Reinforcement learning in home energy management systems: tutorial, survey and application

by

Omar Ayad Abduljabbar Al-Ani

B.Sc., Al-Mustansiriya University, 2013

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
Carl R.Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2022

Approved by:

Major Professor
Dr. Sanjoy Das

Copyright

Abstract

This thesis provides a comprehensive overview of all major reinforcement learning algorithms and a review of all areas of home energy management systems. The last chapter of the research proposes a subjective method to improve human environmental comfort using imitation learning and a state-of-the-art tabular machine learning algorithm called deep attentive tabular neural network (TabNet). Previous studies have focused on using reinforcement learning as a method to replace human based control, are limited because human comfort is not a subjective value that you can easily measure. This research takes an alternate route; it applies *imitation learning*, a paradigm that is strongly connected to reinforcement learning, in order to simulate human behavior, while maintaining energy consumption at desirable levels. In treating humans as the 'experts', it eliminates the need to quantify occupant comfort during the learning process. Policy models were developed using TabNet. TabNet is trained on the expert's samples, which are obtained from switching the HVAC thermostat settings, then this learned policy is applied to the period when the occupant is inactive (sleeping/lazy). Our primary objective is to minimize the difference of the indoor temperature and humidity of the occupants when he is active, sleeping or lazy. Predicted mean vote (PMV) is used here to measure if that objective have been met or not. PMV is a parametric equation that is used extensively to measure the indoor comfort.

Our results show that applying our learned policy improved the comfort level by about 9% and close the comfort's gap between the active and sleeping periods of the occupants.

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

1.1 Introduction

The largest group of consumers of electricity in the US are residential units. In the year 2020, this sector alone accounted for approximately 40% of all electricity usage [1]. The average daily residential consumption of electricity is 12 kWh per person [2]. Therefore, effectively managing the usage of electricity in homes, while maintaining acceptable comfort levels, is vital to address global challenges of dwindling natural resources and climate change. Rapid technological advances have now made *home energy management systems* (HEMS) an attainable goal that is worth pursuing. HEMS consist of automation technologies that can respond to continuously or periodically changing home environmental as well as relevant external conditions, without human intervention [3], [4]. In this review, the term 'home' is taken in a broad context to also include all residential units, classrooms, apartments, offices complexes, and other buildings in the smart grid [5], [6], [7], [8].

Artificial Intelligence (AI), more specifically machine learning, is one of the key contributing factors that have helped realize HEMS today [9], [10], [11]. *Reinforcement learning* (RL) is the class of machine learning algorithms that is making deep inroads in various applications in HEMS. RL allows an algorithmic entity from experience to make sequences of decisions and implement actions in the same manner as a human being [12], [13], [14], [15], [16], [17]. DNNs has proven to be a powerful tool in RL, for it endows the RL agent with the capability to adapt to a wide variety of highly complex domains [18], [19]. In [20] it has been proposed that RL can attain the ultimate goal of artificial general intelligence [21].

Consequently, RL is making deep inroads into many application domains today. It has been applied extensively to robotics [22]. Specific applications in this area include manipulation

of soft bio-inspired robots with many degrees of freedom [23], navigation and path planning applications of mobile robots and UAVs [24], [25], [26]. RL finds widespread applications in communications and networking [27]. It has been used in 5G enables UAVs with cognitive capabilities [28], cybersecurity [29], and edge computing [30]. Other domains where RL has been used include hospital decision-making [31], precision agriculture [32], and fluid mechanics [33]. It is of little surprise that RL has been extensively used to solve various problems arising in energy systems [34], [35], [36], [37], [38]. Another review article on the use of RL [38] considers three application areas frequency and voltage control as well as energy management.

RL is increasingly being used in HEMS applications and several review papers have already been published. The review article in [39] focuses on RL for HVAC and water heaters. The paper in [40] is based on research published between 1997 and 2019. The survey observes that only 11% of published research reports deployment of RL in actual HEMS. The article in [41] specifically focuses on occupant comfort in residences and offices. A more recent review on building energy management [42] focuses on deep neural networks based RL. A recent article [43] considers RL along with model predictive control in smart building applications. The article in [44] is a survey of RL in demand response.

1.2 RL for Environment Control in HEMS

Environment control in the smart home, primarily indoor temperature control, may be carried out while maintaining acceptable comfort levels of the home's occupants. Conversely, comfort may be improved using automated scheduling, without increasing the energy consumption. A few recently published articles aim at making the second objective an attainable one. Research attention has been directed towards investigating deep reinforcement learning

algorithms for smart home applications [45], [46], [47], [48]. Deep reinforcement learning algorithms that concentrate specifically on environment control have begun to appear [SBM+21].

Recently, RL has been used widely to control the indoor environment. Most of the published papers in this field have focused on temperature control by controlling the HVAC system using RL as in [49], [50, [51]. However, other published papers have been focused on controlling other equipment to manage the indoor temperature such as the split air conditioner (AC) as in [52], the ventilation fan [53], [54], the underfloor heating (UFH) as in [55], heat pumps [56], and windows opening and closing [57].

The main objective of most of the reviewed papers is to minimize the energy cost while maintaining or increasing the occupant's comfort as in [54]. However, only a few papers have focused on improving the occupant's comfort alone as in [57]. It is worth mentioning that most of the papers have considered indoor temperature as the main factor for occupant's comfort, just a slight percentage of the papers take into consideration the air quality (CO2 concentration) and visual comfort (lighting) as in [58].

In [56], the study has reduced energy consumption by 4%-11% and kept the thermal comfort within an acceptable limit. In [59], the energy cost has slightly reduced compared to other algorithms and kept the comfort within an acceptable limit. The predicted mean vote (PMV), which is also used to calculate comfort, has not been used in these papers. However, the thermal comfort and thermal violation have been assumed to be enough to measure the occupant's satisfaction. The proposed algorithm in [48] is another example where the energy cost has been reduced by 3.51% in winter and 4.05% in summer compared with the DDQN algorithm with minimum comfort violation. In general, the mentioned papers and other similar papers are

not trying to improve the indoor environmental condition but rather to reduce the thermal violation of a predefined thermal limit.

In [60], the comfort level has been improved roughly by 15% but the energy cost has not been discussed. In [57], windows opening and closing have been used to improve the thermal comfort and the air quality by more than 90% without using any electricity. In [58], the comfort level has been kept at a very high level and the energy cost of a commercial building has been reduced by more than 50% by turning off the HAVC system and the lighting when there are no present occupants in the building. The other method to enhance comfort is to let the RL agent controls the energy storage system and the HVAC system simultaneously. This method has been used in [61] and it has been able to improve comfort by 17% over the DDPG RL agent and reduced the power cost over a period of two months of evaluation by 1057 Japanese yen compared to a baseline method.

An algorithm called exploitation reinforced DDPG, which is a variation of DDPG, has been applied in [61] in which 10 iterations and 100.000 steps/hours for each iteration have been used. Moreover, different models have been trained and only the top 3 models have been utilized for the evaluation process. RLlib has been used for implementing the RL agent, OpenAI Gym has been used for the simulation, TEPCO's rate plan has been used for electricity prices, and Japan Meteorological Agency data has been used for environmental data. The proposed algorithm (double Q-learning) in [53] has been able to reduce the CO2 level by 10% and the energy by 4-5% while keeping an acceptable PMV value. Python has been used for training the DRL agent and EnergyPlus for simulation. The climate data from Taipei official site has been collected for 60 months to train the agent. The algorithm has been able to recognize complex

interactions between the thermal environments and air-conditioning systems which led to a good performance in energy, PMV, and CO2.

1.3 Contributions

1.3.1 Review Contributions

In contrast to the previous reviews, the scope of our review is broad enough to cover all areas of HEMS, including HEMS interfacing with the energy grid. More importantly, it provides a comprehensive overview of all major RL methods, providing a sufficient level of explanation for readers' understanding. Therefore, this article would be of benefit for researchers in other areas of the energy systems, and beyond, to acquire a theoretical level understanding of basic RL techniques.

1.3.2 Tutorial Contributions

This research explores avenues to automate environmental control by leveraging some of the latest advancements in machine learning. More specifically, it applies *deep attentive tabular neural networks* (TabNets) [62], [63] for the twofold purposes of controlling the temperature settings within a smart home environment as well as predicting the temperature and humidity levels arising from it. It is the first among all DNNs known to outperform XGBoost – the current best method to handle tabular data, for benchmark machine learning problems. Consequently, it is being investigated for a wide range of applications, such as Covid19 [64], hyperspectral imagery [65], and malware detection [66] and traffic prediction [67]. TabNet is based on the popular attention mechanism in deep learning [68].

Previous studies that have focused on using reinforcement learning as a method to replace human based control, are limited owing to the fact that human comfort is not easily quantifiable. This research takes an alternate route; it applies *imitation learning*, a paradigm that

is strongly connected to reinforcement learning, in order to simulate human behavior, while maintaining energy consumption at desirable levels. In treating humans as the ‘experts’, it eliminates the need to quantify occupant comfort during the learning process (which are only used as an assessment tool in this article). Recent research has investigated adopting imitation learning for several tasks ranging from dexterous robotics [69] to taxi driving [70]. Related energy applications include renewable energy sharing [71], smart grid load modeling [72], optimal scheduling [73], HVAC control [74], and distribution service restoration [75].

The research contributions of this study are along the following directions:

(*i*) To the best of my knowledge, this is the only smart home related research that uses TabNets, a DNN model that now outperforms all known machine learning algorithm for tabular data.

(*ii*) The research proposes applying imitation learning that has seldom been applied to environmental control in the smart home.

(*iii*) In using imitation learning, this research obviates the need for the difficult task of quantifying occupant comfort in the smart home.

(*iv*) TabNet models have been taken up for two different tasks of prediction and environmental control, and with separate, promising results for each.

The rest of this article is organized in the following manner. chapter 2 introduces basic ideas on reinforcement learning algorithms. Chapter 3 covers the various elements of HEMS in greater details. Chapter 4 explain the deep attentive tabular neural network (TabNet), describe the thesis problem and discuss the results. The thesis concludes in chapter 5.

Chapter 2 - Overview of Reinforcement Learning

2.1. Deep Neural Networks

A deep neural network (DNN) is a trainable highly nonlinear function approximator of the form $\mathbf{y}(\boldsymbol{\theta}): \mathcal{R}^M \rightarrow \mathcal{R}^N$ where M and N are the dimensionalities of the input and output spaces and $\boldsymbol{\theta}$ is its set of parameters. Structurally, the DNN consists of an input layer, and output layer, and at least one hidden layer. The input layer receives the DNN input vector $\mathbf{x}$. The neurons in any other layer receive as their inputs, the weighted outputs of neurons in the preceding layer. The weights of the DNN make up its weight parameter $\boldsymbol{\theta}$. For simplicity, we consider DNNs with scalar outputs so that $y(\boldsymbol{\theta}): \mathcal{R}^M \rightarrow \mathcal{R}$. The (true) output of the DNN is denoted as $y(\mathbf{x}|\boldsymbol{\theta})$, which is that of the sole neuron in the output layer.

In a typical regression application, the DNN's training set $\mathcal{S}$ consists of pairs $\big(\mathbf{x}(n), t(n)\big) \in \mathcal{S}$ where $n = 1, \ldots, |\mathcal{S}|$ is the sample index (for the sake of conciseness, this relationship is often denoted as $n \in \mathcal{S}$ in this article). The quantity $t(n)$ is the *target*, or desired output. During training, $\boldsymbol{\theta}$ is updated in steps so that for each input $\mathbf{x}(n)$, the DNN's output $y(n)$ is as close as possible to $t(n)$. Supervised learning algorithms aim to minimize the DNN's *loss function* $J_{\mathcal{S}}(\boldsymbol{\theta})$. The subscript $\mathcal{S}$ indicates that the loss is empirically estimated over the sample set $\mathcal{S}$. A popular choice of the latter is the sum of squared L_2 norms of the difference between the target and output for all samples in $\mathcal{S}$. In this case,

$$J_{\mathcal{S}}(\boldsymbol{\theta}) = \frac{1}{2}\frac{1}{|\mathcal{S}|}\sum_{n\in\mathcal{S}}\big(y(\mathbf{x}(n)|\boldsymbol{\theta}) - t(n)\big)^2. \tag{2.1}$$

Training the DNN comprises of multiple passes called epochs; each epoch comprising of one pass through all samples in $\mathcal{S}$. In *stochastic gradient descent*, with $\eta \ll 1$ being the *learning rate* the parameter $\boldsymbol{\theta}$ is incremented once for every sample $n \in \mathcal{S}$ as,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\big(y(\mathbf{x}(n)|\boldsymbol{\theta}) - t(n)\big)\nabla_{\boldsymbol{\theta}} y(\mathbf{x}(n)|\boldsymbol{\theta}). \tag{2.2}$$

This increment is equivalent to a single gradient step with $J(\boldsymbol{\theta}) = \frac{1}{2}\big(y(\mathbf{x}(n)|\boldsymbol{\theta}) - t(n)\big)^2$.

While SGD is useful in many online applications, *minibatch gradient descent* is the most common training method. In each epoch $\mathcal{S}$ are divided into non-overlapping minibatches $\mathcal{B}_k \subset \mathcal{S}$ (i.e. $\bigcup_k \mathcal{B}_k = \mathcal{S}$, and $k \neq l \Rightarrow \mathcal{B}_k \cap \mathcal{B}_l = \phi$). The parameter $\boldsymbol{\theta}$ is updated once for every minibatch $\mathcal{B}$ as,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{1}{|\mathcal{B}|} \sum_{n \in \mathcal{B}} \big(y(\mathbf{x}(n)|\boldsymbol{\theta}) - t(n)\big)\, \nabla_{\boldsymbol{\theta}} y(\mathbf{x}(n)|\boldsymbol{\theta}). \tag{2.3}$$

One of the advantages of training in minibatches is that the trajectory taken by the training algorithm is straightened out, thereby speeding up convergence. It can be seen that the loss function is $J_{\mathcal{B}}(\boldsymbol{\theta})$, which is identical to that in Eqn. (2.1), with sample set $\mathcal{B}$.

Typically, the loss function includes an additional regularization term designed to keep the weights in $\boldsymbol{\theta}$ low in order to prevent overfitting; overfitting results in poorer performance after the DNN is deployed into the real world. Nowadays, faster training is accomplished by using extensions of gradient descent such as ADAM. These as well as many other important aspects of DNNs and training algorithms have not be addressed here; the above discussion minimally suffices to understand how DNNs are used in reinforcement learning (RL). A brief exposition to DNNs is available at [76]. For a rigorous treatment of DNNs, the interested reader is referred to [77].

2.2. Reinforcement Learning

An *agent* in RL is a learning entity, such as a deep neural network (DNN) that exerts control over a stochastic, external *environment* by means of a sequence of *actions* over time. The agent learns to improve the performance of its environment using *reward* signals that are derived

from the latter. Rewards are quantitative metrics that indicate the immediate performance of the environment (e.g. average instantaneous user comfort). The sets $\mathbb{S}$ and $\mathbb{A}$ are the state and action spaces and can be discrete or continuous. Although RL can be extended to continuous domains, everywhere in this article it is assumed that all temporal signals are sampled at discrete, regularly spaced intervals. At each discrete time instance t, the current state $s_t \in \mathbb{S}$ of the environment is known to the agent, which then implements an action $a_t \in \mathbb{A}$. The environment transitions to the next state s_{t+1} with a probability $p(s_{t+1}|s_t, a_t)$ while returning an immediate reward signal $r_t \equiv r(s_t, a_t, s_{t+1})$; with $r: \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to \mathcal{R}$ denoting the environment's reward function that it unknown to the agent. The transition can be denoted concisely as $s_t \xrightarrow{a_t, r_t} s_{t+1}$. The overall schematic is depicted in Fig. 2.1 [76].

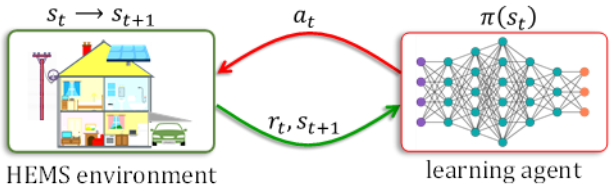

Figure 2.1. Reinforcement Learning. The quantities shown are associated with the transition $\boldsymbol{s_t \xrightarrow{a_t, r_t} s_{t+1}}$.

Instead of greedily aiming to improve the immediate reward r_t at every time instance t, the agent may be iteratively trained to maximize the sum of the immediate and the weighted future rewards, which is called the return,

$$R_t = r_t + \sum_{t'=1}^{T-t} \gamma^{t'} r_{t+t'} . \tag{2.4}$$

The quantity $\gamma \in [0,1]$ is called the discount factor. This lookahead feature prevents the agent to learn to implement greedy actions that fetch large instantaneous rewards, while adversely affect the environment later on. The process begins at time $t = 0$ and terminates at

time $t = T$, the time horizon. The environment's initial state at $t = 0$ is $s_0 \in \mathbb{S}$ which may be probabilistic, with an initial state distribution p_0. It should be noted that if $T = \infty$, then the discount must necessarily be less than unity ($\gamma < 1$) so that the return R_t is bounded at all times t. The 5-tuple $(\mathbb{S}, \mathbb{A}, p, r, \gamma)$ defines a Markov decision process (MDP). The initial distribution p_0 is assumed to be subsumed by the transition probabilities p. The MDP can be viewed as an extension of a discrete Markov model.

During this episode, the action taken by the agent is in accordance with a policy $\pi \in \Pi$, where Π is the policy space. From the Markovian (memoryless) property of the MDP, it follows that the optimal action of an agent at each state in terms of its stated goal of maximizing the total reinforcement, is independent of all previous states of the environment, i.e. its history. Therefore, the action a_t taken by the agent at time t under policy π is based solely on the state s_t and the prior history of states and actions need not be taken into account. The policy can be deterministic or stochastic. A deterministic policy can be treated as a function $\pi: \mathbb{S} \to \mathbb{A}$ (see Fig. 2.1) so that $a_t = \pi(s_t)$, whereas a stochastic policy π represents a probability distribution over $\mathbb{A}$ such that $a_t \sim \pi(s_t)$. In several domains, the probability distribution is determined from the nature of the application itself.

The entire sequence of states, actions, and rewards is an episode, denoted $\mathcal{E}$, so that,

$$\mathcal{E} \equiv s_0 \xrightarrow{a_0, r_0} s_1 \xrightarrow{a_1, r_1} s_2 \cdots \xrightarrow{a_{T-1}, r_{T-1}} s_T. \tag{2.5}$$

The overall objective of reinforcement learning is to maximize an objective function $J(\cdot)$. Let $J(\mathcal{E})$ denote the total return R_0 of a given episode $\mathcal{E}$. If the MDP is initialized to any state $s \in \mathbb{S}$ at $t = 0$ (such that $s_0 = s$), the expected value of this return which is dependent on policy π may be expressed as,

$$J^{\pi}(s) = \mathbb{E}_{\pi}[R_0 | s_0 = s]. \tag{2.6}$$

The operator $\mathbb{E}_\pi[\cdot]$ is the expectation when all episodes are generated by the MDP under policy π. When follows the MDP's initial state distribution, i.e. $s_0 \sim p$, the expectation may be denoted simply as J^π without any argument. This informal, function overloaded convention is adopted throughout this manuscript as there are other ways to define the objective function.

2.3. Taxonomy of Algorithms

RL methods can be classified in several ways. In on-policy RL, a referential policy π^*, such as that of a human, which is considered to be the optimal policy and is known a priori, and the goal is to learn a policy $\pi \cong \pi^*$. In off-policy approaches, the goal is to obtain the optimal policy π^* itself.

In model-free training, RL takes place with the agent connected to the real world environment, whereas in model-based RL, the agent is trained using a simulation platform to represent the environment. The classification of various approaches used in HEMS applications is shown in Fig. 2.2.

In model based RL, as the transition probabilities and the reward function are available, the algorithm can be implemented in an offline manner. Of more practical interest is online RL where the agent can be trained in real time by interacting with the real physical environment as shown in Fig. 2.1. Although online RL is considered to be model-free, practically all research papers report the use of HEMS models for training [40].

