POLITECNICO DI TORINO

Master of Science in Energy and Nuclear Engineering

Master Thesis

Data-driven coordinated building cluster energy management to enhance energy efficiency, comfort and grid stability

Supervisors

Prof. Alfonso CAPOZZOLI

Dott. Giuseppe PINTO

Dott. Silvio BRANDI

Candidate

Davide DELTETTO

Academic Year 2019/2020

Acknowledgements

To my family,
which allowed me to achieve this result

Abstract

The environmental constraints related to emissions reduction and to tackle climate change require an increasing penetration of renewable energy generation. Since renewable energy sources like solar and wind are not programmable and often unpredictable, grid balancing is becoming more and more challenging. In this scenario, buildings energy flexibility could play a key role through demand side management/load control and demand response programs. However, when switching from a single building to a cluster of buildings, an uncoordinated energy management could generate new undesirable power peaks. The study presented in this thesis work aims to explore a coordinated approach between four commercial buildings, in order to reduce the impact that their energy consumption has on the electrical grid. Moreover, the proposed methodology shows the possibility and the feasibility to develop a fully data-driven control scheme able to manage a cluster made by four different buildings, characterized by different occupancy profiles and internal loads. This goal is achieved by taking advantage of machine learning techniques, in particular Long Short Term Memory (LSTM) Neural Network and Deep Reinforcement Learning (DRL), in order to control the HVAC cooling power and the operation of chilled water and DHW storages.

The first part of the thesis is related to the study of black box modelling of building indoor temperature evolution through artificial neural networks. These data-driven techniques have gained popularity in last years, also in the building modelling field, thanks to their ability to deal with nonlinearities. A clear advantage with respect to physics-based models, is related to the lower computational cost, which made it suitable for real time control techniques. The study is specifically focused on the development of four LSTM models, which are a particular class of Recurrent Neural Network especially used in the field of time series forecasting, in order to predict the mean indoor temperature of considered buildings.

In the second part of the work, the developed models are then integrated into CityLearn. CityLearn is an OpenAI Gym environment for the implementation of reinforcement learning agents in a multi-agent demand response setting with the aim of reshaping the aggregated curve of electrical demand of the considered buildings. Reinforcement Learning is an adaptive model-free control algorithm, which is able to learn an optimal control policy by trial and error. Through the interaction with the LSTM models, the centralized agent learns the optimal control policy, taking into account both thermal comfort and energy consumption. The main contribution of the work lies in the fact that the agent has the possibility to

control both the HVAC cooling power and the storage operation of each building. The developed controller was trained for 20 epochs and tested against a manually optimized Rule Based Controller (RBC). The results shown that the DRL algorithm was able to slightly reduce the overall electricity cost of the cluster of buildings while decreasing the peak power consumption of 10.7%, without excessively penalize thermal comfort.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview of Buildings Energy Consumption

According to [1], the building sector is responsible for approximately 32.2% of total energy consumption in the world.

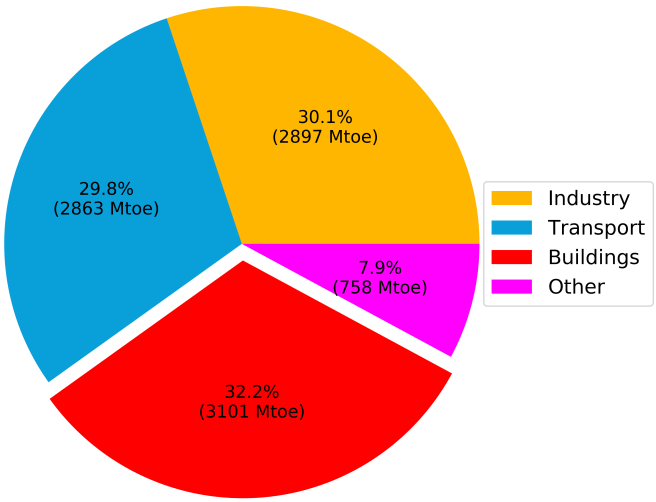

Figure 1.1: Energy consumption by sector, 2018

Direct and indirect emissions from electricity and commercial heat used in buildings rose to 10 GtCO2 in 2019, the highest level ever recorded. Buildings account for about 28% of total energy-related CO2 emissions, two-thirds of which is attributable to emissions from electricity generation for use in buildings, since they are responsible for more than 55% of global electricity consumption [2]. Since buildings sector

plays a critical role in global energy consumption and emissions, there is a great opportunity in reducing its impact by improving efficiency, both on generation and use sides. Following the *Faster Transition Scenario* [3], it will be possible to drop the energy intensity of the power sector of 90% and that of end use sectors of 65% thanks to energy efficiency, uptake of renewable energy technologies and shifts to low-carbon electricity. An important aspect that would lead to the reduction of CO2 emissions is the electrification of final uses. Electricity's share in final energy consumption will reach about 35% by 2050, compared to less than 20% today. This is due to the necessity of reducing fossil fuels consumption in the field of buildings by replacing traditional boilers with Heat Pumps: by using these device it is possible to exploit on site renewable energy generation and greatly reduce CO2 emissions from buildings.

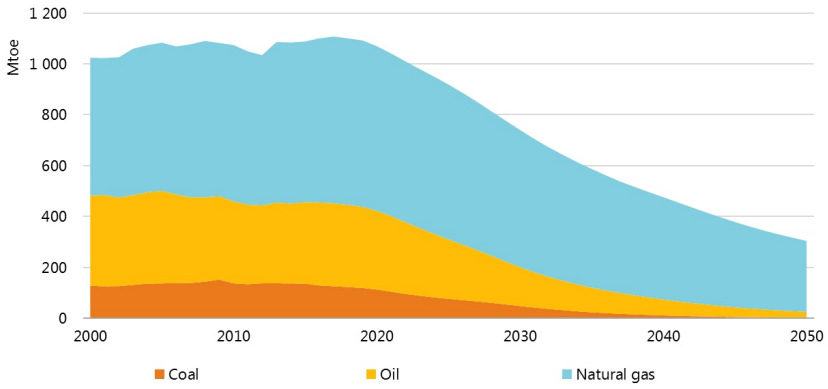

Figure 1.2: Evolution of fossil fuel use in buildings in the Faster Transition Scenario, 2017-50 [2]

To make this transition sustainable, both renewable energy generation and energy efficiency must increase: high performance buildings equipped with efficient heat pumps, air conditioners, lighting and appliances would drastically reduce sector's energy use by nearly 30% to 2050, despite an expected global doubling of the floor area. The electrification of end uses (such as space heating and cooling) would lead to a growth in electricity demand in the buildings sector: in the *Faster Transition Scenario* it is expected a further increase of the equivalent of more than one-fourth of global electricity demand in 2017 by 2050. Even with energy efficiency measures, the increased demand would place pressure on the power system. For example, global electricity demand for space cooling is expected to increase by 35%: in many countries, this will stress peak electricity loads if uncontrolled, even if using efficient air conditioners. By contrast, high performance equipments coupled with demand-side measures can reduce the impact of electrification, while supporting power system flexibility and higher penetration of variable renewables in the electricity mix.

1.2 Energy Flexibility of Buildings

According to [4],[5] the Energy Flexibility of a building is the ability to manage its energy demand and generation according to local climate conditions, user needs and grid requirements. Energy Flexibility of buildings will thus allow for demand side management and load control and thereby demand response based on the requirements of the surrounding grids. Demand side flexibility (DSF), in particular buildings energy flexibility, has become more relevant in recent years, as an alternative to supply side flexibility.

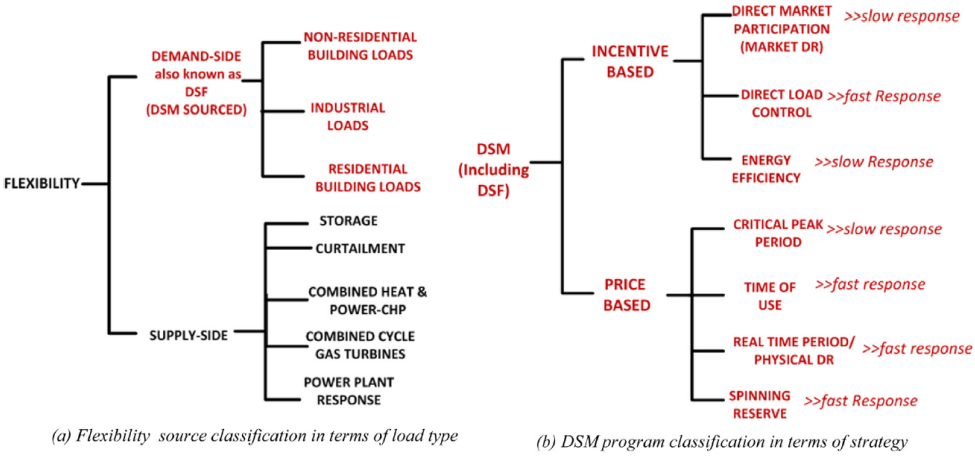

Figure 1.3: Flexibility sources in modern electricity power networks [6]

The reasons behind the increasing interest in DSF is mainly due to the high cost of operating and maintaining flexibility sources on supply-side. Moreover, through demand side flexibility it is possible to reduce thermal costs and electricity prices since less peaking plants would need to operate. Another key aspect is related to the reduction of investments in generation, transmission and distribution facilities, since they are designed according to the peak demand.
Energy flexibility of a building is generally achieved by decoupling energy demand and energy delivery by means of storage that allows to shift energy use from period of high electricity price (e.g. periods in which the production from RES is low) to period of low electricity price. There are different ways to achieve flexibility, those of particular interest are:

- Thermal storage, both cold and hot storage filled with chilled water (or ice) or hot water, respectively; also PCM could be used to store thermal energy .

- Batteries, they are used to directly store electricity. They are charged during periods with surplus of electricity in the grid and thereby cheap, and discharged during periods when there is a shortage of electricity or otherwise beneficial for the grid. They could also be used to increase the self consumption percentage of PV electricity. Batteries could either be the battery of an electrical vehicle or the battery of e.g. a PV system.

- Building thermal mass: walls, floors, ceilings and furniture of buildings contain a certain amount of mass and thereby a certain thermal capacity, which can be utilized to store energy. During shortage of energy the heating or cooling system could either be switched off or could reduce its power consumption for a period without decreasing the comfort of the users. How long a period depends on the thermophysical properties and the heat loss of the buildings, but can be from a few hours up to a couple of days. However, care should be taken, as the storage is directly connected to the indoor climate and the thermal comfort must not be jeopardized.

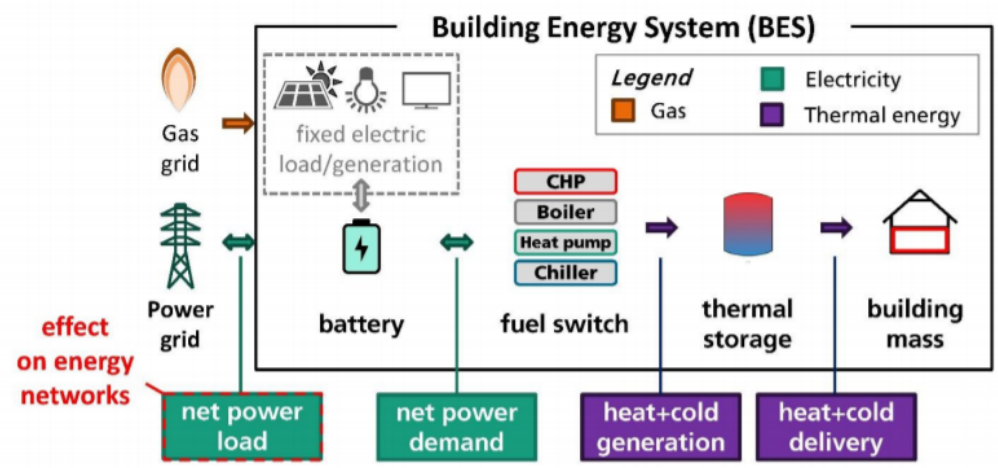

Figure 1.4: Sources for obtaining buildings energy flexibility [7]

To efficiently exploit energy systems, different control techniques have been used. Rule based control (RBC) strategies are a common approach for controlling energy systems of buildings. They use pre-defined conditions (or decision rules) to change the current state of a system. Their first goal is to maintain comfort conditions during occupied hours while consuming less energy as possible. Moreover, advance RBCs, depending on the decision criteria (e.g. weather, electricity price, occupancy, PV production), can aim at activating the energy flexibility of the building to improve grid interaction, lower energy costs, perform load shifting or reduce energy needs by varying the temperature set points of the buildings zones or the water storage tanks. Classical control strategies, such as thermostatic on/off control, P, PI or PID (Proportional-Integrative-Derivative) control belongs to rule based controls field and are the state-of-the-art for HVAC applications. The On/ Off controller uses an upper and lower threshold to regulate the process within the given bounds. The P, PI, and PID controllers use error dynamics and modulate the controlled variable to achieve accurate control of the process. Although the on/off controller is the most intuitive and easiest to implement, it is unable to control moving processes with time delays. Because of the high thermal inertia of many HVAC processes, a process that is controlled using an on/off controller displays large swings from the set points. The PID controller produces promising results, but tuning the controller parameters is time consuming, and the performance of

the controller degrades if the operating conditions vary from the tuning conditions. For these reasons, these control techniques are not suited to adapt to time-varying disturbances or changes in environmental conditions [8] and thus may fail to provide flexibility in a dynamic manner.
To overcome previous limitations, advanced control strategies were investigated. In particular, Model Predictive Control (MPC) is a control strategy that enables the flexibility of the HVAC system operation and allows the optimization of the energy consumption while preserving, or even improving, thermal comfort. MPC has a great potential for deploying demand side flexibility because it can deal with time-varying operating conditions and can interact with the energy system and the grid. It is seen as one of the most promising developments as it can take into account future weather, electricity price forecasts as well as occupant behavior when computing an optimal consumption decision. MPC is based on models that, depending on their structure, could be classified in:

- Engineering or white-box models: they are based on physical laws, in particular on principles of thermodynamics; they do not require any measured data but, on the other hand, they necessitate of a significant amount of information about the interactions between the system and the external environment.

- Inverse or data-driven models: these models are used to describe system behaviour through the use of real measured data: the system must be already in operation. These methods are further divided into:

  - Grey-box models which combine a partial theoretical (physical) structure with data to complete the model.
  - Black-box models: they include all the algorithms which identify a model which links the output with a number of inputs. They have the peculiarity that their coefficients cannot be correlated with any physical meaning. They could approximate any kind of function but to do this they need to be trained on large data sets. Artificial Neural Networks belongs to this category of models.

MPC has been extensively researched in the past 10-15 years and it has showed very good performances if well designed. However, it is expensive to implement because it requires a model of the system that has to be controlled. Of course, its performances are strongly dependent on the quality of the model. Moreover, they lack of generalization, since it is necessary to build a model for each plant and building to control. Another drawback of MPC is that it is not adaptive, if something changes in the system it would not be able to adapt to the change automatically, without re-design the model.
For these reasons, research is trying to develop more efficient methods characterized by the absence of a model; in particular, great interest is devoted to model-free control, such as Reinforcement Learning (RL). Reinforcement learning is less expensive to implement because it does not require a model of the system and could learn both through the interaction with the environment and through historical data. It is similar to a plug and play controller since it could be implemented and it could

automatically learn from the system through interactions. Its performance improve in time as the controller learns the optimal control policy by direct interaction with the environment, obtaining a reward based on the control action performed from a specific state. Another very interesting aspect of Reinforcement Learning is that it is adaptive, it can automatically adapt to the environment's changes as well as to human preferences by directly integrating user feedback into its control logic.
In this context, an interesting framework is represented by CityLearn: it is an OpenAI Gym environment for the implementation of reinforcement learning agents in a demand response settings, with the objective of facilitating and standardizing the evaluation and the comparison of different RL agents and algorithms. The final goal of CityLearn is to flatten, smooth, and reduce the curve of electrical demand through Demand response (DR). CityLearn allows to control the storage of domestic hot water (DHW), and chilled water. It also includes energy models of air-to-water heat pumps, electric heaters, and the pre-computed energy loads of the buildings, which include space cooling, dehumidification, appliances, DHW, and solar generation.

1.3 Contributions and structure of the thesis

The objective of this thesis work is to develop a framework which allows the application of advanced control techniques (RL) to several commercial building HVAC systems. The main goal of the controller is to reduce the impact of its actions on the electrical grid (through DR) while maintaining user comfort. Through the development of an LSTM neural network it has been possible to map the relation between uncontrollable variables (such the external Temperature, the occupancy, the solar radiation), controllable variables (the HVAC cooling power) and the Indoor Temperature, allowing the possibility to consider thermal comfort. Successively, the model has been integrated into CityLearn, in order to create a training environment for the RL agent. This environment would enable the Reinforcement Learning agent to learn the optimal control policy and exploit Demand Response actions with the aim of reshaping and flattening the curve of electricity demand. The main contribution is that the building cooling load is not an input of CityLearn but is learned from the agent through the interaction with the neural network models: in this way the agent could exploit the possibility of giving buildings a cooling energy different from the ideal one, as an additional resource of DR, without violating thermal comfort constraints.
The thesis is organized as follow:

- **Chapter 2** deals with the description of machine learning techniques used in this thesis work; in particular, for introductory purposes, the description starts from simpler algorithms, such as the perceptron and and logistic regression. More complex algorithms, like feed forward neural networks and recurrent neural networks are subsequently addressed: the problem of exploding/vanishing gradient is used to introduce the description of Long Short Term Memory neural networks, which are one of the key algorithms used in this work.

- **Chapter 3** describes the approach used to model the buildings thermodynamic behaviour: the chapter starts with the literature review of previous works in the field of indoor temperature prediction through black box models. Moreover, the buildings analyzed in this work are briefly described and particular attention is devoted to the description of the data-set generation process through EnergyPlus software. The second part of the chapter is devoted to the analysis of the LSTM models used in this work: the focus is related to the data preparation and pre-process work, followed by the analysis of models architecture and hyperparameters selection and tuning. The last part of the chapter is related to the analysis of the performances of the models, both in open loop and in closed loop. The last part of the chapter concerns CityLearn environment and its related work, in order to highlight the contributions of this thesis work.
- **Chapter 4** analyzes the Reinforcement Learning algorithm; a brief description of one of the most used algorithm in this field, Q-learning, is followed by the analysis of two other algorithms: the first one is Deep Q-learning, the second one is Soft Actor Critic algorithm, which is implemented in the case study.
- **Chapter 5** describes the case study: the process of integration of the developed models into CityLearn is analyzed; particular attention is paid to the reward function design.
- **Chapter 6** deals with the analysis of the obtained results.
- **Chapter 7** is about the conclusions and the possible future works.

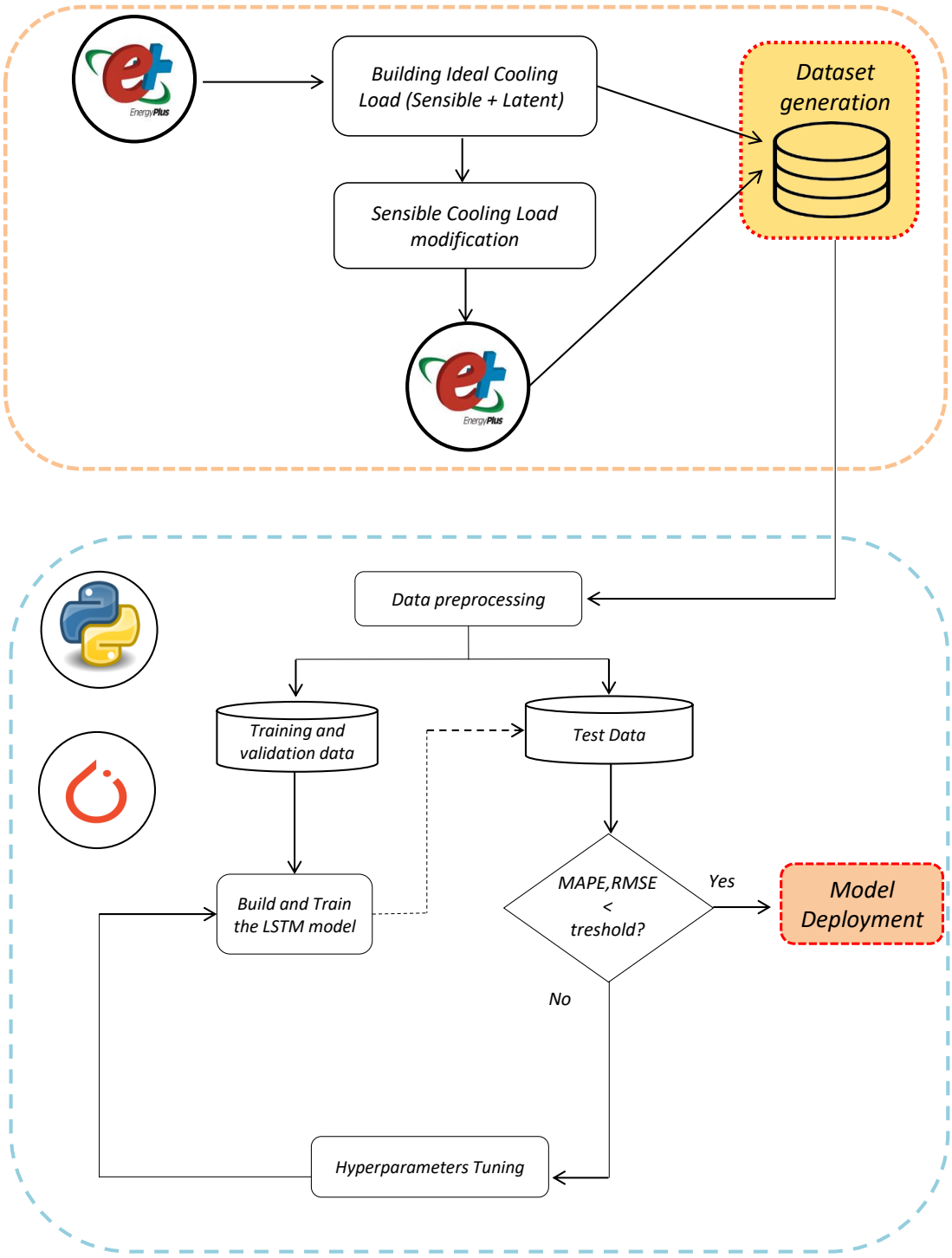

Figure 1.5: Workflow of LSTM models development

.

Chapter 2

Machine Learning overview and Neural Networks

Machine learning (ML) is a field of computer science which aim is teaching computers how to learn and act without being explicitly programmed. More specifically, machine learning is an approach to data analysis that involves building and adapting models, which allow programs to "learn" through experience. Machine learning involves the construction of algorithms that adapt their models to improve their ability to make predictions [9]. The main difference between traditional algorithms and ML algorithms is that the former require input data and a program to provide an output while the latter use both input and output data to provide a program as output.

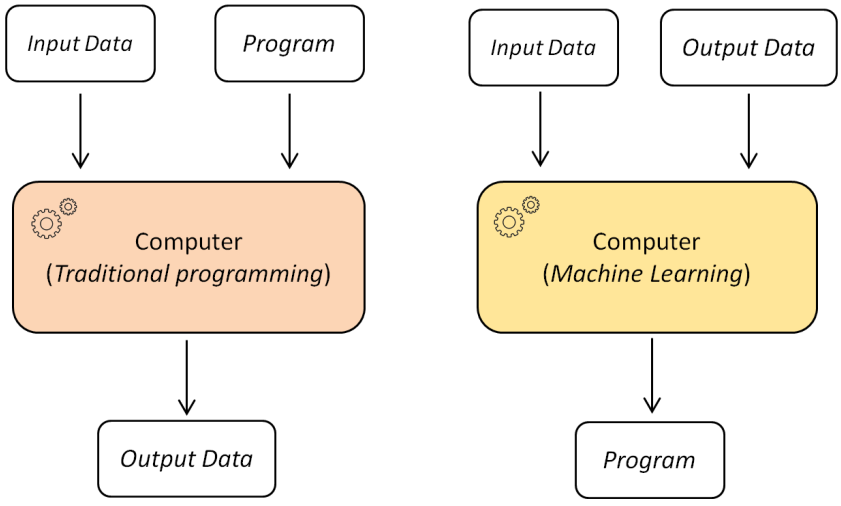

Figure 2.1: Paradigm shift from traditional algorithms to Machine Learning algorithms

Traditionally, machine learning algorithms could be divided in three categories:

- Supervised Learning: is the field of ML generally used to perform regression or

classification tasks, it requires both input and output data to train the model (labelled data set). The comparison between the real output data and the predictions is used to compute an error which is needed to adjust the weights of the model.

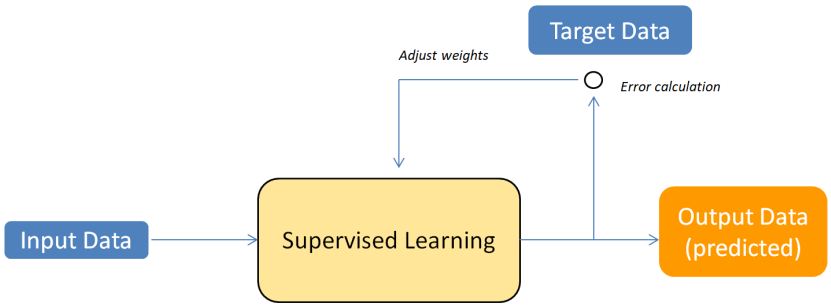

Figure 2.2: Supervised Learning

- Unsupervised Learning: unlike the previous one, the outputs are not provided, there are only input variables. This ML technique is used to find patterns in the input data through mathematical tools like *Euclidean Distances*. The data are not labeled, algorithms must therefore be formulated such that they can find suitable patterns and structures in the data on their own. Some example tasks are clustering (grouping a set of objects in such a way that objects in the same group, called a cluster, are more similar to each other than to those in other clusters) and dimensionality reduction.

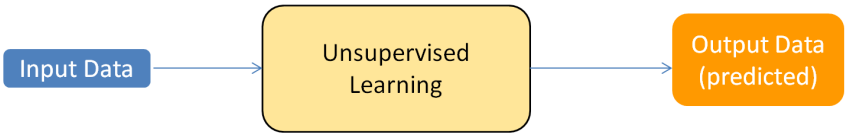

Figure 2.3: Unsupervised Learning

- Reinforcement Learning: it is completely different from the other two techniques, it is about controlling a system through the long term maximization of a reward. According to [10], a reinforcement Learning problem consists of an agent that exists within an environment where, on each time step, it can observe a current state, take actions, and as consequence of those actions, receives a reward signal and updates the state. The Agent's learning objective is to create a Policy (a mapping from State to Action) that maximises the expected reward received over time. The output of RL are the actions that the agent can take in order to modify the behaviour of the controlled system.

For more information and a detailed description of RL, see chapter 4.

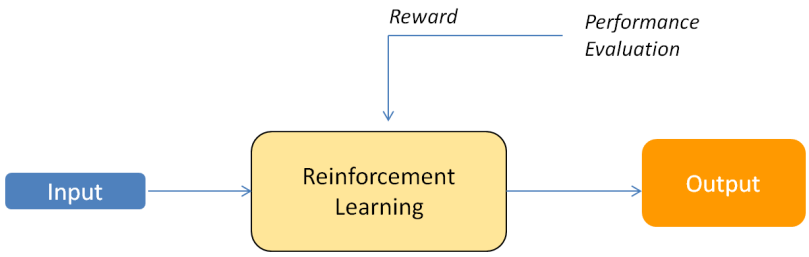

Figure 2.4: Reinforcement Learning

2.1 Single Computational Layer

2.1.1 Perceptron

An artificial neural network (ANN) is a class of algorithms designed to simulate the way the human brain analyzes and processes information. The simplest neural network is referred to as the perceptron. This neural network contains a single input layer and an output node. Its structure is made by a single input layer which transmits the input features to the output node. The input layer contains n features and it does not perform any computation; on the other hand, the output node compute a linear function $\vec{w} \cdot \vec{x}$, then add a bias b and apply the sign function:

$$\hat{y} = sign\{\sum_{i=0}^{n} w_i x_i + b\}$$

The sign function maps a real value to either -1 or $+1$, which is appropriate for binary classification. The sign function serves the role of an *activation function*. Different choices of activation functions can be used to simulate different types of models used in machine learning, like least-squares regression with numeric targets, the support vector machine, or a logistic regression classifier.
In order to have a model which guarantees accurate predictions, a training process is necessary. Training is in fact an iterative process with the aim of minimizing the error of predictions. Generally this error is calculated on the basis of the difference between the prediction and the true output. In particular, weights and biases can be updated through the resolution of an optimization problem. At the time that the perceptron algorithm was proposed by Rosenblatt in 1957 [11], these optimizations were performed in a heuristic way with actual hardware circuits: the original *Mark I perceptron* in fact, was intended to be a machine rather than an algorithm, and custom-built hardware was used to create it. The perceptron algorithm was, therefore, heuristically designed to minimize the number of misclassifications and

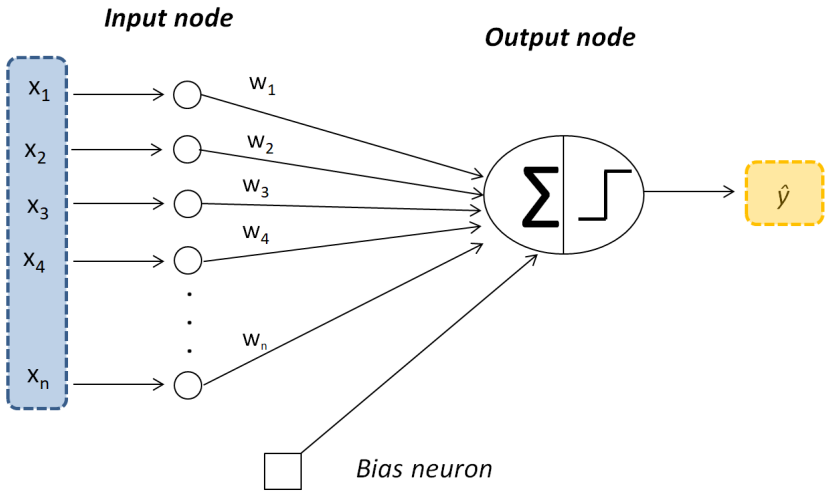

Figure 2.5: The basic architecture of the Perceptron

it provided correct results in simplified settings such as classification tasks for linearly separable classes. The aim of the perceptron was to minimize the following objective function:

$$L = \sum_{(\vec{x},y)\in D} (y - \hat{y})^2$$

The function to be minimized is known as *Loss function*. This loss function is similar to least squares regression, however, the latter is defined for continuous-valued target variables, and the corresponding loss is a smooth and continuous function of the variables. On the other hand, for the least-squares form of the objective function, the sign function is nondifferentiable, with step-like jumps at specific points. Furthermore, the sign function takes on constant values over large portions of the domain, and therefore the exact gradient takes on zero values at differentiable points. This results in a staircase-like loss surface, which is not suitable for gradient-descent. The perceptron algorithm (implicitly) uses a smooth approximation of the gradient of this objective function with respect to each example:

$$\nabla L \approx \sum_{(\vec{x},y)\in D} (y - \hat{y})\vec{x}$$

It is important to highlight that the above gradient is not the true gradient of the staircase-like surface of the objective function, but it is an approximation. Therefore, the staircase is smoothed out into a sloping surface defined by the *perceptron criterion*: this concept was proposed later than the original paper by Rosenblatt [11], in order to explain the heuristic gradient-descent steps. The general goal was to minimize the number of classification errors with a heuristic update process (in hardware) that changed weights in the "correct" direction whenever errors were made. Considering a single training example, weights were updated as follow:

$$\vec{w} =\leftarrow \vec{w} + \alpha(y - \hat{y})\vec{x}$$

The parameter α regulates the *learning rate* of the neural network.
This heuristic update strongly resembled gradient descent but it was not derived as a gradient-descent method. Gradient descent is defined only for smooth loss functions in algorithmic settings.

