Adaptive Reinforcement Learning with LLM-augmented Reward Functions

Alex Place [1]

[1]DB Labs

December 7, 2023

Abstract

Learning in sequential decision making problems can be significantly affected by the choice of reward function. Although crucial for learning success, reward function design remains a complex task, requiring expertise and not always aligned with human preferences. Large Language Models (LLMs) offer a promising avenue for reward design through textual prompts that leverage the prior knowledge and contextual reasoning of LLMs. Despite their potential, LLM responses lack guarantees and their reasoning abilities are poorly understood. To mitigate potential LLM errors, we introduce an alternative approach: learning a new reward function utilizing LLM outputs as auxiliary rewards. This problem is tackled through a bi-level optimization framework, showcasing the method's proficiency in both optimal reward acquisition and adaptive reward shaping. The proposed approach demonstrates robustness and effectiveness, offering a novel strategy for enhancing reinforcement learning outcomes.Â Â Â Â Â Â

Adaptive Reinforcement Learning with LLM-augmented Reward Functions

Alex Place
alplace@deltabluelabs.com
USA

ABSTRACT

Learning in sequential decision making problems can be significantly affected by the choice of reward function. Although crucial for learning success, reward function design remains a complex task, requiring expertise and not always aligned with human preferences. Large Language Models (LLMs) offer a promising avenue for reward design through textual prompts that leverage the prior knowledge and contextual reasoning of LLMs. Despite their potential, LLM responses lack guarantees and their reasoning abilities are poorly understood. To mitigate potential LLM errors, we introduce an alternative approach: learning a new reward function utilizing LLM outputs as auxiliary rewards. This problem is tackled through a bi-level optimization framework, showcasing the method's proficiency in both optimal reward acquisition and adaptive reward shaping. The proposed approach demonstrates robustness and effectiveness, offering a novel strategy for enhancing reinforcement learning outcomes.

ACM Reference Format:
Alex Place. 2023. Adaptive Reinforcement Learning with LLM-augmented Reward Functions. In *Proceedings of the ACM Computing Surveys (ACS '23), January, 2023.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The nature of the reward function is a critical determinant of the success of RL methods in solving a task. While specifying sparse rewards for tasks might be straightforward, the absence of a dense and instructive reward often presents challenges for RL algorithms[17, 25]. On the other hand, rewards that offer frequent and effective feedback to track progress can significantly enhance the learning process [24, 39, 43], but designing such rewards is far from straightforward. Consequently, a substantial body of research has focused on reward-shaping and related methods [17, 25, 34**?**].

The process of constructing reward functions is further compounded by the difficulty of providing helper rewards from agent trajectories or demonstrations, which typically requires domain expertise. Furthermore, translating human intuition into concrete rewards remains a formidable task, even with the utilization of reward-shaping methods. For instance, in many video-games, an agent must attentively monitor diverse resources such as health, mana, and prioritize the development of skills and acquisition of inventory items [37]. This is further exacerbated by the fact that human intuition, often falls short in crafting rewards that guide an agent's learning in a desirable fashion [3, 18] and often leads to reward-hacking.

In this context, incorporating background knowledge about tasks using Large Language Models (LLMs) emerges as a natural solution. LLMs have demonstrated significant prior-knowledge based reasoning capabilities of human behavior, which allows users to articulate a handful of desired behaviors intuitively using natural language. LLMs can serve as a source of information to guide the reward design process. Finally, these models can also go beyond being mere passive information sources and suggest potentially valuable behaviors for specific tasks, via in-context learning.

Nonetheless, embracing LLMs as proxy reward functions introduces its own set of challenges. While LLM outputs hold promise, they are not always guaranteed to be helpful for the given task and can even lead to counterproductive outcomes. Moreover, LLMs can show surprising biases and behaviours in different contexts. Finally, even if the LLM's reasoning is perfect, incorrect instruction or state and task-description can lead to errors [**? ?**]. As such rewards derived from LLMs might result in suboptimal behavior [34**?**]. To address these complexities, there is a need to develop a methodology that harnesses the informative potential of LLM-derived rewards while ensuring policy invariance guarantees and maintaining safety and robustness in the face of potential errors from the LLMs.

In order to tackle this challenge, we propose an optimal-reward learning problem that focuses on adapting an auxiliary reward signal to align it with the primary reward. The primary reward refers to the underlying true reward of the task. The auxiliary reward is provided by an LLM, and is distinct from the primary reward. The LLM's output is intended to be utilized as an additional signal provided to the agent to speed up learning. Our objective is to leverage the valuable guidance offered by the given reward function while disregarding any inaccuracies or unhelpful signals it may contain.

This problem lends itself naturally to a bi-level optimization formulation. The inner level pertains to standard policy optimization with a parameterized auxiliary reward function, while the outer level revolves around optimizing primary returns. This outer-level solver refines the auxiliary reward function such that the optimal policy under the auxiliary reward maximizes the primary return. Solving this optimization problem is computationally challenging. We adopt the implicit gradients technique [23], which offers an efficient method to compute gradients for the outer-level optimizer. We first investigate our method's efficacy for both adaptive reward shaping and then demonstrate its utility in LLM-augmented reinforcement learning.

ACS '23, January, 2023,
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 PRELIMINARIES

We first describe the notations followed in the paper followed by a description of and . We also summarize the implicit gradient method which is used in this work for the outer optimization via gradient based methods.

2.1 Reinforcement Learning with Learnt Rewards

Notation. A Markov Decision Process (MDP) is specified by a tuple $(\mathcal{S}, \mathcal{A}, P, r_p, \gamma)$. $\mathcal{S}$ is the state space, $\mathcal{A}$ is the set of actions, $P : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \rightarrow [0,1]$ are the transition probabilities, $r_p : \mathcal{S}\times\mathcal{A} \rightarrow [-R_{\max}, R_{\max}]$ is the primary reward function, $\gamma \in [0,1]$ is the discount factor [1]. A policy $\pi : \mathcal{S} \times P(\mathcal{A})$, is a function that maps a state s to a probability distribution over the action space. At any given time step t, the agent following policy π executes the action a when observing state s, according to the distribution $\pi(s)$. For brevity, the probability of taking action a under $\pi(s)$ i.e. $\pi(s)(a)$ will be shortened to $\pi(s,a)$. The goal in this case is to optimize the cumulative discounted primary rewards obtained by the policy. If we denote by S_t, A_t, R_t be the state, action and reward at time t, then the objective is given by:

$$V(\theta, r_p) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} \gamma^t r_p(S_t, A_t)] \tag{1}$$

Here, we explicitly write V as a function of r_p, as we will use the same function with a different reward function as well.

