Robustified Learning for Online Optimization with Memory Costs

Pengfei Li
UC Riverside

Jianyi Yang
UC Riverside

Shaolei Ren
UC Riverside

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10228998

***Abstract*—Online optimization with memory costs has many real-world applications, where sequential actions are made without knowing the future input. Nonetheless, the memory cost couples the actions over time, adding substantial challenges. Conventionally, this problem has been approached by various expert-designed online algorithms with the goal of achieving bounded worst-case competitive ratios, but the resulting average performance is often unsatisfactory. On the other hand, emerging machine learning (ML) based optimizers can improve the average performance, but suffer from the lack of worst-case performance robustness. In this paper, we propose a novel expert-robustified learning (ERL) approach, achieving both good average performance and robustness. More concretely, for robustness, ERL introduces a novel projection operator that robustifies ML actions by utilizing an expert online algorithm; for average performance, ERL trains the ML optimizer based on a recurrent architecture by explicitly considering downstream expert robustification. We prove that, for any $\lambda \geq 1$, ERL can achieve λ-competitive against the expert algorithm and $\lambda \cdot C$-competitive against the optimal offline algorithm (where C is the expert's competitive ratio). Additionally, we extend our analysis to a novel setting of multi-step memory costs. Finally, our analysis is supported by empirical experiments for an energy scheduling application.**

I. Introduction

Online optimization is a classic sequential decision problem where the agent chooses irrevocable actions at runtime without knowing the future input. Moreover, in many practical applications, action smoothness over time is highly desired. For example, for motion planning, a robot cannot move arbitrarily due to velocity and/or acceleration limitations; for data center capacity provisioning, servers cannot be turned on/off frequently to avoid excessive wear-and-tear costs and setup delays; and for energy scheduling in smart grids, quickly adjusting energy production can be very costly [23], [27], [32], [35]. Consequently, the long list of real-world applications have led to the emergence of online optimization with *memory* costs that penalize frequent action changes over time.

Adding a memory cost provides crucial regularization for online action smoothness, but also presents significant algorithmic challenges. More concretely, the memory cost essentially couples the online actions across multiple time steps, making it very challenging, if ever possible, to obtain optimal actions without knowing the future. Conventionally, this challenge has been approached by expert-designed online algorithms under various settings [24], [27], [28], [35], [42]. These expert algorithms typically have worst-case performance robustness in terms of guaranteed competitive ratios even for adversarial inputs, but their conservative nature also means that they may not perform very well on average in many typical cases.

More recently, the abundance of historical data in practical applications has been fueling machine learning (ML) approaches to solve optimization problems [3], [23], [32]. In particular, optimizers based on *offline*-trained recurrent neural networks or reinforcement learning have been emerging for various online optimization problems, including online resource allocation [15], online knapsack [21], among others. These ML-based optimizers exploit the statistical information about problem inputs and the strong prediction power of neural networks, empirically achieving unprecedented *average* performance. But, they also have a significant drawback — lack of performance robustness. Specifically, unlike expert online algorithms that have guaranteed robustness, the competitive ratio of ML-based optimizers can be arbitrarily bad, e.g., when training-testing distributions differ, testing inputs are adversarial, and/or the model capacity is stringently limited [6], [23], [32]. As a result, the lack of robustness invalidates the existing ML-based optimizers for online optimization in many real applications, especially those high-stake ones.

To exploit the power of both ML and expert designs, ML-augmented online algorithms have been recently proposed [7], [11], including in the context of online optimization with memory costs that we focus on [6], [23], [32]. The most common goal of these studies is to achieve a finite competitive ratio (i.e., robustness) to bound the worst-case performance for arbitrarily bad ML outputs and a low competitive ratio (i.e., consistency) in order to approximately retain good average-case performance enabled by ML models. Nonetheless, there exist substantial challenges to *simultaneously* achieve good robustness and consistency for our problem setting (see broadly relevant algorithms [6], [23], [32]), let alone that a good consistency may not always translate into a good average performance. Moreover, the existing ML-augmented algorithms often view the ML model as an exogenous blackbox that is pre-trained as a standalone model without being aware of the downstream expert algorithm. This essentially creates a *mismatch* between training and testing — the ML model is trained alone but tested together with a downstream algorithmic procedure — which can unnecessarily hurt the resulting average performance.

In this paper, we focus on online optimization with memory costs and propose a novel expert-robustified learning (ERL) approach, achieving both good average performance and guaranteed robustness. The key idea of ERL is to *let the expert* *and ML do what they are best at respectively*: for guaranteed robustness, ERL utilizes an expert online algorithm to robustify the ML actions by projecting them into a carefully designed robust action space; for good average performance, ERL trains the ML model by explicitly considering the downstream expert robustification process, thus avoiding the mismatch between training and testing. We prove that, for any trust hyperparameter $\lambda \geq 1$ governing how much flexibility we allow for ML actions, ERL can achieve λ-competitive against the expert online algorithm and hence $\lambda \cdot C$-competitive against the optimal offline algorithm (where C is the expert's competitive ratio). The added robustification step is an implicit layer, making it non-trivial to perform backpropagation. Thus, we also derive gradients of the robustification step with respect to their inputs for efficient end-to-end training, thus improving the average-case performance. We subsequently extend our analysis to a novel setting, where the memory cost spans multiple steps. Finally, we run experiments to empirically validate ERL for an energy scheduling application, demonstrating that it can offer the best average cost and competitive ratio tradeoff.

II. Related Works

Online optimization with (single-step) memory costs has been extensively approached under various settings by expert algorithms, such as online gradient descent (OGD) [43], online balanced descent (OBD) [10], and regularized OBD (R-OBD) [17]. Additionally, expert algorithms with the knowledge of future inputs include receding horizon control (RHC) [13] committed horizon control (CHC) [9], receding horizon gradient descent (RHGD) [24], [26]. These algorithms are judiciously designed to have bounded competitive ratios and/or regrets, but they may not perform well on average.

ML-augmented algorithm designs have also been emerging in the context of online optimization with memory costs [6], [32]. Nonetheless, these algorithms simply take the actions produced by an exogenous ML-based optimizer as additional inputs; they still focus on on manual designs, which cannot achieve good worst-case and average performance simultaneously. For example, in order to retain the good average performance of ML actions by setting a hyperparameter $\delta \to 0$, the competitive ratio when ML actions are arbitrarily bad is as high as $\frac{12+o(1)}{\delta}\left(\frac{2}{\alpha+\delta(1+\alpha)}\right)^{2/(\delta\alpha)}$ for α-polyhedral cost functions [32]. The study [23] considers a squared single-step switching cost and trains an ML model to regularize online actions, but its worst-case competitive ratio is unbounded. In orthogonal contexts, by assuming a given downstream algorithm, [16] re-trains an ML model for the count-min sketch problem. Therefore, the novel expert robustification (for tunable and bounded performance robustness), end-to-end training (for good average performance), and new problem settings altogether separate our work far apart from the literature.

Learning to optimize (L2O) based on offline-trained recurrent neural networks or reinforcement learning [22] has been recently applied for online optimization, including online resource allocation and online bipartite matching [15], [21]. Nonetheless, even with the help of adversarial training [14], a crucial drawback of the existing ML-based optimizers is the lack of guaranteed performance robustness, making them inapplicable for high-stake applications. Naive techniques that choose whichever is better between L2O and a conventional solver [20] do not apply to online optimization due to unknown future inputs and irrevocable actions.

ERL is relevant to the recent decision-focused learning framework [38]. But, ERL goes beyond simply training the ML model by proposing a novel expert robustification framework. Moreover, ERL directly uses the robustified actions to determine the training loss, whereas the existing decision-focused learning requires groundtruth labels in the training loss.

Finally, ERL intersects with conservative exploration in bandits and reinforcement learning [40]. Conservative exploration focuses on unknown reward functions (and transition models if applicable) and uses an existing policy to guide the exploration process for robustness. But, its design is dramatically different in the sense that it does not need to account for future input uncertainties when making an action for each step (or choosing a policy for each episode in case of episodic reinforcement learning), i.e., only the cumulative rewards matter. By contrast, ERL must hold a reservation cost to ensure that it always has a feasible solution given any future inputs, achieving a guaranteed deterministic worst-case competitive ratio (rather than probabilistic guarantees). This key point can also be highlighted by noting that, even assuming perfect reward functions (and transition models), the robustification rule used by the the existing conservative bandits/reinforcement learning [40] cannot apply to our problem to achieve a guaranteed competitive ratio. Other related problems include constrained policy optimization and safe reinforcement learning [30], [39]. These studies focus on constraining the *average* safety costs and/or avoiding certain dangerous states (possibly with a high probability). By contrast, ERL has a different goal and guarantees a bounded competitive ratio in *any* case by introducing a novel expert robustification step.