2.1.2 Logistic regression

Logistic regression is a probabilistic model that classifies the instances in terms of probabilities. It is used for binary classification tasks and it could be mathematically described as:

$$\hat{y} = P(y = 1 \mid x)$$

The output of logistic regression is obtained by applying the sigmoid function to the results of the sum between the bias and the scalar product between the weights vector and the input feature vector:

$$\hat{y} = \sigma(\omega^T x + b)$$

The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where z is equal to $\omega^T x + b$.

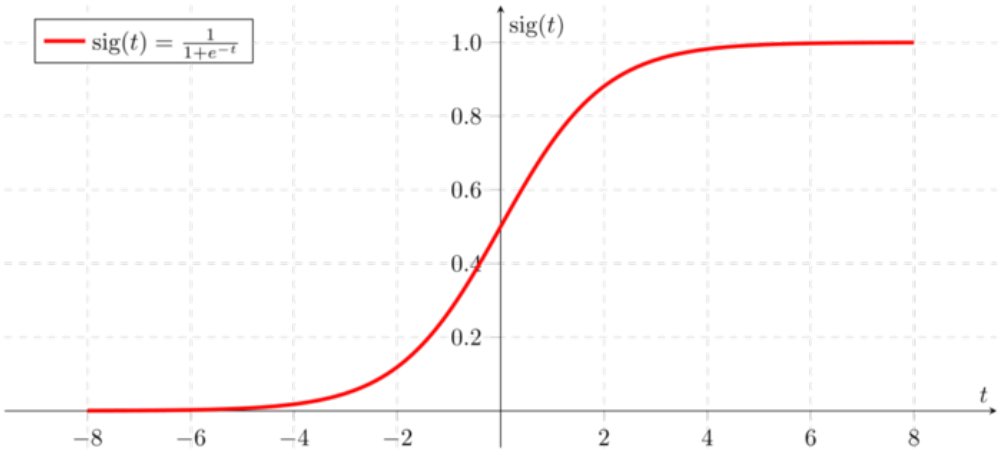

Figure 2.6: Sigmoid Activation Function

A common loss function used in machine learning field is the squared error, however, in logistic regression, this function is not used, because it generates a non-convex optimization problem, with a lot of local minima, which could cause the failure of gradient descent algorithm. For this reason, a common loss function used in logistic regression is defined as follow:

$$\mathscr{L}(\hat{y}, y) = -\Big(y\ log(\hat{y}) + (1 - y)\ log(1 - y)\Big)$$

This loss function is called *Cross-Entropy Loss*. It makes the optimization problem convex and is suitable for gradient descent algorithm.
Generally, the loss function is defined with respect to a single training example and describes how the model performs on a single training examples. When more examples are considered, a new function is introduced, which is called *Cost Function J*. In this case, the cost function could be described as:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathscr{L}(\hat{y},y) = -\frac{1}{m}\sum_{i=1}^{m}\Big[y\ log(\hat{y}) + (1-y)\ log(1-y)\Big]$$

The cost function measures how the model is performing on the entire training data set (or on more than one training example).

2.1.3 Logistic regression Gradient Descent

In order to learn optimal parameters w and b, it is necessary to find w and b which minimize the overall cost function J. In this case, the convex cost function make the optimization problem easier to solve.

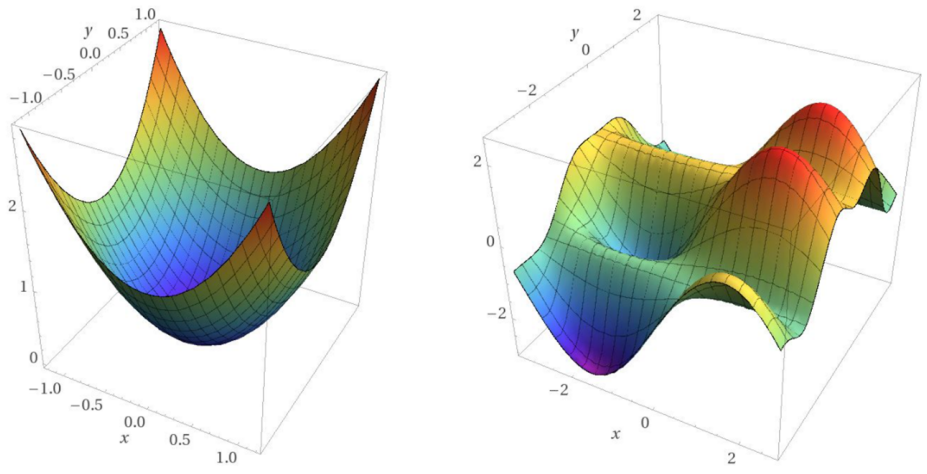

Figure 2.7: *(a)* Convex Cost Function *(b)* Non-Convex Cost Function considering only two parameters

When only two parameters are considered, J is a surface, however, since in machine learning problems there are a lot of parameters to optimize, the cost function is not a surface anymore, but is defined in multiple dimensions. Initially, model parameters are initialized either randomly or to zero (in logistic regression, since the cost function is convex, the algorithm would converge to the same value), then what gradient descent does is to take a step in the steepest downhill direction; this step is repeated until the global optimum is reached. Model parameters are then updated as follow:

$$w = w + \alpha \frac{\partial J(w,b)}{\partial w}$$
$$b = b + \alpha \frac{\partial J(w,b)}{\partial b}$$

When only one training example is considered, the partial derivative of the loss function is equal to the partial derivative of the cost function; on the other hand, considering m examples, the derivative of the cost function are computed as:

$$\frac{\partial\, J(w,b)}{\partial\, w} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial\, \mathscr{L}(\hat{y},y)}{\partial\, w}$$

$$\frac{\partial\, J(w,b)}{\partial\, b} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial\, \mathscr{L}(\hat{y},y)}{\partial\, b}$$

2.1.4 Deriving the formula for Gradient Descent

By looking at the following computation graph, it is possible to better understand the derivation of the formula for logistic regression gradient descent:

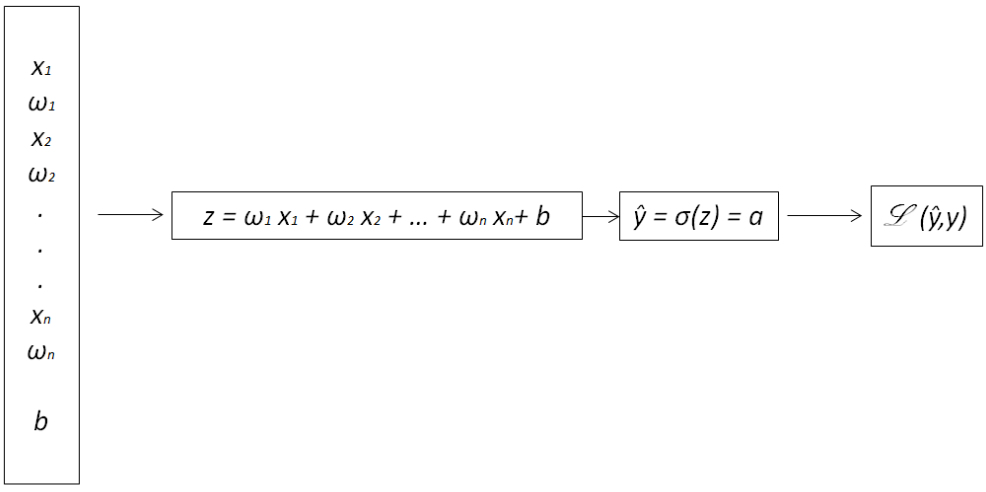

Figure 2.8: Logistic Regression computation graph

To compute the partial derivative of the loss function with respect to the weights and bias, it is possible to exploit the *chain rule*:

$$\frac{\partial \mathscr{L}}{\partial w_i} = \frac{\partial \mathscr{L}}{\partial a}\,\frac{\partial a}{\partial z}\,\frac{\partial z}{\partial w_i}$$

where:

$$\frac{\partial z}{\partial w_i} = x_i$$

$$\frac{\partial \mathscr{L}}{\partial a} = -\frac{y}{a}\,\frac{1-y}{1-a}$$

$$\frac{\partial a}{\partial z} = a(1-a)$$

Finally, the partial derivative of the loss function with respect to w_i is

$$\frac{\partial \mathscr{L}}{\partial w_i} = (a-y)\; x_i$$

and w_i are updated as follow:

$$w_i = w_i - \alpha\;(a-y)\; x_i$$

Similarly, considering the bias b, it is updated in this way:

$$b = b - \alpha\;(a-y)$$

The absence of x_i is due to the fact that the partial derivative of z with respect to b is equal to 1.

2.2 Neural Networks

Artificial neural networks are popular machine learning techniques that emulate the mechanism of learning in biological organisms. The human nervous system contains cells called neurons. Neurons are connected through axons and dendrites, and their connections are known as synapses. The strength of synaptic connections changes in response to external stimuli. This biological mechanism is emulated in artificial neural networks, which contain computation units that are referred to as *neurons*. The computational units are connected to one another through weights, which serve the same role as the strengths of synaptic connections in biological organisms. Each input to a neuron is scaled with a weight, which affects the function computed at that unit.
In recent years they have become increasingly popular due to the fact that the computing power needed to manage them is now available. Neural networks,in fact, were already mathematically treated decades ago, however, due to the scarcity of computational resources available in the nineteenth century they were never used.

2.2.1 Feed Forward Neural Networks

Neural networks contain more than one computational layer, previous examples contain only an input and an output layer: the input layer transmits the data to the output layer, and all computations are completely visible to the user. Neural networks contain multiple computational layers; the additional intermediate layers (between input and output) are referred to as *hidden layers* because the computations performed are not visible to the user. The specific architecture of multilayer neural networks is referred to as *feed forward* networks, because successive layers feed into one another in the forward direction from input to output. The default architecture of feed-forward networks assumes that all nodes in one layer are connected to those of the next layer. Therefore, the architecture of the neural network is almost fully defined, once the number of layers, the number of neurons per layer and the activation functions in each layer have been defined. The only remaining detail is the loss function that is optimized in the output layer. Neural networks with only one hidden layer are referred as *shallow networks* while models with more than one hidden layer are called *Deep Neural Networks*; over the last years, the machine learning community realized that there are functions that very deep neural networks can learn that shallower models are often unable to. However, the number of hidden layer could not be predicted in advance so, when constructing a models, it is always reasonable to start from shallower networks and then add hidden layers one by one. The architecture of a Feed Forward neural network with only one hidden layer is represented in the following picture:

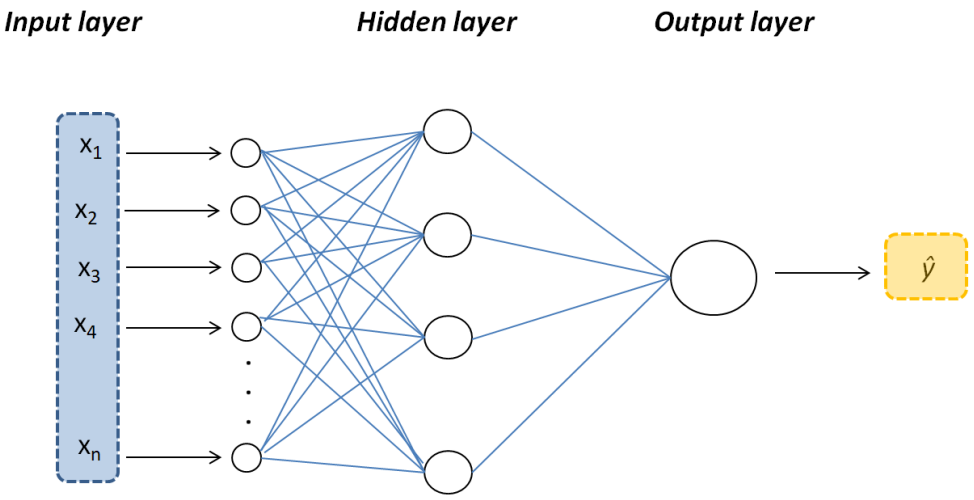

Figure 2.9: Feed Forward Neural Network with only one hidden layer

After having fixed the structure of the network, the learning process of a neural network consist in finding the optimal parameters that weigh the various signals carried by the network, from input layer to output layer. During the training process, examples are processed by the network in order to produce output values; on the basis of these outputs, error are computed and used to adjust weights and biases: this iterative mechanism is called *Forward Progagation* and *Backpropagation*,

respectively. The user of neural network libraries does not have to worry about implementing backpropagation algorithm. Its task is to select the appropriate network structure in terms of *hyperparameters*, which is a delicate and not easy task. With the term hyperparameters are identified all the parameters which control the ultimate parameter, w and b. in Feed forward neural networks, hyperparameters are:

- the learning rate α: it controls w and b with respect to the gradient of the loss function. It defines how quickly the neural network updates the concepts it has learned. A desirable learning rate is low enough that the network possibly converges to the global minimum, but high enough that it can be trained in a reasonable amount of time. Smaller learning rates require more training epochs (requires more time to train) due to the smaller changes made to the weights in each update, whereas larger learning rates result in rapid changes and require fewer training epochs. However, larger learning rates often result in a sub-optimal final set of weights.

- the number of epochs which describes the number of times the algorithm sees the entire data set.

- the number of hidden layers

- the number of neurons in hidden and output layers

- the activation function which defines the output of neurons. There are many kind of activation functions which are suitable for different applications. The most relevant are:

  – Sigmoid Function

  $$f(x) = \frac{1}{1 + e^{-x}}$$

  it has a characteristic "S"-shaped curve and it gives outputs between 0 and 1. The problem associated with this function is that, for large input values, both negative and positive, it saturates at 0 or 1: this generates a gradient very close to zero, which makes the backpropagation algorithm almost ineffective. Therefore, sigmoid activation function is very prone to vanishing gradient problem.

  – Hyperbolic Tangent Function

  $$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

  generally it works better than the sigmoid function, outputs are between -1 and 1 and their mean is 0, this tends to make each layer output more or less centered around 0. Despite a bigger derivative around zero with respect to the sigmoid function, is also suffers from vanishing gradients problem.

– Rectified Linear Unit ReLU Function

$$f(x) = max(0, x)$$

its derivative is equal to 1 for $x > 0$ and equal to 0 for $x < 0$, it is not differentiable in $x = 0$ but this has very few implications in machine learning, since the chances that the input of the function (which is the scalar product of weights and inputs) is equal to zero are almost impossible. There are some advantages with respect to the previous functions, in particular it is faster to compute and does not saturate for large positive input values. Further, the derivative or gradient of ReLU has a constant value when $x > 0$ and this results in reduced likelihood of vanishing gradient problem. One of the main drawbacks of ReLU is known as dying ReLUs that happen when some neurons stop outputting anything other than 0. This happens when a neuron's weights get updated such that weighted sum of neuron's input is negative.

– Leaky ReLU

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha \, \mathrm{x} & x < 0 \end{cases}$$

it is defined to reduce the problem of dying neurons through the introduction of a linear term with slope α for $x < 0$. α is generally equal to 0.01. This small slope ensures that leaky ReLUs never die.

– Exponential Linear Unit ELU

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha \, (\mathrm{e}^{-x} - 1) & x < 0 \end{cases}$$

it takes on negative values when x < 0 and this allows neurons average output to be closer to 0. The hyper-parameter α defines the value that the ELU function approaches when x is a large negative number. It is usually set to 1, but can be tuned as any other hyper-parameter. Further, ELU activation function has non-zero gradient for x < 0 and this solves the dying neurons problem. Moreover, the function is smooth everywhere, including around x=0. This helps to speed up the learning process. The major drawback of ELU is that it is slower to compute as compare to ReLU and its variants. This is due to the use of computationally expensive exponential function in ELU. However, during training this is compensated by faster convergence rate. But during test time an ELU network will be slower than the ReLU neural network.

– Scaled Exponential Linear Unit SELU

$$f(x) = \lambda \begin{cases} x & x \geq 0 \\ \alpha \, (\mathrm{e}^{-x} - 1) & x < 0 \end{cases}$$

it is the last activation function, it is similar to ELU except for the parameter λ; for standard scaled inputs (mean 0, standard deviation

1), the suggested values values are $\alpha = 1.6732$, $\lambda = 1.0507$. The major advantage of using SELU is that it provides self-normalization (that is output from SELU activation will preserve the mean of 0 and standard deviation of 1) and this solves the vanishing or exploding gradients problem. However, this will only happen under certain conditions: (a) the neural network consists only a stack of dense layers, (b) all the hidden layers use SELU activation function, (c) the input features must be standardized (having mean of 0 and standard deviation of 1), (d) the hidden layers weight must be initialized with LeCun normal initialization, and lastly, (e) the network must be sequential [12].

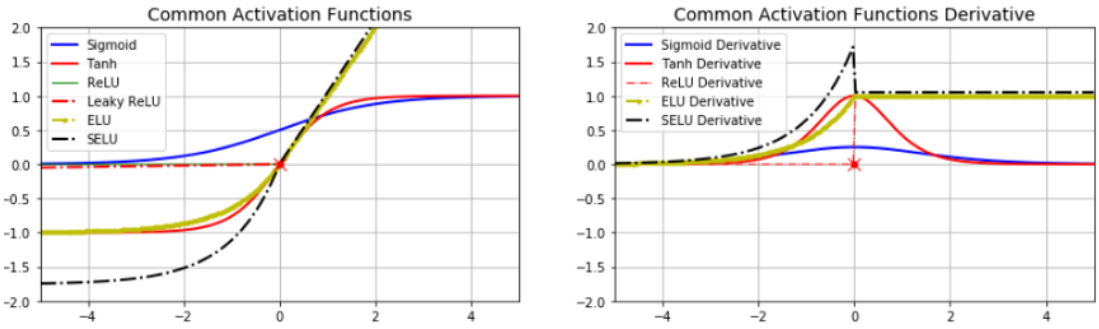

Figure 2.10: Common activation functions in neural networks and their derivatives

- batch size: it is an important hyperparameter which regulates the update frequency of network's parameters . Based on this hyperparameter, the gradient descent algorithm assumes different variants. Depending on the batch size, we can distinguish:

  – Batch Gradient Descent: in this case the batch size is equal to the number of training examples in the training data set; all the training data is taken into consideration to take a single step. We take the average of the gradients of all the training examples and then use that mean gradient to update our parameters. So that's just one step of gradient descent in one epoch. If the optimization problem is convex, it ensures the convergence to the global minimum; in non-convex problem, to a local minimum. The main disadvantage is related to its computational cost: especially for very large training data set its convergence to the global minimum is very low.

  – Mini-Batch Gradient Descent: it differs from the previous one due to the fact that weights are updated more than once per epoch. In fact, the training data set is split into smaller batches, then, for each batch, the mean gradient is computed and used to update the weights.

  – Stochastic Gradient Descent: in Batch gradient descent, all the examples must be considered for every step of Gradient Descent and this could generate too high computational costs if the data set is large. To fix this issue Stochastic gradient descent is introduced: it allows to consider just

one example at a time to take a single step. For every training example, the gradient is computed and used to update the weights.

To summarize:

- Batch size = Size of the training set $\longrightarrow$ Batch Gradient Descent
- Batch size < Size of the training set $\longrightarrow$ Mini-Batch Gradient Descent
- Batch size = 1 $\longrightarrow$ Stochastic Gradient Descent

2.2.2 Recurrent Neural Networks

The first part of the thesis work is based on Recurrent Neural Network, in particular on Long Short Term Memory Neural Networks. Recurrent neural networks, or RNNs, are a family of neural networks for processing sequential data, in particular, a recurrent neural network is a neural network that is specialized for processing a sequence of values $x^1, \ldots, x^t$. They are used in the field of Natural Language Processing, but their application is also common in time series forecasting problems, as an alternative to more traditional statistical methods. The main difference with respect to a standard neural network is that RNNs can share features between layers, each layer is not treated independently. In this way, parameter sharing makes it possible to extend and apply the model to examples of different forms and generalize across them. The general architecture of a Recurrent Neural Network is shown in the following picture:

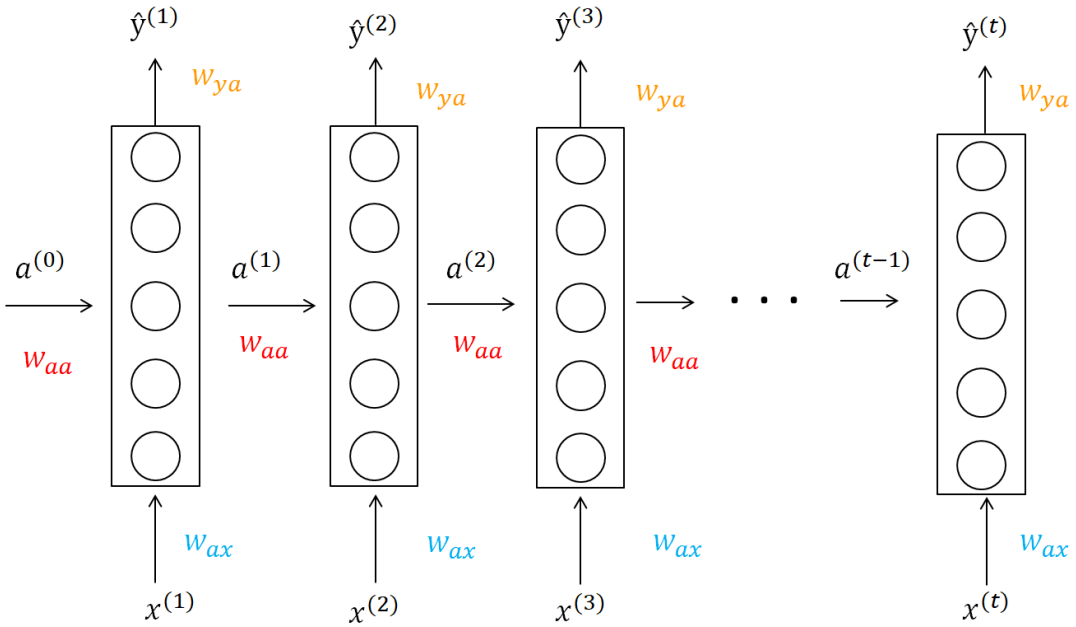

Figure 2.11: Unfolded Structure of a Recurrent Neural Network

At each time step, RNN passes an activation a^{t-1} to the next layer, generally the first activation a^0 is initialized to zero, despite some researchers prefer to initialize it randomly. RNN scans through the data from left to right and its predictions $\hat{y}^t$ are

not only based on current input x^t but also on previous inputs, since informations are shared through the activation a. The weights w_{aa} , w_{ax} , w_{ya} are the same for every layer: RNNs, in fact, could be represented in a more general way as:

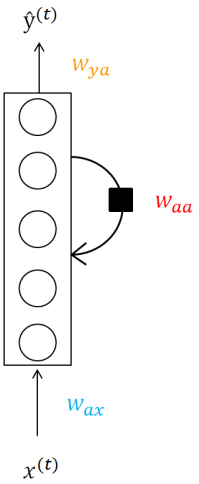

Figure 2.12: General Structure of a Recurrent Neural Network

The forward propagation computed by an RNN could be mathematically summarized as follow:

$$a^{(1)} = g_1\Big(w_{aa}\, a^{(0)} + w_{ax}\, x^{(1)} + b_a\Big)$$
$$\hat{y}^{(1)} = g_2\Big(w_{aa}\, a^{(0)} + w_{ax}\, x^{(1)} + b_a\Big)$$

where g_1 is generally a *tanh* function or a ReLU function while the choice of g_2 would depend on the type of output, for example, in binary classification problems, g_2 is generally a sigmoid function while for multi-class problems is typically used a Softmax function.

2.2.3 Backpropagation through time

Backpropagation through time is actually a specific application of backpropagation used to train RNNs. It requires to expand the computational graph of an RNN one time step at a time to obtain the dependencies among model variables and parameters. The loss function is computed for each individual time step, then, each loss is summed to the others to obtain the total loss. Starting from the total loss it is possible to compute the gradients of the loss with respect to model's parameter. These computations are based on the chain rule, which was analyzed before.
In this procedure, the most significant recursive calculation is the horizontal one, which goes from the right to the left. The mathematical description of this algorithm is more complex than that of standard backpropagation; for a detailed explanation is suggested to see [13]. The computation graph of backpropagation through time is shown in the following figure:

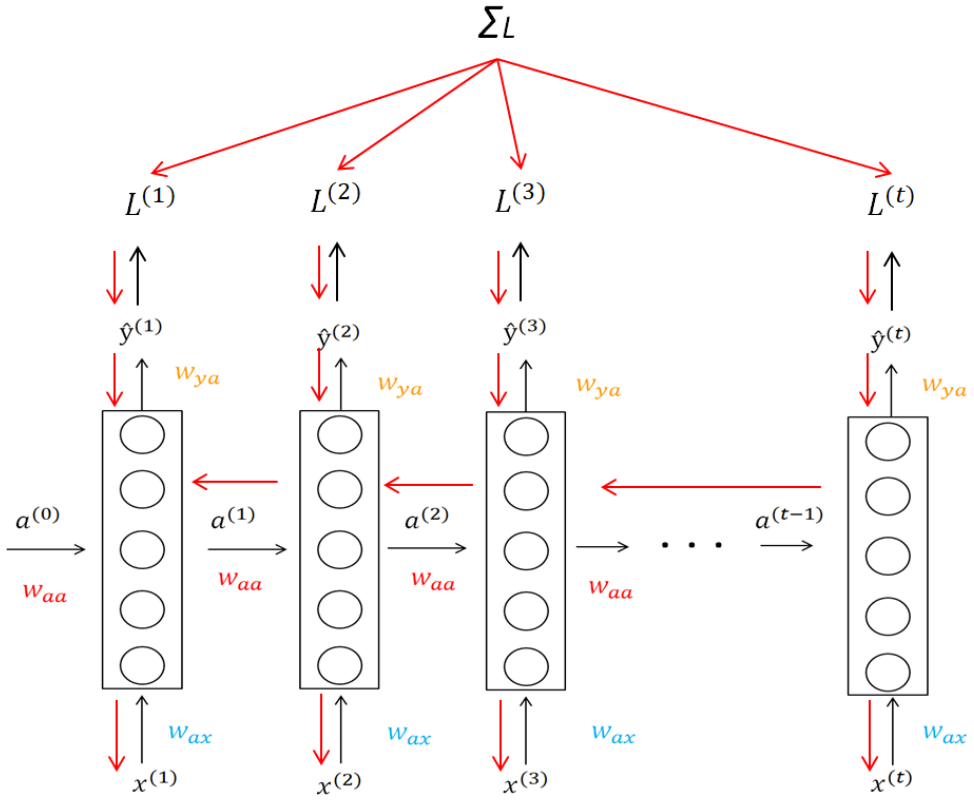

Figure 2.13: Computation graph of backpropagation through time

2.2.4 Vanishing and Exploding Gradients in RNNs

Deep neural networks have several stability issues associated with training. In particular, networks with many layers may be hard to train because of the way in which the gradients in earlier and later layers are related. In fact, basic Recurrent Neural Networks are not good in capturing very long term dependencies, this is mainly due to a problem called vanishing gradient: especially in very deep neural networks, backpropagation has less effect on the computation of parameters in the first layers: the error computed on later time steps hardly influences earlier computations. Because of this problem, basic RNNs have many local influences which means that the local output $\hat{y}_t$ has influence only on neighboring layers and this of course is one of the mainly weakness of the basic RNN algorithm. To have a mathematical intuition of the problem, it is possible to consider a simple RNN with only one neuron in each hidden layer, with sigmoid activation function:

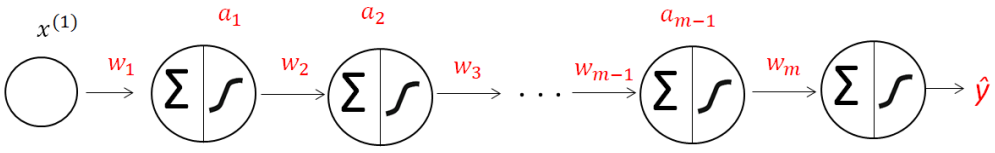

Figure 2.14: Example of vanishing gradient problem

It is relatively easy to use the backpropagation update to show the following relationship:

$$\frac{\partial \mathscr{L}}{\partial a_t} = \frac{\partial \sigma(z_{t+1})}{\partial z_{t+1}} \, w_{t+1} \, \frac{\partial \mathscr{L}}{\partial a_{t+1}}$$

Making the assumption that weights are initialized from a standard normal distribution, each weight would have an expected average magnitude of 1. To understand how each of those terms behave it is important to observe the graph of the derivative of the sigmoid function:

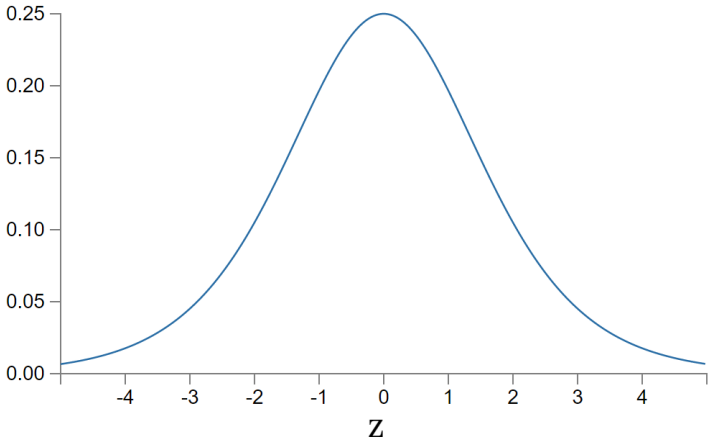

Figure 2.15: Derivative of the Sigmoid Function

The derivative reaches a maximum equal to 0.25 at $z = 0$, since the value of w_{t+1} is expected to be 1, it follows that each weight update will (typically) cause the value of $\frac{\partial \mathscr{L}}{\partial a_t}$ to be less than 0.25 that of $\frac{\partial \mathscr{L}}{\partial a_{t+1}}$. Therefore, after moving by about r layers, this value will typically be less than 0.25^r. Just to get an idea of the magnitude of this drop, if we set r = 10, then the gradient update magnitudes drop to 10^{-6} of their original values. Therefore, when backpropagation is used, the earlier layers will receive very small updates compared to the later layers. To solve this problem it is possible to use an activation function with larger gradients and also initializing the weights to be larger. However, it is easy to end up in the opposite situation where the gradient explodes in the backward direction instead of vanishing. This phenomenon is called *Exploding Gradient*. Exploding gradients are easier to spot, since parameters values become very large; a possible solution to fix this issue is called *Gradient Clipping* which is a method that re-scale gradients which are bigger than a certain threshold. However, vanishing gradient problems are harder to solve. A very effective solution to vanishing gradients is represented by a particular kind of RNN called *Long Short Term Memory*, which allows to consider long-term dependencies.

2.2.5 Long Short Term Memory (LSTM)

Long Short Term Memory networks, usually just called "LSTMs", are a special kind of RNN, capable of learning long-term dependencies. They were introduced in 1997 by Hochreiter & Schmidhuber and their study had a huge impact on sequence modeling. LSTMs can manage the vanishing/exploding gradient problem by remembering important informations and forgetting those that are useless. This property of LSTMs is due to a particular gating mechanism and to the presence of two states:

- Hidden state which is responsible of mantaining the short term memory
- Cell state which is responsible of maintaining the long term memory and capturing long term dependencies.

