What to do when your discrete optimization is the size of a neural network?

by

Hugo Luis Andrade Silva

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Oftentimes, machine learning applications using neural networks involve solving discrete optimization problems, such as in pruning, parameter-isolation-based continual learning and training of binary networks. Still, these discrete problems are combinatorial in nature and are also not amenable to gradient-based optimization. Additionally, classical approaches used in discrete settings do not scale well to large neural networks, forcing scientists and empiricists to rely on alternative methods. Among these, two main distinct sources of top-down information can be used to lead the model to good solutions: (1) extrapolating gradient information from points outside of the solution set (2) comparing evaluations between members of a subset of the valid solutions. We take continuation path (CP) methods to represent using purely the former and Monte Carlo (MC) methods to represent the later, while also noting that some hybrid methods combine the two. The main goal of this work is therefore to compare both approaches. For that purpose, we first overview the two classes while also discussing some of their drawbacks analytically. Then, on the experimental section, we compare their performances, starting with smaller Microworld experiments, which allow more fine-grained control of problem variables, and gradually moving towards larger problems in the overparametrized regime, including neural network regression and neural network pruning for image classification, where we additionally compare against magnitude-based pruning. A future version of this work will also include experiments on sequential task learning, which are currently underway.

Acknowledgements

I would like to thank my supervisor Martha White for supporting me, being patient and giving me the opportunity to work with her since right after I joined the University of Alberta, when the world was on lockdown and I was working remotely from the other side of the world. Her being always in contact with research on many different topics and taking the time to share relevant papers with me has greatly helped me keep this thesis more up-to-date. I acknowledge also that access to Compute Canada significantly reduced the time taken to run the experiments. Finally, I would also like to thank my parents, siblings and my spouse for supporting me and being here with me throughout this entire journey. With their love, support and humour, my life has been meaningful and fun.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Problems in the mathematical sciences often involve the minimization of a function $J(\cdot)$ over some set $\mathcal{S}$. Initializing some tentative parameters, then changing them according to how $J(\cdot)$ behaves locally is one way to approach these problems. Assuming the current local direction in which $J(\cdot)$ decreases the most leads to reasonable next values of the tentative parameters, we can recalculate and follow it iteratively until local improvement no longer seems possible. If such a direction is determined using the gradient, methods following the above procedure belong to the field of gradient-based optimization (Boyd *et al.* 2004; Chong and Zak 2013).

Gradient-based optimization, however, is not applicable in many problems of interest to mathematicians. Firstly, the objective function has to be differentiable, otherwise it is not possible to calculate the gradients. Furthermore, following the direction of the gradient often involves changing the tentative parameters slightly in the desired direction. In general, for the next potential solutions to also belong to $\mathcal{S}$, all points in a small enough neighbourhood around the current parameter values have to also belong to $\mathcal{S}$. Tabular problems, where S consists of a finite set of values, for example, are not suitable for gradient-based optimization, since no two elements of S are arbitrarily close to each other.

Pseudo-Boolean (PB) functions are mappings from d dimensional binary vectors to $\mathbb{R}$ and correspond to tabular functions that map each of the 2^d

possible entries to a real number (Boros and Hammer 2002).[1] Notably, there is a similarity between these problems and the multi-armed bandit problem (Lattimore and Szepesvári 2020; R. S. Sutton and Barto 2018), where the learner has to test each of K different choices, here called arms, until finding the best one, receiving stochastic rewards in the process. The main difference is that PB optimization involves deterministic rewards. Exact optimization of a PB function is of combinatorial nature: in general, finding the best binary value involves trial-and-error evaluation of all elements in the set, which can be prohibitive in case d is also large (Korte *et al.* 2011).

Then again, such problems appear in many different fields, including game theory (Hammer and Holzman 1992), computer science (Karp 2010; Madhulatha 2012), VLSI design (Boros, Hammer, *et al.* 1999; Jünger *et al.* 1994), statistical mechanics (Phillips and Rosen 1994), finance (Hammer and Shlifer 1971), manufacturing (Kubiak 1995) and operations research (Picard and Ratliff 1978). For a more comprehensive list, refer to Tavares (2008). The maximum satisfiability problem (Karp 2010), for example, consists on finding the Boolean input that satisfies the maximum number of clauses from a given set of Boolean expressions and we can write it as a PB optimization problem. Similarly, the maximum independent set problem (Luby 1985), a well-known problem in graph theory that consists on finding the independent set of largest cardinality from an input graph, also fits this framework. Yet another example is training binary neural networks (Courbariaux *et al.* 2015; Hubara *et al.* 2016), where the weights are restricted to be either 1 or -1, usually to comply with computational constraints, such as model deployment in a resource-limited setup where it is only possible to represent each weight using a single bit.

In this work, we will study PB optimization problems that involve training Neural Networks. Particularly, we will focus on pruning and a future version of this work will also include results on sequential task learning. The first problem corresponds to finding the sparsest subnetwork possible without incurring

[1] They are called pseudo because they map to the reals, whereas a Boolean function has binary outputs. See O'Donnell (2014) for an overview of these.

significant loss of performance compared to the dense version (Blalock *et al.* 2020; Gale *et al.* 2019; Reed 1993). Similarly to the training of binary networks, this can be specially useful if deploying on a resource-constrained environment, where it is only possible to store a small percentage of the full network. Deep Learning frameworks or languages that make it possible to customize lower level computation, such as Julia, can take advantage of such a sparse structure to reduce inference times. Algorithmic and hardware support for these sparse structures is an active area of research (Hoefler *et al.* 2021, chapter 7). Since pruning corresponds to finding an optimal binary mask to apply to the neural network weights, it also falls under the framework of optimizing PB functions.

Sequential task learning, on the other hand, refers to the learning of multiple tasks in sequence using the same model (De Lange *et al.* 2021; Mai *et al.* 2022; Parisi *et al.* 2019). Performing well on this problem is imperative to the goal of developing a general artificial intelligence, for the deployed agent will have to constantly be learning and using the acquired knowledge to solve a continuing stream of incoming tasks. One of the main challenges that appear in this context is catastrophic forgetting, where the adaptation to the current task causes forgetting of the older ones (French 1999; McCloskey and Cohen 1989; R. Sutton 1986). Protecting weights important to the initial tasks is one of the approaches to retain performance and applying binary masks to these weights is one way of protecting them. This mechanism for protecting weights also has some neuroscience basis (Cichon and Gan 2015; Kuchibhotla *et al.* 2017; Otazu *et al.* 2009) and finding the correct binary mask can be cast under PB optimization.

The combinatorial nature of finding the optimal point on these large machine learning problems makes it necessary to forfeit exact optimization in favour of approximate solutions. Local search methods (Glover 1990; Hansen 1986; Johnson *et al.* 1988), for example, settle for points whose value of $J(\cdot)$ is the lowest in a local neighbourhood measured by the Hamming distance. Other methods involve reducing the original problem to an equivalent quadratic PB optimization form (Boros and Hammer 2002), a simpler version that is solvable

in polynomial time if it satisfies some additional properties (Kolmogorov and Rother 2007). Unfortunately, theoretical guarantees for local search methods only exist for some specific cases and, although quadratic PB optimization is solvable in polynomial time if the submodularity condition is satisfied, the general problem is NP-hard. More recently, to cope with this high complexity, studies have started applying quantum computing to speed up optimization of some PB problems (McGeoch and Wang 2013; Montanaro 2016), although these are still preliminary and of limited application. Even some polynomial time algorithms, however, may not be suitable for neural network contexts, where d can correspond to millions of parameters.

With this is mind, in this study we will focus mainly on two ways of approaching PB optimization: (1) construction of smooth continuation paths (Allgower and Georg 2003) and (2) Monte-Carlo gradient estimation (Mohamed *et al.* 2020). The first involves replacing the original problem with a surrogate differentiable alternative that can be annealed back to the original problem according to a parameter τ. The model then learns in a curricular fashion, starting from a version that is different from the desired one but gradually becomes closer to it, being possible to make it equivalent as $\tau \to 0$. The second corresponds to changing the PB optimization to the optimization over parameters of a factorized Bernoulli. The new problem becomes finding $\min_{\theta} \mathop{\mathbb{E}}_{z \sim p_\theta(\cdot)} [J(z)]$ and its minimum is the same as the original one (Boros and Hammer 2002).

This study will start by analyzing the behaviour of Monte Carlo (MC) and continuation path (CP) methods in Microworld scenarios, where it is possible to have greater control over problem variables, as well as calculate closed-form expectations, variances and the exact solutions to the PB problems. Afterwards, we move to larger neural network pruning tasks on common benchmark datasets. Here, closed-form solutions are no longer possible. On this last scenario, when training of neural network weights happens alongside the search for masks, determining masks based on the magnitude of the weights also becomes possible (Frankle and Carbin 2018; S. Han *et al.* 2015; Zhu and Gupta

2017). We will also investigate the so-called magnitude-based pruning (MP) approaches alongside the other two in this study.

The goals of this work can be summarized as follows:

1. To understand the trade-offs of CP and MC methods when used on PB optimization problems, as well as the hidden assumptions necessary for both approaches to work well.

2. To compare the performances of CP, MC and MP on larger scale problems with neural networks.

Investigation of both of these goals will be somewhat focused on the MC methods, which are less explored for pruning (and for sequential task learning as well). For the first goal, we show analytical results and then perform smaller experiments. Our main contributions are as follows:

- Showing failure cases for CP and MC.
- Deriving a bound for the performance of MC methods on PB optimization.
- Deriving and experimenting with a closed-form expression for the optimal MC control-variate.
- Introducing two new benchmarks: one with a tabular $J(\cdot)$ and mappings sampled from an exponential distribution and the other with a multilayer neural network $J(\cdot)$.
- Comparing different MC estimators and parametrizations.
- Introducing a medium scale regression benchmark where the parameters mask weights of a fixed neural network.

For the second goal, we focus on pruning, while noting that a future version will additionally contain multi-head sequential task learning experiments, which are currently underway.[2] Our main contributions are as follows:

[2]For sequential task learning, representative methods for CP and MP approaches are HAT (Serra *et al.* 2018) and Packnet (Mallya and Lazebnik 2018) respectively.

- Developing iterative versions of MC gradient estimators, which enable their use without the memory cost scaling with the number of samples.
- Extending already proposed MC algorithms, allowing them to be used with different estimators and parametrizations. We compare the resulting methods with the CP method from Savarese *et al.* (2020) and with the MP methods from S. Han *et al.* (2015) and Zhu and Gupta (2017).

Chapter 2

Pseudo-Boolean optimization

In this chapter, we first start by presenting some basics of pseudo-Boolean optimization and introducing some notation. Then, we present some problems appearing in different fields and show how they are instances of PB. Finally, we briefly go over some common classes of algorithms used to find (approximate) solutions.

2.1 Basics

We will often use boldface to denote vectors, unless otherwise noted. A pseudo-Boolean function $J(\mathbf{z})$ is a mapping $\{0,1\}^d \to \mathbb{R}$. One simple way to identify such a function is to note that it is a table containing 2^d entries, each entry corresponding to one of the possible inputs. Additionally, there is a one-to-one correspondence between each such input and the binary representation of the first 2^d natural numbers. To denote this relation, we will use $\boldsymbol{\zeta}_h$ to represent the vector in $\{0,1\}^d$ satisfying:

$$\sum_{i=1}^{d} 2^{i-1}(\boldsymbol{\zeta}_h)_i = h \qquad \text{for } h \in \{0, ..., 2^d - 1\}.$$

Where the notation $(\boldsymbol{\zeta}_h)_i$ is used to index the i-th element of the vector $\boldsymbol{\zeta}_h$. Sometimes, to avoid clutter, we will denote the i-th element of a vector $\mathbf{z}$ as simply z_i. To exemplify, considering $d = 2$ we have:

$$\boldsymbol{\zeta}_0 = \begin{bmatrix}0\\0\end{bmatrix}; \boldsymbol{\zeta}_1 = \begin{bmatrix}1\\0\end{bmatrix}; \boldsymbol{\zeta}_2 = \begin{bmatrix}0\\1\end{bmatrix}; \boldsymbol{\zeta}_3 = \begin{bmatrix}1\\1\end{bmatrix}$$

and $J(\mathbf{z})$ can be represented by the table listing all possible $J(\boldsymbol{\zeta}_h)$. Notably, any alternative function $J' : [0,1]^d \to \mathbb{R}$ satisfying:

$$J(\boldsymbol{\zeta}_h) = J'(\boldsymbol{\zeta}_h), \qquad \text{for } h \in \{0, ..., 2^d - 1\} \tag{2.1}$$

can also be used to generate the same mapping, regardless of how $J'(\cdot)$ behaves outside of $\{0,1\}^d$. This observation will be used to motivate some methods in subsequent parts of this work. We can additionally represent $J(\mathbf{z})$ with the following equivalent form:

$$\mathscr{P}_J(\mathbf{z}) = \sum_{h=0}^{2^d-1} \left(\prod_{i=1}^{d} z_i^{(\boldsymbol{\zeta}_h)_i} \overline{z}_i^{(\overline{\boldsymbol{\zeta}}_h)_i} \right) J(\boldsymbol{\zeta}_h). \tag{2.2}$$

Where we denote $\overline{\mathbf{z}} = \mathbf{1} - \mathbf{z}$, for $\mathbf{1} = [1, \cdots, 1]^\top$, and we adopt the convention $0^0 = 1$. For reference, $\overline{z}$ and z are often called literals. Another notation that we will sometimes use is the following, considering $\mathcal{D} = \{1, 2, ..., d\}$:

$$\mathscr{P}_J(\mathbf{z}) = \sum_{\mathcal{S} \subseteq \mathcal{D}} \left(\prod_{i=1}^{d} z_i^{(\mathbf{1}^{\mathcal{S}})_i} \overline{z}_i^{(\overline{\mathbf{1}^{\mathcal{S}}})_i} \right) J(\mathbf{1}^{\mathcal{S}}),$$

where:

$$(\mathbf{1}^{\mathcal{S}})_i = \begin{cases} 1 & \text{if } i \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases}. \tag{2.3}$$

We emphasize that $\mathscr{P}_J(\mathbf{z}) = J(\mathbf{z})$ for all $\boldsymbol{\zeta}_h$, since the only summand that is $\neq 0$ for $\mathbf{z} = \boldsymbol{\zeta}_k$ is the iterate where $h = k$ and the inner product inside the parenthesis will evaluate to 1 for this summand.

Note also that, for two different $J'(\cdot)$ and $J''(\cdot)$ satisfying Equation (2.1), $\mathscr{P}_{J'}(\mathbf{u}) = \mathscr{P}_{J''}(\mathbf{u})$ for $\mathbf{u} \in [0,1]^d$. This will be important later in this work, when we explore the behavior of $\mathscr{P}_J(\cdot)$ outside of $\{0,1\}^d$.

The form in Equation (2.2) is a multilinear polynomial, since it corresponds to a polynomial whose terms never appear exponentiated by more than a factor of 1. The polynomial can be alternatively denoted as:

$$\mathscr{P}_J(\mathbf{z}) = \sum_{\mathcal{S} \subseteq \mathcal{D}} w_{\mathcal{S}} \prod_{i \in \mathcal{S}} z_i. \tag{2.4}$$

Where $w_{\mathcal{S}}$ are weights assigned to subsets of $\mathcal{D}$ and the product is one for the empty set. The following result makes it simpler to notice that the two forms are equivalent:

Theorem 1. *An arbitrary twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ will have a Hessian whose diagonal is equal to $\mathbf{0} = [0, \cdots, 0]^\top$ if and only if it can be represented in the form of Equation* (2.4).

Proof. *Section A.1 in the appendix.*

Inspecting Equation (2.2), we can see that its Hessian has diagonal equal to zero, so it can indeed be expressed in the form of Equation (2.4). Additionally, for an arbitrary pseudo-Boolean function $J(\cdot)$, its multilinear polynomial form is unique (Boros and Hammer 2002, section 4.1). We present an example of these forms below:

Example 1. *Considering the following pseudo-Boolean function for $d = 2$:*

$$J(\boldsymbol{\zeta}_0) = A; J(\boldsymbol{\zeta}_1) = B; J(\boldsymbol{\zeta}_2) = C; J(\boldsymbol{\zeta}_3) = D.$$

Expressing $\mathscr{P}_J(\boldsymbol{z})$ as in Equation (2.2) *corresponds to:*

$$\begin{aligned}\mathscr{P}_J(\boldsymbol{z}) &= (1 - z_1)(1 - z_2)A + z_1(1 - z_2)B + (1 - z_1)z_2C + z_1z_2D \\ &= \overline{z}_1\overline{z}_2A + z_1\overline{z}_2B + \overline{z}_1z_2C + z_1z_2D,\end{aligned}$$

whereas Equation (2.4) *corresponds to:*

$$\mathscr{P}_J(\boldsymbol{z}) = A + (B - A)z_1 + (C - A)z_2 + (A - B - C + D)z_1z_2.$$

The degree of the PB function is the size of the largest $\mathcal{S} \subseteq \mathcal{D}$ for which $w_{\mathcal{S}} \neq 0$ in Equation (2.4). Finally, in PB optimization, some local search methods use the following alternative definition of derivative:

$$\begin{aligned}\Delta_i(\mathbf{z}) = J(z_1, \cdots, &z_{i-1}, 1, z_{i+1}, \cdots, z_d) \\ &- J(z_1, \cdots, z_{i-1}, 0, z_{i+1}, \cdots, z_d).\end{aligned} \tag{2.5}$$

On this study, whenever we mention the term derivative, it will always refer to the conventional derivative, not the one in Equation (2.5), unless specifically stated. Optimization of PB functions therefore involves finding:

$$\mathbf{z}^* = \underset{\mathbf{z} \in \{0,1\}^d}{\arg\min}\ J(\mathbf{z}). \tag{2.6}$$

Note that rewriting the optimization over a categorical variable under the PB framework is also possible. Particularly, if the number of categories is $K = 2^d$, and $\text{Cat} : \{0,1\}^d \to \{1,2,\cdots,K\}$ is some invertible mapping, we have:

$$\begin{aligned}\underset{C\in\{1,2,\cdots,K\}}{\arg\min}\ g(C) &= \underset{h\in\{0,\ldots,2^d-1\}}{\arg\min}\ g(\text{Cat}(\boldsymbol{\zeta}_h)) \\ &= \underset{\mathbf{z}\in\{0,1\}^d}{\arg\min}\ g(\text{Cat}(\mathbf{z})) \\ &= \underset{\mathbf{z}\in\{0,1\}^d}{\arg\min}\ J(\mathbf{z}).\end{aligned}$$

Even if $K \neq 2^d$, it is straightforward to extend the above. For $K < 2^d$, for example, we can make $\text{Cat}(\cdot)$ a surjective (but not injective) mapping, although this extension would lead to a less natural interpretation.

2.2 Examples

Many combinatorial optimization problems can be easily formulated under the PB optimization framework, we present some examples below.

Example 2 (Maximum Satisfiability, adapted from section 3 of Boros and Hammer (2002))**.** *This is a very frequently studied problem in applied mathematics/theoretical computer science. The input is a family $\mathscr{C}$ of clauses C, where each of them is a set $C \subseteq \{z_1, \overline{z}_1, \cdots, z_d, \overline{z}_d\}$. A clause is satisfied when any of its literals evaluate to one. In logic notation, considering $u_i \in C$ as the literals in an arbitrary clause, the clause is satisfied when:*

$$u_1 \vee u_2 \vee \cdots \vee u_{|C|} = 1.$$

Mathematically, this happens when:

$$\prod_{u\in C} \overline{u} = 0,$$

the problem corresponds to the following minimization:

$$\underset{\boldsymbol{z}\in\{0,1\}^d}{\arg\min} \sum_{C\in\mathscr{C}} \prod_{u\in C} \overline{u}.$$

Example 3 (Maximum Independent Set, adapted from section 3 of Boros and Hammer (2002))**.** *Considering a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$, where $\mathcal{V} = \{1, \cdots, d\}$, an independent set is one in which no two vertices belong to the same edge $e \in \mathcal{E}$. The problem becomes finding:*

$$\arg\max_{\mathcal{U} \subseteq \mathcal{V}} |\mathcal{U}| \qquad s.t.\ N(v) \cap \mathcal{U} = \emptyset,\ \forall v \in \mathcal{U}.$$

Where $N(v)$ represents the neighbours of vertex v. Tavares (2008, theorem 2.1) shows that the maximum of this problem is equal to:

$$\max_{\boldsymbol{z} \in \{0,1\}^d} \sum_{i \in \mathcal{V}} z_i - \sum_{(i,j) \in \mathcal{E}} z_i z_j.$$

Although this equation corresponds to the cardinality of the maximum independent set, an arbitrary maximizing $\boldsymbol{z} = \boldsymbol{z}^*$ is not guaranteed to be independent. Nonetheless, its conversion to a maximizing independent set will take at most $(|V| - \sum z_i^*)$ iterations. Note that we present this as a maximization problem for consistency with the literature, but since:

$$\max_x J(x) = -\min_x -J(x),$$

every maximization problem has a minimization equivalent.

Example 4 (Computer Vision)**.** *These problems often involve assigning discrete labels for every pixel in an image. For example, in image segmentation, one may want to discern foreground from background, or to identify pixels corresponding to human skin in an image. In tracking, the goal is to follow some object on a sequence of frames, which also involves labeling pixels as either pertaining to the object or not (Szeliski 2022).*

Assuming we have two labels, such as in binary image segmentation, the loss can often be formulated using energy minimization (LeCun, Chopra, et al. *2006; Szeliski* et al. *2008):*

$$E(\boldsymbol{z}) = E_d(\boldsymbol{z}) + \lambda E_s(\boldsymbol{z}).$$

Where E_d is the data energy, which penalizes deviation from the data, whereas $E_s(\boldsymbol{z})$ is the smoothness energy, which enforces spacial coherence, and λ is just a tradeoff parameter. Interestingly, $E_d(\boldsymbol{z})$ corresponds to the negative log-likelihood of a Markov Random Field whereas $E_s(\boldsymbol{z})$ corresponds to that of the prior (S. Z. Li 2012). This probabilistic interpretation is reminiscent of the ones done for classification and regression in Machine Learning (White 2022, chapter 3).

The problem then becomes that of finding:

$$\arg\min_{\boldsymbol{z}\in\{0,1\}^d} E_d(\boldsymbol{z}) + \lambda E_s(\boldsymbol{z})$$

Where d can correspond, for example, to the width times the height of an image, or to their sum in case of multiple images, such as a video or a dataset.

Example 5 (Training of Binary Neural Networks)**.** *Considering problems where the model is parametrized by a neural network, minimizing the loss corresponds to:*[1]

$$\textit{minimize } J(\boldsymbol{w}) = \mathop{\mathbb{E}}_{X,Y\sim\mathcal{D}} \left[\ell(f(X;\boldsymbol{w}), Y)\right], \qquad \textit{for } \boldsymbol{w}\in\mathcal{W}.$$

Where X and Y are the data and labels respectively, $\mathcal{D}$ is the data distribution, $\boldsymbol{w}$ represents the weights of the neural network, which belong to the set $\mathcal{W}$. The neural network evaluation is denoted by $f(\cdot)$ and $\ell(\cdot)$ is the loss, which can be, for instance, a regression or a classification loss.

Sometimes it is desirable to constrain Neural Network weights to only have a discrete set of possible values, which can be due to memory constraints. Work done in Courbariaux et al. *(2015) and Hubara* et al. *(2016), for example, allows weights to be either* 1 *or* -1*. The problem becomes finding:*

$$\arg\min_{\boldsymbol{z}\in\{0,1\}^d} \mathop{\mathbb{E}}_{X,Y\sim\mathcal{D}} \left[\ell(f(X;2\boldsymbol{z}-\mathbf{1}), Y)\right].$$

[1] We emphasize that some problems have an infimum, but not a minimum. Still, a solution close to the lower bound is often desired even in these cases. When we say the goal is to minimize the problems we refer to both situations.

2.3 Algorithmic approaches

In this section, we make a brief and non-comprehensive overview of some algorithmic approaches commonly used on PB optimization. By reason of the exponential number of solutions, it may be necessary to waive the search for the global optimum and instead stop it when a local solution is reached. Local search methods are a broad class of iterative approaches that look for the best solution in a neighbourhood $N(\mathbf{z})$ around the current solution $\mathbf{z}$ (see, for example, Schuurmans and Southey (2001) or sections 4.3 and 4.6 of Boros and Hammer (2002)). Considering, for instance, a neighbourhood using the Hamming distance, which here corresponds to the number of coordinates where $\mathbf{1}^{\mathcal{S}}$ is different from $\mathbf{1}^{\mathcal{S}'}$, for $\mathcal{S}$ and $\mathcal{S}'$ both $\in \{1..d\}$, the method may return a local solution $\hat{\boldsymbol{z}}^*$ satisfying:

$$J(\hat{\boldsymbol{z}}^*) \leq J(\mathbf{z}), \qquad \text{for } \mathbf{z} \in N(\hat{\mathbf{z}}^*).$$

The high level approach the deterministic version of these methods take is summarized in Algorithm 1. Nevertheless, a greedy search around the current solution is likely to get stuck on a poor local optimal, justifying the emergence of alternative approaches that allow some degree of exploration. As an example, tabu search methods (Glover 1989; Glover 1990) accept changing the solution for another whose value of $J(\cdot)$ is higher, in case it is no longer possible to get improvement in the current neighbourhood. Moreover, they discourage re-visiting previous solutions by means of a technique they call *prohibitions*.

Algorithm 1 General procedure for approximation algorithms

Input: PB function $J(\cdot)$, initial solution $\mathbf{z}_0$, neighbourhood function $N(\cdot)$;
$t \leftarrow 0$
while $\mathbf{u} < \mathbf{z}_t$, for $\mathbf{u} \in N(\mathbf{z}_t)$ **do**
  $\mathbf{z}_{t+1} \leftarrow \mathbf{u}$
  $t \leftarrow t + 1$
Return: $\mathbf{z}_t$

Similarly, simulated annealing (Salamon *et al.* 2002; Van Laarhoven *et al.* 1987) allows some degree of exploration, controlled by a temperature parameter. When the temperature is high, the model will have a higher probability

of choosing a next solution worse than the current one. As the temperature goes to zero, the probability of the worse choice becomes arbitrarily close to zero. The method then follows a temperature schedule based on the computational budget. The name annealing comes from an technique from metallurgy, where the material goes through heating and then controlled cooling to alter its physical properties.

Yet another popular approach surfaces by interpreting some problems as graph optimization. In computer vision, for example, Boykov *et al.* (2001) and Szeliski *et al.* (2008) re-cast problems involving pixel labelling as finding the minimum cut of a graph. The *max-flow min-cut theorem* (Dantzig and Fulkerson 2003) draws a connection between this problem and the maximum value of the flow in a flow network, allowing the reuse of efficient algorithms from graph theory to speed-up the search for approximate solutions (Dinitz 1970; Edmonds and Karp 1972).

Alternatively, genetic algorithms take inspiration from natural selection to find the approximate solution while avoiding getting stuck on poor local optima (Kramer and Kramer 2017). They start with a population of randomly initialized solutions (called individuals) which correspond to an iterate (called generation) of the algorithm. The method assigns each candidate solution a fitness score, which, in our case, is based on the function $J(\cdot)$. Then, selection of a subset of the individuals happens according to this fitness score[2] and is succeeded by a merging phase. Merging follows a problem-specific formulation and outputs the candidates of the next generation. In our case, to illustrate, one possibility is to select a cutoff index and swap the values before and after the index between the two parents. Moreover, to further increase stochasticity, randomly changing some part of the children via mutation is common practice. The algorithm ends when a desired fitness level is achieved, the fitness plateaus or the computational budget is used up. Notably, this method can help avoid poor local minima due to the multiple distinct initializations.

[2]Sometimes it may be wise to also include less fit candidates to make the population more diverse.

Meanwhile, branch-and-bound algorithms (Lawler and Wood 1966) are also prominent. The essence of such methods is to break down the search, recursively splitting the original search space into smaller subspaces. For each subspace $\mathcal{S}$, the algorithm will compute a lower bound on $J(\mathbf{z})$ for $\mathbf{z} \in \mathcal{S}$ and will use its value to discard subspaces where it is certain that the desired solution will not be found. The problem instance (and the sub-problem instances generated thereafter) must define three operations: (1) *branching*, which breaks the instance into sub-instances (2) *bounding*, which computes the lower bound on the function for the current instance (3) *returning candidate*, which returns a candidate solution from the set of values in the current instance.

Finally, quadratization methods reduce the degree of the original multilinear polynomial from Equation (2.2), rewriting it as:

$$\mathscr{P}'(\mathbf{z}') = a_\emptyset + \sum_{i \in \mathcal{D}'} b_i z'_i + \sum_{\substack{i,j \in \mathcal{D}' \\ i<j}} c_{ij} z'_i z'_j.$$

Where $d \leq d'$, $\mathcal{D} \subseteq \mathcal{D}' = \{1, \cdots, d'\}$ and $\mathbf{z}' \in \{0,1\}^{d'}$. One can transform the polynomial iteratively by increasing the size of the set by one each time. Adding a new element can, for example, be performed by selecting a pair from the previous iteration set and having the new member correspond to their product. Weights from the old $\mathscr{P}(\cdot)$ can then be changed in a way that reduces the degree of some of the summands by one while also enforcing that the new element is indeed the product of the chosen pair. For more details of the complete example, refer to Boros and Hammer (2002, section 4.4). Rosenberg (1975) shows that this procedure can be completed in polynomial time. Furthermore, if the following submodularity condition is satisfied:

$$\mathscr{P}'(\mathbf{1}^A) + \mathscr{P}'(\mathbf{1}^B) \geq \mathscr{P}'(\mathbf{1}^{A \cup B}) + \mathscr{P}'(\mathbf{1}^{A \cap B}),$$

finding the solution $\mathscr{P}(\mathbf{z}^*)$ in polynomial time is possible (Grötschel *et al.* 1981). In general, however, this condition is not satisfied and the problem is NP-hard. Nonetheless, development of algorithms tailored to the quadratic case is an active area of research. Tavares (2008) overviews some quadratic PB optimization methods.

