January 2018

Regional And Residential Short Term Electric Demand Forecast Using Deep Learning

Tareq Hossen

REGIONAL AND RESIDENTIAL SHORT TERM ELECTRIC DEMAND FORECAST USING DEEP LEARNING

by

Tareq Hossen

Bachelor of Science, Chittagong University of Engineering Technology

A Thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota
May
2018

This thesis, submitted by Tareq Hossen, in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.

Prakash Ranganathan, Ph.D, Chairperson

Hossein Salehfar, Ph.D.

Saleh Faruque, Ph.D.

This thesis meets the standards for appearance, conforms to the style and format requirements of the Graduate School of the University of North Dakota, and is hereby approved.

Grant McGimpsey
Dean of the Graduate School

May 1, 2018
Date

PERMISSION

Title Regional and Residential short term electric demand forecast using Deep learning

Department Electrical Engineering

Degree Master of Science

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in his absence, by the chairperson of the department or the dean of the Graduate School. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my thesis.

Tareq Hossen
May 2018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Dr. Prakash Ranganathan for his continuous guidance, support, and advice which he had provided from the day one. I'm very fortunate to have him as my advisor.

I am thankful to the members of my committee, Dr. Hossein Salefar and Dr. Saleh Faruque for their acceptance of being on my committee as well as their guidance. I would also like to acknowledge the role of the Department of Electrical Engineering at University of North Dakota (UND), Grand Forks, North Dakota for providing me the opportunity to study and conduct research.

My thanks also goes to my friends and colleagues here in Grand Forks for supporting me both academically and socially.

Special thanks to my parents, brother, sister and my wife for their continuous support, patience, and encouragements.

Above all, I am grateful to the Almighty God for giving me the capability and the chance to complete this thesis work.

ABBREVIATIONS

SG	Smart Grid
ARIMA	Auto Regressive Integrated Moving Average
SVM	Support Vector Machine
AI	Artificial Intelligence
DSM	Demand Side Management
STLF	Short Term Load Forecasting
DNN	Deep neural network
RNN	Recurrent neural network
ELU	Exponential linear unit
Re-LU	Rectified linear unit
LSTM	Long short-term memory
GRU	Gated recurrent unit
TF	Tensor flow
MAPE	Mean absolute percentage error
MSE	Mean square error
RMSE	Root mean square error
MAE	Mean absolute error
MIBEL	Iberian Electricity Markets
AMPDs	The Almanac of Minutely Power Dataset

ABSTRACT

For optimal power system operations, electric generation must follow load demand. The generation, transmission, and distribution utilities require load forecasting for planning and operating grid infrastructure efficiently, securely, and economically. This thesis work focuses on short-term load forecast (STLF), that concentrates on the time-interval from few hours to few days. An inaccurate short-term load forecast can result in higher cost of generating and delivering power. Hence, accurate short-term load forecasting is essential. Traditionally, short-term load forecasting of electrical demand is typically performed using linear regression, autoregressive integrated moving average models (ARIMA), and artificial neural networks (ANN). These conventional methods are limited in application for big datasets, and often their accuracy is a matter of concern. Recently, deep neural networks (DNNs) have emerged as a powerful tool for machine-learning problems, and known for real time data processing, parallel computations, and ability to work with large dataset with higher accuracy. DNNs have been shown to greatly outperform traditional methods in many disciplines, and they have revolutionized data analytics. Aspired from such a success of DNNs in machine learning problems, this thesis investigated the DNNs potential in electrical load forecasting application. Different DNN Types such as multilayer perception model (MLP) and recurrent neural networks (RNN) such as long short-term memory (LSTM), Gated recurrent Unit (GRU) and simple RNNs for different datasets were evaluated for accuracies. This thesis utilized the following data sets: 1) Iberian electric market dataset; 2) NREL residential home dataset; 3) AMPds smart-meter dataset; 4)

UMass Smart Home datasets with varying time intervals or data duration for the validating the applicability of DNNs for short-term load forecasting. The Mean absolute percentage error (MAPE) evaluation indicates DNNs outperform conventional method for multiple datasets. In addition, a DNN based smart scheduling of appliances were also studied. This work evaluates MAPE accuracies of clustering-based forecast over non-clustered forecasts.

1 Introduction

1.1 Motivation

Load forecasting play an important role for planning and operation of electric utilities. In today's world, with the high development of electricity market and rapid expansion of power system, load forecasting is becoming an important factor of power system operation scheduling. If the load forecasting is accurate, there will be a great potential savings in the control operations and decision making, such as dispatch, unit commitment, fuel allocation, power system security assessment, and off-line analysis. Thus, how to improve the accuracy of short-term load forecasting has always been the focus of load forecasting study.

Based on various time intervals, load forecasting can be divided into three main categories: short-term forecasting, medium-term forecasting, and long-term forecasting. Short-term forecasting usually forecasts one hour to one-week, medium-term forecasting concerns the future electric load from a week to a month, and long-term forecasting often predicts load of a year or even longer. Short term load forecasting plays an important role in estimating load flows and in making decisions to prevent overloading in power system. The medium-term and long-term forecasting are applied to determine the capacity of generation, transmission, or distribution system. In addition, these types of forecasting required for transmission planning, maintenance scheduling, etc.

Among these three types of forecasting, Short-term load forecasting draws much attention. Short term load forecasting is essentially a time varying signal processing problem. Normally the load forecasting is carried out by making use of past historical data along with influencing factor that effects the load consumption. In recent years, different approach of load forecasting has been proposed. It can be classified as four major categories time series approach, linear and non-linear regression approach, expert system approach and neural network approach[1].

The energy consumption in different region and different level are not same. Therefore, no prediction technique can be claimed as best technique in load forecasting. But it is interesting to notice that neural network-based forecasting model has been widely used in this area based on its powerful nonlinear modeling capability for utility companies in different locations and different level. There are several important key factors that influence the neural network-based forecasting model for example selection of inputs variables, the size of the network, learning rate of the neural network, selection of activation function and the number of epochs of neural training etc.

The goal of this thesis is to adapt the Deep Neural Network (DNN) technology to short-term forecasting of different level (Regional, Residential, Appliance) of electrical power demand and to evaluate the DNNs performance as a forecaster.

1.2 Thesis Contributions:

The following are the objectives of this work:

Objective 1. Develop a deep neural network-based load forecasting of varying data durations using regional, residential and appliance level datasets

To accomplish this objective, following tasks were carried out:

Task 1. Conduct literature review on various load forecasting algorithms suitable for time-series demand datasets.

Task 2. Develop deep neural network forecasting model with different activation functions such as Sigmoid, Rectifier linear unit (Re-LU), and Exponential linear unit (ELU) on 90 days of Iberian electric market (MIBEL) regional datasets. For this purpose, a multi-layered deep neural network was tested using Google's machine learning Tensor-Flow platform.

Task 3. Develop recurrent neural network-based forecasting model to forecast individual residential consumer demand. Then, accuracies for different types of recurrent neural network were compared. Investigated RNNs include Long short-term memory (LSTM), gated recurrent unit (GRU) and simple RNN on a single user with 1 – minute resolution based one year of historical dataset. Conventional forecasting technique such as such as auto regressive integrated moving average (ARIMA), random forest (RF), support vector machines (SVM), Generalized linear models (GLM) also developed to forecast this individual level dataset.

Task 4. Develop a novel smart home appliance scheduling technique using deep learning. Parameters considered include price, demand, energy ratings and constraint formulations using linear programming.

Objective 2. Investigate the effects of clustered versus non-clustered forecasts on MAPE accuracies.

Task 5. This task considers 200 residential user and demand profiles to study clustered versus non-clustered based forecasts.

1.3 Thesis Organization

The thesis is organized as follows. **Chapter 2** presents regional short-term load forecasting using deep neural network with multiple type of activation functions. **Chapter 3** talks about the individual residential consumer demand forecasting using different recurrent neural network and other conventional method of forecasting. **Chapter 4** discusses scheduling of smart home appliance using deep learning. **Chapter 5** presents clustering based short term load forecasting using deep neural network. Finally, **Chapter 6** summarizes the conclusion and provides direction for the future work.

1.4 Publications

- Tareq Hossen, Siby Jose Plathottam, Radha Krishnan Angamuthu, Prakash Ranganathan, and Hossein Salehfar. "Short-term load forecasting using deep neural networks (DNN)." In Power Symposium (NAPS), 2017 North American, pp. 1-6. IEEE, 2017.
- Tareq Hossen, Arun Sukumaran Nair, Sima Noghanian, Prakash Ranganathan “Optimal Operation of Smart Home Appliances using DNN” submitted in Power Symposium (NAPS), 2018 North American, pp. 1-6. IEEE, 2018.
- Tareq Hossen, Arun Sukumaran Nair, Radha Krishnan Angamuthu, Prakash Ranganathan “Residential Load Forecasting Using Deep Neural Networks (DNN)” Submitted in International Electro/Information Technology 2018 North American, pp. 1-6. IEEE, 2018.
- Tareq Hossen, Mitch campion, Prakash Ranganathan, “Improving residential load forecasting based on Deep Neural Network (DNN) and K-shape clustering” (Under preparation).
- A. Sukumaran Nair, P. Ranganathan, T. Hossen, and N. Kaabouch, “Multi-Agent Systems for Resource Allocation and Scheduling in a Smart Grid: A Survey,” Technology and Economics of Smart Grids and Sustainable Energy (Under Rev).

2 Regional short-term load forecasting using deep neural network

2.1 Overview

Load forecasting is an important electric utility task for planning resources in Smart grid. This function also aids in predicting the behavior of energy systems in reducing dynamic uncertainties. The efficiency of the entire grid operation depends on accurate load forecasting. This chapter proposes and investigates the application of a multi-layered deep neural network to the Iberian electric market (MIBEL) forecasting task. Ninety days of energy demand data are used to train the proposed model. The ninety-day period is treated as a historical dataset to train and predict the demand for day-ahead markets. The network structure is implemented using Google's machine learning Tensor-flow platform. Various combinations of activation functions were tested to achieve a better Mean Absolute percentage error (MAPE) considering the weekday and weekend variations. The tested functions include Sigmoid, Rectifier linear unit (Re-LU), and Exponential linear unit (ELU). The preliminary results are promising, and show significant savings in the MAPE values using the ELU over other activation functions.

2.2 Background and related work

Load forecasting play a vital role in operations and planning of electrical power and energy systems. Data related to load forecasting are non-linear in nature, making the load prediction task a challenging one. One simple approach, however, to load forecasting is using regression techniques. Regression is a statistical procedure for estimating the relationship between dependent and predictor (independent) variables. It allows one to see how the dependent variable changes with respect to changes in the independent variable. The advantage of this method is that it is easily understandable. The limitation of this approach is there exist a high degree of over fit. The other disadvantage the linear regression is too simple to capture the complex relationship in multi-variate data sets [2]. Logistic regression is the adaptation of linear regression to problem classification (e.g., yes/no questions, groups etc.) This method also has high probability and challenges to over-fit the model [2]. In decision trees, a graph-based branching method is used to match all the possible outcomes for a decision. It is used normally for a simple problem and not potent enough to solve complex data[3][4]. Other popular method of forecasting is the Random forest. Random forest takes the mean of many decision trees–each of which is made with some random samples. Each tree is weaker than a full-decision tree. When this individual tree is combined with others, it yields a better result. This method is fast to train and can work with high quality models[3]. Gradient boosting uses weaker trees. In this method, a small change in training set can create radical change in the model. This could be its limitation[4]. The other well-known method of load forecasting is using neural-network. In neural network,

the interconnection of neuron passes messages to each other with one or more hidden layers in between them. In deep learning, several hidden layers are placed one after the other. This method can handle extremely complex tasks with high accuracy. The disadvantage of this method is it is very slow to train and require a lot of power. The other disadvantage of this method is it is almost impossible to understand the prediction[5]. A neural network consists of an input layer, varying no of hidden layers, and an output layer. These layers are connected by neuron which processes the data. Neural network with a single layer is not capable of understanding the complex relationship between input and output. A neural network with more than three hidden layers is known as a deep neural network. A deep neural network has a better capability of feature abstraction of input and output pattern[6]. Recent development in the field of big data and internet of things (IoT) increase the acceptance of deep neural network (DNN) in multiple research disciplines [6]. In [7], a three-layered neural network based backpropagation technique was developed for load forecasting. In this work multilayer perception neural (MLP) networks have been used to forecast the demand from domestic users.

2.3 Short term regional load forecasting

The Electricity demand forecast process involves multiple steps that include data cleansing, data preparation, and data evaluation. The following are steps involved:

1- (Gather Load data): We collected the load consumption data through the web-link provided by the Iberian Electricity market.

2 – Glean and order the data: We used the 90 days of the dataset to build the model and evaluate how well the model generalizes to future results.

3 – Training a model on the data

To model the relationship between the predictor variables used in modeling and the electricity demand, we used deep neural networks multi-layer perception model. The neural network model of this work is developed using the Tensor-Flow deep learning platform[8]. Tensor-Flow is an open source software library for numerical computation using data flow graphs. The nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicating between the nodes. The flexible architecture of Tensor-Flow allows one to deploy computations to one or more Central or Graphical Processing Units (CPUs or GPUs) on a desktop, server, or mobile device with a single Application Process Interface (API). Tensor-Flow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research. However, the Tensor-Flow system is general enough to be applicable to other domains as well [8].

The choice of a neural network structure depends on several factors. In general, neural network modeling is divided into five steps [9]

a. Select input and output variables.

b. Build the neural network model.

c. Cluster training, test, and validation data.

d. Train the neural network model with the training data set.

e. Validate the neural network model.

The selection of inputs to the neural network is an important aspect of the design. In this work, we consider temperature, wind-speed, solar irradiance, and the prior load data as input. These data sets are normalized to accommodate the application of activation functions. The following expression is used to normalize load:

$$Normalized\ \ load = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{1}$$

Where x is he actual data value. To build the neural network model, a trial and error method select the number of hidden layers and neurons. The datasets are categorized into three sets: training, validation, and testing sets. The test and validation data sets are not used to train the neural network. These datasets are used to evaluate the error by comparing the obtained results with the actual data. Training of the neural network is a process of determining the network weights that provide a minimum error. As an acceptable minimum level of training error does not ensure the same level of performance with all other related input datasets, it is often necessary to validate the network performance once the training process is completed [9].

2.4 Implementation of deep neural networks

In this work, we developed both shallow and deep neural networks on a Tensor Flow platform. After building the model, it is tested using various combinations of activation functions and neurons.

2.5 Structure of DNN

The numbers of layers and neurons are key in modeling neural network structures. In shallow neural network (i.e., single hidden layer), we can only vary the number of neurons that are in the single hidden layer, while both the width and the depth of the network can be changed in DNN. For both shallow neural network (SNN) and deep neural network (DNN), the number of input and output neurons is predetermined according to the dimension of the training set and the forecasting period [10]. A heuristics method was used to choose the number of hidden neurons in the shallow neural network (SNN) in [10]. In DNN, it is not possible to do the same as SNN, as one has to consider both the width and the depth of the network [11]. The structure of DNN model used in this work is given below:

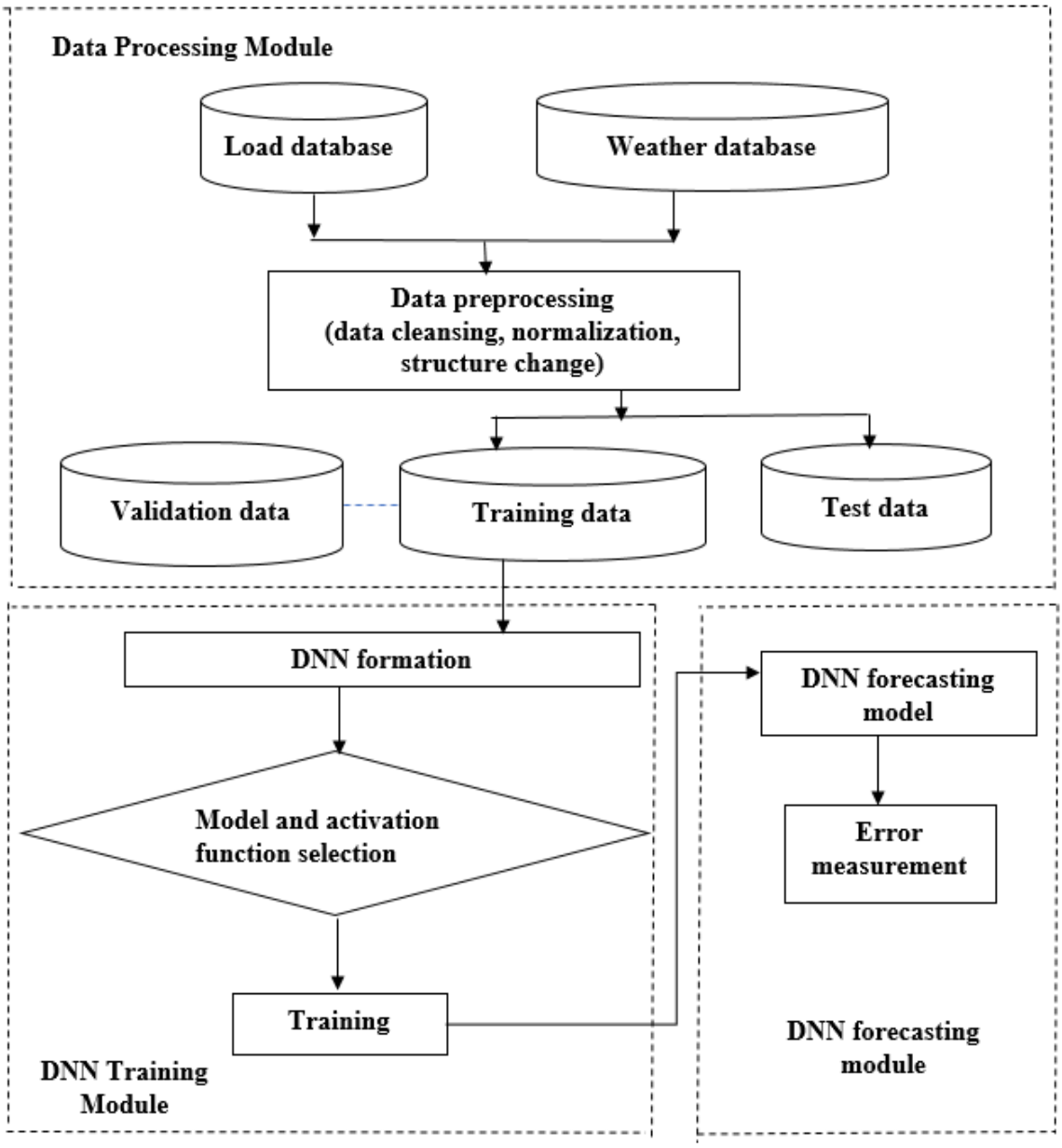

Figure 2-1: DNN based short term load forecast

2.6 Dataset

The size of dataset impacts the accuracy, training, and transfer of learning within the deep neural network [12]. For example, the 2016 DNN competition [13] uses 456,567 images for visual recognition applications. The authors of the present

work used 90-days hourly data of Iberian Energy Market Operator (MIBEL). We selected a single day (July 31, 2015) to forecast and validate the performance with different deep neural network model.

