Direct Policy Gradients: Direct Optimization of Policies in Discrete Action Spaces

Guy Lorberbom
Technion

Chris J. Maddison
DeepMind

Nicolas Heess
DeepMind

Tamir Hazan
Technion

Daniel Tarlow
Google Research, Brain Team

Abstract

Direct optimization [25, 37] is an appealing framework that replaces integration with optimization of a random objective for approximating gradients in models with discrete random variables [21]. A* sampling [24] is a framework for optimizing such random objectives over large spaces. We show how to combine these techniques to yield a reinforcement learning algorithm that approximates a policy gradient by finding trajectories that optimize a random objective. We call the resulting algorithms *direct policy gradient* (DirPG) algorithms. A main benefit of DirPG algorithms is that they allow the insertion of domain knowledge in the form of upper bounds on return-to-go at training time, like is used in heuristic search, while still directly computing a policy gradient. We further analyze their properties, showing there are cases where DirPG has an exponentially larger probability of sampling informative gradients compared to REINFORCE. We also show that there is a built-in variance reduction technique and that a parameter that was previously viewed as a numerical approximation can be interpreted as controlling risk sensitivity. Empirically, we evaluate the effect of key degrees of freedom and show that the algorithm performs well in illustrative domains compared to baselines.

1 Introduction

Many problems in machine learning reduce to learning a probability distribution (or policy) over sequences of discrete actions so as to maximize a downstream utility function. Examples include generating text sequences to maximize a task-specific metric like BLEU and generating action sequences in reinforcement learning (RL) to maximize expected return. A main challenge is that evaluating the objective requires integrating over all possible sequences, which is intractable, and thus approximations like REINFORCE are needed [40] to learn these policies.

A line of work has emerged in recent years that allows replacing integration and sampling with optimization of noisy objective functions [29, 12, 38, 24, 21]. While this does not immediately remove the intractability of the integration problem, casting the problem in terms of optimization gives access to a different toolbox of ideas, which can provide new perspectives and methods for these hard problems. For example, Maddison et al. provide a new way of leveraging bounds from convex duality for use in sampling from continuous probability distributions [24]. Our aim in this work is the analog for reinforcement learning: we will replace the integral that is typically approximated by REINFORCE with an alternative that requires only optimization over a noisy objective function. The benefit is that this opens up techniques from heuristic search for use in reinforcement learning (e.g., variants of A* search) and provides an opportunity to express domain knowledge, all while retaining the conceptual simplicity that comes from optimizing a standard expected return objective function.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

The resulting algorithm is quite different from standard approaches to computing a policy gradient, but it estimates the same quantity up to one finite difference approximation. We provide a comprehensive analysis of the new algorithm from both theoretical and empirical perspectives. In total, this work provides a new perspective on computing a policy gradient and expands the toolbox of techniques and domain knowledge that can be used to tackle this fundamental problem.

2 Background

Reinforcement learning. We consider a standard problem of RL, in which an agent interacts with a Markov Decision Process (MDP) for a finite number of steps[1] and attempts to maximize its accumulated reward. At any time $t \geq 0$ the environment is in state $s_t \in \mathcal{S}$ in the given state space $\mathcal{S}$; there is a fixed initial state $s_0 \in \mathcal{S}$. At each time t the agent interacts with the environment by taking an action a_t from a finite set of actions $a_t \in \mathcal{A}$ according to a policy parameterized by $\theta \in \mathbb{R}^d$, $\pi_\theta(a_t \mid s_t)$. The environment follows a transition distribution $p(r_t, s_{t+1} \mid s_t, a_t)$ over rewards r_t and next states s_{t+1} given previous state s_t and action a_t. The agent interacts with the environment in this way for $T > 0$ steps generating a sequence of states $\boldsymbol{s} = (s_1, \ldots, s_T)$, actions $\boldsymbol{a} = (a_0, \ldots, a_{T-1})$, and rewards $\boldsymbol{r} = (r_0, \ldots, r_{T-1})$. This corresponds to the following generative model,

$$\begin{aligned} a_t &\sim \pi_\theta(\cdot \mid s_t) \text{ for } t \in \{0, \ldots, T-1\} \\ r_t, s_{t+1} &\sim p(\cdot, \cdot \mid a_t, s_t) \text{ for } t \in \{0, \ldots, T-1\} \end{aligned} \tag{1}$$

given $s_0 \in \mathcal{S}$. Taken together this defines the following joint distribution,

$$p_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{r}) = \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t)\, p(r_t, s_{t+1} \mid s_t, a_t). \tag{2}$$

The sum of rewards r_t over an interaction is called the return, and the goal of the agent is to maximize the expected return over its policy parameters, $\max_{\theta \in \mathbb{R}^d} \mathbb{E}_{\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{r} \sim p_\theta} \left[\sum_{t=0}^{T-1} r_t \right]$.

Policy gradients. Policy gradient algorithms are a family of methods for optimizing expected return by estimating gradients. A common variant is REINFORCE [40], which samples a trajectory $\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{r} \sim p_\theta$, computes the return $R = \sum_{t=0}^{T-1} r_t$, and then approximates the gradient as $R \cdot \nabla_\theta \log p_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{r})$.

Gumbel-max reparameterizations. A random variable $G \sim \mathrm{Gumbel}(m)$ is Gumbel-distributed with location m if $p(G \leq g) = \exp(-\exp(-g + m))$. The Gumbel-max trick is a way of casting sampling from a softmax as an argmax computation by using the fact that if $G(i)$ are drawn i.i.d. as $\mathrm{Gumbel}(m_i)$, then $i^* = \mathrm{argmax}_i\, G(i) \sim \exp(m_i) / \sum_{i'} \exp(m_{i'})$. Moreover, $G^* = \max_i G(i) \sim \mathrm{Gumbel}(\log \sum_{i'} \exp m_{i'})$ and i^* and G^* are independent random variables. See [10, 24, 23].

Direct optimization. Direct optimization [25, 37, 21] approximates gradients of a loss function over discrete configurations that are computed as the argmax of a (possibly noisy) underlying potential function. Following [21] and letting $\boldsymbol{a}$ be a discrete variable, f_θ be a scoring function, ϵ be an auxiliary variable, $G(\boldsymbol{a}) \sim \mathrm{Gumbel}(0)$ be independent Gumbel noise, and r be a negative loss function, the method is based on a *direct* objective $D_\theta(\boldsymbol{a}, G, \epsilon) = f_\theta(\boldsymbol{a}) + G(\boldsymbol{a}) + \epsilon \cdot r_{\boldsymbol{a}}$. The main result is that $\nabla_\theta \mathbb{E}_G \left[r_{\mathrm{argmax}_{\boldsymbol{a}} f_\theta(\boldsymbol{a}) + G(\boldsymbol{a})} \right] = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \mathbb{E}_G \left[\nabla_\theta f_\theta(\mathrm{argmax}_{\boldsymbol{a}} D_\theta(\boldsymbol{a}, G, \epsilon)) - \nabla_\theta f_\theta(\mathrm{argmax}_{\boldsymbol{a}} D_\theta(\boldsymbol{a}, G, 0)) \right]$.

