2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)

Short-term Load Forecasting with LSTM based Ensemble Learning

Lingxiao Wang, Shiwen Mao, and Bogdan Wilamowski
Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849-5201
Email: lzw0039@auburn.edu, smao@ieee.org, wilambm@auburn.edu

***Abstract*—In this paper, a short-term load forecasting framework with long short-term memory (LSTM)-based ensemble learning is proposed. To fully exploit the correlation in data for accurate load forecasting, the data is first clustered and each cluster is used to train an LSTM model. Then a Fully Connected Cascade (FCC) Neural Network is incorporated for ensemble learning, which is solved by an enhanced Levenberg-Marquardt (LM) training algorithm. The proposed framework is tested with a public dataset, where its superior performance over several baseline schemes is demonstrated.**

***Index Terms*—Short-term load forecasting; Deep learning; Ensemble learning; Long short-term memory (LSTM); Levenberg-Marquardt (LM) algorithm.**

I. Introduction

With the rapid advances in sensing and acquisition, transmission, storage, computing and analytics, the era of big data has come [1]. Many advanced data analytic techniques have been proposed and found wide applications in our society. In the power industry, data analytics play an essential role in daily power system operation and planning. One major challenge for energy management in the emerging smart grid is the uncertainty in both power supply (e.g., renewable energy generation) and demand (e.g., load demand from the service area). There is a compelling need to accurately predict future generation and load for efficient power management [2]–[5]. Such predictions will help to make intelligent decisions for improving power quality, saving energy, better utilizing renewable energy sources, and reducing cost [6].

Among the various forecasting timescales, short-term load forecasting (STLF) (e.g., hourly, rather than daily) is of particular interest, which allows the energy management system to achieve more timely situation awareness, and to react to power dynamics more quickly to guarantee stability of the grid and more efficient use of energy. In the literature, a variety of methods has been proposed for STLF, which can be summarized into two categories: statistical methods and machine learning approaches. The statistical methods include regression, exponential smoothing, and least absolute shrinkage and selection operator (LASSO), etc. [2]–[4]. In the machine learning category, support vector machine (SVM), neural network, fuzzy system, and wavelet transform have been applied [7]–[10]. In [7], a second-order optimization algorithm is used to train a Radial Basis Function (RBF) neural network. An ensemble of Extreme Learning Machines (ELM) is proposed in [8], [9], while a wavelet neural network is investigated in [10] as well. A recent interest is to apply deep learning to the STLF problem. In [11], the authors apply the Convolutional Neural Network (CNN) model to cluster the input data and connect them with a three hidden-layer neural network to predict the electric load. Deep residual networks have also been shown to be effective in accurate load forecasting in [12]. In [5] and [13], Long Short-term Memory (LSTM) is utilized for short-term solar power generation and residential load forecasting, respectively.

In this paper, we propose an ensemble learning approach to tackle the STLF problem. The proposed framework utilizes LSTM models in the first level learner, and a fully connected cascade (FCC) neural network model in the second-level learner for model fusion. Our proposed framework has three salient features. First, it is a *deep learning* based approach where LSTM recurrent neural networks (RNN) are incorporated. Deep learning, with its considerable successful applications in many fields, has been shown to achieve superior performance in time series analysis [14]. When applied to time series data, the LSTM models can capture the temporal dynamic behavior [15], making them highly suited for load prediction.

Second, unlike most existing work that belong to supervised learning, the proposed framework *integrates an unsupervised learning with a supervised learning model* for load forecasting. Specifically, it employs clustering, an unsupervised learning, to partition the data into clusters, each is then used to train an LSTM model. This way, the intrinsic correlation among the dataset can be better exploited for more accurate forecasting. Then the forecasting results from the LSTM models will be fused by the FCC neural network as a supervised learning, to further improve the accuracy. Third, the proposed framework is an *ensemble learning*, which first trains multiple LSTM models, one for each types of data. Then the LSTM model outcomes will be combined by the FCC neural network. The boosted combined model (ensemble) is often stronger than the base learners, which can make more accurate predictions [16].

Our key contributions in this work can be summarized as follows. First, we propose an ensemble learning approach, which integrates several state-of-the-art machine learning algorithms into a holistic framework for accurate load forecasting. Second, we evaluate and compare different models including different clustering algorithms in the proposed framework for generating the learning models in the first level learner. Third, we propose to adopt the FCC neural network model for model fusion in the second-level learner. Most existing works use a

978-1-7281-2980-8/19/$31.00 ©2019 IEEE
DOI 10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00145

linear combination by assigning weights to the predictors and solving a linear programming (LP) problem that minimizes a loss function. However, we believe that model fusion is a more complex problem; using an FCC NN can capture both the linear/nonlinear relationship among individual models, thus leading to higher prediction accuracy. Fourth, for weight training, a modified Levenberg-Marquardt (LM) optimization algorithm is employed, which is fast converging and stable. The proposed framework is validated with a public dataset and compared with several state-of-the-art schemes, where its superior performance on highly accurate load predictions is demonstrated.

The remainder of this paper is organized as follows. In Section II, we present the problem statement and an overview of the proposed framework. We then discuss data preprocessing in Section III, generation of the LSTM models in the first level learner in Section IV, and the FCC neural network based ensemble learning model in the second-level learner in Section V. Our experimental validation of the proposed framework is presented in Section VI. Section VII concludes this paper.