To conclude, sometimes deriving sub-optimality bounds on an approximation of the true solution and finding this approximation in polynomial time is feasible. Algorithms that search for solutions this way are called approximation algorithms. In general, not all PB optimization problems admit polynomial-time algorithms with reasonable approximation guarantees as discussed in Williamson and Shmoys (2011, chapter 16).

Nevertheless, in our case d is high and a single pass through all of the dimensions might be prohibitive, causing even the best-case polynomial algorithms to be unsuitable. Namely, d will be the number of weights on a neural network, sometimes reaching the order of millions. High dimensions therefore warrant alternative approaches, some of which are explored in this study.

Chapter 3

Numerical Continuation

This chapter introduces concepts of numerical continuation, which serve as the basis for the continuation path methods (CP) that we will study in this work. The first section introduces basics of numerical continuation, the second one explains how to use numerical continuation to approximately solve pseudo-Boolean optimization problems. We note that there are alternative ways to approach PB optimization with CP methods, but we will restrict our discussion to its recent use on pruning and sequential task learning. Finally, the third section presents very simple failure cases of CP applied to PB optimization and we reason about when these methods can fail.

3.1 Basics

Continuation methods, sometimes called embedding or homotopy methods, are useful when solving a system of equations of the form

$$F(\mathbf{x}) = \mathbf{0} \tag{3.1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is smooth. There exist iterative methods for finding roots of $F(x)$, such as Newton-Raphson, where

$$\mathbf{x}_{t+1} = \mathbf{x}_t - (\nabla F(\mathbf{x}_t))^{-1} F(\mathbf{x}_t)$$

with $\nabla F(\mathbf{x}_t)$ being the $n \times n$ Jacobian matrix evaluated at $\mathbf{x}_t$. These methods often depend on a good initial value $\mathbf{x}_0$ and are unlikely to work if such an initialization is not available. Continuation methods start with a simpler problem, where finding (reasonable approximations of) the solution is relatively easy, and then transition iteratively to the original problem. The solutions found for the k-th problem should serve as reasonable initial values when applying methods akin to Newton-Raphson to the $k+1$-th problem, simplifying the search. Namely, we construct a smooth homotopy or continuation path $H : \mathbb{R}^{n+1} \to R^n$, where

$$H(\mathbf{x}, 0) = F(\mathbf{x}), \quad H(\mathbf{x}, 1) = G(\mathbf{x})$$

and $G(\mathbf{x})$ is some simpler function where finding roots is easier. One possible homotopy, for example, is to simply use the interpolation between $G(\mathbf{x})$ and $F(\mathbf{x})$. Based on that idea, Algorithm 2, often called embedding algorithm, describes the most direct way of approaching the problem.

Algorithm 2 General procedure for embedding algorithms

Input: initial value $\mathbf{x}_{init}$, homotopy $H(\cdot)$;
Define τ schedule $1 = \tau_0 > \tau_1 > \cdots > \tau_K = 0$
for $k = 0 \cdots K$ **do**
  Find $\hat{\mathbf{x}}_k \approx \arg\min_{\mathbf{x} \in \mathbb{R}^n} H(\mathbf{x}, \tau_k)$, starting with $\mathbf{x}_{init}$
  ▷ (e.g. using Newton-Raphson)
  $\mathbf{x}_{init} \leftarrow \hat{\mathbf{x}}_k$
Return: $\hat{\mathbf{x}}_K$

Denoting by $H^{-1}(\mathbf{0})$ the set of solutions of $H(\cdot)$ such that

$$H^{-1}(\mathbf{0}) = \{(\mathbf{x}, \tau) \mid H(\mathbf{x}, \tau) = \mathbf{0}\},$$

implicitly, we attempt to trace a curve with elements $\in H^{-1}(\mathbf{0})$ that starts with an easy to find solution $(\mathbf{x}_0^*, 1)$ and ends with $(\mathbf{x}^*, 0)$, where $\mathbf{x}^*$ is a solution to Equation (3.1). We denote this curve by $c(s)$, with $c : \mathbb{R} \to \mathbb{R}^{n+1}$. Note that this curve does not have to be parametrized with $s = \tau$. *Predictor-corrector*, for instance, parametrize s using the arc length instead.

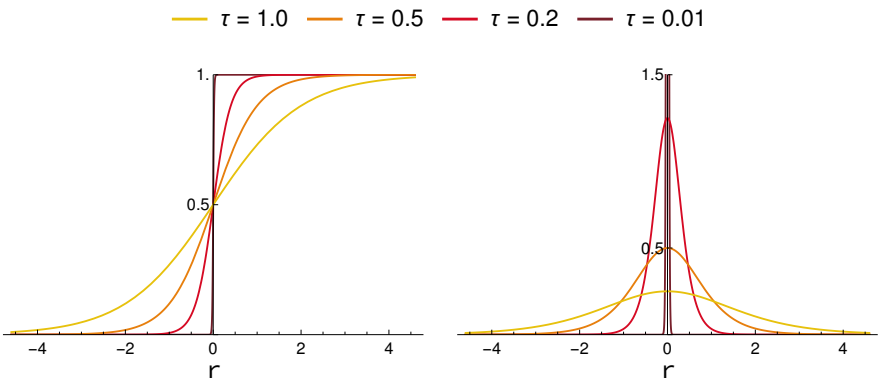
The existence of such a smooth curve $c(s)$ starting from $c(0) = (\mathbf{x}_0^*, 1)$ is guaranteed if the Jacobian $\nabla H(\cdot)$ is full-rank at $c(0)$. Furthermore, this curve will have non-zero derivative at $s = 0$. If the Jacobian is also full-rank for the other points from $H^{-1}(\mathbf{0})$, the curve will have similar structural properties to a circle. Additionally, for the curve to also contain $(\mathbf{x}^*, 0)$, as opposed to going to infinity or returning to $(\mathbf{x}_0^*, 1)$, the problem must satisfy some boundary conditions. For more details, see Allgower and Georg (2012, chapter 11).

Alternatively to Algorithm 2, some methods approach the problem by varying the s parameter directly. Particularly, one can write $H(c(s)) = \mathbf{0}$ and differentiate both sides with respect to s, combining the resulting equation with the requirement that $H(\mathbf{x}_0^*, 1) = \mathbf{0}$ to arrive at an initial value problem. Approximately solving the resulting differential equation can then yield a solution close to $c(s)$. One of the main branches of numerical continuation methods, called *predictor-corrector* methods, alternate between coarse numerical integration of the differential equation and using a local stabilizing step to eventually arrive at a solution close to $(\mathbf{x}^*, 0)$. Another main branch of homotopy methods, the *piecewise linear* methods, instead consider piecewise linear approximation of the homotopy map. These are more general than *predictor-corrector*, but they tend to perform worse in cases where both are applicable. Our discussion, nevertheless, will be centered in approaches following Algorithm 2.

For a more detailed presentation of numerical continuation methods, see Allgower and Georg (2012). Horst and Pardalos (2013, chapter Homotopy Methods, section 2) additionally list many different applications of CP methods for engineering, economics and overall mathematical problems.

3.2 Application to PB optimization

There are multiple ways to apply CP methods for PB optimization. One could, for example, take Equation (2.2) and use τ as a multiplicative factor zeroing out the higher order terms on the easier problems, then recovering them as $\tau \to 0$. Alternatively, one could attempt to approach the problem as constrained optimization, use Lagrange multipliers and then have τ control the strength



(a) $\sigma\left(\frac{x}{\tau}\right)$ (b) $\frac{\mathrm{d}}{\mathrm{d}x}\sigma\left(\frac{x}{\tau}\right)$

Figure 3.1: Effect of varying the temperature of a sigmoid.

of the constraints. We will, however, focus on parametrizing $\mathbf{z}$ as a sigmoid. This idea surfaces by firstly noting that

$$\min_{\mathbf{z}\in\{0,1\}^d} J(\mathbf{z}) = \min_{\mathbf{x}\in\mathbb{R}^d} J(\mathbb{1}[\mathbf{x} \geq 0])$$

where $\mathbb{1}[\cdot]$ is the indicator function, which is applied element-wise and can represent any $\mathbf{z}$ as long as we choose $\mathbf{x}$ in the appropriate quadrant. Secondly, we note that

$$\lim_{\tau\to 0} \sigma\left(\frac{x}{\tau}\right) = \begin{cases} \mathbb{1}[x > 0] & x \neq 0 \\ 0.5 & x = 0 \end{cases}$$

where

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}$$

is the logistic sigmoid function, which here we consider to be applied element-wise when the input is a vector. We will ignore the case $x = 0$, as it rarely occurs in practice when the algorithms get to $\tau \approx 0$. Combining the two observations, we can then take some arbitrary τ and write the problem as

$$\text{minimize} \lim_{\tau\to 0} J\left(\sigma\left(\frac{\mathbf{x}}{\tau}\right)\right), \qquad \text{for } \mathbf{x} \in \mathbb{R}^d. \tag{3.2}$$

Figure 3.3 shows how the plot of $\sigma(\cdot/\tau)$ changes as we vary the temperature τ, as well as what happens to its derivative. Notably, the derivative becomes large near $x = 0$ and almost zero everywhere else, causing gradient-based optimization to become infeasible for τ close to zero.

Nonetheless, we can take an approach similar to the one from Section 3.1. Starting from $\tau = 1$, which corresponds to a problem amenable to gradient-based optimization and iteratively changing τ to be closer to zero, we make the problem gradually closer to the desired one. Conveniently, the procedure also yields good starting solutions for the problems corresponding to the intermediate values of τ. To make the connection to Section 3.1 more explicit, for an arbitrary τ, our goal is to find $\mathbf{x}$ close to stationary, which satisfies

$$\nabla_{\mathbf{x}} J\left(\sigma\left(\frac{\mathbf{x}}{\tau}\right)\right) \approx \mathbf{0}. \tag{3.3}$$

We reiterate that the gradient might not ever be exactly zero. For example, if $d = 1$ and $J(\cdot)$ is the identity mapping, the gradient can get arbitrarily close, but it will never be zero. Despite that, we can take:

$$H(\mathbf{x}, \tau) = \nabla_{\mathbf{x}} J\left(\sigma\left(\frac{\mathbf{x}}{\tau}\right)\right)$$

and then follow Algorithm 2. Instead of using Newton-Raphson, here it is more straightforward to simply use stochastic gradient descent (SGD).

Finally, the reader may have noted that we cannot use $\tau = 0$, as that involves division by zero. Be that as it may, a small enough τ will have the same effect, for limited computer precision will cause $\sigma(x/\tau) \in \{0, 1\}$ if $x \neq 0$. Interestingly, this method is reminiscent of curriculum learning, where the learner starts with an easier problem that also becomes harder over time. Luo and Wu (2020), Savarese *et al.* (2020), and Yuan *et al.* (2020) have successfully used this homotopy for pruning, whereas Azarian *et al.* (2020) also suggest it, but adopt a simplified version. Serra *et al.* (2018) have used it for sequential task learning.

3.3 Drawbacks

One concerning aspect of PB methods from Section 3.2 is the extrapolation of information from evaluations inside the hypercube (i.e. $[0, 1]^d$) to the differences between evaluations at the vertices (i.e. $\{0, 1\}^d$). To clarify, $\mathbf{x}$ is initialized to some finite value, usually $\mathbf{0}$, so that $\sigma(x_i/\tau) = 0.5$, and SGD uses gradients evaluated there to arrive at a new $\mathbf{x}$. The goal, however, is to find one of

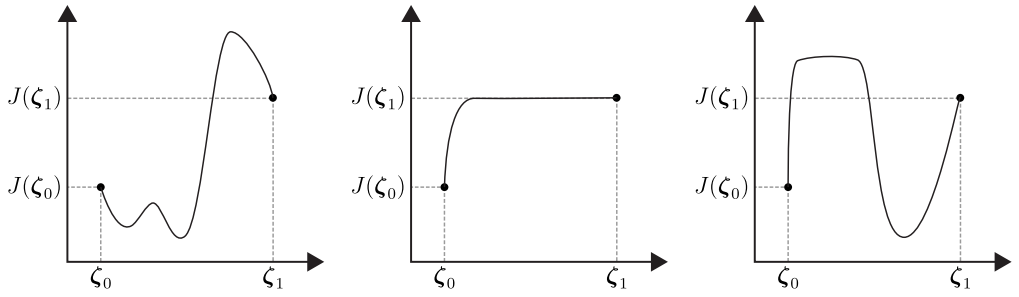
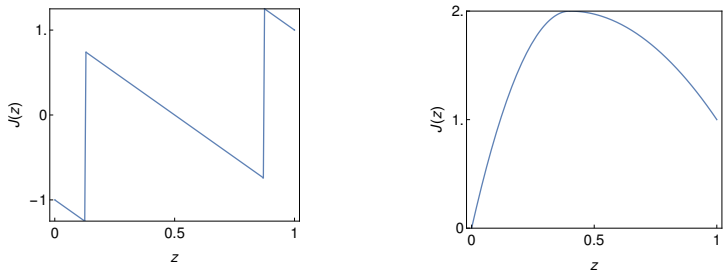


Figure 3.2: Completely different choices of $J(\cdot)$ lead to the same PB optimization problem.



(a) Example 6

(b) Example 7

Figure 3.3: $J(\cdot)$ for the counter examples.

the vertices where $J(\cdot)$ is lower than in other elements from $\{0,1\}^d$. Since the gradient only indicates changes in a small neighbourhood around the current value, can one reliably use it to draw conclusions about $J(\cdot)$ in regions far from the current $\mathbf{x}$?

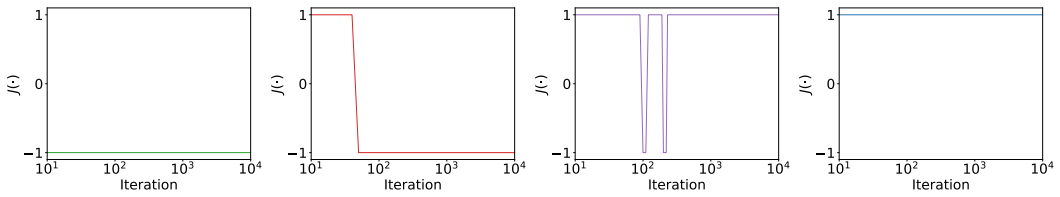
Furthermore, as we mentioned in Section 2.1, any function whose evaluations on the corners are the same as from the multi-linear polynomial will correspond to the same PB optimization problem (Equation (2.1) and the subsequent discussion). Figure 3.2 illustrates one such example, for $d = 1$. The $J(\cdot)$ at hand will often not be the multilinear polynomial $\mathscr{P}(\cdot)$ from Equation (2.2). Since these derivatives can be completely different depending on the choice of $J(\cdot)$ is their information really relevant? For example, the derivative at $x = 0$ can make it seem that the right choice is to go towards $z = 1$, but the (negative) gradient might as well have pointed to $z = 0$ had a different $J(\cdot)$ been used. With all of these in mind, we present some failure cases for CP methods implementing the ideas from Section 3.2.

Example 6. *Consider the following function, whose plot is shown in Figure 3.3a*

$$J(z) = \begin{cases} -1 - 2z & z < 0.125 \\ -51 + 398z & 0.125 \leq z < 0.13 \\ 1 - 2z & 0.13 \leq z < 0.87 \\ -347 + 398z & 0.87 \leq z < 0.875 \\ 3 - 2z & 0.875 \leq z \end{cases}.$$

In this case, we have $J(0) < J(1)$*. However, for most of the* $(0, 1)$ *interval, the negative gradient points towards* $z = 1$*. Taking conclusions about the behaviour of* $J(\cdot)$ *in* $\{0, 1\}$ *using these derivatives is therefore incorrect.*

Luckily, however, for $z \in [0.125, 0.13) \cup [0.87, 0.875)$*, the negative gradient has a large magnitude and points towards* $z = 0$*. Therefore, depending on the learning rate, the model might be able to reach these regions, which will in turn help lead it to the correct solution. Figure 3.4 depicts running the CP algorithm for this loss starting with* $x = 0$ *and training for* 100 *iterations before updating* τ*. On these examples, we use* $J(\sigma(x/\tau))$ *during training, but the curves only show* $J(\mathbb{1}[x > 0])$ *at each timestep, since only* 0 *and* 1 *are valid solutions to the problem. Only the higher learning rates got to the correct solution.*



(a) $\alpha = 0.5$ (b) $\alpha = 0.1$ (c) $\alpha = 0.05$ (d) $\alpha = 0.01$

Figure 3.4: Performance of different learning rates (α) for the problem in Example 6.

Example 7. *Consider the following function, depicted in Figure 3.3b:*

$$J(z) = \begin{cases} 2 - 12.5(z - 0.4)^2 & z < 0.4 \\ 2 - \frac{1}{0.36}(z - 0.4)^2 & 0.4 \leq z \end{cases}.$$

In this case, whenever $z = \sigma(x/\tau)$ is greater than 0.4, the derivative is negative and gradient descent will go towards $z = 1$, even though the correct solution is $z = 0$. This will happen regardless of the learning rate being used. For the CP algorithm to get to the correct solution, it has to be initialized with $x_{init} < \text{Logit}(0.4)$. Since in general we cannot assume prior knowledge about the problem, it is customary to use $x_{init} = 0$, making it impossible to solve the problem with this CP method. For problems like these, initializing the method in the right region amounts to knowing what the solution is before running the method, which would defeat the purpose of using CP.

These examples showcase how these extrapolations can be inaccurate. We note that gradient-based optimization has a similar problem: general theoretical guarantees only exist for local minima, not global, and there is an underlying assumption that these minima will be good enough solutions. Nevertheless, here the problem is exacerbated, for the comparisons between corners are all that matters to find good solutions. The failure cases arise because the methods only take them into account indirectly. Truly, they may not even reach a local minimum in a neighbourhood of Hamming distance 1, such as in Example 7. In the experimental section, we will compare CP methods to alternative approaches in a range of different settings to see how well the extrapolations hold.

Chapter 4

Monte Carlo gradient estimation

This chapter starts by presenting an alternative way of writing pseudo-Boolean optimization by optimizing probabilities instead (Section 4.1.1). Then, in Section 4.1.2, we talk about how to use Monte Carlo estimation on this framework. Section 4.2 starts by categorizing MC gradient estimation methods and explaining why some of them cannot be used here, then explaining the ones this work will focus on. Section 4.3 presents counter-examples elucidating some of the drawbacks of the stochastic approach, as well as presenting a bound that the loss must satisfy for the problem to be solvable, regardless of the variance reduction achieved. Finally, Section 4.4 introduces alternative ways of parametrizing the probabilities, all of which will be investigated in this work.

4.1 Basics

The current section presents the basic framework used as basis for all methods from this chapter by firstly showing how to write the PB optimization problem using probabilities. Then we discuss how to make such a framework more practical by estimating expectations with samples.

4.1.1 A Probabilistic Framework

As mentioned before, in pseudo-Boolean optimization, we want to find:

$$\begin{aligned}\mathbf{z}^* &= \arg\min_{\mathbf{z}\in\{0,1\}^d} J(\mathbf{z}) \\ &= \arg\min_{\mathbf{z}\in\{0,1\}^d} \mathscr{P}_J(\mathbf{z}).\end{aligned}$$

The probabilistic approach involves parametrizing the dimensions as factorized Bernoulli random variables and optimizing their expectation instead. The equivalence becomes more explicit after noticing the following:

Theorem 2. *Assuming a vector $\boldsymbol{\theta} \in [0,1]^d$ and considering $\boldsymbol{z} \in \{0,1\}^d$ such that $z_i \sim Ber(\theta_i)$ and z_i, z_j independent for $i \neq j$, with $\mathscr{P}_J(\cdot)$ as defined in Section 2.1 we have:*

$$\mathop{\mathbb{E}}_{z_i\sim\mathrm{Ber}[\theta_i]}[J(\boldsymbol{z})] = \mathscr{P}_J(\boldsymbol{\theta}). \tag{4.1}$$

Proof Proposition 5 of Boros and Hammer (2002, section 4.2) ■

We rewrite the objective as:

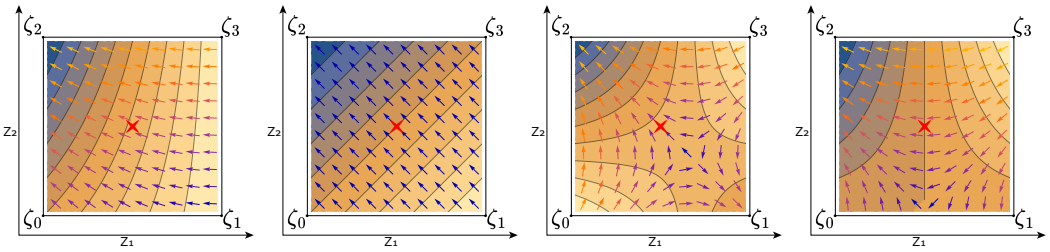
$$\text{minimize} \mathop{\mathbb{E}}_{z_i\sim\mathrm{Ber}[\theta_i]}[J(\mathbf{z})], \qquad \text{for } \boldsymbol{\theta} \in [0,1]^d. \tag{4.2}$$

It is straightforward to verify the equivalence. On the one hand, by Equation (4.1), the optimization in Equation (4.2) has the same objective function as the previous one: $\mathscr{P}_J(\cdot)$, but now it considers the whole hypercube, not only the vertices. Therefore:

$$\begin{aligned}\min_{\boldsymbol{\theta}\in[0,1]^d} \mathscr{P}_J(\boldsymbol{\theta}) &\leq \min_{\mathbf{z}\in\{0,1\}^d} \mathscr{P}_J(\mathbf{z}) \\ \min_{\boldsymbol{\theta}\in[0,1]^d} \mathop{\mathbb{E}}_{z_i\sim\mathrm{Ber}[\theta_i]}[J(\mathbf{z})] &\leq \min_{\mathbf{z}\in\{0,1\}^d} J(\mathbf{z}).\end{aligned}$$

On the other hand, any such expectation is a weighted average of the evaluations in the corners and thus cannot be smaller than the lowest of those values:

$$\min_{\boldsymbol{\theta}\in[0,1]^d} \mathop{\mathbb{E}}_{z_i\sim\mathrm{Ber}[\theta_i]}[J(\mathbf{z})] \geq \min_{\mathbf{z}\in\{0,1\}^d} J(\mathbf{z}).$$



(a) $\left[\begin{smallmatrix} m & M \\ l & M \end{smallmatrix}\right]$ (b) $\left[\begin{smallmatrix} m & l \\ l & M \end{smallmatrix}\right]$ (c) $\left[\begin{smallmatrix} m & M \\ M & l \end{smallmatrix}\right]$ (d) $\left[\begin{smallmatrix} m & M \\ l & l \end{smallmatrix}\right]$

Figure 4.1: Contour plots, captions indicate the matrix $\left[\begin{smallmatrix} J(\zeta_2) & J(\zeta_3) \\ J(\zeta_0) & J(\zeta_1) \end{smallmatrix}\right]$, blue regions indicate low $J(\cdot)$, arrows indicate gradients.

Therefore, the two minima have to be the same. The main change from the original problem is that, in case of multiple maximizing $\mathbf{z}$ values, the probabilistic form will also include solutions that sample randomly between them. Daulton *et al.* (2022, theorem 1) formalize this result and extend it to ordinal and categorical formulations.

We illustrate the optimization surface obtained from this parametrization in the following example:

Example 8. *Similarly to Example 1, we use* $d = 2$*, then:*

$$\begin{aligned} \mathop{\mathbb{E}}_{z_i \sim \mathrm{Ber}[\theta_i]} [J(\boldsymbol{z})] &= \mathscr{P}_J(\boldsymbol{\theta}) \\ &= \overline{\theta}_1 \overline{\theta}_2 J(\boldsymbol{\zeta}_0) + \theta_1 \overline{\theta}_2 J(\boldsymbol{\zeta}_1) + \overline{\theta}_1 \theta_2 J(\boldsymbol{\zeta}_2) + \theta_1 \theta_2 J(\boldsymbol{\zeta}_3). \end{aligned}$$

Assuming global optimum $m = 0.0$*, local optimum* $l = 0.5$ *and maximum* $M = 1.0$*, Figure 4.2 illustrates the contour plots of* $\mathop{\mathbb{E}}_{z_i \sim \mathrm{Ber}[\theta_i]} [J(\boldsymbol{z})]$ *for configurations listed in the captions.*

At least in this simple example, the optimization surface of the probabilistic objective looks reasonable.

Using the probabilistic objective instead of the discrete one benefits from the fact that we can now piggyback on advances from related fields which solve similar problems. One can, for example, attempt to discover latent structure on some simpler space and find the solution using variational inference (Blei *et al.* 2017). Alternatively, one can also attempt to adapt novelty search approaches from machine learning methods or exploration strategies from reinforcement

learning (Ladosz *et al.* 2022). In this chapter, we will focus on applying Monte Carlo methods, described in the next sections.

4.1.2 Using samples

Although we can rewrite the problem using the probabilistic formulation, we remind the reader that the expectation in Equation (4.2) consists of a sum of 2^d terms and therefore still suffers from the problems of the original formulation. Nonetheless, we can use Monte Carlo methods, which approximate the expected value by sampling. In its simplest form, we could, for example, approximate the expected cost by using

$$\mathop{\mathbb{E}}_{z_i \sim \mathrm{Ber}[\theta_i]} [J(\mathbf{z})] \approx \frac{1}{n} \sum_{s=1}^{n} J(\mathbf{z}^{(s)}), \qquad \text{for } \mathbf{z}^{(s)} \sim \prod_{i=1}^{d} \mathrm{Ber}[\theta_i], \tag{4.3}$$

where $\mathbf{z}^{(s)}$ indicates the s-th sample and $\mathbf{z} \sim \prod_{i=1}^{d} \mathrm{Ber}[\theta_i]$ is an alternative notation to indicate that z_i are independent Bernoulli random variables. Here, in addition to the d per-sample coordinates (i.e. $z_i^{(s)}$ for sample s) being independent, the n samples (i.e. $\mathbf{z}^{(s)}$ for $s \in \{1 \dots n\}$) are iid.