3 Basic Algorithm, Motivating Example, and Summary of Results

To motivate the approach as simply as possible, we first present a minimal version of our new *Direct Policy Gradient (DirPG)* algorithm and an example where it has an exponentially larger probability of sampling an informative gradient compared to REINFORCE. In later sections we will handle the full complexity of RL, justify correctness, and describe how to efficiently compute the needed quantities.

DirPG utilizes optimization to find an informative gradient that improves the reward of its policy. In contrast, REINFORCE samples from its current policy. This inherent difference can allow DirPG

[1] Technically, everything in the paper works with an unbounded numbers of steps as long as trajectories terminate with probability 1, but we assume a maximum number of steps to simplify some parts of the exposition.

to find a policy gradient more efficiently than REINFORCE. In the following we formalize this difference by considering a simple environment where rewards $r_{\boldsymbol{a}}$ are a function of action sequences $\boldsymbol{a} \in \mathcal{A}^T$ for a large action space $|\mathcal{A}|^T$. Further suppose that we are in a sparse reward regime such that $r_{\boldsymbol{b}} = m > 0$ for one trajectory $\boldsymbol{b}$ and $r_{\boldsymbol{a}} = 0$ for all others. The REINFORCE gradient is $r_{\boldsymbol{a}} \nabla \log \pi_\theta(\boldsymbol{a})$ where $\boldsymbol{a} \sim \pi$. Since $r_{\boldsymbol{a}}$ is zero for most $\boldsymbol{a}$, k samples from a uniform policy π_θ (like would arise at the start of learning) will result in a nonzero gradient with probability roughly $\frac{k}{|\mathcal{A}|^T}$.

In this setting DirPG can be described as follows. Let $G(\boldsymbol{a}) \sim \mathrm{Gumbel}(0)$ be independent Gumbel noise for each trajectory $\boldsymbol{a}$ and ϵ a hyperparameter. There are two trajectories of interest:

$$\boldsymbol{a}_{opt} = \mathrm{argmax}_{\boldsymbol{a}} \left[\log \pi_\theta(\boldsymbol{a}) + G(\boldsymbol{a})\right] \tag{3}$$

$$\boldsymbol{a}_{dir} = \mathrm{argmax}_{\boldsymbol{a}} \left[\log \pi_\theta(\boldsymbol{a}) + G(\boldsymbol{a}) + \epsilon \cdot r_{\boldsymbol{a}}\right]. \tag{4}$$

The direct policy gradient is defined as

$$\nabla_\theta \, \mathbb{E}_{\boldsymbol{a} \sim \pi_\theta} \, r_{\boldsymbol{a}} \approx \frac{1}{\epsilon} \left[\nabla_\theta \log \pi_\theta(\boldsymbol{a}_{dir}) - \nabla_\theta \log \pi_\theta(\boldsymbol{a}_{opt})\right]. \tag{5}$$

A key benefit of DirPG is that domain knowledge may be inserted to guide a search for $\boldsymbol{a}_{dir}$. Suppose we have a powerful search heuristic that leads directly to the optimum. Then $\boldsymbol{a}_{dir}$ can be computed at the same cost as a single sample, and the total cost of an update requires only two samples (for $\boldsymbol{a}_{opt}$ and $\boldsymbol{a}_{dir}$), hence its computational complexity is equivalent to REINFORCE with $k = 2$. DirPG computes an informative (non-zero) gradient iff $\boldsymbol{a}_{dir} = \boldsymbol{b}$ and $\boldsymbol{a}_{opt} \neq \boldsymbol{b}$. The probability of $\boldsymbol{a}_{opt} \neq \boldsymbol{b}$ is large ($1 - \frac{1}{|\mathcal{A}|^T}$), so this mainly comes down to whether $\boldsymbol{a}_{dir} = \boldsymbol{b}$, which is equivalent to the event

$$\log \pi_\theta(\boldsymbol{b}) + G(\boldsymbol{b}) + \epsilon \cdot m > \max_{\boldsymbol{a} \neq \boldsymbol{b}} \left[\log \pi_\theta(\boldsymbol{a}) + G(\boldsymbol{a})\right]. \tag{6}$$

When π_θ is uniform, this simplifies to $\epsilon \cdot m + G(\boldsymbol{b}) > \max_{\boldsymbol{a} \neq \boldsymbol{b}} G(\boldsymbol{a})$, which has the form of sampling $\boldsymbol{b}$ via Gumbel-max. The RHS has distribution $\mathrm{Gumbel}(\log(|\mathcal{A}|^T - 1))$ and thus the probability of sampling $\boldsymbol{a}_{dir} = \boldsymbol{b}$ is $\frac{\exp(\epsilon \cdot m)}{\exp(\epsilon \cdot m) + \exp(\log(|\mathcal{A}|^T - 1))}$. If ϵ scales logarithmically with $|A|^T$, then DirPG has an exponentially higher chance than REINFORCE to sample an informative gradient in this example.

This example motivates DirPG and also raises a number of questions. In the remainder, we provide a comprehensive analysis of the new algorithm. Since the algorithm has many facets, we prioritize the following, leaving developing finely-tuned variants that outperform state of the art to future work:

Full complexity of RL. We show how to handle general stochastic environments (Sec. 4).

Correctness. We show that DirPG computes a policy gradient up to a one-dimensional finite difference approximation that leads to the appearance of ϵ (Sec. 4).

Utilizing existing heuristics. We assumed above that a perfect heuristic enables computing $\boldsymbol{a}_{dir}$ at the cost of a single rollout. This elides an important detail, which is that the heuristic must not only guide search to maximize return, it must also consider the $\log \pi_\theta + G$ terms in (4). By extending A* sampling, we show how to convert a heuristic over returns to a heuristic for computing $\boldsymbol{a}_{dir}$ (Sec. 5).

Approximate optimization. With imperfect heuristics, exactly computing $\boldsymbol{a}_{dir}$ can be intractable. We define a notion of *improvement* over $\boldsymbol{a}_{opt}$ and prove (in a restricted setting) that approximate optimization of $\boldsymbol{a}_{dir}$ still leads to learning an optimal policy (Appendix B).

