Convolutional Neural Networks For Automatic State-Time Feature Extraction in Reinforcement Learning Applied to Residential Load Control

Bert J. Claessens, Peter Vrancx, Frederik Ruelens

***Abstract*—Direct load control of a heterogeneous cluster of residential demand flexibility sources is a high-dimensional control problem with partial observability. This work proposes a novel approach that uses a convolutional neural network to extract hidden state-time features to mitigate the curse of partial observability. More specific, a convolutional neural network is used as a function approximator to estimate the state-action value function or Q-function in the supervised learning step of fitted Q-iteration. The approach is evaluated in a qualitative simulation, comprising a cluster of thermostatically controlled loads that only share their air temperature, whilst their envelope temperature remains hidden. The simulation results show that the presented approach is able to capture the underlying hidden features and successfully reduce the electricity cost the cluster.**

***Index Terms*—Convolutional neural network, deep learning, demand response, reinforcement learning.**

I. Introduction

DIRECT load control of a large heterogeneous cluster of residential demand flexibility sources is a research topic that has received considerable attention in the recent literature. The main driver behind direct load control is that it can be a cost-efficient technology supporting the integration of distributed renewable energy sources in a power system. Typical applications of direct load control considered in the literature are ancillary services [1], voltage control [2] and energy arbitrage [3]. When designing a controller for a large heterogeneous cluster of residential flexibility sources, one is confronted with several challenges. A first important challenge is that most residential flexibility sources are energy constrained, which results in a sequential decision-making problem. A second challenge is the large dimensionality of the state-action space of the cluster of flexibility sources, since each source has its own state vector and control action. Furthermore, there is heterogeneity accompanying the dynamics of each state, which is intrinsically stochastic. As a consequence, a *mature* control solution needs to be scalable, adapt to the heterogeneity of the cluster and take the intrinsic uncertainty into account. An important challenge, receiving less attention in the literature is that of partial observability, in the sense that there are states that are relevant for the dynamics of the system that are not observed directly. For example in the context of building climate control, only the operational air temperature is measured, whilst the temperature of the building envelope is not [4].

B. J. Claessens is with the energy department of VITO/EnergyVille, Mol, Belgium (bert.claessens@restore.eu).

Peter Vrancx is with the AI-lab, Vrije Universiteit Brussel, Brussels, Belgium (pvrancx@vub.ac.be)

F. Ruelens is with the Department of Electrical Engineering, KU Leuven/EnergyVille, Leuven, Belgium (frederik.ruelens@kuleuven.be).

When considering energy constrained flexibility, energy storage is the most important source, either through direct electric storage or through power to heat conversion. Examples of direct electric storage are the battery in an electric vehicle [5] or a stand alone domestic battery. Thermostatically Controlled Loads (TCLs) [1] are an important example of storage through power to heat conversion, e.g. building climate control [4] and domestic hot water storage [1]. As TCLs are an abundant source of flexibility, they are the focus of this work [6]. When designing a controller for a large cluster of TCLs one not only faces technical challenges from a control perspective. One should also take into account the economic cost of the control solution at the level of individual households as the economic potential per household is limited and on the order of €50 a year [2], [6]. A popular control paradigms found in the recent literature regarding residential load control is model-based control such as Model Predictive Control (MPC) [7]. In MPC, control actions are selected at fixed time intervals by solving an optimization problem with a finite time horizon, following a receding horizon approach. The optimization problem includes a model and constraints of the system to be controlled and forecasts of the exogenous information, such as user behaviour, outside temperature and solar radiance. When controlling a large cluster of flexibility sources however, solving the optimization problem centralised quickly becomes intractable [8]. To mitigate these scalability issues, distributed optimization techniques provide relief by decomposition of the master problem into sub-problems. Another approach (*aggregate-and-dispatch*) gaining interest is to use a *bulk* model of reduced order making the centralised optimization tractable by defining a setpoint for the entire cluster. Dissaggregation of the setpoint occurs through a heuristic dispatch strategy.

The mathematical performance of the aforementioned approaches however, is directly related to the fidelity of the model used in the optimization problem [3], [9]. Obtaining and maintaining an accurate model, is a non-trivial task [10] where the cost of obtaining such a model can outweigh its financial benefits.

Given this context, model-free control solutions are considered a valuable alternative or addition to model-based control solutions [11]. The most popular model-free control paradigm is Reinforcement Learning (RL) [12]. For example when using Q-learning, an established form of RL, a control policy is learned by estimating a state-action value function, or Q-function based upon interactions with the system to be controlled. In [13], [14], [15], RL has been been applied to the setting of residential demand response at the device level, whilst in [14] and [16], RL has been used in an *aggregate-and-dispatch* setting with a large cluster of TCLs.

In the literature, different approaches are presented that obtain an estimate of the Q-function. In this work, as in [17], batch RL [18] is used, where an estimate of the Q-function is obtained offline using a batch of historical tuples. A regression algorithm is used to generalize the estimate of the Q-function to unobserved state-action combinations. In [14], extra-trees [18] has been used as a regression algorithm in combination with hand-crafted features, furthermore perfect state information was assumed. This resulted in a low-dimensional state space, evading the curse of dimensionality. A next important step is to add automatic feature extraction as this enables to work with higher dimensional state representations. This in turn, allows to add historic observations to the state, following Bertsekas [19], as it can compensate for the partial observability of the state by extracting state-time features. Recent developments in the field of RL, and more specific deep reinforcement learning [20], have demonstrated how by using a Convolutional Neural Network (CNN) automatic feature extraction can be obtained in a high-dimensional state space with local correlation. Inspired by these findings, this work applies deep reinforcement learning to the setting of *aggregate-and-dispatch* with a high-dimensional state space, which includes past observations, allowing to extract state-time features, thus mitigating the effect of incomplete state information.

The remainder of this work is summarized as follows. In Section II an overview of the related literature is provided and the contributions of this work are explained. Section III sketches the main motivation behind this paper. Following the approach presented in [17], in Section IV a Markov decision process formulation is provided. In Section V, the implementation of the controller is detailed, while Section VI presents a quantitative and qualitative assessment of its performance. Finally, Section VII outlines the conclusions and discusses future research.

II. Related Work and Contribution

This section provides a non-exhaustive overview of related work regarding the control of heterogeneous clusters of TCLs, batch RL applied to load control and automated feature extraction.

A. Aggregate and dispatch

As mentioned in Section I, two important challenges in model-based control of a cluster of TCLs are the dimensionality of the state-action space and obtaining a high-fidelity model. To mitigate these challenges, two important *schools* can be identified (amongst others), i.e. that of distributed optimization and *aggregate-and-dispatch*. The general concept behind aggregate-and-dispatch techniques [1], [4] is to use a bulk model representing the dynamics of the cluster of TCLs instead of individual modelling at TCL level. Typically, this bulk model is of a reduced dimensionality, making a centralised MPC approach tractable. The subsequent set-points are dispatched at device level by using a simple heuristic, requiring little intelligence at device level. For example in [16], the TCLs are clustered based upon their relative position within their dead-band, resulting in a state vector at cluster level containing the fraction of TCLs in each state bin. A linear state bin transition model describes the dynamics of this state vector, the dimensionality of which is independent of the number of TCLs in the cluster. This model is in turn used in an MPC, resulting in a control signal for each state bin. A simple heuristic is used to dispatch the control signals to individual control actions at device level. The results presented in [3] show that careful system identification is required and a generic *tank* model is preferred for example for energy arbitrage. Moreover, in [21] it is argued that the first order TLC model presented in [3] needs to be extended with a representative *bulk* temperature that is not directly observable, necessitating a model-based state estimation such as a Kalman filter [8]. Also in[22], a low-order tank model is used as a bulk model to describe the flexibility of a large cluster of TCLs allowing for tractable stochastic optimization. A kindred approach is presented by Iacovella *et al.* in [4]. Here, a small set of representative TCLs are identified to model the dynamics of the entire cluster. Through this, the heterogeneity of cluster is accounted for and the auction-based dispatch dynamics can be included in the central optimization. This work is an extension of the work presented in [5] where a tank model has been used for a cluster of electric vehicles, again in combination with an auction-based dispatch strategy. The main advantage of aggregate-and-dispatch is that it mitigates the curse of dimensionality allowing to hedge against uncertainty at a centralised level [22] and requires little and transparent intelligence at the level of a TCL, restricting the local cost. A system identification step at centralised level however, is still required.