In particular, memory cells use gates to regulate the information to be kept or discorded at each time step before passing on the the long term and short term information to the next cell gates: this mechanism is able to filter the irrelevant information. The scheme of the LSTM cell is shown in the following figure:

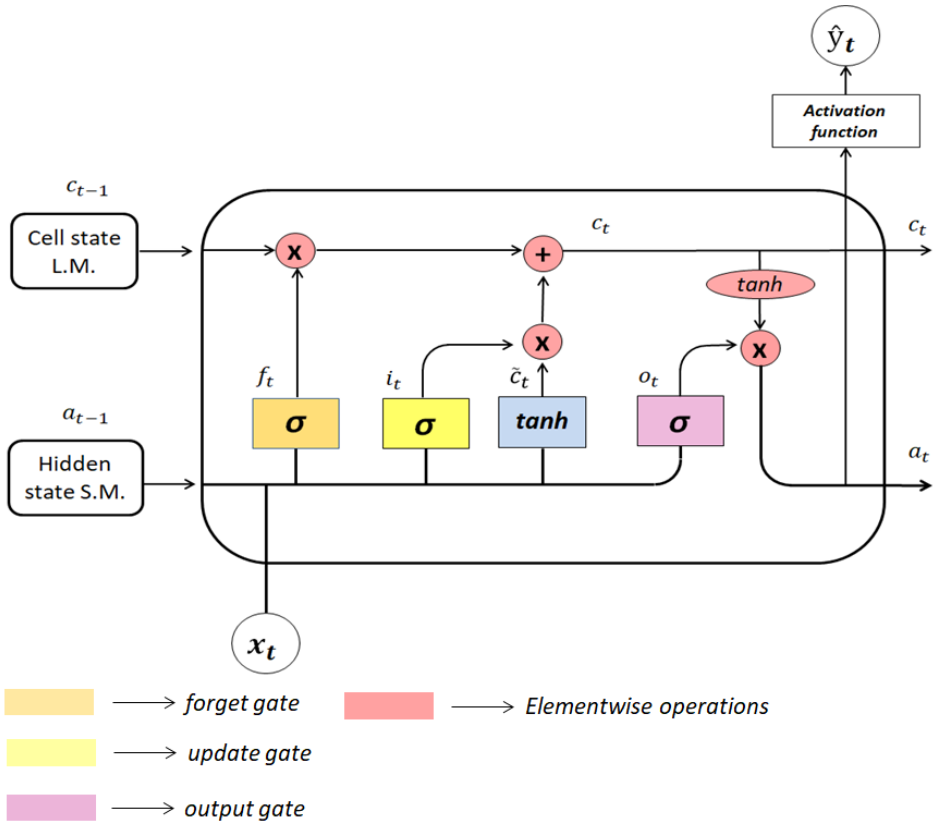

Figure 2.16: LSTM cell scheme

The key to LSTMs is the cell state, which maintains long term dependencies: it runs straight down the entire sequence, with only some minor linear interactions.

It's very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means that the component is not relevant and could be discarded while a value of 1 means that the component is important and has to be maintained.

Forget gate

The first step in an LSTM is to decide what information are going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at a_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state c_{t1}

$$f_t = \sigma\left(W_f[a_{t-1}, x_t] + b_f\right)$$

where W_f is obtained by stacking horizontally w_{aa} and w_{ax}.

Update gate

The next computations are used to decide what new informations are going to be stored in the cell state.This process is divided in two parts:

- the first one involves a sigmoid layer called *"input gate layer"* which decides which values are going to be updated

$$i_t = \sigma\left(W_i[a_{t-1}, x_t] + b_i\right)$$

- the second one involves a *tanh* layer which creates a vector of new candidate values, $\tilde{c_t}$, that could be added to the state.

$$\tilde{c_t} = \tanh\left(W_c[a_{t-1}, x_t] + b_c\right)$$

The elementwise multiplication of i_t and $\tilde{c_t}$ generates a vector that is used to update the cell state c_t; in particular this vector represents the new candidate values, scaled by how much we decided to update each state value.

Output gate

The output of the LSTM cell will be based on a filtered version of the cell state. First, a sigmoid layer decides what parts of the cell state are going to be output. Then, the cell state pass through a *tanh* (to push the values to be between 1 and 1) and is multiplied by the output of the sigmoid gate.

$$o_t = \sigma\left(W_o[a_{t-1}, x_t] + b_o\right)$$

$$a_t = o_t * \tanh\left(c_t\right)$$

2.2.6 Types of RNN architectures

The basic RNN architecture could be modified in order to solve different kind of problems. In particular it is possible to distinguish:

- One to One: it is also known as vanilla neural network and it is used for basic machine learning problems
- One to Many: it has a single input and many outputs. Its applications are mainly in the field of music generation or image captioning.
- Many to One: it has many inputs and a single output. This is basically used in sentiment classification. Another important application is in the field of time series forecasting, where the input is a sequence of n values and the output is the prediction of the single value at position n+1. This is the configuration used in this thesis work.
- Many to Many: it takes a sequence of inputs and generate a sequence of outputs. There are two configurations of Many to Many, the simpler has one output for each input in the sequence, the most complex has a structure called Encoder-Decoder and is used when the input and the output length is different. Their application are common in the field of Machine Translation but also in time series forecasting.

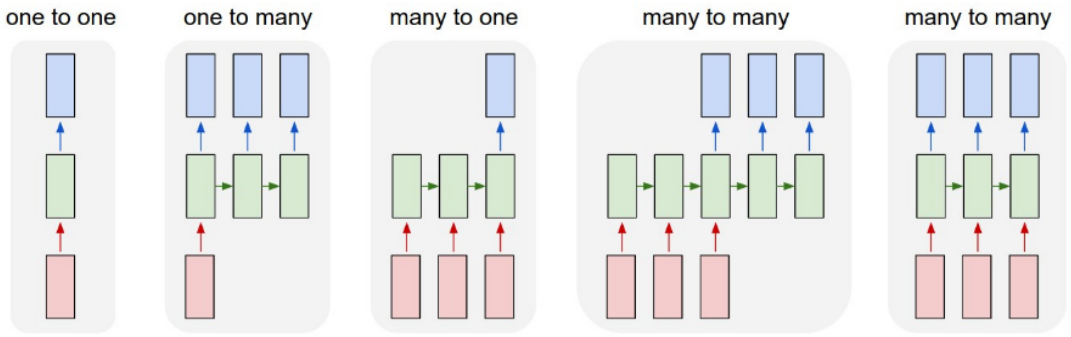

Figure 2.17: Different types of RNNs

Chapter 3

Development of LSTM Building Models

3.1 Literature survey on black box modelling for Indoor Temperature Prediction

Indoor Temperature prediction through machine learning techniques has gained increased interest in recent years, due to the increasing data availability and to the necessity of computationally lightweight models, which can exploit data coming from sensors. Moreover, ANN models are suitable for building dynamics modelling due to their abilities to deal with nonlinear, multivariable modelling problems and generally outerform statistical models which are linear, time-invariant and can easily lose accuracies when strong nonlinearities and uncertainties are presented in the systems. One of the first works in this field has been carried out in 2005 by Ruano et al.[14]. The aim of its study was to develop a radial basis function (RBF) neural network to predict the indoor air temperature of a public secondary scholar building situated in the south of Portugal. The network performances were compared with those of a physical model, showing slightly better predictions. Due to the low computational expenses, neural network is used to build a air-conditioning systems controller, in order to improve on-off schedules and save energy. The study used real data which were acquired every minute. Another interesting work was performed by Mustafaraj et al. [15] that proposed two models, a simpler autoregressive model with exhogenous inputs (ARX) and a more complex neural network-based non linear autoregressive model with exhogenous inputs (NNARX), to predict the indoor temperature and the relative humidity in an office building located in London. The results show that the second model has better performances since it is able to better approximate the complex relations between inputs and outputs such as room temperature and relative humidity. In [16] the authors developed a single zone and a multi zone NNARX model to forecast the indoor temperature in an university campus, in order to improve energy management strategies. Moreover, Huang et al. [17] developed a neural network in order to predict the indoor air temperature of a multizone building and verify the effectiveness of a neural network

based predictive control in terms of energy saving. In [18] authors describes the application of a combined neuro-fuzzy model for indoor temperature dynamic and automatic regulation: NNARX temperature forecasts are used to feed a fuzzy logic control which regulates the HVAC system. Ellis et al. [19] developed an Encoder-Decoder LSTM to model an air handling unit-variable air volume (AHU-VAV) HVAC system. Its predictions are used to optimize energy costs through an MPC and its performance are evaluated by means of E+ software. Another very interesting paper is represented by [20] in which researchers used an LSTM to predict the indoor air temperature in a multi zone building, using the Sequence to Sequence approach to perform multi step forecast. The results of the study have been evaluated on the base of two case studies on real smart buildings using variable air volume (VAV) and constant air volume (CAV) systems.

3.2 Description of Modeled Buildings

The buildings considered in this thesis work are prototypes of typical commercial buildings in the U.S. The study is focused on four buildings:

- Small Office
- Medium Office
- Restaurant full-service
- Retail Stand-alone

The following figure shows the rendering of the models by means of SketchUp software:

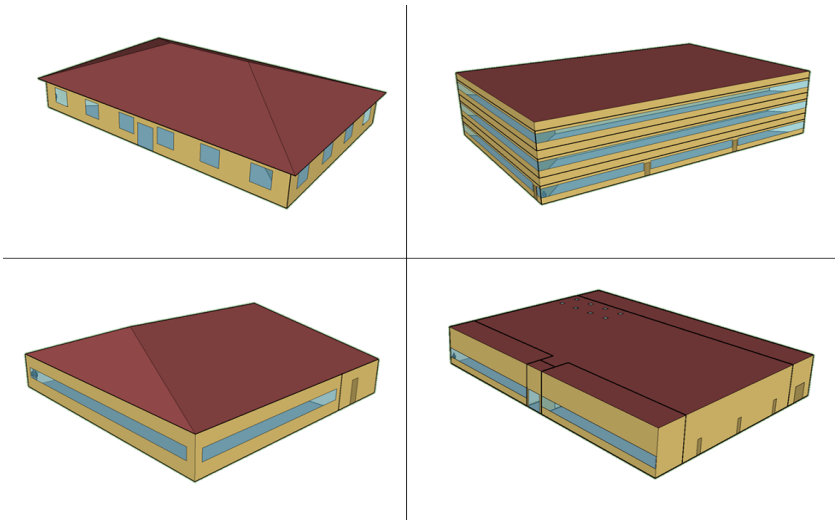

Figure 3.1: Buildings Geometric Model, starting from the top left corner and following the clockwise direction: Small Office, Medium Office, Retail stand-alone, Restaurant

The following tables report the general characteristics of considered buildings in terms of dimensions, occupancy, thermal zones, outdoor flow rate and building envelope (both opaque and transparent) thermophysical properties:

Dimension and Geometry			
Length	27.7 m	Gross Floor Area	511 m2
Height	3.1 m	Gross Roof Area	598.8 m2
Gross Wall Area	281.5 m2	Total gross Volume	2279.6 m3
Window Opening Area	59.7 m2	Surface to Volume Ratio	0.61 m-1
Gross Window-Wall Ratio	21.2 %		
Main Boundary Conditions of Energy Simulations			
Climatic data	EPW file	Floor Max Occupancy	31
Number of thermal zones	6 (5 conditioned, 1 unconditioned)		
Outdoor air flow rate	0.011 m3/s/person		
Building Opaque Envelope			
Construction	Reflectance	U-Factor with Film [W/m2-K]	U-Factor no Film [W/m2-K]
Exterior wall	0.3	0.363	0.384
Ground floor slab	1	-	0.227
Attic Soffit	0.3	3.082	7.547
Attic Roof	0.3	2.858	4.706
Building Transparent Envelope			
Construction	Glass U-Factor [W/m2-K]	Glass SHGC	Glass Visible Transmittance
Window	2.045	0.355	0.397

Table 3.1: General characterization of Small Office Building

Dimension and Geometry			
Length	49.9 m	Gross Floor Area (3 floors)	4981 m2
Height	11.9 m	Gross Roof Area	1660 m2
Gross Wall Area	1369 m2	Total gross Volume	19776.6 m3
Window Opening Area	652 m2	Surface to Volume Ratio	0.27 m-1
Gross Window-Wall Ratio	26.81 %		
Main Boundary Conditions of Energy Simulations			
Climatic data	EPW file	Bottom Floor Max Occupancy	89
Number of thermal zones	18 (15 conditioned, 3 unconditioned)	Mid Floor Max Occupancy	89
Outdoor air flow rate	0.011 m3/s/person	Top Floor Max Occupancy	89
Building Opaque Envelope			
Construction	Reflectance	U-Factor with Film [W/m2-K]	U-Factor no Film [W/m2-K]
Exterior wall	0.3	0.363	0.384
Ground floor slab	1	-	0.237
Roof	0.3	0.182	0.184
Building Transparent Envelope			
Construction	Glass U-Factor [W/m2-K]	Glass SHGC	Glass Visible Transmittance
Window	2.045	0.355	0.397

Table 3.2: General characterization of Medium Office Building

Dimension and Geometry			
Length	22.61 m	Gross Floor Area	511.21 m2
Height	3.05 m	Gross Roof Area	569.51 m2
Gross Wall Area	275.72 m2	Total gross Volume	2414.66 m3
Window Opening Area	47.12 m2	Surface to Volume Ratio	0.58 m-1
Gross Window-Wall Ratio	17.11 %		
Main Boundary Conditions of Energy Simulations			
Climatic data	EPW file	Dining Occupancy	280
Number of thermal zones	3 (2 conditioned, 1 unconditioned)	Kitchen Occupancy	8
Kitchen Outdoor air flow rate	0.0165 m3/s/m2	Dining Outdoor air flow rate	0.01 m3/s/person

Building Opaque Envelope			
Construction	Reflectance	U-Factor with Film [W/m2-K]	U-Factor no Film [W/m2-K]
Exterior wall	0.3	0.363	0.384
Ground floor slab	0.4	4.386	15.157
Roof	0.5	2.858	4.706
Building Transparent Envelope			
Construction	Glass U-Factor [W/m2-K]	Glass SHGC	Glass Visible Transmittance
Window East-West	2.447	0.305	0.271
Window South	2.371	0.397	0.444

Table 3.3: General characterization of Restaurant Building

Dimension and Geometry			
Length (major side)	54.27 m	Length (major side)	42.27 m
Height	6.10 m	Gross Floor Area	2294 m2
Gross Wall Area	1177.02 m2	Gross Roof Area	2294 m2
Window Opening Area	83.94 m2	Total gross Volume	13993.36 m3
Gross Window-Wall Ratio	7.13 %	Surface to Volume Ratio	0.418 m-1
Main Boundary Conditions of Energy Simulations			
Climatic data	EPW file	Back Space Occupancy	61
Number of thermal zones	5 (4 conditioned, 1 unconditioned)	Core Retail Occupancy	259
		Point of Sale Occupancy	24
		Front Retail Occupancy	24
		Front Entry Occupancy	2
Outdoor air flow rate	0.0115 m3/s/person		

Building Opaque Envelope			
Construction	Reflectance	U-Factor with Film [W/m2-K]	U-Factor no Film [W/m2-K]
Exterior wall	0.3	0.591	0.648
Ground floor slab	0.3	2.25	3.54
Roof	0.23	0.181	0.186
Building Transparent Envelope			
Construction	Glass U-Factor [W/m2-K]	Glass SHGC	Glass Visible Transmittance
Skylight Window	2.956	0.335	0.452
Window Point of Sale South	2.371	0.397	0.444
Window Front Retail South	2.371	0.397	0.444
WindowFront Entry South	2.371	0.397	0.444

Table 3.4: General characterization of Retail stand-alone Building

Buildings were simulated taking into account the EPW climate file of Albuquerque International Sunport, New Mexico, which belongs to the 4B climate zone. They were modeled starting from commercial reference buildings developed by U.S. Department of Energy (DOE) [21]. These models, defined by the National Renewable Energy Laboratory (NREL), are designed to cover among 70% of commercial buildings types in United States. The aim of these models is to represent both new and existing buildings in order to verify the effectiveness of new technologies in terms of energy efficiency, advanced control system, indoor air quality, ecc. Starting from these models, the real HVAC plant was changed with an ideal one, using the

object "HVAC Template: IdealLoadsAirSystem", available in EnergyPlus. This object provides an ideal system to supply conditioned air to the zone that meets all the load requirements and it is often used for load calculations. Each building is equipped with an HVAC system that operates according the following schedules:

Small Office																							
Hour	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Week days	30	30	30	30	30	30	28	26	26	26	26	26	26	26	26	26	26	26	30	30	30	30	30	30
Saturday	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Sunday	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30

Medium Office																							
Hour	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Week days	30	30	30	30	30	30	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	30	30
Saturday	30	30	30	30	30	30	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	30	30
Sunday	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30

Restaurant Full-service																							
Hour	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Week days	30	30	30	30	30	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26
Saturday	30	30	30	30	30	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26
Sunday	30	30	30	30	30	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26

Retail Stand-alone																							
Hour	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Week days	30	30	30	30	30	28	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	30	30	30
Saturday	30	30	30	30	30	28	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	30	30
Sunday	30	30	30	30	30	30	30	28	26	26	26	26	26	26	26	26	26	26	26	30	30	30	30	30

Table 3.5: Temperature Schedules

Small Office																							
Hour	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Week days	-	-	-	-	-	-	-	50	50	50	50	50	50	50	50	50	50	50	-	-	-	-	-	-
Saturday	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sunday	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Medium Office																							
Hour	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Week days	-	-	-	-	-	-	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	-	-
Saturday	-	-	-	-	-	-	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	-	-
Sunday	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Restaurant Full-service																							
Hour	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Week days	-	-	-	-	-	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
Saturday	-	-	-	-	-	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
Sunday	-	-	-	-	-	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50

Retail Stand-alone																							
Hour	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Week days	-	-	-	-	-	-	50	50	50	50	50	50	50	50	50	50	50	50	50	50	-	-	-	
Saturday	-	-	-	-	-	-	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	-	-	
Sunday	-	-	-	-	-	-	-	-	50	50	50	50	50	50	50	50	50	50	50	-	-	-	-	-

Table 3.6: Relative Humidity Schedules

3.3 Model development process

The aim of this section is to explain how a robust training data set was generated. In particular, the main idea behind the work is to implement a data driven model for each building which is able to map the relations between uncontrollable variables such as the outdoor temperature, the direct solar radiation, the occupancy profile, controllable variables as the total HVAC cooling power and the mean indoor temperature. Of course, since the cooling power of the HVAC system is the only input variable which is controllable, the data set creation is mainly focused on it. The aim of the control is to act only on the sensible contribution: in an all-air systems, sensible and latent load are not physically separable, since there is a single cooling coil per each air handling unit. To overcome this limitation, the proposed methodology was applied to mixed air-water systems: these kind of HVAC systems combine both advantages of all-air and all-water systems; in particular, the temperature control is performed by water terminals (e.g.fan coils) while ventilation and humidity control is performed by the AHU. For this reason, in this way it is possible to separately control latent and sensible power. The main control objective, in fact, is to maintain the mean indoor temperature within a comfort band which goes from 25 °C to 27 °C. The difficult part of the data generation process is related to the fact that the thermodynamic of a building is strongly non linear so it is not easy to have a model which is able to efficiently predict the indoor air temperature. Moreover, in this case study, the building can receive a cooling load different from the ideal one, which further complicates the analysis. To overcome this issue, the training data set does not only contain the ideal scenario, but also the results of a simulation in which the sensible cooling load differs from the ideal one. Since the actions of a Reinforcement Learning Agent could not be predicted a priori, the additional data set introduce a sensible cooling load which is randomly distributed around the ideal one. In particular, the hourly ideal sensible power is multiplied by a random variable sampled between 0.3 and 1.7 according to a uniform distribution; however, this multiplication does not happen always, but with a probability of 60%. This data set is built in order to highlight, given certain values of occupancy, outdoor temperature, solar radiation, the influence between the sensible cooling power and the indoor air temperature. This procedure has been possible thanks to two Energy plus objects: the first one is called *Schedule: File* and the second one is called *OtherEquipment*. The former is used to read a csv file and to use the read values as a schedule. The latter is then used to introduce the cooling power in the building: in this case, in fact, the ideal HVAC plant was substituted from this object. A particular feature of this object is that it allow to specify the latent and the convective fraction of cooling energy. In particular, the latent power was considered fully latent while the sensible one fully convective. The data generation procedure could be summarized in two steps, the first one is running an EnergyPlus simulation with the ideal HVAC plant; the second one consists in taking the ideal load, both latent and sensible, multiply the latter by the random coefficient previously explained, and use them as inputs in a simulation performed without the ideal HVAC system but with the *OtherEquipment* Object. The output of these simulations have been stored in a csv file. The process is schematized is

the following figure:

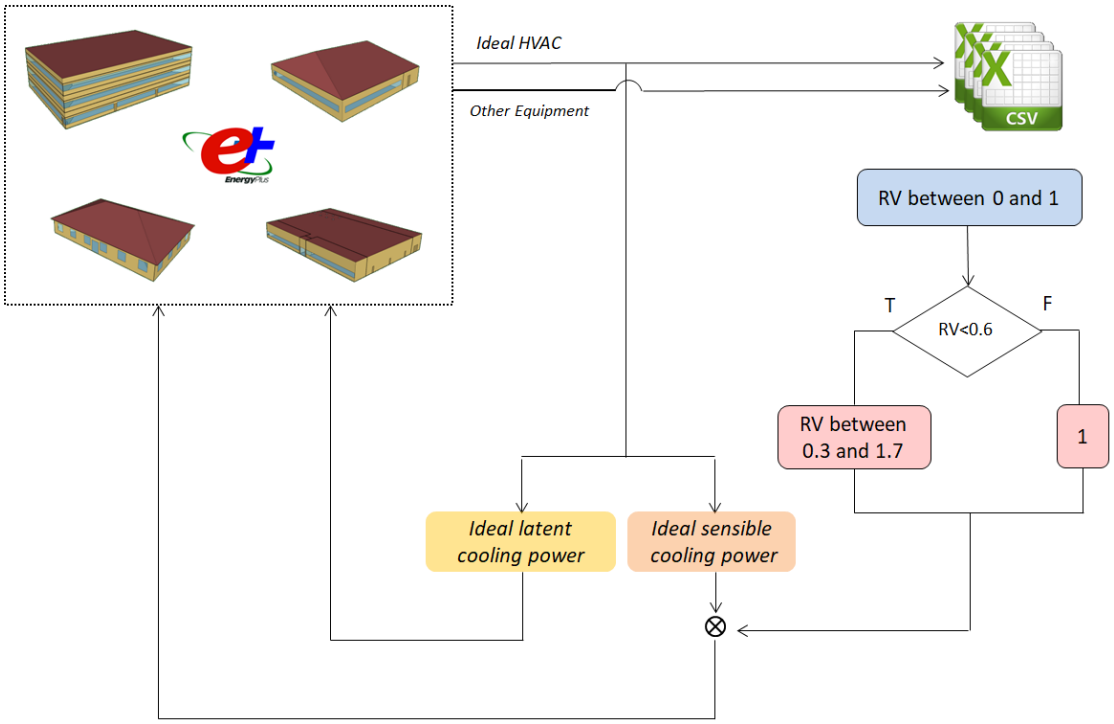

Figure 3.2: Conceptual scheme of training data set generation process

3.3.1 Input selection and data preprocessing

The main idea behind the choice of model's input is to use variables which could be easily available also in a real case application; for this reason, the chosen variables are:

- the hour of the day, from 1 to 24
- the day of the week, from 1 to 7, where 1 indicates Sunday
- the month, from 6 to 9, since the case study deals with the cooling season
- the Outdoor Air Temperature
- the Direct Solar Radiation
- the Occupancy
- the Total Cooling Load
- the Mean Indoor Air Temperature at previous timestep

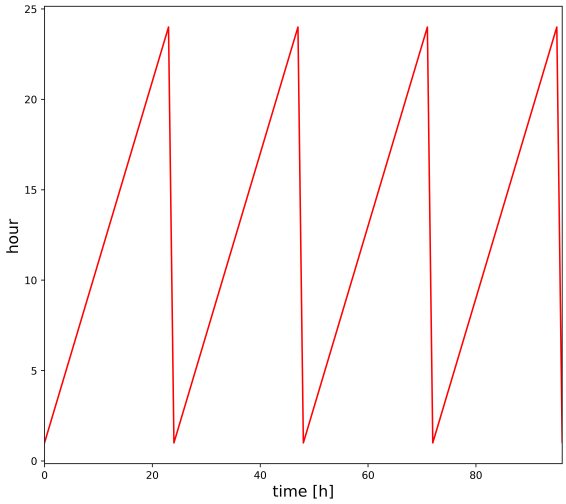

Figure 3.3: Hour without cyclical encoding

Temporal variables of course are cyclical, the problem is letting the deep learning algorithm know that these features occur in cycles. For this reason cyclical data must be transformed to help the model to understand their behaviour.
This graph illustrates the problem with presenting cyclical data to a machine learning algorithm: there are jump discontinuities in the graph at the end of each day, when the hour value goes from 24 to 01. If hour features remains unencoded, it is possible to observe a failure in current representation: it seems that 1 and 24 are 23 hours apart, when, in reality, they differs only of 1 hour. To overcome this limitation it is possible to change the encoding system: a common method for encoding cyclical data is to transform the data into two dimensions using a sine and cosine transformation:

$$x_{cos} = \cos\left(\frac{2\pi x}{\max(x)}\right)$$

$$x_{sin} = \sin\left(\frac{2\pi x}{\max(x)}\right)$$

This procedure has to be repeated for each temporal feature; both sin and cosine transformation are required in order to properly identify each time variable. The results of the transformation are shown in the following figure:

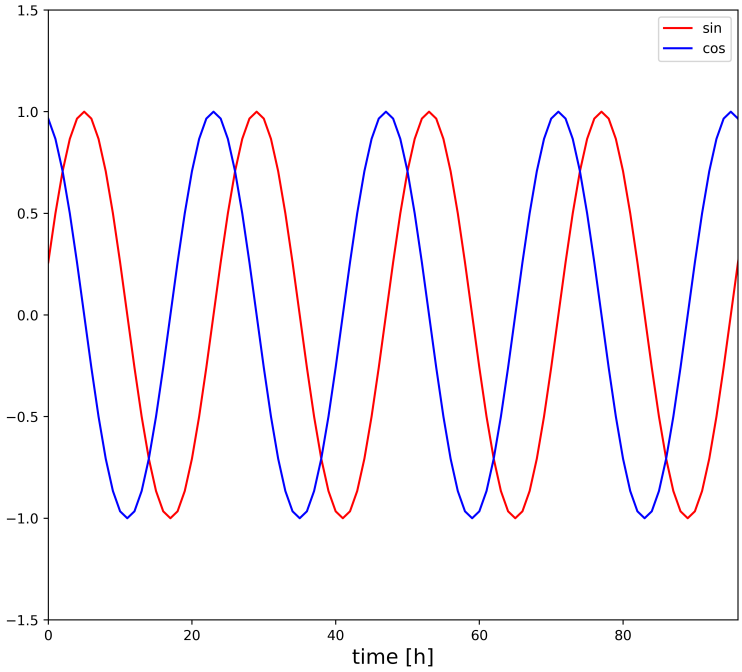

Figure 3.4: Hour with cyclical encoding

An important step during data preprocessing for time series forecasting with machine learning models is to convert the time series into a supervised learning problem. In time series analysis, this procedure is performed through a technique called *Sliding Window*: the window contains all the variables related to the forecasted one, including the past values of the latter. In general, the window has a fixed length which is called *'lookback'* which indicates how many past time steps are used in the forecasting procedure. In this case study, data are reshaped in a slightly different way with respect to standard forecasting problems: each variable has the same lookback value but the indoor temperature has a lag of one hour with respect to the other variables; this is due to the fact that the aim of the analysis is to have a model that forecasts the indoor temperature at a certain time t knowing all the other variables at the same time step. In the case of one step ahead forecast, the window slides of one step at time: at each time step, the input is a matrix of *lookback* rows and $n°$ *features* columns. Each example is then stacked one behind the other, along a third dimension: the time series is converted into an input tensor. A tensor is a generalization of vectors and matrices and is easily understood as a multidimensional array; a vector is a one-dimensional or first order tensor and a matrix is a two-dimensional or second order tensor.
A conceptual scheme of the *time series to supervised learning* process using the sliding window technique to create input tensor is shown in the following figure:

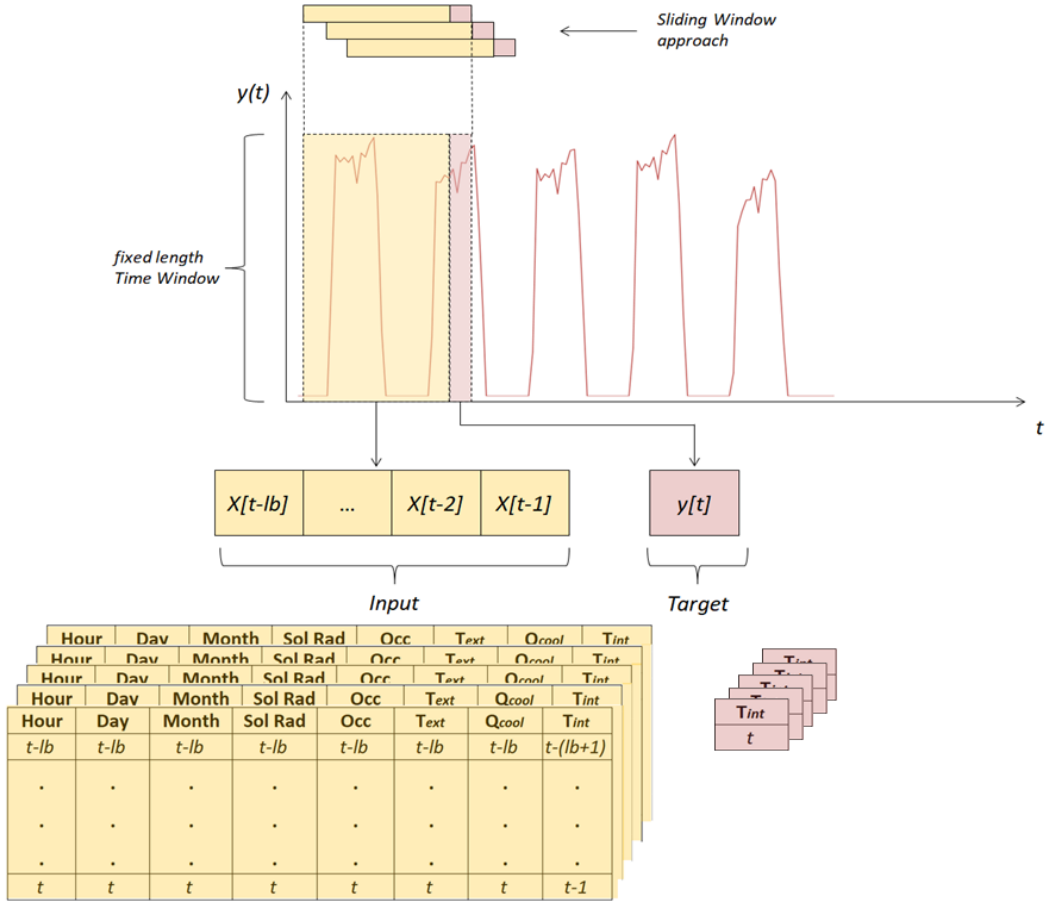

Figure 3.5: Sliding window approach used to frame the forecasting problem into a supervised machine learning problem