Another fundamental trichotomy of the plethora of RL approaches used today includes value-based RL, policy-based RL and actor-critic RL, the latter having emerged more recently. Actor-critic methods are hybrid approaches that borrow features from value-based as well as policy-based RL.

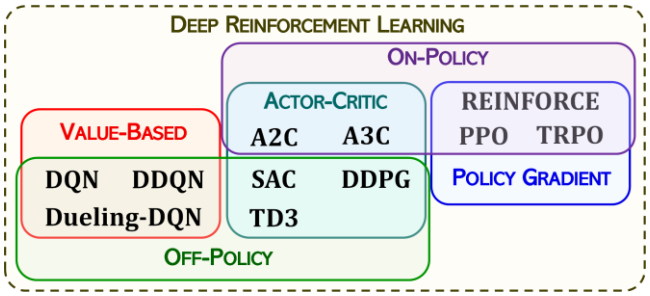

Figure 2.2. Taxonomy of Deep Reinforcement Learning. Classification of all deep reinforcement learning methods that are described in this article are shown. Also see [80].

2.4. Value Based Reinforcement Learning

Value based approaches are based on dynamic programming. Historically value-based RL, first proposed in [81], heralds the advent of the broad area of reinforcement learning as a distinct branch of AI. The formal definition of an MDP was introduced shortly thereafter [82], [83]. As noted earlier, an MDP is memoryless. An implication of this feature is that when the environment is in any given state $s_t = s$ at any instant t, the prior history $s_0 \xrightarrow{a_0,r_0} s_1 \xrightarrow{a_1,r_1} \cdots \xrightarrow{a_{t-1},r_{t-1}} s_t$ is not of any consequence in deciding the future course of actions. Accordingly, one can define the state-action value, or Q-value of the state s and for every available action a as the expected return from state s when implementing action a,

$$q(s,a) \triangleq \mathbb{E}_\pi[R_t|s_t = s, a_t = a]. \tag{2.7}$$

Referring to a specific policy π may be achieved by using a superscript in the above equation, so that the left hand side of Eqn. (2.7) is written as $q^\pi(s,a)$. Even under a deterministic policy, $q^\pi(s,a)$ can still be defined for any action $a \neq \pi(s)$ merely by treating a as an evaluative action and following the policy at all future times. The Q-value function can be expressed recursively in the following manner,

$$q^{\pi}(s,a) = \sum_{s' \in \mathbb{S}} p(s'|s,a) \left(r(s,a,s') + \gamma \sum_{a' \in \mathbb{A}} \pi(a'|s')\, q^{\pi}(s',a') \right). \tag{2.8}$$

The Q-value function $q^{\pi}: \mathbb{S} \times \mathbb{A} \to \mathcal{R}$ can be defined irrespective of whether the policy is deterministic or stochastic.

A stochastic policy is intrinsic to many real world applications. For instance, in order to decrease the ambient temperature by manually lowering the thermostatic setting, the final setting involves a degree of randomness arising from human imprecision. In multiagent environments, the best course may often be to adopt a stochastic policy. As an example, in a repeated game of rock-paper-scissors, randomly selecting each action ('rock', 'paper', or 'scissors') with equal probabilities of ⅓ is the only policy that would ensure that the probability of losing does not exceed that of winning.

From a machine learning standpoint, stochastic policies help explore and assess the effects of the entire repertoire of actions available in $\mathbb{A}$. Such exploration is critical during the initial stages of the learning algorithm. The two most commonly used stochastic policies are the ϵ-greedy and the softmax policies. Under an ϵ-greedy policy π, the probability of picking an action a when the environmental state s is given by,

$$\pi(a|s) = \begin{cases} \dfrac{\epsilon}{|\mathbb{A}|} + (1-\epsilon), & a = \underset{a' \in \mathbb{A}}{\operatorname{argmax}}\, q(s,a') \\ \dfrac{\epsilon}{|\mathbb{A}|}, & a \neq \underset{a' \in \mathbb{A}}{\operatorname{argmax}}\, q(s,a') \end{cases}. \tag{2.9}$$

It is always a good idea to lower the parameter ϵ steadily so that as learning progresses, the agent is greedier - being likelier to select actions with the highest Q-values, $\operatorname{argmax}_{a'} q(s,a')$.

The softmax policy is another popular method to incorporate exploration into a policy. The expression below is the probability of action a being selected under such a policy π,

$$\pi(a|s) = \left[\sum_{a' \in \mathbb{A}} e^{\frac{q(s,a')}{\tau}}\right]^{-1} e^{\frac{q(s,a)}{\tau}}. \tag{2.10}$$

Initialized to a high value, the Gibbs-Boltzmann parameter τ may be steadily lowered as the learning algorithm progresses, so that the policy becomes increasingly exploitative, that is take the action with the highest Q-value. Unless specified otherwise, it shall be assumed hereafter that the policy space Π is stochastic so that actions follow probability distribution ($a \sim \pi(s)$).

Instead of an evaluative action a, suppose the policy π is applied from state s (so that either $a = \pi(s)$ or $a \sim \pi(s)$), then the expected return is called the state's value,

$$v(s) \triangleq \mathbb{E}_{\pi}[R_t | s_t = s]. \tag{2.11}$$

As with the Q-value function, the policy π becomes explicit if the value of s is written as $v^{\pi}(s)$. The value of s can also be expressed in terms of Q-values as,

$$v(s) \equiv \sum_{a} \pi(a|s) q(s,a) = \mathbb{E}_{a \sim \pi(s)}[q(s,a)]. \tag{2.12}$$

The difference between value of any state s and the Q-value of implementing an action a from s under policy is the advantage function, so that,

$$A(s) \equiv q^{\pi}(s,a) - v^{\pi}(s). \tag{2.13}$$

Although the preferred notation in this manuscript is to use lowercase letters to denote variables, the advantage function is represented using uppercase as the lowercase a is reserved to denote an action.

The value function $v: \mathbb{S} \to \mathcal{R}$ allows the optimal policy π^* to be defined in a formal manner. If the objective function with the MDP initialized to some $s_0 \in \mathbb{S}$ is defined as in Eqn. (2.6), then it is evident from Eqn. (2.10) that $J^{\pi}(s_0) = v(s_0)$. Furthermore, if the MDP visits state $s_t = s$ at the instant t, then from Eqn. (2.4),

$$\mathbb{E}_{\pi}[R_0] = r_0 + \gamma r_1 + \cdots + \gamma^{t-1} r_{t-1} + \gamma^t v^{\pi}(s). \tag{2.14}$$

At this stage, we invoke the memoryless property of the underlying MDP. At instant t the partial sum of the terms $r_0 + \gamma r_1 + \cdots + \gamma^{t-1} r_{t-1}$ in the right hand side are part of the episode's history, while $\gamma^t v^{\pi}(s)$ is the expected future return. The optimal policy at every such state s is to adopt the one that maximizes $v^{\pi}(s)$, so that,

$$\pi^*(s) \triangleq \underset{\pi \in \Pi}{\operatorname{argmax}}\, v^{\pi}(s). \tag{2.15}$$

When the policy π^* is deterministic, it can also be inferred that the optimal action from state s is to select the action with the highest Q-value. From Eqn. (2.15) it directly follows that,

$$a^* \triangleq \underset{a \in \mathbb{A}}{\operatorname{argmax}}\, q^*(s, a). \tag{2.16}$$

The Q-value $q^*(s, a)$ is equal to $q^{\pi^*}(s, a)$. It can be established that the Q-values corresponding to the optimal policy are higher than those associated with other policies, i.e. $q^*(s, a) \geq q^{\pi}(s, a)$ [84].

The Bellman's equation for optimality follows from the above consideration,

$$q^*(s, a) = \sum_{s' \in \mathbb{S}} p(s'|s, a) \left(r(s, a, s') + \gamma \max_{a' \in \mathbb{A}} q^*(s', a') \right). \tag{2.17}$$

The difference between Eqn. (2.8) and Eqn. (2.17) is in the second term in each summand. The policy-based Q-value in Eqn. (2.8) is replaced with the maximum Q-value in Eqn. (2.17). A mathematically rigorous coverage of various RL methods can be found in the seminal book [85] that is available online.

2.4.1 Tabular Q-Learning

The simplest possible implementation of the Q-learning algorithm is tabular Q-learning where an $|\mathbb{S}| \times |\mathbb{A}|$ sized array is maintained to store $q(s, a)$ for every state-action pair [86]. Initialized to either zeros or small random values, the tabular entries are periodically updated. As

it is an online approach, Q-learning does not use transition probabilities ($p(s'|s,a)$). For each transition $s \xrightarrow{a,r} s'$, the tabular entry for $q(s,a)$ is incremented as show below,

$$q(s,a) \leftarrow (1-\eta)q(s,a) + \eta t. \tag{2.18}$$

The quantity η is the learning rate; usually $\eta \ll 1$. The quantity t is the target,

$$t = r + \gamma \max_{a' \in \mathbb{A}} q(s',a'). \tag{2.19}$$

In order to impart an exploratory component to Q-learning, the action a must be selected probabilistically as in Eqn. (2.9) or Eqn. (2.10). In many cases, increments are applied in real time at the end of each time instance. It can be shown mathematically that the tabular entries converge towards the maximum values, $q^*(s,a)$ [84], implying that Q-learning is an off-policy approach. The fully trained agent can select actions as per Eqn. (2.16) during actual use, with the reasonable expectation that the optimal policy is implemented.

SARSA (State-Action-Reward-State-Action) [87], [85] is the on-policy RL algorithm that can be implemented in a tabular manner. The update rule for SARSA is identical to the earlier expression in Eqn. (2.19). However, since SARSA is an on-policy algorithm, the target is policy specific and is given by,

$$t = r + \gamma q\big(s', \pi(s')\big). \tag{2.20}$$

Both Q-learning and SARSA use the tabular entries $q(s',a')$ of the environment's new state s' following the transition $s \xrightarrow{a,r} s'$. The difference is in how the entries are used. Whereas Q-learning uses the tabular entry corresponding to the action a' with the highest $q(s',a')$, SARSA applies the specified policy π, using $q(s',a')$, the Q-value of the action $a' = \pi(s')$. This difference is analogous to that between Eqn. (2.17) and Eqn. (2.8).

Tabular Q-learning and SARSA can handle continuous state as well as action spaces by discretizing them into a finite and tractable number of subdivisions. Unfortunately, these tabular

learning methods cannot be applied in many large scale application domains. Replacing tables with DNNs helps obviate this limitation. Such an arrangement is called a Deep Q-network (DQN), which is discussed next.

When the state space $\mathbb{S}$ is too large (e.g. $|\mathbb{S}| \approx 7.73 \times 10^{45}$ in chess), tabular learning becomes prohibitively expensive not only in terms of storage requirements but also in terms of computational time required by the training algorithm. DQNs, which are meant for off-policy training, are well equipped to handle such large discrete as well as continuous state spaces [88]. It must be noted that $|\mathbb{A}|$, the cardinality of the action space must be tractably small. In DQNs, the mapping from every state action pair (s, a) to its Q-value is carried out by means of DNNs.

In reality, the DNN input is some feature vector $\boldsymbol{\varphi}(s)$ of the state s, where $\boldsymbol{\varphi}: \mathbb{S} \rightarrow \mathcal{R}^D$ and D is the dimensionality of the feature space. In the same manner, the action a can be represented in terms of its feature vectors. However, as $|\mathbb{A}|$ is small, it is assumed that the action a itself is the other input. In practice, a unary encoding scheme may be used to represent actions. For instance, if $|\mathbb{A}| = 4$, the four discrete actions may be encoded as 0001, 0010, 0100, and 1000. Under these circumstances, the actual input to the DNN is $(\boldsymbol{\varphi}(s), a)$, and its actual output is $y(\boldsymbol{\varphi}(s), a|\boldsymbol{\theta})$. For simplicity we will treat the DNN input as (s, a) and the output as $q(s, a|\boldsymbol{\theta})$, that is, $(s, a) \equiv (\boldsymbol{\varphi}(s), a)$, and $q(s, a|\boldsymbol{\theta}) \equiv y(\boldsymbol{\varphi}(s), a|\boldsymbol{\theta})$. The Q-value $q(s, a|\boldsymbol{\theta})$ is conditioned in terms of the weight parameter $\boldsymbol{\theta}$ in this manner so as to explicitly reflect its dependence on the latter.

2.4.2 Deep Q-Networks

There are two possible ways in which the mapping of a state-action pair (s, a) to its Q-value $q(s, a|\boldsymbol{\theta})$ can be accomplished, which are as follows (see Fig. 2.3).

(*i*) A different DNN for each action is maintained, so that the total of DNNs is $|\mathbb{A}|$. The state s, encoded in a suitable manner, serves as the common input to all the DNNs.

(*ii*) A single DNN with separate inputs for state s and action a is maintained and it is output is $q(s, a|\boldsymbol{\theta})$. While this manner of storing Q-values requires the use of only a single DNN, in order to obtain $\max q(s, a|\boldsymbol{\theta})$, the actions must be applied sequentially to it.

Stochastic gradient descent can be applied in a straightforward fashion to train the weight parameter $\boldsymbol{\theta}$ as in Eqn. (2.2) for the squared error loss $\frac{1}{2}\big(t - q(s, a|\boldsymbol{\theta})\big)^2$, and with the DNN's output y now being $q(s, a|\boldsymbol{\theta})$,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta(q(s, a|\boldsymbol{\theta}) - t)\nabla_{\boldsymbol{\theta}} q(s, a|\boldsymbol{\theta}). \tag{2.21}$$

This simple approach is the neural-fitted Q-iteration (NFQ) that was proposed in [89]. The target $t(n)$ is determined in accordance with Eqn. (2.19) with $q(s', a'|\boldsymbol{\theta})$ used to obtain the target t so that $t = r + \gamma \max_{a' \in \mathbb{A}} q(s', a'|\boldsymbol{\theta})$.

When the learning agent interacts with the environment, the actions are generally selected using the ϵ-greedy method shown in Eqn. (2.9).

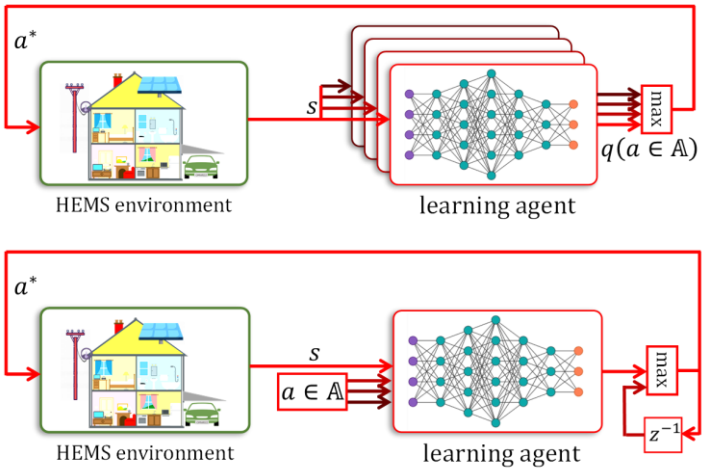

Figure 2.3. Deep Q-Network Layouts. One scheme uses a separate DNN for each action (top). uses only one DNN that receives actions as another input (bottom).

Temporal correlation in real-time training samples is an unfortunate drawback when directly implementing stochastic gradient descent. Unlike in tabular learning, in DQN updating $\boldsymbol{\theta}$ changes not only the output $q(s,a|\boldsymbol{\theta})$ for the relevant state-action pair (s,a) but the Q-values $q(s',a'|\boldsymbol{\theta})$ of every other pair $(s',a') \neq (s,a)$ as well. The change may be barely noticeable when s' is at a large distance from s within the feature space $\boldsymbol{\varphi}(\mathbb{S})$; unfortunately this is not usually the case in most real world domains. Consider two successive transitions $s_t \xrightarrow{a_t,r_t} s_{t+1} \xrightarrow{a_{t+1},r_{t+1}}$. Due to the property of temporal correlation between successive states, it is highly reasonable to expect that $\|\boldsymbol{\varphi}(s_t) - \boldsymbol{\varphi}(s_{t+1})\|$ is very small. Therefore, applying Eqn. (2.21) to update $q(s_t,a_t|\boldsymbol{\theta})$ will have an undesirable yet pronounced effect on $q(s_{t+1},a_{t+1}|\boldsymbol{\theta})$. A similar argument holds for time sequences of actions as well.

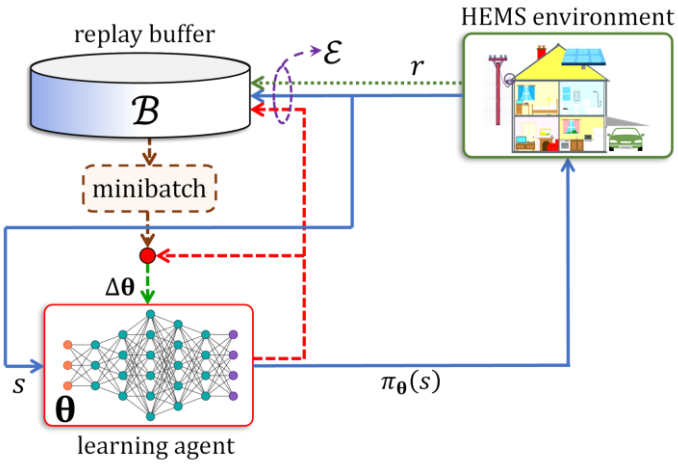

Figure 2.4. Replay Buffer. Shown are the replay buffer, the environment and the agent. The pathways are involved during the agent's interaction with the environment (solid blue) and training (dashed red).

To address the ill effects of temporal correlatedness, DNN training is carried out only after the completion of an entire episode or multiple episodes, during which time the DQN agent is allowed to exert control over the environment, while $\boldsymbol{\theta}$ remains unchanged. All training samples are stored in an experience replay buffer $\mathcal{B}$ [90], which plays the role of a minibatch

described in the previous section. After a large number of training samples are gathered in $\mathcal{B}$, it is shuffled randomly before incrementing $\boldsymbol{\theta}$ which may be carried out either as in Eqn. (2.21), or through minibatch gradient descent as indicated earlier in Eqn. (2.3) with $q(s,a|\boldsymbol{\theta})$ replacing $y(\mathbf{x}|\boldsymbol{\theta})$. For convenience, such a minibatch incremental rule is provided below,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{1}{|\mathcal{B}|} \sum_{n \in \mathcal{B}} \big(q(s(n),a(n)|\boldsymbol{\theta}) - t(n)\big) \nabla_{\boldsymbol{\theta}} q(s(n),a(n)|\boldsymbol{\theta}). \tag{2.22}$$

The buffer $\mathcal{B}$ is flushed before the next cycle begins with the updated parameter $\boldsymbol{\theta}$. An improvement over this scheme is prioritized replay [91], where the probability of a getting selected chosen for a training step is proportional to $\big(t - q(s,a|\boldsymbol{\theta})\big)^2 + \epsilon$. The relatively small constant $\epsilon > 0$ is added to the squared loss term to ensure that all samples have non-zero probabilities. Fig. 2.4 illustrates the use of a replay buffer.

Target non-stationarity is another closely related problem that arises in DQNs, one that is not seen in tabular Q-learning. For any given sample transition $s(n) \xrightarrow{a(n),r(n)} s'(n)$ as the DNN weight parameter $\boldsymbol{\theta}$ is incremented in accordance with Eqn. (2.21), an undesirable effect is that the target $t(n)$ also changes. This is because the same DNN is used to obtain $q(s'(n),a'|\boldsymbol{\theta})$. Target non-stationarity is handled by storing an older copy $\boldsymbol{\theta}^{\text{target}}$ of the primary DNN $\boldsymbol{\theta}$ in memory, effectively using a separate target DNN which is parametrized by $\boldsymbol{\theta}^{\text{target}}$. The target $t(n)$ is obtained using as,

$$t(n|\boldsymbol{\theta}^{\text{target}}) = r(n) + \gamma \max_{a' \in \mathbb{A}} q(s'(n),a'|\boldsymbol{\theta}^{\text{target}}). \tag{2.23}$$

The target DNN's weight parameter is updated infrequently, and only after $\boldsymbol{\theta}$ undergoes a significant amount of training. In this manner, the targets remain stationary when training the primary DNN's parameter $\boldsymbol{\theta}$ so that gradient descent steps can be implemented in a

straightforward manner using terms $\frac{1}{2}\left(q(s(n),a(n)|\boldsymbol{\theta}) - t(n|\boldsymbol{\theta}^{\text{target}})\right)^2$ in the loss function $J(\boldsymbol{\theta})$. This scheme is shown in Fig. 2.5.