2.7 Activation function

Activation function is a deciding parameter that evaluate and capture the trends or feature patterns from within the data. If the output value from the activation function is zero, the feature is absent and if the value is one the feature is present in the data. In computational networks, the activation function of a node defines the output from that node given an input or a set of inputs. A standard computer chip circuit be a digital network of activation functions that can be either "ON" (1) or "OFF" (0), depending on the input. This is like the behavior of the linear perceptron in neural networks. However, the nonlinear activation function allows such networks to compute nontrivial problems using only a small number of nodes. In artificial neural networks, this function is also called the transfer function. In training the multi-layer neural network model, activation function plays an important role in adjusting the weights. In this work, the authors have used a non-linear sigmoid and a rectified linear unit function for hidden layers in the model.

2.7.1 Sigmoid function

The sigmoid function is defined as

$$f(x) = \frac{1}{1+e^{-x+\alpha}} \qquad (2)$$

where x is input to a neuron. α is the offset parameter and sigmoid function evaluates its value as zero. The sigmoid function is very similar to the step function, which acts like threshold. When x is a large positive number, the output of the sigmoid function is near to 1.

The difference between the linear and sigmoid functions is that the sigmoid one saturates to a high value of x. The sigmoidal function can make the gradient-based learning difficult. For this reason, use of linear function in deep neural network is discouraged [14].

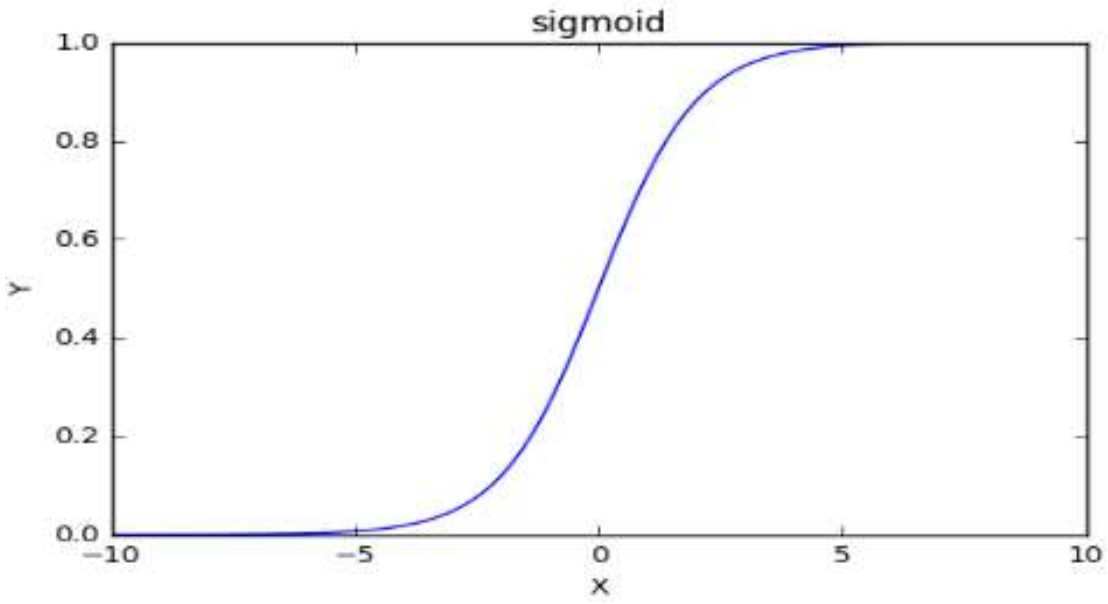

Figure 2-2: Sigmoid function

2.7.2 Rectified linear unit (RELU)

Rectified linear unit is defined as

$$R(x) = \max(0, x) \tag{3}$$

where x is the input to the neuron.

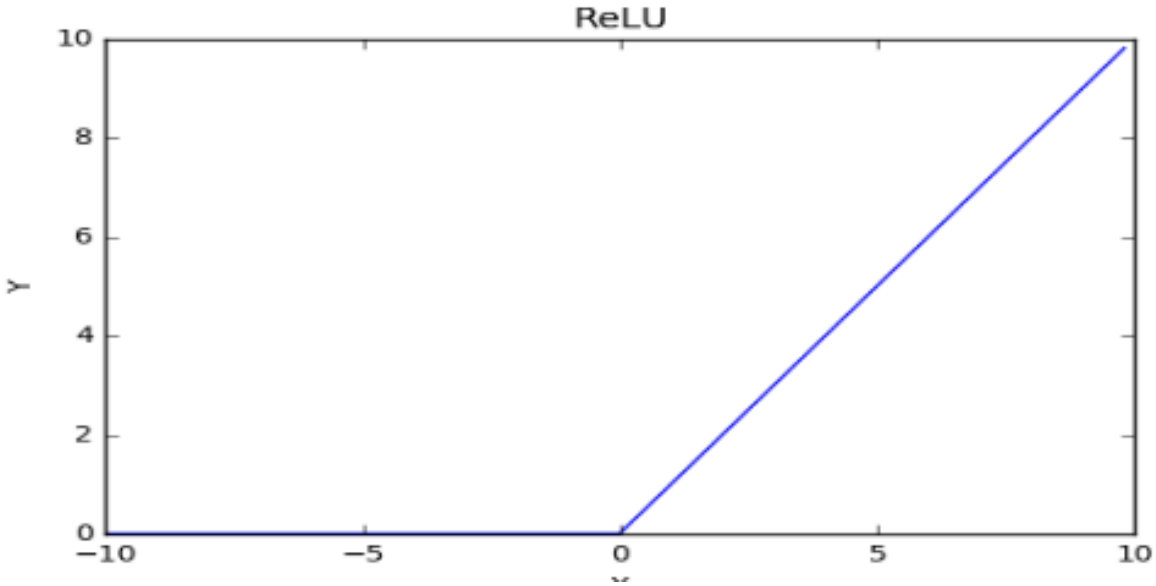

Figure 2-3: Rectified linear unit

ReLU is similar to linear function with the only change having an output that is zero across half of its domain. Specifically, ReLU has the two following advantages [15]:

i) It is very quick to use and train when compared with other activation functions.

ii) It is not the subject to the vanishing gradient problem. The limitation of ReLU, however, is that its mean output is not zero. For deep neural networks, this function

introduces a bias for the next-layer which can slow down the learning process of the network.

2.7.3 Exponential linear unit (ELU)

The exponential linear function is defined as

$$f(x) = a(e^x - 1), x < 0 \text{ otherwise } f(x) = x \quad (4)$$

where, α is a parameter to be chosen and x is the input.

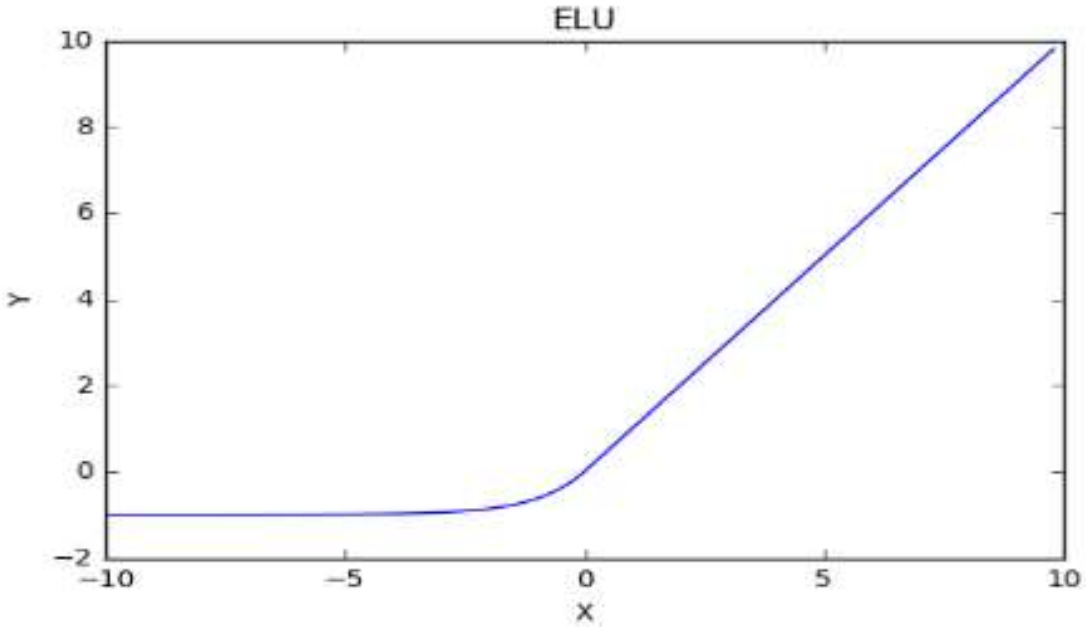

Figure 2-4: Exponential linear unit

ELU function behaves like the ReLU unit for values of x that are positive, but for all negative values, this function is bounded by a fixed value of α is -1 for. This behavior helps to push the mean activation of neurons closer to zero which is beneficial for learning and is robust to noise [16].

2.8 System model

For a proper development and implementation of neural networks, a sound training data set is an absolute necessity. This requires several parameters during the design phase such as the availability of historical load data; number of neurons; selection of the number of layers; and the activation functions. For our system model, temperature, wind-speed, solar irradiance, and the prior load data used as input. In this chapter, several short-term load forecasting cases using various combinations of activation functions were investigated. An ADAM optimizer inside Tensor-flow framework was used to train our model. This optimizer uses a gradient- descent algorithm. This method has a faster convergence rate compared to the stochastic gradient descent (SGD) approach [17].

2.8.1 Case 1. ReLU Activation Function with Single Hidden Layer

In this case, five input variables, ReLU activation function, were selected to do the forecast. The number of neurons is varied randomly until a better mean absolute percentage error (MAPE) result is obtained.

2.8.2 Case 2. ReLU Activation Function with two hidden layers

In this case, ReLU activation function, and two hidden layers were selected to do the forecast. The number of neurons is varied randomly until a better MAPE result is obtained.

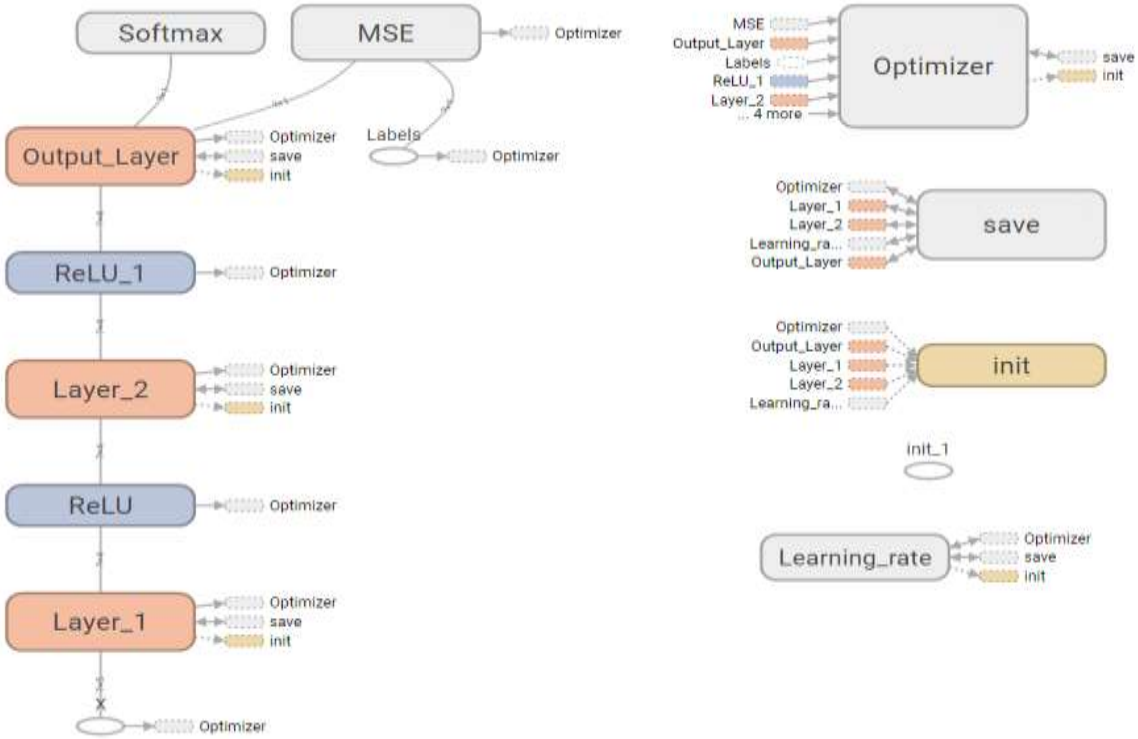

Figure 2-5: Tensor graph for case 2

2.8.3 Case 3. ReLU - Sigmoid Combination

In this case, ReLU and Sigmoid activation functions, and two hidden layers were selected to do the forecast. The number of neurons are varied randomly until a better MAPE result is obtained. ReLU is used in layer 1 and sigmoid is used in layer 2.

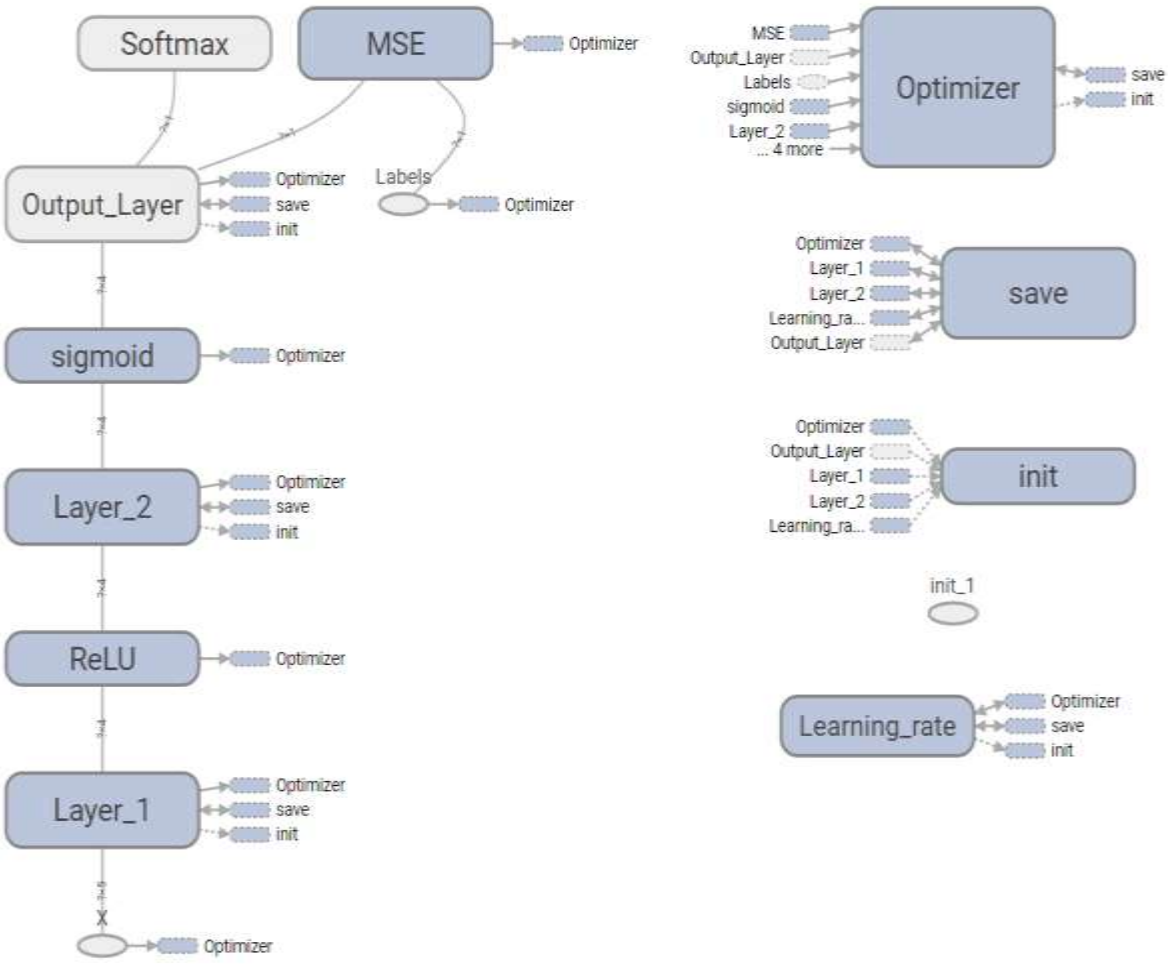

Figure 2-6: Tensor graph for case 3

2.8.4 Case 4. ELU Activation Function with Single Hidden Layer

In this case, ELU activation function, and one hidden layer is selected to do the forecast. The number of neurons are varied randomly until a better MAPE result is obtained. We used ELU in layer 1.

2.8.5 Case 5. ELU Activation Function in two hidden layers

In this case, ELU activation function, and two hidden layers were selected to do the forecast. The number of neurons are varied randomly until a better MAPE result is obtained. We used ELU in both layers 1 and layers 2.

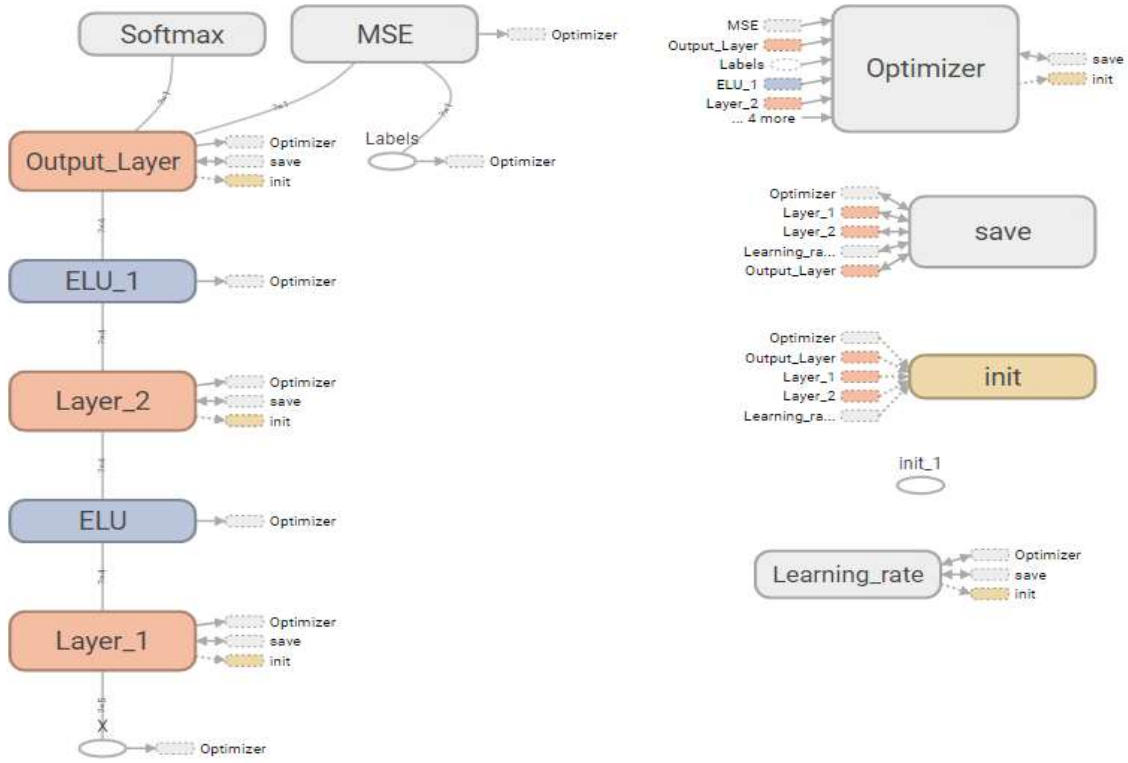

Figure 2-7: Tensor graph for case 5

2.8.6 Case 6. ReLU-ELU Combination

In this case, ReLU is used as the activation function in layer 1 and ELU is used for that in layer 2.

