of weights, to minimize the loss function. It is accomplished by *optimizer* running the *Backpropagation algorithm* (see Fig. 3). Within the training loop, the weights get adjusted with every new processed data instance, and the loss function decreases [75].

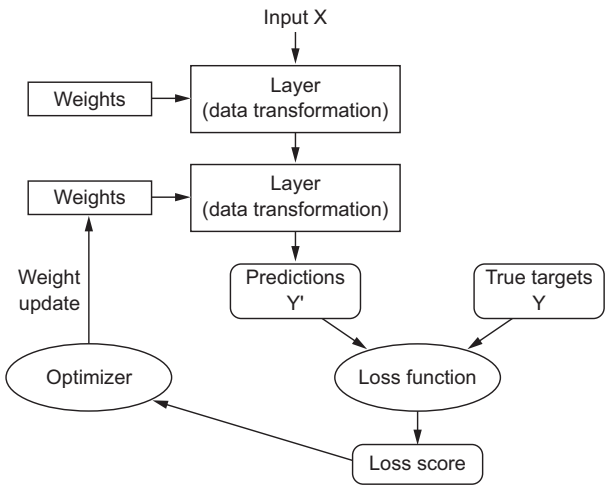


Fig. 3. Schematic representation of deep learning approach (see [75], Figure 1.9)

The idea of neural network was inspired by the neuron in human brain. The neuron contains a cell body, or *soma*, inputs — *dendrites* and outputs – *axons* (See Fig. 28 in Appendix). On receiving electromechanical inputs from neighboring neurons, connected by dendrites, the neuron can get activated or not, depending on the strength of the input. An activated neuron processes the signal

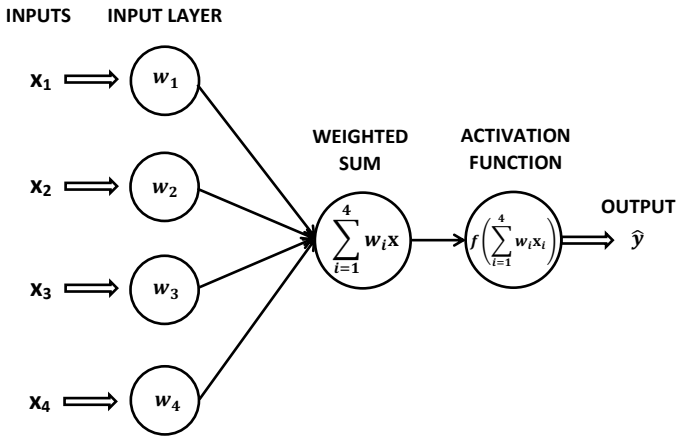


Fig. 4. Schematic representation of artificial neural network. See Fig. 10.3 [76]

in soma and then transmits it along the axon to the dendrites of nearest neurons. Therefore, neuron can be in two states – either it fires or not, thus it can perform binary operations [76].

Even though the idea of artificial neural network was borrowed from neurobiology, the deep learning networks are not brain models, since there is no evidence that the neurophysiological and neurobiological mechanisms, responsible for processing data in our brains resemble the algorithms used in deep learning models. The deep learning is just a mathematical framework for learning concepts and representations from the data [75].

Assume our data is stored in the form of multidimensional *design matrix*, containing data points and associated class labels. A basic artificial neural network performs a weighted summation of inputs (data points x_1, x_2, x_3 on Fig. 4). The fourth input x_4 is a bias added to design matrix (constant value 1). Each data point x_i is connected to a neuron via the *weight vector* $w = (w_1, w_2, \dots, w_n)$. The *output node* passes the weighted sum through the activation function f to get the output value of $f(\sum_{i=1}^{n} w_i x_i)$.

The following activation functions are widely used [76] :

- Step function: $f(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{otherwise} \end{cases}$, where $z = \sum_{i=1}^{n} w_i x_i$
- Sigmoid function: $f(z) = \frac{1}{1+e^{-z}}$
- Hyperbolic tangent: $f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- Rectified Linear Unit (ReLU) : $f(z) = \max(0, z)$
- Leaky ReLU: $f(z) = \begin{cases} z, & \text{if } z \geq 0 \\ \alpha \times z, & \text{otherwise} \end{cases}$
- Exponential Linear Units (ELUs): $f(z) = \begin{cases} z, & \text{if } z \geq 0 \\ \alpha \times (e^z - 1), & \text{otherwise} \end{cases}$, where $\alpha = const$

The main goal of network training is to learn the weights vector w given the input vector $x = (x_1, x_2, \dots, x_n)$ and the actual output y. It is achieved with *backpropagation*, which incorporates gradient descent optimization of the loss function:

$$L(y, \hat{y}) = (y - \hat{y})^2,$$

where $\hat{y} = f(z) = f(\sum_{i=1}^{n} w_i x_i)$ – network prediction of y , $z = \sum_{i=1}^{n} w_i x_i$ and f is the activation function (see Fig. 4 for the case of $n = 4$).

First calculate the first partial derivative of the lost function with respect to the network output $\hat{y}$:

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}(y - \hat{y})^2 = -2(y - \hat{y}).$$

Than calculate the $\frac{\partial L}{\partial z}$ as a function of $\frac{\partial L}{\partial \hat{y}}$. This depends on the mathematical representation of the activation function $f(z)$. We consider simplest case of sigmoid, tanh or ReLU functions to ensure the derivatives are elementwise functions. Applying chain rule we get:

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = \frac{\partial L}{\partial \hat{y}} \frac{\partial f}{\partial z} = f'(z) \frac{\partial L}{\partial \hat{y}}$$

In the case of sigmoid activation function $f(z) = \frac{1}{1+e^{-z}}$ we get: $f'(z) = f(z)\big(1 - f(z)\big)$.

Next, using derivatives $\frac{\partial L}{\partial z}$ we apply the chain rule to calculate partial derivatives of the weights:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial w_i} = \frac{\partial L}{\partial z} \frac{\partial}{\partial w_i} \left(\sum_{i=1}^{n} w_i x_i \right) = x_i \frac{\partial L}{\partial z}$$

Finally, we calculate derivatives of the loss function with respect to the input activations $x_i$using $\frac{\partial L}{\partial z}$:

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x_i} \left(\sum_{i=1}^{n} w_i x_i \right) = w_i \frac{\partial L}{\partial z}.$$

This approach can be extended for multiple layers. In this case the outputs of the previous layer are used as input activations x_i [77].

Neural networks, consisting of more than one computational layer are known as *multilayer networks*. The layers between input and output layers are so called "*hidden layers*," since they hide computations from the user [78]. The most commonly used multilayer network is a *feedforward network*, where the connection between nodes is only allowed from nodes in layer i to nodes in layer $i + 1$. *Recurrent neural networks* are the same feedforward networks with connections fed back to the inputs (*feedback connections*) [76]. The architecture of feedforward network is fully

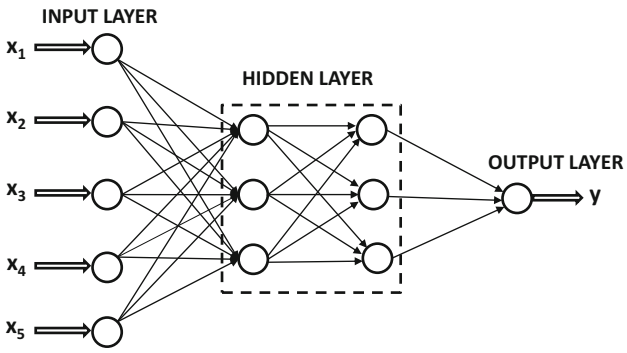


Fig. 5. The basic architecture of a feed-forward network with two hidden layers and a single output layer. See Figure 1.11(a) [78].

defined and the lost function is optimized in output layer. Bias neurons can be found both in hidden and output layers. For example, in Fig. 5 the network contains three layers (usually input layer is not taken into account). The number of units in each layer determines the *dimensionality of network*.

The weight matrix of connections between the input layer of d neurons and the first hidden layer containing p_1 neurons is W_1 $[d \times p_1]$. We denote weight matrix between r-th hidden layer (p_r neurons) and $r+1$-th hidden layer (p_{r+1} neurons) as W_r $[p_r \times p_{r+1}]$. The final matrix between k-th hidden layer (p_k neurons) and output layer (o neurons) is W_r $[p_k \times o]$. Given the activation function f the transformation of vector $X = (x_1, x_2, \dots, x_n)$ into the output vector $\hat{X}$ can be formally written as [78]:

- $h_1 = f(W_1^{\mathrm{T}} X)$ (input to first hidden layer)
- $h_{p+1} = f\left(W_{p+1}^{\mathrm{T}} h_p\right) \forall\, p = 1, \dots, k-1$ (hidden to hidden layer)
- $\hat{X} = f\left(W_{k+1}^{\mathrm{T}} h_k\right)$ (last hidden to output layer)

These equations are only valid for layer-wise feedforward networks and does not take into account other unconventional architectures [78].

The loss function in single-layered network is a direct function of the weights, thus its gradient is easy to compute, which makes the training quite straightforward. In multilayer networks loss function is composition of functions of weights from each layer and its gradient can be computed with the *backpropagation algorithm*, that utilizes *dynamic programming* and chain rule of calculus. The algorithm incorporates *forward* and *backward* phases. At the forward phase, training data are supplied to the network inputs and corresponding weights are calculated across network's layers. The output is compared to target and the derivative of loss function is obtained. At the backward phase this derivative of the loss is computed across all layers with respect to different weights and this process occurs in backward direction, starting from the output layer. This two-stage process of learning weights at different nodes is a computationally complex and requires iterating through the training data in multiple epochs.

The deep learning network can:

- automatically learn complex arbitrary relationships between inputs and outputs
- support multiple inputs and outputs
- *Convolutional Neural Networks (CNN)* are capable to learn features and concepts from data
- *Long-Short Term Memory Networks (LSTM)* learns temporal dependencies in the data
- *Hybrid models* incorporate abilities of different network architecture for more efficient prediction [79]

Convolutional Neural Networks (CNNs)

Originally inspired by information processing in brain's visual cortex, the most successful of all deep learning approaches, *Convolutional Neural Networks* (CNNs) are comprised of three dimensional layers, that are able to encode both the spatial and depth representations, depending on the number of features. This architecture reflects the organization of RGB color channels in input (*convolution*) layer and therefore, CNN is widely used for image classification. The hidden (*subsampling*) layers of CNN encode primitive shapes in lower layers, gradually progressing towards encoding more complex shapes, like loops, in higher layers. In the case of grayscale input the input layer has a depth of 1 but higher layer preserve their 3D structure. The convolution layer is termed after the *convolution operation*, that relies on filter to map the activations from one layer to the next. During the convolution a 3D filter of weights is used and selection of spatial region in a layer, passed through the activation function (e.g. ReLU), defines the value of the hidden state of the next layer. In the next layer the activation preserve their spatial relationships from the previous layer. Since the any activation in every hidden layer is a function of a tiny region in previous layer, the convolutional network is sparsely connected. To compress the spatial residuals coming from previous layers, an additional subsampling layer is added, that averages the values of local regions of size 2×2 (*subsampling operation*). A typical convolutional network of 6 layers is presented at Fig. 6. Traditionally, CNNs were designed for working with two-

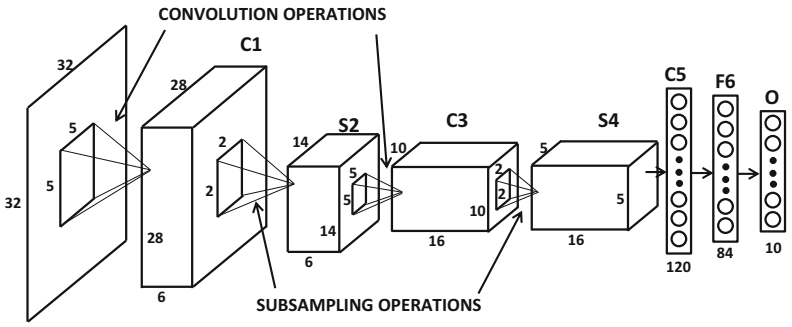
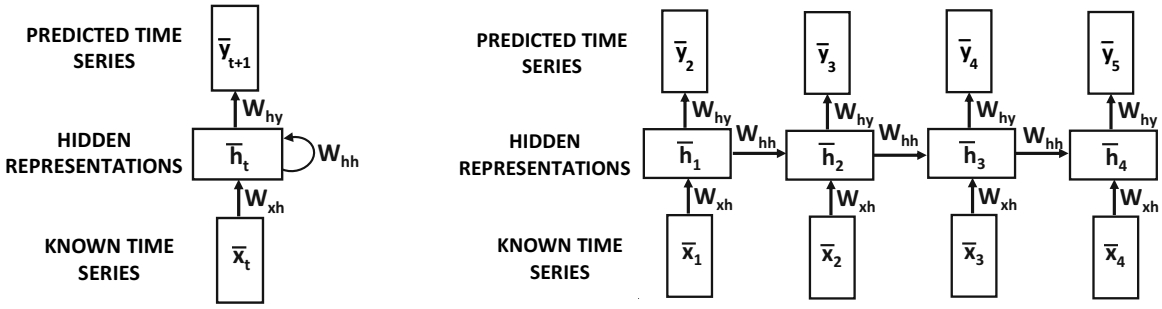


Fig. 6. An example of Convolutional Neural Network (LeNet-5). See Figure 1.18 [78].

dimensional image data and they become very popular for image recognition, object localization and text processing [78]. In this work, we demonstrate how the CNNs can be applied for time series forecasting.



(a) RNN (b) Time-layered representation of RNN

Fig. 7. A Recurrent Neural Network (a) and its time-layered representation (b) (See Fig. 7.2, [78])

Long-Short Term Memory Networks

Time series comprised of sequentially dependent values, i.e. values of neighboring time stamps are closely interrelated. To avoid loss of information, values in time series should not be treated as independent entities. In other words, the network should be able to:

- receive and process inputs in the same order as they are present in the sequence
- the treatment of inputs at each time-stamp in a similar manner in relation to previous history of inputs [78].

This functionality is achieved by *Recurrent Neural Networks* (RNNs), that create one-to-one correspondence between the layers of network and time-stamps in time series. The term recurrent points on the network's architecture, where each layer uses same set of fixed parameters to ensure similarity in modeling of every time-stamp. This uniform layer-wise structure is repeated in time, i.e. *recurrently*. In other words, RNNs, in contrast to other feed-forward networks, take as their inputs not just a currently available data, but also the data they have processed previously. This time-layering approach in feed-forward structure of RNNs allows the networks to transform a sequence of inputs into a sequence of outputs, thus making them useful for sequence-to-sequence learning (e.g. time-series forecasting, machine translation and etc.) [78].

The schematic representation of RNNs' architecture is given in Fig. 7(a). The presence of self-loop in structure results in modification of hidden state in network after the input of each new instance of time-stamped data. This loop can be unfolded into a time-layered network with a different node for hidden state at each time-stamp and self-loop converts into feed-forward

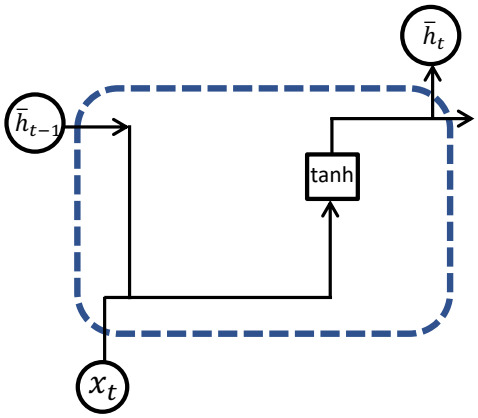


Fig. 8. Block diagram of RNN network unit. See Fig. 3a [47].

network (see Fig. 7(b) and Fig. 8). Here $x_t, \dim(x_t) = d$ is the input vector at time t, the hidden state at time t is $\bar{h}_t, \dim(\bar{h}_t) = p$ and the output vector for time t is the prediction for next time step $t+1$, i.e. $\bar{y}_{t+1}, \dim(\bar{y}_{t+1}) = d$. The hidden state at time t is given by a function of input vector at time t and the hidden vector at time $(t-1)$, i.e. $\bar{h}_t = f(\bar{h}_{t-1}, x_t)$. The function f is defined in terms of weight matrices and activation functions with a requirement that the same weights are used at each time-stamp. Another function, $\bar{y}_{t+1} = g(\bar{h}_t)$ is used to learn the output values from the hidden states. If we define input-hidden matrix $W_{xh}, \dim(W_{xh}) = p \times d$, hidden-hidden matrix W_{hh}, $\dim(W_{hh}) = p \times p$ and hidden-output matrix $W_{hy}, \dim(W_{hy}) = d \times p$, than conditions for the outputs are:

$$\bar{h}_t = \tanh\left(W_{xh}x_t + W_{hh}\bar{h}_{t-1}\right)$$

$$\bar{y}_{t+1} = W_{hy}\bar{h}_t$$

In the very first time-stamp, $\bar{h}_{t-1}$ is initialized with zeros, because there is no input from the hidden layer at the first time-step of series. Although the hidden states change at each time-stamp, the weight matrices stay fixed over the various time-stamps [78]. The mathematical formulations above can be extended for the case of multiple layers (e.g. the hidden state of k-th layer can be denoted as $\bar{h}_t^{(k)}$).

RNNs relay on backpropagation through time mechanism, however, they are prone to the *vanishing* and *exploding gradient problem*. Backpropagation utilizes the chain rule to compute the gradient of loss function with respect to the network's weights. Depending on the architecture, with the increase in the number of layers, updates in earlier layers can be very small (*vanishing gradient*) or very large (*exploding gradient*). This type of instability is the direct result of successive multiplication with the (recurrent) weight matrix at various time-stamps. In other words, networks relying on multiplicative updates learn well over short sequences, i.e. have good short-term memory but poor long –term memory [78].

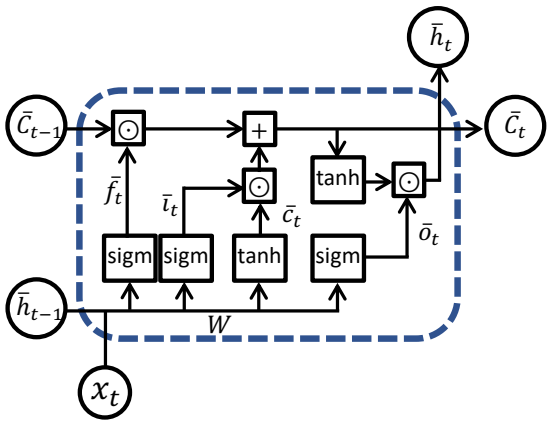


Fig. 9. Block diagram of the LSTM recurrent neural network cell unit. See. Fig. 2 [84]

Long-Short Term Memory (LSTM) networks address this issue by introducing modification in the recurrence equation for the hidden layers. Assume $\bar{h}_t^{(k)}$is the hidden state of k-th layer (see Fig. 9). LSTM adopts a multilayer RNN's architecture with the difference in recurrence conditions in propagation of hidden states $\bar{h}_t^{(k)}$. In LSTM a *cell state* vector $\bar{C}_t^{(k)}, \dim\left(\bar{C}_t^{(k)}\right) = p$ is introduced to retain information of earlier states. The update matrix $W^{(k)}$in LSTM is of size $4p \times 2p$, i.e. multiplying column vector $\left[\bar{h}_t^{(k-1)}, \bar{h}_{t-1}^{(k)}\right]^T$ of size $2p$ results in a vector of size $4p$. The updates rely on four p −dimensional vector variables $\bar{\imath}$ (*input*), $\bar{f}$(*forget*), $\bar{o}$ (*output*), and $\bar{c}$. To determine the hidden state vector $\bar{h}_t^{(k)}$and the cell state vector $\bar{C}_t^{(k)}$, first the intermediate variables $\bar{\imath}$, $\bar{f}$, $\bar{o}$, and $\bar{c}$ are computed to obtain the hidden variables from these intermediate variables.

$$\begin{matrix}\text{Input Gate:}\\ \text{Forget Gate:}\\ \text{Output Gate:}\\ \text{New C.-Gate:}\end{matrix}\begin{bmatrix}\bar{\imath}\\ \bar{f}\\ \bar{o}\\ \bar{c}\end{bmatrix} = \begin{pmatrix}\text{sigm}\\ \text{sigm}\\ \text{sigm}\\ \tanh\end{pmatrix} W^{(k)} \begin{bmatrix}\bar{h}_t^{(k-1)}\\ \bar{h}_{t-1}^{(k)}\end{bmatrix} \text{[Setting up intermediates]} \quad (1)$$

$$\bar{C}_t^{(k)} = \bar{f} \odot \bar{c}_{t-1}^{(k)} + \bar{\imath} \odot \bar{c} \text{ [Selectively forget and add to long} - \text{memory]} \quad (2)$$

$$\bar{h}_t^{(k)} = \bar{o} \odot \tanh\left(\bar{C}_{t-1}^{(k)}\right) \text{ [Selectively leak long} - \text{memory to hidden state]} \quad (3)$$

"⊙" is elementwise product of vectors and "sigm" is sigmoid function. In the case of $k = 1$ (first layer) $\bar{h}_t^{(k-1)} = \bar{x}_t$ and dim $(W^{(k)}) = 4p \times (p + d)$.

At the beginning the intermediate variable vectors $\bar{\imath}$, $\bar{f}$, $\bar{o}$, and $\bar{c}$ are initialized on interval $(0,1)$. The first three of them, i.e. $\bar{\imath}$, $\bar{f}$ and $\bar{o}$, are referred as input, forget and output gates and conceptually regarded as Boolean variables, although they are continuous on the interval $(0,1)$ to ensure differentiability for gradient updates. Vector $\bar{c}$ is used to increment cell states and contains p values, varying in the interval $[-1, +1]$ (because of tanh function). These three vectors determine: a) whether to add a cell state, b) whether to forget a cell state or c) whether to allow leakage into

a hidden state from a cell state. Vector $\bar{c}$ stores the newly updated contents of the cell state and the input and forget gates regulate the extent of modifications in the cells state. All four intermediate variable vectors $\bar{\imath}$, $\bar{f}$, $\bar{o}$, and $\bar{c}$ are calculated using the weight matrices $W^{(k)}$ for the k-th layer in (1). Equation (2) consists of two members. The first member, $\bar{f} \odot \bar{C}_{t-1}^{(k)}$ utilizes the p values in vector $\bar{f}$ to decide on which p cell states, coming from the previous time steps will be reset to 0. It also takes the p values in vector $\bar{\imath}$ to decide if corresponding components from vector $\bar{c}$ will be added to each of the cell states. The additive character of updates prevents the vanishing gradients. Therefore, the cell vector $\bar{c}$ updates the long-term memory and the forget and input gates decide if it is necessary to reset the cells states from the previous time stamp forget the past and whether to increment the cell states from the previous time-stamp to incorporate new information into the long-term memory. At the last step the hidden states $\bar{h}_t^{(k)}$ are updated and equation (3) is a representation of information leakage from $\bar{C}_t^{(k)}$ to $\bar{h}_t^{(k)}$, i.e. we perform copying values from each of the p cell states to each of the p hidden states, depending on whether the output gate $\bar{o}$ is 1 or 0. Since $\bar{o}$ and other variables are continuous, only partial gating occurs with only a fraction of signal being copied from each cell state to the corresponding hidden state. To ensure a slow gradient flows decay the biases of the forget gates are initialized with large values. Expressing the hidden states in terms of the cell states with equation (3) ensures a simple derivation operation, that facilitates gradients' leakage into hidden states. All these improvements results in better organization of gradient flows in LSTM in comparison with RNNs [78].

