Sustainable Cities and Society 45 (2019) 243–257

Contents lists available at ScienceDirect

Sustainable Cities and Society

journal homepage: www.elsevier.com/locate/scs

Fusing TensorFlow with building energy simulation for intelligent energy management in smart cities

José R. Vázquez-Canteli[a,b], Stepan Ulyanin[c], Jérôme Kämpf[d], Zoltán Nagy[a,*]

[a] Intelligent Environments Laboratory, Department of Civil, Architectural and Environmental Engineering, The University of Texas at Austin, 301 E Dean Keeton St Stop C1700, 78712 Austin, TX, USA
[b] National Renewable Energy Laboratory, 1617 Cole Boulevard, Golden, CO, 80401-3393, USA
[c] School of Computer Science, Georgia Institute of Technology, North Avenue, Atlanta, GA 30332, USA
[d] Idiap Research Institute Centre du Parc, Rue Marconi 19, PO Box 592, CH-1920, Switzerland

ARTICLE INFO

Keywords:
Artificial intelligence
Q-learning
Deep learning
Smart cities
Smart grid
HVAC control
Machine learning
Building simulation
Reinforcement learning

ABSTRACT

Buildings account for 35% of the global final energy demand. Efficiency improvements and advanced control strategies have a significant impact in the reduction of energy costs and CO_2 emissions. Building energy simulation is widely used to help planners, contractors, and building owners analyse diverse options regarding the planning and management of energy consumption in buildings. Furthermore, recent advances in data processing and computing have led to the development of sophisticated machine learning algorithms that can learn from large datasets, e.g., sensor data from buildings, and use them to develop building-specific adaptive and automatic energy controllers. Control algorithms, such as deep reinforcement learning can tune themselves, are model-free, and economical to implement. In this paper, we introduce an integrated simulation environment that combines CitySim, a fast building energy simulator, and TensorFlow, a platform for efficient implementation of advanced machine learning algorithms. The integration is achieved via Keras—an API for TensorFlow—and a set of text and csv files for data transfer between the applications. This new environment will allow researchers to investigate novel learning control algorithms, and demonstrate their robustness and potential for diverse applications in the built environment. We present two case studies for energy savings and demand response, respectively.

1. Introduction

Heating and cooling constitutes a large portion of the energy that buildings consume, and buildings account for approximately 35% of the global final energy use (IEA, 2013). Furthermore, population growth and rapid urbanization are increasing the amount of energy being consumed in urban environments. This contributes to increased CO_2 emissions (Leibowicz et al., 2018), exacerbating the effects of climate change, and causing instabilities to the electrical grid, which ultimately lead to higher prices of electricity (Dupont, De Jonghe, Olmos, & Belmans, 2014).

Building automation and real-time control systems have been proven effective in significantly increasing energy savings in the built environment (Marinakis, Doukas, Karakosta, & Psarras, 2013). Integration of distributed renewable energy resources, combined with energy storage systems, can support buildings to consume energy more efficiently and reduce their dependency on the electrical grid. However, energy storage devices must be controlled properly, storing energy when it is most efficient to generate it and releasing it when the energy supply systems are less efficient. Adequate control strategies for heating ventilation and air conditioning (HVAC) can increase energy savings in buildings and reduce CO_2 emissions as well as the cost of the energy that the buildings consume. Some advanced control systems such as Model-Predictive Control (MPC) (Rault, 1978) are often too costly to be implemented in small residential buildings because they require identifying the dynamic model of the system they control (Prívara et al., 2013). Furthermore, any building retrofit and/or system upgrade would require model-based controllers to be redesigned, or at least recalibrated. In contrast, potentially model-free control approaches, such as reinforcement learning (RL) (Sutton & Barto, 1998), can allow an easier implementation without the need to develop and identify a thermal model of the building.

Given the increasing presence of sensors in HVAC systems (Park & Nagy, 2018), and new advances in cloud computing for HVAC control

* Corresponding author.
E-mail address: nagy@utexas.edu (Z. Nagy).

https://doi.org/10.1016/j.scs.2018.11.021
Received 20 July 2018; Received in revised form 13 November 2018; Accepted 14 November 2018
Available online 15 November 2018

(Javed et al., 2017), there is an increasing amount of historical sensor data that can be used to optimize HVAC operations (Zhun, Haghighat, & Fung, 2016), (Capozzoli, Piscitelli, & Brandi, 2017). RL is particularly useful for this purpose as it can find improved control policies after being trained either *off-line*, e.g., using historical data, or *on-line*, e.g., when it controls the system. In fact, a RL controller can learn from a different controller by observing its actions, whether it is a rule-based controller (RBC) or an MPC. It can then improve the control actions over time.

Vázquez-Canteli and Nagy reviewed various RL algorithms and techniques that have been used in the built-environment in the past (Vázquez-Canteli & Nagy, 2019). The seminal contribution to this field was the work by Mozer in the Neural Network House project in 1998 (Mozer, 1998): An artificial neural network controlled the HVAC, domestic hot water (DHW), and lighting systems of a house to minimize user discomfort and energy costs. Liu and Henze reduced the energy costs of a passive thermal storage inventory (Liu & Henze, 2007). It is noteworthy that this is a nascent research field with most contributions having been published after 2013. Yang et al. proved RL to be an effective algorithm for optimal energy control in low exergy buildings (Yang, Nagy, Goffin, & Schlueter, 2015). Urieli and Stone, and Ruelens et al. applied different RL algorithms to control a heat pump with a setback strategy and an auxiliary heater for optimal energy conservation and comfort maximization (Ruelens, Iacovella, Claessens, & Belmans, 2015; Urieli & Stone, 2013). De Somer et al. maximized the self-consumption of local PV generation by storing the energy in DHW buffers in an experimental setting (De Somer et al., 2017). More recently, Mocanu et al. proposed a deep reinforcement learning controller to minimize the cost of consuming electricity by some electrical loads that could be found in a building (Mocanu et al., 2017). Despite these advances, most of these contributions use very simplified building energy models, very specialized RL implementations, or are difficult to reproduce. As a consequence, there is a need to bridge high performance building energy simulation models with state-of-the art machine learning tools in a systematic and reproducible manner.

In this paper, we introduce an integrated simulation environment that combines CitySim (Robinson, 2011), a fast building energy simulator, and TensorFlow (Agarwal et al., 2015), a platform for efficient implementation of advanced machine learning algorithms. This new environment will help researchers test novel control algorithms based on deep learning and demonstrate their robustness and potential for diverse applications in the built environment. Furthermore, it will help planners and researchers analyze the performance of different controllers through building simulation and for different scenarios such as climate change. We apply this simulation environment in two case studies for energy savings, and demand response respectively. By using commercial and open-source software, freely available under academic licenses, the results generated are reproducible by the research community and practitioners.

In the first case-study, we investigate a deep reinforcement learning (DRL) controller to minimize the energy consumed by a heat pump from the electrical grid in combination with a chilled water tank, and a photovoltaic array. We show how DRL can tune itself and adapt to sudden changes in the physical characteristics of the buildings, e.g., after a retrofit, and with the addition of new energy supply systems, e.g. PV panels. We also show how the controller can learn both *on-line*, and *off-line* from historical data. The results show that DRL outperforms a rule-based controller (RBC) in every case. Lastly, we perform a sensitivity analysis on the number of neurons of the neural network.

In the second case study, we apply the simulation environment to two residential buildings in Austin, TX, which share the same price for electricity. In each building, a heat pump provides the necessary cooling, and a water tank provides storage capabilities. The price for electricity increases with the combined electrical demand of both buildings. As there is no communication between the two buildings, the RL controller of each building must learn how to compete with the controller of the other building to achieve cost savings and energy reduction. We also demonstrate how RL can adapt to the installations of PV panels on top of the buildings, which changes the dynamics of the system.

This article is organized as follows. Section 2 introduces reinforcement learning and DRL. Section 3 introduces the simulation environment, specifically, the data exchange between CitySim and TensorFlow. In Sections 4 and 5, we present the two case studies, respectively. In Section 6, we discuss our findings, and Section 7 concludes the paper.

2. Reinforcement learning

Reinforcement learning (RL) is formalized using a Markov Decision Process (MDP). The MDP contains four elements: a set of states S, a set of actions A, a reward function R: $S \times A$ and transition probabilities between the states P: $S \times A \times S \in [0,1]$. The policy π then maps states to actions as $\pi: S \rightarrow A$, and the value function $V^{\pi}(s)$ is the expected return for the agent when starting in the state s and following the policy π, i.e.,

$$V^{\pi}(s) = \sum_{a} \pi(s, a) \sum_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma V^{\pi}(s')] \tag{1}$$

where $R^{a}_{ss'}$, sometimes denoted as $r(s, a)$, is the reward for taking the action $a = \pi(s_k)$, and transitioning from the current state s to the next state s', and $\gamma \in [0,1]$ is a discount factor for future rewards. An agent that uses $\gamma = 1$ will give greater importance to long-term rewards, whereas for $\gamma = 0$, greater values are assigned to states that lead to high immediate rewards. RL is particularly useful when the model dynamics (P and R) are unknown and must be determined or estimated through interaction of the agent with the environment (see Fig. 1). Two approaches can be used to determine the values V^{π} of every state. In the model-based approach, the rewards and transition probabilities of the model are first learned, and then used to find the values by solving iteratively the system of equations represented by Equation (1). In the model-free approach, the agent learns the values associated to every (s, a) pair without explicitly calculating the transition probabilities or the expected rewards (Nagy, Park, & Vázquez-Canteli, 2018; Sutton & Barto, 1998).

In this paper, we use Q-learning, which is the most widely used model-free RL technique due to its simplicity (Watkins & Dayan, 1992). For tasks with small finite state sets, all transitions can be represented with a table that stores the state–action values, or Q-values. Each entry in the table represents a state–action tuple (s, a), and the Q-values are updated as

$$Q(s, a) \longleftarrow Q(s, a) + \alpha [r(s, a) + \gamma \max_{a} Q(s', a) - Q(s, a)] \tag{2}$$

where s' is the next state, and $\alpha \in (0,1)$ is the learning rate, which explicitly defines to what degree new knowledge overrides old knowledge. For $\alpha = 0$, no learning happens, while for $\alpha = 1$, prior Q-values are completely overridden by the new Q-values in every iteration.

There is a trade-off between exploring actions that have not yet been taken and exploiting others from which the algorithm expects high cumulated rewards in the long-term. Two of the most common methods to select the actions are the ε-greedy policy, which selects a random

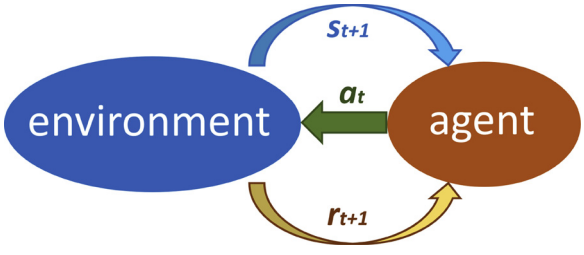

Fig. 1. Agent and environment interaction.

action with a probability ε, and the greedy action with a probability $(1-\varepsilon)$, and the soft-max action-selection, which selects the actions with a probability that is related to their Q-value as

$$\Pr(as) = \frac{e^{\frac{Q(a,s)}{\tau}}}{\sum_{b=1}^{n} e^{\frac{Q(s,b)}{\tau}}} \tag{3}$$

where τ is the Boltzmann temperature constant: high values of τ result in equiprobable actions, regardless of their Q-values, while low values of τ lead the action-selection process towards a greedy policy, in which actions with the highest Q-values are selected with a higher probability. It is typical to start the learning process with high values of τ to increase exploration, and then reduce τ to exploit the acquired knowledge and maximize the obtained rewards.