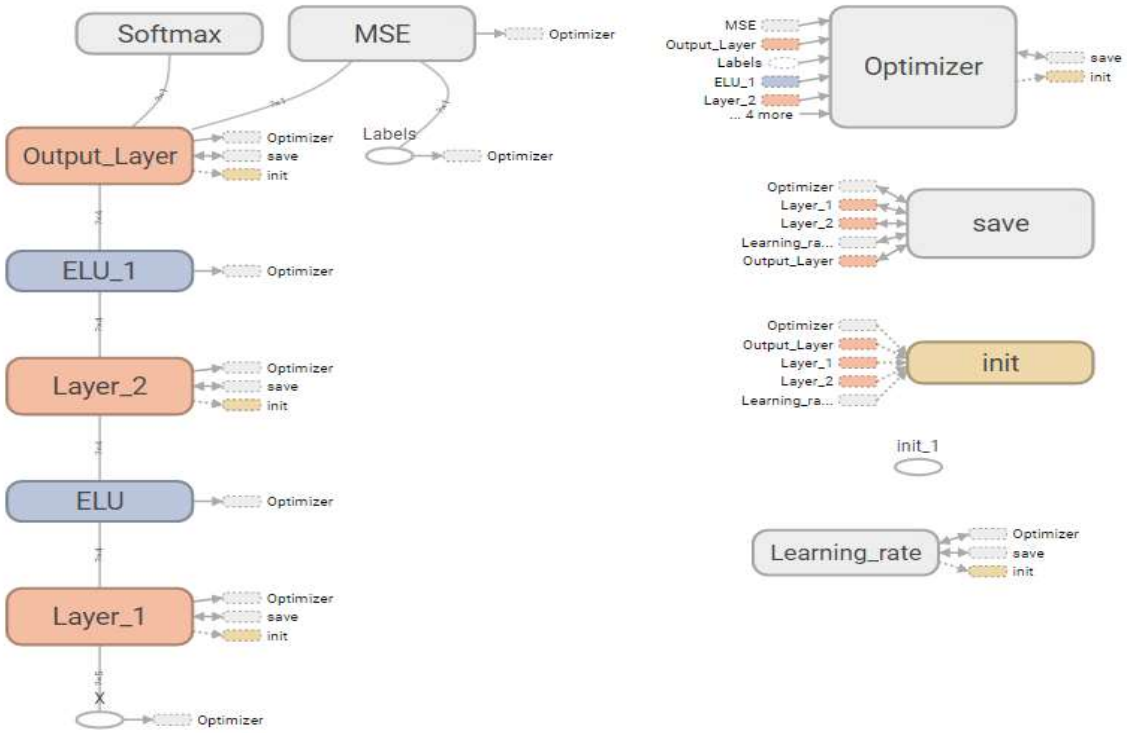

Figure 2-8: Tensor graph for case 6

2.8.7 Case 7. Sigmoid-ELU Combination

In this case, sigmoid and ELU activation functions, and two hidden layers were selected to do the forecast. The number of neurons are varied randomly until a better MAPE result is obtained. We used sigmoid in layer 1 and ELU in layer 2.

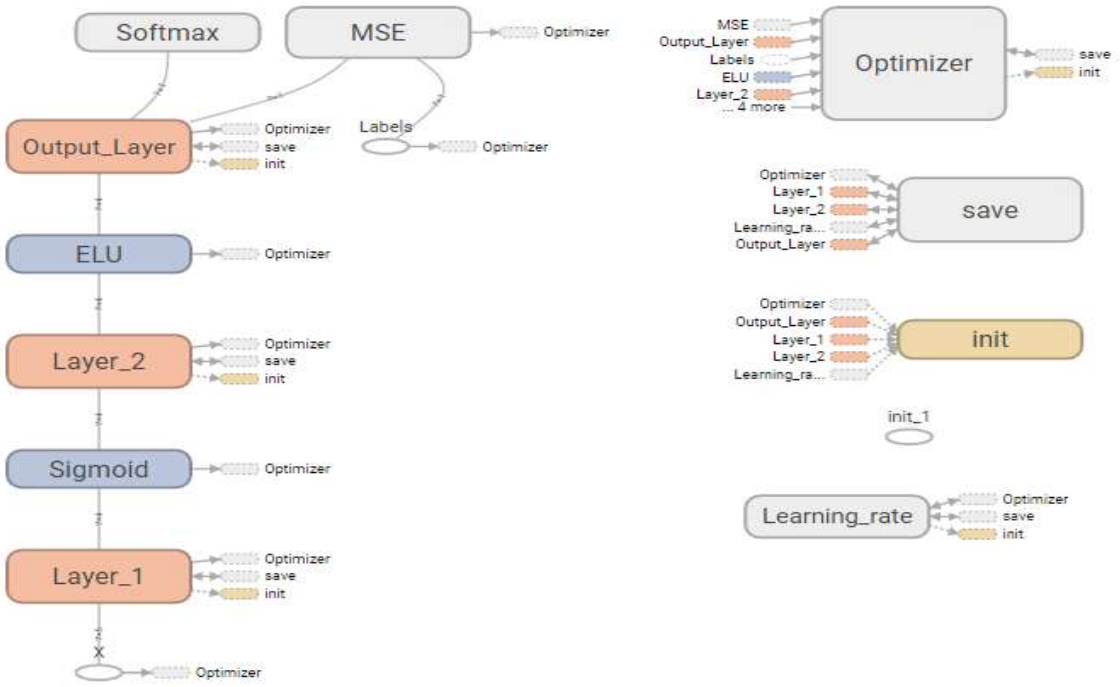

Figure 2-9: Tensor graph for case 7

We also tested the sigmoid-sigmoid combination for layer 1 and 2 respectively, but this combination yielded very poor MAPE values. So, we disregarded the result. Tensor-flow is used to training and test the designed model. In case of designing and training deep neural network it becomes complex and confusing.

2.9 Error Metrics for Evaluation

To assess the accuracy of the forecasting model, three metrics are used: mean absolute percentage error (MAPE), mean square error (MSE), and root mean square error (RMSE).

$$MAPE = \frac{1}{T}\sum_{t=0}^{t=T}\frac{F_T - A_T}{A_T} \times 100\% \quad (5)$$

$$MSE = \frac{\sum_{t=1}^{t=T} (A_T - F_T)^2}{T} \quad (6)$$

$$RMSE = \sqrt{\frac{\sum_{t=1}^{t=T} (A_T - F_T)^2}{T}} \quad (7)$$

where,

F_T = forecasted load, A_T = actual load and T= test set size.

2.10 Simulation Parameters

The simulation settings, and parameters used in this work are listed in Table 1.

Table 1: Tensor-flow parameters

Parameters	Values
Total number of samples	2208
Training samples	2184
Test samples	24
Minimum load data at same hour on previous day, P.DD-1	3640.1 KW
Minimum load at same hour on previous week, P.DD-6	3640.1 KW
Minimum temperature	3 degrees Celsius
Maximum load at same hour on previous day, P.DD-1	7177.2 KW
Maximum load at same hour on previous day, P.DD-6	7177.2 KW
Maximum temperature	35.7 degrees Celsius
Maximum irradiance	975.9 kw
Maximum wind-speed	8.1 m/s
Learning rate	0.001
Training epochs	500000

From Table 2, it is obvious that ELU function outperformed the other functions. This is because ELU does not saturate for the larger values of input x, and the mean of the ELU activation function is closer to zero. This is because of the negative portion of the function characteristics that ensures faster and accurate learning of DNNs.

Table 2: MAPE evaluations for entire 90-day data sets

Activation function	Number of neurons	MAPE(%)	RMSE
Case 1: Single layer ReLU	4	1.641	111.02
	6	1.66	112.88
	8	1.77	118.17
	10	1.62	110.86
	12	1.628	109.86
Case 2: Double layer ReLU- ReLU	3	1.52	115.92
	4	1.73	104.4
	5	1.83	120.14
	7	1.6	109.9
	9	1.48	104.29
Case 3: Double layer ReLU- Sigmoid	3	3.7	297.3
	4	2.2	172.8
	6	1.9	154.2
	8	2.7	205.8
	10	3.9	254.5
Case 4: Single layer ELU	3	1.79	119.46
	4	1.5	103.54
	6	1.71	113.96
	8	1.56	107.74

	10	1.49	103.90
Case 5: Double layer ELU- ELU	3	1.008	73.42
	4	1.192	85.74
	6	1.52	107.9
	8	1.36	98.06
Case 6: Double layer RELU- ELU	3	1.86	121.16
	4	1.58	106.52
	6	1.39	97.67
	8	1.77	117.49
	10	1.93	127.11
Case 7: Double layer Sigmoid- ELU	3	2.34	189.75
	4	2.63	182.63
	6	2.6	185.01
	8	3.05	213.62
	10	3.29	211.667

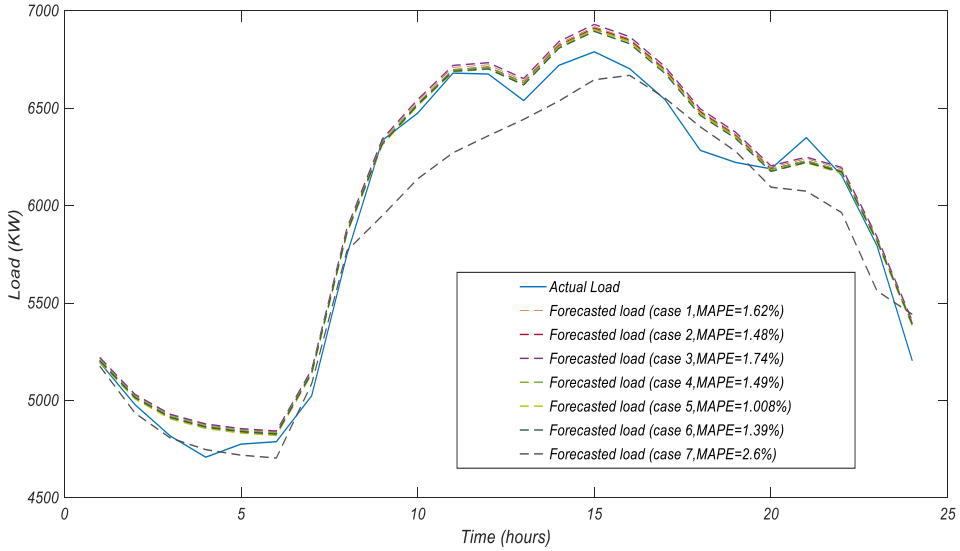

Figure 2-10: Comparison of different cases load forecast with actual data versus MAPE values

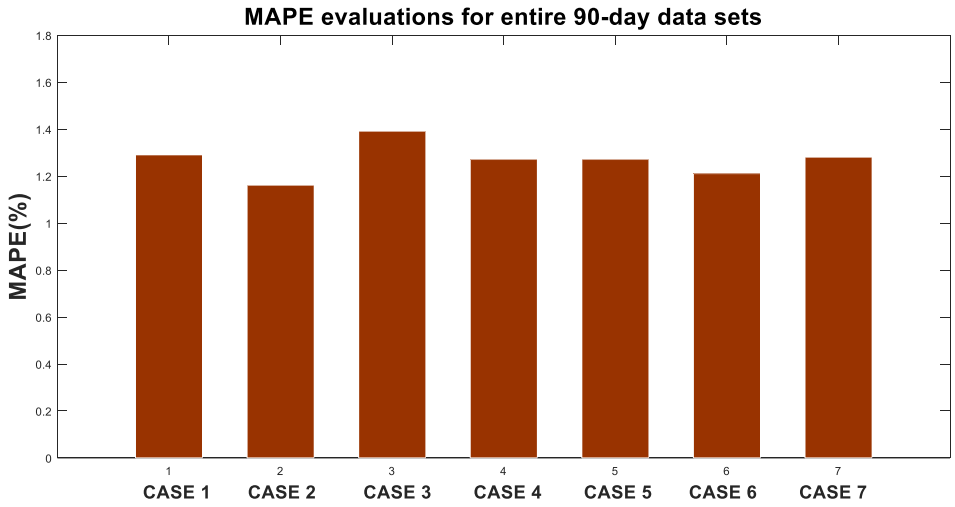

Figure 2-11: MAPE evaluations for entire 90-day data sets

Figure 2-11 show the forecasted graph for all 7 cases with actual demand data. It is observed that case 5 yield better results, and case 7 yield poor results. The ELU-ELU combination with two layers performs better with the 90-day data set. This isdue to the closer correlation of the 90-day pattern profile with the ELU activation function.

Table 3: Weekend MAPE evaluations

Activation function	**Number of neurons**	**MAPE(%)**	**RMSE**
Case 1:	3	1.29	73.22
Single layer	4	1.3	74.53
ReLU	6	1.31	74.95
	8	1.32	75.29
Case 2:	4	1.25	70.94
Double layer	6	1.22	63.81

ReLU- ReLU	8	1.16	64.88
	10	1.20	67.66
Case 3: Double layer Sigmoid -ReLU	4	1.39	82.31
	6	1.51	100.52
	8	2.03	119.65
	10	1.59	94
Case 4: Single layer ELU	4	1.34	76.1
	6	1.27	72.29
	8	1.36	77.53
	10	1.34	76.35
Case 5: Double layer ELU- ELU	4	1.53	79.2
	6	1.52	86.3
	8	1.31	75.09
	10	1.27	72.34
Case 6: Double layer RELU- ELU	4	1.53	86.47
	6	1.21	68.43
	8	1.41	80.24
	10	1.39	78.47
Case 7: Double layer Sigmoid- ELU	4	1.28	73.17
	6	1.69	106
	8	2.37	133.6
	10	1.96	126.22

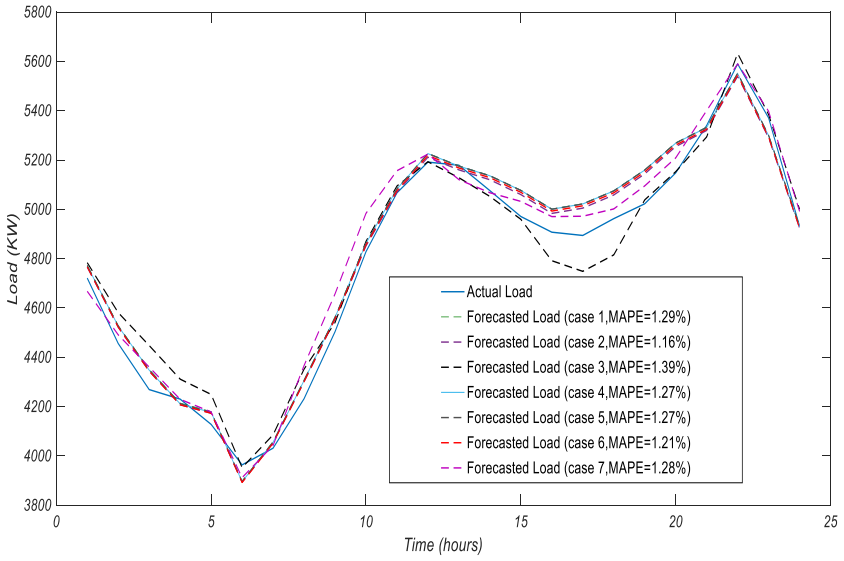

Figure 2-12: Weekend Forecasts

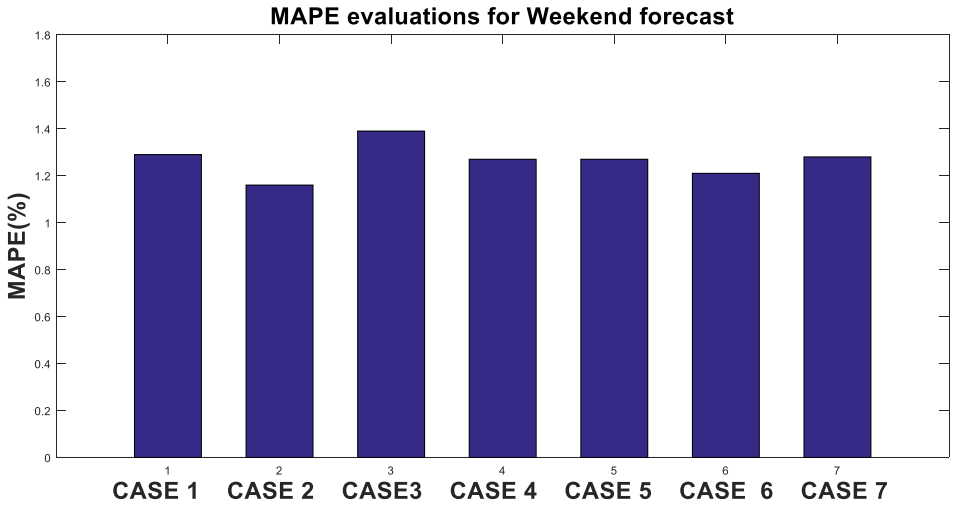

Figure 2-13: MAPE evaluations for Weekend forecast

Table 4: Weekday MAPE evaluations

Activation function	Number of neurons	MAPE(%)	RMSE
Case 1:	4	2.19	150.21
Single layer	6	2.20	150.02
ReLU	8	2.29	157.66
	10	2.31	159.67
Case 2:	4	2.31	158.5
Double layer	6	2.34	161.65
ReLU- ReLU	8	2.36	162.63
	10	2.30	158.748
Case 3:	4	3.27	241.92
Double layer	6	3.44	249.11
Sigmoid -ReLU	8	3.13	206.62
	10	3.24	206.56
Case 4:	4	2.25	154.44
Single layer	6	2.24	154.27
ELU	8	2.19	150.3
	10	2.20	151.39
Case 5:	4	2.29	158.67
Double layer	6	2.03	142.15
ELU- ELU	8	2.43	169.89
	10	2.42	167.86
Case 6:	4	2.31	159.2
Double layer	6	2.12	143.24
RELU- ELU	8	2.38	165.08
	10	2.27	155.84
Case 7:	4	2.75	182.01
Double layer	6	3.97	250.13

Sigmoid- ELU	8	3.28	216.19
	10	3.6	234.53

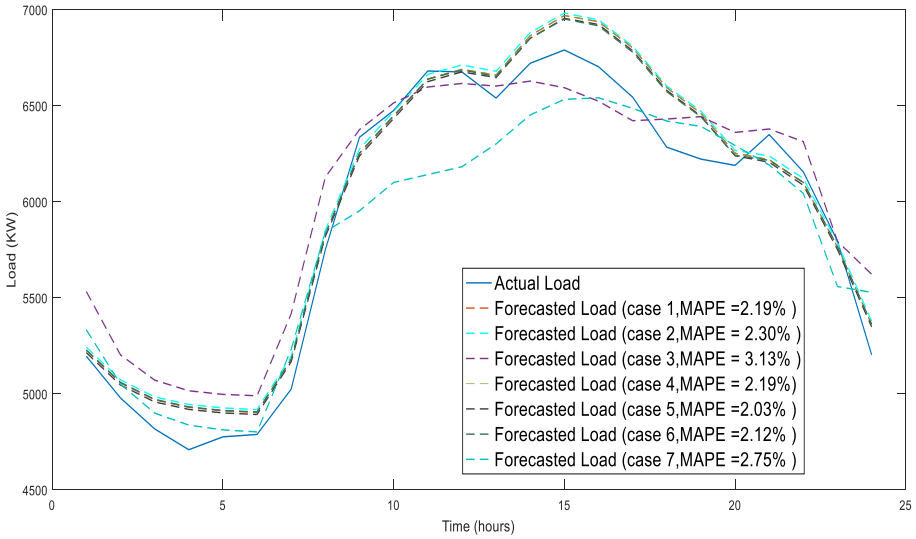

Figure 2-14: Weekday Forecasts

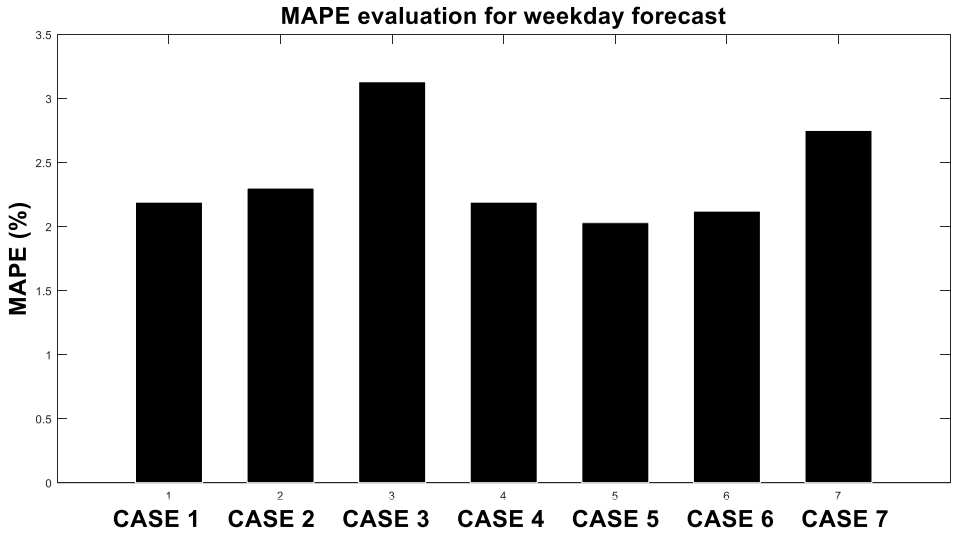

Figure 2-15: MAPE evaluation for weekday forecasts

ELU function performed, because it has improved learning characteristics compared to other activation functions. In contrast to the Sigmoid and ReLU activation functions, the ELU values are negative in x-axis which allows to make the mean value closer to zero, and this speed up the learning process, and enable the gradient closer to the unit's natural gradient.