II. The Proposed Framework

A. Problem Statement

The power load forecasting problem is formulated as follows. A given time series dataset $\boldsymbol{S}_t = \{\boldsymbol{m}_1, \boldsymbol{m}_2, ..., \boldsymbol{m}_f, \boldsymbol{\ell}\}$ is composed of $(f+1)$ historical data series, where $\boldsymbol{\ell} = \{\ell_1, \ell_2, ..., \ell_t\}$ is the load historical data and $\boldsymbol{m}_i = \{m_{i1}, m_{i2}, ..., m_{it}\}$ is the historical data of the ith variable that affects load. The *goal* is to accurately predict the load at a future time $(t+\tau)$, $\tau > 0$, denoted by $\hat{\ell}_{t+\tau}$. The forecasted value $\hat{\ell}_{t+\tau}$ is obtained by a fitting function of the combination of lagged forms of $\boldsymbol{S}_t$, as

$$\hat{\ell}_{t+\tau} = g(\boldsymbol{S}_t). \tag{1}$$

The fitting function $g(\cdot)$ is to be established by our machine learning based predictive method. In this paper, we focus on hourly load forecasting. So the given dataset $\boldsymbol{S}_t$ consists of historical hourly data of power load and historical hourly data of various factors that affect power load.

B. The Concept of Stacking

Our predictive framework is based on the concept of *stacking* [17], which is a general procedure where machine learning tools are trained to integrate individual learners [16]. The procedure has two levels of learners. The first-level learner consists of multiple *individual learning models* and the second-level learner is a *combiner*. The data for stacking is divided into three parts. The first part is used to generate the first-level learning models. The second part of data, which is combined with the new data generated from the first-level classifiers, are used for training the second-level learner. The third part of data is combined with the new data generated by the first-level learner to test the framework. In this paper, we propose to use a number of homogeneous LSTM models for the first-level learning and an FCC neural network for the second-level learning.

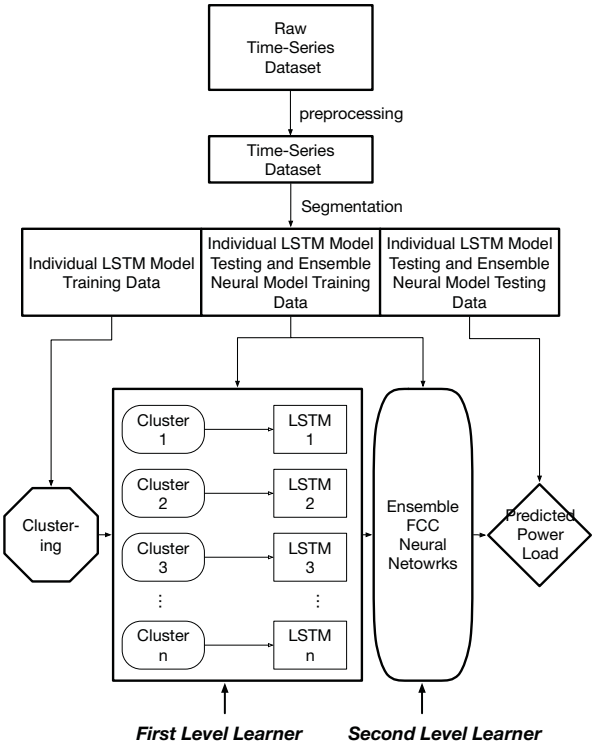

Fig. 1. Hourly power load prediction framework with two levels of learners.

C. The Proposed Framework

An overview of the proposed hourly power load forecasting framework is presented in Fig. 1. The proposed framework is based on the stacking model and consists of two levels of learners, which are:

- First level learner: generates a set of LSTM predictive models using a clustering algorithm.
- Second level learner: integrates the set of LSTM predictions with a fully connected, cascaded neural network using a modified Levenberg-Marquardt (LM) algorithm.

III. Dataset Preprocessing and Segmentation

A. Sliding Window and Data Splitting

We use a sliding window technique in our framework for load prediction: when predicting load, historical data in a window of four immediate previous years is used. Specifically, the window of four historical years is divided into three time periods, as shown in Fig. 2. The data in the first time period P1 is used for data clustering and then to establish the corresponding LSTM model for each data cluster in the first-level learner. The data in the second time period P2 is used to train the neural network in the second-level learner for model combining. The data in the last time period P3 is used to test

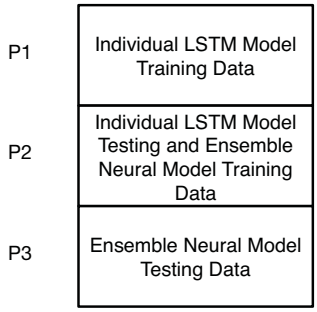

Fig. 2. Illustration of time period splitting: a window of four historical years is dived into three time periods with ratios 2:1:1.

the established forecasting model. The ratios of splitting for the three time periods are 2:1:1.

B. Normalization

Normalization is an essential data preprocessing technique in machine learning. In this paper, we apply an unsupervised clustering algorithm to group time series into different clusters and a supervised learning algorithm based on a deep neural network to extract features from the clustered time series data. These two learning models both benefit from normalized data. In [18], the authors show that in clustering, normalizing data reveals the true similarity between two time-series under Euclidean distance. In [19], [20], it is shown that normalization of input data speeds up the convergence of the neural network, even when the features are not decorrelated.

In time-series prediction, one issue that needs to be careful of is *data snooping*, which occurs when a dataset is used more than once for purposes of inference [21]. In our model only training data, i.e., for time periods P1 and P2, are normalized. In each period, we normalize the data for each variable, $\mathbf{X}$, as

$$X_i^{norm} = \frac{X_i - \min(\mathbf{X})}{\max(\mathbf{X}) - \min(\mathbf{X})}, \tag{2}$$

where X_i and X_i^{norm} are the original and normalized ith data sample in $\mathbf{X}$, respectively. In the testing phase, we first restore the new data generated by the first-level learner from normalized form to the original form, and then add them to the original testing dataset to avoid data snooping.

IV. LSTM Predictive Model Generation by Clustering

A. Motivations for Clustering

The first-level learning model generation procedure is given in Algorithm 1. For electricity load forecasting, the load at a given future time maybe more or less correlated with the historical data. For example, the future load at noon is usually more correlated with the load of the noon of the previous day, but less correlated with the load of an early morning hour of the previous day. Also, the load of a hot summer day is more correlated with the historical data of a summer day than a winter day. Naturally, it would be beneficial to cluster the data into suitable subsets, and to train a different learning model for each cluster, to better exploit the correlation in the dataset [5].