A different paradigm is that of distributed optimization [23], [24], where the centralised optimization problem is decomposed over distributed agents who interact iteratively through *virtual* prices which are the Lagrangian multipliers related to coupling constraints. For example in [24], distributed MPC through dual decomposition was presented as a means for energy arbitrage of a large cluster of TCLs subject to a coupling constraint related to an infrastructure limitation. Although distributed optimization techniques converge to a global optimum under sufficient conditions [25], the technical implementation is not straightforward. This results from the need for a system identification step for each sub-problem (often at the level of a TCL) and the fact that on the order of ten iterations are necessary before convergence is obtained. Although the merits of distributed optimization are recognized, this work focuses on aggregate-and-dispatch techniques in the scope of residential flexibility, as the local intelligence at device level is simple and transparent.

B. Reinforcement Learning for demand response

As discussed in Section I, RL is a model-free control technique whereby a control policy is learned from interactions with its environment. When integrated in an aggregate-and-dispatch approach, it allows to replace or assist a model-based controller [15]. This paves the way for generic control solutions for residential demand response. When considering RL applied to aggregate-and-dispatch techniques, Kara *et al.* applied Q-learning [16] to the binning method presented in [1]. In [14], Ruelens *et al.* applied batch RL in the form of Fitted Q-Iteration (FQI) to a cluster of TCLs using an auction-based dispatch technique effectively learning the dynamic of the cluster, including uncertainty and effects of the dispatch strategy. A second implementation is presented in [26], where FQI was used to obtain an accurate day-ahead consumption schedule for a cluster of electric vehicles. The focus was on finding a day-ahead schedule that results in the lowest electricity cost considering day-ahead and intra-day electricity prices. Although the results demonstrated that RL is of interest for demand response, the state dimensionality was small and the features in the state were handcrafted, furthermore full observability was assumed. This is a limitation when considering a very heterogeneous cluster and partial observability. In this setting, a richer state description is required, e.g. as in [1] using a state bin distribution. Furthermore, following [19] the state vector needs to be extended to include previous observations as it allows to extract state-time features that can be representative for non-observable state information. This however, requires automatic high-dimensional feature extraction.

C. High-Dimensional Feature Extraction

As mentioned in Section I, recent results show that deep approximation architectures such as CNNs can be used as a regression algorithm in RL applied to a problem with a high-dimensional state vector. Artificial neural networks offer an attractive option for value function approximation. They can approximate general nonlinear functions, scale to high-dimensional input spaces and can generalize to unseen inputs. Furthermore, deep network architectures stack multiple layers of representations and can be used with low level sensor inputs (e.g. image pixels) to learn multiple levels of abstraction and reduce the need to manually define features.

Unfortunately, when used with sequential updating and correlated observations, as is typical in online reinforcement learning, neural networks can suffer from issues such as divergence of the estimates or catastrophic forgetting [27], [28]. The FQI algorithm [29] used in this paper, sidesteps this problem by relying on offline approximation of the value function using batch training of the function approximators. It has previously been applied to a range of control applications [30], [31], [32]. Additionally, FQI was successfully combined with deep architectures by Lange *et al.* [33], [34], who extended the algorithm using deep autoencoders to learn features from image pixel inputs. Another approach to combine neural networks with RL was recently proposed by Mnih *et al.* [20]. Here a database of transition samples is used with experience replay [35] to break correlations in the training set in online value function approximation [36]. As is the case in this paper, the proposed deep Q-network (DQN) algorithm uses CNN architecture to map low level inputs to Q-values. Following this result a number of approaches combining reinforcement learning with CNNs have been proposed. Guo *et al.* [37] combine a DQN agent with offline planning agents for sample generation. In [38], Lillicrap *et al.* use the DQN architecture in an actor-critic setting with continuous action spaces. Finally, Levine *et al.* [39] introduce a different approach using a CNN to represent policies in a policy search method. In this paper, we combine a CNN and a multilayer perceptron to approximate Q-values in the batch FQI setting. It offers the following contributions:

- A merged Artificial Neural Network (ANN), comprising a CNN [20] and a multilayer perceptron, tailored to a demand response setting, is used as a regression algorithm within FQI. To the best of our knowledge, this work is the first description of such a network to be used in combination with a batch reinforcement learning algorithm.
- By presenting the CNN with a series of state-bin distributions, state-time features that are relevant to learn near-optimal policies can be extracted.
- The resulting control strategy is evaluated on a simplified and qualitative test scenario comprising a heterogeneous cluster of TCLs with partial observability, exposed to a time varying price. The results demonstrate that the presented approach can be successfully applied to residential load control.

III. Background and Motivation

As detailed in [21], [40], [41], the dynamics of a Thermostatically Controlled Load (TCL) is dominated by at least two time scales, a fast one (related to the operational air temperature) and a slow one (related to the building mass). A detailed description of the second-order dynamics of a TCL can be found in Section VI-A. This model describes the temperature dynamics of the indoor air and of the building envelope. Typically, only the operational air temperature is available from which all information needs to be extracted. In a model-based implementation all non-observable states are determined using a Kalman filter [41]. However, before one can implement this filter, one first needs a calibrated model. This is typically a non-linear optimization problem as presented in [41]. In a model-free approach, information regarding the non-observable states needs to be extracted from the last N observations. This results in a severe extension of the state space. Driven by this challenge, this paper combines a batch RL technique with a convolutional neural network to make up for the partial observability of the problem.

IV. Problem Formulation

Before presenting the control approach in Section V, the decision-making process is formulated as a Markov Decision Process (MDP) [19] following the procedure presented in [17]. An MDP is defined by its state space X, its action space U, and its transition function f:

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k), \tag{1}$$

which describes the dynamics from $\boldsymbol{x}_k \in X$ to $\boldsymbol{x}_{k+1}$, under the control action $\boldsymbol{u}_k \in U$, and subject to a random process $\boldsymbol{w}_k \in W$, with probability distribution $p_w(\cdot, \boldsymbol{x}_k)$. The cost c_k accompanying each state transition is defined by:

$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{x}_{k+1}) = \rho(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k). \tag{2}$$

The objective is to find a control policy $h: X \rightarrow U$ that minimizes the T-stage cost starting from state $\boldsymbol{x}_1$, denoted by $J^h(\boldsymbol{x}_1)$:

$$J^h(\boldsymbol{x}_1) = \mathbb{E}\left(R^h(\boldsymbol{x}_1, \boldsymbol{w}_1, ..., \boldsymbol{w}_T)\right), \tag{3}$$

with:

$$R^h(\boldsymbol{x}_1, \boldsymbol{w}_1, ..., \boldsymbol{w}_T) = \sum_{k=1}^{T} \rho(\boldsymbol{x}_k, h(\boldsymbol{x}_k), \boldsymbol{w}_k). \tag{4}$$

A convenient way to characterize the policy h is by using a state-action value function or Q-function:

$$Q^h(\boldsymbol{x}, \boldsymbol{u}) = \underset{w \sim p_{\mathcal{W}}(\cdot|x)}{\mathbb{E}} \left[\rho(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) + J^h(f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}))\right]. \tag{5}$$

The Q-function is the cumulative return starting from state $\boldsymbol{x}$, taking action $\boldsymbol{u}$, and following h thereafter.

