Learning Rewards to Optimize Global Performance Metrics in Deep Reinforcement Learning

Junqi Qian
Shanghai Jiao Tong University
Shanghai, China
kyle1994@sjtu.edu.cn

Paul Weng
Shanghai Jiao Tong University
Shanghai, China
paul.weng@sjtu.edu.cn

Chenmien Tan
University of Nottingham Ningbo
China
Ningbo, China
chenmientan@outlook.com

ABSTRACT

When applying reinforcement learning (RL) to a new problem, reward engineering is a necessary, but often difficult and error-prone task a system designer has to face. To avoid this step, we propose LR4GPM, a novel (deep) RL method that can optimize a global performance metric, which is supposed to be available as part of the problem description. LR4GPM alternates between two phases: (1) learning a (possibly vector) reward function used to fit the performance metric, and (2) training a policy to optimize an approximation of this performance metric based on the learned rewards. Such RL training is not straightforward since both the reward function and the policy are trained using non-stationary data. To overcome this issue, we propose several training tricks. We demonstrate the efficiency of LR4GPM on several domains. Notably, LR4GPM outperforms the winner of a recent autonomous driving competition organized at DAI'2020.

KEYWORDS

Deep reinforcement learning; Reward learning; Non-linear performance metric

1 INTRODUCTION

Reinforcement learning (RL) is a versatile machine learning technique to learn how to make good decisions by trial and error. Although it has achieved excellent results in various domains [18, 40, 51], its application to real-world problems is still difficult due to various limitations of current RL techniques, e.g., large sample requirements, robustness of trained policies, or generalization issues. In this paper, we focus on two specific related limitations, difficulty of reward engineering and insufficiency of the reward hypothesis, which we discuss next.

Reward engineering is one of the first steps a system designer has to go through when trying to solve a new control problem using RL. It consists in designing a reward function, which serves two roles: (1) to guide the RL agent during training and (2) to define what is the correct behavior expected from a trained RL agent.

While sparse reward are easy to specify, learning from them may be unacceptably slow, especially on long horizon problems [15]. In contrast, dense rewards can accelerate RL training, but they may lead to undesirable behaviors [6, 32].

The reward hypothesis [46] roughly states that intelligence may be formulated as the maximization of an expected accumulated rewards. Although some argued for this conjecture [41], recent work [1] shows some specific definition of tasks may not fit this hypothesis. Actually, given the axiomatization of expected utility [27], it is well-known in decision theory that some ordering over policies cannot be represented by it, e.g., when dealing with risk-sensitive decision-making [5].

In this paper, we propose an RL method that works without the need of designing a reward function, but assumes instead that an overall performance metric is given to measure the quality of a policy. This situation is actually common in practice. For instance, in data center control, one may aim at optimizing power usage effectiveness [19]. In network control, a classic objective function would combine both throughput and delay [43]. In finance, one may consider the Sharpe ratio [38], a performance metric that aggregates performance and risk. In online advertising, one may focus on metrics like cost per action [29].

As a first step in this direction, we assume that the form of the performance metric is known in this work. We propose to automatically learn a reward function such that an approximation of the performance metric can be optimized via the learned rewards by the RL agent. The learned reward function may be vectorial to take into account the fact that the performance metric may be an aggregation of different objectives.

Main Contributions. We propose a novel (deep) RL method that can directly optimize a given global performance metric without the need of a reward function. We experimentally demonstrate its excellent performance over various domains (Mujoco, Iroko [35], SMARTS [55]) in various settings and various difficulty levels (standard RL performance metric to non-linear ones). Notably, in the SMARTS simulator that was used in a competition[1], our method performs better than the winner and runner-up's submissions. Moreover, our experiments also provide some empirical evidence of the insufficiency of the reward hypothesis.

The remaining of the paper is organized as follows. Section 2 discusses the most related work. Section 3 recalls the necessary background and notations. In Section 4, we formally state the problem tackled in this paper and in Section 5, we detail our proposed

---

[1]Organized during the international conference on distributed artificial intelligence (DAI) in 2020

method, LR4GPM, discuss the difficulties of our problem, and motivates the design of LR4GPM. Section 6 presents the experimental results that validate our approach. Finally, we conclude in Section 7.

2 RELATED WORK

The design of the reward function, which is related to the issue of value alignment [8], has been recognized as a difficult problem in the RL literature, as attested by the numerous publications in inverse RL [9, 36] and behavior cloning [7, 49]. In a related direction, reward shaping [30], some work [16, 45] aims at learning a potential function to complement an existing sparse reward function. Although various extensions [14, 22, 25] of theses techniques have been investigated, most work assumes that (often near expert) demonstrations are available or that an oracle is present to provide preferential feedback, which is not suitable in many domains. In this paper, we do not make these assumptions.

Closer to our paper, some work proposes to learn a reward function, often online while training a policy. Singh et al. [42] formulated the *optimal reward design* problem where the goal is to learn a reward function that an agent would learn from such that it can maximize effectively the expected discounted reward computed with the system designer's reward function. This setting is notably useful when the agent has some limitations (e.g., expressivity of parametric policy). In this setting, Sorg et al. [44] show that it may be worthwhile to learn a reward function, even if the designer's reward function is known, due notably to limitations of the policy representation. Zheng et al. [54] learn intrinsic rewards for policy gradient methods. In a related work, Agarwal et al. [3] learn a reward function from sparse or underspecified rewards. The multi-agent extension of optimal reward design has also been studied [21, 26]. Our work can be interpreted as a generalization of optimal reward design [42] to more general performance metrics. As a first step in this direction, we only consider the single-agent case in this paper. We leave the extension of our approach to the multi-agent setting for future work.

Unlike all the previous methods that aim at maximizing the standard RL criterion, our approach can also deal with non-linear objective functions. Thus, another related research direction regards extensions of RL to various considerations, e.g., risk-adversity [13, 48], fairness [39, 56], multi-objective settings [11, 33, 34], constraints [2, 12], or safety [20, 53]. In all these studies, the reward function is assumed to be given in contrast to our work.

3 BACKGROUND

In this section, we recall the necessary background and notations about Markov decision process and (deep) reinforcement learning.

3.1 Markov Decision Process

A *Markov Decision Process* (MDP) is a model for representing sequential decision-making problems under uncertainty. In such model, defined as a tuple $(\mathcal{S}, \mathcal{A}, P, d_0, r)$, an agent repeatedly chooses some action a from an action space $\mathcal{A}$ after observing a state s from a state space $\mathcal{S}$. The agent obtains an immediate reward $r(s, a)$ after performing a in s. The transition function P defines the probability distribution $P(s' \mid s, a)$ over next states and d_0 is the initial state distribution. The goal in this model is to determine how to select

actions in states such as some performance metric is maximized. We explain the details of the standard setting next.