2.1. Deep reinforcement learning

In the tabular approach, only one Q-value is updated at each iteration, and the states and actions must be discrete. When the state-action space is larger, and has potentially continuous values, the tabular approach becomes unpractical. As a remedy to this *curse of dimensionality,* the table can be substituted by regression algorithms which map states and actions directly to their Q-values (Busoniu, Babuska, De Schutter, & Ernst, 2010).

In this work, we use a non-linear mapping function, namely a Deep Neural Network (DNN), shown in Fig. 2, (Gullapalli, 1990), which results in deep reinforcement learning (DRL). The number of neurons in the input layer of the DNN is the sum of states and actions of the learning problem, and the output layer has a single neuron that outputs the Q-value. All the layers of the DNN, except the output layer, may use different non-linear activation functions (we used the tanh activation function). The output layer uses a linear activation function because the Q-values are not limited to $[-1,\ 1]$.

The DRL algorithm is shown in Table 1. The experience of the agent is stored in batches D that contain the tuples $(s_k,\ a_k,\ r_k,\ s_k^{'})$, before any updates are made (Ernst, Geurts, & Wehenkel, 2003). When a batch is completed, the Q-values are updated using Eq. (2), with $\alpha = 1$. The resulting Q-values are then defined as targets to fit for the DNN regression using all available batches, and the DNN is re-trained (Kalyanakrishnan, Stone, & Liu, 2008). Thus, the parameters of the DNN, θ, are updated after training it with known Q-values to estimate the Q-values of state-action tuples that have not yet been observed. The weights of the DNN are randomly initialized before performing back-propagation on the first batch of collected data. For the following batches of training data, the network weights are initialized using the weights of the previously trained DNN. This reduces the time required to train the network, and makes it less sensitive to random initializations.

As an example, DRL has been successfully applied in applications such as playing video games (Mnih et al., 2013).

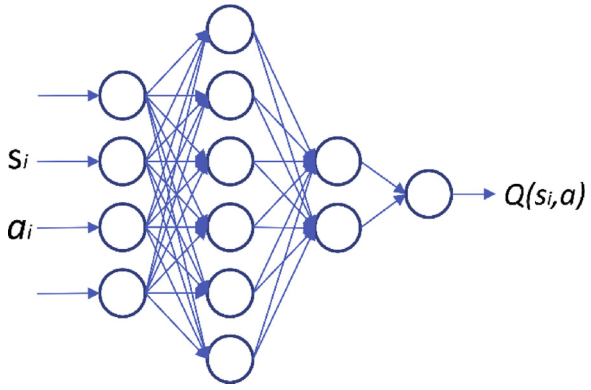

Fig. 2. Artificial Neural Network for DRL with fitted Q-iteration.

Table 1
Deep reinforcement learning for non-episodic task.

1:	$\tilde{Q}$⟵Initialize DNNs randomly
2:	k⟵0
4:	**Repeat**
5:	s_k⟵scale(s_k)
6:	a_k ⟵ *actionSelection*(s_k, $\tilde{Q}$)
7:	$s_k^{'}$⟵ *takeAction*(s_k, a_k)
8:	r_k⟵ *getReward*(s_k, a_k, $s_k^{'}$)
9:	D⟵(s_k, a_k, r_k, $s_k^{'}$)
10:	**If** (k+1) % batchSize= 0
	$p = 0$
11:	**Repeat for all** s_k, a_k, r_k, $s_k^{'}$ **in D**
12:	a^*⟵$arg\ max_a \tilde{Q}(s',\ a,\ \theta)$
13:	$Q(s,\ a)$⟵$r(s,\ a) + \gamma\ \tilde{Q}(s',\ a^*,\ \theta)$
14:	θ⟵trainDNN(s, a, Q)
	$p++$
15:	**Until** $p = epochs$
16:	**end if**
18:	$k++$
21:	**until** k= simulation time

3. Simulation environment: CitySim and TensorFlow

We now introduce the integrated simulation environment (see Fig. 3), that allows us to benefit from the versatility of CitySim, a building energy simulator, and TensorFlow, an extensive and fast machine learning library. In this environment, we can implement a DRL controller for the HVAC systems, and make changes to the building energy model, e.g. add or remove energy supply systems, or apply building retrofitting, and then analyze how the DRL controller continuously adapts to those changes.

This approach also allows to evaluate the performance of the controller under different weather scenarios and in buildings with different physical characteristics. Additionally, TensorFlow allows us to easily implement different types of machine learning techniques in our control algorithm, e.g., different types of DNNs.

CitySim is a building energy simulator developed at the École Polytechnique Fédérale de Lausanne (EPFL). It computes the hourly energy demand for heating, cooling, and lighting in each building. Its radiation models have been validated (Walter & Kämpf, 2015), and used, e.g., in urban retrofit analysis (Vázquez-Canteli & Kämpf, 2016), and in the analysis of occupant's behaviour on building energy demand (Haldi & Robinson, 2011).

For our simulation environment, we extended the features of CitySim, by allowing it to export data at each hourly time step during the simulation via the use of a C++ library. This is necessary so that the controller can take actions at each hour. Furthermore, we programmed part of our controller within the library to achieve faster computation speeds.

TensorFlow is an open source machine learning library created for efficient numerical computation, using data-flow graphs (Agarwal et al., 2015). We opted to use TensorFlow to make use of the available deep learning libraries. Furthermore, there exist high-level APIs for implementation of machine learning algorithms, such as Keras (Chollet, 2015), an open source library that uses TensorFlow as a backend engine.

For our DRL controller, we train and test the DNNs with Keras, and the resulting information is then processed in C++ to take the corresponding control decisions. One of the challenges was the integration of Python libraries, used by Keras, into the C++ code of CitySim. The iterations of the simulator are dynamic, therefore there is no easy way to render Python code in CitySim or C++ code in Python. Hence, we chose to build a read/write hub between the two frameworks using CSV

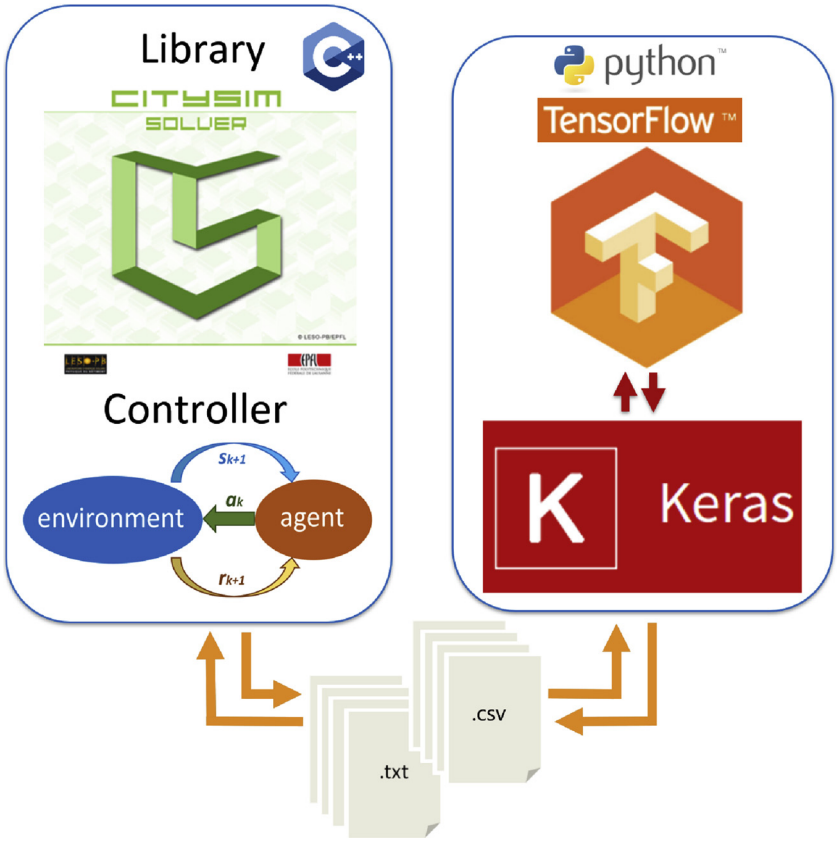

Fig. 3. Simulation environment: merging CitySim and TensorFlow.

and TXT files. This proved to be a simple and computationally not very heavy solution to the problem. Both, the controller and Keras update and exchange flags to guarantee that only one of the two is executing at any given time.

Fig. 4 shows the flow chart of the DRL algorithm. First, the weights of the DNN are randomly initialized. Then, the algorithm saves the current states provided by CitySim (e.g., the outdoor and indoor temperatures, the hour of the day, the temperature of the tank, and the photovoltaic power generation). These states are stored as s_k and $s_k^{'}$, where the subscript k is the time-step in the simulation, s is the current set of states, and s' is the set of states following s after taking the action a. The resulting array is then fed-forward through the DNN to obtain the Q-values associated with every possible action the controller can take under the present state. The Q-values are computed with TensorFlow, and sent back to the controller, which uses soft-max action selection to determine the action to take. The chosen action is then passed on to CitySim to perform the energetic analysis of the building at that time-step.

Once CitySim simulates the building with the new action, the controller can obtain the relevant data to compute the reward r_k. Then, the tuple $(s_k, a_k, r_k, s_k^{'})$ is stored in a batch. If the batch is not yet completed, the simulation advances to the new time-step and CitySim provides the next states to the controller, which are saved. When the batch is completed, the controller will compute the new Q-values and re-train the DNN iteratively as depicted in Fig. 4. The batches are created incrementally, and the Q-values are always updated using the data from all the previous batches. Because of the use of batches in the learning process, this is also sometimes referred to as Batch Reinforcement Learning (BRL). Thus RLC, DRL, and BRL are used interchangeable in this work.

The result is a powerful environment, which can take advantage of any the machine learning functions developed for TensorFlow, and that allows the user to easily develop building energy models, make changes on them, and modify their sequence of operation.

We now move on to apply this simulation environment in two case studies.

4. Case study 1: energy savings of a single building

We apply the simulation environment for the cooling control of a building in Austin, TX. We chose a period of hot weather comprised of 122 days for our simulation, between May 19th and September 7th, as Fig. 5 illustrates. We selected this period because it has a relatively steady range of temperatures that allow us to study the performance of our controller when implemented in a cooling system. The weather file was obtained from Meteonorm (Meteonorm, 2018).

