Les Cahiers du GERAD ISSN: 0711–2440

Autonomous control in smart buildings: A deep reinforcement learning approach

Y. Desage, F. Bouffard,
F. Bastin, J.-S. Venne

G–2020–30

May 2020

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2020
– Bibliothèque et Archives Canada, 2020

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2020
– Library and Archives Canada, 2020

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Autonomous control in smart buildings: A deep reinforcement learning approach

Ysaël Desage [a]

François Bouffard [b]

Fabian Bastin [a]

Jean-Simon Venne [c]

[a] *Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal (Québec) Canada, H3T 1J4*

[b] *GERAD & Department of Electrical and Computer Engineering, McGill University, Montréal (Québec) Canada, H3A 0E9*

[c] *Brainbox AI, Montréal (Québec) Canada, H3A 2L1*

ysael.desage@me.com
francois.bouffard@mcgill.ca
bastin@iro.umontreal.ca
jsvenne@brainboxai.com

May 2020
Les Cahiers du GERAD
G–2020–30

Abstract: Deep learning has redefined modern standards and performance in several areas such as computer vision and natural language processing. With increasing amounts of frequently sampled data in advanced metering infrastructure, similar opportunities are readily available for smart grid actors' optimization. In this regard, we consider the problem of remote high-granularity control with low computational power in deployment and intermittent connectivity for heating, ventilation, and air-conditioning components in smart buildings. Thereupon, we introduce an adapted autonomous multi-system command infrastructure based on deep reinforcement learning. Through several deployment safety measures, we demonstrate significant improvements in expenses, thermal comfort, energy consumption, power peaks and equipment cycling using an adaptation of the Deep Q-Learning algorithm on case studies of physics-based simulations relying on real historical weather data. We quantify the resulting optimization and illustrate both the scalability and flexibility of our approach by comparing the trained controller to its classical reactive counterparts on instances requiring simultaneous control on up to seven parallel systems.

Keywords: Buildings, deep learning, deep reinforcement learning, energy consumption, optimal control, optimization, power consumption, smart grid

Résumé : L'apprentissage profond a redéfini les normes modernes et la performance dans les domaines de la vision informatique et du traitement du langage. Avec l'accroissement des données provenant des infrastructures de mesure de l'énergie dans les réseaux électriques, nous assistons à un foisonnement des occasions d'optimisation dans ces réseaux. Nous considérons donc ici le problème de commande optimale des systèmes de chauffage, climatisation et ventilation d'un bâtiment intelligent avec une infrastructure de calcul à basse puissance et ne nécessitant que peu de communications. Pour ce, nous introduisons une méthode de commande autonome, multi-système basée sur l'apprentissage par renforcement profond. Malgré l'application de plusieurs mesures assurant un niveau de service thermique minimal, nous démontrons des améliorations notoires en termes de coûts d'énergie, de puissance, de confort thermique et de nombre de cycles en utilisant une adaptation de l'approche de Q-apprentissage profond sur des cas d'espèce basés sur des simulations de modèles physiques calibrés sur des données météo historiques. Nous quantifions les résultats de l'optimisation et illustrons sa flexibilité et comment cette méthode peut facilement être étendue à des édifices de plus en plus grands en comparant sa performance avec les contrôleurs classiques réactifs.

Acknowledgments: This work was supported in part by Mitacs and by BrainBox AI labs.

1 Introduction

Energy consumed in the buildings sector, both residential and commercial, accounts for near 40% of the total worldwide energy consumption [1], and beyond 30% of CO_2 emissions [2]. Considering that 230 billion square meters of new construction is expected over the next 40 years [3], such numbers make smart buildings one of the major actors in the modern power grid's infrastructure.

From this perspective, in this paper we consider the multi-objective problem of optimizing expenses, thermal comfort, power peaks, energy consumption and equipment cycling in smart buildings equipped with multiple air-handling units (AHU) such as Roof Top Units (RTU), and connected to the electrical grid. We assume control will be applied with high granularity (decision epochs every 15 minutes), from a remote low-computational power device having local access to all the heating, ventilation and air-conditioning (HVAC) components inside the building (see Figure 1). For application realism purposes, we also consider sparse intermittent connectivity with the remote control device, making it independent from the central computing source except for occasional data transfers. Finally, we impose a set of pre-defined fixed constraints on thermal values for the safety of the users inside the buildings, which will automatically trigger a fall-back position to classical controls if needed.

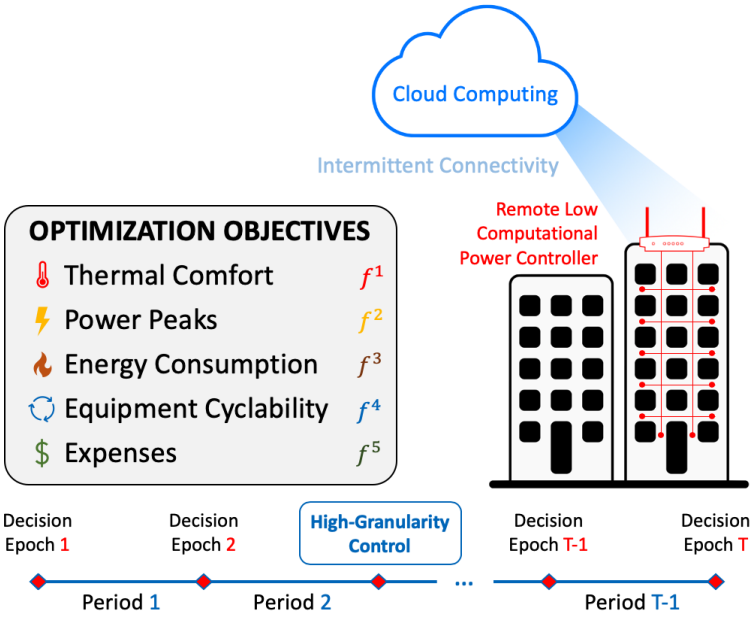

Figure 1: Remote high-granularity control problem considered.

The considered multi-objective optimization problem can be expressed as a single objective problem using weighted contributions of the form:

$$\max \sum_{t=1}^{T} \left\{ \alpha_t^1 f_t^1 + \alpha_t^2 f_t^2 + \alpha_t^3 f_t^3 + \alpha_t^4 f_t^4 + \alpha_t^5 f_t^5 \right\} , \tag{1}$$

where t indexes time up to a horizon T, and f_t^i and α_t^i, $i \in \{1, 2, ..., 5\}$, are the different objective functions as depicted in Figure 1, and their respective weighting coefficients. This optimization is subject to the thermodynamics of the building, the occupants' safety measures, and the technical specificities of the HVAC equipments.