The policy gradient theorem [48] provides a way to obtain gradient estimates of the value $V(\theta, r_p)$ via sample runs on the MDP. The optimal parameter $\theta^* = \operatorname{argmax} V(\theta, r_p)$ can then be obtained using gradient based optimization using these estimated gradients. The policy gradient is given by:

$$\nabla V(\theta, r_p) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \partial_\theta \log(\pi_\theta(s,a)) \sum_{j=t}^{T} \gamma^{j-t} r_p(S_j, A_j)\right] \tag{2}$$

Auxiliary Rewards and Reward Shaping. In many applications, the objective $V(\theta, r_p)$ can be difficult to optimize due to lack of strong supervision [34]. A natural approach to help training is to augment the learning agent with an auxiliary reward function which incorporates domain knowledge. Typically, experts often design useful helper rewards which can be helpful for the agent by providing rewards for the agent to visit a certain state or perform an action sequence [35]. These helper rewards are added to or substituted in place of the primary reward. To ensure, that the new reward function does not create behaviour different from the intended behaviour, Ng et al. [34] suggest a potential-based approach. Their key idea is that invariance can be maintained by a potential function based auxiliary reward viz. $r_{aux} = \gamma\Phi(s') - \Phi(s)$, where $\Phi : \mathcal{S} \rightarrow \mathbb{R}$ is a potential function. Some reward shaping proposals utilizing the idea of a potential function include potential-based advice (PBA) [52] and dynamic potential reward shaping (PBRS) [9].

[1]Often there is an initial state distribution as well, but for our purpose we omit it

2.2 Implicit Gradient Method

Implicit gradient technique allows one to analytically obtain the gradient from the resultant of an optimization process directly. A particular instance of this technique relevant for the current work is that for a optimization problem of the form:

$$\begin{aligned} &\eta^* = \operatorname*{argmin}_{\eta} L_{\text{Prim}}(\theta^*(\eta), \eta) \\ &\text{such that} \\ &\theta^*(\eta) = \operatorname*{argmin}_{\theta} L_{\text{Aux}}(\theta, \eta) \end{aligned} \tag{3}$$

one can find the 'total-gradient' of η as:

$$\frac{\partial(L_{\text{Prim}}(\theta^*(\eta), \eta))}{\partial\eta} - \underbrace{\left[\frac{\partial}{\partial\eta}\frac{\partial}{\partial\theta}(L_{\text{Aux}}(\theta,\eta))\right]\left[\frac{\partial}{\partial\theta}\frac{\partial}{\partial\theta}(L_{\text{Aux}}(\theta,\eta))\right]^{-1}\frac{\partial(L_{\text{Prim}}(\theta^*(\eta), \eta)}{\partial\theta}}_{\text{meta}G} \tag{4}$$

Note that in Equation 4, the total gradient consists of two terms: i) is the direct partial derivative of L_{Prim} w.r.t η and ii) the meta-gradient term labeled meta$G$. This term characterizes the change in objective L_{Prim} due to the fact that the optimal θ^* also is dependent on η. The reason we call it a meta-gradient is in many meta-learning models (for eg. MAML), η is equivalent to the initial parameters of the meta-model, which is updated to $\theta*$ based on the sampled tasks (analogous to optimizing L_{Aux}) after which $\theta*$ is evaluated on a specific task.

The above result is valuable as it only requires the value of the optimal θ^* to obtain the gradient wrt η, and does not depend on how the inner optimization is conducted. In contrast, when the inner optimization is conducted via gradient descent, one can use the automatic differentiation to backpropagate through the optimization (as is frequent in meta-learning applications). It can be proven that the two results are identical [23]; but when the required hessians multiplications can be done efficiently, then the implicit method is more scalable. Moreover if the inner optimization is not perfect then the backprop through optimization route is dependent on choice of optimization, number of steps and other choices. This makes further using implicit gradient technique attractive compared to choosing hyperparameters for the inner optimization process.

3 DYNAMIC ADAPTATION OF LLM REWARD FUNCTIONS

3.1 Using LLMs for Reward

As mentioned earlier, designing useful auxiliary rewards can be challenging requiring human expertise and resources. As such, researchers are now trying to use LLMs for the purpose of designing rewards for RL agents [].The LLM can function as an in-context learner, a knowledge-base for queries, or an oracle suggesting actions, goals, and sub-tasks. We describe a versatile framework for employing an LLM, with the specific intention of general applicability in all aforementioned models; as such, we refrain from delving into the nuances of the exact prompt structure or prompt-tuning methods.

Mathematically, the LLM operates as a function, denoted as $LLM : \Sigma^* \rightarrow \Sigma*$, where Σ signifies a carefully selected alphabet. Operatively, the LLM takes as input a composite text comprising distinct components. Foremost is the general prompt, a user-provided string forming part of the LLM's instructions. Subsequently, a contextual string augments this input, equipping the LLM with additional information for reasoning. This contextual portion accommodates various elements, such as target state descriptions, task elucidations, or demonstrations. This part of the input can also contain examples and outputs for using the LLM as an in-context learner. The third component contains the current status of the agent, encompassing descriptions of the current state and actions. These descriptions are sourced either from a captioner model or by directly learning state-action embeddings. This section may also encompass information concerning recent state-action pairs. Lastly, the inquiry component prompts the LLM to generate an output string. These queries can range from specific action-related questions ('Should you take action A?') to broader queries about sub-goals or sequences of actions ('Should we attempt to reach the door?').