4.1. Building energy model

We selected and modelled a seven-storey residential building (Fig. 6). We estimated its physical characteristics based on typical values complying with the ASHRAE 90.1 energy standard (ASHRAE, 2014) as shown in Table 2. We used a ventilation rate (infiltration plus mechanical ventilation) of $0.9\,h^{-1}$, windows with a U-value of $2.135\,W/m^2K$, and a solar energy transmittance coefficient (G-value) of 0.49.

The building energy system is comprised of an air-source-to-water heat pump that provides cooling energy to a chilled water tank. The tank stores the water and provides cooling to the building. The objective of the controller is to reduce the dependency of the heat pump on the electrical grid by storing and releasing cooling energy from the chilled water tank at different times of the day, while maintaining the indoor temperature at or below 24 °C.

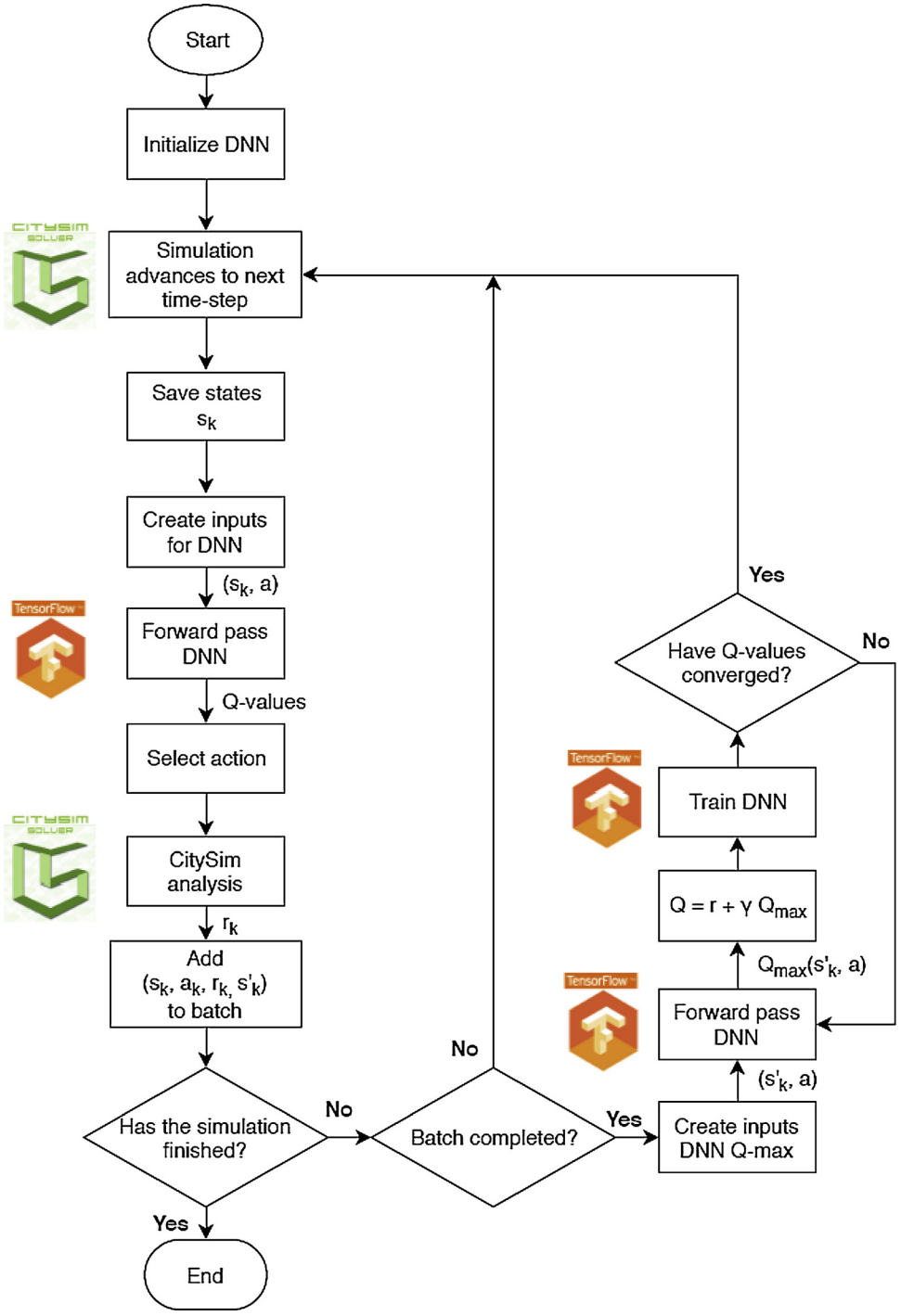

Fig. 4. Flow chart of the deep reinforcement learning controller. (the subscript *k* indicates the current timestep).

4.2. Application scenarios

We analyzed three scenarios to test the ability of the reinforcement learning controller (RLC) to tune itself and adapt to different changes in the energy system it controls. In all cases, the electricity consumption of the RLC is compared to that of a rule-based controller (RBC). The three different scenarios are:

- Scenario S1 (Base case): the resulting electricity consumption of the RLC is compared to that of the RBC. We also perform a sensitivity analysis to the number of neurons in each hidden layer of the DNN.
- Scenario S2 (Change on supply side): Photovoltaic panels are added on the roof of the building to provide electricity to the heat pump. This provides the controller with an additional energy source. We assume that the generated electricity is not used, i.e., lost, if it is not consumed immediately.
- Scenario S3 (Change on demand side): The building is retrofitted after 1000 h of simulation. The retrofitting event consists of reducing the ventilation rate of the building from 0.9 h^{-1} to 0.5 h^{-1}. The retrofitting event changes the dynamics of the system, and the RLC must learn and adapt to this new dynamical system.

We simulate how the RLC transitions from a passive, learning-only stage, to an active, controlling stage as follows. In each scenario, the

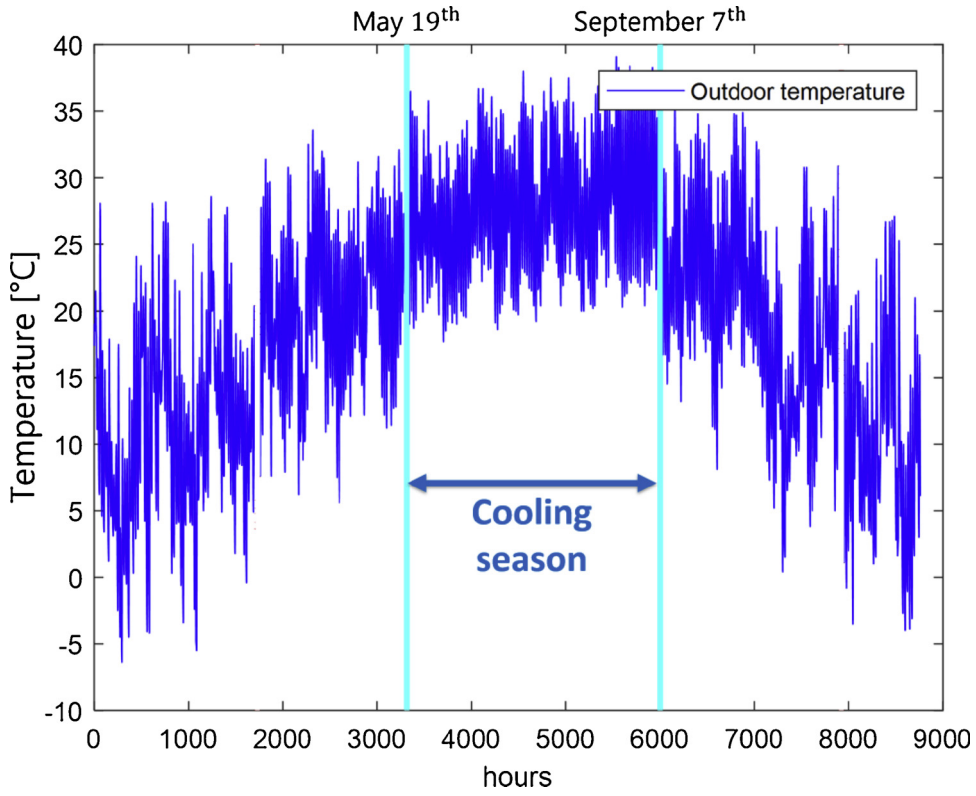

Fig. 5. Outdoor air temperature and the period of cooling used for simulation.

RLC learns off-line by observing the RBC for the initial 360 h. Afterwards, the RLC begins to control the system and learns on-line.

The RBC follows a simple, yet typical sequence of operation: when the temperature of the chilled water tank reaches 20 °C, it is cooled down to 10 °C. Then, the tank provides the building with sufficient cooling to maintain the indoor temperature below 24 °C, and the heat pump does not provide any additional cooling energy to the water tank until it reaches 20 °C again.

4.3. State-action space design

The design of the state-action space for the MDP and the reward function is a key part of setting up the RL controller. Fig. 7 illustrates our choice of states and actions. The variables that are useful in predicting the immediate future reward must be selected as states or actions. Since the problem is modelled as an MDP, the immediate future reward must only depend on the current states and actions, and not on their past values (Markov property). Since the objective of the controller is to minimize the electricity consumption of the heat pump, we choose the penalty to be proportional to the amount of electricity consumed from the grid.

The electrical consumption of the heat pump from the electrical grid depends on its coefficient of performance (COP), the demand for cooling of the water tank, and the photovoltaic power generation. The cooling demand of the tank can be estimated based on its current temperature, and its target temperature for the next time step. Therefore, these two variables are used as a state (s_3) and an action (a_1), respectively. The cooling demand of the building mostly depends on its external temperature and solar gains. The external heat gains depend on the outdoor temperature, which we included as state (s_1). The internal gains are assumed to be dependent on the hour of the day, which is set as state s_2. In scenario S2, we also include photovoltaic generation as state s_4. This not only helps predict how much electricity is needed from the grid, but is also a good estimator of solar irradiance, and consequently of the solar gains. The other scenarios, S1 and S3, have no estimator for the solar gains.

We assume the temperature set-point for cooling is constant and equal to 24 °C. Since we designed the controller to maintain this set-point, the indoor temperature was 24 °C most of the time. Therefore, we did not include neither the indoor temperature, nor the temperature

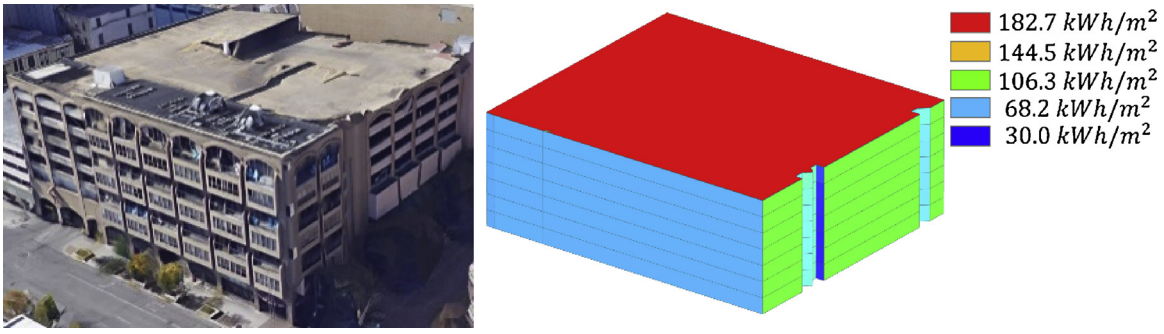

Fig. 6. The building envelope and its representation in CitySim (the color represents the solar irradiance of the first month of the simulated period).