Epsilon. Previous work on direct optimization [25] recognized that ϵ could be positive ("towards good") or negative ("away from bad") but did not provide a precise analysis of its impact. We provide a novel interpretation, deriving the objective optimized under different choices of ϵ and show there is a precise connection to risk-aware RL (Appendix A.3).

Variance Reduction. We show that DirPG "comes with its own variance reduction," by providing an interpretation of the $\nabla_\theta \log \pi_\theta(\boldsymbol{a}_{opt})$ term in (5) as a control variate (Appendix A.2).

Empirical analysis. We study all of the above in a set of carefully designed experiments that illustrate how to leverage the large literature on heuristic-guided search in specific domains, and the effect of key parameters like ϵ and the approximation of $\boldsymbol{a}_{dir}$ (Sec. 6).

4 Direct Policy Gradient

We start by formalizing the full DirPG algorithm in a general stochastic RL environment. Note that there are two places where stochasticity enters into (2): via the agent's policy in the $\pi_\theta(a_t \mid s_t)$ terms

and via the environment in the $p(r_t, s_{t+1} \mid s_t, a_t)$ terms. Given this factorization, we can separately reparameterize them. Once this is done, the direct optimization approach follows straightforwardly. A key requirement of the learning update is that we can explore multiple trajectories for a given realization of environment noise, so the method requires a simulator in order to compute a gradient. However, the result of learning is a standard policy that can be sampled from without any search or simulator, so, e.g., it would be feasible to use in sim-to-real settings.

Reparameterization. The learning rule for DirPG is based on search over trajectories and thus requires a simulator for computing a gradient. Beyond that, we do not want to restrict the environments, so we consider a very general reparameterization, which is simply that there is some source of randomness $\boldsymbol{S}$ that does not depend on $\boldsymbol{a}$ such that there is a deterministic function mapping $\boldsymbol{S}$ and a sequence of $(s_0, a_0, \ldots, s_t, a_t)$ to the next r_t, s_{t+1} pair. This implies, for example, that if $\boldsymbol{S}$ is held fixed and an agent performs the same sequence of actions, then the same environment transitions and rewards will be produced. We denote the state (reward) resulting from a sequence of actions $\boldsymbol{a}$ as $s_{\boldsymbol{a}}$ ($r_{\boldsymbol{a}}$). When clear from context, we omit the explicit dependence on $\boldsymbol{S}$ for brevity.

Now it becomes straightforward to define a *per-trajectory* Gumbel-max reparameterization. Let the total log probability that a policy assigns to a sequence of actions be

$$\Pi_\theta\left(\boldsymbol{a} \mid \boldsymbol{S}\right) = \prod_{t=0}^{T-1} \pi_\theta\left(a_t \mid s_{(a_0 \ldots a_{t-1})}\right), \tag{7}$$

and let $\Gamma(\boldsymbol{a}) \sim \mathrm{Gumbel}(0)$ for each trajectory $\boldsymbol{a}$. This yields a trajectory-level Gumbel-max trick:

$$G_\theta(\boldsymbol{a}; \Gamma, \boldsymbol{S}) = \log \Pi_\theta\left(\boldsymbol{a} \mid \boldsymbol{S}\right) + \Gamma(\boldsymbol{a}) \tag{8}$$

$$\boldsymbol{a}^* = \mathrm{argmax}_{\boldsymbol{a}}\, G_\theta(\boldsymbol{a}; \Gamma, \boldsymbol{S}). \tag{9}$$

G_θ are distributed as Gumbels with shifted locations and $\boldsymbol{a}^*$ is a sample from (7).

We emphasize that the reparameterization is equivalent to the standard RL formulation. Specifically, let $P(\boldsymbol{S})$ be the distribution over $\boldsymbol{S}$ resulting from different realizations of environment stochasticity and let the return of a trajectory $\boldsymbol{a}$ be $R(\boldsymbol{a}, \boldsymbol{S}) = \sum_{t=0}^{T-1} r_{(a_0, \ldots, a_{t-1})}$. Then

$$\mathbb{E}_{\boldsymbol{a},\boldsymbol{s},\boldsymbol{r}\sim p_\theta}\left[\sum_{t=0}^{T-1} r_t\right] \;=\; \mathbb{E}_{\boldsymbol{S}\sim P}\left[\mathbb{E}_{\boldsymbol{a}\sim\Pi_\theta(\cdot|\boldsymbol{S})}\left[R(\boldsymbol{a}, \boldsymbol{S})\right]\right] \;=\; \mathbb{E}_{\boldsymbol{S}\sim P,\Gamma}\left[R(\boldsymbol{a}^*, \boldsymbol{S})\right]. \tag{10}$$

Direct Policy Gradient. The above reparameterizations allow defining the general DirPG algorithm and showing its correctness. Define *direct objective* D_θ and *prediction generating function* f:

$$D_\theta(\boldsymbol{a}; \Gamma, \boldsymbol{S}, \epsilon) = G_\theta(\boldsymbol{a}; \Gamma, \boldsymbol{S}) + \epsilon R(\boldsymbol{a}, \boldsymbol{S}), \tag{11}$$

$$f(\theta, \epsilon) = \mathbb{E}_{\boldsymbol{S}\sim P,\Gamma}\left[\max_{\boldsymbol{a}}\left\{D_\theta(\boldsymbol{a}; \Gamma, \boldsymbol{S}, \epsilon)\right\}\right], \tag{12}$$

$$\boldsymbol{a}^*(\epsilon) = \mathrm{argmax}_{\boldsymbol{a}}\, D_\theta(\boldsymbol{a}; \Gamma, \boldsymbol{S}, \epsilon). \tag{13}$$

When clear from context, we drop the explicit dependence on noise terms $\boldsymbol{S}$ and Γ for brevity. Differentiating f with respect to ϵ and θ in either order and evaluating at $\epsilon = 0$ yields the same value because f is smooth [21] (or see [37] for an alternative proof):

$$\frac{\partial}{\partial\theta_i}\mathbb{E}\left[R(\boldsymbol{a}^*(0), \boldsymbol{S})\right] = \left.\frac{\partial^2 f(\theta, \epsilon)}{\partial\theta_i \partial\epsilon}\right|_{\epsilon=0} = \left.\frac{\partial^2 f(\theta_i, \epsilon)}{\partial\epsilon \partial\theta_i}\right|_{\epsilon=0} = \left.\frac{\partial}{\partial\epsilon}\mathbb{E}\left[\frac{\partial}{\partial\theta_i}\log\Pi_\theta\left(\boldsymbol{a}^*(\epsilon) \mid \boldsymbol{S}\right)\right]\right|_{\epsilon=0}. \tag{14}$$