By the law of large numbers, the RHS of Equation (4.3) will converge to the true expectation as $n \to \infty$. We can also show that[1]

$$\mathop{\mathbb{E}}_{\{\mathbf{z}^{(s)} \sim p_{\boldsymbol{\theta}}(\cdot)\}_{s=1}^{n}} \left[\frac{1}{n} \sum_{s=1}^{n} J(\mathbf{z}^{(s)}) \right] = \mathop{\mathbb{E}}_{z_i \sim \mathrm{Ber}[\theta_i]} [J(\mathbf{z})]$$

and

$$\mathrm{Var}\left[\frac{1}{n} \sum_{s=1}^{n} J(\mathbf{z}^{(s)}) \right] = \frac{1}{n} \mathrm{Var}\left[J(\mathbf{z}) \right]. \tag{4.4}$$

Alternative estimators to that of the RHS in Equation (4.3) are also possible, where each technique has its own pros and cons. In case the expected value of the estimator is equal to the desired quantity, the estimator is called unbiased. In some cases, a biased estimator may have lower variance or better convergence rate than unbiased alternatives. In this work, however, we will focus on unbiased estimators. Monte Carlo estimation is an active area of

[1] Where $\{\mathbf{z}^{(s)} \sim p_{\boldsymbol{\theta}}(\cdot)\}_{s=1}^{n}$ denotes that $\mathbf{z}^{(s)} \sim p_{\boldsymbol{\theta}}(\cdot)$ for $s \in \{1, \dots, n\}$. We will also use $\{\mathbf{z}^{(s)}\}_{s=1}^{n}$ to denote $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(n)}\}$.

research and techniques improving estimation include: control variates, Rao-Blackwellization, stratification, importance sampling and antithetic sampling. For a more in-depth introduction to these, refer to Owen (2013)

In our problem, however, we do not simply aim at estimating the objective for some fixed $\boldsymbol{\theta}$, but at finding the minimizing $\boldsymbol{\theta}$. For this, we can apply a SGD procedure where we choose an initial $\boldsymbol{\theta}_0$, estimate the gradient of the objective and use it to update $\boldsymbol{\theta}$. Note, however, that, on typical SGD usage in machine learning problems we have that

$$\nabla_{\mathbf{w}} \mathop{\mathbb{E}}_{X,Y\sim\mathcal{D}} \left[\ell(f(X,\mathbf{w}),Y)\right] \approx \frac{1}{B}\sum_{b=1}^{B} \nabla_{\mathbf{w}} \ell(f(X^{(b)},\mathbf{w}),Y^{(b)}).$$

This estimator is unbiased because the gradient can simply be moved outside (or inside) the expectation:

$$\begin{aligned}\mathop{\mathbb{E}}_{X^{(b)},Y^{(b)}\sim\mathcal{D}} \left[\frac{1}{B}\sum_{b=1}^{B} \nabla_{\mathbf{w}} \ell(f(X^{(b)},\mathbf{w}),Y^{(b)})\right] &= \nabla_{\mathbf{w}} \mathop{\mathbb{E}}_{X^{(b)},Y^{(b)}\sim\mathcal{D}} \left[\frac{1}{B}\sum_{b=1}^{B} \ell(f(X^{(b)},\mathbf{w}),Y^{(b)})\right] \\ &= \nabla_{\mathbf{w}} \mathop{\mathbb{E}}_{X,Y\sim\mathcal{D}} \left[\ell(f(X,\mathbf{w}),Y)\right].\end{aligned}$$

Interchanging the order of expectation and gradient is only possible because there is no dependency between the sampling probabilities and the desired parameter (i.e. $\mathcal{D}$ is not a function of $\mathbf{w}$ in the above example). If there was such a dependency, the estimator would change. For example:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)} \left[f(\mathbf{z},\boldsymbol{\theta})\right] &= \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\mathbf{z}) f(\mathbf{z},\boldsymbol{\theta}) \mathrm{d}\mathbf{z} \\ &= \int \left(\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z})\right) f(\mathbf{z},\boldsymbol{\theta}) + p_{\boldsymbol{\theta}}(\mathbf{z}) \nabla_{\boldsymbol{\theta}} f(\mathbf{z},\boldsymbol{\theta}) \mathrm{d}\mathbf{z} \\ &= \int p_{\boldsymbol{\theta}}(\mathbf{z}) \left(\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z})\right) f(\mathbf{z},\boldsymbol{\theta}) + p_{\boldsymbol{\theta}}(\mathbf{z}) \nabla_{\boldsymbol{\theta}} f(\mathbf{z},\boldsymbol{\theta}) \mathrm{d}\mathbf{z} \\ &= \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)} \left[f(\mathbf{z},\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z})\right] + \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)} \left[\nabla_{\boldsymbol{\theta}} f(\mathbf{z},\boldsymbol{\theta})\right] \\ &\neq \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)} \left[\nabla_{\boldsymbol{\theta}} f(\mathbf{z},\boldsymbol{\theta})\right]. \end{aligned} \tag{4.5}$$

Estimating $\nabla_{\boldsymbol{\theta}} \mathbb{E}\left[f(\mathbf{z},\boldsymbol{\theta})\right]$ with:

$$\frac{1}{n}\sum_{s=1}^{n} \nabla_{\boldsymbol{\theta}} f(\mathbf{z}^{(s)},\boldsymbol{\theta}) \qquad \text{for } s = \{1,\ldots,n\}$$

like before would only account for $\mathbb{E}\left[\nabla_{\boldsymbol{\theta}} f(\mathbf{z}, \boldsymbol{\theta})\right]$. The cost function in Equation (4.2) presents a dependency of the probability on the gradient parameter. Therefore, gradient estimators have to account for that. Particularly, given samples $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}$ and denoting $(\mathbf{z}^{(s)})_{s=1}^{n}$ as the ordered set containing these samples, we want $\hat{\boldsymbol{g}}(\cdot)$ such that:

$$\mathbb{E}\left[\hat{\boldsymbol{g}}((\mathbf{z}^{(s)})_{s=1}^{n}; \boldsymbol{\theta})\right] = \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{z_i \sim \mathrm{Ber}[\theta_i]} \left[J(\mathbf{z})\right]$$

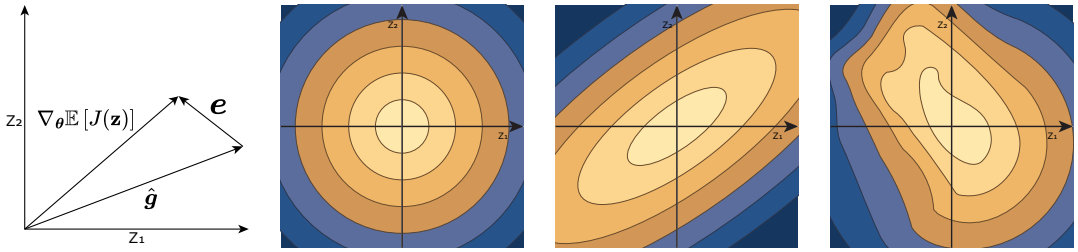
Section 4.2 will present some different $\hat{\boldsymbol{g}}(\cdot)$ to estimate the gradient. When estimating scalars, such as in Equation (4.3), different unbiased estimators are usually chosen based simply on how much they can reduce the variance in the desired problems. For gradient estimation, nonetheless, d is usually greater than one and there are multiple scalar variances, one per dimension. In that case, variance is usually considered element-wise or as a sum of these d individual scalar variances. Notice that the estimated gradient $\hat{\boldsymbol{g}}$ is a random variable and that its deviation from the true gradient can be written as:

$$\boldsymbol{e} = \nabla_{\boldsymbol{\theta}} \mathbb{E}\left[J(\mathbf{z})\right] - \hat{\boldsymbol{g}}.$$

Figure 4.2a illustrates these vectors. The expected squared L_2-norm of the error random variable $\boldsymbol{e}$ is given by:

$$\begin{aligned}
\mathbb{E}\left[\sum_{i=1}^{d} e_i^2\right] &= \mathbb{E}\left[\sum_{i=1}^{d} (\hat{g}_i - (\nabla_{\boldsymbol{\theta}} \mathbb{E}\left[J(\mathbf{z})\right])_i)^2\right] \\
&= \sum_{i=1}^{d} \mathbb{E}\left[(\hat{g}_i - \frac{\partial}{\partial \theta_i} \mathbb{E}\left[J(\mathbf{z})\right])^2\right] \\
&= \sum_{i=1}^{d} \mathrm{Var}\left[\hat{g}_i\right].
\end{aligned}$$

Which corresponds to the sum we mentioned. For unbiased estimators, $\boldsymbol{e}$ has mean equal to zero, but its covariance matrix will depend on the distribution of $\hat{\boldsymbol{g}}$. The variable $\boldsymbol{e}$ may have identity covariance matrix such as in Figure 4.2b, but it may also behave irregularly, such as in Figure 4.2c or Figure 4.2d. The dimensions are often looked at independently, but it is possible that, for example, some alignment between the eigenvectors of the covariance matrix and the true gradient may lead to desirable properties. To our knowledge, covariance between dimensions has not been sufficiently investigated yet.



(a) (b) (c) (d)

Figure 4.2: (a) Illustration of the main multivariate gradient estimation vectors and (b-d) example contour plots of the PDF for different $\boldsymbol{e}$.

4.2 Methods

Firstly, we give a broad overview of Monte Carlo gradient estimation. Then, we detail the methods used in our experiments.

4.2.1 Overview

On a recent survey, Mohamed *et al.* (2020) categorized approaches for MC gradient estimation, arriving in three fundamentally distinct strategies: score function estimation, pathwise gradient estimation and measure-valued gradients. The first one follows the derivation outlined in Equation (4.5), substituting $f(\mathbf{z}, \boldsymbol{\theta})$ for $J(\mathbf{z})$, this estimator becomes:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)} [J(\mathbf{z})] &= \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)} [J(\mathbf{z}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z})] \\ &\approx \frac{1}{n} \sum_{s=1}^{n} J(\mathbf{z}^{(s)}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}). \end{aligned} \tag{4.6}$$

For our purposes, $p_{\boldsymbol{\theta}}(\cdot)$ is a discrete distribution, but the estimator is also applicable in the continuous case.[2] In practice, this estimator is known to have high variance. In some cases, it is possible to instead use pathwise gradient estimators, also known as reparametrization trick, by writing:

$$\mathbf{z} = f_z(\boldsymbol{\epsilon}; \boldsymbol{\theta}) \qquad \text{For } \boldsymbol{\epsilon} \sim p(\cdot). \tag{4.7}$$

[2]This estimator will be biased in some cases, as explained in Mohamed *et al.* (2020, section 4.3.2) Particularly, for vector $\mathbf{z}$, scalar θ and $h \to 0$, they show that $p(\mathbf{z}; \theta + h)$ has to be greater than zero for every $\mathbf{z}$ where $p(\mathbf{z}; \theta) > 0$. See the referred section for more details, as well as a simple case where this condition is not met. For our use cases, however, it is unbiased.

Where the underscript in $f_z(\cdot)$ is merely to indicate that it maps to $\mathbf{z}$ (i.e. not a parameter relation, like in $p_{\boldsymbol{\theta}}(\cdot) = p(\cdot; \boldsymbol{\theta})$). Note that $p(\cdot)$ above no longer depends on $\boldsymbol{\theta}$. In that case, we have:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\mathbf{z} \sim p_{\boldsymbol{\theta}}(\cdot)} [J(\mathbf{z})] &= \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{\epsilon} \sim p(\cdot)} [J(f_z(\boldsymbol{\epsilon}; \boldsymbol{\theta}))] \\
&= \mathop{\mathbb{E}}_{\boldsymbol{\epsilon} \sim p(\cdot)} [\nabla_{\boldsymbol{\theta}} J(f_z(\boldsymbol{\epsilon}; \boldsymbol{\theta}))] \\
&= \mathop{\mathbb{E}}_{\boldsymbol{\epsilon} \sim p(\cdot)} \left[\nabla_{\boldsymbol{\theta}} f_z(\boldsymbol{\epsilon}; \boldsymbol{\theta}) (\nabla_{\mathbf{z}} J(\mathbf{z})) \big|_{\mathbf{z} = f_z(\boldsymbol{\epsilon}; \boldsymbol{\theta})} \right] \\
&\approx \frac{1}{n} \sum_{s=1}^{n} \nabla_{\boldsymbol{\theta}} f_z(\boldsymbol{\epsilon}^{(s)}; \boldsymbol{\theta}) (\nabla_{\mathbf{z}} J(\mathbf{z})) \big|_{\mathbf{z} = f_z(\boldsymbol{\epsilon}^{(s)}; \boldsymbol{\theta})}. \qquad (4.8)
\end{aligned}$$

One very popular distribution where it is possible to write $\mathbf{z}$ as in Equation (4.7) is the multivariate Normal. This method is well-known empirically for having lower variance compared to the score function estimator from Equation (4.6), with successes in generative modeling (Kingma and Welling 2013) and reinforcement learning (Haarnoja *et al.* 2018). More recently Lan *et al.* (2021) derived alternatives to some of the main theorems in reinforcement learning to use pathwise gradient estimators instead of the score function counterparts.

Interestingly, the variance of the reparametrized estimators can be shown to be bounded by the squared Lipschitz constant of the cost function $J(\cdot)$ (Glasserman 2004, section 7.2.2 and Fan *et al.* 2015, section 10) without the presence of the dimension d in the bound. Conversely, as we will see in Section 4.3, the dimensionality of $\mathbf{z}$ can have catastrophic consequences for the score function estimators from Equation (4.6). Xu *et al.* (2019) compares both estimators theoretically in a simplified variational inference setting, attributing the increased variance of the score function estimators to the presence of higher order terms in the variance formulas. These successes led to generalizations of the reparametrization trick, including relaxations allowing $p(\cdot)$ in Equation (4.7) to also be a function of the $\boldsymbol{\theta}$ parameter[3], which in turn allowed the method to be applicable to the Beta, Gamma and Dirichlet distributions.

[3]In this case, more terms appear in the formula to correct for the dependency.

Nonetheless, this method requires differentiable cost functions and continuous inputs, while $\mathbf{z} \in \{0,1\}^d$ in our problem setting. As mentioned in Bengio *et al.* (2013), a reparametrization exists for Bernoulli variables, since, for $\mathbf{u} \sim \prod_{i=1}^{d} U[0,1]$:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{z_i \sim \mathrm{Ber}[\theta_i]} [J(\mathbf{z})] &= \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{u_i \sim U[0,1]} [J(\mathbb{1}[\mathbf{u} \leq \boldsymbol{\theta}])] \\ &= \mathop{\mathbb{E}}_{u_i \sim U[0,1]} [\nabla_{\boldsymbol{\theta}} J(\mathbb{1}[\mathbf{u} \leq \boldsymbol{\theta}])] \,. \end{aligned} \tag{4.9}$$

Still, similarly to what happened in Section 3.2, the coordinates of the gradient of $J(\mathbb{1}[\mathbf{u} \leq \boldsymbol{\theta}])$ are undefined for $\theta_i = u_i$ and zero everywhere else, rendering the reparametrization trick unusable.

Moreover, variance when using reparametrization is not guaranteed to be lower than when using the score function. As mentioned before, it depends on the Lipschitz constant of $J(\cdot)$, which can act as a double-edged knife and cause them to perform worse than the score function methods (Mohamed *et al.* 2020, figure 3, section 3 and section 5.3.2). Despite all that, multiple methods attempt to incorporate the reparametrization trick in discrete settings somehow, culminating in estimators that are hybrid, with characteristics from both the CP approaches from Section 3.2 as well as approaches from the current chapter. These approaches will be discussed in Chapter 5. One of the goals of this work is to understand if incorporating the gradients is indeed beneficial, as well as to explain why.

Finally, measure-valued gradients, which correspond to the third category, are not very common in machine learning. They are also unbiased and their derivation uses Hahn decomposition theorem to write the gradient of some signed probability measure as a difference of two unsigned measures (Mohamed *et al.* 2020, chapter 6). However, these estimators require $O(nd)$ evaluations of $J(\cdot)$, the same problem the PB optimization methods from Section 2.3 suffer from. For the reasons discussed, this chapter will focus only on score function estimators.

4.2.2 REINFORCE

REINFORCE (Williams 1992), which also sometimes appears in the literature with alternative names (Glynn 1990; Rubinstein and Shapiro 1990), corresponds to the simplest form of score function estimator, which we derived and discussed above. For convenience, we repeat its equation in this section:

$$\hat{\boldsymbol{g}}_{REINFORCE}((\mathbf{z}^{(s)})_{s=1}^{n};\boldsymbol{\theta}) = \frac{1}{n}\sum_{s=1}^{n} J(\mathbf{z}^{(s)})\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}), \qquad \text{for } \mathbf{z}^{(s)} \sim p_{\boldsymbol{\theta}}(\cdot). \tag{4.10}$$

In our case, we also have:

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{z}) &= \prod_{i=1}^{d} \text{Ber}[\theta_i] \\ &= \prod_{i=1}^{d} \theta_i^{z_i} \overline{\theta}_i^{\overline{z}_i}. \end{aligned} \tag{4.11}$$

For our experimental section, each $\mathbf{z}$ will be a mask applied element-wise to large neural network weights. To avoid storing all n samples in memory simultaneously, which would have memory cost $O(nd)$, we resort to iterative versions of the estimators, changing the memory cost to $O(d)$. For REINFORCE, this simply corresponds to following Algorithm 3.

Algorithm 3 Iterative form of REINFORCE

$\hat{\boldsymbol{g}} \leftarrow \mathbf{0}$;
for $s = 1 \cdots n$ **do**
  Sample $\mathbf{z}^{(s)}$ according the distribution from Equation (4.11)
  $\hat{\boldsymbol{g}} \leftarrow \frac{s-1}{s}\hat{\boldsymbol{g}} + \frac{1}{s} J(\mathbf{z}^{(s)})\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)})$
Return: $\hat{\boldsymbol{g}}$

As mentioned above, REINFORCE has very high variance. For this reason, it is common to apply some form of variance reduction technique to $\hat{\boldsymbol{g}}_{REINFORCE}$. One of the main ones, which will be used in most of the methods discussed in this paper, is called control variates. Briefly going back to the case of estimating $\mathbb{E}\left[J(\mathbf{z})\right]$ instead of its gradient, notice that, for arbitrary $h(\cdot)$ and considering:

$$\mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)}\left[h(\mathbf{z})\right] = \mu_h,$$

where μ_h a known constant[4], we can write:

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)}[J(\mathbf{z})] &= \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)}[J(\mathbf{z}) - h(\mathbf{z})] + \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)}[h(\mathbf{z})] \\ &\approx \left(\frac{1}{n}\sum_{s=1}^{n} J(\mathbf{z}^{(s)}) - h(\mathbf{z}^{(s)})\right) + \mu_h.\end{aligned}$$

The above quantity will be an unbiased estimator of $\mathbb{E}[J(\mathbf{z})]$ and its variance is equal to:

$$\frac{1}{n}\mathrm{Var}\left[J(\mathbf{z}) - h(\mathbf{z})\right].$$

Comparing it to Equation (4.4), we can see that, if the difference $J(\mathbf{z}) - h(\mathbf{z})$ has smaller variance than $J(\mathbf{z})$, using control variates will be beneficial. Going back to estimating gradients, we denote:

$$\boldsymbol{g}_h = \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)}[h(\mathbf{z})]$$

and similarly, by adding and subtracting this quantity, we can arrive at:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)}[J(\mathbf{z})] &= \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)}[(J(\mathbf{z}) - h(\mathbf{z}))\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z})] + \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)}[h(\mathbf{z})] \\ &\approx \left(\frac{1}{n}\sum_{s=1}^{n}(J(\mathbf{z}^{(s)}) - h(\mathbf{z}^{(s)}))\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)})\right) + \boldsymbol{g}_h. \qquad (4.12)\end{aligned}$$

By using this principle, some methods select $h(\cdot)$ close to $J(\cdot)$, but whose corresponding $\nabla_{\boldsymbol{\theta}}\mathscr{P}_h(\cdot)$ is feasible to compute, such as a Taylor expansion of $J(\cdot)$ (Gu *et al.* 2015). More generally, other methods subtract and re-add the expectation, but use a reparametrization estimator instead of $\boldsymbol{g}_h$, combining score function and pathwise gradient estimation and resulting in hybrid methods (Grathwohl *et al.* 2017; Tucker *et al.* 2017). As mentioned before, some of these methods will be discussed in Chapter 5. In this section, we consider control variates which do not rely on $\nabla J(\cdot)$.

[4]We omit its dependence on $\boldsymbol{\theta}$ to simplify the notation.

4.2.3 LOORF

Proposed initially by Kool *et al.* (2019), this method arises by deriving a control variate similarly to Equation (4.12), but now considering all of the samples simultaneously when designing the baseline to be subtracted. Particularly, for a single $s \in \{1, \dots, n\}$, we denote the ordered set containing all other samples by

$$(\mathbf{z}^{(s')})_{s' \neq s} = (\mathbf{z}^{(s')} \mid s' \in \{1, \dots, n\} \setminus \{s\}).$$

Then, considering $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(n)}$ iid, we have:

$$\begin{aligned}
\mathop{\mathbb{E}}_{\mathbf{z}^{(s)} \sim p_{\boldsymbol{\theta}}(\cdot)} \left[J(\mathbf{z}^{(s)}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}) \right] &= \mathbb{E}\left[(J(\mathbf{z}^{(s)}) - h((\mathbf{z}^{(s')})_{s' \neq s})) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}) \right] + \\
&\qquad \mathbb{E}\left[h((\mathbf{z}^{(s')})_{s' \neq s}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}) \right] \\
&= \mathbb{E}\left[(J(\mathbf{z}^{(s)}) - h((\mathbf{z}^{(s')})_{s' \neq s})) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}) \right] + \\
&\qquad \mathbb{E}\left[h((\mathbf{z}^{(s')})_{s' \neq s}) \right] \cancelto{0}{\mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}) \right]} \\
&= \mathbb{E}\left[(J(\mathbf{z}^{(s)}) - h((\mathbf{z}^{(s')})_{s' \neq s})) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}) \right].
\end{aligned} \tag{4.13}$$

Where we used that the expected value of the score function is zero. This means that we can use any function of the other $n-1$ samples to compose the control variate. To keep $h(\cdot)$ and $J(\cdot)$ close, Kool *et al.* (2019) use:

$$h((\mathbf{z}^{(s')})_{s' \neq s}) = \frac{1}{n-1} \sum_{s' \neq s} J(s'),$$

for iid $\mathbf{z}^{(s)} \sim p_{\boldsymbol{\theta}}(\cdot)$. By linearity of expectations, we can average Equation (4.13) for all s and the estimator then becomes:

$$\begin{aligned}
\hat{\boldsymbol{g}}_{LOORF}((\mathbf{z}^{(s)})_{s=1}^{n}; \boldsymbol{\theta}) &= \frac{1}{n} \sum_{s=1}^{n} \left(J(\mathbf{z}^{(s)}) - \frac{1}{n-1} \sum_{s' \neq s} J(\mathbf{z}^{(s')}) \right) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}) \\
&= \frac{1}{n-1} \sum_{s=1}^{n} \left(J(\mathbf{z}^{(s)}) - \frac{1}{n} \sum_{s=1}^{n} J(\mathbf{z}^{(s')}) \right) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}).
\end{aligned} \tag{4.14}$$

Where the proof of equivalence between the two forms is shown in Kool *et al.* (2019). Although this is a simple estimator, it is a strong baseline which has outperformed more advanced variance reduction techniques (Dimitriev and M. Zhou 2021). Richter *et al.* (2020) shows an alternative derivation of this estimator using a variational inference perspective and also enumerates conditions in which it behaves closely to the optimal control variate. Similarly to REINFORCE, we require a way of sampling iteratively, which is shown in Algorithm 4

Algorithm 4 Iterative form of LOORF

Start accumulators:
$\hat{J} \leftarrow 0$;
$\Lambda_{\nabla \log} \leftarrow \mathbf{0}$;
$\Lambda_{J\nabla \log} \leftarrow \mathbf{0}$;
for $s = 1 \cdots n$ **do**
  Sample $\mathbf{z}^{(s)}$ according the distribution from Equation (4.11)
  $\hat{J} \leftarrow \frac{s-1}{s}\hat{J} + \frac{1}{s}J(\mathbf{z}^{(s)})$
  $\Lambda_{\nabla \log} \leftarrow \frac{\max\ (s-2,1)}{\max\ (s-1,1)}\Lambda_{\nabla \log} + \frac{1}{\max\ (s-1,1)}\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)})$
  $\Lambda_{J\nabla \log} \leftarrow \frac{\max\ (s-2,1)}{\max\ (s-1,1)}\Lambda_{J\nabla \log} + \frac{1}{\max\ (s-1,1)}J(\mathbf{z}^{(s)})\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)})$
$\hat{\boldsymbol{g}} \leftarrow \Lambda_{J\nabla \log} - \Lambda_{\nabla \log}\hat{J}$
Return: $\hat{\boldsymbol{g}}$

4.2.4 ARMS

This estimator extends the idea of antithetic sampling from Monte Carlo estimation to gradient estimation. The base principle is to use samples that are opposite in some way and rely on their error cancellation to reduce variance. To understand it, we once more go back to the case where we are estimating a scalar quantity. Say we want to estimate $\mathbb{E}[f(\mathbf{u})]$, for $\mathbf{u} \in \mathcal{D}$ (instead of $\{0,1\}^d$). Furthermore, assume that $\mathbf{u} \sim p(\cdot)$, where $p(\cdot)$ is a symmetric density with respect to point $\mathbf{c}$. Namely, we define a reflection of $\mathbf{u}$ through $\mathbf{c}$, here called $\overline{\mathbf{u}}$, such that:

$$\mathbf{u} - \mathbf{c} = -(\overline{\mathbf{u}} - \mathbf{c}).$$

For $p(\cdot)$ to be symmetric, we must have:

$$p(\mathbf{u}) = p(\overline{\mathbf{u}}).$$

Notice that requiring such a density is not too restrictive. The uniform density on $\mathcal{D} = [0,1]^d$ with $\mathbf{c} = [0.5 \;\; 0.5 \ldots]^\top$ and $\overline{\mathbf{u}} = \mathbf{1} - \mathbf{u}$, for example, satisfies this condition. This density is commonly used as the basis for sampling from some distributions, since applying specific functions to uniform random variables, such as when using inverse transform sampling or the Gumbel max trick, is often statistically equivalent to sampling from the desired distributions (Owen 2013, chapter 4). The antithetic sampling estimate is obtained by:

$$\mathop{\mathbb{E}}_{\mathbf{u}\sim p(\cdot)}[f(\mathbf{u})] \approx \frac{1}{n}\sum_{s=1}^{n/2} f(\mathbf{u}^{(s)}) + f(\overline{\mathbf{u}}^{(s)}).$$

The efficacy of this estimator for variance reduction will depend heavily on how $f(\cdot)$ behaves. Its variance is equal to:

$$\frac{1}{n}\mathrm{Var}\left[f(\mathbf{u})\right](1 + \mathrm{Corr}\left[f(\mathbf{u}), f(\overline{\mathbf{u}})\right]),$$

as opposed to $(1/n)\mathrm{Var}\left[f(\mathbf{u})\right]$ as in simple Monte Carlo estimation. In a monotonic function, for instance, the correlation should be closer to -1, causing antithetic sampling to be efficient. More generally, we can write $f(\cdot)$ as a sum of even part $f_E(\cdot)$ and odd part $f_O(\cdot)$:[5]

$$f(\mathbf{u}) = \underbrace{\frac{f(\mathbf{u}) + f(\overline{\mathbf{u}})}{2}}_{f_E(\mathbf{u})} + \underbrace{\frac{f(\mathbf{u}) - f(\overline{\mathbf{u}})}{2}}_{f_O(\mathbf{u})}.$$

The odd part has expectation zero and the even part has the same expectation as $f(\cdot)$. Antithetic sampling has benefit of eliminating the variance from the odd part, but has the drawback of doubling the variance from the even part. Going back to the discrete case, estimating the expected loss can be done as follows:

$$\begin{aligned}\mathop{\mathbb{E}}_{z_i\sim\mathrm{Ber}[\theta_i]}[J(\mathbf{z})] &= \mathop{\mathbb{E}}_{u_i\sim U[0,1]}[J(\mathbb{1}[\mathbf{u} \le \boldsymbol{\theta}])] \\ &\approx \frac{1}{n}\sum_{s=1}^{n/2} J(\mathbb{1}[\mathbf{u}^{(s)} \le \boldsymbol{\theta}]) + J(\mathbb{1}[\overline{\mathbf{u}}^{(s)} \le \boldsymbol{\theta}]).\end{aligned}$$

[5]Even and odd are meant with respect to $\mathbf{c}$, not $\mathbf{0}$.

For $\mathbf{u}^{(s)} \sim \prod_{i=1}^{d} U[0,1]$ and $\overline{\mathbf{u}}^{(s)} = \mathbf{1} - \mathbf{u}^{(s)}$.

We now move the discussion back to gradient estimation. In the context of Neural Networks, the first paper that tried to incorporate this technique was ARM (Yin and M. Zhou 2018), which was later improved by two concurrent works: DisARM (Dong *et al.* 2020) and U2G (Yin, Ho, *et al.* 2020), both discovering an equivalent improvement of ARM independently. Notably, however, these methods seem to underperform LOORF for larger n as noted by Dimitriev and M. Zhou (2021). Specifically, this work argues that LOORF leverages all possible combinations of pairs among the n samples, whereas ARM/U2G/DisARM only combine the antithetic pairs.

To use the idea of antithetic sampling while still taking advantage of all pairs, they propose ARMS. Starting with $d = 1$ and $n = 2$, considering $z^{(1)}, z^{(2)} \overset{iid}{\sim} \text{Ber}[\theta]$ we can write the expected value of LOORF as:

$$\begin{aligned}\mathop{\mathbb{E}}_{p_\theta(\cdot)}\left[\hat{g}_{LOORF}((z^{(1)}, z^{(2)}); \theta)\right] &= \mathop{\mathbb{E}}_{\tilde{p}_\theta(\cdot)}\left[\frac{p_\theta(\tilde{z}^{(1)}, \tilde{z}^{(2)})}{\tilde{p}_\theta(\tilde{z}^{(1)}, \tilde{z}^{(2)})}\hat{g}_{LOORF}((\tilde{z}^{(1)}, \tilde{z}^{(2)}); \theta)\right] \\ &= \mathop{\mathbb{E}}_{\tilde{p}_\theta(\cdot)}\left[\frac{p_\theta(\tilde{z}^{(1)})p_\theta(\tilde{z}^{(2)})}{\tilde{p}_\theta(\tilde{z}^{(1)}, \tilde{z}^{(2)})}\hat{g}_{LOORF}((\tilde{z}^{(1)}, \tilde{z}^{(2)}); \theta)\right].\end{aligned} \tag{4.15}$$

Where we use importance sampling (Owen 2013, chapter 9) to change the sampling distribution, introducing dependency between the random variables.[6] Dimitriev and M. Zhou (2021) also require that the marginals of the importance distribution remain the same as the ones from the nominal distribution, namely:

$$\int \tilde{p}_\theta(\tilde{z}^{(1)}, \tilde{z}^{(2)}) \mathrm{d}\tilde{z}^{(2)} = \tilde{p}_\theta(\tilde{z}^{(1)}) = p_\theta(\tilde{z}^{(1)}),$$

and similarly for $\tilde{p}_\theta(\tilde{z}^{(2)})$. Using the fact that the $\hat{g}_{LOORF}((\tilde{z}^{(1)}, \tilde{z}^{(2)}); \theta) = 0$ whenever $\tilde{z}^{(1)} = \tilde{z}^{(2)}$ as well as that the marginals remain the same, they show that the RHS of Equation (4.15) reduces to:

$$\mathop{\mathbb{E}}_{\tilde{p}_\theta(\cdot)}\left[\frac{1}{1-\rho}\hat{g}_{LOORF}((\tilde{z}^{(1)}, \tilde{z}^{(2)}); \theta)\right], \qquad \text{where } \rho = \text{Corr}\left[\tilde{z}^{(1)}, \tilde{z}^{(2)}\right].$$

[6] Some authors denote importance sampling without changing the variable inside the expectation (i.e. $\mathop{\mathbb{E}}_{p(\cdot)}[f(x)] = \mathop{\mathbb{E}}_{q(\cdot)}\left[\frac{p(x)}{q(x)}f(x)\right]$), we opt to use $\tilde{z}$ instead of z to emphasize that the sampling distribution changes.

