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 $\kappa$-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 $f(\cdot, \theta) : 2^V \to \mathbb{R}$, where $V$ is a finite set ($n := |V|$), are modeled with continuous-valued parameter $\theta \in \Theta$. For example, $f$ representing influence spread has link probability parameter $\theta$. In this paper, we consider the following parametric submodular maximization:

$$\max_{X \subseteq V} f(X, \theta) \quad \text{subject to} \quad X \in \mathcal{I},$$

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

Once $\theta$ 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 $\theta$ largely deviates from unknown true $\tilde{\theta}$, the greedy algorithm may return a poor solution to the problem of maximizing $f(\cdot, \tilde{\theta})$. This motivates us to study how changes in $\theta$ values affect outputs of the greedy algorithm. Furthermore, it is desirable if we can learn $\theta$ from data so that the greedy algorithm can achieve high $f(\cdot, \tilde{\theta})$ values.

A major approach to studying such subjects is to differentiate outputs of algorithms w.r.t. $\theta$. 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 $\theta$ 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 $\theta$, 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 $\text{argmax}$ 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:

1. 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 guar-
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 $(1 - 1/e)$- and $\frac{1}{n+1}$-approximation guarantees in expectation for the cases of cardinality and $\kappa$-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 $\theta$ 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 $(1 - 1/e)$-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 $\frac{1}{n+1}$-approximation if $(V, \mathcal{I})$ is an intersection of $\kappa$ matroids; later, this result was extended to the class of $\kappa$-systems (Calinescu et al., 2011), which includes $\kappa$-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 $f(\cdot, \theta)$ 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 (Pogancić 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 $f_X(Y) := f(X \cup Y) - f(X)$. 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, $f(\cdot, \theta)$, to be normalized, monotone, and submodular for any $\theta \in \Theta$. Note that this is the case with many set functions, e.g.,
We first explain details of Algorithm 1 Smoothed Greedy.

Algorithm 1 Smoothed Greedy
1: $S ← ∅$
2: for $k = 1, 2, \ldots$ do
3: $U_k = \{u_1, \ldots, u_{n_k}\} \leftarrow \{v \in \mathbb{R}^n \mid v \notin S \cup \{v\} \in \mathcal{I}\}$
4: $g_k(\theta) \leftarrow (f_S(u_1, \theta), \ldots, f_S(u_{n_k}, \theta))$
5: $p_k(\theta) \leftarrow \arg\max_{p \in \Delta^n_k} \{g_k(\theta), p) - \Omega_k(p)\}$
6: $s_k \leftarrow u \in U_k$ with probability $p_k(u, \theta)$
7: $S \leftarrow S \cup \{s_k\}$
8: if $S$ is maximal then
9: return $S$

We say $(V, \mathcal{I})$ is a $\kappa$-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 $Z \subseteq Y \setminus X$ that satisfies $|Z| \leq \kappa$ and $Y \setminus Z \cup \{v\} \in \mathcal{I}$. 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 $\kappa$ matroids defined on a common ground set always forms a $\kappa$-extendible system. We say $X \in \mathcal{I}$ is maximal if no $Y \in \mathcal{I}$ strictly includes $X$. We define $K := \max_{X \in \mathcal{I}} |X|$, which is so-called the rank of $(V, \mathcal{I})$.

For any positive integer $n$, we let $0_n$ and $1_n$ be $n$-dimensional all-zero and all-one vectors, respectively. For any finite set $V$ and $S \subseteq V$, we let $1_S \in \mathbb{R}^{|V|}$ 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 $f : \mathbb{R}^m \rightarrow \mathbb{R}^{m \times n}$ denotes its gradient or Jacobian, respectively.

2 SMOOTHED GREEDY

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

$$p_k(\theta) = \arg\max_{p \in \Delta^n_k} \{g_k(\theta), p) - \Omega_k(p)\}, \quad (2)$$

where $\Delta^n_k := \{x \in \mathbb{R}^{n_k} \mid x \geq 0_{n_k}, \langle x, 1_{n_k}\rangle = 1\}$ is the $(n_k - 1)$-dimensional probability simplex and $\Omega_k : \mathbb{R}^{n_k} \rightarrow \mathbb{R}$ is a strictly convex function; we call $\Omega_k$ a regularization function. Note that $p_k(\theta)$ is unique due to the strict convexity. We then choose an element, $u \in U_k$, with probability $p_k(u, \theta)$ 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 $\Omega_k$, we have $s_k \in \arg\max_{u \in U_k} f_S(u, \theta)$.

We then show theoretical guarantees of Smoothed Greedy (see, Appendix B for proofs). Let $\delta \geq 0$ be a constant that satisfies $\delta \geq \Omega_k(p) - \Omega_k(q)$ for all $k = 1, \ldots, |S|$ and $p, q \in \Delta^n_k$. As show later, smaller $\delta$ values yield better guarantees. We present examples of $\Omega_k$ and their $\delta$ 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 $\delta$ 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., $S = \{s_1, \ldots, s_{k-1}\}$ is arbitrarily fixed), we have $\mathbb{E}\left[f_S(s_k, \theta)\right] \geq f_S(u, \theta) - \delta K$ 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 $\mathcal{I} = \{X \subseteq V \mid |X| \leq K\}$, we have $\mathbb{E}\left[f(S, \theta)\right] \geq (1 - 1/e)f(O, \theta) - \delta K$.