III. Formulation for Single-Step Memory Cost

To facilitate readers' understanding, we begin with a single-step memory cost (a.k.a. switching cost [27], [28], [42]). Consider a sequence of T time steps as a problem instance. At each step $t = 1, \cdots, T$, the agent receives a context vector/parameter $y_t \in \mathcal{Y} \in \mathbb{R}^m$ for the hitting cost, makes an *irrevocable* action $x_t \in \mathcal{X} \subseteq \mathbb{R}^d$, and then incurs a hitting cost of $f(x_t, y_t) \geq 0$. To encourage smoothed actions over time, the agent also incurs a memory cost $d(x_t, x_{t-1}) \geq 0$ defined in terms of the distance between two adjacent actions in a metric space. Concretely, we consider $d(x_t, x_{t-1}) = \|x_t - x_{t-1}\|$, where $\|\cdot\|$ denotes l_p norm with $p \geq 1$. Thus, the goal of the agent is to minimize the sum of the hitting costs and the memory costs over a sequence of T steps as follows:

$$\min_{x_1, \cdots x_T} \sum_{t=1}^{T} f(x_t, y_t) + d(x_t, x_{t-1}), \tag{1}$$

where the initial action x_0 is provided as an additional input. While we can alternatively impose a constraint on the total memory cost, our formulation of adding the memory cost as a smoothness regularizer for online actions is consistent with the existing literature [23], [27], [32], [42].

The key challenge for optimally solving Eqn. (1) comes from the memory cost that couples online actions over time, but y_t is not revealed to the agent until the beginning of each step $t = 1, \cdots, T$. Given any online algorithm π, we denote its total cost for a problem instance with input context $\boldsymbol{s} = (x_0, \boldsymbol{y}) \in \mathcal{S} = \mathcal{X} \times \mathcal{Y}^T$ as $\mathrm{cost}(\pi, \boldsymbol{s}) = \sum_{t=1}^{T} f(x_t^\pi, y_t) + d(x_t^\pi, x_{t-1}^\pi)$, where x_t^π, $t = 1, \cdots, T$, are the actions produced by the algorithm π. While $\boldsymbol{s}$ follows a general distribution that can be well addressed by ML-based optimizers, it can still contain adversarial cases. For simplicity, we will omit the context parameters, and denote $\mathrm{cost}(x_{i:j}^\pi) = \sum_{t=i}^{j} f(x_t^\pi, y_t) + d(x_t^\pi, x_{t-1}^\pi)$, where $x_{i:j}^\pi = (x_i^\pi, \cdots, x_j^\pi)$ are the actions for $t = i, \cdots, j$ under the algorithm π.

Definition 1 (α**-polyhedral):** Given a context parameter $y \in \mathcal{Y}$, the hitting cost function $f(x, y) : \mathcal{X} \mapsto \mathbb{R}^+$ is called α-polyhedral for $\alpha > 0$ if it has a unique minimizer $x^* \in \mathcal{X}$ and satisfies $f(x, y) - f(x^*, y) \geq \alpha \cdot d(x, x^*)$ for any $x \in \mathcal{X}$.

*Definition 2 (***Competitive ratio):** For $\lambda \geq 1$, an online algorithm ALG is called λ-competitive against the algorithm π subject to an additive factor $B \geq 0$ if its total cost satisfies $\mathrm{cost}(ALG, \boldsymbol{s}) \leq \lambda \cdot \mathrm{cost}(\pi, \boldsymbol{s}) + B$, for any input $\boldsymbol{s} = (x_0, \boldsymbol{y})$.

The α-polyhedral definition is commonly considered in the literature [32], [42] to derive competitive ratios against the optimal offline algorithm. The deterministic competitive ratio in Definition 2 is general, and the additive factor B is independent of the problem input $\boldsymbol{s} = (x_0, \boldsymbol{y})$. By setting $B = 0$, it becomes the strict competitive ratio [19], [31], [32]. Further, with $\lambda = 1$, the additive factor B in Definition 2 captures the *regret* incurred by ALG with respect to the algorithm π. When π is not specified, the competitive ratio is against the optimal offline algorithm OPT by default.

IV. ERL: Expert-Robustified Learning

In this section, we consider a single-step memory cost and show the design of ERL.

A. A Primer on Pure ML-based Optimizers

To solve online optimization with memory costs, a natural idea is to exploit the power of ML to discover the mapping from the available online information to actions. More concretely, we can pre-train an ML model offline based on a recurrent neural network (RNN) or equivalently using reinforcement learning. We denote the ML action at time t as $\tilde{x}_t = h_w(\tilde{x}_{t-1}, y_t)$, where w is the ML model parameter. The recurrent nature comes from sequential online optimization with memory costs: given the previous action $\tilde{x}_{t-1}$ and the current input y_t, we recurrently output an online action $\tilde{x}_t$. With a set of training problem instances, the ML model parameter w can be learnt by minimizing a loss function, which can be the sum of costs in Eqn. (1) [3].

Drawbacks: It is well-known that such ML-based optimizers have significant drawbacks — lack of robustness. Specifically, the competitive ratio can be arbitrarily bad for a variety of reasons, such as distributional shifts, hard problem instances or even adversarial inputs, and/or finite ML capacity [5], [6], [32]. While distributionally robust learning can partially mitigate the lack of robustness in an average sense [36], [41], it still cannot guarantee that the ML model has a bounded competitive ratio for *any* problem instance.

B. Expert Robustification

There have been several expert algorithms to solve online optimization with memory costs under different settings [25], [42]. While these algorithms may not perform well on average due to their conservative nature, they offer worst-case performance robustness for any input. Thus, this motivates us to leverage an expert algorithm π to robustify ML actions. For each t, we denote the pre-robustification ML action as $\tilde{x}_t$, expert action as x_t^π, and post-robustification action as x_t.

A naive idea is to add a proper regularizer during the training process that imposes penalty when the total cost exceeds λ times of the expert's cost. But, this will not work, because the ML actions can still violate the robustness requirement when bad problem instances arrive during online inference. Alternatively, one may want to constrain the robustified actions such that for any $t = 1, \cdots, T$, the cumulative cost up to time t satisfies $\mathrm{cost}(x_{1:t}) \leq \lambda \mathrm{cost}(x_{1:t}^\pi) + B$, where $\mathrm{cost}(x_{1:t})$ and $\mathrm{cost}(x_{1:t}^\pi)$ are the cumulative costs of ERL and the expert (assuming that the expert would run its algorithm alone), respectively. But, this can easily result in an empty set of feasible actions for ERL, because the actions are coupled over time by memory costs. To see this point, let us consider that $\mathrm{cost}(x_{1:t}) = \lambda \mathrm{cost}(x_{1:t}^\pi) + B$ but $x_t \neq x_t^\pi$ at time t. Then, at time $t + 1$, the expert can have such a low total cost of $f(x_{t+1}^\pi, y_{t+1}) + d(x_{t+1}^\pi, x_t^\pi)$ that even setting $x_{t+1} = x_{t+1}^\pi$ (i.e., following the expert) would violate the constraint $\mathrm{cost}(x_{1:t+1}) \leq \lambda \mathrm{cost}(x_{1:t+1}^\pi) + B$ due to the large memory cost $d(x_{t+1}^\pi, x_t)$. Consequently, no actions can guarantee robustness in this case.