Algorithm 1: Procedure of Building the First Level LSTM Predictors

1 Using a clustering algorithm to partition the input data in dataset P1 into k clusters $\boldsymbol{C} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, ..., \boldsymbol{C}_k\}$;
2 According to the clustering results, divide dataset P1 into k individual datasets (i.e., including both input/output data): $\{\mathrm{P1}_1, \mathrm{P1}_2, ..., \mathrm{P1}_k\}$, where $\mathrm{P1}_i$ is the ith dataset produced by the ith cluster $\boldsymbol{C}_i$;
3 Train an individual LSTM model i using each dataset $\mathrm{P1}_i$, $i = 1, 2, ..., k$;

Clustering, as an unsupervised machine learning technique, is the process of partitioning data into clusters of similar features [22]. It is different from classification, which is provided with labeled training data and extracts features to label new data. Clustering works with unlabeled dataset and captures the natural structure within the dataset [23]. We advocate the use of an unsupervised learning in our forecast model, because the power load is affected by various obvious and latent factors. Usually the load contains considerable uncertainty, while still exhibiting a rough temporal pattern (e.g., daily, weekly, or monthly patterns). We assume that the short-term electric load change is mainly affected by the historical data of the time before the current time. The historical data is grouped based on the similarity amount the data samples.

After the dataset is clustered and the learning model for each cluster is well trained, the next question is when a new input data sample arrives, how to use the trained models to make a prediction. In our prior work [5], the new input data is classified into the most similar cluster, and the corresponding learning model is used for prediction. In this paper, we propose to adopt ensemble learning. We assume each of the homogeneous models has an impact on power load prediction. The new data is fed into each trained model, and the outputs from all the models are then fused using a neural network model, to generate the prediction value (see Section V).

B. Clustering Algorithms

Unlike a formal static dataset, the power load time series data is usually time-variant and susceptible to interference such as noise, shift, and deformation, etc. [24]. The dataset could be extremely large as it might include 15-minute samples over a period of several years. It is challenging to choose an appropriate clustering method to handle such data.

There are many types of clustering algorithms available. Three categories of methods are often used: (i) partitioning, (ii) hierarchical, and (iii) density based. In our framework, we choose one of the most widely used algorithms in each of the three categories, which are K-means++, DBSCAN, and BIRCH, and examine their performance on load prediction.

Algorithm 2: K-means++ Clustering Algorithm

```
Randomly choose one center c_1;
Choose a new center c_i using D^2 weighted probabilities;
Repeat Step 2, until k centers C = {c_1, c_2, ..., c_k} are chosen;
while C is changed do
    Assign each data sample to its closest cluster c_i;
    Compute and update the centroid of each cluster;
end
```

1) K-Means++ Clustering Algorithm: K-means is one of the most popular partitional prototype-based clustering algorithm. In order to improve both the speed and accuracy of the K-means clustering algorithm (which is NP-hard), K-means++ was proposed in [25] to choose the seeds (i.e., the initial cluster centers) for K-means. Initiated by K-means++, K-means can guarantee a solution that is $\mathcal{O}(\log k)$-competitive to the optimal K-means solution. The detailed K-means++ algorithm is presented in Algorithm 2.

2) BIRCH Clustering Algorithm: BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is a hierarchical clustering algorithm, which represents a more informative structure and does not require a prescribed number of clusters [24]. Usually hierarchical methods suffer from some drawbacks, such as time-consuming, sensitive to outliers, and that it cannot correct mistaken decisions that once have been taken, as the step of merging or splitting cannot be canceled [26]. The BIRCH clustering algorithm is utilized in our framework for the following reasons [27]:

- BIRCH is a hierarchical clustering method, which does not need a predetermined number k of clusters.
- BRICH BRICH is suited for large datasets.
- BRICH allows each clustering decision made without checking all data as it is local.
- BRICH can remove noise (outliers).
- BRICH minimizes the running time and makes full use of memory to derive the best results.
- BRICH produces highly competitive clusters with a single scan of the dataset.

BIRCH is based on the concept of Clustering Feature (CF) and CF tree. CF is the a vector summarizing information about clusters, which is defined as $\boldsymbol{CF} = (N, LS, SS)$, where N is the number of samples in the cluster, LS is the linear sum of the samples, and SS is the square of the sum of the points. CF tree is a height-balanced tree with two parameters: branching factor B and threshold T [27]. Each interior leaf node in the tree has at most B entries form $[CF_i, child_i]$, where $i \in [1, B]$ and each leaf node has at most L entries in $[CF_i]$, where $i \in [1, L]$. "$child_i$" is a pointer to the ith child node. Moreover, the threshold T ensures that the diameter of each entry in the leaf to be less than T. An example of a CF tree with $B = 4$, $L = 3$, and $T = 1$ is shown in Fig. 3. A CF tree is built while new data is being inserted dynamically. The detailed BIRCH algorithm is given in Algorithm 3.

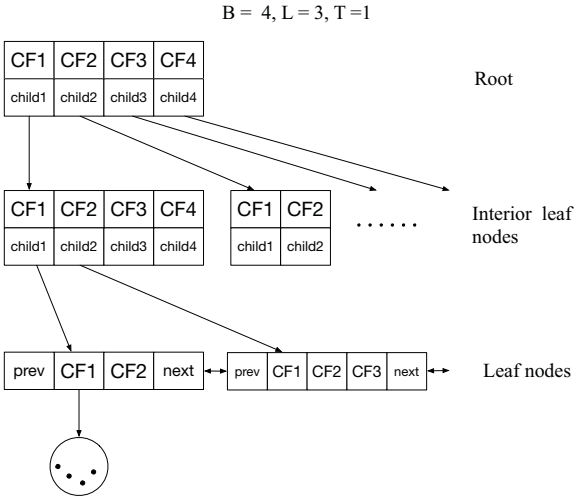

Fig. 3. A CF tree example.

