Experimental evaluation of model-free reinforcement learning algorithms for continuous HVAC control

Marco Biemann[a,b], Fabian Scheller[a], Xiufeng Liu[a,*], Lizhen Huang[b,*]

[a]*Department of Technology, Management and Economics, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark*
[b]*Department of Manufacturing and Civil Engineering, Norwegian University of Science and Technology, 2815 Gjøvik, Norway*

Abstract

Controlling heating, ventilation and air-conditioning (HVAC) systems is crucial to improving demand-side energy efficiency. At the same time, the thermodynamics of buildings and uncertainties regarding human activities make effective management challenging. While the concept of model-free reinforcement learning demonstrates various advantages over existing strategies, the literature relies heavily on value-based methods that can hardly handle complex HVAC systems. This paper conducts experiments to evaluate four actor-critic algorithms in a simulated data centre. The performance evaluation is based on their ability to maintain thermal stability while increasing energy efficiency and on their adaptability to weather dynamics. Because of the enormous significance of practical use, special attention is paid to data efficiency. Compared to the model-based controller implemented into EnergyPlus, all applied algorithms can reduce energy consumption by at least 10% by simultaneously keeping the hourly average temperature in the desired range. Robustness tests in terms of different reward functions and weather conditions verify these results. With increasing training, we also see a smaller trade-off between thermal stability and energy reduction. Thus, the Soft Actor Critic algorithm achieves a stable performance with ten times less data than on-policy methods. In this regard, we recommend using this algorithm in future experiments, due to both its interesting theoretical properties and its practical results.

Keywords: reinforcement learning, continuous HVAC control, actor-critic algorithms, robustness, energy efficiency, Soft Actor Critic

1. Introduction

Energy consumption in buildings accounts for about 40% of global energy consumption. Heating, cooling and ventilation contribute most in building energy consumption and therefore play a pivotal role in mitigating global warming [1]. For example, heating accounts for more than 50% of energy consumption in cold regions during the winter [2]. Effective HVAC control can significantly improve building energy efficiency and indoor thermal comfort, thus supporting the international sustainable development goals. On the other hand, the growing complexities associated with

*Corresponding author
Email addresses: marcob@dtu.dk (Marco Biemann), fjosc@dtu.dk (Fabian Scheller), xiuli@dtu.dk (Xiufeng Liu), lizhen.huang@ntnu.no (Lizhen Huang)

energy transformations requires innovative smart control systems. In this context, energy-related demand-response control operations should be able to cope with stochastic environmental influences, volatile energy pricing, potential power shortages, the intermittency of renewable energy sources, and changes in consumption behaviour.

Nowadays, the HVAC management systems of most current residential buildings use classical algorithms, such as rule-based controllers or proportional, integral and derivative controllers (PID). However, due to the high inertia of thermodynamics, these controllers often overshoot the target temperature, leading to additional energy consumption and reduced thermal comfort. These controllers do not include domain-specific knowledge and cannot use historical data or model predictions. Model Predictive Control (MPC) formulates these challenges as a constrained optimisation problem [3]. For example, based on temperature, demand, and price forecasts, the controller determines when to preheat a building, thus reducing peak energy demand and saving energy costs. However, it only works properly with an accurate and detailed model, requiring expert knowledge for development and calibration. This implies that this paradigm is hardly scalable, as each building is unique and has its own requirements. Therefore, developing a common building energy management model for different types of buildings presents a serious challenge to the widespread deployment of MPC. Moreover, growing complexities such as the stochastic nature of renewables are increasingly difficult to address mathematically.

Due to the increasing popularity of Machine Learning (ML) methods, we aim to devise a good management strategy using data instead of complex mathematical models. Reinforcement Learning (RL) controllers have started being increasingly assessed in HVAC management systems [4, 5]. The most widely used approach is to create a simulation environment to generate the necessary data required to train the algorithm [6]; the idea being to copy the controller into a physical building and continue training there. In contrast to MPC, the model is only required for generating the training data, not for computing the control strategy. Further advantages of RL are that the operation of the algorithms does not require weather or price forecasts, as they can be learned using training data. Once the training has been completed, the operation is associated with much lower computational costs than MPC. However, RL comes with its own limitations, mostly data efficiency, meaning that it requires large amounts of data for training. This makes it inconvenient to train directly in a physical building. Therefore, the need for a model cannot yet be eliminated, as a simulation environment is still required for training. In this paper, we focus on the evaluation of model-free algorithms, but also mention that MPC and RL can be combined into model-based RL to learn to predict future states for control [7, 8].

Demand-response or HVAC management requires adaptation to external factors that cannot be influenced by the agent, such as the weather, prices and human indoor activities. These parameters vary continuously, so in order to keep the temperature within a fixed range, we are interested in managing the setpoint temperature continuously as well, as this allows for finer operation than if we would set it to predefined values. The most commonly used algorithm [5], Deep Q-learning (DQN), is unable to handle such problems, as it only works in discrete action spaces. This motivates us to investigate other RL algorithms that have rarely been used in HVAC applications. As noted by Wölfle et al. [9], most existing studies in building energy management do not compare algorithms with each other, contrary to what

the ML literature suggests [10]. This makes it challenging to emphasise the possible strengths and downsides of the algorithms for this application. Note also that the environments in the building management sector are significantly different from the mostly deterministic environments the algorithms are commonly tested on [11], as the agent needs to react to multiple stochastic factors.

We evaluate the algorithms on a simulated medium-sized data centre environment, which represents a typical problem for HVAC management. The objective is to minimise energy consumption while keeping the indoor temperature within a pre-defined range for the servers in a data centre. This is important, as a high indoor temperature has a significant impact on computing performance and server lifespan, while overcooling can increase energy consumption [12]. This paper will evaluate four state-of-the-art RL algorithms for continuous control: Soft Actor Critic (SAC), Twin Delayed Deep Deterministic Policy Gradients (TD3), Trust Region Policy Optimisation (TRPO) and Proximal Policy Optimisation (PPO) on a common open-source environment. We evaluate the performance in terms of energy savings, thermal stability, robustness and data efficiency. In summary, this paper makes the following contributions:

- We conduct experiments to evaluate and compare the performance of RL algorithms in an open-source benchmark project in terms of energy consumption and indoor climate management. We show that all algorithms are reliably able to maintain temperature, while reducing energy consumption. This emphasises the ability of these RL algorithms to learn the task consistently without having to run time-intensive hyperparameter searches, which is often challenging for real-world implementations.

- We conduct experiments to study the adaptability of the algorithms to the weather dynamics. We demonstrate the robustness of the agents, which reliably manage to maintain the temperature under new weather conditions.

- We show that there is a trade-off between energy consumption and thermal stability, in the sense that some algorithms have a lower energy consumption, whereas others are better at keeping the temperature within the desired range. With longer training, the results of the algorithms become more similar.

- We analyse the data efficiency of the algorithms and demonstrate that SAC is able to learn the task with significantly less data than state-of-the-art on-policy algorithms, while enjoying a stable learning process. Surprisingly, TD3 does not reach a performance comparable to SAC, suggesting that deterministic policies may not be adapted for stochastic environments. The theoretical properties that may explain these results are presented with the current case study in mind.

The rest of this paper is structured as follows. Section 2 surveys the related work. Section 3 introduces the theoretical background of RL. Section 4 explains the RL algorithms. Section 5 presents a case study of HVAC control in data centres. Section 6 conducts the experiments and analyses the results. Section 7 concludes the paper.

2. Related work

The idea of using RL in HVAC was first proposed by Mozer [13], who installed a smart control system into a former school building. The control system can adapt both HVAC and lighting to the occupants' wishes. In recent years, RL has received increasing attention in the building energy domain, where HVAC controllers have been used to automating energy control, taking external constraints into account such as indoor comfort, occupancy and energy price. RL-based applications have been developed for the management of HVAC, water-heaters, energy storage, battery-charging, smart appliances and more. For more information about the applications of RL in demand-response applications, we refer to multiple reviews in the area. For example, Vázquez-Canteli and Nagy [14] reviewed the use of RL in various demand response applications, Han et al. [15] focused on occupant comfort, Wang and Hong [5] reported on the quantitative use of algorithms in the literature, and Perera and Kamalaruban [4] concluded that state-of-the-art algorithms such as TD3 or SAC were too rarely applied, hindering performance improvements.

With regard to the building energy sector, Henze and Schönmann [16] and Liu and Henze [17] used RL to manage a thermal energy storage system. The results showed that an RL-based controller can reduce costs, but requires significant amounts of data for training in order to achieve a performance similar to traditional predictive methods. In [18, 6], Liu and Henze developed a method of using a simulated environment to pre-train the agent before deploying it into a real-world building, as is commonly done today. The simulation model does not have to be perfectly accurate. Instead, it only requires the same states and actions as the learning controller during deployment. The same network and weights are used, but they will have to be continuously updated to improve performance further and minimise the impact of the shift from simulation to reality.

In recent years, models that are based on simplified thermodynamic equations have been increasingly replaced by realistic, complex building simulators, such as EnergyPlus. Moriyama et al. [19] explain how to combine EnergyPlus with RL agents and provided an open-source example of such an environment, which we use in this paper. Zhang et al. [20] discuss how to use EnergyPlus for real-world deployment and its challenges.

In the ML community, it is common to evaluate the performance of algorithms using large standardised open-source data sets or learning environments. This facilitates algorithm benchmarking and comparison, permitting a discussion of the advantages and limitations of the algorithms. As noted by Vázquez-Canteli and Nagy [14] and Wölfle et al. [9], this is significantly different in the building simulation community. The algorithms are trained and evaluated on similar problems, but with different physical properties and dynamics. Only a few papers evaluate and compare multiple algorithms in the same case study, which would help users in selecting algorithms. In addition, the evaluation is often done on a single building. The generality of the approach can be questioned.

Recent years have seen some major breakthroughs in RL, such as attaining super-human performances in video games [21] and Go [22]. This led to new algorithms, significantly improving on classical methods, as well as the increased visibility and popularity of the field. The first approaches [16, 17] used tabular Q-learning, but in order to apply this, the state-action space had first to be made discrete. The controller was able to learn the task, but the

approach proved to be data-inefficient and sensitive to the choice of the discretisation. Ruelens et al. [23] used an autoencoder to reduce the complexity of the states, allowing applications with infinite state spaces. They used the fitted Q-iteration algorithm based on experience replay to deal with sample efficiency. In subsequent work [24, 25], the trained policy was modified to incorporate domain knowledge to obtain a better performance. Recent studies favour end-to-end approaches, where the Q-function is directly approximated by a neural network. The DQN algorithm is a popular choice for HVAC control due to its simplicity and data-efficiency. For example, Wei et al. [26] used DQN to deal directly with the large state space in an application to a multi-zone building. However, in order to be applied, the action space needed to be discretised. A sufficiently fine discretisation increases the number of actions exponentially, making the algorithm increasingly difficult to train for problems requiring the control of additional parameters [27]. To address this problem, they trained a separate network for each zone, which is computationally expensive.

To deal with continuous action spaces, Wang et al. [28] used on-policy actor-critic methods. Li et al. [29] used the Deep Deterministic Policy Gradient (DDPG) algorithm to maintain similar data-efficiency to DQN. The study by Gao et al. [30] showed that continuous control algorithms, such as DDPG, can outperform DQN or tabular Q-learning. Similar conclusions were drawn by Du et al. [31]. On-policy algorithms, such as PPO, are often used as a baseline [7, 8], because they are stable and fast to train. However, policy gradient algorithms are not yet widespread in the building energy community. A possible reason for this is that early algorithms, such as DDPG, are notoriously difficult to train [32], as their performance is sensitive to hyperparameters. On the other hand, PPO suffers from data efficiency, requiring large numbers of samples to train the algorithm, and making it unsuitable for real-world applications. Algorithms that are both data-efficient and stable have since been introduced in the RL community. SAC has recently been applied to multi-agent problems [33, 34], but the advantages compared to other algorithms were not highlighted.

Finally, we uncovered only few papers in the literature that study robustness and generalisation, important points to consider for real-world applications. Xu et al. [35] and Lissa et al. [36] discussed how well the agent generalises to different environment dynamics. For example, in Xu et al. [35], the agent was trained in a given environment and evaluated on a slightly different environment with different building layouts, weather data and construction materials.

In summary, RL has increasingly been applied to HVAC case studies in recent years. Nevertheless, we identified two gaps that slowed down the research in the field. First, multiple papers use tabular methods or DQN for problems that would naturally be defined as continuous control tasks. Second, most papers discuss the implementation of one algorithm in a novel case study; there are few papers comparing different algorithms. This motivated a comparative study of continuous control algorithms that are better suited to handling such problems. We also used this opportunity to discuss related technical properties, such as robustness and data efficiency that were not given the importance they deserve, given their practical importance for potential real-world deployments.

3. Theoretical background

This section will first present the theoretical framework of RL, which is based on Markov Decision Processes, and then present theory on policy gradients. These results will motivate the updates of the presented RL algorithms.

3.1. Reinforcement learning

Reinforcement learning (RL) is a computational approach to decision-making under conditions of uncertainty [37, 38]. The learning problem can be defined as a *Markov Decision Process* (MDP), which represents the interaction between an agent and the environment. Formally, an MDP is a quintuple $(\mathcal{S}, \mathcal{A}, p, \rho, r)$, where

- $\mathcal{S} \subset \mathbb{R}^n$ is the *state space*.
- $\mathcal{A} \subset \mathbb{R}^m$ is the *action space*.
- $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the *state-transition probability*, corresponding to the probability of going from state s to state s' by means of action a.
- $\rho : \mathcal{S} \to [0, 1]$, is the *initial state probability*, corresponding to the probability of starting at state s.
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *reward (or cost) function*.