The optimal Q-function corresponds the best Q-function that can be obtained by any policy:

$$Q^*(\boldsymbol{x}, \boldsymbol{u}) = \min_h Q^h(\boldsymbol{x}, \boldsymbol{u}). \tag{6}$$

Starting from an optimal Q-function for every state-action pair, the optimal policy is calculated as follows:

$$h^*(\boldsymbol{x}) \in \arg\min_{\boldsymbol{u} \in U} Q^*(\boldsymbol{x}, \boldsymbol{u}), \tag{7}$$

where Q^* satisfies the Bellman optimality equation [42]:

$$Q^*(\boldsymbol{x}, \boldsymbol{u}) = \underset{\boldsymbol{w} \sim p_{\mathcal{W}}(\cdot|x)}{\mathbb{E}} \left[\rho(\boldsymbol{x}, \mathbf{u}, \boldsymbol{w}) + \min_{u' \in U} Q^*(f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}), \boldsymbol{u}')\right]. \tag{8}$$

A. State description

Following [17], the state space X consists of: time-dependent state information X_{t}, controllable state information X_{phys}, and exogenous (uncontrollable) state information X_{ex}.

Since the problem of scheduling a TCL includes time dependence, i.e. the system dynamics are non-stationary, it is important to include at time-dependent state component to capture these patterns. As in [15], the time-dependent state component X_{t} contains information related to timing. In this work, this component contains the hour of the day:

$$x_{\mathrm{t}} \in X_{\mathrm{t}} = \{1, \ldots, 24\}. \tag{9}$$

By adding a time-dependent state component to the state vector, the learning algorithm can capture the behavioral patterns of the end users. The rationale is that most consumer behavior tends to be repetitive and tends to follows a diurnal pattern.

The controllable state information $\boldsymbol{x}_{\mathrm{phys},k}$ comprises the operational temperature T_k^i of each TCL $i \in \mathcal{D}$:

$$\underline{T}_k^i < T_k^i < \overline{T}_k^i \tag{10}$$

where $\underline{T}_k^i$ and $\overline{T}_k^i$ denote the lower and upper bound set by the end user.

The exogenous state information $\boldsymbol{x}_{\mathrm{ex},k}$ cannot be influenced by the control action $\boldsymbol{u}_k$, but has an impact on the physical dynamics. In this work, the exogenous information comprises the outside temperature $T_{\mathrm{o},k}$. A forecast of the outside temperature $\hat{T}_{\mathrm{o}}$ [1] is assumed available when calculating the Q-function, as detailed in Section V-B1.

The observable state vector $\boldsymbol{x}_k^{\mathrm{obs}}$ of the cluster is defined as:

$$\boldsymbol{x}_k^{\mathrm{obs}} = \left(x_{\mathrm{t},k}, T_k^1, \ldots, T_k^{|\mathcal{D}|}, T_{\mathrm{o},k}\right). \tag{11}$$

B. Backup controller and physical realisation

The control action for each TCL is a binary value indicating if the TCL needs to be switched ON of OFF:

$$u_k^i \in \{0, 1\}. \tag{12}$$

Similar as in [1] and [17], each TCL is equipped with a backup controller, acting as a filter for the control action resulting from the policy h. At each time step k, the function B maps the requested control action u_k^i of device i to a physical control action $u_k^{\mathrm{phys},i}$, depending on its indoor air temperature T_k^i:

$$u_k^{\mathrm{phys},i} = B(T_k^i, u_k^i, \boldsymbol{\theta}^i), \tag{13}$$

where $\boldsymbol{\theta}^i$ contains the minimum and maximum temperature boundaries, $\underline{T}_k^i$ and $\overline{T}_k^i$ set by the end user and $B(\cdot)$ is defined as:

$$B(T_k^i, u_k^i, \theta^i) = \begin{cases} 1 & \text{if} \quad T_k^i \leq \underline{T}_k^i \\ u_k^i & \text{if} \quad \underline{T}_k^i \leq T_k^i \leq \overline{T}_k^i. \\ 0 & \text{if} \quad T_k^i > \overline{T}_k^i \end{cases} \tag{14}$$

The backup controller guarantees the comfort settings of the end user by overruling the requested control action u_k^i when the comfort constraints of the end user are violated. For example, if the temperature of TCL i drops below $\underline{T}_k^i$ the backup controller will activate the TCL, independent of the requested control action, resulting in $u_k^{\mathrm{phys},i}$, which is needed to calculate the cost (15).

C. Cost function

Different objectives are considered in the literature when controlling a large cluster of TCLs, for example, tracking a balancing signal [1] or energy arbitrage [3]. In this work, we consider energy arbitrage, where TCLs can react to an external price vector $\boldsymbol{\lambda}$. The cost function ρ is defined as:

$$\rho\left(\boldsymbol{x}_k^{\mathrm{obs}}, u_k^{\mathrm{phys},1}, \ldots, u_k^{\mathrm{phys},|\mathcal{D}|}, \lambda_k\right) = \Delta t \lambda_k \sum_{i=1}^{|\mathcal{D}|} u_k^{\mathrm{phys},i} P^i, \tag{15}$$

where P^i is the average power consumption of the ith TCL during time interval Δt and λ_k is the electricity price during time step k.

[1] The notation $\hat{x}_{\mathrm{ex}}$ is used to indicate a forecast of the exogenous state information x_{ex}.

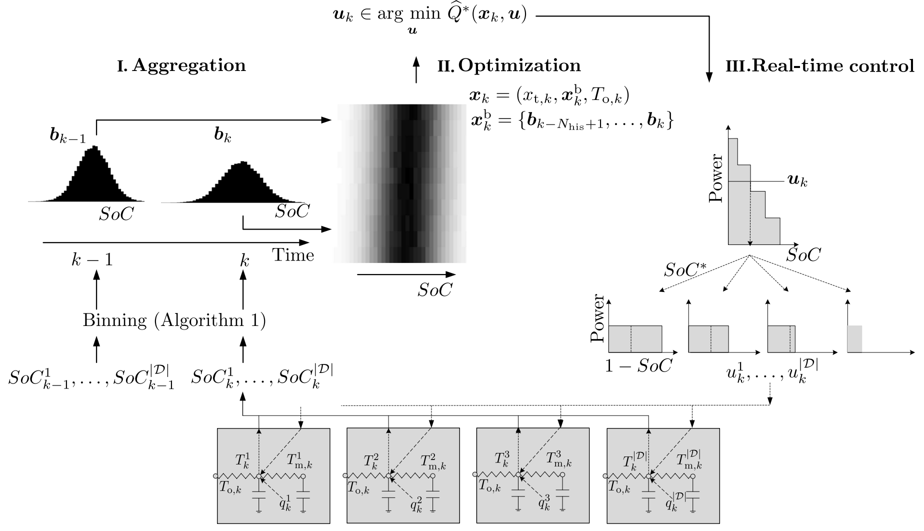
$u_k \in \arg\min_u \widehat{Q}^*(x_k, u)$
I. Aggregation
II. Optimization
III. Real-time control
$x_k = (x_{t,k}, x_k^b, T_{o,k})$
$x_k^b = \{b_{k-N_{his}+1}, \dots, b_k\}$
b_{k-1}
b_k
SoC
Time
$k-1$
k
Binning (Algorithm 1)
$SoC_{k-1}^1, \dots, SoC_{k-1}^{|\mathcal{D}|}$
$SoC_k^1, \dots, SoC_k^{|\mathcal{D}|}$
Power
u_k
SoC^*
$1 - SoC$
$u_k^1, \dots, u_k^{|\mathcal{D}|}$
T_k^1
$T_{m,k}^1$
$T_{o,k}$
q_k^1

Fig. 1: Three steps are identified, i.e. aggregation, optimization and real-time control. The models indicated at the bottom only provide access to the room temperature T, whilst the temperature of the building mass T_{m} is hidden.