The action selection is carried out via a *(stochastic) policy* π : $\mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, which is a mapping from state-action pairs to probabilities. For convenience, we write $\pi(s, a)$ as $\pi(a \mid s)$, which specifies the probability of choosing an action a in a state s. When the probabilities of π are either 0 or 1, it is said to be *deterministic*.

We denote by $\eta = (s_0, a_0, \cdots, s_{T-1}, a_{T-1}, s_T)$ a T-step *trajectory*, and by $\mathbb{H} = \bigcup_n \mathcal{S} \times (\mathcal{A} \times \mathcal{S})^n$ the set of all trajectories. Under the standard RL criterion, the *return* $v(\eta)$ of trajectory η is defined by:

$$v(\eta) = \sum_t \gamma^t r(s_t, a_t) \tag{1}$$

where $\gamma \in [0, 1)$ is a discounted factor.

The *(state) value function* V_π of a policy π from an initial state s is defined by:

$$V_\pi(s) = \mathbb{E}_{\pi, P}[v(\eta) \mid s_0 = s] \tag{2}$$

Similarly, the *action value function* $Q_\pi(s, a)$ is the expected return for applying from state s and action a followed by policy π:

$$Q_\pi(s, a) = \mathbb{E}_{\pi, P}[v(\eta) \mid s_0 = s, a_0 = a] \tag{3}$$

The usual goal in an MDP is to find an (optimal) policy π^* that maximizes the *expected discounted total reward* criterion:

$$\mathbb{E}_{d_0}[V_\pi(s)] \tag{4}$$

where the expectation is with respect to $s \sim d_0$. The value function of an optimal policy is said to be optimal. In the RL setting, this goal is achieved without assuming the knowledge of the transition function or reward function. In this paper, we refer to the expected discounted total reward criterion as the standard RL criterion.

3.2 Standard Deep RL Methods

When the state space or action space becomes too large or continuous, parametric function approximation (e.g., neural network or linear function) is needed to allow generalization. In RL, the state (or action) value function, or the policy can be approximated by V_ϕ (or Q_ϕ), or π_θ where ϕ or θ denotes the parameters to be learned.

In value-based model-free RL, the optimal action value function is usually approximated. For instance, in DQN [28], the parameters ϕ are learned by iteratively minimizing the squared difference between a regression target and the current approximation Q_ϕ:

$$\mathbb{E}\left(r + \gamma \max_{a'} Q_{\phi^-}(s', a') - Q_\phi(s, a)\right)^2 \tag{5}$$

where the expectation is with respect to the random sampling of tuples (s, a, s', r), composed of a state, an action, a next state, and a reward respectively, from a replay buffer, and ϕ^- is a stabler copy of ϕ. The replay buffer stores the tuples (s, a, s', r) experienced by the RL agent as it interacts with its environment.

In contrast to value-based methods, policy-based methods search for a good policy in a space of parametrized policies π_θ. Sutton et al. [47] proposes that parameters θ should be updated in the direction of the following gradient $A_\pi(s, a)\nabla_\theta \log \pi_\theta(a \mid s)$ where $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$ is the *advantage function*, which can be estimated in various manners. In REINFORCE [52], the advantage function is estimated by the return from which a baseline is subtracted: $\hat{A}(s_t, a_t) = v_t - b_t(s_t)$, where v_t is the return at time step t and

Table 1: Instances of our general framework

Framework	ρ	$\nu(\eta)$
Standard RL	identity fun.	$\sum_t \gamma^t r(s_t, a_t)$
Fairness [39]	fair function	$\sum_t \gamma^t r(s_t, a_t)$
Risk-sensitivity [17]	$\frac{1}{\beta} \log$	$\exp(\beta \sum_t \gamma^t r(s_t, a_t))$
General [11]	concave fun.	$\sum_t \gamma^t r(s_t, a_t)$
General [34]	identity fun.	$u(\sum_t \gamma^t r(s_t, a_t))$

$b_t(s_t)$ is usually an estimated value function. In Proximal Policy Optimization (PPO) [37], the advantage of π_θ, approximated by $\hat{A}_\theta$, is estimated with λ-returns and a learned value function V_ϕ. PPO updates the parameters of π_θ in order to maximize the *clipped surrogate objective function* defined as:

$$J_{PPO}(\theta) = \mathbb{E}[\min(\omega_t(\theta)\hat{A}_{\bar{\theta}}(s_t, a_t), \text{clip}_\epsilon(\omega_t(\theta))\hat{A}_{\bar{\theta}}(s_t, a_t))] \quad (6)$$

where $\omega_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\bar{\theta}}(a_t|s_t)}$, $\bar{\theta}$ corresponds to the parameters of the policy that generated the training data, and $\text{clip}_\epsilon(x)$ clips x in $[1 - \epsilon, 1 + \epsilon]$. This objective function is defined as such to prevent large policy changes.

3.3 Vector Rewards

In our proposed method, we allow the learned reward that is given to the RL agent after performing action a in state s to be a vector $\boldsymbol{r}(s, a) \in \mathbb{R}^D$ (with $D \geq 1$) instead of the usual scalar value $r(s, a)$. The return defined in Equation (1), the (state) value function defined Equation (2), and the action value function Equation (3) can naturally be extended to the vector setting. To make this clear, the vector counterparts are denoted in bold, $\boldsymbol{v}$, $\boldsymbol{V}$, and $\boldsymbol{Q}$. Note that in contrast to multi-objective RL, we do not consider vector maximization.

4 PROBLEM STATEMENT

In this paper, we aim at learning a policy that maximizes a given performance metric, assumed to be of the following form:

$$\max_\pi \rho(\mathbb{E}_{\pi, \boldsymbol{P}}[\boldsymbol{v}(\eta)]) \quad (7)$$

where π is a policy, $\rho : \mathbb{R}^D \to \mathbb{R}$ is an aggregation function, $\mathbb{E}_{\pi, \boldsymbol{P}}$ is an expectation with respect to the distribution over trajectories η induced by policy π, and $\boldsymbol{v} : \mathbb{H} \to \mathbb{R}^D$ is a trajectory evaluation function. The interpretation of the overall objective function is simple: histories are first evaluated via function $\boldsymbol{v}$ possibly over several dimensions if $D > 1$, then the expected evaluations are aggregated into one score via function ρ. We refer to this overall objective function as *(global) performance metric*.

Note that in Equation (7), $\boldsymbol{v}(\eta)$ does not have to decompose additively like in the standard RL setting. Furthermore, we assume that ρ is differentiable, but $\boldsymbol{v}$ does not have to be. This provides more flexibility in formulating how to evaluate a trajectory. For instance, in the simplest case, the evaluation can be Boolean indicating whether a trajectory was successful or not. However, since the evaluation applies to the whole trajectory, it can integrate diverse information along the trajectory to form an overall assessment.