PERFORMANCE METRICS

The following standard metrics are used to evaluate the performance of proposed forecasting approach in this work:

Mean Absolute Percentage Error (MAPE)

Mean Absolute Percentage Error (MAPE) is the percentage of average absolute error occurred and is defined as:

$$MAPE = \frac{100\%}{N} \sum_{n=1}^{N} \left| \frac{x_n - \hat{x}_n}{x_n} \right|$$

where N is the number of the samples. x_n is the actual value and $\hat{x}_n$ is the forecast value. MAPE is independent of the scale of measurement, however, is affected by data transformation. The

opposite signed errors does not cancel each other in summation due to the absolute values of addends. It is not directional, i.e. it does not provide any information about the direction of the error [57].

Mean Relative Error (MRE)

Mean Relative Error (MRE) represents the percentage of absolute average error occurred, and is defined as:

$$MRE = 100 \times \frac{1}{N} \sum_{n=1}^{N} \frac{|\hat{x}_n - x_n|}{\bar{x}}$$

where $\hat{x}_n$ and x_n are predicted and actual demand at hour h respectively, $\bar{x}$ is the mean demand of the day and N is the number of predicted hours. MRE is independent of the scale of measurement. The opposite signed errors does not cancel each other in summation due to the absolute values of addends. It is not directional, i.e. it does not provide any information about the direction of the error.

Mean Absolute Error (MAE)

Mean Absolute Error (MAE) measures average absolute deviation of forecasted values from original ones and is defined as:

$$MAE = \frac{1}{N} \sum_{n=1}^{N} |\hat{x}_n - x_n|$$

where $\hat{x}_n$ and x_n are predicted and actual demand at hour h respectively, $\bar{x}$ is the mean demand of the day and N is the number of predicted hours. MAE shows the magnitude of overall forecasting error. However, MAE lacks directionality, i.e. it does not provide any information about the direction of the error. It also depends on the scaling and transformation of the data. The forecast is good, if the values of MAE are small [57].

Root Mean Square Error (RMSE):

Root Mean Square Error (RMSE) estimates the average squared deviation of forecasted values and gives an overall idea of the error occurred during forecasting. It emphasizes the effect of large individual errors on the total forecast error and is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\hat{x}_n - x_n)^2}$$

where $\hat{x}_n$ and x_n are predicted and actual demand at hour h respectively, $\bar{x}$ is the mean demand of the day and N is the number of predicted hours. While calculating RMSE the opposite signed errors do not offset one another. It is also lacks directionality and sensitive to data scaling and transformations [57].

ELECTRICAL LOAD FORECASTING WITH LSTM AND CNN NETWORKS

In this work we deploy LSTM and CNN deep learning networks as separate experts for electricity load and price forecasting. Both experts consist of a sequential stack of hidden layers. LSTM-based expert (see Fig. 11) contains two hidden layers with 500 and 700 LSTM units and a dense layer (a layer with each input neuron connected to the output neuron). The dense layer has a linear activation, i.e. it performs linear operation on the layer's input vector, e.g. $f(x) = x$. Rectified linear unit (ReLU) is set as an activation function for both LSTM hidden layers. While experimenting with architecture we noticed that adding more layers flattens the forecasting performance. It might be the case of model overfitting [4].

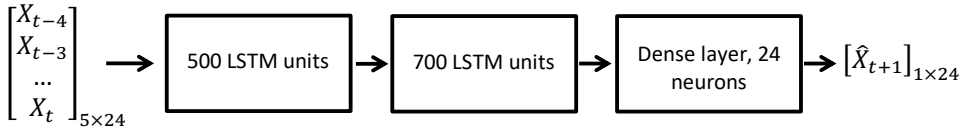


Fig. 11. Structural representation of LSTM deep learning network.

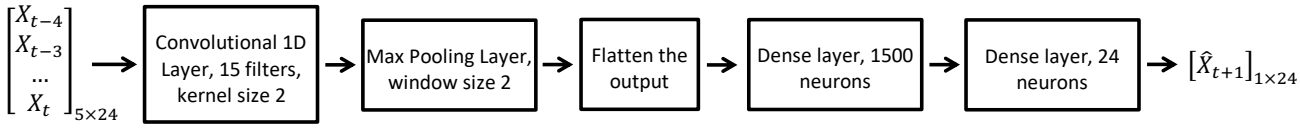


Fig. 10. Structural representation of CNN deep learning network.

The CNN-based expert (Fig. 10) learns a function that maps a sequence of past observations as input to an output observation. It consists of an input layer, that depicts the information on the past load, and the output layer represents the predicted future electricity load. The feature extraction is performed by the first convolution layer and the following pooling layer. The convolutional layer

incorporates 15 different windows (filters), each window (convolutional kernel) is of size 2. Convolving with the layer's input over a single dimension this kernel results into an output tensor. The sequential time series must be transformed into multiple examples from which the CNN can learn. Therefore, this layer takes 5 preceding days[1] as input and passes the extracted feature map through ReLU activation function after applying convolutional kernel.

The next comes the pooling layer, that performs downsampling to reduce the dimension of feature map transferred to the subsequent layers. Despite rigorous reduction the information in the feature map transformed from the previous layer is still enough to learn important dependences. The size of max pooling window is 2.

To perform the forecasting, we flatten the output from the pooling layer to create a fully connected structure between the pooling and output layer. The values are then transferred to two Dense layers, one is with 1500 neurons with ReLU activation and the other is a linear layer with 24 neurons i.e. equal to the dimensionality of the output.

Both networks are fit using the efficient Adam version of stochastic gradient descent and optimized using the mean squared error (MSE) loss function.

DEEP CNN FOR ELECTRICITY PRICE PREDICTION

The idea of multi-layer CNN for price prediction was inspired by the work of Singhal et al [12], where author consider a three layer back propagation network taking as inputs time indices, forecasted demand and historical time-lagged actual price information to predict the future price value. In this work we use deep convolutional neural network with three 1D convolutional layers, three intermediary pooling layers and dense layer as output layer (see Fig. 12). The choice of CNN is influenced by the fact, that electricity price data and electrical load are correlated, even though the relationship is non-linear [80]. The price is affected by a variety of external factors, including fluctuation in end-users' consumption, seasonality, weather conditions and other social, economic and political factors and the dynamics of the market, which result in high volatility of the price data. Most of these factors are intrinsically encoded in load and historical price data. Since CNNs are capable of learning complex relationships, we anticipate that CNN with multilayer architecture

[1] The choice of 5 preceding days is explained in the next chapter (see "Search for optimal parameters").

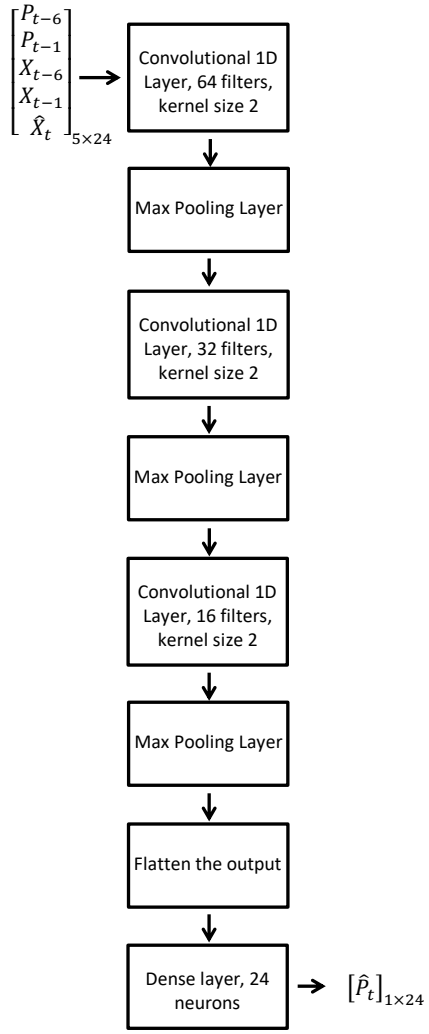


Fig. 12. Block diagram of proposed Deep Learning CNN for electricity price prediction.

will be able to learn an accurate mapping between load and price data. To our best knowledge, the deep learning CNN-based approach for predicting price from load values has not been addressed in the literature. The difference of our approach and the one proposed in [12] is that we are using deep CNN instead of backpropagation network of two hidden layers. Our algorithm takes less parameters than the one in [12] (5 vs 9) to avoid overfitting and to allow the hidden layers of CNN to infer subtle dependences from the inputs.

Since there is no theory yet for determining an optimal number of layers and neurons in the deep learning network, we adopted a trial-and-error process for developing architecture of multilayer deep CNN for mapping between load and price data. We assumed that the first two layers of the network will learn the general and daily trend from the load and historical price data. The third layer is accountable for capturing other external factors, such as seasonality, outages and the dynamicity of the market. Also, as it is pointed in [53], a network with more than one layers is capable of modeling the same dependence as the single-layered network, but with the less number of neurons. Therefore, we decided to adopt a simple multilayered architecture to avoid overfitting.

Experimenting with number of network layer demonstrated that three convolutional 1D layers for learning temporal dependencies and with three pooling layers for dimensionality reduction result in optimal performance of the model. The proposed network takes actual prices for a day before the day of interest P_{t-1} and the similar day a week ago P_{t-6} along with the actual loads for a day before the day of interest X_{t-1} and the similar day a week ago X_{t-6}. The fifth parameter is a predicted load for the day of interest $\hat{X}_t$. And the output is the predicted price $\hat{P}_t$ for the day of

interest t. The network has been trained on a data for one year preceding the day t, to learn the dependencies between input $\left[P_{t-6}, P_{t-1}, X_{t-6}, X_{t-1}, \hat{X}_t\right]^T_{5\times 24}$ and output vectors $[P_t]_{1\times 24}$, where P_t are actual price values. The choice of load and price a week before the day of interest is justified by the presence of weekly trend in price and load variation patterns. Also, we consider load and price a day before the day of interest due to the similarity of load profile with the predicted day. The addition of more historical data as price/load two/three/four weeks ago does not produce a significant effect to the forecasting accuracy, but might result in network overfitting. We choose ReLU as activation for all layers, since it is more efficient for multilayer networks. The significant drop in loss of proposed model has been detected after 10-th epoch, however, due to the periodic spikes in validation loss we decided to increase the number of training epochs till 100. In overall, the proposed architecture demonstrated optimal performance on the price dataset for New York City district (NYISO, [2]) and for the New South Wales (AEMO) [3].

EVALUATING PERFORMANCE OF CONVENTIONAL STATISTICAL METHODS

First we investigated the forecasting accuracy of traditional statistical models. We applied the ARIMA model for forecasting actual monthly electricity balance in Moldova provided by "Moldelectrica" Transmission System Operator [4]. We used the dataset of historical monthly electricity balance values for 50 months from 2016-th to 2020-th. The 66% of the dataset used for training the model and the remaining 34 for testing. The lag order was set to 5, the degree of differencing was 2 and the size of moving average window was 1. The results of prediction are presented in Fig. 13. As we can notice, the ARIMA model is capable of capturing the main trend, however, the accuracy of prediction is too low.

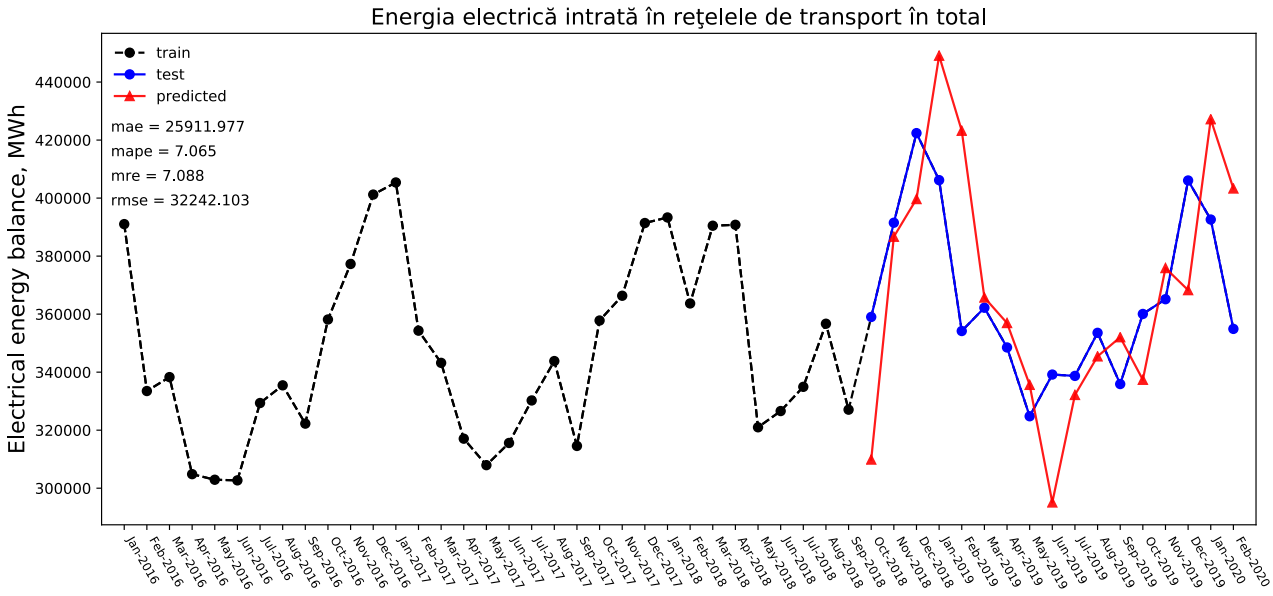


Fig. 13. Applying ARIMA model for forecasting electrical energy balance in Moldova.

We also applied ARIMA for predicting the hourly electricity load from the New York Independent System Operator (NYISO) for New York City [2] and from Australian Energy Market Operator (AEMO) [4] for New South Wales. Initially, we consider the same split for testing and training data as for Moldova dataset (66% training and 34% testing, roughly 48 hours to train in order to test for one day). The forecast was conducted in online manner – first using the data for the past 48 hours predict the result for next hour, than add the actual load for that predicted hour to the training data and predict the next hour. The mean absolute percentage error for the year of

2019 reached to 40.72%. To determine how the size of the training set affects the prediction accuracy we preformed 12-fold cross validation of ARIMA model, trained on the preceding 7, 14, 21 and 30 days. The errors, averaged over folds, are summarized in Table 1.

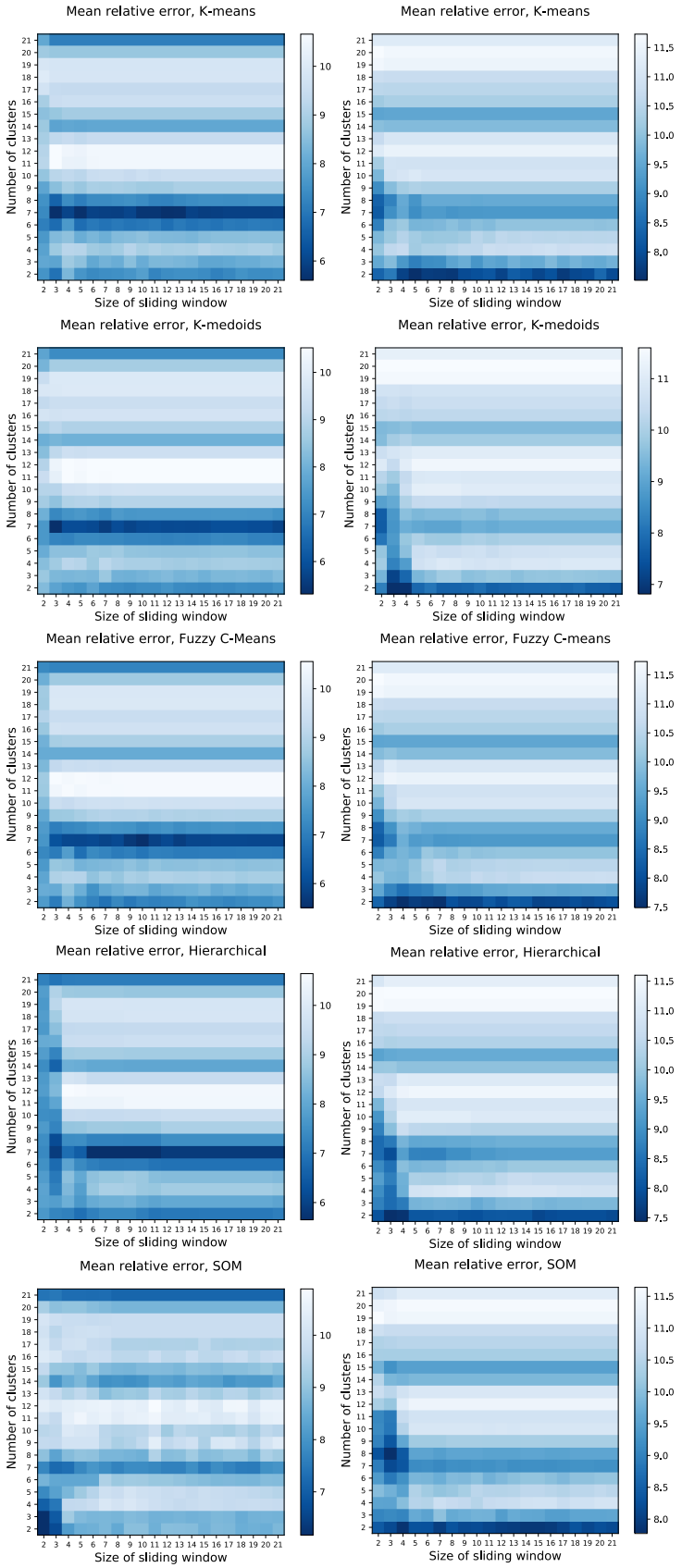
Size of training set (days)	MAE	MAPE	MRE	RMSE
2	2476.9	40.71	41.17	2588.44
7	126.68	2.07	2.11	155.37
14	80.87	1.36	1.36	103.23
21	74.91	1.26	1.26	95.95
30	74.63	1.26	1.26	95.66

Table 1. 12-fold cross-validation results for ARIMA model trained on sets with different sizes.

It is obvious that longer training sets result in better prediction of the ARIMA model. However, there is no significant improvement after the sizes exceeding 3 weeks of training data. Therefore, for future evaluations we decide to train ARIMA model on the hourly electricity load data for the past month (30 days).

SEARCH FOR OPTIMAL PARAMETERS

The main parameters for PSF-based clustering algorithms are the number of clusters and size of the sliding window. There are different approaches to determine these values. For example, in [17] authors deployed Silhouette index [20] to determine the number of clusters. Other authors, e.g. [24] along with Silhouette index used Dunn [21] and Davies-Bouldin indexes [22] or degree of fuzziness in case of Fuzzy C-means algorithm. While these indexes can be a great tool for estimating the number of clusters they can produce ambiguous estimates in some cases, as it has been pointed in in [23]. Besides, the other parameter, size of the window is still determined with 12-fold cross validation. Therefore, we decided to utilize the 12-fold cross validation to estimate the optimal number of clusters and window size via the grid search, one month in a fold. As a training set we have chosen time series of electricity load from the New York Independent System Operator (NYISO) for New York City [2] (NYC) and Australian Energy Market Operator (AEMO) [4] for New South Wales (NSW) for the year of 2015. For every day in a selected one-month-long fold we have iterated over 20 values for two parameters, namely number of clusters and size of sliding window, calculating four error measures (MAE, MAPE, MRE and RMSE) and then averaging over the fold. To obtain the final error measure for a given pair (number of clusters,



a. b.

Fig. 14. Grid search for best parameters a.) NYC, b.) NSW

window size), we averaged these monthly error measures across all 12 folds. The results are presented on Fig. 14.

From the results we can conclude that the best performance of k-means and k-medoids algorithms for NYC is achieved with 7 clusters and window of 3 days (the MRE values are also low for 5, 12 and 13 window sizes, but we are interested in the lowest values to ensure the best performance for shorter computation time). For New South Wales the optimal number of clusters is 2 with the window size of 5 for k-means and 4 for k-medoids. In the case of Fuzzy C-means algorithm, the MRE is lowest for 7 clusters for window of 10 days for NYC and 3 days for NSW. In Hierarchical clustering, the best performance is achieved with 6-7 days for NYC and 3 days for NSW with 7 clusters for both. Finally, for SOM, the lowest MRE is achieved at 2 days-long window and 2-3 clusters for NYC and 3 days with 8 clusters for NSW. These optimal parameters are summarized in Table 2. Note, that the best performance with cluster of 7 days can be explained by the presence of weekly patterns in electricity load

time series. In other words, the same day a week/two weeks/three weeks, etc. ago is important for predicting the load for a given day. This assumption is confirmed by the observation of relatively low MRE values for clusters of 14 days (bi-weekly patterns) and 21 days (tri-weekly patterns) (the darker horizontal strips on the heatmaps). Even in the case of SOM, even though the best performance is achieved for relatively low numbers of clusters and window sizes, the periodic drop in MRE at clusters of 7, 14 and 21 days is also observed for some window sizes (see Fig. 14). Thus, clustering algorithms can infer weekly variations in electricity load.

		k-means	k-medoids	Fuzzy C-means	Hierarchical	SOM
NYC (NYISO)	*k*	5	7	10	6	3
	w	7	3	7	7	2
NSW (AEMO)	*k*	2	2	7	7	8
	w	5	4	3	3	3

Table 2. Optimal values of *k* and *w* for load forecasting with clustering algorithms.