A finite-difference approximation in ϵ of the RHS of (14) yields the direct policy gradient (DirPG):

$$\nabla_\theta\, \mathbb{E}_{\boldsymbol{a},\boldsymbol{s},\boldsymbol{r}\sim p_\theta}\left[\sum_{t=0}^{T-1} r_t\right] \approx \frac{1}{\epsilon}\mathbb{E}_{\boldsymbol{S}\sim P,\Gamma}\left[\nabla_\theta \log\Pi_\theta\left(\boldsymbol{a}^*(\epsilon) \mid \boldsymbol{S}\right) - \nabla_\theta \log\Pi_\theta\left(\boldsymbol{a}^*(0) \mid \boldsymbol{S}\right)\right]. \tag{15}$$

Following terminology of [25] we name $\boldsymbol{a}_{opt} = \boldsymbol{a}^*(0)$ as the optimum in Eq. 9, and $\boldsymbol{a}_{dir} = \boldsymbol{a}^*(\epsilon)$ as the trajectory that defines the update direction. Because the LHS of (14) is the gradient of the expected return, DirPG approaches the standard policy gradient as $\epsilon \to 0$.

Intuitively, (13) reduces to (9) when $\epsilon = 0$, so $\boldsymbol{a}_{opt}$ is a trajectory sampled from the current policy. $\boldsymbol{a}_{dir}$ is a trajectory that is close to a sample from the current policy but that has higher or lower return, where the strength and direction of this pull comes from the magnitude and sign of ϵ. The gradient increases the probability of the better trajectory and decreases the worse.

Algorithms. The general form of algorithms we consider is given in Algorithm 1. The basis is a $\mathrm{TrajectoryGenerator}$ (see Sec. 5) that produces a stream of pairs of trajectories $\boldsymbol{a}$ and associated direct objectives $D_\theta(\boldsymbol{a}; \epsilon)$. The first step of Algorithm 1 is to find $\boldsymbol{a}_{opt}$ and $d_{opt} = D_\theta(\boldsymbol{a}_{opt}; 0)$ and initialize $\boldsymbol{a}_{dir} = \boldsymbol{a}_{opt}, d_{dir} = d_{opt}$. The algorithms in Sec. 5 naturally produce $\boldsymbol{a}_{opt}$ and d_{opt} as the first result, so we assume that behavior. The algorithm then applies heuristic search to find a trajectory $\boldsymbol{a}_{dir}$ with direct objective d_{dir} better than d_{opt} (lines 5-13). If no improvement is found before a budget is exceeded, then $\boldsymbol{a}_{opt}$ is equal to $\boldsymbol{a}_{dir}$ and the result of line 14 is a zero gradient. Given enough budget and no early termination, the algorithm exactly implements (15). One variant is to terminate the search upon finding any improvement (line 9). This automatically adapts the search budget as training progresses. At first it is easy to improve over $\boldsymbol{a}_{opt}$ (a sample from a random policy), but more search is needed after training for longer. In Appendix B, we prove that this variant still learns an optimal policy (in a restricted setting).

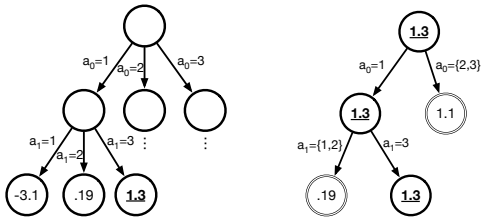
Algorithm 1 Direct Policy Gradient (General Form)

1: $\boldsymbol{S} \sim P(\boldsymbol{S})$
2: $\Gamma(\boldsymbol{a}) \sim \mathrm{Gumbel}(0)$ for all $\boldsymbol{a}$
3: $\mathrm{trajectories} = \mathrm{TrajectoryGenerator}(\boldsymbol{S}, \Gamma, \epsilon)$
4: $\boldsymbol{a}_{opt}, d_{opt} \leftarrow \boldsymbol{a}_{dir}, d_{dir} \leftarrow \mathrm{trajectories.next}()$
5: **while** budget not exceeded **do**
6: $\quad \boldsymbol{a}_{cur}, d_{cur} \leftarrow \mathrm{trajectories.next}()$
7: $\quad$ **if** $d_{cur} > d_{dir}$ **then**
8: $\quad\quad \boldsymbol{a}_{dir}, d_{dir} \leftarrow \boldsymbol{a}_{cur}, d_{cur}$
9: $\quad\quad$ **if** terminate on first improvement **then**
10: $\quad\quad\quad$ break
11: $\quad\quad$ **end if**
12: $\quad$ **end if**
13: **end while**
14: **return** $\frac{1}{\epsilon} \nabla_\theta \left[\log \pi_\theta\left(\boldsymbol{a}_{dir} \mid \boldsymbol{S}\right) - \log \pi_\theta\left(\boldsymbol{a}_{opt} \mid \boldsymbol{S}\right)\right]$

5 Generating trajectories using $A^\star$ sampling

$A^\star$ sampling provides a starting point for computing $\boldsymbol{a}_{opt}$ and $\boldsymbol{a}_{dir}$, but it is inefficient in its use of environment interactions. Here, we develop a new variant tailored to the RL setting that uses a lazier sampling strategy that minimizes the number of environment interactions. Despite $\boldsymbol{a}_{opt}$ being an argmax over $|\mathcal{A}|^T$ trajectories, the algorithm produces an exact solution in T steps. Computing $\boldsymbol{a}_{dir}$ is more challenging, but DirPG can leverage heuristics to guide the search, and it benefits relative to REINFORCE by actively searching for an informative gradient.

Search Space. The search over $\boldsymbol{S}$ for $\boldsymbol{a}_{opt}$ and $\boldsymbol{a}_{dir}$ is structured into a search tree over sets of action sequences that share a common prefix that we refer to as *regions*. Region $\mathcal{R}(\tilde{\boldsymbol{a}}, \mathcal{B}; \boldsymbol{S})$ is the set of trajectories that start with prefix $\tilde{\boldsymbol{a}} = (a_0, \ldots, a_{t-1})$ and then take a next action from $\mathcal{B} \subseteq \mathcal{A}$. The root region $\mathcal{R}(\varnothing, \mathcal{A})$ is the set of all trajectories. An example search tree is shown in Fig. 1 (b). The root (top) is the set of all trajectories and its right child is the set of trajectories $\{\boldsymbol{a} : a_0 \in \{2, 3\}\}$.