Along with the LLM, we have a score function g that takes both the LLM's output and input, yielding a numeric value within the interval [0, 1] that serves as the reward. This value can be obtained in different ways including the probability of specific tokens ('yes' or 'no') as indicated by LLM logits, or through a similarity score between LLM's output with particular state-action descriptions (either via embedding similarity or string similarity). This resulting numeric value is subsequently employed as the auxiliary reward, r_{LLM}. This additional reward function r_{LLM} is then included in the optimization objective for the agent $V(\theta, r_p + r_{\text{LLM}})$ instead of the standard RL objective of $V(\theta, r_p)$.

Formally we include a new reward function r_{aux} such that the objective to optimize is now $V(\theta, r_p + r_{\text{LLM}})$.

3.2 Learning Auxiliary Rewards

When the reward function, denoted as r_p, exhibits sparsity or lacks informative qualities, optimizing the objective function $V(\theta, r_p)$ becomes challenging due to the absence of robust supervision [34]. To address this limitation, a natural approach is to introduce auxiliary rewards, denoted as r_{LLM}, resulting in an updated objective function to optimize: $V(\theta, r_p + r_{\text{LLM}})$. However, the inclusion of such auxiliary rewards can significantly impact the optimal solutions of the objective function and potentially cause the agent to learn a policy that deviates from the desired behavior [34]. While potential bases methods, can retain policy invariance, such methods have inherent limitations [25, 51]. Furthermore designing rewards can demand substantial expertise in translating intuitive concepts into numerical reward structures [21, 43]. To circumvent these challenges, an alternative approach is the adoption of the intrinsic optimal rewards framework [42, 44], which involves simultaneously learning a new reward function while optimizing a policy based on these rewards. By integrating reward learning and policy optimization, one can enhance the agent's ability to discover the desired behavior more effectively [57].

Building on this idea of learning intrinsic rewards, we want to change the auxiliary rewards r_{LLM} in such a way that learning a policy π_θ using them, improves the primary the primary objective $V(\theta, r_p)$. If we parameterize the auxiliary rewards as r_η, this can be written as the following bi-level objective:

$$\eta^* = \underset{\eta}{\operatorname{argmax}}\, V(\theta(\eta), r_p) \quad \text{s.t.} \quad \theta(\eta) = \underset{\theta}{\operatorname{argmax}}\, V(\theta, r_\eta)$$

Intuitively θ is learnt to improve via the rewards r_η Bechtle et al. [4], Zheng et al. [57] solve this problem via alternate maximization of θ and η, where θ is updated o improve $V(\theta, r_\eta)$, while η is updated to improve $V(\theta(\eta), r_p)$. However their method unrolls the inner optimization only a few steps. We instead are going to use the implicit gradient method to directly obtain the "meta-gradients" of η.

Reward Shaping with Auxiliary Rewards. Often experts can design useful helper rewards which can be helpful for the agent by providing rewards for the agent to visit a certain state or perform an action sequence [35]. Such helper rewards can be made available to r_η. Similarly the primary rewards can also be used in r_η. In fact, Hu et al. [19] explicitly consider learning state based weights of auxililary helper rewards; and Zheng et al. [57] optimize a combination of intrinsic/auxiliary rewards and extrinsic/primary rewards. In this work we use the LLM specified auxiliary rewards r_{LLM} and give a generic form for r_η and as:

$$r_\eta = (z_\eta(s, a) r_{\text{LLM}}(s, a) + f_\eta(s, a))$$

.

Remark 1. *Note that while we specify z_η as a function of only s, a; in principle it can also be dependent on the LLM output and/or its embeddings. The description above used f as an additional reward f which the model can learn independent of r_{LLM}, but in practice we set it to 0.*

Objective regularization. An ideal reward function is not only dense and instantaneous, but may also need to guide the agent towards long term goals. However, if one uses high discounting factors, this can be challenging as the agent can be very myopic. Since, we use a bounded reward scenario, it is helpul to allow for adjusting the discount function $\gamma_\eta \in (0, 1)$. The latter allows our method to modify temporal distribution of auxiliary rewards as well. Putting it all together we can now specify the exact objective we use for learning auxiliary rewards.

$$\begin{aligned} r_\eta(s, a) &= r_p(s, a) + z_\eta(s, a) r_h(s, a) + f_\eta(s, a) \\ \eta^* &= \underset{\eta}{\operatorname{argmax}}\, V(\theta(\eta), r_p, \gamma) - \lambda_\gamma |\gamma_\eta|^2 + \lambda_r (|f_\eta|^2) \\ \text{s.t.} \quad \theta(\eta) &= \underset{\theta}{\operatorname{argmax}}\, V(\theta, r_\eta, \gamma_\eta) \end{aligned} \tag{5}$$

where λ_γ and λ_η are hyperparameters and σ is the sigmoid function.

3.3 Estimation of Implicit Gradient

By Equation 4, the implicit gradient of η is given by:

$$\partial_\eta V(\theta, \eta) = -\underbrace{\partial^2_{\eta,\theta} V(\theta, \eta))}_{H_\eta} \left(\underbrace{\partial^2_{\theta,\theta} V(\theta, \eta)}_{H_\theta} \right)^{-1} \partial_\theta V(\theta, \eta) \tag{6}$$

where $\partial^2_{\eta,\theta}$ and $\partial^2_{\theta,\theta}$ refers to the second order derivatives.

For deterministic computation (such as in supervised learning), all the partial derivatives in Equation equation 6 are exactly computable by automatic differentiation. However this is not so for the reinforcement learning case. As such we need to find efficient ways to compute these terms, ideally in the same way as the standard policy gradient is computed. Notice that the first term in the above expression is just the partial gradient for J with respect to policy parameters. As such this can be replaced by the usual policy gradient expression (Equation 2). Next one also observes that the same policy gradient terms also appears inside the second order derivatives. Hence if one can compute the partials of the the policy gradient with respect to the parameters η, θ, one can use the corresponding expressions to compute the "meta-gradient".