This chapter of thesis investigates the application of DNN's in electric power system analysis. This sections of thesis uses a 90-day Iberian market dataset to predict the day-ahead loads. Multiple combinations of activation functions were trained, and tested with both single and double-layer neural networks. The results indicate that the combination of ELU with ELU performs better than other combinations when evaluated against MAPE values. On weekend data sets, the ReLU-ReLU combination outperform other combinations.

3 Residential load forecasting using deep neural networks (DNN)

3.1 Overview

Forecasting consumer electricity usage plays an important role for reliable smart grid. As the activities of an individual residential consumer has many uncertain variables, it is hard to accurately forecast the varying residential load levels. For planning electrical resources and to balance demand and supply, accurate forecasting are critical tasks. This chapter presents Deep Neural Network (DNN) based short-term load forecasting for residential consumers. In this work, we compare the Mean Absolute Percentage Error (MAPE) values for residential electricity dataset using different types recurrent neural networks (RNN). Our preliminary results indicate that Long short-term memory (LSTM) based RNN performed better compared with simple RNN and gated recurrent unit (GRU). This method is deployed for a single user with 1-minute resolution based on one year of historical data sets.

3.2 Background and Related work

Load forecasting has remained an important research area for conducting planning operations in electrical power systems. The advanced metering infrastructure (AMI) technology revolutionized the mass adaptation of smart meters at residential consumer levels by utilities. A significant portion (e.g., 20% to 40%) of the total electricity energy production is consumed by residential loads [18]. Load forecasting based on smart metering datasets that are within 1-minute intervals are relatively new area of research. In [19], authors introduced a methodology of

short-term functional time-series based forecast to predict household-level electricity demand. A Kalman -filter based forecasting model to predict the residential load is discussed in [18] and forecasting using conditional kernel density estimation is discussed in [20]. A hybrid model approach for forecasting future residential electricity consumption for buildings is developed in [21]. An occupancy model has been developed for residential load forecasting and discussed in [22]. Artificial Intelligence (AI) based forecasting techniques such as Fuzzy logic [23], artificial neural networks [9], support vector machines [24] and wavelets neural networks [25] are investigated under short term load forecasting. The AI methods are most conducive due to their ability to handle non-linear relationships between dependent and independent variables. Recently, DNN has been successfully used in application such as image processing, automatic speech recognition, natural language processing and for time-series modeling tasks such as load forecasting. There are several review papers on load forecasting focused at aggregated level for commercial users. However, there are limited work on residential level data sets. We think that this is because short term load forecasting at granular level is extremely challenging due to uncertainty and volatility [18]. Most short-term load forecasting models focuses on regular pattern that are easily predictable. Residential loads are more uncertain due to erratic and stochastic nature of consumer behavior that are hard to predict. This chapter represents a deep learning- based method for the meter-level load forecasting for residential consumers. We have used recurrent neural network for residential load forecasting. We compare our forecasting accuracy by using different recurrent

neural network (RNN) models for residential dataset. We also test our dataset with other conventional time-series analysis such as ARIMA, Generalized linear model (GLM), Random Forest (RF) and machine learning approaches Neural network and Support Vector machine (SVM) methods. The rest of the chapter is organized as follows: section 2 discusses background information on RNN, LSTM, GRU and some related work, section 3 discusses implementation platforms, and section 4 focuses on preliminary results and discussions.

3.3 Implementation of recurrent neural networks for residential load forecasting

In this chapter, the effectiveness of different DNN models is investigated for residential level forecasting. We have developed different RNN models to predict day ahead residential demand using smart meter dataset. We also used different conventional method of forecasting such as ARIMA, GLM, Random Forest, neural network and support vector machine for day ahead forecasting.

3.4 Structure of RNN

Recurrent neural networks (RNNs) are fundamentally different from traditional feedforward neural network. They are sequence-based models, which are able to establish the temporal correlations between previous information and the current circumstances. For time series problem, this means that the decision an RNN made at time step $t-1$ could affect the decision it will reach at time step later t .Such characteristic of RNNs is ideal for the load forecasting problems of individual households, since it has been pointed out that residents intrinsic daily routines may

be one of the most important factors to the energy consumption at the later time intervals.

RNN can have a signal traveling in both directions by introducing a loop in the network. The computations in RNN derived from earlier input are fed back to the network, which gives them a special memory. A RNN can be thought of as multiple copy of same network, each passing a message to successor [26]. An unrolled structure of RNN is given below: In Figure 1, a chunk of neural network A, looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one stage of the network to the next stage.

3.5 Recurrent neural network model for short term load forecasting

We have used three different recurrent model structures for our load forecasting. The structure of these three RNN is given below:

3.5.1 Simple RNN

Simple RNN accepts input x_t at time t and the status is updated by a non linear mapping f from time to time. One simple way defining the recurrent unit f is linear transformation plus a non-linear activation.

$$h_t = \tanh(w[h_{t-1}, x_t] + b) \tag{8}$$

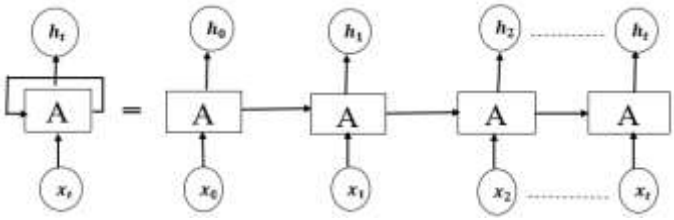

Figure 3-1: An unrolled simple recurrent neural network

3.5.2 Long short term memory (LSTM)

LSTM is a special kind of RNN that was introduced by Hochreiter & Schmidhuber [27]. This type of RNN is apparently modeled to avoid the long term dependency problem [28]. All the RNN models are of the form having a chain of repeating modules of neural networks. In standard RNNs, this repeating module will have a simpler structure, such as a single tanh layer. In RNN, instead of having a single neural network layer, there are four layers. The step by step operation of LSTM can be represented by the following equation:

$$f_t = \sigma(w_f.[h_{t-1}, x_t] + b_f) \tag{9}$$

$$\bar{C}_t = \tanh(w_c).[h_{t-x}.x_t] + b_c \tag{10}$$

$$C_t = f_t \times C_{t-1} + i_t \times C_t \tag{11}$$

$$O_t = \sigma(w_o.[h_{t-1}, x_t] + b_o) \tag{12}$$

$$h_t = O_t \times \tanh(C_t) \tag{13}$$

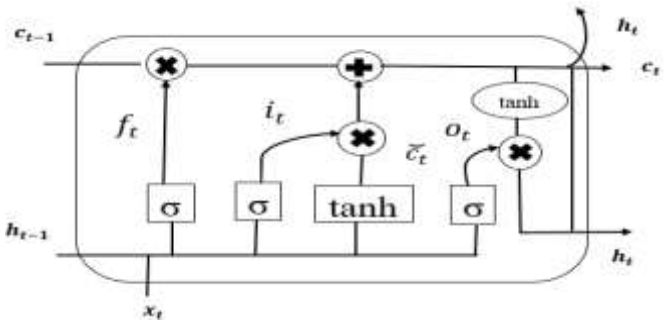

Figure 3-2: Long short term memory (LSTM).

From Figure. 2, of a LSTM network, each block has two parallel lines going in and out. The top line is the cells, the essential point is the hidden state information. Finally, there is a third line going in from the bottom, representing X. In total, three inputs and two outputs. x_t would be X_train,h_{t-1} would be h_previous, x_{t+1} would be X_train.next, and h_t would be h_current.

3.5.3 Gated recurrent unit(GRU)

Gated Recurrent Unit, or GRU is a modified version of the LSTM . GRU is introduced by Cho in 2014 [28] that combines the forget and input gates into a single update gate. In contrast to LSTM, a GRU network only has two inputs and one output and no cell layers [29]. GRU unit takes X_train and h_previous as inputs. They perform certain calculations and then pass along h_current. In the next iteration X_train.next and h_current are used for more calculations.

The step by step operation of GRU can be represented by the following equation.

$$z_t = \sigma(w_z.[h_{t-1,}x_t]) \quad (14)$$

$$r_t = \sigma(w_r.[h_{t-1,}x_t]) \quad (15)$$

$$\tilde{h}_t = \tanh(w.[r_t * h_{t-1,}x_t]) \quad (16)$$

$$h_t = (1-z_t) * h_{t-1} + z_t * \tilde{h}_t \quad (17)$$

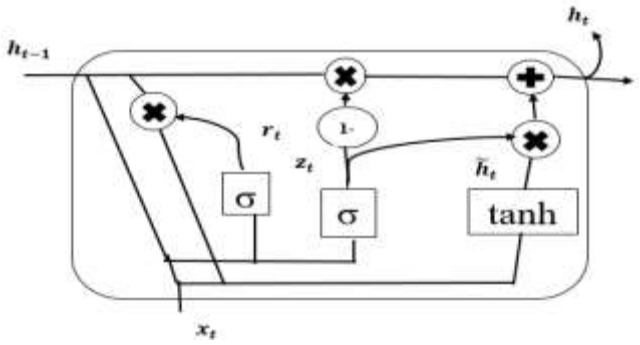

Figure 3-3: Gated recurrent unit (GRU).

3.6 Implementation of recurrent neural networkS (RNN) for residential load forecasting

In this chapter, we have used keras on top of tensor-flow to implement the recurrent neural network. Keras is a high-level neural network library written in python and it can run on top of either Theano or tensor flow [30]. Tensor-flow platform is one of the most leading machine learning library used in developing deep learning models. However, Tensor-flow is not simple to use. On the contrary, keras is a high-level API built on tensor-flow that are more user friendly and simple to use.

3.6.1 Data set

In this chapter , we used AMPds dataset. This dataset contains one year of data that includes 11 measurements at one minute intervals for 21 sub-meters. AMPds also includes natural gas and water consumption data [31]. We convert energy consumption data to KWh to mimic the commonly available smart meter data.

3.6.2 Load Forecasting structure based on keras

We experimented several parameters to train our model with the keras deep learning packages. The following Fig 4. shows a set-up carried for residential short term load forecasting cases in keras.

Case1: Simple RNN forecasting model based on keras

```
model = Sequential()      Input (length {S})
layers = [1, 50, 100, 1]
model.add(SimpleRNN(layers[1],input_shape=(None,layers[0]),return sequences=True))
model.add(Dropout(0.2))
model.add(SimpleRNN(layers[2],return sequences=False))
model.add(Dropout(0.2))
model.add(Dense(layers[3]))
model.add(Activation("linear"))
```

Case2: GRU forecasting model based on keras

```
model = Sequential()      Input (length {S})
layers = [1, 50, 100, 1]
model.add(GRU(layers[1],input_shape=(None,layers[0]),return sequences=True))
model.add(Dropout(0.2))
model.add(GRU(layers[2],return sequences=False))
model.add(Dropout(0.2))
model.add(Dense(layers[3]))
model.add(Activation("linear"))
```

Case3: LSTM forecasting model based on keras

```
model = Sequential()      Input (length {S})
layers = [1, 50, 100, 1]
model.add(LSTM(layers[1],input_shape=(None,layers[0]),return sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(layers[2],return sequences=False))
model.add(Dropout(0.2))
model.add(Dense(layers[3]))
model.add(Activation("linear"))
```

Here, S= Number of sequence in recurrent neural network

Figure 3-4: Keras code structures for three different RNN scenario

3.6.3 Optimization technique

The challenging aspects of modeling deep neural network is the optimization of training criteria over millions of parameters. In this work, we choose RMS Prop optimization. RMS Prop is a biased estimator proposed in neural networks for

machine learning [17]. It is a gradient based optimization. RMS Prop has remained the method of choice for most recurrent neural network modeling. In this chapter, we utilize the RMS Prop as our parameters update optimizer, with parameter update rule according to the following formulae.

$$E[g^2]_{t-1} = \eta E[g^2]_{t-1} + (1-\eta) g_t^{\,2} \tag{18}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t \tag{19}$$

where, $t \varepsilon R$ is the iteration times. The θ is the parameters of neural network. E is the weighted sum operation. g^2 is vector of gradient square. The $\eta \epsilon (0,1)$ is the learning rate. The ε is a smoothing term that avoids division by zero.

3.7 Error Metrics for Evaluation

$$MAE = \frac{1}{T} \sum_{t=1}^{t=T} \frac{|A_t - F_t|}{A_t} \tag{20}$$

$$MAPE = \frac{1}{T} \sum_{t=1}^{t=T} \frac{|A_t - F_t|}{A_t} \times 100\% \tag{21}$$

where,

F_t = forecasted load, A_t = actual load and T= test set size.

From Table 1, LSTM based RNN with sequence 40 performed better compared to other forecasting models. The training and prediction time of LSTM model are also smaller compared with other models.

Table 5: MAPE summaries for residential load forecasts

Scenario	S	MAPE	Train time (s)	Prediction time (s)	Validation loss	Training loss
LSTM	30	35%	1116.7	0.0190	3.1197e-05	4.2183e-05
LSTM	40	24%	1505	0.0210	2.0085e-05	4.2721e-05
LSTM	50	29%	2145	0.0200	2.5080e-05	4.3218e-05
Simple RNN	30	59%	213	0.019049	2.6471e-05	1.1500e-04
Simple RNN	40	37.7%	294	0.019050	2.4830e-05	2.7659e-04
Simple RNN	50	45.7%	398.01	0.021054	2.6893e-05	1.7445e-04
GRU	30	34.3%	1095	0.0210	2.4373e-05	4.2037e-05
GRU	40	24.7%	1491.87	0.0210	2.1842e-05	3.9982e-05
GRU	50	39.7%	1993.13	0.0200	2.6099e-05	3.9602e-05

Table 6: MAPE summaries for conventional methods

Conventional time series model for residential load forecast	
ARIMA	74%
GLM	75%
RF	74%
SVM	50%
FFNN	73.54%

Table 2 show the MAPE values for conventional time-series methods. It is important to note that we were not able to perform the day-ahead forecasting computation with 2-year datasets for conventional methods, due to larger computation run-time. Instead, we used smaller 1-year dataset. For RNN, the computational run-time is less over the conventional methods. Some inferences are: Support vector machine (SVM) perform better than other conventional methods; RNNs are much better than the conventional methods.

In order to visualize how each method perform, the first 100 time steps for each forecasting model is plotted Figs.5-7.

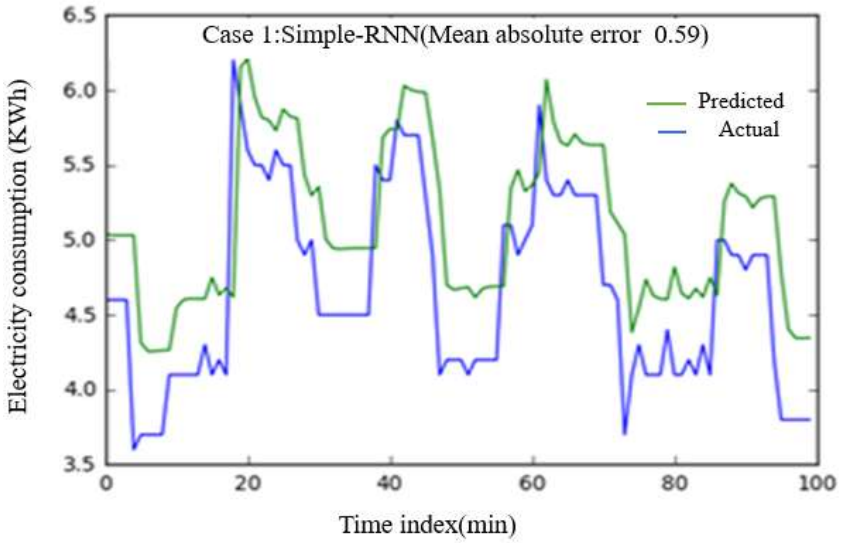

Figure 3-5: Residential load forecast using Simple RNN.

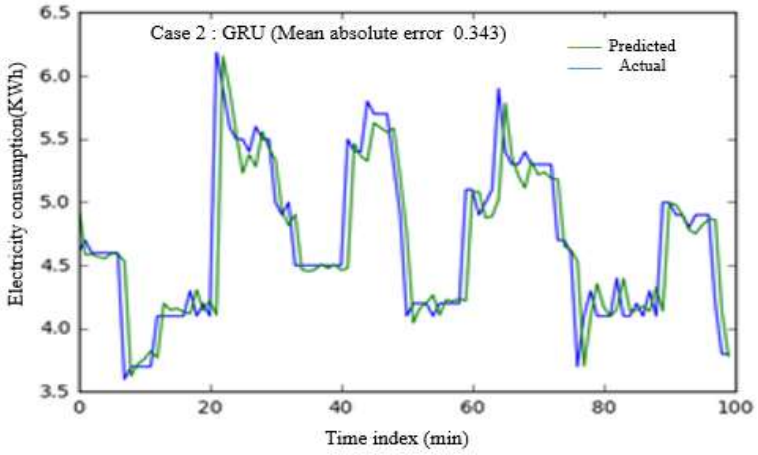

Figure 3-6: Residential load forecast using GRU.