We now present our novel robustification framework, called ERL. To achieve robustness, the crux of ERL is to hedge against the risk of deviating from the expert action to account for future uncertainty. Specifically, at each step t, we project the ML action $\tilde{x}_t$ into a robust action space specified by the expert π by solving:

$$\begin{aligned} x_t &= \arg\min_{x \in \mathcal{X}} \frac{1}{2} \| x - \tilde{x}_t \|^2 \\ s.t. \quad & \mathrm{cost}(x_{1:t-1}) + f(x, y_t) + d(x, x_{t-1}) + d(x, x_t^\pi) \\ & \leq \lambda \mathrm{cost}(x_{1:t}^\pi) + B \end{aligned} \tag{2}$$

where $\lambda \geq 1$ and $B \geq 0$ are hyperparameters indicating the level of robustness requirement. We denote this projection step as $x_t = \mathrm{proj}(\tilde{x}_t, x_t^\pi, \mathrm{cost}(x_{1:t-1}), \mathrm{cost}(x_{1:t}^\pi))$. Importantly, the key is to add a reservation cost $d(x_t, x_t^\pi)$ when constraining the post-robustification cumulative cost at time t in Eqn. (2). By doing so, we ensure that if the constraint is satisfied at

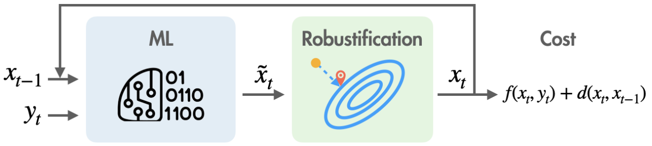


Fig. 1: ERL. Given each online input, we first run forward inference to obtain the ML action, and then project it into an expert-robusitfied action space as the actual action.

Algorithm 1 Expert-Robustified Learning (ERL)

```
Require: $\lambda \geq 1$, $B \geq 0$, initial $x_0$, trained ML model (Section IV-C), and expert online algorithm $\pi$
for $t = 1, \cdots, T$
    Receive the context $y_t$
    Expert chooses $x_t^\pi$ and ML chooses $\tilde{x}_t \leftarrow h(x_{t-1}, y_t)$
    $x_t \leftarrow \text{proj}(\tilde{x}_t, x_t^\pi, \text{cost}(x_{1:t-1}), \text{cost}(x_{1:t}^\pi))$ based on Eqn. (2)   //Robustification
```

time t, then it will be also satisfied at time $t+1$ regardless of the input — following the expert by choosing $x_{t+1} = x_{t+1}^\pi$ is always a *feasible* solution.

The ERL inference process is shown in Fig. 1 and described in Algorithm 1. At each step t, we run the ML model to produce an action $\tilde{x}_t$, get the expert's action x_t^π, and then project $\tilde{x}_t$ into a robustified action space by solving Eqn. (2). Note that the expert online algorithm takes context y_t as its input and outputs its action x_t^π independently following its own trajectory without being affected by the ML action. Next, we formally provide the robustness analysis for ERL.

Theorem 4.1: Let π be any expert online algorithm for the problem in Eqn. (1). For any $\lambda \geq 1$ and $B \geq 0$, ERL is λ-competitive against π subject to an additive factor of B, i.e., $\text{cost}(\texttt{ERL}, \boldsymbol{s}) \leq \lambda \cdot \text{cost}(\pi, \boldsymbol{s}) + B$ for any input $\boldsymbol{s} = (x_0, \boldsymbol{y})$.

Theorem 4.1 is proved in Appendix B and demonstrates the power of ERL by showing that it can achieve any competitive ratio of $\lambda \geq 1$ with respect to any expert algorithm π for $B \geq 0$. Here, given any $B \geq 0$, the hyperparameter $\lambda \geq 1$ can be viewed as the *trust* parameter: the higher λ, the more we trust the ML action $\tilde{x}_t$, thus potentially achieving a lower average cost at the expense of a higher competitive ratio. The additive factor $B \geq 0$ represents a slackness, and $B = 0$ reduces to the strict competitive ratio definition. If we set $\lambda = 1$ and B sublinear in T, ERL is guaranteed to be asymptotically no worse than the expert π even in the worst case as $T \to \infty$.

Competitive ratio of ERL against OPT. One may also desire a bounded competitive ratio against the optimal offline algorithm OPT. To this end, we consider a state-of-the-art expert online algorithm, called Robust, which minimizes the hitting cost at each step without considering the memory cost. This simple online algorithm surprisingly achieves a good competitive ratio of $\max\left(\frac{2}{\alpha}, 1\right)$ against OPT for α-polyhedral hitting cost functions [42]. By applying Theorem 4.1, we have the following corollary.

Corollary 4.1: Consider Robust as the algorithm π that chooses $x_t = \arg\min_{x \in \mathcal{X}} f(x, y_t)$ for any t. Assume that the hitting cost functions $f(x, y_t)$ are α-polyhedral. For any $\lambda \geq 1$ and $B \geq 0$, ERL is $\lambda \cdot \max\left(\frac{2}{\alpha}, 1\right)$-competitive against OPT subject to an additive factor of B, i.e., $\text{cost}(\texttt{ERL}, \boldsymbol{s}) \leq \lambda \cdot \max\left(\frac{2}{\alpha}, 1\right) \cdot \text{cost}(\texttt{OPT}, \boldsymbol{s}) + B$ for any input $\boldsymbol{s} = (x_0, \boldsymbol{y})$.

C. End-to-end Training

In conventional ML-augmented algorithms [6], [12], the ML model is trained to produce good actions on its own, without being aware of the downstream modification (i.e., expert robustification in ERL). While the designed algorithm may sometimes retain good ML actions (i.e., termed as consistency [6]), this still creates a mismatch between training and testing processes — the training process yields good *pre*-robustification ML actions, but it is *post*-robustification actions that are actually being used for testing [3], [23], [29], [38]. Thus, to improve the *average* performance of ERL, we need to explicitly consider the projection step for ML model training.

End-to-end training is highly non-trivial, since the projection step itself is an optimization problem in Eqn. (2) and hence an *implicit* layer. Additionally, unlike typical differentiable optimizers [1], [4], [38], we need to derive gradients to perform backpropagation through time due to the recurrent nature of our online optimization problem. Let w be the weight for each base ML model $\tilde{x}_t = h_w(x_{t-1}, y_t)$ in the RNN as illustrated in Fig. 1. We need to derive the gradient of $\text{cost}(x_{1:T})$ with respect to w as follows:

$$\nabla_w \text{cost}(x_{1:T}) = \sum_{t=1}^{T} \nabla_w \big(f(x_t, y_t) + d(x_t, x_{t-1})\big), \qquad (3)$$

where $x_t = \text{proj}(\tilde{x}_t, x_t^\pi, \text{cost}(x_{1:t-1}), \text{cost}(x_{1:t}^\pi))$ is the post-robustification action at step $t = 1, \cdots, T$. Thus, the total gradient can be calculated by summing up all the gradients over T steps. By applying the chain rule for step t, we have $\nabla_w\big(f(x_t, y_t) + d(x_t, x_{t-1})\big) = \nabla_{x_t}\big(f(x_t, y_t) + d(x_t, x_{t-1})\big)\nabla_w x_t + \nabla_{x_{t-1}} d(x_t, x_{t-1}) \cdot \nabla_w x_{t-1}$, where $\nabla_w x_t = \big(\nabla_{\tilde{x}_t} x_t \nabla_w \tilde{x}_t + \nabla_{\text{cost}(x_{1:t-1})} x_t \nabla_w \text{cost}(x_{1:t-1})\big)$. The gradients $\nabla_{x_t}\big(f(x_t, y_t) + d(x_t, x_{t-1})\big)$ and $\nabla_{x_{t-1}} d(x_t, x_{t-1})$ can be obtained given explicit forms of f and d, $\nabla_w \tilde{x}_t$ can be calculated easily through the backpropagation within the ML model (e.g., a neural network), and $\nabla_w \text{cost}(x_{1:t-1})$ is calculated recursively back to $t = 1$. Thus, the key is to derive the gradients of the projection operator $x_t = \text{proj}(\tilde{x}_t, x_t^\pi, \text{cost}(x_{1:t-1}), \text{cost}(x_{1:t}^\pi))$ with respect to the ML action $\tilde{x}_t$ and $\text{cost}(x_{1:t-1})$. We provide the result based on KKT conditions [8] in the following proposition.