Directly evaluating the meta-gradient in Equation 6 requires computing the matrices corresponding to the second-order derivatives. As mentioned before, this is not straightforward in the RL setting. Secondly, even when these matrices are computable, it can be prohibitively expensive to compute the inverse Hessian. Research in numerical methods (Jorge Nocedal and Stephen Wright., Pearlmutter) has led to various proposals for efficient inverse Hessian vector products (iHVP) including methods like conjugate gradient or Lanczos iteration. However, in our preliminary experiments with this idea, the cost of the Hessian-vector products is still very high. Furthermore as the Pearlmutter HVP trick uses gradient evaluations and the policy gradient is quite stochastic, the resulting iHVP operation is of limited utility. As such we take another approach, using the power series expansion of the inverse matrix (also known as the von-Neumann approximation). For a matrix A, with eigenvalues $\lambda(A) \in (0, 1)$, the inverse is given by:

$$A^{-1} = \sum_{i=0}^{\infty}(I - A)^i = I + (I - A) + (I - A)^2 + ..$$

Using this series, one can compute $A^{-1}b$ via the Chhebyshev-Richardson iteration: $x_0 = b$ and $x_{t+1} = (I - A)x_t + b$

Let us assume that the Hessian H_θ is positive definite, with an operator norm bound $||H_\theta|| < \lambda_M$, and let $\alpha < 1/\lambda_M$ be the conditioner step, then we use the Richardson iteration with following substitution, $A = \alpha H_\theta, b = \partial_\theta L_{\text{Prim}}$, which leads to the following iterative procedure which we run for T steps.

$$\begin{aligned}
x_{t+1} &= (I - \alpha H_\theta)x_t + \partial_\theta L_{\text{Prim}} \\
&= x_t + \partial_\theta L_{\text{Prim}} - \alpha H_\theta x_t \\
&= x_t + \partial_\theta L_{\text{Prim}} - (\partial_\theta(\partial_\theta L_{\text{Aux}}))\alpha x_t \\
&\approx x_t + \partial_\theta L_{\text{Prim}} + \underbrace{\partial_\theta [L_{\text{Aux}}(\theta - \alpha x_t) - L_{\text{Aux}}(\theta)]}_{A}
\end{aligned}$$

Note that the term A above is just the gradient of the difference of expected return, at slightly perturbed policy θ' and θ. As such unbiased estimate of this term can be obtained via standard policy gradient (Equation 2). At the end of this process, we have $x_T \approx \alpha H_\theta^{-1}\partial_\theta J$. To compute the meta-gradient we need to multiply this with H_η, for which we use the Pearlmutter trick. Once again, we would need to evaluate the gradient of the return for which we will use the policy gradient equation.

Algorithm 1 Algorithm for computing total gradient $\frac{dV(\theta(\eta), r_p)}{d\eta}$

```
Require: LLm rewards $r_{\text{LLM}}$,
  Conditioner $\alpha$,
  No. of Iterations $K$,
  Parameter $\theta^*, \eta$
  ———————————————————
  $g = \frac{\partial(L_{\text{Prim}}(\theta,\eta))}{\partial\theta}$ at $\theta = \theta^*$
  $x_0 \leftarrow 0$
  for $k \in K$ iterations do
      $L \leftarrow L_{\text{Aux}}(\theta^* - \alpha x_t) - L_{\text{Aux}}(\theta^*)$
      $x_{t+1} \leftarrow x_t + g + \frac{\partial L}{\partial\theta}$
  end for
  $g \leftarrow x_K$
  $L \leftarrow L_{\text{Aux}}(\theta^* + \alpha g) - L_{\text{Aux}}(\theta^*)$
  $g \leftarrow \frac{\partial L}{\partial\eta}$
  Return vector $g$
```

Remark 2. *Note that the gradients with respect to η are to be computed from the optimized value of θ. As such an inefficiency arises from the need to rerun the new policy $\pi_{\theta^*}(\eta)$ to compute the required gradients. One can improve the efficiency by reusing the data generated for estimating the inner optimization gradient (i.e using π_θ and do off-policy correction for estimation of the gradients at π_{θ^*}. A similar correction is used when doing policy gradient evaluation for the Hessian multiplications.*

4 EXPERIMENTS

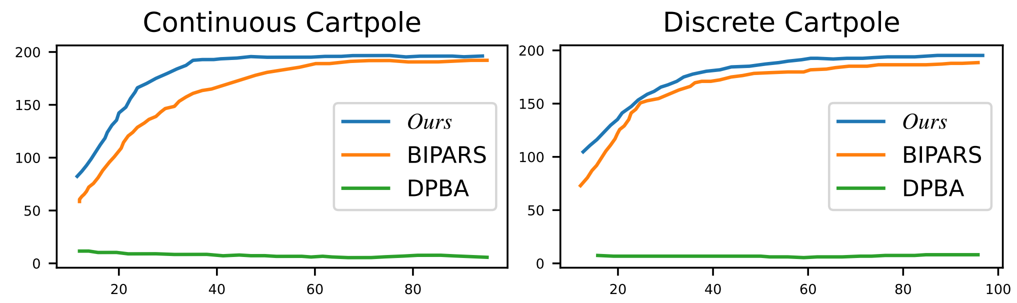
In this section, we present the results from our experiments. We conducted three groups of experiments. First we do exploratory experiments on a simple environment (*cartpole*) to explore how our model works in presence of beneficial, irrelevant and harmful auxiliary rewards. Next, we

4.1 Exploratory Experiments

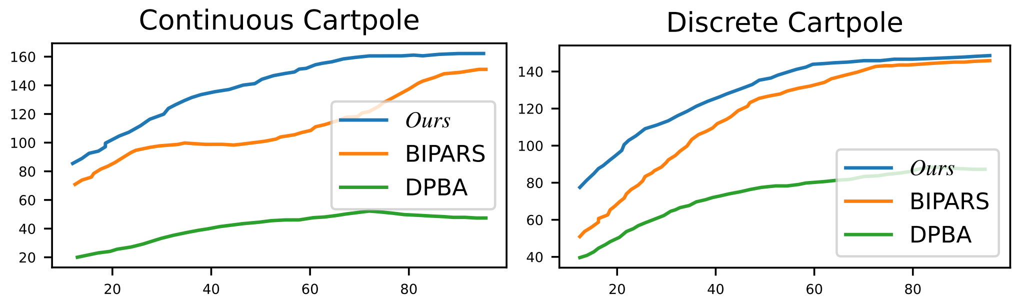
In these experiments how and whether our method is robust to random and actively harmful helper rewards. For this we conduct experiments in the cartpole environment. For the harmful rewards case, we give the external reward of -0.1 when the deviation angle of the pole becomes small. In the second experiment, we choose a random reward function with values in $[-1, 1]$.