Numerous approaches have been proposed in literature to address smart building control, including dynamic programming [4], [5], model-predictive control [6], and machine learning [7], [8], to name but a few. While demonstrating successful results, such methods are either very compositionally expensive in deployment, require continuous two-way communication protocols, imply numerous hours of building-specific engineering, or simply do not scale up properly to large instances.

In contrast, our proposed deep reinforcement learning solution leverages the rich amount of temporal observations through a specialized recurrent deep learning architecture, resolves the important exponential action space growth of larger parallel control instances, and allows an efficient global optimization through a shared parameter setting. Lastly, our solution can be easily deployed in its full potential requiring only a few matrix multiplications on the deployment device.

2 Reinforcement Learning

Reinforcement Learning (RL) is one of the three main machine learning (ML) paradigms where training information is used to evaluate actions (rather than instruct), in order to maximize a numerical reward signal defined in accordance to a specific goal [9]. RL relies on the framework of Markov Decision Processes (MDP), where an agent a undergoes continuous or episodic interactions with its environment. At each step of a sequence of discrete time steps t, up to a horizon $T \in [0, \infty)$, the decision maker receives a partial observation $o_t \in \mathcal{O}$ of the real state $s_t \in \mathcal{S}$ the environment E is currently in, where $\mathcal{O}$ and $\mathcal{S}$ represent the discrete finite sets of observations and states, respectively. Based on the perceived observation, the controller then applies its policy π to choose a control u_t from a set of allowable actions $\mathcal{U}_s$,[1] which triggers a transition of the environment according to the probability function $P_t(s_{t+1}|s_t, u_t)$ into a new state s_{t+1} and returns a new (partial) observation o_{t+1} along with a reward $r_t \in \mathbb{R}$. The objective of the decision maker is to maximize the expected cumulative sum of such reward, called the *return*:

$$R_t = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t(s_t, u_t)\right] \ , \tag{2}$$

where the discounting factor $\gamma \in [0, 1)$ accounts for the desirability of short versus long-term rewards. From this, we define the Q-function (or Q-factors) [10] of a given policy π as the total expected return from being in state s, applying action u and thereafter following policy π:

$$Q^\pi(s, u) = \mathbb{E}\big[R_t|s_0 = s, u_0 = u, \pi\big] \ . \tag{3}$$

It is common to use a function approximator to estimate this or other quantities in RL. When a neural network is used for this purpose, we typically use the term *deep reinforcement learning* (DRL).

2.1 Deep Q-Learning

The optimal Q-factors can be defined as [11]:

$$Q^\star(s, u) = \max_\pi \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t(s_t, u_t)|s_0 = s, u_0 = u\right] \ , \tag{4}$$

which obey an important recursive relation known as the *Bellman Equation*: starting from the intuition that if the optimal value $Q^\star(s', u')$ of the state $s_{t+1} = s'$ at the next time-step was known for all possible actions u_{t+1}, then the optimal strategy is to select that action u' maximizing the expected value of $r + \gamma Q^\star(s', u')$. Algorithms of the classical *Q-Learning* [12] family leverage this principle and convert the Bellman equation into an iterative update of the form

$$Q_{i+1}(s_t, u_t) = \mathbb{E}_{s_{t+1}}\big[r + \gamma \max_{u_{t+1}} Q_i(s_{t+1}, u_{t+1})|s_t, u_t\big] \ . \tag{5}$$

The *Deep Q-Network* (DQN) [13] algorithm uses a deep neural network as a function approximator to predict $Q(s, u; \theta) \approx Q^\star(s, u)$, where θ denotes the parameters of the network. These parameters are updated periodically as a supervised learning task using a mean-squared error loss $\mathcal{L}$ of the difference between the target $y = r + \gamma \max_{u'} Q(s', u'; \theta_i^-)$ and the old estimate $Q(s, u; \theta)$:

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,u,r}\Big[\big(y - Q(s, u; \theta_i)\big)^2 + \mathbb{E}_{s,u,r}\big[\mathrm{Var}'_s(y)\big]\Big] \ , \tag{6}$$

[1] It is typical in RL to consider the same set of actions $\mathcal{U}_s = \mathcal{U}\ \forall s \in \mathcal{S}$.

where θ_i^- and $\mathbb{E}_{s,u,r}\left[\mathrm{Var}'_s(y)\right]$ represent the network's parameters from a previous iteration and the expected variance of the target, respectively. Crude Monte Carlo estimates, or Sample Average Approximation (SAA) are then typically used in conjunction with incremental gradient methods to converge to a solution.

Lastly, [13] also introduces *experience replay* to stabilize learning and smooth out the training distribution, which consists of storing experience tuples $e_t = (s_t, u_t, r_t, s_{t+1})$ in a memory buffer. During the inner training loop of the algorithm, training examples are then randomly selected and mini-batch Q-learning updates are performed with respect to (6). This characteristic makes the DQN an *offline* algorithm, where the agent learns a different optimal policy from the one used to act in the environment during training.

3 Methodology

Given the sequential nature arising from time dependency in the partial observations o_t, we consider a sequence-adapted bidirectional Long Short-Term Memory (LSTM) architecture as function approximator for the DRL (details are illustrated in Figure 2). We refer the interested reader to Appendix A for a more thorough presentation of essential deep learning concepts.

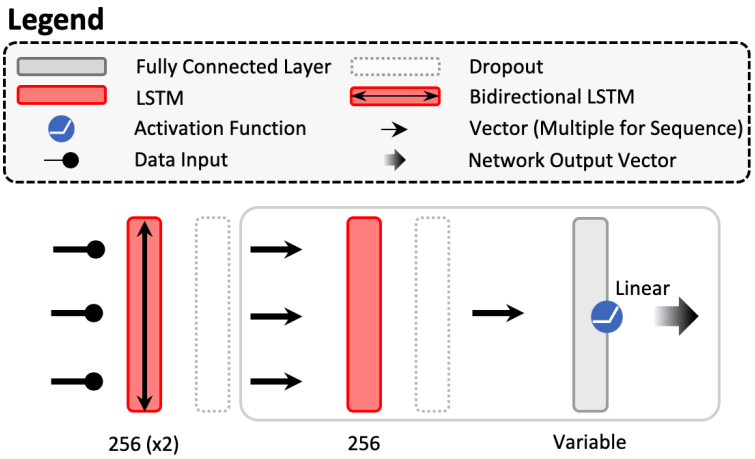

Figure 2: Deep bidirectional LSTM architecture used as function approximator. The grey squares represent the duplicated components for each independent Q-function output. Activations are not explicitly shown for the LSTM layers.