Algorithm 1 Calculate the binning vector $\boldsymbol{b}_k$.

```
Input: $\boldsymbol{x}_k^{\mathrm{obs}}, \boldsymbol{b}_{\mathrm{s}}$
1: let $\boldsymbol{b}_k$ be zeros everywhere on $\mathbb{N}^{|\boldsymbol{b}_{\mathrm{s}}|}$
2: for $i = 1, \dots, |\mathcal{D}|$ do
3:     $SoC_k^i = \frac{T_k^i - \underline{T}_k^i}{\overline{T}_k^i - \underline{T}_k^i}$
4:     $j^* = \arg\max_j b_{\mathrm{s},j}$
5:         s.t. $b_{\mathrm{s},j} \leq SoC_k^i$
6:     $b_{k,j^*} \leftarrow b_{k,j^*} + 1$
7: end for
Output: $\boldsymbol{b}_k$
```

V. Implementation

In this section the implementation details of the presented controller are described. Similar as in [1] and [5] a three-step approach is used (Fig. 1).

A. Step 1: Aggregation

In the first *aggregation* step, an aggregated state representation is created from N_{his} historical observations $\boldsymbol{x}_{k-N_{\mathrm{his}}+1}^{\mathrm{obs}}, \dots, \boldsymbol{x}_k^{\mathrm{obs}}$. Each observation $\boldsymbol{x}_k^{\mathrm{obs}}$ is processed similar as in [1], i.e. each TCL in the state vector $\boldsymbol{x}_k^{\mathrm{obs}}$ is binned according to its state of charge (SoC_k^i) in the binning vector $\boldsymbol{b}_k$ with support points $\boldsymbol{b}_{\mathrm{s}}$. The vector $\boldsymbol{b}_{\mathrm{s}}$ contains N_{bin} equidistant points between the minimum and maximum state of charge of the cluster. For each TCL in the cluster, Algorithm 1 calculates the corresponding SoC_k^i (line 3) and allocates this SoC_k^i within the corresponding state of charge interval (line 4 and 5), indexed by $j \in \{1, \dots, N_{\mathrm{bin}}\}$.

In a second step, the binning vectors of subsequent time steps are concatenated, resulting in $\boldsymbol{x}_k^{\mathrm{b}} \in \mathbb{R}^{N_{\mathrm{bin}} \times N_{\mathrm{his}}}$:

$$\boldsymbol{x}_k^{\mathrm{b}} = \{\boldsymbol{b}_{k-N_{\mathrm{his}}+1}, \dots, \boldsymbol{b}_k\}, \tag{16}$$

where N_{his} denotes the number of historical time steps included in $\boldsymbol{x}_k^{\mathrm{b}}$. As a result, the final state vector is defined as:

$$\boldsymbol{x}_k = \left(x_{\mathrm{t},k}, \boldsymbol{x}_k^{\mathrm{b}}, T_{\mathrm{o},k}\right). \tag{17}$$

B. Step 2: Batch Reinforcement Learning

In the second step, a control action for the entire cluster is determined following (7). In this work, FQI is used to obtain an approximation $\widehat{Q}^*$ of the state-action value function Q^* from a batch of four tuples $\mathcal{F}$, as detailed in [18]:

$$\mathcal{F} = \{(\boldsymbol{x}_l, u_l, \boldsymbol{x}_l', c_l), \, l = 1, ..., \#\mathcal{F}\}, \tag{18}$$

1) Fitted Q-Iteration: Building upon recent results [17], [15], FQI is used to obtain $\widehat{Q^*}(\boldsymbol{x}, u)$. The cost function is assumed known (15) and the resulting actions of the backup controller can be measured. As a consequence, Algorithm 2 uses tuples of the form $\left(\boldsymbol{x}_l, u_l, \boldsymbol{x}_l', u_l^{\mathrm{phys}}\right)$. Here $\boldsymbol{x}_l'$ denotes the successor state to $\boldsymbol{x}_l$. To leverage the availability of forecasts, in Algorithm 2 the observed exogenous information (outside temperature) in $\boldsymbol{x}_{\mathrm{ex},l}'$ is replaced by its forecasted value $\hat{\boldsymbol{x}}_{\mathrm{ex},l}'$ (line 5 in Algorithm 2). In step 6, a neural network is used $|U|$ times to determine the minimum value of the current approximation of the Q-function $\widehat{Q}_{N-1}(\hat{\boldsymbol{x}}_l', .)$. In step 8, the neural network, used in step 6, is trained using all tuples $(\boldsymbol{x}_l, u_l)$ as input and all Q-values $Q_{N,l}$ as output data.

In our previous work [14], [26], an ensemble of extremely randomized trees [18] was used as a regression algorithm to estimate the Q-function. Given the high dimensionality of the state ($\boldsymbol{x}_k^{\mathrm{b}} \in \mathbb{R}^{N_{\mathrm{bin}} \times N_{\mathrm{his}}}$) and given that state-time features are expected to have strong local correlations, this work proposes an artificial neural network with a convolutional component.

2) Regression Algorithm: The parametrization of the Q-function is given by a neural network architecture consisting of two subcomponents: a convolutional component and a standard multi-layer perceptron. The full architecture is shown in Fig. 2. The network takes as input a state action pair $(\boldsymbol{x}, u)$ and returns an approximated Q-value $\widehat{Q}^*(\boldsymbol{x}, u)$. The inputs of the neural network are split into two parts. The first part contains a $N_{\mathrm{bin}} \times N_{\mathrm{his}}$ grid corresponding to the binned state representations $\boldsymbol{x}^{\mathrm{b}}$ and the second part contains the time-dependent state information x_{t}, the exogenous state information $\boldsymbol{x}_{\mathrm{ex}}$ and the action u_k. The binned state representation $\boldsymbol{x}^{\mathrm{b}}$

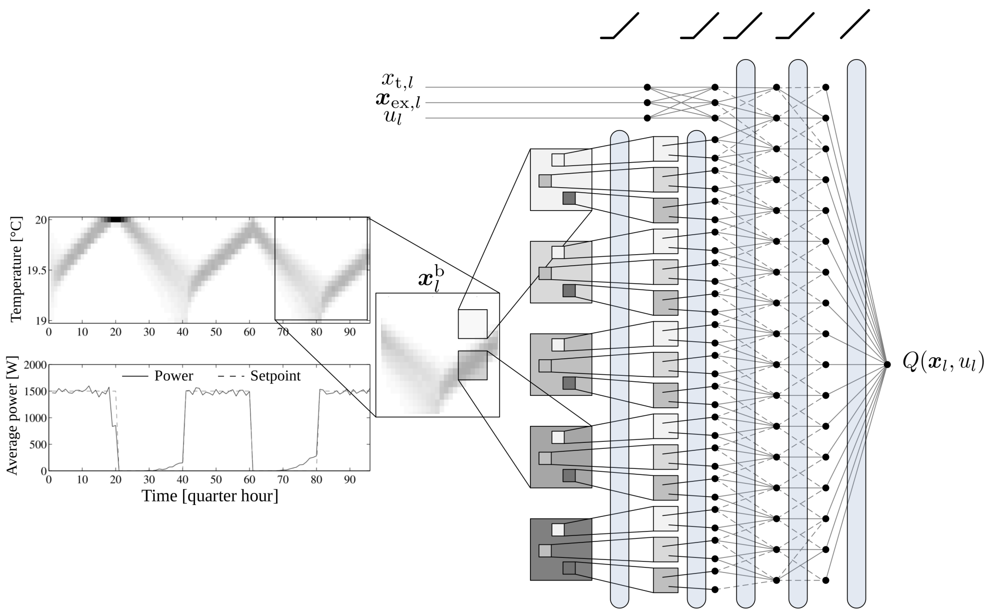

Fig. 2: Overview of the regression step as used in the fitted Q-iteration implementation (line 8 in Algorithm 2). The controllable state information, represented in the form of a matrix, goes into two convolutional layers that identify state-time features. The other parts of the state, i.e. time-dependent and exogeneous state information together with the control action go through a dense neural network where also features can be extracted. Finally both layers are merged followed by two fully connected layers.