Table 2
Materials of construction for the building envelope.

	Material	Thickness [m]	Thermal Conductivity [W/(mK)]	Cp [J/(kgK)]	Density [kg/m^3]
Walls	Rendering	0.02	0.870	1100	1800
	PS 30 polystyrene	0.10	0.036	1400	30
	Reinforced concrete	0.17	2.400	1000	2350
	Plaster	0.01	0.430	1000	1200
Floors	Reinforced concrete	0.3	2.400	1000	2350
Roof	Rendering	0.02	0.870	1100	1800
	PS 30 polystyrene	0.10	0.036	1400	30
	Reinforced concrete	0.17	2.400	1000	2350
	Plaster	0.01	0.430	1000	1200

set-point as states.

4.4. Action selection

We used soft-max action-selection, and gradually reduced τ from 0.2 to 0.02 to evolve the controller from an almost random search policy towards a greedy policy. Fig. 8 illustrates the convergence of the soft-max action-selection algorithm for the three different actions low, medium, and high tank temperatures (note that in Fig. 8, $\tau = T_{boltzman}$). As τ decreases over time, the probability of taking the greediest action increases. When the simulation convergences, the algorithm has learned the greedy policy and chooses the actions with the highest Q-value very often.

As mentioned above, the RLC controller first learns off-line from 360 h of historical control data, obtained from the sequence of operation of the RBC, and begins the soft-max action-selection process thereafter. Over time, it learns which actions provide the best long-term rewards for each state. After collecting about 1000 h of data for training, the RLC begins to take the actions it considers to be the best (greediest action) very often. This is when we consider that the controller has converged. Fig. 8 illustrates how, after 1000 h of data collection for training, the probability of taking the greediest action increases substantially, while the likelihood of taking the 2nd and the 3rd greediest actions decrease. As an example, consider Fig. 9, which shows all the Q-values associated with every possible action the controller can take at every time-step (temperature set-point of the tank). The red line displays the action taken by the controller. Clearly, the maximum (or close to maximum) Q-values are chosen, demonstrating that the controller follows the greedy policy.

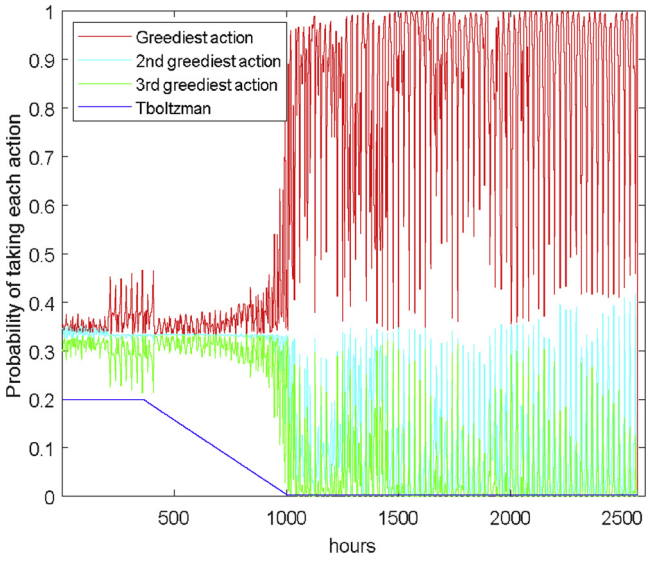

Fig. 8. Convergence of the soft-max action-selection algorithm during the simulation period.

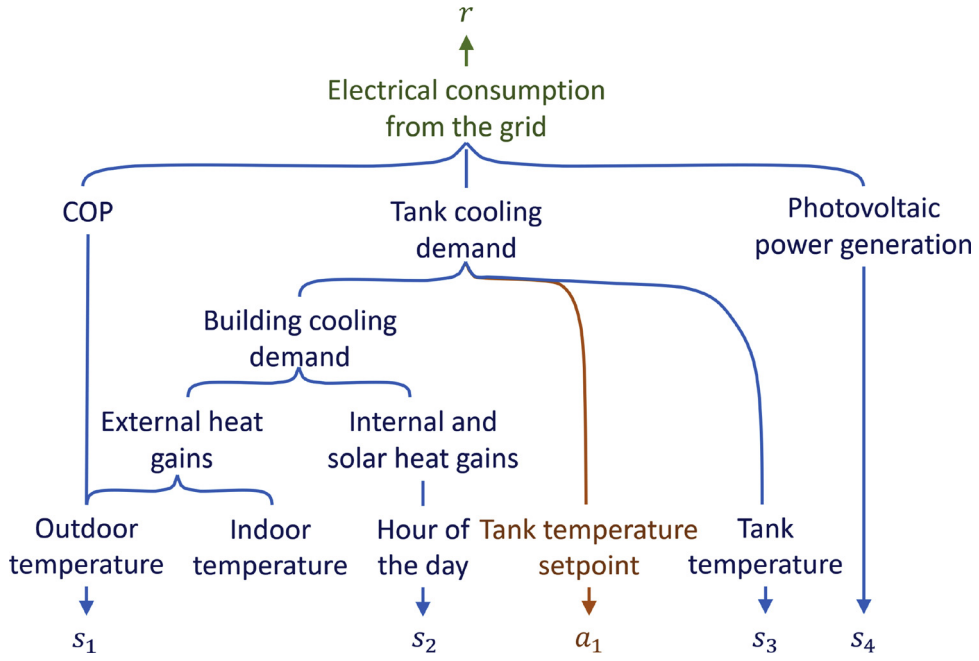

Fig. 7. Selection of the states, the action, and the reward.

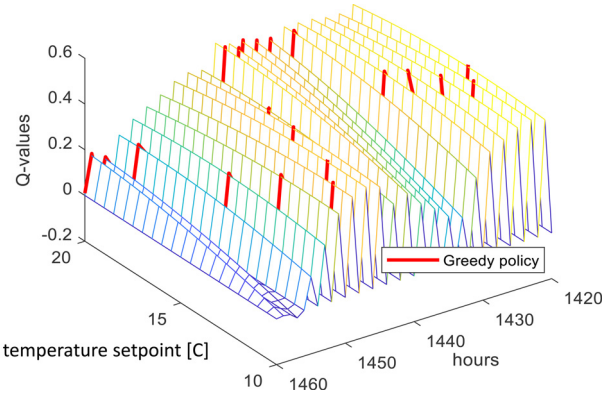

Fig. 9. 3D mapping between the state-action space and the associated Q-values.

4.5. Security and limits of operation

Since the RLC does not require a model of the system, it is essentially a black-box controller. Thus, it is crucial to maintain operational safety limits, e.g., guarantee that it only takes actions between certain bounds. In this case study, we set a minimum and maximum temperature of the chilled water tank of 10 °C and 20 °C, respectively. If the RLC takes any action that would lead the system variables to exceed their limits, the action is overridden by an alternative action that sets those variables to their limit.

Fig. 10 shows two snapshots during the learning process, one early one (hours 800–1000) and one later one (hours 1520–1680), when the controller has learned more. The green line represents the actions that the RLC *wants to* take, and the blue line is the action that it is *actually* taking after considering the safety limits. It is apparent that, in the beginning, Fig.10 (top), many more actions of the RLC are overridden, while after 1500 h of simulation, Fig.10 (bottom), the number of actions that are overridden has been considerably lowered, as the planned and executed RLC actions are essentially the same.

4.6. Results

The total simulation time of the case study with the RLC was 13 min and 21 s (CPU: i7-6700 K 4.0 GHz, RAM: 64.0 GB, GPU: NVIDIA Quadro M2000). This includes the time required to compute the Q-values and train the DNN in every training step. The batch size for DRL was 100 data points, and the Q-values were updated iterating 100 times for each new batch collection. The RLC gathered data or selected new set-points for the water tank with a 2 h resolution, while the rest of the calculations of the simulation were computed for every hour. While the temperature of the water tank is controlled by reinforcement learning with a 2 h resolution (which controls how much cooling energy is stored), the temperature control of the building is done by a separate controller. In this case, the building is always supplied with enough cooling to ensure that its indoor temperature matches the setpoint. This cooling comes first from the water tank, and, if the tank does not have any cooling energy stored, it is supplied directly by the heat pump. The temperature of the building is controlled with a 1 h resolution, which is the time resolution of the building simulator CitySim. By contrast, the simulation of the building energy model using the rule-based controller instead of the RLC controller lasted about 2 s.

We now analyze the performance of the RLC. For each scenario, the electricity consumption, shown in Fig. 11, was normalized by the total cumulated electricity consumption of the RBC of the base case (S1).

4.6.1. Scenario S1: base case and sensitivity analysis

As shown in Fig. 11 (top), the RLC achieved electricity savings of about 4% with respect to the RBC. This demonstrates that RLC can learn off-line by observing the sequence of operation of the RBC, and then improve upon it. While the savings seem modest, it should be noted that they have been achieved without knowledge of the building thermal dynamics or predictive controllers to make optimal decisions. As such, the RLC essentially achieved the savings for free. Of course, a longer training period could potentially lead to even higher savings. After collecting about 1000 h of data for training, the controller starts to take greedier actions very often to minimize the electricity consumed.

To analyse *what* the RLC has learned, consider Fig. 12, which shows the indoor, outdoor, and water tank temperatures for both RBC and RBL at the (top), while the control sequence that the RLC has learned is shown at the (bottom). It can be seen that each day, when the outdoor temperature reaches its minimum, the heat pump provides cooling energy to the water tank, decreasing its temperature. This strategy maximizes the COP of the heat pump, by consuming electricity only when the difference between the outdoor temperature and the temperature of the water is low. When the outdoor temperature increases, and the building requires more cooling, the water tank releases this cooling energy into the building, increasing the temperature of the water tank. Fig. 12 (bottom) depicts that, when converged, the RLC achieves an overall greater and steadier COP than the RBC, i.e., a more efficient heat pump operation.

Table 3 contains the sensitivity analysis. Specifically, the results of scenario S1 for various number of neurons of each of the two hidden layers of the DNN are shown. The DNN with 6 neurons in the first hidden layer, and 2 neurons in the second hidden layer provided the best results: 4% electricity savings. This DNN structure was used to generate the results for each of the three scenarios shown in Figs. 11 and 12.

4.6.2. Scenario S2: installation of photovoltaic PV panels

In this scenario, shown in Fig. 11 (top), PV panels are added at the beginning of the simulation. This change in the energy supply changes the dynamics of the system. Since the objective of the RLC is to minimize the amount of electricity consumed from the electrical grid, the addition of PV panels provides “free” electricity during the day.