Notably, if ρ is the smallest negative value possible, we recover the antithetic pair from above (i.e. $(\tilde{z}^{(1)}, \tilde{z}^{(2)}) = (\mathbb{1}[u^{(1)} \leq \theta], \mathbb{1}[\overline{u}^{(1)} \leq \theta])$). After some algebra, the expression above can be generalized to $d > 1$ and $n > 2$, yielding the ARMS estimator:

$$\hat{\boldsymbol{g}}_{ARMS}((\tilde{\mathbf{z}}^{(s)})_{s=1}^{n}; \boldsymbol{\theta}) = \frac{1}{1-\boldsymbol{\rho}} \hat{\boldsymbol{g}}_{LOORF}((\tilde{\mathbf{z}}^{(s)})_{s=1}^{n}; \boldsymbol{\theta}), \tag{4.16}$$
$$\text{where } \rho_i = \text{Corr}\left[\tilde{z}_i^{(s)}, \tilde{z}_i^{(s')}\right], \text{ for } s \neq s'.$$

Importantly, for each dimension $i \in \{1, \ldots, d\}$, they assume that the correlation between all pairs of scalar samples $(\tilde{z}_i^{(s)}, \tilde{z}_i^{(s')})$ will be the same value.

To obtain samples satisfying such a structure, they rely on copula sampling (Owen 2013, section 5.6). To summarize, applying a CDF of some random scalar varible to that same variable results in an output distributed according to $U[0, 1]$. Therefore, if we sample a d-dimensional random variable and apply each marginal CDF to the corresponding dimension, we get d (marginally) uniform random variables. If the original distribution causes the d original variables to have mutual negative dependence, the uniforms should keep some of this dependence. Then, these uniform variables can be used to produce Bernoulli samples by the reparametrization from Equation (4.9).

Dimitriev and M. Zhou (2021) propose two different ways to get these uniforms: Dirichlet copula and Gaussian copula. On their experiments, the Dirichlet copula performed the best, and for that reason it is the one we are going to use in this work. The full procedure as proposed in their paper is summarized Algorithm 5. As before, we require the algorithm to be iterative and therefore modify their sampling procedure[7] to the one outlined in Algorithm 6, shown only for $d = 1$ to simplify exposition. Proof of equivalence between the two sampling procedures is provided in Section A.2, as well as the full iterative algorithm to compute the ARMS estimator, obtained by combining Algorithms 4 to 6.

[7]Note that $\sum_{s'=1}^{n} \log u_i^{(s')}$ in Algorithm 5 requires all n samples to have been computed beforehand.

Algorithm 5 Original ARMS with Dirichlet copulas

▷ This loop can be parallelized with vectorized implementations
for $i \in 1 \dots d$ **do**
  Sample $u_i^{(s)} \overset{iid}{\sim} U[0,1]$ for $s \in \{1, \dots, n\}$
  $d_i^{(s)} \leftarrow \frac{\log u_i^{(s)}}{\sum_{s'=1}^{n} \log u_i^{(s')}}$ ▷ Convert iid uniforms to Dirichlet r.v.
  $\tilde{u}_i^{(s)} \leftarrow 1 - (1 - d_i^{(s)})^{n-1}$ ▷ Apply marginal CDF
  if $\theta_i > 0.5$ **then** ▷ Additional steps from ARMS paper
    $\tilde{z}_i^{(s)} \leftarrow \mathbb{1}[\tilde{u}_i^{(s)} \leq \theta_i]$
    $\rho_i \leftarrow \frac{\max(0, 2(1-\theta_i)^{\frac{1}{n-1}} - 1)^{n-1} - (1-\theta_i)^2}{\theta_i(1-\theta_i)}$ ▷ Correlation from ARMS paper
  else
    $\tilde{z}_i^{(s)} \leftarrow \mathbb{1}[1 - \tilde{u}_i^{(s)} \leq \theta_i]$
    $\rho_i \leftarrow \frac{\max(0, 2\theta_i^{1/(n-1)} - 1)^{n-1} - \theta_i^2}{\theta_i(1-\theta_i)}$ ▷ Correlation from ARMS paper
$\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}}_{ARMS}((\tilde{\mathbf{z}}^{(s)})_{s=1}^{n}; \boldsymbol{\theta})$
Return: $\hat{\boldsymbol{g}}$

Algorithm 6 Iterative sampling from Dirichlet copula ($d = 1$)

Sample $\sum_{s'} E^{(s')} \sim \text{Gamma}[n, 1]$
$R \leftarrow \sum_{s'} E^{(s')}$
for $s \in 1 \dots n$ **do**
  if $s < n$ **then**
    Sample $U \sim U[0,1]$
    $E^{(s)} \leftarrow -RU^{\frac{1}{n-s}} + R$
  else
    $E^{(s)} \leftarrow R$
  $R \leftarrow R - E^{(s)}$
  ▷ Get single Dirichlet r.v. (PS: $\sum_{s'} E^{(s')}$ was already computed)
  $d^{(s)} \leftarrow \frac{E^{(s)}}{\sum_{s'} E^{(s')}}$
  $\tilde{u}^{(s)} \leftarrow 1 - (1 - d^{(s)})^{n-1}$ ▷ Apply marginal CDF
  yield $\tilde{u}^{(s)}$

4.2.5 Beta*

As shown in Equation (4.13), if the control variate is of the form $h(\cdot)\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\cdot)$, as long as $h(\cdot)$ is not a function of the current sample $\mathbf{z}^{(s)}$, the correction summand is simply zero. We here consider only constant functions (i.e. $h(\cdot) = \beta$), but allow different β_i for different dimensions. Theorem 3 shows a closed-form expression for the optimal control variate in this case.

Theorem 3. *Consider the problem of estimating:*

$$\frac{\partial}{\partial \theta_i} \mathop{\mathbb{E}}_{\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot)} [J(\boldsymbol{z})] = \mathop{\mathbb{E}}_{\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot)} \left[\frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{z})}{\partial \theta_i} J(\boldsymbol{z}) \right]$$

by sampling $\boldsymbol{z}^{(s)} \overset{iid}{\sim} p_{\boldsymbol{\theta}}(\cdot)$*, for* $s \in \{1, \ldots, n\}$ *and using:*

$$\hat{g}_{\beta_i}((\boldsymbol{z}^{(s)})_{s=1}^n; \boldsymbol{\theta}) = \frac{1}{n} \sum_{s=1}^{n} \Big(J(\boldsymbol{z}^{(s)}) - \beta_i \Big) \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{z}^{(s)})}{\partial \theta_i}. \tag{4.17}$$

The optimal β_i^* *such that*

$$\beta_i^* = \mathop{\arg\min}_{\beta_i \in \mathbb{R}} \operatorname{Var} \left[\hat{g}_{\beta_i}((\boldsymbol{z}^{(s)})_{s=1}^n; \boldsymbol{\theta}) \right]$$

is given by

$$\beta_i^* = \mathop{\mathbb{E}}_{\boldsymbol{z} \sim q_i(\cdot; \boldsymbol{\theta})} [J(\boldsymbol{z})], \qquad \text{where } q_i(\cdot; \boldsymbol{\theta}) \propto p_{\boldsymbol{\theta}}(\cdot) \left(\frac{\partial \log p_{\boldsymbol{\theta}}(\cdot)}{\partial \theta_i} \right)^2.$$

Proof. *Section A.3 in the appendix.*

The quantity $p_{\boldsymbol{\theta}}(\cdot) \left(\frac{\partial \log p_{\boldsymbol{\theta}}(\cdot)}{\partial \theta_i} \right)^2$ above is the integrand of the (diagonal) Fisher information metric for the corresponding i. The Fisher information corresponds to the integral of this function and quantifies how much a random variable $\mathbf{z}$ distributed according to $p_{\boldsymbol{\theta}}(\cdot)$ is predictive of the parameter θ_i. If this quantity is high then it should take less observations of $\mathbf{z}$ to compute θ_i accurately via Monte Carlo. The distribution $q_i(\boldsymbol{\zeta}_h; \boldsymbol{\theta})$, for $h \in \{0, \ldots, 2^d - 1\}$, weights the relative contribution of the value $\boldsymbol{\zeta}_h$ among the possible values $\mathbf{z}$ can assume.

To exemplify, if $d = 1$ and $\theta = 0.999$, sampling z will yield multiple 1 values, but it is the zeros that are going to enable better discernment of whether θ is 0.9, 0.99 or 0.999. Theorem 4 shows what the optimal β_i^* is when $\mathbf{z}$ is sampled from a factorized Bernoulli.

Theorem 4. *For the same setting as in Theorem 3, but with the additional condition that* $p_{\boldsymbol{\theta}}(\cdot) = \prod_{i=1}^{d} \text{Ber}[\theta_i]$*, the optimal* β_i^* *becomes:*

$$\beta_i^* = \mathop{\mathbb{E}}_{z \sim p_{\boldsymbol{\theta}}(\cdot)} \left[J(z_1, \ldots, z_{i-1}, 1 - z_i, z_{i+1}, \ldots) \right].$$

Proof. *Section A.3 in the appendix.*

This control variate changes the input on the i-th coordinate from z_i to $1-z_i$ while keeping the other dimensions as they were, bearing some resemblance to the alternative definition of derivative used in classical PB approaches from Equation (2.5).

Since it is not feasible to compute this estimator in closed-form for the larger experiments, we will only use it for the smaller ones. It should be seen as an upper-bound of variance reduction achievable by the other methods, as well as an indicator of the implications of such variance reduction when solving the desired PB optimization problems.

4.3 Drawbacks

This section overviews issues with the methods presented throughout this chapter.

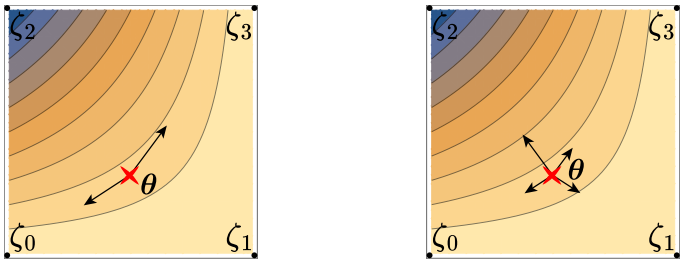
4.3.1 Dependence on the current distribution

When put into the perspective of solving a PB optimization problem, one of the main limitations of the methods presented in this section is their dependence on $p_{\boldsymbol{\theta}}(\cdot)$. To exemplify, we note that the REINFORCE expression

$$\frac{1}{n} \sum_{s=1}^{n} J(\mathbf{z}^{(s)}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}),$$

corresponds to a weighted average of multiple $J(\boldsymbol{\zeta}_h) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_h)$ terms, for $h \in \{0, \ldots, 2^d - 1\}$, where some of the summands are likely to have higher weights if $p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_h)$ is higher. In other words, we can rewrite it as:

$$\sum_{h=0}^{2^d-1} \left(\frac{n_{\boldsymbol{\zeta}_h}}{n} \right) J(\boldsymbol{\zeta}_h) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_h) = \sum_{h=0}^{2^d-1} \left(\frac{n_{\boldsymbol{\zeta}_h}}{n} \right) \left(\frac{1}{p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_h)} \right) J(\boldsymbol{\zeta}_h) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_h). \quad (4.18)$$



(a) Sampled (b) True

Figure 4.3: Contour plots of $J(\cdot)$ for $d = 2$. Arrows indicate the summands from (a) Equation (4.18), with $n = 2$ (b) Equation (4.19). If $p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_2)$ is low, the gradient may not point towards $\mathbf{z}^* = \boldsymbol{\zeta}_2$, such as in (a).

Where $n_{\boldsymbol{\zeta}_h}$ is the number of samples satisfying $\mathbf{z}^{(s)} = \boldsymbol{\zeta}_h$. In each summand, the term $\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_h)$ is the direction that increases the probability of $\boldsymbol{\zeta}_h$ being sampled again, while $J(\boldsymbol{\zeta}_h)$ scales that direction according to the cost function and $(1/p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_h))$ up-scales rare values. We can see the sum as multiple $\boldsymbol{\zeta}_h$ values attracting $\boldsymbol{\theta}$ towards themselves (or repelling if $J(\cdot) < 0$), as illustrated in Figure 4.3.

A big difference between MC methods and the ones using true gradients, such as a method using the vector field from Example 8, is that, in the true gradient case, the sum becomes:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)} [J(\mathbf{z})] &= \nabla_{\boldsymbol{\theta}} \sum_{h=0}^{2^d-1} p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_h) J(\boldsymbol{\zeta}_h) \\ &= \sum_{h=0}^{2^d-1} J(\boldsymbol{\zeta}_h) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_h). \end{aligned} \tag{4.19}$$

Notably, it includes all $\boldsymbol{\zeta}_h$ values, whereas a value that is unlikely to be sampled may not be present in Equation (4.18). Hence, while the true gradient vector field might indicate a smooth surface leading towards the minimizing $\mathbf{z}^*$, these gradients always account for the entirety of $\{0, 1\}^d$. Conversely, MC implementations may never sample $\mathbf{z}^*$ with low $p_{\boldsymbol{\theta}}(\cdot)$. Even in the rare occasion that they do, some optimizers normalize the gradient updates, diminishing the joint effect of $J(\boldsymbol{\zeta}_h)(1/p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_h))$ and making it less likely that $p_{\boldsymbol{\theta}}(\cdot)$ will get close to placing significant probability mass on $\mathbf{z}^*$.

Although one might consider using importance sampling to make the optimization more exploratory, a large d severely limits its applicability. To exemplify, in case the importance distribution is a factorized Bernoulli with parameters $\boldsymbol{\theta}_q$, the sampling estimate will be

$$\frac{1}{n}\sum_{s=1}^{n}\left(\frac{p_{\boldsymbol{\theta}}(\tilde{\mathbf{z}}^{(s)})}{p_{\boldsymbol{\theta}_q}(\tilde{\mathbf{z}}^{(s)})}\right) J(\tilde{\mathbf{z}}^{(s)})\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tilde{\mathbf{z}}^{(s)}) = \\ \frac{1}{n}\sum_{s=1}^{n}\left(\prod_{i=1}^{d}\frac{(\boldsymbol{\theta})_i^{\tilde{z}_i^{(s)}}(\overline{\boldsymbol{\theta}})_i^{1-\tilde{z}_i^{(s)}}}{(\boldsymbol{\theta}_q)_i^{\tilde{z}_i^{(s)}}(\overline{\boldsymbol{\theta}}_q)_i^{1-\tilde{z}_i^{(s)}}}\right) J(\tilde{\mathbf{z}}^{(s)})\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tilde{\mathbf{z}}^{(s)}).$$

The IS ratios will be large whenever a $\tilde{\mathbf{z}}^{(s)}$ that is more likely under $p_{\boldsymbol{\theta}}(\cdot)$ than under $p_{\boldsymbol{\theta}_q}(\cdot)$ is sampled. Since there are d per-sample coordinates where this can happen, it might be infeasible to avoid exploding gradients without limiting $\boldsymbol{\theta}_q$[8] or resorting to some biased approach, such as weighted importance sampling (Owen 2013, section 9.2).

Generally it is a rule of thumb to not use IS for large dimensions (Au and Beck 2003; B. Li *et al.* 2005). See Owen (2013, example 9.3) for an example using mean zero Gaussian distributions where slight changes in variance from the importance distribution result in massive variance increments for large d. ARMS manages to bypass this problem because of the condition that the marginals remain unchanged, leading to some cancellation. Nonetheless, since the search space is so large, it is infeasible to cover it entirely, causing the design of an adequate importance distribution to either be heuristic or to rely on domain knowledge.

4.3.2 Unwanted generalization

The problems outlined in the previous section revolved mostly around how hard it can be for the sampling to get close to minimizing values. In this section, we show that issues with the probabilistic formulation from Section 4.1.1 go beyond the stochasticity introduced by sampling. To understand why, we remind the reader that the problem involves finding the best solution among

[8] One possibility to control the variance introduced by IS is to use defensive importance sampling, where the importance distribution is designed to be a mixture between the nominal distribution and some desired alternative (Owen 2013, section 9.11).

2^d candidates and that, in general, the value of a single $J(\boldsymbol{\zeta}_h)$ does not imply anything about evaluations in the rest of the set.

A straightforward approach to solve this problem might be to store a logit vector with 2^d entries, one corresponding to each vertex, and increase or decrease them according to comparisons between multiple $J(\cdot)$ evaluations. Naturally, these logits could map to probabilities by the use of some function, a common choice being the softmax. That way, increasing the probability of some solution $\mathbf{z}$ would not affect the probabilities of unseen $\mathbf{z}'$ relative to each other.

Yet, because we cannot store such a large vector, we resorted to using multiple Bernoulli in this chapter. Although this choice can make gradient-based optimization possible, it also means that increasing the probability of some solution $\mathbf{z}$ will also increase $p_{\boldsymbol{\theta}}(\cdot)$ in points that are close to it. When we say close, we refer to the Hamming distance between both vectors, which corresponds to the number of dimensions in which they differ. In our particular case, the Hamming distance between $\mathbf{z}$ and $\mathbf{z}'$ is given by:

$$d_H(\mathbf{z}, \mathbf{z}') = \sum_{i=1}^{d} z_i(1 - z'_i) + (1 - z_i)z'_i.$$

In Example 9, we show a simple case where this is problematic

Example 9. *For $d = 2$, we take $J(\cdot)$ such that:*

$$\begin{bmatrix} J(\zeta_2) & J(\zeta_3) \\ J(\zeta_0) & J(\zeta_1) \end{bmatrix} = \begin{bmatrix} m & M \\ M & m \end{bmatrix}, \qquad \text{where } m < M.$$

Assume the update rule is:[9]

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \left(\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot)} [J(\boldsymbol{z})] \right) \Bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{t-1}}.$$

If we initialize $\boldsymbol{\theta}_0 = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}^\top$, we have that:

$$\left(\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot)} [J(\boldsymbol{z})] \right) \Bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0} = \mathbf{0}$$

and the optimization will be stuck at $\boldsymbol{\theta}_0$. In that case, the final solution will have 50% change of sampling M instead of m. Figure 4.4 shows the contour plot for this problem.

[9] In general, $\boldsymbol{\theta}_t$ would additionally have to be projected to $[0, 1]^d$ in case it falls outside of this region. In this section we assume that it will always stay there.

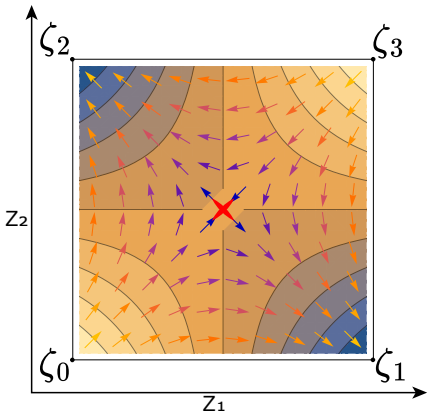


Figure 4.4: Illustration for Example 9. Blue regions correspond to lower $J(\cdot)$, arrows correspond to the gradient field.

Initializing $\boldsymbol{\theta}$ in the middle of the hypercube is a reasonable choice when nothing else is known about the optimal solution, since it is equidistant to all vertices. For this simple example, using MC instead of true gradients would actually help getting the optimization away from the saddle point. Nonetheless, it illustrates how different $\boldsymbol{\zeta}_h$ can interfere with each other. Theorem 5 shows another example of undesired behaviour due to this generalization.

Theorem 5. *For some arbitrary $\boldsymbol{z}^*$ in $\{0,1\}^d$, define $J : \{0,1\}^d \to \mathbb{R}$ as:*

$$J(\boldsymbol{z}) = \begin{cases} m & \text{if } \boldsymbol{z} = \boldsymbol{z}^* \\ M_0 - d_H(\boldsymbol{z}, \boldsymbol{z}^*)\dfrac{\Delta M}{d} & \text{otherwise} \end{cases},$$

where $M_0 \in \mathbb{R}$, $\Delta M \in \mathbb{R}_{>0}$ and $m = \min_{\boldsymbol{z}\in\{0,1\}^d} J(\boldsymbol{z})$ is unique. Particularly, this is only satisfied if:

$$m < M_0 - \Delta M,$$

where $J(\mathbf{1} - \boldsymbol{z}^*) = M_0 - \Delta M$ is the second lowest value. For any arbitrary dimension i and assuming $p_{\boldsymbol{\theta}}(\cdot)$ is a factorized Bernoulli with parameter $\boldsymbol{\theta}$, we have:

$$\left(-\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_{\boldsymbol{\theta}}(\cdot)} [J(\boldsymbol{z})]\right)_i \left(\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{z}^*)\right)_i \geq 0 \tag{4.20}$$

$$\iff \quad m \leq M_0 - \frac{\Delta M}{d \prod_{j\neq i} p_{\theta_j}(z_j^*)}.$$

Proof. *Section A.4 in the appendix.*

$\mathbf{z}$	$d_H(\mathbf{z}, \mathbf{z}^*)$	$J(\mathbf{z})$
$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^\top$	3	$M_0 - \Delta M$
$\begin{bmatrix}0 & 0 & 1\end{bmatrix}^\top$; $\begin{bmatrix}0 & 1 & 0\end{bmatrix}^\top$; $\begin{bmatrix}1 & 0 & 0\end{bmatrix}^\top$	2	$M_0 - 2\Delta M/3$
$\begin{bmatrix}0 & 1 & 1\end{bmatrix}^\top$; $\begin{bmatrix}1 & 1 & 0\end{bmatrix}^\top$; $\begin{bmatrix}1 & 0 & 1\end{bmatrix}^\top$	1	$M_0 - \Delta M/3$
$\begin{bmatrix}1 & 1 & 1\end{bmatrix}^\top$	0	m

Table 4.1: Example of $J(\cdot)$ following the conditions from Theorem 5 for $d = 3$ and $\mathbf{z}^* = \mathbf{1}$.

The LHS of Equation (4.20) corresponds to the i-th summand of the inner product between the direction used in gradient descent (i.e. $-\nabla_{\boldsymbol{\theta}}\mathbb{E}[J(\mathbf{z})]$) and the direction that increases the probability of $\mathbf{z}^*$ the most (i.e. $\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z}^*)$). Their sign being the same implies the (negative) gradient is pointing towards the optimal solution in that dimension. Table 4.1 illustrates one $J(\cdot)$ following the description from the theorem. The intuition behind $J(\cdot)$ is that, if not for $\mathbf{z}^*$, the optimal value would be $\mathbf{1}-\mathbf{z}^*$, the furthest point from the true solution. Additionally, the closer $\mathbf{z}$ is to $\mathbf{1}-\mathbf{z}^*$, the better $J(\mathbf{z})$ gets. To overcome the joint effect where most of the gradients (in the PB perspective of Equation (2.5)) evaluated on the set $\{0,1\}^d \setminus \mathbf{z}^*$ point away from $\mathbf{z}^*$, m must be much less than $M_0 - \Delta M$.

Corollary 6. *For the setting described in Theorem 5, define error ϵ such that $\max_{i\in\{1,\dots,d\}} p_{\theta_i}(z_i^*) = 1-\epsilon$ and assume ϵ is greater than zero.*[10] *Equation* (4.20) *is satisfied if and only if:*

$$\begin{aligned} m &\leq -\Theta\left(\frac{1}{d}\left(\frac{1}{1-\epsilon}\right)^d\right) \\ &= -\Theta\left(\frac{c^d}{d}\right), \qquad \text{for } c > 1. \end{aligned}$$

It is worth it to mention that m, M_0 and ΔM are pre-defined values and none of them needs to have any dependence on d for $J(\cdot)$ to be of the described form (e.g. m does not need to go to $-\infty$ as $d \to \infty$ to be the minimum). Moreover, $J(\cdot)$ will always be bounded to the interval $[m, M_0)$. What happens as d increases is that the differences between evaluations of neighbours (e.g.

[10] Similar results can be derived if some coordinates have error zero, but we opt to use these assumptions to simplify the exposition.

$J(\mathbf{z}) - J(\mathbf{z}') = \Delta M/d$) become smaller and multiple ($J(\mathbf{z}), J(\mathbf{z}')$) become closer together. Because the number of vertices increases exponentially, but $\Delta M/d$ only decreases linearly, relative contribution of the $2^d - 1$ points towards $\mathbf{1} - \mathbf{z}^*$ will increase at a large rate in this example.

From an optimization perspective, this means that, asymptotically, the solution will not converge to $\mathbf{z}^*$, even if $p_{\boldsymbol{\theta}}(\cdot)$ is initialized close to it. Once more, we point out that this is all considering true gradients, which correspond to either using MC with $n \to \infty$ or using 2^d evaluations per timestep. This number of evaluations of $J(\cdot)$ would be enough to solve the problem by simply searching among solutions, without even needing any method. Therefore, the probabilistic approach has a hidden assumption that good solutions must also be close to each other according to the Hamming distance, or, alternatively, the best solution must be low enough to compensate for the difference. Although we presented results on a global scale, it is likely that the same effect will occur locally in the optimization landscape. The assumption could very well be broken in large Neural Networks, given that they operate in complex ways.

Another point that becomes apparent after looking at this example is the dependence of these methods on the scale of the loss. The relative ordering of $J(\mathbf{z})$ remains the same, regardless of which m, M_0 and ΔM are chosen. Still, values of m that do not comply with the bound from Theorem 5 will cause the method to fail, which would not be the case had m been small enough. We finalize this section by citing the following result:

Theorem 7. *Let $\hat{\boldsymbol{\theta}}_{t,m}$ be the best solution after running stochastic gradient descent for t time steps on the probabilistic objective $\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot)}[J(\boldsymbol{z})]$ from m starting points with unbiased gradient estimators. Assume the sequence of step sizes, here denoted by $\{\alpha_t\}_{t=1}^{\infty}$, satisfies $0 < \sum_{t=1}^{\infty} \alpha_t^2 < \infty$ and $\sum_{t=1}^{\infty} \alpha_t = \infty$. Further, assume that $\theta_i = \sigma\left(\frac{r_i - 1/2}{\tau}\right)$ for all dimensions.*[11] *Let $\hat{\boldsymbol{z}}_{t,m} \sim p_{\hat{\boldsymbol{\theta}}_{t,m}}(\cdot)$. Then, as $n \to \infty$, $m \to \infty$ and $\tau \to 0$, we have that $\hat{\boldsymbol{z}}_{t,m} \to \boldsymbol{z}^*$ in probability.*

Proof Theorem 2 of Daulton *et al.* (2022) ■

[11] This reparametrization was used by the authors because the score function $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z})$ is not defined if $p_{\boldsymbol{\theta}}(\mathbf{z}) = 0$.

This theorem outlines convergence guarantees for MC methods presented in this chapter. Nevertheless, it assumes infinitely many initializations and time steps for some fixed size problem. Corollary 6 and the subsequent discussion, on the other hand, investigate behaviour as $d \to \infty$ for a constant number of initializations. In the overparametrized regime, we should have a d much larger than the number of possible parallel runs, so our assumption is likely more reflective of the neural network problems we will focus on.

4.4 Alternative parametrizations

Throughout this chapter, we focused on the case where:

$$p_{\boldsymbol{\theta}}(\mathbf{z}) = \prod_{i=1}^{d} p_{\theta_i}(z_i), \qquad \text{for } p_{\theta_i}(z_i) = \begin{cases} \theta_i & \text{if } z_i = 1 \\ 1 - \theta_i & \text{if } z_i = 0 \end{cases},$$

which we alternatively denoted as:

$$p_{\boldsymbol{\theta}}(\mathbf{z}) = \prod_{i=1}^{d} \theta_i^{z_i} \overline{\theta}_i^{\overline{z}_i}.$$

Note that, when updating $\boldsymbol{\theta}$, care must be taken so that it does not fall out of $[0, 1]^d$. It is common practice to, instead of updating $\boldsymbol{\theta}$ directly, reparametrize it by writing:

$$\boldsymbol{\theta} = \theta(\mathbf{r}), \qquad \text{for } \mathbf{r} \in \mathcal{R},$$

where $\theta(\cdot)$ is applied element-wise and usually chosen such that its range is $[0, 1]$. The most common choice for $\theta(\cdot)$ is the sigmoid function due to its simplicity.

Nevertheless, there are some results in the literature questioning the effectiveness of the sigmoid for this purpose. We note that it corresponds to the softmax function when we have only two outputs. Mei *et al.* (2020) establish, both theoretically and empirically, two problems that appear whenever optimizing an expectation with respect to the softmax: high sensitivity to the initialization ("softmax gravity well") and slow convergence ("softmax damping"). Y. Li and Ji (2020) use sigmoid gates as probability masks on NN pruning, where they point out the slow transition between ones and zeros, which

they then attempt to reduce by using fixed temperature parameters. Similarly, Serra *et al.* (2018) apply annealed sigmoid gates directly as NN masks in sequential task learning, and also point out that the low gradient magnitudes of the sigmoid (with $\tau = 1$) harmed performance, which led them to add a compensation to their annealing schedule.