This framework is very general. For instance, it can formalize some notions of fairness [39], or risk-sensitivity [17]. It also includes the general multi-objective framework with concave objective functions proposed by [11] or that based on Expected Utility in [34]. We summarize in Table 1 how ρ and $\boldsymbol{v}$ are defined in all these cases. In that table, $u : \mathbb{R}^D \to \mathbb{R}$ refers to a utility function. The interested readers can obtain more details in the respective references. Note that our general framework covers many other cases. Two other examples (networking and autonomous driving) are introduced in Section 6 for our experimental validation.

Problem (7) is not straightforward to solve via standard RL methods, since no reward function is readily available and function ρ may be non-linear. In the next section, we explain our proposed approach.

5 PROPOSED METHOD

We present first the general structure of our algorithm and its motivation. Then, we explain why we include several training tricks. We finish with a discussion about some possible alternative approaches.

5.1 Basic Algorithmic Scheme

Facing Problem (7), one may think of different straightforward approaches. The simplest one consists in resorting to a REINFORCE-like algorithm. However, such method is not sample efficient, and its performance is subpar in our experiments. This observation suggests that learning an approximation of the performance metric is needed for both sample efficiency and performance.

Therefore, we propose to solve Problem (7) by alternating between two phases: (i) approximating its objective function (i.e., performance metric) and (ii) learning a policy. Intuitively, a good approximation of the performance metric is obtained to guide policy training, which can then be used to generate training data for improving the approximation of the performance metric around the current policy. This process can be repeated until convergence to a hopefully good policy.

Apart from our method LR4GPM, we also discuss at the end of this section several natural approaches to approximate the performance metric. However, we have not managed to obtain very promising performances with any of them. We report some of those results in Section 6, since they are used as baselines. Below, we provide a detailed description of our approach.

Assuming that π is a fixed policy, phase (i) corresponds to this first problem:

$$\min_{\xi, \psi} (\rho(\mathbb{E}_{\pi, \boldsymbol{P}}[\boldsymbol{v}(\eta)]) - \hat{\rho}_\xi(\mathbb{E}_{\pi, \boldsymbol{P}}[\boldsymbol{r}_\psi(\eta)]))^2 \quad (8)$$

where $\hat{\rho}_\xi$ is a parametric function (i.e., polynomial parametrized by ξ) and $\boldsymbol{r}_\psi(\eta)$ is the return of trajectory η computed using a reward network $\boldsymbol{r}_\psi$ (i.e., neural network parametrized by ψ).

Learning a reward network $\boldsymbol{r}_\psi$ enables the usual temporal decomposition of the value of a trajectory, which helps learn a critic. Moreover, it may allow to obtain a denser reward feedback that can better guide the RL agent during training. Instead of using ρ directly, an approximate $\hat{\rho}_\xi$ is learned to account for the approximation introduced by the reward network. Moreover, learning both $\hat{\rho}_\xi$ and $\boldsymbol{r}_\psi$ can also be motivated and understood from the point of view of optimal reward design [42].

If parameters ξ and ψ are updated to solve Problem (8), they would then form a good approximation of the objective function

---

Algorithm 1 LR4GPM

Require: ρ non-linear performance metric, N number of iterations
   of the outer loop, m number of policies, n_B size of a minibatch,
   K number of the extra sets, n_η number of trajectories in a set,
   n_H number of timesteps, n_E number of trajectories.
   Randomly initialize parameters ξ (performance metric), θ (policy),
   ϕ (critic), and ψ (reward)
   Initialize ψ^- (target network) as copy of ψ
   Initialize replay buffer $\mathcal{B}$ of trajectories with trajectories gener-
   ated by a uniformly-random policy
   for $i = 1, \ldots, N$ **do**
      Sample $n_B + K$ sets of n_η trajectories from $\mathcal{B}$
      Keep the top n_B sets of trajectories w.r.t. ρ
      Update ξ and ψ using these n_B sets of traj. to minimize L_2 loss
      (10)
      for $j = 1, \ldots, n_H$ **do**
         Generate a set E of trajectories with π_θ (the rewards are the
         weighted sum of values obtained with ψ and ψ^-)
         Update θ (actor) with E to optimize the extended J_{PPO} (11)
         Update ϕ (critic) with the usual L_2 loss (13)
      end for
      Generate m sets of trajectories $E_1, \ldots, E_m$ with π_θ
      Update θ (actor) to get θ_1 to θ_m with each of these $E_1, \ldots, E_m$
      to minimize the extended J_{PPO} (11)
      Generate set E'_k of trajectories with π_{θ_k} for $k \in \{1, 2, \ldots, m\}$
      Evaluate each π_{θ_k} with respect to ρ using E'_k
      $\theta \leftarrow \theta_k$ for the highest evaluated policy π_{θ_k}
      Generate n_E trajectories with θ and add them in $\mathcal{B}$
   end for

---

of Problem (7) around trajectories generated by policy π. However,
since policy π may be far from optimal, there is no reason to reach
the best approximation possible of the performance metric around
it. Therefore, ξ and ψ only need to be updated to improve the
approximation so that they can be used to improve policy π.

Assuming now that ξ and ψ are fixed, phase (ii) corresponds to
this second problem:

$$\max_\pi \hat{\rho}_\xi(\mathbb{E}_{\pi, P}[r_\psi(\eta)]) . \qquad (9)$$

We assume this optimization problem to be on a space of parametrized
policies π_θ. If we remove $\hat{\rho}_\xi$ in Problem (9) and $D = 1$, it would be a
standard RL problem. With $\hat{\rho}_\xi$, the problem is actually structurally
similar to some recent extensions of RL to non-linear objective
functions (e.g., [11, 39]).

Again, if we were to update θ to solve Problem (9), we may
obtain a good policy for the current reward function r_ψ (in com-
bination with $\hat{\rho}_\xi$). However, since ξ and ψ may not form a good
approximation of the performance metric around a good policy for
the original problem (7), it is not necessary to solve Problem (9)
completely. Therefore, θ only needs to be updated so that it can help
improve the approximation produced by ξ and ψ. In the remaining,
we drop subscript ψ to avoid cluttering the notations since there is
no risk of confusion because in our framework, the RL agent does
not receive any other types of rewards.

More precisely, our algorithm (see Algorithm 1), called *Learning
Rewards for Global Performance Metric* (LR4GPM), consists of two

nested loops. In the outer loop, function $\hat{\rho}_\xi$ and the reward network
r are trained such that the following loss $\mathcal{L}(\psi, \mathbb{B})$ is reduced:

$$\frac{1}{n_{\mathbb{B}}} \sum_{k=1}^{n_{\mathbb{B}}} \left(\rho\left(\frac{1}{|B_k|} \sum_{\eta \in B_k} \nu(\eta)\right) - \hat{\rho}_\xi\left(\frac{1}{|B_k|} \sum_{\eta \in B_k} r(\eta)\right) \right)^2 \qquad (10)$$

where $\mathbb{B} = \{B_1, \ldots, B_{n_{\mathbb{B}}}\}$ is a mini-batch of set of histories sampled
from a replay buffer $\mathcal{B}$. To make the training of the reward network
ν and the learning of $\hat{\rho}_\xi$ sample efficient, replay buffer $\mathcal{B}$ is filled
with trajectories generated by the current policy. This loss approxi-
mates the objective function of (8) via empirical means. In addition,
we perform an extra average to reduce variance.