Algorithm 2 Fitted Q-iteration using a convolutional neural network to extract state-time features.

```
Input: $\mathcal{F} = \{\boldsymbol{x}_l, u_l, \boldsymbol{x}'_l, u_l^{\text{phys}}\}_{l=1}^{\#\mathcal{F}}, \hat{X}_{\text{ex}} = \{\hat{\boldsymbol{x}}_{\text{ex},k}\}_{k=1}^{T}, \boldsymbol{\lambda}$
1: let $\widehat{Q}_0$ be zero everywhere on $X \times U$
2: for $N = 1, \ldots, T$ do
3:     for $l = 1, \ldots, \#\mathcal{F}$ do
4:         $c_l \leftarrow \rho(\boldsymbol{x}_l, u_l^{\text{phys}}, \boldsymbol{\lambda})$
5:         $\hat{\boldsymbol{x}}'_l \leftarrow (x'_{\text{t},l}, \boldsymbol{x}_l^{\text{b}\prime}, \hat{\boldsymbol{x}}'_{\text{ex},l})$
6:         $Q_{N,l} \leftarrow c_l + \min_{u \in U} \widehat{Q}_{N-1}(\hat{\boldsymbol{x}}'_l, u)$
7:     end for
8:     use the convolutional neural network in Fig. 2 to obtain
       $\widehat{Q}_N$ from $\mathcal{T} = \{((\boldsymbol{x}_l, u_l), Q_{N,l}), l = 1, \ldots, \#\mathcal{F}\}$
9: end for
Output: $\widehat{Q}^* = \widehat{Q}_N$
```

is processed using a CNN. CNNs process inputs structured as a 2-dimensional grid (e.g. images, video) by convolving the input grid with multiple linear filters with learned weights. In this way, CNNs can learn to detect spatial features in the local structure of the input grid. A convolutional layer consists of multiple filters W^k, each giving rise to an output *feature map*. The feature map h^k corresponding to the kth filter weight matrix W^k is obtained by:

$$h_{ij}^k = \sigma(W^k * x)_{ij} + b^k, \tag{19}$$

where $*$ represents a 2d convolution operation, x are the layer inputs, b^k is a bias term and σ is a nonlinear activation function. Multiple layers can be stacked to obtain a deep architecture. Convolutional layers are often alternated with pooling layers that downsample their inputs to introduce an amount of translation invariance into the network. Convolutional and pooling layers are followed by fully connected layers that combine all the feature maps produced by the convolutional part and produce the final network outputs. The CNN processes the binned $\boldsymbol{x}^{\text{b}} \in \mathbb{R}^{N_{\text{bin}} \times N_{\text{his}}}$ with 1 dimension of the input grid corresponding to the N_{bin} bins and the other dimension representing observations at N_{his} previous time steps. Time and state dimensions are treated equally and 2d convolution operations are applied over both these dimensions. This results in the identification of spatio-temporal features that identify local structure in the state information and history. This enables the network to identify features corresponding to events that occur over multiple time steps. These features are then used as input by higher network layers. The time-dependent state information $x_{\text{t},k}$, exogenous input values $\boldsymbol{x}_{\text{ex},k}$ and actions u_k are fed into a separate fully-connected feedforward architecture. This multi-layer perceptron first maps the inputs to an intermediate hidden representation. This hidden representation is then combined with the CNN output and both networks are merged into fully connected layers. A final linear output layer maps the combined hidden features of both networks to the predicted Q-value of the input state-action pair. This 2-stream network structure that combines different sources of information is similar to the organisation used in supervised problems with multimodal inputs [43].

C. Step 3: Real-time control

In the third step, a control action for the entire cluster u_k is selected using an ε-greedy strategy, where the exploration probability is decreased on a daily basis according to a harmonic sequence [12]. Following [5], u_k is dispatched over the different TCLs using an auction-based multi-agent system. In this market, each TCL is represented by a bid function

f^i_{bid}, which defines the power consumed versus a heuristic p_r, resulting in the following expression for each TCL i:

$$f^i_{\text{bid}}(p_\text{r}) = \begin{cases} P^i & \text{if} \quad 0 < p_\text{r} \leq p^i_\text{c} \\ 0 & \text{if} \quad\quad p_\text{r} > p^i_\text{c} \end{cases}, \tag{20}$$

where p^i_c is the corner priority. The corner priority indicates the wish (priority) for consuming at a certain power rating P^i. The closer the state of charge drops to zero, the more urgent its scheduling (high priority), the closer to 1, the lower the scheduling priority. The corner priority of the ith TCL is given by $p^i_\text{c} = 1 - SoC^i$, where SoC^i is the state of charge of TCL i. At the aggregated level, a clearing process is used to translate the aggregated control action u_k to a clearing priority $p^*_{\text{r},k}$:

$$p^*_{\text{r},k} \; = \; \arg\min_{p_\text{r}} \left| \sum_{i=1}^{|\mathcal{D}|} f^i_{\text{bid}}(p_\text{r}) - u_k \right|. \tag{21}$$

Note, in (21) the clearing priority $p^*_{\text{r},k}$ is found by matching the aggregated control action u_k in the aggregated bid function of the cluster. This clearing priority $p^*_{\text{r},k}$ is sent back to the different TCLs, who start consuming according $u^i_k = f^i_{\text{bid}}(p^*_{\text{r},k})$.

VI. Results

In order to evaluate the functionality of the controller presented in Section V, a set of numerical experiments were performed on a qualitative scenario, the results of which are presented here. The simulation scenario comprises a cluster of 400 TCLs exposed to a dynamic energy price [44]. The thermal inertia of each TCL is leveraged as a source of flexibility allowing the electric demand to be shifted towards moments when the energy price is low. A backup controller (14) deployed at each TCL safeguards the comfort constraints. In the following simulation experiments, we define a control period of 1 hour and an optimization horizon of 24 control periods. At the start of each optimization horizon Algorithm 2 is used to compute a control policy for the next 24 control periods. This control policy is updated every 24 hours using a new price profile and forecast of the outside temperature. During the day, an ε-greedy exploration strategy is used to interact with the environment and to collect new transitions that are added systematically to the given batch. Since more interactions result in a better coverage of the state-action space, the exploration probability ε_d is decreased on a daily basis, according to the harmonic sequence $1/d^n$, where n is set to 0.7 and d denotes the current day.