Algorithm 3: BIRCH Clustering Algorithm

```
Build a CF tree after loading the dataset;
(Optional) Create a smaller tree if necessary for Step 3;
Perform global clustering;
(Optional) Use the centroids of clusters to redistribute the data samples by Step 3, and then refine the clusters;
```

3) DBSCAN Clustering Algorithm: Density Based Spatial Clustering of Application with Noise (DBSCAN) is designed to discover the clusters and noise in a spatial database, which is simple and effective [28]. It uses parameters $(\epsilon, Minpts)$ to characterize the density of the data space, which defines the concept of ϵ-neighborhood, core project, directly density-reachable, density-reachable, and density-connected. Clusters are formed by iteratively connecting the directly density-reachable instances starting from the core objects [16].

C. Long Short-Term Memory (LSTM)

LSTM is an improved RNN inspired by the idea of introducing self-loops to store long-term conditions of the gradient [29]. The LSTM neural cell is illustrated in Fig. 4.

Each LSTM cell has four components: input gate i_t, forget gate f_t, output gate o_t, and state unit c_t. Input gate is a neuron cell with a sigmoid activation function, which is connected with the current time input matrix x_t and the former time output h_{t-1}, as

$$i_t = \sigma(\boldsymbol{W}^i h_{t-1} + \boldsymbol{U}^i x_t + \boldsymbol{b}^i), \tag{3}$$

where $\boldsymbol{W}^i$ are the recurrent weights of the input gate, $\boldsymbol{U}^i$ are the input weights of the input gate, and $\boldsymbol{b}^i$ are the biases of the input gate. Forget gate is similar to input gate but with its own parameters $\boldsymbol{W}^f$, $\boldsymbol{U}^f$, and $\boldsymbol{b}^f$, which is defined by

$$f_t = \sigma(\boldsymbol{W}^f h_{t-1} + \boldsymbol{U}^f x_t + \boldsymbol{b}^f). \tag{4}$$

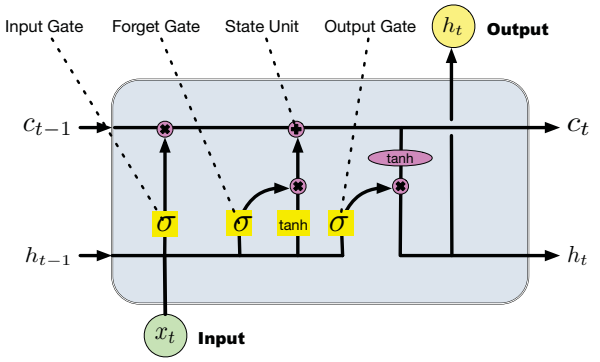

Fig. 4. The LSTM neural cell structure.

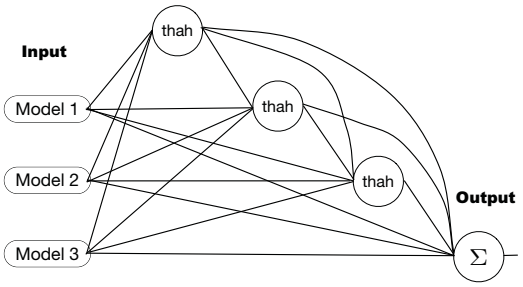

Fig. 5. A simple example of FCC ensemble neural network.

State unit c_t is the most important part of LSTM. It determines how much information from former cells is contained that affect the current statement. Correspondingly, the state unit has its own parameters $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{b}$, which is defined by

$$c(t) = f_t \cdot c(t-1) + i_t \cdot \sigma(\boldsymbol{W} h_{t-1} + \boldsymbol{U} x_t + \boldsymbol{b}). \quad (5)$$

The output gate o_t has parameters $\boldsymbol{W}^o$, $\boldsymbol{U}^o$, and $\boldsymbol{b}^o$ as well, and is defined by

$$o_t = \sigma(\boldsymbol{W}^o h_{t-1} + \boldsymbol{U}^o x_t + \boldsymbol{b}^o). \quad (6)$$

Finally, the function of output h_t is defined as

$$h_t = \tanh(c_{t-1}) \cdot o_t. \quad (7)$$

V. LSTM-based Ensemble Learning

After training the LSTM models in the first-level learner with dataset P1, we next use dataset P2 to train the learner in the second level. In particular, we feed data samples in P2 to the trained LSTM predictors. The outputs from the LSTM predictors are then used to train the ensemble neural network in the second-level learner for model combination (i.e., fusion).

We propose to use the FCC neural network for ensemble learning. A simple example of the FCC ensemble neural network is shown in Fig 5. The example has three base models to be fused, i.e., Model 1, Model 2, and Model 3, and four neurons. The first three neurons use the $\tanh(\cdot)$ activation function, while the last neuron uses a simple linear summation function. In [30], it is shown that FCC is superior to traditional neural network structures. This is because with the same number of neurons, FCC requires more connections/weights than traditional neural networks to connect from neurons to neurons, making it *deeper* than traditional neural networks. In addition, the identity mapping from every input and from every latent variables is similar to that in Deep Residual Networks [31]. The optimization of model combination is described as follows.