(a) Gumbels for trajectories (b) Gumbels for regions

Figure 1: Example search tree and associated values. **(a)** Gumbel values $G_\theta(\boldsymbol{a}; \Gamma, \boldsymbol{S})$ associated with each trajectory $\boldsymbol{a}$. The trajectory with maximum value (underlined) is $\boldsymbol{a}_{opt}$. **(b)** State of the search tree after sampling $\boldsymbol{a}_{opt}$. Nodes on the queue are drawn with double outline.

A search queue is initialized with the root region, and then the search tree is repeatedly expanded by choosing a region $\mathcal{R} = \mathcal{R}(\tilde{\boldsymbol{a}}, \mathcal{B}; \boldsymbol{S})$ from the queue and a next action $a_t \in \mathcal{B}$. $\mathcal{R}$ is split into two child regions. The first appends a_t to the prefix and allows any next action to follow; i.e., $\mathcal{R}_1 = \mathcal{R}(\tilde{\boldsymbol{a}} \oplus a_t, \mathcal{A})$ where $\oplus$ denotes concatenation. The second leaves the prefix unchanged and eliminates a_t as a possible next action; i.e., $\mathcal{R}_2 = \mathcal{R}(\tilde{\boldsymbol{a}}, \mathcal{B} \backslash \{a_t\})$. If $s_{\tilde{\boldsymbol{a}} \oplus a_t}$ is a terminal state then $\mathcal{R}_1$ contains a single trajectory and is not expanded further. If $\mathcal{B} \backslash \{a_t\}$ is empty, then $\mathcal{R}_2$ can be discarded. An interaction with the environment is generated only for the first new region, and the resulting state is stored so that it can be re-used by all other nodes sharing the same prefix. In Fig. 1 (b), the first split chose $a_0 = 1$ and created regions $\mathcal{R}_1 = \mathcal{R}((1), \mathcal{A})$ and $\mathcal{R}_2 = \mathcal{R}(\varnothing, \mathcal{A} \backslash \{1\})$.

Optimal completions. For any region $\mathcal{R}(\tilde{\boldsymbol{a}}, \mathcal{B}; \boldsymbol{S})$ popped from the queue, it is possible to optimally complete it with respect to G_θ without any backtracking in the search. That is, letting $\tilde{\boldsymbol{a}} = a_0, \ldots, a_t$, we can compute $\mathrm{argmax}_{a_{t+1}, \ldots, a_T | a_{t+1} \in \mathcal{B}} G_\theta(\tilde{\boldsymbol{a}} \oplus (a_{t+1}, \ldots, a_T); \Gamma, \boldsymbol{S})$ using only $T - t$ interactions with the environment. The key idea is to define random variables $G_\theta(\boldsymbol{a}; \Gamma, \boldsymbol{S})$ not only for full trajectories $\boldsymbol{a}$ but also for every region in the search tree. The random variable

for a region is assigned to be the max over G_θ of all trajectories in the region: $G_\theta(\mathcal{R}; \Gamma, \boldsymbol{S}) = \max_{\boldsymbol{a} \in \mathcal{R}} G_\theta(\boldsymbol{a}; \Gamma, \boldsymbol{S})$. Since the marginal distributions of the region random variables can be computed efficiently,[2] the top-down algorithm [24] can be applied to sample child region random variables conditional on the parents. By always following the search tree downwards towards the child with maximum $G_\theta(\mathcal{R}; \Gamma, \boldsymbol{S})$, we descend straight to the optimal completion. Notably, if we follow this strategy starting at the root region, we sample $\boldsymbol{a}_{opt}$ using only T environment interactions.

Top-down sampling of trajectories. Putting the above two sections together and simplifying expressions results in Algorithm 2, a new variant of top-down sampling. Note that the algorithm produces an endless stream of $(\boldsymbol{a}, D_\theta(\boldsymbol{a}))$ pairs (line 15) and does not specify the order in which nodes are popped from the queue (various choices are discussed below). The algorithm begins by sampling $G_\theta(\mathcal{R})$ for the root region $\mathcal{R}$ that contains all trajectories (line 4). Line 6 pops a node from the queue and line 7 samples the action a_t associated with the child region with maximum G_θ. Line 8 queries the environment for the s_{t+1} and r_t that result from taking a_t as the next action, and the result is stored until $\boldsymbol{S}$ is reset. Then regions are divided as described above (line 17 corresponds to $\mathcal{R}_1$; lines 9-13 correspond to $\mathcal{R}_2$), and upon creation of new regions, their G_θ values are sampled conditional upon the parent's G_θ value (lines 11, 17).

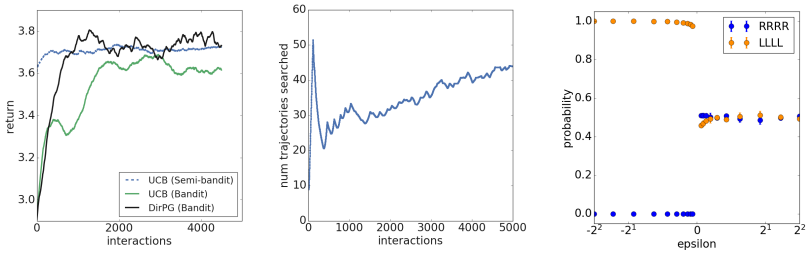
Algorithm 2 Top-Down Sampling $\boldsymbol{a}$

1: **In:** environment env, actions $\mathcal{A}$, ϵ.
2: **Out:** Stream of $(\boldsymbol{a}, D_\theta(\boldsymbol{a}))$ pairs.
3: $Q, \boldsymbol{S} \leftarrow$ Queue, StateRewardTree
4: $Q.\text{push}(\varnothing, \mathcal{A}, \text{Gumbel}(0))$
5: **while** Q is not empty **do**
6: $\quad \tilde{\boldsymbol{a}}, \mathcal{B}, G \leftarrow Q.\text{pop}()$
7: $\quad a \leftarrow$ Sample $\pi_\theta(a \mid s_{\tilde{\boldsymbol{a}}})\, 1\{a \in \mathcal{B}\}$
8: $\quad s_{\tilde{\boldsymbol{a}} \oplus a}, r_{\tilde{\boldsymbol{a}} \oplus a} \leftarrow env.\text{step}(a, s_{\tilde{\boldsymbol{a}}})$
9: $\quad$ **if** $\mathcal{B} \backslash \{a\}$ is not empty **then**
10: $\quad\quad \mu \leftarrow \log \Pi_\theta(\mathcal{R}(\tilde{\boldsymbol{a}}, \mathcal{B} \backslash \{a\}) \mid \boldsymbol{S})$
11: $\quad\quad G' \leftarrow \text{TruncGumbel}(\mu, G)$
12: $\quad\quad Q.\text{push}(\tilde{\boldsymbol{a}}, \mathcal{B} \backslash \{a\}, G')$
13: $\quad$ **end if**
14: $\quad$ **if** $s_{\tilde{\boldsymbol{a}} \oplus a}$ is terminal **then**
15: $\quad\quad$ **yield** $(\tilde{\boldsymbol{a}} \oplus a, G + \epsilon R(\tilde{\boldsymbol{a}} \oplus a, \boldsymbol{S}))$
16: $\quad$ **else**
17: $\quad\quad Q.\text{push}(\tilde{\boldsymbol{a}} \oplus a, \mathcal{A}, G)$
18: $\quad$ **end if**
19: **end while**