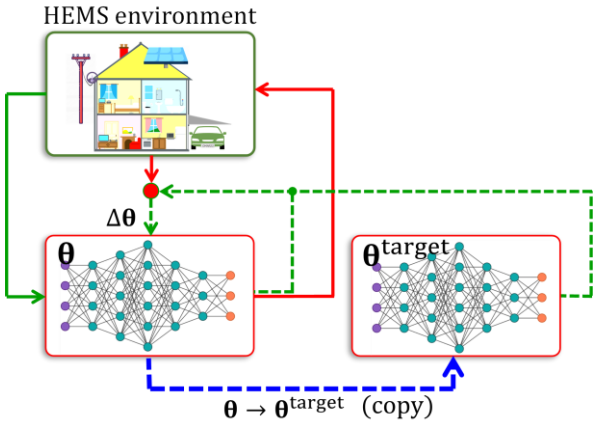

Figure 2.5. Use of Target Network. The scheme used to correct temporal correlatedness is shown. Pathways for control (solid red), learning (dashed green), and intermittent copying (dashed, thick blue) are shown. The replay buffer has been omitted for simplicity.

Overestimation bias [92], [93] is another problem frequently encountered in stochastic environments. This is an outcome of the maximization operation. As an example, consider an MDP with $\mathbb{S} = \{A, B, C\}$ where C is the terminal state. This is shown in Fig. 2.6. The action space $\mathbb{A} = \{a_k | k = 1,2,\ldots,N\}$ where N is relatively large, is available to the agent. From state A, only action a_N leads to B whereas the remaining ones a_1 thru a_{N-1} lead to C. The reward received from state A is always zero, (i.e. $r(A, a_k) = 0$. From state B all actions lead to C, with the reward being either -3 or $+1$ and with equal probabilities of N^{-1}. In other words, the possible transitions are $A \xrightarrow{a_N,0} B$, $A \xrightarrow{a_{k\neq N},0} C$, $B \xrightarrow{a_k, r\in\{-3,1\}} C$. Since the rewards of -3 and 1 have the same probability when the environment transitions from B to C, the expected reward from B to C is -1 i.e. $\mathbb{E}[r(B, a_k, C)] = -1$. For simplicity, let us assume that $\eta = 1$. The Q-values for some actions would be updated to -3, whereas those of others, to $+1$. Since N is large enough, it is very likely that at least one of them, say a' has the higher of the two. Consider the Q-values

of actions from state A. It is clear that for $k = 1$ through $N - 1$, $q(A, a_k) = 0$. However, when the agent selects action a_N from A, thereby reaching B, the operation $\max_{a \in \mathbb{A}} q(B, a)$ is likely to return $+1$ so that $q(A, a_N)$ would be updated to $\gamma \max_{a \in \mathbb{A}} q(B, a) = \gamma$. This makes a_N seem to be the optimal action from state A, when in fact it is the worst choice in $\mathbb{A}$.

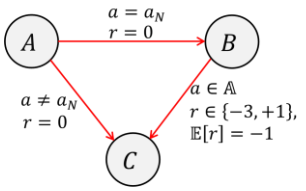

Figure 2.6. Overestimation Bias. This example is used to illustrate overestimation bias (see description for explanation).

Double Q-learning [94] is a popular approach to circumvent overestimation bias in off-policy RL (see Fig. 2.7). Although first proposed in a tabular setting [92], more recent research implements double Q-learning in conjunction with DNNs, which is called the double deep Q-network (DDQN). It incorporates two DNNs with the parameters $\boldsymbol{\theta}^1$ and $\boldsymbol{\theta}^2$. Samples are collected by implementing actions using their mean Q-values, $\frac{1}{2}\left(q(s, a|\boldsymbol{\theta}^1) + q(s, a|\boldsymbol{\theta}^2)\right)$. For each sample transition $s(n) \xrightarrow{a(n), r(n)} s'(n)$ in $\mathcal{B}$ during training, one of the two DNNs, say DNN j ($j \in \{1,2\}$), is picked randomly and with equal probability to compute the target, and the other DNN, $\bar{\jmath}$ is trained with it. Whence,

$$\begin{cases} t(n) = r(n) + \gamma \max_{a' \in \mathbb{A}} q\left(s'(n), a' | \boldsymbol{\theta}^j\right) \\ \boldsymbol{\theta}^{\bar{\jmath}} \leftarrow \boldsymbol{\theta}^{\bar{\jmath}} - \eta \sum_{n \in \mathcal{B}} \left(q(s(n), a(n) | \boldsymbol{\theta}^{\bar{\jmath}}) - t(n)\right) \nabla_{\boldsymbol{\theta}^{\bar{\jmath}}} q(s(n), a(n) | \boldsymbol{\theta}^{\bar{\jmath}}) \end{cases}. \tag{2.24}$$

This is the manner of updating that was originally proposed [92].

An extension of DDQN is clipped DDQN [95], [96], [97], where the target is obtained by taking the minimum of the Q-values, $q(s'(n), a'|\boldsymbol{\theta}^1)$ and $q(s'(n), a'|\boldsymbol{\theta}^2)$ instead of at random as before,

$$t(n) = r(n) + \gamma \min_{j\in\{1,2\}} \max_{a'\in\mathbb{A}} q\left(s'(n), a'|\boldsymbol{\theta}^j\right). \tag{2.25}$$

Each DNN has a 0.5 probability of getting trained with the transition sample.

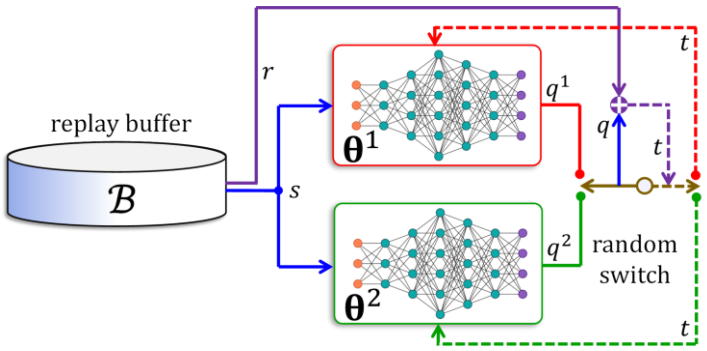

Figure 2.7. Double DQN. One DQN ($\boldsymbol{\theta}^1$ or $\boldsymbol{\theta}^2$) is picked at random and its Q value ($\boldsymbol{q}^1$ or $\boldsymbol{q}^2$) is used to obtain the target ($\boldsymbol{t}$), which is used to train the other DQN. For simplicity only the pathways involved in training are shown. The target pathways are depicted with dotted lines.

Dueling DQN architectures (Dueling-DQN) [98] use a different scheme to avoid overestimation bias (see Fig. 2.8). It divides the state-action value $q(s, a)$ into two parts, the state value $v(s)$ and the state-action advantage $A(s, a)$. As shown in Eqn. (2.13), $q(s, a)$ is the difference between the two quantities. The advantage of action a in state s, $A(s, a)$ is the expected gain in the return obtained by picking action a. The DNN architecture consists of an input layer for the state s. After a few initial preprocessing layers, it splits into two separate pathways, each of which is a fully connected DNN. Letting the symbols $\boldsymbol{\theta}^V$ and $\boldsymbol{\theta}^A$ denote the weight parameters of the pathways, the scalar output of the value pathway is the state's value, $v(s|\boldsymbol{\theta}^V)$ and the output of the advantage pathway is an $|\mathbb{A}|$ dimensional vector comprising of the

advantages $A(s,a|\boldsymbol{\theta}^A)$ of all available actions in $\mathbb{A}$. The Q-value of the state-action pair (s,a) can be obtained in a straightforward manner as,

$$q(s,a|\boldsymbol{\theta}) = v(s|\boldsymbol{\theta}^V) - |\mathbb{A}|^{-1}\sum_{a'} A(s,a'|\boldsymbol{\theta}^A)\,. \tag{2.26}$$

The quantity $\boldsymbol{\theta}$ denotes the set of all weight parameters of the dueling-DQN, including $\boldsymbol{\theta}^V$ and $\boldsymbol{\theta}^A$ as well as those present in the earlier preprocessing layers.

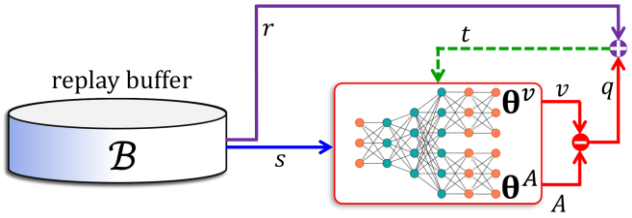

Figure 2.8. Dueling DQN. Shown is the dueling DQN architecture. The two outputs of the DNN are parametrized by $\boldsymbol{\theta}^{\boldsymbol{v}}$ and $\boldsymbol{\theta}^{\boldsymbol{A}}$. The target pathway (dotted green) is for training.

2.5. Policy Based and Actor-Critic Reinforcement Learning

Unlike the methods detailed in the preceding section, policy based RL does not require the estimates of the expected return from any given state. Initialized with an arbitrary policy π, tabular RL is an iterative process comprising of two steps [85], [99]. Policy evaluation is carried out in the first stage, where Q-values $q^{\pi}(s,a)$ are learned as shown in Eqn. (2.18) and Eqn. (2.20). In the second step, the policy is refined by defining the action for each state as shown in Eqn. (2.16). The two step process is repeated until the policy can be refined no further.

Although this kind of RL can be implemented in a tabular fashion, gradient descent methods do not draw upon it in the same way that value based learning did. These methods are realized through parametrized agents, e.g. DNNs. An attractive feature of deep policy RL is its intrinsic ability to handle continuous states as well as continuous actions.

2.5.1. Deep Policy Networks

Policy gradient uses an experience replay buffer $\mathcal{B}$ is the same manner as a DQN. The buffer stores full episodes of sequences. Instead of using Eqn. (2.6), it is convenient to directly express the loss function in terms of episodes $\mathcal{E}$ and the DNN's weight parameter $\boldsymbol{\theta}$, in the following manner,

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]. \tag{2.27}$$

Policy gradient methods try to maximize this loss. The operator $\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[\cdot]$ is the expected with the DNN agent operating under the probabilistic policy $\pi_{\boldsymbol{\theta}}$. The initial state s_0 in the above expression is implicitly defined in $\mathcal{E}$. Moreover, the distribution of s_0 within $\mathbb{S}$ is in accordance with the underlying MDP. The quantity $R(\mathcal{E})$ is the total return R_0 of episode $\mathcal{E}$ starting from $t = 0$.

Note that for a transition $s \xrightarrow{a} s'$, the reward $r(s, a, s')$ is a feedback signal that is determined by the environment (such as a home or residential complex) which is external to the agent. So is the discounted, aggregate return $R(\mathcal{E})$, which is also equal to that in Eqn. (2.4). No function $R: \mathbb{S}^T \times \mathbb{A}^T \to \mathcal{R}$ that maps a sequence of states and action of time horizon T to a return is available to the agent. Consequently, a straightforward gradient descent step in the direction of $\nabla_{\boldsymbol{\theta}} R(\mathcal{E})$ cannot be applied. In an apparent paradox, it turns out that its expected value $\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]$, can be differentiated by the agent, which is also the rationale behind expressing the loss as in Eqn. (2.27). This is due to a mathematical result known as the policy gradient theorem [100], [14], [101].

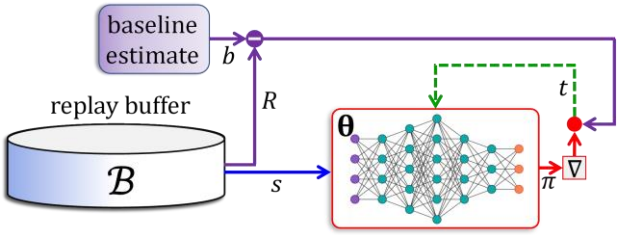

Figure 2.9. Policy Gradient with Baseline. Shown is the overall scheme used in REINFORCE with the baseline. There are different ways to implement the baseline.

The policy gradient theorem establishes the theoretical foundation for the majority of deep policy gradient methods. The theorem can be stated in mathematical terms as below,

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})] = \mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})\nabla_{\boldsymbol{\theta}}\log p(\mathcal{E}|\boldsymbol{\theta})]. \tag{2.28}$$

The significance of the theorem is that the gradient of the expected return $\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]$ does not require the use of the gradient of the return $R(\mathcal{E})$. Only the log probability of episode $\mathcal{E}$ must be differentiated. Fortunately, this gradient can readily be computed by the DNN agent. The probability of a transition $s_t \xrightarrow{a_t,r_t} s_{t+1}$ in $\mathcal{E}$ (see Eqn. (2.5)) is the product $\pi_{\boldsymbol{\theta}}(a_t|s_t)p(s_{t+1},r_t|s_t,a_t)$; its logarithm is $\log\pi_{\boldsymbol{\theta}}(a_t|s_t) + \log p(s_{t+1},r_t|s_t,a_t)$. The second term is intrinsic to the environment, and independent of the DNN so that differentiating it w.r.t. $\boldsymbol{\theta}$ is zero. Since $\log p(\mathcal{E}|\boldsymbol{\theta})$ is the product $p(s_0)\prod_t \pi_{\boldsymbol{\theta}}(a_t|s_t)p(s_{t+1},r_t|s_t,a_t)$, we arrive at the following interesting result,

$$\nabla_{\boldsymbol{\theta}}\log p(\mathcal{E}|\theta) = \frac{1}{T}\sum_t \nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\,. \tag{2.29}$$

The left hand side of Eqn. (2.28) to be estimated rather easily using this expression. This is because the policy $\pi_{\boldsymbol{\theta}}$ is, in fact, is based on the DPN output. Whereas [100] uses softmax policies as in Eqn. (2.10), it is quite usual in later research to adopt Gaussian policies [14]. Since $\pi_{\boldsymbol{\theta}}$ is the same policy that is used to obtain transition samples, Eqn. (2.28) can be used for on-policy learning.

The expected gradient $\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]$ can be estimated as the average of several Monte Carlo samples of episodes (also called rollouts) $\mathcal{E}(n), n = 1, \ldots, N$ that are stored in $\mathcal{B}$. This provides an estimate of the gradient of the loss function defined in Eqn. (2.27). An early policy gradient method, REINFORCE [88] uses Eqn. (2.29) to increment $\boldsymbol{\theta}$. The REINFORCE on-policy update rule is expressed as,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \frac{1}{|\mathcal{B}|T} \sum_{n\in\mathcal{B},t} \big(R\big(\mathcal{E}(n)\big) - b\big) \sum_{t} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\big(a_t(n)|s_t(n)\big). \tag{2.30}$$

In the above expression, it is assumed for simplicity that the time horizon is fixed across all N samples. The quantity b is called the baseline [102]. It can be set to zero in the basic implementation of policy learning.

Unfortunately, when $b = 0$ the variance in $R\big(\mathcal{E}(n)\big) \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\big(a_t(n)|s_t(n)\big)$ becomes too large. This in turn would require a large number of Monte Carlo episode samples. Including the baseline in Eqn. (2.30) that is close to $\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]$ helps reduce the variance to tractable limits. The theoretical optimal baseline estimate is given by,

$$b = \frac{\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})(\nabla_{\boldsymbol{\theta}} \log p(\mathcal{E}|\boldsymbol{\theta}))^2]}{\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[(\nabla_{\boldsymbol{\theta}} \log p(\mathcal{E}|\boldsymbol{\theta}))^2]}. \tag{2.31}$$

There are ways to obtain reasonable baseline estimates in practice that reduce the variance without affecting the bias [102], [103]. The purpose of actor critic architectures, which will be described subsequently, are also designed to obtain reliable bias estimates. Before proceeding further, we will make improvements to Eqn. (2.30) on the basis of the following two observations.

The first observation is that in Eqn. (2.30) the gradient $\sum \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\big(a_t(n)|s_t(n)\big)$ linked with $\mathcal{E}(n)$ is weighted by $R\big(\mathcal{E}(n)\big) - b$ in the outer summation. In this manner, the episode $\mathcal{E}(n)$

would receive a higher weight if it fetched a higher return. However the weighting scheme is rather arbitrary. For instance, with $b = 0$ if all returns were non-negative, then all gradients would receive positive weights. On the other hand, suppose the bias b were to be replaced with the expected return, then the gradients of the episodes with lower than expected returns would receive negative weights, whereas those with better than expected returns would be assigned positive weights. Using Eqn. (2.11) it is observed that the bias b is also the value of the starting state $v(s_0)$. Our first improvement would be to replace the bias with a value function.

The second observation is subtler, requiring the scrutiny of the weighting scheme at each time instant t. To simplify the discussion, it will be assumed that the discount $\gamma = 1$. Consider the episode $\mathcal{E}(n)$ consisting of transitions of the form $s_t(n) \xrightarrow{a_t(n), r_t(n)} s_{t+1}(n)$. Ignoring b, the corresponding term in the inner summation, which is $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\big(a_t(n)|s_t(n)\big)$, is weighted by the return $R\big(\mathcal{E}(n)\big) = r_0(n) + \cdots + r_{t-1}(n) + r_t(n) + \cdots + r_T(n)$. At time instant t, the prior rewards $r_0(n)$ until $r_{t-1}(n)$ represent the past history of the episode $\mathcal{E}(n)$; it has no role in how good the action $a_t(n)$ was from state $s_t(n)$. Removing these terms, the weight becomes $r_t(n) + r_{t+1}(n) + \cdots + r_T(n)$, which is, in fact, $q(s_t(n), a_t(n)|\boldsymbol{\theta})$. Replacing $R\big(\mathcal{E}(n)\big)$ in Eqn. (2.30) with $q(s_t(n), a_t(n)|\boldsymbol{\theta})$ is the other improvement. Once the prior history of the episode is removed, the bias must be set to $v(s_t(n)|\boldsymbol{\theta})$ instead of $v(s_0)$.

From the above discussion, it is seen that each factor $R\big(\mathcal{E}(n)\big) - b$ in Eqn. (2.30) should be replaced with $q(s_t(n), a_t(n)|\boldsymbol{\theta}) - v(s_t(n)|\boldsymbol{\theta})$. From Eqn. (2.13), this is the advantage function $A(s_t(n), a_t(n)|\boldsymbol{\theta})$, so that we replace the step original increment rule with the following,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \frac{1}{|\mathcal{B}|T} \sum_{n \in \mathcal{B}, t} A(s_t(n), a_t(n)|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\big(a_t(n)|s_t(n)\big). \tag{2.32}$$

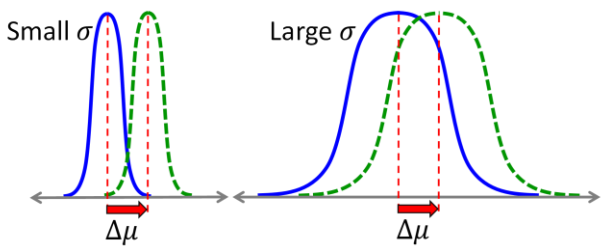

Figure 2.10. K-L Divergence. Two Gaussian distributions (solid, blue) with low $\boldsymbol{\sigma}$ (left) and high $\boldsymbol{\sigma}$ (right) are shown, and $\boldsymbol{\theta} = [\boldsymbol{\mu}, \boldsymbol{\sigma}]$. Incrementing $\boldsymbol{\mu}$ by $\Delta\boldsymbol{\mu}$ (dashed green) results in equal change in the norm $\|\Delta\boldsymbol{\theta}\|_{\mathbf{1}}$whereas $\boldsymbol{D_{KL}}(\boldsymbol{\pi_{\theta}}||\boldsymbol{\pi_{\theta+\Delta\theta}})$ is higher in the distribution in the left.