Another important step during data preparation involves using techniques such as *normalization* and *standardization* to rescale input and output variables prior to training a neural network model. As such, the scale and distribution of the data may be different for each variable. In fact, input variables may have different units (e.g. [°C], [W], [W/m2]) that, in turn, may mean the variables have different scales. Differences in the scales across input variables may increase the difficulty of the problem being modeled. An example of this is that large input values can result in a model that learns large weight values. A model with large weight values is often unstable, meaning that it may suffer from poor performance during learning and sensitivity to input values resulting in higher generalization error. There are two types of data scaling:

- Standardization, which is used when data are normally distributed, in order to rescale the distribution to have mean equal to 0 and the standard deviation equal to 1. The data are standardized according to this formula:

$$x_{std} = \frac{x - \overline{x}}{\sigma_x}$$

- Normalization, which is a rescaling of the data from the original range to 0-1 range and is used when data are not normally distributed. The formula to normalize the data is:

$$x_{norm} = \frac{x - min}{max - min}$$

In this case study, data are normalized because input features were not normally distributed. During data set preprocessing, particular attention was paid to the connection between different data sets. The moving window, in fact, slides of one hour ahead each time step: this process has to be stopped at the end of each set, otherwise an overlap between two different data sets would have occurred and some windows would have contained values both from the beginning of the following processed set and from the end of the previous one. The process is shown, as a flowchart, in the following figure:

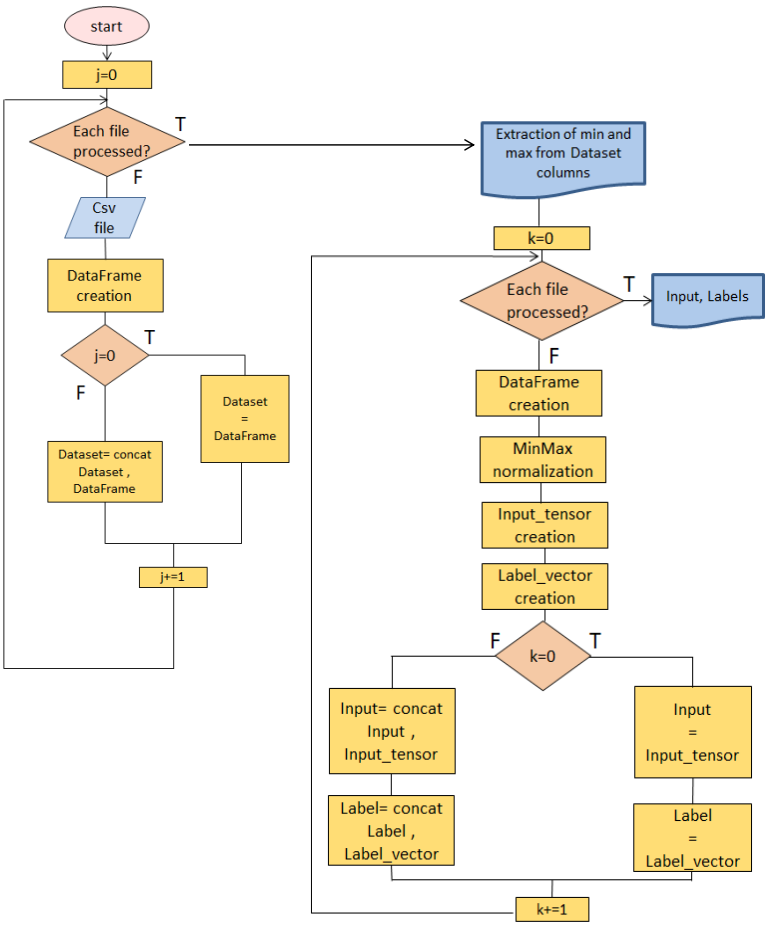

Figure 3.6: Flowchart of data preprocessing

3.3.2 Hyperparameters Selection and Tuning

In machine learning, hyperparameters selection and tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process [22]. They are used to configure various aspects of the learning algorithm and can have wildly varying effects on the resulting model and its performance. First of all is necessary to determine the type of network: in this work, the selected neural network model is the LSTM, which, as previously explained, is often used in time series analysis thanks to its capability to deal with temporal dependencies. Models accuracy was then evaluated considering the following hyperparameters:

- Batch size
- Number of Hidden Layers
- Lookback, which is the number of previous time steps used in the forecasting process
- Learning rate, which determines the step size at each iteration while moving toward a minimum of a loss function

To perform hyperparameters tuning, an additional data set, which is called *validation data set*, was generated: this set was created by multiplying the sensible cooling load by a factor equal to 1.1. Moreover, another data set, called *test data set* was generated, in order to evaluate models accuracy through proper metrics. The test set was generated similarly to the training one, but using a random variable uniformly distributed between 0.8 and 1.2 (instead of 0.3 and 1.7) and a multiplication probability of 20%. The entire procedure could be summarized as follow:

- Split the entire data set data into validation, training, and test sets, in particular 50% training, 25% validation and 25% test
- Identify the hyperparameter set that gives the best performance using the validation data set
- Use the best hyperparameter set to train the final model
- use the trained model (from the best hyperparameter set) to make predictions in test set, and evaluate the performances through proper metrics.

During the tuning process, the following metrics were used:

- Root Mean Square Error, defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

- Mean Absolute Percentage Error, defined as:

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\hat{y}_i - y_i}{y_i}\right|$$

- Determination Coefficient, R^2

For issues related to computational time, the number of LSTM layers was not evaluated as an hyperparameter, however, it was seen that for the Medium Office, only one layer was sufficient to achieve good performances. On the other hand, for the Small Office, the Retail and the Restaurant were necessary two LSTM layers. For the other hyperparameters was prepared a list of values (per each hyperparameter) and the model was trained and evaluated 15 times per each value in the list; for each of this 15 repetitions, the model was trained for 100 epochs. Repetitions are needed because random initial conditions for an LSTM network can generate different results each time a given configuration is trained. The idea is to compare the configurations using summary statistics over a larger number of runs and see exactly which of the configurations might perform better on average. To do so, for each value in the corresponding hyperparameter list, was created a *box plot*. In descriptive statistics, a box plot is a method for graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending from the boxes (whiskers) indicating variability outside the upper and lower quartiles, hence the terms box-and-whisker plot and box-and-whisker diagram. Outliers may be plotted as individual points. To select the best value for each parameter were evaluated both the mean value and the median. Starting from a simple model, with only one hidden neuron, a learning rate equal to 0.001 and a lookback of 12, parameters are sequentially analyzed. The parameter which maximizes metrics was picked and used in the next parameter evaluation: the idea is to increasingly enhance the performance of the model up to a final model, which is the combination of hyperparameters which have maximized metrics across tuning procedure.
Parameters were analyzed in this order:

- Batch size
- Number of Hidden Neurons
- Lookback
- Learning Rate

Results for each building are shown in the following figures:

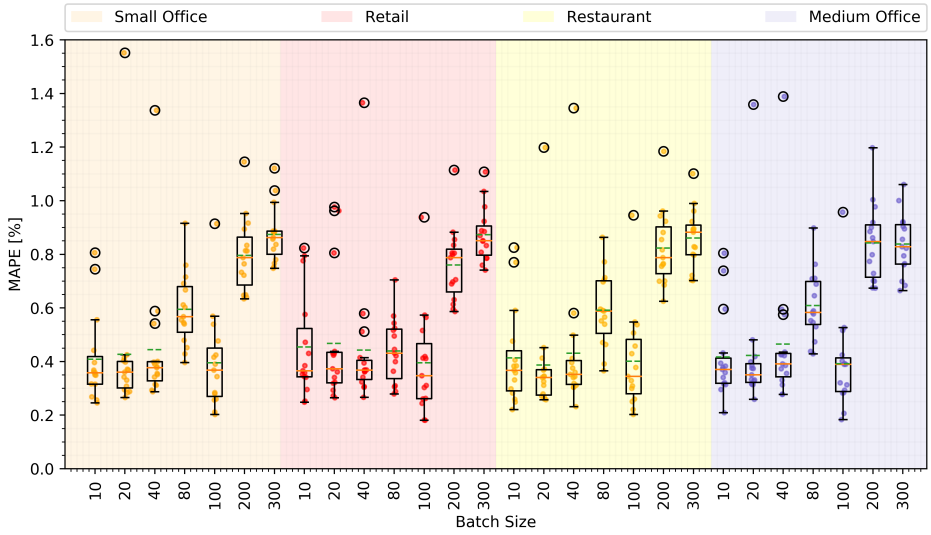

Figure 3.7: MAPE Box plot, hyperparameter = Batch Size

Figure 3.8: RMSE Box plot, hyperparameter = Batch Size

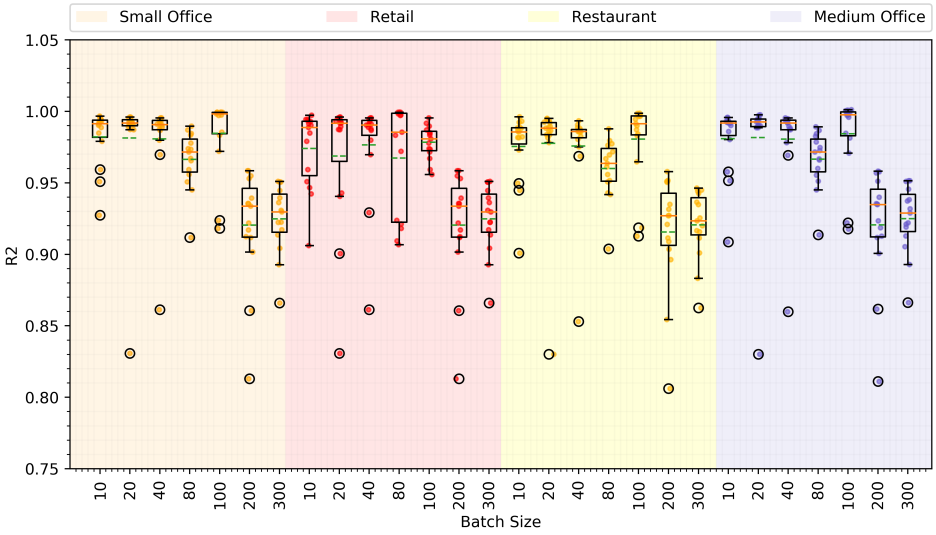

Figure 3.9: R2 Box plot, hyperparameter = Batch Size

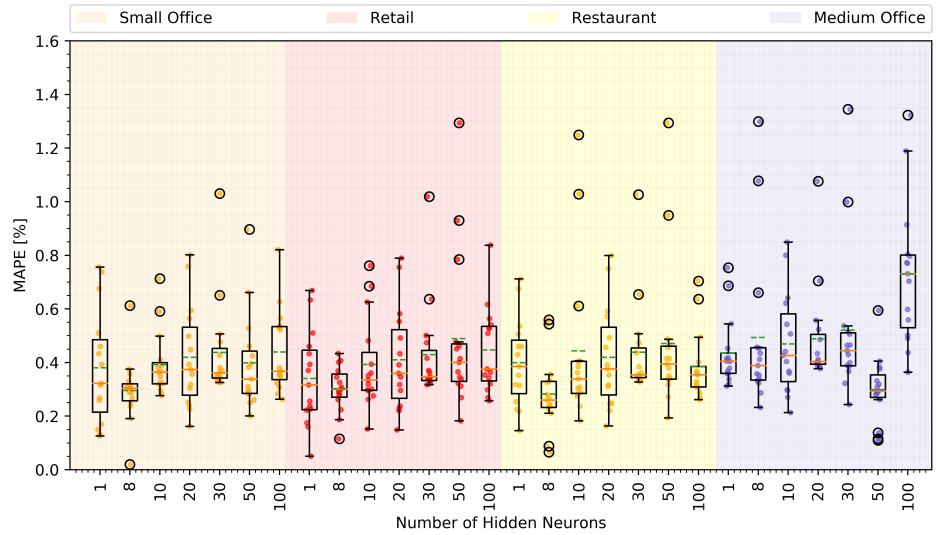

Figure 3.10: MAPE Box plot, hyperparameter = Number of Hidden Neurons

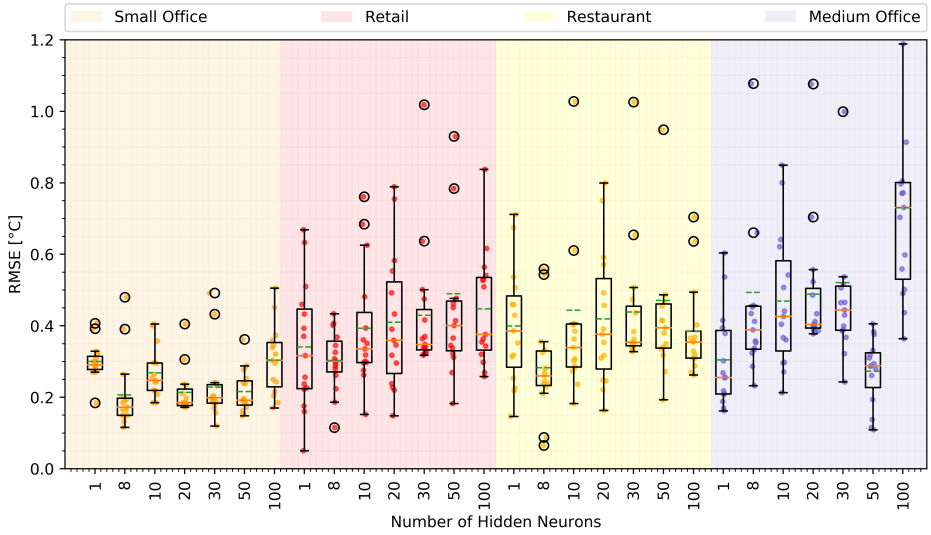

Figure 3.11: RMSE Box plot, hyperparameter = Number of Hidden Neurons

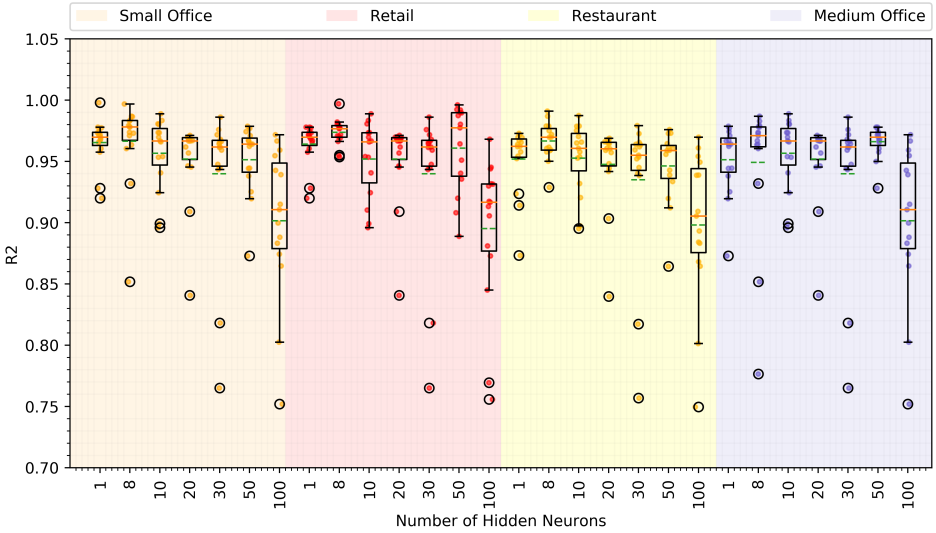

Figure 3.12: R2 Box plot, hyperparameter = Number of Hidden Neurons

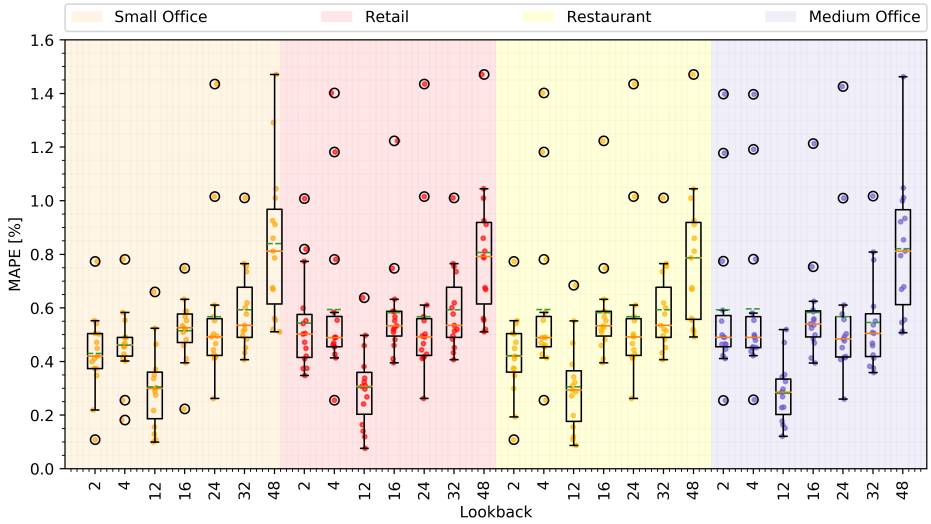

Figure 3.13: MAPE Box plot, hyperparameter = Lookback

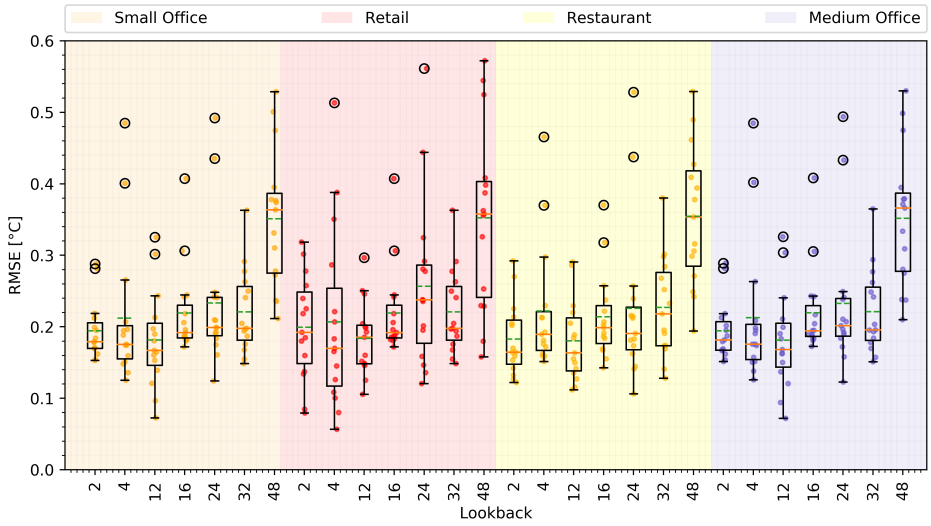

Figure 3.14: RMSE Box plot, hyperparameter = Lookback

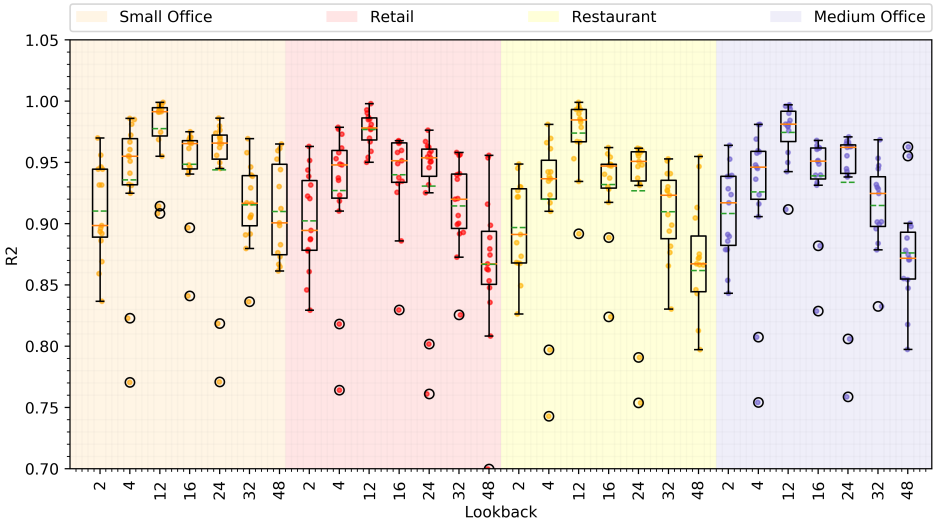

Figure 3.15: R2 Box plot, hyperparameter = Lookback

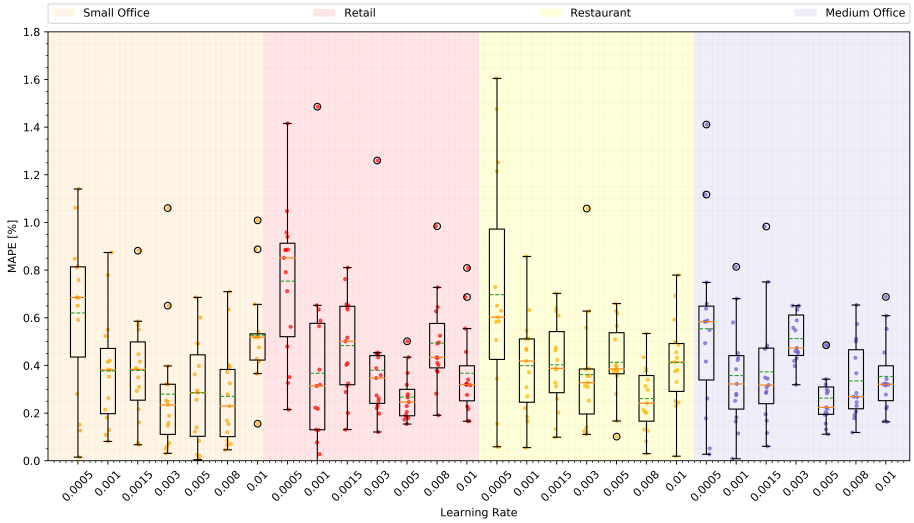

Figure 3.16: MAPE Box plot, hyperparameter = Learning Rate

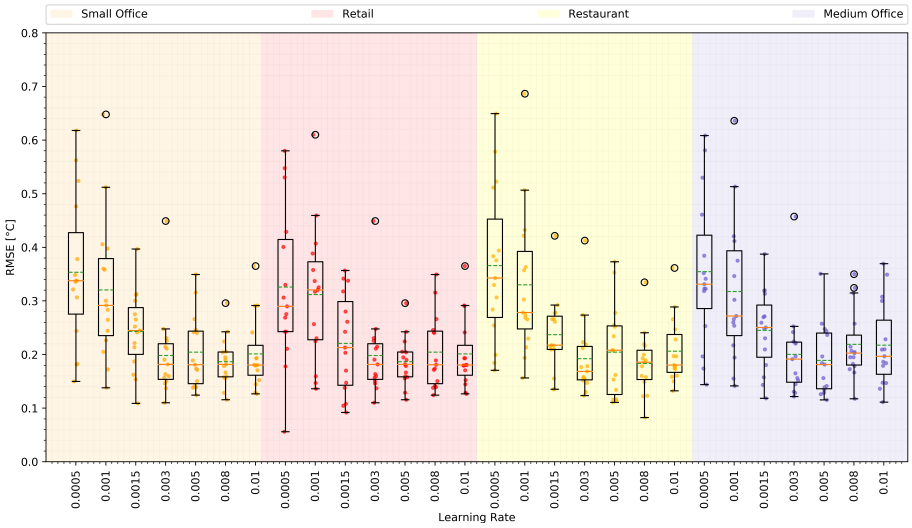

Figure 3.17: RMSE Box plot, hyperparameter = Learning Rate

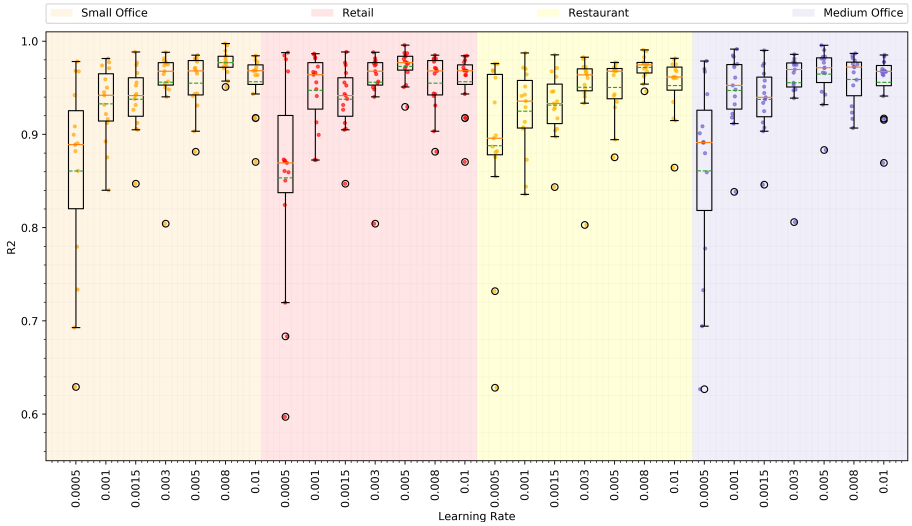

Figure 3.18: R2 Box plot, hyperparameter = Learning Rate

The results of hyperparameters tuning process is shown in the table below:

	Small Office	Retail	Restaurant	Medium Office
Batch size	100	100	100	100
n Hidden	8	8	8	50
Lookback	12	12	12	12
Learning Rate	0.008	0.005	0.008	0.005
n Layers	2	2	2	1

Table 3.7: Hyperparameters for each building model

The final LSTM configurations could be summarized as follow:

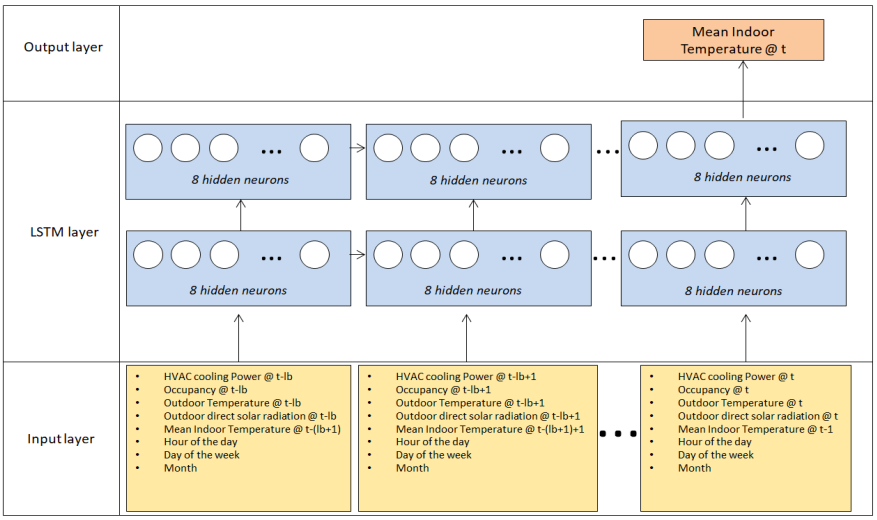

Figure 3.19: Small Office, Retail and Restaurant LSTM architecture

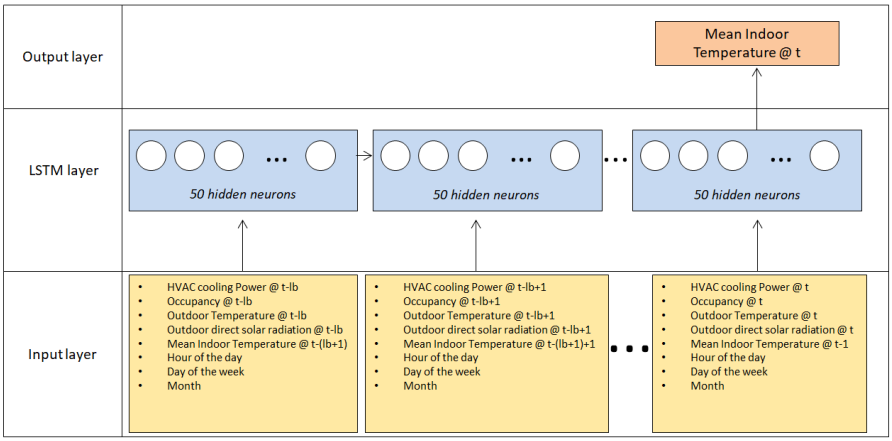

Figure 3.20: Medium Office LSTM architecture

As previously explained, this configuration is called *Many To One*: 12 previously measured sets of eight sample variables are sequentially inputted in order to predict the next indoor air temperature. The output temperature is the result of a linear transformation that is applied to the last hidden state, in order to pass from n *hidden* dimension (8 or 50) to output dimension (1).
Each model was trained for 100 epochs and the trend of the loss function, computed both on training and validation dataset, was monitored; this monitoring is useful in order to avoid *underfitting* or *overfitting*. Underfitting occurs when a model is too simple (informed by too few features or regularized too much) which makes it inflexible in learning from the dataset. An underfit model is a model that fails to sufficiently learn the problem and performs poorly on a training dataset and does not perform well on a holdout sample. On the other hand, an overfit model learns the training dataset too well, it performs well on the training dataset but does not perform well on a hold out sample. The following learning curves show that the training process was effective and that the final models have a good fit with unseen data; in fact, a good fit is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values.

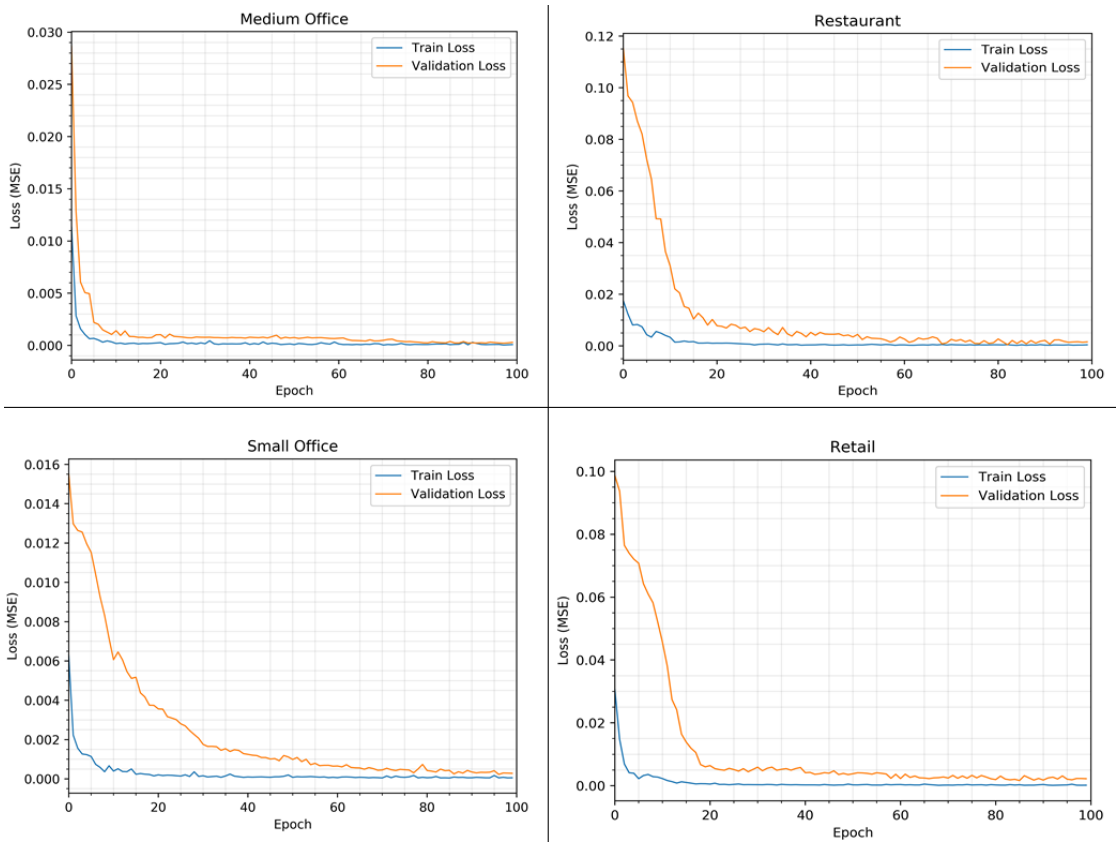

Figure 3.21: Learning Curve to diagnose underfitting/overfitting

3.4 Analysis of developed models

In this section the results of the developed models are shown and analyzed according to the previous metrics. The aim of the entire procedure is to develop a model which is able to do a multi step forecast starting from June to the end of September. Four different strategies are generally used to perform multi step predictions:

- Direct Strategy: this method involves developing and training a separate model for each forecast time step. The input vector is the same to all the predictors. However this approach is unfeasible as the number of time steps to be forecasted increases beyond the trivial.
- Multiple input-Multiple Output (MIMO): this strategy involves developing one model that is capable of predicting the entire forecast sequence in a one-shot manner and, differently from the previous case, the output of the model is not a scalar but a vector. An example of this method is Seq to Seq LSTM.
- Recursive Strategy: it involves using a one-step model multiple times where the prediction for the prior time step is used as an input for making a prediction on the following time step. More in detail, a single model is trained to perform a one-step ahead forecast given the input sequence. Subsequently, during the operational phase, the forecasted output is recursively fed back and considered to be the correct one.
- Direct-Recursive Hybrid Strategy: is a combination of the first and the third strategies. A separate model must be constructed for each time step to be predicted, but each model may use the predictions made by models at prior time steps as input values.