Based on these observations, our experimental sections will also include the following alternative choices for $\theta(\cdot)$:

- Direct parametrization: where $\mathbf{r} = \boldsymbol{\theta}$, similarly to what was done previously in this chapter, with the caveat that each θ_i has to be clamped to $[0, 1]$ after being updated.[12]

- Sinusoid parametrization: we normalize a sinusoid to lie in the $[0, 1]$ interval. Qualitatively, its main distinction from the sigmoid is that high values of r_i oscillate between one and zero, rather than causing z_i to get arbitrarily close to deterministic.

- Escort: Mei *et al.* (2020) propose this is an alternative parametrization to the softmax to avoid the described problems. In this work, we simply use the version they propose in their paper, where, for a categorical random variable with two classes, $\theta(\cdot)$ is input two scalars (i.e. $(\mathbf{r})_i = \mathbf{r}_i \in \mathbb{R}^2$), instead of a single scalar as in sigmoid, where one of the logits from the corresponding softmax is fixed at zero. Following the authors, we use $P = 4$ in the experiments.

For all of the gradient estimators presented in this chapter, changing the parametrization simply amounts to changing the score function $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\cdot)$ for $\nabla_{\mathbf{r}} \log p(\cdot; \theta(\mathbf{r}))$ in all $\hat{\boldsymbol{g}}(\cdot)$ formulas. In our experiments, the methods will also require inverting given probabilities to initialize $\mathbf{r}$ to the desired values. Sometimes it is possible to invert the mapping in multiple ways, in which case we chose an arbitrary inverse map. Table 4.2 summarizes all estimators used, as well as their expressions, inverse maps and score functions.

[12] In practice we had to clamp it to $[\epsilon, 1 - \epsilon]$ instead, for some small ϵ. This avoided the gradient becoming zero prematurely.

Name	Domain $\mathbf{r}_i \in$	Expression $\theta(\mathbf{r}_i)$	Inverse map* $\mathbf{r}^{-1}(\theta_i)$	Score Function** $(\nabla_{\mathbf{r}} \log p(\mathbf{z}; \theta(\mathbf{r})))_i$
Cosine	$\mathbb{R}$	$\frac{1}{2}(1 - \cos(r_i))$	$-\arccos(2\theta_i - 1) + \pi$	$\frac{\cos(r_i)+(2z_i-1)}{\sin(r_i)}$
Direct	$\mathbb{R}$	r_i	θ_i	$\frac{z_i-\theta(r_i)}{\theta(r_i)(1-\theta(r_i))}$
Sigmoid	$\mathbb{R}$	$\frac{1}{1+\mathrm{e}^{-r_i}}$	$\log\theta_i - \log(1-\theta_i)$	$z_i - \theta(r_i)$
Escort	$\mathbb{R}^2$	$\frac{\lvert(\mathbf{r}_i)_1\rvert^P}{\lvert(\mathbf{r}_i)_1\rvert^P+\lvert(\mathbf{r}_i)_2\rvert^P}$	$\begin{bmatrix}\left(\frac{\theta_i}{1-\theta_i}\right)^{\frac{1}{P}} & 1\end{bmatrix}^\top$	$\begin{bmatrix}-\frac{(\theta(r_i)-z_i)P}{(\mathbf{r}_i)_1} & \frac{(\theta(r_i)-z_i)P}{(\mathbf{r}_i)_2}\end{bmatrix}^\top$

Table 4.2: Summary of alternative parametrizations. (*) Inverse map assumes $\theta_i \notin \{0, 1\}$. (**) If $\theta(\mathbf{r}_i) \in \{0, 1\}$, we use zero instead of these formulas.

Chapter 5

Alternative stochastic gradient approaches

Chapter 3 discussed CP approaches where a smooth function is iteratively annealed and minimized until it becomes close to the desired discrete function, whereas Chapter 4 discussed MC methods, which store a parametrized probability distribution that tracks regions with seemingly better performances using a gradient-based procedure. These gradients are essentially comparisons between evaluations at the corners. There, we mentioned the reparametrization trick and explained why it is not applicable here. Nonetheless, due to the aforementioned successes of pathwise gradient estimation in other contexts, the belief that incorporating them in PB optimization should be beneficial has led to attempts to do so. In this section, we briefly mention some of these attempts, while emphasizing that they are not the focus of this study. Intuitively, they correspond to hybrid versions which use both the gradients, like CP approaches, but also rely on sampling and comparing evaluations on $\{0,1\}^d$ like MC methods.

5.1 Overview

One of the simplest and most widely used ways of combining MC methods and gradient information is by using straight-through estimation. Originally introduced by Bengio *et al.* (2013) as a way of adding stochastic units to

Neural Networks, the forward pass consists of a sampling operation,[1] but the backward pass treats the sampling operation as if it had been the identity. Specifically, they propose to use:

$$\text{Forward pass:} \quad \mathbb{E}[J(\mathbf{z})] \approx J(\mathbf{z}) \qquad \text{For } \mathbf{z} \sim \prod_{i=1}^{d} \text{Ber}[\sigma(r_i)]$$

$$\text{Backward pass:} \ \frac{\partial \mathbb{E}[J(\mathbf{z})]}{\partial r_i} \approx \frac{\partial J(\mathbf{z})}{\partial z_i}.$$

Note that, in addition to treating the sampling operation as if it had been an identity, the above steps ignore the derivative of the sigmoid with respect to r_i. Bengio *et al.* (2013) claimed that including the additional derivative harmed performance. Despite this estimator being biased, many works have used it throughout the years, mainly motivated by its empirical performance rather than analytical results (Bethge *et al.* 2019; Bulat, Martinez, *et al.* 2020; Bulat, Tzimiropoulos, *et al.* 2019; Martinez *et al.* 2020; Srinivas, Subramanya, *et al.* 2017; H. Zhou *et al.* 2019).

More recently, Shekhovtsov and Yanush (2021) presented theoretical reasoning behind straight-through estimation considering a more general framework that includes the method above. Particularly, ST estimation corresponds roughly to approximating the PB derivative of Equation (2.5) by using a first order Taylor expansion instead. Additionally, the paper proves that, for dimension i, if the expected $f_i(\mathbf{z}) = \partial J(\mathbf{z})/\partial z_i$ has absolute value larger than the Lipschitz constant of $f_i(\mathbf{z})$, the corresponding (negative) straight-through gradient will be a descent direction.

Another very popular way of combining gradients and sampling in PB optimization is via the Gumbel-softmax estimator (Jang *et al.* 2016; Maddison, Mnih, *et al.* 2016), a technique that is inspired by the Gumbel max trick. In general, sampling from a categorical distribution with probabilities $(\theta_c)_{c=1}^{C}$ can be equivalently done with this trick as follows:

1. Sample $u_c \sim U[0, 1]$.

2. Transform it using inverse transform sampling $T_c = \log \theta_c - \log(-\log u_c)$.

[1] Some works simplify this even further by using a threshold instead of sampling.

3. Output $\arg\max_{c=\{1,\dots,C\}} T_c$ (sometimes in one-hot vector form).

Samples from the Gumbel-softmax distribution, also called Concrete distribution, are obtained by changing the last step for $\exp(T_c/\tau)/\sum \exp(T_{c'}/\tau)$. Naturally, relaxing the estimation of an expectation with respect to a categorical distribution by instead using the Gumbel-softmax results in a biased estimator. On the other hand, this estimator can be promptly combined with the reparametrization trick. Furthermore, this distribution can be annealed towards the categorical as τ goes to zero.[2] Some methods use this biased approach directly (Louizos *et al.* 2017; M. B. Paulus *et al.* 2020; X. Zhou *et al.* 2021). Maddison, Tarlow, *et al.* (2014) extend the Gumbel-softmax to continuous random variables and M. Paulus *et al.* (2020) extend it to more general combinatorial spaces.

Some unbiased approaches also incorporate the Gumbel-softmax to the design of control variates. When computing the estimator following Equation (4.12), $h(\mathbf{z})$ is taken to be $J(\mathbf{z}_G)$, where $\mathbf{z}_G$ is dependent on $\mathbf{z}$, but is ultimately sampled from the Gumbel-softmax. The correction is often done using a reparametrization-based estimator, instead of using closed-form $\boldsymbol{g}_h$. Popular methods approaching the problem this way are REBAR (Tucker *et al.* 2017) and RELAX (Grathwohl *et al.* 2017).

Other works use Taylor expansions as control variates for unbiased estimation, therefore also relying on first-order derivatives, such as Muprop (Gu *et al.* 2015) and the control-variate used in the experiments from Mohamed *et al.* (2020), referred to as delta method. More recently Titsias and Shi (2022) propose combining a two-level application of LOORF and Taylor expansions, which was later improved by Shi *et al.* (2022) by also incorporating Stein operators to the two-level estimator, but ultimately also relying on inclusion of $\partial J(\mathbf{z})/\partial z_i$ in the control variates.

[2] In fact, the same logic used to motivate CP methods could also be used with Gumbel-softmax. We choose to focus on the deterministic algorithms in Chapter 3 due to their more pronounced recent successes as well as to better separate sampling from gradients as two different sources of top-down information.

Some of these methods have known pitfalls. Andriyash *et al.* (2018) point out that the raw Gumbel-sotmax estimators are highly dependent on tuning the temperature and that the apparent improvement obtained by such tuning can be replicated in other methods via simple entropy regularization. Additionally, Tucker *et al.* (2017) show that these same Gumbel estimators fail in a very simple one dimensional problem where $J(z) = (z - C)^2$ and C is chosen to be very close to 0.5. We also verified empirically that the version of straight-through estimation presented above fails in this same problem setting.

Comparisons between unbiased hybrid methods and MC methods that do not use $\nabla_{\mathbf{z}} J(\mathbf{z})$ (i.e. methods from Chapter 4) show mixed results. In a non-exhaustive analysis of recent works, we found five papers reporting mostly superior results of variants of ARMS[3] while three reported superior performances of RELAX/REBAR.[4] We mention that both REBAR and RELAX compute three forward passes and two backward passes for each sample $s = \{1, \ldots, n\}$. Perhaps this computation would have been better used with larger n instead. In more general comparisons between pathwise gradient estimators and score function estimators, Mohamed *et al.* (2020) noted that the later seems to benefit more from larger n. In our experiments, we will occasionally include some of the hybrid methods from this chapter.

[3]Dimitriev and M. Zhou (2021), Dong *et al.* (2020), Shi *et al.* (2022), Yin, Yue, *et al.* (2019), and Yin and M. Zhou (2018).

[4]Andriyash *et al.* (2018), Dong *et al.* (2021), and Lorberbom *et al.* (2019).

Chapter 6

Smaller-scale experiments

Now that we understand the ideas behind CP and MC methods, the next step is to see how well the intuitions developed in the previous chapters generalize to more practical settings. By first scaling experiments down to smaller d, we can compute closed form expressions for expectations and variances for the MC methods. Furthermore, non-overparametrized settings can help us understand how reliant the described CP methods are on large neural networks. Accordingly, we start with two problems where $\mathbf{z}$ is directly input to arbitrary $J(\cdot)$, for $d \leq 10$. Then, we scale up by moving to a regression problem, where $d = 8,050$ and the multiple z_i act as element-wise masks to the weights of a fixed backbone neural network.

6.1 Microworld

In this section, we restrict ourselves to $d \in \{4, 10\}$ and $n \in \{1, 4, 10\}$, as well as using true gradients, which corresponds to $n \to \infty$. Our experiments aim at first comparing the different estimators from Section 4.2, then comparing the different parametrizations proposed in Section 4.4 and finally comparing MC and CP methods, as well as comparing both against some of the methods from Chapter 5. We always initialize $\boldsymbol{\theta}_0 = [0.5, \ldots, 0.5]^\top$ in the experiments and optimize with SGD.

6.1.1 Benchmarks

Most MC gradient estimation papers start by comparing estimators in problems of the form $J(z) = (z - C)^2$, for C close to 0.5 and then moving to larger ones involving discrete Variational Autoencoders, usually combined with image datasets. The former setting serves to eliminate confounding factors present in the later, while also allowing computation of closed-form expressions. Still, we believe it might be overly simplistic and as a result not bring to light the differences between the estimators. For example, a simple second order Taylor expansion is enough to completely reconstruct the quadratic $J(\cdot)$ everywhere.

On the other hand, examples from Sections 3.3 and 4.3 were adversarial, designed specifically to highlight some of the main flaws from MC and CP methods. We propose new benchmarks that, while compatible with small scale settings, also require some additional search, either because of the presence of local minima or because $\nabla_{\mathbf{z}} J(\mathbf{z})$ might not be too informative of $J(\cdot)$ away from the current $\mathbf{z}$.

- **ExponentialTabularLoss:** we first sample 2^d values, where $E_h \overset{iid}{\sim} \text{Exp}[1.5]$ and $h \in \{0, \dots, 2^d - 1\}$. Then, we calculate the costs by normalizing $\{E_h\}_{h=0}^{2^d-1}$ such as to map $\max_h E_h$ to -1 and $\min_h E_h$ to 1. Because of the exponential distribution, most $\mathbf{z}$ points will have $J(\cdot)$ closer to 1, with a few rare exceptions closer to -1, meaning the solutions will be harder to find. As sampling is agnostic to the Hamming distance, $d_H(\mathbf{z}, \mathbf{z}^*)$ will not be indicative of anything about $J(\cdot)$ in $\mathbf{z}' \notin \{\mathbf{z}, \mathbf{z}^*\}$. In fact, sampling might result in multiple local solutions in different parts of $\{0, 1\}^d$. Importantly, this benchmark is not differentiable, so we will only evaluate MC methods using it.

- **NNLoss:** we input $\mathbf{z}$ directly to a fixed neural network with a scalar output corresponding to the cost. In addition to the input and output layers, this network has 9 fully-connected hidden layers with 20 neurons each. Normalization and LeakyRelu follow the respective linear operations of all but the output layer, which simply maps to a scalar that is then only normalized.

We initialize weights to be either 1 or -1 with 50% chance and preprocess the input $\mathbf{z}$ by mapping it to the $[-1, 1]^d$ range. The normalization uses one dimensional "batch norm" layers (Ioffe and Szegedy 2015) without affine parameters. One should run these layers in evaluation mode only, otherwise the output $J(\mathbf{z})$ will depend on the whole input batch instead of only on $\mathbf{z}$. Before fixing moving normalization statistics, we initialize them by running some forward passes on random uniform points from $[0, 1]^d$.

In both cases, for any fixed d, we initialize the loss only once and reuse it across methods and runs from the same method.

6.1.2 Estimators

Experiments in this section use only sigmoid parametrization. We start by comparing the variances of different MC estimators. As mentioned when we introduced control variates in Section 4.2, they will only reduce $\mathrm{Var}\left[(\boldsymbol{g}(\cdot))_i\right]$ if $(J(\mathbf{z}) - h(\mathbf{z}))\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})/\partial\theta_i$ has smaller variance than $J(\mathbf{z})\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})/\partial\theta_i$. Analytical guarantees of variance reduction only exist in limited contexts. Richter *et al.* (2020), for example, show that LOORF can have lower variance than REINFORCE, but enumerate some conditions in their proof. Similarly, Dimitriev and M. Zhou (2021) prove that the ARMS gradient will have lower variance than LOORF for negatively correlated z and z', given that $d = 1$ and $n = 2$. This however, does not automatically imply that the same is true for general n and d.[1]

With that in mind, we run a single trajectory starting from $\boldsymbol{\theta}_0$ in the middle of the hypercube and then following the true gradient of the expected cost for $10,000$ steps. This trajectory serves as a guide, providing $\boldsymbol{\theta}_t$ for different timesteps. For each of them, we calculate the variances for all estimators and compare their across-dimension sums. We consider two settings: $d = 4$ with $n = 4$ and $d = 10$ with $n = 10$. In the first, we compute the variances in closed

[1] Dimitriev and M. Zhou (2021) show that, for general $n > 2$ and $d = 1$, both LOORF and ARMS correspond to an average of their respective $\hat{\boldsymbol{g}}((z, z'); \theta)$ (i.e. $n = 2$ estimates). Still, the variance for $n > 2$ also has to account for the correlation between these summands.

form after leveraging some combinatorial analysis, whereas in the second we estimate them by using $10,000$ per-iteration evaluations of $\hat{\boldsymbol{g}}(\cdot)$. More details can be found in Section B.1.1. Importantly, computing ARMS variance in closed-form requires knowing the IS weights, which we derive algebraically.

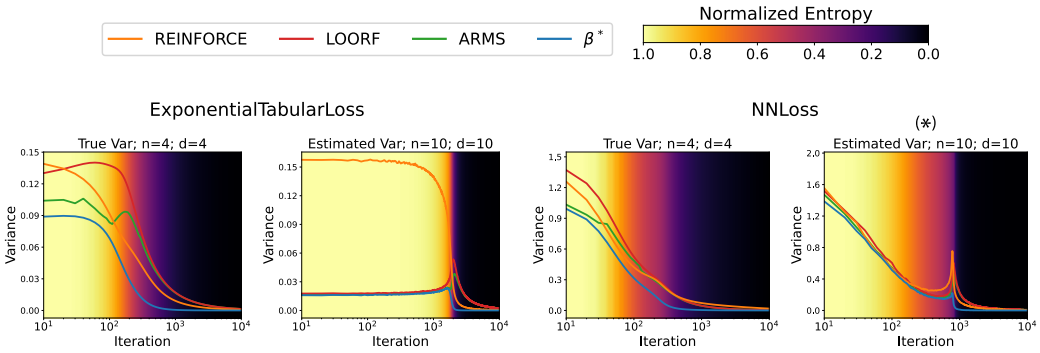


Figure 6.1: Comparing estimator variances following a fixed trajectory on Microworld domains. The model was not able to solve the setting marked with (*).

Figure 6.1 shows the results for both settings and benchmarks, considering only learning rate 0.1. We can see that REINFORCE indeed has lower variance than ARMS or LOORF in some cases. By overlaying the plots with the entropy, which is shared by all methods, we see that REINFORCE tend to perform better when $p_{\boldsymbol{\theta}}(\cdot)$ is almost deterministic.

Despite analytical proof of ARMS superior variance reduction being restricted to $n = 2$ and $d = 1$, here it indeed seems to perform better than LOORF, even when using closed-form results instead of estimates. As expected, β^* always had the lowest variance, indicating perhaps the lower bound achievable by using control variates without further problem-specific information.

After analyzing the variances, the next question in how that impacts the solution of the PB optimization problems on self-generated trajectories. Since the setting with $d = 4$ is not too challenging and all methods perform similarly well on it, we restrict ourselves to $d = 10$ and compare the estimators for varying n. Figure 6.2 shows the results averaged across 100 seeds.

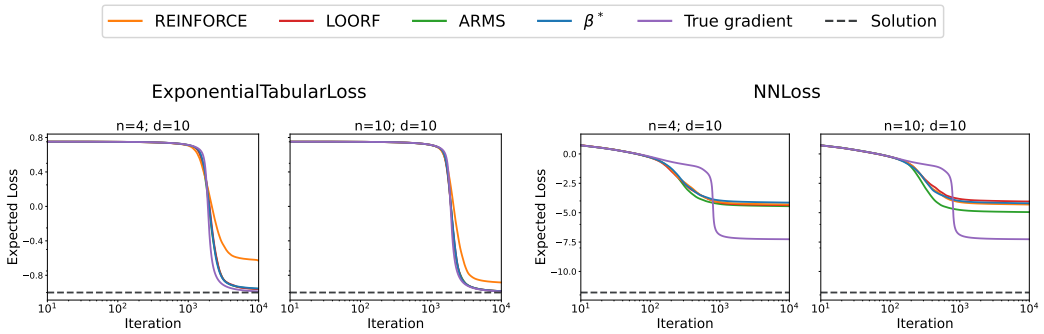


Figure 6.2: Comparing estimators on Microworld domains.

REINFORCE now performs clearly worse than the other methods. The contrast to its previous performance is reminiscent of what happens in imitation learning for sequential decision-making problems. There, an agent is trained to replicate expert behaviour on supplied trajectories, often via supervised learning. Even if training is seemingly successful, deployment can be catastrophic: the stochasticity of the problem often leads the agent to unseen regions of the state-space. There, the learned behaviour can be useless and take the agent even further from the states it has seen before (Ross *et al.* 2011). In Figure 6.2, the higher initial variance of REINFORCE, when the entropy was also high, was likely responsible for the model ending up far from where it would have gone had it followed the true gradient.

For the other estimators, variance reduction relative to each other did not seem to matter much. Despite using the optimal control variate, β^* failed to perform better than LOORF and ARMS in both problems. On NNLoss, even the model using true gradients failed to reach the correct solution, although eliminating the variance still led it to a better local minimum than the MC estimators. We can explain this observation using the generalization discussion from Section 4.3.2, where interactions between $\boldsymbol{\zeta}_h$ impede the model from reaching the correct solution.

6.1.3 Parametrizations

In this section, we run the same experiments as before, but keep the estimator fixed and vary the parametrization instead. Namely, we use REINFORCE for

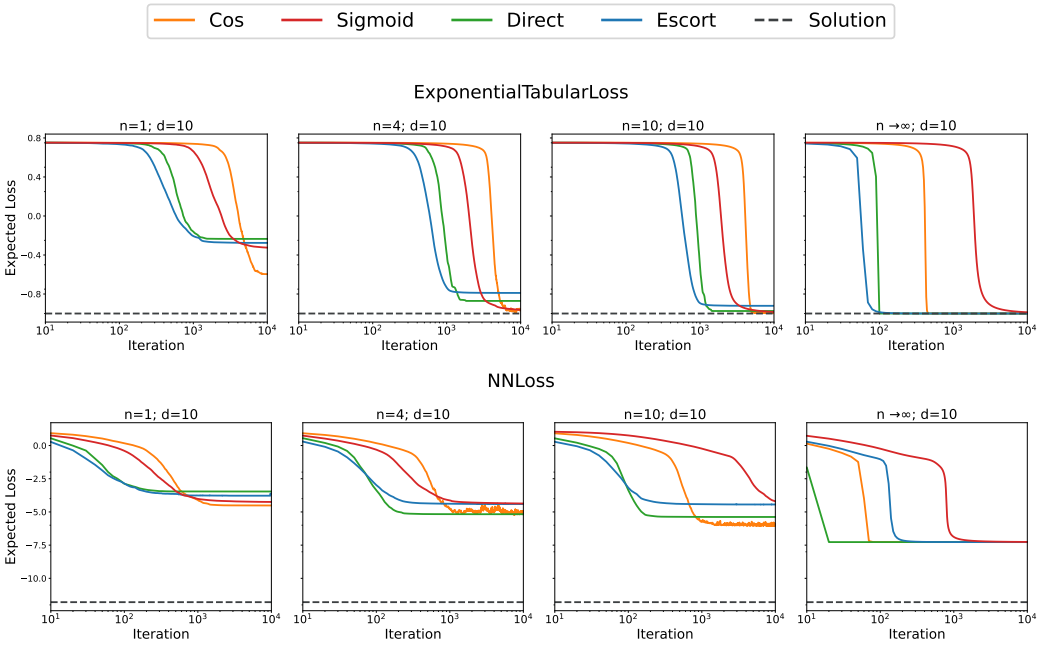


Figure 6.3: Comparing parametrizations on Microworld domains.

$n = 1$ and LOORF for $n \in \{4, 10\}$. We sweep learning rates in $\{0.1, 0.01\}$ and select the best one based on the final loss. Figure 6.3 shows the results averaged over 100 seeds.

By inspecting the figure, we note that the parametrizations can roughly be categorized in two groups: direct and escort; sigmoid and cosine. The first converges much faster, but often to worse values, whereas the second converges slower, but to better final values. As more samples are used, the faster group seems to maintain superior convergence speed while arriving at better final solutions

Figure 6.4 depicts NNLoss scatter plots for this same experiment, where each circle corresponds to the final per-seed loss. Losses are more spread out for low n, but converge towards -8 as samples increase. For $n = 1$, the higher variance caused the model to be more exploratory and even find $\mathbf{z}^*$ in some runs, indicated with arrows. However, these ouliers stopped appearing as variance was reduced with increasing n and the generalization problem stopped the model from ever reaching $\mathbf{z}^*$.

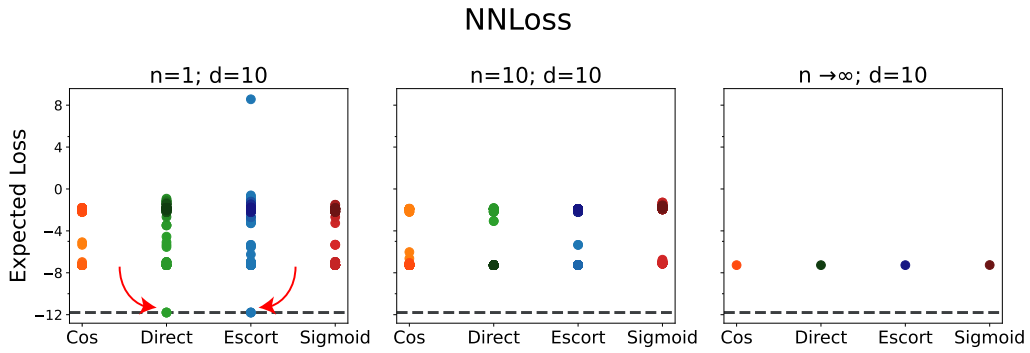


Figure 6.4: Comparing parametrizations on Microworld domains. Alternative visualization of NNLoss from Figure 6.3 showing the final loss for each seed.

6.1.4 Approaches

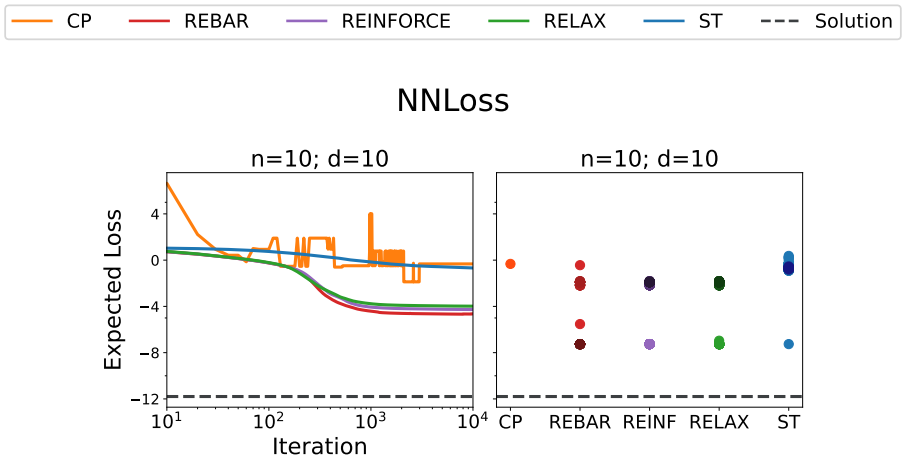


Figure 6.5: Comparing approaches on Microworld domains.

Finally, we compare MC and CP in Figure 6.5. We also include hybrid approaches from Chapter 5, where ST is a biased method and both REBAR and RELAX are unbiased, but use $\nabla_{\mathbf{z}} J(\mathbf{z})$ in their control variates. Similarly to the above, we sweep learning rates in $\{0.1, 0.01\}$ and use 100 seeds on non-deterministic algorithms. Additional details about hyperparameters particular to each method are deferred to Section B.1.2. We parametrize REINFORCE with sigmoid. Differently from the previous experiments, selecting hyperparameters based on the lowest final loss, led to poor choices. Settings often had similar final losses and tie-breaking sometimes chose models that converged

much slower. Therefore, we select by comparing the average value from iteration 10 onward instead.

By inspecting the plots, we can see that CP has a "jagged" format, likely due to sudden temperature changes, causing the model to reach a new solution quickly and stay there until τ changes again. Overall, it did not seem that $\nabla_{\mathbf{z}} J(\mathbf{z})$ was helpful here: ST and CP, which arguably rely on it the most, were stuck in poor local solutions, whereas the methods that combine MC and gradients had similar performances to merely using REINFORCE. Alternative parametrizations and estimators already outperformed REINFORCE for this same benchmark in the previous sections.

We can again put these results in light of previous discussions. In particular, Section 3.3 mentioned that, when applied to PB optimization in the described manner, numerical continuation methods tend to extrapolate local information from inside the hypercube to differences between its corners. The same is true for the other methods relying on the gradient of $J(\cdot)$. This extrapolation is only heuristic and appears to be inaccurate in this benchmark.

6.2 Neural network regression

The goal of this section is to perform comparisons similar to the previous one, but on a larger scale, although not as large as in image classification tasks. This time, it is not feasible to compute expectations in closed form, so we restrict ourselves to $n \in \{2, 10, 100, 1000\}$ and report losses with respect to a sampled $\mathbf{z}$. Just as before, we initialize $\boldsymbol{\theta}_0 = [0.5, \ldots, 0.5]^\top$, but now we optimize parameters with RMSprop. Results are averaged over 10 seeds and d is equal to $8,050$. We select hyperparameters by comparing the average loss in the second half of training.

6.2.1 Benchmark

For these experiments, we use a single benchmark, called MaskedNNRegression, consisting of two fixed neural networks: a backbone NN and a target NN. $\mathbf{z}$ is a binary mask applied weight-wise to the backbone NN and the goal is to

learn the input-output mapping from the target NN while only changing the masks. Furthermore, the target has higher capacity than the backbone and a more complexity-inducing initialization, so these experiments are not in the overparametrized regime yet.