For two deep-learning networks-based experts, i.e. LSTM and CNN the parameters selection is more complicated. First of all, the number of epochs has to be set properly to prevent overfitting. In order to determine the number of epochs we trained LSTM and CNN-based experts on the data for year of 2014 and tested on 2015-th. Fig. 15 and Fig. 16 demonstrate how one of the error metrics, namely, mean absolute percentage error (MAPE), changes with the number of epochs. As we can notice for LSTM network MAPE both for training and validation data flattens after first 100 epochs. For the CNN-based expert gradually drops down till approximately 220 epochs. Therefore, it does not make sense to use more than 100 epochs for LSTM and more than 220 for CNN.

To determine the number of LSTM units in both hidden layers of LSTM network, and number of filters with number of neurons in dense layer for CNN, we employed greed search for these parameters. Both networks were trained on entire year of 2014 and validated on time series data for 2015-th. All four metrics were calculated for an optimal number of epochs (100 for LSTM and 220 for CNN) and averaged along the epochs. The results are presented on Fig. 18 and Fig. 17. Note, that the similar heatmaps could be plotted for other metrics. MAPE is used just for convenience, the identical conclusions about optimal parameters can be reached with other metrics.

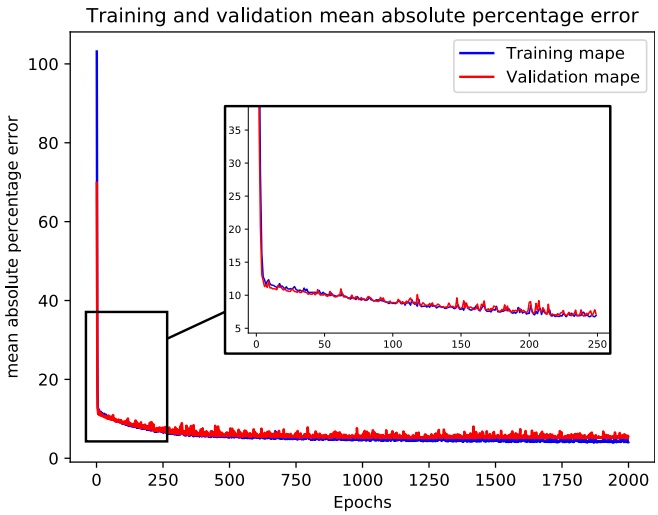


Fig. 15. Determining the number of epochs for CNN

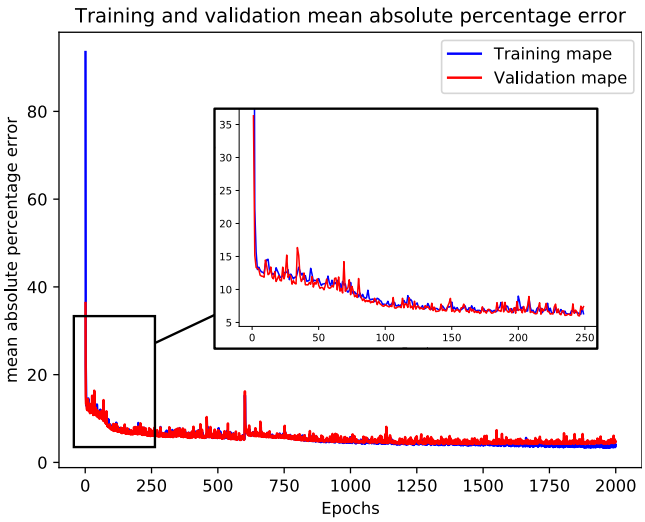


Fig. 16. Determining the number of epochs for LSTM network

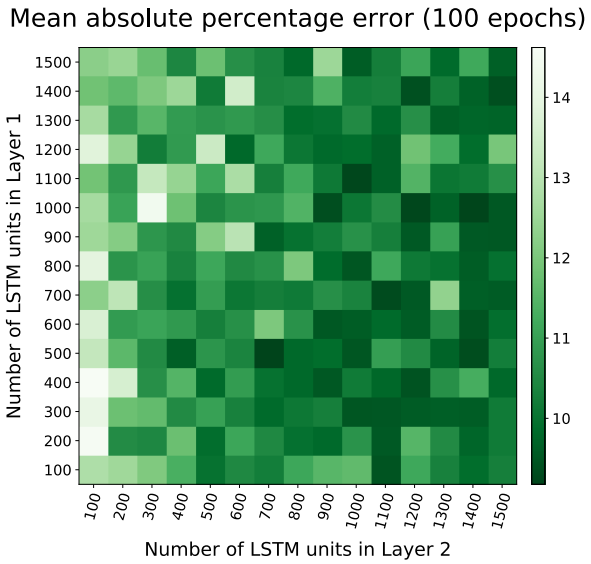


Fig. 18. Mean absolute percentage error values averaged over 100 epochs for validation set, calculated for different numbers of LSTM units first and second hidden layers.

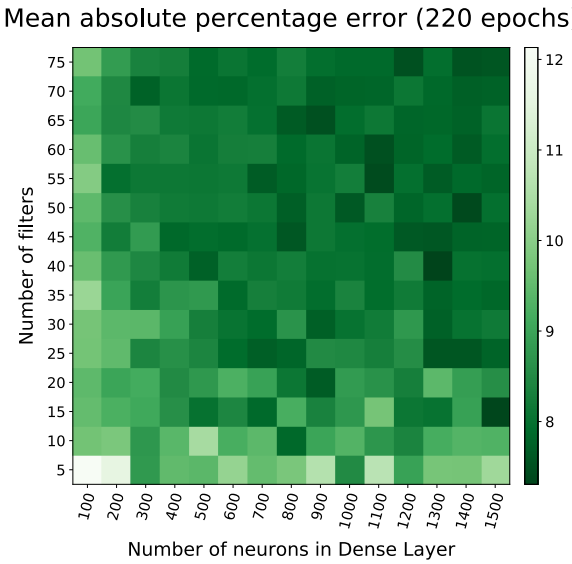


Fig. 17. Mean absolute percentage error values averaged over 220 epochs for validation set, calculated for different filters and number of neurons in Dense layer of CNN network.

From these heatmaps it is easy to conclude that the LSTM achieves the lowest error with 500 LSTM units in first hidden layer and 700 second. For CNN, the minimal MAPE value is achieved

with 15 filters and 1500 neurons in the dense layer. These parameters were used in LSTM and CNN-based experts for electricity load prediction.

Preceding days (n)	MAE	MAPE	MRE	RMSE
5	233.81	3.71	3.83	277.72
7	231.20	3.70	3.81	275.31
14	230.91	3.67	3.79	273.42
21	230.05	3.63	3.78	271.83

Table 3. 12-fold cross-validation results for LSTM model for different preceding days.

We also consider how the forecasting performance depends on the number of preceding days at the input layer (i.e. the size of the vector $\left[X_{t-(n-1)}, \dots, X_t\right]^T$ on Fig. 11 and Fig. 10). The results of 12-fold cross-validation of LSTM model for the year of 2019 with 500 units in first and 700 units in second layer are shown in Table 3. As it is easy to notice there is not so huge difference between errors for different input vector sizes, however, computing with 21 days on input layers takes almost three times longer than with 5 days on MacBook Pro laptop with 2.7 GHz Intel Core i5 processor and 8 Gb 1867 MHz DDR3 RAM, running macOS Catalina. Therefore, to reduce the computing time we decided to supply the data for $n = 5$ preceding days to the input layer both for LSTM and CNN networks.

In general, there is no theoretical approach for selection of number of neurons in hidden layers is known yet. The common logics suggests that if we choose too few hidden neurons, we will get a high training error and high generalization error due to under-fitting. In contrast, if there are too many hidden neurons, the training error would be low, but model will fail to generalize due to over-fitting.

ANALYSIS OF CLUSTERING PATTERNS

We analyzed how the number of clusters affects the segregation of the dataset. The load data representing each cluster differs from each other by shape and amplitude and the number of days per cluster differ as well. We used this observation to evaluate how the segregated daily load and price data instances are distributed by months within one cluster.

Considering the heatmaps in Table 4 it is easy to observe that there is a striking difference in distribution of number of days within each cluster. For $k = 2$ cluster 1 apparently captures the electricity consumption for fall, winter and spring months but second cluster represents the

summer months. The similar pattern repeats for all clustering algorithms, both clusters capturing either fall-winter-spring or summer interchangeably. For $k = 3$ the cluster 2 of k-means and k-medoids mostly represents summer months from June to September. Opposite to it, the third cluster contains daily loads from spring and fall months. Moreover, the pattern for distribution of days across different clusters is identical for k-means and k-medoids in the case of 2 and 3 clusters, meaning that centroids, generated by k-means are very close to the actual data points, selected as medoids by k-medoids. Fuzzy C-means and Self-organizing maps for $k = 3$ also achieve similar distributions, placing all summer days in cluster 3, and the Hierarchical clustering places summer days in cluster 1. In the case of $k = 4$ similar pattern preserves, but clusters capture more subtle differences between days now. For example, k-means algorithm now has two clusters (2 and 4) to represent summer months, however, cluster 2 more reflects the hottest month of the year in New York (July), while days in cluster 4 distribute more-or-less uniformly across the warmer months between June and September, slightly peaking in July-August. For SOM clusters 2 and 4 encode cooler and warmer days of the year correspondingly and in they sense they can be considered reciprocal. In summary, regardless the index of the cluster, we observe very similar patterns in the case of lower numbers of clusters across all clustering techniques.

In the case of higher number of clusters the differences between clustering techniques become more pronounced, each algorithm capturing its own subtle differences between daily electrical load consumption for different months. For instance, for $k = 5$ clusters 2 and 4 in k-means represent electricity consumption in summer, but cluster 4 is capturing electricity load in warmer months of July and August. Cluster 3 represents load for spring and late fall, while there is hard to extract any pattern in cluster 5.

It is also worth to note that increasing k results in replication of some clusters by others. For example, in k-medoids ($k = 6$), clusters 5 and 6 are encoding electrical load for March to May and for October-November. It can be because of too close boundaries between discovered neighboring clusters, i.e. the separation is not optimal. This observation emphasizes the importance of determining the optimal number of clusters before predicting.

Similar conclusion can be achieved for New South Wales. For example, for all algorithms at $k = 2$ the first cluster represents either Fall months from March to May or Spring months from September to December accordingly. The second cluster represents either summer (k-medoids, hierarchical and SOM) or winter months (k-means, Fuzzy C-means). Interestingly, the assignment

of January in the second cluster contradicts the cluster's seasonality (e.g. in k-means for $k = 2$ it is considered as winter month along with June, July and August). It could be explained by similarity in consumption patterns in January and winter months. Increasing k results in assigning a separate cluster to January, for instance in case of k-means with $k = 3$ the second cluster contains the highest number of days in January (see Table 5).

For electricity prices the periodicity of the data is not as pronounced as in the electrical load time series. There are some traces of seasonal trend and some patterns of periodical fluctuations, but in most cases the appearance of the data depends on the situation in the electricity market. Therefore, the distribution of number of days in each cluster by months in most cases lacks the clear pattern. For example, the heatmap for price data for New South Wales in Table 7 shows that in case of two clusters all algorithms discern between spring/fall and summer/winter months, while for NYC (see Table 6) this division is not that clear, for example, k-means and fuzzy C-means assign both clusters the equal number of days for November and December. Almost all algorithms clearly discriminate the prices in July for NYC. For instance, k-means with $k = 2$ assigns only a few days in July to cluster 1, while cluster 2 containing most of the days. The similar pattern is observed for January, February and March. This three months are clearly captured by almost all algorithms with different k values. The key for explanation can be found in seasonal temperature fluctuations. January, February and March are the coolest months of the year and July is the hottest month. Apparently, the electricity consumption for heating /cooling is high, thus, the price is also high and the clustering algorithms are capturing this trend. The similar picture is observed for the NSW, where the coolest days last from June to August and the January is the hottest month. The fact that the load data for NYC and NSW follows the similar pattern confirms this assumption. Also, it is possible to notice the similarity between electricity load and price clustering patterns, which justifies our approach of using load data to forecast the electricity prices.

In overall, the clustering algorithms clearly reflect the seasonal changes in electrical power consumption and electricity price. Thus, clustering approach is an important tool for interweaving these intrinsic seasonal patterns into future forecasts.

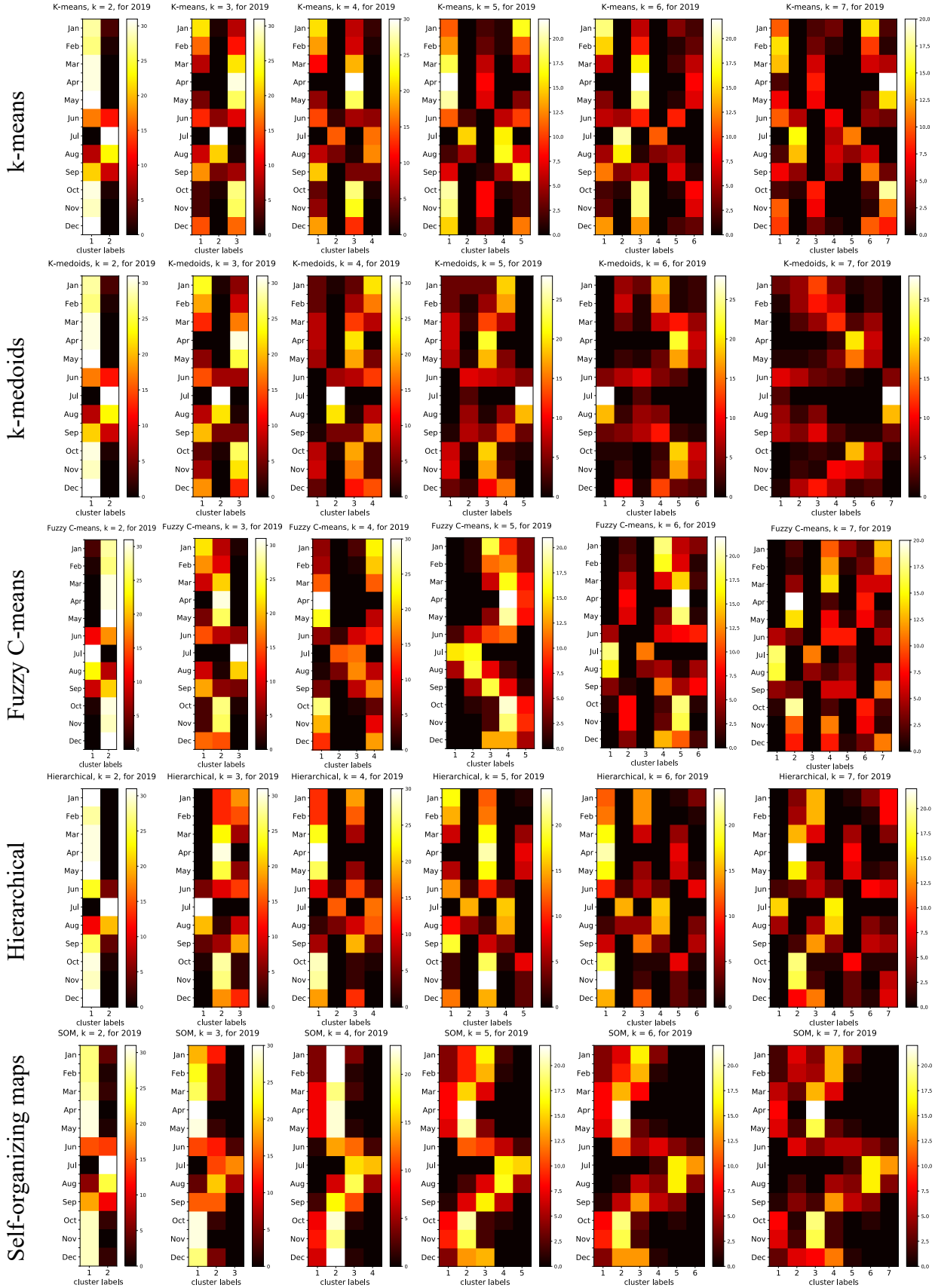


Table 4. Clustered daily electricity load values, distributed by months within each cluster for each of the 5 clustering algorithms (for NYISO, 2019).

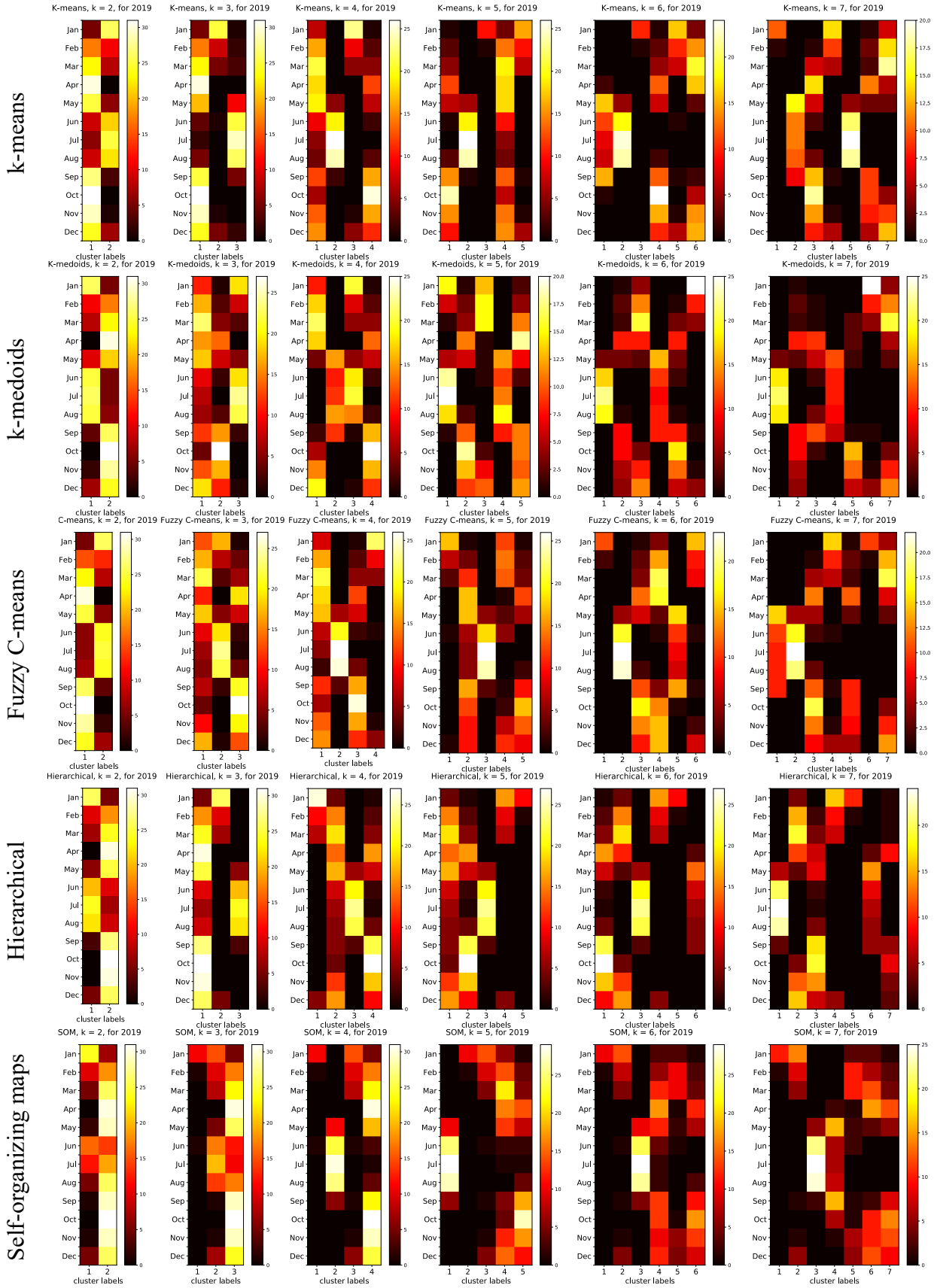


Table 5. Clustered daily electricity load values, distributed by months within each cluster for each of the 5 clustering algorithms (for NSW, 2019).

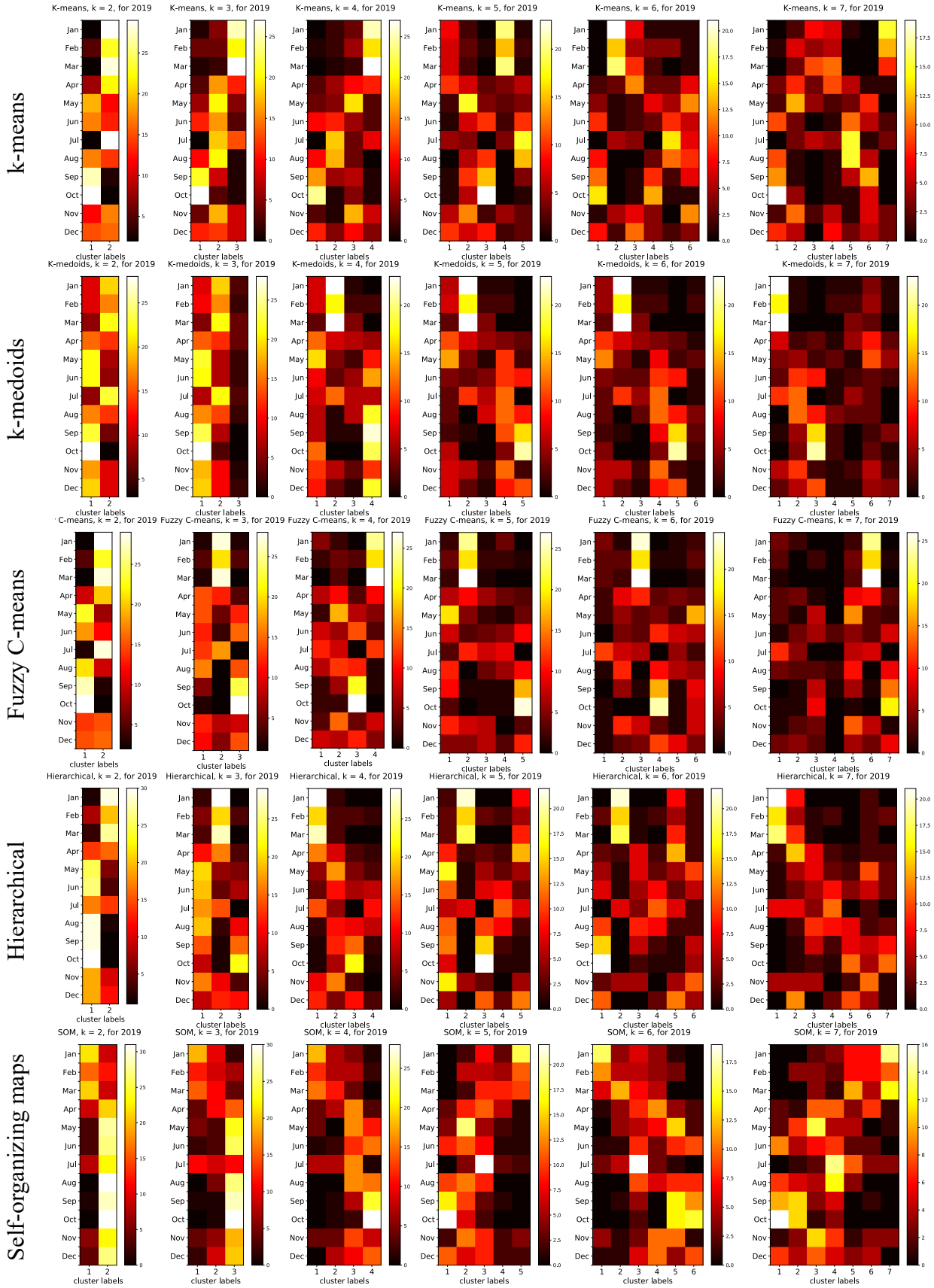


Table 6. Clustered daily electricity prices, distributed by months within each cluster for each of the 5 clustering algorithms (for NYC, 2019).

