1. Deep Recurrent Neural Network & Long-Short Term Memory

The DRNN model proposed in this study is developed to predict heat and electricity consumption values at a one-hour resolution over a medium to long-term timeline, using LSTM units. The proposed DRNN is established for two cases: (I) predicting overall heat demand of buildings at a predetermined zone in Tehran, Iran, aggregated over multiple buildings over a time horizon of two years; and (II) predicting electricity consumption in buildings over the same period.

1.1. DRNN Structure and Formulation

The energy consumption profiles of such buildings is non-linear and transient by nature; subsequently, to have a reliable forecast, a predictive model should provide the following desired features: (I) should be able to account for temporal dependencies while information about buildings' construction and operation schedules is unavailable, (II) needs to be adaptive- adaptively learn from training data without any human intervention, (III) should be able to model non-linear behavior of the loads, and (IV) should be able to account for both short and medium-to-long-term temporal dependencies within energy profiles.

Since an LSTM unit learns adaptively and models both short and long-term dependencies, a DRNN model with LSTM units is suggested in this study. Fig.1 displays the schematic of the proposed model. The proposed DRNN utilizes an encoder-decoder approach (a sequence-based architecture) to exploit the sequential nature of the loads. Using the encoder-decoder structure, a fixed vector representation would be generated by mapping the inputs using an encoder-like block; then, the vector representation converts to a particular target sequence using a decoder-like block. Now, the outputs of encoder-decoder are used as inputs for the following multi-layer perceptron (MLP) model. To exemplify variables indicating happenings on operation schedules, the DRNN layers, corresponding to the encoder-decoder layers, make transient variables in 24-h sequences, thus, allowing for the LSTM unit at each time step to model temporal dependencies. The DRNN model shares an MLP layer that receives a vector at separate time steps within the same 24-h sequence.

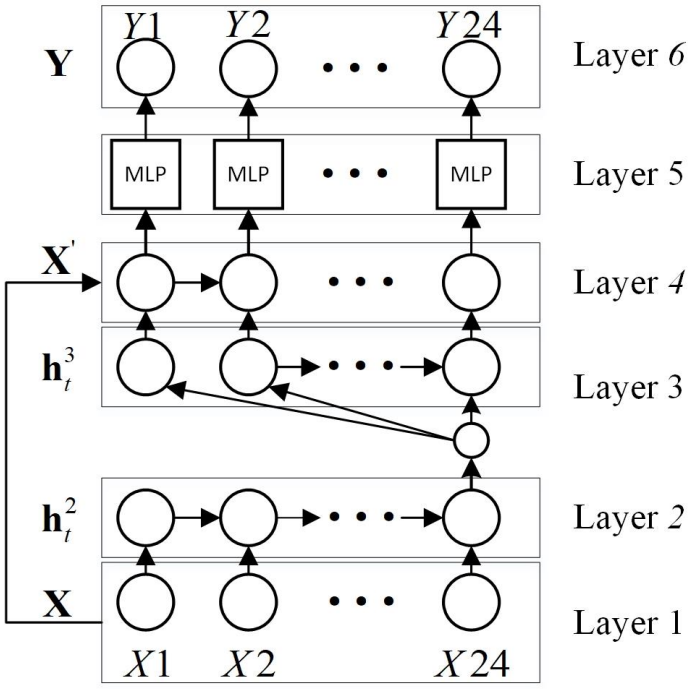

Figure 1. Architecture of the proposed DRNN. Circles' outlines indicate a vector produced at a given timestep. The circles in layer 1 represent the inputs to the model. In layers 2 and 3, each circle represent an output vector at a given timestep after passing through an LSTM activation function. Layer 4 is the concatenation layer emitting a vector at each time step *t*. Layer 5 is a shared MLP hidden layer applied to each timestep. Layer 6 is the output layer indicating the load prediction.

The input is presented in layer 1. The training set comprises *D* samples (where, in this contest, *D* is the total number of days in the training set) – each sample $X^{ts} \in R^{24\times d}$, where *d* is the dimension of input X^{ts}. Layer 2 is the first LSTM layer and is similar to the encoder layer in the encoder-decoder layer. The output at each time step *t* of this layer ($\mathbf{h}^2_t$) can be written as follows:

$$\mathbf{h}_t^2 = LSTM(\mathbf{h}_{t-1}^2, \mathbf{X}_t^{ts}); \forall t \in \{1,2,...,T\} \quad (1)$$

Layer 3: The decoder layer in the encoder-decoder model can be expressed as:

$$\mathbf{h}_t^3 = LSTM(\mathbf{h}_T^2, \mathbf{h}_{t-1}^3); \forall t \in \{1,2,...,T\} \quad (2)$$

where the fixed vector representation $\mathbf{h}^2_T$ is simply the output at the final timestep. The outputs from layer 3, h_t^3, act as surrogates for 'transient' variables, which are not explicitly known. These 'transient' variables are analogous to the schedule variables used as inputs in deterministic energy simulation models. Layer4 only consists of a single operation– a concatenation of outputs from layer 3 with the original data.

$$\mathbf{X}_t^{'} = [\mathbf{X}^{ts}; \mathbf{h}_t^3]; \forall t \in \{1,2,...,T\} \quad (3)$$

In Layers 5 and 6, the multi-layered perceptron is applied to the concatenated vector at each time-step. Thus, the prediction Y_t, for a given timestep *t* can be expressed as follows:

$$Y_t = MLP(\mathbf{X}_t^{'}); \forall t \in \{1,2,...,T\} \quad (4)$$

2. Gradient descent algorithm

To optimize the weights in each layer of the network, we used the ADAM algorithm that demonstrates a faster convergence than other stochastic gradient descent algorithms used conventionally [1]. ADAM algorithm uses a computationally efficient first-order gradient descent optimization approach, thereby making it suitable for optimizing models with a large number of parameters [2]. While other stochastic gradient descent algorithms, for example, the vanilla stochastic gradient descent, simply consider a constant learning rate in order to update the weights, ADAM compensates by considering the squared gradient and bias-corrected estimates of the moving average of the gradient. Further details on the ADAM algorithm are available in the literature [1].