First, we sample some random input data uniformly from $[-1, 1]^{10}$ and map each 10-dimensional vector to a scalar output using the target network. We fix this data and split it into training and validation datasets. Importantly, we sample datasets, backbone and target networks only once and re-use them across different random seeds. The loss is the expected (absolute) difference between target and output of the masked backbone network. More details about the architectures used and experimental setup are in Section B.2

6.2.2 Estimators

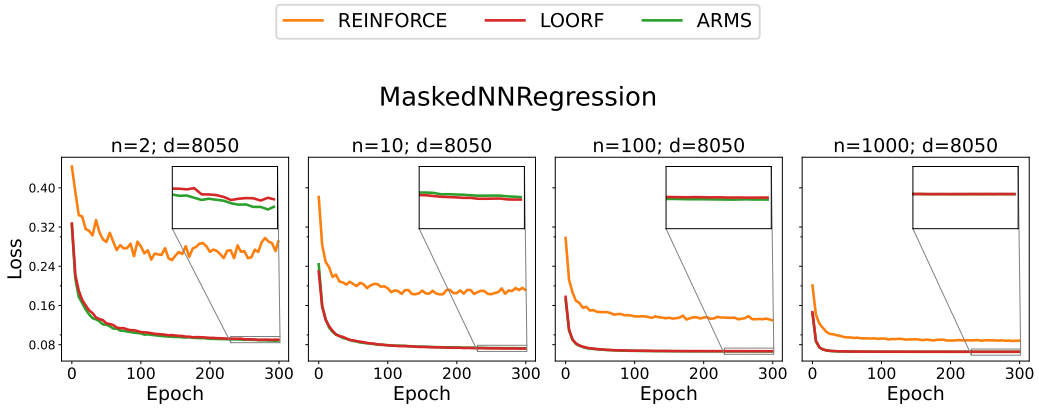


Figure 6.6: Comparing estimators on masked NN regression.

As we show in Figure 6.6 (for learning rate = 0.1 and sigmoid parametrization), REINFORCE still performs worse than ARMS and LOORF, but this time the difference is much higher. Recall that the entropy of a random variable that can assume 2^d values is at most $d \log 2$, which is also the initial entropy in these experiments. Considering the previous REINFORCE results, where entropy, variance and loss were positively correlated, this more entropic initial model should indeed lead to relative worsening of its performance.

Despite that, lower entropy does not necessarily imply lower variance. The former corresponds roughly to the spread of a random variable with respect to the set of values it can assume, whereas the later concept is tied to a scalar function, measuring how much it changes as a consequence of the changes from this variable.

ARMS and LOORF again performed similarly. The ratio between n and samples needed to solve the PB optimization (i.e. $n/2^d$) will get even worse in the later experiments. Perhaps ARMS being superior for $n = 2$ in Figure 6.6 and also having lower variance in Figure 6.1 is indicative that it will perform slightly better than LOORF in the larger experiments.

6.2.3 Parametrizations

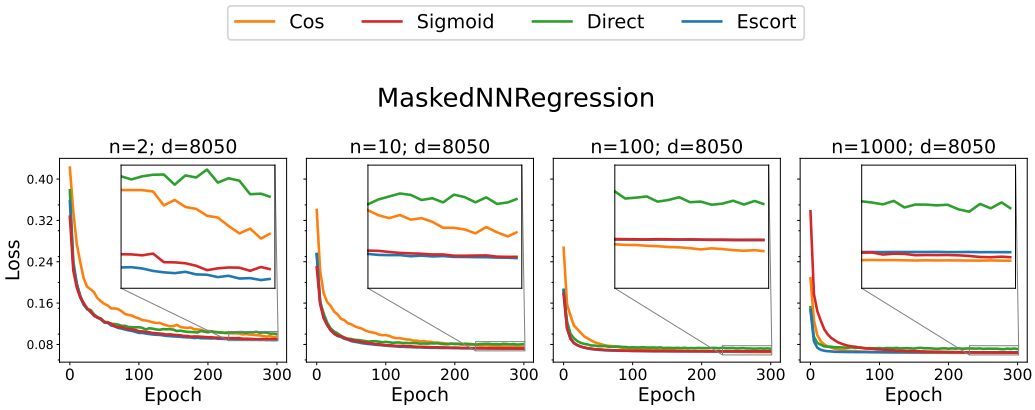


Figure 6.7: Comparing parametrizations on masked NN regression.

Figure 6.7 depicts the results of comparing parametrizations, where we sweep learning rates in $\{0.1, 0.01\}$ and use LOORF (approach-specific parameters are as before). Similarly to the previous experiments, direct parametrization is quick to converge, but to sub-optimal values. This effect is even more pronounced now. Cosine parametrization is still slow, but, in contrast with the previous section, its final values are worse than those of the other estimators. Escort and sigmoid were the best performing parametrizations. The first maintained its convergence speed, but reached better final values, while the later got faster. On the larger experiments, we will only use sigmoid and escort.

6.2.4 Approaches

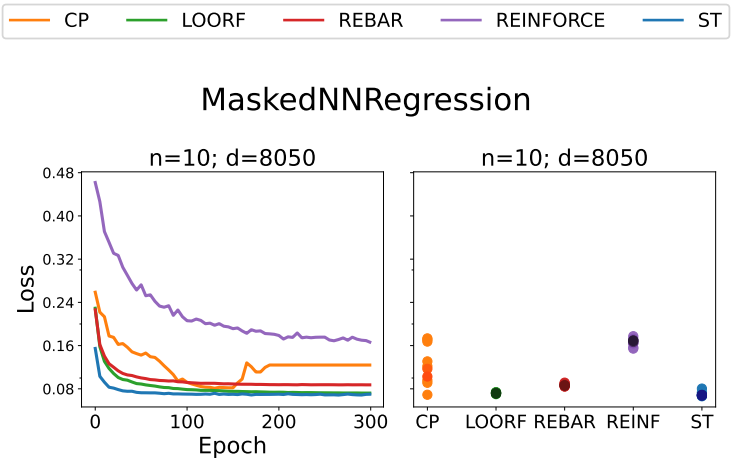


Figure 6.8: Comparing approaches on masked NN regression.

As seen in Figure 6.8 (for learning rates in $\{0.1, 0.01\}$, $n = 10$ for the non-deterministic methods and sigmoid parametrization), results from this experiment were different from those of Section 6.1.4. The combination of larger d and neural network structure is somewhat compatible with approaches reliant on $\nabla_{\mathbf{z}} J(\mathbf{z})$. Differently from last time, simply using REINFORCE was much worse than combining MC and gradients, such as in REBAR and ST. The best performances were from ST and LOORF, where the first converged faster, but the later had more runs converging to the lower loss values (see the scatter plot).

CP often seemed to be getting close to the best performing methods, but, apparently, the sudden temperature changes during the last half of training led $\boldsymbol{\theta}$ away from previous well-performing values. Combined with the extrapolation problem described before, this seems to have caused CP to perform relatively poorly. It remains to be seen if the performance gains when using $\nabla_{\mathbf{z}} J(\mathbf{z})$ relative to MC will become even more prominent in the larger settings.

Chapter 7

Pruning

The goal of this chapter is to compare approaches based on Monte-Carlo gradient estimation (MC), continuation path (CP), as well as magnitude-based methods (MP) on pruning benchmarks. Even though there are papers already comparing CP and MP, MC is not well represented in the literature. Namely, Monte-Carlo gradient estimation is an active area of research, lots of new approaches to reduce variance during estimation and improve convergence to the true gradient have been proposed in the last few years, as described in Chapter 4. Studies comparing against these approaches for pruning, however, are scarce, and mostly focus on straight-through estimators from Chapter 5. One can argue that MC approaches have more theoretical justification than CP and MP, as we saw in the previous sections. Additionally, MC approaches were usually the most successful on the experiments from Chapter 6. With that in mind, we attempt to bridge the gap between the Monte-Carlo gradient estimation literature and the pruning literature.

Section 7.1 starts by introducing and motivating the problem, as well as listing some common approaches. Then, Section 7.2 describes the algorithms used in the experiments. Finally Section 7.3 describes the experimental setup and presents the results.

7.1 Overview

This section starts by presenting the motivation for studying pruning, then we categorize it according to the pruned structure and finally we give a broad overview of the existing methods.

7.1.1 Problem Motivation

Deep learning methods have shown great successes in many areas, including vision, natural language processing and recommendation systems. Conveniently, it automates the discovery of data representations in domains that previously required human expertise. In computer vision, for example, modern approaches based on CNN's (Krizhevsky *et al.* 2017) and R-CNN (Ren *et al.* 2015), specially when used in conjunction with transfer learning, have somewhat replaced older representations such as SIFT (Lowe 1999), HOG (Dalal and Triggs 2005) and Bag-of-Visual-Words (Csurka *et al.* 2004). Yet, even the design of deep neural networks requires humans to make arbitrary decisions. Deciding the exact model size and capacity can be particularly challenging, justifying the search for automated approaches to do so. One such approach is to start with a dense, overparametrized model and sparsify it by pruning some of its connections, which can be either neurons or weights.

The idea of pruning is not new, with early work in the 80's and 90's studying the generalization and interpretability of pruning smaller models (see Reed (1993) for a survey of these older works). Nonetheless, interest in funding AI companies greatly decreased in the late 80's, a period some authors call the "AI winter" (Russell 2021, section 1.3.4). This was mostly due to failure to satisfy expectations on more complex domains, with systems breaking down easily in the face of uncertainty and unable to learn from experience. The "AI winter" lasted until about 2012, when Deep Learning frameworks started better leveraging the available compute resources, for example, by incorporating graphics processing units on their workflows.

Since then, larger models became increasingly common. Inception-V3 (Szegedy *et al.* 2016), has 5.7 billion arithmetic operations and 27 million parameters,

whereas GPT-3 (Brown *et al.* 2020) requires 175 billion parameters. Some works even have models requiring trillions of parameters (Fedus *et al.* 2022; Lepikhin *et al.* 2020). This phenomenon was one of the main motivators for new research on pruning, since many of these pruned models can benefit from faster inference times and smaller memory footprints, specially on deployment, as we will discuss in Section 7.1.2. Recent pruning strategies can sometimes lead to 100x reduction of network size (Hoefler *et al.* 2021). To illustrate the popularity of the field, on their recent survey, Hoefler *et al.* (2021) plot the number of published papers on the topic per year: since 2012, there were more than 266, the curve indicates growth close to exponential.

Pruning of neural networks has also some theoretical justifications. Malach *et al.* (2020) prove that, with high probability, any network can be approximated with arbitrary accuracy by pruning a polynomially larger random network (without even changing the weights). Later, Orseau *et al.* (2020) and Pensia *et al.* (2020) showed that a logarithmically larger network suffices. In addition to that, training overparametrized networks also benefits from good convergence. Brutzkus *et al.* (2017) and Du *et al.* (2018) prove that it leads to a convexity-like property, Allen-Zhu *et al.* (2019) and Neyshabur *et al.* (2018) later supported these results. Then, depending on the level of sparsity desired,[1] pruning can act as a form of regularization for these models, improving generalization and robustness. In fact, sparse models sometimes even outperform their fully dense counterparts (Frankle and Carbin 2018; Lee *et al.* 2019; Pensia *et al.* 2020).

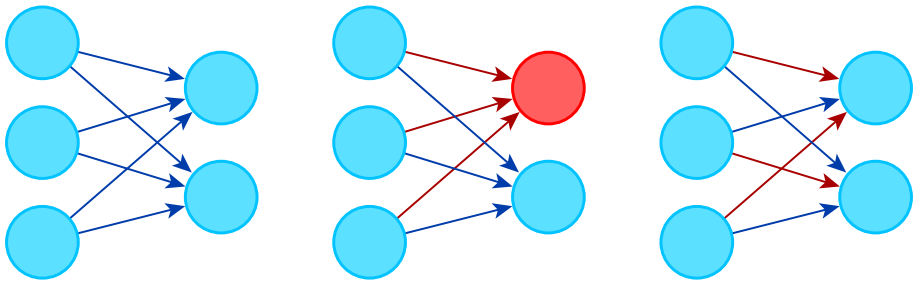
With all that in mind, we choose pruning as a benchmark to compare the different PB optimization approaches. Particularly, for this section, we aim at understanding how well the MC gradient estimation apparent advantage on the Microworld problems still holds on the larger benchmarks. We believe such knowledge can help guide future research when incorporating these methods to the pruning literature. Analyzing its pros and cons can help either improve

[1]For moderate levels of sparsity, there seems to be an increase in test accuracy, often attributed to the reduction on learned noise. At high levels, however, we start to see the performance decrease. See Hoefler *et al.* (2021, section 2.1) for more details.

current MC methods or develop alternative stochastic formulations achieving high performance.

7.1.2 Structured and unstructured sparsity

It should be mentioned that current research often classifies pruning methods as either structured or unstructured. We illustrate both types in Figure 7.1 for a fully connected layer. The main difference is that, for unstructured pruning, weight or neuron removal happens indiscriminately, whereas entire structures are removed at once for structured pruning. For the later, it is common to prune whole matrix columns of fully-connected layers and whole feature maps of convolutional layers. Pruning these structures often reduces memory consumption and inference speed immediately in common frameworks. When a pruning a feature map, for example, one can simply recreate the convolutional filter as a smaller tensor including only the depths that do not correspond to that map. Structured sparsity, nonetheless, is not limited to these structures and recent works have studied pruning contiguous blocks of weights, as well as the development of deep learning accelerators specifically tailored for sparse networks. For an overview of such research, refer to Hoefler *et al.* (2021, chapter 7).



(a) Dense network (b) Structured pruning (c) Unstructured pruning

Figure 7.1: Different ways of pruning. Pruned weights or neurons are repesented in red.

Unstructured sparsity, on the other hand, will not yield any speedups if implemented naively, as the common deep learning frameworks have limited support for weights pruned in a completely arbitrary manner, often performing computations and storing data as if the network was dense. For this reason,

one can see unstructured pruning as an upper bound on the possible sparsity or as a way to advance pruning research more easily by focusing more on the algorithmic aspects and less on the deployment limitations. Nevertheless, frameworks allowing lower level implementations can avoid storing these sparse tensors as dense structures by instead storing only the nonzero weights, along with some metadata indicating how the computations should be performed. Coordinate offset, for example, stores only the non-zero elements, along with their absolute offsets. There are other ways to represent sparse tensors, such as compressed sparse row or compressed sparse column, both used in high performance computing. For a brief overview of the approaches to represent unstructured sparse tensors, refer to Hoefler *et al.* (2021, section 2.2). Hoefler *et al.* (2021, section 7) also summarizes recent successes on accelerating neural network computations using unstructured sparsity.

Finally, although high performance computing has a long history in scientific workflows, its common use cases consist on sparsities larger than 99.9%, which goes beyond what current pruning methods can accomplish (i.e. 50-99%). Nonetheless, if the current trend continues, it is likely to culminate in even sparser models, which will in turn take better advantage of libraries such as Intel's MKL and computing kernels such as sparse BLAS or cuSPARSE. Moreover, scientific computing workflows often leverage non-zero patterns that follow a banded matrix structure, where the non-zero elements are concentrated in few adjacent diagonals. Consequently, successes in reproducing such patterns with structured sparsity are even more likely to benefit from existing high performance computing frameworks. Nonetheless, for simplicity, we will focus on unstructured sparsity in this paper.

7.1.3 Common approaches

We here give a brief and non-exhaustive survey of existing pruning approaches. Notably, magnitude-based methods are ubiquitous in the field. In particular, Hoefler *et al.* (2021, figure 18a) categorize 157 papers published between 1988 and 2020, concluding that magnitude pruning is the most common category, comprising 48 of the surveyed papers (30.57%). Their high presence in the

field, combined with their simplicity, makes them good baselines, justifying our choice to include them in our experiments. The second most common category is regularization, totalling 34 papers (21.66%). As we will see in Section 7.2, writing the pruning objective as L_0 regularized optimization makes its relation to PB optimization more explicit. We use L_0 regularization as a foundation for presenting both the CP and MC methods in Section 7.2. In addition, Hoefler *et al.* (2021, figure 26b) gather reported results of around 90 methods from the last decade that prune a Resnet-50 on Imagenet. There, the Pareto frontier of regularization methods corresponds to the best performances, the second best corresponding to magnitude pruning. Following Hoefler *et al.* (2021), one can roughly categorize other works as follows:

- Data-free methods: although most of these are in fact magnitude pruning, alternative approaches exist, often involving inspection of the final (dense) weights and activations and then fusion or elimination of some based on similarities or regularities (Mussay *et al.* 2021; Srinivas and Babu 2015).

- Input/output sensitivity: inspired by Sietsma and Dow (1988), they also work by inspecting neural network behaviour, but now on a data-driven manner. Particularly, studying how the output of a neuron varies with respect to training data guides the detection of "pruning behaviour". For example, we can eliminate neurons that are almost constant with respect to different training inputs, adding their values back to the network as constant bias terms. Luo, Wu, and Lin (2017) apply this same logic for convolutional filters. Some approaches even use Fourier analysis to detect "pruning behaviour" (H.-G. Han and Qiao 2013; Lauret *et al.* 2006). Alternatively, other approaches prune via searching for redundant activity after looking at the correlation between weights. Cuadros *et al.* (2020), for example, detect such redundancy using PCA.

- First and second order expansion methods: these methods rely on Taylor's theorem. In particular, the later assumes that the second order

expansion approximates the loss behaviour reasonably well near a local minimum of the dense network. Then, they use the per-weight saliency measure to determine which should be pruned (Hassibi and Stork 1992; LeCun, Denker, *et al.* 1989). This requires inverting the Hessian, which one can, in turn, approximate by using the diagonal Fisher matrix (Theis *et al.* 2018), or, more recently, by using K-FAC (Wang *et al.* 2019; Zeng and Urtasun 2019) or revisiting the classic Sherman-Morrison formula (Singh and Alistarh 2020).

- Variational methods: inspired by Kingma, Salimans, *et al.* (2015), they model weights as latent random variables and achieve sparsity by proposing arbitrary prior distributions, such as the log-uniform in Molchanov *et al.* (2017). Recently, Gale *et al.* (2019) showed that these methods can achieve good performance on large models, although results had high variance with respect to both accuracy and sparsity.

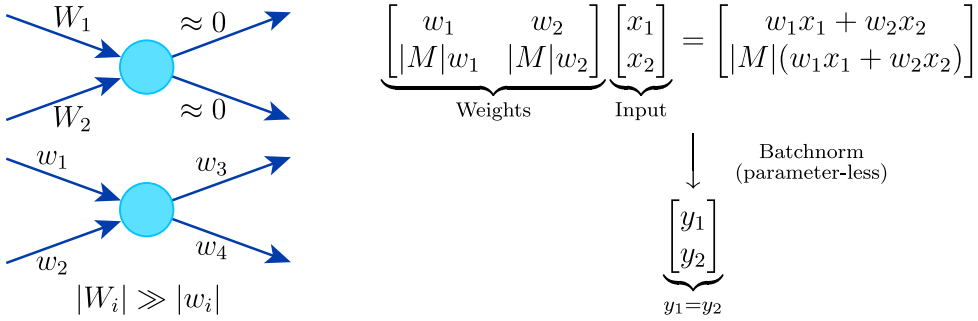
Hoefler *et al.* (2021, section 3) made a more comprehensive list of works pertaining to each of these categories. Even combined, they account for less than half of the studies surveyed there. Furthermore, these methods are more complex and often conceived with only one specific task in mind, making it hard to analyze them in more general PB optimization contexts. For instance, manual heuristics based on inspecting neural network sensitivity to inputs cannot be used in some problems presented in this study. Thus, these alternative methods are not implemented in our experiments.

7.2 Algorithms

In this section, we first talk about magnitude-based approaches in more detail. Then, we discuss how to write pruning as PB optimization and connect it to CP and MC methods from previous chapters.

7.2.1 Magnitude pruning

As mentioned before, magnitude-based methods are dominant in the field. They heuristically rely on L_2-norm summarizing the importance of weights in the neural network, thus choosing the smallest ones for removal. In general, there are some obvious counter-examples to that assumption when taken as such, see Figure 7.2. Nonetheless, when taken in conjunction with gradient-based training, it yields good empirical results. Some of the clear advantages of these methods is their simplicity and the absence of need for storing additional parameters in many of them. In situations where the dense model already occupies most of the available memory, MP approaches are specially convenient. Their performances were among the best in the comparative study by Gale *et al.* (2019).



(a) Output connections (b) Batch normalization

Figure 7.2: Examples where weight magnitudes do not correlate with their importance. On Figure 7.2a, the larger weights have less influence on the model output than the smaller ones due to their outgoing connections. In Figure 7.2b, one row is a scaled version of another. Despite the difference in magnitude, subsequent batch normalization equates their outputs.

In addition to that, under some strong assumptions, magnitude pruning has a theoretical justification. Particularly, it is a special case of the second order methods mentioned in Section 7.1.3 where the Hessian is roughly diagonal and uniform. The saliency measure then becomes proportional to the L_2-norm. Naturally, we also require the other assumptions from Hassibi and Stork (1992) and LeCun, Denker, *et al.* (1989): the gradient has to be approximately zero when pruning (although Singh and Alistarh (2020) extended the framework to waive this assumption); the Hessian has to be invertible; pruning multiple

weights at a time, as opposed to a single one, requires no correlation between them, otherwise we could have the removal of one weight causing a previously unimportant weight to become important.

Pruning can happen following different schedules, such as training and then choosing some percentage of weights to be removed at once. Here, we will follow the gradual magnitude pruning scheme proposed by Zhu and Gupta (2017), which Savarese *et al.* (2020) showed to be a competitive baseline. The schedule removes the lowest norm weights at specified pruning steps to achieve predefined sparsities. Considering final sparsity s_f, initial sparsity s_i, initial pruning timestep t_0, pruning interval Δt and total number of pruning operations n, sparsity at time t is given by:

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3, \qquad \text{for } t \in \{t_0, t_0 + \Delta t, \ldots, t_0 + n\Delta t\}. \quad (7.1)$$

7.2.2 L_0 regularization

This approach involves writing the pruning objective explicitly as:

$$\text{minimize} \underset{X,Y\sim\mathcal{D}}{\mathbb{E}} [\ell(f(X;\mathbf{w}), Y)] + \lambda\|\mathbf{w}\|_0, \qquad \text{for } \mathbf{w} \in \mathcal{W}, \qquad (7.2)$$

where notation is the same as in Example 5, λ is a positive hyperparameter controlling the tradeoff between sparsity and accuracy and the L_0 norm is given by:

$$\|\mathbf{w}\|_0 = \sum_{i=1}^{d} \mathbb{1}[w_i \neq 0].$$

One advantage of using the L_0 norm as opposed to L_1 or L_2 norms, is that it more directly represents what we desire from the optimization. Other approaches, such as Lasso (Tibshirani 1996), rarely cause weights to be exactly zero, requiring subsequent thresholding steps for pruning to occur. Furthermore, L_1 and L_2 regularization are both incompatible with batch normalization layers (Ioffe and Szegedy 2015), since the affine parameters can simply undo the regularizarion, allowing weights with small norms to have significant impact on the output (Azarian *et al.* 2020). The objective in Equation (7.2)

is equivalent to:

$$\text{minimize} \underset{X,Y\sim\mathcal{D}}{\mathbb{E}} [\ell(f(X; \mathbf{w} \odot \mathbf{z}), Y)] + \lambda \sum_{i=1}^{d} z_i, \qquad \text{for } \mathbf{w} \in \mathcal{W},\ \mathbf{z} \in \{0,1\}^d,$$

where we simply defer the L_0 norm of the original weights to an auxiliary discrete variable $\mathbf{z}$. For a fixed $\mathbf{w}$, searching for the optimal $\mathbf{z}$ is an instance of PB optimization and the loss can be written as in Equation (2.2).[2] Then, to use the CP framework described in Section 3.2, one can simply take the homotopy

$$\underset{X,Y\sim\mathcal{D}}{\mathbb{E}} \left[\ell(f(X; \mathbf{w} \odot \sigma\Big(\frac{\mathbf{r}}{\tau}\Big)), Y)\right] + \lambda \sum_{i=1}^{d} \sigma\Big(\frac{r_i}{\tau}\Big), \qquad \text{for } \mathbf{w} \in \mathcal{W},\ \mathbf{r} \in \mathbb{R}^d.$$

We again mention that Luo and Wu (2020), Savarese *et al.* (2020), and Yuan *et al.* (2020) successfully implemented variations of this approach and Azarian *et al.* (2020) also suggested it. Interestingly, again on the same survey, Hoefler *et al.* (2021, section 8.7) define a new parameter efficiency metric using the per-class difficulty on Imagenet, obtained by inspecting Resnet-50 results from papers published the last 10 years. After applying this metric to compare around 90 different papers, the method proposed by Savarese *et al.* (2020) had the best parameter-efficiency of all.

Alternatively, to use the MC framework described in Chapter 4, one can simply minimize:

$$\underset{\substack{X,Y\sim\mathcal{D} \\ z_i\sim\text{Ber}[\theta(\mathbf{r}_i)]}}{\mathbb{E}} \left[\ell(f(X; \mathbf{w} \odot \mathbf{z}), Y) + \lambda \sum_{i=1}^{d} z_i\right], \qquad \text{for } \mathbf{w} \in \mathcal{W},\ \mathbf{r} \in \mathcal{R}. \tag{7.3}$$

By linearity of expectations and independence of the Bernoulli random variables, the above quantity is the same as[3]

$$\underset{\substack{X,Y\sim\mathcal{D} \\ z_i\sim\text{Ber}[\theta(\mathbf{r}_i)]}}{\mathbb{E}} [\ell(f(X; \mathbf{w} \odot \mathbf{z}), Y)] + \lambda \sum_{i=1}^{d} \theta(\mathbf{r}_i). \tag{7.4}$$

[2]When implementing the methods, updates to both main weights and the auxiliary variables happen concurrently, but here we fix $\mathbf{w}$ for ease of exposition.

[3]In some of our preliminary experiments, we attempted to optimize the objective in Equation (7.3) directly, but this proved much more challenging that the one in Equation (7.4).

Notably, in addition to MC methods from Chapter 4, Gumbel-softmax and straight-through estimators described in Chapter 5 can also be combined with Equation (7.4) to be used for pruning. In fact, most of the pruning papers based on stochastic L_0 regularization use the biased estimators from Chapter 5.

L_0 regularized stochastic methods were popularized in pruning by Louizos *et al.* (2017), which combine the Gumbel-softmax with clipping and use a CDF to approximate the regularization term. Some studies report instabilities when implementing their method for image classification (Gale *et al.* 2019; Y. Li and Ji 2020; Savarese *et al.* 2020). On the other hand, for language tasks, it showed superior performance compared to three other baselines in McCarley *et al.* (2019). Subsequent works optimizing stochastic objectives close to Equation (7.4) include Y. Li and Ji (2020), Srinivas, Subramanya, *et al.* (2017), H. Zhou *et al.* (2019), and X. Zhou *et al.* (2021). Among these, most use the biased approaches from Chapter 5, the only exception being Y. Li and Ji (2020). Still, Y. Li and Ji (2020) limit their analysis to structured sparsity and only use 2 samples with the ARM estimator. In our experiments, we will focus on unbiased estimators and include different parametrizations, as well as investigate increasing the number of samples.

7.3 Experiments

We make the transition between smaller and larger scale settings more gradual by first running supermask experiments (H. Zhou *et al.* 2019), where the backbone network is fixed and only the masks change throughout training. Then, we move to the more common pruning setup and train both backbone weights and masks together. We run experiments for 200 epochs and optimize $\mathbf{r}$ with RMSprop, while again initializing $\boldsymbol{\theta}$ in the middle of the hypercube. Results reported for each setting correspond to averages over 5 seeds. We report results on the test set, while noting that the training results were similar.

7.3.1 Supermask

We run experiments on two architecture-dataset combinations: Lenet (LeCun, Bottou, *et al.* 1998) on MNIST and a 6 layer convolutional neural network on CIFAR-10 (Krizhevsky 2009). We exclude MP methods from the comparison, as they rely on training the main network. Further, we only show results for ARMS and escort, since they performed slightly better than sigmoid and LOORF. Additional experimental details can be found in Sections B.3 and B.3.1.

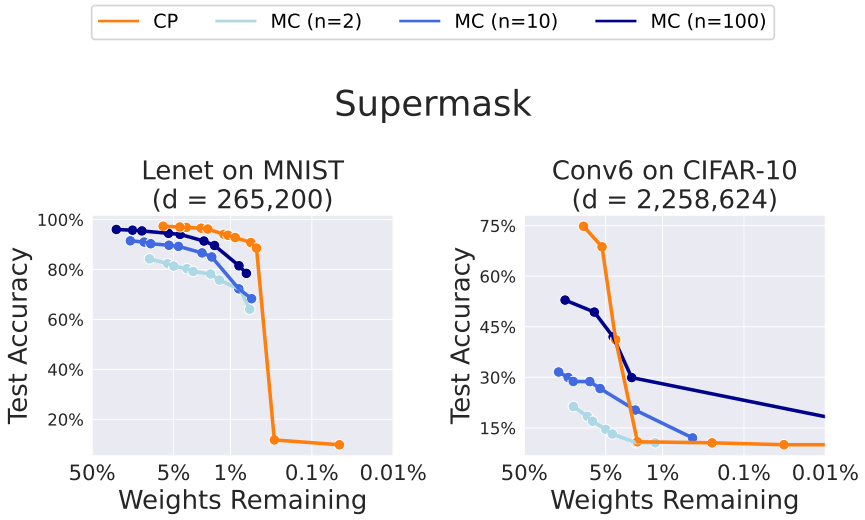


Figure 7.3: Pareto front for supermask experiments.