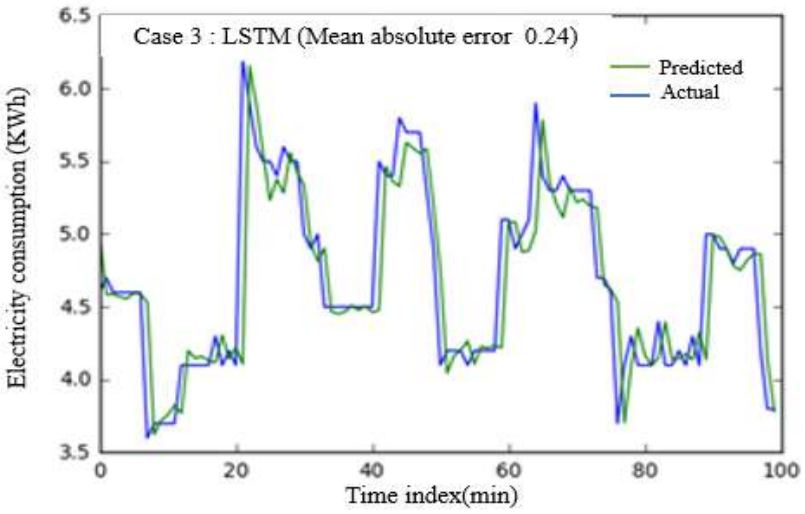

Figure 3-7: Residential load forecast using LSTM.

The performance different RNN models with varied sequences are summarized in Table 1.

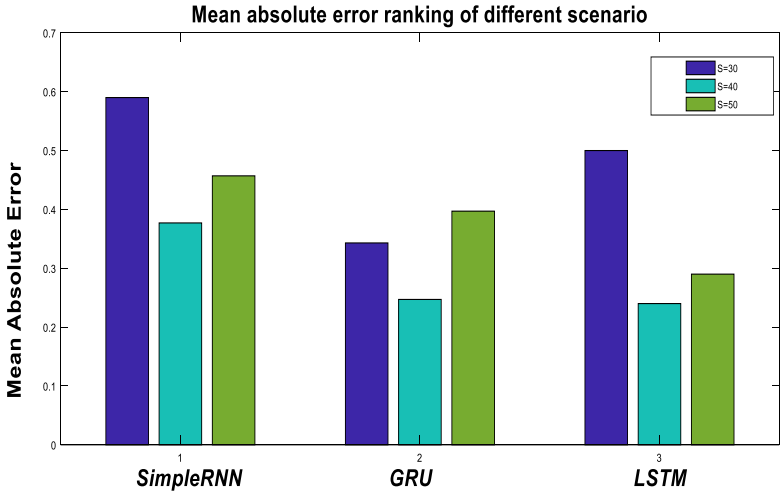

Figure 3-8: MAPE ranking for RNN's.

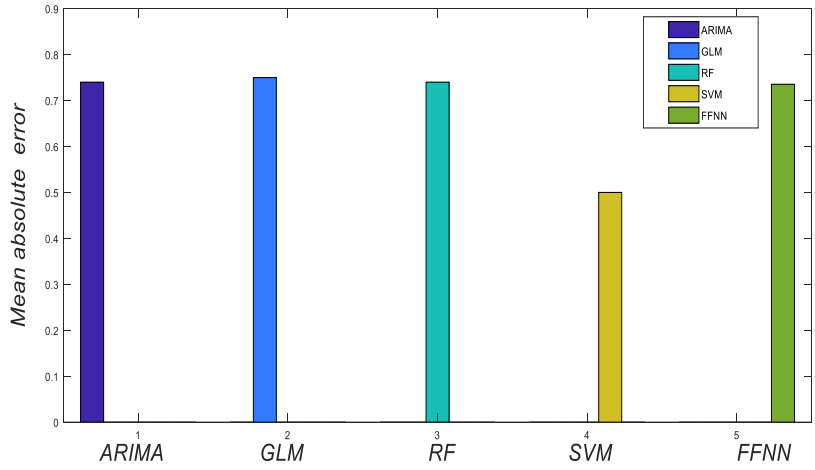

Figure 3-9: MAPE ranking for conventional methods.

Figures 8 and 9 show MAPE values for RNN and conventional methods. All frameworks are built on a desktop PC with a 3.4 GHz Intel i7 processor and 16GB of memory using the Keras library [32] with tensor-flow backend [8][33].

4 Optimal operation of smart home appliances using deep learning

4.1 Summary

This chapter discusses optimal operation and scheduling of smart home appliances using deep learning. One year of data sets were used that include price, demand, rating, and energy constraints. Using deep neural network platform, the Mean Absolute Percentage Errors (MAPE) were computed for 9 appliances using historical data sets. The preliminary results show promising improvement in forecasting accuracies coupled with Linear Programming suitable for demand response and scheduling loads.

4.2 Background and related work

The smart home system is envisioned to contain distributed renewable sources and smart appliances that able to participate in demand response. An important challenge faced by the power generation and distribution system is the sudden surge in energy demand during peak hours [34]. Power system companies are forced to install additional generating units, just to support the peak energy demand. The power transmission infrastructure also presents an additional bottleneck to support the ever-increasing growth in power demand. One way to overcome this surge in demand during peak hours is to encourage consumers to operate their equipment during off peak hours [35]. With advent of smart appliances, the consumers can take the advantage of time-of-use pricing scheme to reduce their electricity cost. This chapter presents a DNN based predictive model to forecast the next day energy consumption at an appliance level from a

one-year historical residential dataset. The forecasted consumption data along with the day-ahead energy price from MISO is used to model the smart home appliance management system. The concept of linear programming is used to do the optimization with the help of the AMPL (A Mathematical Programming Language) software.

4.3 DNN based Day-Ahead energy forecast

It is significant to forecast a particularly household daily consumption in order to design and size suitable renewable energy systems and battery storage. In this work, we did a Short-Term Load Forecasting (STLF) of household equipment. It is a challenge to forecast the household energy consumption in an appliance level because of its uncertainty [36]. Despite the uncertainty associated with household electric power consumption, we were able to forecast the energy consumption with a significant accuracy using DNN.

4.3.1 Designing Deep Neural Networks

Selecting an appropriate design of deep neural network is the first step of DNN-based forecasting system. In the current studies, the network architecture was built based on multilayer perceptron (MLP), full-connected, which is a feed-forward type of neural network, and the training task was performed through a backpropagation learning algorithm.

4.3.2 Backpropagation learning algorithms

The backpropagation learning algorithms, most common in use in feed-forward DNN, are based on steepest-descent methods that perform stochastic gradient descent on the error surface. Backpropagation is typically applied to multiple layers of neurons and works by calculating the overall error rate of an artificial neural network. The output layer is then analyzed to see the contribution of each of the neurons to that error. The neurons weights and threshold values are then adjusted, according to how much each neuron contributed to the error, to minimize the error in the next iteration.

4.3.3 DNN energy Forecasting Models

The neural network model of this work is developed using the Tensor Flow deep learning platform. Tensor Flow is an open source software library for numerical computation using data flow graphs. The nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicating between the nodes. The flexible architecture of Tensor Flow allows one to deploy computations to one or more Central or Graphical Processing Units (CPUs or GPUs) on a desktop, server, or mobile device with a single Application Process Interface (API). Tensor Flow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research. However, the

Tensor Flow system is general enough to be applicable to other domains as well [5]. The structure of load forecasting model of DNN is given below.

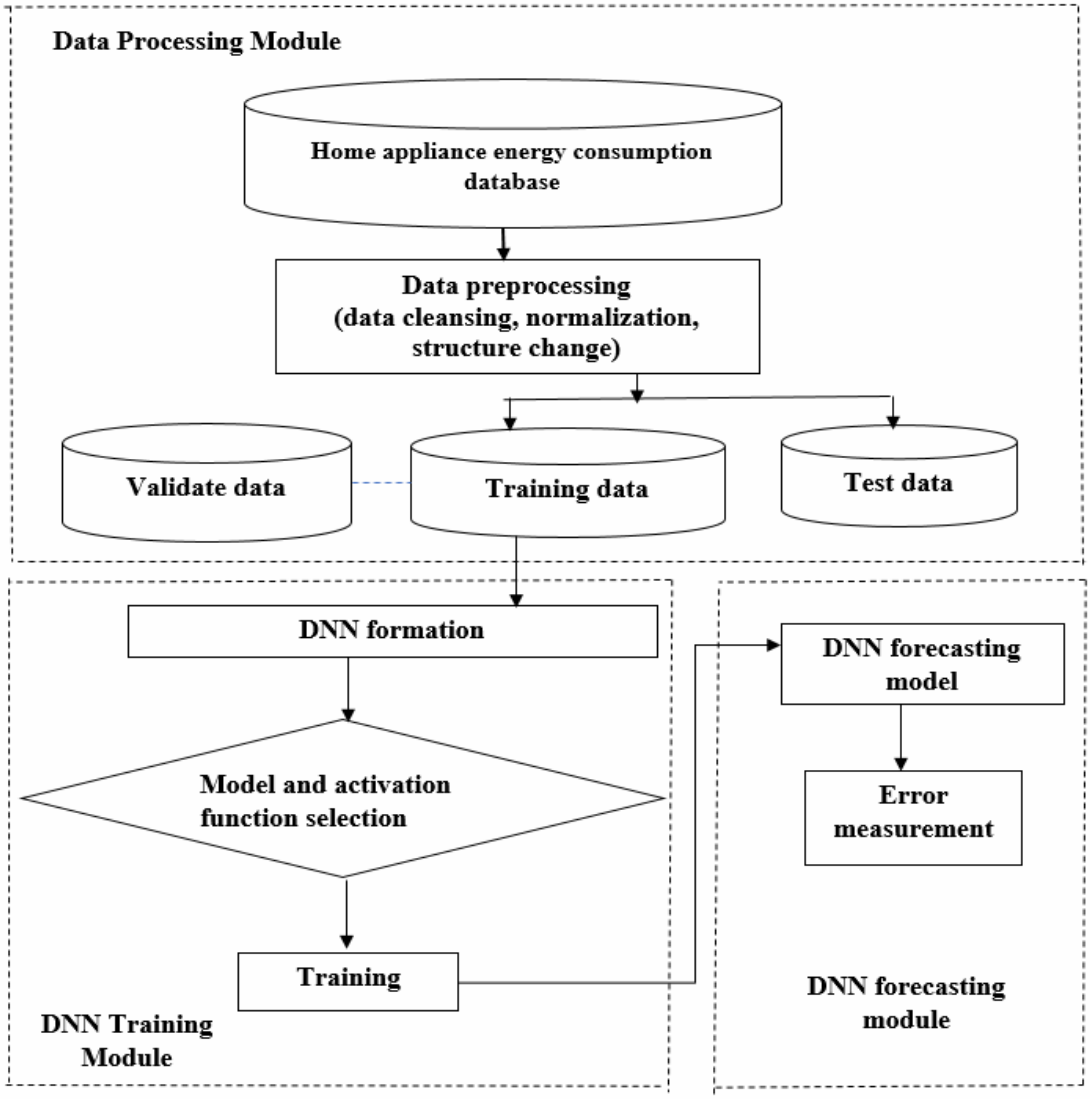

Figure 4-1: DNN energy Forecasting Models[37].

One-year smart home appliance data with one-minute resolution was used to predict the next day[38]. The previous load consumption of different appliance along with weather data was used to train our neural network model. In this work, an ADAM optimizer inside Tensor-flow framework is used to train our model. It

uses gradient descent algorithm. This method is faster in convergence than Stochastic Gradient Descent (SGD) approach [28]. The simulation settings and parameters used are listed in Table 1.

Table 7: DNN simulation settings and parameters

Parameters	**Values**
Total number of samples	236970
Training samples	235530
Validation samples	1440
Shape of training input data set	(235530, 11)
Shape of training target data set	(235530, 1)
Shape of validation input data set	(1440, 11)
Shape of validation target data set	(1440, 1)
epochs	100
Learning rate	0.001
mini_batch_size	100
Activation function	Linear
Number of hidden layer	3
Learning rate	0.001
Training epochs	500000

4.3.4 Dataset

The size of dataset impacts the accuracy, training, and transfer of learning within the deep neural network[37]. In this work, we used 1 year of 1-minute resolution data of UMass Smart Home Data Set. In this dataset contains data for 114 single-family apartments for the period 2014-2016. This data set includes a variety of

traces from three separate smart homes[38]. In our work, we used single smart home data of 2016 to predict next day different home appliance energy consumption.

4.3.5 Error Metrics for Evaluation:

To assess the accuracy of the forecasting model, Mean absolute percentage error (MAPE) is used.

$$MAPE = \frac{1}{T}\sum_{t=1}^{t=T}\frac{A_T - F_T}{A_T} \times 100\% \quad (22)$$

where,

$F_T =$ Forecasted load

$A_T =$ Actual load and

$T =$ Test set size

4.3.6 Forecasting Evaluation of different home appliance

The one day ahead predicted energy usage of different home appliance is given below (figure 2 to figure 10).

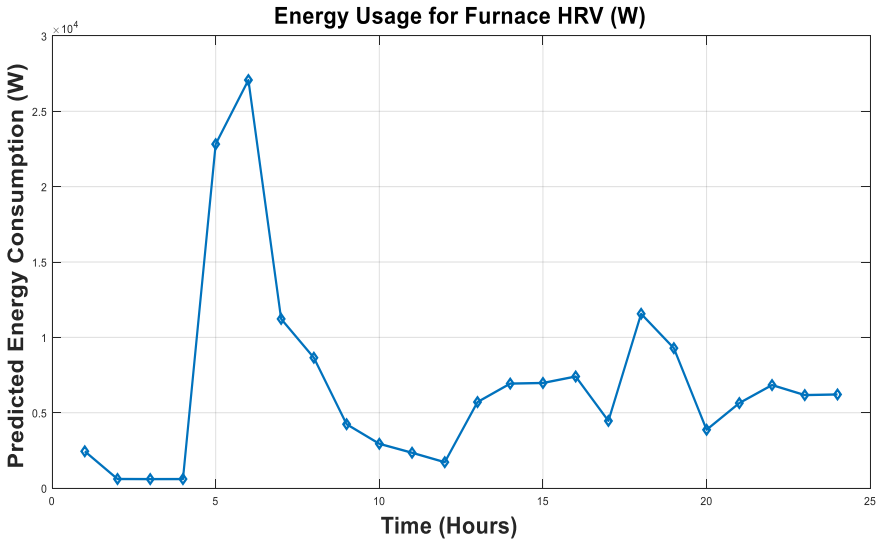

Figure 4-2: Predicted day ahead energy usage for Furnace HRV.

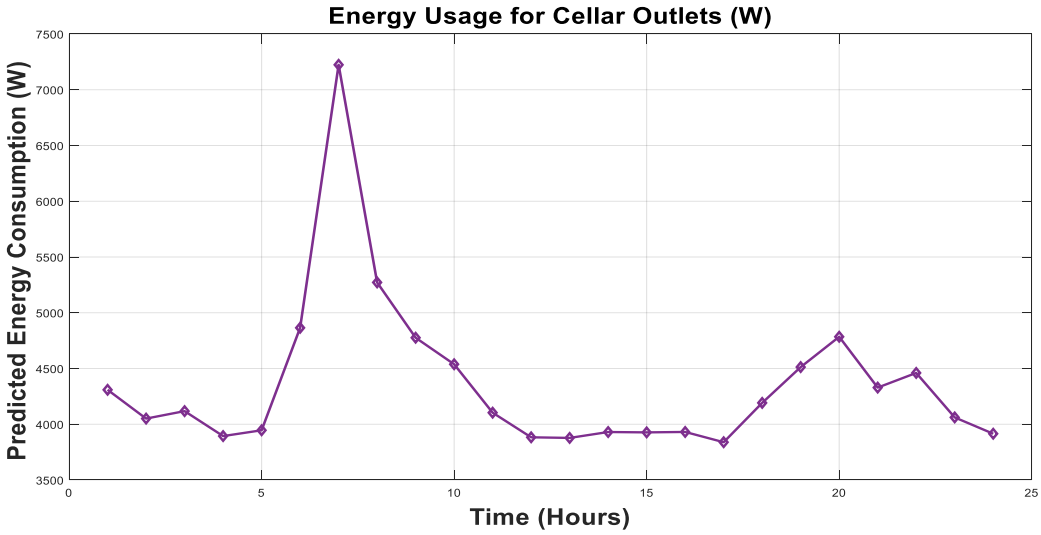

Figure 4-3: Predicted day ahead energy usage for Cellar Outlets.

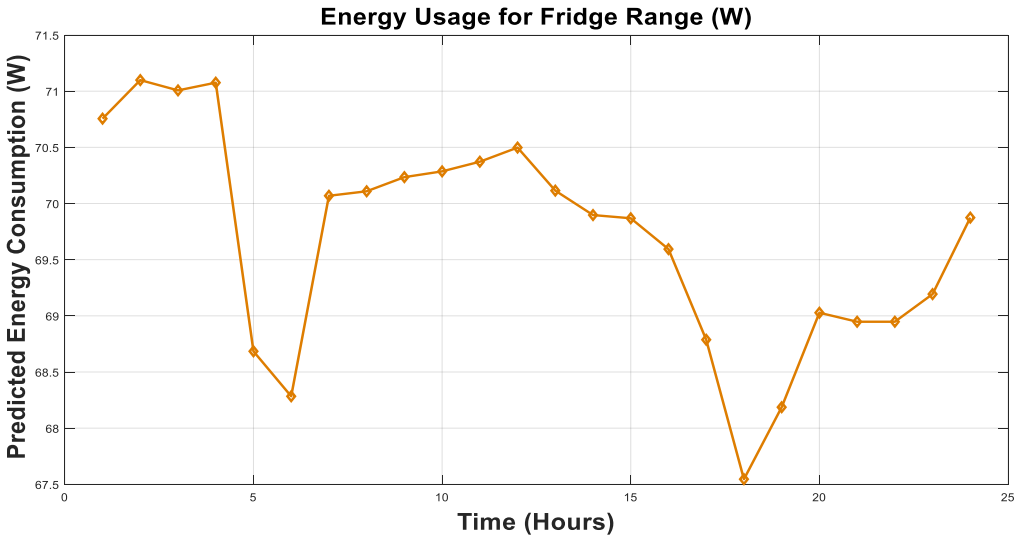

Figure 4-4: Predicted day ahead energy usage for Fridge Range.

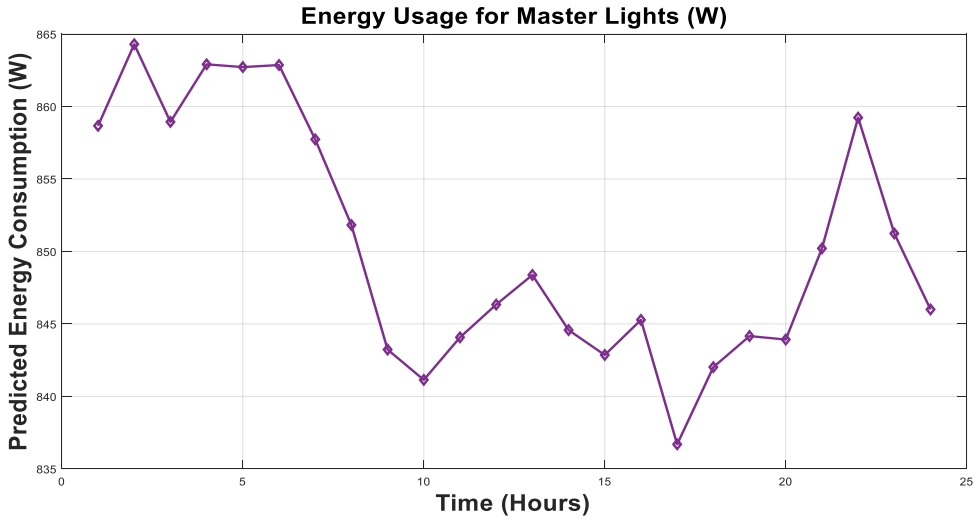

Figure 4-5: Predicted day ahead energy usage for Master Lights.