2.5.2. Natural Gradient Methods

One of the problems associated with policy gradient learning approaches as in Eqn. (2.30) or Eqn. (2.32) is choosing an adequate learning rate η. Too small a value of η would necessitate a large training period, whereas a value that is too large would produce a 'jump' in $\boldsymbol{\theta}$ large enough to yield a new policy that is too different from the previous one. Although there are several reliable methods to address this effect in gradient descent for supervised learning as in Eqn. (2.3), it is too pronounced in RL diminishing the efficacies of such methods. In extreme cases, a seemingly small increment along the direction of the gradient may lead to irretrievable distortion of the policy itself. The underlying reason behind this limitation is that unlike in Eqn. (2.1), the loss function in Eqn. (2.25) incorporates a probability distribution.

The change in any policy whenever a perturbation is applied to the parameter should not be quantified in terms of the norm $\left\|\boldsymbol{\theta} - \boldsymbol{\theta}^{\text{old}}\right\|$, but using the Kullback-Leibler divergence between to the distributions $\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}^{\text{old}}}$ [104]. The K-L divergence is denoted as $D_{KL}\left(\pi_{\boldsymbol{\theta}}||\pi_{\boldsymbol{\theta}^{\text{old}}}\right)$ where the new and previous values of the parameter are $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^{\text{old}}$. The Hessian (2nd derivative) of $D_{KL}\left(\pi_{\boldsymbol{\theta}}||\pi_{\boldsymbol{\theta}^{\text{old}}}\right)$ is known as the Fisher information matrix $\mathbf{F}_{\boldsymbol{\theta}}$. The increment $\Delta\boldsymbol{\theta}$ applied to $\boldsymbol{\theta}$ should be in proportion to $\mathbf{F}_{\boldsymbol{\theta}}^{-1}\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]$, which is referred to as the natural

gradient [105] of the expectation$\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]$. The Fisher information matrix can be estimated as [14],

$$\mathbf{F}_{\boldsymbol{\theta}} \approx \nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]^{\mathrm{T}}\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]. \tag{2.33}$$

Recent policy gradient algorithms use concepts derived from natural gradients [14] to rectify the downside of 'vanilla' gradient descent to eliminate the use of an effective learning rate η. The use of the natural gradient greatly reduces the natural gradient algorithm's dependence on how the policy is parametrized. Unfortunately, the gains of using the natural gradient come at the cost of increased computational overheads associated with matrix inversion. The overheads may outweigh the gains when the Fisher matrix $\mathbf{F}_{\boldsymbol{\theta}}$ is very large. When the policy is represented effectively through the parameter $\boldsymbol{\theta}$ natural gradient becomes unnecessary.

Trust region policy optimization (TRPO) is a class of training algorithms that directly uses the Kullback-Leibler divergence [106]. In TRPO, a hard upper bound is imposed on this divergence produced due to the increment $\Delta\boldsymbol{\theta}$ is applied to the DNN weights $\boldsymbol{\theta}$. Denoting this bound as ε,

$$D_{KL}\left(\pi_{\boldsymbol{\theta}}||\pi_{\boldsymbol{\theta}^{\mathrm{old}}}\right) \leq \varepsilon. \tag{2.34}$$

Under these circumstances it can be shown that the parametric increment in TRPO is,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \left(\frac{2\varepsilon}{\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]^{\mathrm{T}}\mathbf{F}_{\boldsymbol{\theta}}^{-1}\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]}\right)^{\frac{1}{2}}\mathbf{F}_{\boldsymbol{\theta}}^{-1}\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]. \tag{2.35}$$

The expectation $\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathcal{E}\sim\pi_{\boldsymbol{\theta}}}[R(\mathcal{E})]$ can be estimated in the same manner as in Eqn. (2.30) or Eqn. (2.32).

Proximal policy optimization (PPO) [107] is another RL method that uses natural gradients. PPO replaces the bound in TRPO with a penalty term. An expression for the PPO's objective function for a single episode is as shown below,

$$J(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t}\frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}^{\text{old}}}(a_t|s_t)}A(s_t, a_t|\boldsymbol{\theta}) - \lambda D_{KL}\left(\pi_{\boldsymbol{\theta}}||\pi_{\boldsymbol{\theta}^{\text{old}}}\right). \quad (2.36)$$

2.5.3. Off-Policy Policy Learning

So far in this section, only on-policy algorithms have been discussed. This includes TRPO and PPO. Nevertheless, policy gradient can also be applied for off-policy learning. Since such an algorithm would be trained for the optimal policy, the samples in the replay buffer (that are collected using earlier policies) can be recycled multiple times. This aspect is a significant advantage of off-policy learning.

Policy gradient methods for off-policy learning can be implemented using the popular statistical technique of importance sampling. Suppose $f(x)$ is any function of a random variable x that follows an arbitrary distribution $q(x)$, importance sampling can be used to obtain an estimate of the expectation $\mathbb{E}_{x\sim q}[f(x)]$. Samples x are drawn from a more tractable distribution $p(x)$ and $q(x)^{-1}p(x)$ as sample weights, the weighted expectation $\mathbb{E}_{x\sim p}[p(x)^{-1}q(x)f(x)]$ is computed from several such samples. It serves as the estimated value i.e. $\mathbb{E}_{x\sim q(x)}[f(x)] \approx \mathbb{E}_{x\sim p}[p(x)^{-1}q(x)f(x)]$. This approach is adopted in off-policy RL. Suppose samples in the replay buffer are based on the policy $\pi_{\boldsymbol{\theta}}$. The gradient can be empirically estimated as,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{a\sim p}\left[\frac{\pi_{\boldsymbol{\theta}}(a)}{p(a)}A(s, a|\boldsymbol{\theta})\right]. \quad (2.37)$$

2.5.4. Actor-Critic Networks

Actor-critic methods combine policy gradient and value-based RL methods [108]. The actor-critic architecture consists of two learning agents, the actor and the critic. From any environmental state, the actor is trained using policy gradient to respond with an action. The critic is trained with a value-based RL method to evaluate the effectiveness of the actor's output, which is then used to train the latter.

Let us denote their parameters using the symbols $\boldsymbol{\theta}^a$ and $\boldsymbol{\theta}^c$. The critic network can be modeled as a DQN, although in this explanation it is trained with the advantage function as defined in Eqn. (2.13). When the environmental state is $s_t(n)$, for every action $a_t(n)$ critic network provides as its output, the value of the state-action pair $q(s_t(n), a_t(n)|\boldsymbol{\theta}^c)$. The actor is incremented using gradient ascent,

$$\boldsymbol{\theta}^a \leftarrow \boldsymbol{\theta}^a + \eta^a \frac{1}{T} \sum_t q(s_t(n), a_t(n)|\boldsymbol{\theta}^c) \nabla_{\boldsymbol{\theta}^a} \log \pi_{\boldsymbol{\theta}^a}(a_t(n)|s_t(n); \boldsymbol{\theta}^a) \,. \tag{2.38}$$

Eqn. (2.38) closely resembles the policy gradient increment as in Eqn. (2.30) (with $b = 0$). The only difference is that the weights applied to each gradient is determined by the critic. The update rule for the critic network, which is similar to that is Eqn. (2.22), is shown below,

$$\boldsymbol{\theta}^c \leftarrow \boldsymbol{\theta}^c - \eta^c \frac{1}{T} \sum_t \big(q(s_t, a_t|\boldsymbol{\theta}^c) - (r_t + q(s_{t+1}, a_{t+1}|\boldsymbol{\theta}^c)\big) \nabla_{\boldsymbol{\theta}^c} q(s_t, a_t|\boldsymbol{\theta}^c) \,. \tag{2.39}$$

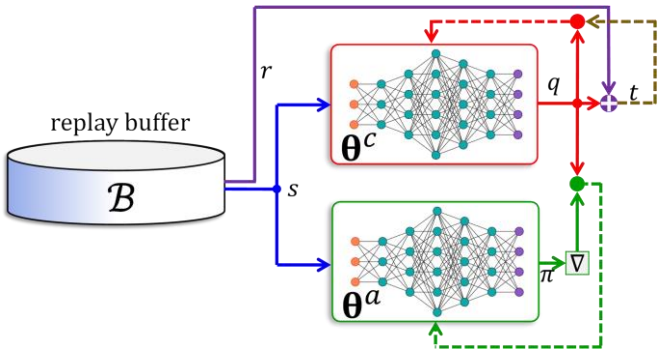

Figure 2.11. Actor-Critic. Shown is the overall scheme used in actor-critic learning.

The advantage actor critic (A2C) algorithm [109] is very effective in reducing the variance in the policy gradient algorithm of the actor. The A2C architecture entails improvements over the above 'vanilla' actor critic method in two directions in the following manner.

(*i*) The actor network uses an advantage function $A(s_t, a_t)$, which is the difference between a return value R and the value of state $v(s_t|\boldsymbol{\theta}^c)$. Accordingly, the critic is trained to approximate the value function.

(*ii*) Computing the reward R uses an τ-step lookahead feature, where the log-gradient is weighted using the sum of the next τ rewards.

To better understand how the τ-step lookahead works, let us turn our attention to Eqn. (2.30). In this expression the gradient $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t)$ at time instant t is weighted by the factor $(R(\mathcal{E}) - b)$ where $R(\mathcal{E})$ is the return of an entire episode from $t = 0$ until $T - 1$, so that $R(\mathcal{E}) = r_0 + \gamma r_1 + \cdots \gamma^t r_t + \cdots \gamma^{t+\tau} r_{t+\tau} + \cdots \gamma^{T-1} r_{T-1}$. The baseline is b is the value of $v(s_t|\boldsymbol{\theta}^C)$. It is reasoned that the sum of the past rewards $r_0 + \gamma r_1 + \cdots \gamma^{t-1} r_{t-1}$ does not have any bearing on the quality of the action a_t taken at the instant t. Hence all past rewards are dropped from R. Furthermore, rewards received in the distant future, i.e. after τ instants are also dropped. In other words, R consists of the sum of the discounted rewards between the instant t and the instant $t + \tau$. Whereupon, the actor's update rule is expressed as,

$$\boldsymbol{\theta}^a \leftarrow \boldsymbol{\theta}^a + \eta^a \frac{1}{T} \sum_{t=0}^{T-\tau} \nabla_{\boldsymbol{\theta}^a} \log \pi_{\boldsymbol{\theta}^a}(a_t|s_t) \left(\sum_{t'=0}^{\tau} r_{t+t'} - v(s_t|\boldsymbol{\theta}^c) \right). \tag{2.40}$$

The critic is updated using the same return R, so that,

$$\boldsymbol{\theta}^c \leftarrow \boldsymbol{\theta}^c - \eta^c \frac{1}{T|\mathcal{B}|} \sum_{t=0}^{T-\tau} \left(v(s_t|\boldsymbol{\theta}^c) - \sum_{t'=0}^{\tau} r_{t+t'} \right) \nabla_{\boldsymbol{\theta}^c} v(s_t|\boldsymbol{\theta}^c) \,. \tag{2.41}$$

The asynchronous advantage actor critic (A3C) method [109] is an extension of A2C that can be applied in parallel processing environments. A global network and a set of 'workers' are maintained in A3C. Each worker receives the actor and critic parameters that it implements on its own independent environment and collecting reward signals. The rewards are then used to

determine increments $\Delta\boldsymbol{\theta}^{A}$ and $\Delta\boldsymbol{\theta}^{C}$, which are then used to asynchronously update the parameters in the global network. An advantage of A3C is that due to the parallel action of multiple workers, an experience replay buffer does not have to be incorporated.

The deterministic policy gradient (DPG) algorithm was described in [100], and more recently in 113]. It was later extended to a deep framework in [110], known as the deep deterministic policy gradient (DDPG). DDPG is an off-policy actor critic method that concurrently learns the optimal Q-function q^*, as well as the optimal policy π^*.

In any off-policy actor critic model the critic must be trained to output the optimal policy π^*. Hence, the term $q(s_{t+1}, a_{t+1})$ in Eqn. (2.39) should be replaced with the maximum over all actions $q(s_{t+1}, a^*)$, where the optimal action $a^* = \underset{a \in \mathbb{A}}{\operatorname{argmax}}\, q(s_{t+1}, a)$ as in Eqn. (2.19). Unfortunately, when the action space $\mathbb{A}$ is continuous ($|\mathbb{A}| = \infty$), neither an exhaustive search to find a^* nor maximizing numerical, feasible from a computational perspective.

In order to circumvent these difficulties in identifying the optimal action that maximizes the Q-value, there three options available for use. These are outlined below.

(*i*) $q(s_{t+1}, a)$ can be sampled for several different actions and a^* be assigned the action corresponding to the sample maximum [111].

(*ii*) A convex approximation of $q(s, a)$ around s_{t+1} can be devised and a^* obtained over the approximate function [112].

(*iii*) A separate off-policy policy network can be used to learn the optimal policy π^* [113].

Out of the above three available options discussed above, the third and last has been adopted in DDPG. The critic parameter $\boldsymbol{\theta}^{c}$ is updated in accordance with the expression shown below,

$$\boldsymbol{\theta}^c \leftarrow \boldsymbol{\theta}^c - \eta^c \frac{1}{T|\mathcal{B}|} \sum_{n\in\mathcal{B},t} \begin{pmatrix} r_t + q(s_{t+1}(n), a_{t+1}(n)|\boldsymbol{\theta}^c) - \\ q\big(s_t(n), \pi(s_t(n)|\boldsymbol{\theta}^a)\big) \end{pmatrix} \nabla_{\boldsymbol{\theta}^c} q(s_t(n), a_t(n)|\boldsymbol{\theta}^c). \quad (2.42)$$

In Eqn. (2.42), $\pi(s_t(n)|\boldsymbol{\theta}^a)$ is the output of DDPG's actor. DDPG uses a replay buffer $\mathcal{B}$ that includes samples from older policies. The actor's parameter $\boldsymbol{\theta}^a$ is trained using any off-policy policy gradient as in Eqn. (2.37).

One of the drawbacks of DDPG is the problem of overestimation [114]. If the function $q(s, a|\boldsymbol{\theta}^c)$ acquires a sharp local peak, further training converges towards the latter, leading to undesirable results. This issue has been tackled by twin delayed deterministic policy gradient (TD3) in [95]. TD3 maintains a pair of critics whose parameters we shall denote as $\boldsymbol{\theta}^{c_1}$ and $\boldsymbol{\theta}^{c_2}$, or more concisely as $\boldsymbol{\theta}^{c_i}$, where $i \in \{1,2\}$.

In Eqn. (2.42), it can be seen that DDPG has a target $r_t + q(s_{t+1}(n), a_{t+1}(n)|\boldsymbol{\theta}^c)$ where $q(s_{t+1}(n), a_{t+1}(n)|\boldsymbol{\theta}^c)$ is obtained from the critic. TD3 has two targets, $q(s_{t+1}(n), a_{t+1}(n)|\boldsymbol{\theta}^{c_i})$, $i \in \{1,2\}$. The actions $a_{t+1}(n)$ in TD3 are clipped to lie within the interval $[a_{\min}, a_{\max}]$. In order to increase exploration, Gaussian noise is added to this action. Finally, the target is obtained as $r_t + \min_{i\in\{1,2\}.} q(s_{t+1}(n), a_{t+1}(n)|\boldsymbol{\theta}^{c_i})$, which is used for training.

The soft actor critic (SAC) RL proposed recently in [96], [115] is an off-policy RL approach. The striking feature of SAC is the presence of an entropy term in the objective function,

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[r_t + \alpha H(\pi_{\boldsymbol{\theta}}|s_t)]. \quad (2.43)$$

Incorporating the entropy $H(\pi_{\boldsymbol{\theta}}|s_t)$ increases the degree of randomness in the policy which helps in exploration. As in TD3, SAC uses two critic networks.

2.6. Imitation Learning

Imitation learning aims to accomplish the reverse of reinforcement learning. The buffer contains samples that are obtained externally from an expert and without the agent's involvement. Such an expert may represent an actual human or multiple humans. It may be assumed that the expert applies the optimal policy $\pi^*(\cdot)$. In such a case, the samples in the buffer are of the form (s_t, a_t^*, s_{t+1}), where actions $a_t^* \sim \pi^*(s_t)$. The difference between reinforcement learning and imitation learning is highlighted in Fig. 2.12.

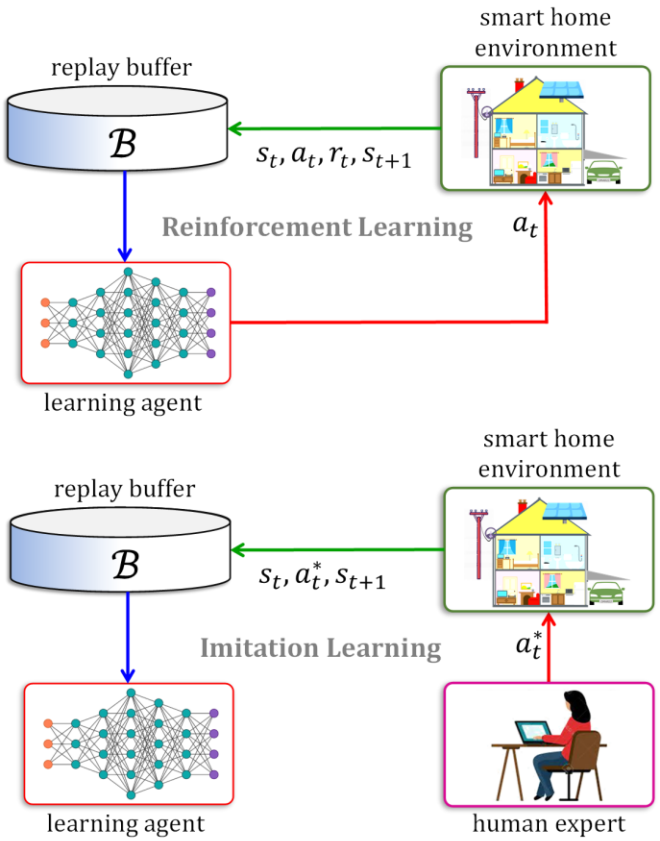

Figure 2.12. Learning Paradigms. Shown for comparison are the schematics in reinforcement learning (top) and imitation learning (bottom) paradigms. Arrows show how data is shared during training.

Chapter 3 - Home Energy Management Systems

3.1. HEMS Enabling Technologies

3.1.1. Enabling Networking & Communication Technologies

All HEMS devices must have the ability to send/receive data with each other using the same communication protocol. HEMS provides the occupants with the tools that allow them to monitor, manage and control all the activities within the system. The advancements in technologies and more specifically in IoT-enabled devices and wireless communications protocols such as ZigBee, Wi-Fi, and Z-Wave made HEMS feasible [116], [117]. These smart devices are connected through a home area network (HAN) and/or to the internet, i.e. a wide area network (WAN).

ZigBee and Z-Wave are the two dominant communication protocols in today's home automation market. Z-Wave is better than ZigBee with the range of transmission (120m with three devices as repeaters vs 60m with two devices working as repeaters). When it comes to inter-brand operability Z-Wave still has the advantage over ZigBee. However, ZigBee win the competition when it comes to the data rate of transmission and the maximum number of connected devices. Both protocols share many common features like they both use RF communication mode, and both are two-way communication protocols. Z-Wave is specially founded for home automation applications, while ZigBee is used in a wider range of places such as industrial, research, health care, and home automation [118]. Both protocols have established various connections with different companies and tens of hundreds of smart devices are using one of these protocols. A study done by [119] concluded that ZigBee is most likely to be the standard communication protocol for HEMS. However, it is still difficult to tell with high

certainty what the study concluded is going to happen as different factors might affect this conclusion or even a new player might emerge and prevail.

HEMS needs this level of connectivity to be able to get the electricity price from the smart grid through the smart meter and control all the system's elements accordingly (e.g., turn on/off the TV, set the thermostat settings, charge/discharge the battery., etc.). In some scenarios, HEMS uses the forecasting electricity prices to schedule the shiftable loads (e.g., washing machine, dryer, electric vehicle charging) [116].

3.1.2. Enabling Sensors & Controller Platforms Technologies

HEMS consists of smart appliances with sensors, these IoT-enabled devices communicate with the controller by sending and receiving data. They collect information from the environment and/or about their electricity usage using built-in sensors. The smart meter gathers information regarding the total consumers' consumption from the appliances, the peak load period, and electricity price from the smart grid.