From the reinforcement learning perspective, we extend the classical DQN framework by implementing its double Q-learning variant [14]. We then use the aforementioned neural network architecture to directly map observation sequences $\{o_{t-H}, ..., o_{t-1}, o_t\}$ to Q-values, where H denotes the *observation history* considered by the agent.[2] Moreover, we further leverage the deep learning infrastructure by considering not only a single Q-value vector for all the possible actions like in the original DQN algorithm, but rather parallel Q-value output streams for every present control system (see Figure 3). This can be seen as a simplistic multi-agent framework adaptation, as it allows both an important action space factorization, and the possibility to provide system-specific actions and granularities for each controller. It is also computation-efficient because it requires only a single forward pass calculation, while still benefiting from shared parameters which offer a global generalization and optimization scheme.

Despite the high potential for performance exploration and tuning, we purposely fix the hyper-parameters of the DRL algorithm with the objective of testing application robustness and flexibility in a broad range of different instances, while reducing the associated computational burden. This also useful for assessing deployment autonomy, as no post-implementation interventions are performed.

[2] To rigorous intents, $\{o_{t-H}, ..., o_{t-1}, o_t\}$ can be considered as the DRL agent's *state* in itself, to respect the Markovian property.

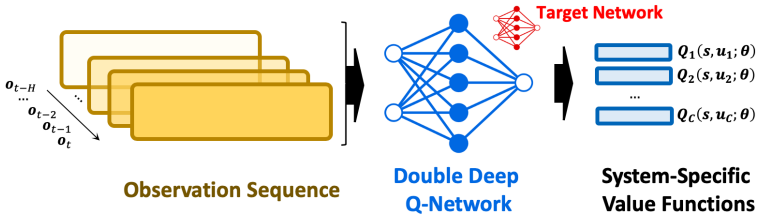

Figure 3: Proposed Double Deep Q-Network using time series adapted deep learning architectures to map a sequence of observations $\{o_{t-H}, ..., o_{t-1}, o_t\}$ to individual parallel Q-value streams, one for each of the C control systems.

4 Case studies

4.1 Case studies' setup

We perform our proof of concept through building simulations by accounting the interaction of all the major thermodynamics actors in the system: shared walls and doors between zones, open spaces and room content, external temperature, users activities, solar radiance, and HVAC components behavior. Real historical weather temperature is used to drive the boundary conditions of the simulation, internal thermal properties are chosen to represent archetype building types and characteristics, and the HVAC systems considered are RTUs with two heating stages and two cooling stages for a total of five possible control indexes if we include the "off" position. Finally, human activities and solar contributions are sampled from normal distributions with parameters varying depending on each building's nature and schedule.

For building types, we consider the three following distinct instances: a house (or an apartment) with two AHUs consuming 10kW for stage 1, and 15kW for stage 2; a retail store with three similar but larger systems having a 20kW stage 2; and finally a small commercial center with seven independent controllers identical to the retail store. We assume a 90% efficiency on all equipment, meaning that 90% of the input power is converted into heating or cooling. Each building instance (see Figure 4) is represented by its own thermal circuit (the reader can refer to the Appendix B for more technical thermodynamics and modeling details) and includes heat contributions from users, solar radiance and HVAC components in each individual zone (see Figure 5 for simulation parameters). The comfort zone is defined to be between 19.5°C and 22.5°C, but while it is kept constant in instance A, set points are scheduled in the two other instances to adjust the dead band between 16°C and 26°C during the non-occupancy hours, from 6 PM to 6 AM. The unitary kWh electricity price varies during the day, taking the value of 0.132$ from 7 AM to 11 AM, 0.095$ from 11 AM

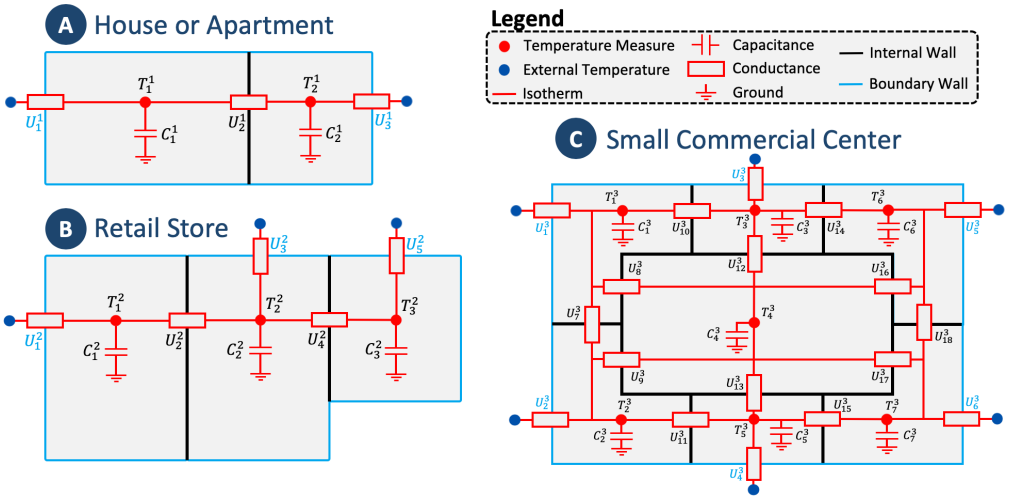

Figure 4: Thermal RC circuit instances considered in the building simulation. Independent solar, human and HVAC heat contributions are added at every internal temperature measurement point, and the external temperature is provided from real historical data.

to 3 PM, 0.132$ from 3 PM to 7 PM, and 0.05$ in the remaining hours. Power is charged based on the month's highest peak, at the rate of 14.58$/kW. While a realistic simulation is targeted, our main objective is also to introduce a high level of variation and uncertainty, to illustrate the generalization and adaptation capacity of our DRL controller.