The state and action spaces are domain-specific; this will be described later. The transition probability p represents the physics of the system. Unlike MPC, it is unknown to the agent, to which it tries to adapt by trial and error. The reward function is similar to a cost function, which is usually the objective function in a control problem.

The *policy* is a probability distribution $\pi : \mathcal{S} \times \mathcal{A} \longrightarrow [0, 1]$, giving the probability of taking an action $a \in \mathcal{A}$ in an state $s \in \mathcal{S}$. The policy learned can be either deterministic or stochastic, and depends on the specific algorithm.

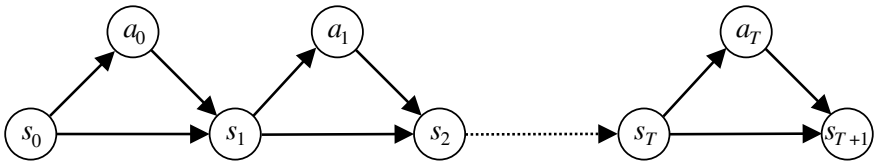

Figure 1: Illustration of a graphical model of an MDP.

To define the optimisation objective, we introduce the notion of a *trajectory* τ, that is a sequence of state-action pairs $(s_t, a_t)_{t\geq 0}$, where the states are returned by the environment and the actions follow the policy π. As illustrated in Figure 1, the probability of a given trajectory τ can be factorised as [39]:

$$p_\pi(\tau) = \rho(s_0) \prod_{t\geq 0} \pi(a_t \mid s_t) p(s_{t+1} \mid a_t, s_t). \tag{1}$$

The *(maximum entropy) RL objective* (see e.g. [40]) is defined as follows, where for $\gamma \in (0, 1)$ and $\alpha \geq 0$[1], we have:

$$J^{\text{soft}}(\pi) = \mathbb{E}_{\tau \sim p_\pi}\left[\sum_{t \geq 0} \gamma^t \left(r(s_t, a_t) - \alpha \log \pi(a_t \mid s_t)\right)\right]. \tag{2}$$

The objective of RL is to find a policy π^* that maximises $J^{\text{soft}}(\pi)$. This objective generalises slightly the traditional RL objective presented in [38], which we can recover by setting $\alpha = 0$ and denote by $J(\pi)$.

The majority of RL algorithms aims to learn a value function $V_\pi(s)$ or $Q_\pi(s, a)$. The first tells what the expected reward is at a given state s, while the second tells us what the expected reward is, when taking action a in state s. The *state-value function* is defined as follows (it can be reverted to the traditional definition of $V_\pi(s)$ by setting $\alpha = 0$):

$$V_\pi^{\text{soft}}(s) = \mathbb{E}_{\tau \sim p_\pi}\left[\sum_{k \geq 0} \gamma^k \left(r\left(s_{t+k}, a_{t+k}\right) - \alpha \log \pi\left(a_{t+k} \mid s_{t+k}\right)\right) \;\middle|\; s_t = s\right]. \tag{3}$$

We take the expectation over all trajectories starting at state s. This definition is strongly related to the optimisation objective, yielding the relationship $J^{\text{soft}}(\pi) = \mathbb{E}_{s_0 \sim \rho}[V_\pi^{\text{soft}}(s_0)]$.

The *action-value function* or simply Q-function (where Q stands for quality) is strongly related to $V_\pi(s)$. It can be defined by the following equation (see [38] for more details):

$$Q_\pi^{\text{soft}}(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)}\left[V_\pi^{\text{soft}}\left(s'\right)\right]. \tag{4}$$

In addition, we have the *Bellman equation*:

$$V_\pi^{\text{soft}}(s) = \mathbb{E}_{a \sim \pi(\cdot|s)}\left[Q_\pi^{\text{soft}}(s, a) - \alpha \log \pi(a \mid s)\right]. \tag{5}$$

If we can find the optimal Q-function $Q^*(s, a) = \max_\pi Q_\pi^{\text{soft}}(s, a)$, we can express an optimal policy π^* in terms of Q^*. Value-based methods work by iterating the recurrent relationship of the two equations (4) and (5). In the traditional RL setting ($\alpha = 0$), an optimal policy is given by the greedy policy:

$$\pi^*(a \mid s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q^*(s, a), \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

In the maximum entropy RL setting, an optimal policy can be given by the *softmax* policy (see [40] for the proof), given by:

$$\pi_{\text{soft}}^*\left(a \mid s\right) = \frac{\exp\left(\alpha^{-1} Q^*(s, a)\right)}{\int_{\mathcal{A}} \exp\left(\alpha^{-1} Q^*\left(s, a'\right)\right) da'}. \tag{7}$$

Note that if $\alpha \to 0$, we have $\pi_{\text{soft}}^*(a \mid s) \to \pi^*(a \mid s)$, recovering the greedy policy (6) of the traditional RL objective. Hence, if α is small, the policy tends to be deterministic, whereas when α becomes large, it tends to take

[1]The *discount factor* γ can be regarded as a moving effective time horizon. It is introduced to make the series converge mathematically in infinite-horizon tasks, typically close to 1. In the literature, α is called *temperature* and addresses the trade-off between reward and entropy, but we do not insist on this terminology here to avoid confusion with our case study. The SAC algorithm aims to maximise this more general objective.

completely random actions. Note that this policy (sometimes called Boltzmann exploration) has been used in the HVAC setting. The studies in [17, 23] have validated the fact that exploration policy performs better than the greedy (and ε-greedy) policy in the tabular setting.

A learning algorithm maximising the maximum entropy RL objective introduces some randomness into the policy, in contrast to the traditional value-based methods that learn a deterministic policy. This is useful, as it handles exploration naturally during training. Besides, a policy optimising the maximum entropy objective learns to solve a task in multiple ways, while a traditional agent aims to solve the task in an optimal, but unique way. This implies that stochastic policies can provide a better initialisation for transfer learning than deterministic functions. Furthermore, it is probable that in our application the dynamics of the system change over time, e.g., climate, energy prices, occupancy and the building's physical conditions. The soft agent (e.g. from SAC) should have fewer difficulties in adapting to the stochasticity of the environment, the domain shift between the simulated and physical environments, or to non-stationary dynamics [41].

3.2. Policy gradients

Policy gradients provide the theoretical basis for the actor-critic methods. Since commonly used value-based methods (based solely on the Bellman equation) are unable to handle continuous control (we explain why in Section 4.1), a discretisation of the action space becomes necessary, as discussed in Section 2. Policy gradient methods do not require it, so it is more natural to apply these methods to HVAC applications.

To learn a policy, we parameterise the policy $\pi_\theta(a \mid s)$, using weights $\theta \in \mathbb{R}^d$. The policy can be described using various function approximators, but it is now common to use a neural network. Let us denote by $J(\theta) = J(\pi_\theta)$ the expected reward. We define the *advantage function* $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$, quantifying how much better it is to take action a in state s over the average action. A fundamental result is the policy gradient theorem [42], i.e.,

$$\nabla_\theta J(\theta) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho^{\pi_\theta},a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s) A_{\pi_\theta}(s,a)\right], \tag{8}$$

where

$$\rho^{\pi_\theta}(s) = (1-\gamma)\mathbb{E}_{s'\sim\rho}\left[\sum_{t\geq 0}\gamma^t p(s' \to s, t, \pi_\theta)\right] \tag{9}$$

and $p(s' \to s, t, \pi_\theta)$ is the probability of the transition from $s_0 = s'$ to $s_t = s$, when following the policy π_θ.

For continuous action spaces, there is a similar result [43] for deterministic policies $\mu_\theta : \mathcal{S} \to \mathcal{A}$,[2] i.e.,:

$$\nabla_\theta J(\theta) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho^{\mu_\theta}}\left[\nabla_a Q_{\mu_\theta}(s,a)|_{a=\mu_\theta(s)}\nabla_\theta \mu_\theta(s)\right]. \tag{10}$$

The actor is typically updated using the gradient estimates of the theorems above, via a gradient descent algorithm. For example, Advantage Actor-Critic (A2C) [44] uses the policy gradient theorem (8), and Deep Deterministic Policy Gradient (DDPG) [27] uses the deterministic gradient theorem (10). The drawback of these estimates is their high

[2] We refer to the deterministic policy as $\mu_\theta(s)$ to distinguish it from $\pi_\theta(a \mid s)$, which is stochastic.

variance, making the learning process unstable. For example, the A2C algorithm needs crucially multiple actors to collect data in order to reduce the risk of catastrophic updates. Schulman et al. [45] developed theoretical policy improvement guarantees to provide more stable learning. They are used in the Trust Region Policy Optimisation (TRPO) [45] algorithm to provide a more stable on-policy algorithm than A2C. The subsequent Proximal Policy Optimisation (PPO) [46] uses heuristics to increase the reliability of the learning process.

4. Algorithms

In general, RL algorithms can be divided into two families: value-based methods and actor-critic methods. While value-based methods are inspired by the value iteration algorithm in dynamic programming, actor-critic methods are inspired by the policy iteration algorithm [38]. The latter are more convenient in handling continuous control, so this paper focuses on them.

RL algorithms can be off-policy or on-policy. Off-policy methods can use data sampled from any policy, as well as expert demonstrations (by a human or rule-based controller). In contrast, on-policy methods should only use data sampled from the current policy to update the network. An overview of the most popular algorithms and its main properties is given in Table 1.

Algorithm	Data usage	Action space	Critic update, eq no.	Actor	Actor update, eq. no
DQN [21]	Off-policy	Discrete	(12)	-	-
A2C [44]	On-policy	Discrete/Continuous	(D.2)	Gaussian	(8)
DDPG [27]	Off-policy	Continuous	(13)	Deterministic	(16)
TD3 [47]	Off-policy	Continuous	(14)	Deterministic	(16)
SAC [48]	Off-policy	Continuous	(15)	Gaussian	(17)
TRPO [45]	On-policy	Discrete/Continuous	(D.2)	Gaussian	(D.4)
PPO [46]	On-policy	Discrete/Continuous	(D.2)	Gaussian	(D.8)

Table 1: Description of the main properties of popular RL algorithms.

In applications where data is expensive or slow to generate (e.g., robotics, HVAC), off-policy methods are generally preferred because of their data-efficiency (e.g., we can store the data and reuse them later in the training process). The drawback is the absence of policy improvement heuristics, making the algorithms more difficult to train. On-policy methods are generally faster and more stable, which is useful for applications where data can be generated quickly (e.g., if we have access to a simulator). However, on-policy methods are not sample-efficient, as they must discard the collected data every time the policy is updated. In real-world HVAC applications, data collection is slow, therefore, off-policy algorithms can offer a great advantage. For this reason, this paper will mainly focus on off-policy algorithms.

The learning scheme for critic-actor methods is similar to the policy iteration algorithm in dynamic programming, as shown in Figure 2. An actor-critic algorithm is composed of two networks: a critic for the approximation of $Q^*(s,a)$

or $V^*(s)$, and an actor to learn the optimal policy $\pi^*(a \mid s)$. The critic's predictions are used to update the actor. Once the actor is updated, we re-run the policy in the environment to obtain better samples for training.

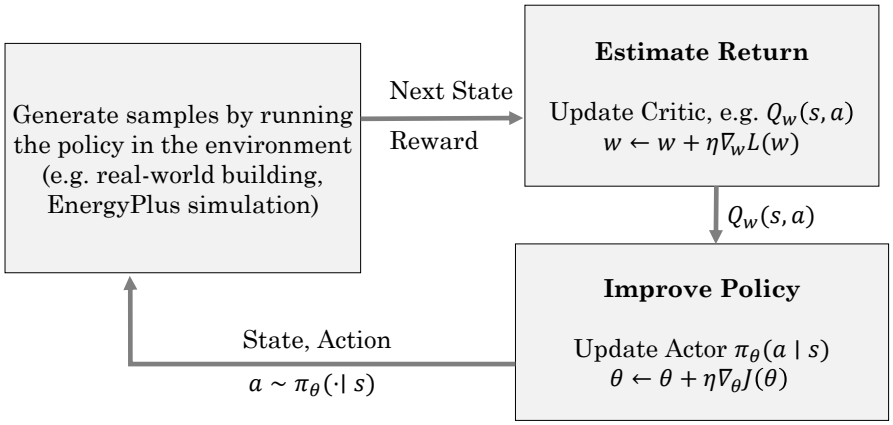

Figure 2: A typical scheme for generating data and training an off-policy actor-critic algorithm. An on-policy algorithm learns $V_w(s)$, instead of $Q_w(s,a)$ and generates trajectories, instead of single samples. The main difference between the algorithms will be the definition of the loss $L(w)$.

4.1. Off-policy methods

Off-policy algorithms focus on obtaining good estimates of the critic $Q_w(s,a)$, which is learned using variants of the Bellman equation. The policy is learned using the policy gradient theorem.

4.1.1. Critic network

The *critic* is a function telling the agent the value of states or actions to guide its decisions. Typical examples are $Q_\pi(s,a)$ or the value function $V_\pi(s)$. Algorithms learning only the critic are called *value-based methods* and have been studied in depth for the tabular case (finite state-action space), e.g. in [38].

Tabular methods are not convenient here, as many real-world applications have an infinite state space. Hence, we are interested in algorithms that approximate the optimal Q-function. It is then common to approximate the optimal Q-function by a neural network $Q^*(s,a) \approx Q_w(s,a)$. However, as learning this function is challenging, two important concepts are introduced: experience replay and target networks.