The goal of the two tests is to show that our methods can identify the harmful shaping rewards and useless shaping rewards and downweigh or negate them. In Figure 1(a), we plot the performance of the different algorithms with the actively negative reward on both discrete as well as continuous cartpole. Similarly in Figure 1(b), we plot the performance of the different algorithms with the random reward on both environments. Since it is meaningless to provide the result of a standard algorithm such as PPO with these rewards, we plot the methods which use these rewards as advice viz DPBA [17], BIPARS[19] and our method.

The results illustrate distinct behaviors between the different shaping methods. DPBA struggles with random and detrimental rewards, which is expected given its goal of using helpful rewards. Both our proposed method and BIPARS exhibit adaptability to the mis-specified auxiliary rewards. In the first experiment, both our



(a) Experiment with harmful rewards



(b) Experiment with random rewards

Figure 1: Return curves for all methods on discrete ad continuous cartpole with a) Harmful rewards and b) Random rewards. The x-axis is time steps durin training. The y-axis is the moving window of average return.

method and BIPARS effectively identify harmful rewards and transform them into advantageous ones. Additionally, they demonstrate the capability to discern learn even with random rewards. Notably, our approach showcases a marginal advantage over BIPARS, which can be attributed to the inherent strengths of implicit gradient methods when faced with imperfect inner optimization.

4.2 Learning to Utilize LLM Rewards

Description. We test our LLM based reward learning method in the Crafter environment, a 2D version of Minecraft [16]. Like Minecraft, Crafter is a procedurally generated and partially observable world that enables collecting and creating a set of artifacts organized along an achievement tree which lists all possible achievements and their respective prerequisites. The environment is a grid-world features top-down observation and discrete action space of size 17. The agents performance is measured in terms of its health and the number of achievements it is able to unlock in an episode [16]

For our LLM, we employ GPT, and use the prompts and procedures suggested by Du et al. [12] for generating rewards. Specifically, the LLM-generated text is used to form a set of potential goals which are used for reward determination. One major difference in the experimental setting is that while Du et al. [12] use the LLM in a pretraining procedure for generating intrinsic rewards. Instead, we use these rewards for r_{LLM} which forms a components of the auxiliary reward to optimize during the inner optimization.

Result. Figure 2 offers a comparative analysis of our method, against ELLM [12], RND [5], APT [28] and standard PPO [40]. RND [5] and APT[28] are knowledge-based intrinsic motivation (IM) method that rewards the agent for novelty. ELLM [12] on the other hand is close to our motivation, and rewards the agent for achieving the goals suggested by the LLM. During policy learning, the IM rewards are combined with random exploration in order for the agent to solve the task. We instead provide an LLM based reward function into the auxiliary reward model, which updates itself over the episodes. As can be from the Figure, our method matches or outperforms all other algorithms, demonstrating its efficacy in utilizing the LLM generated signals. Note that while the APT and RND are primarily an intrinsic motivation (IM) methods, they are very helpful in achieving high scores in Crafter. Standard RL algorithms require significantly more time and often fail to unlock many achievements [16] in crafter environment. Thus while these methods are not necessarily the most optimal baselines, improving over these methods highlights the potential of our proposal.

5 RELATED WORK

Meta Learning. Meta-learning broadly refers to the idea of learning to learn and has been for tasks like multi-task and structural transfer [15, 33], adaptive learning [31, 32] and other RL applications [36, 55]. A common aspect in all these applications is to joint learning and adaptation of the model. This allows the model to learn suitable parameters which improve further learning [13, 22, 49]. As such the ideas in this work are closely related to meta-learning. As rewards are the RL analog of training objectives in supervised

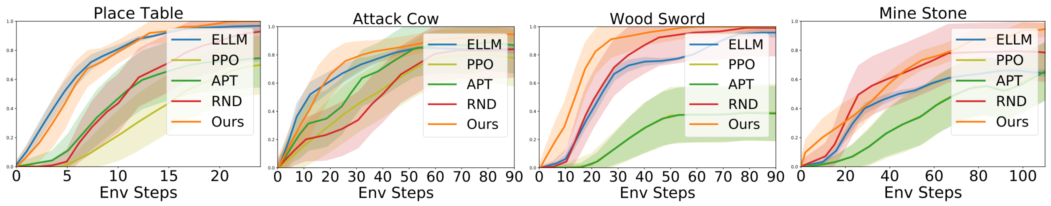


Figure 2: Completion performance of various methods for different tasks in the Crafter environment. Each line shows the mean across 10 seeds with shaded area showing the standard deviations. The primary reward is the standard sparse reward. The x-axis is time steps durin training. The y-axis is the moving window of success rate.

learning, learning a new reward function is similar to the idea of learning parameterized objectives (in our case η) explored in supervised learning [47]. Earlier works such as that of Huang et al. [20], Wu et al. [53] have attempted to learn training losses for supervised learning with neural networks; and our bi-level learning framework is the RL analog of their proposed framework.

Implicit Gradients The method of implicit gradients [10, 23] naturally occurs whenever one deals with hierarchical optimization. They provide an analytical technique for differentiable optimization [50], and so have been used for building such layers in neural-networks [1, 2]. They have also been used for few-shot learning [26, 38] and automatic hyper-parameter optimization [29].