As a result, while in scenario S1 the RLC consumed and stored additional electricity during the night to maximize the COP of the heat pump, in this scenario, S2, the controller attempts to maximize the self-consumption from the PV array. During the day, the photovoltaic array reduces the electricity that the heat pump consumes from the grid, but the coefficient of performance (COP) of the heat pump will typically be lower than during the night time. During the night, the heat pump will usually benefit from a higher COP but will lack the electricity generated by the photovoltaic array. Overall, the RLC outperforms the RBC by achieving energy savings of about 8.2%, i.e., about twice as much as without PV panels (S1).

4.6.3. Scenario S3: building retrofit

In this scenario, shown in Fig. 11 (bottom), we made a sudden change in the demand side after 1000 h of simulation to model a building retrofit. Comparing scenarios S3 to S1, the retrofit reduced the electricity demand of the building. However, the RLC still improved on the RBC, which demonstrates that a) the RLC adapted to the changed thermal properties of the building, and b) continued to learn and improve its behaviour.

5. Case study 2: cost savings in a demand response scenario

For this case study, we modelled a seven-story, and a nine-story residential building, both located in downtown Austin, TX, and illustrated in Fig. 13. The energy systems of each building are—as in the first case study—an air-source to water heat pump that provides cooling energy to a chilled water tank, which stores the water and provides cooling to the building. The goal of the controller is—again—to reduce the dependency of the heat pump on the electrical grid by storing and releasing cooling energy from the chilled water tank.

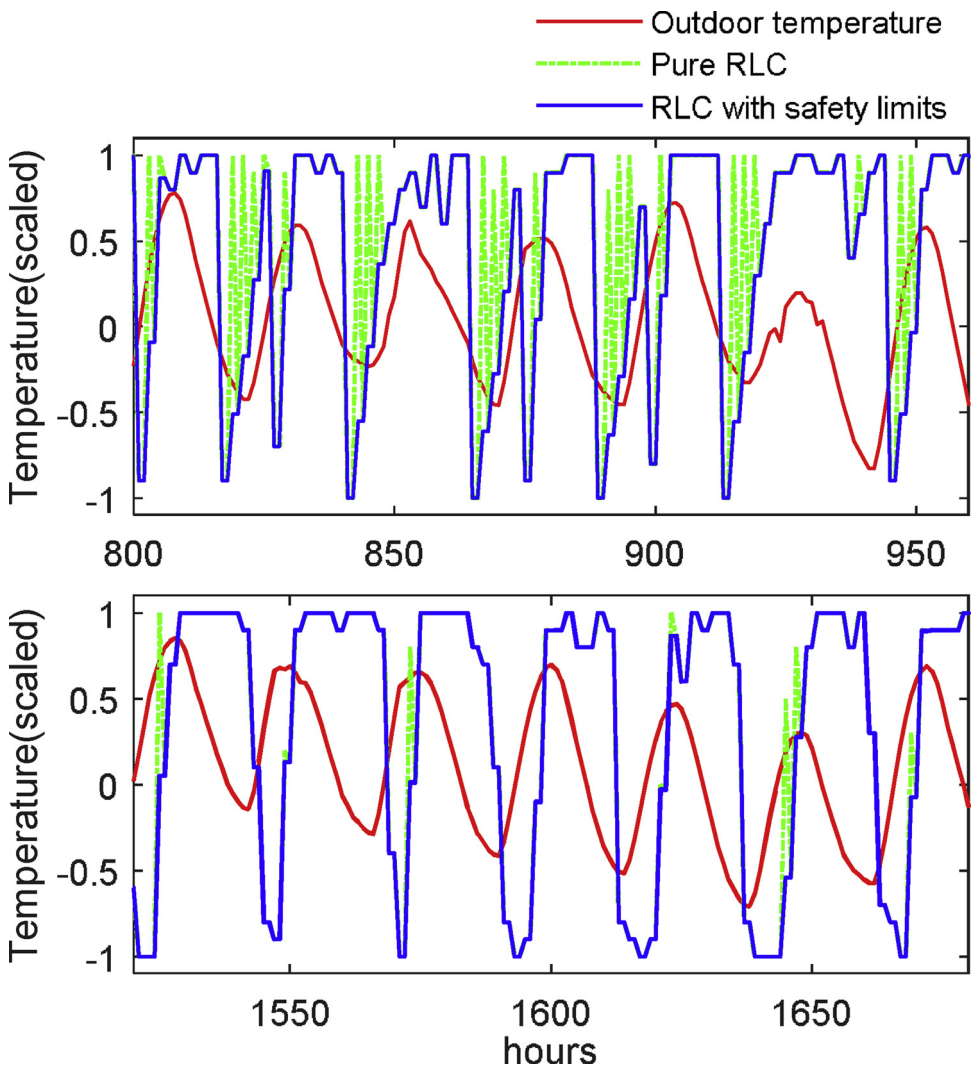

Fig. 10. Evolution of the pure RLC and the RLC with the safety limits (temperature set-points of the chilled water tank) over time, during the initial (top) and final (bottom) learning stage.

If a photovoltaic array is present (second experiment), it can supply the heat pump with electricity, and we assume that the buildings must either store or consume the electricity they generate. The excess energy cannot be sold to other buildings for a profit. Therefore, if the PV panels have the capacity of generating more electricity that can be stored or consumed at a given time, such capacity is not utilized. During the daytime, the PV panel reduces the electricity that the heat pump consumes from the grid, but the coefficient of performance (COP) of the heat pump will be lower during these hours. At night, the heat pump benefits from a higher COP, but lacks the electricity supplied by the PV panel.

5.1. Electricity prices

The price of electricity is modelled proportionally to the sum of the electrical consumption of both buildings at any given time. This constitutes an incentive for the buildings not to consume electrical energy simultaneously. The purpose of using a simplified linear model of electricity prices, which is dependent on the energy consumption of both buildings, is not to model a real electricity market scenario but rather to create conditions under which the buildings need to learn from each other and achieve some degree of coordination: if one building increases its energy consumption from the grid, the electricity prices for both buildings will rise. However, the buildings do not actively exchange information (states or actions) between each other, only the observed combined electricity price after consumption is shared. The relation between the price of electricity P (in USD), and the total electricity consumption E (in kWh) of both buildings at any given time is

$$\boldsymbol{P} = 3{\cdot}10^{-5}{\cdot}\boldsymbol{E} + 0.045 \tag{4}$$

In reality, the increase of electrical demand produces increases in the wholesale prices for electricity, which utilities eventually pass on to the consumers in the form of higher retail prices of electricity. Eq. 4 is based on an estimation of actual retail prices provided by Austin Energy. As mentioned, in this case study, we do not intend to simulate the actual electricity market, but to show that buildings can learn and adapt to changes in the prices of electricity caused by the actions of other buildings, and compete for a lower energy bill using reinforcement learning in our integrated simulation environment.

5.2. The controller state-action space

The objective of the batch reinforcement learning (BRL) controller is to minimize the cost of the electricity consumed by the heat pump from the power grid. Therefore, the reward that the controller receives is the cost of electricity. Fig. 14 shows all the variables we chose as states and actions that help predicting the future reward of the system.

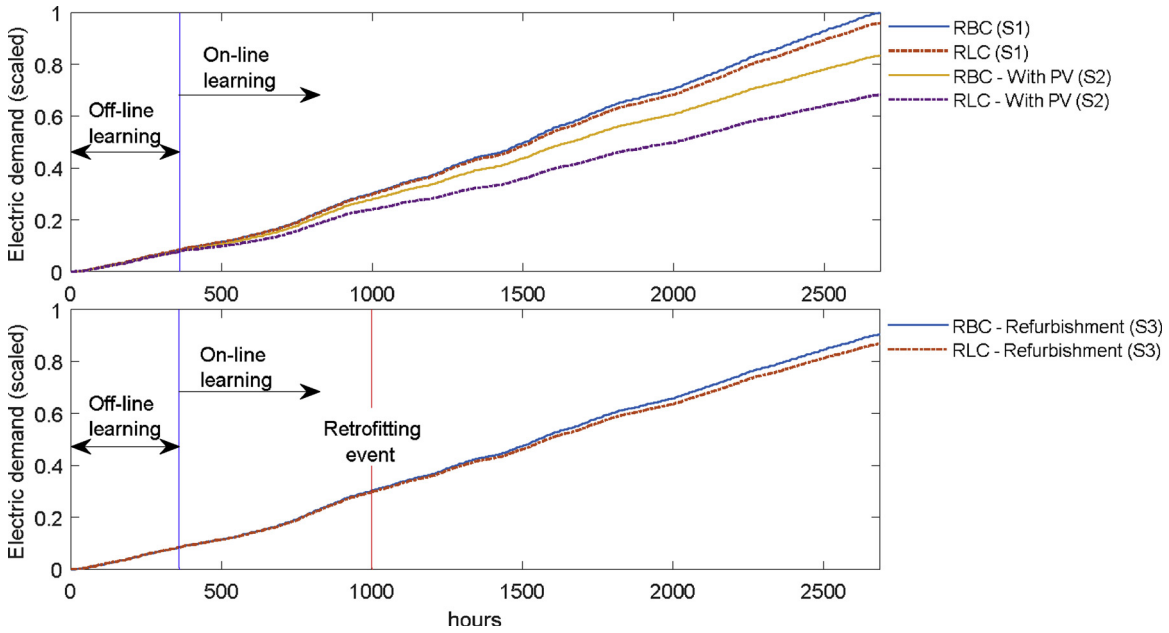

Fig. 11. Electricity consumption of the RLC with respect to the RBC. Top: Scenarios S1 (base case) and S2 (PV panels). Bottom: Scenario S3 (retrofit). Results are scaled to the RBC data of S1 for comparison.