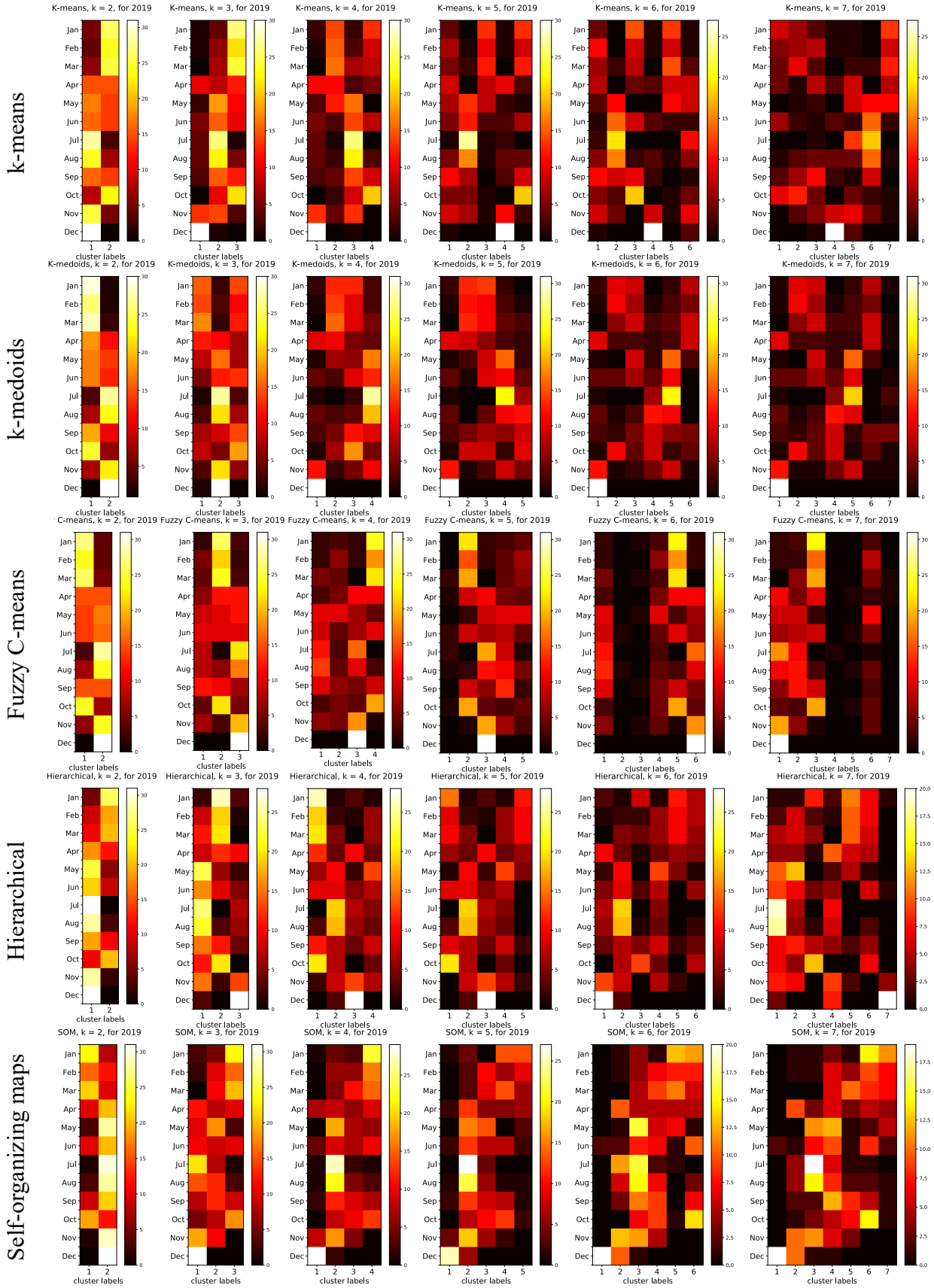


Table 7. Clustered daily electricity prices, distributed by months within each cluster for each of the 5 clustering algorithms (for NSW, 2019).

First we run every model separately to evaluate the average performance. The electrical load data were obtained from the New York Independent System Operator (NYISO) for New York City [2] and Australian Energy Market Operator (AEMO) [4] for New South Wales (NSW). The frequency of load recordings in the original dataset is 12 records per hour (each five minutes). To reduce the size of the data we decided to shrink it to one record per hour, leaving only the records collected at the beginning of each hour, thus reducing it to 24 records daily. On the data preprocessing stage the initial trend in the data is suppressed by normalizing the data across all the records:

$$X_j = \frac{X_j}{\frac{1}{N}\sum_{i=1}^{N} X_j}$$

where X_j is the load for the j −th hour of the day and N the number of samples considered per day. In this case, $N = 24$ since each sample represents one hour of the day [17], [27]. However, in order to calculate the denominator we need to know the data for all 24 hours, which can impose additional limitations. Therefore, we adopted a unity-based normalization approach [18], allowing to map all values to the interval $[0,1]$:

$$X_j' = \frac{X_j - \min(X_j)}{\max(X_j) - \min(X_j)},$$

where X_j' is the normalized load for the j −th hour of the day, $i = 1, \dots, N$.

Another difference from the original PSF algorithm is that we discounted outliers from the predicted day value calculations. Using 1.5 of interquartile range (IQR) as a criteria, all the days whose absolute values exceeded 1.5 IQR (i.e. if they have found at the head of the similar sequence preceding the day of interest, they have been removed from the average). To demonstrate the outliers can affect the accuracy of prediction, we applied k-means-based expert (original LBF) to predict the electricity load with and without outliers for the year of 2017 (using the optimal parameters found in previous paragraph). While the average values of MAE, MAPE, MRE and RMSE for dataset with outliers were 740.68, 11.78, 11.92 and 812.74 correspondingly, the error values slightly decreased for dataset with removed outliers (MAE = 681.33, MAPE = 10.93, MRE = 11.04 and MSE = 751.65). This observation motivated us to adapt the outlier detection for the

future analyses. In general, the outlies constitute about 5% of the dataset and usually, these are values exceeding the upper bound of 1.5 IQR. After outlier removal we applied each expert algorithm for predicting electricity load values for every day in the year of 2019. The results for are summarized in Table 8 for NYC and in Table 9 for NSW.

		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Average
ARIMA	MAE	77.51	70.98	69.02	70.42	69.36	84.55	89.49	83.75	73.88	66.81	69.78	70.02	**74.63**
	MAPE	1.3	1.22	1.24	1.36	1.31	1.37	1.14	1.17	1.19	1.28	1.29	1.23	**1.26**
	MRE	1.31	1.23	1.25	1.36	1.3	1.35	1.13	1.17	1.19	1.26	1.3	1.24	**1.26**
	RMSE	98.44	93.47	89.45	91.87	90.05	107.59	114.48	105.6	93.46	85.95	87.87	89.64	**95.66**
k-Means	MAE	366.5	477.16	393.83	270.1	344.58	742.34	800.73	761.78	658.89	406.77	254.08	399.74	**489.71**
	MAPE	6.17	8.15	7.11	5.6	6.51	11.22	9.51	11.48	10.79	7.76	4.64	7.13	**8.01**
	MRE	6.17	8.2	7.2	5.5	6.51	11.35	9.66	11.37	10.79	7.78	4.73	7.16	**8.04**
	RMSE	405.41	506.39	436.08	304.59	394.86	793.3	870.92	817.4	723.63	456.89	284.27	426.92	**535.06**
k-Medoids	MAE	470.4	583.4	461.92	380.37	467.16	692.05	843.1	773.55	758.88	486.84	335.79	573.06	**568.88**
	MAP	7.87	10.15	8.34	7.45	8.7	10.35	10.24	11.54	12.28	9.17	6.17	10.03	**9.36**
	MRE	8	10.2	8.5	7.63	8.89	10.46	10.25	11.5	12.25	9.37	6.3	10.22	**9.46**
	RMS	522.59	639.91	524.27	438.32	538.62	753.4	906.12	846.12	848.1	560.56	383.45	641.83	**633.61**
Fuzzy C-Means	MAE	314.58	373.33	260.5	141.73	209.59	736.83	799.34	648.96	639.28	338.99	222.36	405.49	**424.25**
	MAPE	5.24	6.49	4.84	2.8	3.7	11.31	9.6	9.88	10.5	6.36	4.13	7.26	**6.84**
	MRE	5.21	6.47	4.82	2.79	3.78	11.34	9.75	9.65	10.5	6.33	4.14	7.28	**6.84**
	RMSE	344.92	398.64	280.8	157.85	244.33	785.25	885.81	706.71	701.57	384.92	249.12	429.01	**464.08**
Hierarchical	MAE	325.72	366.67	236.45	184.98	242.29	736.83	798.69	642.33	638.29	344.59	255.37	377.3	**429.13**
	MAPE	5.41	6.38	4.38	3.69	4.43	11.31	9.38	9.64	10.48	6.48	4.73	6.76	**6.92**
	MRE	5.38	6.36	4.35	3.65	4.44	11.34	9.64	9.52	10.49	6.43	4.76	6.78	**6.93**
	RMSE	361.25	392.8	257.38	198.9	274.47	785.25	883.79	703.26	700.86	389.15	282.14	399.34	**469.05**
SOM	MAE	477.22	395.56	360.88	323.12	382.64	596.01	845.57	625.08	812.82	475.14	289.06	420.57	**500.31**
	MAPE	8	6.79	6.57	6.7	7.38	9.23	10.28	8.86	13.18	9.02	5.36	7.38	**8.23**
	MRE	8.05	6.86	6.66	6.57	7.28	9.42	10.35	8.93	13.25	8.93	5.43	7.52	**8.27**
	RMSE	524.91	438.3	410.26	359.05	431.54	652.36	905.77	678.44	880.36	526.05	328.17	468.49	**550.31**
LSTM	MAE	195.5	161.81	125.14	121.62	212.04	333.75	407.69	333.67	401.21	189.24	140.78	183.3	**233.81**
	MAPE	3.3	2.73	2.24	2.3	3.75	5.1	5	4.67	6.21	3.33	2.62	3.25	**3.71**
	MRE	3.36	2.8	2.27	2.39	3.89	5.3	5.18	4.83	6.45	3.48	2.67	3.35	**3.83**
	RMSE	226.17	193.29	146.86	146.81	248.84	396.16	482.98	401.92	480.11	224.12	168.18	217.24	**277.72**
CNN	MAE	297.94	214.14	179.45	185.19	308.45	379.31	596.96	405.64	612.88	371.35	208.59	246.41	**333.86**
	MAPE	4.89	3.71	3.28	3.54	5.56	6	7.34	5.71	9.87	6.92	3.92	4.36	**5.42**
	MRE	5.03	3.78	3.29	3.63	5.64	6.12	7.52	5.79	10.01	7	3.96	4.44	**5.52**
	RMSE	351.92	251.78	210.24	224.55	353.03	439.41	695.6	468.23	714.89	436	243.24	282.82	**389.31**

Table 8. Electricity load prediction errors for each algorithm after 12-fold cross validation averaged for every fold (month) (NYC, 2019).

We can notice from the table that the average lowest errors for non-statistical methods are achieved predicting with LSTM. CNN-based expert also demonstrates quite a good performance. The worst performing expert is k-medoids, with MAPE exceeding 9%. For most of the algorithms September is the hardest month to predict load. It could be explained by change in people's daily routine, when significant number of population return from summer holidays. Also in September most of schools/universities resume their work, that might result in unpredictable patterns of electricity

consumption. Interestingly, that the early spring months, March and April are the easiest to predict. Below, we demonstrate best and worst prediction results for each algorithm along with the predictions from LSTM and CNN (see Table 10 and Table 11).

		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Average
ARIMA	MAE	144.25	153.39	333.43	157.26	222.76	250.33	368.88	285.79	246.28	148.54	152.6	142.85	**217.2**
	MAPE	1.63	1.89	4.25	2.13	2.75	2.83	4.13	3.27	3.2	2.07	2.07	1.87	**2.67**
	MRE	1.6	1.87	4.32	2.14	2.81	2.89	4.33	3.37	3.27	2.06	2.05	1.86	**2.71**
	RMSE	179.28	191.32	391.25	200.86	287.41	316.82	480.56	364.44	307.28	181.76	185.1	175.4	**271.79**
k-Means	MAE	1092.4	876.87	618.57	371.06	536.46	714.36	600.21	745.38	709.21	412.4	449.98	783.13	**659.18**
	MAPE	11.34	10.47	7.75	5.22	6.56	8.21	7.05	9.02	9.71	5.93	6.04	9.92	**8.1**
	MRE	11.98	10.77	7.92	5.2	6.7	8.31	7.06	8.98	9.63	5.89	6.15	10.33	**8.24**
	RMSE	1329.5	1036.9	731.64	446.29	637.39	820.54	703.19	879.27	834.48	491.04	549.73	948.4	**784.04**
k-Medoids	MAE	1104.8	814.12	617.49	371.4	515.87	653.97	632.37	671.74	720.78	428.05	474.26	655.52	**638.37**
	MAP	11.4	9.63	7.72	5.23	6.37	7.58	7.37	8.18	9.86	6.16	6.39	8.22	**7.84**
	MRE	12.1	9.96	7.9	5.21	6.49	7.66	7.41	8.16	9.8	6.12	6.49	8.58	**7.99**
	RMS	1372.6	966.09	732.4	448.55	620.49	759.14	746.55	778.43	837.13	509.22	577.23	808.42	**763.03**
Fuzzy C-Means	MAE	966.95	833.39	603.08	544.16	621.77	595.28	574.84	586.04	778.23	532.75	533.39	704.47	**656.2**
	MAPE	10.19	9.89	7.6	7.4	7.86	6.9	6.67	7.08	10.4	7.49	7.07	8.83	**8.11**
	MRE	10.61	10.24	7.77	7.46	7.97	6.99	6.74	7.12	10.44	7.54	7.24	9.25	**8.28**
	RMSE	1118.0	985.68	714.56	629.02	710.91	682.35	666.1	688.07	893.63	631.23	643.34	884.61	**770.63**
Hierarchical	MAE	960.63	909.4	619.15	502.92	565.71	615.15	547.73	600.89	698.68	544.02	573.09	703.46	**653.4**
	MAPE	10.16	10.85	7.8	6.87	7.16	7.09	6.38	7.36	9.4	7.57	7.6	8.96	**8.1**
	MRE	10.57	11.24	8	6.91	7.24	7.17	6.44	7.37	9.41	7.64	7.75	9.28	**8.25**
	RMSE	1115.3	1075.7	744.68	582.49	645.44	698.27	633.99	698.38	799.4	637.47	695.62	849.28	**764.68**
SOM	MAE	905.91	898.21	663.62	519.72	586.79	614.3	583.49	591.58	772.53	530.28	576.37	743.34	**665.51**
	MAPE	9.65	10.74	8.33	7.07	7.37	7.19	6.76	7.15	10.35	7.42	7.63	9.38	**8.25**
	MRE	10.06	11.09	8.52	7.13	7.48	7.26	6.83	7.19	10.39	7.48	7.78	9.79	**8.42**
	RMSE	1065.3	1060.5	788.91	602.82	682.33	709.42	674.44	690.1	877.39	623.73	688.35	914.42	**781.48**
LSTM	MAE	690.34	415.62	355.86	232.01	206.82	310.63	236.71	315.48	313.12	288.03	366.88	358.5	**340.83**
	MAPE	7.17	4.91	4.33	3.16	2.54	3.52	2.7	3.71	4.15	3.97	4.78	4.48	**4.12**
	MRE	7.59	5.14	4.52	3.2	2.6	3.62	2.76	3.75	4.19	4.01	4.94	4.71	**4.25**
	RMSE	838.25	519.85	455.81	279.06	257.13	380.72	289.07	380.44	375.05	361.18	469.31	460.77	**422.22**
CNN	MAE	918.42	707.51	407.51	345.78	284.94	367.57	272.07	450.52	506.35	400.64	513.15	612.18	**482.22**
	MAPE	9.39	8.33	5.06	4.78	3.53	4.16	3.14	5.31	6.77	5.59	6.9	7.92	**5.91**
	MRE	9.94	8.66	5.23	4.79	3.6	4.25	3.2	5.36	6.81	5.62	7	8.11	**6.05**
	RMSE	1136.1	877.8	509.57	421.74	351	442.95	328.98	555.58	601.48	487.94	625.22	740.88	**589.94**

Table 9. Electricity load prediction errors for each algorithm after 12-fold cross validation averaged for every fold (month) (NSW, 2019).