In the inner loop, an RL agent is trained over a fixed number
of steps to solve Problem (9) assuming that $\hat{\rho}_\xi$ and r are fixed. This
problem, which is a variation of standard RL, can be solved
efficiently [39]. For concreteness, we instantiate our proposition
with PPO, but other RL algorithms would be possible.

Therefore, following the design of the PPO algorithm, the actor
update is formulated such as it maximizes the following extended
clipped surrogate objective function $J_{PPO}(\theta, E)$ defined as:

$$\frac{1}{H \cdot |E|} \sum_{\eta \in E} \sum_{t=1}^{H} \min\left(\omega_t(\theta)\hat{A}_{\bar{\theta}}(s_t, a_t), \text{clip}_\epsilon(\omega_t(\theta))\hat{A}_{\bar{\theta}}(s_t, a_t)\right)$$

$$(11)$$

where E is a batch of trajectories sampled by the current policy $\pi_{\bar{\theta}}$,
H is the length of the trajectories, $\omega_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\bar{\theta}}(a_t|s_t)}$, $\text{clip}_\epsilon(x)$ clips
x in $[1 - \epsilon, 1 + \epsilon]$ and:

$$\hat{A}_{\bar{\theta}}(s_t, a_t) = \hat{\rho}_\xi\left(\frac{1}{|E|} \sum_{\eta \in E} \sum_{k=t}^{H} \gamma^{k-t} r_k^\eta\right) - \hat{\rho}_\xi\left(V_\phi(s_t)\right) \qquad (12)$$

corresponds to a generalized advantage function estimated for pol-
icy $\pi_{\bar{\theta}}$ that generated the trajectories in E. Note that the two terms
in the advantage are transformed by $\hat{\rho}_\xi$. Moreover, V_ϕ is the critic
learned in our extended PPO. It is trained using a standard L_2 loss:

$$\frac{1}{H \cdot |E|} \sum_{\eta \in E} \sum_{t=1}^{H} \left(\sum_{k=t}^{H} \gamma^{k-t} r_k^\eta - V_\phi(s_t, a_t)\right)^2 \qquad (13)$$

where all the operations are performed componentwisely. Recall
both the return r and the state value function V_ϕ take values in $\mathbb{R}^D$.
Therefore, the sum of the components of this critic loss is actually
minimized at the end.

Given Equation (11), the policy gradient $\nabla_\theta J(\pi_\theta, E)$ to update
policy π_θ can be obtained easily via automatic differentiation when
π_θ is represented as a neural network. Note that the policy gradient
used in our work is computed differently from that proposed by
Siddique et al. [39]. In their work, the non-linear aggregator (ρ
in our work) is applied on a clipped vector surrogate objective
function, which is computed from a vector advantage function
obtained from the difference of a vector Q-function and a vector
value function. While this is justified in fair optimization, such
approach would not work in our setting, since applying ρ on such
advantages would not make sense.

We have now finished describing the basic algorithmic structure
of LR4GPM, which consists in repeatedly improving the approxi-
mation of the objective function in (7) and improving the current

policy. Note that both in the inner and outer loops, the parameters are updated several times using different randomly-sampled mini-batches, as it is common practice. Yet, this basic scheme is not sufficient, because of several difficulties, which we discuss next.

5.2 Difficulties

First of all, the environment for the policy learning is non-stationary since the reward function is updated dynamically in the outer loop. To alleviate the effects of the non-stationarity, we propose the following tricks that help stabilize the agent training process.

Target reward network To avoid the issues of sudden and large reward modifications in the policy learning, we keep a target reward network like in DQN [28]. The online reward network is copied at regular interval to the target one, which is used to train the RL agent.

Adaptive learning rates Since the reward network keeps changing, the policy is trained in a non-stationary environment. The usual approach in that case is to resort to constant learning rates. Moreover, following theoretical analysis [23] of actor-critic-like algorithms, we enforce the following constraints on the learning rates: the reward learning rate ≥ the critic learning rate ≥ the actor learning rate.

Without careful generation of the training data for the outer loop, the whole process can converge to a local optimum. To alleviate this issue, we propose the following tricks:

Optimistic policy update For the last policy update in one inner loop iteration, we sample m sets of trajectories $E_1, \ldots, E_m$ and update the current policy with each set to obtain m new policies. Each of those policies is evaluated with respect to the performance metric by sampling from it a new set of trajectories. Only the best one is kept and a new set of trajectories, generated from it, is added to the replay buffer $\mathcal{B}$. This favors the current policy to be updated towards better policies and biases the data generation to include better trajectories.

Optimistic sampling For approximating the performance metric, $n_B + K$ sets of mini-batches of trajectories are sampled and then evaluated with respect to the performance metric. Only the best n_B mini-batches are kept and used for the update of the reward network. Here, K is a hyperparameter controlling the optimism level. This biases the mini-batch sampling to include better trajectories and consequently improves the approximation around better trajectories.

Reward diversity To differentiate the components of the vector reward network, we measure the distance/divergence over all pairs of returns $\nu(\eta)$ and maximize the sum of distance/divergences in the update of reward network. We experimented with several possibilities, e.g., squared L_2, KL-divergence, JS-divergence, and Wasserstein distance (Recall that we can always normalize vector returns and treat them as probability distributions). We choose the squared L_2-distance because of its simplicity and its performance (see Section 6). Thus, the following additional loss, when averaged over several trajectories η, can then be combined with the loss in (10):

$$\lambda_{RD} \sum_{i,j} \|\nu_i(\eta) - \nu_j(\eta)\|_2^2 \tag{14}$$

where λ_{RD} is a hyperparameter controlling how much weight to put on this loss that promotes reward diversity.

Using these tricks in combination, LR4GPM can solve Problem (7) effectively and achieve excellent performance.

5.3 Alternative Approaches

In this subsection, we list a few alternative straightforward methods one could think of for approximating the performance metric:

- Following the reward hypothesis, one may aim at learning one scalar reward function to approximate the performance metric as an expected discounted reward.
- Instead of learning a reward function, a Q-network could directly be trained to approximate the performance metric. However, this requires the strong assumption that the environment can be reset in any state to train the Q-network in various states, since the performance metric in (7) needs to be computed as an average of many returns.
- If the performance metric is an aggregation of several components, a vector reward function can be learned such that each of its components approximates one of those components (again with an expected discounted reward).