Experience replay aims to improve data efficiency and is used in [49, 21] and most subsequent off-policy algorithms. The idea is to store the transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ into a replay buffer. To improve data-efficiency, we sample mini-batches from the replay buffer to update the weights of the network. Target networks improve the stability of the Bellman targets. We introduce a network $Q_{\bar{w}}(s,a)$, which has the same architecture as the original network, but its weights are updated more slowly. The target weights are often updated using Polyak averaging [27].

Training the critic often involves learning the Q-function by minimising the loss function:

$$L(w) = \frac{1}{2}\mathbb{E}_{(s,a,s')\sim\mathcal{U}(D)}\left[\left(y(s,a,s') - Q_w(s,a)\right)^2\right], \tag{11}$$

where the data is sampled from the replay buffer, and where $y(s,a,s')$ is the Bellman target that varies between different algorithms. In DQN [21], the following target is used:

$$y_{DQN}(s,a,s') = r(s,a) + \gamma \max_{a'\in\mathcal{A}} Q_{\bar{w}}(s',a'). \tag{12}$$

The Bellman update of DQN in (12) does not use an actor, but uses instead the greedy policy to choose the action with the highest Q-value. This approach works well in small action spaces, but not for continuous control tasks. The reason is that the maximum in (12) becomes expensive to compute in infinite action spaces. We will focus on the following two algorithms: TD3 [47] and SAC [48]. TD3 is an algorithm that provides improvements over the DDPG algorithm [27], which is commonly applied in the HVAC settings, e.g., [30, 29]. It is the continuous analogue of the Double DQN algorithm [50, 51], aiming to learn the greedy policy $\mu_\theta(s) \approx \text{argmax}_{a\in\mathcal{A}}\, Q_w(s,a)$ using a neural network. TD3 aims to combat an overestimation bias using two different critic networks, resulting in more stable learning and better performance. In contrast, the SAC algorithm optimises the maximum entropy RL objective, aiming to learn the softmax policy, which is stochastic. Inspired by TD3, the SAC algorithm also learns two critics to give better estimates of the Q-values.

We briefly described the Bellman targets for the loss function in (11), which are used to update the critic. The DDPG algorithm uses the following update:

$$y_{DDPG}\left(s,a,s'\right) = r(s,a) + \gamma Q_{\bar{w}}\left(s', \mu_{\bar{\theta}}\left(s'\right)\right), \tag{13}$$

in which we use the target networks to compute the Bellman target to increase stability, but replace the maximisation of (12) by using the actor. The TD3 algorithm uses the minimum between the two target networks to calculate the residuals:

$$y_{TD3}\left(s,a,s'\right) = r(s,a) + \gamma \min_{j=1,2} Q_{\bar{w}_j}\left(s', \mu_{\bar{\theta}}\left(s'\right) + \varepsilon\right), \tag{14}$$

where $\varepsilon \sim \mathcal{N}(0,\sigma)$ is some noise (with σ small).[3] For the SAC algorithm, we use the following loss, which minimises the Bellman error of the maximum entropy RL objective:

$$y_{SAC}\left(s,a,s'\right) = r(s,a) + \gamma\left(\min_{j=1,2} Q_{\bar{w}_j}\left(s',a'\right) - \alpha \log \pi_\theta\left(a' \mid s'\right)\right), \tag{15}$$

where we sample $a' \sim \pi_\theta(\cdot \mid s')$. Note that all these updates are variants of the Bellman Equation (5).

[3] Policy smoothing is introduced to ensure that states located in the same neighbourhood have similar Q-values. This makes the agent less sensitive to perturbations.

4.1.2. Actor network

For both DDPG and TD3, the policy network is updated using the deterministic policy gradient theorem (10). The estimates of the gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{s\sim\mathcal{U}(D)}\left[\nabla_a Q_w(s,a)|_{a=\mu_\theta(s)}\nabla_\theta\mu_\theta(s)\right] \tag{16}$$

can be used in a gradient descent type algorithm to update the weights of the policy network. Note that this gradient uses the estimates of the critic to update the weights.

The policy π_θ in the SAC algorithm is typically parameterised by a Gaussian distribution.[4] The idea, justified theoretically in [48], is to minimise the Kullback-Leibler divergence between π_θ and the softmax policy. It aims to find the weights of the neural network that match the closest the softmax policy. This is equivalent to minimising the following loss:

$$L_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{U}(D),a\sim\pi_\theta}\left[\alpha\log\pi_\theta(a\mid s) - Q_w(s,a)\right]. \tag{17}$$

The gradient of this loss can be computed analytically and strongly resembles the DDPG update (16) (see [52] for the exact expression).

4.2. On-policy methods

For on-policy methods, the difficulty is to find good updates for the policy network, that give stable policy improvement guarantees. We refer to Appendix D for more details. This is motivated by the fact that our discussion will focus on off-policy methods, due to their increased data efficiency.

5. Case study

In this section, we use a data-centre case study to evaluate RL algorithms for continuous HVAC control.

5.1. Simulation environment

We simulate the data centre HVAC system with the building energy model (BEM) tool EnergyPlus, and train the RL algorithms with the OpenAI Gym framework [11]. The interaction between the Gym environment and the BEM is handled by the open-source wrapper from [19]. The wrapper sends the agent's actions to the simulator and waits until a state is returned. The wrapper then computes the reward and returns it along with the state back to the agent.

To obtain a fair comparison with other approaches, we implement the same environment as in [19, 7]. The data centre case study is chosen, as it has complex HVAC systems. It corresponds to an example environment in EnergyPlus, representing a two-room data centre[5]. The original file is modified by replacing the existing controller with an agent-based controller, as described in [19]. The data centre consists of two zones, each having an independent

[4] The algorithm uses a neural network, taking a state s as an input and the mean $\mu_\theta(s)$ and a diagonal covariance matrix $\sigma_\theta(s)$ as output. Then, the mean and covariance can be used to describe the Gaussian distribution. The output a is then sampled from this distribution.

[5] The name of the file is *2ZoneDataCenterHVAC_wEconomizer.idf*.

HVAC system. It consists of an air economiser, a variable volume fan, a direct-indirect evaporative cooler and a cooling coil (see Figure 3). To differentiate between the two zones, the west zone uses a direct expansion cooling coil, the east zone a chilled water cooling coil. The target temperatures of the evaporative coolers and the cooling coil are set to a common temperature, called the setpoint temperature, which is measured after the cooling coil. The air supply is adjusted by the variable volume fan.

The outdoor air enters through a damper and passes through evaporative coolers. The direct evaporative cooler humidifies the air, which causes the evaporation of water molecules, thus reducing the air temperature. The indirect evaporative cooler exchanges heat with a secondary air loop to cool the air down further without adding humidity into the principal air loop. It then passes through a cooling coil (in Figure 3, it uses chilled water, connected to a cooling tower) to cool down further if necessary. The setpoint temperature is measured after the cooling coil. Then, the air passes through a variable volume air fan, where the airflow rate is controlled by the agent. It controls the amount of air entering the server room. As cold air enters, warm air exits and leaves the building. Some of the exhaust air may re-enter the loop if necessary.

5.2. Problem formulation

We formulate the problem in the form of a MDP, in terms of states and actions and a reward function. We assume that the state and action space take continuous values. We take actions every fifteen minutes, and obtain a total of 35,040 steps for an episode (one year).

5.2.1. State space

We model the state space shown in Table 2, which includes the outdoor air temperature, the indoor air temperature in both zones, the power demand of the IT equipment and HVAC system. The power demand of the IT equipment is due to the electrical demand of the servers, which also depends on the indoor temperature. If it is colder in the server room, the computers require less internal cooling to operate, resulting in energy savings.

Table 2: Description of the state space.

Description	Notation	Range	Unit
Outdoor air temperature	T_{out}	$[-20, 50]$	$^{\circ}C$
West zone air Temperature	T_{west}	$[-20, 50]$	$^{\circ}C$
East zone air temperature	T_{east}	$[-20, 50]$	$^{\circ}C$
IT equipment demand power	P_{it}	$[0, 1]$	GW
HVAC electric demand power	P_{hvac}	$[0, 1]$	GW

5.2.2. Action space

We model the action space shown in Table 3, which includes the temperature setpoints and fan mass flow rates in both zones. Note that the range of actions corresponds to the range of possible actions, which is much broader than

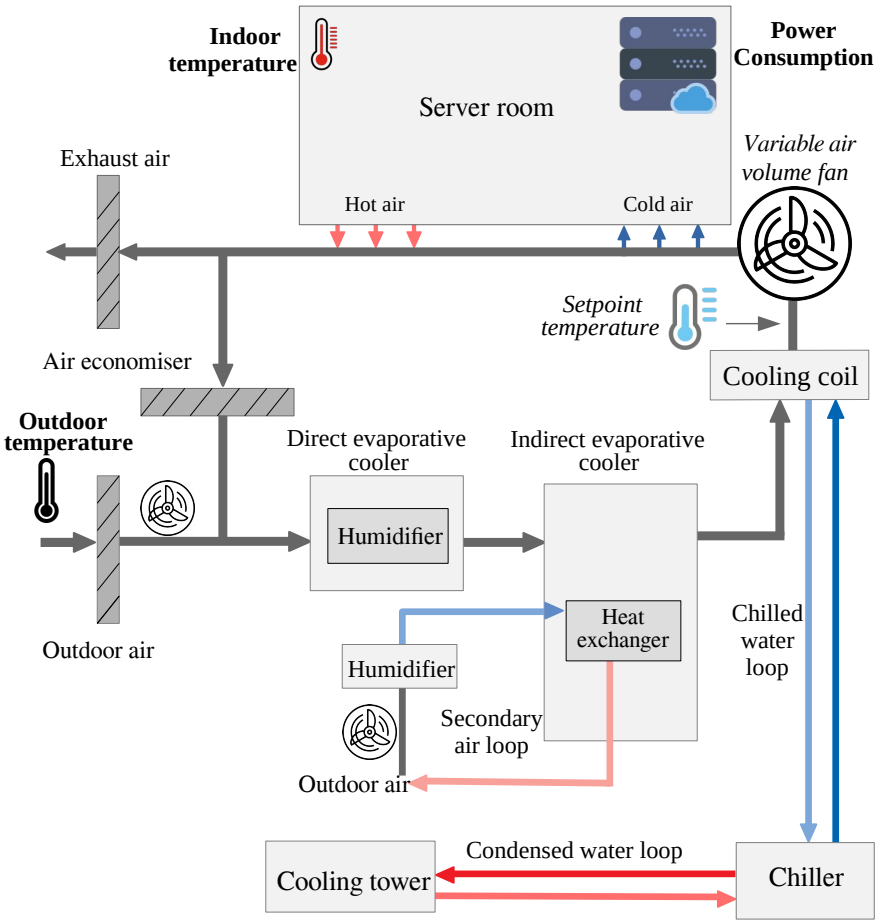

Figure 3: Simplified description of the air loop in the HVAC system of the east zone. The controllable parameters (belonging to the action space) are written in italics. The variables of the state space are written in bold.

the range of "good" actions. The agent should learn to manage the task using the feedback from the reward function. Domain knowledge should be incorporated into the reward function and not be used to define the desired range.

Table 3: Description of the action space.

Description	Notation	Range	Unit
West zone setpoint temperature	$T^{\text{set}}_{\text{west}}$	$[10, 40]$	$°C$
East zone setpoint temperature	$T^{\text{set}}_{\text{east}}$	$[10, 40]$	$°C$
West zone fan mass flow rate	$\dot{m}_{\text{west}}$	$[1.75, 7.0]$	kg/s
East zone fan mass flow rate	$\dot{m}_{\text{east}}$	$[1.75, 7.0]$	kg/s

5.2.3. Reward function

The objective of the task is to minimise energy consumption, while maintaining the indoor temperature within a pre-defined range. We define the reward function as:

$$r(s,a) = R_{\text{west}} + R_{\text{east}} - \lambda_P \left(P_{\text{it}} + P_{\text{hvac}}\right), \tag{18}$$

where R_i is a given reward when the temperature is within an acceptable range in the zone i, representing either the western or eastern zone and P corresponds to the power consumption. The minus sign is due to the fact that we want to minimise energy consumption, not maximise it. The ASHRAE guidelines for power equipment in data centres, recommend a range of between $18°C$ and $27°C$ [53], as undercooling may increase the risk of battery failure, resulting in server failure. To increase operational safety, we define a tighter range $[T_{\min}, T_{\max}]$ than the safe operation range. Inspired by [19], we define the following reward function:

$$R_i = \exp\left(-\lambda_1 \left(T_i - T_{\text{tgt}}\right)^2\right) - \lambda_2 \left([T_{\min} - T_i]_+ + [T_i - T_{\max}]_+\right), \tag{19}$$

where $T_{\text{tgt}} = (T_{\min} + T_{\max})/2$ is the target temperature, i.e., the midpoint of the interval and $[x]_+ = \max(x, 0)$. The plot of this function is presented in Figure 4. The reward will be close to 1 if the temperature is within the desired range, and will be small or negative when the temperature is too cold or too hot. We use a Gaussian shape to motivate the agent to maintain the temperature close to the centre, which is more robust than a simple trapezoidal reward function. We add a trapezoid penalty to regulate the learning when the temperatures are out of the desired range, as the Gaussian tends too quickly to 0, which may result in sparse rewards. The penalty allows the agent to distinguish between very and moderately bad actions, which can help at the start of the training. The weight λ_P is around 10^{-5}, as the power consumption is expected to be around $100\,kW$. This parameter addresses the trade-off between temperature control and energy savings. We will evaluate the importance of hyperparameters and their impact on performance. An example of the hyperparameter settings is as follows: $T_{\min} = 22°C$, $T_{\max} = 25°C$, $\lambda_1 = 0.2$, $\lambda_2 = 0.1$ and $\lambda_P = 10^{-5}$.