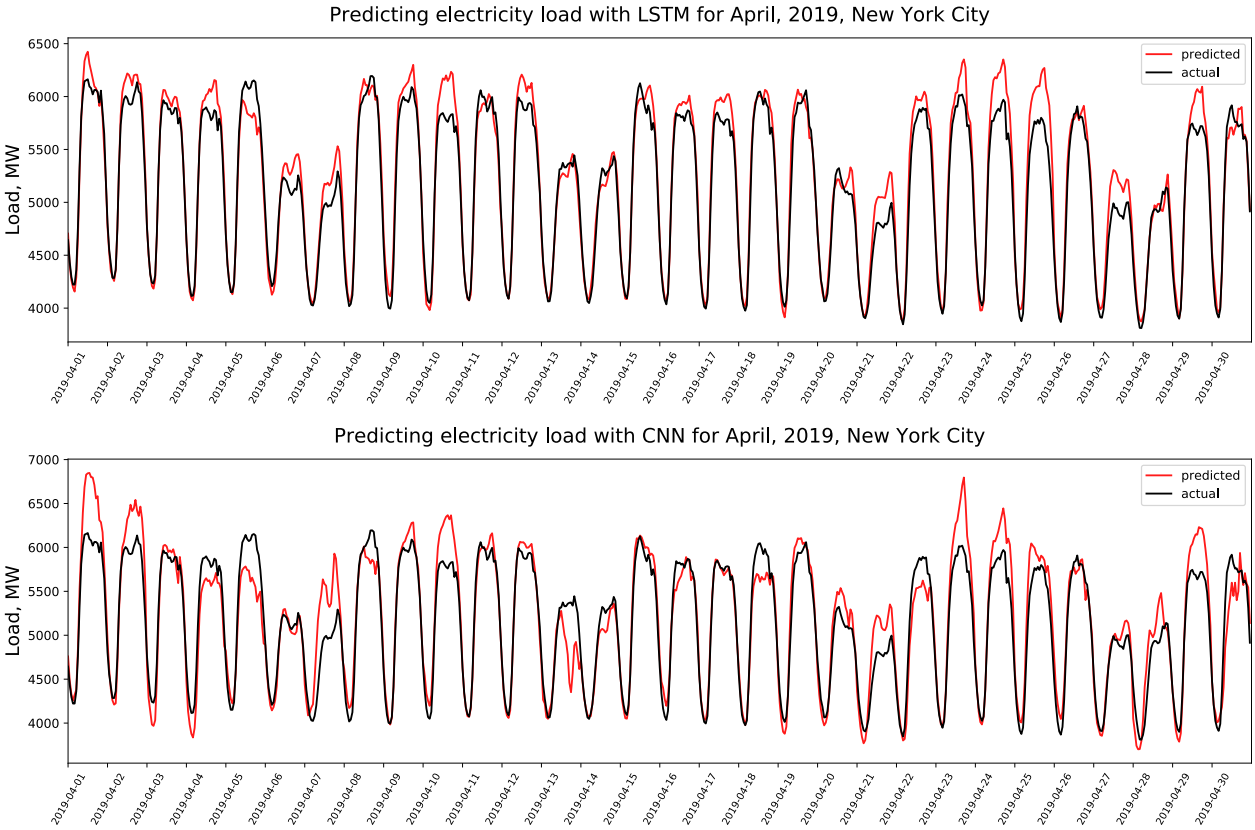


Fig. 19. Electricity load, predicted for April 2019 for NYC

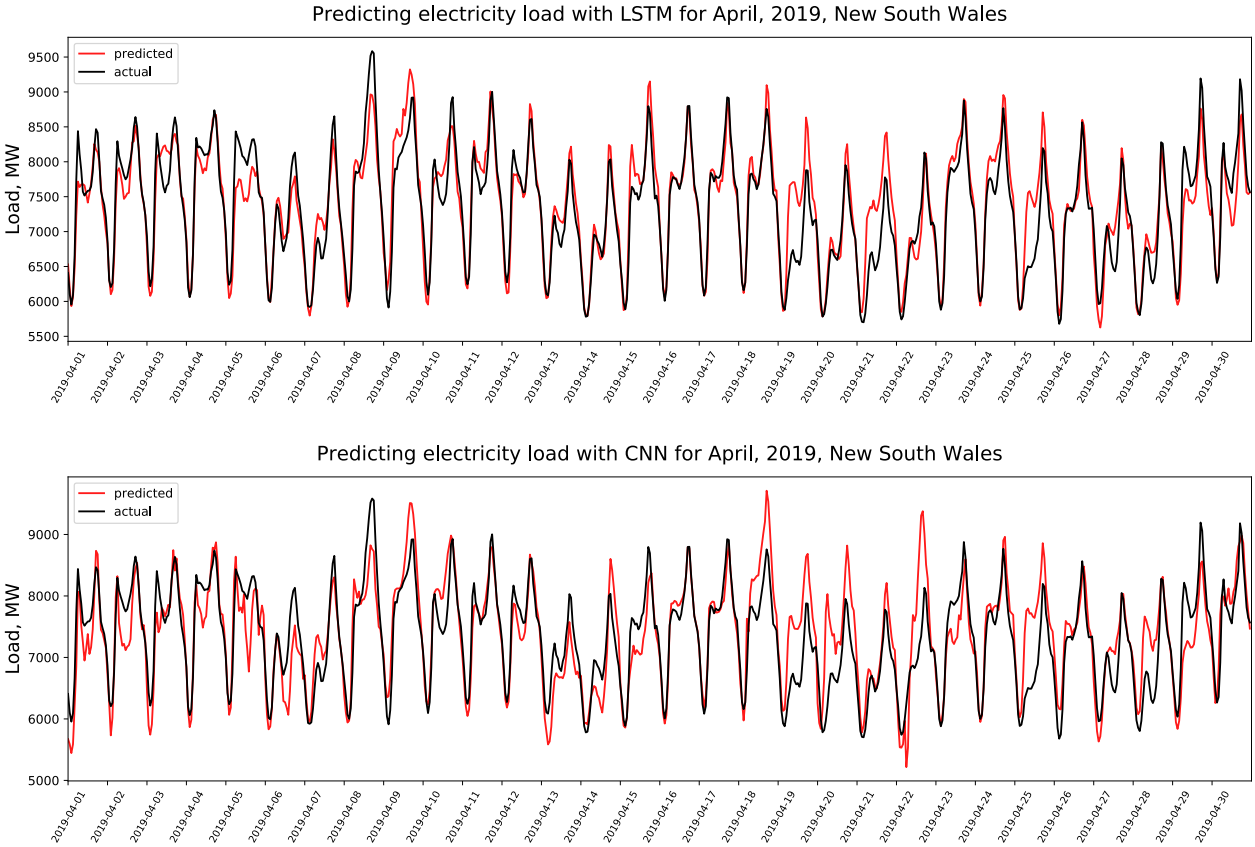
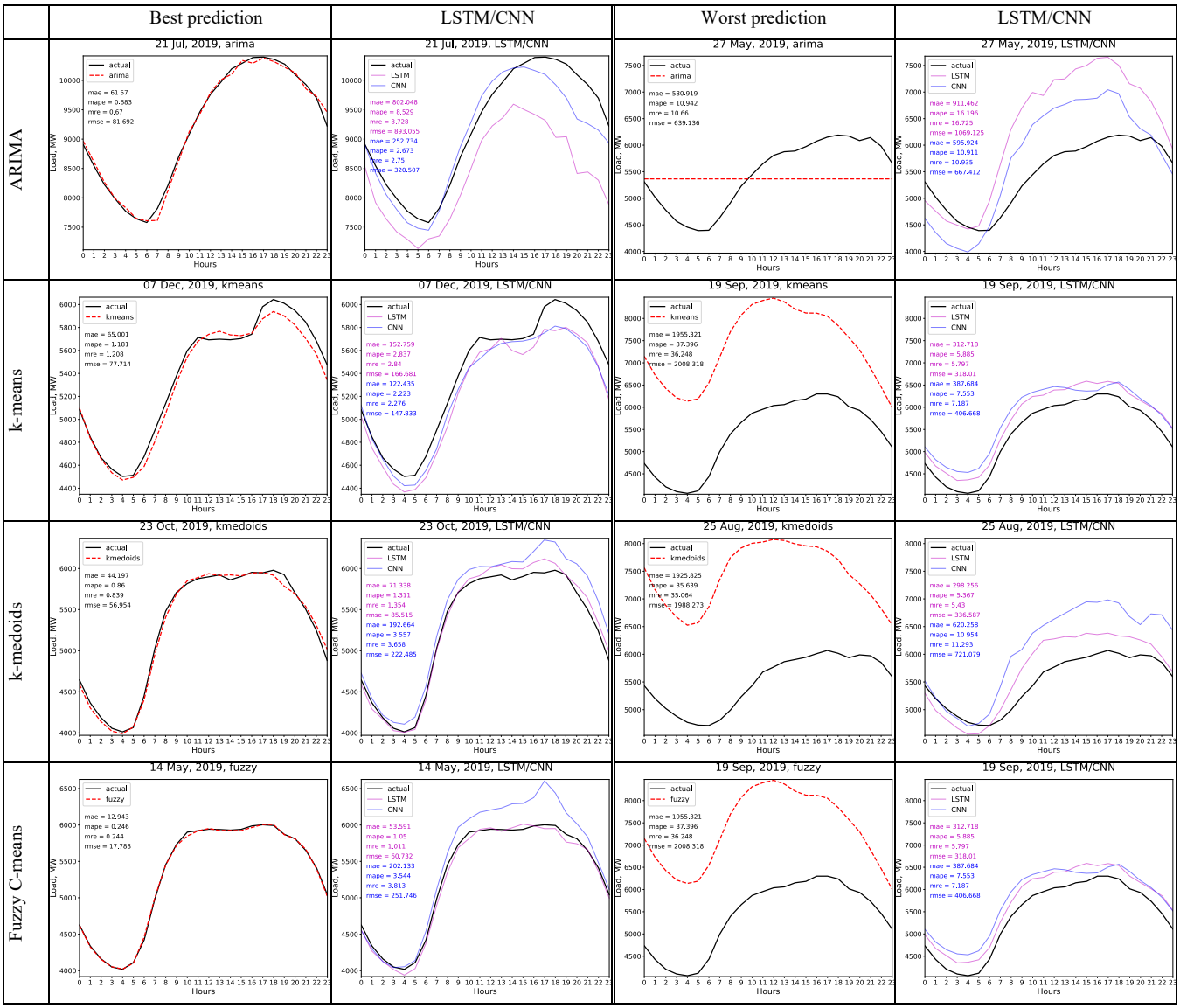


Fig. 20. Electricity load, predicted for April 2019 for NSW

Despite the fact that error metrics for ARIMA in average achieve lowest values in comparison with other methods, in majority of cases predictions with ARIMA are unreliable since the model fails to converge on non-stationary data while performing likelihood function optimization. It is clearly illustrated in Table 10for the worst case of ARIMA prediction. Therefore, while ARIMA can achieve high accuracy on stationary data, it should be applied to load data with caution for its sensitivity to non-stationary fluctuations in electricity load time-series. Fig. 19 and Fig. 20 demonstrate predictions for one month for (April, 2019) for NYC and NSW.

In general, LSTM is better suited to handle the spikes than CNN This can be explained by the ability of LSTM networks to capture intrinsic temporal dependences in the time-series data. On average the LSTM-based expert achieved better performance than the rest of the non-statistical forecasting methods considered in this work.



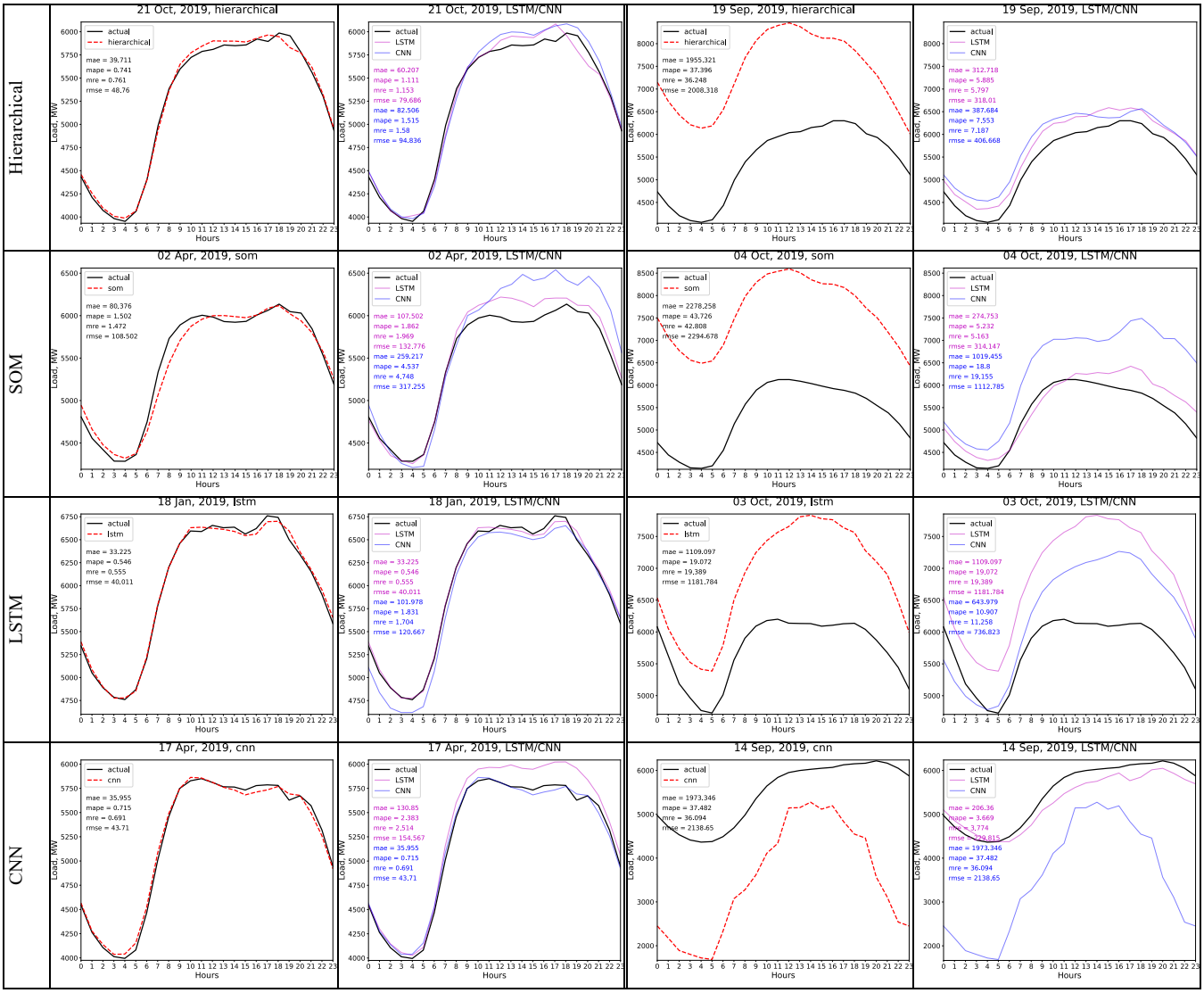
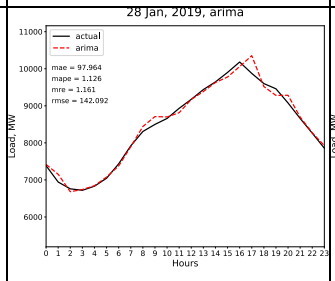
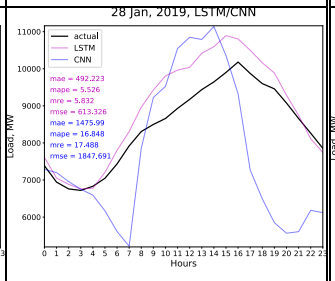
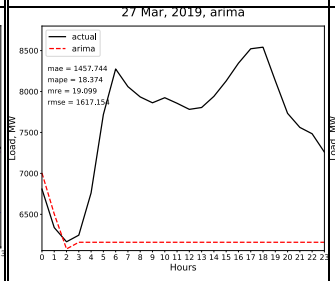
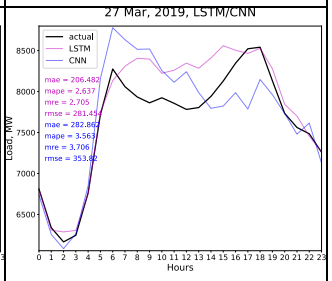
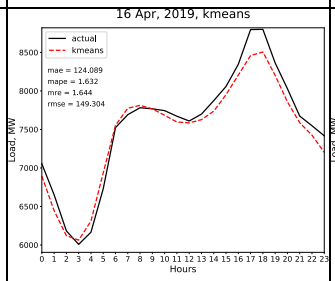
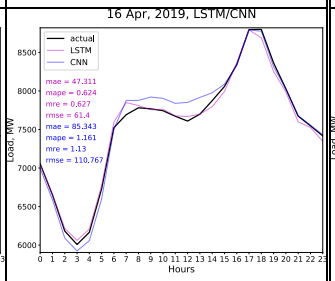
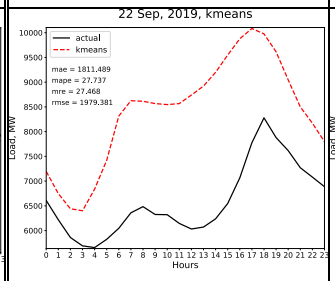
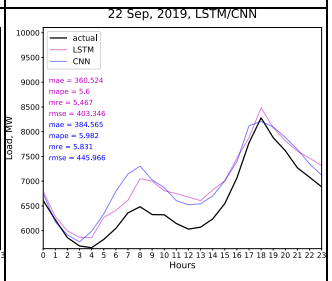
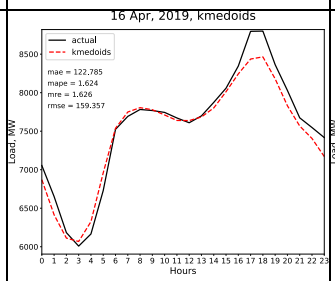
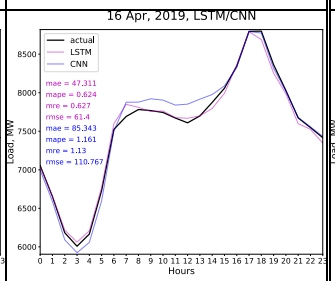
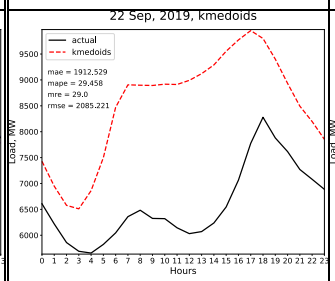
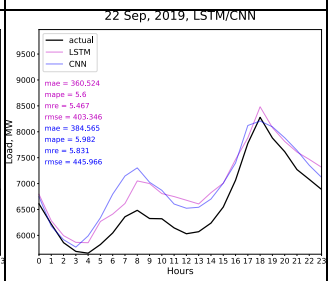
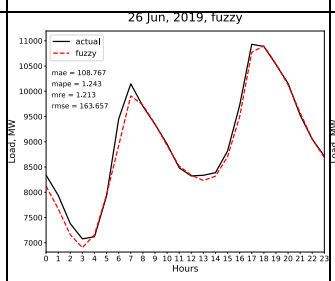
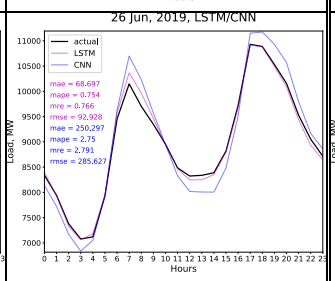
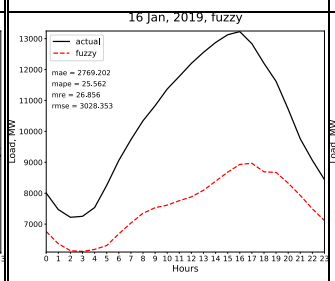
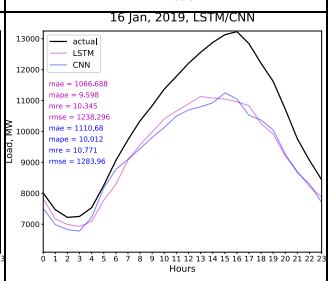
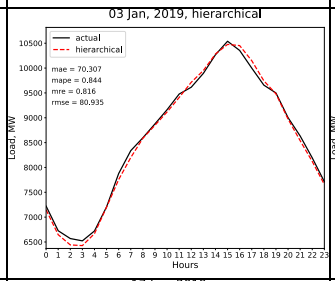
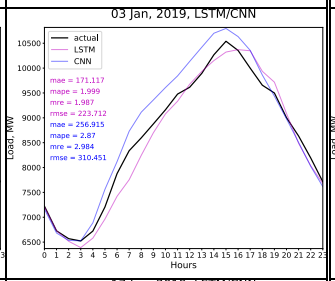
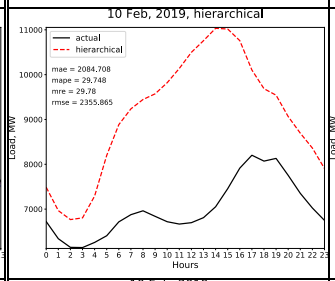
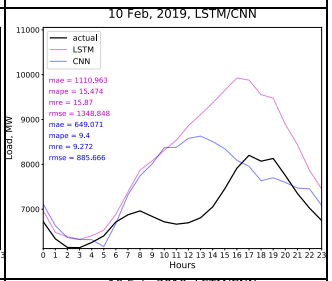
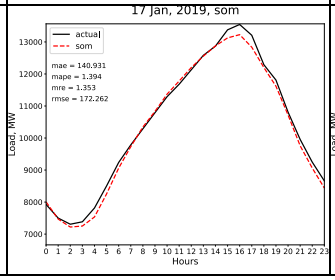
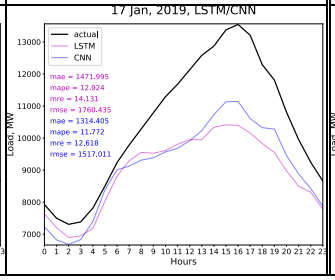
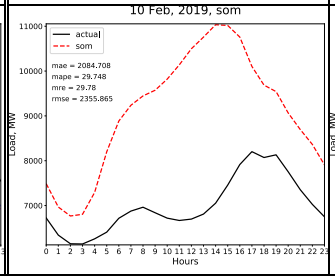
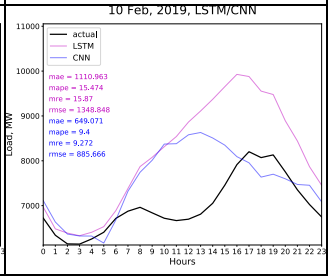


Table 10. Best and worst electricity load forecasts of each model for NYC

	Best prediction	LSTM/CNN	Worst prediction	LSTM/CNN
ARIMA				
k-means				
k-medoids				
Fuzzy C-means				
Hierarchical				
SOM				

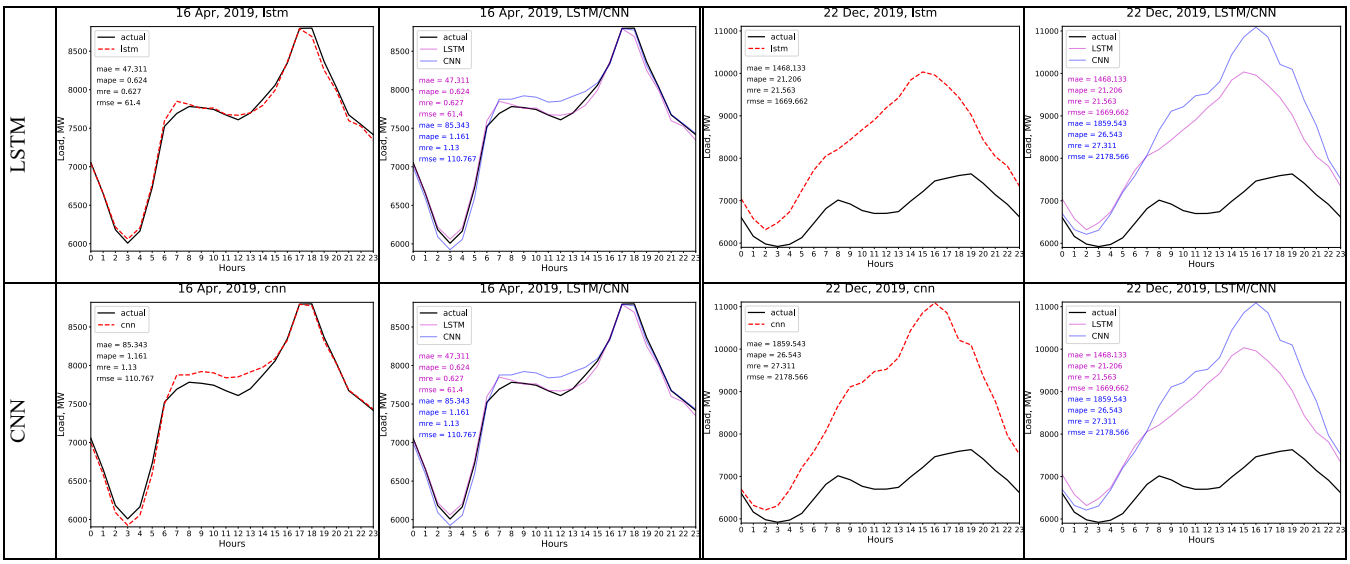


Table 11. Best and worst electricity load forecasts of each model for NSW

The end of 2019 and the most of the year of 2020 is overshadowed by the outbreak of global COVID-19 pandemic of infectious disease, caused by coronavirus [81]. Initially identified in Wuhan, China at the end of December 2019, it quickly spread across the globe, taking over 300,000 lives and causing a severe disruption in global economics, leading to its future recession. In the US the first cases were recorded in January [82] with the first death case registered on February [83]. Till the end of the March the COVID-19 spread all over the states, and New York became the epicenter of the pandemic in US, with the metropolitan area worst affected. The declared national emergency on March 13 aimed to curb the community transmission of COVID -19. The partial closure of businesses in NYC area since March 20-th affected the citywide electrical energy consumption during the quarantine, causing the downward trend in electricity load and higher price volatility. We investigate to what extent the forecasting algorithms are capable to cope with unexpected patterns in electrical power load and the unpredictable situation in the electricity market (see Table 12).

In New South Wales the consequences of pandemics were not that severe as in the US due to the timely intervention and strict preventive measures. Australia started initial screenings of passengers on January 23-rd and finally closed its borders on March 20-th. The first case of COVID-19 has been recorded on January 25-th, and the government announced the Australian Health Sector Emergency Response Plan for Novel Coronavirus at the end of the February. The number of cases was increasing starting from March 10-th, which prompted the Australian government to introduce a human biosecurity emergency state on March 18-th, imposing the quarantine and social distancing. The most cases were recorded in New South Wales, totaling to 3074 with 45 deaths and 2611 recovering as of May 16th. The restrictions were lifted on May 15-th. In overall, the Australian government was able to localize the pandemic and eliminate the spread due to organized and prompt measures. There is a drop in power load after March 3-rd, following the relatively flat trend (see Table 13).

