Overview of reinforcement learning

The control task is modeled as a Markov decision process (MDP) in RL, defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$, where $\mathcal{S}$ is the set of states $s$, $\mathcal{A}$ is a set of actions $a$, $\mathcal{T}$ indicates the world dynamics as transition probabilities $P(s_{t+1}|s_t, a_t)$, $r(s,a)$ is the reward at state $s$ and action $a$, and $\gamma \in (0, 1]$ is the factor to discount future rewards. A control strategy is defined by a (stochastic) policy $\pi(a|s)$ which determines a probability distribution over actions given the current state $s$. We can also define a deterministic policy $\mu(s) = \text{arg max}_a \pi(a|s)$, which chooses an action based on the mode of the probability distribution $\pi(a|s)$. Reinforcement learning chooses a policy $\pi^* = \text{arg max}_\pi R(\pi)$ to maximize the expected reward:

$$R(\pi) = E_{(s_t, a_t) \sim \pi} \left[ \sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right],$$

where the expectation is taken over both the policy to choose an action given the current state and the world dynamics. Existing methods typically learn a neural network $\pi^\theta$ with parameters $\theta$ to approximate the policy, and from a practitioner point of view, they have four groupings based on how the optimal policy is determined:

**Policy gradient methods** directly optimize the policy parameters $\theta$ by estimating the gradient of the expected return

$$\nabla_\theta R(\pi^\theta) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi(a_t^i|s_t^i; \theta) (R_t^i - b_t^i)$$

where $R_t^i = \sum_{t'=t}^{T} \gamma^{t'-t} r(s_t^i, a_t^i)$, and $b_t^i$ is a baseline value that only depends on the state $s_t^i$ at time $i$ in trajectory $i$ to reduce variance. To improve the policy $\pi^\theta$, an ascent step is taken in the direction of the estimated gradient (i.e., stochastic gradient method). Examples in this category include REINFORCE [10], natural policy gradient [1], and Trust Region Policy Optimization [7].

**Value-based algorithms** like Q-learning do not aim at optimizing the policy directly, but instead approximates the Q-value $Q(a|s)$ of the optimal policy for the available actions $a \in \mathcal{A}$ at state $s \in \mathcal{S}$:

$$Q(a|s) = r(s, a) + \gamma \mathbb{E}_{a'} \text{max}_{a'} Q(s', a')$$

where the expectation is taken over the next state $s'$ given by the physical dynamics. The Q-value function can be represented by a neural network (e.g., Deep Q-network [5]) trained to minimize the Bellman error (i.e., the mismatch of the terms on both sides of (3)). The corresponding (deterministic) policy is then determined by taking the action to maximize the Q-value:

$$\mu(s) = \text{arg max}_a Q(s, a)$$
**Actor-critic algorithms** keep an estimate of the value function (critic) as well as a policy that maximizes the value function (actor). For example, deep deterministic policy gradient (DDPG) sample a minibatch data of size $B$ from a “replay pool”, and trains the critic $Q$ via gradient descent on the Bellman error $\frac{1}{B} \sum_{i=1}^{B} (y_i - Q_{\phi}(s_i, a_i))^2$, where $y_i = r(s_i, a_i) + \gamma Q'_{\phi'}(s'_{i}, \mu'_{\theta'}(s'_{i}))$ is the target Q-value given by the previous Q-value function [3]. The gradient step for training the actor is given by the chain rule, namely:

$$\nabla_{\theta} R(\mu_{\theta}) = \sum_{i=1}^{B} \nabla_{a} Q_{\phi}(s_{i}, a)|_{a=\mu_{\theta}(s_{i})} \nabla_{\theta} \mu_{\theta}(s_{i}).$$  \hspace{1cm} (5)

Another method known as Asynchronous Advantage Actor-Critic (A3C) uses asynchronous gradient update to speed up and stabilize learning of a stochastic policy [4].

**Model-based methods** focus on the learning of the transition model for the underlying dynamics, and then use it for planning or to improve a policy. For example, the Dyna architecture simultaneously uses experience to build a model and to adjust the policy, yielding strategies that are both more effective than model-free learning and more computationally efficient than the certainty-equivalence approach [9]. The guided policy search combines iteratively refitted local linear models with complex neural network policy learning to optimize trajectory distributions for large, continuous problems [2].

In practice, it is unlikely to know all the operating points of building equipment and occupancy activities to fully observe the state (due to sensor noise, sensor limitations, and dynamic and unpredictable environment), resulting in the automation task as a partially observable Markov decision process (POMDP), which requires additional notations to denote the observations and the observation probability. To deal with partially observable task, it is necessary to integrate past observations and actions to infer about the current state, such as by adopting recurrent policies

**RL framework for HVAC control**

*Building control RL framework*

We model the task of HVAC control as a reinforcement learning, with the following specifications:

**State space $S$:** Because a building is a system of systems such as lighting, HVAC, water and electrical network, a building state can be characterized by the operation status of each sub system (e.g., device operating schedules, water temperature), as well as occupancy (e.g., occupant activities and events) and context (e.g., weather, time of the day, indoor environment, energy consumption). These information will be incorporated into the state as *features* with continuous or discrete values.