INSTANCE A	
$U_1 = 300, U_2 = 100, U_3 = 250$	W/K
$C_1 = 11 \times 10^6, C_2 = 11 \times 10^6$	J/K
INSTANCE B	
$U_1 = 400, U_2 = 1000, U_3 = 200, U_4 = 750, U_5 = 300$	W/K
$C_1 = C_2 = 12 \times 10^6, C_3 = 10 \times 10^6$	J/K
INSTANCE C	
$U_1 = U_2 = U_3 = U_4 = U_5 = U_6 = 200$	
$U_7 = U_{10} = U_{11} = U_{14} = U_{15} = U_{18} = 500$	W/K
$U_8 = U_9 = U_{12} = U_{13} = U_{16} = U_{17} = 1000$	
$C_1 = C_2 = C_6 = C_7 = 11 \times 10^6, C_4 = 12 \times 10^6$	
$C_3 = C_5 = 10 \times 10^6$	J/K

Figure 5: Value of the parameters used in the simulation.

To increase application realism, we perform training in a centralized fashion from a cloud computing resource, which then transfers the weights matrix to the local remote controller. During deployment, only the results of the forward pass are accessible to this device, with weight updates possible only through a predefined schedule. To assess performance on different deployment steps, with increasing amounts of available data, we divide the simulation into two phases:

- Phase 1: One continuous month deployment with an initial training on two months of similar conditions.
- Phase 2: Year-round deployment, with 8 months of initial training data and no weight update.

Phase 1 is evaluated on January 2020 (one of the coldest months) and trained from November 2019 to the end of December 2019, while phase 2 is trained from July 7, 2018 to March 14, 2019, then tested from March 15, 2019 to March 15, 2020 (see Figure 6).

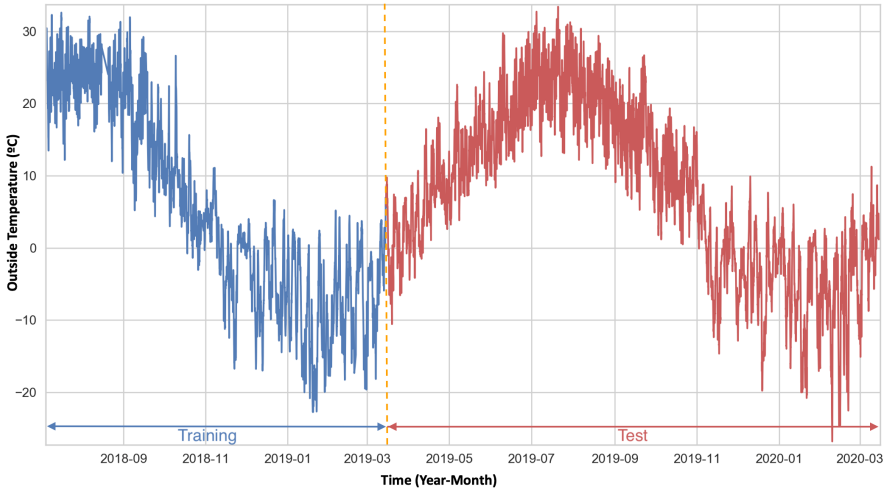

Figure 6: Outside temperature in Montreal (QC), Canada, from July 7 2018 to March 15 2020. Data originates from the Dark Sky API [15], and the training data for phase 2 is depicted in blue, while the test deployment data is shown in red.

For the RL side of the simulation, the environment's partial observation includes the following information from the previous hour up to 15 minutes before the present time: temporal iteration's value t, and cyclic features of the form $\sin(2\pi t/T)$ and $\cos(2\pi t/T)$ where T is the episode's length; the HVAC systems' index; internal and external temperature; temperature set points; energy consumed in each zone; power calls in each

zone; and highest building power peak since the beginning of the month. Three actions are considered for each HVAC controller: heat (increases control index), do nothing (leaves index unchanged), or cool (lowers control index). Cooling stage 2 is represented by the lowest control index 0, while heating stage 2 is indexed by the maximum, 4. The reward process is built as follows: the agent receives a periodic reward of 50 at each time step while being between the temperature set points, and -20 if outside of the range, to stimulate comfort; a -1000 penalty each time the temperature is 2 degrees above or below the set points; a negative reward proportional to the power consumption each time the current highest power peak of the month is exceeded; a penalty linearly scaling to the energy cost at each time step; and finally a -15 reward for each action different from "do nothing", to reduce useless toggling. As a safety measure, the agent is replaced by classical controls until the temperature is back within comfort range when temperature exceeds or falls bellow 2 degrees of the set points. The resulting mathematical system for the agent is to maximize its return for a 24-hours period, while being exposed to all the combined reward mechanisms described above.

4.2 DQN training parameters

The training phase of the modified DQN consists in 50 000 epochs of 24-hours episodes simulation, using an ϵ-greedy policy where either a random action is chosen for every controller with probability ϵ, or a stochastic policy using the vector of normalized Q-values as a probability distribution over the actions (sometimes referred to as *Boltzmann Policy* [16]) is applied instead. Using this methodology bolsters appropriate exploratory behavior in the long term, while still allowing convergence towards a final random optimal policy, if needs be (unlike the original algorithm always deterministically using the *arg max* of the Q-factors). Starting with $\epsilon = 1$, we perform a linear decrement to reach a floor value of 0.05 at 80% of the training epochs. Optimization is achieved with the *Adam* [17] gradient descent algorithm, with constraints to limit the gradient norm between -1 and 1 to avoid instability arising from a major update. Finally, a scheduled learning rate is applied with a starting value of 0.001, reduced to 5×10^{-4} at 30% of training, then finally set to 1×10^{-4} from 60% to the end of the computation.

4.3 Results and discussion

We assess the test simulations by defining Key Performance Indicators (KPI) with respect to each initial target objective: total and individual costs for the expenses; average daily discomfort time for thermal comfort; average daily energy consumed for energy consumption; highest overall peak for power peaks; and total number of performed cycles for equipment cyclability. Following these definitions lead to the conclusion that the multi-objective optimization is successful overall, as the DQN controller outperforms or equals its classical reactive peer for all KPIs in phase 1 (see Figure 7), and nearly all in phase 2 (see Figure 8).

Measure	Instance A		Instance B		Instance C	
Total Cost ($)	841.8	865.7	1303.7	1331.6	1240.4	2266.2
Power Cost ($)	437.4	437.4	874.8	874.8	1020.6	2041.2
Energy Cost ($)	404.4	428.3	428.9	456.8	219.8	225.0
Average Daily Discomfort Time (Minutes)	31.5	106.25	43.2	108.7	39.2	91.1
Average Energy Consumed Daily (kWh)	158.3	167.2	168.9	176.9	87.4	89.0
Highest Power Peak (kW)	30.0	30.0	60.0	60.0	70.0	140.0
# Equipment Cycles (Cycles)	260	315	394	418	925	1038
: DQN : Classical						

Figure 7: KPI for phase 1 deployment, comparing both the DRL controller and its classical reactive counterpart. Results are highlighted in green when DRL performances exceeds classical controls, and in red in the opposite case.