If Q is a priority queue with priority $G_\theta(\mathcal{R})$, then the algorithm will yield pairs in descending order of $G_\theta(\boldsymbol{a})$, which also means that $\boldsymbol{a}_{opt}$ will be found after T node expansions. We assume regions are prioritized this way until the first yield so that line 4 in Algorithm 1 produces $(\boldsymbol{a}_{opt}, D_\theta(\boldsymbol{a}_{opt}; \epsilon))$. We are then free to change the priority function as in the next subsection and reorder the queue. However if we do not, then this can generate "Gumbel Top-K" [17] by running Algorithm 2 with priority $G_\theta(\mathcal{R})$ and return the first K results. Algorithm 2 is better for RL than other $A^\star$ sampling algorithms [24, 15], because the others would roll-out an entire trajectory for each region expanded and thus make inefficient use of interactions with the environment. We expand on these details in Appendix C.

Searching for large D_θ using A* sampling. The final algorithm prioritizes regions on the queue using the return achieved so far and (if available) an upper bound on the return-to-go. It is the same as Algorithm 2, except before pushing a region on the queue (lines 4, 12, 17), we compute a priority for a region based on all the terms in (11). Let $L(\mathcal{R}) = \sum_{t'=0}^{t-1} r_{(a_0, \ldots, a_{t'-1})}$ be the reward accumulated so far by the prefix and $U(\mathcal{R}) \geq \sum_{t'=t}^{T} r_{(a_0, \ldots, a_{t'-1})}$ be an upper bound on the return-to-go for any trajectory in region $\mathcal{R}$. Recall the $G_\theta(\mathcal{R})$ computed during the search is the maximum G_θ for any trajectory in the region. We can then upper bound $D_\theta(\mathcal{R}; \epsilon) = \max_{\boldsymbol{a} \in \mathcal{R}} D_\theta(\boldsymbol{a}; \epsilon) \leq G_\theta(\mathcal{R}) + \epsilon \cdot (L(\mathcal{R}) + U(\mathcal{R}))$. We can also prune regions from the search if their upper bound is worse than $D_\theta(\boldsymbol{a}; \epsilon)$ for the best $\boldsymbol{a}$ found so far. Using the upper bound as a priority yields a stochastic version of A* search (i.e., it is A* Sampling). In practice, there is a large literature on heuristic search methods that relax optimality guarantees of $A^\star$ search in order to arrive at good solutions faster (see, e.g., [30, 11]). We have found benefit to adapting these methods to the search for $\boldsymbol{a}_{dir}$. In particular, we adapt static weighted $A^\star$ search [32] to our setting by modifying the priority to be $G_\theta(\mathcal{R}) + \epsilon \cdot (L(\mathcal{R}) + \alpha U(\mathcal{R}))$ for $0 \leq \alpha < 1$, though we expect other methods to also be fruitful.

[2] By properties of Gumbel distributions, the marginals are $G_\theta(\mathcal{R}; \Gamma, \boldsymbol{S}) \sim \text{Gumbel}(\log \Pi_\theta(\mathcal{R} \mid \boldsymbol{S}))$ where $\Pi_\theta(\mathcal{R} \mid \boldsymbol{S}) = \sum_{\boldsymbol{a} \in \mathcal{R}} \Pi_\theta(\boldsymbol{a} \mid \boldsymbol{S})$. It can efficiently be computed by pushing the sum inwards through the shared prefix: $\Pi_\theta(\mathcal{R}(\tilde{\boldsymbol{a}}, \mathcal{B}; \boldsymbol{S}) \mid \boldsymbol{S}) = \prod_{t'=0}^{t-1} \pi_\theta\left(a_{t'} \mid s_{(a_0, \ldots, a_{t'-1})}\right) \sum_{a \in \mathcal{B}} \pi_\theta(a \mid s_{\tilde{\boldsymbol{a}}})$.



(a) (b) (c)

Figure 2: Bandits and risk sensitivity. (a) Average return vs # of interactions. (b) Number of steps needed to find $\boldsymbol{a}_{dir}$. (c) DeepSea results showing learned Π(LLLL) (safe) and Π(RRRR) (risky) vs ϵ.

6 Experiments

Combinatorial Bandits. We experiment with combinatorial bandits and compare DirPG to Upper Confidence Bound (UCB) algorithms [2, 5]. The environment is defined by a graph $G = (V, E)$ where $V = \{1, \ldots, n\}$ is the set of nodes and $E \subseteq V \times V$ is the set of undirected edges. For each edge $e \in E$ a parameter μ_e determines per-edge rewards as $r_e \sim \text{Uniform}(0, 2\mu_e)$. An agent queries the environment with tree $\mathcal{T}$ and receives reward $r_{\mathcal{T}} = \sum_{e \in \mathcal{T}} r_e$. Fresh realizations of r_e are drawn for each episode. UCB algorithms end an episode after a single interaction, while DirPG uses multiple interactions per episode (at the cost of seeing fewer realizations). We compare to a "semi-bandit" version of UCB that observes more information (per-edge contributions to rewards) and a "full bandit" version that receives the same observations as DirPG, the total reward $r_{\mathcal{T}}$ after producing a full tree. Note the similarity of the full bandit version to, e.g., CUCB [7].