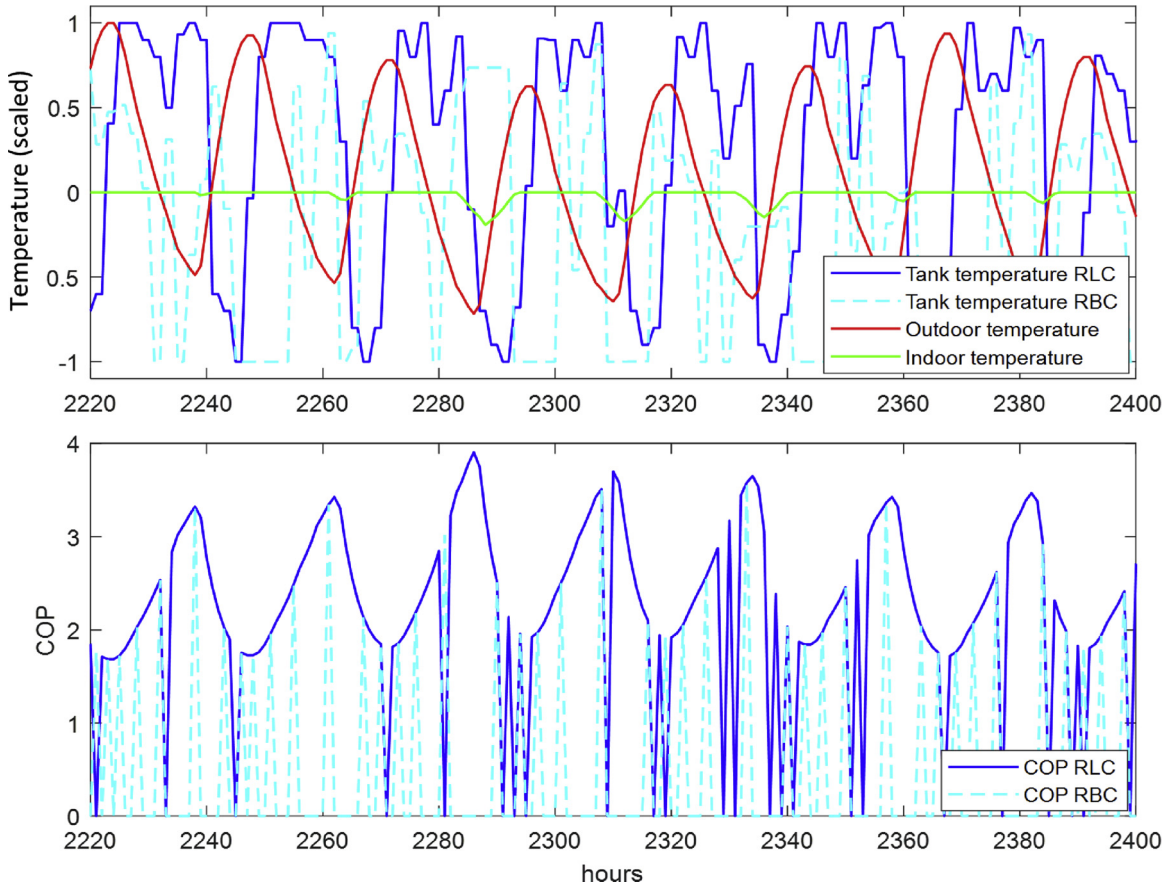

Fig. 12. Top: Indoor, outdoor and chilled water tank temperatures for both RBC and RLC. Bottom: COP of the heat pump for RBC and RLC.

When formulating the RL problem, we must define as states the variables we can observe, but not control, that have an impact on the probability of receiving a specific reward. As the actions, we must define those variables that we can control and which also have an impact on the probability of receiving a specific reward.

The action of the RL controller is the target temperature of the chilled water tank (for the next time-step), while the states are defined as the current temperature of the water in the tank, the outdoor temperature (which is a predictor of the energy demand in the building as well as of the coefficient of performance (COP) of the heat pump), the

Table 3
Electricity consumption (scaled) for different structures of DNN.

		1st hidden layer			
		3	4	5	6
2nd hidden layer	2	0.966	0.973	0.963	0.960
	3	0.986	0.969	0.968	0.972
	4	0.989	0.968	0.970	0.970

hour of the day, and the price of electricity. Indoor temperatures are always maintained between the appropriate temperature set-points, and they are never increased to achieve greater cost savings at the expense of thermal comfort. Since indoor temperature is maintained constant most of the time, it is not used as a state. In this case study, both the electricity prices and the COP of the heat pump have an influence on the overall cost of the electricity.

5.3. Reinforcement learning controllers

We compared three different controllers: a rule-based controller (RBC), a single-agent BRL controller, and a multi-agent BRL controller. As in the previous case study, the RBC cools the water in the chilled water tank every time it reaches 20 °C until it reaches 10 °C. On the other hand, both the single-agent, and the multi-agent BRL controllers use reinforcement learning in each building to adjust the temperature of the tank every two hours.

The single-agent BRL controllers reward their respective buildings by calculating an energy cost that uses a virtual electricity price, which is calculated from Eq. 4 using their individual electricity consumption as the input *E*. Thus, for this controller, reducing energy consumption and reducing electricity cost are equivalent. Note that this virtual price is only used to calculate the reward for each controller, whereas the real price both buildings pay is computed using the total electricity consumption of both buildings as the input *E* in Eq. 4. This virtual price (at the previous time-step) was also used as the state "Price of electricity" that Fig. 14 shows.

The multi-agent BRL controllers reward their respective building using their real cost of electricity, which is calculated using the electricity price the buildings share (using Eq. 4 with the sum of both electrical demands as the input *E*). Therefore, the multi-agent BRL controllers penalizes more the buildings if they consume electricity simultaneously, while the single-agent BRL controller only penalizes each of them, separately, for increasing their individual electricity consumption. The real price of electricity both buildings share (at the previous time-step) was also used as a state for every controller as Fig. 14 illustrates.

In a second experiment, we add a photovoltaic array on one of the buildings, covering 20% of its roof surface. We analyze how this can affect the electricity prices and whether the different controllers can adapt to this new situation.

5.4. Results

Fig. 15 shows how the single-agent BRL controller learned to cool the water tank when the outdoor temperature is low and the COP high, and discharge the cooling energy from the tank into the building when the outdoor temperature is high and the COP low. This control achieves greater energy cost savings than the RBC, which switches on and off without considering the outdoor temperature. This is the same behaviour as in the previous case study. On the other hand, the multi-agent BRL controller not only considers the outdoor temperature, but also the price of electricity both buildings share, which depends on the electrical consumption of the other building. Therefore, it did not follow a pattern intended to maximize the COP.

Fig. 16 illustrates the energy cost of each building using the three different controllers. The cost of electricity is scaled for a better visualization of the cost reductions. The RBC led to the highest electricity costs in both buildings. The reason for this is that it is an on-off controller, which makes use of the heat pump at full power capacity when the water tank reaches 20 °C until it reaches 10 °C. This creates spikes in the energy consumption of both buildings, leading to high electricity prices and costs.

On the other hand, both the single-agent, and the multi-agent BRL controllers achieved the same improvement with respect to the RBC. However, the single-agent BRL controller optimized the COP of the heat pump, while the multi-agent BRL controller did not. Therefore, the multi-agent BRL controller achieved some level of coordination between the buildings to make sure that the best price was paid rather than the COP maximized. This shows that even though we violate the Markovian property, we achieve similar results than the single-agent controller, which shows that coordination did happen.

5.5. Addition of a photovoltaic array

For the second experiment, Fig. 17 illustrates the temperature of the tanks for the building that does not have a photovoltaic array. While the single-agent BRL controller still focused on COP maximization to increase the energy cost savings, the multi-agent BRL controller learned how, during the day, the other building generated electricity and reduced the price of electricity. Therefore, the multi-agent BRL controller tended to cool the water tank when the power output from the photovoltaic array of the other building was higher.

Fig. 18 shows, how, in this second experiment, the multi-agent BRL controller of the building without PV panels achieves greater cost savings than the single-agent BRL controller. This is because it takes into consideration the impact that the photovoltaic generation of the other building has on the price of electricity.

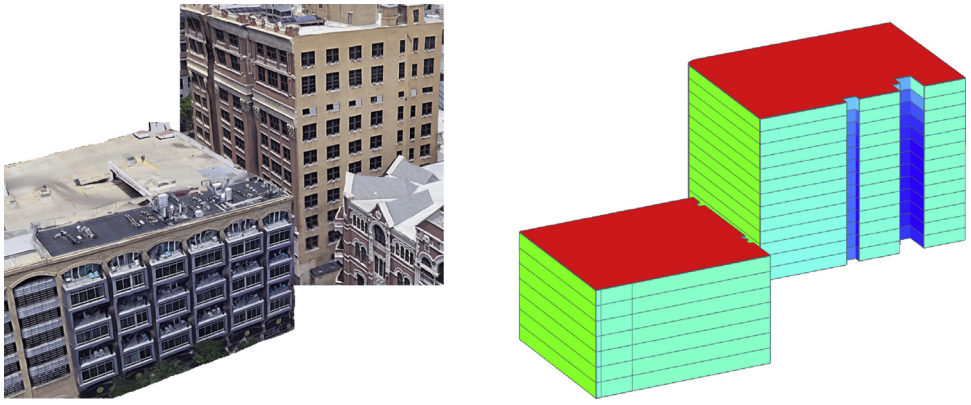

Fig. 13. Building envelopes and the representation of their building energy models in CitySim.

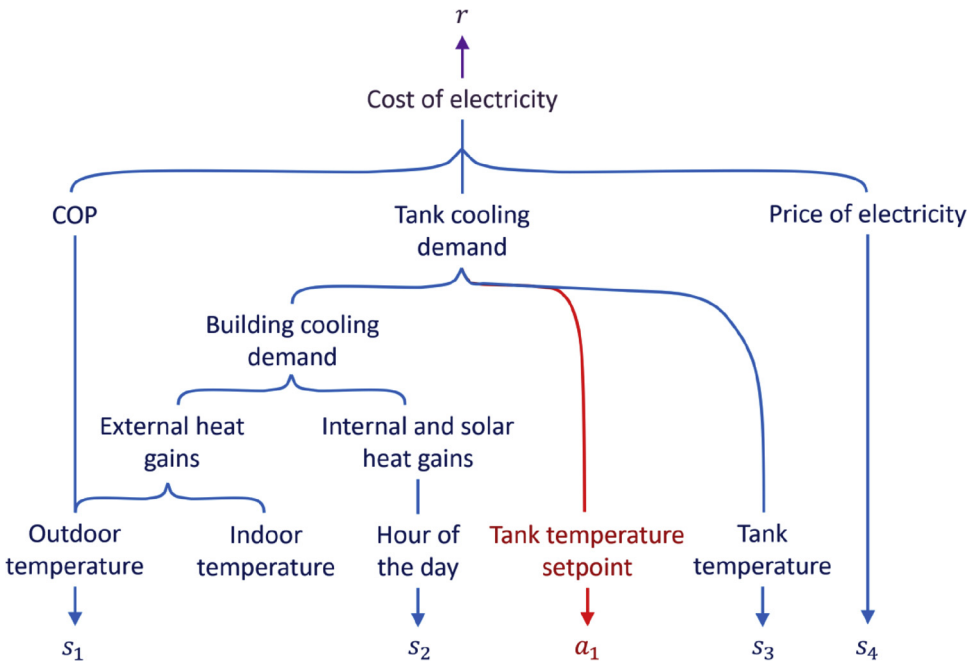

Fig. 14. State-action space to predict the reward.