5.3. Exogenous input

The state space can be divided into a controllable component and an exogenous component. The agent can only influence a subset of the state space (e.g. indoor temperature and HVAC demand power), but cannot influence the outdoor temperature. For the exogenous component, we use external data to train the agent, such as weather data provided with EnergyPlus.

We design our experiments to train and evaluate the algorithms in different weather conditions. In this way, we can test whether the agent manages to maintain the temperature under new weather conditions, which is absolutely essential in a practical application. If we train and evaluate on the same dataset, it is unclear whether the agent learns the task or overfits to the weather data. This could lead to an agent that only manages to control the system under specific weather conditions. To reduce the risk of overfitting, we alternate weather data from various locations during training, even though it may yield a less stable training process.

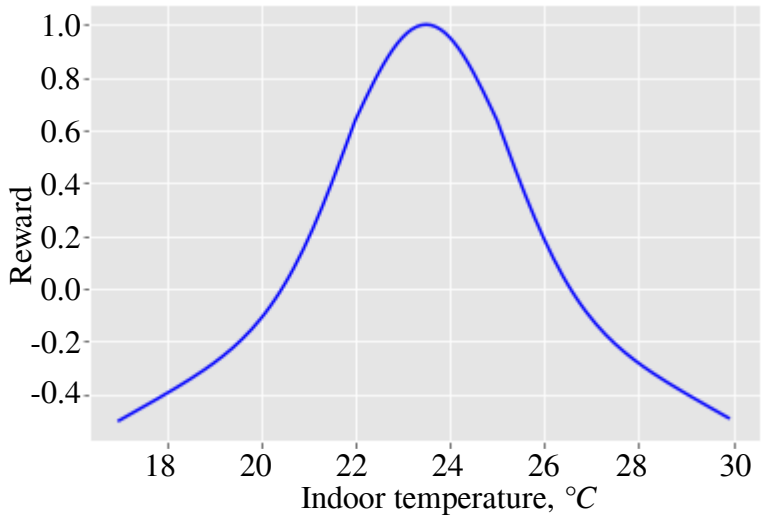

Figure 4: Graph of the reward R_t with the parameters $\lambda_1 = 0.2$ and $\lambda_2 = 0.1$. λ_1 corresponds to the precision of the Gaussian function, i.e how steep the peak is. λ_2 controls the slope of the trapezoid function. We give no trapezoid penalty for temperatures between 22 and 25 $°C$.

5.4. Performance metrics

We use the following two metrics to evaluate the algorithm's performance: energy saving and thermal stability. Energy consumption is measured by integrating the total electric demand power $P_{\text{tot}} = P_{\text{it}} + P_{\text{hvac}}$ of the whole data centre over a year. The average power consumption of the data centre is around 100 kW, corresponding to an annual energy consumption of about 3.2 TWh. In our experiments, we report the average yearly power consumption.

To evaluate thermal stability, we assume that the temperature distribution in the two zones approximately follows a Gaussian distribution. Therefore, we use the inferred mean μ and standard deviation σ to measure the agent's ability to control the temperature in the given zone. Ideally, μ is close to the target temperature of 23.5 $°C$ and the standard deviation σ as small as possible.

5.5. Choice of algorithms

The experiments will focus on evaluations of the following four algorithms: SAC, TD3, TRPO and PPO. Other algorithms, such as DDPG and A2C, are not selected as they were not able to learn the task successfully in our case study. Note that TD3 and PPO are improved versions of DDPG and A2C respectively. The TRPO algorithm is included for comparison with prior work [19].

5.6. Implementation details

We implement the algorithms using the Pytorch version of the Stable Baselines framework [54].[6] We use a discount factor of $\gamma = 0.99$ for all algorithms, meaning that the agent takes the actions that maximise the expected

[6]The TRPO algorithm is implemented with the original version of Baselines, as in [19].

reward over an effective time horizon of a hundred steps (about one day in real-time). The states are normalised before being fed into the neural network for numerical stability. The actions a sampled from π are in the range of $[-1, 1]$, which need to be scaled back to the correct range for EnergyPlus to interpret them correctly.

We use the neural network structure and hyperparameters recommended by Stable Baselines [54], with only minor differences, e.g., using a larger horizon for the on-policy algorithms to improve stability (the parameters used are listed in Appendix E, often corresponding to the hyperparameters of the original implementations.). Although some algorithms may obtain a greater reward by fine-tuning the hyperparameters, we argue that the algorithms should learn the task without it. Furthermore, when deployed in a physical building, a hyperparameter search becomes impossible, so an algorithm needs to be able to learn the task consistently even with suboptimal hyperparameters.

6. Experiments

We perform a series of experiments to demonstrate the robustness of the algorithms with respect to changing weather dynamics and different hyperparameters. For all experiments, we test all algorithms to see if the same trends can be observed for all of them. Then, we take a closer look at the results and discuss the observed trade-off between energy consumption and indoor climate management.

6.1. Robustness to unseen weather conditions

We first evaluate the robustness of different actor-critic algorithms using weather data as an exogenous input. We perform three experiments, described in Table 4. The algorithms are trained for twenty episodes using weather data from one location (Helsinki, Berlin or San Francisco), where we use the same weather data for each episode. Then the algorithm is evaluated at a different location, with weather data from Copenhagen. This experiment aims to demonstrate that the algorithms are robust to changes in temperature dynamics.

Table 4 shows the results. We can observe that the controller has better results (in terms of energy savings) when trained with weather data from Helsinki rather than San Francisco. Therefore, we conclude that it is be preferable to use data from locations that have similar weather conditions to the location where the controller would be deployed. The results are promising, as all the tested algorithms work reasonably well under unseen weather conditions, even when trained with data from a different climate.

Motivated by the results of this first experiment, we perform a second experiment, asking whether it helps to use weather files from various locations to increase the robustness of the agent. Instead of using the same weather information for each episode, we use episode weather information from another location. For example, for the first episode we use data from Oslo, for the second year from Bergen and then continue as described in Table C.9. As these locations are in northern Europe, we assume that they have similar weather conditions to Copenhagen. We evaluate the algorithms using weather data from Copenhagen as well.

Table 5 shows the results. We can observe that training using various locations can significantly increase the performance of the off-policy algorithms on the test environment, in terms of both energy savings and thermal stability.

Algorithm	Weather data	Training location			Test location		
		P_{tot}	μ	σ	P_{tot}	μ	σ
SAC	Helsinki	102.2	23.2	1.2	102.8	23.2	1.2
TD3	Helsinki	96.4	23.0	1.6	95.2	23.2	1.4
PPO	Helsinki	98.5	23.0	1.6	98.1	22.9	1.3
TRPO	Helsinki	96.5	22.5	2.8	96.5	22.4	2.5
SAC	Berlin	104.7	23.3	1.2	102.8	23.4	1.2
TD3	Berlin	109.7	22.6	2.1	105.6	23.0	1.9
PPO	Berlin	107.8	22.6	1.6	105.7	22.6	1.3
TRPO	Berlin	102.3	22.3	2.8	98.6	22.3	2.6
SAC	San Francisco	112.1	23.2	1.1	106.5	23.3	1.3
TD3	San Francisco	107.5	23.0	1.7	105.8	23.5	2.5
PPO	San Francisco	112.1	23.2	1.1	106.5	22.9	1.3
TRPO	San Francisco	109.4	22.4	2.7	104.4	22.7	2.8

Table 4: Influence of different weather data for training on the performance of the algorithms, tested on Copenhagen weather data. The values are averaged over the whole year. The following hyperparameters were used: $T_{\text{min}} = 22°C$, $T_{\text{max}} = 25°C$, $\lambda_1 = 0.2$, $\lambda_2 = 0.1$ and $\lambda_P = 10^{-5}$.

The PPO algorithm manages to reduce energy consumption, but at the cost of a worse indoor climate. This can be explained by pointing out that changing the weather dynamics each year reduces the stability of the learning process. In general, this experiment shows that using weather data from various locations can increase the agent's understanding of the weather transitions and is, therefore, better able to adapt to them. This validates experiments performed by Moriyama et al. [19], who also showed that using multiple weather locations during training can improve the performance of the algorithms. We will take a closer look at the results of this experiment in Section 6.3.

Algorithm	Weather data	Training location			Test location			Energy consumption
		P_{tot}	μ	σ	P_{tot}	μ	σ	
SAC	Northern Europe	102.1	23.3	1.2	100.4	23.3	1.2	-13.0 %
TD3	Northern Europe	95.6	22.8	1.5	93.4	22.7	1.2	-19.0 %
PPO	Northern Europe	95.9	22.0	2.6	94.2	22.0	2.5	-18.3 %
TRPO	Northern Europe	100.7	22.4	2.7	98.5	22.4	2.6	-14.6 %
Baseline	-	-	-	-	115.3	23.7	0.4	-

Table 5: Table comparing various reinforcement learning algorithms, trained using alternative weather data every year from locations in Northern Europe (see Table C.9). We used the same hyperparameters as in Table 4. We compared the results to the model-based controller implemented into EnergyPlus, which uses knowledge of the environment to compute the actions.

We compare the results to a baseline controller, commonly used in the industry, that uses a model-based set point

manager with a zone thermostat cooling point at $23.0°C$. We refer the reader to Li et al. [29] for more details. That controller, which aimed to maintain the temperature precisely, is clearly better at keeping the temperature within the desired range, but it uses internal simulation information to compute the strategy, which the RL controllers do not have access to. On the other hand, all RL algorithms reduce energy consumption by at least 10 %. Note that the baseline controller is more complex than a simple rule-based controller.

6.2. Sensitivity of the reward function

We now perform a sensitivity analysis of the algorithms by tuning the hyperparameters of the reward function, evaluating their performance in Table 6. They include the temperature range used in the definition of the reward function, i.e. $[T_{\min}, T_{\max}]$, the precision λ_1 of the Gaussian, the slope of the trapezoid λ_2, and the weight of the trade-off between temperature maintenance and energy savings λ_P.

By changing the hyperparameters, we can shape the range of acceptable temperatures. Notice that having a larger range $[T_{\min}, T_{\max}]$ and a lower precision λ_1 can increase energy savings. On the other hand, a high precision λ_1 and a tight range implies that the mean temperature is generally closer to the target temperature of $23.5°C$. The value of the parameter λ_2 is not important, as it is only used to guide the agent at the beginning of the training. We also noticed that the on-policy algorithms (PPO and TRPO) are sensible to the choice of the range (showing a lower mean with a larger range). In contrast, SAC shows similar means, standard deviations and energy consumption for all parameters, making it possibly the most robust algorithm. For the other algorithms, the reward function needs to be shaped more precisely to achieve the desired results.

The weight λ_P is used to find a balance between thermal stability and energy savings. A high value means that energy savings are more important, while a low value means that the agent is aimed at maintaining the temperature precisely. The initial parameters performed the best, probably because the other parameters were optimised for the value of $\lambda_P = 10^{-5}$. The dependency between the parameters is complex, making it difficult to define the reward function so as to obtain the desired results.

While the results are noisy, this at least shows that all algorithms are able to learn the task under all tested hyperparameters. This implies, that while better parameters can lead to better performance, the exact choice is not essential. It is possible to train an algorithms without much trial-and-error, which is important in practice.

6.3. Thermal stability and energy saving

Given the above results, it is interesting to make a further study on temperature control and energy savings for the off-policy algorithms, SAC and TD3. We first plot the temperature evolution over one year and the distribution for both zones for the SAC algorithm in Figures 5 and 6, respectively. The results in Figure 5 indicate that SAC can successfully keep the temperature within the desired range (between the two green lines), with a relatively low variance.