In Figure 7.3, we show the Pareto front for CP and MC. For multiple objectives and a set of solutions $\mathcal{S}$, this is the largest subset where no element surpasses another in all objectives. Visually, it is a curve with the best performances for each sparsity level.

These results reaffirm those of MaskedNNRegression: it seems that large neural networks change the discrete problem in a way that benefits approaches reliant on $\nabla_{\mathbf{z}} J(\mathbf{z})$. Extrapolations from evaluations outside of $\{0,1\}^d$, which were misleading before (Sections 3.3 and 6.1.4), now produce good results. On top of that, this happens even without the co-adaptation of the backbone network.

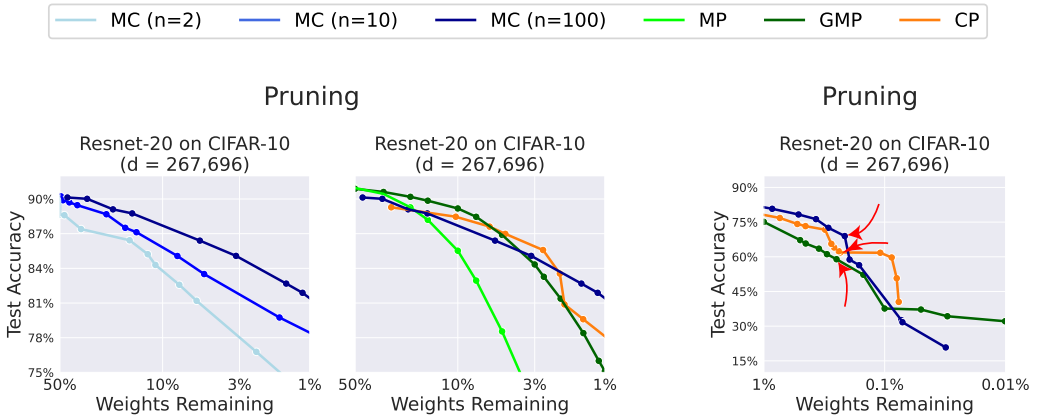
On the other hand, it seems that the difficulty of finding better solutions by mere search, such as in MC methods, increases significantly with larger d. Even using 100 samples is now worse than using a single combination of forward and backward passes. MC methods seem to catch up to CP for very high sparsity, but, at this point, none of the methods performs particularly well.

The contrast between these results and those from Section 6.1 can be connected to previous discussions. Chapter 3 showed the unreliability of the gradient extrapolations from CP methods on simple examples, later confirmed by Microworld experiments. However, there were also signs that performance of such methods might be more correlated with properties of $J(\cdot)$ other than d. As an example, when introducing pathwise gradients from Section 4.2, we noted that the reparametrization estimators had their variance linked to the Lipschitz constant of $J(\cdot)$ instead. Similarly, in Chapter 5, we also cited analytical results linking the Lipschitz constant and "correctness" of ST.

Conversely, the drawbacks discussed in Section 4.3 (i.e. dependence on the current $\boldsymbol{\theta}$ and generalization between $\boldsymbol{\zeta}_h$), are both worsened by high d. The introduction of NN, combined with the initializations, activations and losses used, might have helped control how fast $J(\cdot)$ changes in $(0, 1)^d$, which benefits CP methods, but not MC. Nonetheless, this does not explain what exactly changes, nor how overparametrization leads CP to good solutions or how far these solutions are from the best possible. Those are still open questions whose answers might lead to better algorithms and architecture designs.

7.3.2 Joint pruning

Next, we move to deeper convolutional networks and start training the main weights. Since running sweeps with $n = 100$ on these larger models is challenging, we reuse the best hyperparameters from $n \in \{2, 10\}$. See Section B.4 for alternative visualizations of data from supermask experiments suggesting this is a valid extrapolation. Other experimental details are in Section B.4.1.



(a) Moderate sparsity (b) High sparsity

Figure 7.4: Pareto front for joint pruning experiments.

In Figure 7.4 we first plot MC settings only, then we show MC (n=100) together with CP[4] and the magnitude based approaches. Lastly, we plot only the best performing methods from each category on extreme sparsity ranges. MP does one-shot pruning and GMP follows Equation (7.1).

Surprisingly, MC methods now start getting more competitive against CP, even outperforming it in almost all of the 1% to 0.1% range (note the log scale). Concurrently training $\mathbf{w}$ and $\mathbf{r}$ provides the model with a chance to adapt to the new masks, which seems to significantly benefit MC.

For high sparsity, the best method varies by range of weights remaining: MC when closer to 1%, CP when closer to 0.1% and GMP when closer to 0.01%. To better understand the solutions, we generate Figure 7.5 by selecting one run (out of 5) from each of the settings indicated by arrows in Figure 7.4 and plotting their per-layer sparsities over time.[5] For each method, left plots indicate the percentage of weights remaining relative to the dense layer, whereas right plots show how much of the computation is allocated to each layer at a particular time.

As seen in the left plot, all methods prioritized the protection of the first layer and pruned mostly the higher-level ones. From the right plots, it is evident

[4]Savarese *et al.* (2020) reported better results for this same benchmark. We attribute the difference to the hyperparameter sweep and initialization, see Section B.3 for details.

[5]We emphasize that we studied this same plots for settings from Figure 7.4 other than the selected. Per-method results were similar across high sparsity settings.

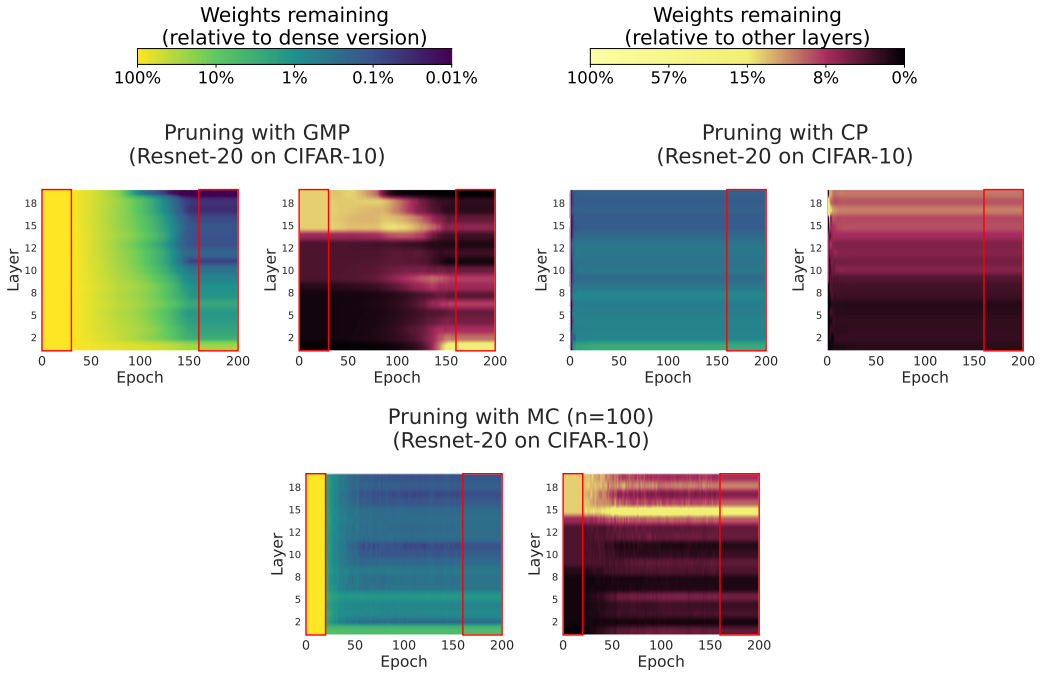


Figure 7.5: Per-layer sparsity for the runs indicated in Figure 7.4. Rectangles correspond to epochs with manually frozen masks.

that the dense Resnet-20 architecture has more (prunable) weights on high-level layers. Even though this initial allocation is mostly maintained in CP and MC, GMP seems to have benefitted from reverting it. It appears to have been an effective way to keep its performance much higher for the sparsest of settings (weights remaining $\approx 0.01\%$).

In addition to that, CP plots consisted mostly of fixed horizontal stripes, in contrast with the "jagged" stripes from MC. CP seems to change the (discrete) masks only in the very first epochs, while MC takes longer to find its solutions. Even after that, MC keeps switching between masks (close in Hamming distance) until fine-tuning begins. GMP is the most gradual of the three, utilizing the whole range of epochs.

Chapter 8

Conclusion

Perhaps one of the main difficulties of adapting PB optimization to neural network contexts is that most of their knowledge base was built around smooth optimization. Techniques such as backpropagation, initialization schemes, optimizers, regularization, normalization, architectures, activations and even loss functions were designed with continuous optimization in mind. In multivariate calculus, the gradient of the loss is already indicative of the best solution in a small neighbourhood. At least locally, a short step in that direction should give the desired outcome. Then, researchers and practitioners formulate their objectives in terms of such functions, while attempting to make the problem as "convex"-like as possible. Finally, neural network design connects this objective with a computational graph that takes advantage of this differentiable framework.

In contrast, a similar set ot techniques is lacking when combining larger-scale PB optimization and neural networks. Problem formulation will not necessarily involve functions that interact particularly well with discrete parameters. The unavailability of equivalent closed-form expressions for the best immediate direction signifies searching exhaustively for such a solution. Still, PB optimization problems resurface when attempting to improve certain aspects of machine learning.

As a consequence of the referred disparity, approaches trying to extend smooth optimization to discrete contexts have to overcome the incompatibility between the two settings. One such approach involves utilizing the gradient of

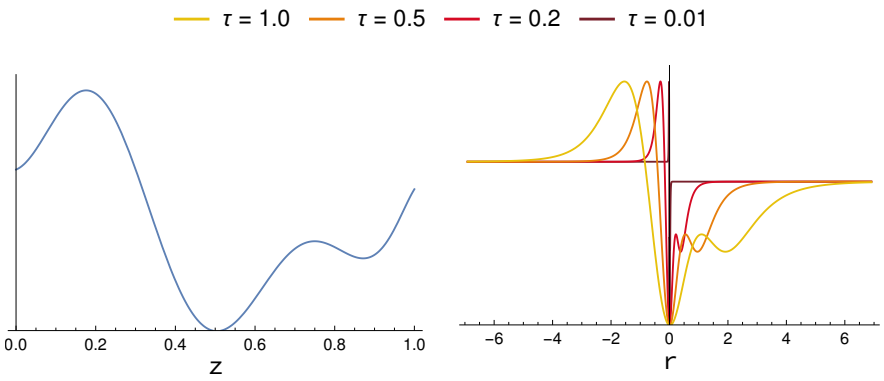
the loss and annealing the problem, as described in this study. Unfortunately, most such methods are afterthoughts and still poorly understood. Perhaps one future direction for improving the PB optimization solutions is to adapt problem and architecture designs by also considering the discrete variables.

Table B.5 summarizes all experimental results from this work. By analyzing them and the background discussion, CP and MC appear to be poor choices for general PB optimization. MC methods are greatly harmed by larger dimensionality. In addition to the inherent difficulty of searching in a large space, exploding importance sampling ratios and unwanted generalization between evaluations make it even less applicable. Despite renewed interest in proposing new estimators with increasingly lower variance, our underwhelming results using optimal control-variates or even true gradients suggest this might be insufficient. Notably, even using 100 samples was not enough to outperform CP in the supermask section.

In spite of concerns regarding the sigmoid, our results indicate that using it is unlikely to be the bottleneck for this use case. Widely different parametrizations appear to have the same limitations. Alternative ideas for improving MC estimation include modeling dependency between the discrete variables and mapping a smaller parameter vector to $\mathbf{z}$, instead of working directly with d dimensions.[1] As none of these avoid the limitations discussed, we believe they are unlikely to lead to long-term advances, serving at best to slightly improve results on some benchmark. Clearly, regardless of having smaller parameters or dependent variables, the resulting method will still ultimately be searching a complex function in an enormous space by mere trial and error.

On the other hand, there is no specific reason why adapting numerical continuation ideas to bridge the gap between continuous and discrete domains should work. Studies often rely on the general intuition of curriculum learning instead. As explained in Section 3.1, homotopy methods were designed to find roots of a non-linear system of equations. Given some assumptions, which we overviewed then, a path between the simplified problem and the original

[1] As a sidenote, depending on how one implement these changes, the equivalence of the PB and MC objectives discussed in Section 4.1.1 might be lost.



(a) $J(z)$ (b) $J\left(\sigma\left(\frac{r}{\tau}\right)\right)$

Figure 8.1: Temperature annealing collapses $J(\cdot)$ at the origin.

one will exist. Gradient-based optimization has a similar motivation: the system equations correspond to the dimensions of the gradient vector and the roots are the critical points. Thus, when CP methods follow the the negative gradient of the smooth loss, there is an implicit attempt to find such points for increasingly lower temperatures. Still, it is not clear why finding a critical point is desirable for infinitely small temperature, or how that connects to the differences between multiple $\boldsymbol{\zeta}_h$. The original assumptions from numerical continuation are likely false in this context.

What ends up happening in these methods is the extrapolation of local behaviour. Although reconstructing $J(\cdot)$ away from the current point is no new endeavour, with Taylor expansions and Padé approximant being some examples, precision decays for far-away regions, which are exactly the ones needed here. Moreover, accurate results require higher-order terms, whose computation is prohibitive if d is large. Our results indicate that these extrapolations are indeed unreliable.

In addition to that, CP methods resulted in solutions oscillating considerably throughout training in all experiments. To understand why, we can take $d = 1$ and note that the chained function $J(\sigma(\cdot/\tau))$ has an active range $(-\Delta r, \Delta r)$, where behaviour is similar to that of $J(\cdot)$ when restricted to $[0, 1]$. For r outside of this range, the chained function will be constant: $J(\zeta_0)$ for lower values or $J(\zeta_1)$ for higher values. As $\tau \to 0$, the whole active range col-

lapses towards the origin, causing r_t to traverse many of the critical points from $J(\cdot)$. The parameter will then change direction depending on whether its per-timestep neighbourhood is ascending or descending. See Figure 8.1 for an illustration.

We did not discuss magnitude-based approaches much in this work. Since the (implicit) masks come from training the main weights, which utilize the gradient of the loss, they are naturally closer to CP methods than to MC. The connection between the two goes beyond that. Considering a single weight w from the neural network, CP rewrites it as:

$$w' = w\sigma\left(\frac{r}{\tau}\right)$$

and weights with $r > 0$ by the end of training are not pruned. The chain rule in this case gives

$$-\frac{\partial J}{\partial r} = w\frac{\partial \sigma(r/\tau)}{\partial r}\left(-\frac{\partial J}{\partial w'}\right).$$

The derivative of the sigmoid is never negative, regardless of τ, whereas the rightmost factor corresponds to the feedback signal from the problem to w'. The LHS is the direction used when updating r. From this expression, r increases in two situations: when w is positive and the gradient feedback indicates it should go up, or when w is negative and the same signal indicates it should go down.

Interestingly, a positive weight that always goes up or a negative weight that always goes down will tend to have larger magnitude, which explains the connection. When used explicitly, this exact criterion: pruning weights by removing those moving toward zero, was shown to work by itself multiple times (Bellec *et al.* 2017; Sanh *et al.* 2020; H. Zhou *et al.* 2019). This observation can perhaps help explain recent successes when generating sparse masks via CP approaches.

One remaining open question is what would happen to joint training if computation was not a problem and we could find the optimal solution of the PB optimization. Using pruning as an example, on the one hand, this perhaps could lead to major improvements and neural networks significantly smaller

than the current state-of-the-art would perform closely to dense models. Conveniently, this could unlock a multitude of applications, specially if deployment speedups using the scientific workflows from Section 7.1.2 became possible.

In contrast, it could also be the case that training of the main network ends up being robust to worse mask choices. MC results from the pruning section seem to be one such example: supermask experiments indicated that the high d handicapped MC methods, but their performance was significantly improved when pruning jointly. This robustness could then imply that using the optimal $\mathbf{z}^*$ does not improve the current state-of-the-art much.

One key point for future work is answering what changed for the methods using $\nabla_{\mathbf{z}} J(\mathbf{z})$ when combined with neural networks. Despite the main NN being fixed, MaskedNNRegression and supermask had them performing well. We believe studying similar phenomena more analytically is paramount for future progress, as it provides understanding that might not be achievable by more practical investigation on larger models. Broadly speaking, the scale of such systems and the excess of confounding factors limit the range of possible tools when interpreting results.

Of equal importance is the design of smaller experiments that, despite the scale, still capture the essence of the problem. Relative computational feasibility allows for more trial-and-error of new ideas, while also benefiting from a wider range of tools. Using pruning as an example once more, if the input-output mapping is part of the experimental design, we can compare the size of solutions found to the actual number of degrees of freedom, which is not possible for image classification problems.

Better understanding the aforementioned behaviour is a more immediate step that perhaps can lead to more conscious model and algorithmic design. These advances, in turn, can likely be adapted to a wide array of different fields and problems, including continual learning, multi-task learning, pruning, transfer-learning, ticket search, Bayesian optimization, training binary/ternary NN and mixed-precision NN quantization.

References

[1] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *International Conference on Machine Learning*, PMLR, 2019.

[2] E. L. Allgower and K. Georg, *Introduction to numerical continuation methods*. Society for Industrial and Applied Mathematics, 2003.

[3] E. L. Allgower and K. Georg, *Numerical continuation methods: an introduction*. Springer Science & Business Media, 2012, vol. 13.

[4] E. Andriyash, A. Vahdat, and B. Macready, "Improved gradient-based optimization over discrete distributions," *Computing Research Repository*, 2018. arXiv: 1810.00116.

[5] S.-K. Au and J. Beck, "Important sampling in high dimensions," *Structural safety*, vol. 25, no. 2, 2003.

[6] K. Azarian, Y. Bhalgat, J. Lee, and T. Blankevoort, "Learned threshold pruning," *Computing Research Repository*, 2020. arXiv: 2003.00075.

[7] G. Bellec, D. Kappel, W. Maass, and R. Legenstein, "Deep rewiring: Training very sparse deep networks," *Computing Research Repository*, 2017. arXiv: 1711.05136.

[8] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *Computing Research Repository*, 2013. arXiv: 1308.3432.

[9] J. Bethge, H. Yang, M. Bornstein, and C. Meinel, "Back to simplicity: How to train accurate BNNs from scratch?" *Computing Research Repository*, 2019. arXiv: 1906.08637.

[10] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" *Proceedings of machine learning and systems*, vol. 2, 2020.

[11] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American statistical Association*, vol. 112, no. 518, 2017.

[12] E. Boros and P. L. Hammer, "Pseudo-Boolean optimization," *Discrete applied mathematics*, vol. 123, no. 1-3, 2002.

[13] E. Boros, P. L. Hammer, M. Minoux, and D. J. Rader Jr, "Optimal cell flipping to minimize channel density in VLSI design and pseudo-Boolean optimization," *Discrete applied mathematics*, vol. 90, no. 1-3, 1999.

[14] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[15] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 11, 2001.

[16] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, 2020.

[17] A. Brutzkus, A. Globerson, E. Malach, and S. Shalev-Shwartz, "SGD learns over-parameterized networks that provably generalize on linearly separable data," *Computing Research Repository*, 2017. arXiv: 1710.10174.

[18] A. Bulat, B. Martinez, and G. Tzimiropoulos, "High-capacity expert binary networks," *Computing Research Repository*, 2020. arXiv: 2010.03558.

[19] A. Bulat, G. Tzimiropoulos, J. Kossaifi, and M. Pantic, "Improved training of binary networks for human pose estimation and image recognition," *Computing Research Repository*, 2019. arXiv: 1904.05868.

[20] E. K. Chong and S. H. Zak, *An introduction to optimization*. John Wiley & Sons, 2013, vol. 75.

[21] J. Cichon and W.-B. Gan, "Branch-specific dendritic Ca2+ spikes cause persistent synaptic plasticity," *Nature*, vol. 520, no. 7546, 2015.

[22] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[23] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Workshop on statistical learning in computer vision, European Conference on Computer Vision*, vol. 1, 2004.

[24] X. S. Cuadros, L. Zappella, and N. Apostoloff, "Filter distillation for network compression," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020.

[25] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE computer society conference on computer vision and pattern recognition*, IEEE, vol. 1, 2005.

[26] G. Dantzig and D. R. Fulkerson, "On the max flow min cut theorem of networks," *Linear inequalities and related systems*, vol. 38, 2003.

[27] S. Daulton, X. Wan, D. Eriksson, M. Balandat, M. A. Osborne, and E. Bakshy, "Bayesian optimization over discrete and mixed spaces via probabilistic reparameterization," *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[28] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, 2021.

[29] A. Dimitriev and M. Zhou, "ARMS: Antithetic-REINFORCE-multi-sample gradient for binary variables," in *International Conference on Machine Learning*, PMLR, 2021.

[30] Y. Dinitz, "Algorithm for solution of a problem of maximum flow in networks with power estimation," *Soviet Mathematics Doklady*, vol. 11, 1970.

[31] Z. Dong, A. Mnih, and G. Tucker, "DisARM: An antithetic gradient estimator for binary latent variables," *Advances in neural information processing systems*, vol. 33, 2020.

[32] Z. Dong, A. Mnih, and G. Tucker, "Coupled gradient estimators for discrete latent variables," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[33] S. S. Du, X. Zhai, B. Poczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," *Computing Research Repository*, 2018. arXiv: `1810.02054`.

[34] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM*, vol. 19, no. 2, 1972.

[35] K. Fan, Z. Wang, J. Beck, J. Kwok, and K. A. Heller, "Fast second order stochastic backpropagation for variational inference," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[36] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *The Journal of Machine Learning Research*, vol. 23, no. 1, 2022.

[37] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *Computing Research Repository*, 2018. arXiv: `1803.03635`.

[38] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, 1999.

[39] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *Computing Research Repository*, 2019. arXiv: 1902.09574.

[40] P. Glasserman, *Monte Carlo methods in financial engineering*. Springer, 2004, vol. 53.

[41] F. Glover, "Tabu search—part i," *ORSA Journal on computing*, vol. 1, no. 3, 1989.

[42] F. Glover, "Tabu search—part ii," *ORSA Journal on computing*, vol. 2, no. 1, 1990.

[43] P. W. Glynn, "Likelihood ratio gradient estimation for stochastic systems," *Communications of the ACM*, vol. 33, no. 10, 1990.

[44] W. Grathwohl, D. Choi, Y. Wu, G. Roeder, and D. Duvenaud, "Backpropagation through the void: Optimizing control variates for black-box gradient estimation," *Computing Research Repository*, 2017. arXiv: 1711.00123.

[45] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, vol. 1, 1981.

[46] S. Gu, S. Levine, I. Sutskever, and A. Mnih, "Muprop: Unbiased backpropagation for stochastic neural networks," *Computing Research Repository*, 2015. arXiv: 1511.05176.

[47] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *Computing Research Repository*, 2018. arXiv: 1812.05905.

[48] P. L. Hammer and R. Holzman, "Approximations of pseudo-Boolean functions; applications to game theory," *Zeitschrift für Operations Research*, vol. 36, no. 1, 1992.

[49] P. L. Hammer and E. Shlifer, "Applications of pseudo-Boolean methods to economic problems," *Theory and decision*, vol. 1, 1971.

[50] H.-G. Han and J.-F. Qiao, "A structure optimisation algorithm for feedforward neural network construction," *Neurocomputing*, vol. 99, 2013.

[51] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[52] P. Hansen, "The steepest ascent mildest descent heuristic for combinatorial programming," in *Congress on numerical methods in combinatorial optimization*, 1986.

[53] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," *Advances in neural information processing systems*, vol. 5, 1992.

[54] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *Computing Research Repository*, 2012. arXiv: 1207.0580.

[55] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *The Journal of Machine Learning Research*, vol. 22, no. 1, 2021.

[56] R. Horst and P. M. Pardalos, *Handbook of global optimization*. Springer Science & Business Media, 2013, vol. 2.

[57] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[58] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pmlr, 2015.

[59] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with Gumbel-softmax," *Computing Research Repository*, 2016. arXiv: 1611.01144.

[60] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, "How easy is local search?" *Journal of computer and system sciences*, vol. 37, no. 1, 1988.

[61] M. Jünger, A. Martin, G. Reinelt, and R. Weismantel, "Quadratic 0/1 optimization and a decomposition approach for the placement of electronic circuits," *Mathematical programming*, vol. 63, 1994.

[62] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 2010.

[63] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *Computing Research Repository*, 2013. arXiv: 1312.6114.

[64] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," *Advances in neural information processing systems*, vol. 28, 2015.

[65] V. Kolmogorov and C. Rother, "Minimizing nonsubmodular functions with graph cuts - a review," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 7, 2007.

[66] W. Kool, H. van Hoof, and M. Welling, "Buy 4 reinforce samples, get a baseline for free!" *International Conference on Learning Representations Workshop*, 2019.

[67] B. H. Korte, J. Vygen, B. Korte, and J. Vygen, *Combinatorial optimization*. Springer, 2011, vol. 1.

[68] O. Kramer and O. Kramer, *Genetic algorithms*. Springer, 2017.

[69] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009.

[70] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, 2017.

[71] W. Kubiak, "New results on the completion time variance minimization," *Discrete Applied Mathematics*, vol. 58, no. 2, 1995.

[72] K. V. Kuchibhotla, J. V. Gill, G. W. Lindsay, E. S. Papadoyannis, R. E. Field, T. A. H. Sten, K. D. Miller, and R. C. Froemke, "Parallel processing by cortical inhibition enables context-dependent behavior," *Nature neuroscience*, vol. 20, no. 1, 2017.

[73] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," *Information Fusion*, vol. 85, 2022.

[74] Q. Lan, S. Tosatto, H. Farrahi, and A. R. Mahmood, "Model-free policy learning with reward gradients," *Computing Research Repository*, 2021. arXiv: 2103.05147.

[75] T. Lattimore and C. Szepesvári, *Bandit algorithms*. Cambridge University Press, 2020.

[76] P. Lauret, E. Fock, and T. A. Mara, "A node pruning algorithm based on a Fourier amplitude sensitivity test method," *IEEE transactions on neural networks*, vol. 17, no. 2, 2006.

[77] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations research*, vol. 14, no. 4, 1966.

[78] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998.

[79] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, "A tutorial on energy-based learning," *Predicting structured data*, vol. 1, no. 0, 2006.

[80] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989.

[81] N. Lee, T. Ajanthan, S. Gould, and P. H. Torr, "A signal propagation perspective for pruning neural networks at initialization," *Computing Research Repository*, 2019. arXiv: 1906.06307.

[82] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *Computing Research Repository*, 2020. arXiv: 2006.16668.

[83] B. Li, T. Bengtsson, and P. Bickel, "Curse-of-dimensionality revisited: Collapse of importance sampling in very large scale systems," *Rapport technique*, vol. 85, 2005.

[84] S. Z. Li, *Markov random field modeling in computer vision*. Springer Science & Business Media, 2012.

[85] Y. Li and S. Ji, "L0-ARM: Network sparsification via stochastic binary optimization," in *The European Conference on Machine Learning*, Springer, 2020.

[86] G. Lorberbom, A. Gane, T. Jaakkola, and T. Hazan, "Direct optimization through argmax for discrete variational auto-encoder," *Advances in neural information processing systems*, vol. 32, 2019.

[87] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," *Computing Research Repository*, 2017. arXiv: 1712.01312.

[88] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, IEEE, vol. 2, 1999.

[89] M. Luby, "A simple parallel algorithm for the maximal independent set problem," in *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, 1985.

[90] J.-H. Luo and J. Wu, "Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference," *Pattern Recognition*, vol. 107, 2020.

[91] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017.

[92] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *Computing Research Repository*, 2016. arXiv: 1611.00712.

[93] C. J. Maddison, D. Tarlow, and T. Minka, "A* sampling," *Advances in neural information processing systems*, vol. 27, 2014.

[94] T. S. Madhulatha, "An overview on clustering methods," *Computing Research Repository*, 2012. arXiv: 1205.1117.

[95] Z. Mai, R. Li, J. Jeong, D. Quispe, H. Kim, and S. Sanner, "Online continual learning in image classification: An empirical survey," *Neurocomputing*, vol. 469, 2022.

[96] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir, "Proving the lottery ticket hypothesis: Pruning is all you need," in *International Conference on Machine Learning*, PMLR, 2020.

[97] A. Mallya and S. Lazebnik, "Packnet: Adding multiple tasks to a single network by iterative pruning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018.

[98] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, "Training binary neural networks with real-to-binary convolutions," *Computing Research Repository*, 2020. arXiv: 2003.11535.

[99] J. McCarley, R. Chakravarti, and A. Sil, "Structured pruning of a BERT-based question answering model," *Computing Research Repository*, 2019. arXiv: 1910.06360.

[100] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*, vol. 24, Elsevier, 1989.

[101] C. C. McGeoch and C. Wang, "Experimental evaluation of an adiabiatic quantum system for combinatorial optimization," in *Proceedings of the ACM International Conference on Computing Frontiers*, 2013.

[102] J. Mei, C. Xiao, B. Dai, L. Li, C. Szepesvári, and D. Schuurmans, "Escaping the gravitational pull of softmax," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[103] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning.," *J. Mach. Learn. Res.*, vol. 21, no. 132, 2020.