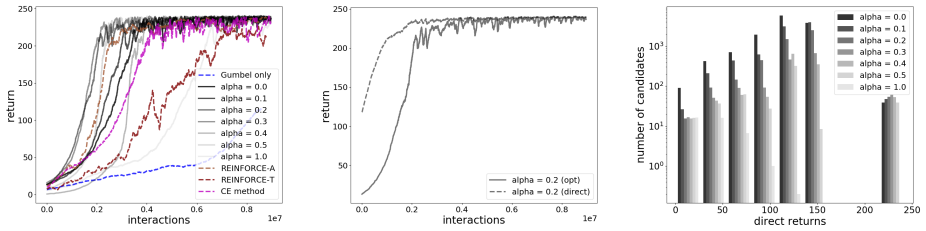
To apply DirPG, we let $\boldsymbol{a}$ be a sequence of $|E|$ binary decisions of whether to include each edge in the spanning tree. Learnable parameters θ_e determine the probability of inclusion via $\sigma(\theta_e)$ where σ is the sigmoid function. The environment presents a legal set of actions at each step (see Appendix D.1 for details). To compute $\boldsymbol{a}_{dir}$, we give a budget of 100 interactions and use priority $G_\theta(\boldsymbol{a})$ in the search, enabling the early termination option in Algorithm 1. Results appear in Fig. 2 (a), which shows the moving average return versus number of interactions, averaged over 10 runs. The DirPG curve is for samples of $\boldsymbol{a}_{opt}$, which is noisier due to there being fewer realizations. DirPG is competitive with a UCB variant using more information, and it outperforms the comparable variant. Fig. 2 (b) shows the number of steps taken to find an improvement. Aside from initial noise due to the moving average, the number of interactions used in the search automatically grows as learning progresses.

DeepSea. Previously ϵ was considered a nuisance parameter, but we show that it controls an agent's preference for risk-seeking (positive ϵ) versus risk-avoiding (negative ϵ) behavior. Analysis making this claim precise and a further experiment appears in Appendix A.3.

We use an adaptation of the DeepSea environment that was used by [28] to study risk sensitivity. The environment is a 5x5 grid where the agent starts from the top-left cell and the goal is in the bottom-right. The agent has a choice of left (L) or right (R) at each step. If the agent chooses L, it gets 0 reward and moves down and left. If it chooses R, it gets a reward sampled from $\mathcal{N}(1, 1)$ if transitioning to the bottom-right corner and otherwise $-\frac{1}{3}$. This is interesting because any policy that is a mixture of LLLL and RRRR has optimal return (mixture of 0, $\mathcal{N}(0, 1)$ respectively), but the policies have different variance and thus we expect the choice of ϵ to affect what the agent learns.

In Fig. 2 (c) we train policies with a range of ϵ values for $400,000$ episodes to ensure convergence and plot the probability assigned to trajectories LLLL and RRRR in the learned policy. For $\epsilon < 0$, most mass is put on LLLL, which has no variance and is thus favorable to a risk-avoiding agent. For $\epsilon > 0$, mass is split evenly, which has highest "controllable risk" (see Appendix A.3).

MiniGrid. In our final experiments we use the **MiniGrid-MultiRoom-N6-v0** environment [8] to study how to prioritize nodes within the search for $\boldsymbol{a}_{dir}$. MiniGrid is a partially observable grid-world where the agent observes an egocentric 7×7 grid around its current location and has the choice of 7 actions including moving right, left, forward, or toggling doors. We use environments of 25×25 grids with a series of 6 connected rooms separated by doors that need to be opened. Intermediate rewards are given for opening doors and reaching a final goal state. As baselines we compare to



(a) (b) (c)

Figure 3: Minigrid results. (a) Return vs number of interactions. (b) Direct objective of $\boldsymbol{a}_{dir}$ and $\boldsymbol{a}_{opt}$ vs iteration. (c) Histograms showing quality-vs-quantity tradeoff for various search priorities.

REINFORCE and the cross entropy method. In all of the methods we utilized the simulator to reset the environment so that multiple trajectories could be sampled starting from the same environment seed. In all cases, we use a total of 3000 interactions per environment seed (episode). In our method, we use 100 interactions to sample $\boldsymbol{a}_{opt}$ (the trajectory length) and 2900 interactions to search for $\boldsymbol{a}_{dir}$. In REINFORCE and in the cross entropy method we sample 30 independent trajectories, where each is 100 interactions long. Details on their implementation are in Appendix D.3.

We explore variations on how to set the priority of nodes in the search for $\boldsymbol{a}_{dir}$. First, in the "Gumbel only" priority, we use just $G_\theta(\mathcal{R})$ as a region's priority. In the others, we use $G_\theta(\mathcal{R}; \boldsymbol{S}, g) + \epsilon(L(\mathcal{R}) + \alpha U(\mathcal{R}))$, where U is based on the Manhattan distance to the goal and the number of unopened doors. Setting $\alpha = 0$ trades off enumerating by descending order of $G_\theta(\mathcal{R}; \boldsymbol{S}, g)$ with favoring prefixes that have already achieved high return. Setting $\alpha = 1$ yields A* search. Fig. 3 (a) shows average return versus training episode. $\alpha = 0$ provides good results, and increasing α up to $\alpha = 0.3$ gives improved performance. Beyond that, performance degrades, with $\alpha = 1$ performing worst.

To better understand this, we partially trained a model for 1.2M interactions and then froze the parameters and ran several searches for the same number of interactions but with different priority functions. Fig. 3 (c) shows the results. For smaller α, more trajectories are finished to completion but the returns achieved are worse. As α increases, fewer full trajectories are found but they have better returns, but past $\alpha = 0.4$ not enough full trajectories are found, and both the quality and the quantity shrink. Thus, setting α too high leads to "breadth-first behavior" where too much time is spent exploring prefixes and not completing trajectories. In Fig. 3 (b), we show the relationship between $D_\theta(\boldsymbol{a}_{dir})$ and $D_\theta(\boldsymbol{a}_{opt})$ over the course of learning. This shows that $\boldsymbol{a}_{dir}$ does not need to find a trajectory with the optimal return in order to provide signal for the policy to improve.

7 Related Work

Similarities can be drawn to the body of work casting RL as probabilistic inference, in particular in Expectation-Maximization (EM) Policy Search methods [31, 39, 34, 20, 19, 26, 6, 1, 4]. Broadly, these methods alternate a step akin to posterior inference that improves a trajectory distribution with an update to the policy parameters using in an EM formulation. In this context our work would be most similar to an incremental variant [27] of Monte Carlo EM [18], though DirPG has significant differences, including the use of A* sampling to guide the sampling and the use of direct optimization, which can be interpreted as a variance reduction strategy. We discuss this in detail in Appendix A.