A. Simulation model

Following [4], [21], a second-order model has been used to describe the dynamics of each building as illustrated in Fig. 1:

$$\begin{aligned} \dot{T}^i &= \tfrac{1}{C^i_\text{a}}\left(T_\text{o} - T^i\right) \quad + \tfrac{1}{C^i_\text{m}}\left(T^i_\text{m} - T^i\right) + P^i u^i + q^i, \\ \dot{T}^i_\text{m} &= \tfrac{1}{C^i_\text{m}}\left(T^i - T^i_\text{m}\right), \end{aligned} \tag{22}$$

where T^i is the measured indoor air temperature and T^i_m is the not observable building mass temperature. For each TCL in the simulation, the values $1/C^i_\text{a}$ and $1/C^i_\text{m}$ are selected random from a normal distributions $\mathcal{N}(0.004, 0.0008)$ and

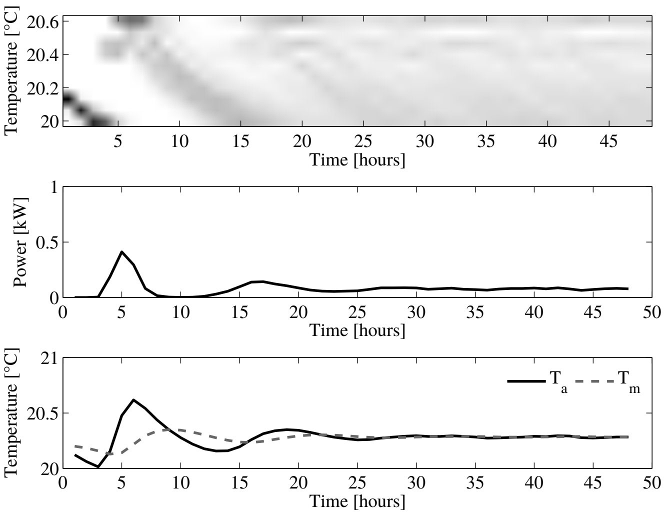

Fig. 3: Top graph, evolution of the distribution of the TCL population over time. Middle graph, the average power evolving over time, after about 30 hours full decoherence is observed. Lower graph, the average observable and non-observable temperature as a function of time.

$\mathcal{N}(0.2, 0.004)$ respectively. The internal heating q is sampled from $\mathcal{N}(0, 0.01)$ for each time step. The power P^i is set to 0.5 kW for each TCLs and the minimum and maximum temperatures are set at 20 and 22°C for each TCL. To illustrate the effect of the heterogeneity of parameters, Fig. 3 depicts the temperature dynamics of 1000 TCLs where a backup-back controller [15] is used. The top graph in Fig. 3 shows the evolution of the temperature distribution, initially all TCLs have the same state, however, after one day de-phasing has occurred which is a direct measure for the heterogeneity of the cluster. The middle graph shows the aggregated power as a function of time. The initial coherence, resulting in strongly correlated consumption, is gone after around 30 hours. The lower graph shows the average values of T and T_m respectively. As discussed in Section V, only T, is assumed available, whilst features representing T_m are inferred from past measurements of T by the convolutional section in the regression algorithm. The coefficients of the model were chosen as such that C_m has a small but significant effect on the objective.

B. Theoretical benchmark

An optimal solution of the considered control problem is found by using a mathematical solver [45]. The objective of the benchmark is to minimize the electricity cost of the cluster:

$$\min \sum_{i=1}^{|\mathcal{D}|} \sum_{t=1}^{T} \lambda_t u^{\text{phys},i}_t P^i \tag{23}$$

subject to the second-order models (22) and comfort constraints (14) of the individual TCLs in the cluster. Note, the benchmark optimization has *perfect* information about the model and has *full* access to the temperature of the building mass, resulting in a mathematical optimal result. This result can be seen as a lower limit on the cost, indicating how far from the optimum the controller is.

To demonstrate the impact of ignoring the non-observable state T_{m} on the objective, the theoretical benchmark optimization was performed for coefficients corresponding to the mean of previous normal distributions. Ignoring the non-observable state T_{m} resulted in a cost increase of 2.5%.

C. Deep regression architecture

This subsection describes the exact architecture of the neural network depicted in Fig. 2 used during the simulations. A fragment of the Python code of the neural network can be found in the appendix section. The input of the CNN is provided by $\boldsymbol{x}^{\mathrm{b}} \in \mathbb{R}^{N_{\mathrm{bin}} \times N_{\mathrm{his}}}$. In the simulations, the number of bins N_{bin} is set to 28 and the number of previous time steps N_{his} to 28, resulting in a 28×28 grid. The first hidden layer convolves four 7×7 filters with stride 1 with the input $\boldsymbol{x}^{\mathrm{b}}$ and applies a rectifier nonlinearity (ReLu). The second hidden layer convolves eight 5×5 filters with stride 1, again followed by a rectifier nonlinearity. The convolutional layers are followed by a single fully connected layer mapping the feature maps to 32 hidden nodes. The time-dependent state component $x_{\mathrm{t},k}$, exogenous state component $\boldsymbol{x}_{\mathrm{ex},k}$ and action u_k are processed using a single, fully connected hidden layer of 16 units. The combined output of the CNN and feedforward network are processed using two fully connected layers, each consisting of 24 units. All layers used ReLu activations and no pooling layers were used. The final hidden representation is mapped to a single output using a fully connected linear output layer with a single hidden output. The network was trained using the RMSProp algorithm [46] with minibatches of size 16.

D. Results

The simulations span a period of 80 days, each simulation taking about 16 hours[2]. During the last eight days, the exploration probability was set to zero, resulting in a completely greedy policy according to (7). In Fig. 4 one can see a selection of the results of the presented approach after different number of days for different outside temperatures. The number of days are indicated in the titles of the top row, i.e. after 20, 60 and 70 days. The bottom row depicts the corresponding outside temperatures. Added in the graph are the results of a benchmark optimization as discussed above. It is observed that the results obtained after 60 and 70 days are close to optimal and this for different outside temperatures.

Since a random exploration term is used, the experiments were repeated 6 times. The results of these 6 simulation runs can be seen in Fig. 5. This figure depicts the power consumption profiles and corresponding electricity prices of the cluster for different days during the learning process. The last two subplots in the bottom row of the figure correspond to the last eight days obtained with a pure greedy policy.

This is presented more quantitatively in Fig. 6, where the scaled performance is depicted. The scaled performance is defined as the daily cost using our approach (Algorithm 2) scaled with the daily cost obtained using the theoretical benchmark. The results in Fig. 6 are obtained by averaging the scaled performance over 6 statistical runs. It is observed that it takes on the order of 30 days before the control policy converges to a scaled performance of 0.95, after which its performance remains stable. Note that in Fig. 6, a scaled performance of one corresponds to the solution of the theoretical benchmark.

[2]Intel$^{\mathrm{R}}$ Core $^{\mathrm{TM}}$ i5 2.5 GHz, 8192 MB RAM

TABLE I: Overview of simulation results.
A: without state-time features. B: with state-time features

Experiment	I	II	III	IV	V	VI
A	1.0176	1.0255	0.9924	1.0043	0.9958	1.0157
B	0.9774	1.009	0.9922	0.9917	0.9902	0.9873
t-test	p	3.2%				

E. State-time features

To identify the contribution of taking into account the history of observations into the state, a set of numerical experiments (spanning 80 days) has been conducted, the results of which are presented in Table I. Six numerical experiments have been performed where the history of observations is added as discussed in Section V. Similar, six numerical experiments have been conducted where the history of observations was omitted. In order to evaluate the contribution of the information present in the past observations, the network architecture has been left unchanged. The state however, has been constructed by copying the last observation b_k, N_{his} times. Table I presents the scaled cumulated cost for both sets of simulations, i.e. with and without taking into account the history of observations. The cumulated cost is calculated for the last 30 days to make the results less sensitive for the effects of exploration. For clarity, the results are normalised with the mean cumulated cost over the twelve experiments. As the expected difference is on the order of one to two percent, a two-sample t-test has been conducted indicating that with almost 97% probability the results originate from distributions with a different mean. Adding the history of observations to the state, reduces the average cost by approximately 1.2%.

VII. CONCLUSIONS

Driven by recent successes in the field of deep learning [20], this work has demonstrated how a neural network, containing convolutional layers, can be used within fitted Q-iteration in a realistic demand response setting with partial observability. By enriching the state with a sequence of past observations, the convolutional layers used to obtain an approximation of the Q-function were able to extract state-time features that mitigate the issue of partial observability. The approach has been evaluated in a qualitative simulation, comprising a heterogeneous cluster of thermostatically controlled loads, that only share their operational air temperature, whilst their *envelope* temperature remains hidden. The simulation experiments have demonstrated that our approach was able to obtain near-optimal results and that the regression algorithm was able to benefit from the sequence of past observations.