**Action space $A$:** For HVAC control, a range of parameters can be tuned, such as heating water temperature, room-level air handler fan speed, and building-level supply air temperature set point and flow rate. In practice, the set of controllable parameters is building dependent and equipment bound.

**Building dynamics $T$:** The thermal environment evolution inside a building is a physical process, and can be generally encoded in a from of transition probabilities, e.g., the probability that the room temperature increases 1 degree Celsius given a 10% increase in supply air flow rate. Nevertheless, it is challenging to accurately model the system due to various factors such as location of air vents, equipment operating mechanism, and stochastic changes of occupancy.
**Reward function** $r$: Peoples preferences can be expressed by designing the reward at each state $s$ and action $a$, e.g., energy savings and occupant comfort. As for the discount factor $\gamma$, a value close to 1, e.g., 0.99, is meaningful, and we also typically define a horizon $T$ of optimization, e.g., 24 hours.

**Building control strategy** $\pi(a|s)$: A desirable HVAC controller maximizes people’s preferences during building operations, and can be formulated as a (stochastic) policy $\pi(a|s)$ which provides a probability distribution over actions given the current observation of building state $s$. For practical reasons, we can also define a deterministic policy $\mu(s) = \arg\max_a \pi(a|s)$, which chooses the best available action.

**Learning the optimal RL policy**

Based on the policy gradient method, we directly optimize the policy parameters $\theta$ by estimating the gradient of the expected return given in (1). In each iteration, we perform the current policy and collect $N$ traces $\{\tau^i\}$, where each trace consists of $T$ instantiations $(s^t_i, a^t_i, r^t_i)$. We then use this batch of samples to estimate the gradient in (2), and perform stochastic gradient ascent to update the policy. We then iterate the process of data collection using the updated policy until convergence.

However, we noticed that the learning performance is unstable (e.g., the evaluated reward consistently decreases instead of increases), and the learned policy produce results that can not be interpreted (e.g., the action changes dramatically at a short time scale). Thus, we investigated a new strategy to stablize the learning. Let $\mu_\theta(s^t_i)$ denote the deterministic action strategy for state $s^t_i$ at time $t$, and let $a^t_{i-1}$ represent the previous action taken by the controller. We added the following term to penalize the erratic behavior of neural network:

$$L_{\text{smooth}}(\theta) = -\frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \| \mu_\theta(s^t_i) - a^t_{i-1} \|_2^2,$$

where $\| \cdot \|_2^2$ is the standard squared loss function. This penalty is added to the original objective (1) with a positive weight $\lambda$. This significantly enhanced the stability during training, and the learned policy produce smooth actions that can be interpreted by a domain expert.

**Expert knowledge encoding**

Because people have been operating building HVAC for decades, there is an accumulated knowledge base of experiences and heuristics. While these rules might not be data-driven and not necessarily optimized for individual buildings, they provide informative baselines to help speed up policy gradient training. We explore the possibilities of encoding expert knowledge in two distinct approaches:

**Guidance via experiences replay**: When the baseline control experiences in the form of state action pairs $\{s^i, a^E_i\}_{i=1}^B$ are available but the baseline policy itself is unknown, we can initialize our neural network controller $\mu_\theta$ to clone the behavior of the baseline policy by minimizing the mismatch loss $\sum_i \| a^E_i(s^i) - \mu_\theta(s^i) \|_2^2$, as inspired by the idea of imitation learning [6]. Throughout the training, we continuously use the baseline data pool for tuning and stabilizing the algorithm using importance sampling [8].

**Guidance via expert policy**: When the baseline policy $f_E(s)$ is known and can be accessed, we can directly encode it during both training and evaluation. One challenge in HVAC control is the nonstationary environment that is changing over time (e.g., the cooling demand during summer
is much higher than during winter). This will make the rewards fluctuating dramatically during training. To reduce the variance of rewards, we propose to evaluate the baseline reward \( r(s, f_E(s)) \), which will be used to offset the nominal reward obtained by the current policy. In essence, our goal is transformed from a generic requirement of learning an optimal policy to more specific task of learning a better policy than the baseline, which has been found highly effective to improve the policy performance. Another mechanism that we investigated is to learn a policy on top of the baseline strategy, \( \mu(s) + f_E(s) \), which “compliments” or “corrects” the baseline. We demonstrated the effectiveness in our case study.

References