A. Problem Formulation

Assuming there are k trained LSTM models in the first-level learner, the predicted load i by the ensemble model is

$$\hat{\ell}_E^{(i)}(\boldsymbol{\omega}) = f(\hat{\boldsymbol{\ell}}^{(i)}; \boldsymbol{\omega}), \quad (8)$$

where $f(\boldsymbol{x}; \boldsymbol{\omega})$ is the output of the ensemble neural network, $\hat{\boldsymbol{\ell}}^{(i)}$ is the load forecast vector predicted by the k LSTM models for sample i in P2, and $\boldsymbol{\omega}$ is the weight vector of the ensemble neural network. We define an average of the cost functions $C^{(i)}(\boldsymbol{\omega})$ over the validation and ensemble dataset P2 as the *lost function*, given by

$$\mathcal{L}(\boldsymbol{\omega}) = \frac{1}{M} \sum_{i=1}^{M} C^{(i)}(\boldsymbol{\omega}), \quad (9)$$

where M is the total number of samples in the validation and ensemble dataset P2, and $C^{(i)}(\boldsymbol{\omega})$ is the cost function for each sample i, calculated by sum squared error (SSE) as

$$C^{(i)}(\boldsymbol{\omega}) = \text{SSE}(\ell^{(i)}; \hat{\ell}_E^{(i)}(\boldsymbol{\omega})), \quad (10)$$

where $\ell^{(i)}$ is the actual value of power load in sample i (i.e., the label) and $\hat{\ell}_E^{(i)}(\boldsymbol{\omega})$ is the predicted value calculated by the ensemble model for sample i. An L1 regularizer is added to the per-sample cost function (10). We have

$$C^{(i)}(\boldsymbol{\omega}) = \text{SSE}(\ell^{(i)}; \hat{\ell}_E^{(i)}(\boldsymbol{\omega})) + \alpha \cdot \|\boldsymbol{\omega}\|. \quad (11)$$

Substituting (11) into (9), the lost function $\mathcal{L}(\omega)$ is defined as

$$\mathcal{L}(\boldsymbol{\omega}) = \frac{1}{M} \sum_{i=1}^{M} (\hat{\ell}_E^{(i)}(\boldsymbol{\omega}) - \ell^{(i)})^2 + \alpha \|\boldsymbol{\omega}\|. \quad (12)$$

B. Second-order Gradient Descent Algorithm

Once the loss function is defined, we use the data samples in P2 to tune the parameters $\boldsymbol{\omega}$ to minimize the loss function $\mathcal{L}(\boldsymbol{\omega})$. Unlike traditional optimization problems, training neural networks is based on empirical risk minimization [15]. Error backpropagation, a first order gradient based algorithm, is the most commonly used method to train neural networks. However, ill-conditioning and local-minima are common challenges during the training process.

To this end, second-order gradient descent algorithms provide an effective solution [32]. The Levenberg-Marquardt (LM) Algorithm, also known as the quasi-Newton method, is able to not only achieve stability but also speed up convergence by approximating the second-order derivative [33]. Although the LM algorithm is quite successful, it sometimes fails when handling a compact neural network architecture. The failing of this training is known as the flat-spot problem [34]. It is

mainly caused by large weight values that keep the neurons stuck in the saturated region. The derivative of gradient tends to zero, thus causing a vanishing gradient condition.

A potential solution is to adopt a large-sized neural network, which, however, may cause *overfitting*. The Vapnik-Chervonenkis dimension (or, the VC dimension), is introduced to measure the capacity of neural networks [35]. The VC dimension of neural networks with activation function $\tanh(\cdot)$ and output node $\text{sign}(\cdot)$ is estimated by [36]

$$d_{VC} \approx \mathcal{O}(V \cdot |\boldsymbol{\omega}|), \tag{13}$$

where V is the number of neurons and $|\boldsymbol{\omega}|$ is the number of weights. Overfitting usually occurs when using a neural network with a high d_{VC} value. The general guideline is to use as few neurons as possible, while providing a sufficient learning capacity for the training samples. This observation motivates us to adopt the FCC ensemble neural network for model combination.

As mentioned, the LM algorithm usually achieves a better performance than first-order gradient methods, but it is easier to get stuck at local minima in the process of weight updates when training compact sized neural networks. Therefore, we adopt a modified LM algorithm to overcome the flat-spot problem in this paper [37].

C. Modified Levenberg-Marquardt (LM) Algorithm

The LM Algorithm approximates the Hessian matrix by $\boldsymbol{J}^T\boldsymbol{J}$, where $\boldsymbol{J}$ is the Jacobian Matrix. Inspired by the concept of *trust region* [38], the LM algorithm introduces a dumping factor μ, whose value is updated iteratively according to the loss function (see (16)). When μ is large, the neural network weights are updated with the gradient descent method; when μ is small, the neural network weights are updated with the GaussNewton algorithm. The weights are updated as

$$\Delta\boldsymbol{\omega}_k = -[\boldsymbol{J}(\boldsymbol{\omega}_k)^T\boldsymbol{J}(\boldsymbol{\omega}_k) + \mu_k\boldsymbol{I}]^{-1}\boldsymbol{J}(\boldsymbol{\omega}_k)^T\mathcal{L}(\boldsymbol{\omega}_k), \tag{14}$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{J}(\boldsymbol{\omega}_k)$ is the Jacobian Matrix in the kth iteration, given by

$$\boldsymbol{J}(\boldsymbol{\omega}_k) = \left[\frac{\partial\mathcal{L}(\boldsymbol{\omega}_k)}{\partial\omega_{1k}}, \frac{\partial\mathcal{L}(\boldsymbol{\omega}_k)}{\partial\omega_{2k}}, \frac{\partial\mathcal{L}(\boldsymbol{\omega}_k)}{\partial\omega_{3k}}, ..., \frac{\partial\mathcal{L}(\boldsymbol{\omega}_k)}{\partial\omega_{Wk}}\right], \tag{15}$$

where ω_{ik} is the value of weight i in iteration k, and W is the total number of weights. In each iteration, the dumping factor μ_k is updated as

$$\mu_k = \|\mathcal{L}(\boldsymbol{\omega}_k)\|^\beta, \tag{16}$$

where $\beta \in (0, 2]$. It has been proved that the convergence order of LM method is under the local bound condition [39].