In overall, the deep-learning based technics were capable to maintain high forecasting accuracy and capture the trend variations. ARIMA demonstrated the best performance and the clustering algorithms were reluctant to learn the new consumption patterns, failing to capture sudden spikes at the beginning of pandemics in NSW (see Table 13). Thus the statistical and deep learning methods can provide reliable prediction even in the changing conditions.

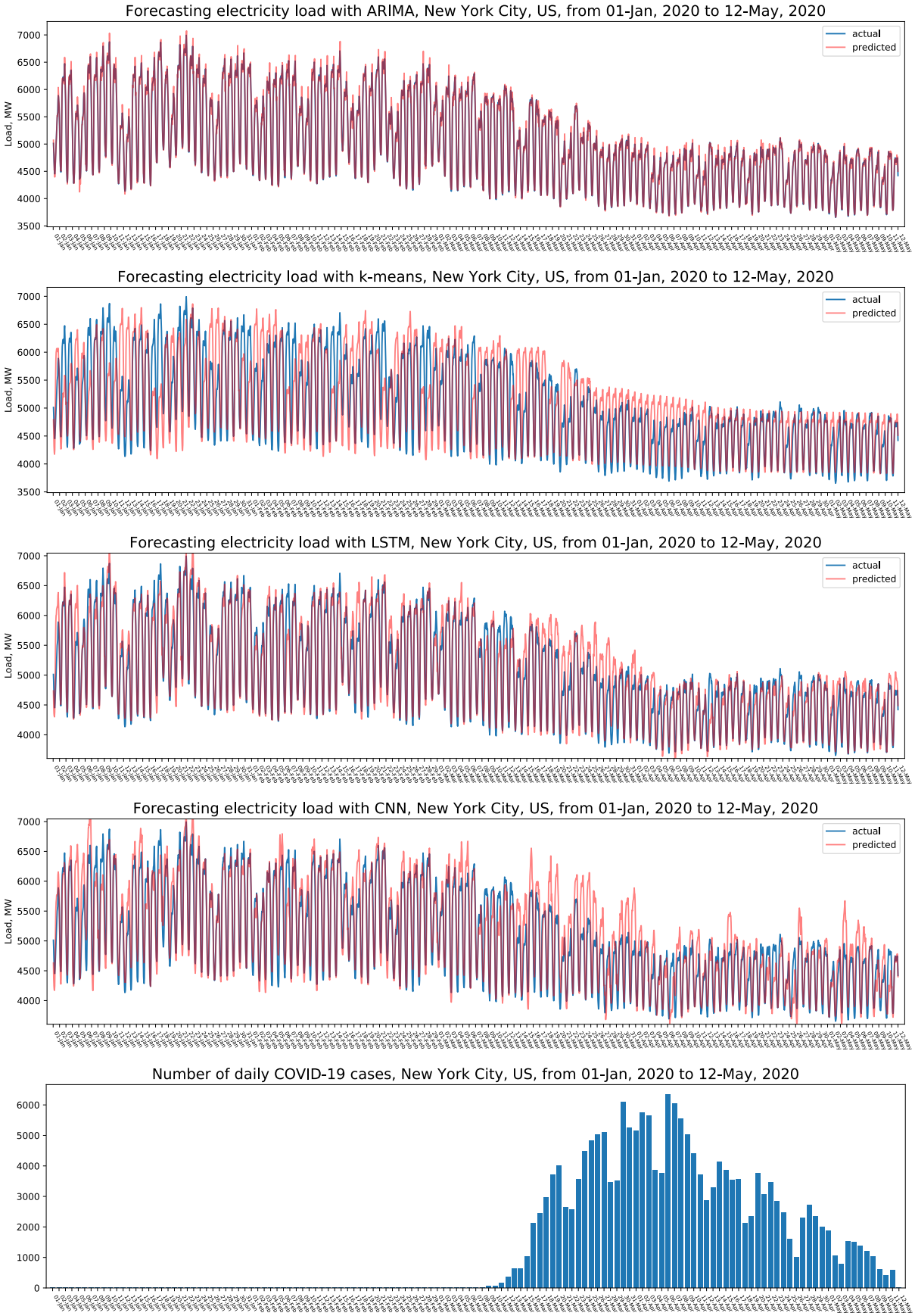


Table 12. Predicting electrical load in NYC during the COVID-19 pandemics.

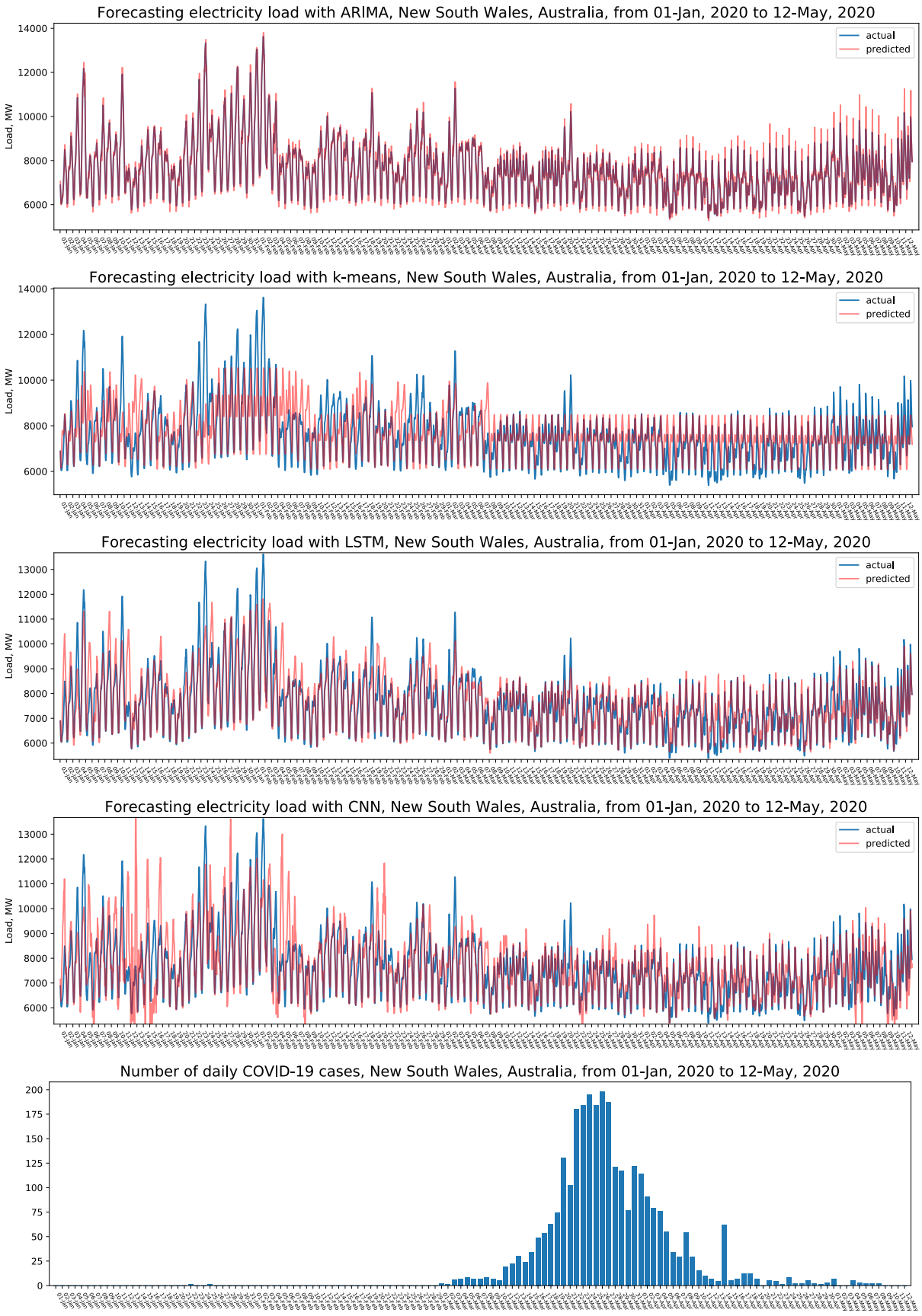
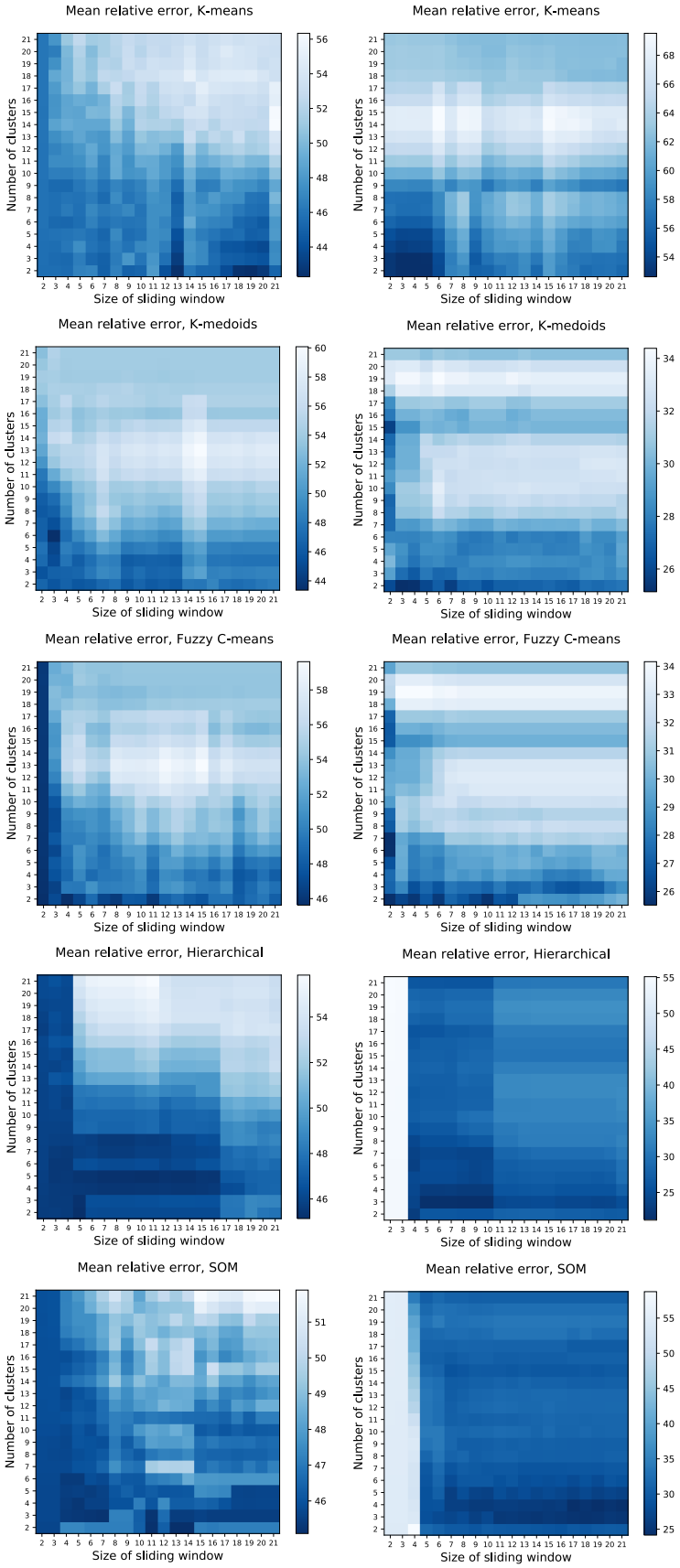


Table 13. Predicting electrical load in NSW during the COVID-19 pandemics.



a. b.

Fig. 21. Grid search for best parameters a.) NYC, b.) NSW

First we applied clustering algorithms for forecasting the price data from the New York Independent System Operator (NYISO) for New York City [2] (NYC) and Australian Energy Market Operator (AEMO) [4] for New South Wales (NSW). Fig. 21 presents the results for grid search for optimal parameters: number of clusters and window size. Since the price is volatile and hard to predict, the MRE values are relatively higher than corresponding MRE values for load data. In general, there are also some traces of weekly/biweekly trends (the darker horizontal strips on the heatmaps at $k = 7, 14, 21$), however, they are not that pronounced as for the grid search results for load. It could be attributed to the less periodical structure of the price as well as its quite high volatility and dependency on the external factors. Due to these reasons the trends, captured by clustering algorithms, are diluted with the external noise. The external factors severely affect the forecasting accuracy. The found

optimal parameters, used in clustering algorithms for electricity price forecasting are summarized in Table 14.

		k-means	k-medoids	Fuzzy C-means	Hierarchical	SOM
NYC (NYISO)	*k*	2	6	2	5	3
	w	13	3	4	11	11
NSW (AEMO)	*k*	3	2	7	3	4
	w	5	4	2	10	17

Table 14. Optimal values of *k* and *w* for price forecasting with clustering algorithms.

Using these parameters we conducted a 12 fold cross validation of 5 clustering algorithms, two deep-learning network based approaches (LSTM and CNN) and ARIMA model on the price data for NYC and New South Wales for the year of 2019. The results are summarized in Table 15 and Table 16.

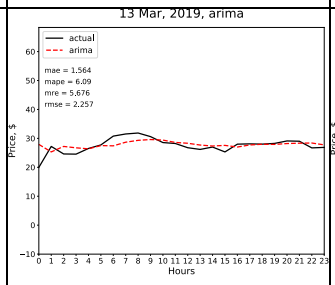
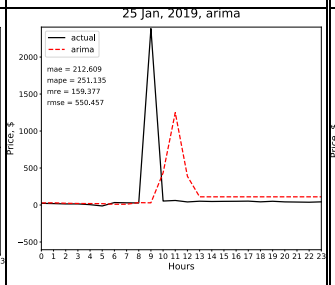
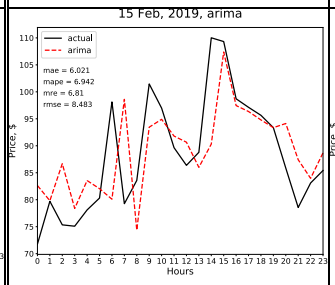
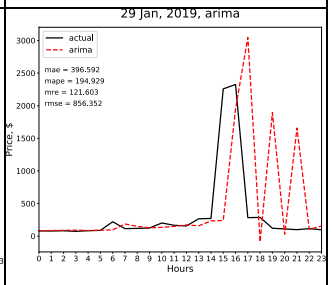
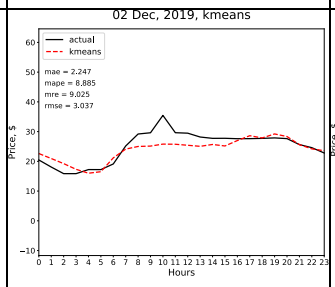
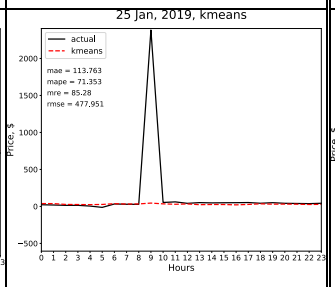
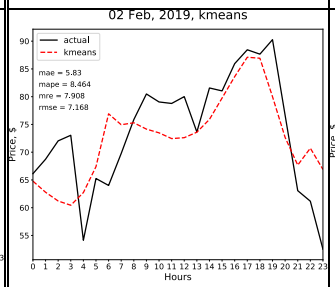
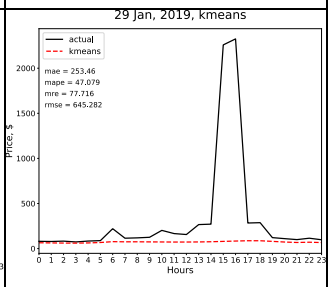
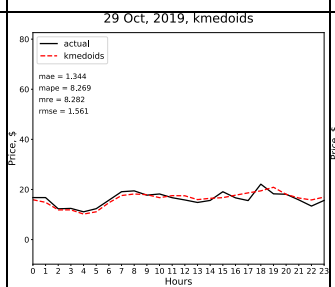
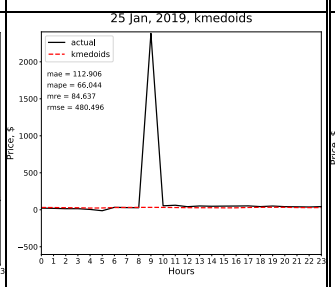
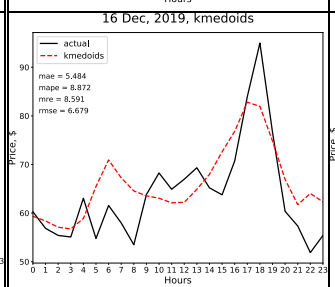
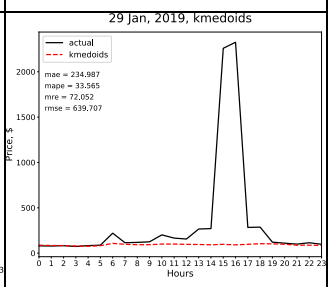
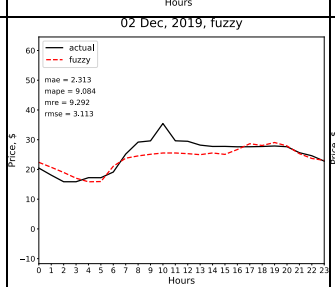
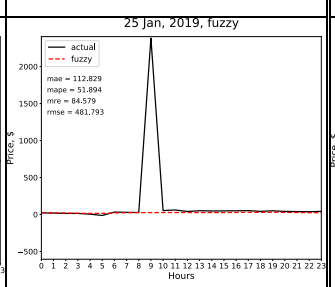
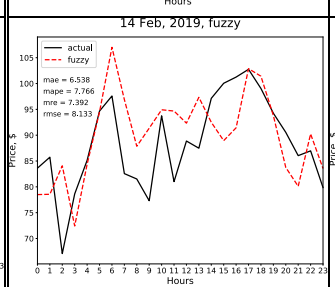
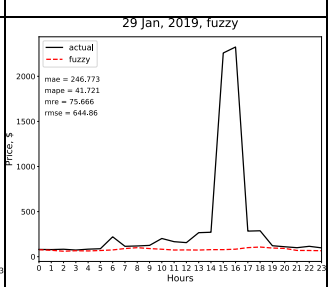
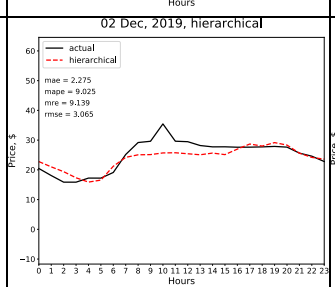
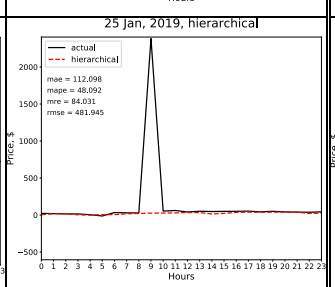
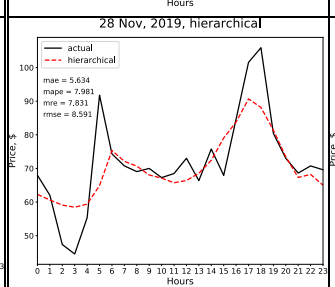
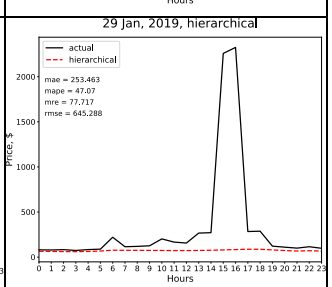
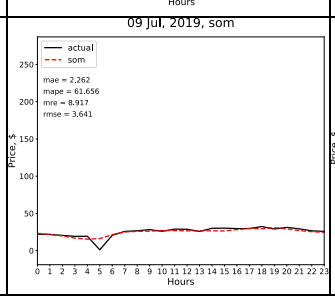
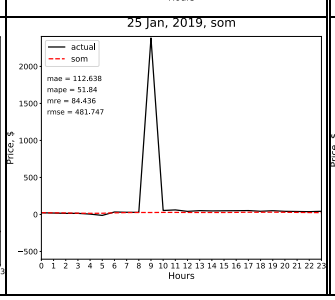
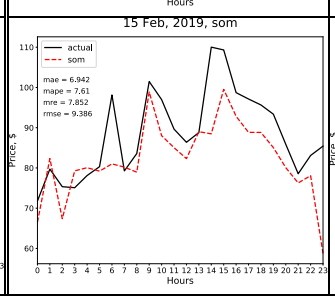
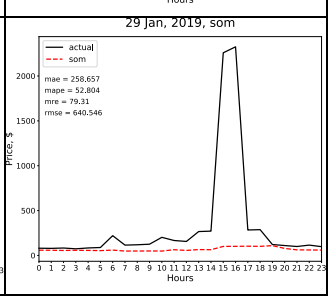
It's easy to notice that all algorithms performed poorly due to high volatility of price data and the presence of spikes. Since these outbursts can even reach negative values, MAPE can get absurdly high values because of the division on actual value, which can get close to 0. Since the MRE contains an average value of the interval of interest in denominator, it makes this error metrics the most reliable metrics for evaluating the classifier on highly volatile price data with outliers. Other two error metrics can also give a general idea of algorithm's forecasting accuracy. Looking at the MRE for all algorithms it appears that the price for November for NYC and March for NSW is most predicable. In overall, the ARIMA model performed relatively well in comparison with the rest of the methods. However, ARIMA and the rest of the algorithms, was not capable to handle the random outbursts, as it can be seen from Table 17. Interestingly, the clustering algorithms outperformed the deep learning networks on both datasets. Most probably it is due to the elimination of the sequences of similar patterns, containing outliers greater than 1.5 IQR, from predicting the future values in improved PSF-style algorithm. Also, apparently, the shallow architecture of networks fails to capture the patterns causing high volatility in the price data. Modifying the architecture of deep learning networks and introducing methods for spike detection and prediction is a good topic for future research.