Such approximated performance metric can then be used to train a good policy in a similar scheme as in LR4GPM. However, these simple approaches either work poorly or are worse than LR4GPM in our experiments, which show the difficulty of Problem (7).

6 EXPERIMENTAL RESULTS

We carried out several sets of experiments to validate our method, LR4GPM, against several baselines in three different domains: simple Mujoco control environments (Section 6.1), a network control task (Section 6.2), and an autonomous driving task (Section 6.3). We have different global performance metrics for different domains, from linear performance metric to non-linear performance metric that cannot be decomposed into direct sum of rewards. We evaluated the policies obtained from LR4GPM and baseline methods with respect to the global performance metric. The baseline methods applied in each domain are listed in their respective sections.

Hyperparameter optimization is performed in each domain for our method and the baseline methods, and each hyperparameter configuration is evaluated over 5 random seeds during the hyperparameter optimization. The hyperparameter configuration corresponding to the best agent performance would then be rerun with 20 new random seeds to compute an average result. Both during training and testing, a policy is evaluated with an empirical mean over 5 sets of trajectories, whose size is n_η (see Algorithm 1). For experiments in Sections 6.1 and 6.2, simple grid search is applied for hyperparameter optimization. For experiments in Section 6.3, Optuna [4], a hyperparameter optimization framework, is adopted to automatically find the hyperparameter configurations corresponding to best agent performance. Optuna is used because the baselines used in this domain are very strong. The hyperparameters used in our experiments are listed in the appendix[2].

---

[2]Available in the longer version

In the next three subsections, we present our experiments in the three domains. For each of them, we provide a short description of the environment, the definition of the global performance metric that is used, and finally the experimental results with some comments. In all our figures, we plot the average value, standard error, and standard deviation of the performance measure for each method with solid line, dark shadowed area, and light shadowed area respectively. We also performed an ablation study in Section 6.4 to understand the contributions of our different tricks. More experiments can be found in the appendix.

6.1 Mujoco Control Tasks

To check that our method can work in the simplest setting (i.e., standard RL), we test it on some classic control tasks. In this setting, although a reward function is defined, note that our LR4GPM agent never sees the ground-truth rewards. This reward function is only used to defined a performance metric (see below for more details).

Environment Description. We conduct our experiments in some MuJoCo [50] environments, specifically Hopper, Half-Cheetah, Reacher, and Walker2D, where in each environment, the agent with body parts connected by joints are simulated in 2 dimensions. Observations provided by the simulator include the positions and (angle) velocities of the joints. The agent controls the torque applied on the joints to move forward. While the reward functions in Hopper, Half-Cheetah, and Walker-2D are dense, that in Reacher is sparse.

Global Performance Metric. In the MuJoCo control tasks, we simply adopt the discounted sum of the rewards defined in those tasks as the global performance metric. This specific scenario illustrates a low-level case where the aggregation function ρ is the identity function and the trajectory evaluation function is additively decomposed. Therefore, here $D = n_\eta = 1$.

Experimental Results. In this setting, the most natural baseline is PPO [37] trained with standard rewards. We trained all agents with 1 million steps. The performance of LR4GPM along with the standard PPO is illustrated in Figure 1a, Figure 1b, Figure 1c and Figure 1d. LR4GPM can nearly match the performance of PPO in Hopper, Half-Cheetah, and Walker2D. The biggest performance gap can be observed in Walker2D. Note that LR4GPM solves a harder problem than PPO. While PPO is guided by dense rewards, LR4GPM needs to learn its own rewards. Surprisingly, LR4GPM achieves a better performance than the standard PPO in the environment of Reacher, indicating that our method is able to learn an effective and dense reward function from a global performance metric which is composed of sparse rewards. To conclude, in dense reward tasks LR4GPM can achieve competitive performance with little inferior convergence speed and optimality compared with the usual PPO. In sparse reward tasks, LR4GPM can be superior to PPO. These points suggest the potential of learning the near-optimal reward function from the global performance metric.

6.2 Networking

We perform experiments in this domain to demonstrate a practical use case of LR4GPM. In real world applications, a global performance is generally available, but a reward function is not.

Environment Description. We conduct our experiments in the domain of data center control, where a centralized controller manages a computer network, which is shared by hosts, in order to optimize the bandwidth of each host. For this problem, we use a Gym environment called Iroko [35], which is based on the Mininet [24] network emulator, to simulate an RL environment for this data center control problem. The agent observes statistics from switches in the network and stores them as a $d \times n$ matrix which contains traffic information going through n ports and d network features. The agent controls the percentage of the maximum bandwidth each host is allocated. In this domain, we focus on a relatively complex scenario, which corresponds to a fat-tree topology.

Global Performance Metric. In networking, it is common to evaluate a control by integrating the following two aspects:

- **Throughput**: summing the number of bytes received by all hosts divided by the delivery time (last receiving time - first sending time) for all packets.

- **Delay**: summing all the differences between the receiving time and the sending time divided by the number of packets.

We therefore define the global performance metric in this domain using the following equation [43]:

$$\log(\text{throughput}) - \delta \log(\text{delay}) \tag{15}$$

where δ is a weighted factor for the delay and it is set for simplicity to be 1.0 in the experiments.

Experimental Results. Note that when they propose Iroko, Ruffy et al. [35] engineered a parametric reward function to achieve a good trade-off between throughput and delay. Inspired by the work of Sivaraman et al. [43], they define it as follows:

$$\sum_{i \in hosts} bw_i / bw_{max} - \omega \cdot (q_i / q_{max})^2 - \text{std} \tag{16}$$

where bw_i (resp. bw_{max}) is the bandwidth of host i (resp. max bandwidth), q_i (resp. q_{max}) is the queue size of host i (resp. max queue size), ω is a hyperparameter, and std is the standard deviation of the bandwidth allocation (to favor fairness). Following this reward function, the agent is therefore encouraged to find a strategy to maximize the bandwidth usage as well as minimize the occurrence of queuing on the switch interface. We trained the policies with 1 million steps for our method, PPO with the default reward, and the separate component learning method introduced in Section 5.

The performance measure of the policies for different methods are shown in Figure 1e. For PPO trained with the default reward, we show the agent with best performance among agents trained with different values for the factor ω. We see that separate reward learning works as well as PPO trained with the default reward, but LR4GPM performs better than the others in this networking problem, although it converges slightly slower. This is probably due to the fact that LR4GPM needs some time to learn good rewards.

