Differentiable Greedy Algorithm for Monotone Submodular Maximization: Guarantees, Gradient Estimators, and Applications

Shinsaku Sakaue
The University of Tokyo
sakaue@mist.i.u-tokyo.ac.jp

Abstract

Motivated by, e.g., sensitivity analysis and end-to-end learning, the demand for differentiable optimization algorithms has been increasing. This paper presents a theoretically guaranteed differentiable greedy algorithm for monotone submodular function maximization. We smooth the greedy algorithm via randomization, and prove that it almost recovers original approximation guarantees in expectation for the cases of cardinality and κ-extendible system constraints. We then present how to efficiently compute gradient estimators of any expected output-dependent quantities. We demonstrate the usefulness of our method by instantiating it for various applications.

1 INTRODUCTION

Submodular function maximization is ubiquitous in practice. In many situations such as influence maximization (Alon et al., 2012) and data summarization (Mirzasoleiman et al., 2016), submodular functions $f(\cdot, \boldsymbol{\theta}) : 2^V \to \mathbb{R}$, where V is a finite set ($n := |V|$), are modeled with continuous-valued parameter $\boldsymbol{\theta} \in \Theta$. For example, f representing influence spread has link probability parameter $\boldsymbol{\theta}$. In this paper, we consider the following parametric submodular maximization:

$$\underset{X \subseteq V}{\text{maximize}} \quad f(X, \boldsymbol{\theta}) \qquad \text{subject to} \quad X \in \mathcal{I}, \qquad (1)$$

where $\mathcal{I} \subseteq 2^V$ consists of all feasible solutions. In what follows, we assume $(V, \mathcal{I})$ to be a κ-extendible system and $f(\cdot, \boldsymbol{\theta})$ to be normalized, monotone, and submodular for any $\boldsymbol{\theta} \in \Theta$.

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130. Copyright 2021 by the author(s).

Once $\boldsymbol{\theta}$ is fixed, we often apply the greedy algorithm to problem (1) due to its theoretical strength (Nemhauser et al., 1978; Fisher et al., 1978) and high empirical performances. However, if $\boldsymbol{\theta}$ largely deviates from unknown true $\tilde{\boldsymbol{\theta}}$, the greedy algorithm may return a poor solution to the problem of maximizing $f(\cdot, \tilde{\boldsymbol{\theta}})$. This motivates us to study how changes in $\boldsymbol{\theta}$ values affect outputs of the greedy algorithm. Furthermore, it is desirable if we can learn $\boldsymbol{\theta}$ from data so that the greedy algorithm can achieve high $f(\cdot, \tilde{\boldsymbol{\theta}})$ values.

A major approach to studying such subjects is to differentiate outputs of algorithms w.r.t. $\boldsymbol{\theta}$. Regarding continuous optimization algorithms, this approach has been widely studied in the field of sensitivity analysis (Rockafellar and Wets, 1998; Gal and Greenberg, 2012), and is used by recent decision-focused (or end-to-end) learning methods (Donti et al., 2017; Wilder et al., 2019a), which learn to predict $\boldsymbol{\theta}$ based on outputs of optimization algorithms. When it comes to the greedy algorithm for submodular maximization, however, its outputs are generally non-differentiable and, even at differentiable points $\boldsymbol{\theta}$, the derivatives are constant zero. Hence, we need some smoothing techniques for using the well-established differentiation-based methods.

Tschiatschek et al. (2018) opened the field of differentiable submodular maximization; they proposed greedy-based differentiable learning methods for monotone and non-monotone submodular functions. Their algorithm for monotone objective functions was obtained by replacing non-differentiable argmax with differentiable softmax. Since then, this field has been attracting increasing attention; another softmax-based algorithm that forms a neural network (NN) (Powers et al., 2018) and applications (Kalyan et al., 2019; Peyrard, 2019) have been studied. However, this field is still in its infancy and the following two problems remain open:

Can we (1) smooth the greedy algorithm without losing its theoretical guarantees and (2) efficiently compute derivatives in an application-agnostic manner?

The first problem is important since, without the guarantees, we cannot ensure that the differentiation-based methods work well. The existing studies (Tschiatschek et al., 2018; Powers et al., 2018) state that the $(1-1/\mathrm{e})$-approximation for the cardinality constrained case is obtained if the temperature of softmax is zero (i.e., equal to argmax). This, however, provides no theoretical guarantees for the smoothed differentiable algorithms.

As regards the second problem, the existing methods (Tschiatschek et al., 2018; Powers et al., 2018) focus on differentiating some functions defined with subsets $X_1, X_2, \ldots \subseteq V$ given as training data. This restricts the scope of application; for example, we cannot apply them to sensitivity analysis as detailed in Appendix A. The efficiency also matters when computing derivatives; in (Tschiatschek et al., 2018), a heuristic approximation method is used since the computation of exact derivatives generally incurs exponential costs in n.

Our contribution is a theoretically guaranteed versatile framework that resolves the two problems, thus greatly advancing the field of differentiable submodular maximization. Below we present the details.

Smoothed Greedy We consider a stochastically perturbed version of the greedy algorithm, called Smoothed Greedy, which generalizes the existing algorithms (Tschiatschek et al., 2018; Powers et al., 2018). We prove that Smoothed Greedy achieves almost $(1-1/\mathrm{e})$- and $\frac{1}{\kappa+1}$-approximation guarantees in expectation for the cases of cardinality and κ-extendible system constraints, respectively, where a subtractive term depending on the perturbation strength affects the guarantees.

Gradient estimation Owing to the perturbation, we can differentiate expected outputs of Smoothed Greedy. However, the computation cost is exponential in n as with (Tschiatschek et al., 2018). To avoid this, we show how to estimate derivatives of any expected output-dependent quantities by sampling Smoothed Greedy outputs. Our estimator is unbiased, and can optionally be biased for reducing its variance. Experiments reveal how the perturbation strength affects estimator's variance.

Applications We demonstrate that our framework can serve as a bridge between the greedy algorithm and differentiation-based methods in many applications. When used for sensitivity analysis, it elucidates how outputs of Smoothed Greedy can be affected by changes in $\boldsymbol{\theta}$ values. Results of decision-focused learning experiments suggest that our greedy-based approach can be a simple and effective alternative to a recent continuous relaxation method (Wilder et al., 2019a).

1.1 Related Work

Nemhauser et al. (1978) proved the well-known $(1-1/\mathrm{e})$-approximation guarantee of the greedy algorithm for the cardinality constrained case, and this result is known to be optimal (Nemhauser and Wolsey, 1978; Feige, 1998). Fisher et al. (1978) proved that the greedy algorithm achieves the $\frac{1}{\kappa+1}$-approximation if $(V, \mathcal{I})$ is an intersection of κ matroids; later, this result was extended to the class of κ-systems (Calinescu et al., 2011), which includes κ-extendible systems.

Differentiable greedy submodular maximization is studied in (Tschiatschek et al., 2018; Powers et al., 2018). Our work is different from them in terms of theoretical guarantees, differentiation methods, and problem settings as explained above (see, also Appendix A). The closest to our result is perhaps that of the continuous relaxation method (Wilder et al., 2019a). Specifically, they use the multilinear extension (Calinescu et al., 2011) of $f(\cdot, \boldsymbol{\theta})$ and differentiate its local optimum computed with the stochastic gradient ascent method (SGA) (Hassani et al., 2017), which achieves a 1/2-approximation. Their method can be used for matroid constraints, but their analysis focuses on the cardinality constrained case. Compared with this, our method is advantageous in terms of approximation ratios and empirical performances (see, Section 5.2).

Differentiable learning methods have been studied in other situations: submodular minimization (Djolonga and Krause, 2017), quadratic programming (Amos and Kolter, 2017), structured prediction (Mensch and Blondel, 2018), mixed integer programming (Ferber et al., 2020), optimization on graphs (Wilder et al., 2019b), combinatorial linear optimization (Pogančić et al., 2020), satisfiability (SAT) instances (Wang et al., 2019), and ranking/sorting (Cuturi et al., 2019).

Perturbation-based smoothing is used for, e.g., online learning (Abernethy et al., 2016), linear contextual bandit (Kannan et al., 2018), linear optimization (Berthet et al., 2020), and sampling from discrete distributions (Gumbel, 1954; Jang et al., 2017; Maddison et al., 2017), but it has not been theoretically studied for smoothing the greedy algorithm for submodular maximization.

1.2 Notation and Definition

For any $f : 2^V \to \mathbb{R}$, we define $f_X(Y) \coloneqq f(X \cup Y) - f(X)$. We say f is normalized if $f(\emptyset) = 0$, monotone if $X \subseteq Y$ implies $f(X) \leq f(Y)$, and submodular if $f_X(v) \geq f_Y(v)$ for all $X \subseteq Y$ and $v \notin Y$. In this paper, we assume the objective function, $f(\cdot, \boldsymbol{\theta})$, to be normalized, monotone, and submodular for any $\boldsymbol{\theta} \in \Theta$. Note that this is the case with many set functions, e.g.,

Algorithm 1 Smoothed Greedy

```
1: $S \leftarrow \emptyset$
2: for $k = 1, 2 \dots$ do
3:     $U_k = \{u_1, \dots, u_{n_k}\} \leftarrow \{v \notin S \mid S \cup \{v\} \in \mathcal{I}\}$
4:     $\mathbf{g}_k(\boldsymbol{\theta}) \leftarrow (f_S(u_1, \boldsymbol{\theta}), \dots, f_S(u_{n_k}, \boldsymbol{\theta}))$
5:     $\mathbf{p}_k(\boldsymbol{\theta}) \leftarrow \mathrm{argmax}_{\mathbf{p} \in \Delta^{n_k}} \{\langle \mathbf{g}_k(\boldsymbol{\theta}), \mathbf{p} \rangle - \Omega_k(\mathbf{p})\}$
6:     $s_k \leftarrow u \in U_k$ with probability $p_k(u, \boldsymbol{\theta})$
7:     $S \leftarrow S \cup \{s_k\}$
8:     if $S$ is maximal then
9:         return $S$
```

weighted coverage functions with non-negative weights $\boldsymbol{\theta}$, probabilistic coverage functions with probabilities $\boldsymbol{\theta}$, and deep submodular functions (Dolhansky and Bilmes, 2016) with non-negative linear-layer parameters $\boldsymbol{\theta}$.

We say $(V, \mathcal{I})$ is a κ-extendible system (Mestre, 2006) if the following three conditions hold: (i) $\emptyset \in \mathcal{I}$, (ii) $X \subseteq Y \in \mathcal{I}$ implies $X \in \mathcal{I}$, and (iii) for all $X \in \mathcal{I}$ and $v \notin X$ such that $X \cup \{v\} \in \mathcal{I}$, and for every $Y \supseteq X$ such that $Y \in \mathcal{I}$, there exists $Z \subseteq Y \backslash X$ that satisfies $|Z| \le \kappa$ and $Y \backslash Z \cup \{v\} \in \mathcal{I}$. It is known that $(V, \mathcal{I})$ is a matroid iff it is a 1-extendible system, which includes the cardinality constrained case. The intersection of κ matroids defined on a common ground set always forms a κ-extendible system. We say $X \in \mathcal{I}$ is maximal if no $Y \in \mathcal{I}$ strictly includes X. We define $K \coloneqq \max_{X \in \mathcal{I}} |X|$, which is so-called the rank of $(V, \mathcal{I})$.

For any positive integer n, we let $\mathbf{0}_n$ and $\mathbf{1}_n$ be n-dimensional all-zero and all-one vectors, respectively. For any finite set V and $S \subseteq V$, we let $\mathbf{1}_S \in \mathbb{R}^{|V|}$ denote the indicator vector of S; i.e., the entries corresponding to S are 1 and the others are 0. For any scalar- or vector-valued differentiable function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$, $\nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x}) \in \mathbb{R}^{m \times n}$ denotes its gradient or Jacobian, respectively.

2 SMOOTHED GREEDY

We first explain details of Smoothed Greedy (Algorithm 1). Fix $\boldsymbol{\theta} \in \Theta$ arbitrarily. In the k-th iteration, we compute marginal gain $f_S(u, \boldsymbol{\theta})$ for every addable element $u \in U_k \coloneqq \{v \notin S \mid S \cup \{v\} \in \mathcal{I}\}$; we define $n_k \coloneqq |U_k|$ and index the elements in U_k as $u_1, \dots, u_{n_k}$. Let $\mathbf{g}_k(\boldsymbol{\theta}) = (g_k(u_1, \boldsymbol{\theta}), \dots, g_k(u_{n_k}, \boldsymbol{\theta})) \in \mathbb{R}^{n_k}$ denote the marginal gain vector. We compute a probability vector, $\mathbf{p}_k(\boldsymbol{\theta}) = (p_k(u_1, \boldsymbol{\theta}), \dots, p_k(u_{n_k}, \boldsymbol{\theta}))$,[1] as

$$\mathbf{p}_k(\boldsymbol{\theta}) = \underset{\mathbf{p} \in \Delta^{n_k}}{\mathrm{argmax}} \{\langle \mathbf{g}_k(\boldsymbol{\theta}), \mathbf{p} \rangle - \Omega_k(\mathbf{p})\}, \tag{2}$$

where $\Delta^{n_k} \coloneqq \{\mathbf{x} \in \mathbb{R}^{n_k} \mid \mathbf{x} \ge \mathbf{0}_{n_k}, \langle \mathbf{x}, \mathbf{1}_{n_k} \rangle = 1\}$ is the $(n_k - 1)$-dimensional probability simplex and $\Omega_k : \mathbb{R}^{n_k} \to \mathbb{R}$ is a strictly convex function; we call Ω_k a regularization function. Note that $\mathbf{p}_k(\boldsymbol{\theta})$ is unique due to the strict convexity. We then choose an element, $u \in U_k$, with probability $p_k(u, \boldsymbol{\theta})$ and add the chosen element, denoted by s_k, to S. The above procedure can be seen as a stochastically perturbed version of argmax; without Ω_k, we have $s_k \in \mathrm{argmax}_{u \in U_k} f_S(u, \boldsymbol{\theta})$.