Proposition 4.2 (Gradient by KKT conditions): Assume that x_t and μ are the primal and dual solutions to Eqn. (2), respectively. Let $\Delta_{11} = I + \mu\Big(\nabla_{x_t, x_t}\big(f(x_t, y_t) + d(x_t, x_{t-1})\big) + \nabla_{x_t, x_t} d(x_t, x_t^\pi)\Big)$, $\Delta_{12} = \nabla_{x_t}\big(f(x_t, y_t) + d(x_t, x_{t-1})\big) + \nabla_{x_t} d(x_t, x_t^\pi)$, $\Delta_{21} = \mu\Big(\nabla_{x_t}\big(f(x_t, y_t) + d(x_t, x_{t-1})\big) + \nabla_{x_t} d(x_t, x_t^\pi)\Big)^\top$, $\Delta_{22} = f(x_t, y_t) + d(x_t, x_{t-1}) + d(x_t, x_t^\pi) +$

$\text{cost}(x_{1:t-1}) - [\lambda \text{cost}(x^{\pi}_{1:t}) + B]$. The gradients of the projection operation $x_t = \text{proj}(\tilde{x}_t, x^{\pi}_t, \text{cost}(x_{1:t-1}), \text{cost}(x^{\pi}_{1:t}))$ with respect to $\tilde{x}_t$ and $\text{cost}(x_{1:t-1})$ are

$$\nabla_{\tilde{x}_t} x_t = \Delta_{11}^{-1}[I + \Delta_{12} Sc(\Delta, \Delta_{11})^{-1} \Delta_{21} \Delta_{11}^{-1}],$$

$$\nabla_{\text{cost}(x_{1:t-1})} x_t = \Delta_{11}^{-1} \Delta_{12} Sc(\Delta, \Delta_{11})^{-1} \mu,$$

where $Sc(\Delta, \Delta_{11}) = \Delta_{22} - \Delta_{21}\Delta_{11}^{-1}\Delta_{12}$ is the Schur-complement of Δ_{11} in the blocked matrix $\Delta = \big[[\Delta_{11}, \Delta_{12}], [\Delta_{21}, \Delta_{22}]\big]$.

We remark that if the Schur-complement $Sc(\Delta, \Delta_{11})$ is not full-rank (e.g., ML action $\tilde{x}_t$ lies in the boundary of the action space in Eqn. (2)) or the hitting cost function f or memory cost d is not differentiable for certain x_t, we can still approximate the gradients based on Proposition 4.2 for backpropagation. Concretely, the pseudo-inverse of $Sc(\Delta, \Delta_{11})$ can be used if $Sc(\Delta, \Delta_{11})$ is not full-rank; if f or d is not differentiable at x_t, we can use its its subgradient as a substitute. This is also a common technique to handle non-differentiable points when training ML models, especially neural networks [18]. For example, we often use 0 as a subgradient for $ReLu(x)$ at $x = 0$. Importantly, Proposition 4.2 provides a practically convenient way to perform backpropagation.

Training. As in typical ML-based approaches for online optimization [2], [3], [14], [21], we train the ML model based on pre-collected historical problem instances by using the gradients in Proposition 4.2 and explicitly considering the projection process. Additionally, we can also update the ML model online by collecting batches of new problem instances during online inference. The training process can be supervised by using the total cost $\sum_i \text{cost}_i(x_{1:T})$ as the loss where i is the index for training problem instances.

V. Extension to Multi-Step Memory Cost

Motivated by smoothness in higher-order dynamics, we now turn to a more general case where the memory cost can span multiple steps: $d(x_t, x_{t-q:t-1}) = \|x_t - \sum_{i=1}^{q} C_i x_{t-i}\|$, where $q \geq 1$ is the memory length and $C_i \in \mathbb{R}^{d \times d}$ is problem-specific. For example, let us consider a robot motion planning problem where x_t represents the position at time t and acceleration smoothness is highly desired. In this case, the memory cost can be written as $d(x_t, x_{t-2:t-1}) = \|\,(x_t - x_{t-1}) - (x_{t-1} - x_{t-2})\,\| = \|x_t - 2x_{t-1} + x_{t-2}\|$, for which we can set $C_1 = -2 \cdot I$, $C_2 = I$ and $q = 2$ where I is the identity matrix in $\mathbb{R}^{d \times d}$. Note that the expert algorithm in [35] uses the same form of multi-step memory structure, but considers a *squared* memory cost along with other strong assumptions (e.g., strongly convex hitting costs) that require entirely different techniques [42]. To our knowledge, our work is the first to consider multi-step memory costs in metric space.

Expert robustification. Given multi-step memory costs, the input to our ML model includes y_t and $x_{t-q:t-1}$ and outputs $\tilde{x}_t$, which is then robustified by solving the following:

$$\begin{aligned} x_t = \arg\min_{x \in \mathcal{X}} \frac{1}{2}\|x - \tilde{x}_t\|^2 \\ s.t. \quad \text{cost}(x_{1:t-1}) + f(x, y_t) + d(x, x_{t-q:t-1}) \\ + G(x, x_{t-q:t-1}, x^{\pi}_{t-q:t}) \leq \lambda \text{cost}(x^{\pi}_{1:t}) + B, \end{aligned} \tag{4}$$

where the reservation cost $G(x, x_{t-q:t-1}, x^{\pi}_{t-q:t})$ is given by

$$\begin{aligned} & G\left(x, x_{t-q:t-1}, x^{\pi}_{t-q:t}\right) \\ = & \sum_{k=1}^{\min(q, T-t)} \left\| C_k x + \sum_{i=1}^{q-k} C_{k+i} x_{t-i} - \sum_{i=0}^{q-k} C_{k+i} x^{\pi}_{t-i} \right\|. \end{aligned} \tag{5}$$

The key insight for Eqn. (5) is that we need to account for the potentially higher memory costs incurred by ERL compared to the expert algorithm π over up to future q steps. By holding the reservation cost for the cumulative cost at each step, we can ensure that ERL can always roll back to the expert's actions in the future without violating the robustness requirement. The ERL inference process still follows Algorithm 1, except for that the projection step for expert robustification in Line 5 is based on Eqn. (4).

Competitive ratio of Robust. Robust has a bounded competitive ratio in the single-step memory setting [42], but it is unclear in the multi-step setting. Here, we prove that Robust is also competitive in the multi-step memory case. The proof is in Appendix A.

Theorem 5.1: Assume that $f(\cdot, y_t) : \mathcal{X} \mapsto \mathbb{R}$ is α-polyhedral and that the memory cost is given by $d(x_t, x_{t-q:t-1}) = \|x_t - \sum_{i=1}^{q} C_i x_{t-i}\|$ for $t = 1, \cdots, T$, where $C_i \in \mathbb{R}^{d \times d}$ and $\sum_{i=1}^{q} \|C_i\| = \beta$ with $\|C_i\|$ being the matrix norm induced by the l_p vector norm. The Robust algorithm that chooses $x_t = \arg\min_{x \in \mathcal{X}} f(x, y_t)$ for any $t = 1, \cdots, T$ is strictly $\max\left(\frac{\beta+1}{\alpha}, 1\right)$-competitive against OPT, i.e., $\text{cost}(\texttt{Robust}, \boldsymbol{s}) \leq \max\left(\frac{\beta+1}{\alpha}, 1\right) \cdot \text{cost}(\texttt{OPT}, \boldsymbol{s})$ for any input $\boldsymbol{s} = (x_0, \boldsymbol{y})$.

Competitive ratio of ERL. In the multi-step memory case, Theorem 4.1 still holds. That is, for any $\lambda \geq 1$ and $B \geq 0$, ERL is still λ-competitive against any expert online algorithm π subject to an additive factor B. Also, by combining this result with Theorem 5.1, we obtain the following corollary (proof in Appendix B).

Corollary 5.1: Let the expert π be Robust that chooses $x_t = \arg\min_{x \in \mathcal{X}} f(x, y_t)$ for any $t = 1, \cdots, T$. Under the same assumptions as in Theorem 5.1, for any $\lambda \geq 1$ and $B \geq 0$, ERL is $\lambda \max\left(\frac{\beta+1}{\alpha}, 1\right)$-competitive against OPT subject to an additive factor of B where $\beta = \sum_{i=1}^{q} \|C_i\|$, i.e., $\text{cost}(\texttt{ERL}, \boldsymbol{s}) \leq \lambda \max\left(\frac{\beta+1}{\alpha}, 1\right) \cdot \text{cost}(\texttt{OPT}, \boldsymbol{s}) + B$ for any input $\boldsymbol{s} = (x_0, \boldsymbol{y})$.