The only poorer performances in the year-round deployment are the ones related to discomfort and energy. It is however important to note that the difference between both controllers is smaller than the model's time step in the first case, making it negligible in the context of this simulation, while in the second case energy cost

savings were still observed despite slightly more consumed energy. Running more training epochs or having a complete year of data would probably solve or improve these aspects. Lastly, it is worth mentioning from a safety perspective that security measures were not triggered at any point for phase 1, and only twice for phase 2 during the coldest days of winter, illustrating the ability to operate reliably within strictly established constraints.

Measure		Instance A		Instance B		Instance C	
	Spring	1301.1	1770.3	2050.8	2952.2	3246.9	6327.2
	Summer	951.6	1424.0	1414.5	2755.3	2682.5	4179.2
Total Overall Cost ($)	Autumn	1713.7	1972.7	2690.8	3304.5	3356.2	6436.4
	Winter	2445.0	2510.3	3527.1	3897.2	4535.5	6749.0
		874.8	1312.1	1603.8	2478.6	3061.8	6123.6
Total Power Cost ($)		874.8	1312.2	1312.2	2624.4	2624.4	4082.4
		1093.5	1312.2	2041.2	2624.4	3061.8	6123.6
		1312.2	1312.2	2332.8	2624.4	3936.6	6123.6
		426.2	458.1	447.0	473.6	185.2	203.6
Total Energy Cost ($)		76.8	111.8	102.3	130.9	58.1	96.9
		620.2	660.5	649.6	680.1	294.4	312.8
		1132.8	1198.1	1194.3	1273.1	598.9	625.5
		44.7	69.8	44.3	59.0	53.2	39.9
Average Daily Discomfort Time (Minutes)		27.7	23.3	13.7	23.6	28.3	19.5
		33.5	52.7	54.6	71.67	53.0	39.7
		40.5	111.2	61.0	112.3	49.3	87.5
		54.2	56.5	57.8	56.9	25.1	24.7
Average Energy Consumed Daily (kWh)		9.2	11.9	11.8	13.9	6.5	10.0
		149.8	156.9	85.0	85.0	39.7	40.0
		150.1	159.8	159.9	168.0	80.6	83.8
		20.0	30.0	40.0	60.0	70.0	140.0
Highest Power Peak of the Season (kW)		20.0	30.0	30.0	60.0	60.0	140.0
		30.0	30.0	40.0	60.0	70.0	140.0
		30.0	30.0	60.0	60.0	100.0	140.0
		704	729	879	918	1316	1463
Total Equipment Cycles (Cycles)		298	256	370	385	563	782
		751	836	944	1030	1685	1903
		819	962	1042	1293	2591	2980
: DQN : Classical							

Figure 8: KPI for phase 2 deployment, comparing both the DRL controller and its classical reactive counterpart. Results are highlighted in green when DRL performances exceeds classical controls, and in red in the opposite case.

Figure 9 visually depicts a combination of important optimal HVAC behaviors autonomously learned by the DRL controller, which match suggested theoretical guidelines found in the HVAC literature [18], [19]: 1) temperature barely touches the set points and reacts pre-emptively just before doing so, 2) pre-heating in the morning is linear and with a precise phase shift, yet respects the more restrictive set point right on time during the occupancy schedule, 3) the central zone never triggers HVAC as it knows it will benefit from the heat transfer and inertia of all other rooms, and 4) temperature continuously oscillates between set points, with lagged power calls and with timing at the end of the day to reach the larger dead band without falling into the discomfort zone, and while successfully avoiding electricity consumption during the expensive high price hours of the evening.

Just like Figure 9 showed the hourly energy price awareness of the DQN controller, Figure 10 illustrates an important flattening behavior in the power calls of the January 2020 month, and Figure 11 an important peaks distribution shift toward lower values over the whole year. Such improvements are particularly important, as power-related costs usually account for the majority of the total expenses in buildings, and are prone to change depending on different factors like geographic location and electrical operator contracts. This inherent variability makes a flexible solution like DRL an ideal approach to fit a broad range of specific eventualities by simply adapting the reward definition of the control agent.

Despite being a competing objective to thermal comfort, equipment cyclability also shows enhancements over normal operations: for the largest instance, the number of cycles were reduced by 10% in Spring, 28% in Summer, 11% in Autumn, and 13% in Winter. The phenomenon can be directly observed in the control sequences comparison of instance B for a typical day in January 2020 (see Figure 12).