3. Model regularization

When a machine learning algorithm fits the small perturbations and noise while training, the model is suffering from over-fitting, which, in practice, results in the algorithm's prediction error that is drastically larger than the training error. To avoid this issue, the following regularization approaches are implemented in this investigation:

3.1. Early-Stopping

In general, the training practice should be ceased before the maximum number of iterations are completed. This could be done through monitoring the validation loss [3]. As such, the model is said to be over-fitted when the validation error begins to rise with increasing number of epochs during training (i.e., one iteration over the entire training data). We employed the early-stopping while, after a predetermined number of iterations, the validation error has not improved. Moreover, the early-stopping practice is only utilized once the training has exceeded a minimum number of epochs. In this analysis, the minimum number of epochs is selected to be 30, and the predetermined threshold number for iterations is chosen to be 20.

3.2. Weight decay regularization

Machine learning algorithms often suffer from weight vectors being too large. Beside computational burden, large vectors impose several challenges on the model- discussed in [4]. Weight decay regularization has been considered as a significant step to constrain the magnitude of the vectors, such that the weight vectors are penalized for being too large. Restricting the weight vectors, accordingly, guarantees that the layers' outputs are not as sensitive to noise as inputs of that given layer. Consequently, the error function can be stated as:

$$\varepsilon(\mathbf{w}) = (y^p(\mathbf{w}) - y^a(\mathbf{w}))^2 - \frac{\lambda}{2M}\mathbf{w}^T\mathbf{w} \quad (5)$$

here, λ is equal to its default value in Keras [5], which is 0.01. This setting (λ=0.01) makes sure that the error is penalized when ||w||≥1.

4. Hyper-Parameter Optimization

Parameters in the model, which can be calculated by optimization of the model structure itself, not optimized through the data itself, or can be assigned empirically are higher-level choices known as Hyper-parameters. The proposed DRNN model comprises the following hyper-parameters: (I) vectors' length from output layers and (II) number of surrogates for transient variables. Since selection of hyper-parameters is case-dependent (i.e. specific to different load profiles), a global technique is needed to optimize hyper-parameters applicable to all loads.

Concerning a loss function in graph-structured spaces that are implemented in the study, *Hyperopt*, a Python library to optimize hyper-parameters, is most appropriate for the optimization process. *Hyperopt* utilizes Sequential Model-based Optimization (SMBO) method using a surrogate function to estimate the function $f(\theta)$, which on this occasion, is a deep recurrent neural network. As a result, the optimization process turns into finding θ^* while maximizing the surrogate function, for an expected improvement criterion (EI). The EI principle, as expressed in (2), penalizes the score (i.e. accuracy) of the model when it exceeds the threshold γ.

$$EI_\gamma(\theta) = \int_{-\infty}^{\infty} \max(y - score, 0)\, p(score \mid \theta)\, dy \quad (6)$$

As represented, the EI criterion seeks for $score = f(\theta)$ to be lower than γ. Using the conditional probability theory, we have:

$$EI_\gamma(\theta) = \int_{-\infty}^{\infty} \max(y - score, 0)\, \frac{p(\theta \mid score)\, p(score)}{p(\theta)}\, dy \quad (7)$$

To model/optimize the EI, an approach- namely, Tree of Parzen Estimator (TPE)- is applied. In TPE, the quantity $p(\theta \mid score)$ is calculated using non-parametric probability distributions; yet, we used the TPE model only as a black-box to optimize the set of hyper-parameters in the study. Advanced mathematical analysis on SMBO and the TPE approach can be found in [6].

5. References

1 Kingma, D.P., Ba, J.: 'Adam: A method for stochastic optimization'*arXiv Prepr. arXiv1412.6980*, 2014.

2 Rahman, A., Srikumar, V., Smith, A.D.: 'Predicting electricity consumption for commercial and residential buildings using deep recurrent neural networks'*Appl. Energy*, 2018, 212, (October 2017), pp. 372–385.

3 Saha, O., Sathish, R., Sheet, D.: 'Fully Convolutional Neural Network for Semantic Segmentation of Anatomical Structure and Pathologies in Colour Fundus Images Associated with Diabetic Retinopathy'*arXiv Prepr. arXiv1902.03122*, 2019.

4 Bottou, L.: 'Large-scale machine learning with stochastic gradient descent,' in 'Proceedings of COMPSTAT'2010' (Springer, 2010), pp. 177–186

5 Gulli, A., Pal, S.: 'Deep Learning with Keras' (Packt Publishing Ltd, 2017)

6 Hutter, F., Hoos, H.H., Leyton-Brown, K.: 'Sequential model-based optimization for general algorithm configuration,' in 'International conference on learning and intelligent optimization' (Springer, 2011), pp. 507–523