Finally, for end-to-end training, the gradients of projection in Eqn. (4) with respect to $\tilde{x}_t$ and $\text{cost}(x_{1:t-1})$ can be derived and the ML model can be trained following the steps in Section IV-C. Hence, we omit them for brevity.

VI. Experimental Results

To empirically validate ERL, we consider the dynamic energy scheduling application in the presence of uncertain renewables. Specifically, renewable energy such as wind and solar energy is being massively incorporated into the power grid for sustainability. But, their availability is highly intermittent subject to a variety of factors such as weather conditions and equipment efficiency. On the other hand, balancing the power demand and generation is crucial to ensure grid stability — a mismatch requires rapid offsetting using alternative and potentially more expensive energy sources. Thus, a challenging problem faced by grid operators is *how to dynamically schedule energy production to meet net demands based on real-time renewable availability*. A mismatch between the production x_t and net demand y_t needs offsetting using expensive energy sources/storage and hence causes a *hitting* cost $f(x_t, y_t) = \alpha\|x_t - y_t\|$, and varying the production level over time incurs a *memory* cost $d(x_t, x_{t-1}) = \|x_t - x_{t-1}\|$ (due to generator ramp-up/-down costs). Thus, this is a typical online optimization problem with memory cost [23], [25], [42].

A. Dataset

We consider intermittent renewable energy generated using trace data and empirical equations. Specifically, for wind power, the amount of energy generated at step t is modeled based on [33] as $P_{\text{wind},t} = \frac{1}{2}\kappa_{\text{wind}}\varrho A_{\text{swept}}V^3_{\text{wind},t}$. The sympols are explained as follows: κ_{wind} is the conversion efficiency (%) of wind energy, ϱ is the air density (kg/m^3), A_{swept} is the swept area of the turbine (m^2), and $V_{\text{wind},t}$ is the wind speed (kW/m^2) at time step t. The amount of solar energy generated at step t is given based on [37] as $P_{\text{solar},t} = \frac{1}{2}\kappa_{\text{solar}}A_{\text{array}}I_{\text{rad},t}(1 - 0.05 * (\text{Temp}_t - 25))$. The symbols are explained as follows: κ_{solar} is the conversion efficiency (%) of the solar panel, A_{array} is the array area (m^2), and $I_{\text{rad},t}$ is the solar radiation (kW/m^2), and Temp_t is the temperature (°C) at step t. Thus, at time step t, the total energy generated by the renewables $P_{\text{r},t} = P_{\text{wind},t} + P_{\text{solar},t}$. Suppose at time step t, the net energy demand is $y_t = \max(P_{\text{s},t} - P_{\text{r},t}, 0)$, where $P_{\text{s},t}$ is the demand before renewable integration. The amount of energy generation is the agent's online action x_t. We model the hitting cost as the scaled l_2-norm of the difference between the action x_t and the context y_t, i.e. $f(x_t, y_t) = \alpha\|x_t - y_t\|$. Additionally, we model the switching cost by the l_2-norm of the difference between two consecutive actions, i.e. $c(x_t, x_{t-1}) = \|x_t - x_{t-1}\|$. The hitting cost parameter is set as $\alpha = 0.2$. The parameters for wind energy are set as $\kappa_{\text{wind}} = 30\%$, $\varrho = 1.23kg/m^3$, $A_{\text{swept}} = 500,000m^2$. The parameters of solar energy are set as $\kappa_{\text{wind}} = 10\%$, $A_{\text{array}} = 10,000m^2$. The other parameters, such as wind speed, solar radiation and temperature data, are all collected from the National Solar Radiation Database [34], which contains detailed hourly data for the year of 2015.

To generate datasets for training and testing, we use a sliding window to generate multiple sequences of hourly data, with each sequence length being 25 (i.e., 24 action steps plus 1 initial step). For each sequence of 25 consecutive hourly data, we can calculate the contextual information for each step/hour. We define the energy generation of the first hour as the initial action x_0. The problem can be formulated as: $\min_{x_1,\cdots x_T}\sum_{t=1}^{T}\alpha\|x_t - y_t\| + \|x_t - x_{t-1}\|$. We use the CVXPY Library to find the optimal offline solution.

B. Experimental Setup

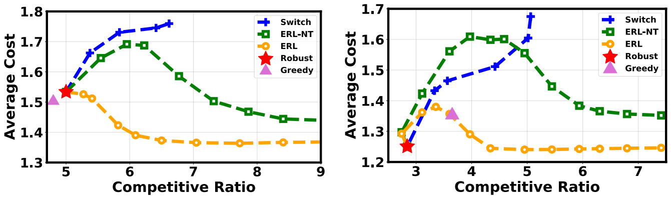
We use a RNN with 2 hidden layers, each with 8 neurons. To train this model, we use the data from the first two months (January–February) of 2015, which contains 1440 hourly weather data samples in total. Specifically, we generate 1416 data sequences using a sliding window. We train the RNN model for 140 epochs with batch size of 50. The model is implemented in PyTorch Library and the training process usually takes around 3 minutes on a 2020 MacBook Air with 8GB memory and a M1 chipset. In ERL, we set the slackness parameter $B = 0$ to follow the strict definition of competitive ratio. By default, we train ERL with $\lambda = 1.4$.

To evaluate the performance of different algorithms, we divide the remaining 10 months of 2015 into five segments, each with two months. There are three different cases: when ML empirically works better than Robust in terms of both average and worst-case performance; ML is better than Robust on average but worse in the worst case; and ML is worse than Robust both on average and in the worst case. The first case occurs for the testing segment of March–April, because the data in both training and testing datasets well consistent due to their similar weather patterns. Next, we focus on the other two cases, which are more interesting and typical since data distributional shifts between training and testing datasets are very common in practice. This is also consistent with our main contribution — robustifying ML-based optimizers.

While we can also re-train/update the ML models (in ERL, ML, and ERL-NT) based on online collected data, the existing ML-based optimizers are typically pre-trained offline [2], [15]. Thus, we keep the ML model unchanged when testing its performance, in order to highlight the role of our expert robustification step in ERL— regardless of the testing distributions, ERL offers a provable worst-case competitive ratio guarantee against the expert.

C. Baselines

We compare ERL with the following baselines. **Optimal offline (OPT):** OPT has all the context information to optimally solve Eqn. (1); **Robust expert (Robust):** Robust is the state-of-the-art expert that chooses $x_t = \arg\min_{x\in\mathcal{X}} f(x, y_t)$ for $t = 1, \cdots, T$ with guaranteed competitive ratios [42]; **Simple greedy (Greedy):** Greedy greedily minimizes the total hitting cost and memory cost at each step; **Pure ML-based optimizer (ML):** ML uses the same recurrent neural network as ERL but does not use expert robustification for training or inference; **Dynamic switching (Switch):** Switch dynamically switches between Robust and ML based on a threshold hyperparameter [6]; **ERL-NoTraining (ERL-NT):** ERL-NT uses Algorithm 1 for inference but the ML model is trained as a standalone optimizer without end-to-end training.



(a) September-December testing

(b) May-August testing

Fig. 2: Normalized average cost vs. empirical competitive ratio. ML is off the charts: (left): average cost 1.45 and competitive ratio 50+; (right): average cost 1.367 and competitive ratio 11.167.

Although Greedy may empirically perform better than Robust, it does not have a provably-bounded competitive ratio whereas Robust has one (see [42] for the single-step memory case and our Theorem 5.1 for the multi-step memory case). Thus, we use Robust as our expert in ERL.