In [37], a modified LM algorithm is proposed. At each integration, the weights are updated in two steps: (i) the first step is the normal LM step as $\boldsymbol{d}_k = \Delta\boldsymbol{\omega}_k$, and (ii) the second step is a line search for approximating the LM step $\Delta{\boldsymbol{\omega}_k}'$. A combination of these two steps is given by

$$\boldsymbol{s}_k = \Delta\boldsymbol{\omega}_k + \alpha_k\Delta{\boldsymbol{\omega}_k}', \tag{17}$$

Algorithm 4: The Modified Levenberg MarQuardt Method

1 Set $0 < m < \mu_1$ and $0 < p_0 < p_1 < p_2 < 1$, where $\mu_1 = 10^{-6}$, $m = 10^{-7}$, $p_0 = 10^{-4}$, $p_1 = 0.2$, $p_2 = 0.8$, and $k = 1$;

2 If $\boldsymbol{J}(\boldsymbol{\omega}_k)^T\boldsymbol{J}(\boldsymbol{\omega}_k) = 0$, then stop;

3 Compute $r_k = Ared_k/Pred_k$, and let

$$\boldsymbol{\omega}_{k+1} = \begin{cases} \boldsymbol{\omega}_k + \boldsymbol{s}_k, & \text{if } r_k > p_0 \\ \boldsymbol{\omega}_k, & \text{otherwise;} \end{cases} \tag{23}$$

4 Compute

$$\alpha_{k+1} = \begin{cases} 4\alpha_k, & \text{if } r_k < p_1 \\ \alpha_k, & \text{if } r_k \in [p_1, p_2] \\ \max(0.25\alpha_k, m), & \text{if } r_k > p_1; \end{cases} \tag{24}$$

5 Set $k = k + 1$, and go to Step 2;

where α_k is a parameter iterative updated as in (24). Thus the weights of the neural network are updated as

$$\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k + \Delta\boldsymbol{\omega}_k + \alpha_k\Delta{\boldsymbol{\omega}_k}', \tag{18}$$

where α_k is the step size for $\Delta{\boldsymbol{\omega}_k}'$ and $\Delta{\boldsymbol{\omega}_k}'$ is given by

$$\begin{aligned}\Delta\boldsymbol{\omega}_k' = &-[\boldsymbol{J}(\boldsymbol{\omega}_k + \Delta\boldsymbol{\omega}_k)^T\boldsymbol{J}(\boldsymbol{\omega}_k + \Delta\boldsymbol{\omega}_k) + \mu_k'\boldsymbol{I}]^{-1} \\ &\cdot \boldsymbol{J}(\boldsymbol{\omega}_k + \Delta\boldsymbol{\omega}_k)^T\mathcal{L}(\boldsymbol{\omega}_k + \Delta\boldsymbol{\omega}_k),\end{aligned} \tag{19}$$

where $\mu_k' = \|\mathcal{L}(\boldsymbol{\omega}_{k+1})\|^\beta$. In order to reduce the computational complexity, $\boldsymbol{J}(\boldsymbol{\omega}_k + \Delta\boldsymbol{\omega}_k)$ can be approximated by $\boldsymbol{J}(\boldsymbol{\omega}_k)$ and μ_k' can be approximated by μ_k. Therefore, Eq. (19) can be rewritten as

$$\Delta\boldsymbol{\omega}_k' = -[\boldsymbol{J}(\boldsymbol{\omega}_k)^T\boldsymbol{J}(\boldsymbol{\omega}_k) + \mu_k\boldsymbol{I}]^{-1}\boldsymbol{J}(\boldsymbol{\omega}_k)^T\mathcal{L}(\boldsymbol{\omega}_k + \Delta\boldsymbol{\omega}_k). \tag{20}$$

The trust region technique is used to justify whether $\boldsymbol{s}_k$ is a good step or not. The actual reduction at the kth iteration is

$$Ared_k = \|\mathcal{L}(\boldsymbol{\omega}_k)\|^2 - \|\mathcal{L}(\boldsymbol{\omega}_k + \boldsymbol{s}_k)\|^2, \tag{21}$$

while the new predicted reduction is

$$\begin{aligned}&Pred_k \\ &= \|\mathcal{L}(\boldsymbol{\omega}_k)\|^2 - \|\mathcal{L}(\boldsymbol{\omega}_k) + \boldsymbol{J}(\boldsymbol{\omega}_k)\boldsymbol{d}_k\|^2 + \|\mathcal{L}(\boldsymbol{\omega}_k + \boldsymbol{d}_k)\|^2 - \\ &\quad \|\mathcal{L}(\boldsymbol{\omega}_k + \boldsymbol{d}_k) + \alpha_k\boldsymbol{J}(\boldsymbol{\omega}_k)\Delta{\boldsymbol{\omega}_k}'\|^2.\end{aligned} \tag{22}$$

The modified LM algorithm is presented in Algorithm 4.

VI. Experimental Results

In this section, the ISO-NE dataset [40] is used to verify the efficacy of the proposed framework. The hourly load and temperature data from Jan. 1, 2007 (2007-1-1) through Dec. 31, 2018 (2018-12-31) are used as our dataset. The New England area is divided into eight zones: Connecticut (CT), Maine (ME), New Hampshire (NH), Rhode Island (RI), and Vermont (VT), Massachusetts of NEM (NEMASS), Massachusetts of SEM (SEMASS), and Massachusetts of WC (WCMASS), which are all served by the ISO-NE transmission

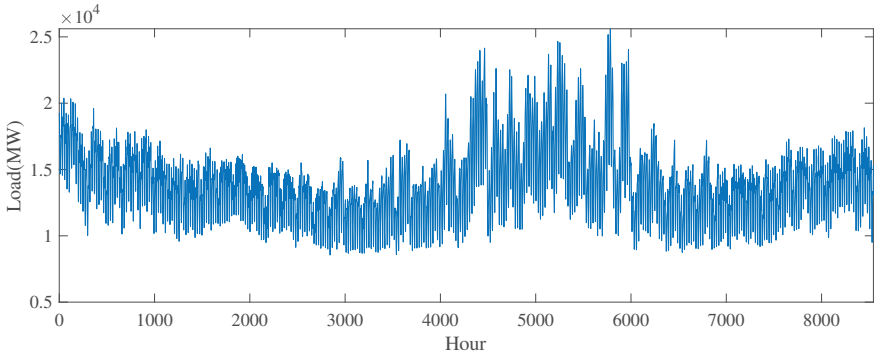

Fig. 6. The 2018 overall system load of ISO-New England.