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

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

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

Existing guarantees (Tschiatschek et al., 2018; Powers...
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 functions $\Omega_k$: 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 $\Omega_k$, an additional differentiability condition (see, Assumption 2 in Section 3) must be satisfied for making expected outputs of Smoothed Greedy differentiable.

### Entropy Function
If $\Omega_k(p) = \epsilon \sum_{i=1}^{n_k} p(u_i) \ln p(u_i)$, where $p(u_i)$ is the $i$-th entry of $p \in [0, 1]^{n_k}$ and $\epsilon > 0$ is an arbitrary constant, we have $\delta = \epsilon \ln n_k$. That is, the hyper-parameter, $\epsilon$, controls the $\delta$ 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 $p_k(u, \theta) \propto \exp(f_k(u, \theta))/\epsilon$ (see, Appendix C.1 for details).

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

As above, the $\delta$ 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, $\delta$ 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. $\theta$. 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 known 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 $f(X, \theta)$ to be differentiable w.r.t. $\theta$.

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 $\Omega_k$ (thus, it is rather a requirement when designing $\Omega_k$ than an assumption, but we here state it as an assumption for convenience).

**Assumption 2.** Let $p_k(g_k)$ be the maximizer, $p_k(\theta)$, in (2) regarded as a function of $g_k(\theta)$. We assume $p_k(g_k)$ to be differentiable w.r.t. $g_k$.

For example, if $\Omega_k$ is the entropy function, we have $\nabla g_k p_k(g_k) = -1/\epsilon (\text{diag}(\nabla f_k(u, g_k)) - p_k g_k) p_k(g_k)^\top$; 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 $\Omega_k$ to satisfy Assumption 2.

We then introduce the probability distribution of Smoothed Greedy outputs.\(^3\)

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

We let $p(S, \theta) \in [0, 1]$ denote the probability that $S \in \mathcal{I}$ is returned by Smoothed Greedy. Specifically, if it returns a sequence $S = (s_1, \ldots, s_{|S|}) \in \mathcal{I}$, we have $p(S, \theta) = \prod_{k=1}^{|S|} p_k(s_k, \theta)$, where $p_k(s_k, \theta)$ is the entry of $p_k(\theta)$ 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 $\nabla_\theta \mathbf{E}_{S \sim p(\theta)}[Q(S)] = \mathbf{E}_{S \sim \mathcal{I}}[Q(S) \nabla_\theta p(S, \theta)]$. Since the size of $\mathcal{I}$ is exponential in $K = O(n)$, 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 $S_j = (s_1, \ldots, s_{|S_j|}) \sim p(\theta)$ ($j = 1, \ldots, N$) be outputs of SMOOTHED GREEDY. Then

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

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

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

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

The above differentiation is usually not computationally expensive. If $\Omega_k$ is the entropy function, once $\nabla_{\theta} \Omega_k(\theta)$ is obtained, we can compute $\nabla_{\theta} \mathbf{p}_k(\theta)$ in $O(n_k \times \dim(\Theta)$ time (see, Appendix C for details). The cost of computing $\nabla_{\theta} \Omega_k(\theta)$ is instance-dependent, but it is often as cheap as $O(n_k)$ 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 $\beta$ is some coefficient. If $\beta$ is a constant, the estimator remains unbiased since $\mathbb{E}_{S \sim p(\theta)}[\nabla_{\theta} \ln p(S, \theta)] = \nabla_{\theta} \mathbb{E}_{S \sim p(\theta)}[1] = 0$. By appropriately setting the $\beta$ value, we can reduce the variance. In practice, to set $\beta$ 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., $f(X, \theta)$ is not differentiable w.r.t. $\theta$, 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 $f(X, \theta)$.

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 $\delta$ 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 $\Omega_k$, and thus the $\delta$ value is controlled by $\epsilon > 0$ as $\delta = \epsilon \ln n_k$.

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

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

\[
\mathbb{E}_{S \sim p(\theta)}[1_S] = \sum_{S \in \mathcal{S}} 1_{S^p(S, \theta)}.
\]

By using our method shown in Section 3 with \( Q(S) = 1_S \), we can estimate the Jacobian matrix as