6.3 Autonomous Driving

We chose this third evaluation domain to fully demonstrate the potential of our approach. This domain was proposed in a recent RL competition organized at DAI 2020. In this competition, a global performance metric was defined to evaluate the submissions of the
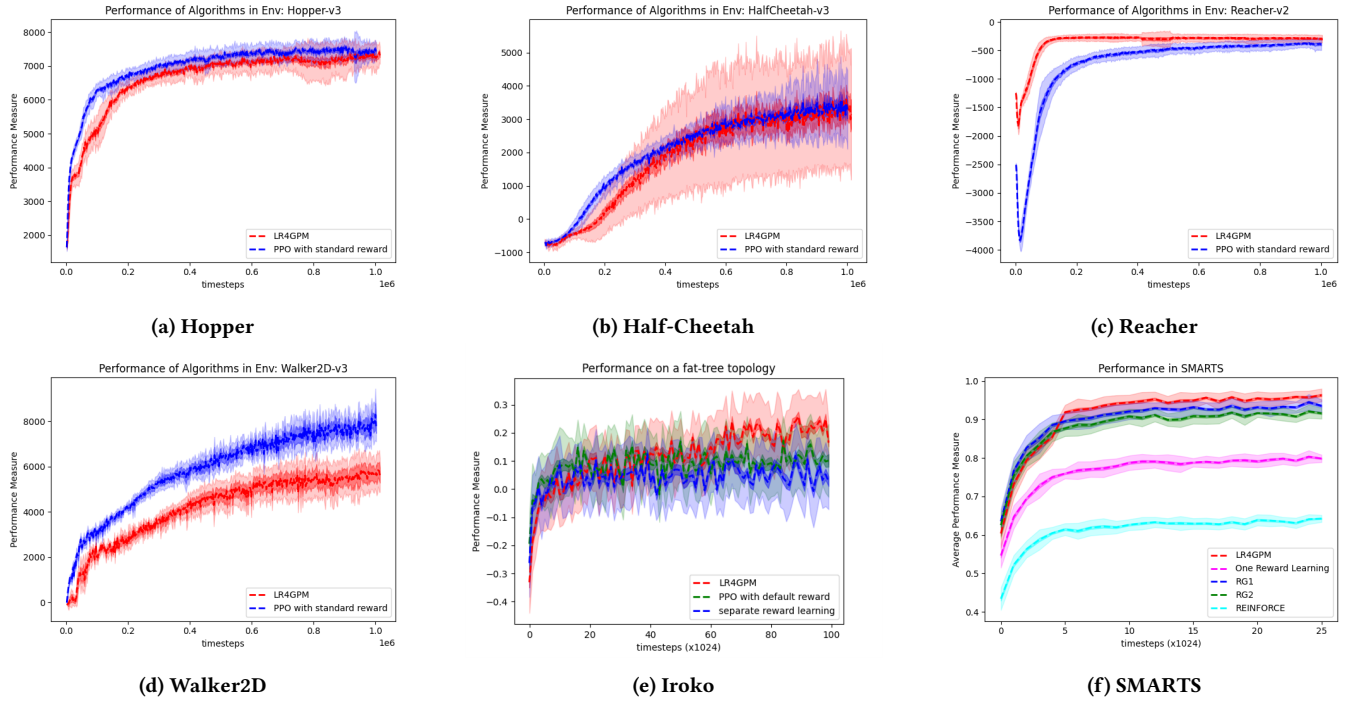
Figure 1: Algorithm performance. Performance measure of learned policy varied over timesteps. Mean and standard deviation are showed in the figure. Our method outperforms baselines in most cases.

participants. A default reward function was also provided, but it lead to a poor performance in terms of this performance metric. Therefore, many participants spent a lot of effort to design a more efficient and informative reward function. As we will show now, LR4GPM can bypass the reward engineering step by learning a good reward function to train the RL agent.

Environment Description. The last domain we conduct our experiments on is the SMARTS [55] (Scalable Multi-Agent RL Training School) platform which provides a simulation environment for reinforcement learning in autonomous driving. SMARTS offers a Gym interface [10] for RL agents. We focus on the single agent situation in our experiments, where the agent is a specific moving vehicle in the scenario. The observation space consists of the dynamic features of the moving vehicle and the road condition around the vehicle. The agent can make a decision (accelerating, braking, turning) every step to determine the next vehicle state.

Global Performance Metric. The performance of the agent is evaluated according to the following global performance metric:

$$\alpha \cdot \left(A \cdot \left(1 - \frac{B}{B_{max}} \right) \right) + \beta \cdot \left(1 - \frac{C}{W_{road}} \right) + \gamma \cdot \frac{D}{L_{route}} \qquad (17)$$

where $\alpha + \beta + \gamma = 1$ and A, B, C, and D are defined as follows:

- **Safety:** A is the percentage of routes or missions completed without critical infractions (e.g., crashes with other vehicles).
- **Time:** B is the average time took to finish the task, also the timestep.

- **Control quality:** C is the average distance to the center of the lane.
- **Valid distance travelled:** D is the average driving distance along the predefined route.

The final score for a single map will be mapped into [0,1] by normalizing each independent metric. As shown in the formulation, B_{max} represents the maximum time consumption; W_{road} is half of the road width; L_{route} is the length of the predefined route. In this domain, we focus on the loop scenario where α is set to 0.4, β is set to 0.1, and γ is set to 0.5.

Baselines. We compare our approach with the following baselines in the experiments of autonomous driving:

RG1, RG2: PPO using human-specified reward to optimize the global performance metric. **RG1** and **RG2** are the reward function engineered by the winner and the runner up of the competition organized at DAI 2020. Those are therefore strong baselines.

One Reward Learning: PPO using only one scalar reward to learn the global performance metric.

REINFORCE: REINFORCE-like algorithm that directly optimizes the global performance metric in Problem (7).

Experimental Results. We present the results in Figure 1f. We notice that LR4GPM outperforms other methods in terms of performance measure. The agent of LR4GPM effectively learns from the rewards trained with the global performance metric. The two
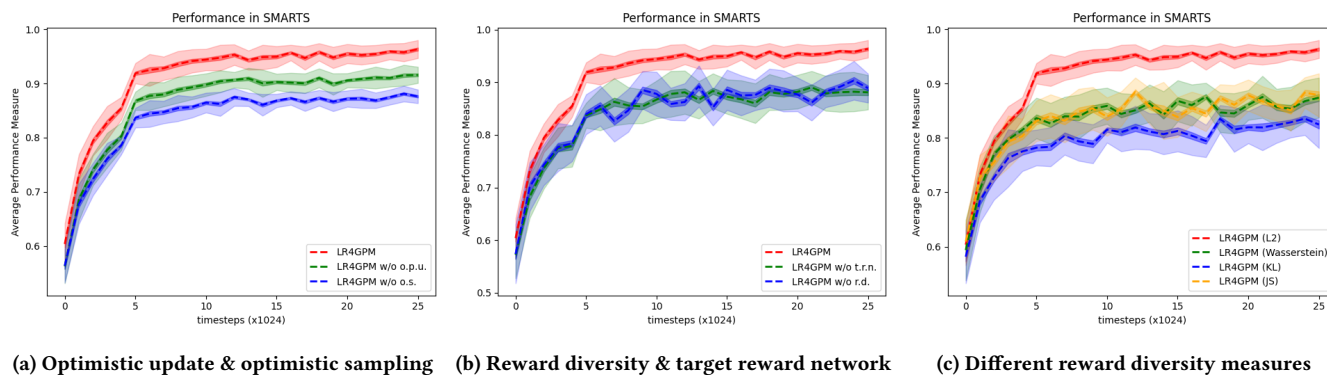
(a) Optimistic update & optimistic sampling (b) Reward diversity & target reward network (c) Different reward diversity measures

Figure 2: Ablation study. Comparison of ablated training tricks in LR4GPM performed in SMARTS platform shows their contributions. Mean and standard deviation are showed in the figure.

reward engineering methods also achieve good performance. Learning with one scalar reward is worse than LR4GPM because the non-linear performance metric cannot be directly decomposed into sum of rewards. The REINFORCE-like algorithm also fails to achieve good performance.