Algorithm	Range		Hyperparameters			Test Set		
	$T_{\min}$	$T_{\max}$	λ_1	λ_2	λ_P	P_{tot}	μ	σ
SAC	23.0	24.0	0.2	0.1	10^{-5}	103.2	23.2	1.2
TD3	23.0	24.0	0.2	0.1	10^{-5}	**92.9**	23.2	1.3
PPO	23.0	24.0	0.2	0.1	10^{-5}	106.3	23.3	1.2
TRPO	23.0	24.0	0.2	0.1	10^{-5}	105.1	23.4	**1.1**
SAC	23.0	24.0	0.5	0.1	10^{-5}	100.8	23.2	1.2
TD3	23.0	24.0	0.5	0.1	10^{-5}	97.5	23.3	1.3
PPO	23.0	24.0	0.5	0.1	10^{-5}	105.1	23.4	**1.1**
TRPO	23.0	24.0	0.5	0.1	10^{-5}	100.8	22.3	2.7
SAC	23.0	24.0	0.5	0.5	10^{-5}	100.8	23.2	1.2
TD3	23.0	24.0	0.5	0.5	10^{-5}	101.2	22.9	1.7
PPO	23.0	24.0	0.5	0.5	10^{-5}	105.1	23.4	1.1
TRPO	23.0	24.0	0.5	0.5	10^{-5}	102.4	22.4	2.7
SAC	22.5	24.5	0.5	0.1	10^{-5}	104.8	23.3	1.3
TD3	22.5	24.5	0.5	0.1	10^{-5}	99.2	23.1	1.4
PPO	22.5	24.5	0.5	0.1	10^{-5}	102.1	23.4	1.5
TRPO	22.5	24.5	0.5	0.1	10^{-5}	98.4	22.3	2.7
SAC	22.0	25.0	0.5	0.1	10^{-5}	102.5	23.2	1.2
TD3	22.0	25.0	0.5	0.1	10^{-5}	103.7	23.3	1.7
PPO	22.0	25.0	0.5	0.1	10^{-5}	99.4	22.9	1.3
TRPO	22.0	25.0	0.5	0.1	10^{-5}	104.7	22.7	1.4
SAC	22.0	25.0	0.2	0.1	$8 \cdot 10^{-6}$	104.3	23.1	1.2
TD3	22.0	25.0	0.2	0.1	$8 \cdot 10^{-6}$	100.9	23.1	1.6
PPO	22.0	25.0	0.2	0.1	$8 \cdot 10^{-6}$	102.8	21.7	2.1
TRPO	22.0	25.0	0.2	0.1	$8 \cdot 10^{-6}$	98.6	22.4	2.4
SAC	22.0	25.0	0.2	0.1	$1.2 \cdot 10^{-5}$	103.4	23.3	1.2
TD3	22.0	25.0	0.2	0.1	$1.2 \cdot 10^{-5}$	96.6	23.0	1.3
PPO	22.0	25.0	0.2	0.1	$1.2 \cdot 10^{-5}$	101.8	22.0	1.9
TRPO	22.0	25.0	0.2	0.1	$1.2 \cdot 10^{-5}$	**97.6**	22.2	2.7
SAC	22.0	25.0	0.2	0.1	10^{-5}	**100.4**	23.3	**1.2**
TD3	22.0	25.0	0.2	0.1	10^{-5}	93.4	22.7	**1.2**
PPO	22.0	25.0	0.2	0.1	10^{-5}	**94.2**	22.0	2.5
TRPO	22.0	25.0	0.2	0.1	10^{-5}	98.5	22.4	2.6

Table 6: Influence of the hyperparameters of the reward function on the performance of the algorithms, trained on Northern Europe data and tested on Copenhagen weather data.

According to our study in Section 6.1, TD3 has the same standard deviation, and can save more energy than SAC, but has a lower mean. This suggests that TD3 is better than SAC. However, if we explore its temperature evolution (see Figure 7) and temperature distribution (see Figure 8) further, we find that TD3 does not manage to maintain the temperature as well as SAC. Note that the distributions in Figure 8 are not well-approximated by a Gaussian distribution. Using the standard deviation σ to compare two distributions makes only sense if both are Gaussian. As this is not the case here, it is an inappropriate measure for evaluating thermal stability. From Figure 7, we see that TD3 is worse at maintaining temperature than SAC, when trained with equal data. Nevertheless, it often saves more energy, as illustrated in Tables 4 and 6.

We therefore conclude that there is a trade-off between thermal stability and energy savings. Depending on different real-world applications, some fields can use more energy to achieve more precise temperature control, such as the biochemistry field, while others could relax it, but prioritise energy saving, such as residential buildings. Data centres and industrial buildings are in between, where a lower energy consumption significantly reduces costs, but the temperature needs to be maintained for safe operation. As noted in Section 6.2, we can partially address this trade-off by adapting the hyperparameters of the reward function to shape the desired behaviour to the case study, such as the range $[T_{\text{min}}, T_{\text{max}}]$ or the weight λ_P. In Section 6.4, we see that part of this trade-off may be due to the TD3 algorithm not having yet converged to a stable policy. We will discuss in Section 6.6 how potentially to improve both energy savings and thermal stability.

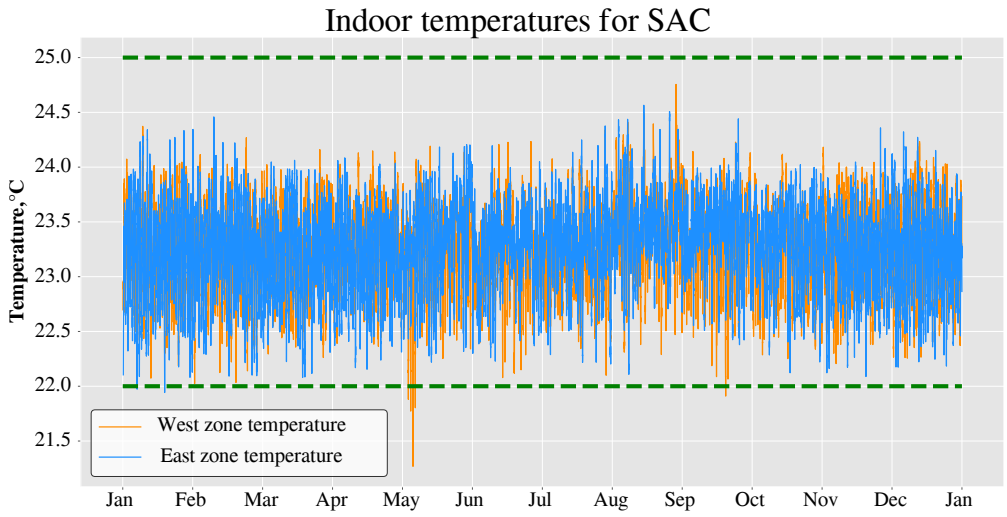

Figure 5: Evolution of temperatures over a year for the SAC algorithm, tested with Copenhagen weather data. The values correspond to the moving average over six hours.

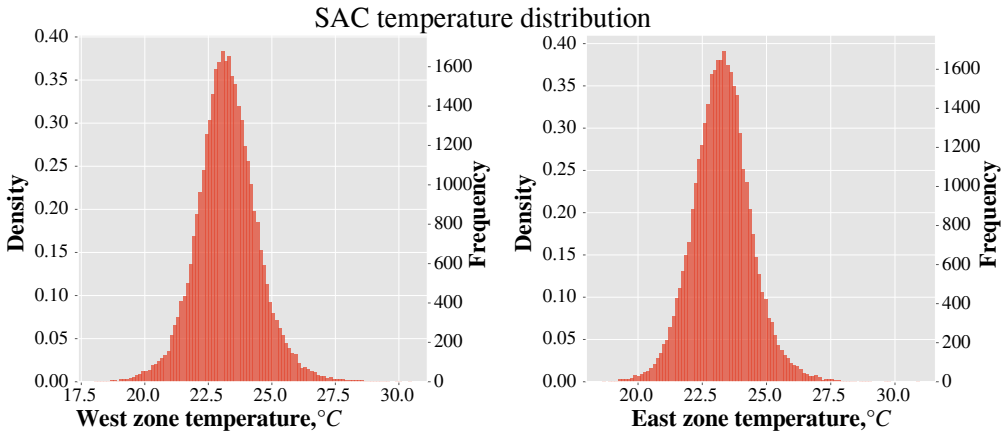

Figure 6: Temperature distributions in both zones over a year. The distribution follows a Gaussian distribution when trained with SAC.

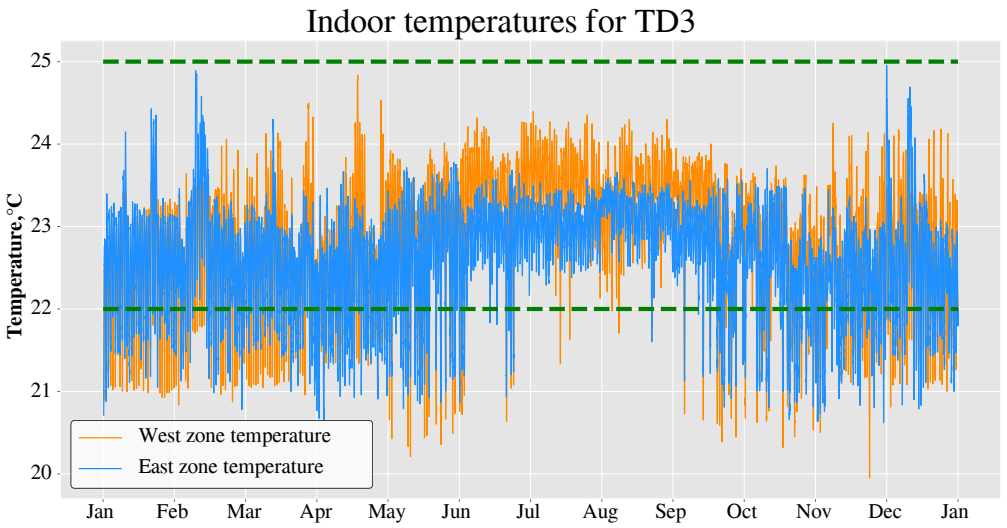

Figure 7: Evolution of temperatures over a year for the TD3 algorithm, tested on Copenhagen weather data. The values correspond to the moving average over six hours. The training comprised 20 episodes. The temperature control is not yet satisfactory.

6.4. Analysis of data efficiency

We now evaluate the impact on control performance by increasing the training sample size from 20 to 60 episodes.[7] We first compare the off-policy algorithms, SAC and TD3. The result of SAC with 60 episodes does not show much difference with the training size of 20 episodes (see Figure 5 and 6), thus we do not plot the results here. However, the TD3 results are becoming more similar to the results of SAC (see Figure 9), where temperatures are almost within the desired range, but with higher average power consumption than before (103.2 kW). In addition, the temperature distribution is also closer to a Gaussian distribution (see Figure 10), although the distributions still show differences between the two zones. This difference in the distribution indicates another property of the algorithm: the stability

[7]For the 21st episode, we reuse the same weather data as for the first episode and continue alternating the locations in the same order, as described in Table C.9.

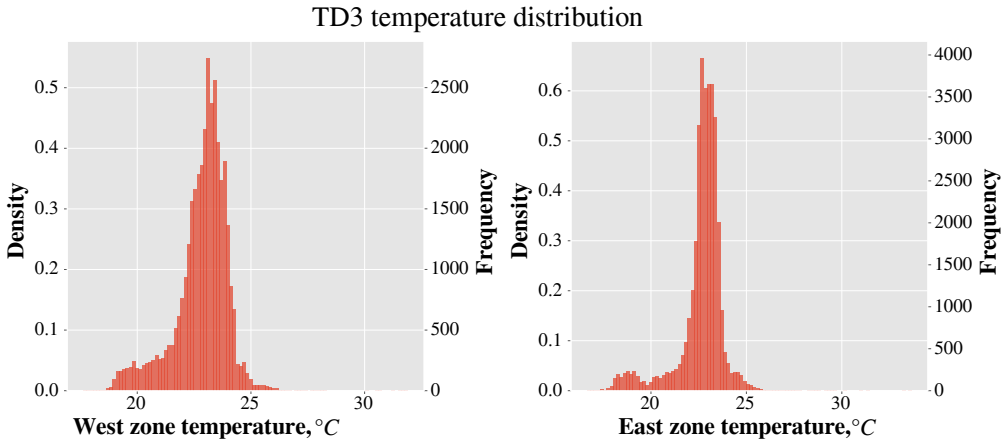

Figure 8: Temperature distributions in both zones over a year. The distribution for TD3 is not well approximated by a Gaussian distribution. The training comprised 20 episodes.

of the learning process. SAC obtains similar results for both zones (see Figure 6), whereas TD3 obtains significantly different results for both zones (see Figure 8 and 10), even though the zones are very similar. Therefore, in comparison, not only can the SAC algorithm achieve faster convergence (in terms of episodes), but the training process is more reliable.

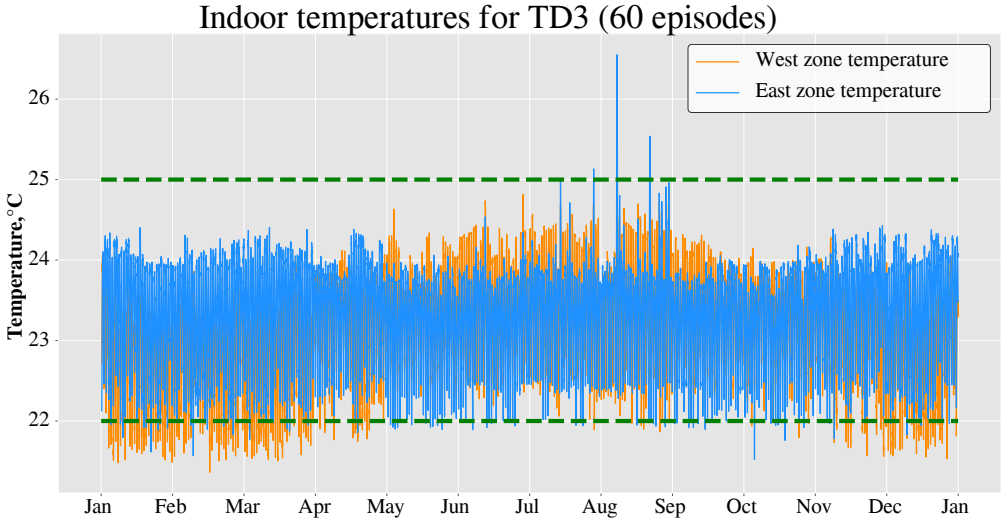

Figure 9: Evolution of temperatures over a year for the TD3 algorithm, tested on Copenhagen weather data. The values correspond to the moving average for six hours. We trained for 60 episodes, instead of 20. We obtain a policy that maintains the temperature better (but uses more energy) compared to the less trained version.