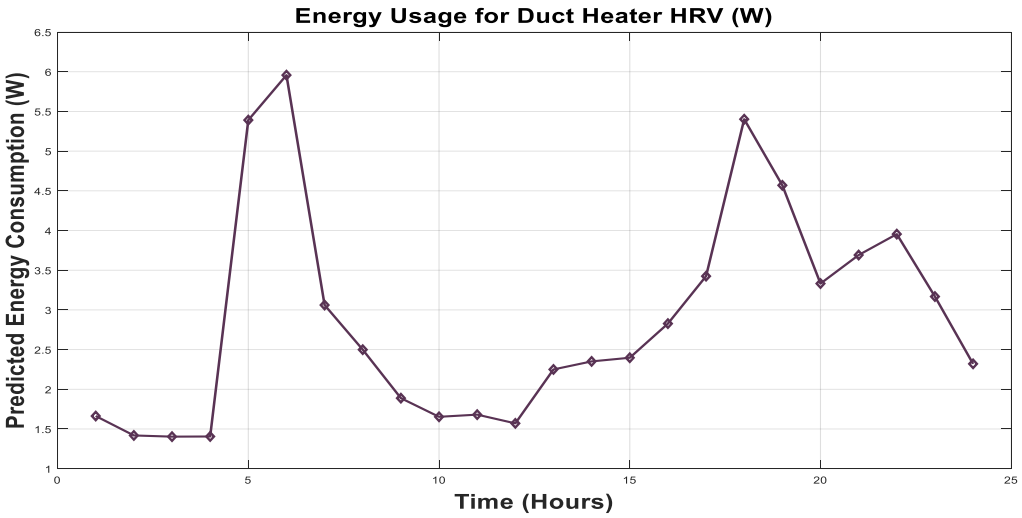

Figure 4-6: Predicted day ahead energy usage for Duct Heater HRV.

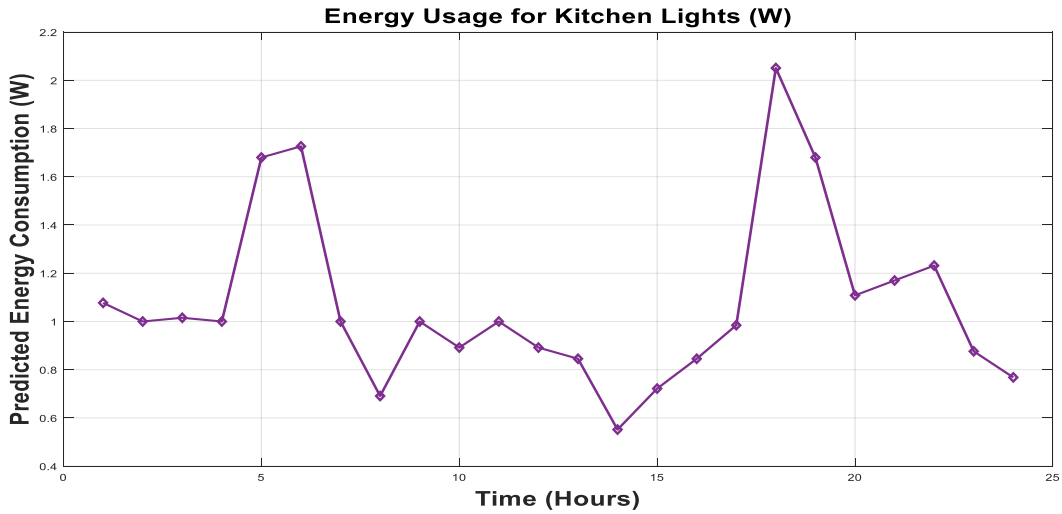

Figure 4-7: Predicted day ahead energy usage for Kitchen Lights.

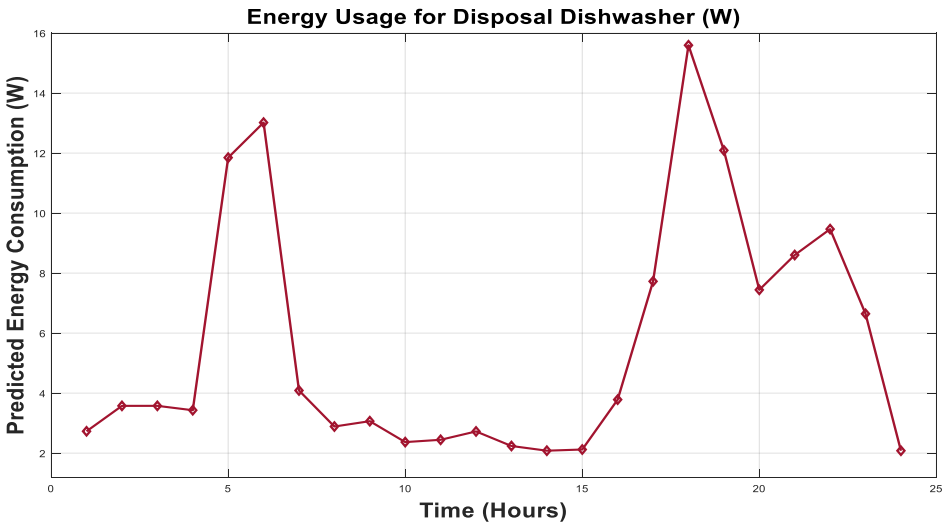

Figure 4-8: Predicted day ahead energy usage for Disposal Dishwasher.

Table 8: MAPE evaluation for smart home appliance

Sr. no	**Appliance**	**MAPE value (%)**
AP1	Furnace HRV	1.575
AP2	Cellar Outlets	3.23
AP3	Fridge Range	5.24
AP4	Master Lights	0.509
AP5	Bedroom Outlets	1.7359
AP6	Duct Heater HRV	10.8561
AP7	Bedroom Lights	1.7359
AP8	Kitchen Lights	2.23
AP9	Disposal Dishwasher	3.2

Table 2 and figure 11 shows the MAPE evaluation of different appliance. We got higher accuracy (MAPE of 0.509%) for masters' lights (AP4) and lower accuracy (MAPE of 10.8561%) for Duct Heater (AP6). It can be explained by the actual energy consumption behavior of this appliance. Figure 12 and 13 shows the actual

load consumption of master lights and Duct heater. It shows the Master lights have more continuous energy consumption pattern compared to Duct Heater.

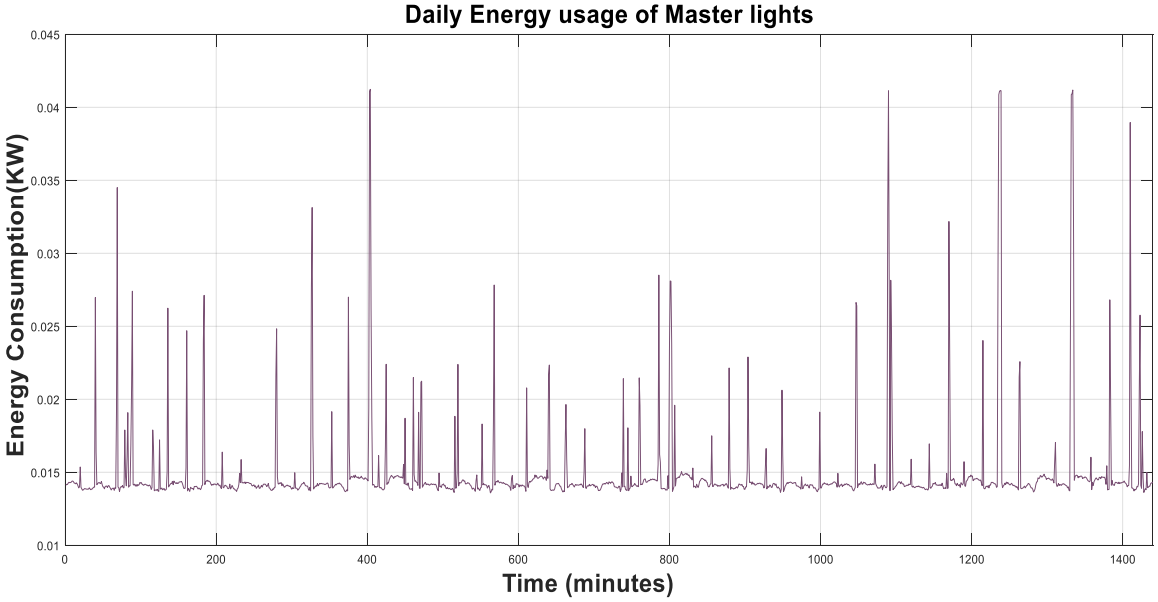

Figure 4-9: Daily energy usage for Disposal Dishwasher.

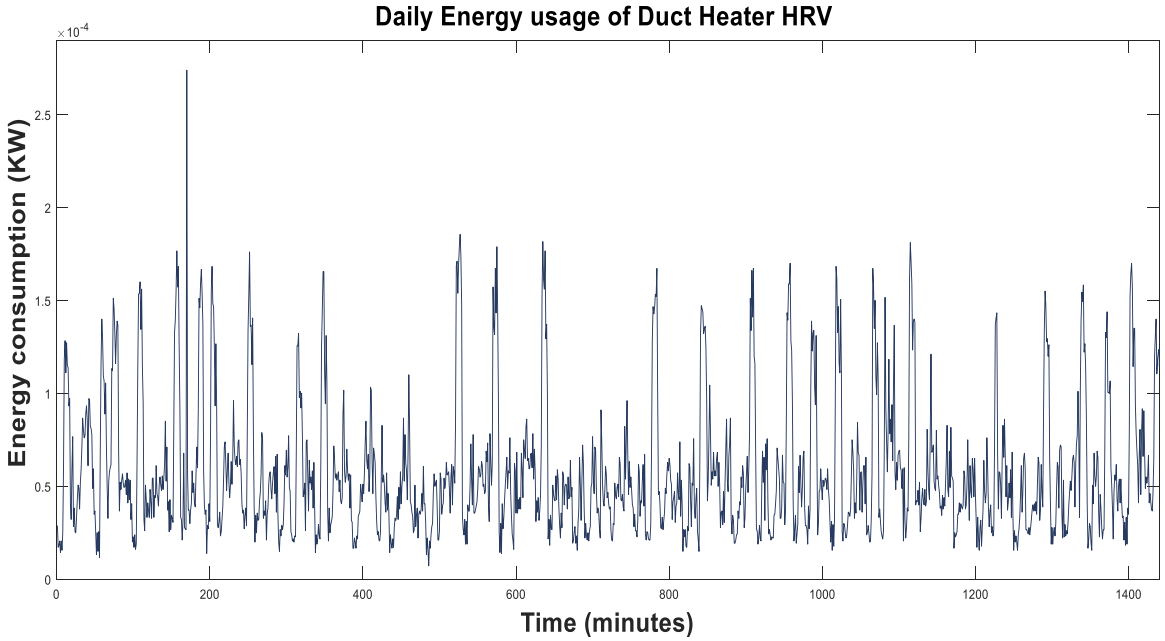

Figure 4-10: Daily energy usage for Duct Heater HRV.

4.4 Residential Appliance Scheduling

4.4.1 Smart Home Management System

The introduction of distributed energy sources has necessitated the need for improved metering and home management systems. The new generation residential appliances are becoming smarter and energy efficient to reduce the energy cost and improve the convenience for the customer.

4.4.2 System Modeling

The objective of the proposed optimization model is to reduce the total electricity cost for a residential customer. In this chapter cost is calculated on 24-hour period (e.g. USD per Wh). Here, C_j denote the day-ahead energy price in each time period. P_{ij} represents the energy used by appliance i in time period j

$$Min \sum_{i=1}^{M} \sum_{j=1}^{N} E_{ij} C_i \tag{23}$$

The optimization is performed subjected to several constraints [39]–[44]. They are listed below.

4.4.2.1 Energy Constraint

This constraint makes sure that the scheduling process allocates the required energy requirements for all the appliances in a residential home.

$$\sum_{i=1}^{M} E_{ij} = E_j \tag{24}$$

For each appliance j , the total allocated energy in all the time-period should meet the predicted electricity usage for each appliance.

4.4.2.2 Power safety constraint

The allotted energy in each time-period for each appliance should not cross a power limit. Residential appliance would not be able to use more the rated power of the appliance. If more power is allocated to the appliance, it won't be able to use more than its rated power and will result not meeting the energy requirements for a day.

$$E_{ij} \leq Rating[i] \tag{25}$$

Rating[i], stands for the rated power of each appliance in the scheduling problem. The scheduled power allocated to an appliance in each time period should be less than the maximum power for the appliance.

4.4.2.3 Production capacity constraint

The energy usage during peak hours creates a challenge for power companies to generate and transmit the required power. An important requirement of bringing forth the demand-response and time-of-use pricing scheme is to reduce the peak energy usage. This constraint prevents the allocation of all the appliance to the lowest price time slot. If all the house hold appliances are scheduled to the same time, it can create transmission constraints for the power companies.

$$\sum_{j=1}^{N} E_{ij} = K * \sum_{j=1}^{N} E_j \tag{26}$$

Here, K percentage of total demand that can be scheduled in each hour. In this work, the percentage was assumed to be 10%.

4.4.2.4 Consumer Preference

The user preference for the scheduling of the appliances is also considered in this optimization model. The consumer can specify the preferred time of use each individual appliance. The optimization model will not schedule appliances in other time-slots.

4.4.2.5 Equipment Flexibility

All the consumer residential appliances won't be committed to do scheduling based on energy price. Consumers prefer some appliances to be keep running without interruption for their comfort. The residential appliance committed to the scheduling and those which are supposed to be run at customers preferred time can be achieved in this model.

4.4.3 Optimization Tool

AMPL is an algebraic modeling language to solve large scale optimization and scheduling problems. It was developed at Bell Laboratories by Robert Fourer, David Gay, and Brian Kernighan. The software supports several open-source and commercial solvers. The AMPL coding syntax is similar to mathematical notation of optimization problems, which helps developers to develop their model [45], [46]. The day-ahead energy price for the residential customer is shown in Table 9. The day-ahead energy price data for Minnesota hub region (MISO-Midwest Independent Transmission System Operator), was used as the residential energy price [47].

Table 9: Residential day-ahead energy price ($/KWhr)

Hours	Energy Price	Hours	Energy Price
0	0.01976	12	0.03376
1	0.01828	13	0.02172
2	0.01595	14	0.02297
3	0.01545	15	0.02397
4	0.0151	16	0.02611
5	0.00778	17	0.03861
6	0.01488	18	0.03391
7	0.00725	19	0.02094
8	0.01823	20	0.01891
9	0.01655	21	0.01643
10	0.0234	22	0.01885
11	0.02247	23	0.0165

Table 10 lists the equipment rating, flexibility, and preferred time of appliance usage.

Table 10: Residential appliance rating and user preference

Equipment	Rating	Flexibility	Preference
Furnace	12000	0	1AM:11AM
Cellar Outlets	8000	1	12AM-12PM
Fridge Range	1200	0	2AM-11PM
Master Lights	2000	1	12AM-8AM 4PM-11PM
Bedroom Outlets	2500	1	12AM-8AM 4PM-11PM
Bedroom Lights	2000	1	12AM-8AM 4PM-11PM

Duct Heater	3000	1	3AM-8AM 10AM-4PM 6PM-8PM 10PM-11PM
Kitchen Lights	10	0	12AM-12PM
Dishwasher	45	1	12AM-5AM

The scheduling model developed in this work allows the consumer the flexibility to decide which appliances to commit for scheduling. If the flexibility parameter in the table is set to 0, the appliance will not be committed for scheduling and will be run at the costumer preferred timing without any interruption. The optimized appliance scheduling for the household appliances based on the consumer preferred timing and flexibility is shown in figure 11.

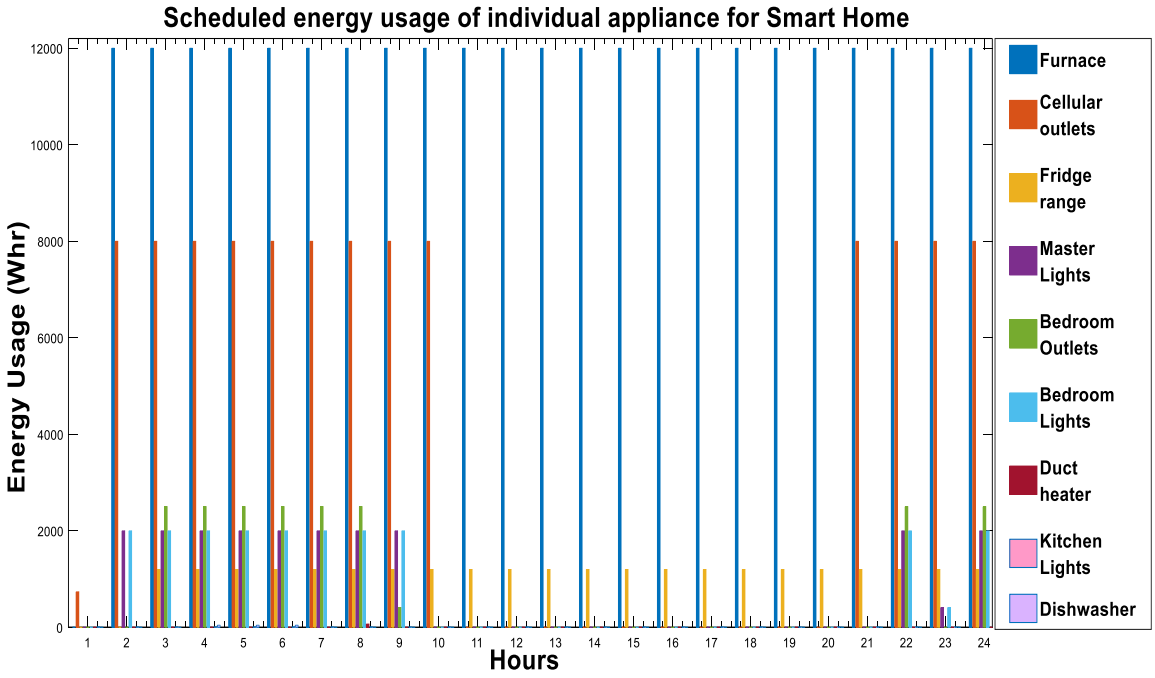

Figure 4-11: Scheduled energy usage of individual appliance for the residential customer.

5 Residential load forecasting based on deep neural network(DNN) and k-shape clustering

5.1 Overview

One of the most important tasks for utility companies is load forecasting in order to plan future demand for generation capacity and infrastructure. Improving load forecasting accuracy over a short period is a challenging open problem due to the variety of factors that influence the load. This chapter proposes a new approach for short term load forecasting using an effective new combination of clustering and deep learning methods. Our evaluation using 200 residential home demand data from a publicly available real-life dataset.