Learning Rewards Recent advancements in reinforcement learning research have embraced the fundamental meta-learning framework for acquiring knowledge about functions concerning trajectory returns [56], TD learning targets [55], and update rules [22, 36]. Additionally, meta-learning has also been employed by Bechtle et al. [4] to learn novel rewards for multi-task reinforcement learning applications. Drawing inspiration from the optimal reward framework [27, 41, 42, 45], our work builds upon this foundation. The core framework of learning within a multi-level optimization paradigm explored in our study bears striking similarities to the approaches pursued by Zheng et al. [57] and Hu et al. [19]. In a similar vein to the literature on learnable losses [4, 53], these works treat the inner and outer optimization steps as computations within a unified graph, enabling the backpropagation of gradients through the truncated optimization process. However, it is important to note that these approaches suffer from an inadequate characterization of the model and optimizee parameters. Unlike in supervised learning, the gradients in this context are highly stochastic and necessitate extensive unrolling over numerous time steps to capture the true dynamics of the optimization process.

Bilevel Optimization. Multi-stage optimization has a long-history in optimization and operations research, both for deterministic (Bertsimas & Caramanis, 2010) and stochastic objectives (Pflug & Pichler, 2014). In general, these problems exhibit NP-hardness if the objectives are general non-convex functions. Therefore, much work in this area restricts focus to classes of problems, such as affine (Bertsimas & Caramanis, 2010; Bertsimas et al., 2010). From both the mathematical optimization community (Bracken & McGill, 1973) and algorithmic game theory (Von Stackelberg, 2010), extensive interest has been given to specifically two-stage problems. A non-exhaustive list contains the following works (Ghadimi & Wang, 2018b; Yang et al., 2021; Khanduri et al., 2021; Li et al., 2022; Ji & Liang, 2022; Huang et al., 2022). Predominately, one these works do not consider that either stage is a MDP, and therefore while some of the algorithmic strategies are potentially generalizable to the RL agent alignment problem, they are not directly comparable to the problem studied her

Inverse RL and Behavioral Cloning. Implicitly contained in the RL with preferences framework is the assumption that humans should design a reward function to drive an agent towards the correct behavior through its policy optimization process. This question of reward design has also been studied through an alternative lens in which one provides demonstrations or trajectories, and seeks to fit a reward function to represent this information succinctly. This class of problems, broadly referred to as inverse RL (Ng et al., 2000), has a long history in experimentally driven RL research – see (Arora & Doshi, 2021) for a survey. Rather than design a reward function as an intermediary between demonstrations and policy optimization, behavioral cloning directly seeks to mimic the behavior of demonstration information (Torabi et al., 2018; Wen et al., 2020). In doing so, it is able to sidestep some of the questions of whether a reward function can be well-posed for a given collection of trajectories

Reward Shaping and Partially Correct rewards. Reward shaping is a common approach to handle partially correct rewards, by modifying the provided partial reward function [30] Early research on reward shaping [11, 39] primarily focused on the design of a shaping reward function. However, these approaches were essentially designed via trial-and-error required lot of expertise. Another crucial aspect in these endeavors was the potential impact of shaping rewards on the optimal policy. Building upon this, Ng et al. [34] made significant contributions by demonstrating that potential-based reward shaping preserves policy optimality. An innovative approach known as dynamic potential-based advice (DPBA), introduced by Harutyunyan et al. [17], offers a direct means of incorporating external auxiliary rewards by transforming them into potentials. Research has shown the effectiveness of potential-based reward shaping (PBRS) in numerous applications [9, 14**?**]. Works such as that of Laud & DeJong [25], Wiewiora [51], have outlined theoretical limitations of these potential-based shaping techniques. Other approaches to automate reward shaping such as population-based method [21] and automatic reward shaping [7, 19] have also been developed.

Other works have used the key ideas of reward shaping for multi-agent reward shaping [8, 46], belief shaping [**?**], ethics shaping [54], and meta learning rewards [57, 58]. Similar to this work, Jaderberg et al. [21] propose a population based optimization process for learning intrinsic rewards and achieve spectacular performance on Quake III.