The results can be explained with reference to the theoretical properties of the algorithms. The policy of the SAC algorithm follows a Gaussian distribution, explaining the shape of the results in Figure 6, as the setpoint temperature is closely correlated with the indoor temperature. In contrast, the TD3 policy is a deterministic function, specifying a

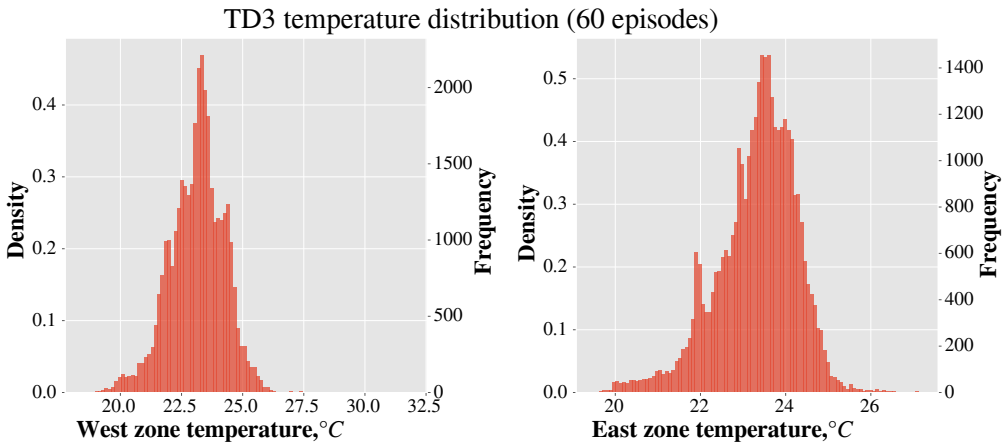

Figure 10: Temperature distributions in both zones over a year. We trained for 60 episodes instead of the usual 20. The distribution for TD3 became closer to a Gaussian.

given action. Such policies are biased towards taking similar actions in similar states, explaining the sharp peaks in Figure 8. This behaviour is often desirable in deployment, but it makes training difficult in practice. At the beginning of the training, it is important to explore the environment to be able to distinguish between good and bad states. With stochastic policies, exploration is handled naturally, while deterministic policies have to rely on the stochasticity of the environment to end up in different states.[8] In SAC, the agent needs to reduce the Q-values of bad state-action pairs significantly, so that they do not happen frequently anymore, allowing the agent to identify quickly which actions to take. For TD3, if an action is bad, it takes time before the agent realises it and adapts the network's weights in order to take different actions in such states. Furthermore, entropy regularisation makes the policy easier to optimise with gradient descent, which implies a more stable learning process than for deterministic policies [55]. With more training episodes, the policy becomes more stable and can outperform SAC, but this requires a large amount of data for training, e.g., covering nearly sixty years in this paper. However, this is an unacceptable amount of data for many real-world applications. In contrast, SAC requires much less data, less than ten years and shows clear signs of learning during the first year (see Figure 14).

We perform the same experiment for the on-policy algorithms (PPO and TRPO). PPO performs similarly to TD3, achieving similar thermal stability after 60 training episodes (see Figure 11), although its results are clearly worse than TD3 for previous episodes. PPO also obtains a Gaussian temperature distribution (see Figure 12). TRPO shows significant improvements in maintaining the temperature and a remarkably stable learning process, but its results are still worse than for the other three algorithms. The on-policy algorithms are more stable than TD3, which can be subject to a large performance drop, as seen in Figure 13. This figure also shows how quickly the SAC is able to learn the task compared to the other algorithms. SAC is both stable and data-efficient, making it a promising algorithm to

[8] In deterministic environments, it is common to add noise to the actions to encourage exploration. In our experiments, it performed significantly worse with the noise. This may be due to the fact that we did not reduce the noise during the course of training.

consider for future work.

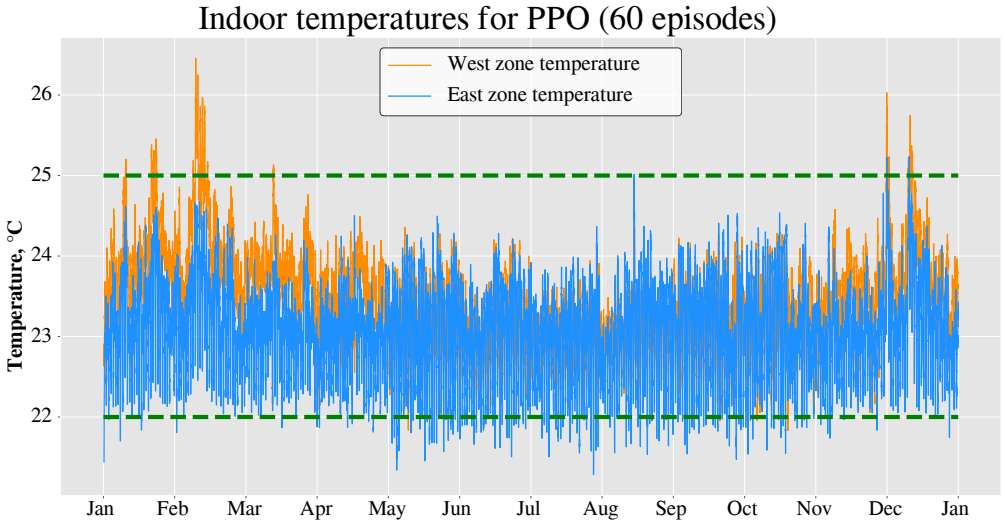

Figure 11: Evolution of temperatures over a year for the PPO algorithm, tested on Copenhagen weather data. The values correspond to a moving average over 6 hours and trained for 60 episodes. The algorithm is eventually able to maintain temperature, but requires far more data than SAC.

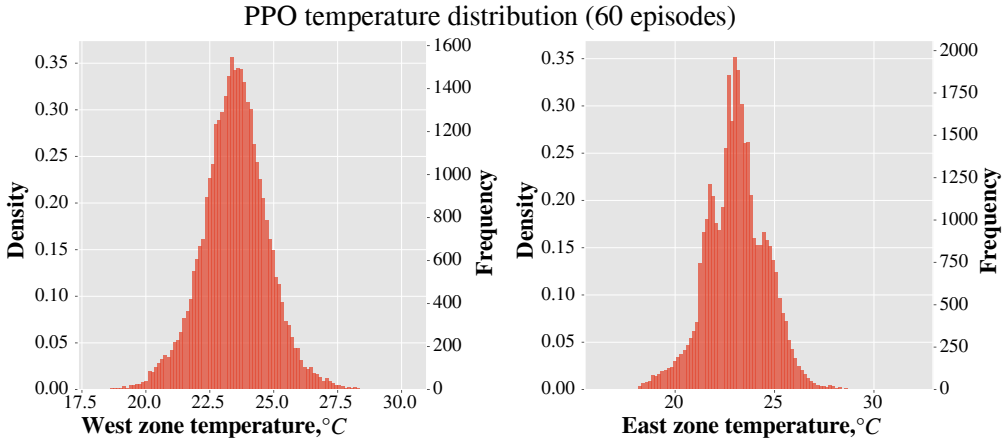

Figure 12: Temperature distributions in both zones over a year. The distribution for PPO also follows a Gaussian distribution. We trained for 60 episodes, instead of the usual 20.

6.5. Research implications

Our experiments show consistently that all applied continuous control RL algorithms are able to manage the HVAC systems, while keeping the temperature of the data centre within the desired range. The energy consumption is reduced by up to 15% compared to the model-based EnergyPlus controller. The SAC algorithm is even able to reach these results after fewer than 10 episodes. The amount of data required by SAC to stabilise the indoor temperature is up to ten times less than for the other algorithms. This might be ascribable to the use of experience replay and the

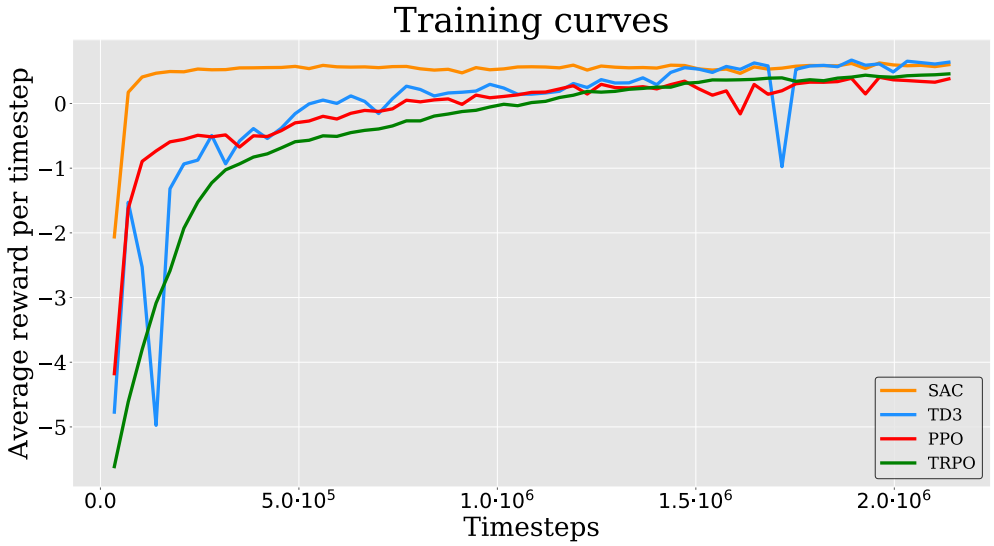

Figure 13: Training process of the RL algorithms. We show the reward $r(s, a)$ received per timestep, averaged over the whole episode. SAC reaches a high reward very quickly, showing its data-efficiency. Ultimately, all algorithms have a similar performance.

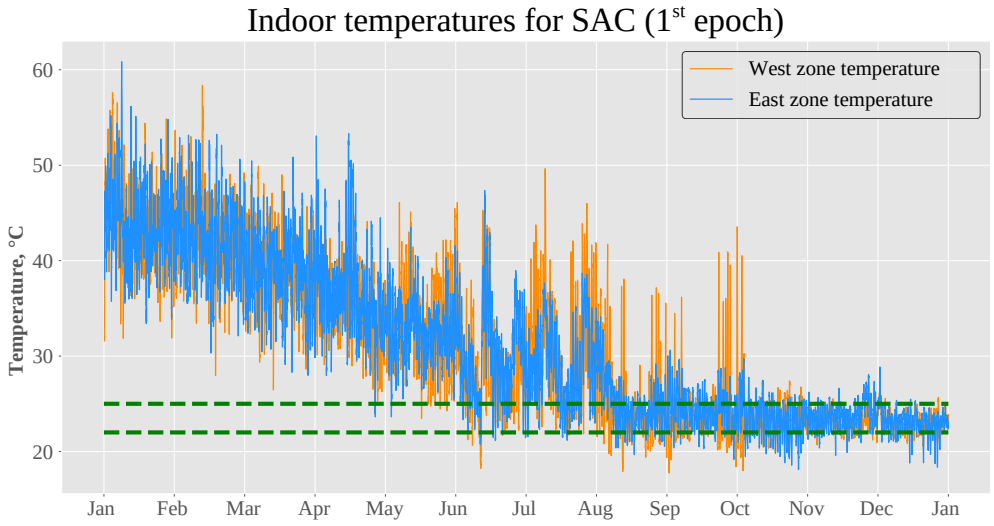

Figure 14: Evolution of the inside temperatures during the first year of training for SAC. While the policy is inadmissible at the beginning of the training (this issue can potentially be solved using imitation learning [56]), it clearly shows improvements and ends up with an almost acceptable policy at the end of the year.

learning stability of entropy regularisation. Due to this and the good capability of handling domain shifts, SAC seems to be predestined for real-world deployment despite the slightly higher energy consumption.

In contrast, the on-policy algorithms show a stable training process, are fast in terms of wall-clock time and can obtain excellent policies, as shown in Moriyama et al. [19] (trained with 360 episodes). However, the amount of data required by these algorithms before obtaining a good policy is inadmissible, as they cannot reuse past experience.

While also using experience replay, the TD3 algorithm needs significantly more data to be able to maintain the temperature as well as SAC. In the RL literature, the SAC and TD3 perform similarly, but in our case study, SAC performed significantly better. This is possibly explained by the fact that our environment is noisy, whereas most environments in the RL literature [11] are deterministic. It is possible that stochastic policies are able to handle noisy environments better.

With regard to robustness and generalisation, we demonstrate that all algorithms are able to learn the task with different hyperparameters in the reward function. This shows improvements over commonly applied algorithms, such as DDPG, that have been applied successfully in other tasks (as in [31, 30]), but had to rely on smaller networks than the original architecture [27] to learn the task successfully. The benchmarked algorithms work without having to run a hyperparameter search. This is again important with a real-world deployment in mind, because we cannot restart the training to change the hyperparameters. Furthermore, our results confirm that all algorithms generalise to unseen weather dynamics, which is essential for real-world applications, as the weather during deployment will be different than that during training. This confirms results from the literature that showed similar results for other algorithms, such as tabular Q-learning [36], DQN [35] and DDPG [31].