[1] Note that $\mathbf{p}_k(\boldsymbol{\theta})$ depends on the past $k-1$ steps, which, however, is not written explicitly for simplicity.

We then show theoretical guarantees of Smoothed Greedy (see, Appendix B for proofs). Let $\delta \ge 0$ be a constant that satisfies $\delta \ge \Omega_k(\mathbf{p}) - \Omega_k(\mathbf{q})$ for all $k = 1, \dots, |S|$ and $\mathbf{p}, \mathbf{q} \in \Delta^{n_k}$. As show later, smaller δ values yield better guarantees. We present examples of Ω_k and their δ values at the end of this section.

As is often done, we begin by bounding the marginal gain. The following lemma elucidates the effect of δ and plays a key role when proving the subsequent theorems.

Lemma 1. *In any k-th step, conditioned on the $(k-1)$-th step (i.e., $S = \{s_1, \dots, s_{k-1}\}$ is arbitrarily fixed), we have $\mathbb{E}[f_S(s_k, \boldsymbol{\theta})] \ge f_S(u, \boldsymbol{\theta}) - \delta$ for any $u \in U_k$.*

Let S and O be an output of Algorithm 1 and a maximal optimal solution to problem (1), respectively. In the cardinality constrained case, we can obtain the following guarantee. We also show in Theorem 3 (Appendix F.1) that the faster stochastic variant (Mirzasoleiman et al., 2015) can achieve a similar approximation guarantee.

Theorem 1. *If $\mathcal{I} = \{X \subseteq V \mid |X| \le K\}$, we have $\mathbb{E}[f(S, \boldsymbol{\theta})] \ge (1 - 1/\mathrm{e}) f(O, \boldsymbol{\theta}) - \delta K$.*

For the more general case of κ-extendible systems, we can prove the following theorem.

Theorem 2. *If $(V, \mathcal{I})$ is a κ-extendible system with rank K, we have $\mathbb{E}[f(S, \boldsymbol{\theta})] \ge \frac{1}{\kappa+1} f(O, \boldsymbol{\theta}) - \delta K$.*

Proof sketch of Theorem 2. First, we briefly review the proof for the standard greedy algorithm (Calinescu et al., 2011). For a series of subsets $\emptyset = S_0 \subseteq S_1 \subseteq \dots \subseteq S_{|S|} = S$ obtained in $|S|$ steps of the greedy algorithm, we construct a series of subsets $O = O_0, O_1 \dots, O_{|S|} = S$ that satisfies $S_i \subseteq O_i \in \mathcal{I}$ and $\kappa \cdot (f(S_i, \boldsymbol{\theta}) - f(S_{i-1}, \boldsymbol{\theta})) \ge f(O_{i-1}, \boldsymbol{\theta}) - f(O_i, \boldsymbol{\theta})$ for $i = 1, \dots, |S|$. The $\frac{1}{\kappa+1}$-approximation is obtained by summing both sides for $i = 1, \dots, |S|$. Our proof extends this analysis to randomized Smoothed Greedy. We construct $O_0, O_1 \dots$ for each realization of the randomness, and prove $\kappa \cdot (\mathbb{E}[f(S_i, \boldsymbol{\theta})] - \mathbb{E}[f(S_{i-1}, \boldsymbol{\theta})] + \delta) \ge \mathbb{E}[f(O_{i-1}, \boldsymbol{\theta})] - \mathbb{E}[f(O_i, \boldsymbol{\theta})]$ for $i = 1, \dots, K$ by using Lemma 1. Here, unlike the deterministic case, $|S|$ may differ depending on realizations, and thus we must carefully construct $O_0, O_1, \dots$. By summing both sides for $i = 1, \dots, K$, we obtain Theorem 2. □

Existing guarantees (Tschiatschek et al., 2018; Powers et al., 2018) consider only a special case of Theorem 1 with $\delta = 0$; in this case the algorithm becomes non-differentiable. Therefore, our results, which hold even if $\delta > 0$ and deal with a wider class of constraints, bring significant progress in theoretically understanding differentiable submodular maximization.

Below we showcase two examples of regularization function Ω_k: entropy and quadratic functions. We can also use other strictly convex functions, e.g., a convex combination of the two functions. Note that when designing Ω_k, an additional differentiability condition (see, Assumption 2 in Section 3) must be satisfied for making expected outputs of Smoothed Greedy differentiable.

Entropy Function If $\Omega_k(\mathbf{p}) = \epsilon \sum_{i=1}^{n_k} p(u_i) \ln p(u_i)$, where $p(u_i)$ is the i-th entry of $\mathbf{p} \in [0,1]^{n_k}$ and $\epsilon > 0$ is an arbitrary constant, we have $\delta = \epsilon \ln n_k$. That is, the hyper-parameter, ϵ, controls the δ value. Moreover, Steps 4 to 6 can be efficiently performed via softmax sampling as with (Tschiatschek et al., 2018; Powers et al., 2018); i.e., we have $p_k(u, \boldsymbol{\theta}) \propto \exp(f_S(u, \boldsymbol{\theta})/\epsilon)$ (see, Appendix C.1 for details).

Quadratic Function We can use strongly convex quadratic functions as Ω_k. To be specific, if we let $\Omega_k(\mathbf{p}) = \epsilon \|\mathbf{p}\|_2^2$, then $\delta = \epsilon(1 - 1/n_k) \le \epsilon$. In this case, we need to solve quadratic programming (QP) for $k = 1, 2, \ldots$ for obtaining $\mathbf{p}_k(\boldsymbol{\theta})$. To this end, we can use an efficient batch QP solver (Amos and Kolter, 2017). When using the same Ω_k for every k, preconditioning (e.g., decomposition of Hessian matrices) is also effective for efficiently computing $\mathbf{p}_k(\boldsymbol{\theta})$.

As above, the δ value is often controllable, and thus we can use it as a hyper-parameter that balances the trade-off between the approximation guarantees and smoothness; we will experimentally analyze this in Section 3.1. In practice, δ value should be set depending on applications, which we discuss in Section 4.

3 GRADIENT ESTIMATION

We show how to differentiate outputs of Smoothed Greedy w.r.t. $\boldsymbol{\theta}$. Our idea is to utilize the score-function method (Rubinstein et al., 1996)[2] for computing gradient estimators. In Appendix F.2, we show that our method also works with the faster stochastic version (Mirzasoleiman et al., 2015) of Smoothed Greedy.

[2]The method is also know as the likelihood estimator (Glynn, 1990) and REINFORCE (Williams, 1992). Other than this, there are several major gradient estimators, e.g., (Jang et al., 2017; Mohamed et al., 2019). However, it is difficult to use them in our submodular maximization scenario as discussed in Appendix D.

In this section, we assume the following condition to hold:

Assumption 1. *For any $X \subseteq V$, we assume $f(X, \boldsymbol{\theta})$ to be differentiable w.r.t. $\boldsymbol{\theta}$.*

Assumption 1 is inevitable; the existing studies (Tschiatschek et al., 2018; Powers et al., 2018; Wilder et al., 2019a) are also based on this condition. Examples of functions satisfying Assumption 1 include weighted coverage functions (w.r.t. weights of covered vertices), probabilistic coverage functions (Wilder et al., 2019a), and deep submodular functions with smooth activation functions (Dolhansky and Bilmes, 2016). At the end of this section, we discuss what occurs if Assumption 1 fails to hold and possible remedies for such cases.

We also assume the following condition to hold. Note that, as we will see shortly, we can always satisfy it by appropriately choosing Ω_k (thus, it is rather a requirement when designing Ω_k than an assumption, but we here state it as an assumption for convenience).

Assumption 2. *Let $\mathbf{p}_k(\mathbf{g}_k)$ be the maximizer, $\mathbf{p}_k(\boldsymbol{\theta})$, in (2) regarded as a function of $\mathbf{g}_k(\boldsymbol{\theta})$. We assume $\mathbf{p}_k(\mathbf{g}_k)$ to be differentiable w.r.t. $\mathbf{g}_k$.*

For example, if Ω_k is the entropy function, we have $\nabla_{\mathbf{g}_k} \mathbf{p}_k(\mathbf{g}_k) = \epsilon^{-1}(\mathrm{diag}(\mathbf{p}_k(\mathbf{g}_k)) - \mathbf{p}_k(\mathbf{g}_k)\mathbf{p}_k(\mathbf{g}_k)^\top)$; i.e., the desired derivative can be computed in a closed form (see, Appendix C.1). In Appendix C.2, we present a sufficient condition for Ω_k to satisfy Assumption 2.

We then introduce the probability distribution of Smoothed Greedy outputs.[3]

Definition 1 (Output distribution)**.** *Let $\mathscr{S}_\mathcal{I}$ be the set of all sequences of elements that form a feasible solution $S \in \mathcal{I}$. For any fixed $\boldsymbol{\theta} \in \Theta$, we define $p(\boldsymbol{\theta}) : \mathscr{S}_\mathcal{I} \to [0,1]$ as the probability distribution function of* Smoothed Greedy *outputs, i.e., $S \sim p(\boldsymbol{\theta})$, which we refer to as the output distribution.*

We let $p(S, \boldsymbol{\theta}) \in [0,1]$ denote the probability that $S \in \mathscr{S}_\mathcal{I}$ is returned by Smoothed Greedy. Specifically, if it returns a sequence $S = (s_1, \ldots, s_{|S|}) \in \mathscr{S}_\mathcal{I}$, we have $p(S, \boldsymbol{\theta}) = \prod_{k=1}^{|S|} p_k(s_k, \boldsymbol{\theta})$, where $p_k(s_k, \boldsymbol{\theta})$ is the entry of $\mathbf{p}_k(\boldsymbol{\theta})$ corresponding to $s_k \in U_k$.

We now present our derivative computation method. Let $Q(S)$ be any scalar- or vector-valued function (see, Section 4 for examples of $Q(S)$). We aim to compute $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{S \sim p(\boldsymbol{\theta})}[Q(S)] = \Sigma_{S \in \mathscr{S}_\mathcal{I}} Q(S) \nabla_{\boldsymbol{\theta}} p(S, \boldsymbol{\theta})$. Since the size of $\mathscr{S}_\mathcal{I}$ is exponential in $K = \mathrm{O}(n)$, the exact derivative is unavailable in practice. Instead, we consider using the following unbiased estimator of the derivative:

[3]A similar notion is used in (Tschiatschek et al., 2018), but our way of using it is different (see, Appendix A).

Proposition 1. *Let $S_j = (s_1, \dots, s_{|S_j|}) \sim p(\boldsymbol{\theta})$ $(j = 1, \dots, N)$ be outputs of* Smoothed Greedy. *Then*

$$\frac{1}{N}\sum_{j=1}^{N} Q(S_j) \otimes \nabla_{\boldsymbol{\theta}} \ln p(S_j, \boldsymbol{\theta})$$

is an unbiased estimator of $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{S\sim p(\boldsymbol{\theta})}[Q(S)]$, where $\otimes$ denotes the outer product.

Proof. The claim is obtained from $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{S\sim p(\boldsymbol{\theta})}[Q(S)] = \Sigma_{S\in\mathscr{I}_{\mathcal{I}}} Q(S) \otimes (p(S,\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} \ln p(S,\boldsymbol{\theta})) = \mathbb{E}_{S\sim p(\boldsymbol{\theta})}[Q(S) \otimes \nabla_{\boldsymbol{\theta}} \ln p(S,\boldsymbol{\theta})]$, where an unbiased estimator of the RHS can be computed as described in the proposition. $\square$

The remaining problem is how to compute $\nabla_{\boldsymbol{\theta}} \ln p(S, \boldsymbol{\theta})$ for sampled $S = (s_1, \dots, s_{|S|})$. Since $\nabla_{\boldsymbol{\theta}} \ln p(S, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \ln \prod_{k=1}^{|S|} p_k(s_k, \boldsymbol{\theta}) = \sum_{k=1}^{|S|} \frac{1}{p_k(s_k,\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} p_k(s_k, \boldsymbol{\theta})$, it suffices to compute $\nabla_{\boldsymbol{\theta}} p_k(s_k, \boldsymbol{\theta})$ for $k = 1, \dots, |S|$. From Assumptions 1 and 2, we can differentiate $\mathbf{p}_k(\boldsymbol{\theta})$ by using the chain rule as $\nabla_{\boldsymbol{\theta}} \mathbf{p}_k(\boldsymbol{\theta}) = \nabla_{\mathbf{g}_k} \mathbf{p}_k(\mathbf{g}_k) \cdot \nabla_{\boldsymbol{\theta}} \mathbf{g}_k(\boldsymbol{\theta})$, and the row corresponding to $s_k \in U_k$ gives $\nabla_{\boldsymbol{\theta}} p_k(s_k, \boldsymbol{\theta})$. In some cases where $\mathbf{p}_k(\boldsymbol{\theta})$ can be analytically expressed as a function of $\boldsymbol{\theta}$, we can directly compute $\nabla_{\boldsymbol{\theta}} \ln p(S, \boldsymbol{\theta})$ via efficient automatic differentiation (Paszke et al., 2017; Baydin et al., 2018).