6.4 Ablation Study

In our method, we apply the following tricks in reward learning and agent training: optimistic policy update, optimistic sampling, reward diversity, and target reward network. We perform an analysis of their contributions in the performance of our LR4GPM.

Optimistic Policy Update. After we update the policy for the last time in the agent training (inner loop), multiple minibatches are sampled from the replay buffer to update the same number of candidate policies. The updated policy would be evaluated with respect to the global performance metric. We choose the one with best performance measure as the optimistic policy for the next reward update. In order to disentangle the effects it has on LR4GPM, we compare the performance of the agent trained with and without the optimistic policy update (green curve) in Figure 2a. We notice that it has little difference in the beginning and expand to some extent as the agent learns from the dynamically updated rewards.

Optimistic sampling. When sampling the minibatch from the replay buffer, we sample much more than the size of the minibatch and then keep the best ones as a minibatch. We test the effects of removing the optimistic sampling and sample the minibatch randomly from the replay buffer. The blue curve in Figure 2a shows that with the help of trajectory optimization, the agent can learn faster and achieve much better performance.

Reward diversity. To alleviate the effects of non-stationarity of the dynamic environment, we add one more term in the objective function that increases the diversity of the different components of the reward function. We choose the L_2 distance criterion to measure the differences between pairs of reward components. Figure 2b shows that the introduction of reward diversity significantly helps in obtaining better performance and stabilizing the policy learning.

We also tried the following types of distance/divergence computation in the loss of reward learning: squared L_2 distance (applied in

LR4GPM), Wasserstein distance, KL divergence, and JS divergence. Figure 2c shows their effects on the performance measure. The agent achieves the best performance when applying L_2 distance in the loss formulation. The agent trained with the loss where KL divergence is applied obtains the worst performance measure. It could be highly related with the mismatch between the asymmetry property of KL divergence and the symmetry property in the reward vector.

Target reward network In our method, we apply the moving average of the rewards in agent training to take into account the impact of the past reward from the target reward network, which helps to stabilize the policy learning. We test the effects of removing the target reward network and directly applying the current rewards in the agent training. Figure 2b shows that the performance measure of the agent decreases and the variance increases due to the lack of moving average rewards.

7 CONCLUSION

In this paper, we introduce a new RL problem where the reward function is assumed not to be known, but the RL task is defined with a known overall performance metric. For this problem, we design a novel algorithm that simultaneously (1) learns to approximate the performance metric with a (potentially vector) reward network and (2) trains a policy using this reward network. Due to the difficulty of this problem, several tricks are called for to face the issues of non-stationarity training and early convergence to suboptimal policies. Our proposition, LR4GPM, is validated on several domains and shown to outperform various alternatives approaches. Notably, LR4GPM surpasses a reward engineered approach that wins in a recent RL competition, which is very strong baseline.

This indicates that LR4GPM could be a promising approach to replace reward engineering and human expertise. Indeed, our work can also be seen as a step towards automatic RL [31] where rewards, interpreted as hyperparameters, are adapted online. Future work includes designing alternative algorithms for our new problem and extending the approach to the (cooperative or competitive) multi-agent setting.

ACKNOWLEDGMENTS

REFERENCES

[1] David Abel, Will Dabney, Anna Harutyunyan, Mark K. Ho, Michael L. Littman, Doina Precup, and Satinder Singh. 2021. On the Expressivity of Markov Reward. In *NeurIPS*. Curran Associates, Inc., 7799–7812.

[2] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained Policy Optimization. In *ICML*. JMLR.org, Sydney, NSW, Australia, 10.

[3] Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. 2019. Learning to Generalize from Sparse and Underspecified Rewards. In *ICML*. PMLR, Long Beach, 130–140.

[4] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*. Association for Computing Machinery, New York, NY, USA, 2623–2631. https://doi.org/10.1145/3292500

[5] M. Allais. 53. Le comportement de l'homme rationnel devant le risque : critique des postulats de l'école américaine. *Econometrica* 21 (53), 503–546.

[6] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete Problems in AI Safety. (2016). ArXiv 1606.06565.

[7] B. Argall, S. Chernova, M. Veloso, and B. Browning. 2009. A Survey of Robot Learning from Demonstration. *Robotics and Autonomous Systems* 57, 5 (2009), 469–483.

[8] Thomas Arnold, Daniel Kasenberg, and Matthias Scheutz. 2017. Value Alignment or Misalignment – What Will Keep Systems Accountable?. In *AAAI*. AAAI, San Francisco, California, USA.

[9] Saurabh Arora and Prashant Doshi. 2021. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence* 297 (2021).

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. http://arxiv.org/abs/1606.01540 arXiv:1606.01540 [cs].

[11] Wang Chi Cheung. 2019. Regret Minimization for Reinforcement Learning with Vectorial Feedback and Complex Objectives. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/a02ffd91ece5e7efeb46db8f10a74059-Paper.pdf

[12] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. 2017. Risk-constrained Reinforcement Learning with Percentile Risk Criteria. *JMLR* 18, 1, 6070–6120.

[13] Yinlam Chow, Aviv Tamar, Shie Mannor, and Marco Pavone. 2015. Risk-Sensitive and Robust Decision-Making: a CVaR Optimization Approach. In *NIPS*, Vol. 28. Curran Associates, Inc.

[14] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep Reinforcement Learning from Human Preferences. In *NIPS*, Vol. 30. Curran Associates, Inc., 4299–4307. http://papers.nips.cc/paper/7017-deep-reinforcement-learning-from-human-preferences.pdf

[15] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. 2021. First return, then explore. *Nature* 590, 7847 (2021).

[16] Layla El Asri, Romain Laroche, and Olivier Pietquin. 2012. Reward Function Learning for Dialogue Management. In *STAIRS*.

[17] Yingjie Fei, Zhuoran Yang, Yudong Chen, and Zhaoran Wang. 2021. Exponential Bellman Equation and Improved Regret Bounds for Risk-Sensitive Reinforcement Learning. In *NeurIPS*.