D. *Results*

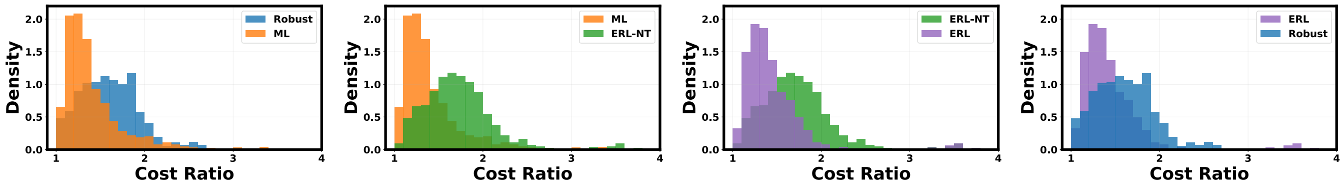
September-December testing. We obtain the empirical results of average cost vs. competitive ratio in Fig. 2. All the average costs are normalized with respect to the average cost of OPT. ML achieves a lower average cost than Robust, but its empirical competitive ratio is way larger due to the common drawback of ML-based optimizers — lack of performance robustness. Specifically, the training and testing distributions are rarely identical in practice, which can lead to an extremely bad competitive ratio for ML. While Greedy empirically performs better than Robust in this setting, it does not have any competitive ratio guarantees. We see that Switch performs badly compared to Robust, because it imposes a hard switch based on a pre-set threshold regardless of the actual performance of Robust or ML. Compared to ML, ERL-NT can have a much lower competitive ratio due to expert robustification, but the average cost also increases dramatically and can be even higher than Robust (because the ML model training in ERL-NT is not aware of the robustification step). On the other hand, ERL achieves a guaranteed competitive ratio and a much lower average cost than ERL-NT. This highlights the benefits of training the ML model in ERL by explicitly considering the downstream expert robustification process. Interestingly, we also observe that by properly setting the hyperparameter λ (around $1.4 \sim 1.8$ in our case), ERL can have an even lower average cost than ML. This is because for those *hard* problem instances that ML cannot solve well, ERL has Robust as its guidance to provide reasonably good solutions.

Cost ratio distribution. To provide further insights, we also show in Fig. 3 the detailed comparison between different algorithm pairs in terms of the cost ratio distribution density. By looking at Robust vs. ML in Fig. 3(a), we can see that ML has low cost ratios in more cases than Robust, although it has a long tail (not shown in the figure due to the axis limit). This explains that ML can have good average performance than the expert algorithm Robust, when the training-testing distributions are not very different. Nonetheless, ML still suffers from the lack of robustness, while Robust does not. Comparing ML with ERL-NT in Fig. 3(b), we can see that expert robustification can shift the cost ratios rightwards (i.e., increasing the average cost), but ERL-NT has guaranteed robustness. Next, we observe from Fig. 3(c) that the cost ratios of ERL are shifted leftwards compared to ERL-NT, demonstrating the importance of training ERL with explicit consideration of the expert robustification process. Fig. 3(d) shows that ERL has many smaller cost ratios than Robust. Again, this shows the importance of considering expert robustficniation during the training process.

Impacts of λ. While both ERL and ERL-NT can guarantee robustness due to the expert robustification step during inference, the ML model in ERL is trained with explicit consideration of the expert whereas ERL-NT simply trains the ML model as a standalone optimizer. Thus, ERL can further improve the average performance compared to ERL-NT. To further highlight the necessity of being aware of the expert robustification step in the training of ERL, we show the results for different algorithms in Table I. By training ERL using the same λ as testing it, we can obtain both the best average cost and the best competitive ratio empirically. In particular, the difference in terms of the average performance is more prominent when $\lambda = 1.4$ than when $\lambda = 1.2$. This can be explained by noting that with a larger $\lambda \geq 1$, the expert plays a less significant role by placing less emphasis on robustness and providing the ML model with more freedom. Then, when $\lambda = 1.4$, the average performance is better than when $\lambda = 1.2$, although its guaranteed competitive ratio is higher (which is also empirically verified in Table I). For reference, we also show the performance of other algorithms that are not affected by $\lambda \geq 1$.

May-August testing. Next, we turn to a more challenging case in which ML is outperformed by Robust both on average and in the worst case (May–August, due to the different weather patterns and hence large training-testing distributional shifts). This is not uncommon in practice, since ML models can have arbitrarily bad performances due to the lack of robustness. We show the results in Fig. 2(b). All the average values are normalized with respect to the average cost of OPT.

Again, ML is off the charts, with its competitive ratio as 11.167 and average cost as 1.367 (both normalized with respect to OPT). Like in the previous case, Switch is not as good as Robust, since it utilizes a *hard* switching between Robust and ML whenever a pre-defined threshold is reached without looking at the actual performance of Robust or ML. Due to the lack of robustness guarantees, Greedy is also worse than Robust in this setting. By varying $\lambda \geq 1$, we see that ERL-NT can have very large average costs (even larger than ML), although its competitive ratio is still guaranteed to be λ-competitive against the expert Robust. On the other hand, ERL, which is trained with $\lambda = 1.4$ and tested with different $\lambda \geq 1$ has a much lower average cost than ERL-NT, while also being able to guarantee competitive ratios. This shows the importance of being aware of expert robustification during the training stage. Moreover, ERL has a lower average cost than



(a) (b) (c) (d)

Fig. 3: Cost ratio probability distribution for September-December testing. The density values are obtained by dividing the number of testing instances within each bin by the total number of instances and the bin width. We use $\lambda = 1.4$ for testing ERL−NT and ERL. The horizontal axis is limit to 4 for better visualization, and cost ratios larger than 4 are off the charts.

	ERL−NT		ERL (λ = 1.4)		ERL (λ = 1.2)		Switch		Robust		Greedy	
λ for Testing	Avg	CR	Avg	CR	Avg	CR	Avg	CR	Avg	CR	Avg	CR
λ = 1.4	1.6977	6.0912	**1.3903**	6.0910	1.4343	6.0910	1.7454	6.4130	1.5336	5.000	1.5030	4.800
λ = 1.2	1.6457	5.5457	1.4832	5.5456	**1.4587**	5.5456	1.7454	6.4130	1.5336	5.000	1.5030	4.800

TABLE I: September-December testing. "Avg" and "CR" represent the empirical average cost and competitive ratio (normalized w.r.t. OPT), respectively. Bold texts mean the best AVG performance. ERL ($\lambda = x$) means we train ERL with $\lambda = x$.

ML: even in the presence of large training-testing distributional discrepancies, the expert can help correct many of the bad pre-robustification actions, thus significantly improving the average performance of ERL over ML. Interestingly, the average performance of ERL is not monotonic in the parameter of $\lambda \geq 1$ used for testing. This is partly because λ is different for training and testing, and partly because the large training-testing distributional discrepancies result in irregular average performance for the ML model used by ERL. By $\lambda = 1$, we essentially have no trust on the ML model in ERL, and hence ERL will follow the expert Robust at each step.

Summary. Our experiments highlight the key point that ERL guarantees worst-case robustness in terms of the competitive ratio by utilizing expert robustification, while exploiting the power of ML to improve the average performance. Naturally, when training-testing distributions are reasonably similar, we expect the average performance of ERL (and other ML-based optimizers like ML) to be better than that of Robust. But, even when the pure ML performs arbitrarily badly, ERL can still offer a good average cost performance due to the introduction of expert robustification. Last but not least, with explicit awareness of the expert robustification process, ERL has a much better average performance than otherwise (i.e., ERL−NT).

VII. Conclusion

In this paper, we propose ERL, a novel expert-robustified learning approach to solve online optimization with memory costs. For guaranteed robustness, ERL introduces a projection operator that robustifies ML actions by utilizing an expert online algorithm; for good average performance, ERL trains the ML optimizer based on a recurrent architecture by explicitly considering downstream expert robustification process. We prove that, for any $\lambda \geq 1$, ERL can achieve λ-competitive against the expert algorithm for any problem inputs. We also extend our analysis to a novel setting of multi-step memory costs. Finally, we run experiments for an energy scheduling application to validate ERL, showing that ERL can offer the best tradeoff in terms of the average and worst performance.

Acknowledgement

This work was supported in part by the NSF under grant CNS-1910208.

Appendix

In the more general case, the memory cost may span multiple steps (e.g. acceleration smoothness), which has not been well studied. We first show that Robust is still an competitive expert, by providing its competitive ratio in the multi-step memory setup in Appendix A. Then, in Appendix B, we prove that ERL is still λ-competitive against any expert, and this automatically proves Theorem 4.1 for the single-stem memory case.