The above differentiation is usually not computationally expensive. If Ω_k is the entropy function, once $\nabla_{\boldsymbol{\theta}} \mathbf{g}_k(\boldsymbol{\theta})$ is obtained, we can compute $\nabla_{\boldsymbol{\theta}} \mathbf{p}_k(\boldsymbol{\theta})$ in $\mathrm{O}(n_k \times \dim \Theta)$ time (see, Appendix C for details). The cost of computing $\nabla_{\boldsymbol{\theta}} \mathbf{g}_k(\boldsymbol{\theta})$ is instance-dependent, but it is often as cheap as $\mathrm{O}(n_k)$ times evaluations of f due to the cheap gradient principle (Griewank and Walther, 2008).

Variance Reduction The variance of the gradient estimators sometimes becomes excessive, which requires us to sample too many outputs of Smoothed Greedy. Fortunately, there are various methods for reducing the variance of such Monte Carlo gradient estimators (Greensmith et al., 2004; Tucker et al., 2017; Mohamed et al., 2019). A simple and popular method is the following baseline correction (Williams, 1992): we use $Q(S) - \beta$ instead of $Q(S)$, where β is some coefficient. If β is a constant, the estimator remains unbiased since $\mathbb{E}_{S\sim p(\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}} \ln p(S, \boldsymbol{\theta})] = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{S\sim p(\boldsymbol{\theta})}[1] = 0$. By appropriately setting the β value, we can reduce the variance. In practice, to set β at the running average of $Q(\cdot)$ values is often effective, although this causes a small bias (see, (Mohamed et al., 2019) and references therein). We will use this variance reduction method (VR) in the experiments in Sections 3.1 and 5.

Non-differentiable Cases If Assumption 1 does not hold, i.e., $f(X, \boldsymbol{\theta})$ is not differentiable w.r.t. $\boldsymbol{\theta}$, the above discussion is not correct since the chain rule fails to hold (Griewank and Walther, 2008). This issue is common with many machine learning scenarios, e.g., training of NNs with ReLU activation functions. The current state of affairs is that we disregard this issue since it rarely brings harm in practice. Recently, Kakade and Lee (2018) developed a subdifferentiation method for dealing with such non-differentiable cases. Their result may enable us to extend the scope of our framework to non-differentiable $f(X, \boldsymbol{\theta})$.

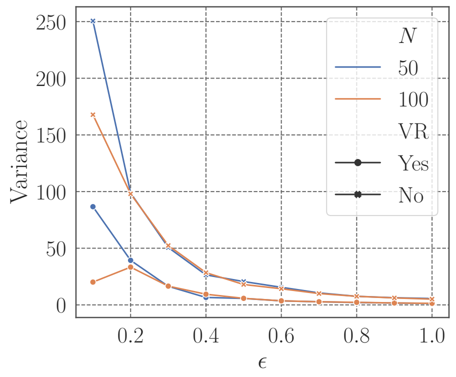


Figure 1: Variance of Estimated Derivatives

3.1 Experimental Study on Variance

The behavior of our estimator depends on regularization functions, sample size N, and whether we use VR or not; in particular, the effect of regularization strength δ on the estimator's quality is non-trivial. Unfortunately, the theoretical analysis of the effects is too difficult because of the complicated structure of the output distribution, which is specified with the iterative perturbed argmax over marginal gains of instance-dependent function f. To gain an empirical understanding of the effects, we here present an experiment on the variance of the estimator. We use a bipartite influence maximization instance, which is the same as the one detailed in Section 5.1. We use the entropy function as Ω_k, and thus the δ value is controlled by $\epsilon > 0$ as $\delta = \epsilon \ln n_k$.

For notational ease, we let $G_j = Q(S_j) \otimes \nabla_{\boldsymbol{\theta}} \ln p(S_j, \boldsymbol{\theta})$ be a derivative estimated with the j-th output sample $(j = 1, \dots, N)$, and $\bar{G}$ denotes their average. We study how the variance $\frac{1}{N}\sum_{j=1} \|G_j - \bar{G}\|^2$, where $\|\cdot\|$ denotes the Frobenius norm, is affected by ϵ, N, and VR.

Figure 1 shows the result. We see that the decrease in ϵ increases the variance. This is because, as ϵ decreases, $\mathbf{p}_k(\mathbf{g}_k)$ becomes close to being non-smooth; consequently, $\nabla_{\boldsymbol{\theta}} \ln p(S_j, \boldsymbol{\theta})$ largely fluctuates among sampled outputs, resulting in large variances. Thus, as mentioned in Section 2, we can regard ϵ (or δ) as a hyper-parameter that controls the trade-off between the approximation guarantee and the variance (or smoothness of $\mathbf{p}_k(\mathbf{g}_k)$). We can also see that VR is effective. The increase in N appears to decrease the variance when ϵ is small, but its effect is subtle when ϵ is large.

4 APPLICATIONS

Our framework accepts any computable $Q(S)$, and thus we can use it in various situations. We here show how to apply it to sensitivity analysis and decision-focused learning. We also present another application related to learning of submodular models in Appendix E.

4.1 Sensitivity Analysis

When addressing parametric optimization instances, the sensitivity—how and how much changes in parameter values can affect outputs of algorithms—is a major concern. In continuous optimization settings, most sensitivity analysis methods are based on derivatives of outputs (Rockafellar and Wets, 1998; Gal and Greenberg, 2012; Bertsekas, 2016). By contrast, those for combinatorial settings are diverse (Gusfield, 1980; Kleijnen and Rubinstein, 1996; Bertsimas, 1988; Ghosh et al., 2000; Varma and Yoshida, 2019) probably due to the non-differentiability. As explained below, our gradient estimation method can be used for analyzing the sensitivity of Smoothed Greedy, which can become close to the greedy algorithm by decreasing δ. This provides, to the best of our knowledge, the first method for analyzing the sensitivity of the greedy algorithm for submodular function maximization.

We analyze the sensitivity of the probability that each $v \in V$ appears in a Smoothed Greedy output, which can be expressed as $\mathbb{E}_{S\sim p(\boldsymbol{\theta})}[\mathbf{1}_S] = \Sigma_{S\in\mathscr{I}_\mathcal{I}} \mathbf{1}_S p(S,\boldsymbol{\theta})$. By using our method shown in Section 3 with $Q(S) = \mathbf{1}_S$, we can estimate the Jacobian matrix as

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{S\sim p(\boldsymbol{\theta})}[\mathbf{1}_S] \approx \frac{1}{N}\sum_{j=1}^{N} \mathbf{1}_{S_j} \otimes \nabla_{\boldsymbol{\theta}} \ln p(S_j,\boldsymbol{\theta}).$$

Here, given $\boldsymbol{\theta}$, the (v, j) entry of the matrix indicates how and how much the infinitesimal increase in the j-th entry of $\boldsymbol{\theta}$ affects the probability that $v \in V$ is chosen, which quantifies the sensitivity of each $v \in V$ to uncertainties in $\boldsymbol{\theta}$. This information is beneficial to practitioners who address submodular maximization tasks with uncertain parameters (e.g., advertisers who want to know how to reliably promote products); if Smoothed Greedy outputs are found to be too sensitive, we can consider using more robust methods, e.g., (Staib et al., 2019). Experiments in Section 5.1 demonstrates how this sensitivity analysis method works.

4.2 Decision-Focused Learning

We consider a situation where $\boldsymbol{\theta}$ is computed with some predictive models (e.g., NNs). Let $m(\cdot,\mathbf{w})$ be a predictive model that maps some observed feature $\mathbf{X}$ to $\boldsymbol{\theta}$, where $\mathbf{w}$ represents model parameters. We train $m(\cdot,\mathbf{w})$ by optimizing $\mathbf{w}$ values with training datasets $(\mathbf{X}_1,\boldsymbol{\theta}_1),\dots,(\mathbf{X}_M,\boldsymbol{\theta}_M)$. Given test instance $(\tilde{\mathbf{X}},\tilde{\boldsymbol{\theta}})$, where $\tilde{\boldsymbol{\theta}}$ is the unknown true parameter, the trained model predicts $\boldsymbol{\theta} = m(\tilde{\mathbf{X}},\mathbf{w})$, and we obtain solution $S \in \mathcal{I}$ (or, make a decision) by approximately maximizing $f(\cdot,\boldsymbol{\theta})$. Our utility (decision quality) is measured by $f(S,\tilde{\boldsymbol{\theta}})$. This situation often occurs in real-world scenarios, e.g., budget allocation, diverse recommendation, and viral marketing (see, (Wilder et al., 2019a)). For example, in the case of viral marketing on a social network, $\boldsymbol{\theta}$ represents link probabilities, which we predict with $m(\cdot,\mathbf{w})$ for observed feature $\mathbf{X}$. A decision is a node subset S, which we activate to maximize the influence. Our utility is the influence spread $f(S,\tilde{\boldsymbol{\theta}})$, where $\tilde{\boldsymbol{\theta}}$ represents unknown true link probabilities.

With the decision-focused learning approach (Wilder et al., 2019a), we train predictive models in an attempt to maximize the decision quality, $f(S,\tilde{\boldsymbol{\theta}})$. This approach is empirically more effective for the above situation involving both prediction and optimization than the standard two-stage approach, which trains predictive models separately from the downstream optimization problems. By combining our framework with the decision-focused approach, we can train predictive models with first-order methods so that Smoothed Greedy achieves high expected objective values.

Below we detail how to train predictive models. We consider maximizing the empirical utility function, $\frac{1}{M}\sum_{i=1}^{M}\mathbb{E}_{S\sim p(m(\mathbf{X}_i,\mathbf{w}))}[f(S,\boldsymbol{\theta}_i)]$, where $p(\cdot)$ is the output distribution. In each iteration, we sample a training dataset, $(\mathbf{X}_i,\boldsymbol{\theta}_i)$, and compute $\boldsymbol{\theta} = m(\mathbf{X}_i,\mathbf{w})$ with the current $\mathbf{w}$ values. We then perform N trials of Smoothed Greedy to estimate the current function value, $\mathbb{E}_{S\sim p(\boldsymbol{\theta})}[f(S,\boldsymbol{\theta}_i)]$. Next, with the outcomes of N trials, we estimate the gradient by using our method with $Q(S) = f(S,\boldsymbol{\theta}_i)$. More precisely, for each j-th trial of Smoothed Greedy, we compute $\nabla_{\boldsymbol{\theta}} \ln p(S_j,\boldsymbol{\theta})$ as explained in Section 3 and estimate the gradient, $\nabla_{\mathbf{w}}\mathbb{E}_{S\sim p(m(\mathbf{X}_i,\mathbf{w}))}[f(S,\boldsymbol{\theta}_i)]$, as follows:[4]

$$\frac{1}{N}\sum_{j=1}^{N} f(S_j,\boldsymbol{\theta}_i)\nabla_{\boldsymbol{\theta}} \ln p(S_j,\boldsymbol{\theta})\cdot\nabla_{\mathbf{w}} m(\mathbf{X}_i,\mathbf{w}),$$

where $\boldsymbol{\theta} = m(\mathbf{X}_i,\mathbf{w})$. We then update $\mathbf{w}$ with the above gradient estimator. When using mini-batch up-

[4]If $m(\cdot,\mathbf{w})$ is not differentiable, the chain rule, $\nabla_{\mathbf{w}} \ln p(S_j,\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \ln p(S_j,\boldsymbol{\theta})\cdot\nabla_{\mathbf{w}} m(\mathbf{X}_i,\mathbf{w})$, fails to hold. This issue is essentially the same as what we discussed in Section 3, which we may usually disregard in practice.

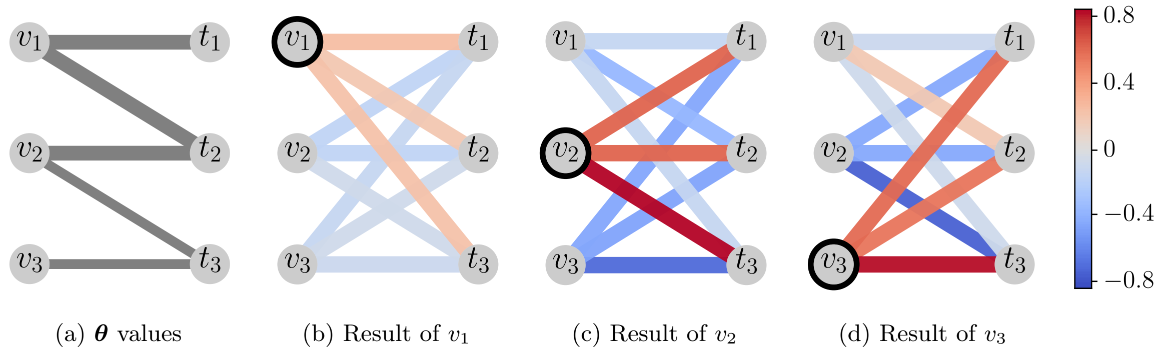


Figure 2: Sensitivity Analysis Instance and Results. (a): The $\boldsymbol{\theta}$ values; thick and thin edges have link probabilities 0.4 and 0.2, respectively. (b) to (d): Illustration of results; edge colors in (b), (c), and (d) indicate how the increase in the corresponding $\boldsymbol{\theta}$ entries can affect the probability of choosing v_1, v_2, and v_3, respectively.

dates, we accumulate the loss values and gradient estimators over datasets in a mini-batch, and then update $\mathbf{w}$. Note that the N trials of Smoothed Greedy, as well as the computation of $\nabla_{\boldsymbol{\theta}} \ln p(S_j, \boldsymbol{\theta})$, can be performed in parallel. Experiments in Section 5.2 confirm the practical effectiveness of the above method.