In our reward function and analysis, we focused on maintaining indoor temperatures in order to reduce energy consumption. A more involved case-study would also have to consider other parameters, such as humidity, air quality and more (they are also monitored in the simulation, but have not been used by the algorithm). To control these effectively, we would need to incorporate these as well into the reward function. This can present additional challenges in designing the reward function. Even when controlling few parameters, we might want to modify the reward function to punish other undesirable behaviour[9]. It is important to consider all exogenous parameters that can influence the indoor temperature (such as insulation or human activities) and whether they can be effectively measured. Furthermore, we might want to react to other constraints such as energy prices, available renewable energy, and more. As the algorithms support multi-dimensional state and action spaces, it is straightforward to apply the same algorithms in such applications. However, this leads to additional practical challenges, such as simulating physically realistic environments, monitoring the desired parameters that define the state space and especially defining a reward function that addresses these additional trade-offs.

Moreover, it is important to select appropriate metrics for the evaluation of thermal management. In this paper, we use the mean and standard deviation of the temperature distributions. Comparing algorithms using these metrics only makes sense if the distributions are all Gaussian. However, as we saw for TD3, this is not always the case. This can result in making wrong conclusions about the qualities of the algorithms[10]. A statistical analysis of the results may be required in order to evaluate the algorithms properly. Other metrics to measure comfort in residential buildings,

[9] In our case study, we did not include the airflow rate in the reward function, which may result in undesirably high airflow or noise in the building. In addition, it would certainly be of benefit to try to address fluctuation by penalising actions that differ greatly from previous actions.

[10] For instance, in Table 4, TD3 appears to be better than SAC, due to better energy consumption and similar standard deviation. However, as discussed in Section 6.3, this may not be the case.

such as predictive mean value (used in [28]) or predicted percentage dissatisfied (used in [20]) are subjective and are computed with parameters that are not easily tractable. Average temperature violation, used in [31] is unable to distinguish between good policies. Another commonly used metric is simply the expected reward $J(\pi)$. This could seem natural, as it corresponds to the objective the algorithms optimise, but it has little meaning for engineers. While the choice of an appropriate metric is case-study dependent, we argue that this aspect should be considered carefully in future.

6.6. Future directions

We discussed the trade-off between energy consumption and thermal stability. It is difficult to assess this trade-off quantitatively, as it is difficult to measure thermal stability. It is certainly impossible to reduce the trade-off completely, as an algorithm that was required to control the temperature precisely would have less freedom to operate than a policy with laxer requirements that could reduce energy consumption more. However, we believe that the results can be improved with respect to both metrics if we use better algorithms and neural network architectures that are able to take account of temporal dependencies. A possible direction for improving model-free methods is the use of distributional RL [57], which would improve estimates of the expected reward, leading to additional safety. This has shown to help improve the performance in real-world applications in complex, stochastic environments [58]. Another important direction is using networks that take not only the current state as input, but a sequence of previous states. This can give the agent important additional information, which can help its decisions. For example, it can determine if the temperatures have been increasing in the previous hours and take actions accordingly. For instance, convolutional neural networks (CNN) and recurrent neural networks (RNN) can be used for this purpose. However, it is challenging to apply RNNs to off-policy algorithms, because of memory issues.

Testing the algorithms in other environments is necessary in order to examine further the generalisation properties of the studied algorithms [9]. It is also essential to deploy such controllers into real-world environments to see how the algorithms handle additional challenges, as well as the domain shift between simulation and reality. Furthermore, we can reduce cold starts using imitation learning [56], where we use existing data to initialise the networks and continue the training from there. It is important to test more model-based approaches as well. This can be done by using the building models that have been developed for MPC applications, where the agent predicts the trajectories using that model, while learning the policy from the data. To make model-based RL more scalable, we can also learn the model directly using the training data (in addition to the actor and critic networks). This has been done notably in [7, 8] and could be used to increase data efficiency even further. These methods should be combined in future work to reduce the gap in training RL controllers directly in the real world, at least in an experimental environment. The current method, namely to train the controller first in a simulated environment before deploying it into the real-world is not ideal, as one of the main motivations in using RL methods is to be able to manage HVAC systems efficiently without designing these simulations.

We believe that this evaluation study will encourage the further democratisation of algorithms for the continuous

control of HVAC systems. It should be noted that many of the major challenges posed by the RL for HVAC control are not unique to this setting. Efficiency, robustness, safety, scalability, interpretability, reward function design, and transfer from simulation to real-world deployment have also been the subject of significant research efforts in other artificial intelligence-based fields, such as robotics. We therefore believe that the smart building sector should closely monitor progress in the related fields, as it can greatly benefit from it. We believe that robustness and data efficiency are important topics that deserve further investigation.

7. Conclusion

Reinforcement Learning (RL)-based strategies are important for smart building systems, due to their ability to learn from experience in stochastic environments and their scalability. Realistic case studies require controllers that are able to manage multiple parameters (temperature, humidity, air quality) in multiple zones. For these problems, the commonly used value-based methods are not straightforward to apply. Therefore, we discussed the theoretical background needed to define algorithms that are able to handle such problems. We evaluated the algorithms on a simulated data centre case study, using EnergyPlus. The objective was to reduce energy consumption, while keeping indoor temperature within the pre-defined range. We addressed technical issues regarding the real-world deployment of RL-based controllers, including data efficiency and robustness to different weather conditions and reward functions. We analysed the trade-off between energy consumption and thermal stability in this case study. Although a growing number of RL-based applications have emerged for building management, only a few studies have aimed to compare different RL algorithms with each other and discuss more technical questions. This paper fills this gap, helping users understand better the properties of different algorithms for indoor climate and energy management, and facilitating the selection of RL algorithms for specific applications.

The experiments showed that all algorithms are able to maintain indoor temperatures, while reducing energy consumption with respect to model-based controllers by more than 13 %. The algorithms can learn the task under different hyperparameters and show robustness when using unseen weather conditions. This is promising with regard to the scalability and generalisation of RL-based controllers. The temperature distributions of all algorithms became more similar in the end, except that Soft Actor Critic can obtain these consistently with up to ten times less data and shows clear improvements the first year. Its yearly average indoor temperature lies at 23.3°C, close to the target temperature of 23.5°C with a low standard deviation of 1.2°C. Due to its high data efficiency and stability, we believe that this algorithm can reduce the gap in training RL controllers directly in the real world.

Acknowledgements

This research was supported by a Nordic5Tech PhD fellowship, the Reinforcing the Health Data Infrastructure in Mobility and Assurance through Data Democratization project (288856) funded by the Norwegian Research Council, the Heat4.0 project (8090-00046A), and by the Flexible Energy Denmark project (8090-00069B) funded by Innovation Fund Denmark.

References

[1] L. Huang, R. A. Bohne, J. Lohne, Shelter and residential building energy consumption within the 450 ppm co2eq constraints in different climate zones, Energy 90 (2015) 965 – 979. doi:10.1016/j.energy.2015.07.129.

[2] I. IEA, World energy outlook, International Energy Agency, Paris (2008).

[3] A. Afram, F. Janabi-Sharifi, Theory and applications of hvac control systems–a review of model predictive control (mpc), Building and Environment 72 (2014) 343–355. doi:10.1016/j.buildenv.2013.11.016.

[4] A. Perera, P. Kamalaruban, Applications of reinforcement learning in energy systems, Renewable and Sustainable Energy Reviews 137 (2021) 110618. doi:10.1016/j.rser.2020.110618.

[5] Z. Wang, T. Hong, Reinforcement learning for building controls: The opportunities and challenges, Applied Energy 269 (2020) 115036. doi:10.1016/j.apenergy.2020.115036.

[6] S. Liu, G. P. Henze, Experimental analysis of simulated reinforcement learning control for active and passive building thermal storage inventory: Part 2: Results and analysis, Energy and buildings 38 (2) (2006) 148–161. doi:10.1016/j.enbuild.2005.06.002.

[7] C. Zhang, S. R. Kuppannagari, R. Kannan, V. K. Prasanna, Building hvac scheduling using reinforcement learning via neural network based model approximation, in: Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation, 2019, pp. 287–296. doi:10.1145/3360322.3360861.

[8] X. Ding, W. Du, A. E. Cerpa, Mb2c: Model-based deep reinforcement learning for multi-zone building control, in: Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, 2020, pp. 50–59. doi:10.1145/3408308.3427986.

[9] D. Wölfle, A. Vishwanath, H. Schmeck, A guide for the design of benchmark environments for building energy optimization, in: Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, 2020, pp. 220–229. doi:10.1145/3408308.3427614.

[10] Y. Duan, X. Chen, R. Houthooft, J. Schulman, P. Abbeel, Benchmarking deep reinforcement learning for continuous control, in: International Conference on Machine Learning, 2016, pp. 1329–1338.

[11] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym (2016). arXiv:arXiv:1606.01540.

[12] C. Nadjahi, H. Louahlia, S. Lemasson, A review of thermal management and innovative cooling strategies for data center, Sustainable Computing: Informatics and Systems 19 (2018) 14–28. doi:10.1016/j.suscom.2018.05.002.

[13] M. C. Mozer, The neural network house: An environment hat adapts to its inhabitants, 1998.

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

[15] M. Han, R. May, X. Zhang, X. Wang, S. Pan, D. Yan, Y. Jin, L. Xu, A review of reinforcement learning methodologies for controlling occupant comfort in buildings, Sustainable Cities and Society 51 (2019) 101748. doi:10.1016/j.scs.2019.101748.

[16] G. P. Henze, J. Schoenmann, Evaluation of reinforcement learning control for thermal energy storage systems, HVAC&R Research 9 (3) (2003) 259–275. doi:10.1080/10789669.2003.10391069.

[17] S. Liu, G. P. Henze, Evaluation of reinforcement learning for optimal control of building active and passive thermal storage inventory, in: Solar Energy, ASMEDC, 2005. doi:10.1115/isec2005-76085.

[18] S. Liu, G. P. Henze, Experimental analysis of simulated reinforcement learning control for active and passive building thermal storage inventory: Part 1. theoretical foundation, Energy and Buildings 38 (2) (2006) 142–147. doi:10.1016/j.enbuild.2005.06.002.

[19] T. Moriyama, G. De Magistris, M. Tatsubori, T.-H. Pham, A. Munawar, R. Tachibana, Reinforcement learning testbed for power-consumption optimization, in: Methods and Applications for Modeling and Simulation of Complex Systems, Springer Singapore, Singapore, 2018, pp. 45–59.

[20] Z. Zhang, A. Chong, Y. Pan, C. Zhang, K. P. Lam, Whole building energy model for hvac optimal control: A practical framework based on deep reinforcement learning, Energy and Buildings 199 (2019) 472–490. doi:10.1016/j.enbuild.2019.07.029.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, nature 518 (7540) (2015) 529–533.

[22] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, nature 529 (7587) (2016) 484–489. doi:10.1038/nature16961.

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

[24] G. T. Costanzo, S. Iacovella, F. Ruelens, T. Leurs, B. J. Claessens, Experimental analysis of data-driven control for a building heating system, Sustainable Energy, Grids and Networks 6 (2016) 81–90. doi:10.1016/j.segan.2016.02.002.

[25] F. Ruelens, B. J. Claessens, S. Vandael, B. De Schutter, R. Babuška, R. Belmans, Residential demand response of thermostatically controlled loads using batch reinforcement learning, IEEE Transactions on Smart Grid 8 (5) (2016) 2149–2159. doi:10.1109/tsg.2016.2517211.

[26] T. Wei, Y. Wang, Q. Zhu, Deep reinforcement learning for building hvac control, in: Proceedings of the 54th annual design automation conference 2017, 2017, pp. 1–6. doi:10.1145/3061639.3062224.

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

[28] Y. Wang, K. Velswamy, B. Huang, A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems, Processes 5 (3) (2017) 46. doi:10.3390/pr5030046.

[29] Y. Li, Y. Wen, D. Tao, K. Guan, Transforming cooling optimization for green data center via deep reinforcement learning, IEEE transactions on cybernetics 50 (5) (2019) 2002–2013. doi:10.1109/tcyb.2019.2927410.

[30] G. Gao, J. Li, Y. Wen, Energy-efficient thermal comfort control in smart buildings via deep reinforcement learning, arXiv preprint arXiv:1901.04693 (2019).

[31] Y. Du, H. Zandi, O. Kotevska, K. Kurte, J. Munk, K. Amasyali, E. Mckee, F. Li, Intelligent multi-zone residential hvac control strategy based on deep reinforcement learning, Applied Energy 281 (2021) 116117. doi:10.1016/j.apenergy.2020.116117.

[32] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, D. Meger, Deep reinforcement learning that matters, arXiv preprint arXiv:1709.06560 (2017).

[33] A. Kathirgamanathan, K. Twardowski, E. Mangina, D. P. Finn, A centralised soft actor critic deep reinforcement learning approach to district demand side management through citylearn, in: Proceedings of the 1st International Workshop on Reinforcement Learning for Energy Management in Buildings & Cities, 2020, pp. 11–14.

[34] J. R. Vazquez-Canteli, G. Henze, Z. Nagy, Marlisa: Multi-agent reinforcement learning with iterative sequential action selection for load shaping of grid-interactive connected buildings, in: Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, 2020, pp. 170–179. doi:10.1145/3408308.3427604.

[35] S. Xu, Y. Wang, Y. Wang, Z. O'Neill, Q. Zhu, One for many: Transfer learning for building hvac control, in: Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, 2020, pp. 230–239. doi:10.1145/3408308.3427617.

[36] P. Lissa, M. Schukat, E. Barrett, Transfer learning applied to reinforcement learning-based hvac control, SN Computer Science 1 (3) (2020) 1–12.

[37] D. P. Bertsekas, Dynamic Programming and Optimal Control 3rd Edition , Volume II, Control (2010).

[38] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[39] M. L. Puterman, Markov decision processes: discrete stochastic dynamic programming, John Wiley & Sons, 2014.