Given the considerable computational requirements of LSTMs during training and the necessity to integrate the models into CityLearn environment (where the Cooling load is decided time step by time step by RL agent), the work was focused on multi-step forecasting through Recursive strategy. This is a great difference with respect to standard forecasting problems, since there is the necessity to maintain as low as possible the error accumulation during time. For this reason, during the test phase, a model was considered acceptable if the MAPE error across the considered months was lower than 1.5%. In the following page is shown a conceptualization of the recursive strategy used in this thesis work.

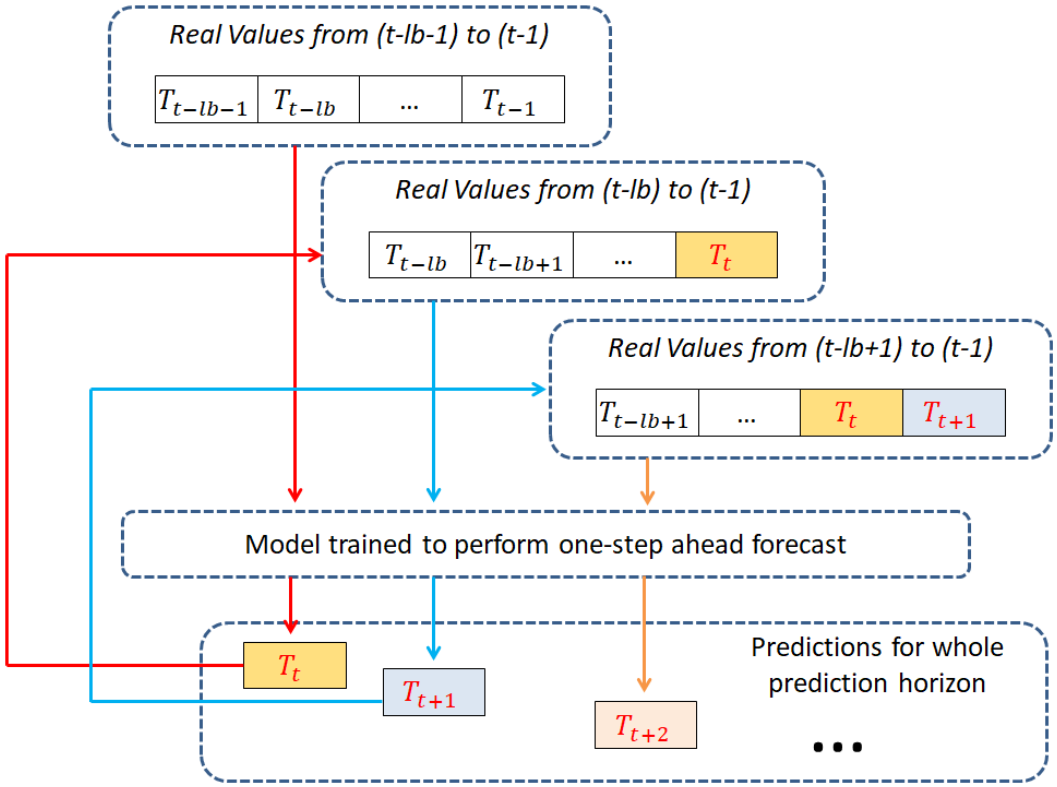

Figure 3.22: Recursive approach adopted to perform multi step forecasts

3.4.1 Test Results: Small Office

Here are presented the results for Small Office Model:

	One step ahead model	Recursive Model
MAPE %	0.422	0.696
RMSE [°C]	0.145	0.253
R2	0.98	0.94

Table 3.8: Model Evaluation Metrics: Small Office

The following figures show the results both for the one step ahead prediction and for the recursive approach:

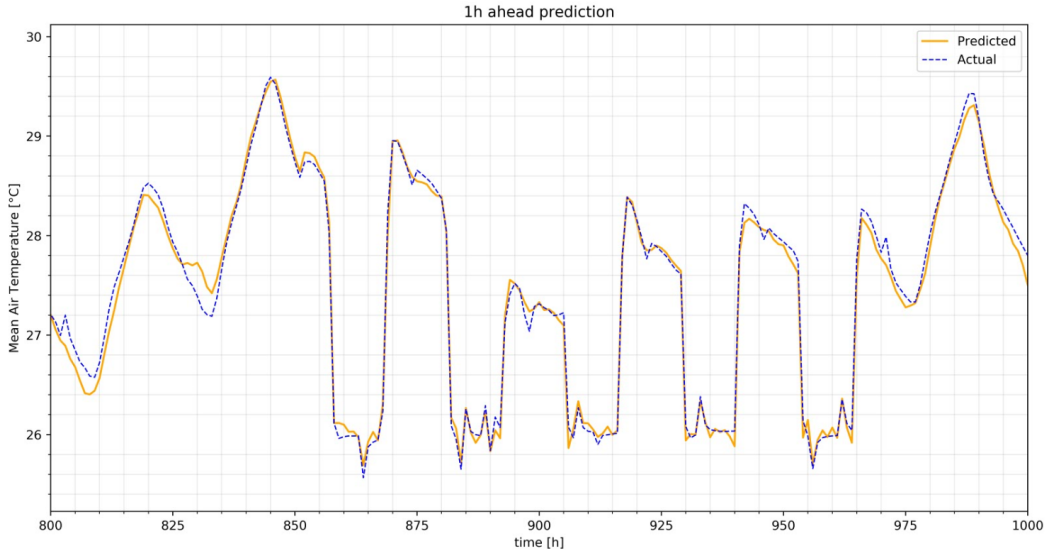

Figure 3.23: Small Office: detail of simulated mean indoor temperature and 1 step ahead forecasted mean indoor temperature

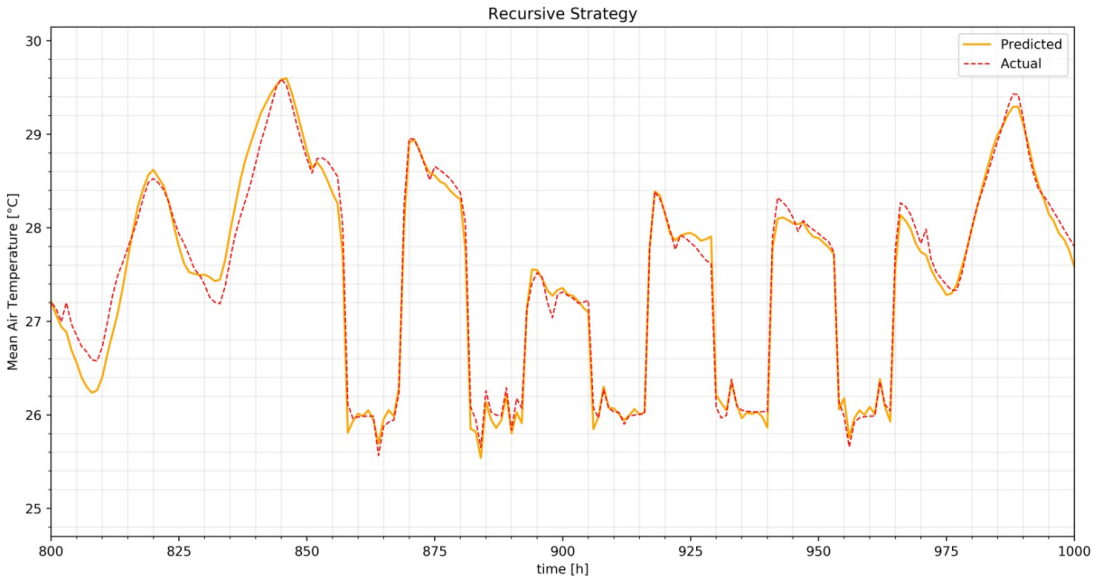

Figure 3.24: Small Office: detail of simulated mean indoor temperature and forecasted mean indoor temperature through recursive strategy

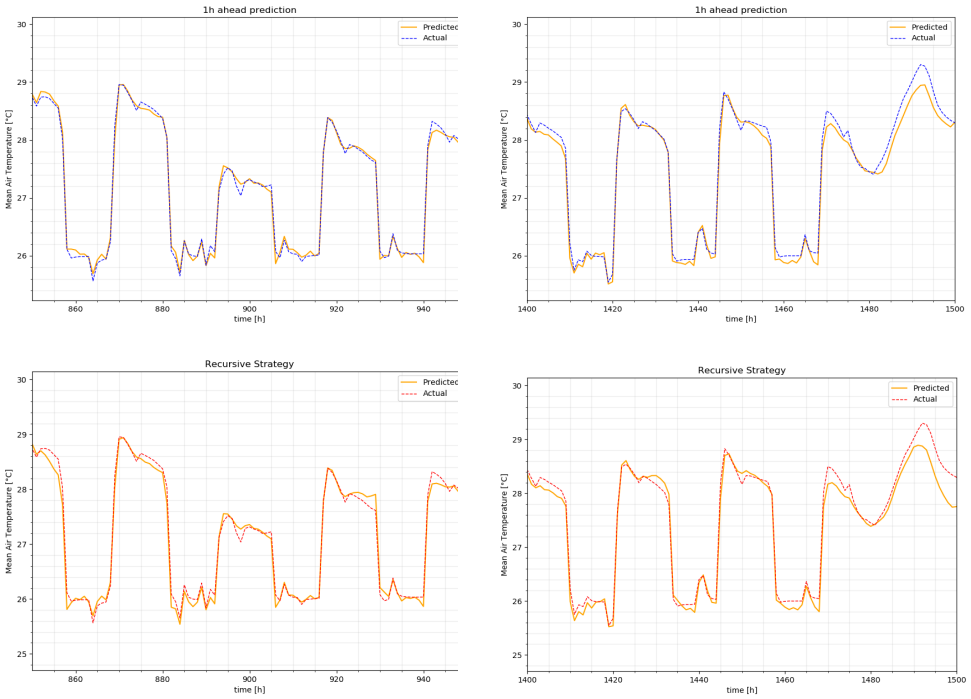

Figure 3.25: Small Office: comparison of 1 step ahead and recursive predictions

It is possible to observe that there are not substantial differences between the 1 step ahead approach and the recursive one: this means that the training was very effective and the error accumulation does not cause the model to diverge. Another observation is that the model captures very well the effects that the variation of cooling load (with respect to the ideal one) has on the indoor temperature, since the forecasted one follow very accurately those computed by EnergyPlus. Moreover it is interesting to observe the relative error distribution and the compare the distribution of recursively forecasted temperature and simulated one:

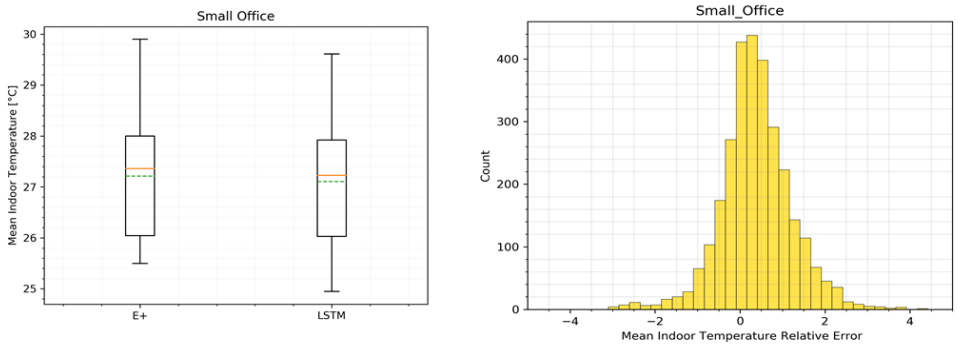

Figure 3.26: Box plot of simulated and recursively forecasted mean indoor temperature and distribution of relative error

In order to understand how much the error between the forecasted temperature and the ideal one influences the cooling energy consumption, another interesting analysis was performed. The forecasted mean indoor temperature was used as a set point for the Ideal HVAC thermostat; since the forecasted indoor temperature is the results of a weighted average on conditioned thermal zones volume, the thermostat was set to each conditioned zone in the building. The following figures show the distribution of the relative error calculated between the test cooling power and the one calculated by means of thermostat and the box plot which highlights cooling power distribution.

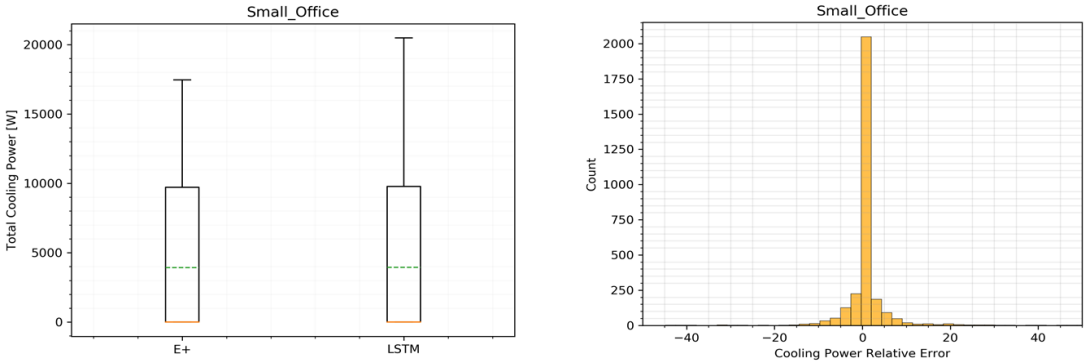

Figure 3.27: Box plot of test cooling power and that calculated through thermostat object and distribution of relative error

What could be seen from this figures is that the distribution of the cooling power is very close to the original one; moreover, the relative error has a Gaussian shape centered in zero. The result of this analysis is that the cooling energy computed considering temperature predictions is equal to 11.49 [MWh] against 11.58[MWh], which means a relative error of 0.83% In the following pages are shown the results for the other building models

3.4.2 Test Results: Retail Stand Alone

	One step ahead model	Recursive Model
MAPE %	0.163	0.448
RMSE [°C]	0.058	0.15
R2	0.992	0.944

Table 3.9: Model Evaluation Metrics: Retail Stand-Alone

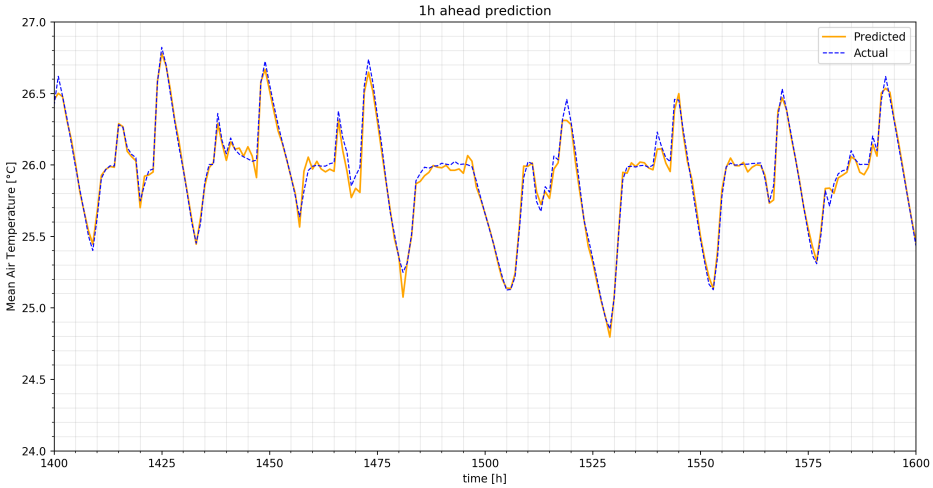

Figure 3.28: Retail Stand Alone: detail of simulated mean indoor temperature and 1 step ahead forecasted mean indoor temperature

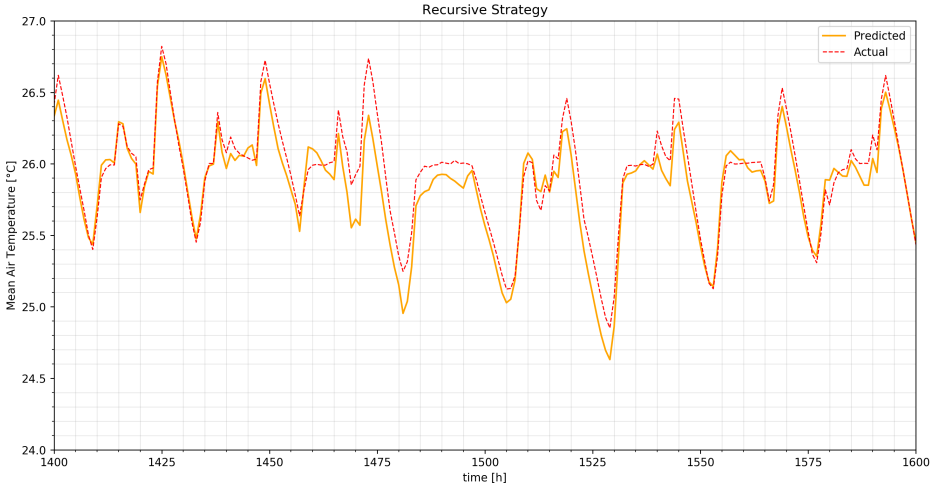

Figure 3.29: Retail Stand Alone: detail of simulated mean indoor temperature and forecasted mean indoor temperature through recursive strategy

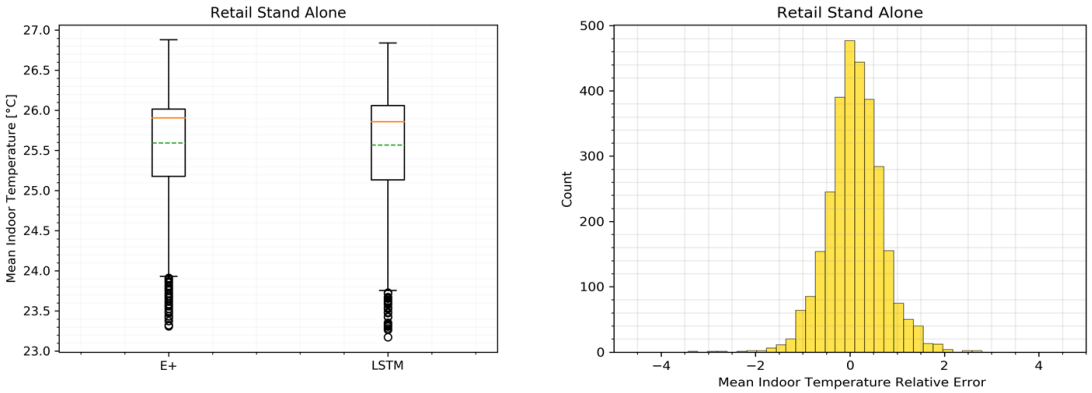

Figure 3.30: Retail Stand Alone: Box plot of simulated and recursively forecasted mean indoor temperature and distribution of relative error

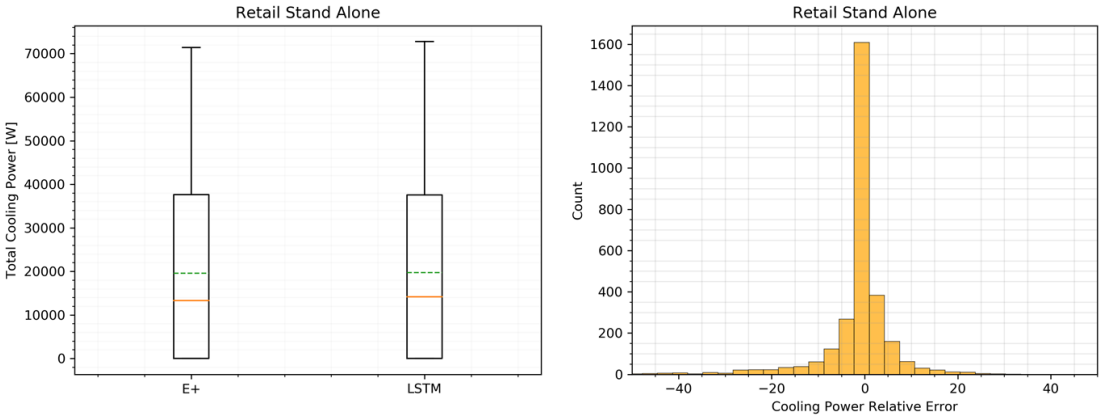

Figure 3.31: Retail Stand Alone: Box plot of test cooling power and that calculated through thermostat object and distribution of relative error

As could be seen from these figures, also the Retail Stand Alone model is very accurate, temperature and cooling distributions are very close to the original one and, in both cases, the distribution of error has a Gaussian shape centered in zero. In this case the cooling energy computed considering temperature predictions is equal to 57.69 [MWh] against 57.31[MWh], which means a relative error of 0.67% The accuracy of the model is also demonstrated from the computed metrics: one step ahead predictions are extremely close to the simulated values; however, also recursive forecasts are very accurate. Also in this case, error accumulation is avoided and limited as much as possible. It is important to highlight that having a very accurate 1 step ahead prediction model is a necessary condition to have accurate predictions with recursive strategy.

3.4.3 Test Results: Restaurant

	One step ahead model	Recursive Model
MAPE %	0.489	0.775
RMSE [°C]	0.170	0.258
R2	0.972	0.934

Table 3.10: Model Evaluation Metrics: Restaurant

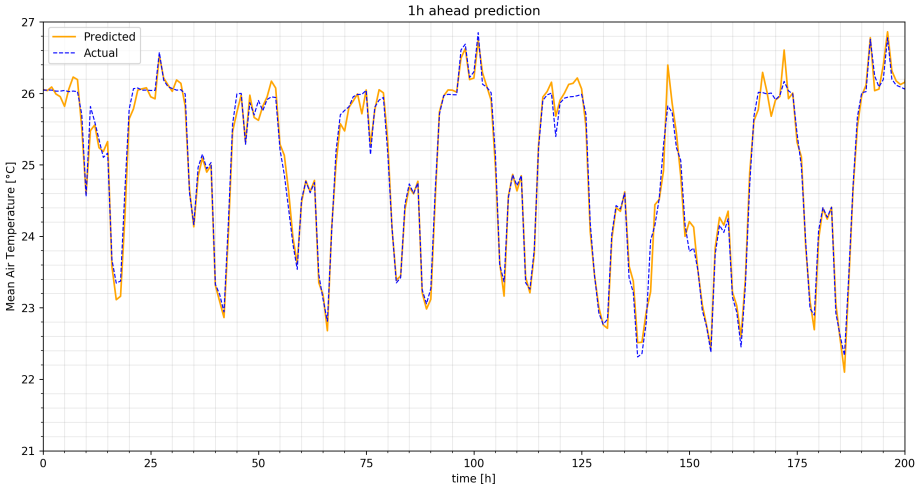

Figure 3.32: Restaurant: detail of simulated mean indoor temperature and 1 step ahead forecasted mean indoor temperature

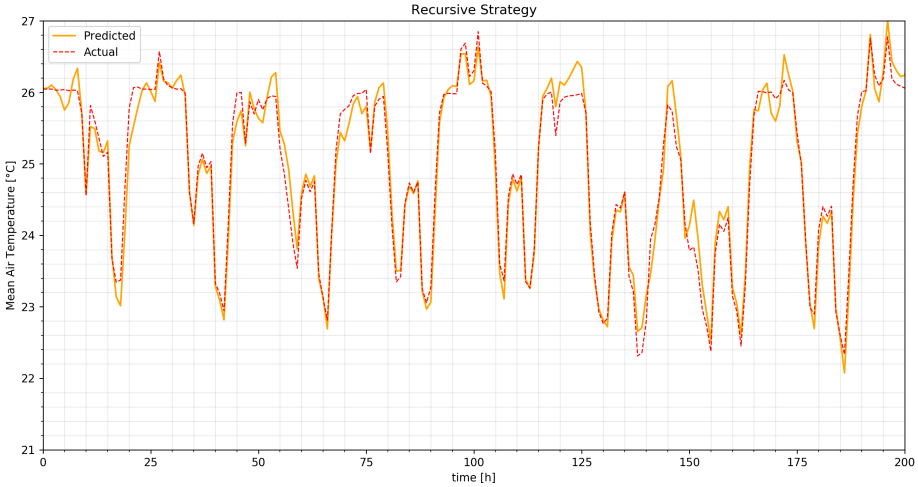

Figure 3.33: Restaurant: detail of simulated mean indoor temperature and forecasted mean indoor temperature through recursive strategy

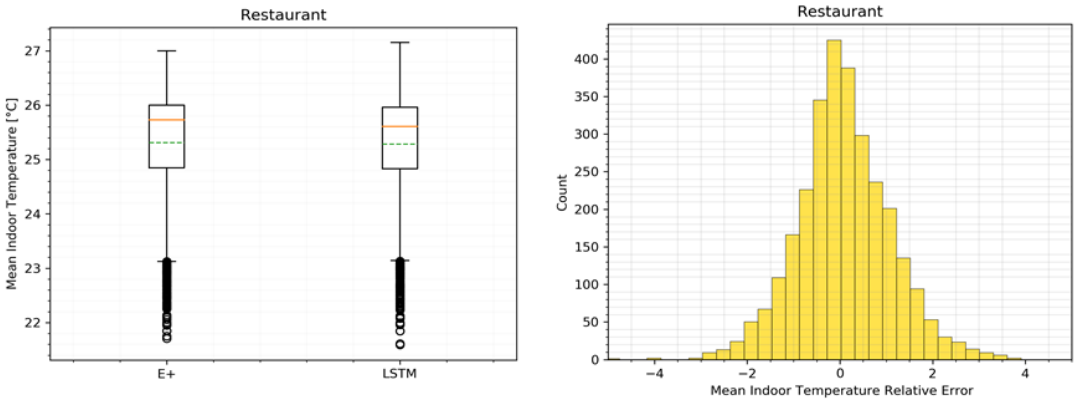

Figure 3.34: Restaurant: Box plot of simulated and recursively forecasted mean indoor temperature and distribution of relative error

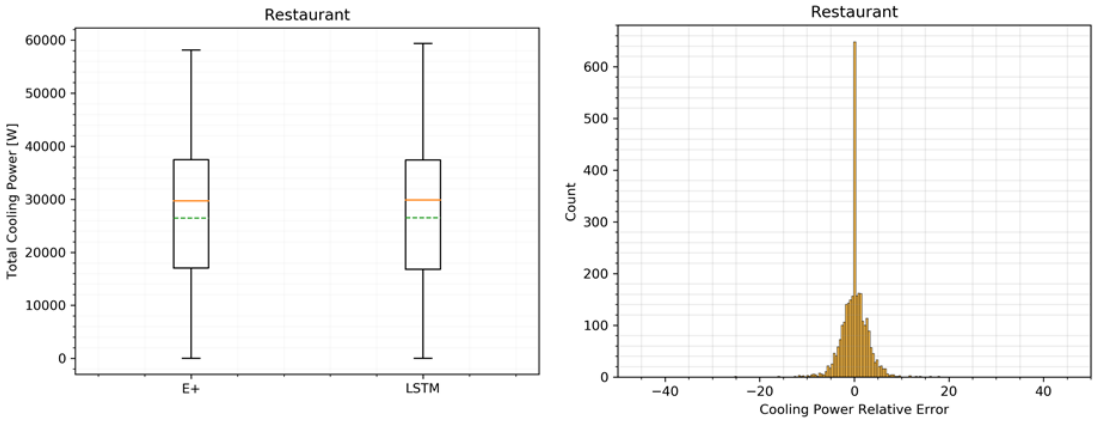

Figure 3.35: Restaurant: Box plot of test cooling power and that calculated through thermostat object and distribution of relative error

Box plots indicate a very similar distribution of both temperature and cooling power, also in this case the distribution of the relative error follows a Gaussian centered in zero. In this case, the cooling energy computed considering temperature predictions is equal to 77.64 [MWh] against 77.46[MWh], which means a relative error of 0.23%.