In this setting, we let regularization strength δ be large to some extent. This is because, Smoothed Greedy with too small δ may overfit to outputs of a predictive model in early stages, where it is not well trained yet.

5 EXPERIMENTS

We evaluate our method with sensitivity analysis and decision-focused learning instances. We use the entropy function as Ω_k; as the effect of ϵ has been studied in Section 3.1, we here fix $\epsilon = 0.2$ for simplicity. All experiments are performed on a 64-bit macOS machine with 1.6GHz Intel Core i5 CPUs and 16GB RAMs.

We use the following bipartite influence maximization instance. Let V and T be sets of items and targets, respectively, and $\boldsymbol{\theta} \in [0,1]^{V \times T}$ be link probabilities. We aim to maximize the expected number of influenced targets, $f(X, \boldsymbol{\theta}) = \sum_{t \in T} \left(1 - \prod_{v \in X} (1 - \theta_{v,t})\right)$, by choosing up to K items.

In Appendix E.2, we perform experiments with another setting, where we consider learning deep submodular functions under a partition matroid constraint. Experiments in Appendix F.3 analyze performances of the faster stochastic version of Smoothed Greedy.

5.1 Sensitivity Analysis

We perform sensitivity analysis with a synthetic instance such that $V = \{v_1, v_2, v_3\}$, $T = \{t_1, t_2, t_3\}$, and $K = 2$. Let $\theta_{i,j}$ denote the link probability of (v_i, t_j); we set $(\theta_{1,1}, \theta_{1,2}, \theta_{1,3}) = (0.4, 0.4, 0)$, $(\theta_{2,1}, \theta_{2,2}, \theta_{2,3}) = (0, 0.4, 0.2)$, and $(\theta_{3,1}, \theta_{3,2}, \theta_{3,3}) = (0, 0, 0.2)$ as in Figure 2a. We analyze the sensitivity of Smoothed Greedy by estimating $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{S \sim p(\boldsymbol{\theta})}[\mathbf{1}_S]$ as in Section 4.1. We let $N = 100$ and use VR as explained in Section 3.

Figures 2b, 2c, and 2d illustrate how and how much the increase in each $\theta_{i,j}$ value can affect the probability of choosing v_1, v_2, and v_3, respectively. In this setting, the objective values of the three maximal solutions, $\{v_1, v_2\}$, $\{v_1, v_3\}$, and $\{v_2, v_3\}$, are 1.24, 1.00, and 0.76, respectively. Therefore, Smoothed Greedy returns $\{v_1, v_2\}$ or $\{v_1, v_3\}$ with a high probability. This remains true even if the $\boldsymbol{\theta}$ values slightly change, and thus the probability of choosing v_1 is relatively insensitive as in Figure 2b. By contrast, as in Figures 2c and 2d, the probabilities of choosing v_2 and v_3, respectively, are highly sensitive. For example, if $\theta_{2,3}$ increases, the probability that the algorithm returns $\{v_1, v_2\}$ ($\{v_1, v_3\}$) increases (decreases), which means the probability of choosing v_2 (v_3) is positively (negatively) affected by the increase in $\theta_{2,3}$. We can also see the that the opposite occurs if $\theta_{3,3}$ increases.

5.2 Decision-Focused Learning

We evaluate the performance of our method via decision-focused learning experiments with MovieLens 100K dataset (Harper and Konstan, 2015), which contains 100,000 ratings (1 to 5) of 1,682 movies made by 943 users. We set the link probabilities at $0.02, 0.04, \dots, 0.1$ according to the ratings; those of unrated ones are set at 0. We randomly sample 100 movies and 500 users, which form item set V and target set T, respectively. We thus make 100 random (V, T) pairs with link probabilities. Each movie $v \in V$ belongs to some of 19 genres, e.g., action and horror; we use

Table 1: Function Values Achieved with Each Method.

	$K=5$		$K=10$		$K=20$	
	Training	Test	Training	Test	Training	Test
SG-1	26.3 ± 4.0	26.4 ± 4.4	46.0 ± 5.9	45.9 ± 6.5	69.7 ± 23.8	69.6 ± 24.1
SG-10	29.0 ± 3.7	28.1 ± 4.9	47.0 ± 12.1	46.1 ± 12.4	71.5 ± 28.0	70.6 ± 28.1
SG-100	33.6 ± 2.4	32.0 ± 3.8	54.3 ± 2.0	53.5 ± 4.2	82.6 ± 21.8	82.3 ± 21.7
VR-SG-10	35.2 ± 6.1	33.7 ± 6.2	57.9 ± 1.6	56.2 ± 3.4	90.8 ± 16.5	89.5 ± 16.7
VR-SG-100	$\mathbf{36.8 \pm 0.9}$	$\mathbf{35.6 \pm 2.2}$	$\mathbf{59.9 \pm 1.6}$	$\mathbf{58.0 \pm 2.9}$	$\mathbf{96.8 \pm 1.1}$	$\mathbf{94.5 \pm 2.6}$
Continuous	24.0 ± 4.5	23.2 ± 4.9	43.2 ± 6.1	42.3 ± 7.1	81.7 ± 6.8	81.3 ± 6.6
Two-stage	17.3 ± 1.2	17.3 ± 2.1	35.6 ± 0.9	35.6 ± 2.7	65.5 ± 4.0	64.8 ± 5.1
Random	17.5 ± 1.0	17.6 ± 2.2	33.8 ± 0.8	34.0 ± 2.7	64.0 ± 1.3	64.5 ± 2.6

the 19-dimensional indicator vector as a movie feature. Each user $t \in T$ has information of their age, sex, and occupation categorized into 21 types, e.g., writer and doctor; we concatenate them and use the resulting 24-dimensional vector as a user feature. A feature of each $(v, t) \in V \times T$ is a concatenation of the 19- and 24-dimensional vectors. As a result, each of the 100 random (V, T) pairs has feature $\mathbf{X}$ of form $100 \times 500 \times 43$. The predictive model, which outputs $\theta_{v,t} \in [0, 1]$ for the feature of each $(v, t) \in V \times T$, is a 2-layer NN with a hidden layer of size 200 and ReLU activation functions, where the outputs are clipped to $[0, 1]$. Since the features are sparse, the predictive model with default weight initialization returns 0 too frequently; to avoid this, we set initial linear-layer weights at random non-negative values drawn from $[0, 0.01]$.

We split the 100 random instances into 80 training and 20 test instances. We train the predictive model with $(\mathbf{X}_1, \boldsymbol{\theta}_1), \dots, (\mathbf{X}_{80}, \boldsymbol{\theta}_{80})$ and test the performance with $(\tilde{\mathbf{X}}_1, \tilde{\boldsymbol{\theta}}_1), \dots, (\tilde{\mathbf{X}}_{20}, \tilde{\boldsymbol{\theta}}_{20})$. We make 30 random training/test splits; we present all results with means and standard deviations over the 30 random splits. Given 80 training datasets, we train the model over mini-batches of size 20 for 5 epochs. We use Adam with learning rate 10^{-3} for updating parameter $\mathbf{w}$ of the predictive model.[5]

We compare **SG-**N, **VR-SG-**N, **Continuous**, **Two-stage**, and **Random**. **SG-**N is our method based on Smoothed Greedy (see, Section 4.2), where N indicates the number of output samples; we let $N = 1$, 10, and 100. **VR-SG-**N (variance-reduced **SG-**N) uses the baseline correction method when estimating gradients; we let $N = 10$ and 100 (omit $N = 1$) since if $N = 1$, the baseline value is equal to the single output value, which always yields zero gradients. Both **SG-**N and **VR-SG-**N use the greedy algorithm when making decisions. **Continuous** (Wilder et al., 2019a) maximizes the continuous relaxation (multilinear extension) of the objective function with SGA and differentiates local optima (we use their original implementation). **Two-stage** trains the model by minimizing the mean square error and then maximizes the objective function with SGA (the implementation is based on that of (Wilder et al., 2019a)). **Continuous** and **Two-stage** make decisions $S \in \mathcal{I}$ by choosing elements corresponding to the top-K entries of solution $\mathbf{x} \in [0, 1]^n$ returned by SGA. **Random** is a baseline method that makes decisions $S \in \mathcal{I}$ uniformly at random.

Table 1 shows the objective function values (averaged over the 80 training and 20 test instances) achieved by each method for $K = 5$, 10, and 20. **VR-SG-**100 achieves the highest objective value for every case, and (**VR-**)**SG** with other settings also performs comparably to or better than **Continuous**. These results are consistent with the theoretical guarantees. More precisely, while **Continuous** trains the predictive model so that SGA, a 1/2-approximation algorithm, returns high objective values, our method trains the predictive model based on outputs of Smoothed Greedy, which achieves an almost $(1 - 1/\mathrm{e})$-approximation. The results also show that VR is effective for improving the performance of our method. The standard deviation of (**VR-**)**SG** becomes sometimes high; this is because they are sometimes trapped in poor local optima and result in highly deviated objective values. Considering this, the performance of our method would be further improved if we can combine it with NN training techniques for escaping from poor local optima. Regarding running times, for updating $\mathbf{w}$ once, **SG-**1 takes 2.81, 3.38, and 3.77 seconds on average for $K = 5$, 10, and 20, respectively, while **Continuous** takes 5.86, 5.87, and 6.11 seconds, respectively.[6] Hence, our method can run faster by performing Smoothed Greedy in parallel as mentioned in Section 4.2.

[5]The settings mostly replicate those of (Wilder et al., 2019a), but we use the public MovieLens dataset instead of the original one, which is not open to the public. Accordingly, some parts are slightly changed.

[6]Note that the above instance is a special case where the multilinear extension has a closed-form expression that is computable in polynomial time; this makes **Continuous** particularly fast. When such an expression is unavailable, our method, is far more efficient than **Continuous**.

Acknowledgements

This work was supported by JST ERATO Grant Number JPMJER1903.

References

J. Abernethy, C. Lee, and A. Tewari. Perturbation techniques in online learning and optimization. In *Perturbations, Optimization, and Statistics*. MIT Press, 2016.

N. Alon, I. Gamzu, and M. Tennenholtz. Optimizing budget allocation among channels and influencers. In *Proceedings of the 21st International Conference on World Wide Web*, pages 381–388. ACM, 2012.

B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 136–145. PMLR, 2017.

E. Balkanski and Y. Singer. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing*, pages 1138–1151. ACM, 2018.

R. Barnes and T. Burkett. Structural redundancy and multiplicity in corporate networks. *Connect.*, 30(2): 4–20, 2010.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, 18(153): 1–43, 2018.

Q. Berthet, M. Blondel, O. Teboul, M. Cuturi, J.-P. Vert, and F. Bach. Learning with differentiable perturbed optimizers. *arXiv preprint arXiv:2002.08676*, 2020.

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 3rd edition, 2016.

D. Bertsimas. *Probabilistic Combinatorial Optimization Problems*. PhD thesis, Massachusetts Institute of Technology, 1988.

G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6): 1740–1766, 2011.

M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems 26*, pages 2292–2300. Curran Associates, Inc., 2013.

M. Cuturi, O. Teboul, and J.-P. Vert. Differentiable ranking and sorting using optimal transport. In *Advances in Neural Information Processing Systems 32*, pages 6861–6871. Curran Associates, Inc., 2019.

J. Djolonga and A. Krause. Differentiable learning of submodular models. In *Advances in Neural Information Processing Systems 30*, pages 1013–1023. Curran Associates, Inc., 2017.

B. W. Dolhansky and J. A. Bilmes. Deep submodular functions: Definitions and learning. In *Advances in Neural Information Processing Systems 29*, pages 3404–3412. Curran Associates, Inc., 2016.

A. L. Dontchev and R. T. Rockafellar. *Implicit Functions and Solution Mappings*. Springer, 2nd edition, 2014.

P. Donti, B. Amos, and J. Z. Kolter. Task-based end-to-end model learning in stochastic optimization. In *Advances in Neural Information Processing Systems 30*, pages 5484–5494. Curran Associates, Inc., 2017.

U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

A. Ferber, B. Wilder, B. Dilina, and M. Tambe. MIPaaL: Mixed integer program as a layer. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (to appear)*, 2020. arXiv:1907.05912.

M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions–II. In *Polyhedral combinatorics*, pages 73–87. Springer, 1978.

T. Gal and H. J. Greenberg. *Advances in Sensitivity Analysis and Parametric Programming*, volume 6. Springer, 2012.

D. Ghosh, N. Chakravarti, and G. Sierksma. Sensitivity analysis of the greedy heuristic for binary knapsack problems. Research Report 00A18, University of Groningen, Research Institute SOM (Systems, Organisations and Management), 2000.

P. W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Commun. ACM*, 33(10):75–84, 1990.

E. Greensmith, P. L. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *J. Mach. Learn. Res.*, 5(Nov): 1471–1530, 2004.

A. Griewank and A. Walther. *Evaluating Derivatives*. SIAM, 2nd edition, 2008.

E. J. Gumbel. Statistical theory of extreme values and some practical applications: A series of lectures. *US Govt. Print. Office*, 33, 1954.

D. M. Gusfield. *Sensitivity Analysis for Combinatorial Optimization*. PhD thesis, University of California, Berkeley, 1980.

F. M. Harper and J. A. Konstan. The MovieLens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, 2015. URL https://grouplens.org/datasets/movielens/100k/.

H. Hassani, M. Soltanolkotabi, and A. Karbasi. Gradient methods for submodular maximization. In *Advances in Neural Information Processing Systems 30*, pages 5841–5851. Curran Associates, Inc., 2017.