[40] O. Nachum, M. Norouzi, K. Xu, D. Schuurmans, Bridging the gap between value and policy based reinforcement learning, in: Advances in Neural Information Processing Systems, 2017, pp. 2775–2785.

[41] T. Haarnoja, H. Tang, P. Abbeel, S. Levine, Reinforcement learning with deep energy-based policies, in: Proceedings of the 34th International

Conference on Machine Learning-Volume 70, 2017, pp. 1352–1361.

[42] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: Advances in neural information processing systems, 2000, pp. 1057–1063.

[43] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, 2014.

[44] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International conference on machine learning, PMLR, 2016, pp. 1928–1937.

[45] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International conference on machine learning, 2015, pp. 1889–1897.

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347 (2017).

[47] S. Fujimoto, H. van Hoof, D. Meger, Addressing function approximation error in actor-critic methods, Proceedings of Machine Learning Research 80 (2018) 1587–1596.

[48] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: International Conference on Machine Learning, 2018, pp. 1861–1870.

[49] M. Riedmiller, Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method, in: European Conference on Machine Learning, Springer, 2005, pp. 317–328.

[50] H. V. Hasselt, Double q-learning, in: Advances in neural information processing systems, 2010, pp. 2613–2621.

[51] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-Learning, in: 30th AAAI Conference on Artificial Intelligence, AAAI 2016, 2016. arXiv:1509.06461.

[52] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al., Soft actor-critic algorithms and applications, arXiv preprint arXiv:1812.05905 (2018).

[53] A. TC, et al., Data center power equipment thermal guidelines and best practices, ASHRAE TC 9.9, ASHRAE, USA (2016).

[54] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, N. Dormann, Stable baselines3, https://github.com/DLR-RM/stable-baselines3 (2019).

[55] Z. Ahmed, N. Le Roux, M. Norouzi, D. Schuurmans, Understanding the impact of entropy on policy optimization, in: International Conference on Machine Learning, PMLR, 2019, pp. 151–160.

[56] B. Chen, Z. Cai, M. Bergés, Gnu-rl: A precocial reinforcement learning solution for building hvac control using a differentiable mpc policy, in: Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, 2019, pp. 316–325. doi:10.3389/fbuil.2020.562239.

[57] W. Dabney, M. Rowland, M. Bellemare, R. Munos, Distributional reinforcement learning with quantile regression, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, 2018.

[58] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, Z. Wang, Autonomous navigation of stratospheric balloons using reinforcement learning, Nature 588 (7836) (2020) 77–82. doi:10.1038/s41586-020-2939-8.

[59] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, in: Proceedings of the International Conference on Learning Representations (ICLR), 2016.

[60] S. Kakade, J. Langford, Approximately optimal approximate reinforcement learning, in: Proceedings of the Nineteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., 2002, pp. 267–274.

Appendix A. Abbreviations

Abbreviation	Meaning
(D)RL	(Deep) Reinforcement Learning
HVAC	Heating, Ventilation, Air Conditioning
BEM	Building Energy Management
PID	Proportional, Integral, Derivative
MPC	Model Predictive Control
ML	Machine Learning
MDP	Markov Decision Process
DQN [21]	Deep Q-network
DDPG [27]	Deep Deterministic Policy Gradient
A2C/A3C[44]	(Asynchronous) Advantage Actor Critic
TD3 [47]	Twin Delayed DDPG
SAC [48]	Soft Actor Critic
TRPO[45]	Trust Region Policy Optimisation
PPO [46]	Proximal Policy Optimisation
GAE [59]	Generalised Advantage Estimation

Table A.7: Abbreviations

Appendix B. Nomenclature

Appendix C. Weather data

Appendix D. On-Policy Methods

Appendix D.0.1. Critic network

On-policy methods generally aim to learn the value function, instead of the Q-function. This allows estimates of the advantage function $A_\pi(s, a)$ to be obtained, which are needed in the policy updates. Using Generalised Advantage Estimation (GAE) [59], we can estimate $\hat{A}(s_t, a_t) \approx A_\pi(s_t, a_t)$. To do this, we generate trajectories $(s_1, a_1, \cdots, s_T, a_T)$ by running the policy π_θ in the environment, and approximate the value-function $V_\pi(s)$ using a neural network $V_w(s)$, similarly to what we did for the Q-function in the off-policy methods. Using current estimates of $V_w(s)$, we can estimate the advantage:

$$\hat{A}(s_t, a_t) = \sum_{l=0}^{T-t-1} (\lambda\gamma)^l \delta_{t+l}, \tag{D.1}$$

Notation	Meaning
s	State
a	Action
$p(s' \mid s, a)$	State-transition probability distribution
$\rho(s)$	Initial state probability distribution
$r(s, a)$	Reward function
τ	Trajectory (sequence of state-action pairs)
$\pi(a \mid s)$	Stochastic policy
$\mu(s)$	Deterministic policy
γ	Discount factor
α	Temperature parameter (in maximum entropy RL)
$J(\pi)$	Optimisation objective (total expected reward)
$V_\pi(s)$	(State-) Value function
$Q_\pi(s, a)$	Action-value function
$A_\pi(s, a)$	Advantage function
θ	Weights of the actor
w	Weights of the critic
$L(w)$, $L(\theta)$	Loss function
$y(s, a, s')$	Bellman residual
λ	Trace-decay parameter (in GAE)
D_{KL}	Kullback-Leibler divergence
$\mathcal{N}$	Gaussian (normal) distribution
$\mathcal{U}(D)$	Uniform distribution, sampled from the replay buffer
P_{it}	Power consumption of the IT equipment
P_{hvac}	Power consumption of the HVAC system
P_{tot}	$P_{\text{tot}} = P_{\text{it}} + P_{\text{hvac}}$ Total power consumption
μ	Mean
σ	Standard deviation

Table B.8: Notations

where $\lambda \in [0, 1]$ and $\delta_l = r(s_l, a_l) + \gamma V_w(s_{l+1}) - V_w(s_l)$. The critic aims to minimise the mean squared error, $L_V(w) = \frac{1}{2}\mathbb{E}_{\tau\sim p_\pi}[(V_w(s_t) - y(s_t, a_t))^2]$, where

$$y(s_t, a_t) = \sum_{l=0}^{T-t-1} \gamma^l r(s_{t+l}, a_{t+l}) + \gamma^{T-t} V_{w_{\text{old}}}(s_T) \tag{D.2}$$

Years 1-10	Years 11-20
Oslo	Hamburg
Bergen	Düssledorf
Stockholm	Bremen
Ostersund	Aberdeen
Karlstad	Saint Petersburg
Kiruna	Dundee
Göteborg	Amsterdam
Tampere	Groningen
Helsinki	Gdansk
Berlin	Szczecin

Table C.9: Weather data from towns located in northern Europe. The files are available on: `https://www.energyplus.net/weather`.

is the multi-step Bellman residual (see [38]).

Appendix D.0.2. Actor network

Although the policy gradient theorem is an important theoretical result, the gradient estimates suffer from high variance. This makes it impractical for a learning algorithm, due to noisy updates. Theoretical results [60, 45] suggest maximising another objective that is easier to analyse, which has the same gradient as $J(\pi)$ when evaluated at $\tilde{\pi} = \pi$. The idea is to maximise the surrogate objective

$$L_{\pi}(\tilde{\pi}) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho^{\pi},a\sim\pi}\left[\frac{\tilde{\pi}(a \mid s)}{\pi(a \mid s)}A_{\pi}(s,a)\right] \tag{D.3}$$

under some trust region constraints. For example, the TRPO [45] algorithm requires that the expected Kullback-Leibler divergence between the old and new policy is bounded by a constant δ. More precisely, it solves the following convex optimisation problem:

$$\theta_{\text{new}} = \max_{\theta} \nabla L_{\theta_{\text{old}}}(\theta)^{T}(\theta - \theta_{\text{old}}), \tag{D.4}$$

subject to:

$$\frac{1}{2}(\theta - \theta_{\text{old}})^{T} F(\theta, \theta_{\text{old}})(\theta - \theta_{\text{old}}) \leq \delta, \tag{D.5}$$

where

$$F_{ij}(\theta, \theta_{\text{old}}) = \frac{\partial^2}{\partial\theta_i \partial\theta_j}\mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}}\left[D_{KL}(\pi_{\theta}(\cdot \mid s), \pi_{\theta_{\text{old}}}(\cdot \mid s)\right] \tag{D.6}$$

is the Fisher information matrix and where $L_{\theta_{\text{old}}}(\theta) = L_{\pi_{\theta_{\text{old}}}}(\pi_{\theta})$ corresponds to the surrogate loss of Equation (D.3). This ensures that the data collected using the old policy is meaningful to the new policy.

Instead of solving an optimisation problem, the PPO algorithm [46] requires the importance sampling ratio to be close to 1:

$$\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{old}}(a \mid s)} \in [1-\varepsilon, 1+\varepsilon]. \tag{D.7}$$

To ensure this constraint, the PPO algorithm maximises the following objective:

$$L_\pi(\theta) = \mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}, a\sim\pi_{\theta_{\text{old}}}}\left[\min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)}\hat{A}(s,a), g(\varepsilon, \hat{A}(s,a))\right)\right], \tag{D.8}$$

where

$$g(\varepsilon, A) = \begin{cases} (1+\varepsilon)A & \text{if } A \geq 0, \\ (1-\varepsilon)A & \text{if } A < 0. \end{cases} \tag{D.9}$$

Intuitively, this means increasing the probabilities of actions leading to a higher reward and decreasing the probabilities of bad actions, while keeping updates small enough to avoid causing an accidental drop in performance.

Appendix E. Hyperparameters

Appendix E.1. SAC algorithm

Both networks are feed-forward neural networks, where between each layer, we use the Relu activation function. The input of the critics is the state-action pair (s, a) and calculates $Q_w(s, a)$. The actor uses the state and calculates the latent variables μ and σ, which are used to describe the Gaussian distribution. We also adjust the temperature parameter α automatically, as described in [52]. We use the same optimiser for the networks (two critics, one actor) and the temperature parameter. Note that the hyperparameters are identical to [52, 54].

Critic networks	$9 \to 256 \to 256 \to 1$
Actor network	$5 \to 256 \to 256 \to (2 \times 4)$
Activation function	Relu
Optimiser	Adam
Learning rate	$3 \cdot 10^{-4}$
Batch size	256
Discount factor (γ)	0.99
Polyak averaging (τ)	$5 \cdot 10^{-3}$
Buffer size	10^6
Temperature (α)	Automatically adjusted

Table E.10: SAC hyperparameters.

Appendix E.2. TD3 algorithm

Both networks are feed-forward networks, where between each layer, we use the Relu activation function. The architectures are analogous to SAC with the difference that the actor directly calculates the action. The policy uses delayed updates, as described in [47] for reduced training time. Contrary to the recommended parameters, we did not use exploration noise, as it reduces the stability of the training process. The other hyperparameters are identical to the original implementation [47].

Critic networks	$9 \rightarrow 400 \rightarrow 300 \rightarrow 1$
Actor network	$5 \rightarrow 400 \rightarrow 300 \rightarrow 4$
Activation function	Relu
Optimiser	Adam
Learning rate	10^{-3}
Batch size	100
Discount factor (γ)	0.99
Polyak averaging (τ)	$5 \cdot 10^{-3}$
Buffer size	10^6
Target policy noise	clip($\mathcal{N}(0, 0.2)$, -0.5, 0.5)
Exploration noise	None
Delayed policy update	2

Table E.11: TD3 hyperparameters.

Appendix E.3. PPO algorithm

The networks are similar to SAC, except smaller. We modified the parameters of the Stable Baselines [54] implementation, by using a horizon of 4096 instead of 2048 and updating the networks for 15 epochs instead of 10. The number of epochs tells how often we reuse the sampled data in order to update the networks, before collecting new trajectories. The gradient clipping means they minimise the Huber loss, instead of the mean squared error. Other optimisations, such as entropy regularisation (as in SAC) and early stopping were not used.

Critic network	$5 \to 64 \to 64 \to 1$
Actor network	$5 \to 64 \to 64 \to (2 \times 4)$
Activation function	Tanh
Optimiser	Adam
Learning rate	$3 \cdot 10^{-4}$
Batch size	64
Discount factor (γ)	0.99
Trace-decay parameter (λ)	0.95
Horizon (T)	4096
Number of epochs	15
Clipping range (ε)	0.2
Global gradient clipping	0.5
Entropy regularisation	None
Target KL early stopping	None

Table E.12: PPO hyperparameters.

Appendix E.4. TRPO algorithm

We use the same hyperparameters as in [19]. They are identical to the Baselines implementation with the exception of the horizon, which is larger in order to increase stability in the learning process. The neural networks are similar to PPO, but smaller. The value function is learned the same way as for PPO. However, the policy weights are updated differently. TRPO solves each iteration a convex optimisation problem. This involves solving a linear equation with the conjugate gradient algorithm. We refer the reader to [45] for more details.

Critic network	$5 \to 32 \to 32 \to 1$
Actor network	$5 \to 32 \to 32 \to (2 \times 4)$
Activation function	Tanh
Value function optimiser	Adam
Learning rate	10^{-3}
Discount factor (γ)	0.99
Trace-decay parameter (λ)	0.98
Horizon (T)	16348
Maximum KL-divergence	0.01
Conjugate gradient iterations	10
Conjugate gradient dumping	0.01
Number of epochs (critic updates)	5
Entropy regularisation	None

Table E.13: TRPO hyperparameters.