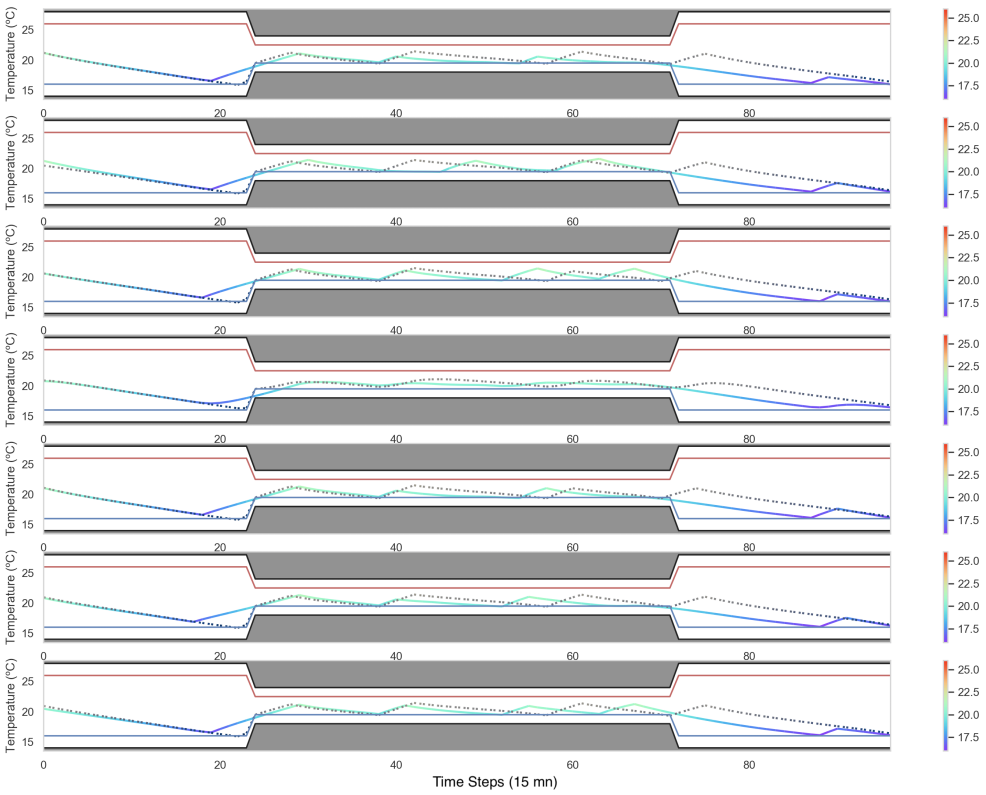

Figure 9: Temperature variation in each zone for the small commercial center instance during a typical day in January 2020. The fully colored line represents the RL agent's result, while the grey dashed line depicts what a classical controller would have performed under the same conditions.

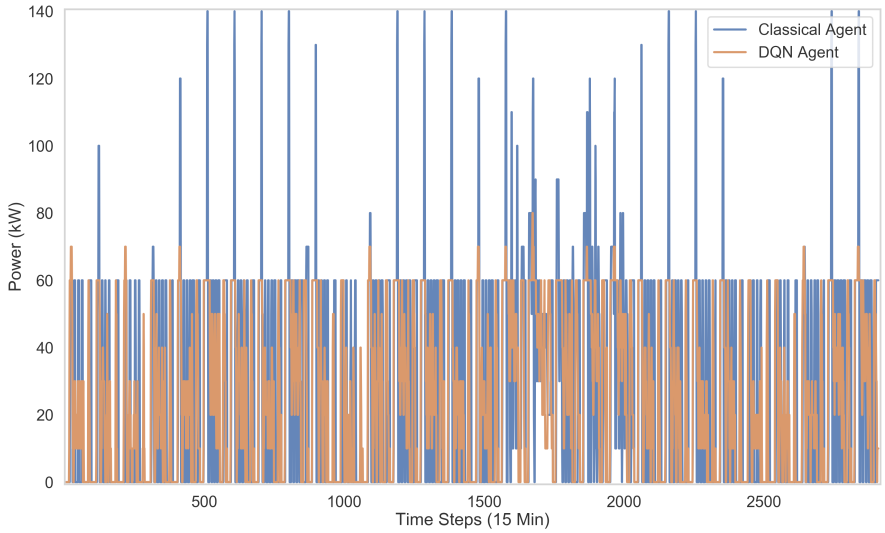

Figure 10: Instantaneous 15 minutes power calls for phase 1 of instance C.

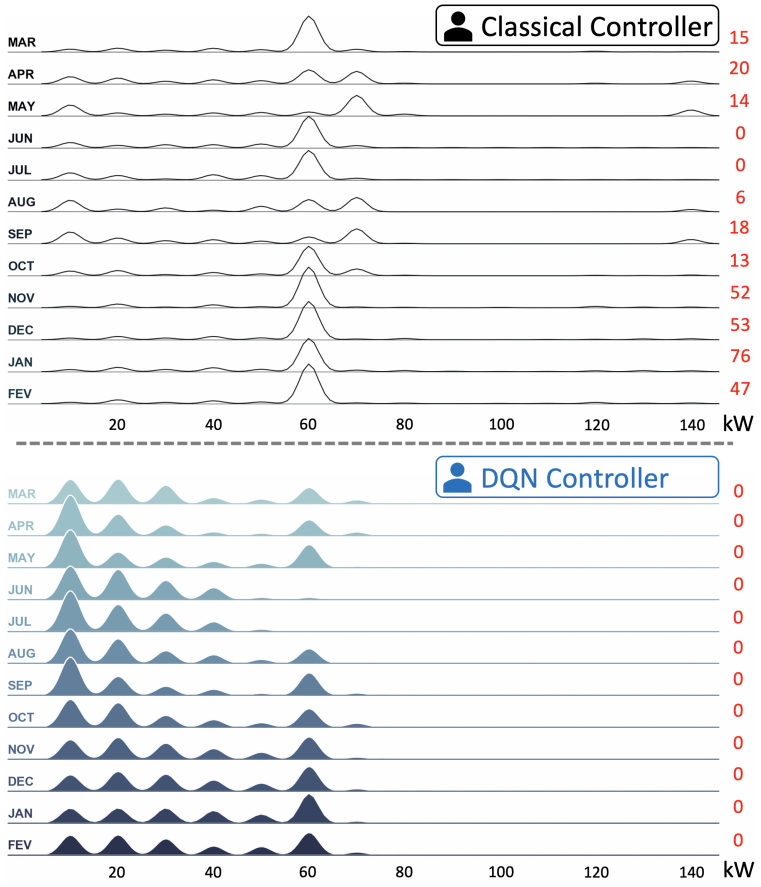

Figure 11: Monthly non-zero power call distributions for instance C, from March 15 2019 to March 15 2020. The red numbers on the right denote the number of power calls over 120 kW.

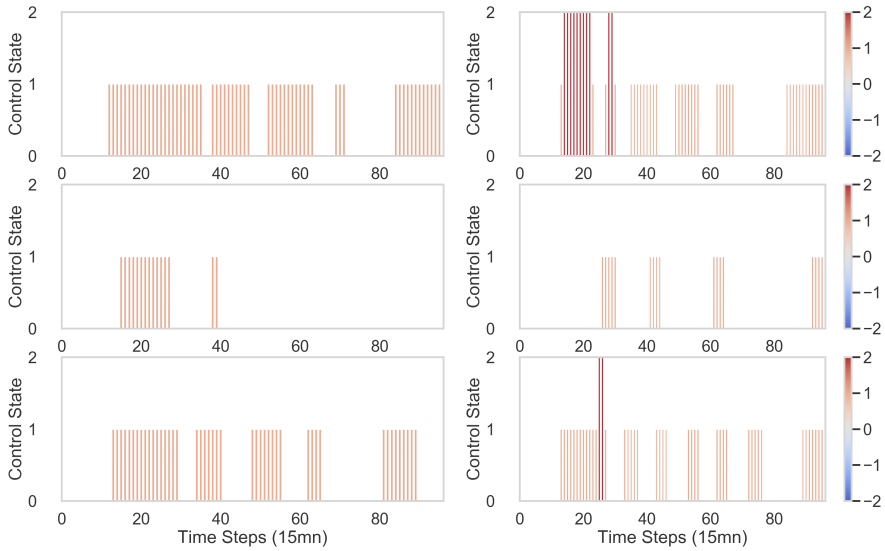

Figure 12: 24-hour control sequences on instance B for the DQN controller (left) and classical controller (right) during a typical day of January 2020.