E. Jang, S. Gu, and B. Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017.

S. M. Kakade and J. D. Lee. Provably correct automatic subdifferentiation for qualified programs. In *Advances in Neural Information Processing Systems 31*, pages 7125–7135. Curran Associates, Inc., 2018.

A. Kalyan, P. Anderson, S. Lee, and D. Batra. Trainable decoding of sets of sequences for neural sequence models. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 3211–3221. PMLR, 2019.

S. Kannan, J. H. Morgenstern, A. Roth, B. Waggoner, and Z. S. Wu. A smoothed analysis of the greedy algorithm for the linear contextual bandit problem. In *Advances in Neural Information Processing Systems 31*, pages 2227–2236. Curran Associates, Inc., 2018.

J. P. C. Kleijnen and R. Y. Rubinstein. Optimization and sensitivity analysis of computer simulation models by the score function method. *European J. Oper. Res.*, 88(3):413–427, 1996.

J. Kunegis. KONECT: The Koblenz Network Collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013. URL http://konect.uni-koblenz.de/networks/brunson_corporate-leadership.

C. J. Maddison, A. Mnih, and Y. W. Teh. The Concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

A. Mensch and M. Blondel. Differentiable dynamic programming for structured prediction and attention. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 3462–3471. PMLR, 2018.

J. Mestre. Greedy in approximation algorithms. In *Proceedings of the 17th Annual European Symposium on Algorithms*, pages 528–539. Springer, 2006.

B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause. Lazier than lazy greedy. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 1812–1818. AAAI Press, 2015.

B. Mirzasoleiman, A. Badanidiyuru, and A. Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, pages 1358–1367. PMLR, 2016.

S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih. Monte Carlo gradient estimation in machine learning. *arXiv preprint arXiv:1906.10652*, 2019.

G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978.

G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions–I. *Math. Program.*, 14(1):265–294, 1978.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS 2017 Workshop Autodiff*, 2017.

M. Peyrard. *Principled Approaches to Automatic Text Summarization*. PhD thesis, Technische Universität, 2019.

M. V. Pogančić, A. Paulus, V. Musil, G. Martius, and M. Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2020.

T. Powers, R. Fakoor, S. Shakeri, A. Sethy, A. Kainth, A. Mohamed, and R. Sarikaya. Differentiable greedy networks. *arXiv preprint arXiv:1810.12464*, 2018.

R. T. Rockafellar and R. J-B Wets. *Variational Analysis*, volume 317. Springer, 1998.

R. Y. Rubinstein, A. Shapiro, and S. Uryasev. The score function method. *Encyclopedia of Management Sciences*, pages 1363–1366, 1996.

M. Staib, B. Wilder, and S. Jegelka. Distributionally robust submodular maximization. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, volume 89, pages 506–516. PMLR, 2019.

P. Stechlinski, K. A. Khan, and P. I. Barton. Generalized sensitivity analysis of nonlinear programs. *SIAM J. Optim.*, 28(1):272–301, 2018.

S. Tschiatschek, A. Sahin, and A. Krause. Differentiable submodular maximization. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2731–2738. IJCAI Organization, 2018.

G. Tucker, A. Mnih, C. J. Maddison, J. Lawson, and J. Sohl-Dickstein. REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems 30*, pages 2627–2636. Curran Associates, Inc., 2017.

N. Varma and Y. Yoshida. Average sensitivity of graph algorithms. *arXiv preprint arXiv:1904.03248*, 2019.

P.-W. Wang, P. Donti, B. Wilder, and Z. Kolter. SAT-Net: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 6545–6554. PMLR, 2019.

B. Wilder, B. Dilkina, and M. Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pages 1658–1665. AAAI Press, 2019a.

B. Wilder, E. Ewing, B. Dilkina, and M. Tambe. End to end learning and optimization on graphs. In *Advances in Neural Information Processing Systems 32*, pages 4672–4683. Curran Associates, Inc., 2019b.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3):229–256, 1992.

M. Zhang, L. Chen, H. Hassani, and A. Karbasi. Online continuous submodular maximization: From full-information to bandit feedback. In *Advances in Neural Information Processing Systems 32*, pages 9210–9221. Curran Associates, Inc., 2019.

Differentiable Greedy Algorithm for Monotone Submodular Maximization: Guarantees, Gradient Estimators, and Applications (Supplementary Material)

A COMPARISONS WITH EXISTING GREEDY METHODS

We present detailed comparisons of our work and the exiting studies (Tschiatschek et al., 2018; Powers et al., 2018) on the differentiable greedy methods, which use softmax instead of argmax. As explained below, the existing methods are devoted to differentiating some functions defined with subsets $X_1, X_2, \dots \subseteq V$ given as training data. By contrast, we do not assume such subsets to be given and consider differentiating the expected value of any output-dependent quantities, $\mathbb{E}_{S\sim p(\boldsymbol{\theta})}[Q(S)]$. Note that this design of our framework is the key to dealing with a wide variety of applications including sensitivity analysis and decision-focused learning. Our framework can also provide more reasonable approaches to their problem settings as described below.

Tschiatschek et al. (2018) consider differentiating the likelihood function, which quantifies how close an output of their algorithm can be to some good solutions, $X_1, X_2, \dots$, given as training data. To this end, we need to differentiate $P(X) \coloneqq \Sigma_{\sigma\in\Sigma(X)} P(\sigma, \boldsymbol{\theta})$, where $X \in \{X_1, X_2, \dots\}$ is a given subset, $\Sigma(X)$ is the set of all permutations of elements in X, and $P(\sigma, \boldsymbol{\theta})$ is the probability that their algorithm returns sequence $\sigma \in \mathscr{S}_{\mathcal{I}}$. Since the computation of the summation over $\Sigma(X)$ is too costly, they employ the following heuristic approximation: if the temperature of softmax is low, we let $P(X) \approx P(\sigma^G, \boldsymbol{\theta})$, where σ^G is obtained by the greedy algorithm, and if the temperature is high, we let $P(X) \approx |X|! \times P(\sigma^R, \boldsymbol{\theta})$, where σ^R is a random permutation. As a result, the computed derivative has no theoretical guarantees unlike our gradient estimator, which is guaranteed to be unbiased. Note that with our method, we can compute an unbiased estimator of the desired derivative as follows: we let $Q(S)$ return 1 if S and X consist of the same elements and 0 otherwise, and we estimate $\nabla_{\boldsymbol{\theta}} P(X) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{S\sim p(\boldsymbol{\theta})}[Q(S)]$ as explained in Section 3.

Powers et al. (2018) focus on some cases where we can compute derivatives more easily. They consider some loss function $L(X, \mathbf{p}_1(\boldsymbol{\theta}), \dots, \mathbf{p}_K(\boldsymbol{\theta}))$ that is differentiable w.r.t. $\mathbf{p}_1(\boldsymbol{\theta}), \dots, \mathbf{p}_K(\boldsymbol{\theta})$, where X is given as training data. In their setting, $\mathbf{p}_i(\boldsymbol{\theta})$ is given by softmax and $f(X, \boldsymbol{\theta})$ is differentiable w.r.t. $\boldsymbol{\theta}$. Thus, once X is fixed, $\nabla_{\boldsymbol{\theta}} L(X, \mathbf{p}_1(\boldsymbol{\theta}), \dots, \mathbf{p}_K(\boldsymbol{\theta}))$ can be readily computed via automatic differentiation. From the perspective of our method, we can regard their method as the one dealing with the case of $N = 1$. More precisely, if we take X to be a single output of Smoothed Greedy and let $L(X, \mathbf{p}_1(\boldsymbol{\theta}), \dots, \mathbf{p}_K(\boldsymbol{\theta})) = Q(X) \ln p(X, \boldsymbol{\theta})$, then the derivative computed by their method coincides with the one obtained by using our method with $N = 1$. Note that the above design of $L(\cdot)$, which is the key to obtaining unbiased gradient estimators, and the case of $N > 1$ are not studied in (Powers et al., 2018). Furthermore, if we apply our framework to their problem setting, we can use (non-differentiable) loss functions, $L(X, S)$, that measure the distance between given X and output S (e.g., Hamming and Levenshtein distances); we let $Q(S) = L(X, S)$ and estimate $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{S\sim p(\boldsymbol{\theta})}[L(X, S)]$.

B PROOFS OF APPROXIMATION GUARANTEES

We present proofs of the lemma and theorems. In the following discussion, S_k denotes the solution constructed in the k-th step of Algorithm 1; we let $S_0 = \emptyset$. For simplicity, we omit the fixed parameter, $\boldsymbol{\theta}$, in the proofs.

Lemma 1. *In any k-th step, conditioned on the $(k-1)$-th step (i.e., $S = \{s_1, \dots, s_{k-1}\}$ is arbitrarily fixed), we have $\mathbb{E}[f_S(s_k, \boldsymbol{\theta})] \ge f_S(u, \boldsymbol{\theta}) - \delta$ for any $u \in U_k$.*

Proof. From the rule of choosing s_k, we have $\mathbb{E}[f_S(s_k)] = \langle \mathbf{g}_k, \mathbf{p}_k \rangle$. Let $\mathbf{1}_u \in \mathbb{R}^{n_k}$ be the indicator vector of

$u \in U_k$. Since $\langle \mathbf{g}_k, \mathbf{1}_u \rangle = f_S(u)$ and $\mathbf{1}_u \in \Delta^{n_k}$ hold, we can obtain the lemma as follows:

$$\begin{aligned}\mathbb{E}[f_S(s_k)] = \langle \mathbf{g}_k, \mathbf{p}_k \rangle &= \max_{\mathbf{p} \in \Delta^{n_k}} \{\langle \mathbf{g}_k, \mathbf{p} \rangle - \Omega_k(\mathbf{p})\} + \Omega_k(\mathbf{p}_k) \\ &\geq \langle \mathbf{g}_k, \mathbf{1}_u \rangle - (\Omega_k(\mathbf{1}_u) - \Omega_k(\mathbf{p}_k)) \geq f_S(u) - \delta,\end{aligned}$$

where the last inequality comes from $\delta \geq \Omega_k(\mathbf{p}) - \Omega_k(\mathbf{q})$ for any $\mathbf{p}, \mathbf{q} \in \Delta^{n_k}$. □

Theorem 1. *If $\mathcal{I} = \{X \subseteq V \mid |X| \leq K\}$, we have $\mathbb{E}[f(S, \boldsymbol{\theta})] \geq (1 - 1/\mathrm{e}) f(O, \boldsymbol{\theta}) - \delta K$.*

Proof. Fix $k \in \{1, \dots, K\}$ arbitrarily and take all random quantities to be conditioned on the $(k-1)$-th step. From Lemma 1 with $O \backslash S_{k-1} \subseteq U_k$ and the submodularity, we obtain

$$\mathbb{E}[f_{S_{k-1}}(s_k)] \geq \frac{1}{K} \sum_{v \in O \backslash S_{k-1}} f_{S_{k-1}}(v) - \delta \geq \frac{1}{K} f_{S_{k-1}}(O) - \delta.$$

By taking expectation over all possible realizations of the $(k-1)$-th step and using the monotonicity, we obtain

$$\mathbb{E}[f(S_k)] - \mathbb{E}[f(S_{k-1})] \geq \frac{1}{K}(\mathbb{E}[f(O \cup S_{k-1})] - \mathbb{E}[f(S_{k-1})]) - \delta \geq \frac{1}{K}(f(O) - \mathbb{E}[f(S_{k-1})]) - \delta.$$

Therefore, as is often the case with the analysis of the greedy algorithm, we can obtain the following inequality by induction:

$$\mathbb{E}[f(S_K)] \geq \left(1 - \left(1 - \frac{1}{K}\right)^K\right) f(O) - \delta \sum_{k=0}^{K-1} \left(1 - \frac{1}{K}\right)^k \geq \left(1 - \frac{1}{\mathrm{e}}\right) f(O) - \delta K,$$

where we used $f(\emptyset) = 0$. Hence we obtain the theorem from $\mathbb{E}[f(S)] = \mathbb{E}[f(S_K)]$. □

Theorem 2. *If $(V, \mathcal{I})$ is a κ-extendible system with rank K, we have $\mathbb{E}[f(S, \boldsymbol{\theta})] \geq \frac{1}{\kappa+1} f(O, \boldsymbol{\theta}) - \delta K$.*

Proof. For each realization of $S_0 \subset S_1 \subset \cdots \subset S_{|S|} = S \in \mathcal{I}$, we define $S_{|S|+1}, S_{|S|+2}, \dots, S_K$ as S. We thus construct a series of feasible solutions, $S_0, S_1, \dots, S_K$, for every realization. Note that $\mathbb{E}[f(S)] = \mathbb{E}[f(S_K)]$ holds since we always have $S = S_K$.

We consider constructing a series of subsets $O_0, O_1, \dots, O_K$ for each realization of $S_0, S_1, \dots, S_K$. We aim to prove by induction that we can construct such $O_0, O_1, \dots, O_K$ satisfying the following conditions: $O_0 = O$, $S_i \subseteq O_i \in \mathcal{I}$ $(i = 0, \dots, K-1)$, $S_K = O_K \in \mathcal{I}$ for every realization, and

$$\kappa \cdot (\mathbb{E}[f(S_i)] - \mathbb{E}[f(S_{i-1})] + \delta) \geq \mathbb{E}[f(O_{i-1})] - \mathbb{E}[f(O_i)] \tag{A1}$$

for $i = 1, \dots, K$.

In the case of $i = 0$, we let $O_0 = O$, which satisfies $S_0 = \emptyset \subseteq O = O_0 \in \mathcal{I}$. In this case, (A1) is not required to hold.