A. Proof of Theorem 5.1

When $t \geq q$, Robust satisfies the following condition:

$$\begin{aligned}
& f(x_t^\pi, y_t) + ||x_t^\pi - \sum_{i=1}^{q} C_i x_{t-i}^\pi|| \\
\leq & f(x_t^\pi, y_t) + ||x_t^* - \sum_{i=1}^{q} C_i x_{t-i}^*|| + ||x_t^\pi - x_t^*|| \\
& + \sum_{i=1}^{q} ||C_i|| \cdot ||x_{t-i}^\pi - x_{t-i}^*|| \\
\leq & f(x_t^\pi, y_t) + ||x_t^* - \sum_{i=1}^{q} C_i x_{t-i}^*|| + \frac{1}{\alpha}\big(f(x_t^*, y_t) - f(x_t^\pi, y_t)\big) \\
& + \frac{1}{\alpha}\sum_{i=1}^{q} ||C_i||\big(f(x_{t-i}^*, y_{t-i}) - f(x_{t-i}^\pi, y_{t-i})\big).
\end{aligned}$$

The first and second inequalities come from the triangle inequality of l_p norm, and the third inequality comes from the α-polyhedral assumption of the hitting cost function. For

$t < q$, since $x_t^\pi = x_t^* = x_t, \forall t \in [-q+1, 0]$, the above inequality also holds. We sum up all the single-step costs:

$$
\begin{aligned}
&\sum_{t=1}^{T} f(x_t^\pi, y_t) + \left\| x_t^\pi - \sum_{i=1}^{q} C_i x_{t-i}^\pi \right\| \\
\leq &\sum_{t=1}^{T} f(x_t^\pi, y_t) + \sum_{t=1}^{T} \left\| x_t^* - \sum_{i=1}^{q} C_i x_{t-i}^* \right\| \\
&+ \frac{1}{\alpha} \sum_{t=1}^{T} \big(f(x_t^*, y_t) - f(x_t^\pi, y_t) \big) \\
&+ \frac{1}{\alpha} \sum_{i=1}^{q} \|C_i\| \sum_{t=1}^{T} \big(f(x_{t-i}^*, y_{t-i}) - f(x_{t-i}^\pi, y_{t-i}) \big) \\
\leq &\sum_{t=1}^{T} f(x_t^\pi, y_t) + \sum_{t=1}^{T} \left\| x_t^* - \sum_{i=1}^{q} C_i x_{t-i}^* \right\| \\
&+ \frac{1}{\alpha} \sum_{t=1}^{T} \big(f(x_t^*, y_t) - f(x_t^\pi, y_t) \big) \\
&+ \frac{1}{\alpha} \sum_{i=1}^{q} \|C_i\| \sum_{t=1}^{T} \big(f(x_t^*, y_t) - f(x_t^\pi, y_t) \big) \\
= &\sum_{t=1}^{T} f(x_t^\pi, y_t) + \sum_{t=1}^{T} \left\| x_t^* - \sum_{i=1}^{q} C_i x_{t-i}^* \right\| \\
&+ \frac{1}{\alpha}(1 + \sum_{i=1}^{q} \|C_i\|) \sum_{t=1}^{T} \big(f(x_t^*, y_t) - f(x_t^\pi, y_t) \big),
\end{aligned}
$$

where the second inequality holds because $x_t^\pi = x_t^* = x_t, \forall t \in [-q+1, 0]$ and $f(x_t^*, y_t) - f(x_t^\pi, y_t) \geq 0$. Thus, we have

$$
\begin{aligned}
&\sum_{t=1}^{T} f(x_t^\pi, y_t) + \left\| x_t^\pi - \sum_{i=1}^{q} C_i x_{t-i}^\pi \right\| \\
\leq &(1 - \frac{1+\beta}{\alpha}) \sum_{t=1}^{T} f(x_t^\pi, y_t) + \sum_{t=1}^{T} \left\| x_t^* - \sum_{i=1}^{q} C_i x_{t-i}^* \right\| \\
+ &\frac{1+\beta}{\alpha} \sum_{t=1}^{T} f(x_t^*, y_t)
\end{aligned}
\tag{6}
$$

If $\alpha \leq 1 + \beta$, then $1 - \frac{1+\beta}{\alpha} \leq 0$, the inequality (6) becomes

$$
\begin{aligned}
&\sum_{t=1}^{T} f(x_t^\pi, y_t) + \left\| x_t^\pi - \sum_{i=1}^{q} C_i x_{t-i}^\pi \right\| \\
\leq &\frac{1+\beta}{\alpha} \sum_{t=1}^{T} f(x_t^*, y_t) + \sum_{t=1}^{T} \left\| x_t^* - \sum_{i=1}^{q} C_i x_{t-i}^\pi \right\|.
\end{aligned}
$$

If $\alpha > 1 + \beta$, since $x_t^\pi = v_t$ minimizes $f(\cdot, y_t)$, then $f(x_t^\pi, y_t) \leq f(x_t^*, y_t)$ and, based on (6), `Robust` is optimal. This completes the proof.

B. Proof of Theorem 4.1 and Corollary 5.1

We denote the accumulated cost of the first t_1 steps as $\text{cost}(x_{1:t_1}) = \sum_{t=1}^{t_1} \big(f(x_t, y_t) + \tilde{d}(x_t, \sum_{i=1}^{q} C_i x_{t-i}) \big)$. When $t = 1$, x_1^π is clearly a feasible solution to (4). Then, suppose that for $t \geq 1$, $x_{1:t-1}$ satisfies the constraint, i.e. $\text{cost}(x_{1:t-1}) + G(x_{t-1}, x_{t-q-1:t-2}, x_{t-q-1:t-1}^\pi) - \big(\lambda \text{cost}(x_{1:t-1}^\pi) + B\big) \leq 0$. We need to prove that x_t^π is a feasible solution of the projection (4). By the constraint in the projection, we have

$$
\begin{aligned}
&\big(\text{cost}(x_{1:t-1}) + f(x_t^\pi, y_t) + \tilde{d}(x_t^\pi, \sum_{i=1}^{q} C_i x_{t-i})\big) \\
&+ G(x_t^\pi, x_{t-q:t-1}, x_{t-q:t}^\pi) - \big(\lambda \text{cost}(x_{1:t}^\pi) + B\big) \\
= &\text{cost}(x_{1:t-1}) - \big(\lambda \text{cost}(x_{1:t-1}^\pi) + B\big) \\
&+ G(x_t^\pi, x_{t-q:t-1}, x_{t-q:t}^\pi) \\
&+ \tilde{d}(x_t^\pi, \sum_{i=1}^{q} C_i x_{t-i}) - \tilde{d}(x_t^\pi, \sum_{i=1}^{q} C_i x_{t-i}^\pi).
\end{aligned}
$$

By the triangular inequality, we have $\tilde{d}(x_t^\pi, \sum_{i=1}^{q} C_i x_{t-i}) - \tilde{d}(x_t^\pi, \sum_{i=1}^{q} C_i x_{t-i}^\pi) \leq \tilde{d}(\sum_{i=1}^{q} C_i x_{t-i}, \sum_{i=1}^{q} C_i x_{t-i}^\pi)$ and

$$
\begin{aligned}
&\big(\text{cost}(x_{1:t-1}) + f(x_t^\pi, y_t) + \tilde{d}(x_t^\pi, \sum_{i=1}^{q} C_i x_{t-i})\big) \\
&+ G(x_t^\pi, x_{t-q:t-1}, x_{t-q:t}^\pi) - \big(\lambda \text{cost}(x_{1:t}^\pi) + B\big) \\
\leq &\text{cost}(x_{1:t-1}) - \big(\lambda \text{cost}(x_{1:t-1}^\pi) + B\big) \\
&+ G(x_t^\pi, x_{t-q:t-1}, x_{t-q:t}^\pi) \\
&+ \tilde{d}(\sum_{i=1}^{q} C_i x_{t-i}, \sum_{i=1}^{q} C_i x_{t-i}^\pi) \\
= &\text{cost}(x_{1:t-1}) - \big(\lambda \text{cost}(x_{1:t-1}^\pi) + B\big) \\
&+ \sum_{k=1}^{q} \tilde{d}(\sum_{i=1}^{q-k} C_{k+i} x_{t-i}, \sum_{i=1}^{q-k} C_{k+i} x_{t-i}^\pi) \\
&+ \tilde{d}(\sum_{i=1}^{q} C_i x_{t-i}, \sum_{i=1}^{q} C_i x_{t-i}^\pi) \\
= &\text{cost}(x_{1:t-1}) + G(x_{t-1}, x_{t-q-1:t-2}, x_{t-q-1:t-1}^\pi) \\
&- \big(\lambda \text{cost}(x_{1:t-1}^\pi) + B\big) \leq 0,
\end{aligned}
\tag{7}
$$