This work uses a two steps process:

1) Apply the time-series clustering (K-shape) for 200 residential home

2) Apply deep neural network method for short term load forecasting

5.2 Background and related work

Smart meter are the key components of smart grid technology. Smart meter provides fine-grained electric power consumption information at different sampling intervals (10,30,60 minutes). One of the most promising applications for such large volumes of data from smart meters is to improve the accuracy of electrical load forecasting. One of methods of improving load forecasts using the data generated from smart meters of individual customers is based on the use of clustering [48]. In this method, the knowledge about load consumption behavior of customers is used to improve the accuracy of forecasting. Instead of developing a single

forecasting model for the aggregated load consumption of all customers, clustering can be used to divide the customers into sub-class with similar demand profiles. Then, a forecasting model can be provided for each cluster of customers according to their load consumption profile. Thus, a more accurate model can be provided for each cluster and then the load forecast for all consumers can be obtained by combining the forecasts of these models. Applying clustering as an initial step in electric load forecasting has been the focus of several studies [49] [50]. The K-means clustering method has been widely used in previous works for this purpose [51] [52]. However, little attention has been paid in previous studies to the choice of clustering method with the aim of improving the aggregate level of forecasting accuracy. In this chapter we show that the accuracy of load forecasting can be improved by clustering method on demand profile.

5.3 Methodology for clustering based DNN forecast

Deep learning techniques such as Recurrent neural network (RNN) has higher performance in solving STLF by treating STLF as a time-series forecasting problem [53], [54]. However, in this study, a novel method of combining DNN and clustering techniques for forecasting loads on an electricity big data is proposed. There are two phases in the procedure of creating the proposed method. In phase 1, raw data is preprocessed by removing noises and numerical processing. And then related factor analysis on the clean data is performed for feature extraction and selection. Millions of load samples consist of the chosen features and target electricity loads to form a big data set. On the data set, we utilized the clustering technique to partition the set into subsets. Furthermore, these subsets are divided

into training sets and testing sets. On the training sets, the method is constructed based on DNN models.

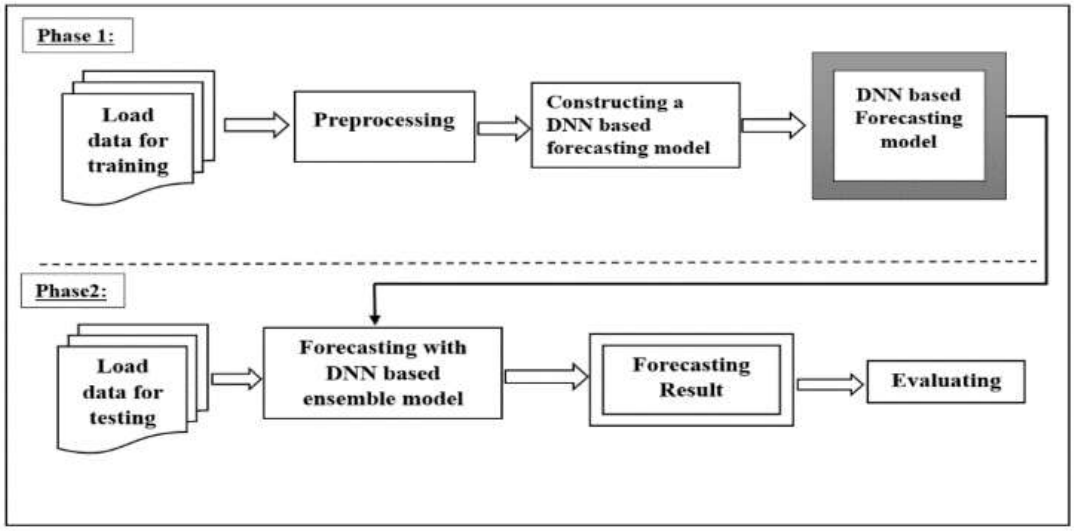

Figure 5-1: Framework of constructing the model.

In phase 2, the method is adopted to forecast loads for 10 minutes interval on these testing sets. The framework of the presented method is shown in Figure 1. K-Means algorithm is utilized to divide the dataset into small clusters. Then, these clusters are segmented into training with the DNN Based Model. In 2nd phase we use the method to forecast loads on each testing subset so as to conduct results on each testing subset. Prediction loads are generated by these prediction results by these DNN models, which is shown in figure.

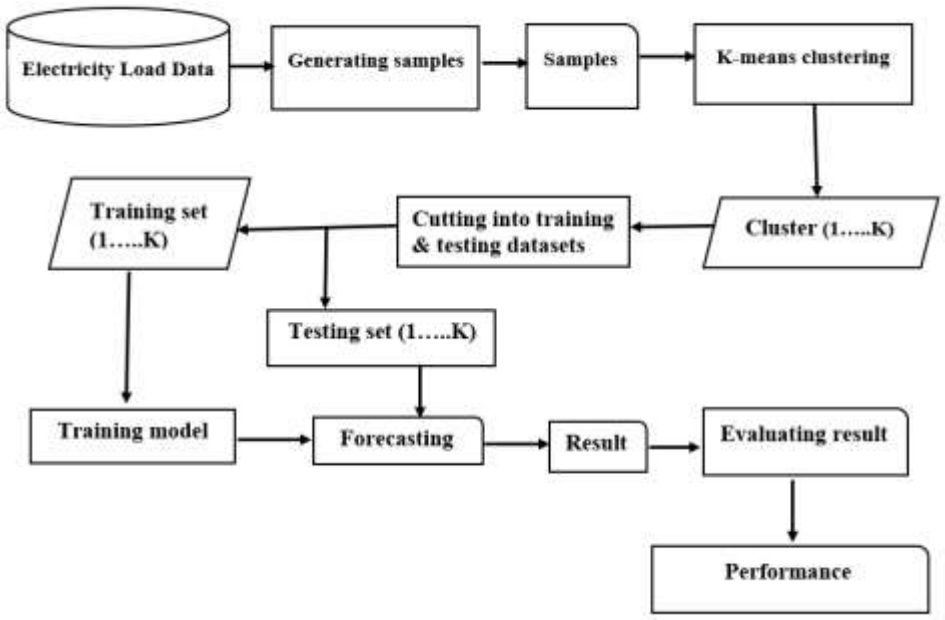

Figure 5-2: Flows of constructing the DNN based method with K-Means algorithm.

5.4 Methods of network level clustering

The smart meter records consist of a set of M consumers $x_1, x_2, ..., x_M$ .The consumption history of consumer i can be define $x_i = \{x_i^1, x_i^2, ...x_i^T\} \forall \in \{1, 2, ..., M\}$ where,

$M =$ the total number of consumers

$T =$ the total number of historical time periods

In our work we have used three approaches of network level clustering-

a) Completely aggregated method
b) Completely disaggregated method
c) Clustering based forecasting method

The brief description of this three networks level clustering is given below-

5.4.1 Completely aggregated method

In completely aggregated method [3], the load consumption of all consumers that belong to the network into a vector, $X_{aggr} = (X^1_{aggr}, X^2_{aggr}, ..., X^T_{aggr})$ and take this as the input feature vector for forecasting.

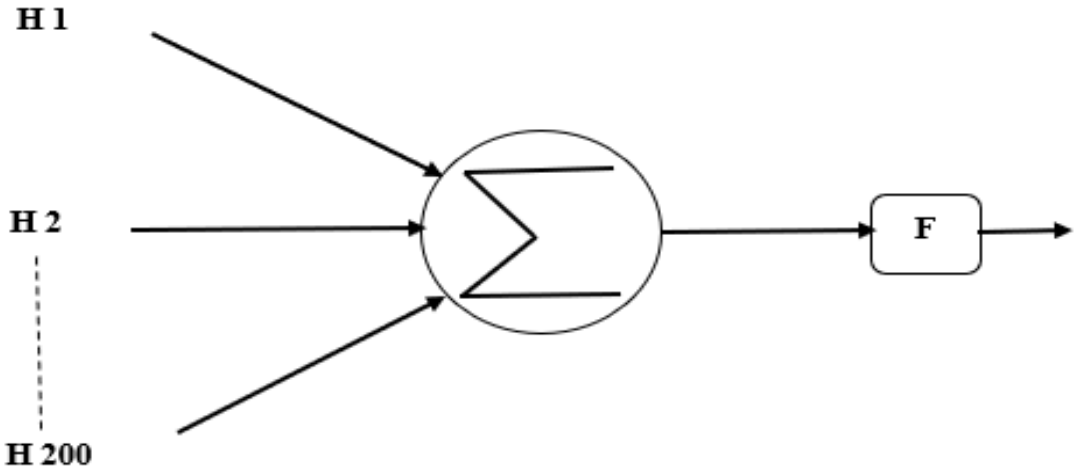

Figure 5-3: Completely aggregated method**.**

5.4.2 Completely disaggregated method

In completely disaggregated method load forecasting applies to individual consumers and then adds individual consumers prediction to prediction the aggregated level.

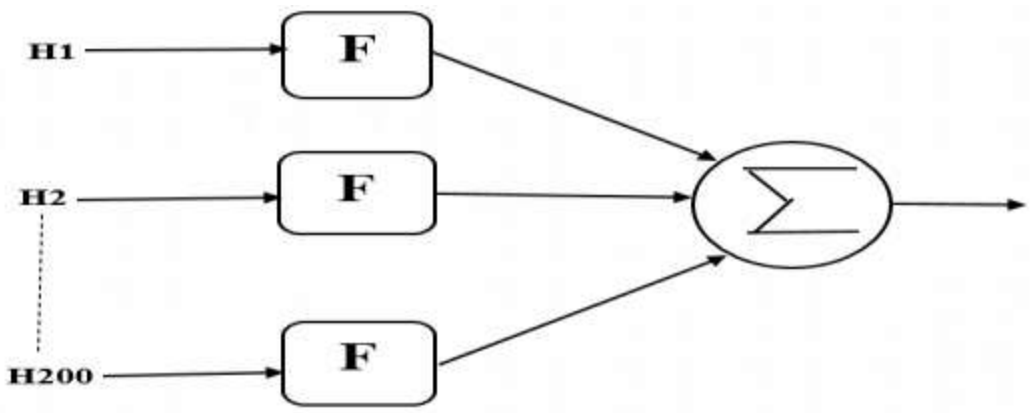

Figure 5-4: Completely disaggregated method.

5.4.3 Clustering based forecasting method

In clustering-based forecasting method, consumer household demand are grouped into several clusters based on actual demand profile of the consumers. We consider that by applying a clustering algorithm, K clusters $C = \{c_1, c_2, ..., c_k\}$ are obtained, so that each cluster of households in C can then be used to train a neural network. Therefore, K prediction models $F_c = \{F_{c1}, F_{c2}, ...F_{ck}\}$ are generated from the k groups of consumers. To calculate the final prediction for period of interest, we take the sum over the predictions from the clusters as follows:

$$F_{aggr} = \sum_{i=1}^{i=k} F_{ci} \tag{27}$$

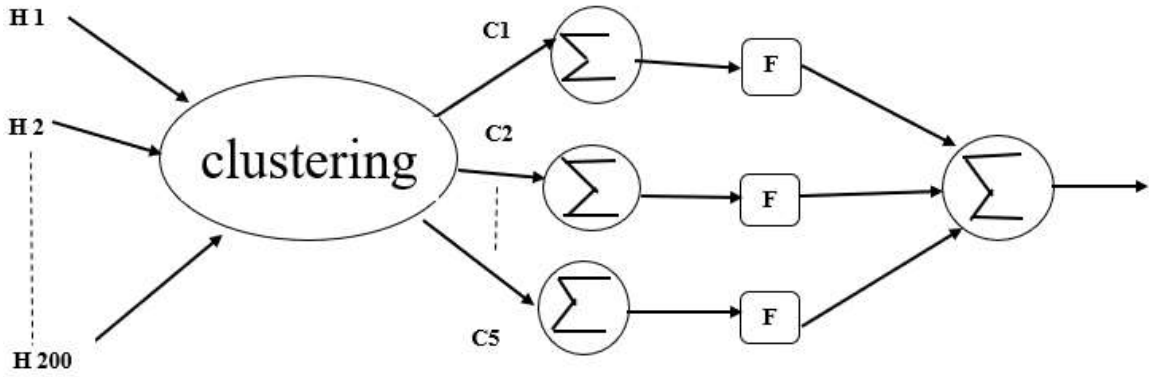

Figure 5-5: Clustering based forecasting method.

5.5 Designing Deep Neural Networks

Selecting an appropriate design is the first step of DNN-based forecasting system. In our work we utilized multilayer perception model and different recurrent neural such as simple RNN, long short term memory (LSTM) and gated recurrent unit (GRU) for designing forecasting model.

Table 11: Deep learning tensor-flow parameter

Parameters	Values
Total number of samples	52560
Training samples	52416
Validation samples	1440
Epochs	100
Learning rate	0.001
Mini batch size	100
Activation function	Linear

5.6 Evaluation Metrics:

Mean absolute percentage error (MAPE), root-mean-square error (RMSE), normalized mean absolute error (NMAE), and normalized root-mean-square error (NRMSE) are the commonly used evaluation metrics to measure performance of models. We choose MAPE to measure the accuracy of our model. The MAPE is a measure of prediction accuracy in statistics. It defines accuracy as a percentage, and is defined by the following equation.

$$MAPE = \frac{1}{T}\sum_{t=1}^{t=T}\frac{A_T - F_T}{A_T} \times 100\% \tag{28}$$

5.7 MAPE evaluation

MAPE value of different individual house and for clusters is presented. We get better MAPE value with clustering then without clustering based forecasting. The MAPE evaluation for different scenario is given below-

Table 12: MAPE evaluations of residential home dataset using RNN

Cluster1 (28 household)	0.352 (LSTM)
	0.375(GRU)
	0.320(SimpleRNN)
Cluster2 (14 household)	0.133(LSTM)
	0.118(GRU)
	0.153(SimpleRNN)
Cluster3 (80 household)	1.086(LSTM)
	1.168(GRU)
	2.409(SimpleRNN)
Cluster4 (41 household)	0.756(LSTM)
	0.463(GRU)
	0.479(SimpleRNN)
Cluster5	0.007(LSTM)

(37 household)	0.418(GRU)
	0.458(SimpleRNN)
Whole 200 household	8.544(SimpleRNN)
	7.305(LSTM)
	4.096(GRU)

6 Conclusion and future work

6.1 Conclusion

This work investigated the performance of different DNNs to forecast short term electricity. We used different dataset such as Iberian electricity markets data, AMPds smart meter data, NREL residential home dataset to evaluate the accuracies of DNN. It is observed that deep neural networks are powerful forecasts that can provide better individual forecasts over conventional forecasting models.

Here, the state-of-the-art tool named TensorFlow and keras based deep learning library utilized to develop the forecasting models. The key findings of this research are as follows:

1. The first part of research investigated the application of DNN's in electric power system analysis. This part uses a 90-day Iberian market dataset to predict the day-ahead loads. Multiple combinations of activation functions were trained, and tested on single and double-layer neural networks. The MAPE results indicate that the combination of ELU with ELU perform better than other combinations. On weekend data sets, the ReLU-ReLU combination outperform other combinations.
2. The second part of this research investigated the performance of recurrent neural network in load forecasting for a single residential customer. Load forecasting for a single residential customer at a 1-minute interval using recurrent neural networks for smart-metering data sets are investigated using DNN and conventional methods.

3. The third part of this research investigated a smart home management to schedule the appliances. A deep learning based (DNN) forecasting model was used to predict the consumer pattern and a linear programming based optimization model was formulated to develop a day-ahead scheduling scheme based on price, demand, rating, and energy constraints.

6.2 Future work

1. Convolutional neural networks (CNN) can be utilized for the short-term load forecasting problem.

2. Further exploration is required to determine the reliability of DNN based load forecasting. The following questions are crucials for effectiveness on the applicability of DNN's:

   a) How large should a DNN be and which configurations should be considered?

   b) What is the right sample size and data intervals are required for sufficient training for short term forecasts?

   c) How can a deep neural network be adopted for a real-time market operation.

One possible solution is using hybrid combination of DNN and fuzzy systems.

REFERENCES

[1] G. D. Garson, “A Comparison of Neural Network and Expert Systems Algorithms with Common Multivariate Procedures for Analysis of Social Science Data,” *Soc. Sci. Comput. Rev.*, vol. 9, no. 3, pp. 399–434, Oct. 1991.

[2] “Prediction Algorithms in One Picture - Data Science Central.” [Online]. Available: https://www.datasciencecentral.com/profiles/blogs/prediction-algorithms-in-one-picture. [Accessed: 18-Mar-2018].

[3] A. K. Srivastava, A. S. Pandey, and D. Singh, “Short-Term Load Forecasting Methods : A Review,” 2016.

[4] “No Title.” [Online]. Available: http://www.datasciencecentral.com/profiles/blogs/prediction-algorithms-in-one-picture.

[5] M. Q. Raza and A. Khosravi, “A review on artificial intelligence based load demand forecasting techniques for smart grid and buildings,” *Renew. Sustain. Energy Rev.*, vol. 50, pp. 1352–1372, 2015.

[6] S. Hosein and P. Hosein, “Load Forecasting using Deep Neural Networks.”

[7] L. Hernández, C. Baladrón, J. M. Aguiar, and B. Carro, “A Multi-Agent-System Architecture for Smart Grid Management and Forecasting of Energy Demand in Virtual Power Plants,” vol. 51, pp. 106–113, 2013.

[8] “TensorFlow.” [Online]. Available: https://www.tensorflow.org/. [Accessed: 18-Mar-2018].

[9] H. Shayeghi, H. A. Shayanfar, and G. Azimi, “Intelligent Neural Network Based STLF,” 2009.

[10] W. Charytoniuk and M. S. Chen, “Neural Network Design for Short-Term Load Forecasting,” no. 817, pp. 554–561.

[11] H. Shi, M. Xu, and R. Li, “Deep Learning for Household Load Forecasting – A Novel Pooling Deep RNN,” *IEEE Trans. Smart Grid*, vol. 3053, no. c, pp. 1–1, 2017.

[12] D. Soekhoe, P. Van Der Putten, and A. Plaat, “On the Impact of data set Size in Transfer Learning using Deep Neural Networks.”

[13] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[14] “ReLu compared against Sigmoid, Softmax, Tanh - my notebook - Quora.” [Online]. Available: https://algorithmsdatascience.quora.com/ReLu-compared-against-Sigmoid-Softmax-Tanh. [Accessed: 18-Mar-2018].

[15] “CS231n Convolutional Neural Networks for Visual Recognition.” [Online].

Available: http://cs231n.github.io/neural-networks-1/. [Accessed: 18-Mar-2018].

[16] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),” Nov. 2015.

[17] D. P. Kingma and J. L. Ba, “A : a m s o,” pp. 1–15, 2015.

[18] M. Ghofrani *et al.*, “Smart Meter Based Short-Term Load Forecasting for Residential Customers,” pp. 13–17, 2019.

[19] I. H. L. Curves, “Clustering-Based Improvement of Nonparametric Functional Time Series Forecasting : Application to,” vol. 5, no. 1, pp. 411–419, 2014.

[20] S. Arora and J. W. Taylor, “Forecasting electricity smart meter data using conditional kernel density estimation,” *Omega*, vol. 59, pp. 47–59, 2016.

[21] B. Dong, Z. Li, S. M. M. Rahman, and R. Vega, “A hybrid model approach for forecasting future residential electricity consumption,” *Energy Build.*, vol. 117, pp. 341–351, 2016.

[22] L. G. Swan and V. I. Ugursal, “Modeling of end-use energy consumption in the residential sector : A review of modeling techniques,” vol. 13, pp. 1819–1835, 2009.

[23] M. Chow, “Application of Fuzzy Logic Technology for Spatial Load Forecasting MO-yuen Chow Hahn Tram,” 1996.

[24] U. Nonfully, C. Artificial, and N. Network, “L!aYuA,” vol. 7, no. 3, pp. 1098–1105, 1992.