We assume all random quantities to be conditioned on an arbitrary realization of the $(k-1)$-th step, where $S_0, \dots, S_{k-1}$ and $O_0, \dots, O_{k-1}$ satisfying $S_i \subseteq O_i \in \mathcal{I}$ $(i = 0, \dots, k-1)$ are given. If S_{k-1} is maximal, we let $O_k = S_k$ $(= S_{k-1} = O_{k-1})$, which satisfies $S_k = O_k \in \mathcal{I}$ and

$$\kappa \cdot (\mathbb{E}[f(S_k)] - f(S_{k-1}) + \delta) = \kappa \cdot \delta \geq 0 = f(O_{k-1}) - \mathbb{E}[f(O_k)].$$

If S_{k-1} is not maximal, from the definition of κ-extendible systems, for any choice of $s_k \notin S_{k-1}$, there exists $Z_k \subseteq O_{k-1} \backslash S_{k-1}$ such that $O_{k-1} \backslash Z_k \cup \{s_k\} \in \mathcal{I}$ and $|Z_k| \leq \kappa$ hold. We let $O_k = O_{k-1} \backslash Z_k \cup \{s_k\}$. Note that thus constructed O_k satisfies $S_k \subseteq O_k \in \mathcal{I}$ for any realization of the k-th step; moreover, if $k = K$, we always have $S_K = O_K \in \mathcal{I}$ since S_K is maximal. Considering expectation over realizations of the k-th step, we obtain

$$\begin{aligned}&f(O_{k-1}) - \mathbb{E}[f(O_k)] \\ &= f(O_{k-1}) - \mathbb{E}[f(O_{k-1} \backslash Z_k)] \\ &\qquad + \mathbb{E}[f(O_{k-1} \backslash Z_k)] - \mathbb{E}[f(O_k)]\end{aligned}$$

$$
\begin{aligned}
&\leq \mathbb{E}[f_{O_{k-1}\backslash Z_k}(Z_k)] && \because O_{k-1}\backslash Z_k \subseteq O_k \text{ and monotonicity}\\
&\leq \mathbb{E}\left[\sum_{v\in Z_k} f_{S_{k-1}}(v)\right] && \because S_{k-1} \subseteq O_{k-1}\backslash Z_k \text{ and submodularity}\\
&\leq \kappa\cdot(\mathbb{E}[f_{S_{k-1}}(s_k)]+\delta) && \because Z_k \subseteq O_{k-1}\backslash S_{k-1} \subseteq U_k\text{, Lemma 1, and } |Z_k|\leq\kappa\\
&= \kappa\cdot(\mathbb{E}[f(S_k)]-f(S_{k-1})+\delta)
\end{aligned}
$$

Therefore, in any case we have

$$\kappa\cdot(\mathbb{E}[f(S_k)]-f(S_{k-1})+\delta) \geq f(O_{k-1})-\mathbb{E}[f(O_k)].$$

By taking expectation over all realizations of the $(k-1)$-th step, we obtain (A1) for $i=k$.

For every realization of the k-th step, $O_0,\dots,O_k$ constructed above satisfy $S_i \subseteq O_i \in \mathcal{I}$ for $i=0,\dots,k$ (if $k=K$, we have $S_k = O_k \in \mathcal{I}$). Therefore, the induction hypothesis for proving the statement in the next step (i.e., the $(k+1)$-step) is satisfied. Consequently, (A1) holds for $i=1,\dots,K$ by induction. Summing both sides of (A1) for $i=1,\dots,K$, we obtain

$$\kappa\cdot(\mathbb{E}[f(S_K)]-f(\emptyset)+\delta K) \geq \mathbb{E}[f(O_0)]-\mathbb{E}[f(O_K)].$$

Since we have $f(\emptyset)=0$, $O_0=O$, and $O_K=S_K$ for every realization, it holds that

$$\mathbb{E}[f(S_K)] \geq \frac{1}{\kappa+1}f(O)-\frac{\kappa}{\kappa+1}\delta K \geq \frac{1}{\kappa+1}f(O)-\delta K.$$

Hence we obtain the theorem from $\mathbb{E}[f(S)]=\mathbb{E}[f(S_K)]$. $\square$

C REGULARIZATION FUNCTIONS

We first detail the case where Ω_k is the entropy function. We then present a sufficient condition for satisfying Assumption 2, which is useful when designing regularization functions.

C.1 Entropy Regularization

We consider using the entropy function as a regularization function: $\Omega_k(\mathbf{p}) = \epsilon\sum_{u\in U_k} p(u)\ln p(u)$, where $\epsilon>0$ is a constant that controls the perturbation strength. Note that we have $\Omega_k(\mathbf{p})-\Omega_k(\mathbf{q}) \leq \epsilon\cdot 0-\epsilon\sum_{i=1}^{n_k}\frac{1}{n_k}\ln\frac{1}{n_k} = \epsilon\ln n_k$ for any $\mathbf{p},\mathbf{q}\in\Delta^{n_k}$.

From the relationship between the entropy regularization and softmax, each iteration of Smoothed Greedy can be performed via softmax sampling; below we will see this in detail. From the Karush–Kuhn–Tucker (KKT) condition of problem (2), $\max_{\mathbf{p}\in\Delta^{n_k}}\{\langle\mathbf{g}_k,\mathbf{p}\rangle-\Omega_k(\mathbf{p})\}$, we have

$$\epsilon(\ln\mathbf{p}+\mathbf{1}_{n_k})-\mathbf{g}_k+\mathbf{1}_{n_k}\mu=\mathbf{0}_{n_k} \quad\text{and}\quad \mathbf{1}_{n_k}^\top\mathbf{p}=1, \tag{A2}$$

where ln operates in an element-wise manner and $\mu\in\mathbb{R}$ is a multiplier corresponding to the equality constraint. Note that we need not take the inequality constraints, $\mathbf{p}\geq\mathbf{0}_{n_k}$, into account since the entropy regularization forces every $p(u)$ to be positive. Since Ω_k is strictly convex and every feasible solution satisfies the linear independence constraint qualification (LICQ), the maximizer, $\mathbf{p}_k$, is characterized as the unique solution to the KKT equation system (A2). From (A2), we see that $\mathbf{p}_k$ is proportional to $\exp(\mathbf{g}_k/\epsilon)$. Thus, Steps 4 to 6 in Algorithm 1 can be performed via softmax sampling: $p_k(u,\boldsymbol{\theta}) \propto \exp(f_{S_{k-1}}(u,\boldsymbol{\theta})/\epsilon)$ for $u\in U_k$, which we can compute by making $\mathrm{O}(n_k)$ time queries to $f(\cdot;\boldsymbol{\theta})$.

We then discuss how to compute $\nabla_{\mathbf{g}_k}\mathbf{p}_k(\mathbf{g}_k)$. While this can be done by directly differentiating $p_k(u,\mathbf{g}_k)\propto\exp(g_k(u)/\epsilon)$, we here see how to compute it by applying the implicit function theorem (see, e.g., (Dontchev and Rockafellar, 2014)) to the KKT equation system (A2) as a warm-up for the next section. In this case, the requirements for using the implicit function theorem are satisfied (see the next section). By differentiating the KKT equation system (A2) w.r.t. $\mathbf{g}_k$, we obtain

$$\begin{bmatrix}\epsilon\mathrm{diag}(\mathbf{p}_k)^{-1} & \mathbf{1}_{n_k}\\ \mathbf{1}_{n_k}^\top & 0\end{bmatrix}\begin{bmatrix}\nabla_{\mathbf{g}_k}\mathbf{p}_k\\ \nabla_{\mathbf{g}_k}\mu\end{bmatrix}=\begin{bmatrix}\mathbf{I}_{n_k}\\ \mathbf{0}_{n_k}^\top\end{bmatrix},$$

where $\mathrm{diag}(\mathbf{p}_k)$ is a diagonal matrix whose diagonal entries are $\mathbf{p}_k$ and $\mathbf{I}_{n_k}$ is the $n_k \times n_k$ identity matrix. We can compute $\nabla_{\mathbf{g}_k}\mathbf{p}_k$ by solving the above equation as follows:

$$\begin{bmatrix}\nabla_{\mathbf{g}_k}\mathbf{p}_k \\ \nabla_{\mathbf{g}_k}\mu\end{bmatrix} = \begin{bmatrix}\epsilon\mathrm{diag}(\mathbf{p}_k)^{-1} & \mathbf{1}_{n_k} \\ \mathbf{1}_{n_k}^\top & 0\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{I}_{n_k} \\ \mathbf{0}_{n_k}^\top\end{bmatrix} = \begin{bmatrix}\epsilon^{-1}(\mathrm{diag}(\mathbf{p}_k) - \mathbf{p}_k\mathbf{p}_k^\top) \\ \mathbf{p}_k^\top\end{bmatrix}.$$

Note that once we obtain $\nabla_{\boldsymbol{\theta}}\mathbf{g}_k(\boldsymbol{\theta})$, we can compute the desired derivative, $\nabla_{\boldsymbol{\theta}}\mathbf{p}_k(\boldsymbol{\theta}) = \nabla_{\mathbf{g}_k}\mathbf{p}_k(\mathbf{g}_k)\cdot\nabla_{\boldsymbol{\theta}}\mathbf{g}_k(\boldsymbol{\theta}) = \epsilon^{-1}(\mathrm{diag}(\mathbf{p}_k) - \mathbf{p}_k\mathbf{p}_k^\top)\nabla_{\boldsymbol{\theta}}\mathbf{g}_k(\boldsymbol{\theta})$, by matrix-vector products in $\mathrm{O}(n_k \times \dim\Theta)$ time.

Connection to Entropy-Regularized Optimal Transport One may get interested in the link between problem (2) with the entropy regularization and the optimal transport (OT) with entropy regularization (Cuturi, 2013). While (2) has a vector variable with one equality constraint, OT has a matrix variable with two equality constraints; in this sense, (2) considers a simpler setting. Thanks to the simplicity, we can analyze the theoretical guarantees of Smoothed Greedy. By contrast, if we consider using OT, we can employ more sophisticated operations (e.g., ranking and sorting (Cuturi et al., 2019)) than argmax. In return for this, however, it becomes more difficult to prove approximation guarantees; for example, how to design appropriate transportation costs is non-trivial. This OT-based approach to designing differentiable combinatorial optimization algorithms will be an interesting research direction, which we leave for future work.

C.2 Sufficient Condition for Satisfying Differentiability Assumption

We study the case where Ω_k is a general strictly convex differentiable function; although a similar discussion is presented in (Amos and Kolter, 2017) for the case where Ω_k is quadratic, we here provide a detailed analysis with general Ω_k for completeness. The KKT condition of problem (2) can be written as

$$\nabla_{\mathbf{p}}\Omega_k(\mathbf{p}) - \mathbf{g}_k - \boldsymbol{\lambda} + \mathbf{1}_{n_k}\mu = \mathbf{0}_{n_k}, \qquad \boldsymbol{\lambda}\odot\mathbf{p} = \mathbf{0}_{n_k}, \qquad \text{and} \qquad \mathbf{1}_{n_k}^\top\mathbf{p} = 1,$$

where $\boldsymbol{\lambda} \geq \mathbf{0}_{n_k}$ consists of multipliers corresponding to the inequality constraints (i.e., $\mathbf{p} \geq \mathbf{0}_{n_k}$) and $\odot$ denotes the element-wise product. Since every feasible point in Δ^{n_k} satisfies LICQ, if Ω_k is strictly convex on Δ^{n_k}, the optimal solution is uniquely characterized by the KKT condition. Let $(\tilde{\mathbf{p}}, \tilde{\boldsymbol{\lambda}}, \tilde{\mu})$ be a triplet that satisfies the KKT condition, where $\tilde{\mathbf{p}} = \mathbf{p}_k$. If the following three conditions hold, $\nabla_{\mathbf{g}_k}\mathbf{p}_k$ can be calculated from the KKT condition as detailed later:

1. Ω_k is twice-differentiable,
2. the Hessian, $\nabla^2_{\mathbf{p}}\Omega_k(\mathbf{p})$, is positive definite for any $\mathbf{p} \in \Delta^{n_k}$, and
3. the strict complementarity, $\tilde{\boldsymbol{\lambda}} + \tilde{\mathbf{p}} > \mathbf{0}_{n_k}$, holds at the unique optimum, $\tilde{\mathbf{p}} = \mathbf{p}_k$.

Note that the second condition implies the strict convexity of Ω_k on Δ^{n_k}. Therefore, a sufficient condition for satisfying Assumption 2 is given by the above three conditions. In practice, given any twice-differentiable convex function Ω', we can obtain a regularization function Ω_k that satisfies the sufficient condition by adding the entropy function multiplied by a small constant to Ω'.