where the first inequality is because of the triangular inequality of l_p norm, the first equality is from

$$
\begin{aligned}
&G(x_t^\pi, x_{t-q:t-1}, x_{t-q:t}^\pi) \\
= &\sum_{k=1}^{q} \tilde{d}(\sum_{i=1}^{q-k} C_{k+i} x_{t-i}, \sum_{i=1}^{q-k} C_{k+i} x_{t-i}^\pi),
\end{aligned}
\tag{8}
$$

and the second equality is from

$$
\begin{aligned}
&\sum_{k=1}^{q} \tilde{d}(\sum_{i=1}^{q-k} C_{k+i} x_{t-i}, \sum_{i=1}^{q-k} C_{k+i} x_{t-i}^\pi) \\
&+ \tilde{d}(\sum_{i=1}^{q} C_i x_{t-i}, \sum_{i=1}^{q} C_i x_{t-i}^\pi) \\
= &\sum_{k=0}^{q} \tilde{d}(\sum_{i=1}^{q-k} C_{k+i} x_{t-i}, \sum_{i=1}^{q-k} C_{k+i} x_{t-i}^\pi) \\
= &G(x_{t-1}, x_{t-q-1:t-2}, x_{t-q-1:t-1}^\pi)
\end{aligned}
\tag{9}
$$

Thus, Theorem 4.1 is proved by setting $q = 1$. By Theorem 5.1, we also prove Corollary 5.1.

References

[1] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[2] Mohammad Ali Alomrani, Reza Moravej, and Elias B. Khalil. Deep policies for online bipartite matching: A reinforcement learning approach. *CoRR*, abs/2109.10380, 2021.

[3] Brandon Amos. Tutorial on amortized optimization for learning to optimize over continuous domains. *CoRR*, abs/2202.00665, 2022.

[4] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J. Zico Kolter. Differentiable MPC for end-to-end planning and control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[5] Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. A regression approach to learning-augmented online algorithms. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[6] Antonios Antoniadis, Christian Coester, Marek Eliáš, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *ICML*, pages 345–355, 2020.

[7] Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *SIGACT News*, 47(3):93–129, August 2016.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[9] Niangjun Chen, Joshua Comden, Zhenhua Liu, Anshul Gandhi, and Adam Wierman. Using predictions in online optimization: Looking forward with an eye on the past. *SIGMETRICS Perform. Eval. Rev.*, 44(1):193–206, June 2016.

[10] Niangjun Chen, Gautam Goel, and Adam Wierman. Smoothed online convex optimization in high dimensions via online balanced descent. In *COLT*, 2018.

[11] Jakub Chłędowski, Adam Polak, Bartosz Szabucki, and Konrad Tomasz Żołna. Robust learning-augmented caching: An experimental study. In *ICML*, 2021.

[12] Nicolas Christianson, Tinashe Handina, and Adam Wierman. Chasing convex bodies and functions with black-box advice. In *COLT*, 2022.

[13] Joshua Comden, Sijie Yao, Niangjun Chen, Haipeng Xing, and Zhenhua Liu. Online optimization in cloud resource provisioning: Predictions, regrets, and algorithms. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(1), March 2019.

[14] Bingqian Du, Zhiyi Huang, and Chuan Wu. Adversarial deep learning for online resource allocation. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 6(4), feb 2022.

[15] Bingqian Du, Chuan Wu, and Zhiyi Huang. Learning resource allocation and pricing for cloud profit maximization. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.

[16] Elbert Du, Franklyn Wang, and Michael Mitzenmacher. Putting the "learning" into learning-augmented algorithms for frequency estimation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2860–2869. PMLR, 18–24 Jul 2021.

[17] Gautam Goel, Yiheng Lin, Haoyuan Sun, and Adam Wierman. Beyond online balanced descent: An optimal algorithm for smoothed online optimization. In *NeurIPS*, volume 32, 2019.

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

[19] Elad Hazan. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

[20] Howard Heaton, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. Safeguarded learned convex optimization, 2020.

[21] Weiwei Kong, Christopher Liaw, Aranyak Mehta, and D. Sivakumar. A new dog learns old tricks: RL finds classic optimization algorithms. In *ICLR*, 2019.

[22] Ke Li and Jitendra Malik. Learning to optimize. In *ICLR*, 2017.

[23] Pengfei Li, Jianyi Yang, and Shaolei Ren. Expert-calibrated learning for online optimization with switching costs. In *SIGMETRICS*, 2022.

[24] Yingying Li, Xin Chen, and Na Li. *Online Optimal Control with Linear Dynamics and Predictions: Algorithms and Regret Analysis*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[25] Yingying Li and Na Li. Leveraging predictions in smoothed online convex optimization via gradient-based algorithms. In *NeurIPS*, volume 33, 2020.

[26] Yingying Li, Guannan Qu, and Na Li. Online optimization with predictions and switching costs: Fast algorithms and the fundamental limit. *IEEE Transactions on Automatic Control*, 2020.

[27] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. In *INFOCOM*, 2011.

[28] Yiheng Lin, Gautam Goel, and Adam Wierman. Online optimization with predictions and non-convex losses. *Proc. ACM Meas. Anal. Comput. Syst.*, 4(1), May 2020.

[29] Heyuan Liu and Paul Grigas. Risk bounds and calibration for a smart predict-then-optimize method. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[30] Yecheng Jason Ma, Dinesh Jayaraman, and Osbert Bastani. Conservative offline distributional reinforcement learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[31] Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8 (4):265–368, 2013.

[32] Daan Rutten, Nico Christianson, Debankur Mukherjee, and Adam Wierman. Online optimization with untrusted predictions. *CoRR*, abs/2202.03519, 2022.

[33] Asis Sarkar and Dhiren Kumar Behera. Wind turbine blade efficiency and power calculation with electrical analogy. *International Journal of Scientific and Research Publications*, 2(2):1–5, 2012.

[34] Manajit Sengupta, Yu Xie, Anthony Lopez, Aron Habte, Galen Maclaurin, and James Shelby. The national solar radiation data base (nsrdb). *Renewable and Sustainable Energy Reviews*, 89:51–60, 2018.

[35] Guanya Shi, Yiheng Lin, Soon-Jo Chung, Yisong Yue, and Adam Wierman. Online optimization with memory and competitive control. In *NeurIPS*, volume 33. Curran Associates, Inc., 2020.

[36] Matthew Staib and Stefanie Jegelka. Distributionally robust optimization and generalization in kernel methods. *Advances in Neural Information Processing Systems*, 32:9134–9144, 2019.

[37] Can Wan, Jian Zhao, Yonghua Song, Zhao Xu, Jin Lin, and Zechun Hu. Photovoltaic and solar power forecasting for smart grid energy management. *CSEE Journal of Power and Energy Systems*, 1(4):38–46, 2015.

[38] Kai Wang, Sanket Shah, Haipeng Chen, Andrew Perrault, Finale Doshi-Velez, and Milind Tambe. Learning MDPs from features: Predict-then-optimize for sequential decision making by reinforcement learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[39] Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J. Ramadge. Projection-based constrained policy optimization. In *International Conference on Learning Representations*, 2020.

[40] Yunchang Yang, Tianhao Wu, Han Zhong, Evrard Garcelon, Matteo Pirotta, Alessandro Lazaric, Liwei Wang, and Simon Shaolei Du. A reduction-based framework for conservative bandits and reinforcement learning. In *International Conference on Learning Representations*, 2022.

[41] Jingzhao Zhang, Aditya Krishna Menon, Andreas Veit, Srinadh Bhojanapalli, Sanjiv Kumar, and Suvrit Sra. Coping with label shift via distributionally robust optimisation. In *International Conference on Learning Representations*, 2021.

[42] Lijun Zhang, Wei Jiang, Shiyin Lu, and Tianbao Yang. Revisiting smoothed online learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[43] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pages 928–936, 2003.