The controller can be in the form of a physical computer located within the premises, that is equipped with the ability to run complex algorithms. An alternate approach is to leverage any of the cloud services that are available to the consumers through cloud computing firms.

The controller gathers information from the following sources: (*i*) the energy grid through the smart meter, which includes the power supply status and electricity price, (*ii*) the status of renewable energy and the energy storage systems, (*iii*) the electricity usage of each smart device at home, and (*iv*) the outside environment. Then process all the data through a computational algorithm to take specific action for each device in the whole system separately [5].

3.1.3. Control Algorithms

AI and machine learning methods are making deep inroads into HEMS [120], [10]. HEMS algorithms incorporated into the controller might be in the form of simple knowledge-based systems. These approaches embody a set of if-then-else rules, that may be crisp or fuzzy. However, due to their reliance on a fixed set of rules, such methods may not be of much practical use with real-time controllers. Moreover, they cannot effectively leverage the big amount of data available today [5]. Although it is possible to impart a certain degree of trainability to fuzzy knowledge based systems, the structural bottleneck of consolidating all inputs using only conjunctions (and) and disjunctions (or) still persists.

Numerical optimization comprises of another class of computational methods for the smart home controller. These methods entail an objective function that is to be either minimized (e.g. cost) or maximized (e.g. occupant comfort), as well as a set of constraints imposed by the underlying physical HEMS appliances and limitations. Due to its simplicity, linear programing is a popular choice for this class of algorithms. More recently, game theoretic approaches have emerged as an alternative approach for HEMS optimization problems [5].

In recent years, artificial intelligence and machine learning, more specifically deep learning techniques have become popular for HEMS applications. Deep learning takes advantage of all the available data for training the neural network to predict the output and control the connected devices. It is so helpful to forecast the weather, load, and electricity price and it can solve a non-linear problem without using any mathematical models. Reinforcement learning is a class of machine learning algorithms that consists of trainable decision-making agents that interact with the environment and the consumer to adjust the end action. Since 2013, there have

been significant efforts directed at using deep neural networks within a reinforcement learning framework [121], [122], that has met with great success.

3.2. RL Usage in HEMS

3.2.1. Application Classes

There are many elements of HEMS that can effectively leverage RL techniques. These fall into the broad classes (see Fig. 3.1) that are outlined below.

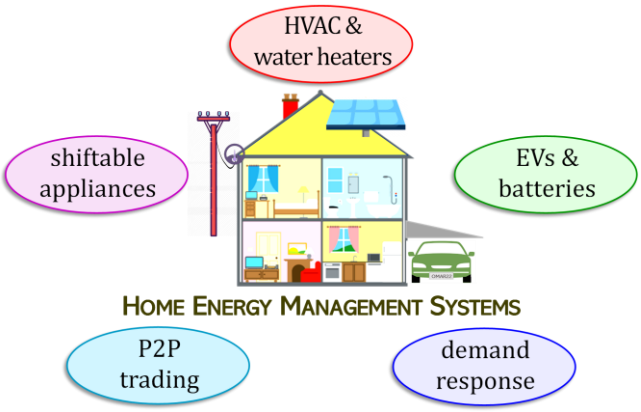

Figure 3.1. Classes of Applications. All applications of reinforcement learning in home energy management systems are classified into the five categories shown.

(*i*) *Heating, Ventilation & Air Conditioning, Fans and Water Heaters*: Heating, ventilation, and air conditioning (HVAC) systems alone are responsible for about half of the total electricity consumption [39], [123], [124], [125], [126]. In this survey, HVAC, fans and WH have been placed under a single category. Effective control of these loads is a major research topic in HEMS.

(*ii*) *Electric Vehicles, Energy Storage, & Renewable Generation*: The charging of electric vehicles (EVs) and energy storage (ES) devices, i.e. batteries are studied in the literature as in [127], [128]. Wherever applicable, EV and ES must be charged in coordination with renewable generation (RG) such as solar panels and wind turbines. The aim is to make decisions in order to

save energy costs, while addressing comfort and other consumer rsuirements. Thus, EV, ES, and RG have been placed under a single class for the purpose of this survey.

(*iii*) *Other Loads*: Suitable scheduling of several home appliances such as dishwasher, washing machine, etc. can be achieved through HEMS to save energy usage or cost. Lighting schedules are important in buildings with large occupancy. These loads have been lumped into a single class.

(*iv*) *Demand Response*: With the rapid proliferation of green energies into homes and buildings, and these sources merged into the grid, demand response (DR) has acquired much research significance in HEMS. DR programs help in load balancing, by scheduling and/or controlling shiftable loads and in incentivizing participants [129], [130] to do so through HEMS. RL for DR is one of the classes in this survey.

(*v*) *Peer-to-Peer Trading*: Home energy management has been used to maximize the profit for the prosumers by trading the electricity with each other directly in peer-to-peer (P2P) trading or indirectly through a third party as in [131]. Currently, theoretical research on automated trading is receiving significant attention. P2P trading is the last class that is considered here.

Fig. 3.2 shows the number of research articles that applied RL to each class. Note that a significant proportion of these papers addressed more than one class. More than third of the papers we reviewed focused only on Heating, Cooling and Ventilation. Just above 10% of the papers studied RL control for the energy storage (ES) systems. Only 7% of the papers focused on the energy trading. However, most of the papers (46%) are targeting more than one object. These results are shown in Fig. 3.3.

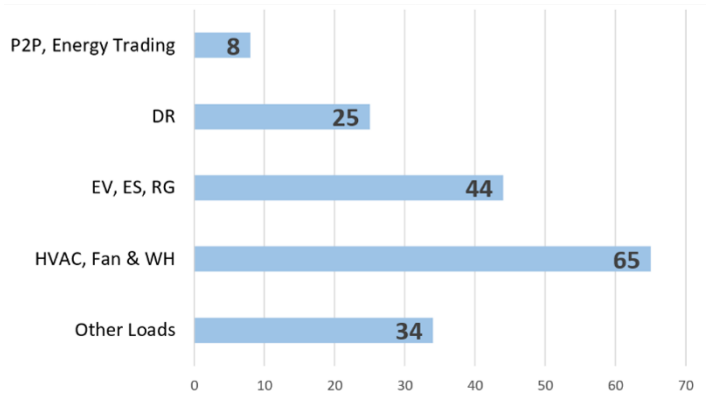

Figure 3.2. Number of Articles in each Application Class.

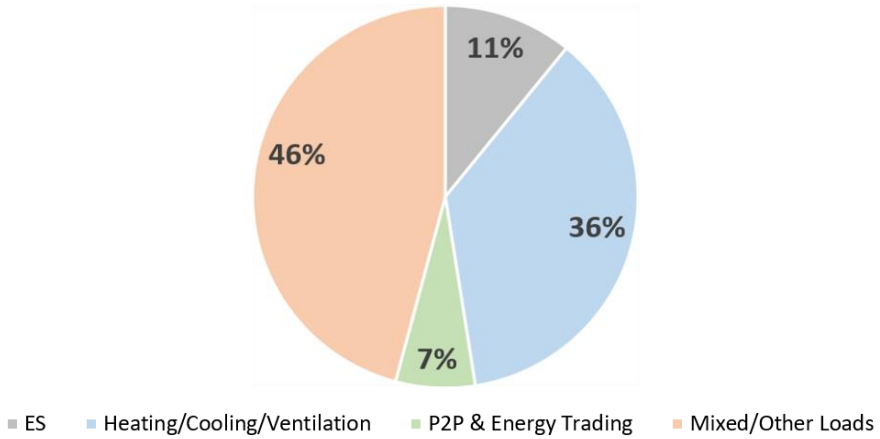

Figure 3.3. Proportion of Articles in each Application Class.

3.2.2. Objectives

Within these HEMS applications, RL has been applied in several ways. It has been used to reduce energy consumption within residential units and buildings [132]. It has also been used to achieve a higher comfort level of occupants [133]. In operations at the interface between the residential units and the energy grid, RL has been applied to maximize prosumers profit in energy trading as well as for load balancing. For this purpose, we break down the objectives into three different types as listed below.

(*i*) *Energy Cost*: The cost of using any electrical device by the consumer and in most of the cases it is proportionally related to energy consumption. In this paper we use cost and consumption interchangeably.

(*ii*) *Occupant Comfort*: The main factor that can affect the occupant's comfort is the thermal comfort, which depends mainly on the room temperature and humidity.

(*iii*) *Load Balance*: Power supply companies try to achieve load balance by reducing the power consumption of consumers at peak periods to match the station power supply. The consumers are motivated to participate in such programs by price incentives.

Fig. 3.4 illustrates the outcome of this survey. It may be noted that in the largest proportion of articles (42%) the RL algorithm took into account both cost and comfort. About a quarter of the remaining addressed cost alone forming the second largest proportion.

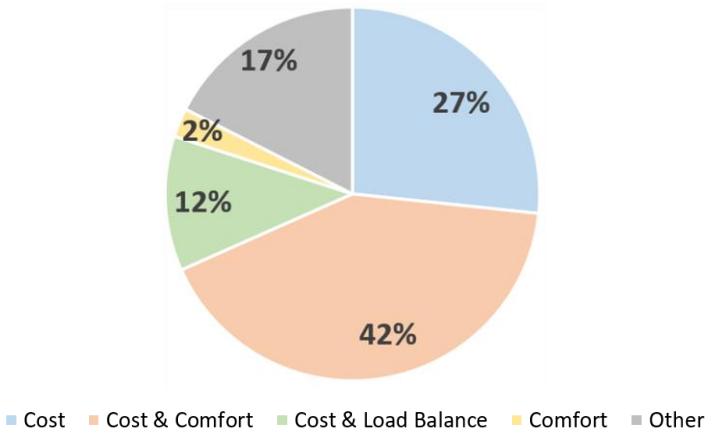

Figure 3.4. Proportion of Articles for each Objectives.

3.2.3. Building Types

(*i*) *Residential*: For the purpose of this survey, individual homes, residential communities, as well as apartment complexes fall under this type of buildings.

(*ii*) *Commercial*: These buildings include offices, office complexes, shops, malls, hotels, as well as industrial buildings.

(*iii*) *Academic*: Academic buildings range from schools, university classrooms, buildings, research laboratories, up to entire campuses.

The proportion of articles for each building is shown in Fig. 3.5. In residential buildings, the research literature revealed that RL was applied in all five application classes shown in Fig. 3.1. However, in case of commercial and academic buildings, RL was typically applied to the first three categories, i.e. to HVAC, fans & WH, to EVs, ESs & RGs, as well as to other loads.

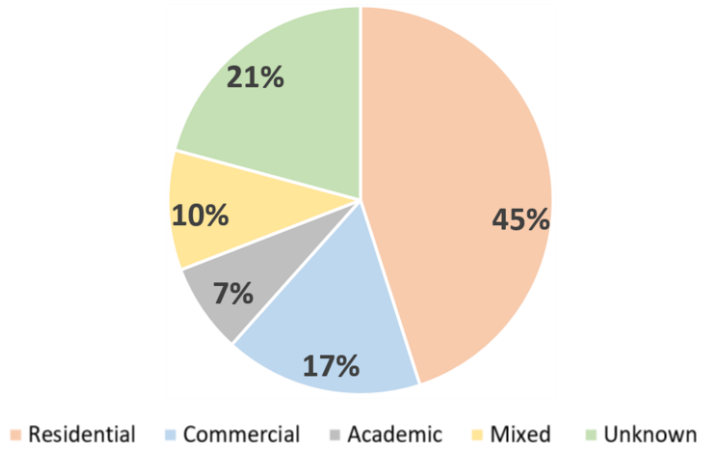

Figure 3.5. Proportion of Articles for each Objectives.

3.2.4. Real-Word Deployment

Lastly, the proportion of research articles where RL was actually deployed in the real world was studied. It was found that only 12% of research articles report results where RL was used with real HEMS. The results are shown in Fig. 3.6. The results are consistent with an earlier survey [40] where this proportion was 11%.

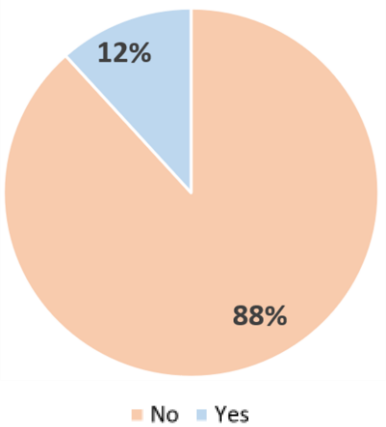

Figure 3.6. Proportion of Articles in Real World HEMS.

Chapter 4 - Deep Attentive Tabular Neural Networks and Imitation Learning and Control in Smart Home

4.1. Background: Deep Attentive Tabular Networks

TabNet consists of several functional DNN layers [63]. Before providing an overview of TabNet's architecture, an outline of each such layer is provided below.

A fully connected (FC) layer consists of a layer of neurons that incorporate a set of trainable weights **W**, as well as sigmoid nonlinearities $\sigma(\cdot)$. Suppose the input vector to an FC is denoted as **x**, its corresponding output **y** is given by the following relationship,

$$\mathbf{y} = \boldsymbol{\sigma}(\mathbf{W}\mathbf{y}). \tag{4.1}$$

The mapping $\boldsymbol{\sigma}(\cdot)$ represents an elementwise application of $\sigma(\cdot)$. Thus if x is a scalar input, the output $y = \sigma(x)$ is given by,

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{4.2}$$

TabNet contains layers of generalized linear units (GLUs). With **x** representing the input to a GLU layer, the output $\mathbf{y} = \mathbf{GLU}(\mathbf{x})$ is,

$$\mathbf{y} = \mathbf{x} \circ \boldsymbol{\sigma}(\mathbf{x}). \tag{4.3}$$

The operator '$\circ$' in the above relationship is the Kronecker product. The main advantage of using GLU layers in TabNet is to allow penetration into deeper layers without encountering exploding or imploding gradients.

Another type of layer consists of rectified linear units (ReLu). If **x** is the input vector to such a layer, its corresponding output **y** is denoted as, $\mathbf{y} = \mathbf{ReLu}(\mathbf{x})$, where $\mathbf{ReLu}(\cdot)$ is the elementwise application of the $ReLu(\cdot)$ function,

$$ReLu(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}. \tag{4.4}$$

Training samples in TabNet are divided into minibatches. A batch normalization (BN) layer contains neurons with linear activations, those only purpose is to perform elementwise normalization of their inputs. If $\mathcal{B}$ is a minibatch, and the vectors **x** and **y**, the input and output of any BN layer, then $\mathbf{y} = \mathbf{BN}(\mathbf{x}|\mathcal{B})$, where $\mathbf{BN}(\cdot)$ represents elementwise batch normalization. For each element x of **x**,

$$BN(x|\mathcal{B}) = \frac{x - \min\limits_{x \in \mathcal{B}} x}{\min\limits_{x \in \mathcal{B}} x - \min\limits_{x \in \mathcal{B}} x}. \tag{4.5}$$

The maximization and minimization operations are carried out over all sample elements in minibatch $\mathcal{B}$. In this manner each element of the vector **y** is restricted to the [0, 1] interval.

Throughout the remainder of this article, **x** is used to represent a vector input to TabNet such that $\mathbf{x} \in \mathbb{R}^D$ where D is the input dimensionality. Although in general its output is also a vector, the TabNet architecture used in this research outputs a scalar, y. Optionally, the nth sample input and output are indicated as $\mathbf{x}(n)$ and $y(n)$. TabNet is designed to mimic decision trees. It follows a multi-step architecture, where each sequential step is analogous to a decision box in such a tree. Fig. 4.1 illustrates the arrangement of a step indexed i in TabNet as well as its overall architecture. Each step includes two transformer blocks, (*i*) a feature transformer and (*ii*) an attentive transformer, whose layouts are depicted separately in Fig. 4.2.

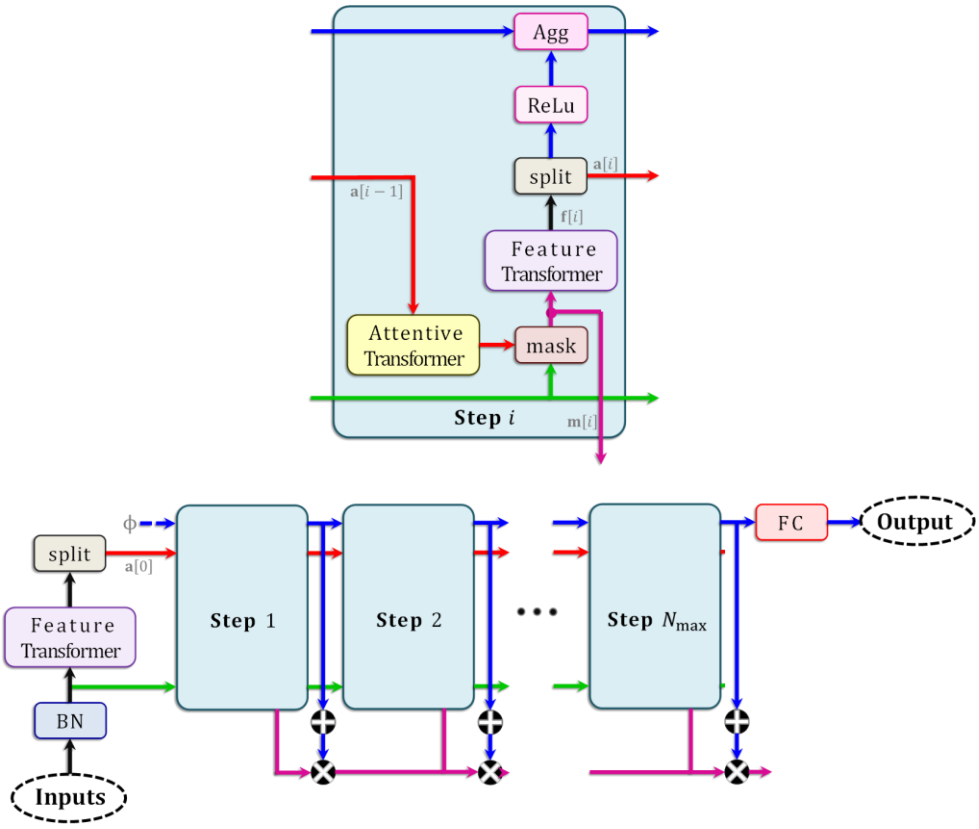

Figure 4.1. TabNet Architecture. Shown here are a single step (top) as well as the overall multi-step layout of TabNet (bottom).

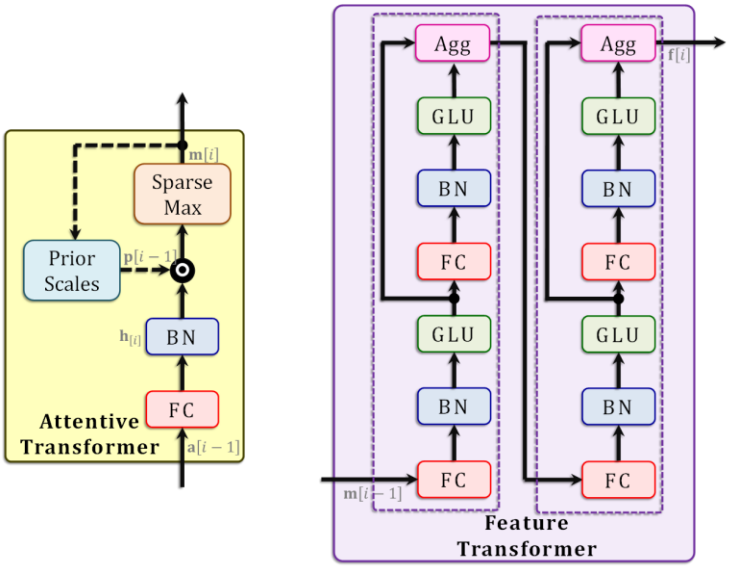

Figure 4.2. TabNet Transformers. Shown are the layouts of the attentive transformer (left) and the feature transformer (right) that are used in TabNet. Black circles marked '∘' are used to implement elementwise multiplication. Dashed lines in the feature transformer are from the prior step, $\boldsymbol{i-1}$.