We then explain how to compute $\nabla_{\mathbf{g}_k}\mathbf{p}_k$. Let $\tilde{U}$ be a subset of U_k such that $\tilde{p}(u) = 0$ iff $u \in \tilde{U}$; the strict complementarity implies $\tilde{\lambda}(u) = 0$ iff $u \notin \tilde{U}$. We let $\mathbf{x} \coloneqq (\mathbf{p}, \boldsymbol{\lambda}_{\tilde{U}}, \mu)$, where $\boldsymbol{\lambda}_{\tilde{U}}$ is a $|\tilde{U}|$-dimensional vector consisting of the entries of $\boldsymbol{\lambda}$ corresponding to $\tilde{U}$. We define $\mathbf{I}_{\tilde{U}}$ as the $n_k \times |\tilde{U}|$ matrix that has columns of $\mathbf{I}_{n_k}$ corresponding to $\tilde{U}$. The KKT equation system at $\tilde{\mathbf{x}} = (\tilde{\mathbf{p}}, \tilde{\boldsymbol{\lambda}}_{\tilde{U}}, \tilde{\mu})$ can be written as

$$H(\mathbf{x}, \mathbf{g}_k) \coloneqq \begin{bmatrix}\nabla_{\mathbf{p}}\Omega_k(\mathbf{p}) - \mathbf{g}_k - \mathbf{I}_{\tilde{U}}\boldsymbol{\lambda}_{\tilde{U}} + \mathbf{1}_{n_k}\mu \\ -\mathbf{I}_{\tilde{U}}^\top\mathbf{p} \\ \mathbf{1}_{n_k}^\top\mathbf{p} - 1\end{bmatrix} = \begin{bmatrix}\mathbf{0}_{n_k} \\ \mathbf{0}_{|\tilde{U}|} \\ 0\end{bmatrix},$$

and its partial Jacobians at $\tilde{\mathbf{x}}$ are given by

$$\nabla_{\mathbf{x}}H(\tilde{\mathbf{x}}, \mathbf{g}_k) = \begin{bmatrix}\nabla^2_{\mathbf{p}}\Omega_k(\tilde{\mathbf{p}}) & -\mathbf{I}_{\tilde{U}} & \mathbf{1}_{n_k} \\ -\mathbf{I}_{\tilde{U}}^\top & \multicolumn{2}{c}{\multirow{2}{*}{$\mathbf{0}_{|\tilde{U}|+1\times|\tilde{U}|+1}$}} \\ \mathbf{1}_{n_k}^\top & & \end{bmatrix} \qquad \text{and} \qquad \nabla_{\mathbf{g}_k}H(\tilde{\mathbf{x}}, \mathbf{g}_k) = \begin{bmatrix}-\mathbf{I}_{n_k} \\ \mathbf{0}_{|\tilde{U}|\times n_k} \\ \mathbf{0}_{n_k}^\top\end{bmatrix}.$$

Note that $|\tilde{U}| < n_k$ always holds; otherwise $\tilde{\mathbf{p}} = \mathbf{0}_{n_k}$, which is an infeasible solution. Therefore, $[-\mathbf{I}_{\tilde{U}} \ \mathbf{1}_{n_k}]$ always has rank $|\tilde{U}| + 1$. From the positive definiteness of $\nabla^2_{\mathbf{p}} \Omega_k(\tilde{\mathbf{p}})$, we have

$$\det\left(\nabla_{\mathbf{x}} H(\tilde{\mathbf{x}}, \mathbf{g}_k)\right) = \det\left(\nabla^2_{\mathbf{p}} \Omega_k(\tilde{\mathbf{p}})\right) \det\left(-[-\mathbf{I}_{\tilde{U}} \ \mathbf{1}_{n_k}]^\top \nabla^2_{\mathbf{p}} \Omega_k(\tilde{\mathbf{p}})^{-1} [-\mathbf{I}_{\tilde{U}} \ \mathbf{1}_{n_k}]\right) \neq 0,$$

where we used the Schur complement. Hence $\nabla_{\mathbf{x}} H(\tilde{\mathbf{x}}, \mathbf{g}_k)$ is non-singular. This guarantees that $\nabla_{\mathbf{g}_k} \mathbf{p}_k$ can be computed by using the implicit function theorem as follows (see, e.g., (Dontchev and Rockafellar, 2014)):

$$\begin{bmatrix} \nabla_{\mathbf{g}_k} \mathbf{p}_k \\ \nabla_{\mathbf{g}_k} \tilde{\boldsymbol{\lambda}}_{\tilde{U}} \\ \nabla_{\mathbf{g}_k} \tilde{\mu} \end{bmatrix} = -\nabla_{\mathbf{x}} H(\tilde{\mathbf{x}}, \mathbf{g}_k)^{-1} \nabla_{\mathbf{g}_k} H(\tilde{\mathbf{x}}, \mathbf{g}_k).$$

Thus, once the KKT triplet, the Hessian, and $\nabla_{\boldsymbol{\theta}} \mathbf{g}_k(\boldsymbol{\theta})$ are obtained, we can compute $\nabla_{\boldsymbol{\theta}} \mathbf{p}_k(\boldsymbol{\theta}) = \nabla_{\mathbf{g}_k} \mathbf{p}_k(\mathbf{g}_k) \cdot \nabla_{\boldsymbol{\theta}} \mathbf{g}_k(\boldsymbol{\theta})$ in $\mathrm{O}({n_k}^3 + {n_k}^2 \times \dim \Theta)$ time in general. For speeding up this step, we can reduce the n_k value by using the stochastic version of the greedy algorithm (Mirzasoleiman et al., 2015) (see, Appendix F.2).

A recent result (Stechlinski et al., 2018) provides an extended version of the implicit function theorem, which may enable us to deal with a wider class of Ω_k. We leave further studies on this topic for future work.

D DISCUSSION ON OTHER GRADIENT ESTIMATORS

The score-function gradient estimator is one of major Monte Carlo gradient estimators. Other than that, the pathwise and measure-valued gradient estimators are widely used (see, (Mohamed et al., 2019) for a survey). The Gumbel-Softmax estimator (Jang et al., 2017; Maddison et al., 2017) has also been used in many recent studies. We discuss why it is difficult to apply those estimators to our case.

The pathwise gradient estimators basically use derivatives of quantities inside the expectation. In our case, however, we cannot differentiate the quantity, $Q(S)$, w.r.t. S since the domain is non-continuous.

The measure-valued gradient estimators require us to decompose $\nabla_{\boldsymbol{\theta}} p(\boldsymbol{\theta})$ into $p^+(\boldsymbol{\theta})$ and $p^-(\boldsymbol{\theta})$, which must satisfy the following conditions: both $p^+(\boldsymbol{\theta})$ and $p^-(\boldsymbol{\theta})$ form some probability distribution functions, and $\nabla_{\boldsymbol{\theta}} p(\boldsymbol{\theta}) = c_{\boldsymbol{\theta}}(p^+(\boldsymbol{\theta}) - p^-(\boldsymbol{\theta}))$ holds with some constant $c_{\boldsymbol{\theta}}$. Once we obtain a decomposition satisfying these conditions, we can estimate the gradient by sampling from $p^+(\boldsymbol{\theta})$ and $p^-(\boldsymbol{\theta})$. It is known that we can obtain such a decomposition when $p(\boldsymbol{\theta})$ has certain structures, e.g., Poisson and Gaussian. In our case, however, $p(\boldsymbol{\theta})$ is the output distribution, whose complicated structure makes it quite difficult to obtain a valid decomposition of $p(\boldsymbol{\theta})$.

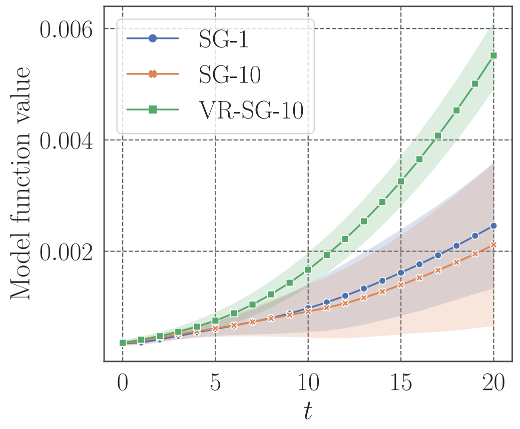
The Gumbel-Softmax estimator is obtained by continuously interpolating discrete categorical distributions (defined on Δ^{n_k} in our case) and computing derivatives at interior points. In our case, however, we must obtain an extreme point, S_{k-1}, in the $(k-1)$-th step to compute the categorical distribution $\mathbf{p}_k(\boldsymbol{\theta})$ used in the k-th step. That is, unlike the cases of (Jang et al., 2017; Maddison et al., 2017), SMOOTHED GREEDY needs to sequentially sample from categorical distributions, which is specified depending on the past samples. Consequently, the continuous interpolation for a single step does not work for smoothing the sequential argmax; hence we cannot apply the Gumbel-Softmax estimator to our setting.

E LEARNING SUBMODULAR MODELS WITH LIMITED ORACLE QUERIES

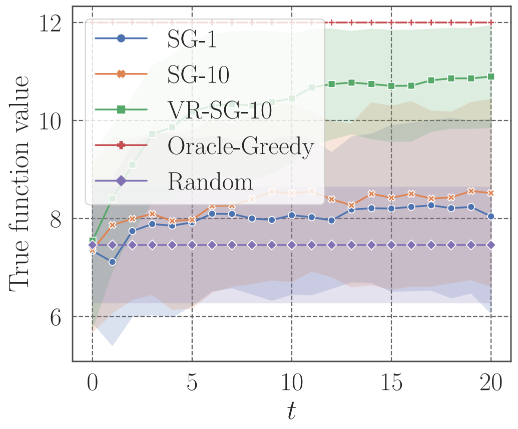
We discuss the application of our framework to learning of parameterized submodular functions with limited oracle queries. In the experiment, we consider learning deep submodular functions (Dolhansky and Bilmes, 2016).

E.1 Problem Description

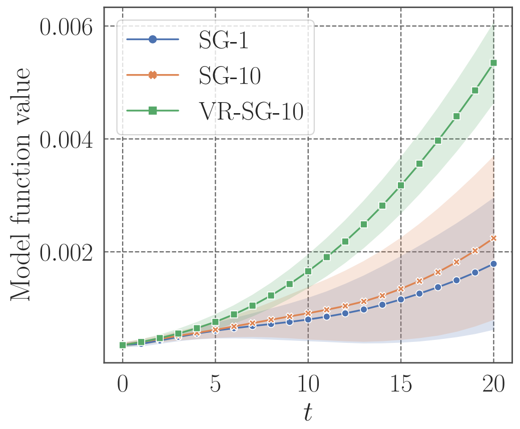
We consider maximizing unknown submodular function $\tilde{f}(\cdot)$ by sequentially querying its values. Specifically, in each t-th round, we can query $\tilde{f}(\cdot)$ values at N points $S_1, \ldots, S_N \in \mathcal{I}$, and by using this feedback, we seek a good



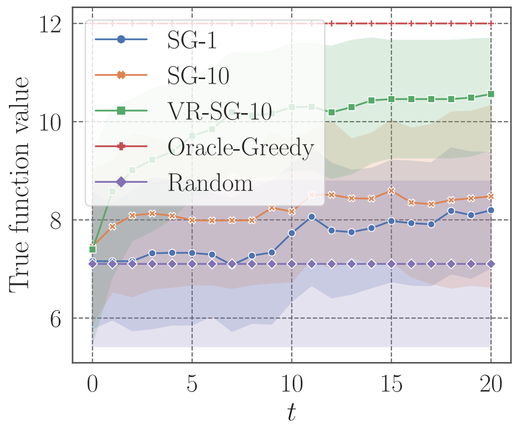
(a) Noise-free



(b) Noise-free



(c) Noisy



(d) Noisy

Figure 3: Mode and true function values for noise-free (upper) and noisy (lower) settings.

solution for maximizing $\tilde{f}(\cdot)$. We suppose that no prior knowledge on the true function, $\tilde{f}(\cdot)$, other than the fact that it is normalized, monotone, and submodular, is available and that to query the true function value is costly and time-consuming. We want to achieve high $\tilde{f}(\cdot)$ values with a small number of rounds and queries. One can think of this setting as a variant of submodular maximization with low adaptive complexities (Balkanski and Singer, 2018) or online submodular maximization with bandit feedback (Zhang et al., 2019).

We consider the following approach: we construct some parameterized submodular model $f(\cdot, \boldsymbol{\theta})$, e.g., a deep submodular function, and update $\boldsymbol{\theta}$ by using our gradient estimators with $Q(S_j) = \tilde{f}(S_j)$. That is, akin to the decision-focused approach described in Section 4.2, we train $f(\cdot, \boldsymbol{\theta})$ so that the greedy algorithm can achieve high $\tilde{f}(\cdot)$ values; the current setting is more difficult since we know nothing about $\tilde{f}(\cdot)$ in advance and features $\mathbf{X}$, which are used by the predictive models, are unavailable.

E.2 Experiments

We consider a situation where we make contact with business leaders to make influences on their companies. We use the corporate leadership network dataset of KONECT (Barnes and Burkett, 2010; Kunegis, 2013), which contains person–company leadership information between 20 people and 24 companies; the companies are indexed with $i = 1, \dots, 24$. We let each $v \in V$ represent a person, who is associated with a subset of companies $I_v \subseteq \{1, \dots, 24\}$. We define $I_X \coloneqq \bigcup_{v \in X} I_v$ for every $X \subseteq V$. We express the importance of the i-th company with a non-negative weight w_i; we let $w_1 = w_3 = \cdots = w_{23} = 1$ and $w_2 = w_4 = \cdots = w_{24} = 0.1$. We use the weighted coverage function as an unknown true function: $\tilde{f}(X) \coloneqq \sum_{i \in I_X} w_i$. We separate the 20 people into two groups of 10 people, and we choose up to two people from each group; i.e., $(V, \mathcal{I})$ forms a partition matroid.

As a model function, $f(\cdot, \boldsymbol{\theta})$, we use a deep submodular function that forms a 2-layer NN. We set the hidden-layer size at 50 and use sigmoid activation functions. We set initial NN parameters $\boldsymbol{\theta}$ at non-negative values drawn uniformly at random from $[0, 0.01]$.