3.4.4 Test Results: Medium Office

	One step ahead model	Recursive Model
MAPE %	0.154	0.810
RMSE [°C]	0.065	0.279
R2	0.997	0.936

Table 3.11: Model Evaluation Metrics: Medium Office

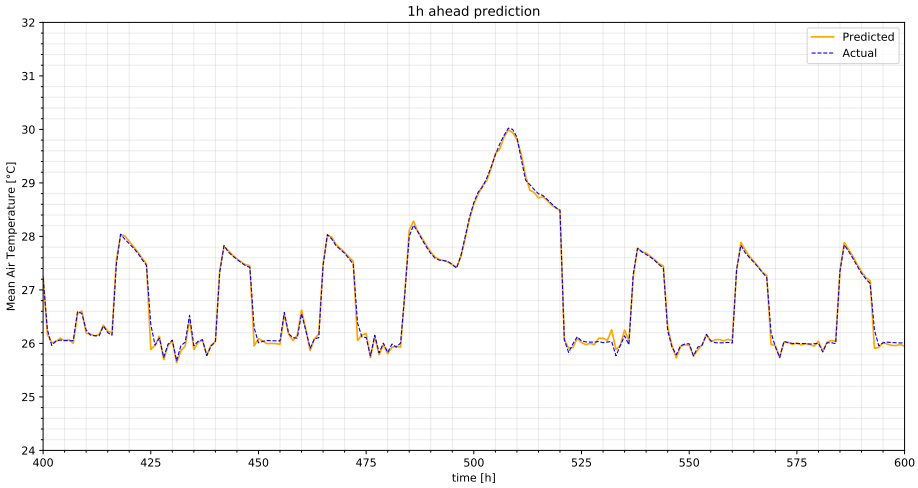

Figure 3.36: Medium Office: detail of simulated mean indoor temperature and 1 step ahead forecasted mean indoor temperature

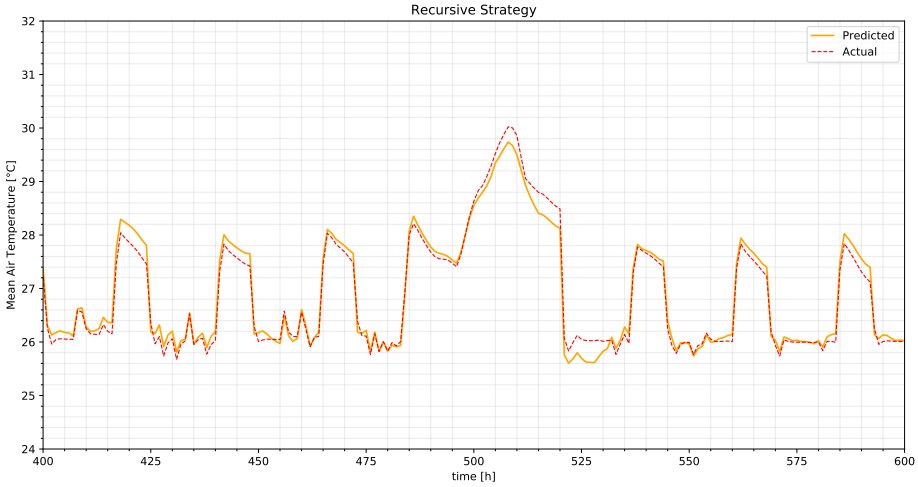

Figure 3.37: Medium Office: detail of simulated mean indoor temperature and forecasted mean indoor temperature through recursive strategy

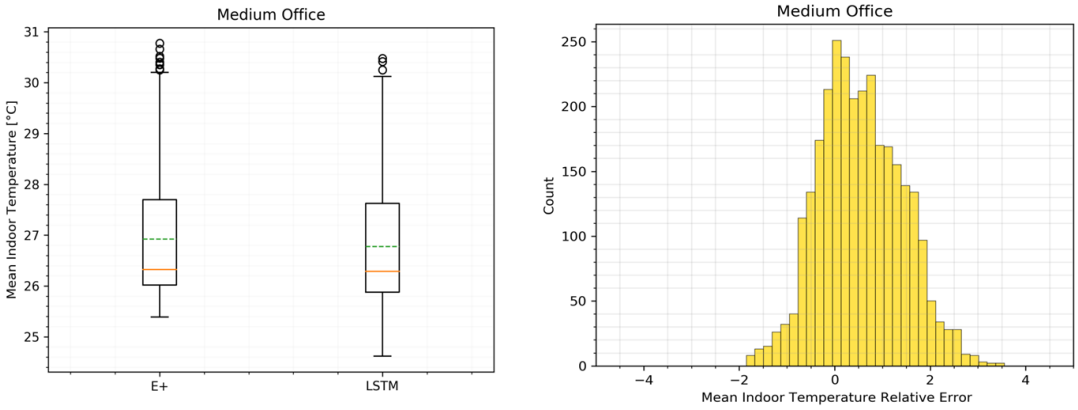

Figure 3.38: Medium Office: Box plot of simulated and recursively forecasted mean indoor temperature and distribution of relative error

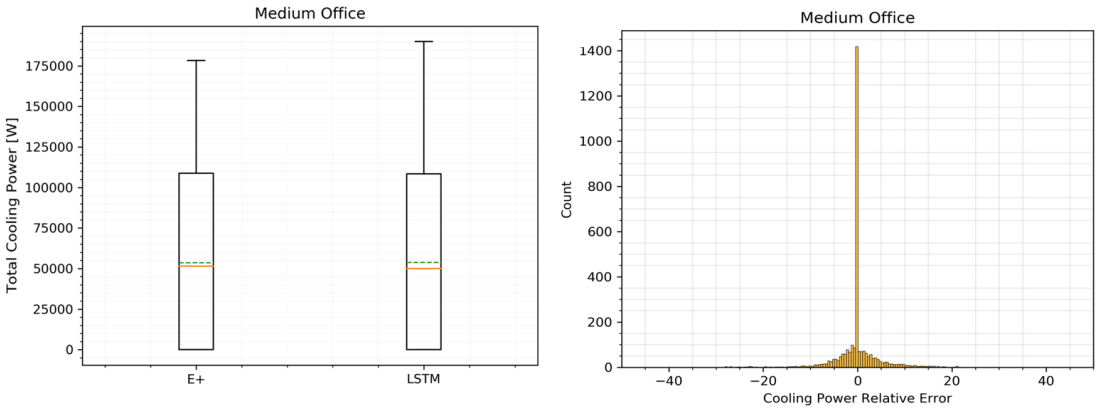

Figure 3.39: Medium Office: Box plot of test cooling power and that calculated through thermostat object and distribution of relative error

Also in this case, the model is very accurate and follows very well the simulated temperature profile; it is important to notice that both the resulting temperature and cooling power distributions are very similar to the original ones: the interquartile distribution is almost the same, despite the minimum LSTM temperature is lower than the one computed by E+. The cooling energy computed considering temperature predictions is equal to 157.518 [MWh] against 156.615[MWh], which means a relative error of 0.577%.
The results of these analysis show that the difference of cooling power due to the error of LSTM predictions is very low, with a relative error on total energy consumption that is maintained well below the threshold of 1%.
The aim of this entire process is to develop data-driven models (based on artificial neural networks) which are able to simulate the thermal-dynamic behaviour of considered buildings. The final goal is to coordinate the energy consumption of a cluster made by four buildings through an innovative control strategy based

on single agent reinforcement learning. The data-driven approach has two main advantages:

- the first one is to avoid physic-based energy simulations that, despite their accuracy, require detailed input, models calibration and increase the computational costs.
- The second is that through data-driven approach is possible to capture the real dynamic of the buildings, through data collected by sensors. In this way, by correctly processing those data, it is possible to build a model that reflects the real behaviour of the buildings.

To evaluate the reinforcement learning agent, the developed models are integrated into CityLearn; in this way the controller would have the possibility to exploit demand response actions not only by controlling the charge/discharge of thermal storages, but also by modulating the HVAC cooling power. Demand response, in fact, is a very interesting technique which has the potential of reducing buildings electricity demand peaks by about 20%. However, buildings are very difficult to model and coordinate, especially thanks to their complex dynamic (due to almost stochastic occupants' behaviour, refurbishment measures, complex thermal-dynamic). RL has the potential to overcome this problems, since it is adaptive, model free and could learn from historical and real time data, through the interaction with its surrounding environment. A lot of research has been conducted in the use of RL to exploit DR actions, however, there is a lack in experiments' reproducibility and standardization. CityLearn [23] is an OpenAI Gym environment that is created to address these issues: its main objective, in fact, is to facilitate and standardize the evaluation of RL agents, enabling an easier comparison of different algorithms. CityLearn allows to control the storage of domestic hot water (DHW), and chilled water. It also includes energy models of air-to-water heat pumps, electric heaters, and the pre-computed energy loads of the buildings, which include space cooling, dehumidification, appliances, DHW, and solar generation. Some previous works have been conducted exploiting CityLearn framework, in [24] the authors implemented the state of the art algorithm *Soft Actor Critic* (SAC) deep reinforcement learning (DRL) agent in order to handle continuous action space (for a detailed explanation of reinforcement learning algorithms is suggested to see chapter 4). The centralised agent was able to effectively control thermal storages, highlighting the potential application of DRL as a plug-and-play controller style. Furthermore, Pinto et al. [25] implemented the SAC algorithm to control a cluster of 4 buildings in order to asses the ability of the DRL agent in reshaping and flattening the electricity consumption curve. The main contribution of this work is that they took into account the influence that the outdoor temperature has on the COP and on the declared capacity (DC) of the heat pump (HP): in CityLearn, in fact, these two parameters are kept constant, which is unreal in a practical implementation. The results of this work show how the DRL agent is effectively able to flatten the load profile and shift the charge of the storages between the different buildings, optimising both energy consumption and costs. Moreover, in [26], the authors implemented a distributed control solution, which consists of a central load

aggregator that optimizes system-level objectives and building-level controllers that track the load profiles planned by the aggregator. The proposed approach was evaluated through CityLearn simulation environment across four climate zones in four nine-building clusters; results demonstrated that was possible to reduce costs of an average equal to 16.8% with respect to a standard RBC controller. Another very interesting work is performed by Canteli et. al. [27], they implemented a multi agent reinforcement learning controller in CityLearn. The agents were evaluated on the average of five normalized metrics (normalization is performed according to RBC performances) and results outperformed both independent/uncooperative reinforcement learning agents and manually optimized RBC. As previously said, the main contribution of this thesis work laid in the fact that the centralized controller has the possibility to control also the cooling power provided by HVAC system: this possibility introduce an additional degree of freedom in terms of demand response, however this increase the complexity of the control problem, since the agent has to choose at least 2 actions per each building (depending on the number of storages per each building).

Chapter 4

Reinforcement Learning overview

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. In other terms, Reinforcement Learning could be defined as the science of decision making, it is a fundamental science which tries to understand the optimal way to make decisions; in engineering, reinforcement learning is applied to the field of optimal control, which is the branch that deals with finding the best sequence of actions to efficiently control a system. The main difference between Reinforcement Learning and Supervised Learning is that the former is characterized by the absence of supervision; the paradigm is completely different: the actions to be taken are not known a priori but are learned by *trial and error* through a reward signal. Moreover, another distinction is that feedbacks related to the quality of the actions could not be instantaneous, but delayed. Furthermore, in RL, time is an essential feature: it deals with sequential decision making processes in which an agent has to pick decisions time step by time step. The agent has to take actions which influence the surrounding environment, in this way the data received by the agent are influenced by its actions: this paradigm is known as *active learning process*.

4.1 The Reinforcement Learning Problem

A Reinforcement Learning problem is mainly characterized by:

- a reward R_t
- an action A_t
- an Observation O_t

The reward is a scalar feedback signal that describes how well the agent is performing at time step t: the aim of the agent is to get as much reward as possible, in other

terms it has to maximize the cumulative reward. The goal of the agent, in fact, is to select actions in order to maximize the total future reward; since actions may have long term consequences and reward may be delayed, it is important that the agent learns to plan ahead.

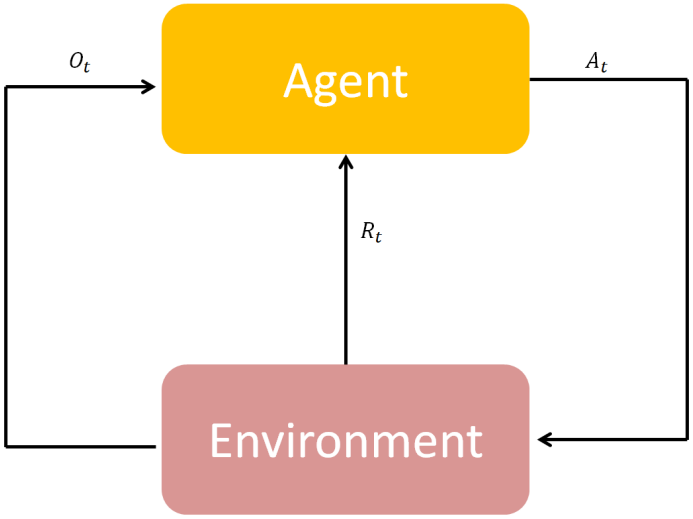

Figure 4.1: Representation of a Reinforcement Learning Problem

A key concept in RL is the state: it is a summary of the information used by the agent to determine its next action; formally, the state is a function of the *history*, where history is a sequence of previous actions, observations and rewards. The *state* could have different definitions:

- Environment state S_t^e which is the environment's internal representation, is represented by all the data the environment uses to determine the next observation and reward. Generally, the agent does not see the environment state
- Agent state S_t^a which is the agent's internal representation, is the information the agent takes to determine the next action.

Mathematically, the state is known as *Information State* or *Markov State* and is defined starting from the *Markov Property*:

$$P\Big[S_{t+1} \mid S_t\Big] = P\Big[S_{t+1} \mid S_1, ..., S_t\Big]$$

which states that the probability of the next state conditioned to the current state is the same is equal to the probability of the next state conditioned to all the previous states. This means that the probability of the next state is only influenced by the current state, the future is independent from the past given the present. In *Fully Observable* problems, the agent directly observe the environment state and

$$O_t = S_t^a = S_t^e$$

However, fully observable environments are uncommon, generally RL problems are characterized by a partial observability, where the agent can only partially observe

the environment, in this case the agent state is different from the environment state.

4.1.1 The RL Agent

A reinforcement learning agent could include one or more of these components:

- Policy: it is a function that describes the behaviour of the agent. it is a map from states to actions. There are two kinds of policy, deterministic where $a = \pi(s)$ and stochastic, which is useful to make random exploratory decisions and see more of the state space $a = \pi(a|s) = P[A = a|S = s]$
- Value Function: it is a prediction of the expected future reward starting from a particular state

$$v_\pi(s) = E_\pi\Big[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... \mid S_t = s\Big]$$

- Model: it is used to predict the evolution of the environment; there are two kinds of model:
  - Transition models which predict the next state, they are used to model the dynamics of the environment
  - Rewards model which predict the next immediate reward

According to these tree components, it is possible to categorize the RL agent; in particular we can distinguish in :

- Value Based algorithms if the agent is only characterized by a value function and the policy is implicit: the policy is not necessary since it is possible to pick actions only by acting greedily with respect to the value function
- Policy based algorithms where there is only an explicit representation of the policy and an implicit representation of value function
- Actor Critic algorithms which are a combination of the previous two algorithms, in this case the agent stores both the policy and the value function
- Model Free algorithms which are characterized by the absence of a model. They could store both policy and value function or explicitly represents only one of these two functions.
- Model Based algorithms which are similar to model free except for the presence of a model of the environment.

These concepts are well summarized in the following figure:

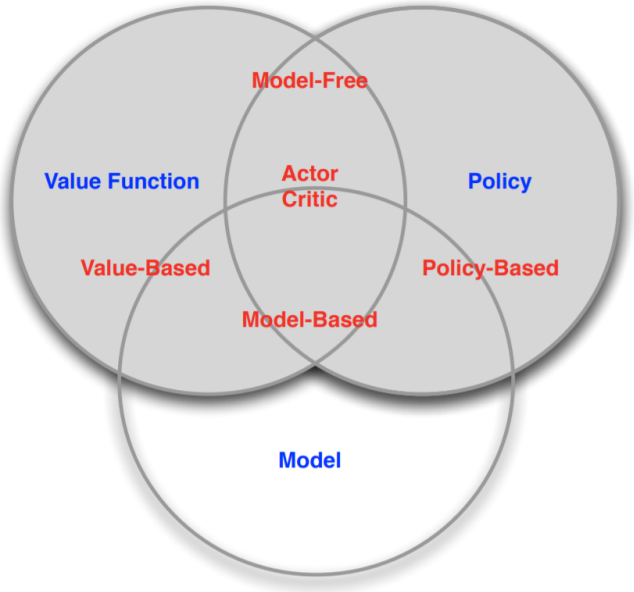

Figure 4.2: RL agent taxonomy [28]

4.2 Markov Process

A Markov process could be described as a sequence of random states where the probability of end up in another state given a current state is not influenced by previous states (memoryless random process). A Markov process could be mathematically described by a tuple which contains a set of states S and and the state transition probability matrix P. This matrix describes the probability to end up in all the other possible states given the current state. The set of states and probabilities is enough to fully describe the dynamic of the system.

4.2.1 Markov Reward Process

A Markov reward process (MRP) is a Markov process with value judgments which indicate how much reward could be accumulated across a particular sequence of states sampled from the Markov process. The difference between a Markov process is the presence of the *Reward Function* R and the discount factor γ. The former describes the immediate reward associated to a particular state. In a Markov reward process, an important quantity is described by the return G_t, which is the total discounted reward from time step t:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

The discount factor γ is a value between 0 and 1 that discounts the future rewards. γ accounts for the fact that could be preferable to have short term reward instead

of delayed reward. The closer γ is to zero, the more immediate reward is important, the closer is to 1, the more delayed rewards are important. The Markov reward process is strictly related to the *value function v(s)*: the state value function of an MRP is the expected return staring from state s:

$$v(s) = E\Big[G_t \mid S_t = s\Big]$$

4.2.2 Bellman Equation for MRPs

The Bellman Equation describes one of the most fundamental relationships in RL, it is based on the fact that the value function can be decomposed in two parts:

- immediate Reward R_{t+1}
- discounted value for successor state $\gamma v(S_{t+1})$

$$\begin{aligned} v(s) &= E\Big[G_t \mid S_t = s\Big] \\ &= E\Big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \mid S_t = s\Big] \\ &= E\Big[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...) \mid S_t = s\Big] \\ &= E\Big[R_{t+1} + \gamma G_{t+1} \mid S_t = s\Big] \\ &= E\Big[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\Big] \end{aligned}$$

.

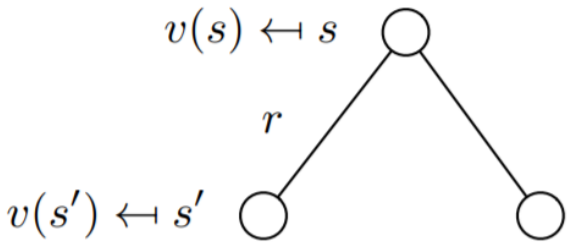

Figure 4.3: Bellman equation scheme

4.2.3 Markov Decision Process

A Markov Decision Process (MDP) is an MRP which includes also decisions in the form of actions. MDPs formally describe an environment for reinforcement learning [29]. In an MDP all states are Markov and the transition probability matrix is influenced by actions. To formalize how actions are taken is necessary to introduce the concept of *policy*: a policy π is a distribution over actions given states.

$$\pi(a|s) = P[A_t = a \mid S_t = s]$$

A policy fully defines the behaviour of an agent. After have defined the policy, it is possible to introduce the *state-value function* which is denoted as $v_\pi(s)$ and represents the expected return when starting in s and following π thereafter [10]; in other terms it says how good is for the agent to execute a given action given a particular state.

$$v_\pi(s) = E_\pi[G_t \mid S_t = t]$$

The state value function has not to be confused with another type of value function , called *action value function* which, instead, tells how convenient is for the agent to take action a in state s, under the policy π:

$$q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a]$$

Having defined this value functions, it is possible to introduce the Bellman equation in the MDP case, which is called *Bellman Expectation Equation*. Considering the state value function, the Bellman expectation equation could be written as:

$$v_\pi(s) = \sum_a \pi(a|s)\ q_\pi(s, a)$$

On the other hand, for the action value function, the Bellman expectation equation is:

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s')$$

By putting these two equations together it is possible to redefine the state value function:

$$v_\pi(s) = \sum_a \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s') \right)$$

The formula could be understood by looking at the following back up diagram:

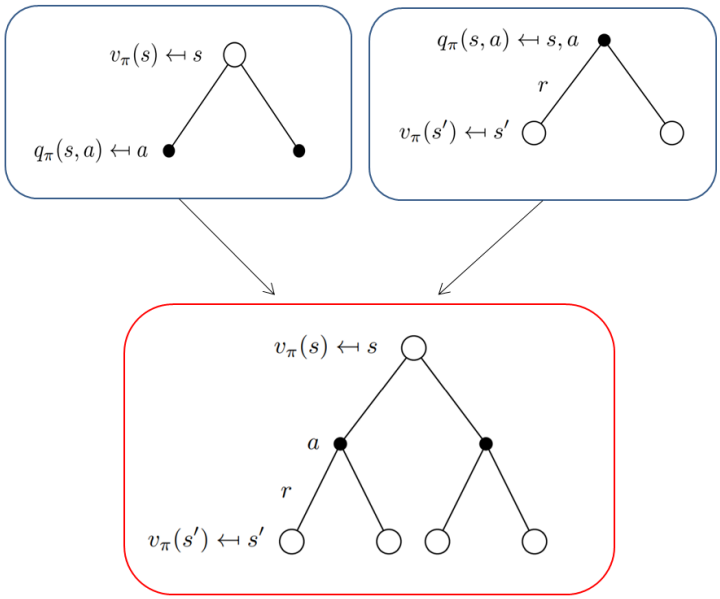

Figure 4.4: Backup Diagram for State-Value Function

The definition of the optimal policy is straightforward; in fact, a policy π is better that another policy π' if the expected return of the former is greater then the latter's:

$$\pi > \pi' \iff v_\pi(s) > v_{\pi'}(s)$$

The optimal state-value function $v_*(s)$ is the maximum value function over all policies:

$$v_* = max\ v_\pi(s)$$

In the same way, the optimal action-value function $q_*(s)$ is the maximum action-value function over all policies:

$$q_* = max\ q_\pi(s)$$

4.3 Dynamic Programming

Dynamic programming (DP) is an optimization method for sequential problems, in fact, the term dynamic accounts for the temporal component of the problem while the term programming accounts for the optimization of a program, in this case, identified by the policy. The great advantage of DP is that it is a very general solution method for problems characterized by sub-optimal substructure and that can be divided into overlapping sub-problems, so once one of them is solved, the solution can be stored and later reused, since they recur many times. The same two proprieties are completely satisfied by a MDP, in particular, the Bellman equation gives recursive decomposition while the value function stores and reuses solutions. To be specific, a DP algorithm is used to solve a problem in which there is the full knowledge of the MDP . This problem is known as *Planning* and it can be divided in two parts:

- Prediction: given a MDP or MRP and a policy π as input, the objective is to evaluate the value function v_π i.e. to demonstrate how good is the input policy.
- Control which has as input an MDP and as output the optimal value function v_* and the optimal policy $\pi_*(s)$

To solve a prediction problem it is possible to follow an iterative procedure trough the iteration of the Bellman expectation equation; however, this approach requires too much computational effort. A better idea is to use an approach called *Synchronous backup* in which, for each iteration $k+1$, for all states $s \in S$, $v_{k+1}(s)$ is updated starting from $v_k(s')$, where s' is the state after s.

$$v_{k+1}(s) = \sum \pi(a|s)\Big(R_s^a + \gamma \sum P_{ss'}^a v_k(s')\Big)$$

Basically for each state s at time step $k+1$, the new value function is computed starting from the old value functions of the future states; the algorithm converges as $k \to \infty$. In a control problem, the aim is to improve a given policy up to the optimal policy. The first step of this procedure is to evaluate the initial policy, which means to figure out the value function for that function. The second step is to improve the current policy by acting greedily with respect to the current value function.

$$\pi' = greedy(v_\pi)$$

This procedure is called *Policy Iteration* and is proven to converge to the optimal policy.

$$\pi'(s) = argmax\ q_\pi(s,a) = argmax\ E_\pi\Big[R_{t+1} + \gamma v_\pi(S_{t+1}|S_t = s, A_t = a\Big]$$

As soon as the policy π is improved, the new policy π' is improved again to obtain a new better policy π''. This procedure perpetuates up to the point where the optimal policy π_* and the optimal value function $v_*(s)$ are found.

4.4 Model Free Prediction

Previously was demonstrated how DP could be a very powerful tool to solve prediction and control problems; however, its biggest limit lays in the fact that the environment has to be fully known. In real world problems this complete knowledge is almost impossible to achieve (e.g. the transition probability matrix is almost impossible to be known a priori). RL algorithms try to provide a solution for both prediction and control, but completely model-free, by learning through a real or simulated experience

4.4.1 Monte-Carlo Reinforcement Learning

Monte Carlo (MC) methods learn directly from episodic experiences, it is model free since it does not require any knowledge of MDP transitions and rewards. Since it learns only from complete episodes it is impossible to use bootstrapping. Starting from a particular state and following a policy π, MC policy evaluation uses empirical mean return of expected return: for each state, the method involves collecting as many samples as possible in order to compute the mean of various collected expected returns. Each occurrence of a state during the episode is called visit. The concept of visit is important because it allows to categorize two different MC approaches:

- First-Visit MC policy evaluation which is defined as the average of the returns considering only the first visit to s, within a given episode: a counter takes into account how many times a state is encountered (taking in to account only the first time in each episode) while the total return takes into account the sum of the return G corresponding to the first visit per each episode. The value is estimated dividing the total return by the counter and, by the law of large numbers, the value function is proven to converge to the value function for the corresponding policy.
- Every-Visit MC policy evaluation which, differently from the previous approach, considers each time a state is encountered to update the counter and the total return. The value function is then computed in the same way.

Another way to compute the value function is based on the *incremental mean* in which the mean is updated in the direction of the error between the old and the new mean. Each time a state S_t with return G_t is encountered, the following computations are performed:

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\Big(G_t - V(S_t)\Big)$$

It is also possible to substitute $\frac{1}{N(S_t)}$ with a constant step size α which is called *learning rate*

4.4.2 Temporal-Difference Learning

It differs from MC learning since Temporal-Difference (TD) learns from incomplete episodes, by *bootstrapping*. TD can learn online every time step, while MC must wait until the end of an episode before knowing the return: for this reason, MC cannot be used in continuous environments where there is not an end of an episode. The simplest temporal-difference learning algorithm is called TD(0) and updates the value $V(S_t)$ towards the estimated return $R_{t+1} + \gamma V(s_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha\Big(R_{t+1} + \gamma V(s_{t+1}) - V(S_t)\Big)$$

The immediate Reward R_{t+1} and the discounted value of value function $\gamma V(s_{t+1})$ in the next step are estimated and used to replace the actual Return G_t. This concept is called *bootstrapping*: the original guess is updated from the subsequent guess.

4.4.3 TD(λ) and eligibility trace

As previously analyzed, in TD(0) target depends just on the estimate of the value function at the next step; however nothing prevents us to look ahead for the n next steps to perform the update. In n-step TD, the target is computed as the n-step return:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

this defines the n-step temporal-difference learning

$$V(S_t) \leftarrow V(S_t) + \alpha\Big(G_t^{(n)} - V(S_t)\Big)$$

Different n generates different results, it is necessary to find an algorithm that allows to combine information from all time steps without increasing the complexity of the problem. This algorithm is called TD(λ) where the λ is a weight used to combine all the n-step returns $G_t^{(n)}$. $G_t^{(\lambda)}$ is called λ return and is the geometrically weighted average of all n.

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^n$$

Actually, G_t^{λ} represents the target in forward-view TD(λ), since the utility is computed looking into the future, and as it happens in MC, it requires the ending of the episode. The other type of TD(λ)is called backward-view TD(λ)and it exploits the concept of *eligibility trace*. The idea of TD(λ) with eligibility trace is to update the utility function by considering not only the next step, but also the previous states visited during the current episode. It assigns credit to both the most frequent states and the most recent states.

4.5 Model Free Control

Differently from prediction, where the objective is to estimate the value function of an unknown MDP, a control aims to optimise the value function of an unknown MDP. As previously discussed, a control problem aims to find the optimal policy and RL takes over DP when the MDP is unknown or is known but too big to be used. Model free control is characterized by two different types of learning:

- On-policy learning, where actions are sampled from a certain policy π and, at the same time, the same policy is evaluated
- Off-policy learning , where the agent learns about policy π from experience acquired through another policy, μ

4.5.1 Monte Carlo evaluation

When MC method is used, a part of the algorithm has to perform the policy evaluation while the other part performs policy improvement. Here, the first problem shows up, since it is not sufficient to adopt a greedy policy improvement because of the trade-off between *exploration* and *exploitation*. To be sure that the provided policy is optimal, exploration cannot be avoided since all the states of the environment must be visited, even if they do not return the highest immediate reward. To perform greedy policy improvement is necessary to use the action value function instead of the state value function, since the latter requires a model of the MDP (to know the transition probability matrix). On the other hand, action value function enables to perform control in a model free setting.

$$\pi'(s) = argmax\ Q(s,a)$$

The simplest way for ensuring continuous exploration is to adopt the ϵ-greedy exploration. The greedy action is chosen according to a probability equal to $1-\epsilon$ while the random action is chosen according to a probability equal to ϵ. Taking into account m possible actions:

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon & a^* = argmax\ Q(s,a) \\ \frac{\epsilon}{m} & otherwise \end{cases}$$

According policy improvement theorem, for any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

4.5.2 SARSA

SARSA is probably one of the most known on-policy algorithm in RL field, it extends the idea of Temporal-Difference to the control problem. The basic idea is to apply TD to Q(s,a) and use ϵ-greedy policy improvement. SARSA is the acronym of *State-Action-Reward-State-Action* which well explains the concept behind this algorithm: starting from a particular state S and performing a particular action A, the agent receives a reward R and ends up in a new state S', where it picks another action A', this time sampled according to the current policy.

$$Q(S,A) \leftarrow Q(S,A) + \alpha\Big(R + \gamma Q(S',A') - Q(S,A)\Big)$$

4.5.3 Q-Learning

SARSA gives an idea of what on-policy learning means, since the agent learns about policy π through experience sampled by policy π itself. On the contrary, in off-policy learning (learning from observation) optimal policy is obtained by the observation of another policy μ that is never updated. One of the biggest advantages that makes off-policy learning a very powerful tool, lies in the fact that it gives to the agent the ability of learning an optimal policy looking to the actions taken by another agent. Furthermore, it opens the doors to the "experience replay"

of deep learning strategies, where a part of information collected in the past is re-used to enhance the current policy. The idea behind Q-learning is that, at each time step two actions are considered: the first one is the real performed action, sampled from the policy μ; the second one is an alternative action sampled from the target policy π. The action value function is then updated in the direction of the target policy:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\Big)$$

The target policy π is greedy with respect to Q(s,a)

$$\pi(S_{t+1}) = argmax\ Q(S_{t+1}, a')$$

while the policy μ is ϵ-greedy. To summarize, the Q-learning update the Q value in the direction of the maximum Q value that could be taken from the successive step. For this reason, this algorithm is also called SARSAMAX:

$$Q(S, A) \leftarrow Q(S, A) + \alpha\Big(R + \gamma\ max\ Q(S', a') - Q(S, A)\Big)$$

4.5.4 Deep Q-Learning

In simple settings, Q-learning uses lookup tables called *Q-Tables*: they are tables of n rows (where n represents the number of states) and m columns (where m indicates the number of possible actions for each state) that store Q-value for each pair of state-action. The Q-values are updated during the learning process thanks to the agent's experience. The problem is that it is not possible to store all the values when the MDP is too large (too many states and actions) or continuous. In these cases it is necessary a function approximator. As discussed in chapter 2, artificial neural networks are known thanks to their capability to approximate every kind of function. A practical example of non-linear function approximation in the RL domain is the deep Q-network (DQN) which overcomes the limits of Q-learning. As suggested by the name, in DQN the goal is to approximate the action value function Q(s,a) through a Deep Neural Network; The difference from standard supervised learning problems is that during training phase, the target function $\hat{Q}(s, a; w)$ is continuously variable and a specific training method for non-stationary data is required. Furthermore, in RL problems there is not a supervisor, but only rewards; in case of MC learning, the supervisor is replaced by the return G_t from a particular state while in case of TD learning, the supervisor is the TD target. The former case is unbiased (noisy sample pf the true value) while the latter is biased; however, also in TD learning, the convergence is ensured at least to a local optimum. DQN uses experience replay and fixed Q-targets. Experience replay consists of storing in memory a given number of past experiences and then sampling randomly a small sized mini-batch of those experiences, that will be used as stable input for the DNN, in order to perform the training process and update network weights. However, training a DQN with only a single network for both Q value and target is not efficient: with a single neural network, in fact, both the first pass (which associate the Q value for the action-state pair) and the second pass

(which is used to compute the target Q value) occur using same network weights. Common weights generates instability problems since, at each iteration, the Q value is updated in order to get closer to the target value that, at the same time, moves in the same direction. The method used to solve this issue consists in using a different network (called target network) to compute the target Q value. The target-network is a DNN used to create the target and whose weights are kept constant for a definite number of iterations. Indeed, in a DQN algorithm there are always two DNN, denoted by weights w and w^-. At the beginning of the algorithm w and w^- are set equal with a random initialization and thereafter w is continuously updated while w^- is kept constant. Just after a certain number of iterations a reset is executed, and again $w^-= w$. The loss function to be minimized could be written as:

$$\mathscr{L} = E_{s,a,r,s' \sim D_i}\Big[\Big(r + \gamma \; max \; Q(s', a'; w_i^-) - Q(s, a; w_i)\Big)^2\Big]$$

where D is the dimension of the experience buffer.