The attentive transformer at each step i incorporates a SparseMax block that implements a sparse version of the well-known softmax function [134]. It produces a sparse vector $\mathbf{m}[i] \in \mathbb{R}^D$, labeled as the block mask. The mask is a sparse vector of probabilities that can be obtained in the following manner,

$$\mathbf{m}[i] = \underset{\mathbf{m}\in\boldsymbol{\Delta}^{D'}}{\operatorname{argmax}} \|\mathbf{m} - \mathbf{z}\|^2 . \tag{4.6}$$

In the above $\boldsymbol{\Delta}^{D'}$ is a subspace of the probability simplex in D dimensions, with only $D' < D$ nonzero elements. In other words, the mask vector in Eqn. (4.6) is such that $\|\mathbf{m}[i]\|_0 = D'$ and $\mathbf{1}_D^{\mathrm{T}}\mathbf{m}[i] = 1$. The vector $\mathbf{z}$ in Eqn. (4.6) shown above is determined as,

$$\mathbf{z} = \mathbf{p}[i-1] \cdot \mathbf{h}_i(\mathbf{a}[i-1]). \tag{4.7}$$

Ignoring the BN block, the function $\mathbf{h}_i(\cdot)$ is a trainable mapping accomplished by the immediately preceding FC layer, as in Eqn (4.1). The FC layer has its own set of trainable weights (not shown).

The block priorScales used the attentive transformer of step i Fig. 4.2 is a vector $\mathbf{p}[i]$. It is used to scale down $\mathbf{m}[i]$ using the masks $\mathbf{m}[j]$ from all prior steps $j < i$. With $\boldsymbol{\gamma} \geq \mathbf{1}$ being a predetermined TabNet parameter vector, the mask is obtained as shown below,

$$\mathbf{p}[i] = \prod_{j=1}^{i-1} (\boldsymbol{\gamma} - \mathbf{m}[j]) . \tag{4.8}$$

The product in the above equation is obtained in an elementwise fashion. The rationale behind this formulation can be explained in the following manner. If a scalar element has been used extensively in some prior step j, the corresponding mask $m[j]$ will be close to one. The corresponding factor $\gamma - m[j]$ in Eqn. (4.8) will be small, so that the same element is scaled

downward. On the other hand, if the element stays unused, the $p[i]$ will be nearly one, ergo likelier to be used in step i.

The feature transform converts input vectors into more suitable representations. It consists of four separate sub-blocks, with each consisting of an FC layer, followed by a BN layer, and then a GLU layer. In a manner shown in Eqn. (4.5), the transformer applies normalization. However, it uses ghost normalization where only a smaller set $\mathcal{B}' \subset \mathcal{B}$ of the minibatch is used for this purpose so that $\mathbf{y} = \mathbf{BN}(\mathbf{x}|\mathcal{B}')$. Although ghost normalization slows down the training, it has been shown to yield better generalization properties [135].

The feature transformer splits the $D \times 1$ vector into two disjoint chunks with dimensionalities D_d and D_a such that $D = D_d + D_a$. This is achieved using the Split block. Aggregation takes place at each block Agg. The final decision is determined as,

$$\mathbf{d} = \sum_{i=1}^{N_{\max}} \mathbf{ReLu}(\mathbf{d}[i])\,. \tag{4.9}$$

The mapping $\mathbf{ReLu}(\cdot)$ appearing above is performed elementwise as in Eqn. (4.4). The quantity $N_{\max}$ is the total number of steps in TabNet. It is obtained empirically based on the dataset.

With the symbol $\boldsymbol{\theta}$ representing the set of all weight parameters of TabNet, and $f(\cdot)$ being another nonlinearity representing the composition of the nonlinear function of each step, its output can be expressed succinctly as,

$$y = f(\mathbf{x}|\boldsymbol{\theta}). \tag{4.10}$$

The training dataset is of the form $\{\mathbf{x}(n), t(n)|n \in \mathcal{N}_T\}$, where $\mathcal{N}_T$ is the set of indices of all training samples, $\mathbf{x}(n)$ is the nth sample input, and $t(n)$, the corresponding target output. The aim of the learning algorithm is to iteratively increment $\boldsymbol{\theta}$ so that for every training sample, the

difference between the target and TabNet's true output $\mathbf{y}(n) = f(\mathbf{x}(n)|\boldsymbol{\theta})$ is steadily reduced. The deviation between the outputs and the targets is quantified in terms of the network's loss. A regularization term is added to the loss, and the regularized loss is denoted as $\mathcal{L}(\boldsymbol{\theta}|\mathcal{N}_T)$, whereupon the TabNet trained parameter is,

$$\boldsymbol{\theta} \cong \underset{\boldsymbol{\theta}'}{\operatorname{arginf}} \mathcal{L}(\boldsymbol{\theta}'|\mathcal{N}_T)\,. \tag{4.11}$$

4.2. Problem Description

4.2.1. Data Set

The dataset used in this research is available online in public domain. It is in the form of time series samples. The time series contains outside (external) and indoor (internal) temperatures τ_t^{int} and τ_t^{ext}, the corresponding humidity levels m_t^{int} and m_t^{ext}, and separate thermostat settings for heating and cooling s_t^{heat} and s_t^{cool}. The subscripts t appearing in all these quantities refer to discrete time instances, with $t \in \{1, \dots, T\}$ where T is the length of a sample train. Two separate settings are used here only because that was the case with the dataset. We presume that this is a residence specific artifact. Depending on the scenario, a single thermostat setting variable may also be used. In addition, the initial dataset included various other fields that were dropped.

4.2.2. Switching Policy Models

As the dataset contains separate controls heating and cooling, two policy models have been used in this research to represent agents for imitation learning. The models share a common input $\mathbf{x}_t$,

$$\mathbf{x}_t = \left(m_t^{\text{int}}, \tau_t^{\text{int}}, m_t^{\text{ext}}, \tau_t^{\text{ext}}\right). \tag{4.12}$$

The models are represented in functional forms as $\boldsymbol{\theta}_{\pi}^{\mathrm{h}}$ (heating) and $\boldsymbol{\theta}_{\pi}^{\mathrm{c}}$ (cooling); the common subscript π is used to distinguish them from the prediction models. For the heating model it is as given below,

$$\hat{s}_t^{\mathrm{h}} = \boldsymbol{\theta}_{\pi}^{\mathrm{h}}(\mathbf{x}_t). \tag{4.13}$$

Similarly, for the cooling model the corresponding function is,

$$\hat{s}_t^{\mathrm{c}} = \boldsymbol{\theta}_{\pi}^{\mathrm{c}}(\mathbf{x}_t). \tag{4.14}$$

It must be noted that the quantities $\hat{s}_t^{\mathrm{h}}$ and $\hat{s}_t^{\mathrm{c}}$ (with the 'hat' $\hat{\cdot}$) are merely the settings that are recommended by the agent model.

Since thermostat settings do not change in each time step, adjusting them at any instant t to their recommended values is implemented in a probabilistic manner. Therefore, with the probability of switching to the recommended setting being p, the instantaneous settings are obtained as,

$$s_t^{\mathrm{h}} = \begin{cases} \hat{s}_t^{\mathrm{h}}, & \text{with probability } p \\ s_{t-1}^{\mathrm{h}}, & \text{otherwise} \end{cases}. \tag{4.15}$$

Similarly,

$$s_t^{\mathrm{c}} = \begin{cases} \hat{s}_t^{\mathrm{c}}, & \text{with probability } p \\ \hat{s}_{t-1}^{\mathrm{c}}, & \text{otherwise} \end{cases}. \tag{4.16}$$

The symbols $\hat{s}_t^{\mathrm{h}}$ and $\hat{s}_t^{\mathrm{c}}$ in the above can also used to distinguish them from the manually controlled settings that is available in the data set.

4.2.3. Environmental Prediction Models

The purpose of the environmental prediction models is to predict the future indoor temperatures and humidity levels. This is because when the policy models $\boldsymbol{\theta}_{\pi}^{\mathrm{c}}$ and $\boldsymbol{\theta}_{\pi}^{\mathrm{h}}$ are used to replace human decisions, the instantaneous indoor temperature and humidity levels from the dataset can no longer be used. Since only single step lookahead predictions are required, two

prediction models have been used with a common subscript p for prediction, which are $\boldsymbol{\theta}_{\mathrm{p}}^{\tau}$ (temperature) and $\boldsymbol{\theta}_{\mathrm{p}}^{\mathrm{m}}$ (humidity). In addition to $\mathbf{x}_t$ in Eqn. (4.12), the inputs to these models are the vector of switching actions obtained using Eqn. (4.15) and Eqn. (4.16) as shown below,

$$\mathbf{s}_t = \left(s_t^{\mathrm{h}}, s_t^{\mathrm{c}}\right). \tag{4.17}$$

The prediction models can be expressed in functional form as,

$$\tilde{\tau}_{t+1}^{\mathrm{int}} = \boldsymbol{\theta}_{\mathrm{p}}^{\tau}(\mathbf{x}_t, \mathbf{s}_t)., \tag{4.18}$$

and,

$$\tilde{m}_{t+1}^{\mathrm{int}} = \boldsymbol{\theta}_{\mathrm{p}}^{\mathrm{m}}(\mathbf{x}_t, \mathbf{s}_t). \tag{4.19}$$

4.2.4. Comfort Indices Models

Human comfort level is determined by a combination of several factors, including the indoor temperature and humidity. As previously noted, comfort is a human sensation that varies from one individual to another, incorporates nonlinearities, and is difficult to quantify. Consequently, comfort is not used anywhere in this article to train the TabNet models. It is only used as a secondary evaluation tool. The two indices used here for secondary evaluation are: (*i*) the *predicted mean vote* (PMV), and (*ii*) the *predicted percentage of dissatisfied* (PPD). The expression used to determine these indices, and the values of their coefficients have been obtained from [136].

PMV estimates the mean response of a large group of people in accordance with the ASHRAE thermal sensation scale. The index is an integer in the interval $[-3, +3]$ with zero representing highest satisfaction. At any given instant t, PMV has been obtained as,

$$v_t = \alpha\tau_t^{\mathrm{int}} + \beta m_t^{\mathrm{int}} + \gamma. \tag{4.20}$$

It should be noted that the three coefficients α, β, and γ have different values for male and female occupants.

The PPD index estimates the percentage of a group of people who report dissatisfaction, with zero reflecting maximum comfort. PPD is obtained indirectly, using PMV v_t according to,

$$d_t = d_0 - d_1 e^{-(a v_t^4 + b v_t^2)}. \tag{4.21}$$

The quantities d_0, d_1, a, and b are four coefficients used in PPD.

Although PMV and PPD reflect dissatisfaction, they are referred to as comfort indices in this article. Other determinants of human comfort such as clothing and activity level could not be incorporated in Eqn. (4.12) as they are not available in the dataset. The values of all coefficients are provided in Table (4.1).

Table 4.1. PMV and PPD Coefficients

Index		**PMV** (v_t)			**PPD** (d_t)			
Symbol		α	β	γ	d_0	d_1	a	b
Value	Male	0.220	0.233	6.673	100	95	0.03353	0.21970
	Female	0.272	0.248	7.245				
	Mixed	0.245	0.248	6.475				

4.3. Results

4.3.1. Preprocessing

The samples in the dataset were collected at regularly spaced intervals of 15 minutes (so that the difference between two consecutive instances t and $t+1$ is 15 mins). Only the columns of the spreadsheet that are associated with the four inputs (τ_t^{int}, m_t^{int}, τ_t^{ext}, m_t^{ext}) as well as the two thermal setting ($s_t^{\text{h}}, s_t^{\text{c}}$) were used in this research. All other columns in the file were ignored. Missing values were filled with the mean values of the relevant entries.

Samples with fixed thermal settings were ignored in the training process of the two switching policy models, as periods with no switching are considered not useful to develop the policy models. A total of 4269 samples were obtained in this manner to train and test the heating policy model and 1258 samples to train and test the cooling policy model. Table (4.2) provides all details.

Table 4.2. Switching Activity

Setting	Total	Changed	Fixed
Heating	47247	09.04%	90.96 %
Cooling		02.66%	97.34%

To simulate the indoor environment, we developed two separate machine learning algorithms, a temperature environment model, and a humidity environment model. We dropped empty cells, then we utilized all the remaining 44978 samples for the training and testing. We split the samples in all our four models into 80% training and 20% testing sets.

After we finished the training process, we divided the data into intervals of 12 consecutive samples (three hours). Then we added the intervals with fixed thermal settings in all it is samples to the inactive period dataset. The rest of the intervals were added to the active period dataset. We will use these terms (active and inactive periods) later in this chapter.

4.3.2. Evaluation Metrics

The mean square error E is given as,

$$E = \frac{1}{N}\sum_{t=1}^{N}(r_t - s_t)^2 \tag{4.22}$$

The accuracy was determined in terms of the absolute difference,

$$A = \frac{1}{N}\sum_{t=1}^{N} \delta_t , \tag{4.23}$$

where δ_t is,

$$\delta_t = \begin{cases} 1, & |r_t - s_t|' - \eta\, s_t < 0 \\ 0, & \text{otherwise} \end{cases}. \tag{4.24}$$

In Eqn. (22) and Eqn. (24), N is the total number of samples, η is the error margin (η is used when we have a regression problem and we want to calculate the accuracy), r_t and s_t are the estimated and real outputs, respectively. $\eta = 0.005$ for the environment models and $\eta = 0.05$ for the policy models. MSE is considered to be the main evaluation metric, while the accuracy is merely another metric that worth showing in this problem.

4.3.3. Model Comparison

It is worth mentioning that TabNet training time is longer than decision tree models. However, this is not considered an issue as we are training the models offline.

Table 4.3. Policy Models' Evaluations of Different Regressors

Regressor Algorithm Name	Switching Policy Model			
	Heating Setting model		Cooling Setting model	
	Accuracy %	MSE	Accuracy %	MSE
TabNet	**86.18**	**8.6**	**98.41**	**2.5**
DNN	81.91	10.5	95.37	4.4
K-nearest Neighbors	76.81	12.7	93.65	3.3
Decision Tree	76.8	20.4	87.3	6.3
Random Forest	84.54	10.3	96.03	2.8
AdaBoost	70.25	11.7	96.83	3.2
Gradient Boosting	85.71	9.2	96.83	2.7
SVR	76.58	18.1	95.24	4.9

Table 4.4. Environment Models' Evaluations of Different Regressors

Regressor Algorithm Name	Environmental Prediction Model			
	Temp Estimation Model		Humidity Estimation Model	
	Accuracy %	MSE	Accuracy %	MSE
TabNet	**75.19**	**0.17**	**64.16**	**0.59**
DNN	67.88	0.37	33.82	1.44
K-nearest Neighbors	33.06	1.27	29.24	1.97
Decision Tree	57.76	0.54	38.62	1.59
Random Forest	70.83	0.34	56.58	0.73
AdaBoost	32.24	0.61	47.02	0.89
Gradient Boosting	72.08	0.29	63.07	0.60
SVR	44.95	0.46	48.29	1.18

4.3.4. TabNet Training

The TabNet models were trained in Python, using PyTorch as a machine learning framework. MSE was used as a loss function to optimize. Adam was used as the optimizer function with a momentum value of 0.3. Training and validation loss curves are shown in Fig. (4.3). Early stopping and learning rate decay were used as well. Fig. (4.4) shows the learning rate values of all the models through the entire training process.

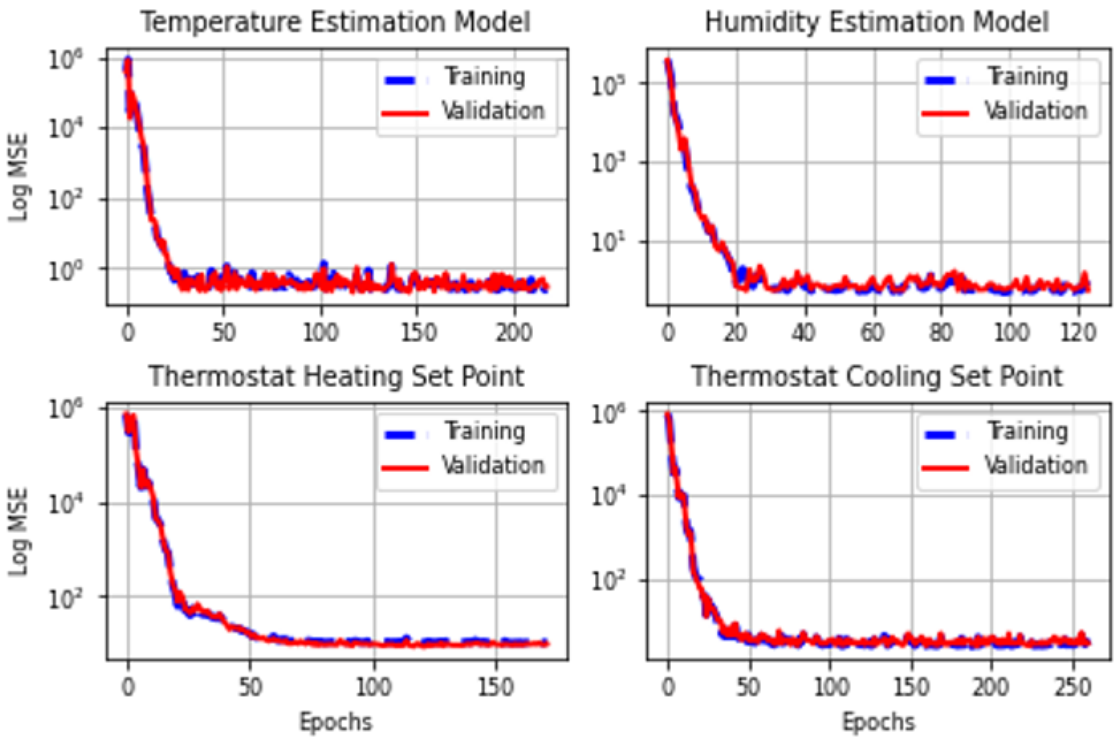

Figure 4.3. Line Plots of Log MSE Error Loss Over Training Epochs.

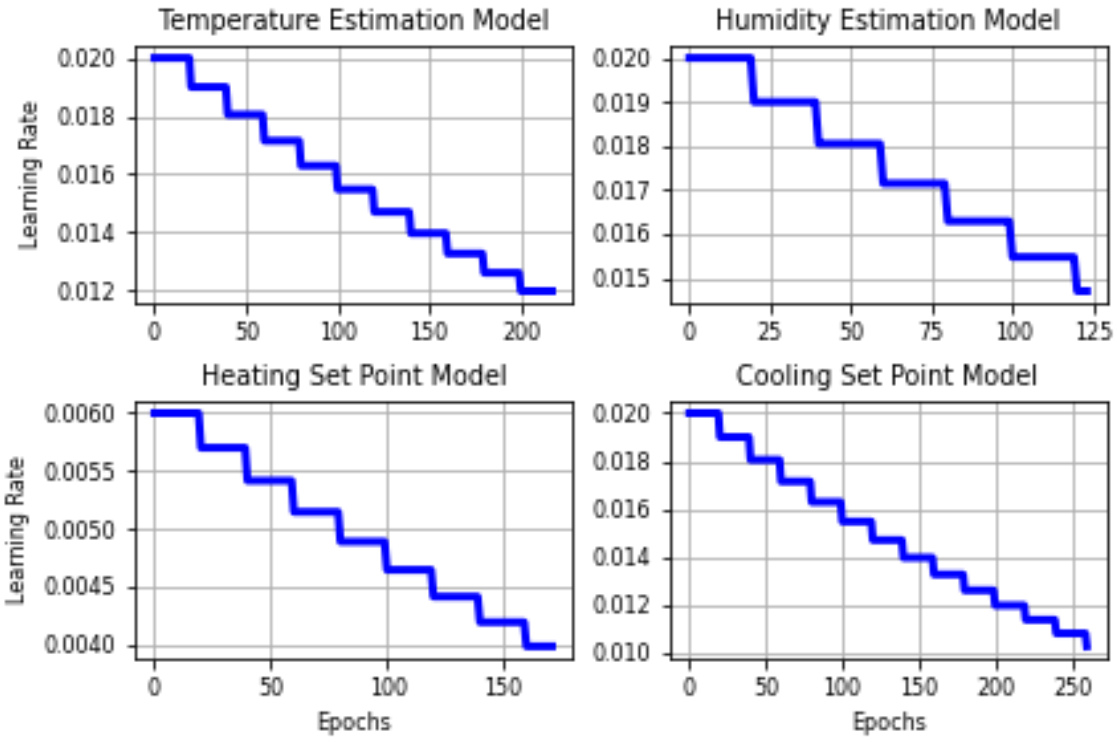

Figure 4.4. Learning Rate Decay with step size = 20.