		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Average
ARIMA	MAE	26.78	10.35	16.6	8.27	8.94	10.17	14.08	7.37	3.96	5.2	5.08	10.96	10.65
	MAPE	210.41	75.32	48.88	216.13	847.68	1840.9	197.57	31.64	33.35	41.43	22.72	43.73	300.82
	MRE	47.71	33.25	43.07	31.79	76.61	36.92	78.89	29.33	18.06	24.41	18.58	34.22	39.4
	RMSE	46.96	14.45	32.17	14.65	15.33	19.81	28.85	11.43	6.38	8.78	7.64	20.85	18.94
k-Means	MAE	26.5	9.26	13.27	8.27	8.78	10.35	11.88	7.67	7.01	8.97	6.7	11.33	10.83
	MAPE	119.17	42.85	28.43	182.49	992.66	1876.3	143.69	37.76	56.67	75.76	25.74	37.15	301.56
	MRE	46.55	28.48	32.52	32.8	68.12	41.3	59.52	35.89	35.79	49.58	22.97	33.74	40.6
	RMSE	48.95	13.92	30.45	14.76	15.41	20.13	27.77	11.85	9.2	12.34	9.17	21.87	19.65
k-Medoids	MAE	23.47	9.92	12.06	9.54	8.82	11.19	12.94	8.28	6.9	6.39	9.13	13.09	10.98
	MAP	93.51	56.02	28.27	173.59	739.18	1386.2	45.28	38.69	46.36	46.87	34.18	47.7	227.99
	MRE	41.1	32.44	30.26	38.27	66.8	43.3	74.27	37.31	32.78	31.75	31.98	42.69	41.91
	RMS	46.55	14.65	29.36	15.63	15.3	20.73	28.58	12.45	9.41	10.46	11.74	23.22	19.84
Fuzzy C-Means	MAE	26.35	9.02	12.94	7.97	8.64	10.21	12.13	7.43	6.69	8.63	6.72	11.25	10.66
	MAPE	112.57	42.28	27.4	177.97	983.65	1830.4	148.08	36.39	54	72.84	25.58	36.83	295.67
	MRE	46.33	27.08	31.44	31.64	67.29	40.48	61.5	34.82	33.96	47.53	22.94	33.73	39.89
	RMSE	49.05	13.62	30.13	14.52	15.28	20	28.04	11.61	8.91	12.04	9.21	21.77	19.52
Hierarchical	MAE	26.73	8.99	13.15	8.33	8.75	10.36	11.94	7.71	7.01	8.94	6.7	11.43	10.84
	MAPE	118.4	46.4	27.95	182.36	991.49	1875.3	145.98	38.1	56.61	75.49	25.74	37.53	301.78
	MRE	46.97	27.13	32.12	32.99	68.01	41.37	60.18	36.25	35.76	49.42	22.96	34.2	40.61
	RMSE	49.4	13.7	30.36	14.89	15.39	20.12	27.9	11.93	9.2	12.32	9.17	22.02	19.7
SOM	MAE	26.84	8.93	13.56	8.34	8.81	10.39	11.82	7.5	7.09	9.11	6.75	11.37	10.88
	MAPE	107.98	42.69	30.74	183.27	999.68	1883.6	141.2	37.22	56.69	76.58	26.16	37.31	301.93
	MRE	46.96	26.89	33.66	33.1	68.34	41.63	59.4	35.5	36.21	50.44	23.27	33.99	40.78
	RMSE	49.54	13.58	30.58	14.93	15.41	20.12	27.84	11.66	9.26	12.47	9.18	21.87	19.7
LSTM	MAE	26.48	94.35	16.46	18.24	19.58	25.75	13.08	11.03	12.65	16.97	16.83	23.7	24.59
	MAPE	157.94	1935.4	1248.0	701.02	1368.8	64.59	1495.8	49.18	1739.3	319.86	440.96	243.18	813.7
	MRE	38.75	75.56	48.97	93.47	48.98	57.27	89.44	50.24	34.79	65.81	105.65	68.01	64.74
	RMSE	40.49	136.49	29.09	22.73	30.68	46.01	20.05	15.2	16.86	24.16	27.81	42.56	37.68
CNN	MAE	104.56	12.43	16.88	13.32	14.43	15.49	16.2	10.61	9.3	9.58	8.28	12.51	20.3
	MAPE	467.09	58.56	45.79	207.95	1370.6	2133.4	88.88	53.84	68.22	73.66	32.85	44.31	387.1
	MRE	230.77	42.28	45.47	55.35	105.75	65.95	82.7	51.39	47.83	52.38	30.36	40.48	70.89
	RMSE	156.13	17.14	35.8	19.78	20.8	23.79	34.81	14.97	12.72	14.16	11.03	22.82	32

Table 15. Electricity price forecast errors for each algorithm after 12-fold cross validation averaged for every fold (month) (NYC, 2019).

		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Average
ARIMA	MAE	32.48	16.66	11.91	11.68	13.35	20.41	13.17	22.69	29.35	25.13	12.86	6.93	18.05
	MAPE	23.39	20.15	14.3	14.97	15.73	20.95	20.09	536.98	35.65	26.4	19.31	13.43	63.45
	MRE	20.82	18.63	13.62	14.82	16.54	21.72	18.21	26.53	30.57	24.19	18.32	13.38	19.78
	RMSE	55.44	20.39	15.65	17.25	20.89	31.6	20.95	34.11	43.06	35.78	18.95	9.77	26.99
k-Means	MAE	47.62	21.28	19.45	17.57	15.27	28.04	16.1	29.57	37.07	38.7	16.42	18.63	25.48
	MAPE	28.69	22.41	21.32	23.06	17.79	26.22	29.26	1006.9	39.15	36.91	27.47	42.01	110.1
	MRE	33.92	22.88	21.68	22.38	18.94	28.58	22.52	34.84	36.52	36.1	23.72	38	28.34
	RMSE	73.14	26.81	23.79	22.67	22.6	41.84	22.83	44.64	54.96	52.7	22.17	21	35.76
k-Medoids	MAE	43.55	19.85	17.93	17.74	17.7	27.37	16.79	30.21	37.01	36.14	16.32	17.22	24.82
	MAP	27.23	23.67	21.86	24.49	22.15	29.36	30.5	1303.6	48.85	37.62	27.98	39.1	136.37
	MRE	32.06	22.68	20.37	23.11	22.33	29.84	23.46	36.17	38.02	34.4	23.78	35.2	28.45
	RMS	70.34	25.67	22.23	23.06	25.3	40.08	24.18	45.42	53.59	50.24	22.25	19.57	35.16
Fuzzy C-Means	MAE	48.78	20.52	17.56	19.14	17.87	22.16	17.59	29.1	39.18	32.63	16.42	14.66	24.63
	MAPE	33.65	23.11	22.52	26.51	22.96	25.68	32.2	1134.7	49.59	37.08	27.7	33.24	122.41
	MRE	36.98	22.75	20.19	24.4	22.79	25.39	24.72	35.36	41.06	32.04	23.69	30.04	28.28
	RMSE	87.42	28.4	21.73	24.88	25.7	33.3	25.08	43.27	56.68	45.54	22.53	17.3	35.99
Hierarchical	MAE	47.63	21.32	19.46	17.61	15.3	28.07	16.21	29.54	37.18	37.24	16.04	19.59	25.43
	MAPE	28.69	22.44	21.33	23.11	17.83	26.28	29.31	1007.5	40.61	33.39	27.09	44.12	110.14
	MRE	33.92	22.92	21.7	22.44	18.98	28.65	22.66	34.41	36.65	34.22	23.33	39.91	28.32
	RMSE	73.15	26.85	23.81	22.71	22.63	41.88	22.91	45.01	55.51	51.99	21.71	21.91	35.84
SOM	MAE	47.87	19.64	19.52	17.73	15.3	28.73	16.88	28.95	41.83	40.31	17.53	19.76	26.17
	MAPE	29.41	21.56	21.38	23.37	17.78	27.47	30.55	728.84	40.45	37.66	30.17	44.64	87.77
	MRE	34.34	21.32	21.76	22.64	18.97	29.71	23.6	32.93	41.19	37.54	25.41	40.25	29.14
	RMSE	73.15	25.57	23.9	22.92	22.6	42.13	23.68	43.41	57.85	54.5	23.32	21.99	36.25
LSTM	MAE	129.01	81.25	22.56	21.73	19.19	31.65	20.03	29.86	45.52	42.92	27.5	15.96	40.6
	MAPE	102.63	95.85	26.14	28.48	23.63	33.56	35.47	38.46	56.02	46.21	44.21	32.82	46.95
	MRE	97.51	85.77	25.58	27.66	24.22	34.1	28.55	36.58	48.39	42.14	39.23	31.33	43.42
	RMSE	168.86	108.35	27.88	29.24	26.68	44.03	26.95	42.74	64.52	55.35	36.33	20.26	54.27
CNN	MAE	105.64	72.69	19.14	23.89	18.98	40.86	23.35	53.26	82.08	84.94	31.52	19.64	48
	MAPE	80.51	87.96	22.93	31.03	23.36	45.63	42.17	1978.5	114.69	93.02	50.27	38.48	217.38
	MRE	83.96	87.61	21.8	29.89	23.93	45.91	33.3	65.84	89.76	82.32	44.17	38.43	53.91
	RMSE	149.16	99.63	24.11	31.19	26.9	58.04	33.44	72.09	110.63	110.44	42.92	25.9	65.37

Table 16. Electricity price forecast errors for each algorithm after 12-fold cross validation averaged for every fold (month) (NSW, 2019).

	Best price forecast, NYC	Worst price forecast, NYC	Best price forecast, NSW	Worst price forecast, NSW
ARIMA				
k-means				
k-medoids				
Fuzzy C-means				
Hierarchical				
SOM				

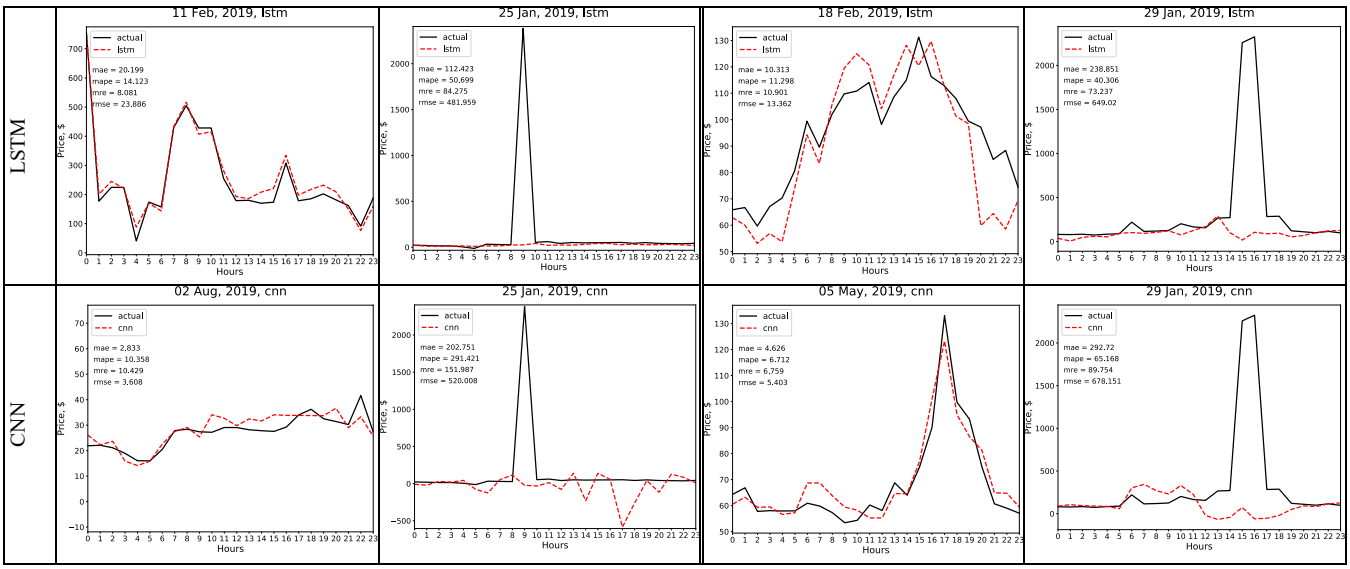


Table 17. Best and worst electricity price forecasts of each model for NYC and NSW

Electricity price prediction is a difficult problem due to volatility, non-stationarity and non-homogenous structure of price time series data. In contrast to electrical load, electricity price data has a less pronounced trend (see Fig. 22). Also, price data is heavily affected by external factors. For example, price spike occurs if there is a transmission congestion detected in the system. After the spike, price usually reverts to a more reasonable level [12]. Regardless of volatile nature, electricity price is not a random variable. As it can be noticed from Fig. 22., there is a periodic

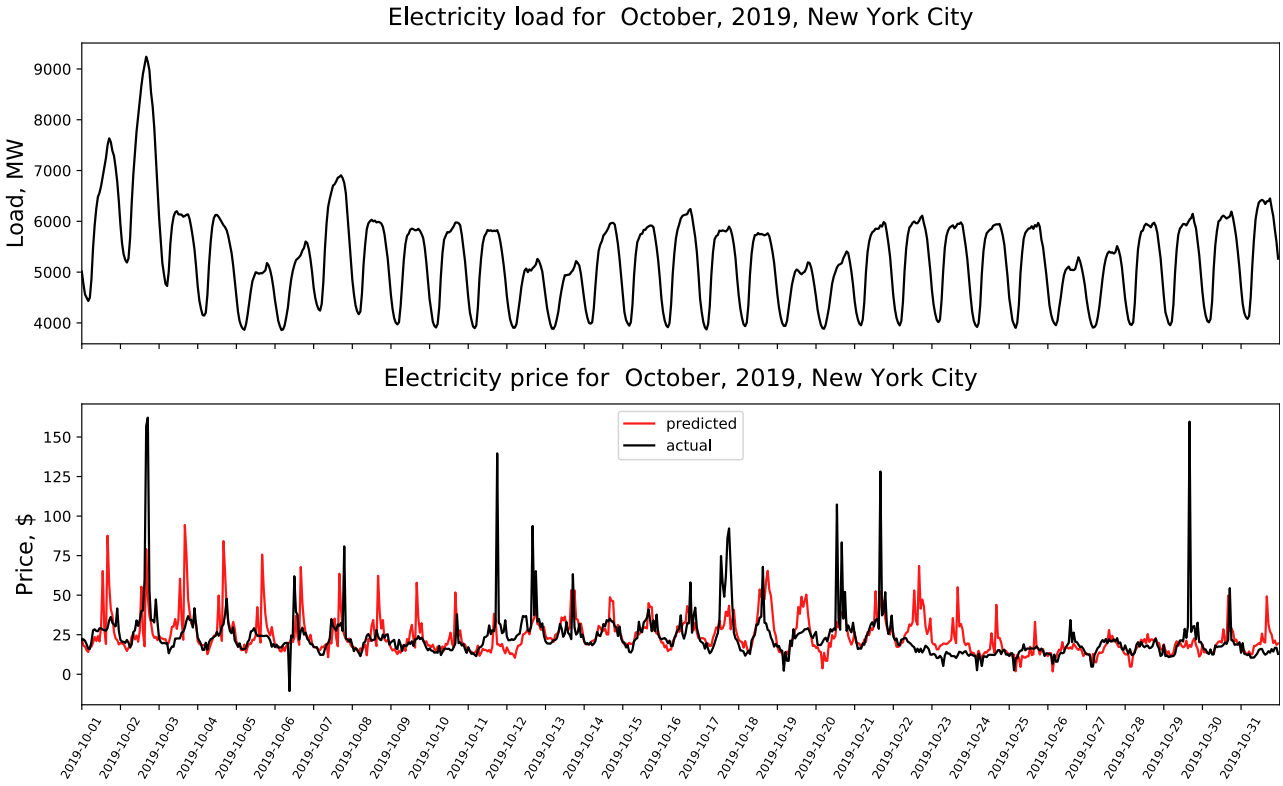


Fig. 22. Load and predicted/actual electricity price values for NYC in October, 2019.

structure and variations in daily trend, correlated with electrical load. Therefore, it is possible to learn hidden patterns and rules that determine price fluctuations from load and historical price data. Knowing these hidden dependences will result in more accurate electricity price forecast.

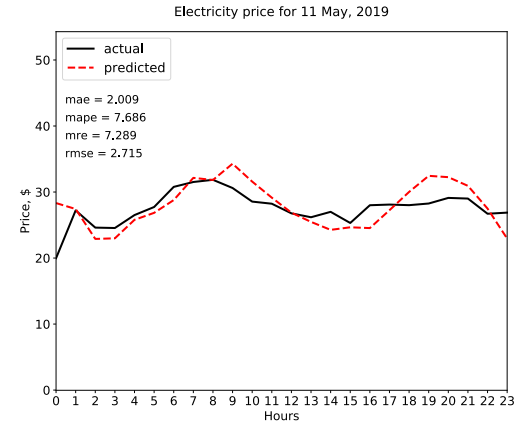


Fig. 24. Best price forecast for NYC, 2019.

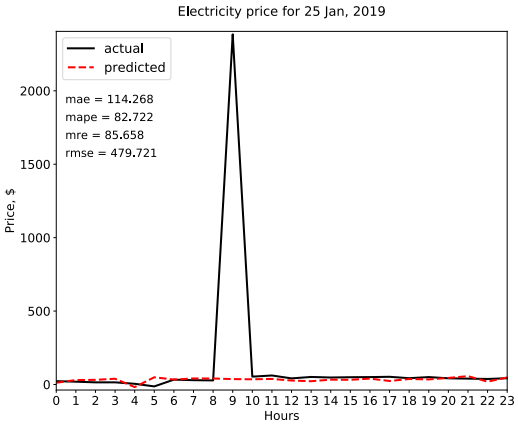


Fig. 23. Worst price forecast for NYC, 2019.

The non-linear correlation between electrical load and price is influenced by different factors. Load is mostly affected by the consumption patterns, weather conditions, social events (e.g. holidays), seasonality and the inability to store the electric power. In the case of electricity price, all the load's factors come into play along with other factors, imposed by market, such as regulations of authorities, competitors' pricing schemes, and other macro- and micro-economic factors. All these factors contribute into high volatility of price data and need to be considered for more accurate mapping between load and prices.

We applied the proposed Convolutional Neural Network to establish the mapping between electricity load and price. As pricing data we used Location Based Marginal Pricing (LBMP) — cost to provide the next MW of Load at a Specific Location in the grid provided by New York Independent System Operator [2] for the same location as load (N.Y.C.).

The validation of CNN-based algorithm for electricity price prediction has been conducted on the data for 2019. To train the model the load and price data for the previous year was used. As a predicted load values we used output from LSTM network, since it demonstrated relatively high average performance in comparison with the rest of the non-statistical models considered in this work. On the validation stage the inputs for each day were tested and after obtaining prediction the corresponding inputs were generated for predicted day's actual value and the whole network has been retrained. This procedure is known as online learning and it allows to generate predictions for the longer forecasting horizons (in this case we need to use the predicted price for the given day instead of actual price, since we assume the actual value is unknown). In overall, the proposed algorithm was able to achieve the average MAE of 28.3 and average MRE 32.48 on NYC and average MAE 22.35 and MRE 25.52 for NSW dataset for 2019. Fig. 24 and Fig. 23 show the best and worst predictions for the year of 2019 for NYC and Fig. 27 with Fig. 26 demonstrate the best and worst predictions for NSW.

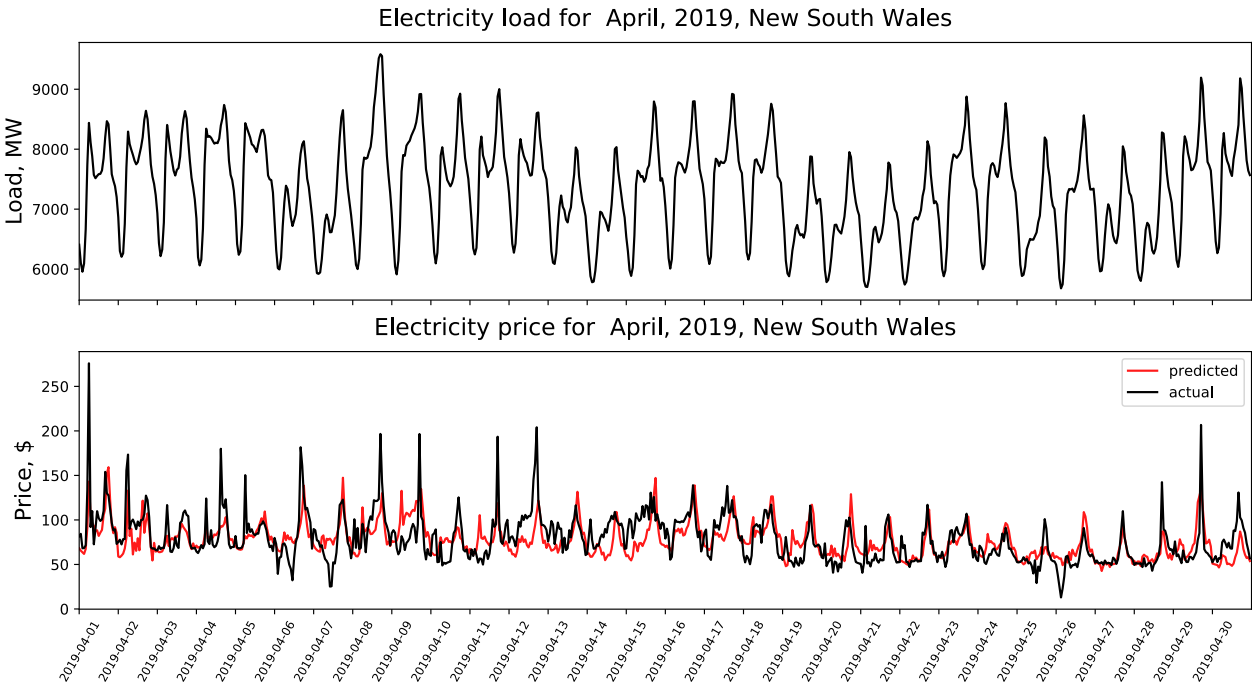


Fig. 25. Load and predicted/actual electricity price values for NSW in April, 2019.

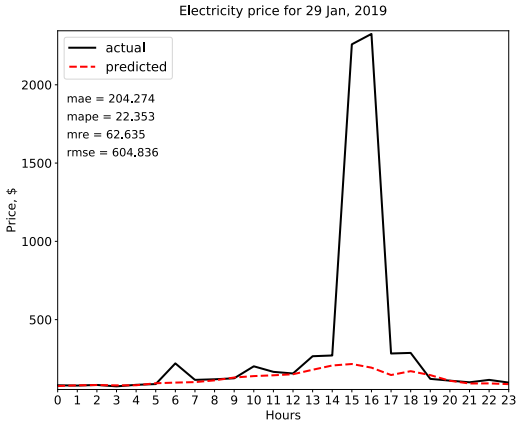


Fig. 26. Worst price forecast for NSW, 2019.