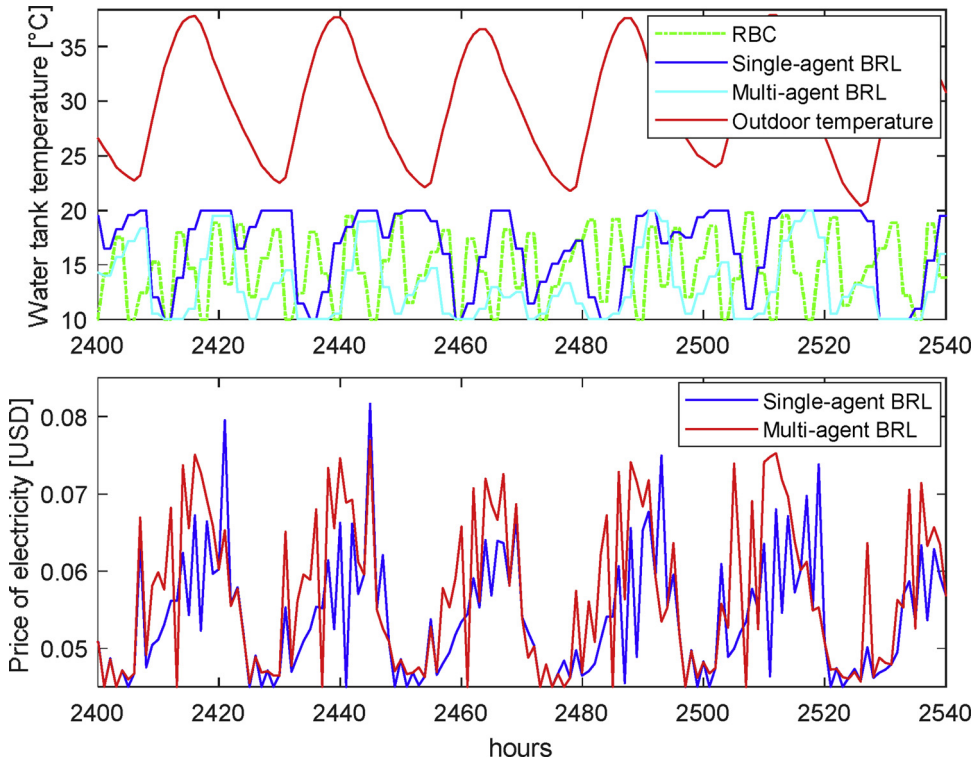

Fig. 15. Variations of the temperature of the water tanks with respect to the outdoor temperature, and electricity prices for one building and different controllers.

6. Discussion

Our simulation environment allows to analyze building energy management scenarios efficiently, and it is particularly useful when a machine-learning based controller must be run during the simulation. Capitalizing on the efficient implementation of algorithms provided by TensorFlow, and the validated building energy simulator CitySim, the results obtained are reproducible and extensible by the research community.

There are still open challenges in the implementation. One of these challenges is to improve the communication between CitySim and TensorFlow such that they do not need to share data through TXT or CSV files. This could be solved by compiling Keras and TensorFlow as a C++ library together with CitySim. This would also increase the computational speed. Another useful tool would be a graphical user interface (GUI) that allowed the user to write the machine learning code, or set hyper-parameters of the algorithms, after the simulation environment is compiled.

The simulation speed depends on both, the controller itself and the building energy simulation. The time of computation of the controller

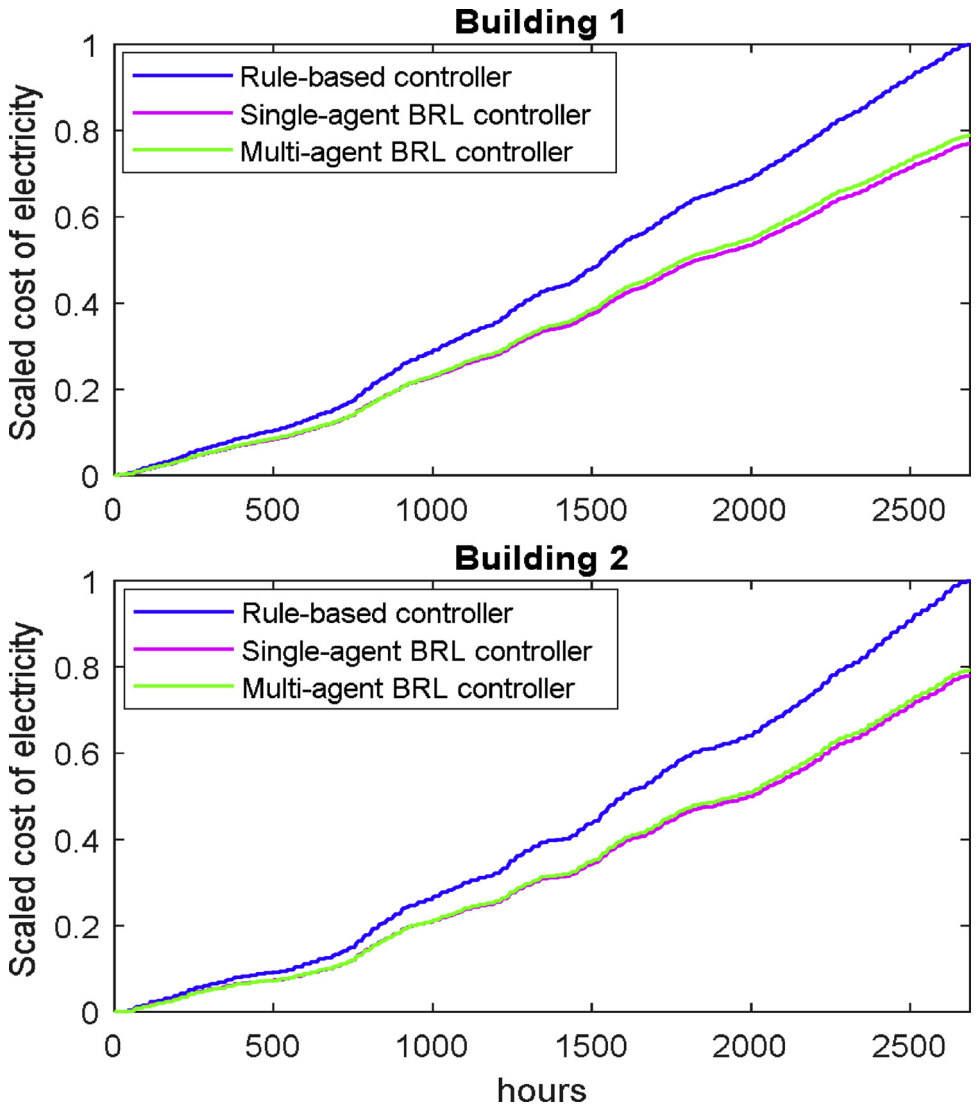

Fig. 16. Scaled electricity cost of both buildings using three different controllers.

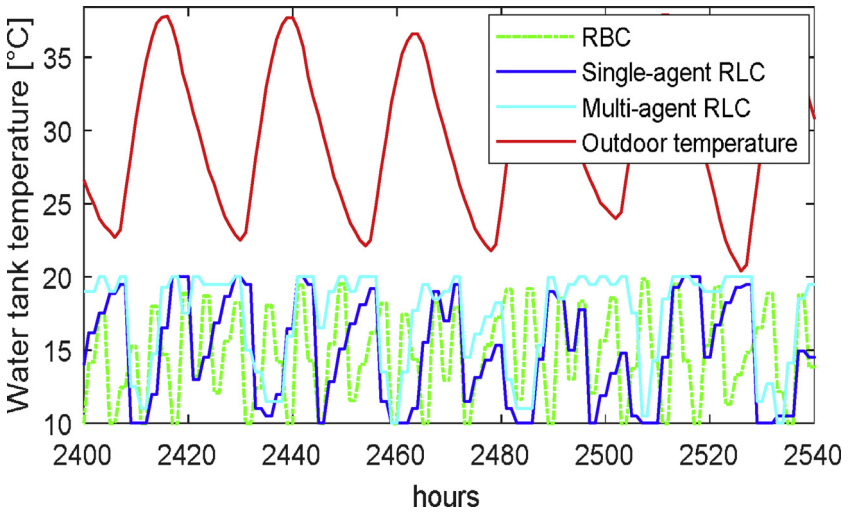

Fig. 17. Variations of the temperature of the water tanks, with respect to the outdoor temperature.

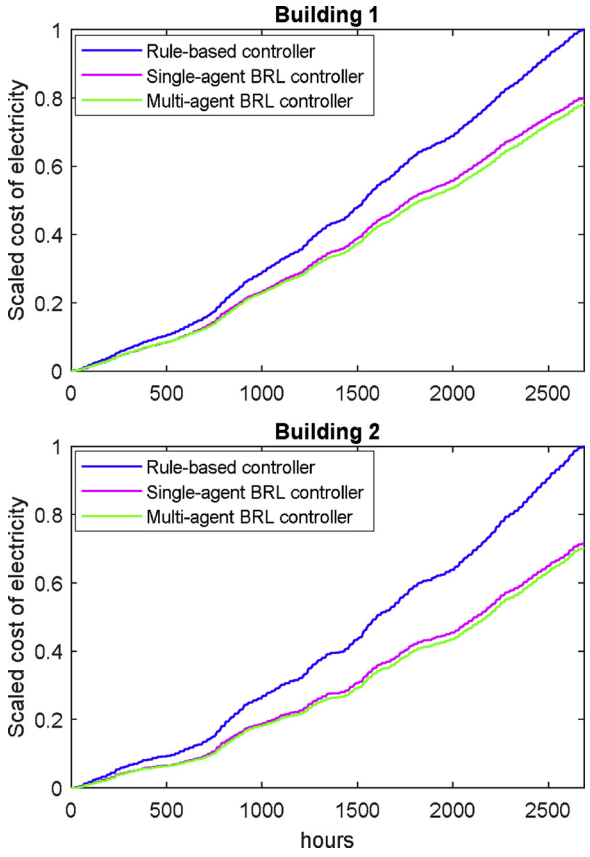

Fig. 18. Scaled electricity cost of both buildings using three different controllers.

mostly depends on are the size of the batches, the number of iterations required to make the Q-values converge, and the number of epochs used to train the neural network in each iteration. The use of a GPU greatly reduces the training time of the neural network, since it allows computing the training process in parallel. CitySim, in turn, benefits from the use of multi-threading in the CPU. However, effective multi-threading when multiple buildings are simulated and learn from each other, e.g., in a demand response scenario is still challenging. The main difficulty relies on the fact that all buildings need to finish their own computations before the simulation can advance to the following hour, at which point they share their data with each other and take new actions. Therefore, the speed of the simulation is limited by the building that takes the longest time to simulate.

We applied the developed integrated simulation environment in two case studies. In the first one, we used a single building in three different scenarios. We demonstrated the advantages of DRL in terms of its adaptability, its ability to learn without a model of the system being controlled, and its potential to learn from historical sensor data. These features are important because they make DRL especially suitable for residential buildings where deploying an MPC can be too expensive because a building energy model must be defined and identified.

Specifically, we showed how the RLC outperformed the RBC by 4% in the base-case scenario. However, it must be noted that the RLC does not start exploiting most of the optimal actions until the hour 1500 approximately (about half of the simulation period). If we only consider the improvement during the simulation period 1500 h–2688 h, then the energy savings would be near 10%, which is approximately what was achieved by refurbishing the building in S3. Surprisingly, in S2, the addition of solar PV panels contributed to the electricity savings as much as the RLC itself (about 15% each, and 30% in total compared to the base case).

Additionally, DRL is robust to changes in both the demand and the supply side, as we showed in our results. Even if the building is retrofitted or PV panels are added while the DRL controller is still learning, it can adapt to these changes and still learn the optimal control sequence.