TABLE I
INPUT DATA CONSTRUCTION AND TEST CONFIGURATION

Input	
ℓ_h^{month}	Load of the hth hour of the day that are 4, 8, 12, 16 and 24 weeks prior to the next day
ℓ_h^{week}	Load of the hth hour of the day that are 1, 2, 3, 4 weeks prior to the next day
ℓ_h^{day}	Load of the hth hour of the day that are 1, 2, 3, 4, 5, 6, 7 days prior to the next day
ℓ_h^{hour}	Load of the most recent 24 hours prior to the hth hour of the next day
T_h^{month}	Temperature of the same hour as ℓ_h^{month}
T_h^{week}	Temperature of the same hour as ℓ_h^{week}
T_h^{day}	Temperature of same hour as ℓ_h^{day}
T_h	Temperature of the hth hour of the next day ℓ_h^{day}
S, W, H	Dummy variables, indicator (1,0) for season (winter, summer), weekend, and holiday
Output	
$\hat{\ell}$	24 hours ahead load

system [41]. Fig. 6 presents the hourly load demands for the year of 2018 for the overall system.

The proposed framework is implemented and executed on a Mac-book Pro laptop with Intel Core i7 2.3Ghz processor and 16.0 GB of Ram. The first-level learner is implemented using Keras 2.2.4 and Sklearn 0.20.0 in the Python 3.5 environment. The neural network for model fusion is implemented using ADNBN in Matlab R2018a.

We use the data for the time period from 2007-1-1 to 2011-12-31. A four-year sliding window is used to include historical data of the immediate past for prediction (see Section III-A). Thus the system load in Year 2010 is forecasted using historical data from 2007-1-1 to 2010-1-1, while the system load in Year 2011 is forecasted using historical data from 2008-1-1 to 2011-1-1. In order to compare our framework's performance with several state-of-the-art schemes, the same input data construction is used in our experiments as in [12], which is summarized in Table I.

In Table II, the performance of our framework, based on the three clustering algorithms, is compared with the existing models proposed in [7], [8], [12], as well as the traditional LSTM RNN model. The table shows that three variants of our

TABLE II
COMPARISON OF THE THREE CLUSTERING METHODS BASED WITH EXISTING MODELS USING THE ISO-NE DATASET FOR 2010 AND 2011

	ISONE (SYS) 2010	ISONE (SYS) 2011
Model	MAPE	MAPE
ErrCorr modified [7]	1.75	1.98
ELM-PLSR [8]	**1.50**	1.80
DRN [12]	**1.50**	1.64
LSTM	1.58	**1.50**
K-means++-LSTM	**1.30**	**1.32**
DBSCAN-LSTM	1.37	1.34
BIRCH-LSTM	1.43	1.34

proposed framework all outperform the four baseline schemes with respect to mean absolute percentage error (MAPE), with a 13.33% reduction in MAPE in 2010 (over ELM-PLSR [8] and DRN [12]) and a 12.0% reduction in MAPE in 2011 (over LSTM). Among the three variants of our framework, the K-means++ based approach achieves the best performance.

Table III presents the performance of the first-level learner based on the K-means++ algorithm and the second-level leaner (combined model) in 2010 and 2011. In this experiment, the dataset is clustered into 8 groups, each is used to train an LSTM model. A four-neuron FCC ensemble network (see Fig. 5) is then used for model fusion. Form the table, we can see that the combined model effectively improves the performance of the first-level learners. In Year 2010, the improvements in MAPE ranges from 48.37% to 80.20%; in Year 2010, the improvements in MAPE ranges from 37.44% to 85.90%, when the FCC ensemble neural network is applied.

VII. CONCLUSIONS

In this paper, we proposed an LSTM based ensemble learning approach for short-term load forecasting. In the stacking framework, the first-level learner consisted of a set of LSTM models, each trained with a data cluster; the second-level learner consisted of an FCC ensemble neural network for model fusion. An enhanced second-order optimization algorithm was proposed to solve the ensemble problem. Superior performance over state-of-the-art schemes was demonstrated by using a real-world dataset.

TABLE III
INDIVIDUAL K-MEANS++ BASED MODEL RESULTS AND THE ENSEMBLE METHOD IMPROVEMENT FOR SYSTEM LOAD IN 2010 AND 2011

	ISONE (SYS) 2010	ISONE (SYS) 2011
Model	MAPE	MAPE
1	**2.522**	4.034
2	4.093	2.274
3	5.632	**2.102**
4	2.549	4.195
5	6.577	7.237
6	**2.522**	5.429
7	5.121	7.041
8	5.859	9.325
Combined Model	**1.302**	**1.315**

ACKNOWLEDGMENT

This work is supported in part by the NSF under Grant DMS-1736470, and through the Wireless Engineering Research and Education Center (WEREC) at Auburn University.

REFERENCES

[1] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Springer Mobile Networks and Applications Journal (MONET)*, vol. 19, no. 2, pp. 171–209, Apr. 2014.

[2] Y. Wang, G. Cao, S. Mao, and R. Nelms, "Analysis of solar generation and weather data in smart grid with simultaneous inference of nonlinear time series," in *Proc. IEEE INFOCOM 2015 Workshop on SmartCity*, Hong Kong, P.R. China, Apr. 2015, pp. 672–677.

[3] Y. Wang, Y. Shen, S. Mao, G. Cao, and R. Nelms, "Adaptive learning hybrid model for solar intensity forecasting," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1635–1645, Apr. 2018.

[4] N. Tang, S. Mao, Y. Wang, and R. Nelms, "Solar power generation forecasting with a LASSO-based approach," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1090–1099, Apr. 2018.