The initial DirPG reparameterization is similar to [13], but the setting and approach are very different. The most prominent example of search in RL is Monte Carlo Tree Search (MCTS) [16, 3]. MCTS is quite different because—unlike DirPG—it uses search and a simulator at test time, but it becomes closer when search results are distilled into a policy as in [36]. However, we are not aware of results showing that MCTS can be used to directly compute a policy gradient. One can imagine an MCTS-style algorithm that explores k trajectories under a fixed realization of environment noise and chooses the one with highest return, then distills into a policy via a gradient update to increase the probability assigned to the chosen trajectory. As $k \to \infty$, this will approach the optimal DirPG update with $\epsilon \to \infty$. Bbased on risk sensitivity results in Appendix A.3, we can see that this algorithm will be very risk-seeking. Thus, DirPG offers a degree of control via ϵ that isn't available to this MCTS-style counterpart.

Another related use of search trees is the *vine* method from [35], which leverages a simulator's ability to reset to previous states to construct a tree over trajectories. Multiple roll-outs are created from tree nodes, and common random numbers are used across the roll-outs to reduce variance.

8 Discussion

We have presented a new method for computing a policy gradient and studied its properties from theoretical and empirical perspectives. This also provides new understandings of direct loss optimization in terms of variance reduction and risk-sensitivity. One limitation is that in its current form, the algorithm only learns in an episodic framework and from complete trajectories. We are currently exploring how this limitation could be removed. Our experiments so far have been geared towards understanding the algorithm and its important degrees of freedom. We are eager to take these learnings and apply them to real-world applications where search and heuristics (upper bounds) have traditionally been successful like navigation, combinatorial optimization, and program synthesis.

Broader Impact

This work presents a general theoretical and algorithmic contribution to reinforcement learning (RL) research. One contribution (Appendix A.3) is an analysis of the risk-sensitive behavior of the algorithm as parameter ϵ is varied. This provides an axis of control beyond simply maximizing expected future reward, which is likely a beneficial analysis to perform (though far-removed from well-defined impacts). We'll refrain from commenting on the future societal consequences of general advances in RL research, because this work is more theoretical and conceptual in nature, and it is a complex topic that is better covered in the context of work that is closer to specific impacts.

Acknowledgments and Disclosure of Funding

References

[1] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Rémi Munos, Nicolas Heess, and Martin A. Riedmiller. Maximum a posteriori policy optimisation. *CoRR*, abs/1806.06920, 2018.

[2] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[3] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[4] Lars Buesing, Theophane Weber, Yori Zwols, Sebastien Racaniere, Arthur Guez, Jean-Baptiste Lespiau, and Nicolas Heess. Woulda, coulda, shoulda: Counterfactually-guided policy search. *arXiv preprint arXiv:1811.06272*, 2018.

[5] Nicolo Cesa-Bianchi and Gábor Lugosi. Combinatorial bandits. *Journal of Computer and System Sciences*, 78(5):1404–1422, 2012.

[6] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path integral guided policy search. *CoRR*, abs/1610.00529, 2016.

[7] Wei Chen, Yajun Wang, and Yang Yuan. Combinatorial multi-armed bandit: General framework and applications. In *International Conference on Machine Learning*, pages 151–159, 2013.

[8] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019.

[9] Stefano Coraluppi. *Optimal control of Markov decision processes for performance and robustness*. PhD thesis, University of Maryland, 1997.

[10] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1954.

[11] Eric A Hansen and Rong Zhou. Anytime heuristic search. *Journal of Artificial Intelligence Research*, 28:267–297, 2007.

[12] Tamir Hazan and Tommi Jaakkola. On the partition function and random maximum a-posteriori perturbations. In *International Conference on Machine Learning*, 2012.

[13] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[14] Ronald A Howard and James E Matheson. Risk-sensitive markov decision processes. *Management science*, 18(7):356–369, 1972.

[15] Carolyn Kim, Ashish Sabharwal, and Stefano Ermon. Exact sampling with integer linear programs and random perturbations. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[16] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[17] Wouter Kool, Herke van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. *arXiv preprint arXiv:1903.06059*, 2019.

[18] Richard A Levine and George Casella. Implementations of the monte carlo em algorithm. *Journal of Computational and Graphical Statistics*, 10(3):422–439, 2001.

[19] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[20] Sergey Levine and Vladlen Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems*, pages 207–215, 2013.

[21] Guy Lorberbom, Andreea Gane, Tommi Jaakkola, and Tamir Hazan. Direct Optimization through $\arg\max$ for Discrete Variational Auto-Encoder. *arXiv e-prints*, page arXiv:1806.02867, Jun 2018.

[22] Chris J. Maddison, Dieterich Lawson, George Tucker, Nicolas Heess, Arnaud Doucet, Andriy Mnih, and Yee Whye Teh. Particle value functions. *arXiv preprint arXiv:1703.05820*, 2017.

[23] Chris J. Maddison and Daniel Tarlow. Gumbel machinery. `https://cmaddis.github.io/gumbel-machinery`. Accessed: 2019-05-21.

[24] Chris J. Maddison, Daniel Tarlow, and Tom Minka. A* Sampling. In *Advances in Neural Information Processing Systems 27*, 2014.

[25] David A McAllester, Tamir Hazan, and Joseph Keshet. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems*, pages 1594–1602, 2010.

[26] William Montgomery and Sergey Levine. Guided policy search as approximate mirror descent. *CoRR*, abs/1607.04614, 2016.

[27] Radford M Neal and Geoffrey E Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

[28] Brendan O'Donoghue. Variational bayesian reinforcement learning with regret bounds. *arXiv preprint arXiv:1807.09647*, 2018.

[29] G. Papandreou and A. Yuille. Perturb-and-MAP Random Fields: Using Discrete Optimization to Learn and Sample from Energy Models. In *International Conference on Computer Vision*, 2011.

[30] Judea Pearl. Heuristic search theory: Survey of recent results. In *IJCAI*, 1981.

[31] Jan Peters, Katharina Mülling, and Yasemin Altün. Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[32] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4):193–204, 1970.

[33] John W Pratt. Risk aversion in the small and in the large. *Econometrica*, 32(1/2):122–136, 1964.

[34] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *(R:SS 2012)*, 2012. *Runner Up Best Paper Award*.

[35] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[36] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[37] Yang Song, Alexander Schwing, Raquel Urtasun, et al. Training deep neural networks via direct loss minimization. In *International Conference on Machine Learning*, pages 2169–2177, 2016.

[38] Daniel Tarlow, Ryan Adams, and Richard Zemel. Randomized optimum models for structured prediction. In Neil D. Lawrence and Mark Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 1221–1229, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

[39] M. Toussaint and A.J. Storkey. Probabilistic inference for solving discrete and continuous state Markov Decision Processes. In *Proceedings of 23nd International Conference on Machine Learning (ICML 2006)*, 2006.

[40] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.