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

Here, given \( \theta \), the \((v, j)\) entry of the matrix indicates how and how much the infinitesimal increase in the \( j \)-th entry of \( \theta \) affects the probability that \( v \in V \) is chosen, which quantifies the sensitivity of each \( v \in V \) to uncertainties in \( \theta \). 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 \( \theta \) 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 \( \theta \), where \( \mathbf{w} \) represents model parameters. We train \( m(\cdot, \mathbf{w}) \) by optimizing \( \mathbf{w} \) values with training datasets \((\mathbf{X}_1, \theta_1), \ldots, (\mathbf{X}_M, \theta_M)\). Given test instance \((\mathbf{X}, \theta)\), where \( \theta \) is the unknown true parameter, the trained model predicts \( \hat{\theta} = m(\tilde{\mathbf{X}}, \mathbf{w}) \), and we obtain solution \( S \in \mathcal{I} \) (or, make a decision) by approximately maximizing \( f(\cdot, \theta) \). Our utility (decision quality) is measured by \( f(S, \theta) \). 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, \( \theta \) 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 \( f(S, \theta) \), where \( \theta \) 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, \( f(S, \theta) \). 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, 

\[
\frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{S \sim p(m(\mathbf{X}_i, \mathbf{w})))}[f(S, \theta_i)]
\]

where \( p(\cdot) \) is the output distribution. In each iteration, we sample a training dataset, \((\mathbf{X}_i, \theta_i)\), and compute \( \theta = m(\mathbf{X}_i, \mathbf{w}) \) with the current \( \mathbf{w} \) values. We then perform \( N \) trials of SMOOTHED GREEDY to estimate the current function value, \( \mathbb{E}_{S \sim p(\theta)}[f(S, \theta_i)] \). Next, with the outcomes of \( N \) trials, we estimate the gradient by using our method with \( Q(S) = f(S, \theta_i) \). More precisely, for each \( j \)-th trial of SMOOTHED GREEDY, we compute \( \nabla_\theta \ln p(S_j, \theta) \) as explained in Section 3 and estimate the gradient

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

where \( \theta = m(\mathbf{X}_i, \mathbf{w}) \). We then update \( \mathbf{w} \) with the above gradient estimator. When using mini-batch up-
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 $\nabla_\theta \mathbb{E}_{E \sim p(\theta)} [1_S]$ 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 $\theta$ 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, . . . , 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
We compare

We split the training data into random splits.

(\theta, \theta) \sim \mathcal{N}(0, \sigma^2 I)

SG-

SG-

VR-SG-

VR-SG-

Continuous

Two-stage

Random

Table 1: Function Values Achieved with Each Method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Training</th>
<th>Test</th>
<th>Training</th>
<th>Test</th>
<th>Training</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>K = 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SG-1</td>
<td>26.3 ± 4.0</td>
<td>26.4 ± 4.4</td>
<td>46.0 ± 5.9</td>
<td>45.9 ± 6.5</td>
<td>69.7 ± 23.8</td>
<td>69.6 ± 24.1</td>
</tr>
<tr>
<td>SG-10</td>
<td>29.0 ± 3.7</td>
<td>28.1 ± 4.9</td>
<td>47.0 ± 12.1</td>
<td>46.1 ± 12.4</td>
<td>71.5 ± 28.0</td>
<td>70.6 ± 28.1</td>
</tr>
<tr>
<td>SG-100</td>
<td>33.6 ± 2.4</td>
<td>32.0 ± 3.8</td>
<td>54.3 ± 2.0</td>
<td>53.5 ± 4.2</td>
<td>82.6 ± 21.8</td>
<td>82.3 ± 21.7</td>
</tr>
<tr>
<td>VR-SG-10</td>
<td>35.2 ± 6.1</td>
<td>33.7 ± 6.2</td>
<td>57.9 ± 1.6</td>
<td>56.2 ± 3.4</td>
<td>90.8 ± 16.5</td>
<td>89.5 ± 16.7</td>
</tr>
<tr>
<td>VR-SG-100</td>
<td>36.8 ± 0.9</td>
<td>35.6 ± 2.2</td>
<td>59.9 ± 1.6</td>
<td>68.0 ± 2.9</td>
<td>90.8 ± 1.1</td>
<td>94.5 ± 2.6</td>
</tr>
<tr>
<td>Continuous</td>
<td>24.0 ± 4.5</td>
<td>23.2 ± 4.9</td>
<td>43.2 ± 6.1</td>
<td>42.3 ± 7.1</td>
<td>81.7 ± 6.8</td>
<td>81.3 ± 6.6</td>
</tr>
<tr>
<td>Two-stage</td>
<td>17.3 ± 1.2</td>
<td>17.3 ± 2.1</td>
<td>35.6 ± 0.9</td>
<td>35.6 ± 2.7</td>
<td>65.5 ± 4.0</td>
<td>64.8 ± 5.1</td>
</tr>
<tr>
<td>Random</td>
<td>17.5 ± 1.0</td>
<td>17.6 ± 2.2</td>
<td>33.8 ± 0.8</td>
<td>34.0 ± 2.7</td>
<td>64.0 ± 1.3</td>
<td>64.5 ± 2.6</td>
</tr>
</tbody>
</table>

the 19-dimensional indicator vector as a movie feature. Each user \( \in \) 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 feature \(X\) 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 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 \((X_1, \theta_1), \ldots, (X_{80}, \theta_{80})\) and test the performance with \((X_1, \tilde{\theta}_1), \ldots, (X_{20}, \tilde{\theta}_{20})\). 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 \(w\) 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 \(\in \) by choosing elements corresponding to the top-\(K\) entries of solution \(x \in [0, 1]^n\) returned by SGA. Random is a baseline method that makes decisions \(\in \) 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 \((1 - 1/e)\)-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 \(w\) 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