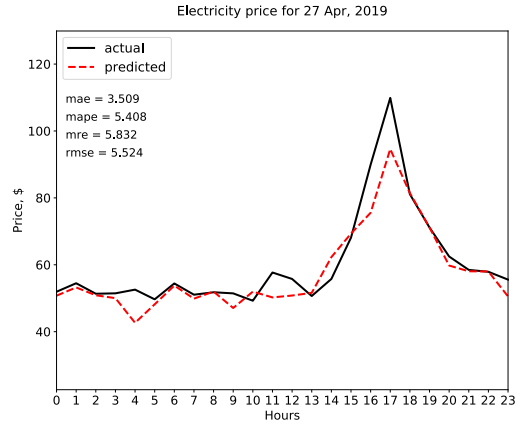


Fig. 27. Best price forecast for NSW, 2019.

As it is easy to notice, the model fails to perform well when it faces the spike in electricity price. In addition, the forecasting accuracy of the model significantly drops down after the spike. Therefore, the forecast for the next few days after the outburst cannot be considered 100% reliable. For instance, on Fig. 22 after the spike occurred on October 21, the model predicted small peaks for October 22 and 23 while the actual trend was almost flat. However, the peak for October 11 did not affect the predictions for October 12 and 13. Thus, the presence of a spike may indicate a need for revising the forecast for the next few days after the outburst.

Usually, spikes are the product of temporal discordance between electricity supply and demand. Even the discrepancies on the scale of seconds can induce significant spikes. The other factors resulting in spikes include power generation outages,

congestion, behavior of market entities, manipulations on the market to name a few.

Significant volatility of price data creates challenges for accurate forecasting of the future prices. As a result, the price forecasting accuracy is usually low that that of the electrical load forecasting. However, the requirements for price forecasting are not as stringent as for the electrical load and some degree of inaccuracy in prediction is acceptable.

CONCLUSION AND RECOMMENDATIONS

Electricity load and price forecasting is very important for electrical power industry and market. In fact, these two problems are interrelated: the knowledge of future load values is vital for power system scheduling, resulting in lower cost of production, thus resulting in lower electricity prices. Accurate prediction of electricity price is important for all market participants: energy producers, traders, providers and consumers. Robust load forecasting mechanism is essential in smart grid settings, where the grid analyzes the behavioral patterns of power suppliers and end-users and tries to optimize electricity generation and distribution.

In this work we study automated methods for electricity load and price forecasting. Initially we consider the statistical methods (ARIMA) and conventional Machine Learning technics (PSF algorithm and its extensions with different clustering approaches). Analyzing the distribution of time series daily values across the clusters we demonstrated that the clustering algorithms clearly reflect the seasonal changes in power consumption and electricity price. We introduced an improvement to classical PSF by excluding days, containing outliers, above and below 1.5IQR of historical data, from prediction process. This improvement resulted in lower values of all four error metrics.

The novel deep learning approaches offer a very promising technology for electricity price prediction. We explored two different deep learning architectures – Convolutional Neural Networks and Long-Short Term Memory Networks. While CNNs are efficient of revealing hidden dependencies between inputs and outputs and automated features learning, the LSTM networks are more suited for extracting temporal patterns. Testing both networks on NYISO load data demonstrated more stable performance of LSTM, which was even capable of handling non-trivial patterns such as peaks in electricity load. While the ARIMA model achieved the highest accuracy, it sometimes failed to provide a reliable forecast, especially facing the unpredictable outbursts in the price data. The LSTM – based approach also demonstrated acceptable average performance in comparison with other conventional methods. We also evaluated the performance of conventional methods and deep learning based approaches in the situation, when electricity consumption patterns change unexpectedly due to pandemics. Both ARIMA and deep-learning approaches demonstrated high accuracy and robustness in capturing the novel trend, while the clustering algorithms failed to adapt fast to the changes. Generally speaking, the main

recommendation would be to use the combination of statistical and deep-learning methods for making robust predictions.

The next contribution of this study is the development of CNN –based model for direct mapping between electricity load and price values. Since the price is very hard to predict due to its volatility, most of the conventional models are unable to capture all hidden dependencies resulting in unpredictable behavior of price values. The proposed CNN–based deep learning approach is capable of learning some of these dependences, which results in more-or-less adequate performance.

As the future work it would be interesting to attempt learning the features inducing the peaks in price data. It can be accomplished by deploying wavelet transform to extract the deterministic and noise component and analyze them individually. It would be also interesting to explore the performance of more advanced deep learning techniques such as Generative Adversarial Networks (GANs) and Deep Reinforcement Learning for time series prediction, or hybrid models, incorporating CNN-s for feature learning and LSTM-s for prediction.

BIBLIOGRAPHY

[1] S. Stoft, Power system economics: designing markets for electricity, Piscataway, NJ: IEEE Press, 2002.

[2] "ENERGY MARKET & OPERATIONAL DATA," New York Independent System Operator, [Online]. Available: https://www.nyiso.com/energy-market-operational-data. [Accessed 20 04 2020].

[3] "Aggregated price and demand data," Australian Energy Market Operator, [Online]. Available: https://www.aemo.com.au/energy-systems/electricity/national-electricity-market-nem/data-nem/aggregated-data. [Accessed 20 04 2020].

[4] "Pronosticul lunar: Pronosticul transportului de energie şi putere electrică de către sistemul energetic al Republicii Moldova, împărţit pe Furnizori," Moldelectrica Operatorul sistemului de transport, [Online]. Available: http://www.moldelectrica.md/ro/electricity/monthly_forecast. [Accessed 20 04 2020].

[5] "COVID-19: Data," NYC government, [Online]. Available: https://www1.nyc.gov/site/doh/covid/covid-19-data.page. [Accessed 15 05 2020].

[6] "COVID-19 cases by notification date," The government of New South Wales , [Online]. Available: https://data.nsw.gov.au/data/dataset/covid-19-cases-by-location/resource/21304414-1ff1-4243-a5d2-f52778048b29. [Accessed 15 05 2020].

[7] F. Chollet, "Keras," 2015. [Online]. Available: https://keras.io. [Accessed 20 04 2020].

[8] F. Pedregosa, V. Gael, G. Alexandre, M. Vincent, T. Bertrand, G. Olivier, M. Blondel, P. Prettenhofer, W. Ron, D. Vincent, V. Jake, P. Alexandre, C. David and B. Matthieu, "Scikit-learn: Machine learning in Python.," *Journal of Machine Learning Research,* p. 2825–2830, 2011.

[9] T. Erkkilä, A. Lehmussola, K. Kiełczewski and Z. Dufour, "K-medoids clustering," 2019. [Online]. Available: https://github.com/scikit-learn-contrib/scikit-learn-extra/blob/master/sklearn_extra/cluster/_k_medoids.py. [Accessed 20 04 2020].

[10] "SciKit-Fuzzy," [Online]. Available: https://pythonhosted.org/scikit-fuzzy/. [Accessed 12 05 2020].

[11] V. Giuseppe, "MiniSom," [Online]. Available: https://github.com/JustGlowing/minisom. [Accessed 20 05 2020].

[12] D. Singhal and S. Swarup, "Electricity price forecasting using artificial neural networks," *International Journal of Electrical Power & Energy Systems ,* vol. 33, pp. 550-555, 2011.

[13] H. Hippert, C. Pedreira and R. C. Souza, Neural Networks for Short-Term Load Forecasting: A review and Evaluation, vol. 16, IEEE Transactions on Power Systems, 2001, pp. 44-55.

[14] D. Belik, D. Nelson and D. Olive, Use of the Karhunen- Loeve expansion to analyze hourly load requirements for a power utility, vol. A78, IEEE Power Engineering Society Winter Meeting, 1987, pp. 225-230.

[15] M. Meng and W. Shang, Research on Annual Electric Power Consumption Forecasting Based on Partial Least-Squares Regression, vol. 2, Proceedings of nternational Seminar on Business and Information Management, 2008, pp. 125-127.

[16] L.-Y. Wei, T. Chih-Hung, Y.-C. Chung, L. Kuo-Hsiung, C. Hao-En and L. Jyh-Shyan, "A Study of the Hybrid Recurrent Neural Network Model for Electricity Loads Forecasting,"

International Journal of Academic Research in Accounting, Finance and Management Sciences, vol. 7, no. 2, pp. 21-29, 2017.

[17] F. Martınez-Alvarez, A. Troncoso, J. Riquelme and J. S. Aguilar-Ruiz, LBF: A labeled-based forecasting algorithm and its application to electricity price time series,, Proceedings of the 8th IEEE International Conference on Data Mining (ICDM'08), 2008.

[18] G. Box, G. Jenkins and G. Reinsel, Time Series Analysis: Forecasting and Control, vol. 734, Hoboken, NJ: John Wiley & Sons, 2011.

[19] F. Martınez-Alvarez, A. Troncoso, J. C. Riquelme and J. S. A. Ruiz, Energy time series forecasting based on pattern sequence similarity, IEEE Transactions on Knowledge and Data Engineering, 2011.

[20] P. Rousseeuw and L. Kaufman, Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley & Sons, 2009.

[21] J. Dunn, Well-Separated Clusters and Optimal Fuzzy Partitions, vol. 4, Journal of Cybernetics and Systems, 1974, pp. 95-104.

[22] D. Davies and D. Bouldin, A Cluster Separation Measure, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1979, pp. 224 - 227.

[23] C. Jin, G. Pok, H. W. Park and K. Ryu, Improved Pattern Sequence-Based Forecasting Method for Electricity Load, IEEJ Transactions on Electrical and Electronic Engineering, 2014.

[24] W. Shen, V. Babushkin, Z. Aung and W. Woon, An ensemble model for day-ahead electricity demand time series forecasting, In Proceedings of the 4th ACM Conference on Future Energy Systems (e-Energy), 2013.

[25] M. Mostafa, Q. Charlie, C. Peter, G. Rajit and P. H. R., "Modified pattern sequence-based forecasting for electric vehicle charging stations.," in *IEEE International Conference on Smart Grid Communications (SmartGridComm)* , 2014.

[26] Y. Fujimoto and Y. Hayashi, "Pattern sequence-based energy demand forecast using photovoltaic energy records," in *International Conference on Renewable Energy Research and Applications (ICRERA)*, 2012.

[27] F. Martınez-Alvarez, A. Troncoso, J. Riquelme and J. S. Aguilar-Ruiz, "Discovery of motifs to forecast outlier occurrence in time series.," *Pattern Recognition Letters,* vol. 32, p. 1652–1665, 2011.

[28] L. Ekonomou, C. Christodoulou and V. Mladenov, "A short-term load forecasting method using artificial neural networks and wavelet analysis," *International Journal of Power Systems,* vol. 1, pp. 64-68, 2016.

[29] A. Reis and A. da Silva, "Feature extraction via multiresolution analysis for short-term load forecasting," in *IEEE Transactions on Power Systems*, 2005.

[30] I. Koprinska, M. Rana, A. Troncoso and F. Martínez-Álvarez, "Combining Pattern Sequence Similarity with neural networks for forecasting electricity demand time series," in *Proceedings of the International Joint Conference on Neural Networks*, 2013.

[31] G. E. Hinton, O. Simon and T. Yee-Whye, "A fast learning algorithm for deep belief nets," *Neural computation,* vol. 18, pp. 1527-54, 2006.

[32] S. Bouktif, A. Fiaz, A. Ouni and M. Serhani, "Optimal Deep Learning LSTM Model for Electric Load Forecasting using Feature Selection and Genetic Algorithm: Comparison with Machine Learning Approaches," *Energies,* vol. 11, p. 1636, 2018.

[33] X. Qiu, Z. Le, Y. Ren, P. N. Suganthan and A. Gehan, "Ensemble deep learning for regression and timeseries forecasting," in *IEEE Symposium on Computational Intelligence in Ensemble Learning (CIEL)*, 2014.

[34] E. I. O. a. S. W. Busseti, "Deep learning for time series modeling, Technical report," Stanford University, 2015.

[35] G. Merkel, R. Povinelli and R. Brown, "Short-Term Load Forecasting of Natural Gas with Deep Neural Network Regression," *Energies,* vol. 11, 2008.

[36] T. Kuremoto, S. Kimura, K. Kobayashi and M. Obayashi, "Time Series Forecasting Using Restricted Boltzmann Machine," *Communications in Computer and Information Science,* vol. 304, pp. 17-22, 2012.

[37] E. Mocanu, P. Nguyen, M. Gibescu and W. Kling, "Deep Learning For Estimating Building Energy Consumption," *Sustainable Energy, Grids and Networks,* Vols. 1-10, 2016.

[38] H. Shi, M. Xu and R. li, "Deep Learning for Household Load Forecasting – A Novel Pooling Deep RNN," *IEEE Transactions on Smart Grid,* vol. PP, pp. 1-1, 2017.

[39] S. Hochreiter and J. Schmidhuber, "Long-short term memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[40] A. a. H. J. Gensler, B. Sick and N. Raabe, "Deep Learning for solar power forecasting — An approach using AutoEncoder and LSTM Neural Networks," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016.

[41] M. Shahzad and A. Afshin, "Short-Term Load Forecasts Using LSTM Networks," *Energy Procedia,* vol. 158, pp. 2922-2927, 2019.

[42] 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*, 2016.

[43] W. Kong, Z. Dong, Y. Jia, D. Hill, Y. Xu and Y. Zhang, "Electric load forecasting in smart grids using Long-Short-Term-Memory based Recurrent Neural Network.," *IEEE Transactions on Smart Grid,* vol. PP, 2017.

[44] S. Du, T. Li, X. Gong, Y. Yang and S.-J. Horng, "Traffic flow forecasting based on hybrid deep learning framework," in *12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, 2017.

[45] P.-H. Kuo and C.-J. Huang, "A High Precision Artificial Neural Networks Model for Short-Term Energy Load Forecasting," *Energies,* vol. 11, p. 213, 2018.

[46] A. Kasun, M. D. L. and M. Milos, "Deep neural networks for energy load forecasting," in *IEEE 26th International Symposium on Industrial Electronics (ISIE)*, 2017.

[47] P.-H. Kuo and C.-J. Huang, "An Electricity Price Forecasting Model by Hybrid Structured Deep Neural Networks," *Sustainability,* vol. 10, p. 1280, 2018.

[48] L. Hu, G. Taylor, H.-B. Wan and M. Irving, "A review of short-term electricity price forecasting techniques in deregulated electricity markets," in *Proceedings of the Universities Power Engineering Conference*, 2009.

[49] J. Contreras, R. Espinola, F. Nogales and A. Conejo, "ARIMA models to predict next-day electricity prices," *Power Engineering Review, IEEE,* vol. 22, 2002.

[50] Z. Tan, J. Zhang, J. Wang and J. Xu, "Day-ahead electricity price forecasting using wavelet transform combined with ARIMA and GARCH models," *Applied Energy,* vol. 87, no. 11, pp. 3606-3610, 2010.

[51] S. Voronin and J. Partanen, "Price forecasting in the day-ahead energy market by an iterative method with separate normal price and price spike frameworks," *Energies,* vol. 6, pp. 5897-5920, 2013.

[52] N. K. Singh, M. Tripathy and A. K. Singh, "A radial basis function neural network approach for multi-hour short term load-price forecasting with type of day parameter.," in *6th International Conference on Industrial and Information Systems*, 2011.

[53] L. Jesus, D. R. Fjo and D. S. Bart, "Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms," *Applied Energy,* vol. 221, pp. 386-405, 2018.

[54] G. Dorffner, Neural Networks for Time Series Processing, vol. 6, Neural Network World, 1996, pp. 447-468.

[55] C. Tian, J. Ma, C. Zhang and P. Zhan, "A Deep Neural Network Model for Short-Term Load Forecast Based on Long Short-Term Memory Network and Convolutional Neural Network," *Energies,* vol. 11, 2018.

[56] G. E. Box and G. M. Jenkins, Time Series Analysis: Forecasting and Control, Holden-Day, 1976.

[57] R. Adhikari and R. K. Agrawal, "An Introductory Study on Time Series Modeling and Forecasting," *CoRR,* vol. abs/1302.6613, 2013.

[58] K. Hipel and A. McLeod, Time Series Modelling of Water Resources and Environmental Systems, Amsterdam: Elsevier, 1994.

[59] K. Hammouda and F. Karray, A comparative study of data clustering techniques, tech. rep., Pattern Analysis and Machine Intelligence Research Group, University of Waterloo, 2000.

[60] A. Reynolds, G. Richards and V. Rayward-Smith, The application of k-medoids and PAM to the clustering of rules, Springer, 2004.

[61] P. Rodrigues, J. Gama and J. Pedroso, LBF: Hierarchical time-series clustering for data streams, Proceedings of the 1st International Workshop on Knowledge Discovery in Data Streams (IWKDDS'04), 2004.

[62] P. Langfelder, B. Zhang and S. Horvath, Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R, vol. 24, Bioinformatics, 2007, p. 719–720.

[63] R. Cannon, J. Dave and J. Bezdek, Efficient Implementation of the Fuzzy C-Means Clustering Algorithms, vol. 8, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986, pp. 248-255.

[64] J. Vesanto and E. Alhoniemi, Clustering of the self-organizing map, vol. 11, IEEE Transactions on Neural Networks, 2000, pp. 586-600.

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

[66] J. Mazanec, Positioning Analysis with Self-Organizing MapsAn Exploratory Study on Luxury Hotels, Cornell Hotel and Restaurant Administration Quarterly, 1995.

[67] D. Larose, Discovering Knowledge in Data, New Jersey: John Wiley & Sons, 2005.

[68] T. Kohonen, Self-Organizing Map, 2-nd ed., Springer Verlag, 1997.

[69] S. Wang and H. Wang, Knowledge Discovery Through Self-Organizing Maps: Data Visualization and Query Processing, vol. 4, Knowl. Inf. Syst., 2002, pp. 31-35.

[70] J. Dayhoff, Neural network architectures — An introduction, Van Nostrand Reinhold, 1990.

[71] A. Richardson, C. Risien and F. Shillington, Using self-organizing maps to identify pat- terns in satellite imagery, vol. 59, Progress In Oceanography, 2003, pp. 223-239.

[72] J. Mazanec, Neural market structure analysis: Novel topology-sensitive methodology, vol. 35, European Journal of Marketing, 2001, pp. 894-914.

[73] A. Chitra and S. Uma, An ensemble model of multiple classifiers for time series prediction, International Journal of Computer Theory and Engineering, 2010, pp. 454-458.

[74] L. Deng and D. Yu, Deep Learning: Methods and Applications, vol. 7, Foundations and Trends in Signal Processing, 2014, pp. 1-199.

[75] C. François, Deep learning with Python, Manning, 2017.

[76] R. Adrian, Deep Learning for Computer Vision with Python: Starter Bundle, PyImageSearch, 2017.

[77] J. Patterson and A. Gibson, Deep Learning: A Practitioner's Approach, O'Reilly Media, Inc., 2017.

[78] C. Aggarwal, Neural Networks and Deep Learning: A Textbook, Springer, Cham, 2018.

[79] B. Jason, Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python, Machine Learning Mastery, 2018.

[80] B. Neupane, W. Woon and Z. Aung, Ensemble Prediction Model with Expert Selection for Electricity Price Forecasting, Energies, 2017.

[81] "Naming the coronavirus disease (COVID-19) and the virus that causes it," (WHO), World Health Organization, [Online]. Available: https://www.who.int/emergencies/diseases/novel-coronavirus-2019/technical-guidance/naming-the-coronavirus-disease-(covid-2019)-and-the-virus-that-causes-it. [Accessed 15 05 2020].

[82] I. Ghinai, T. D. McPherson, J. C. Hunter, H. L. Kirking, D. Christiansen, K. Joshi, R. Rubin, S. Morales-Estrada, S. R. Black, M. Pacilli, M. J. Fricchione, R. K. Chugh, K. A. Walblay and Ahmed, "First known person-to-person transmission of severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) in the USA," *Elsevier Public Health Emergency Collection,* vol. 395, no. 10230, pp. 1137-1144, 2020.

[83] S. Moon, "A seemingly healthy woman's sudden death is now the first known US coronavirus-related fatality," 24 04 2020. [Online]. Available: https://edition.cnn.com/2020/04/23/us/california-woman-first-coronavirus-death/index.html. [Accessed 15 05 2020].

[84] J. Castro, P. Achanccaray Diaz, I. Sanches, L. Cue La Rosa and P. R. Nigri Happ, "Evaluation of recurrent neural networks for crop recognition from multitemporal remote sensing images," in *Anais do XXVII Congresso Brasileiro de Cartografia e XXVI Exposicarta 6*, Rio de Janeiro, 2017.

APPENDIX 1

Link to github repository: https://github.com/babushkinvladimir/mscThesisProject

APPENDIX 2

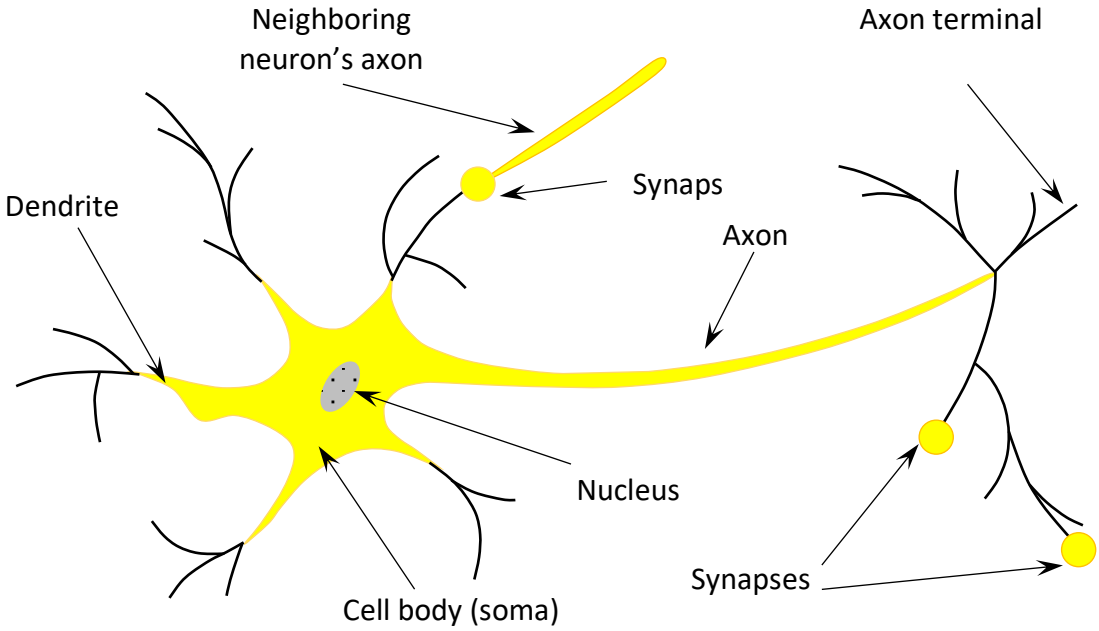


Fig. 28. Diagram of neuron.