4.3.5. Objective Function

Occupant's settings during the active periods are assumed to be the optimal strategy that we need to imitate and apply them to the occupant's inactive period (i.e., sleeping, lazy). In this imitation learning model, I mimicked the occupant's best strategy when he switched actively, then I applied this learned strategy to the inactive period. Stochastic switching process was applied with a discrete probability value ranging from 0-100%. Due to the stochastic switching, we repeated the process 100 times with each probability value and then calculated the average from all the runs to get a more accurate results of the proposed architecture. Fig 4.5 illustrates the process of applying the policy models to the inactive period. We used two objective functions to test our assumption.

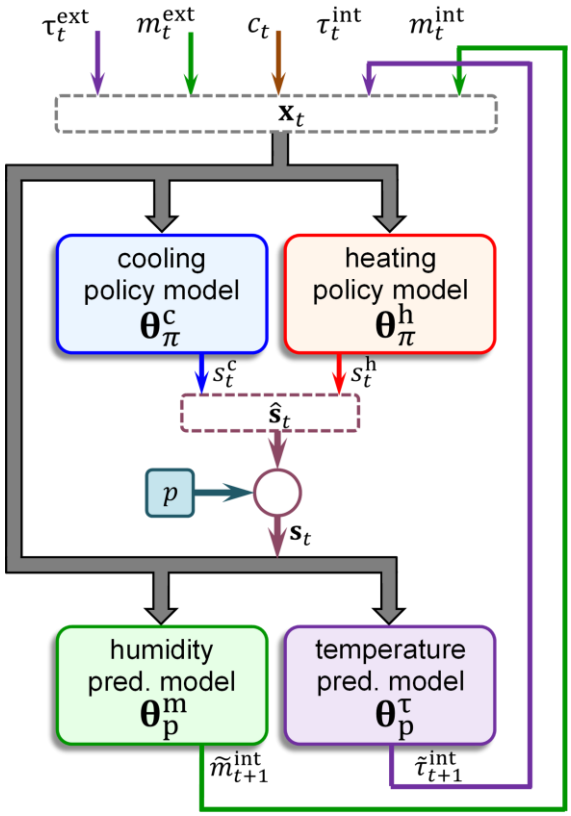

Figure 4.5. TabNet Models. Shown are the four TabNet models used in this research, as well as all inputs and outputs. The input $\boldsymbol{c_t}$ representing unit price is optional and may be excluded. The circle represents probabilistic switching action.

Our primary objective is to minimize the comfort gap between the inactive and active periods of the occupants. We calculated PPD for the occupant during the period where the occupant was active, inactive and when we applied our policy models to the inactive period.

The PPD gap between the active and inactive period was minimized after applying the policy models to the inactive period. PPD of active Period = 12.80, PPD of inactive period = 17.97, PPD of inactive period after applying the policy model = 15.05.

Discomfort and PPD were used as a secondary objective. Our results proved that we were able to optimize the PMV and PPD by switching more frequently as shown in Fig 4.6. We achieved about 9% better performance when we switched every 15 minutes with 100% switching probability (deterministic).

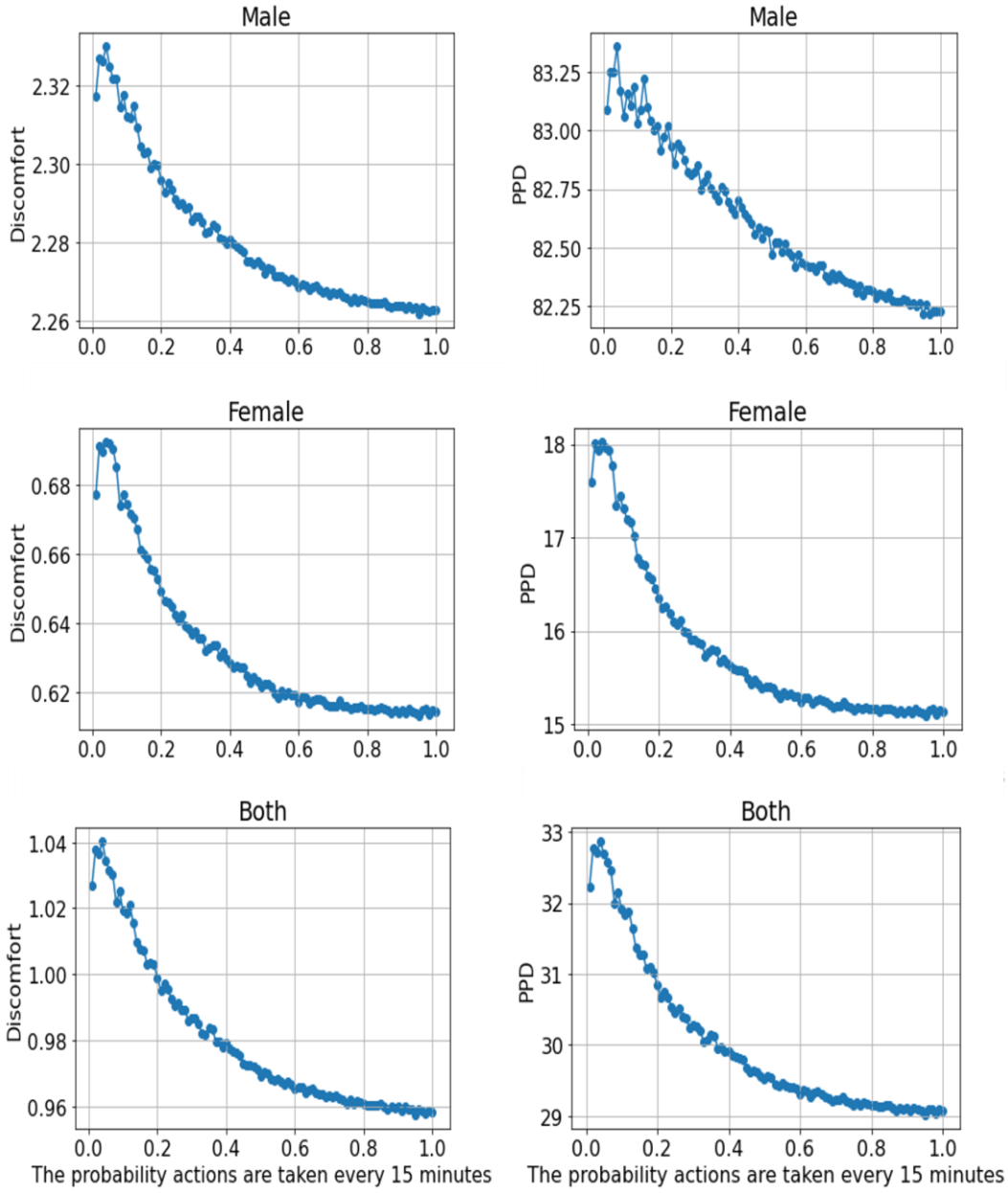

Figure 4.6. Discomfort & PPD values for Male, Female and Both.

Chapter 5 - Conclusion

Reinforcement learning is arguably the most promising area within the rapidly emerging discipline of machine learning. Unlike the more commonly used supervised and unsupervised methods that are limited to handling straightforward data, in using reward signals as feedback, reinforcement learning equips the learning algorithm to directly learn from experience, much like actual human beings. This is the first of the two crucial features of reinforcement learning that forms the backdrop of this thesis.

Traditional machine learning algorithms are severely restricted in terms of their trainability. For example, in a regression tree, the decision nodes must match input data fields. Decisions based on more abstracted data cannot be incorporated. On the other hand, deep neural networks, which are organized into layers of neurons, are modeled roughly after mammalian cortical structures, learn by adjusting their internal weight parameters. This allows deep networks to manipulate data to learn at higher levels of abstraction, the other crucial feature that deep reinforcement learning algorithms possess. This is the other motivation for the research contained in this thesis.

The twin capabilities of learning from experience and learning at higher levels of abstraction, set reinforcement learning apart from other areas of machine learning and (within the broader context) all of artificial intelligence. It is of little surprise that reinforcement learning is at the very heart of artificial general intelligence, the holy grail of artificial intelligence. Algorithmic entities can replace human beings in the real world, including at their very homes, thereby allowing humans a degree of comfort that had hitherto been considered to be an impossible dream. It is this lofty goal that forms the backdrop of this thesis, which entails manifold research contributions, described next.

This thesis investigates the use of deep reinforcement learning in HEMS applications. At the onset, it provides an overview of generic reinforcement learning. This is followed with more detailed discussions on the state-of-the-art methods for value based, policy gradient based, and actor-critic reinforcement learning. Verbal descriptions of these algorithms are accompanied with explanatory figures as well as mathematical expressions that use notations and terminology similar to those which are prevalent in published research in the machine learning community. This is done to make its published literature more accessible to HEMS research.

The thesis describes survey oriented research that was conducted by me, to investigate how effectively has reinforcement learning been leveraged for HEMS applications. In doing so, I have discovered that: (i) a significant volume of this research uses tabular learning and without considering deep neural networks, and (ii) the approaches proposed by this research are rarely deployed in the real world, their use being limited to simulation platforms only.

As an example study, the thesis takes up a specific HEMS application – indoor environmental prediction and control in the smart home. It attempts to benefit directly from human experience by making use of imitation learning. The research uses the state-of-the-art TabNet architecture, an extension of deep neural networks. TabNet is able to outperform all other machine learning algorithms in current use. The promising results demonstrate how current developments in machine learning can be adopted effectively in HEMS research.

Ultimately, I hope that my thesis will help in bridging the gap between the machine learning and the HEMS research communities.

References

[1] U.S. Energy Information Administration, “Electricity explained: use of electricity,” May 14, 2021. Available: https://www.eia.gov/energyexplained/electricity/use-of-electricity.php [Accessed: April 10, 2022].

[2] Center for Sustainable Systems, “U.S. Energy System Factsheet,” Pub. No. CSS03-11, Center for Sustainable Systems, University of Michigan, September 2021. Available: https://css.umich.edu/factsheets/us-energy-system-factsheet [Accessed: April 10, 2022].

[3] Mohammad Shakeri, Mohsen Shayestegan, Hamza Abunima, S.M. Salim Reza, M. Akhtaruzzaman, A.R.M. Alamoud, Kamaruzzaman Sopian, Nowshad Amin, "An intelligent system architecture in home energy management systems (HEMS) for efficient demand response in smart grid," *Energy and Buildings*, Volume 138, 2017, pp. 154-164, ISSN 0378-7788, doi.org/10.1016/j.enbuild.2016.12.026.

[4] J. Leitão, P. Gil, B. Ribeiro, and A. Cardoso, “A survey on home energy management,” *IEEE Access*, vol. 8, pp. 5699–5722, 2020.

[5] H. Shareef, M. S. Ahmed, A. Mohamed and E. Al Hassan, "Review on Home Energy Management System Considering Demand Responses, Smart Technologies, and Intelligent Controllers," in *IEEE Access*, vol. 6, pp. 24498-24509, 2018, doi: 10.1109/ACCESS.2018.2831917.

[6] Bandana Mahapatra, and Anand Nayyar, “Home energy management system (HEMS): Concept, architecture, infrastructure, challenges and energy management schemes,” *Energy Systems*, pp. 1-27, 2019. doi.org/10.1007/s12667-019-00364-w.

[7] G. Dileep, “A survey on smart grid technologies and applications,” *Renewable energy*, vol. 146, pp. 2589-2625, 2020.

[8] U. Zafar, S. Bayhan and A. Sanfilippo, “Home Energy Management System Concepts, Configurations, and Technologies for the Smart Grid," in *IEEE Access*, vol. 8, pp. 119271-119286, 2020, doi: 10.1109/ACCESS.2020.3005244.

[9] Kari Alanne, Seppo Sierla, "An overview of machine learning applications for smart buildings," *Sustainable Cities and Society*, Volume 76, 2022, 103445, ISSN 2210-6707, https://doi.org/10.1016/j.scs.2021.103445.

[10] J. Aguilar, A. Garces-Jimenez, M.D. R-Moreno, Rodrigo García, "A systematic literature review on the use of artificial intelligence in energy self-management in smart buildings," *Renewable and Sustainable Energy Reviews*, Volume 151, 2021, 111530, ISSN 1364-0321, https://doi.org/10.1016/j.rser.2021.111530.

[11] Yassine Himeur, Khalida Ghanem, Abdullah Alsalemi, Faycal Bensaali, Abbes Amira, "Artificial intelligence based anomaly detection of energy consumption in buildings: A

review, current trends and new perspectives," *Applied Energy*, Volume 287, 2021, 116601, doi.org/10.1016/j.apenergy.2021.116601.

[12] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol.13, 1983, pp. 835-846.

[13] G. Tesauro, "TD-Gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, 1994, 215–219.

[14] Jan Peters, and Stefan Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 682-697, 2008.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529-533, 2015.

[16] D. Silver *et al*., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[17] D. Silver *et al*., "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[18] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, Anil Anthony Bharath, "A Brief Survey of Deep Reinforcement Learning," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[19] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare and Joelle Pineau, "An Introduction to Deep Reinforcement Learning", *Foundations and Trends in Machine Learning*, vol. 11, no. 3-4, 2018. DOI: 10.1561/2200000071.

[20] David Silver, Satinder Singh, Doina Precup, Richard S. Sutton, "Reward Is Enough," *Artificial Intelligence*, vol. 299, 2021: 103535.

[21] Ben Goertzel, *Artificial general intelligence* (Eds: Cassio Pennachin), vol. 2, Springer, New York, 2007.

[22] Tengteng Zhang, and Hongwei Mo, "Reinforcement learning for robot research: A comprehensive review and open issues," *International Journal of Advanced Robotic Systems*, vol. 18, no. 3, 2021.

[23] Sarthak Bhagat, Hritwick Banerjee, Zion Tsz Ho Tse, and Hongliang Ren, "Deep reinforcement learning for soft, flexible robots: Brief review with impending challenges," *Robotics*, vol, 8, no. 1, 2019.

[24] H. Shakhatreh et al., "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges," in *IEEE Access*, vol. 7, pp. 48572-48634, 2019, doi: 10.1109/ACCESS.2019.2909530.

[25] F. Zeng, C. Wang and S. S. Ge, "A Survey on Visual Navigation for Artificial Agents With Deep Reinforcement Learning," in *IEEE Access*, vol. 8, pp. 135426-135442, 2020, doi: 10.1109/ACCESS.2020.3011438.

[26] H. Sun, W. Zhang, R. Yu and Y. Zhang, "Motion Planning for Mobile Robots—Focusing on Deep Reinforcement Learning: A Systematic Review," in *IEEE Access*, vol. 9, pp. 69061-69081, 2021, doi: 10.1109/ACCESS.2021.3076530.

[27] N. C. Luong et al., "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," in *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133-3174, 4th quarter 2019, doi: 10.1109/COMST.2019.2916583.

[28] Z. Ullah, F. Al-Turjman and L. Mostarda, "Cognition in UAV-Aided 5G and Beyond Communications: A Survey," in *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 3, pp. 872-891, Sept. 2020, doi: 10.1109/TCCN.2020.2968311.

[29] T. T. Nguyen, and V. J. Reddi, "Deep reinforcement learning for cyber security," *arXiv preprint*, arXiv:1906.05799, 2019.

[30] H. Zhu, Y. Cao, W. Wang, T. Jiang and S. Jin, "Deep Reinforcement Learning for Mobile Edge Caching: Review, New Features, and Open Issues," in *IEEE Network*, vol. 32, no. 6, pp. 50-57, November/December 2018, doi: 10.1109/MNET.2018.1800109.

[31] Siqi Liu, Kay Choong See, Kee Yuan Ngiam, Leo Anthony Celi, Xingzhi Sun, and Mengling Feng, "Reinforcement learning for clinical decision support in critical care: comprehensive review," *Journal of Medical Internet Research*, vol. 22, no. 7, 2020.

[32] D. Elavarasan, and P. M. D. Vincent, "Crop Yield Prediction Using Deep Reinforcement Learning Model for Sustainable Agrarian Applications," in *IEEE Access*, vol. 8, pp. 2020, 86886-86901, doi: 10.1109/ACCESS.2020.2992480.

[33] Paul Garnier, Jonathan Viquerat, Jean Rabault, Aurélien Larcher, Alexander Kuhnle, and Elie Hachem, "A review on deep reinforcement learning for fluid mechanics," *Computers & Fluids* vol. 225, 2021.

[34] D. Zhang, X. Han and C. Deng, "Review on the research and practice of deep learning and reinforcement learning in smart grids," in *CSEE Journal of Power and Energy Systems*, vol. 4, no. 3, pp. 362-370, September 2018, doi: 10.17775/CSEEJPES.2018.00520.

[35] Z. Zhang, D. Zhang and R. C. Qiu, "Deep reinforcement learning for power system applications: An overview," in *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1, pp. 213-225, March 2020, doi: 10.17775/CSEEJPES.2019.00920.

[36] O. Jogunola et al., "Consensus Algorithms and Deep Reinforcement Learning in Energy Market: A Review," in *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4211-4227, 15 March15, 2021, doi: 10.1109/JIOT.2020.3032162.

[37] A.T.D. Perera, Parameswaran Kamalaruban, "Applications of reinforcement learning in energy systems," *Renewable and Sustainable Energy Reviews*, Volume 137, 2021, 110618, ISSN 1364-0321, doi.org/10.1016/j.rser.2020.110618.

[38] Xin Chen, Guannan Qu, Yujie Tang, Steven Low, and Na Li, "Reinforcement learning for selective key applications in power systems: Recent advances and future challenges," *IEEE Transactions on Smart Grid*, 2022, doi: 10.1109/TSG.2022.3154718.

[39] Karl Mason, and Santiago Grijalva, "A review of reinforcement learning for autonomous building energy management," *Computers & Electrical Engineering*, vol. 78, pp. 300-312, 2019.

[40] Zhe Wang, Tianzhen Hong, "Reinforcement learning for building controls: The opportunities and challenges," *Applied Energy*, Vol. 269, 2020, pp. 115036, , doi.org/10.1016/j.apenergy.2020.115036. 1997-2019

[41] Mengjie Han, Ross May, Xingxing Zhang, Xinru Wang, Song Pan, Da Yan, Yuan Jin, and Liguo Xu, "A review of reinforcement learning methodologies for controlling occupant comfort in buildings," *Sustainable Cities and Society*, vol. 51, pp. 101748–101762, Nov. 2019. doi.org/10.1016/j.scs.2019.101748

[42] L. Yu, S. Qin, M. Zhang, C. Shen, T. Jiang and X. Guan, "A Review of Deep Reinforcement Learning for Smart Building Energy Management," in *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12046-12063, 1 Aug.1, 2021, doi: 10.1109/JIOT.2021.3078462.

[43] H. Zhang, S. Seal, D. Wu, F. Bouffard and B. Boulet, "Building Energy Management With Reinforcement Learning and Model Predictive Control: A Survey," *IEEE Access*, vol. 10, pp. 27853-27862, 2022, doi: 10.1109/ACCESS.2022.3156581.

[44] José R.Vázquez-Canteli, and Zoltán Nagy, "Reinforcement learning for demand response: A review of algorithms and modeling techniques," *Applied Energy*, vol. 235, pp. 1072–1089, Feb. 2019. https://doi.org/10.1016/j.apenergy.2018.11.002.

[45] L. Yu, S. Qin, M. Zhang, C. Shen, T. Jiang, and X. Guan, "Deep reinforcement learning for smart building energy management: A survey," *arXiv preprint*, arXiv:2008.05074, 2020.

[46] L. Yu, W. Xie, D. Xie, Y. Zou, D. Zhang, S. Zhixin, L. Zhang, Y. Zhang, and T. Jiang, "Deep Reinforcement Learning for Smart Home Energy Management," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2751-2762, April 2020, doi: 10.1109/JIOT.2019.2957289.