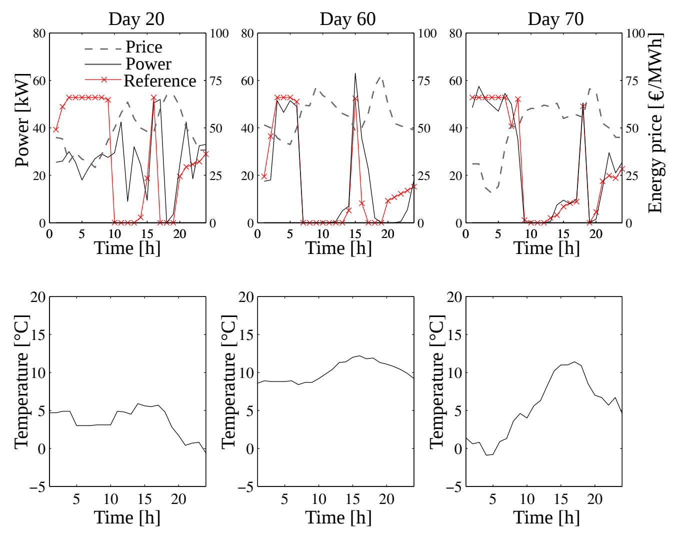

Fig. 4: Illustration of the learning process, the black lines in top row depicts the power profiles obtained with the approach presented in this paper after 20, 60 and 70 days respectively, the red lines indicate profiles corresponding to a benchmark solution. Depicted with the dashed lines are the corresponding price profiles, whilst the lower graphs depict the corresponding outside temperature.

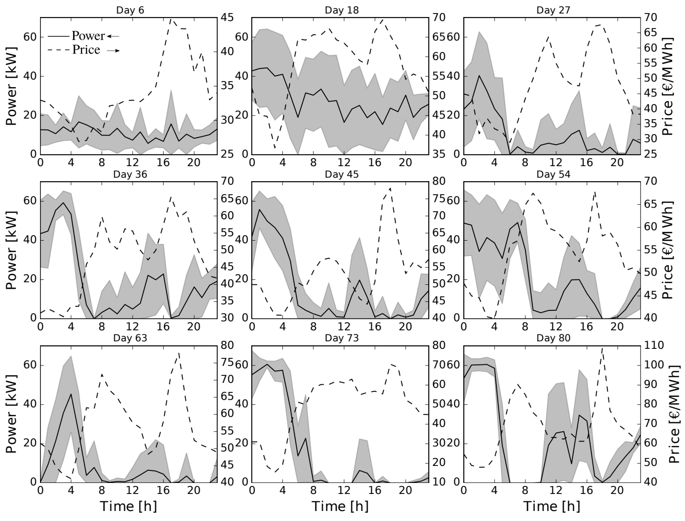

Fig. 5: Average power consumption of the cluster (black line) surrounded by a gray envelope containing 95% of the power consumption profiles (of six simulation runs) for different days during the learning process (left y-axis). Daily price profiles (dashed line, right y-axis).

Future work will be oriented towards the application of high-fidelity building models, which we already started exploring in [47], and testing the performance of the proposed approach for other objectives such as tracking a reference profile. In terms of research, other regression techniques, such as long short-term memory networks, will be investigated [48].

Appendix: Convolutional Neural Network Architecture

The following fragment of Python code shows the construction of the neural network (Fig. 2) used during the simulations (Section VI). The implementation of the neural

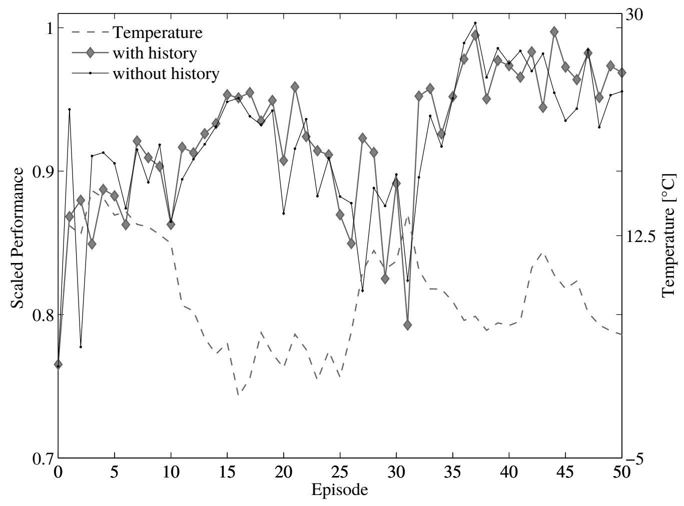

Fig. 6: Illustration of the learning process, depicted is the scaled performance (averaged over six runs) with and without including the state-time features. Also depicted is the average outside temperature.

network was done using Keras [49] and Theano [50].

```
from keras.optimizers import RMSprop
from keras.models import Sequential
from keras.layers.core import (Dense, Activation, Merge,
                               Flatten, Reshape, Convolution2D)

width1 = 7 # width first filter
CNN = Sequential()
CNN.add(Dense(28*28,28*28))
CNN.add(Reshape(1, 28, 28))
CNN.add(Convolution2D(4,1,width1,width1,
                                    border_mode='valid'))
CNN.add(Activation('relu'))

width2 = 5 # width second filter
CNN.add(Convolution2D(8, 4, width2, width2))
CNN.add(Activation('relu'))
CNN.add(Flatten())
scaledGraph = 28-width1+1-width2+1
CNN.add(Dense(8*scaledGraph*scaledGraph, 32))
CNN.add(Activation('relu'))

model = Sequential()
model.add(Merge([CNN,Dense(2,16)], mode='concat'))
model.add(Dense(48,24))
model.add(Activation('relu'))
model.add(Dense(24,24))
model.add(Activation('relu'))
model.add(Dense(24, 1))

# RMSProp optimizer [44]
Rpr = RMSprop(lr=0.001,rho=0.9,epsilon=1e-6)
model.compile(loss='mean_squared_error',optimizer=Rpr)
```

VIII. Acknowledgments

This research is supported by IWT-SBO-SMILE-IT, funded by the Flemish Agency for Innovation through Science IWT through Science and Technology, promoting Strategic Basic Research.

References

[1] S. Koch, J. L. Mathieu, and D. S. Callaway, "Modeling and control of aggregated heterogeneous thermostatically controlled loads for ancillary services," in *Proc. 17th IEEE Power Sys. Comput. Conf. (PSCC)*, Stockholm, Sweden, Aug. 2011, pp. 1–7.

[2] B. Dupont, P. Vingerhoets, P. Tant, K. Vanthournout, W. Cardinaels, T. De Rybel, E. Peeters, and R. Belmans, "LINEAR breakthrough project: Large-scale implementation of smart grid technologies in distribution grids," in *Proc. 3rd IEEE PES Innov. Smart Grid Technol. Conf. (ISGT Europe)*, Berlin, Germany, Oct. 2012, pp. 1–8.

[3] J. L. Mathieu, M. Kamgarpour, J. Lygeros, and D. S. Callaway, "Energy arbitrage with thermostatically controlled loads," in *Proc. 2013 European Control Conference (ECC)*. IEEE, 2013, pp. 2519–2526.

[4] S. Iacovella, F. Ruelens, P. Vingerhoets, B. J. Claessens, and G. Deconinck, "Cluster control of heterogeneous thermostatically controlled loads using tracer devices," *IEEE Trans. on Smart Grid*, vol. PP, no. 99, pp. 1–9, 2015.