Phase 2 results highlight a very important aspect, that is, the capacity of the DRL controller to cope well with all seasonalities and weather transitions. This aspect is usually challenging for this type of applications, making it a noteworthy advantage of the proposed methodology. The action set of the controller was never constrained or modified, and always had all the heating and cooling actions available. The agent autonomously learned how to blend efficiently heating and cooling in seasonality transitions, while properly focusing only on the important possibilities by itself in more extreme weathers. This suggests that while occasional retraining may be required during the first year of deployment, complete autonomy can be reached with very sparse yearly updates once a complete year of data is obtained or right away from the beginning given a proper reliable model of the building.

The training phase demonstrated an important continuous stability for all instances, with non-degrading and monotonically improving performance. Even after 80% of the epochs completed, pursuing training with a fixed exploration rate of $\epsilon = 0.05$ did not change significantly the actual approximated optimal Q-values following new neural network updates. Such behavior is important to monitor, as it proves convergence toward a stable solution, and can even be used to halt training after the optimal number of training iterations for deployment.

Our general synthesis is that multi-system DQN controller deployment value increases with the complexity and opportunity potential of the considered instance. That is, systems exhibiting complex dynamics and transient behaviors which typically induce challenges to building operators and engineers, like fluctuating energy prices or contracts, a high number of simultaneous control components, dynamic building schedules, or user-defined set points, to name but a few. The *learning* and adaptive capability of DRL methodology then becomes highly profitable, and does not suffer from obsolescence in the long term. Applying this solution to smaller-scale buildings like houses or simpler systems still result in improvements, but can prove more impactful in higher volumes and with group coordination.

5 Conclusion

Case studies results demonstrate the successful application of our proposed methodology on the smart building multi-objective optimization problem. Significant improvements were observed in expenses, thermal comfort, energy consumption, power peaks and equipment cycling for different buildings accounting multiple HVAC systems. Our proposed DQN controller outperformed its classical reactive counterpart with respect to all individual objectives in simulation, and showed successful year-round deployment without any retraining.

In addition to its autonomous learning capabilities directly from raw observations of any entity, DRL reward definition can easily be tuned and adapted through a simple graphical user interface (GUI) post-implementation to reflect different optimization objectives. Furthermore, from a calculation perspective, a DQN controller presents the advantage of being light in deployment, as only matrix multiplications are required during the forward pass to access the policy. This can further be leveraged in the GUI to offer visualization and simulation tools as a transparency measure to building managers and operators.

In the light of our results, it is our belief that DRL and its extensions can cope with much more complex smart building instances, both in terms of scale and dynamics. Moreover, generalization success on unseen data leads us to the premise that our approach could be directly applied on a real building in future work, by first performing the learning phase on a digital twin with an appropriate statistical or physical model. Given multiple adoptions, this building-scale solution could then be extended to a larger multi-agent hierarchy at the grid level to help electrical operators and support network resiliency.

Appendix A Deep learning

Artificial neural networks (ANN) are computing systems structured as alternating layers of aggregated parameters and nonlinear activation functions. Their strength comes from their ability to learn complex hierarchical non-linear representations of different abstraction levels from data samples and to scale particularly well with massive datasets. Traditional fully-connected layers can be seen as vector-to-vector mappings, where two

operations are successively applied on a given input x: first, a linear transformation of the form

$$Wx + b \ , \tag{7}$$

where W represents the *weights* matrix, and b the *bias* vector; then, the linearly modified outputs are fed as an argument into a non-linear *activation function* σ. Typical choices of functions are the *rectified linear unit* (ReLU) $\equiv \max(0, x)$, the *hyperpolic tangent* $\equiv \tanh(x)$ or other *sigmoid* functions.

The *forward pass* or *forward propagation* of a neural network is defined as the complete information flow from the input layer to the output [20]. For the multi-layer perceptron (MLP), one of the most standard ANN made of consecutive fully connected layers, this yields:

$$h^{(i)} = \sigma^{(i)}\left(W^{(i)\mathrm{T}} h^{(i-1)} + b^{(i)}\right) \ \forall \ i \in [1, 2, ..., L+1] \ , \tag{8}$$

where $h^{(i)}$ is the output of the i-th layer, $h^{(0)} = x$, and L is the number of hidden layers. The output $h^{(L+1)} = \hat{y}$ typically uses a different activation function, depending on the nature of the considered supervised learning task.

Deriving a scalar cost $\mathcal{L}(\theta)$ from $\hat{y}$ with $\theta = [W^{(1)} \dots W^{(L+1)}; b^{(1)} \dots b^{(L+1)}]$, and applying the maximum likelihood principle on n available samples, the standard ANN training problem has the form

$$\min_{\theta} \sum_{k=1}^{n} \left(\mathcal{L}\big(\hat{y}_{(k)}(x_{(k)}; \theta), y_{(k)}\big)\right) \ , \tag{9}$$

where y denotes the real value of some predicted quantity. From this point, the gradient of the loss with respect to each parameter θ in the network can be computed, in a phase referred to as *back-propagation*. Finally, given the unconstrained nonconvex differentiable optimization nature of the problem, such instances are usually addressed with standard (incremental) gradient-type optimization methods [21].

A.1 Recurrent neural networks

Recurrent neural networks (RNN)s [22] are designed to leverage sequential data of the form $x_1, ..., x_\tau$, by building an internal state h updated at each sequence input x_t by the recursive relationship

$$h_t = f(h_{t-1}, x_t; \theta) \ . \tag{10}$$

This hidden state is used as a partial summary of the task-relevant aspects of past sequence inputs up to time t, and can be used in different fashions, depending on how the computational graph of the outputs is developed by the user. Bidirectional RNNs [23] extend classical RNNs by combining two layers of opposite direction to the same output.