[47] Paulo Lissa, Conor Deane, Michael Schukat, Federico Seri, Marcus Keane, and Enda Barrett, "Deep reinforcement learning for home energy management system control," *Energy and AI*, vol. 3, 100043, 2021.

[48] Ting Yang, Liyuan Zhao, Wei Li, Jianzhong Wu, Albert Y. Zomaya,"Towards healthy and cost-effective indoor environment management in smart homes: A deep reinforcement

learning approach," *Applied Energy*, Volume 300, 2021, 117335, ISSN 0306-2619, https://doi.org/10.1016/j.apenergy.2021.117335.

[49] Siliang Lu, Weilong Wang, Chaochao Lin, Erica Cochran Hameen, "Data-driven simulation of a thermal comfort-based temperature set-point control with ASHRAE RP884," *Building and Environment*, Volume 156, 2019, Pages 137-146, ISSN 0360-1323, https://doi.org/10.1016/j.buildenv.2019.03.010.

[50] P. Macieira, L. Gomes, and Z. Vale, "Energy Management Model for HVAC Control Supported by Reinforcement Learning," *Energies*, vol. 14, no. 24, p. 8210, Dec. 2021, doi: 10.3390/en14248210.

[51] B. Liu, M. Akcakaya and T. E. Mcdermott, "Automated Control of Transactive HVACs in Energy Distribution Systems," *IEEE Transactions on Smart Grid*, vol. 12, no. 3, pp. 2462-2471, May 2021, doi: 10.1109/TSG.2020.3042498.

[52] X. Zhang, D. Biagioni, M. Cai, P. Graf and S. Rahman, "An Edge-Cloud Integrated Solution for Buildings Demand Response Using Reinforcement Learning," *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 420-431, Jan. 2021, doi: 10.1109/TSG.2020.3014055.

[53] William Valladares, Marco Galindo, Jorge Gutiérrez, Wu-Chieh Wu, Kuo-Kai Liao, Jen-Chung Liao, Kuang-Chin Lu, Chi-Chuan Wang, "Energy optimization associated with thermal comfort and indoor air control via a deep reinforcement learning algorithm," *Building and Environment*, Volume 155, 2019, Pages 105-117, ISSN 0360-1323, https://doi.org/10.1016/j.buildenv.2019.03.038.

[54] Z. Zhang, C. Ma, and R. Zhu, "Thermal and Energy Management Based on Bimodal Airflow-Temperature Sensing and Reinforcement Learning," *Energies*, vol. 11, no. 10, p. 2575, Sep. 2018, doi: 10.3390/en11102575.

[55] C. Blad, S. Bøgh, and C. Kallesøe, "A Multi-Agent Reinforcement Learning Approach to Price and Comfort Optimization in HVAC-Systems," *Energies*, vol. 14, no. 22, p. 7491, Nov. 2021, doi: 10.3390/en14227491.

[56] F. Ruelens, S. Iacovella, B. Claessens, and R. Belmans, "Learning Agent for a Heat-Pump Thermostat with a Set-Back Strategy Using Model-Free Reinforcement Learning," *Energies*, vol. 8, no. 8, pp. 8300–8318, Aug. 2015, doi: 10.3390/en8088300.

[57] Mengjie Han, Ross May, Xingxing Zhang, Xinru Wang, Song Pan, Yan Da, Yuan Jin, "A novel reinforcement learning method for improving occupant comfort via window opening and closing," *Sustainable Cities and Society*, Volume 61, 2020, 102247, ISSN 2210-6707, https://doi.org/10.1016/j.scs.2020.102247.

[58] P. Korkidis, A. Dounis, and P. Kofinas, "Computational Intelligence Technologies for Occupancy Estimation and Comfort Control in Buildings," *Energies*, vol. 14, no. 16, p. 4971, Aug. 2021, doi: 10.3390/en14164971.

[59] F. Ruelens, B. J. Claessens, P. Vrancx, F. Spiessens and G. Deconinck, "Direct load control of thermostatically controlled loads based on sparse observations using deep reinforcement learning," *CSEE Journal of Power and Energy Systems*, vol. 5, no. 4, pp. 423-432, Dec. 2019, doi: 10.17775/CSEEJPES.2019.00590.

[60] Alex Dmitrewski, Miguel Molina-Solana, Rossella Arcucci, "CntrlDA: A building energy management control system with real-time adjustments. Application to indoor temperature," *Building and Environment*, Volume 215, 2022, 108938, ISSN 0360-1323, https://doi.org/10.1016/j.buildenv.2022.108938.

[61] N. Kodama, T. Harada and K. Miyazaki, "Home Energy Management Algorithm Based on Deep Reinforcement Learning Using Multistep Prediction," *IEEE Access*, vol. 9, pp. 153108-153115, 2021, doi: 10.1109/ACCESS.2021.3126365.

[62] S. O. Arik, and T. Pfister, "TabNet: Attentive interpretable tabular learning," *arXiv*:1908.07442v5, December 2020. [Online]. [Available: https://doi.org/10.48550/arXiv.1908.07442v4].

[63] Sercan Ö. Arik, and Tomas Pfister, "Tabnet: Attentive interpretable tabular learning," In *Proceedings, AAAI Conference on Artificial Intelligence*, vol. 35, no. 8, pp. 6679-6687, 2021.

[64] Ankit Pal, and Malaikannan Sankarasubbu, "Pay attention to the cough: Early diagnosis of COVID-19 using interpretable symptoms embeddings with cough sound signal processing," In *Proceedings*, 36th *Annual ACM Symposium on Applied Computing*, pp. 620-628, 2021.

[65] Chiranjibi Shah, Qian Du, and Yan Xu, "Enhanced TabNet: Attentive Interpretable Tabular Learning for Hyperspectral Image Classification," *Remote Sensing*, vol. 14, no. 3, 2022, 716.

[66] Christelle Nader, and Elias Bou-Harb, "An attentive interpretable approach for identifying and quantifying malware-infected internet-scale IoT bots behind a NAT," In *Proceedings*, 19th *ACM International Conference on Computing Frontiers*, pp. 279-286, 2022.

[67] C. Sun, S. Li, D. Cao, F. -Y. Wang and A. Khajepour, "Tabular Learning-Based Traffic Event Prediction for Intelligent Social Transportation System," in *IEEE Transactions on Computational Social Systems*, doi: 10.1109/TCSS.2022.3170934.

[68] Alana Correia de Santana, and Esther Luna Colombini, "Attention, please! A survey of neural attention models in deep learning," *Artificial Intelligence Review*, 1-88, 2022.

[69] H. Kim, Y. Ohmura and Y. Kuniyoshi, "Gaze-Based Dual Resolution Deep Imitation Learning for High-Precision Dexterous Robot Manipulation," in *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1630-1637, April 2021, doi: 10.1109/LRA.2021.3059619.

[70] X. Zhang, Y. Li, X. Zhou and J. Luo, "cGAIL: Conditional Generative Adversarial Imitation Learning—An Application in Taxi Drivers' Strategy Learning," in *IEEE Transactions on Big Data*, doi: 10.1109/TBDATA.2020.3039810.

[71] N. Piovesan, D. López-Pérez, M. Miozzo and P. Dini, "Joint Load Control and Energy Sharing for Renewable Powered Small Base Stations: A Machine Learning Approach," in *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 1, pp. 512-525, March 2021, doi: 10.1109/TGCN.2020.3027063.

[72] Jian Xie, Zixiao Ma, Kaveh Dehghanpour, Zhaoyu Wang, Yishen Wang, Ruisheng Diao, and Di Shi, "Imitation and Transfer Q-Learning-Based Parameter Identification for Composite Load Modeling," in *IEEE Transactions on Smart Grid*, vol. 12, no. 2, pp. 1674-1684, March 2021, doi: 10.1109/TSG.2020.3025509.

[73] S. Gao, C. Xiang, M. Yu, K. T. Tan and T. H. Lee, "Online Optimal Power Scheduling of a Microgrid via Imitation Learning," in *IEEE Transactions on Smart Grid*, vol. 13, no. 2, pp. 861-876, March 2022, doi: 10.1109/TSG.2021.3122570.

[74] H. T. Dinh and D. Kim, "MILP-Based Imitation Learning for HVAC Control," in *IEEE Internet of Things Journal*, vol. 9, no. 8, pp. 6107-6120, 15 April15, 2022, doi: 10.1109/JIOT.2021.3111454.

[75] Y. Zhang, F. Qiu, T. Hong, Z. Wang and F. Li, "Hybrid Imitation Learning for Real-Time Service Restoration in Resilient Distribution Systems," in *IEEE Transactions on Industrial Informatics*, vol. 18, no. 3, pp. 2089-2099, March 2022, doi: 10.1109/TII.2021.3078110.

[76] S. Das, "Deep Neural Networks," YouTube, Jan. 31, 2022 [Video file]. Available: https://www.youtube.com/playlist?list=PL_4Jjqx0pZY-SIO8jElzW0lNpzjcunOx4 [Accessed: April 1, 2022].

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

[80] Open AI, "Part 2: Kinds of RL Algorithms" 2018.
Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html [Accessed: April 15, 2022].

[81] R. Bellman, *Dynamic Programming*, Princeton, 1957.

[82] R. Bellman, "A Markovian Decision Process", *Journal of Mathematics and Mechanics*, vol. 6, no. 5, 1957, pp. 679–84, ISSN: 00959057.

[83] R. Howard, *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[84] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang, "A theoretical analysis of deep Q-learning," In *Learning for Dynamics and Control*, PMLR, 2020, pp. 486-489.

[85] R. S. Sutton, and A. G. Barto, *Reinforcement Learning: An Introduction*, Bradford Books, MIT Press, Cambridge, MA, 1998, revised 2018.
https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[86] C. J. C. H. Watkins, *Learning from Delayed Rewards*. PhD Thesis, University of Cambridge, England, 1989.

[87] Gavin A. Rummery, and Mahesan Niranjan, "On-line Q-learning using connectionist systems," *Technical Report*: *Department of Engineering*, *University of Cambridge*, Vol. 37, 1994.

[88] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3–4), pp. 229-256, 1992.

[89] M. Riedmiller, "Neural fitted Q iteration-first experiences with a data efficient neural reinforcement learning method," in *European Conference on Machine Learning*, Springer, 317–328, 2005.

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

[91] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver, "Prioritized experience replay," *arXiv preprint*, arXiv:1511.05952, 2015.

[92] H. Hasselt, "Double Q-learning," *Advances in Neural Information Processing Systems*, vol. 23, pp. 2613-2621, 2010.

[93] Andreas Pentaliotis, "Investigating Overestimation Bias in Reinforcement Learning," M.S. thesis, University of Groningen, Univ., Melbourne, 2020 [Online]. Available: https://www.ai.rug.nl/~mwiering/Thesis-Andreas-Pentaliotis.pdf [Accessed: April 1, 2022].

[94] Hado van Hasselt, Arthur Guez, and David Silver, "Deep reinforcement learning with double Q learning," In *Proceedings of the* 30th *AAAI Conference on Artificial Intelligence*, vol. 30, no. 1. 2016.

[95] Scott Fujimoto, Herke Hoof, and David Meger, "Addressing function approximation error in actor-critic methods," In *International Conference on Machine Learning*, PMLR, pp. 1587-1596, 2018.

[96] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *International Conference on Machine Learning*, PMLR, pp. 1861-1870, 2018.

[97] Haobo Jiang, Jin Xie, and Jian Yang, "Action Candidate Driven Clipped Double Q-learning for Discrete and Continuous Action Tasks," *arXiv preprint*, arXiv:2203.11526, 2022.

[98] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," in 33rd *International Conference on Machine Learning*, *PMLR*, vol 48, pp. 1995-2003, 2016.

[99] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[100] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1057-1063, 1999.

[101] Kamil Ciosek, and Shimon Whiteson, "Expected policy gradients for reinforcement learning," *arXiv preprint*, arXiv:1801.03326, 2018.v

[102] Philip S. Thomas, and Emma Brunskill, "Policy gradient methods for reinforcement learning with function approximation and action-dependent baselines," *arXiv preprint*, arXiv:1706.06643, 2017.

[103] Lex Weaver, and Nigel Tao, "The optimal reward baseline for gradient-based reinforcement learning," in *Proceedings*, 17th *Conference on Uncertainty in Artificial Intelligence*, pp. 538-545, 2001.

[104] Sueli I. R. Costa, Sandra A. Santos, and João E. Strapasson, "Fisher information distance: A geometrical reading," *Discrete Applied Mathematics*, vol. 197, pp. 59-69, 2015.

[105] S. Kakade, "A Natural Policy Gradient," in *Advances in Neural Information Processing Systems*, vol. 14, 2001.

[106] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz, "Trust region policy optimization," *Proceedings of the* 32nd *International Conference on Machine Learning*, *PMLR*, vol. 37, pp.1889-1897, 2015.

[107] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, "Proximal policy optimization algorithms," *arXiv preprint* arXiv:1707.06347, 2017.

[108] Vijay R. Konda, and John N. Tsitsiklis, "On actor-critic algorithms," *SIAM Journal on Control and Optimization*, Vol. 42, no. 4, 2003, pp. 1143-1166.

[109] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," In *International Conference on Machine Learning*, PMLR, pp. 1928-1937, 2016.

[110] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint*, arXiv:1509.02971v6, 2017.

[111] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, “Scalable deep reinforcement learning for vision-based robotic manipulation,” In *Conference on Robot Learning*, *PMLR*, pp. 651-673, October 2018.

[112] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas, “Sample efficient actor-critic with experience replay." *arXiv preprint*, arXiv:1611.01224, 2016.

[113] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller, “Deterministic policy gradient algorithms,” In *International Conference on Machine Learning*, PMLR, pp. 387-395, 2014.

[114] Lingheng Meng, Rob Gorbet, and Dana Kulić, “The effect of multi-step methods on overestimation in deep reinforcement learning,” In 25th *International Conference on Pattern Recognition* (ICPR), pp. 347-353, 2021.

[115] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine, “Soft actor-critic algorithms and applications,” *arXiv preprint*, arXiv:1812.05905, 2018.

[116] Hammou Ou Ali, M. Ouassaid, Mohamed Maaroufi, “Chapter 24: Optimal appliance management system with renewable energy integration for smart homes,” *Renewable Energy Systems*, Academic Press, 2021, pp. 533-552, https://doi.org/10.1016/B978-0-12-820004-9.00025-5.

[117] Swati Sharda, Mukhtiar Singh, Kapil Sharma, "Demand side management through load shifting in IoT based HEMS: Overview, challenges and opportunities," *Sustainable Cities and Society*, Volume 65, 2021, 102517, ISSN 2210-6707, https://doi.org/10.1016/j.scs.2020.102517.

[118] C. Withanage, R. Ashok, C. Yuen and K. Otto, "A comparison of the popular home automation technologies," *IEEE Innovative Smart Grid Technologies - Asia (ISGT ASIA)*, 2014, pp. 600-605, doi: 10.1109/ISGT-Asia.2014.6873860.

[119] G. Van de Kaa, S. Stoccuto, C. Valladolid Calderón, "A battle over smart standards: Compatibility, governance, and innovation in home energy management systems and smart meters in the Netherlands," *Energy Research & Social Science*, Volume 82, 2021, 102302, ISSN 2214-6296, https://doi.org/10.1016/j.erss.2021.102302.

[120] Batchu Rajasekhar, Wayes Tushar, Clement Lork, Yuren Zhou, Chau Yuen, Naran M. Pindoriya, and Kristin L. Wood, “A survey of computational intelligence techniques for air-conditioners energy management,” *IEEE Transactions on Emerging Topics Computational Intelligence*, vol. 4, no. 4, pp. 555–570, Aug. 2020.

[121] C. Huang, H. Zhang, L. Wang, X. Luo and Y. Song, "Mixed Deep Reinforcement Learning Considering Discrete-Continuous Hybrid Action Space for Smart Home Energy Management," *Journal of Modern Power Systems and Clean Energy*, doi: 10.35833/MPCE.2021.000394.

[122] L. Yu et al., "Deep Reinforcement Learning for Smart Home Energy Management," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2751-2762, April 2020, doi: 10.1109/JIOT.2019.2957289.

[123] Mohammad Esrafilian-Najafabadi, Fariborz Haghighat, "Occupancy-based HVAC control systems in buildings: A state-of-the-art review," *Building and Environment*, Vol. 197, 2021, 107810, ISSN 0360-1323, doi.org/10.1016/j.buildenv.2021.107810.

[124] Lizhi Jia, Shen Wei, Junjie Liu, "A review of optimization approaches for controlling water-cooled central cooling systems," *Building and Environment*, Volume 203, 2021, 108100, ISSN 0360-1323, https://doi.org/10.1016/j.buildenv.2021.108100.

[125] L. Yu et al., "Multi-Agent Deep Reinforcement Learning for HVAC Control in Commercial Buildings," *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 407-419, Jan. 2021, doi: 10.1109/TSG.2020.3011739.

[126] Sarah Noye, Rubén Mulero Martinez, Laura Carnieletto, Michele De Carli, Amaia Castelruiz Aguirre, "A review of advanced ground source heat pump control: Artificial intelligence for autonomous and adaptive control," *Renewable and Sustainable Energy Reviews*, Volume 153, 2022, 111685, ISSN 1364-0321, https://doi.org/10.1016/j.rser.2021.111685.

[127] A. Paraskevas, D. Aletras, A. Chrysopoulos, A. Marinopoulos, and D. I. Doukas, “Optimal Management for EV Charging Stations: A Win–Win Strategy for Different Stakeholders Using Constrained Deep Q-Learning,” *Energies*, vol. 15, no. 7, p. 2323, Mar. 2022, doi: 10.3390/en15072323.

[128] Mifeng Ren, Xiangfei Liu, Zhile Yang, Jianhua Zhang, Yuanjun Guo, Yanbing Jia, "A novel forecasting based scheduling method for household energy management system based on deep reinforcement learning," *Sustainable Cities and Society*, Volume 76, 2022, 103207, ISSN 2210-6707, https://doi.org/10.1016/j.scs.2021.103207.

[129] F. Alfaverh, M. Denaï and Y. Sun, "Demand Response Strategy Based on Reinforcement Learning and Fuzzy Reasoning for Home Energy Management," *IEEE Access*, vol. 8, pp. 39310-39321, 2020, doi: 10.1109/ACCESS.2020.2974286.

[130] Ioannis Antonopoulos, Valentin Robu, Benoit Couraud, Desen Kirli, Sonam Norbu, Aristides Kiprakis, David Flynn, Sergio Elizondo-Gonzalez, Steve Wattam, "Artificial intelligence and machine learning approaches to energy demand-side response: A systematic review," *Renewable and Sustainable Energy Reviews*, Volume 130, 2020, 109899, ISSN 1364-0321, https://doi.org/10.1016/j.rser.2020.109899.

[131] T. Chen and W. Su, "Indirect Customer-to-Customer Energy Trading With Reinforcement Learning," *IEEE Transactions on Smart Grid*, vol. 10, no. 4, pp. 4338-4348, July 2019, doi: 10.1109/TSG.2018.2857449.

[132] Mathieu Bourdeau, Xiao qiang Zhai, Elyes Nefzaoui, Xiaofeng Guo, Patrice Chatellier, "Modeling and forecasting building energy consumption: A review of data-driven techniques," *Sustainable Cities and Society*, Volume 48, 2019, 101533, ISSN 2210-6707, https://doi.org/10.1016/j.scs.2019.101533.

[133] Nan Ma, Dorit Aviv, Hongshan Guo, William W. Braham, "Measuring the right factors: A review of variables and models for thermal comfort and indoor air quality," *Renewable and Sustainable Energy Reviews*, Volume 135, 2021, 110436, ISSN 1364-0321, https://doi.org/10.1016/j.rser.2020.110436.

[134] André F. T. Martins, Ramón Fernandez Astudillo, "From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification," In *International Conference on Machine Learning*, pp. 1614-1623, PMLR, 2016.

[135] Elad Hoffer, Itay Hubara, and Daniel Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," *Advances in Neural Information Processing Systems*, 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

[136] N. Djongyang, R. Tchinda, D. Njomo, "Thermal comfort: A review paper," *Renewable and Sustainable Energy Reviews*, Volume 14, Issue 9, 2010, Pages 2626-2640, ISSN 1364-0321, https://doi.org/10.1016/j.rser.2010.07.040.