REFERENCES

[1] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and Zico Kolter. Differentiable convex optimization layers. *arXiv preprint arXiv:1910.12430*, 2019.
[2] Brandon Amos and Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, 2017.
[3] John Asmuth, Michael L Littman, and Robert Zinkov. Potential-based shaping in model-based reinforcement learning. In *AAAI*, pp. 604–609, 2008.
[4] Sarah Bechtle, Yevgen Chebotar, Artem Molchanov, Ludovic Righetti, Franziska Meier, and Gaurav S Sukhatme. Meta-learning via learned loss. 2019.
[5] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.
[6] Bruce Christianson. Automatic hessians by reverse accumulation. *IMA Journal of Numerical Analysis*, 1992.
[7] Rati Devidze, Goran Radanovic, Parameswaran Kamalaruban, and Adish Singla. Explicable reward design for reinforcement learning agents. *Advances in Neural Information Processing Systems*, 34:20118–20131, 2021.
[8] Sam Devlin and Daniel Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'11)*, pp. 225–232, 2011.
[9] Sam Michael Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*, pp. 433–440, 2012.
[10] Asen L Dontchev and R Tyrrell Rockafellar. *Implicit functions and solution mappings*, volume 543. Springer, 2009.
[11] Marco Dorigo and Marco Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial intelligence*, 71(2):321–370, 1994.
[12] Yuqing Du, Olivia Watkins, Zihan Wang, Cdric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. *arXiv preprint arXiv:2302.06692*, 2023.
[13] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1165–1173. JMLR. org, 2017.
[14] Marek Grzes and Daniel Kudenko. Learning potential for reward shaping in reinforcement learning with tile coding. In *Proceedings of the AAMAS 2008 Workshop on Adaptive and Learning Agents and Multi-Agent Systems (ALAMAS-ALAg 2008)*, pp. 17–23, 2008.
[15] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, pp. 5302–5311, 2018.
[16] Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.
[17] Anna Harutyunyan, Sam Devlin, Peter Vrancx, and Ann Nowe. Expressing arbitrary reward functions as potential-based advice. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pp. 2652–2658, 2015.
[18] Mark K Ho, Fiery Cushman, Michael L Littman, and Joseph L Austerweil. People teach with rewards and punishments as communication, not reinforcements. *Journal of Experimental Psychology: General*, 148(3):520, 2019.
[19] Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 2020.
[20] Chen Huang, Shuangfei Zhai, Walter Talbott, Miguel Bautista Martin, Shih-Yu Sun, Carlos Guestrin, and Josh Susskind. Addressing the loss-metric mismatch with adaptive loss alignment. In *International Conference on Machine Learning*. PMLR, 2019.
[21] Max Jaderberg et al. Human-level performance in multiplayer games with population-based rl. 2019.
[22] Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.
[23] Steven George Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications.* Springer Science & Business Media, 2002.
[24] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, pp. 2140–2146, 2017.
[25] Adam Laud and Gerald DeJong. The influence of reward on the speed of reinforcement learning: An analysis of shaping. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, pp. 440–447, 2003.
[26] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10657–10665, 2019.
[27] Richard L Lewis, Satinder Singh, and Andrew G Barto. Where do rewards come from? In *Proceedings of the International Symposium on AI-Inspired Biology*, pp. 2601–2606, 2010.
[28] Hao Liu and Pieter Abbeel. Behavior from the void: Unsupervised active pre-training. *Advances in Neural Information Processing Systems*, 34:18459–18473, 2021.
[29] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pp. 1540–1552. PMLR, 2020.
[30] Ofir Marom and Benjamin Rosman. Belief reward shaping in reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
[31] Franziska Meier, Daniel Kappler, and Stefan Schaal. Online learning of a memory for learning rates. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2425–2432. IEEE, 2018.
[32] Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Guided meta-policy search. *arXiv preprint arXiv:1904.00956*, 2019.
[33] Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Learning unsupervised learning rules. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HkNDsiC9KQ.
[34] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, 1999.
[35] Scott Niekum, Andrew G Barto, and Lee Spector. Genetic programming for reward function search. *IEEE Transactions on Autonomous Mental Development*, 2010.
[36] Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *arXiv preprint arXiv:2007.08794*, 2020.
[37] OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018.
[38] Arvind Rajesvaran, Chelea Finn, Sham Kakade, and Sergey Levin. Meta-learning with implicit gradients. 2019.
[39] Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, pp. 463–471, 1998.
[40] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.
[41] Satinder Singh, Richard L Lewis, Andrew G Barto, and Jonathan Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.
[42] Satinder P. Singh, Andrew G. Barto, and Nuttapong Chentanez. Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems 17 (NIPS'04)*, pp. 1281–1288, 2004.
[43] Shihong Song, Jiayi Weng, Hang Su, Dong Yan, Haosheng Zou, and Jun Zhu. Playing fps games with environment-aware hierarchical reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (AAAI'19)*, pp. 3475–3482, 2019.
[44] Jonathan Sorg, Richard L Lewis, and Satinder Singh. Reward design via online gradient ascent. *Advances in Neural Information Processing Systems*, 2010.
[45] Jonathan Sorg, Satinder Singh, and Richard L Lewis. Optimal rewards versus leaf-evaluation heuristics in planning agents. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.
[46] Fan-Yun Sun, Yen-Yu Chang, Yueh-Hua Wu, and Shou-De Lin. Designing non-greedy reinforcement learning agents with diminishing reward shaping. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society (AIES 2018)*, pp. 297–302, 2018.
[47] Flood Sung, Li Zhang, Tao Xiang, Timothy Hospedales, and Yongxin Yang. Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*, 2017.
[48] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neurips*, 2000.
[49] Sebastian Thrun and Lorien Pratt. *Learning to learn.* Springer Science & Business Media, 2012.
[50] Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. *arXiv:1912.02175*, 2019.
[51] Eric Wiewiora. Potential-based shaping and q-value initialization are equivalent. *J. Artif. Intell. Res.*, 19:205–208, 2003.
[52] Eric Wiewiora, Garrison W Cottrell, and Charles Elkan. Principled methods for advising reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, pp. 792–799, 2003.
[53] Lijun Wu, Fei Tian, Yingce Xia, Yang Fan, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. Learning to teach with dynamic loss functions. *arXiv:1810.12081*, 2018.
[54] Yueh-Hua Wu and Shou-De Lin. A low-cost ethics shaping approach for designing reinforcement learning agents. In *Proceedings of the 32nd AAAI Conference on*

Artificial Intelligence (AAAI'18), pp. 1687–1694, 2018.

[55] Zhongen Xu, Hado Hasselt, Matteo Hessel, Junyuk Oh, Satinder Singh, and David Silver. Metagradient reinforcement learning with objective discovered online, 2020.

[56] Zhongwen Xu, Hado van Hasselt, and David Silver. Meta-gradient reinforcement learning. *arXiv preprint arXiv:1805.09801*, 2018.

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

[58] Haosheng Zou, Tongzheng Ren, Dong Yan, Hang Su, and Jun Zhu. Reward shaping via meta-learning. *arXiv preprint*, arXiv:1901.09330, 2019.

A IMPLICIT GRADIENT

Consider the following bi-level objective

$$\phi^* = \underset{\phi}{\operatorname{argmin}} L_{\text{Prim}}(\theta * (\phi), \phi) \text{ s.t. } \theta^*(\phi) = \underset{\theta}{\operatorname{argmin}} L_{\text{Aux}}(\theta, \phi) \tag{7}$$

Here we have explicitly added the dependence of θ due to the opimization process on ϕ. One approach to find the optimal ϕ^* is to find the partial derivative of $\mathcal{L}(f(\theta^*(\phi)), D_{\text{val}})$ with respect to the ϕ, and use gradient descent based optimization. The corresponding partial derivative is given by

$$\frac{d}{d\phi} L_{\text{Prim}}(\theta^*(\phi), \phi) = \underbrace{\frac{\partial}{\partial\theta}(L_{\text{Prim}}(\theta^*(\phi), \phi))|_{\theta^*}}_{I} \circ \underbrace{\frac{\partial}{\partial\phi}\theta^*}_{II} + \frac{\partial}{\partial\phi} L_{\text{Prim}}(\theta^*(\phi), \phi) \tag{8}$$

Considering that the inner optimization is finished we have direct access to $\theta^*(\phi)$ and the first term I in the previous equation can be computed directly. The second term is a more challenging to compute.