Gated RNNs such as the Long Short-Term Memory [24] (LSTM) or the Gated Recurrent Unit [25] (GRU) build on top of classical RNNs by adding internal memory cells and operations to control information flow through different logical gates implemented with sigmoids. More concretely, LSTMs possess a *forget* gate to control information kept from previous cell states; an *input gate* to filter new information input from x_t and h_{t-1}; and an *output* gate to control the nature of the next hidden state h_{t+1}. Besides being popular for solving calculation issues known as *exploding* and *vanishing* gradient, internal gated recurrent unit components are also very useful because their behavior and properties can be learned along with the rest of the parameters during the training process.

Appendix B Thermodynamics and modeling

Starting from the diffusion term in the general physics *transport equation* of some quantity u, temperature in our context :

$$\frac{\partial u}{\partial t} = \nabla \cdot (D \nabla u) \ , \tag{11}$$

where D is the diffusion coefficient which we assume constant, we define a spatial meshing of the form $x_j = j \times \Delta x,\ j = 0, 1, ..., J-1$ and temporal meshing $t^n = n \times \Delta t,\ n = 0, 1, ..., N$. The time iteration index n goes up to N because $n = 0$ is used to denote the initial condition.

Given the initial and boundary conditions, converting the left-hand side of (11) into a finite difference for the temporal derivate, and the right side into a second-order centered temporal difference for the second derivative of x leads to the well known *explicit* or *Forward-in-Time-Centered-in-Space* (FTCS) algorithm [26]. Evaluating the right-hand term of the FTCS method at time $n+1$ instead of n results in the so-called *Implicit Euler* method:

$$u_j^{n+1} = u_j^n + \Big(\frac{D\Delta t}{(\Delta x)^2}\Big)(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}) \ , \tag{12}$$

that can be expressed under the matrix form

$$K_{ij}\ u_j^{n+1} = u_i^n \ . \tag{13}$$

This method is particularly well suited for applications with bigger disparities between characteristic times, which is the case in thermal modeling of a building.

Focusing on the unidimensional implicit finite-elements heat transfer implementation, we consider that each given node i is exchanging heat with all neighbouring nodes j and k (the latter indexes nodes with known defined temperature, a boundary condition) through conduction, convection and radiation, expressed as an equivalent conductance U; has capacitance or thermal mass $C = mc_p$ (J/K), where c_p (J/kg K) is the specific heat of the medium and m (kg) its mass; and is receiving heat from a source Q. The finite difference equation can then be written:

$$\sum_j U_{ij}^{t+1}(T_j^{t+1} - T_i^{t+1}) + \sum_k U_{ik}^{t+1}(T_k^{t+1} - T_i^{t+1}) \\ -\frac{C_i}{\Delta T}(T_i^{t+1} - T_i^t) + \dot{Q}_i^{t+1} = 0 \ , \tag{14}$$

where U_{ij} (W/K) is the conductance between nodes i and j, $\dot{Q}$ (W) the heat flow into the node and Δt is expressed in seconds. The conductance for the conduction, convection and radiation heat transfer mechanisms is respectively given by Ak/L, $\bar{h}_c A$ and $\bar{h}_r A$, where A (m^2) is the area through which heat is transferred, k (W/m K) the thermal conductivity, L (m) the length between points i and j, and $\bar{h}_c$ (W/K m^2) and $\bar{h}_r$ (W/K m^2) the convection and radiation heat transfer coefficients.

References

[1] International Energy Agency. Energy efficiency: Buildings. https://www.iea.org/topics/energyefficiency/buildings/, 2019. Accessed: 2020-03-08.

[2] Environmental and Energy Study Institute. Buildings and built infrastructure. https://www.eesi.org/topics/built-infrastructure/description. Accessed: 2020-04-1.

[3] UN Environment. Global status report: towards a zero-emission, efficient, and resilient buildings and construction sector. https://www.worldgbc.org/sites/default/files/UNEP%20188_GABC_en%20%28web%29.pdf, 2017. Accessed: 2020-03-15.

[4] D. Lee, S. Lee, P. Karava, and J. Hu. Approximate dynamic programming for building control problems with occupant interactions. In 2018 Annual American Control Conference (ACC), pages 3945–3950, 2018.

[5] Berenger Favre and Bruno Peuportier. Optimization of building control strategies using dynamic programming. pages 2593–2600, 2013.

[6] Model predictive control for smart buildings to provide the demand side flexibility in the multi-carrier energy context: Current status, pros and cons, feasibility and barriers. Energy Procedia, pages 3026–3031, 2019. Innovative Solutions for Energy Transitions.

[7] I. Szilagyi and P. Wira. An intelligent system for smart buildings using machine learning and semantic technologies: A hybrid data-knowledge approach. In 2018 IEEE Industrial Cyber-Physical Systems (ICPS), pages 20–25, 2018.

[8] Djamel Djenouri, Roufaida Laidi, Youcef Djenouri, and Ilangko Balasingham. Machine learning for smart building applications: Review and taxonomy. ACM Computing Surveys, 52, 2019.

[9] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning – An Introduction (2nd Edition). MIT Press, 2018.

[10] Dimitri P. Bertsekas. Dynamic Programming and Optimal Control – Approximate Dynamic Programming – Volume II (4th Edition). Athena Scientific, 2005.

[11] Martin L. Puterman. Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Interscience, 2005.

[12] Christopher Watkins and Peter Dayan. Q-learning. Machine Learning, 8:279–292, 1992.

[13] Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Human-level control through deep reinforcement learning. Nature, 518:529–540, 2015.

[14] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. AAAI, page 2094–2100, 2016.

[15] The Dark Sky Company (2020). Dark sky api. https://darksky.net/dev, 2020. Accessed: 2020-03-18.

[16] Laura Graesser and Wah Loon Keng. Foundations of Deep Reinforcement Learning – Theory and Practice in Python. Addison Wesley, 2019.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. ArXiV, 2014.

[18] Refrigerating American Society of Heating and Air-Conditioning Engineers (ASHRAE). ASHRAE Handbook – Fundamentals. Atlanta, GA, 2017.

[19] Donald R. Wulfinghoff. Energy Efficiency Manual. Energy Institute Press, 1999.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[21] Dimitri P. Bertsekas. Reinforcement Learning and Optimal Control. Athena Scientific, 2019.

[22] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by backpropagating errors. Nature, 323:533–536, 1986.

[23] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. IEEE, 45:2673–2681, 1997.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9:1735–80, 12 1997.

[25] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs/1412.3555, 2014.

[26] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing. Cambridge University Press, USA, 1992.