[5] Y. Wang, Y. Shen, S. Mao, X. Chen, and H. Zou, "LASSO & LSTM integrated temporal model for short-term solar intensity forecasting," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2933–2944, Apr. 2019.

[6] T. Strasser *et al.*, "A review of architectures and concepts for intelligence in future electric energy systems," *IEEE Trans. Ind. Electron.*, vol. 62, no. 4, pp. 2424–2438, Apr. 2015.

[7] C. Cecati, J. Kolbusz, P. Różycki, P. Siano, and B. M. Wilamowski, "A novel RBF training algorithm for short-term electric load forecasting and comparative studies," *IEEE Trans. Ind. Electron.*, vol. 62, no. 10, pp. 6519–6529, Oct. 2015.

[8] S. Li, L. Goel, and P. Wang, "An ensemble approach for short-term load forecasting by extreme learning machine," *Elsevier Appl. Energy*, vol. 170, pp. 22–29, May 2016.

[9] R. Zhang, Z. Y. Dong, Y. Xu, K. Meng, and K. P. Wong, "Short-term load forecasting of australian national electricity market by an ensemble model of extreme learning machine," *IET Generation, Transmission & Distribution*, vol. 7, no. 4, pp. 391–397, Apr. 2013.

[10] Y. Chen *et al.*, "Short-term load forecasting: Similar day-based wavelet neural networks," *IEEE Trans. Power Syst.*, vol. 25, no. 1, pp. 322–330, Feb. 2010.

[11] L. Li, K. Ota, and M. Dong, "Everything is image: CNN-based short-term electrical load forecasting for smart grid," in *Proc. ISPAN-FCST-ISCC'17*, Exeter, UK, June 2017, pp. 344–351.

[12] K. Chen, K. Chen, Q. Wang, Z. He, J. Hu, and J. He, "Short-term load forecasting with deep residual networks," *IEEE Trans. Smart Grid*, to appear.

[13] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on LSTM recurrent neural network," *IEEE Trans. Smart Grid*, vol. 10, no. 1, pp. 841–851, Jan. 2019.

[14] J. C. B. Gamboa, "Deep learning for time-series analysis," *arXiv preprint arXiv:1701.01887*, Jan. 2017.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT press, 2016.

[16] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*. Boca Raton, FL: Chapman and Hall/CRC, 2012.

[17] P. Smyth and D. Wolpert, "Linearly combining density estimators via stacking," *Springer Machine Learning*, vol. 36, no. 1/2, pp. 59–83, July 1999.

[18] E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks: A survey and empirical demonstration," *Springer Data Mining and Knowledge Discovery*, vol. 7, no. 4, pp. 349–371, Oct. 2003.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, Mar. 2015.

[20] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Trans. Nucl. Sci.*, vol. 44, no. 3, pp. 1464–1468, June 1997.

[21] H. White, "A reality check for data snooping," *Wiley Econometrica*, vol. 68, no. 5, pp. 1097–1126, Sept. 2000.

[22] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*. Burlington, MA: Morgan Kaufmann, 2011.

[23] P.-N. Tan, *Introduction to Data Mining*. Chennai, India: Pearson Education India, 2018.

[24] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008.

[25] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proc. ACM/SIAM SODA'07*, New Orleans, LA, Jan. 2007, pp. 1027–1035.

[26] Y. Rani and H. Rohil, "A study of hierarchical clustering algorithm," *International Journal of Information and Computation Technology*, vol. 3, no. 10, pp. 1115–1122, Oct. 2013.

[27] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," *ACM Sigmod Record*, vol. 25, no. 2, pp. 103–114, June 1996.

[28] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. KDD'96*, Portland, Oregon, Aug. 1996, pp. 226–231.

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

[30] D. Hunter, H. Yu, M. S. Pukish III, J. Kolbusz, and B. M. Wilamowski, "Selection of proper neural network sizes and architectures–a comparative study," *IEEE Trans Ind. Informat.*, vol. 8, no. 2, pp. 228–240, May 2012.

[31] X. Wang, X. Wang, and S. Mao, "ResLoc: Deep residual sharing learning for indoor localization with CSI tensors," in *Proc. IEEE PIMRC 2017*, Montreal, Canada, Oct. 2017, pp. 1–6.

[32] J. Martens, "Deep learning via Hessian-free optimization," in *Proc. ICML'10*, Haifa, Israel, June 2010, pp. 735–742.

[33] T. Xie, H. Yu, J. Hewlett, P. Rózycki, and B. Wilamowski, "Fast and efficient second-order method for training radial basis function networks," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 23, no. 4, pp. 609–619, Apr. 2012.

[34] J. E. Vitela and J. Reifman, "Premature saturation in backpropagation networks: mechanism and necessary conditions," *Elevier Neural Networks*, vol. 10, no. 4, pp. 721–735, June 1997.

[35] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and the Vapnik-Chervonenkis dimension," *Journal of the ACM*, vol. 36, no. 4, pp. 929–965, Oct. 1989.

[36] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from Data*. New York, NY: AMLBook, 2012, vol. 4.

[37] J. Fan, "Accelerating the modified levenberg-marquardt method for nonlinear equations," *Mathematics of Computation*, vol. 83, no. 287, pp. 1173–1187, May 2014.

[38] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," *Springer Math. Program.*, vol. 89, no. 1, pp. 149–185, Nov. 2000.

[39] J. Fan and J. Pan, "Convergence properties of a self-adaptive Levenberg-Marquardt algorithm under local error bound condition," *Springer Comput. Opt. Appl.*, vol. 34, no. 1, pp. 47–62, May 2006.

[40] ISO New England, "ISO New England Zonal Information," 2019. [Online]. Available: https://iso-ne.com/isoexpress/web/reports/pricing/-/tree/zone-info

[41] Y. Wang, N. Zhang, Y. Tan, T. Hong, D. S. Kirschen, and C. Kang, "Combining probabilistic load forecasts," *IEEE Trans. Smart Grid*, to appear.