[25] V. S. KODOGIANNIS, M. AMINA, and I. PETROUNIAS, “A CLUSTERING-BASED FUZZY WAVELET NEURAL NETWORK MODEL FOR SHORT-TERM LOAD FORECASTING,” *Int. J. Neural Syst.*, vol. 23, no. 5, p. 1350024, Oct. 2013.

[26] “Understanding LSTM Networks -- colah’s blog.” [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 18-Mar-2018].

[27] B. Zhang, J.-L. Wu, and P.-C. Chang, “A multiple time series-based recurrent neural network for short-term load forecasting,” *Soft Comput.*, 2017.

[28] S. Hochreiter and P. Frasconi, “Gradient Flow in Recurrent Nets : the Di culty of Learning Long-Term Dependencies 1 Introduction 2 Exponential error decay Gradients of the error function.”

[29] R. Jozefowicz and I. G. Com, “An Empirical Exploration of Recurrent Network Architectures,” vol. 37, 2015.

[30] C. Trabelsi *et al.*, “DEEP COMPLEX NETWORKS.”

[31] S. Makonin, F. Popowich, L. Bartram, B. Gill, and I. V Baji, “AMPds : A Public Dataset for Load Disaggregation and Eco-Feedback Research,” no. Section III, 2013.

[32] “Keras Documentation.” [Online]. Available: https://keras.io/. [Accessed: 18-Mar-2018].

[33] “Keras as a simplified interface to TensorFlow: tutorial.” [Online]. Available: https://blog.keras.io/keras-as-a-simplified-interface-to-tensorflow-tutorial.html. [Accessed: 18-Mar-2018].

[34] C. F. S. A, J. F. Franco, M. J. Rider, and R. Romero, “Optimal Charging Coordination of Electric Vehicles in Unbalanced Electrical Distribution System Considering Vehicle-to-Grid Technology.”

[35] N. G. Paterakis, S. Member, O. Erdinc, and A. G. Bakirtzis, “Response Strategies Optimal Household Appliances Scheduling under Day-Ahead Pricing and Load-Shaping Demand Response Strategies,” no. December, 2015.

[36] A. S. Nair, P. Ranganathan, H. Salehfar, and N. Kaabouch, “Uncertainty Quantification of Wind Penetration and Integration into Smart Grid : A Survey.”

[37] T. Hossen, S. J. Plathottam, R. K. Angamuthu, P. Ranganathan, and H. Salehfar, “Short-term load forecasting using deep neural networks (DNN),” *2017 North Am. Power Symp.*, pp. 1–6, 2017.

[38] “Smart - UMass Trace Repository.” [Online]. Available: http://traces.cs.umass.edu/index.php/Smart/Smart. [Accessed: 25-Mar-2018].

[39] K. C. Sou, J. Weimer, H. Sandberg, and K. H. Johansson, “Scheduling Smart Home Appliances Using Mixed Integer Linear Programming,” pp. 5144–5149, 2011.

[40] A. Dubey, S. Santoso, M. P. Cloud, and M. Waclawiak, “Determining Time-of-Use Schedules for Electric Vehicle Loads: A Practical Perspective,” *IEEE Power Energy Technol. Syst. J.*, vol. 2, no. 1, pp. 12–20, 2015.

[41] N. Korolko and Z. Sahinoglu, “Robust optimization of EV charging schedules in unregulated electricity markets,” *IEEE Trans. Smart Grid*, vol. 8, no. 1, pp. 149–157, 2017.

[42] Z. Wu, S. Zhou, J. Li, and X. P. Zhang, “Real-time scheduling of residential appliances via conditional risk-at-value,” *IEEE Trans. Smart Grid*, vol. 5, no. 3, pp. 1282–1291, 2014.

[43] J. Ning, Y. Tang, and W. Gao, “A hierarchical charging strategy for electric vehicles considering the users’ habits and intentions,” *IEEE Power Energy Soc. Gen. Meet.*, vol. 2015–Septe, 2015.

[44] M. L. Crow, “Multi-Objective Electric Vehicle Scheduling Considering Customer and System Objectives.”

[45] R. Fourer, D. M. Gay, M. Hill, B. W. Kernighan, and T. B. Laboratories, “AMPL : A Mathematical Programming Language,” *Manage. Sci.*, vol. 36, pp. 519–554, 1990.

[46] D. M. Gay, “The AMPL modeling language: An aid to formulating and solving optimization problems,” in *Springer Proceedings in Mathematics and Statistics*, 2015, vol. 134, pp. 95–116.

[47] “MISO Day-Ahead price data.” .

[48] S. Haben, C. Singleton, and P. Grindrod, “Analysis and Clustering of Residential Customers Energy Behavioral Demand Using Smart Meter Data,” *IEEE Trans. Smart Grid*, vol. 7, no. 1, pp. 136–144, Jan. 2016.

[49] M. Misiti, Y. Misiti, G. Oppenheim, and J.-M. Poggi, “OPTIMIZED CLUSTERS FOR DISAGGREGATED ELECTRICITY LOAD FORECASTING,” *REVSTAT – Stat. J.*, vol. 8, no. 2, pp. 105–124, 2010.

[50] P. Goncalves, D. Silva, D. Ili, and S. Karnouskos, “The Impact of Smart Grid Prosumer Grouping on Forecasting Accuracy and its Benefits for Local Electricity Market Trading.”

[51] A. Shahzadeh, A. Khosravi, and S. Nahavandi, “Improving load forecast accuracy by clustering consumers using smart meter data,” in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–7.

[52] F. L. Quilumba, W.-J. Lee, H. Huang, D. Y. Wang, and R. L. Szabados, “Using Smart Meter Data to Improve the Accuracy of Intraday Load Forecasting Considering Customer Behavior Similarities,” *IEEE Trans. Smart Grid*, vol. 6, no. 2, pp. 911–918, Mar. 2015.

[53] J. Geng, M. L. Huang, M. W. Li, and W. C. Hong, “Hybridization of seasonal chaotic cloud simulated annealing algorithm in a SVR-based load forecasting model,” *Neurocomputing*, vol. 151, no. P3, pp. 1362–1373, 2015.

[54] J. C. B. Gamboa, “Deep Learning for Time-Series Analysis,” *arXiv*, 2017.

APPENDIX A

FORECASTING OF ELECTRICAL DEMAND

#CODE FOR IMPLEMENTION OF MULTI LAYER PERCEPTION DEEP NEURAL NETWORK USING TENSORFLOW:

```
import numpy as np
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
data_dir ="C:/Users/tareq.hossen/Desktop/load_predict/data/"
model_dir ="C:/Users/tareq.hossen/Desktop/load_predict/model/"
summaries_dir                                                    =
"C:/Users/tareq.hossen/Desktop/load_predict/summaries"


#Extract data from CSV
df1=pd.read_csv(data_dir+"PJM LOAD.csv")
col1 = df1[['P.DD-1','P.DD-6','Temp','irradiance','windspeed']]
col2 = df1[['Actual']]
#Convert to Numpy array
InputX1 = col1.as_matrix()
InputY1 = col2.as_matrix()
InputX1.astype(float, copy=False);
InputY1.astype(float, copy=False);
print("Input:",InputX1)
print("Output:",InputY1)
#print("Samples:",InputX1.shape[0])

#Min-max Normalization
X1_min = np.amin(InputX1,0)
X1_max = np.amax(InputX1,0)
print("Mininum values:",X1_min)
print("Maximum values:",X1_max)
Y1_min = np.amin(InputY1)
Y1_max = np.amax(InputY1)
InputX1_norm = (InputX1-X1_min)/(X1_max-X1_min)
InputY1_norm = InputY1  #No normalization in output
#InputY1_norm = (InputY1-Y1_min)/(Y1_max-Y1_min)

#Reshape
Xfeatures = 5 #Number of input features
Yfeatures = 1 #Number of output features
samples = InputX1.shape[0] # Number of samples
print("Total Samples:",samples)
InputX1_reshape = np.resize(InputX1_norm,(samples,Xfeatures))
```

```
InputY1_reshape = np.resize(InputY1_norm,(samples,Yfeatures))
print("X1 normalized:",InputX1_reshape)
#print("Y1 normalized:",InputY1_reshape)


#Training data
batch_size = 2000
InputX1train = InputX1_reshape[0:batch_size,:]
InputY1train = InputY1_reshape[0:batch_size,:]
#Validation data
v_size = samples-batch_size
InputX1v = InputX1_reshape[batch_size:batch_size+v_size,:]
InputY1v = InputY1_reshape[batch_size:batch_size+v_size,:]
print("Training Samples:",batch_size)
print("Validation Samples:",v_size)
#print(InputX1v)
print(InputY1v)

#Network hyper parametres
learning_rate0 = 0.001
training_epochs = 500000
display_epoch= 50000
summarize_epoch = 500

# reset everything to rerun in jupyter
tf.reset_default_graph()
#Input
X                                                                =
tf.placeholder(tf.float32,shape=(None,Xfeatures),name="X")#[batch
size, input_features]
#Output
Y                                                                =
tf.placeholder(tf.float32,shape=(None,Yfeatures),name="Labels")

#Neurons
L1 = 6 #Number of neurons in 1st layer
#Layer1 weights
with tf.device('/cpu:0'):
    with tf.name_scope('Layer_1'):
        W_fc1                                                    =
tf.Variable(tf.random_uniform([Xfeatures,L1]),name="W")          #
[input_features,Number of neurons])
        b_fc1 = tf.Variable(tf.random_uniform([L1]),name="bias")
        matmul_fc1=tf.matmul(X, W_fc1) + b_fc1  #Weights * Inputs
        tf.summary.histogram("Layer1_Weights",W_fc1)
        tf.summary.histogram("Layer1_biases",b_fc1)
    with tf.name_scope('ReLU'):
        h_fc1 = tf.nn.relu(matmul_fc1)    #ReLU activation
        #h_fc1=tf.sigmoid(matmul_fc1)     #Sigmoid activation

#Output layer
    with tf.name_scope('Output_Layer') as scope:
```

```
        W_fO=
tf.Variable(tf.random_uniform([L1,Yfeatures]),name="W") #  [Number
of neurons in preceding layer,output_features])
        b_fO                                                    =
tf.Variable(tf.random_uniform([Yfeatures]),name="bias")
        matmul_fco= tf.matmul(h_fc1, W_fO) + b_fO
        output_layer = matmul_fco  #linear activation
        tf.summary.histogram("Output_Layer_Weights",W_fO)
        tf.summary.histogram("Output_Layer_biases",b_fO)
    with tf.name_scope('Softmax') as scope:
        output_layer_prob = tf.nn.softmax(output_layer)  #Applying
softmax activation to find probabilities for each class
with tf.device('/cpu:0'):

    #Loss/cost function
    with tf.name_scope('MSE'):
        mean_squared_error    =    tf.losses.mean_squared_error(Y,
output_layer)
        tf.summary.scalar('mean_squared_error',
mean_squared_error)

    #Decreasing learning rate
    with tf.name_scope('Learning_rate'):
        global_step = tf.Variable(0, trainable=False)
        starter_learning_rate = learning_rate0
        learning_rate                                           =
tf.train.exponential_decay(starter_learning_rate,
global_step,1000000, 0.96, staircase=True)

    #Training step
    with tf.name_scope('Optimizer'):
        #train_step=
tf.train.GradientDescentOptimizer(learning_rate).minimize(mean_sq
uared_error,global_step=global_step)
        train_step=
tf.train.AdamOptimizer(learning_rate).minimize(mean_squared_error
,global_step=global_step)
        #train_step=
tf.train.AdagradOptimizer(learning_rate).minimize(mean_squared_er
ror,global_step=global_step)

 #Merge all the summaries
merged = tf.summary.merge_all()

#Operation to save variables
saver = tf.train.Saver()
#tensorboard                                                    --
logdir=C:\Users\tareq.hossen\Desktop\load_predict\summaries

#Initialization and session
init = tf.global_variables_initializer()
init_local = tf.local_variables_initializer()
```

```
with tf.Session() as sess:
    train_writer                                                =
tf.summary.FileWriter(summaries_dir+"/1/",sess.graph)         #For
writing summaries
    sess.run([init,init_local])  #Initializes all variables
    print("Initial                                          Training
loss:",sess.run([mean_squared_error],feed_dict={X:InputX1train,Y:
InputY1train}))

    for i in range(training_epochs):
        train_size = np.random.randint(low=5,high=batch_size)
        Xtrain = InputX1train[0:train_size,:]
        Ytrain = InputY1train[0:train_size,:]

        if i%display_epoch ==0:
            print("Iteration:",i)
            print("Batch size:",train_size)
            print("Training
loss:",sess.run([mean_squared_error],feed_dict={X:Xtrain,Y:Ytrain
}))
            print("Validation
loss:",sess.run([mean_squared_error],feed_dict={X:InputX1v,Y:Inpu
tY1v}))
            print("Learning rate:",sess.run([learning_rate]))
        if i%summarize_epoch ==0:
            summary,_                                                =
sess.run([merged,train_step],feed_dict={X:Xtrain,Y:Ytrain})
            train_writer.add_summary(summary, i)
        else:
            sess.run([train_step],feed_dict={X:Xtrain,Y:Ytrain})

    #Close summary writer
    train_writer.close()
    # Save the variables to disk.
    save_path = saver.save(sess, model_dir+"Load_predict.ckpt")
    #/tmp/Load_predict.ckpt
    print("Model saved in file: %s" % save_path)

    print("Final                                            training
loss:",sess.run([mean_squared_error],feed_dict={X:InputX1train,Y:
InputY1train}))
    print("Final                                          validation
loss:",sess.run([mean_squared_error],feed_dict={X:InputX1v,Y:Inpu
tY1v}))
    print("Labels:",sess.run([Y],feed_dict={Y:InputY1train}))

print("Prediction:",sess.run([output_layer],feed_dict={X:InputX1t
rain}))
with tf.Session() as sess:
    # Restore variables from disk.
    saver.restore(sess, model_dir+"Load_predict.ckpt")
    print("Model restored.")
```

```
    print("Training
loss:",sess.run([mean_squared_error],feed_dict={X:InputX1train,Y:
InputY1train}))

print("Output:",sess.run([output_layer],feed_dict={X:InputX1train
}))
    print("Validation
Output:",sess.run([output_layer],feed_dict={X:InputX1v}))
    print("Validation Acutual:",InputY1v)


# Recover model and re-run training session
with tf.Session() as sess:
    train_writer                                                =
tf.summary.FileWriter(summaries_dir+"/2/",sess.graph)         #For
writing summaries
    # Restore variables from disk.
    saver.restore(sess, model_dir+"Load_predict.ckpt")
    print("Model restored.")

    print("Training
loss:",sess.run([mean_squared_error],feed_dict={X:InputX1train,Y:
InputY1train}))
    for i in range(training_iterations):
        summary,_                                               =
sess.run([merged,train_step],feed_dict={X:InputX1train,Y:InputY1t
rain})
        train_writer.add_summary(summary, i)

        if i%display_iterations ==0:
            print("Iteration:",i)
            print("Training
loss:",sess.run([mean_squared_error],feed_dict={X:InputX1train,Y:
InputY1train}))
            print("Training
accuracy:",sess.run([accuracy],feed_dict={X:InputX1train,Y:InputY
1train}))
            print("Validation
loss:",sess.run([mean_squared_error],feed_dict={X:InputX1v,Y:Inpu
tY1v}))
            print("Learning rate:",sess.run([learning_rate]))

    # Save the variables to disk.
    save_path = saver.save(sess, model_dir+"Load_predict.ckpt")
    #/tmp/RC_classifier.ckpt
    print("Model saved in file: %s" % save_path)

    print("Final                                               training
loss:",sess.run([mean_squared_error],feed_dict={X:InputX1train,Y:
InputY1train}))
```

```
    print("Final                                                    Training
accuracy:",sess.run([accuracy],feed_dict={X:InputX1train,Y:InputY
1train}))
    print("Final                                                validation
loss:",sess.run([mean_squared_error],feed_dict={X:InputX1v,Y:Inpu
tY1v}))

print("Labels:",sess.run([class_labels],feed_dict={Y:InputY1train
}))

print("Prediction:",sess.run([class_pred],feed_dict={X:InputX1tra
in}))
```

#CODE FOR IMPLEMENTION OF RECURRENT NEURAL NETWORK USING KERAS WITH TENSOR-FLOW BACKEND:

```
from __future__ import print_function
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.layers.recurrent import GRU
from keras.layers.recurrent import SimpleRNN
from keras.models import Sequential

# random seed
np.random.seed(1234)
df_raw = pd.read_csv('C:\data\hourly_load_2010.csv', header=None)
# load raw data
df_raw_array = df_raw.values
# daily load
list_daily_load = [df_raw_array[i,:] for i in range(0, len(df_raw))
if i % 24 == 0]
# hourly load (23 loads for each day)
list_hourly_load = [df_raw_array[i,1]/100000 for i in range(0,
len(df_raw)) if i % 24 != 0]
# the length of the sequnce for predicting the future value
sequence_length = 23

# define a function to convert a vector of time series into a 2D
matrix
def convertSeriesToMatrix(vectorSeries, sequence_length):
    matrix=[]
    for i in range(len(vectorSeries)-sequence_length+1):
        matrix.append(vectorSeries[i:i+sequence_length])
    return matrix

# convert the vector to a 2D matrix
matrix_load = convertSeriesToMatrix(list_hourly_load,
sequence_length)

# shift all data by mean
matrix_load = np.array(matrix_load)
shifted_value = matrix_load.mean()
matrix_load -= shifted_value
print ("Data  shape: ", matrix_load.shape)

# split dataset: 90% for training and 10% for testing
train_row = int(round(0.9 * matrix_load.shape[0]))
train_set = matrix_load[:train_row, :]

# shuffle the training set (but do not shuffle the test set)
```

```
np.random.shuffle(train_set)
# the training set
X_train = train_set[:, :-1]
# the last column is the true value to compute the mean-squared-
error loss
y_train = train_set[:, -1]
# the test set
X_test = matrix_load[train_row:, :-1]
y_test = matrix_load[train_row:, -1]

# the input to LSTM layer needs to have the shape of (number of
samples, the dimension of each element)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1],
1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# build the model
model = Sequential()
# layer 1: LSTM
model.add(GRU(input_dim=1,                          output_dim=150,
return_sequences=True))
model.add(Dropout(0.2))
# layer 2: LSTM
model.add(GRU(output_dim=150, return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(output_dim=150, return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(output_dim=150, return_sequences=False))
model.add(Dropout(0.2))
# layer 3: dense
# linear activation: a(x) = x
model.add(Dense(output_dim=1, activation='linear'))

# compile the model
model.compile(loss="mse", optimizer="adam")

# train the model
model.fit(X_train,    y_train,    batch_size=512,    nb_epoch=1,
validation_split=0.05, verbose=1)

# evaluate the result
test_mse = model.evaluate(X_test, y_test, verbose=1)
test_mape = model.evaluate(X_test, y_test, verbose=1)
print ('\nThe mean squared error (MSE) on the test data set is %.3f
over %d test samples.' % (test_mse, len(y_test)))
print ('\nThe mean absolute percentage error (MAPE) on the test
data set is %.3f over %d test samples.' % (test_mape*100,
len(y_test)))

# get the predicted values
predicted_values = model.predict(X_test)
num_test_samples = len(predicted_values)
```

```
predicted_values          =          np.reshape(predicted_values,
(num_test_samples,1))

# plot the results
fig = plt.figure()
plt.plot(y_test + shifted_value)
plt.plot(predicted_values + shifted_value)
plt.xlabel('Hour')
plt.ylabel('Electricity load (*1e5)')
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(y_test[:150])
plt.plot(predicted_values[:150])
plt.show()
```