For $t = 1, \dots, 20$, we perform Smoothed Greedy N times with objective function $f(\cdot, \boldsymbol{\theta})$; we thus obtain $S_1, \dots, S_N \in \mathcal{I}$. We then query $\tilde{f}(S_1), \dots, \tilde{f}(S_N)$ values, with which we compute the gradient estimator, and we update $\boldsymbol{\theta}$ by using Adam with learning rate 10^{-3}. In each t-th round, we evaluate the quality of the trained model, $f(\cdot, \boldsymbol{\theta})$, as follows: we obtain $S \in \mathcal{I}$ by applying the greedy algorithm to $f(\cdot, \boldsymbol{\theta})$, and compute model function value $f(S, \boldsymbol{\theta})$ and true function value $\tilde{f}(S)$. We also consider a noisy setting where observed $\tilde{f}(S_j)$ values are perturbed with random variables drawn from the standard normal distribution. We here use the entropy function with $\epsilon = 0.02$ as a regularization function of Smoothed Greedy.

As in Section 5.2, (**VR-**)**SG-**N stands for (variance-reduced) Smoothed Greedy with N samples. We compare the true function values of (**VR-**)**SG-**N with those of two methods: **Oracle-Greedy** and **Random**. **Oracle-Greedy** is the greedy algorithm directly applied to $\tilde{f}(\cdot)$, which we assume to be unknown in this setting; we use **Oracle-Greedy** to see what if we had full access to the unknown true $\tilde{f}(\cdot)$. **Random** returns $X \in \mathcal{I}$ by randomly choosing two people from each of the two groups.

Figure 3 presents the means and standard deviations of the model and true function values over 30 runs. We see that, by updating the model function parameter $\boldsymbol{\theta}$ with our gradient estimator, we can successfully increase the true function values. As indicated by the results of **SG-**1, if even once we can query $\tilde{f}(\cdot)$ value in each round, we can do better than **Random**. With more queries and VR, we can achieve higher true function values. The results suggest that our method is useful for learning and maximizing submodular functions when very limited prior knowledge and feedback are available.

F DIFFERENTIABLE STOCHASTIC GREEDY ALGORITHM

We show that our framework can be used for making the stochastic greedy algorithm (Mirzasoleiman et al., 2015) differentiable, which is a faster randomized variant of the greedy algorithm. In this section, we focus on the cardinality constrained case.

Algorithm 2 presents the smoothed version of the stochastic greedy algorithm, which we call Stochastic Smoothed Greedy. The only difference from Smoothed Greedy (Algorithm 1) is in Step 3, where we sample n_k elements uniformly at random without replacement from $V \backslash S$. In what follows, we let $n_k = \left\lceil \frac{n}{K} \ln \frac{1}{\varepsilon} \right\rceil$ for every $k = 1, \dots, K$, where $\varepsilon \in (0, 1)$ is a hyper-parameter controlling the speed–accuracy trade-off.

As with the original stochastic greedy algorithm, Stochastic Smoothed Greedy requires only $\mathrm{O}(n \ln 1/\varepsilon)$ evaluations of $f(\cdot, \boldsymbol{\theta})$, while Smoothed Greedy requires $\mathrm{O}(nK)$. Moreover, as explained in Appendix F.2, the gradient estimator for Stochastic Smoothed Greedy can be computed more efficiently than that for Smoothed Greedy. Therefore, Stochastic Smoothed Greedy is useful when n and K are large and/or the evaluation of $f(\cdot, \boldsymbol{\theta})$ is costly. Below we prove the approximation guarantee of Stochastic Smoothed Greedy, and explain how to compute gradient estimators. We also present experiments to see the empirical speed–accuracy trade-off.

F.1 Approximation Guarantee

We prove that Algorithm 2 returns solution S that satisfies the following approximation guarantee for the cardinality constrained case.

Theorem 3. *If $n_k \geq \frac{n}{K} \ln \frac{1}{\varepsilon}$ for $k = 1, \dots, K$, we have $\mathbb{E}[f(S, \boldsymbol{\theta})] \geq (1 - 1/\mathrm{e} - \varepsilon) f(O, \boldsymbol{\theta}) - \delta K$.*

Proof. As with the proofs in Appendix B, we omit $\boldsymbol{\theta}$ and use S_k to denote the solution obtained in the k-th step $(k = 0, \dots, K)$. We take all random quantities to be conditioned on the realization of the $(k-1)$-th step. Once U_k is fixed in Step 3, we can obtain the following inequality from Lemma 1:

$$\mathbb{E}[f_{S_{k-1}}(s_k) \mid U_k] \geq f_{S_{k-1}}(s_k^*) - \delta,$$

Algorithm 2 Stochastic Smoothed Greedy

```
1: $S \leftarrow \emptyset$
2: for $k = 1, 2 \dots, K$ do
3:     $U_k = \{u_1, \dots, u_{n_k}\} \leftarrow$ $n_k$ elements chosen from $V \backslash S$ uniformly at random
4:     $\mathbf{g}_k(\boldsymbol{\theta}) = (g_k(u_1, \boldsymbol{\theta}), \dots, g_k(u_{n_k}, \boldsymbol{\theta})) \leftarrow (f_S(u_1, \boldsymbol{\theta}), \dots, f_S(u_{n_k}, \boldsymbol{\theta}))$
5:     $\mathbf{p}_k(\boldsymbol{\theta}) = (p_k(u_1, \boldsymbol{\theta}), \dots, p_k(u_{n_k}, \boldsymbol{\theta})) \leftarrow \mathrm{argmax}_{\mathbf{p} \in \Delta^{n_k}} \{\langle \mathbf{g}_k(\boldsymbol{\theta}), \mathbf{p}\rangle - \Omega_k(\mathbf{p})\}$
6:     $s_k \leftarrow u \in U_k$ with probability $p_k(u, \boldsymbol{\theta})$
7:     $S \leftarrow S \cup \{s_k\}$
   return $S$
```

where $s_k^* \in \mathrm{argmax}_{u \in U_k} f_{S_{k-1}}(u)$ and $\mathbb{E}[\cdot \mid U_k]$ denotes the expectation conditioned on U_k. By taking the expectation over all possible choices of U_k, we obtain

$$\mathbb{E}[f(S_k)] - f(S_{k-1}) = \mathbb{E}[f_{S_{k-1}}(s_k)] \geq \mathbb{E}[f_{S_{k-1}}(s_k^*)] - \delta. \tag{A3}$$

Here, note that s_k^* is a random variable representing an element, which the original stochastic greedy algorithm adds to the current solution. As proved in (Mirzasoleiman et al., 2015), if $n_k \geq \frac{n}{K} \ln \frac{1}{\varepsilon}$, we have

$$\mathbb{E}[f_{S_{k-1}}(s_k^*)] \geq \frac{1-\varepsilon}{K}(f(O \cup S_{k-1}) - f(S_{k-1})).$$

By substituting this inequality into (A3) and taking the expectation over all possible realizations of S_{k-1}, we obtain

$$\mathbb{E}[f(S_k)] - \mathbb{E}[f(S_{k-1})] \geq \frac{1-\varepsilon}{K}(\mathbb{E}[f(O \cup S_{k-1})] - \mathbb{E}[f(S_{k-1})]) - \delta,$$

which holds for $k = 1, \dots, K$. Therefore, by induction, we obtain the theorem as follows:

$$\mathbb{E}[f(S)] \geq \left(1 - \left(1 - \frac{1-\varepsilon}{K}\right)^K\right) f(O) - \delta \sum_{k=0}^{K-1} \left(1 - \frac{1-\varepsilon}{K}\right)^k \geq \left(1 - \frac{1}{\mathrm{e}} - \varepsilon\right) f(O) - \delta K,$$

where we used $\mathbb{E}[f(O \cup S_{k-1})] \geq f(O)$, $f(\emptyset) = 0$, and $\mathbb{E}[f(S)] = \mathbb{E}[f(S_K)]$. $\square$

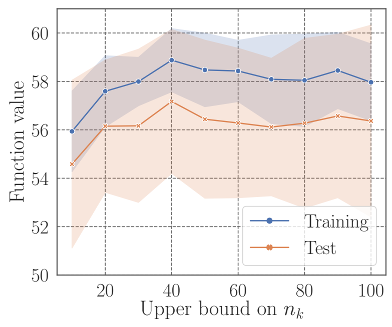
F.2 Gradient Estimation

We show how to compute gradient estimators for Stochastic Smoothed Greedy. As with the case of Smoothed Greedy, outputs of Stochastic Smoothed Greedy are distributed over $\mathscr{S}_{\mathcal{I}}$. Therefore, the score-function gradient estimator can be computed by sampling outputs as in Section 3, i.e.,

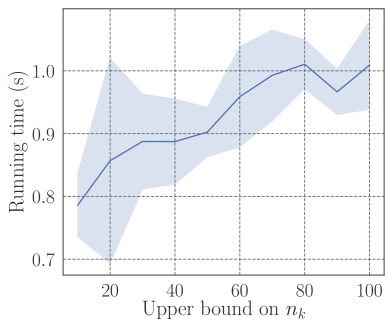
$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{S \sim p(\boldsymbol{\theta})}[Q(S)] \approx \frac{1}{N} \sum_{j=1}^{N} Q(S_j) \otimes \nabla_{\boldsymbol{\theta}} \ln p(S_j, \boldsymbol{\theta}) \quad \text{where} \quad S_j = (s_1, \dots, s_{|S_j|}) \sim p(\boldsymbol{\theta}).$$

Note that here $p(\boldsymbol{\theta})$ denotes the output distribution of the Stochastic Smoothed Greedy; more precisely, for $\mathbf{p}_1(\boldsymbol{\theta}), \dots, \mathbf{p}_K(\boldsymbol{\theta})$ and solution $S = \{s_1, \dots, s_K\}$ computed by Algorithm 2, we let $p(S, \boldsymbol{\theta}) = \prod_{k=1}^{|S|} p_k(s_k, \boldsymbol{\theta})$, where $p_k(s_k, \boldsymbol{\theta})$ is the entry of $\mathbf{p}_k(\boldsymbol{\theta})$ corresponding to $s_k \in U_k$. We can compute $\nabla_{\boldsymbol{\theta}} \ln p(S_j, \boldsymbol{\theta})$ in the same manner as in Section 3.

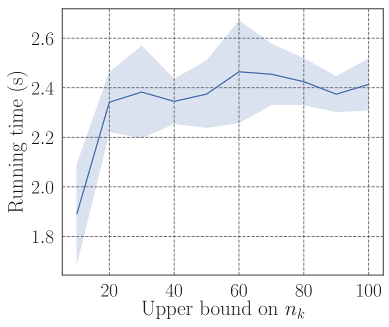
Remember that the computation of $\nabla_{\boldsymbol{\theta}} \ln p(S_j, \boldsymbol{\theta})$ involves the following differentiation based on the chain rule: $\nabla_{\boldsymbol{\theta}} \mathbf{p}_k(\boldsymbol{\theta}) = \nabla_{\mathbf{g}_k} \mathbf{p}_k(\mathbf{g}_k) \cdot \nabla_{\boldsymbol{\theta}} \mathbf{g}_k(\boldsymbol{\theta})$. Here, the dimensionality of $\mathbf{p}_k$ and $\mathbf{g}_k$ is at most $n_k = \left\lceil \frac{n}{K} \ln \frac{1}{\varepsilon} \right\rceil$, while it is up to n in the case of Smoothed Greedy. Therefore, Stochastic Smoothed Greedy is effective for speeding up the computation of gradient estimators.



(a) Training and test scores



(b) Solution computation



(c) Gradient estimation

Figure 4: (a): Objective function values achieved for the training and test instances. (b): Running times of Stochastic Smoothed Greedy for updating $\mathbf{w}$ once (summation over 200 runs). (c): Running times of the gradient-estimator computation.

F.3 Experiments

We study the empirical performance of Stochastic Smoothed Greedy. We use the same settings as those of the decision-focused learning experiments with $K = 10$ (see, Section 5.2), where we have $n = 100$. We apply the Stochastic Smoothed Greedy version of **VR-SG-**10 to the instances. We consider various upper-bound values, $10, 20, \dots, 100$, on n_k; that is, in Step 3 of Algorithm 2, we set n_k at the upper-bound value if it is less than $|V \setminus S|$ and at $|V \setminus S|$ otherwise.

We evaluate objective function values with training and test instances for each upper bound on n_k, where we calculate the means and standard deviations over 30 training/test splits as in Section 5.2. We also observe running times required for computing solutions with Stochastic Smoothed Greedy and estimating gradients. More precisely, we measure those times taken for once updating the predictive-model parameter, $\mathbf{w}$; since the mini-batch size is 20 and we perform $N = 10$ trials, we take the sum of times over 200 runs for obtaining the running time of Stochastic Smoothed Greedy. In this experiment, $\mathbf{w}$ is updated 600 times in total; this is because we have $80/20 = 4$ mini-batches for each of 5 epochs, and we consider 30 random training/test splits, hence $4 \times 5 \times 30 = 600$. The running times of Stochastic Smoothed Greedy and gradient estimation are indicated with means and standard deviations over the 600 iterations.

As shown in Figure 4a, even if n_k decreases, the objective function values do not drop so much with both training and test instances; rather, the highest values are achieved with $n_k = 40$. The results imply that the stochastic greedy algorithm remains empirically effective even if it is smoothed with our framework. Figures 4b and 4c confirm that by decreasing n_k, we can reduce the running times required for computing solutions and estimating gradients. In this experimental setting, since the instance size is not so large and objective function values can be efficiently computed via matrix-vector products, the run-time overhead becomes dominant; this makes the degree of the speed-up yielded by decreasing n_k appears less significant. However, when instance sizes are larger and evaluations of objective functions are more costly, the speed-up achieved by using Stochastic Smoothed Greedy would become more significant.