[5] S. Vandael, B. J. Claessens, M. Hommelberg, T. Holvoet, and G. Deconinck, "A scalable three-step approach for demand side management of plug-in hybrid vehicles," *IEEE Trans. on Smart Grid*, vol. 4, no. 2, pp. 720–728, Nov. 2013.

[6] J. Mathieu, M. Dyson, D. Callaway, and A. Rosenfeld, "Using residential electric loads for fast demand response: The potential resource and revenues, the costs, and policy recommendations," *Proc. of the ACEEE Summer Study on Buildings, Pacific Grove, CA*, 2012.

[7] Y. Ma, "Model predictive control for energy efficient buildings," Ph.D. dissertation, University of California, Berkeley, 2013.

[8] J. Mathieu and D. Callaway, "State estimation and control of heterogeneous thermostatically controlled loads for load following," in *Proc. 45th Int. Conf. on System Science*, Maui, HI, US, Jan. 2012, pp. 2002–2011.

[9] M. Maasoumy, "Selecting building predictive control based on model uncertainty," *IEEE American Control Conference*, 2014.

[10] J. Cigler, D. Gyalistras, J. Širokỳ, V. Tiet, and L. Ferkl, "Beyond theory: the challenge of implementing model predictive control in buildings," in *Proc. 11th REHVA World Congress*, Czech Republic, Prague, 2013.

[11] D. Ernst, M. Glavic, F. Capitanescu, and L. Wehenkel, "Reinforcement learning versus model predictive control: a comparison on a power system problem," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 39, no. 2, pp. 517–529, 2009.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[13] Z. Wen, D. O'Neill, and H. Maei, "Optimal demand response using device-based reinforcement learning," *IEEE Trans. on Smart Grid*, vol. 6, no. 5, pp. 2312–2324, Sept 2015.

[14] F. Ruelens, B. J. Claessens, S. Vandael, S. Iacovella, P. Vingerhoets, and R. Belmans, "Demand response of a heterogeneous cluster of electric water heaters using batch reinforcement learning," in *Proc. 18th IEEE Power Sys. Comput. Conf. (PSCC)*, Wrocław, Poland, 2014, pp. 1–8.

[15] G. Costanzo, S. Iacovella, F. Ruelens, T. Leurs, and B. Claessens, "Experimental analysis of data-driven control for a building heating system," *Sustainable Energy, Grids and Networks*, vol. 6, pp. 81 – 90, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352467716000138

[16] E. C. Kara, M. Berges, B. Krogh, and S. Kar, "Using smart devices for system-level management and control in the smart grid: A reinforcement learning framework," in *Proc. 3rd IEEE Int. Conf. on Smart Grid Commun. (SmartGridComm)*, Tainan, Taiwan, Nov. 2012, pp. 85–90.

[17] F. Ruelens, B. J. Claessens, S. Vandael, B. De Schutter, R. Babuska, and R. Belmans, "Residential demand response of thermostatically controlled loads using batch reinforcement learning," *IEEE Trans. on Smart Grid*, vol. PP, no. 99, pp. 1–11, 2016.

[18] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, pp. 503–556, 2005.

[19] D. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*. Nashua, NH: Athena Scientific, 1996.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[21] W. Zhang, K. Kalsi, J. Fuller, M. Elizondo, and D. Chassin, "Aggregate model for heterogeneous thermostatically controlled loads with demand response," in *IEEE, Power and Energy Society General Meeting*, 2012, pp. 1–8.

[22] Z. Xu, D. Callaway, Z. Hu, and Y. Song, "Hierarchical coordination of heterogeneous flexible loads," *Power Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–11, 2016.

[23] N. Gatsis and G. Giannakis, "Residential load control: Distributed scheduling and convergence with lost AMI messages," *IEEE Trans. on Smart Grid*, vol. 3, no. 2, pp. 770–786, June 2012.

[24] F. D. Ridder, B. J. Claessens, D. Vanhoudt, S. D. Breucker, T. Bellemans, D. Six, and J. V. Bael, "On a fair distribution of consumer's flexibility between market parties with conflicting interests," *Int.Trans. on Electrical Energy Systems*, 2016.

[25] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[26] S. Vandael, B. J. Claessens, D. Ernst, T. Holvoet, and G. Deconinck, "Reinforcement learning of heuristic EV fleet charging in a day-ahead electricity market," *IEEE Trans. on Smart Grid*, vol. 6, no. 4, pp. 1795–1805, July 2015.

[27] B. Goodrich and I. Arel, "Mitigating catastrophic forgetting in temporal difference learning with function approximation," in *2nd Multidisciplinary Conf. on Reinforcement Learning and Decision Making*, 2015.

[28] J. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," *Advances in neural information processing systems*, pp. 369–376, 1995.

[29] M. Riedmiller, "Neural fitted Q-iteration–first experiences with a data efficient neural reinforcement learning method," in *Proc. 16th European Conference on Machine Learning (ECML)*, vol. 3720. Porto, Portugal: Springer, Oct. 2005, p. 317.

[30] R. Hafner and M. Riedmiller, "Neural reinforcement learning controllers for a real robot application," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 2098–2103.

[31] T. Gabel, C. Lutz, and M. Riedmiller, "Improved neural fitted Q-iteration applied to a novel computer gaming and learning benchmark," in *IEEE Symp. on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, Paris, France, April 2011, pp. 279–286.

[32] T. C. Kietzmann and M. Riedmiller, "The neuro slot car racer: Reinforcement learning in a real world setting," in *Proc. 8th IEEE Int. Conf. on Machine Learning and Applications (ICMLA)*, Miami, Florida, US, 2009, pp. 311–316.

[33] S. Lange and M. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," in *Proc. IEEE 2010 Int. Joint Conf. on Neural Networks (IJCNN)*, Barcelona, Spain, July 2010, pp. 1–8.

[34] S. Lange, M. Riedmiller, and A. Voigtlander, "Autonomous reinforcement learning on raw visual input data in a real world application," in *Proc. IEEE Int. Joint Conf. on Neural Networks*, 2012, pp. 1–8.

[35] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[37] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning," in *Proc. 27th Advances in Neural Information Processing Systems*, 2014, pp. 3338–3346.

[38] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[39] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *arXiv preprint arXiv:1504.00702*, 2015.

[40] G. Reynders, J. Diriken, and D. Saelens, "Quality of grey-box models and identified parameters as function of the accuracy of input and observation signals," *Energy and Buildings*, vol. 82, pp. 263–274, 2014.

[41] E. Vrettos, E. C. Kara, J. MacDonald, G. Andersson, and D. S. Callaway, "Experimental demonstration of frequency regulation by commercial buildings-part i: Modeling and hierarchical control design," *arXiv preprint arXiv:1605.05835*, 2016.

[42] R. Bellman, *Dynamic Programming*. New York, NY: Dover Publications, 1957.

[43] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 689–696.
[44] "Belpex - Belgian power exchange," http://www.belpex.be/, [Online: accessed March 21, 2015].
[45] ILOG, Inc, "ILOG CPLEX: High-performance software for mathematical programming and optimization," 2006, see http://www.ilog.com/products/cplex/.
[46] Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio, "RMSProp and equilibrated adaptive learning rates for non-convex optimization," *arXiv preprint arXiv:1502.04390*, 2015.
[47] A. Aertgeerts, B. Claessens, R. De Coninck, and L. Helsen, "Agent-based control of a neighborhood: A generic approach by coupling modelica with python," in *14th Conf. of Int. Building Performance Simulation Association*, Hyderabad, India, Dec., 2015.
[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov 1997.
[49] F. Chollet, "Keras: Deep learning library," *GitHub repository: https://github. com/fchollet/keras*, 2015.
[50] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math compiler in python," in *Proc. 9th Python in Science Conf*, 2010, pp. 1–7.