[18] Chelsea Finn, Sergey Levine, and Pieter Abbeel. 2016. Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization. In *ICML* (New York, NY, USA). 49–58. http://dl.acm.org/citation.cfm?id=3045390.3045397

[19] Jim Gao. 20014. *Machine Learning Applications for Data Center Optimization*. Technical Report. Google.

[20] Javier Garcia and Fernando Fernandez. 2015. A Comprehensive Survey on Safe Reinforcement Learning. *JMLR* 16, 1437–1480 (2015).

[21] Ricardo Grunitzki and Ana L C Bazzan. 2017. A Flexible Approach for Designing Optimal Reward Functions (Extended Abstract). In *International Conference on Autonomous Agents and Multiagent Systems*. São Paulo, Brazil, 3.

[22] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. 2018. Reward learning from human preferences and demonstrations in Atari. In *NIPS*. 8022–8034. http://papers.nips.cc/paper/8025-reward-learning-from-human-preferences-and-demonstrations-in-atari.pdf

[23] Vijaymohan R. Konda and Vivek S. Borkar. 1999. Actor-Critic–Type Learning Algorithms for Markov Decision Processes. *SIAM Journal on Control and Optimization* 38, 1 (1999), 94–123.

[24] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*.

[25] Kimin Lee, Laura M. Smith, and Pieter Abbeel. 2021. PEBBLE: Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-training. In *Proceedings of the 38th International Conference on Machine Learning*.

[26] Bingyao Liu, Satinder Singh, Richard L. Lewis, and Shiyin Qin. 2012. Optimal rewards in multiagent teams. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics*.

[27] M.J. Machina. 1988. Expected Utility Hypothesis. In *The New Palgrave: A Dictionary of Economics*, J. Eatwell, M. Milgate, and P. Newman (Eds.). Macmillan, 232–239.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533. https://doi.org/10.1038/nature14236

[29] Hamid Nazerzadeh, Amin Saberi, and Rakesh Vohra. 2008. Dynamic cost-per-action mechanisms and applications to online advertising. In *WWW*.

[30] Andrew Y. Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*. 278–287.

[31] Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, and Marius Lindauer. 2022. Automated Reinforcement Learning (AutoRL): A Survey and Open Problems. *jair* 74 (June 2022), 517–568. https://doi.org/10.1613/jair.1.13596 arXiv:2201.03916 [cs].

[32] J. Randlov and P. Alstrom. 1998. Learning to drive a bicycle using reinforcement learning and shaping. In *ICML, (1998)*.

[33] Mathieu Reymond, Conor F. Hayes, Diederik M. Roijers, Denis Steckelmacher, and Ann Nowé. 2021. Actor-Critic Multi-Objective Reinforcement Learning for Non-Linear Utility Functions. In *MODeM*.

[34] Diederik M. Roijers, Denis Steckelmacher, and Ann Nowé. 2018. Multi-objective Reinforcement Learning for the Expected Utility of the Return. In *ALA workshop at AAMAS/ICML/IJCAI*.

[35] Fabian Ruffy, Michael Przystupa, and Ivan Beschastnikh. 2019. Iroko: A Framework to Prototype Reinforcement Learning for Data Center Traffic Control. In *Workshop on ML for Systems at NeurIPS*. http://arxiv.org/abs/1812.09975

[36] Stuart Russell. 1998. Learning Agents for Uncertain Environments. In *COLT*.

[37] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. http://arxiv.org/abs/1707.06347 arXiv:1707.06347 [cs].

[38] William F. Sharpe. 1994. The Sharpe Ratio. *The Journal of Portfolio Management* 21, 1 (1994).

[39] Umer Siddique, Paul Weng, and Matthieu Zimmer. 2020. Learning Fair Policies in Multi-Objective (Deep) Reinforcement Learning with Average and Discounted Rewards. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 8905–8915.

[40] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550 (2017), 354–359.

[41] David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton. 2021. Reward is enough. *Artificial Intelligence* 299 (Oct 2021).

[42] Satinder Singh, Richard L Lewis, and Andrew G Barto. [n.d.]. Where Do Rewards Come From? ([n. d.]), 6.

[43] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. 2014. An Experimental Study of the Learnability of Congestion Control. In *ACM SIGCOMM*.

[44] Jonathan Sorg, Richard L. Lewis, and Satinder Singh. 2010. Reward Design via Online Gradient Ascent. In *NeurIPS*, Vol. 23. container-title: Advances in Neural Information Processing Systems.

[45] Halit Bener Suay, Tim Brys, Matthew E. Taylor, and Sonia Chernova. 2016. Learning from Demonstration for Shaping through Inverse Reinforcement Learning. In *AAMAS*. https://dl.acm.org/doi/10.5555/2936924.2936988

[46] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press.

[47] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS'99)*. MIT Press, Cambridge, MA, USA, 1057–1063.

[48] Aviv Tamar, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor. 2015. Policy Gradient for Coherent Risk Measures. In *NIPS*.

[49] G. Tesauro. 1990. Neurogammon: a neural-network backgammon program. In *IJCNN*, Vol. 3.

[50] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2012), 5026–5033.

[51] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yuri Sulsky, James Molly, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.

[52] Ronald J Williams. [n.d.]. Simple statistical gradient-following algorithms for connectionist reinforcement learning. ([n.d.]), 28.

[53] Jianyi Zhang and Paul Weng. 2021. Safe Distributional Reinforcement Learning. In *International Conference on Distributed Artificial Intelligence (DAI)*.

[54] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. 2018. On learning intrinsic rewards for policy gradient methods. In *NeurIPS*.

[55] Ming Zhou, Jun Luo, Julian Villella, Yaodong Yang, David Rusu, Jiayu Miao, Weinan Zhang, Montgomery Alban, Iman Fadakar, Zheng Chen, Chongxi Huang, Ying Wen, Kimia Hassanzadeh, Daniel Graves, Zhengbang Zhu, Yihan Ni, Nhat Nguyen, Mohamed Elsayed, Haitham Ammar, Alexander Cowen-Rivers, Sanjeevan Ahilan, Zheng Tian, Daniel Palenicek, Kasra Rezaee, Peyman Yadmellat, Kun Shao, Dong Chen, Baokuan Zhang, Hongbo Zhang, Jianye Hao, Wulong Liu, and Jun Wang. 2021. SMARTS: An Open-Source Scalable Multi-Agent RL Training School for Autonomous Driving. In *Proceedings of the 2020 Conference on Robot Learning*. PMLR, 264–285. https://proceedings.mlr.press/v155/zhou21a.html ISSN: 2640-3498.

[56] Matthieu Zimmer, Claire Glanois, Umer Siddique, and Paul Weng. 2021. Learning Fair Policies in Decentralized Cooperative Multi-Agent Reinforcement Learning. In *ICML*.