4.5.5 Soft Actor Critic

The Actor-Critic (AC) architecture involve the use of two different entities, e.g. two function approximators, that compose the agent as we know it from the formal definition of an MDP. The actor is the part that chooses the action to be performed, while the critic judges what the actor does and tells it how good it's what it is doing. When talking of Actor-Critic architecture, is convenient to remember that it is used only during the training. When it is over, only the actor is used. As a matter of fact the critic is used to the train the actor, even though the critic requires a training too. The critic may approximate the action-value function using the TD error and the actor may approximate the policy. While the critic usually uses TD error loss function gradient to update its parameters, the actor network may be updated exploiting gradient coming from critic, e.g. using Policy Gradient algorithm. The soft actor critic (SAC) (which is an off policy algorithm) is a newest version of AC, recently introduced by Haarnoja et al [30]. Differently from most existing model-free models, it uses a continuous action space. This algorithm aims to maximize a target function composed not only of the term expected reward but also of an entropy term. The term entropy could be interpreted as how unpredictable a random variable is. This last term, is what expresses the attitude of our agent in the choice of random actions. An high entropy coefficient is desirable since it ensures the exploration of new policies avoiding the collapse into repeatedly selection of a particular action that could generate inconsistency in the approximated Q function. The objective function to minimize consists of both a reward term and an entropy term H weighted by a coefficient α, called *temperature parameter*:

$$J(\pi) = \sum_{t=0}^{T} E_{(s_t,a_t)\sim\rho_\pi}\Big[r(s_t, a_t) + \alpha \; H(\pi(\cdot|s_t))\Big]$$

SAC makes use of six networks: a policy function π, a target policy, two Q-functions Q and two Q targets. In principle, it would not be necessary to have separate

approximators; however, authors demonstrated that in practice having separate function approximators helps in convergence. So it is needed to train six different function approximators. The architecture of SAC is shown in the following figure:

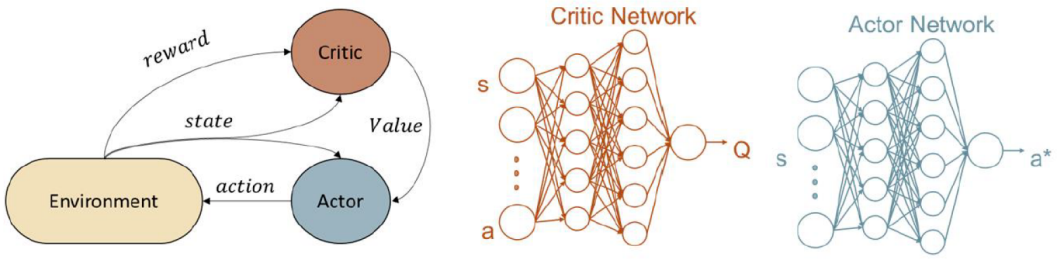

Figure 4.5: Actor-Critic-Environment interaction and neural networks in reinforcement learning [25]

Chapter 5

Case Study: integration of LSTM models into CityLearn environment

This chapter shows the integration of the neural networks models into the CityLearn environment. Through this process the RL agent could interact with building models, in order to learn the optimal control policy that allows to optimize energy consumption and reduce as much as possible the stress on the electrical grid (in terms of peak demand). The main contribution of this thesis is that the presence of neural network models makes it possible to modulate the HVAC power taking into account thermal comfort; during the training process, the centralized agent could decide to reduce or increase the power consumption: the selected power is then used as input to the neural network which returns the corresponding temperature. Then, this temperature value is used in the computation of the reward function: in this way the agent is trained to control the HVAC system in order to optimize energy usage, taking into account building thermal dynamics and thermal comfort. The first part of the chapter is related to the description of the RBC controller which is used as a benchmark; the second part deals with the explanation of the design DRL-SAC control logic: the definition of the action space, the state space and the reward function will be addressed and analyzed.

5.1 Description of the Rule Based Controller

The RBC is used as a benchmark, in order to evaluate the performance of the SAC algorithm. It is designed to always provide to the building the pre-computed cooling demand, which has been obtained from *IdealLoadsAirSystem* object of EnergyPlus. The cooling energy demand is satisfied by a heat pump which is dimensioned in order to cover the building's energy needs at each time step. For this reason the nominal thermal power is computed by multiplying the maximum cooling power of the considered building with a *"safety coefficient"* equal to 1.75 (this coefficient is necessary in order to provide enough cooling energy to satisfy the building cooling

load). The storages are designed in order to cover the maximum energy load for three hours, the cold one is linked to the heat pump while the hot one receives thermal energy from an electric heater. The RBC control actions are manually optimized to reduce energy costs: in particular, storages are uniformly charged during the night, when the cost of electricity is lower, and uniformly discharged during the day. CityLearn automatically sets constraints to the actions from the controllers to guarantee that the DHW and cooling demands are satisfied, and that the building does not receive from the storage units more energy than it needs.
It is important to notice that, by definition, the ideal load is the exact quantity of energy required to maintain the desired set point; for this reason, under the point of view of thermal comfort, it has not sense to compare the RBC and the SAC. However, the RBC is useful as a benchmark, to compare and to explore the potential of reinforcement learning as a control approach for building energy coordination and demand response.

5.2 Description of DRL Control Logic

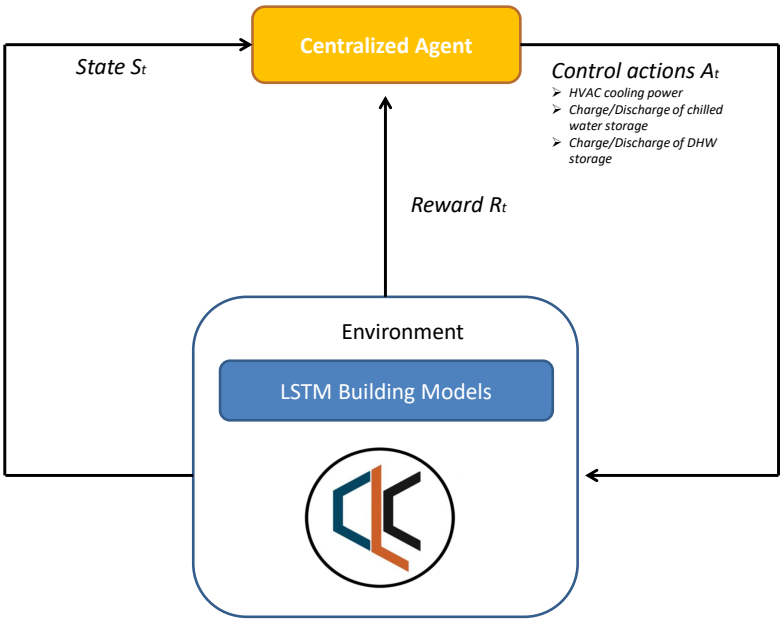

Figure 5.1: Scheme of the RL control problem

5.2.1 Description of the Action space

The action space of this control problem is quite complex because the agent is centralized: the dimension of the action space is equal to 11 since all buildings except the small office have 3 controlled variable: the HVAC cooling power, the chilled water storage charge/discharge and the DHW storage charge/discharge. The small office does not have the DHW storage, since its DHW demand is very low: for this reason it has only two controlled variables. The action related to the HVAC cooling power can vary from 0 to 1; then, the selected value is multiplied by the minimum between the nominal thermal power of the heat pump and the available cooling power in the corresponding time step (which could be lower than the nominal power due to environmental conditions). The actions related to the storages can vary between $-\frac{1}{3}$ and $\frac{1}{3}$, negative values indicate a discharge while positive values, a charge. These values are used to simulate a more realistic environment, where the storages, at each time step (1 hour), can be charged or discharged at a maximum rate of one third of their capacity.
It is necessary to specify that the environment was set in order to shut down the HVAC system during unoccupied hours: the action related to the HVAC power, in fact, is set to 0 during the night, when the building is unoccupied.

5.2.2 Description of the State space

The state represents the environment as it is observed by the control agent [31]. The state space is fundamental because the agent selects actions according to the values that the state assumes. The idea behind the selection of state variables is to include those that well describes the environment: for this reason, weather variables, like outdoor air temperature and direct and diffuse solar radiation, economic variables, like the price of electricity and its predictions from 1 to 3 hours ahead, and action-related variables, as the state of charge of storages, the consumed electrical power, the HVAC cooling power and the temperature difference between the set point and the predicted indoor temperature are included. Furthermore, the state space also encompasses low-level information from each building, regarding non-shiftable loads (e.g., fans, lights) and the COP of the heat pumps (both the nominal and the temperature-dependent one). Moreover, also outdoor temperature and solar radiation forecasts are included in the state space. In the following page it is possible to see in detail all the states which are used for each building, the table reports the min and max values for each variables; however, the *"*"* means that the min and max values for the considered variable depend on the building and, to avoid to displace four different tables, are omitted.

Variable	min value	max value	Unit
hour	1	24	h
day	1	7	-
month	6	8	-
Outdoor Air Temperature	10.8	36.9	°C
Outdoor Air Temperature 6h Forecast	10.7	36.96	°C
Outdoor Air Temperature 12h Forecast	10.5	37	°C
Outdoor Air Temperature 24h Forecast	10.3	37.96	°C
Direct Solar Radiation	0	1037.7	W/m2
Direct Solar Radiation 6h Forecast	0	1052.6	W/m2
Direct Solar Radiation 12h Forecast	0	1075.8	W/m2
Direct Solar Radiation 24h Forecast	0	1118.25	W/m2
Diffuse Solar Radiation	0	523.08	W/m2
Diffuse Solar Radiation 6h Forecast	0	526.36	W/m2
Diffuse Solar Radiation 12h Forecast	0	547.0022	W/m2
Diffuse Solar Radiation 24h Forecast	0	542.9011	W/m2
Non-shiftable load	*	*	kW
Cooling Storage State of Charge	*	*	-
DHW Storage State of Charge	*	*	-
COP	*	*	-
COP(T)	*	*	-
Total Hourly Electric consumption	*	*	kWh
RL HVAC Cooling Power	0	*	kW
Set Point	26	30	°C
ΔT Setpoint - LSTM indoor temperature	0	3	°C
Electricity price	0.03025	0.06605	$
Electricity price 1 hour ahead	0.03025	0.06605	$
Electricity price 2 hour ahead	0.03025	0.06605	$
Electricity price 3 hour ahead	0.03025	0.06605	$

Table 5.1: Variables included in the State space

Each state variable is then scaled between 0 and 1 through the *minmax normalization*, as generally required by neural networks.

5.2.3 Description of the Reward function

The design of the reward function is challenging, since it must be representative of the control problem under attention and it must take into account both the energy consumption and indoor thermal comfort. In this case study, the reward is the result of a combination of different factors which take into account thermal comfort, peak reduction and energy consumption. In particular, it can be expressed as :

$$R = \sum_{i=1}^{n} Comf\_penalty + \sum_{i=1}^{n} En\_penalty + \sum_{i=1}^{n} Stor\_prize + peak\_penalty$$

where n indicates the number of considered buildings.

Thermal Comfort Penalty

This term of the reward function is used to try to minimize the temperature violations; in particular, the objective of the controller is to maintain the mean indoor temperature in a range that goes from 25 °C to 27 °C, while the set point is equal to 26 °C. This term is designed as follow: if the set point is different from 26°C, its value is equal to zero.

$$Comf\_penalty = 0$$

On the other hand, when the temperature set point is 26 °C, the term is structured in this way:

- if $26 \leq T_{in} \leq 27$:
$$Comf\_penalty = 0$$
- if $25 \leq T_{in} < 26$:
$$Comf\_penalty = -m_{25}\ (26 - T_{in})$$
The idea behind this linear term is to encourage the controller to stay as much as possible close to 26 °C, in order to consume less energy.
- if $T_{in} < 25$:
$$Comf\_penalty = -m_{25}\ (26 - T_{in})^3$$
- if $T_{in} > 27$:
$$Comf\_penalty = -m_{27}\ (T_{in} - 26)^2$$

When the indoor temperature exceeds the lower or the upper bound, the comfort penalty is not linear anymore, but becomes non linear; the only difference is that, for lower temperatures, the exponent is cubic instead of quadratic: this is due to the fact that temperatures under 25 °C would generate both thermal discomfort and additional energy consumption.

Energy Penalty

The energy term of the reward function is computed only when the set point is equal to 26 °C (due to the fact that the HVAC cooling power is manually set to zero during unoccupied hours) and is used to minimize the electrical energy consumption during the HVAC operation. This term is calculated as follow:

$$Ener\_penalty = -K\ \frac{electric\_power\_cooling}{Heat\_pump\_nominal\_electric\_power}\ (\rho\ c_{el})^2$$

It is important to notice that the penalty takes into account only the electrical power and not the thermal one: in this way, the controller is encouraged to provide

energy from thermal storages rather than from the heat pump. The terms is weighted by a factor that include the energy price: in this way the electrical energy consumption during off-peak period is less penalized with respect to peak energy consumption. To further highlight this concept, the weight term is squared: the use of a quadratic exponent requires the coefficient ρ, in order to make the electricity price grater than 1.

Storage Prize

This term of the reward function is the only positive term and it is computed only during off-peak periods. Since the action related to the HVAC cooling power is set to zero when the building is unoccupied, the only way for the agent to maximize this term is to consume energy by charging the storages. This term is also related to the previous energy penalty: the former incentives to discharge storages during the day, when the electricity price is higher, while the latter incentives the charge during off peak periods. This term is computed as follow:

$$Stor\_prize = (total\_electrical\_power - non\_shiftable\_electrical\_power)\ c_{el}\ \delta$$

Peak penalty

This term has the purpose of flatten buildings and cluster profiles. To achieve this result, the controller tries to minimise the sum of the squared energy consumption of each building for each time step. This has the role of flattening the single building consumption thanks to the "squared" formulation. Moreover, it minimises the sum of each building squared consumption, avoiding simultaneous charge of storages [25]. It is computed in this way.

$$peak\_penalty = -\beta \sum_{i=1}^{n} (total\_energy\_consumption)^2$$

The design of the reward function highly influences Reinforcement Learning performances, searching compromises between energy savings, thermal comfort and grid stability. The coefficients β, δ, K, ρ, m_{25} and m_{27} weight the relative importance of the energy savings, temperature violations and peak shaving actions. Moreover, since the reward magnitude influences the behaviour of SAC algorithm [20], these coefficients are used to tune exploration-exploitation trade-off of the agent. A sensitivity analysis was performed with the aim of finding the optimal balance between these terms. The values used in this thesis work are summarized in the following table.

Variable	Value
β	0.005
δ	1.4
K	0.002
m_{25}	0.1
m_{27}	0.1
rho	50

A further sensitivity analysis was performed in order to find the best hyperparameter configuration. The selected parameters are summarized in the following table.

Variable	Value
DNN architecture	2 layers
Neurons per hidden layer	256
DNN Optimizer	Adam
Batch size	512
Discount rate γ	0.9
Learning rate λ	0.003
Decay Rate τ	0.005
Temperature Coefficient α	0.08
Target Entropy H	auto
Target model update	1
Episode Length	2196 Control Steps (92 days)*
Training Episodes	20

The agent was thus trained over the entire summer period (June, July, August), which represents an episode; the *"*"* highlights the fact that the control steps are not 2208 because the first 12 values are used as input to the LSTM models (*lookback = 12*). The training process lasted for 20 episodes, followed by a deployment phase of the learned control policy on the previously defined episode.

Chapter 6

Results

In this chapter the results of the thesis work are reported and commented. The following analysis is the result of a static deployment on a single episode, which lasts from the beginning of June to the end of August. In order to evaluate the goodness of the SAC control policy, proper metrics have been calculated: their aim is to asses the ability of the DRL agent to maintain comfort conditions during the occupancy period (mean indoor temperature within a comfort band in the range [25,27]) and to optimize both the energy consumption and the load shape. The comfort performances were evaluated through the calculation of the cumulative sum of temperature violations during the occupancy hours, measured in °C and the average temperature discomfort corresponding to temperature violations (a temperature violation occurs when, during occupied hours, the temperature falls below 25 °C or rise above 27 °C). Furthermore, different *"Key Performance Indicator KPI"* were computed to evaluate the ability of the controller in reducing the stress on the electrical grid. In particular, were considered:

- Peak electric Power, computed as the max electric power consumed by the building over the summer period
- Peak To Average Ratio (PAR), computed as the ratio between the max consumed power and the mean one
- Cluster Electricity Cost
- Flexibility Factor, computed as the sum of hourly average off peak energy consumption over the sum of hourly average energy consumption

In particular, the overall cluster electricity cost was computed by considering a variable electricity price, which varies from 0.03025$ during off-peak hours to 0.06605$ during on-peak periods. For what concerns the energy and economic performances, the controller was compared with a manually optimized Rule Based Controller. Conversely, temperature performances have not been benchmarked: doing a comparison with respect to the ideal load would not have sense, since, by definition, it provides the exact thermal energy to maintain the indoor set point perfectly. Surely, a future work have to take into account also the thermal

performances benchmark with respect to a standard control strategy.
The following pages show the results of the thesis work: firstly, the analysis is focused on thermal comfort aspects related to each buildings, analyzing the ability of the controller to maintain comfort conditions. Then, the analysis is focused on cluster level, where the impact that DRL control strategy has on the electrical grid is evaluated.

6.1 Comfort Analysis: Small Office

The first plot shows a weekly temperature profile related to the small office. The red band indicates the thermal comfort range while the orange one indicates the occupied hours.

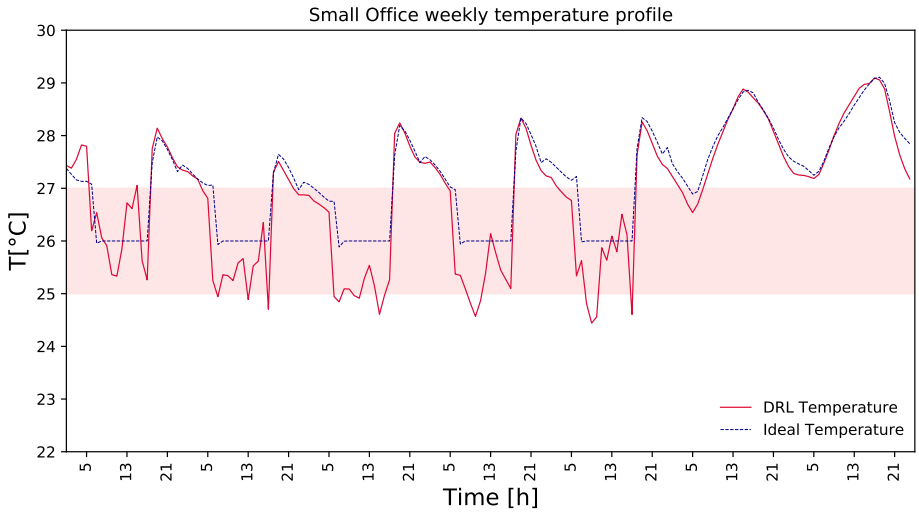

Figure 6.1: Small Office: Example of Weekly Temperature Profile

As could be seen, the agent is effective in maintaining the indoor temperature within the comfort range; temperatures rarely fall below 25 °C and, in case of temperature violation, the controller is able to restore thermal comfort in the following hour. To confirm the agent's ability in maintaining comfort conditions, it is possible to observe the average indoor temperature computed for each hour of the day during the cooling period. To perform this analysis, temperatures were reshaped in order to have a data set with 24 columns (hours) and 92 rows (days); subsequently, since the office is not occupied on weekends, a mask has been created to exclude from the calculation of the average unoccupied days, in order to have averages that effectively represents the indoor temperature conditions during the HVAC operation . A more detailed analysis could be performed by taking into account box plots; they have been generated for each hour of the day, in order to better represent temperature distribution during occupied hours.
These plots are analyzed in the following page:

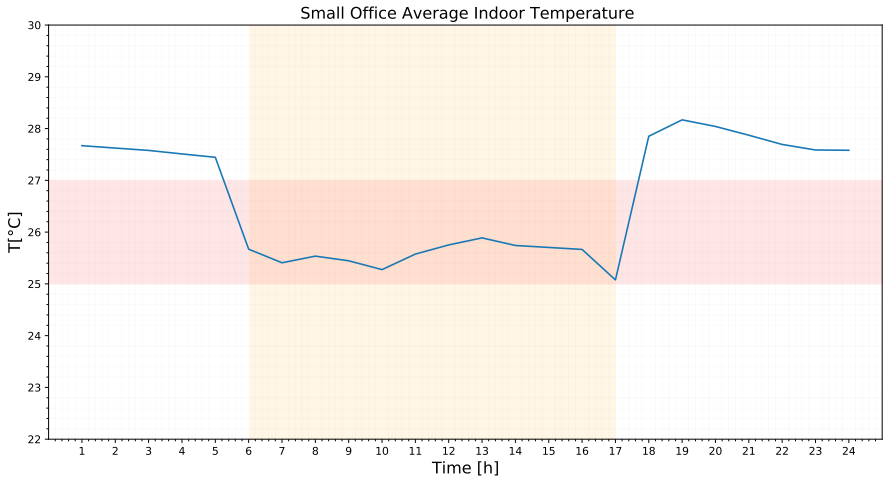

Figure 6.2: Small Office: Cooling Season Hourly Average Temperature

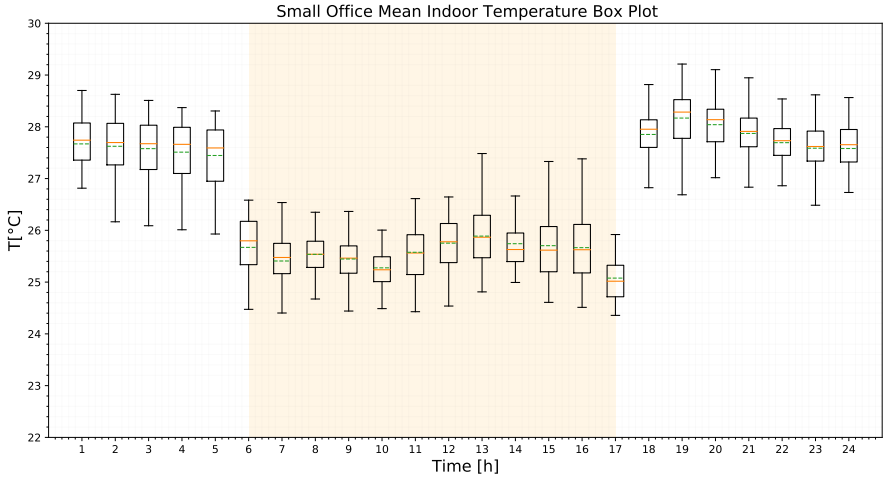

Figure 6.3: Small Office: Box Plot of Hourly Temperature

From these two figures it is possible to notice that the controller is able to maintain comfort conditions: the mean temperature falls always within the comfort band and interquartile range is almost included in [25 °C,27 °C] interval. Another useful tool to understand temperature distribution during occupied hours is the histogram, which is shown in the following page:

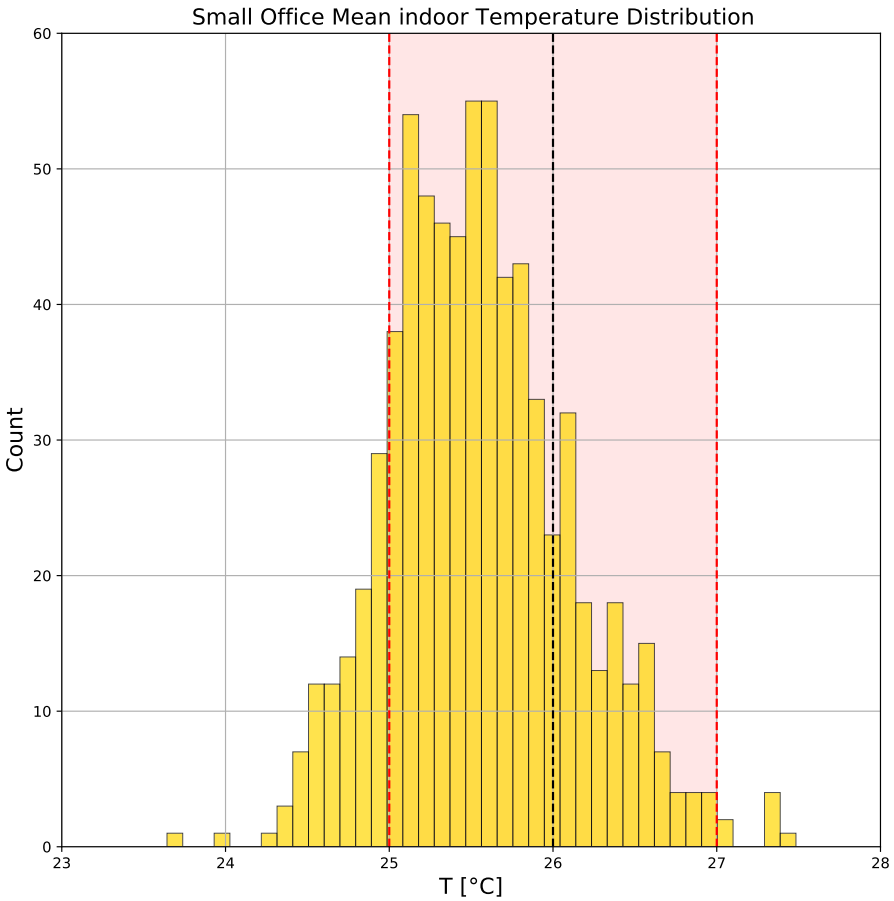

Figure 6.4: Small Office: Mean Indoor Temperature Distribution

The histogram only includes temperatures during occupied hours. As could be seen also from the previous plots, the distribution is not centered in 26°C; however, almost all values fall within the comfort range. The upper bound is almost never overtaken, while temperature falls easily under 25°C.

6.2 Comfort Analysis: Retail Stand Alone

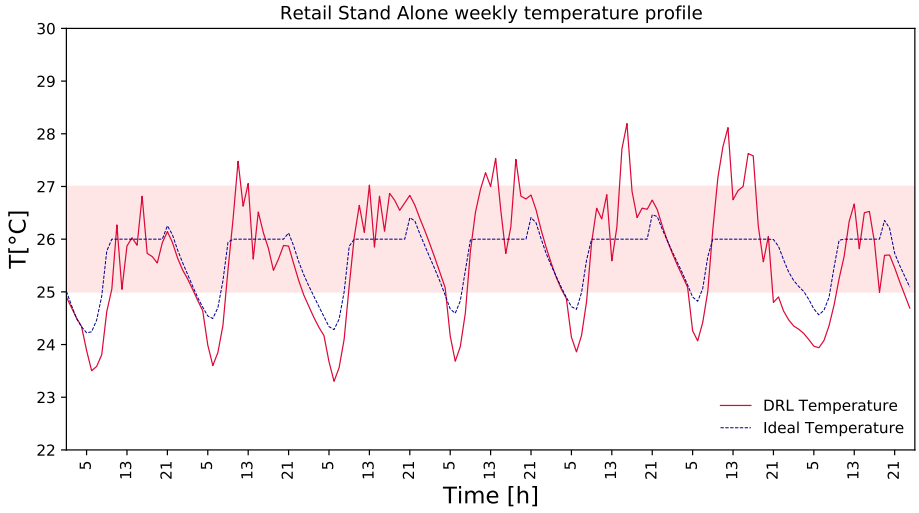

Figure 6.5: Retail Stand Alone: Example of Weekly Temperature Profile

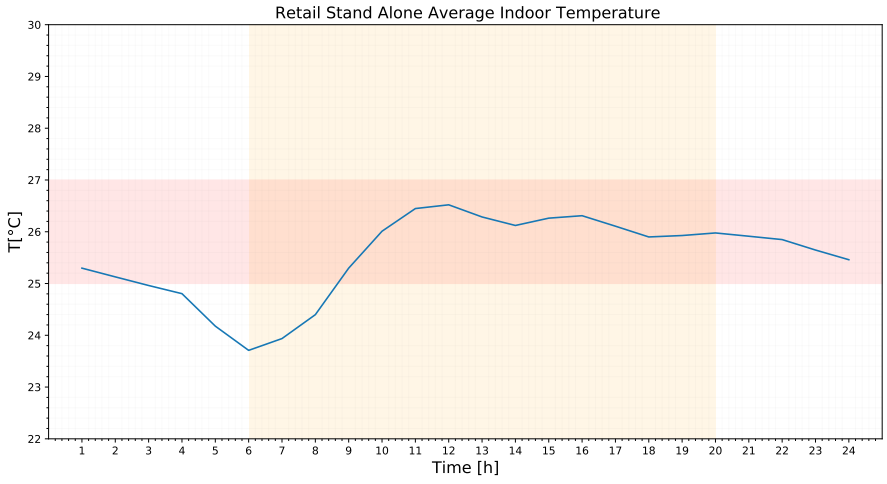

Figure 6.6: Retail Stand Alone: Cooling Season Hourly Average Temperature

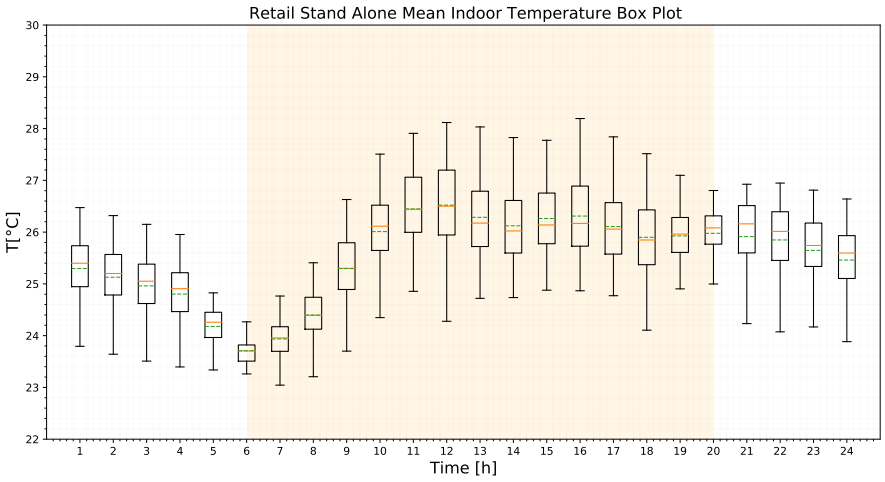

Figure 6.7: Retail Stand Alone: Box Plot of Hourly Temperature

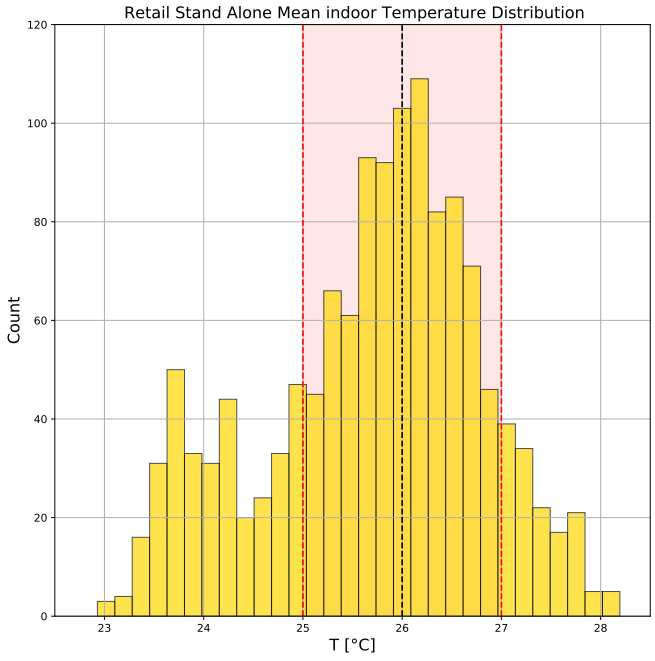

Figure 6.8: Retail Stand Alone: Mean Indoor Temperature Distribution

The Retail Stand Alone Building is characterized by a greater number of temperature violations; however, for what concerns temperatures lower than 25°C, it is important to notice that also the ideal HVAC system violates this lower bound. This happens especially in the morning when the outdoor air temperature is lower

then 25°C: since the HVAC is an air water system, the air system introduce a certain outdoor air flow rate per person that cause the overcooling in the earlier hours of the morning. However, except for these hours, the average indoor temperature falls in the comfort range and the interquartile range is almost always between the lower and upper temperature threshold.