Implicit gradient method computes this gradient via differentiation of the optimality criteria of the inner optimization. The optimality criteria states that the gradient of the inner loss at the optima $\theta^*(\phi)$ is zero i.e.

$$\nabla_\theta L_{\text{Aux}}(\theta, \phi) = 0 \Rightarrow \frac{\partial}{\partial\theta}(L_{\text{Aux}}(\theta, \phi)) = 0 \tag{9}$$

By differentiating this with respect to ϕ one gets:

$$\frac{d}{d\phi}\frac{\partial}{\partial\theta} L_{\text{Aux}}(\theta, \phi) = 0 \Rightarrow \frac{\partial}{\partial\phi}\frac{\partial}{\partial\theta}(L_{\text{Aux}}(\theta, \phi)) + \frac{\partial}{\partial\theta}\frac{\partial}{\partial\theta}(L_{\text{Aux}}(\theta, \phi))\frac{\partial}{\partial\phi}\theta = 0 \tag{10}$$

$$\rightarrow \frac{\partial\theta}{\partial\phi} = -\left[\frac{\partial}{\partial\theta}\frac{\partial}{\partial\theta}(L_{\text{Aux}}(\theta, \phi))\right]^{-1}\left[\frac{\partial}{\partial\phi}\frac{\partial}{\partial\theta}(L_{\text{Aux}}(\theta, \phi))\right] \tag{11}$$

Putting this back in Equation 8 we get

$$\frac{d}{d\phi} L_{\text{Prim}}(\theta^*(\phi), \phi) = -\frac{\partial(L_{\text{Prim}}(\theta^*(\phi), \phi)}{\partial\theta}\left[\frac{\partial}{\partial\theta}\frac{\partial}{\partial\theta}(L_{\text{Aux}}(\theta, \phi))\right]^{-1}\left[\frac{\partial}{\partial\phi}\frac{\partial}{\partial\theta}(L_{\text{Aux}}(\theta, \phi))\right] + \frac{\partial(L_{\text{Prim}}(\theta^*(\phi), \phi))}{\partial\phi} \tag{12}$$

Remark 3. *Notice that in Equation 12, the inverse of second order derivative i.e. a Hessian needs to be computed which can be expensive. In practice, approximations via Conjugate Gradient, Shermon-Morrison Identity, diagonalized Hessian or von-Neumann Expansion could be used. For our structure prediction experiments we used the von-Neumann approximation while for reinforcement learning we used the diagonal Hessian.*

A.1 Approximation for inverse Hessian

von Neumann Approximation. The von Neumann series for inverse of matrix H is given by:

$$H^{-1} = \sum_{i=0}^{\infty}(I - H)^i \approx \sum_{i=0}^{1}(I - H)^i$$

This is convergent for matrices H with singular values less than 1. An approximation is obtained by truncating the series. While it is invalid for general matrices, this approximation has been shown useful when used in the context of gradient based methods [29]. Note that the first order approximation is linear in H and along with automatic differentiation methods allows easy and efficient multiplication with any vector by the Hessian-vector product method [6]. To compute the Hessian-vector product (HVP) with the vector v, one simply changes the parameters by ϵv (for some small ϵ) and computes the gradient. The difference between the two gradient when scaled equals the HVP. Furthermore this also holds when multiplying with the cross-Hessian $\partial_\theta\partial_\phi$, the same trick can be used once again. For further details refer to Christianson [6]

Diagonal Approximation. We use the following diagonal approximation for the matrix $H_\phi \approx \hat{H}_\theta = diag(h)$ and H_ϕ.

$$h = \mathbb{E}_{\mathcal{D}}\left[\sum_{t=0}^{T} \partial_\theta ln(\pi_\theta(S_t, A_t) \times \partial_\theta ln(\pi_\theta(S_t, A_t)\left(\sum_{j}^{T} \gamma^{j-t} r_\phi(S_t, A_t)\right)\right]$$

where $\times$ is the Hadamard product. Similarly H_ϕ is approximated as $diag(a)$ where:

$$a = \mathbb{E}_{\mathcal{D}}\left[\sum_{t=0}^{T} \partial_\theta ln(\pi_\theta(S_t, A_t) \times \left(\sum_{j}^{T} \gamma^{j-t} \partial_\phi r_\phi(S_t, A_t)\right)\right]$$

B REWARD LEARNING

Algorithm 2 Implicit Gradient for Reward Learning

```
Require: Training Data $\mathcal{D}_{\text{init}}$, Execution Window $T$

  $\mathcal{D} \leftarrow \mathcal{D}_{\text{init}}$
  Sample $\theta, \phi$ randomly
  $t = 0$
  $\hat{p}i_\theta \leftarrow \pi_\theta$
  while TRUE do
      $\pi_\theta \leftarrow \hat{\pi}_\theta$
      while $\pi_\theta$ not converged do
          Sample trajectories B from $\mathcal{D}$
          Compute policy gradient $g_\theta$ via Equation 2
          $\theta \leftarrow \theta - \eta g_\theta$
      end while
      $\hat{\pi}_\theta \leftarrow \pi_\theta$
      $\mathcal{D}_2 = \{\}$
      for T iterations do
          Generate trajectory $\tau$ by executing $\pi_\theta$
          $\mathcal{D}_2 \leftarrow \mathcal{D}_2 \cup \{\tau\}$
      end for
      $\mathcal{D} \leftarrow \mathcal{D}_2 \cup \mathcal{D}$
      Compute meta gradient $g_\phi$ from Equation 6
      $\phi \leftarrow \phi - \eta g_\phi$
      if $\phi$ converged and $\pi_\theta$ converged break
  end while
```