Our case studies focused on a single rather constant outdoor season to demonstrate the ability of the RLC to learn. This may give the wrong impression that the operation during multiple seasons throughout the year is not possible. In fact, it has been demonstrated, e.g., in (Yang et al., 2015) that learning throughout the year, is possible by, e.g., using a different neural network for each season.

DRL makes use of a DNN to determine which actions the controller should take. Since a DNN is a black-box model, its use can introduce safety concerns. That is why we demonstrated that DRL can learn off-line from a back-up controller (the RBC in this case). This allows the DRL controller to use historical data first as base knowledge before taking any decisions, and then start improving the control sequence taking its own actions. Limits of operation can be defined, and if the DRL controller crosses those limits, it can switch back to a back-up controller, e.g., RBC, and continue learning off-line from the new data that will be collected (Yang et al., 2015).

In the second case study we demonstrate the feasibility of coordinating buildings using reinforcement learning in this simulation environment for demand response scenarios. We proved the adaptability and robustness of this algorithm by testing it under different scenarios with and without photovoltaic generation. In both cases it manages to reduce the electricity costs of the buildings simulated.

In addition, we observed an interesting, emerging behaviour when one of the buildings had PV panels, i.e., free electricity, to cover some of its demand. The building without PV managed to adapt its behaviour such that it consumed electricity during the times that the other building generated its own, because the lower total net demand meant lower electricity price. We expect to see more of such, emergent behaviour, when we simulate more buildings with heterogeneous HVAC systems, and dissimilar demand profiles.

Both case studies show the reinforcement learning capabilities to find control patterns and generate emerging behaviours that minimize energy consumption and costs in an adaptive, resilient, and self-tuning way. They also prove how this integrated simulation environment allows the easy implementation of these types of controllers based on deep learning, and testing their suitability for the built-environment.

The implementation of an RLC in a building or an urban setting as presented in this paper would require the use of temperature sensors to measure the outdoor temperature, the water temperature in the tank, and potentially the indoor air temperature of the building. A power meter could measure the power output of the photovoltaic PV panels when necessary, and all these variables would be used as states by the RLC. There would be several options for its implementation in a real setting, which would depend on the limitations of the current building energy management system (BEMS):

a) If the current BEMS allows running Python scripts and recording many several months of data, the RLC could be programmed in the BEMS after installing the TensorFlow library.
b) If the current BEMS does not allow running Python scripts, a separate control server that stores all the sensor data, and communicates with the BEMS over a data acquisition and control network protocol (e.g., BACNet) can be set up in the building. The python script would be run on the server and would compute the control signals for the BEMS.
c) For urban scale implementation and/or inter-building communication (i.e. sharing variables or electricity prices), all the data can be stored in a server in the cloud. The RLC would run in this centralized server and communicate the control actions to each individual BEMS.

7. Conclusion

The rise of information and communication technologies and the internet of things, has led to the collection of an increasing amount of data on buildings, and building operation. From these data, novel and improved machine learning techniques, such as deep learning, and the increase of computational power of computers will allow us to extract useful information, and generate actionable knowledge.

In this paper, we have introduced a new simulation environment that is the result of merging CitySim, a building energy simulator, and TensorFlow, a powerful machine learning library. This new simulation environment has the potential for developing building energy scenarios in which machine learning algorithms, such as deep reinforcement learning, are applied to of the major problems and opportunities modern cities face, e.g., the increased demand for heating and cooling due to increasing populations.

This simulation environment allows to study model-free and self-tuning control algorithms, such as deep reinforcement learning (DRL) when integrating distributed renewable energy sources and storage devices into buildings. DRL can learn on-line and off-line from historical sensor data, and it can adapt to diverse changes in the system it controls on both the demand and the supply side. Its off-line learning feature allows it to be safely implemented with a back-up controller and operational constraints.

There are many potential applications beyond what has been demonstrated in our case study. For instance, the results of a building energy simulation at the urban level could be used to derive surrogate models, which can be then used for faster system optimization purposes. Furthermore, provided with econometric, urban development and weather models projected over several decades, the adaptation of the energetic behaviour of the building stock to those variations can be studied.

In conclusion, our powerful simulation environment will enable studying smart cities from an energetic perspective by simplifying the implementation of advanced machine learning algorithms to process the results, implement new control solutions, or feed the simulation models with data. This can support planners and architects by generating guidelines on the energetic impact of their decisions on the built environment, and researchers can demonstrate the effectiveness of advanced control algorithms based on machine learning.

References

Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Greg, S., et al. (2015). *TensorFlow: Large-scale machine learning on heterogeneous distributed systems*.

ASHRAE (2014). *Standard 90.1*.

Busoniu, L., Babuska, R., De Schutter, B., & Ernst, D. (2010). *Reinforcement learning and dynamic programming using function approximators.* 260. https://doi.org/10.1201/9781439821091.

Capozzoli, A., Piscitelli, M. S., & Brandi, S. (2017). Mining typical load profiles in buildings to support energy management in the smart city context. 9th international conference on sustainability in energy and buildings. *Energy Procedia, 134*, 865–874. https://doi.org/10.1016/j.egypro.2017.09.545.

Chollet, F. (2015). *Keras.* https://github.com/fchollet/keras.

De Somer, O., Soares, A., Kuijpers, T., Vossen, K., Vanthournout, K., & Spiessens, F. (2017). *Using reinforcement learning for demand response of domestic hot water buffers: A real-life demonstration.* 1–6. https://arxiv.org/pdf/1703.05486.pdf.

Dupont, B., De Jonghe, C., Olmos, L., & Belmans, R. (2014). Demand response with locational dynamic pricing to support the integration of renewables. *Energy Policy, 67*, 344–354. https://doi.org/10.1016/j.enpol.2013.12.058 Elsevier.

Ernst, D., Geurts, P., & Wehenkel, L. (2003). "Iteratively extending time horizon reinforcement learning. *Machine Learning: ECML 2003: 14th European Conference on Machine Learning, 14*, 96–107. https://doi.org/10.1007/b100702.

Gullapalli, V. (1990). A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks, 3*(6), 671–692. https://doi.org/10.1016/0893-6080(90)90056-Q.

Haldi, F., & Robinson, D. (2011). The impact of occupants' behaviour on building energy demand. *Journal of Building Performance Simulation, 4*(4), 323–338. https://doi.org/10.1080/19401493.2011.558213.

IEA (2013). *Transition to sustainable buildings.* https://doi.org/10.1787/9789264202955-en.

Javed, A., Larijani, H., Ahmadinia, A., Emmanuel, R., Mannion, M., & Gibson, D. (2017). *Design and implementation of a cloud enabled random neural network-based decentralized smart controller with intelligent sensor nodes for HVAC, 4*, 393–403 2.

Kalyanakrishnan, S., Stone, P., & Liu, Y. (2008). Batch reinforcement learning in a complex domain. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 5001 LNAI,* 171–183. https://doi.org/10.1007/978-3-540-68847-1_15.

Leibowicz, B. D., Lanham, C. M., Brozynski, M. T., Vázquez-Canteli, J. R., Castillo, N., & Nagy, Z. (2018). Optimal decarbonization pathways for urban residential building energy services. *Applied Energy, 230*(May), 1311–1325. https://doi.org/10.1016/j.apenergy.2018.09.046 Elsevier.

Liu, S., & Henze, G. P. (2007). Evaluation of reinforcement learning for optimal control of building active and passive thermal storage inventory. *Journal of Solar Energy Engineering, 129*(2), 215. https://doi.org/10.1115/1.2710491.

Marinakis, V., Doukas, H., Karakosta, C., & Psarras, J. (2013). An integrated system for buildings' energy-efficient automation: Application in the tertiary sector. *Applied Energy, 101*, 6–14. https://doi.org/10.1016/j.apenergy.2012.05.03.

Meteonorm (2018). *Meteonorm.* http://www.meteonorm.com/.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013). *Playing atari with deep reinforcement learning.* https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf.

Mocanu, E., Mocanu, D. C., Nguyen, P. H., Liotta, A., Webber, M. E., Gibescu, M., et al. (2017). *On-line building energy optimization using deep reinforcement learning.* 1–9.

Mozer, M. C. (1998). The neural network house: An environment that adapts to its inhabitants. *American Association for Artificial Intelligence Spring Symposium on Intelligent Environments,* 110–114 doi:SS-98-02/SS98-02-017.

Nagy, Z., Park, J. Y., & Vázquez-Canteli, J. R. (2018). Reinforcement learning for intelligent environments: A Tutorial. In P. Gardoni (Ed.). *Handbook of sustainable and resilient infrastructure*Routledge ISBN-13: 978-1138306875.

Park, J. Y., & Nagy, Z. (2018). Comprehensive analysis of the relationship between thermal comfort and building control research - a data-driven literature review. *Renewable and Sustainable Energy Reviews, 82*(October), 2664–2679. https://doi.org/10.1016/j.rser.2017.09.102 Elsevier Ltd.

Prívara, S., Cigler, J., Váňa, Z., Oldewurtel, F., Sagerschnig, C., & Žáčeková, E. (2013). Building modeling as a crucial part for building predictive control. *Energy and Buildings, 56*, 8–22. https://doi.org/10.1016/j.enbuild.2012.10.024.

Rault, A. (1978). *Model predictive heuristic control : Applications to industrial processes *, 14*, 413–428.

Robinson, D. (2011). *Computer modelling for sustainable urban design.* London: Earthscan.

Ruelens, F., Iacovella, S., Claessens, B. J., & Belmans, R. (2015). Learning agent for a heat-pump thermostat with a set-back strategy using model-free reinforcement learning. *Energies, 8*(8), 8300–8318. https://doi.org/10.3390/en8088300.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction.* Cambridge, Massachusetts: MIT Press.

Urieli, D., & Stone, P. (2013). A learning agent for heat-pump thermostat control. In: *Proc. 12th Int'l Conf Autonomous Agents and Multiagent Systems,* 1093–1100.

Vázquez-Canteli, J. R., & Nagy, Z. (2019). Reinforcement learning for demand response: A review of algorithms and modeling techniques. *Applied Energy, 235*, 1072–1089. https://doi.org/10.1016/j.apenergy.2018.11.002.

Vázquez-Canteli, J., & Kämpf, J. (2016). Energy simulation at the urban scale: A focus on Geneva and climate change scenarios. *Sustainable City, 204*. https://doi.org/10.2495/SC160041 Sc.

Walter, E., & Kämpf, J. H. (2015). *A verification of CitySim results using the BESTEST and monitored consumption values. Proceedings of the 2nd Building Simulation Applications Conference*215–222. https://infoscience.epfl.ch/record/214754.

Watkins, C. J. C. H., & Dayan, P. (1992). Technical Note: Q-Learning. *Machine Learning, 8*(3), 279–292. https://doi.org/10.1023/A:1022676722315.

Yang, L., Nagy, Z., Goffin, P., & Schlueter, A. (2015). Reinforcement learning for optimal control of low exergy buildings. *Applied Energy, 156*, 577–586.

Zhun, Y., Haghighat, F., & Fung, B. C. M. (2016). Advances and challenges in building engineering and data mining applications for energy-efficient communities. *Sustainable Cities and Society, 25*, 33–38.