6.3 Comfort Analysis: Restaurant

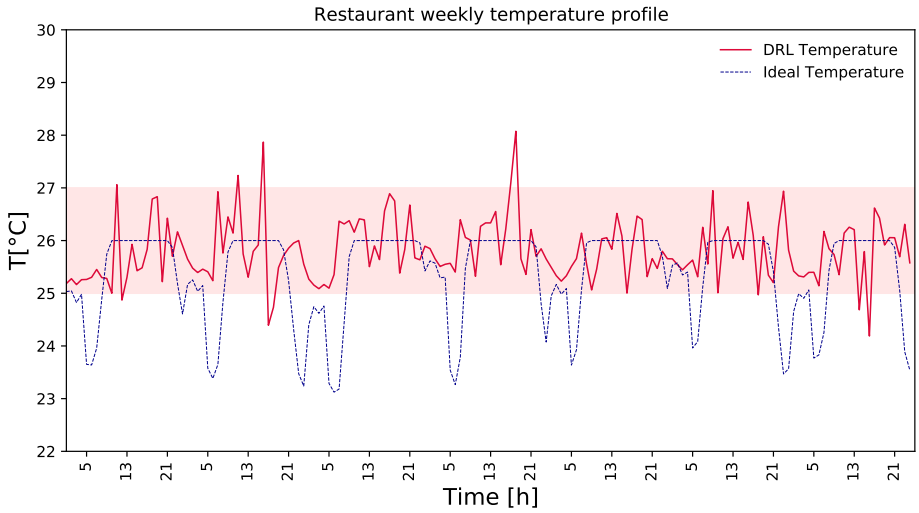

Figure 6.9: Restaurant: Example of Weekly Temperature Profile

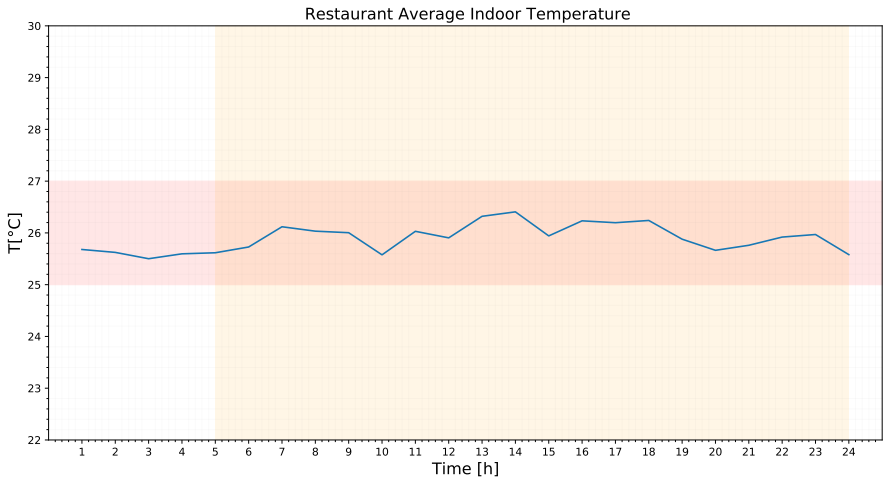

Figure 6.10: Restaurant: Cooling Season Hourly Average Temperature

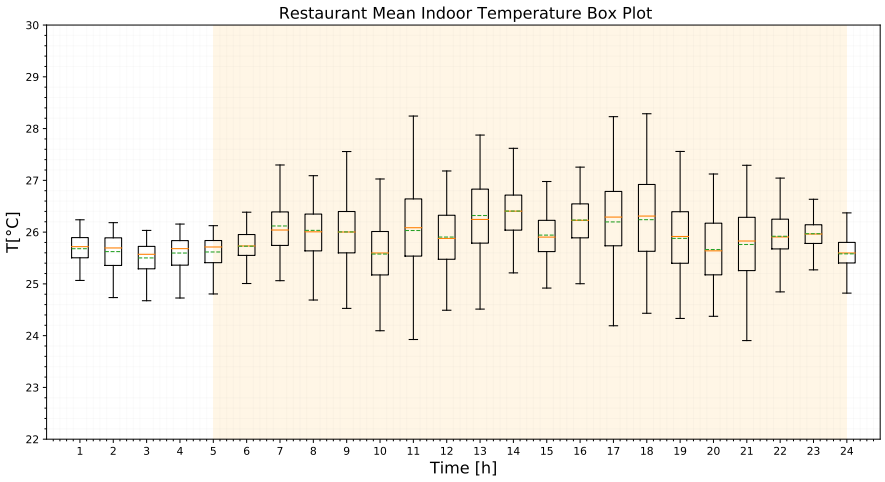

Figure 6.11: Restaurant: Box Plot of Hourly Temperature

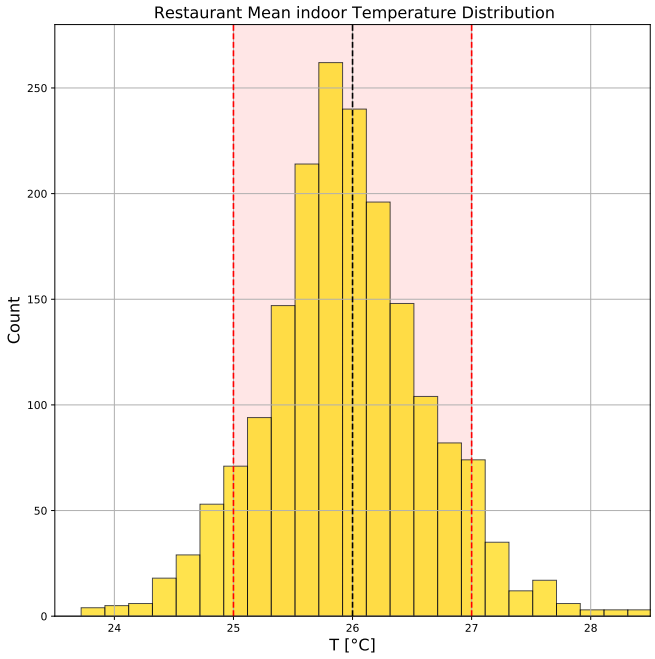

Figure 6.12: Restaurant: Mean Indoor Temperature Distribution

In the case of restaurant, the controller is very effective in maintaining the indoor temperature within the desired band: the average temperature is very close to the set point at each hour, moreover, as could be seen from the histogram, temperature violations are rare. It is important to notice that, in this case, the interquartile

range is wider than that of other buildings, which means that indoor temperature has exceeded the considered thresholds in some hours during the cooling period. However, the temperature distribution profile is very close to a gaussian centered in 26°C.

6.4 Comfort Analysis: Medium Office

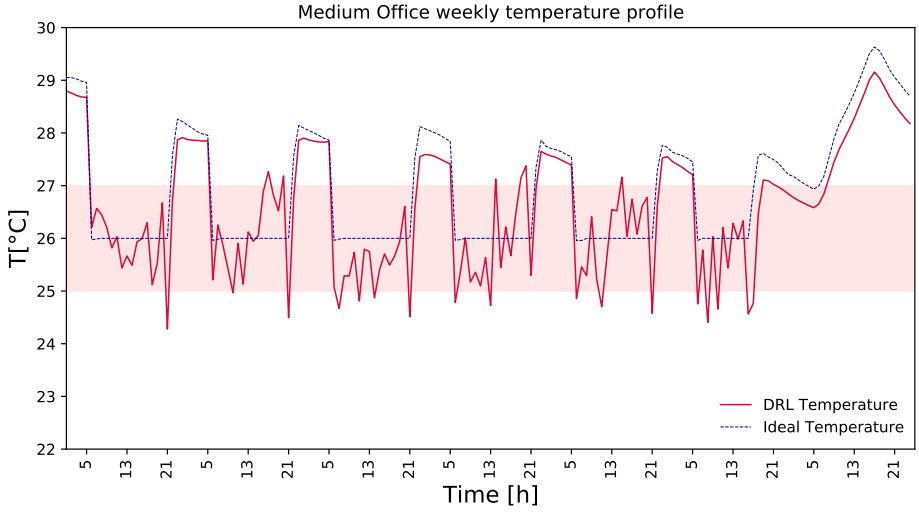

Figure 6.13: Medium Office: Example of Weekly Temperature Profile

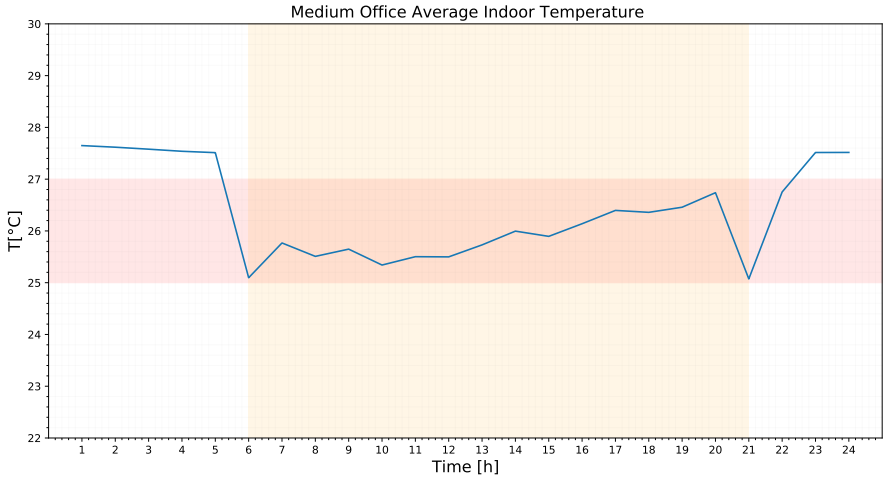

Figure 6.14: Medium Office: Cooling Season Hourly Average Temperature

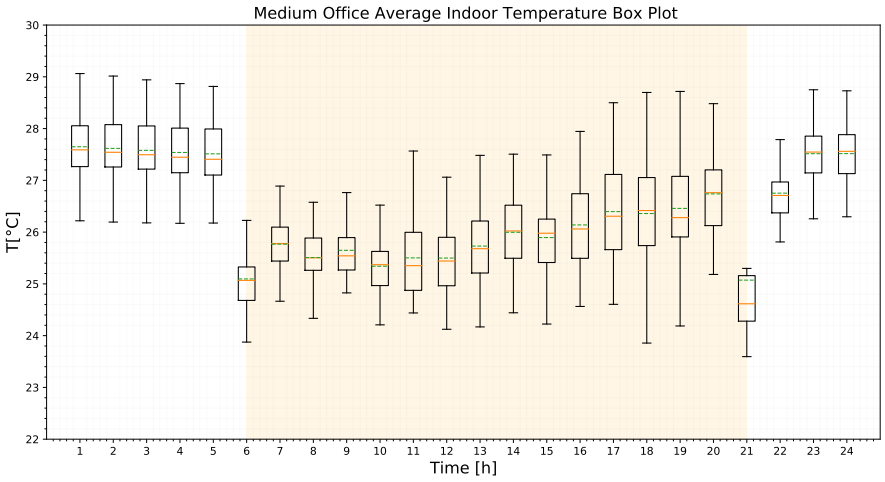

Figure 6.15: Medium Office: Box Plot of Hourly Temperature

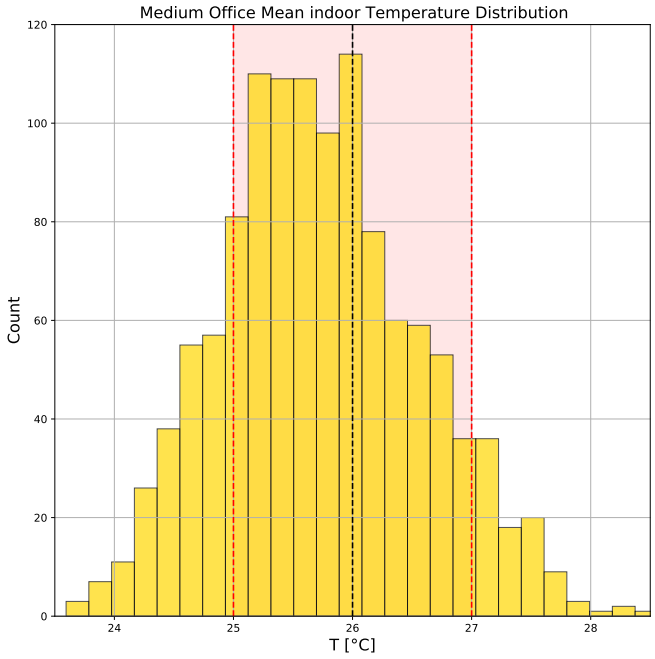

Figure 6.16: Medium Office: Mean Indoor Temperature Distribution

Also in this case, it could be noticed that the SAC controller is able to maintain the comfort conditions: the average indoor temperature is always between 25 °C and 27°C. Moreover, except for the first hour in the morning and the last hour in the evening, the interquartile range falls within the desired temperature band.

Furthermore, the temperature distribution is, also in this case, very close to a gaussian centered in the set point value. The Thermal comfort metrics are reported in the following table:

Building	**Cumulative T<25 [°C]**	**Avg. T Discomfort [°C]**	**Cumulative T>27 [°C]**	**Avg. T Discomfort [°C]**
Small Office	26.7	0.25	1.89	0.27
Retail	110.3	0.49	54.5	0.40
Restaurant	41.9	0.36	49.6	0.41
Medium Office	90.9	0.40	39.25	0.40

Table 6.1: Thermal comfort metrics

The table shows that, despite the presence of temperature violations, the average discomfort is maintained under 0.5 °C with respect the two considered thresholds. Moreover, it is important to notice that the violations above the upper bound are limited with respect to those under 25 °C. In fact, an indoor temperature greater than 28°C could generate very uncomfortable conditions that must be avoided. Temperature violations under 25°C are less relevant in terms of comfort (also because, as previously highlighted, it is possible to have the outdoor air temperature lower then the lower threshold) but could generate additional energy consumption.

6.5 Electrical Load Analysis

Another important analysis concerns the evaluation of how the implemented control strategy influences the electrical grid: a first observation regards the storage control actions, since their charge and discharge can impact on the load profile.

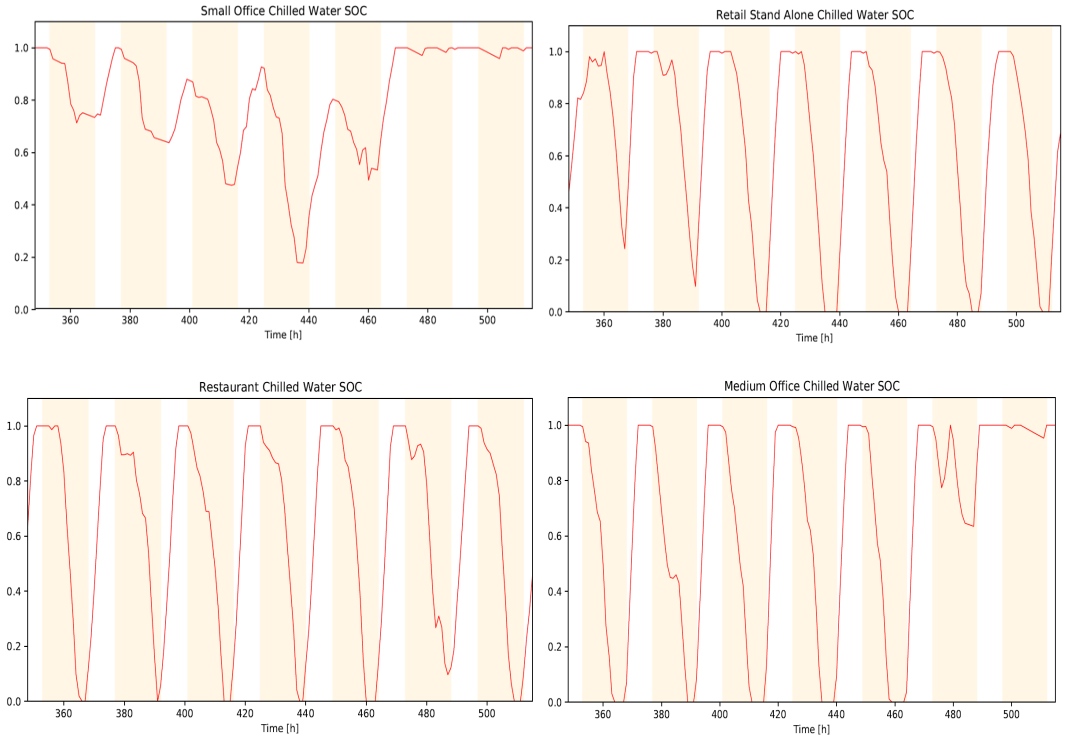

Figure 6.17: State of Charge of Chilled Water Storage

These plots show how the DRL agent controls the chilled water storages; the period between the orange band are those characterized by higher energy cost: it is clear that the controller tries to charge the storages when the electricity price is lower, in order to use stored energy during on-peak periods. Moreover, it is important to highlight that the controller, differently from the RBC which uses an uniform charge, learns to charge the storages as soon as the electricity price become lower, in order to avoid operating the heat pumps at partial load. The small office presents a control strategy less efficient than the other three buildings, since in certain periods it charges the storage during peak periods: this is certainly an aspect to improve in a future work. For what concerns the load shape metrics, the controller achieved very good performances: the peak power consumption is greatly reduced, passing from 161 kW consumed by the RBC, to 144 kW of the SAC controller. Peak reduction is very important because it increase the reliability of the electrical grid by reducing congestion; moreover, a lower peak power implies a reduction of peak plant operation, which are expensive to operate and have a relevant environmental impact. Furthermore, the SAC controller was able to achieve a great reduction of

Peak To Average Metric and showed a slightly enhanced Flexibility Factor. Also the overall cost associated to cluster energy consumption have been slightly reduced by the DRL control policy: it passed from 6661 $ to 6613 $. To visualize the peak reduction it is possible to see the following figures:

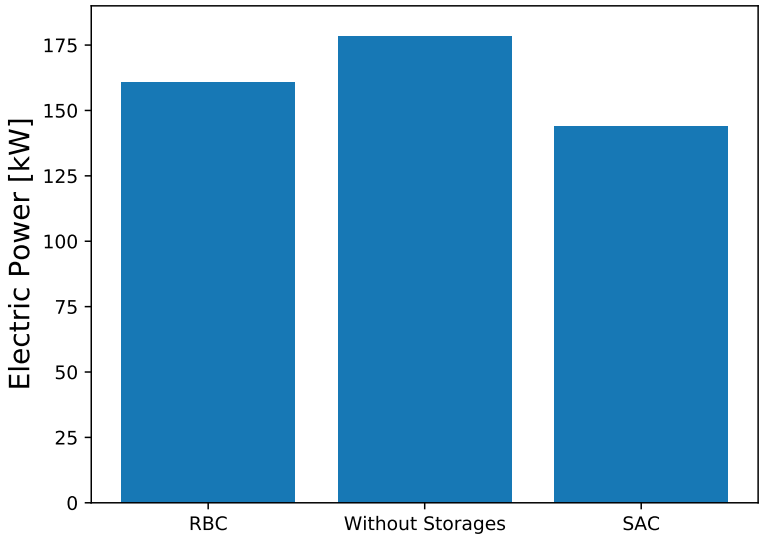

Figure 6.18: Comparison between Electrical Power Peak

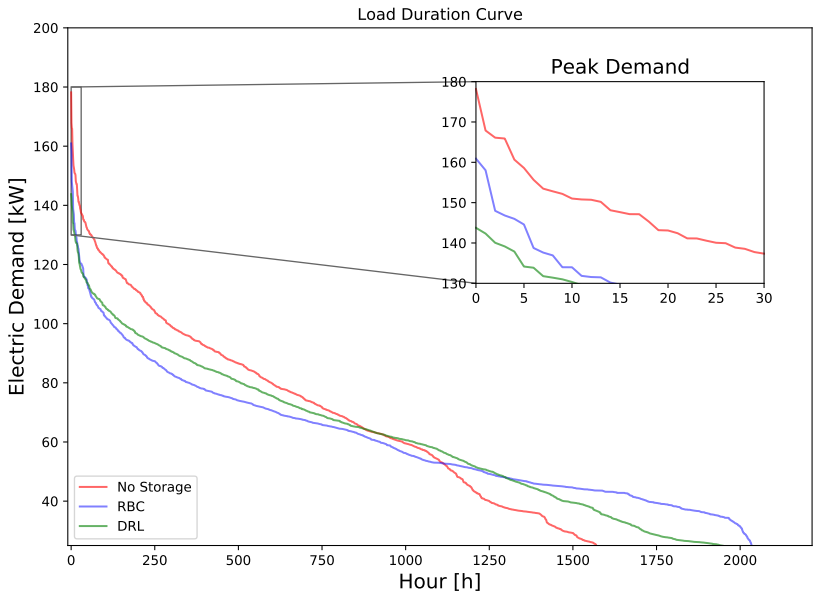

Figure 6.19: Load duration curve to highlight peak reduction

The load duration curve well highlights the peak reduction; moreover, it is interesting to observe that the RBC, thanks to its *if-then logic*, has generated a smoother profile with respect to SAC controller that, on the other hand, has specifically optimized and reduced the peak consumption. The following table summarize the SAC controller performances:

	E. Cost	Peak	PAR	Flexibility Factor
Manually Optimise RBC	1	1	1	1
SAC	0.99279	0.89341	0.89179	0.99721

To visualize KPI comparison is also shown the corresponding graphical representation:

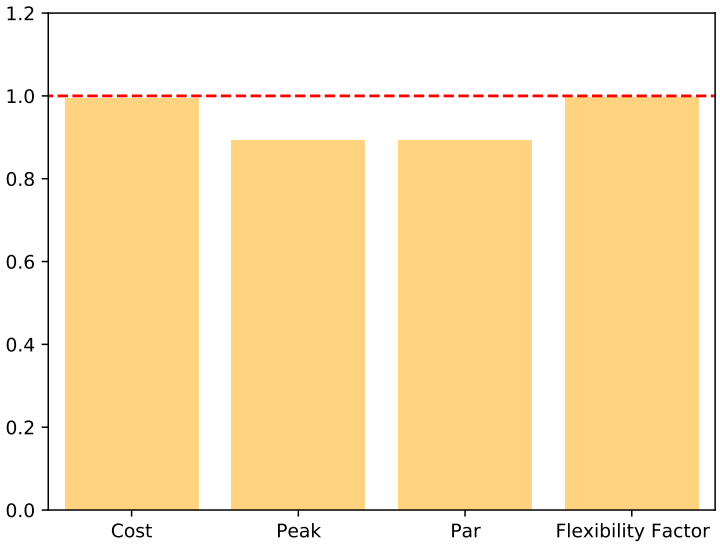

Figure 6.20: KPI bar plot, the red line represent the RBC

Chapter 7

Conclusion and Future Work

This thesis work has shown the possibility and the feasibility to develop a fully data-driven control scheme able to manage a cluster made by four different buildings, characterized by different occupancy profiles and internal loads. In particular, in the first part of the thesis, the study was focused on the development of four LSTM models able to emulate the buildings thermal dynamic, in order to perform multi-step temperature forecast through recursive strategy. LSTM are a particular supervised learning techniques which belong to the Recurrent Neural Network category, they are especially used for time series forecast, due to their particular gating mechanism which make possible to manage both short and long term dependencies. The developed models have shown great performances on the test data set, avoiding the error accumulation and predictions divergence. Moreover, the recursive strategy presented prediction metrics similar to those of 1-step ahead prediction model. Succesively, these four models have been integrated into CityLearn environment, which is an open AI environment for the implementation and the comparison of different reinforcement learning agent, in a demand-response setting. Reinforcement Learning is a particular machine learning technique especially used for optimal control problems. RL has acquired increasingly popularity since it is model free and could learn optimal control policy only by interacting with its surrounding environment. Other works have been focused on CityLearn, however, in those works the building cooling load was known a priori. The contribution of this study regards the possibility to control not only the storages, but also the HVAC system cooling power, as an additional source of demand-response and load shaping. The considered control problem is challenging, since the central agent has to find the optimal trade off between thermal comfort, energy consumption and load shape of each building. However, the implemented DRL controller was able to effectively reduce peak power consumption, optimize costs and load shape, without excessively penalize thermal comfort. One of the main limitation of this work is the absence of a thermal comfort benchmark, since the comparison with the ideal load is infeasible due to the fact that, in a practical implementation, is almost impossible to maintain the temperature fixed to the desired set point

level. For this reason, a future work could be related to the comparison with a standard, state of the art, control technique. Moreover, other possible studies should regard the design of the reward function, in order to enhance even more the load shifting/shaping, without jeopardizing thermal comfort. Another idea is to implement a cooperative/competitive multi agent SAC, instead of a centralized controller: this would reduce the action space for each controller, which could bring benefits for both comfort and energy consumption. Eventually, a further expansion of the work could be done implementing the algorithm of the proposed framework starting from real data, collected by a real building cluster; this idea comes from the necessity to get closer to a real, practical implementation, in order to evaluate the effectiveness of DRL in a real world setting.

Bibliography

[1] IEA. *World Energy Outlook*. 9 rue de la Fédération 75739 Paris Cedex 15 France: IEA Publications, 2019 (cit. on p. 1).

[2] IEA. *The Critical Role of Buildings*. URL: https://www.iea.org/reports/the-critical-role-of-buildings (cit. on pp. 1, 2).

[3] IEA. *World Energy Model*. URL: https://www.iea.org/reports/world-energy-model (cit. on p. 2).

[4] EBC. *Annex 67*. URL: http://www.annex67.org/about-annex-67/ (cit. on p. 3).

[5] IEA. «Review of applied and tested control possibilities for energy flexibility in buildings». In: (2018) (cit. on p. 3).

[6] K.O. Aduda, T. Labeodan, W. Zeiler, G. Boxem, and Y. Zhao. «Demand side flexibility: Potentials and building performance implications». In: *Sustainable Cities and Society* (2016) (cit. on p. 3).

[7] IEA. *Examples of Energy Flexibility in Buildings*. 2019 (cit. on p. 4).

[8] Afram A. and Janabi-Sharifi F. «Theory and applications of HVAC control systems - A review of model predictive control (MPC)». In: *Building and Environment* (2014) (cit. on p. 5).

[9] URL: https://deepai.org/machine-learning-glossary-and-terms/machine-learning (cit. on p. 9).

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2014 (cit. on pp. 10, 67).

[11] F.Rosenblatt. «The Perceptron — a perceiving and recognizing automaton». In: *Report 85–460–1, Cornell Aeronautical Laboratory* (1957) (cit. on pp. 11, 12).

[12] URL: https://medium.com/@kshitijkhurana3010/activation-functions-in-neural-networks-ed88c56b611b (cit. on p. 20).

[13] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2020 (cit. on p. 22).

[14] A.E. Ruano, E.M. Crispim, E.Z.E. Conceição, and M.M.J.R. Lúcio. «Prediction of building's temperature using neural networks models». In: *Energy and Buildings* (2005) (cit. on p. 28).

[15] G. Mustafaraj, G. Lowry, and J. Chen. «Prediction of room temperature and relative humidity by autoregressive linear and nonlinear neural network models for an open office». In: *Energy and Buildings* (2011) (cit. on p. 28).

[16] Z. Afroza, T. Urmee, G.M. Shafiullah, and G. Higgins. «Real-time prediction model for indoor temperature in a commercial building». In: *Applied Energy* (2018) (cit. on p. 28).

[17] H. Huang, L. Chen, and E. Hu. «A neural network-based multi-zone modeling approach predictive control system design in commercial buildings». In: *Energy and Buildings* (2015) (cit. on p. 28).

[18] A. Marvuglia, A. Messineo, and G. Nicolosi. «Coupling a neural network temperature predictor and a fuzzy logic controller to perform thermal comfort regulation in an office building». In: *Building and Environment* (2013) (cit. on p. 29).

[19] M.J. Ellis and V. Chinde. «An encoder–decoder LSTM-based EMPC frameworkapplied to a building HVAC system». In: *Chemical Engineering Research and Design* (2020) (cit. on p. 29).

[20] F. Mtibaa, K. Nguyen, M. Azam, A. Papachristou, J. Venne, and M. Cheriet. «LSTM-based indoor air temperature prediction framework for HVAC systems in smart buildings». In: *Neural Computing and Applications* (2020) (cit. on p. 29).

[21] URL: `https://www.energycodes.gov/development/commercial/prototype_models` (cit. on p. 31).

[22] M. Claesen and B. De Moor. «Hyperparameter Search in Machine Learning». In: *The XI Metaheuristics International Conference* (2015) (cit. on p. 39).

[23] J. R. Vázquez-Canteli, J. Kämpf, G. Henze, and Z. Nagy. «CityLearn v1.0: An OpenAI Gym Environment for Demand Response with Deep Reinforcement Learning». In: *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (2019) (cit. on p. 60).

[24] A. Kathirgamanathan, K. Twardowski, E. Mangina, and D. P. Finn. «A Centralised Soft Actor Critic Deep Reinforcement Learning Approach to District Demand Side Management through CityLearn». In: *arXiv:2009.10562* (2020) (cit. on p. 60).

[25] G. Pinto, S. Brandi, A. Capozzoli, Z. Nagy, and J.R Vazeque-Canteli. «Towards Coordinated Energy Management in Buildings via Deep Reinforcement Learning». In: *15th SDEWES Conference* (2020) (cit. on pp. 60, 75, 81).

[26] B. Chen, W. Yao, J. Francis, and M. Bergés. «Learning a Distributed Control Scheme for Demand Flexibility in Thermostatically Controlled Loads». In: *arXiv:2007.00791* (2020) (cit. on p. 60).

[27] J. R. Vázquez-Canteli, G. Henze, and Z. Nagy. «MARLISA: Multi-Agent Reinforcement Learning with Iterative Sequential Action Selection for Load Shaping of Grid-Interactive Connected Buildings». In: *BuildSys '20: Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (2020) (cit. on p. 61).

[28] David Silver. *Lecture 1: Introduction to Reinforcement Learning*. URL: `https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf` (cit. on p. 65).

[29] David Silver. *Lecture 2: Introduction to Reinforcement Learning*. URL: `https://www.davidsilver.uk/wp-content/uploads/2020/03/MDP.pdf` (cit. on p. 67).

[30] T. Haarnoja. «Soft Actor-Critic Algorithms and Applications». In: (2018) (cit. on p. 74).

[31] S. Brandi, M.S. Piscitelli, M. Martellacci, and A. Capozzoli. «Deep reinforcement learning to optimise indoor temperature control and heating energy consumption in buildings». In: *Energy Buildings* (2020) (cit. on p. 78).