[104] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *International Conference on Machine Learning*, PMLR, 2017.

[105] A. Montanaro, "Quantum algorithms: An overview," *npj Quantum Information*, vol. 2, no. 1, 2016.

[106] B. Mussay, D. Feldman, S. Zhou, V. Braverman, and M. Osadchy, "Data-independent structured pruning of neural networks via coresets," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, 2021.

[107] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro, "Towards understanding the role of over-parametrization in generalization of neural networks," *Computing Research Repository*, 2018. arXiv: 1805.12076.

[108] R. O'Donnell, *Analysis of boolean functions*. Cambridge University Press, 2014.

[109] L. Orseau, M. Hutter, and O. Rivasplata, "Logarithmic pruning is all you need," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[110] G. H. Otazu, L.-H. Tai, Y. Yang, and A. M. Zador, "Engaging in an auditory task suppresses responses in auditory cortex," *Nature neuroscience*, vol. 12, no. 5, 2009.

[111] A. B. Owen, *Monte carlo theory, methods and examples*, 2013.

[112] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural networks*, vol. 113, 2019.

[113] M. Paulus, D. Choi, D. Tarlow, A. Krause, and C. J. Maddison, "Gradient estimation with stochastic softmax tricks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[114] M. B. Paulus, C. J. Maddison, and A. Krause, "Rao-blackwellizing the straight-through Gumbel-softmax gradient estimator," *Computing Research Repository*, 2020. arXiv: 2010.04838.

[115] A. Pensia, S. Rajput, A. Nagle, H. Vishwakarma, and D. Papailiopoulos, "Optimal lottery tickets via subset sum: Logarithmic overparameterization is sufficient," *Advances in neural information processing systems*, vol. 33, 2020.

[116] A. T. Phillips and J. B. Rosen, "A quadratic assignment formulation of the molecular conformation problem," *Journal of Global Optimization*, vol. 4, 1994.

[117] J.-C. Picard and H. D. Ratliff, "A cut approach to the rectilinear distance facility location problem," *Operations Research*, vol. 26, no. 3, 1978.

[118] R. Reed, "Pruning algorithms - a survey," *IEEE transactions on Neural Networks*, vol. 4, no. 5, 1993.

[119] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[120] L. Richter, A. Boustati, N. Nüsken, F. Ruiz, and O. D. Akyildiz, "Vargrad: A low-variance gradient estimator for variational inference," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[121] I. G. Rosenberg, "Reduction of bivalent maximization to the quadratic case.," 1975.

[122] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, The Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

[123] R. Y. Rubinstein and A. Shapiro, "Optimization of static simulation models by the score function method," *Mathematics and Computers in Simulation*, vol. 32, no. 4, 1990.

[124] S. J. Russell, *Artificial intelligence a modern approach, 4th edition*. Pearson Education, Inc., 2021.

[125] P. Salamon, P. Sibani, and R. Frost, *Facts, conjectures, and improvements for simulated annealing*. Society for Industrial and Applied Mathematics, 2002.

[126] V. Sanh, T. Wolf, and A. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[127] P. Savarese, H. Silva, and M. Maire, "Winning the lottery with continuous sparsification," *Advances in neural information processing systems*, vol. 33, 2020.

[128] D. Schuurmans and F. Southey, "Local search characteristics of incomplete SAT procedures," *Artificial Intelligence*, vol. 132, no. 2, 2001.

[129] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *International Conference on Machine Learning*, PMLR, 2018.

[130] A. Shekhovtsov and V. Yanush, "Reintroducing straight-through estimators as principled methods for stochastic binary networks," in *DAGM German Conference on Pattern Recognition*, Springer, 2021.

[131] J. Shi, Y. Zhou, J. Hwang, M. Titsias, and L. Mackey, "Gradient estimation with discrete Stein operators," *Advances in neural information processing systems*, vol. 35, 2022.

[132] Sietsma and Dow, "Neural net pruning - why and how," in *IEEE 1988 International Conference on Neural Networks*, IEEE, 1988.

[133] S. P. Singh and D. Alistarh, "Woodfisher: Efficient second-order approximation for neural network compression," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[134] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *Computing Research Repository*, 2015. arXiv: `1507.06149`.

[135] S. Srinivas, A. Subramanya, and R. Venkatesh Babu, "Training sparse neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017.

[136] R. Sutton, "Two problems with back propagation and other steepest descent learning procedures for networks," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*, 1986.

[137] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. MIT press, 2018.

[138] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[139] R. Szeliski, *Computer vision: algorithms and applications*. Springer Nature, 2022.

[140] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods for markov random fields with smoothness-based priors," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 6, 2008.

[141] G. Tavares, *New algorithms for Quadratic Unconstrained Binary Optimization (QUBO) with applications in engineering and social sciences*. Rutgers The State University of New Jersey, School of Graduate Studies, 2008.

[142] L. Theis, I. Korshunova, A. Tejani, and F. Huszár, "Faster gaze prediction with dense networks and Fisher pruning," *Computing Research Repository*, 2018. arXiv: `1801.05787`.

[143] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, 1996.

[144] M. Titsias and J. Shi, "Double control variates for gradient estimation in discrete latent variable models," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2022.

[145] G. Tucker, A. Mnih, C. J. Maddison, J. Lawson, and J. Sohl-Dickstein, "REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[146] P. J. Van Laarhoven, E. H. Aarts, P. J. van Laarhoven, and E. H. Aarts, *Simulated annealing*. Springer, 1987.

[147] C. Wang, R. Grosse, S. Fidler, and G. Zhang, "Eigendamage: Structured pruning in the Kronecker-factored eigenbasis," in *International conference on machine learning*, PMLR, 2019.

[148] M. White, "Intermediate machine learning," in *University of Alberta course notes*, 2022.

[149] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, 1992.

[150] D. P. Williamson and D. B. Shmoys, *The design of approximation algorithms*. Cambridge university press, 2011.

[151] M. Xu, M. Quiroz, R. Kohn, and S. A. Sisson, "Variance reduction properties of the reparameterization trick," in *The 22nd international conference on artificial intelligence and statistics*, PMLR, 2019.

[152] M. Yin, N. Ho, B. Yan, X. Qian, and M. Zhou, "Probabilistic best subset selection via gradient-based optimization," *Computing Research Repository*, 2020. arXiv: 2006.06448.

[153] M. Yin, Y. Yue, and M. Zhou, "Arsm: Augment-REINFORCE-swap-merge estimator for gradient backpropagation through categorical variables," in *International Conference on Machine Learning*, PMLR, 2019.

[154] M. Yin and M. Zhou, "ARM: Augment-REINFORCE-merge gradient for stochastic binary networks," *Computing Research Repository*, 2018. arXiv: 1807.11143.

[155] X. Yuan, P. Savarese, and M. Maire, "Growing efficient deep networks by structured continuous sparsification," *Computing Research Repository*, 2020. arXiv: 2007.15353.

[156] W. Zeng and R. Urtasun, "MLPrune: Multi-layer pruning for automated neural network compression," *Computing Research Repository*, 2019. arXiv: 2101.06608.

[157] H. Zhou, J. Lan, R. Liu, and J. Yosinski, "Deconstructing lottery tickets: Zeros, signs, and the supermask," *Advances in neural information processing systems*, vol. 32, 2019.

[158] X. Zhou, W. Zhang, H. Xu, and T. Zhang, "Effective sparsification of neural networks with global sparsity constraint," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[159] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," *Computing Research Repository*, 2017. arXiv: 1710.01878.

Appendix A

Proofs

A.1 Multilinear form

We start with the following lemma, then proceed to the main proof.

Lemma 8. *As in the main text, we denote* $\mathcal{D} = \{1, 2, ..., d\}$. *With* $\boldsymbol{x} \in \mathbb{R}^d$, *consider the following:*

$$g(\boldsymbol{x}) = \sum_{\mathcal{S} \subseteq \mathcal{D} \setminus \{i\}} q_{\mathcal{S}}(x_i) \prod_{j \in \mathcal{S}} x_j.$$

$g(\boldsymbol{x})$ *being independent of* x_i *implies* $q_{\mathcal{S}}(x_i)$ *is also independent of* x_i *for all* $\mathcal{S} \subseteq \mathcal{D} \setminus \{i\}$.

Proof To simplify exposition, we will consider only $\mathbf{z} \in \{0, 1\}^d$ instead of $\mathbf{x}$. Extension to $\mathbb{R}^d$ follows a similar argument, which we mention in the end. Using notation from Equation (2.3), $g(\mathbf{z})$ independent of z_i means:

$$g(\mathbf{1}^{\mathcal{S}}) = g(\mathbf{1}^{\mathcal{S} \cup \{i\}}), \qquad \text{for } \mathcal{S} \subseteq \mathcal{D} \setminus \{i\}.$$

Also, for $\mathcal{S} \subseteq \mathcal{D} \setminus \{i\}$, we have:

$$g(\mathbf{1}^{\mathcal{S}}) = \sum_{\mathcal{T} \subseteq \mathcal{S}} q_{\mathcal{T}}(0) \quad \text{and} \quad g(\mathbf{1}^{\mathcal{S} \cup \{i\}}) = \sum_{\mathcal{T} \subseteq \mathcal{S}} q_{\mathcal{T}}(1).$$

We want to show that:

$$q_{\mathcal{S}}(0) = q_{\mathcal{S}}(1), \qquad \text{for } \mathcal{S} \subseteq \mathcal{D} \setminus \{i\},$$

which we prove by induction:

- **Base case:** we consider the empty set

$$g(\mathbf{1}^{\emptyset}) = q_{\emptyset}(0) \quad \text{and} \quad g(\mathbf{1}^{\{i\}}) = q_{\emptyset}(1).$$

By our main assumption from the statement, $g(\mathbf{1}^{\emptyset}) = g(\mathbf{1}^{\{i\}})$. Therefore:

$$q_{\emptyset}(0) = q_{\emptyset}(1).$$

- **General case:** assume that, for $\mathcal{S}' \subseteq \mathcal{D} \setminus \{i\}$ and $|\mathcal{S}'| \leq d-1$, we have $q_{\mathcal{S}'}(1) = q_{\mathcal{S}'}(0)$. Then, for $\mathcal{S} \subseteq \mathcal{D} \setminus \{i\}$ and $|\mathcal{S}| = d$:

$$\begin{aligned} g(\mathbf{1}^{\mathcal{S}\cup\{i\}}) &= \sum_{\mathcal{T}\subseteq\mathcal{S}} q_{\mathcal{T}}(1) \\ &= q_{\mathcal{S}}(1) + \sum_{\mathcal{T}\not\subseteq\mathcal{S}} q_{\mathcal{T}}(1) \\ &= q_{\mathcal{S}}(1) + \sum_{\mathcal{T}\not\subseteq\mathcal{S}} q_{\mathcal{T}}(0) \end{aligned}$$

and we know that:

$$g(\mathbf{1}^{\mathcal{S}\cup\{i\}}) = g(\mathbf{1}^{\mathcal{S}}) = \sum_{\mathcal{T}\subseteq\mathcal{S}} q_{\mathcal{T}}(0).$$

Combining both:

$$\begin{aligned} q_{\mathcal{S}}(1) + \sum_{\mathcal{T}\not\subseteq\mathcal{S}} q_{\mathcal{T}}(0) &= \sum_{\mathcal{T}\subseteq\mathcal{S}} q_{\mathcal{T}}(0) \\ q_{\mathcal{S}}(1) &= q_{\mathcal{S}}(0). \end{aligned}$$

To extend this proof to $\mathbb{R}^d$, simply use $z_i = x'$ and $z_i = x''$ in the comparisons, instead of 0 and 1 and allow arbitrary z_j for $j \neq i$ in place of $z_j = 1$. The additional terms due to the product of the coordinates $\neq i$ will cancel. ■

Theorem 1. *An arbitrary twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ will have a Hessian whose diagonal is equal to $\mathbf{0} = [0, \cdots, 0]^\top$ if and only if it can be represented in the form of Equation* (2.4).

Proof We consider $x \in \mathbb{R}^d$ and start by proving:

$$f(\mathbf{x}) = \mathscr{P}_J(\mathbf{x}) \Rightarrow \frac{\partial^2 f(\mathbf{x})}{\partial x_i^2} = 0, \quad \text{for all } i.$$

Which follows from the fact that no x_i appears squared for $\mathscr{P}_J(\mathbf{x})$ following Equation (2.4). Then, we prove the reverse implication:

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_i^2} = 0 \quad \text{for all } i \Rightarrow f(\mathbf{x}) = \mathscr{P}_J(\mathbf{x}).$$

By induction:

- **Base case:** We assume $f : \mathbb{R} \to \mathbb{R}$. Using integration we have

$$\begin{aligned} \frac{\partial^2 f(x)}{\partial x^2} &= 0 \\ f(x) &= ax + b \\ &= w_1 x + w_\emptyset. \end{aligned}$$

- **General case:** we assume that, for $f' : \mathbb{R}^{d-1} \to \mathbb{R}$

$$\begin{aligned} \frac{\partial^2 f'(\mathbf{x})}{\partial x_i^2} = 0, \quad \text{for } 1 \leq i \leq d-1 \Leftrightarrow f'(\mathbf{x}) &= \mathscr{P}_{J'}(\mathbf{x}) \\ &= \sum_{\mathcal{S} \subseteq \{1,\ldots,d-1\}} w'_{\mathcal{S}} \prod_{i \in \mathcal{S}} x_i. \end{aligned} \tag{A.1}$$

  Then, we consider a function $f^d : \mathbb{R} \to \mathbb{R}$. Assuming that:

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_i^2} = 0, \quad \text{for all } 1 \leq i \leq d,$$

  we want to show that this implies $f(\cdot)$ can be written in the form of Equation (2.4). Fix an arbitrary i, and note that, for any value of x_i, since the other $d-1$ coordinates have corresponing entries on the diagonal of the Hessian equal to zero, we can write $f(\cdot, x_i, \cdot)$ as in the RHS of Equation (A.1) (with coordinates remapped). The specific $d-1$ multilinear polynomial from Equation (A.1) will depend on the value of x_i. Therefore, we can write $f(\cdot)$ as:

$$f(\mathbf{x}) = \sum_{\mathcal{S} \subseteq \mathcal{D} \setminus \{i\}} w_{\mathcal{S}}(x_i) \prod_{j \in \mathcal{S}} x_j.$$

Taking the derivative:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\{i\}} \frac{\partial w_{\mathcal{S}}(x_i)}{\partial x_i} \prod_{j\in\mathcal{S}} x_j.$$

In addition, by assumption, we also have $\partial^2 f(\mathbf{x})/\partial x_i^2 = 0$. Consequently, $\partial f(\mathbf{x})/\partial x_i$ does not depend on x_i. Using Lemma 8, with $g(\mathbf{x}) = \partial f(\mathbf{x})/\partial x_i$ and $q_{\mathcal{S}}(x_i) = \partial w_{\mathcal{S}}(x_i)/\partial x_i$, we have that $\partial w_{\mathcal{S}}(x_i)/\partial x_i$ is also independent of x_i. Meaning:

$$w_{\mathcal{S}}(x_i) = w'_{\mathcal{S}} + x_i w''_{\mathcal{S}}.$$

Finally, this implies:

$$\begin{aligned} f(\mathbf{x}) &= \sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\{i\}} w_{\mathcal{S}}(x_i) \prod_{j\in\mathcal{S}} x_j \\ &= \sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\{i\}} (w'_{\mathcal{S}} + x_i w''_{\mathcal{S}}) \prod_{j\in\mathcal{S}} x_j \\ &= \sum_{\mathcal{S}\subseteq\mathcal{D}} w_{\mathcal{S}} \prod_{i\in\mathcal{S}} x_i. \end{aligned}$$

Where, for any $\mathcal{S} \subseteq \mathcal{D} \setminus \{i\}$, we have $w_{\mathcal{S}} = w'_{\mathcal{S}}$ and $w_{\mathcal{S}\cup\{i\}} = w''_{\mathcal{S}}$.

■

A.2 Iterative ARMS procedure

To prove that the sampling procedure is equivalent, since dimensions are independent, we restrict ourselves to $d = 1$ and omit the index i. Assume:

$$E^{(s)} = -\log U^{(s)}, \qquad \text{for } U^{(1)}, \ldots, U^{(n)} \overset{i.i.d.}{\sim} U[0,1].$$

Note also that the original ARMS procedure from Algorithm 5 samples the Dirichlet by using:

$$d^{(s)} = \frac{\log U^{(s)}}{\sum_{s'=1}^{n} \log U^{(s')}} = \frac{E^{(s)}}{\sum_{s'=1}^{n} E^{(s')}}. \tag{A.2}$$

Furthermore, for $E^{(s)}$ as defined above and considering the exponential and Gamma distributions, the following holds:

$$E^{(s)} \sim \text{Exp}[1] = \text{Gamma}[1,1].$$

The PDF of $E^{(s)}$ is:

$$f_{E^{(s)}}(x) = \mathrm{e}^{-x}, \qquad \text{for } x \geq 0.$$

Additionally, the sum of these multiple independent Gamma random variables with common scale parameter satisfies

$$\sum_{s'=1}^{n} E^{(s')} \sim \text{Gamma}[n, 1], \qquad f_{\sum E^{(s')}}(x) = \frac{x^{n-1}\mathrm{e}^{-x}}{(n-1)!}, \quad \text{for } x \geq 0. \tag{A.3}$$

The procedure involves sampling the whole $\sum_{s'=1}^{n} E^{(s')}$ sum at once and then iteratively sampling each of its composing summands $E^{(s)}$. We therefore want to know the PDF of $E^{(s)}$ given the sum. By Bayes rule:

$$\begin{aligned}
f_{E^{(s)}|\sum E^{(s')}=R}(x) &= \frac{f_{E^{(s)}}(x) f_{\sum E^{(s')}|E^{(s)}=x}(R)}{f_{\sum E^{(s')}}(R)} \\
&= \frac{f_{E^{(s)}}(x) f_{\sum_{s'\neq s} E^{(s')}|E^{(s)}=x}(R-x)}{f_{\sum E^{(s')}}(R)} \\
&= \frac{f_{E^{(s)}}(x) f_{\sum_{s'\neq s} E^{(s')}}(R-x)}{f_{\sum E^{(s')}}(R)} && \triangleright \text{ Since } E^{(s)} \text{ and } E^{(s')} \text{ are i.i.d.} \\
&= \frac{\mathrm{e}^{-x}\left(\frac{(R-x)^{n-2}\mathrm{e}^{-(R-x)}}{(n-2)!}\right)}{\frac{R^{n-1}\mathrm{e}^{-R}}{(n-1)!}}, \qquad \text{for } 0 \leq x \leq R \\
&= \frac{n-1}{R}\left(1 - \frac{x}{R}\right)^{n-2}.
\end{aligned}$$

The CDF of this r.v. is:

$$\begin{aligned}
F_{E^{(s)}|\sum E^{(s')}=R}(x) &= \int_0^x f_{E^{(s)}|\sum E^{(s')}=R}(x')\mathrm{d}x' \\
&= 1 - \left(1 - \frac{x}{R}\right)^{n-1}, \qquad \text{for } 0 \leq x \leq R,
\end{aligned} \tag{A.4}$$

and its inverse is given by

$$\left(F_{E^{(s)}|\sum E^{(s')}=R}\right)^{-1}(y) = -R(1-y)^{\frac{1}{n-1}} + R, \qquad \text{for } 0 \leq y \leq 1. \tag{A.5}$$

By inverse transform sampling, we can sample a random variable with CDF from Equation (A.4) by first sampling $U \sim U[0,1]$ and then applying Equation (A.5) to it. In that case, $1-U$ is also distributed according to $U[0,1]$, so we can use it instead. Therefore, sampling $E^{(s)}$ given $\sum_{s'=1}^{n} E^{(s')} = R$ is the same as sampling $U \sim U[0,1]$, then doing:

$$E^{(s)} = -RU^{\frac{1}{n-1}} + R. \tag{A.6}$$

The iterative procedure outlined in Algorithm 6 then consists of the following steps:

1. Sample $\sum_{s'=1}^{n} E^{(s')}$ (Equation (A.3)).
2. Sample $E^{(s)}$ given this sum (Equation (A.6)).
3. Compute the Dirichlet $d^{(s)}$ (Equation (A.2)).
4. Compute uniform $\tilde{u}$ using the marginal Dirichlet CDF (similar to Algorithm 5).
5. Repeat this process for the next iteration, but, instead of sampling $\sum_{s'=1}^{n} E^{(s')}$ again, use R minus the previous $E^{(s)}$, with the caveat that Equation (A.5) has to be adapted to consider one less sample.

The final algorithm, resulting from combining Algorithms 4 to 6 is outlined in Algorithm 7.

Algorithm 7 Iterative ARMS with Dirichlet copulas

▷ Start accumulators:
$\hat{J} \leftarrow 0$;
$\Lambda_{\nabla \log} \leftarrow \mathbf{0}$;
$\Lambda_{J\nabla \log} \leftarrow \mathbf{0}$;
▷ Start variables for iterative Dirichlet sampling:
Sample $\sum_{s'} E_i^{(s')} \sim \text{Gamma}[n, 1]$ for $i \in \{1, \cdots, d\}$
$R_i \leftarrow \sum_{s'} E_i^{(s')}$
▷ Compute correlations:
for $i \in 1 \dots d$ **do**
  if $\theta_i > 0.5$ **then**
    $\rho_i \leftarrow \frac{\max(0, 2(1-\theta_i)^{\frac{1}{n-1}} - 1)^{n-1} - (1-\theta_i)^2}{\theta_i(1-\theta_i)}$
  else
    $\rho_i \leftarrow \frac{\max(0, 2\theta_i^{1/(n-1)} - 1)^{n-1} - \theta_i^2}{\theta_i(1-\theta_i)}$
▷ Main loop:
for $s \in 1 \dots n$ **do**
  ▷ The loop below can be parallelized with vectorized implementations
  for $i \in 1 \dots d$ **do**
    if $s < n$ **then**
      Sample $U \sim U[0, 1]$
      $E_i^{(s)} \leftarrow -R_i U^{\frac{1}{n-s}} + R_i$
    else
      $E_i^{(s)} \leftarrow R_i$
    $R_i \leftarrow R_i - E_i^{(s)}$
    ▷ Get single Dirichlet r.v. (PS: $\sum_{s'} E_i^{(s')}$ was already computed)
    $d_i^{(s)} \leftarrow \frac{E_i^{(s)}}{\sum_{s'} E_i^{(s')}}$
    $\tilde{u}_i^{(s)} \leftarrow 1 - (1 - d_i^{(s)})^{n-1}$ ▷ Apply marginal CDF
    if $\theta_i > 0.5$ **then** ▷ Additional steps from ARMS paper
      $\tilde{z}_i^{(s)} \leftarrow \mathbb{1}[\tilde{u}_i^{(s)} \leq \theta_i]$
    else
      $\tilde{z}_i^{(s)} \leftarrow \mathbb{1}[1 - \tilde{u}_i^{(s)} \leq \theta_i]$
  ▷ Accumulate values, similarly to LOORF
  $\hat{J} \leftarrow \frac{s-1}{s}\hat{J} + \frac{1}{s} J(\tilde{\mathbf{z}}^{(s)})$
  $\Lambda_{\nabla \log} \leftarrow \frac{\max\ (s-2,1)}{\max\ (s-1,1)} \Lambda_{\nabla \log} + \frac{1}{\max\ (s-1,1)} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tilde{\mathbf{z}}^{(s)})$
  $\Lambda_{J\nabla \log} \leftarrow \frac{\max\ (s-2,1)}{\max\ (s-1,1)} \Lambda_{J\nabla \log} + \frac{1}{\max\ (s-1,1)} J(\tilde{\mathbf{z}}^{(s)}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tilde{\mathbf{z}}^{(s)})$
$\hat{\boldsymbol{g}} \leftarrow \frac{1}{1-\boldsymbol{\rho}} \left(\Lambda_{J\nabla \log} - \Lambda_{\nabla \log} \hat{J} \right)$
Return: $\hat{\boldsymbol{g}}$

A.3 Beta*

Theorem 3. *Consider the problem of estimating:*

$$\frac{\partial}{\partial \theta_i} \mathop{\mathbb{E}}_{\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot)} \left[J(\boldsymbol{z}) \right] = \mathop{\mathbb{E}}_{\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot)} \left[\frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{z})}{\partial \theta_i} J(\boldsymbol{z}) \right]$$

by sampling $\boldsymbol{z}^{(s)} \overset{iid}{\sim} p_{\boldsymbol{\theta}}(\cdot)$*, for* $s \in \{1, \ldots, n\}$ *and using:*

$$\hat{g}_{\beta_i}((\boldsymbol{z}^{(s)})_{s=1}^n; \boldsymbol{\theta}) = \frac{1}{n} \sum_{s=1}^{n} \Big(J(\boldsymbol{z}^{(s)}) - \beta_i \Big) \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{z}^{(s)})}{\partial \theta_i}. \tag{4.17}$$

The optimal β_i^* *such that*

$$\beta_i^* = \arg\min_{\beta_i \in \mathbb{R}} \operatorname{Var}\left[\hat{g}_{\beta_i}((\boldsymbol{z}^{(s)})_{s=1}^n; \boldsymbol{\theta}) \right]$$

is given by

$$\beta_i^* = \mathop{\mathbb{E}}_{\boldsymbol{z} \sim q_i(\cdot; \boldsymbol{\theta})} \left[J(\boldsymbol{z}) \right], \qquad \text{where } q_i(\cdot; \boldsymbol{\theta}) \propto p_{\boldsymbol{\theta}}(\cdot) \left(\frac{\partial \log p_{\boldsymbol{\theta}}(\cdot)}{\partial \theta_i} \right)^2.$$

Proof For arbitrary $f(\cdot)$ and $d = 1$, consider estimating $\mathbb{E}[f(z)]$ by using control variates as follows:

$$\mathbb{E}\left[f(z) \right] \approx \left(\frac{1}{n} \sum_{s=1}^{n} f(z^{(s)}) - \beta h(z^{(s)}) \right) + \beta \mathbb{E}\left[h(z) \right]. \tag{A.7}$$

From Owen (2013, section 8.9), the optimal β is given by

$$\beta^* = \frac{\operatorname{Cov}\left[f(z), h(z) \right]}{\operatorname{Var}\left[h(z) \right]}.$$

Estimation from Equation (4.17) is equivalent to Equation (A.7) for the following choices:

$$f(z) = \frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i} J(\mathbf{z}) \quad \text{and} \quad h(z) = \frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i}.$$

We have:

$$\begin{aligned}
\beta^* &= \frac{\operatorname{Cov}\left[\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i} J(\mathbf{z}), \frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i}\right]}{\operatorname{Var}\left[\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i}\right]} \\
&= \frac{\mathbb{E}\left[\left(\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i}\right)^2 J(\mathbf{z})\right]}{\mathbb{E}\left[\left(\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i}\right)^2\right]} && \triangleright \text{ Since } \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)}\left[\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i}\right] = 0 \\
&= \frac{\int p_{\boldsymbol{\theta}}(\mathbf{z})\left(\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i}\right)^2 J(\mathbf{z})\mathrm{d}\mathbf{z}}{\int p_{\boldsymbol{\theta}}(\mathbf{z}')\left(\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z}')}{\partial \theta_i}\right)^2 \mathrm{d}\mathbf{z}'} \\
&= \int \left(\frac{p_{\boldsymbol{\theta}}(\mathbf{z})\left(\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i}\right)^2}{\int p_{\boldsymbol{\theta}}(\mathbf{z}')\left(\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z}')}{\partial \theta_i}\right)^2 \mathrm{d}\mathbf{z}'}\right) J(\mathbf{z})\mathrm{d}\mathbf{z} \\
&= \mathop{\mathbb{E}}_{\mathbf{z}\sim q_i(\cdot;\boldsymbol{\theta})}\left[J(\mathbf{z})\right].
\end{aligned}$$

Where:

$$\begin{aligned}
q_i(\mathbf{z};\boldsymbol{\theta}) &= \frac{p_{\boldsymbol{\theta}}(\mathbf{z})\left(\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i}\right)^2}{\int p_{\boldsymbol{\theta}}(\mathbf{z}')\left(\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z}')}{\partial \theta_i}\right)^2 \mathrm{d}\mathbf{z}'} \\
&\propto p_{\boldsymbol{\theta}}(\mathbf{z})\left(\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i}\right)^2.
\end{aligned}$$

■

Theorem 4. *For the same setting as in Theorem 3, but with the additional condition that* $p_{\boldsymbol{\theta}}(\cdot) = \prod_{i=1}^d \operatorname{Ber}[\theta_i]$*, the optimal* β_i^* *becomes:*

$$\beta_i^* = \mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_{\boldsymbol{\theta}}(\cdot)}\left[J(z_1, \ldots, z_{i-1}, 1 - z_i, z_{i+1}, \ldots)\right].$$

Proof Since we have $p_{\boldsymbol{\theta}}(\mathbf{z}) = \prod_{i=1}^d p_{\theta_i}(z_i)$, $q_i(\mathbf{z};\boldsymbol{\theta})$ becomes:

$$\begin{aligned}
q_i(\mathbf{z};\boldsymbol{\theta}) &= \frac{p_{\boldsymbol{\theta}}(\mathbf{z})\left(\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i}\right)^2}{\int p_{\boldsymbol{\theta}}(\mathbf{z}')\left(\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{z}')}{\partial \theta_i}\right)^2 \mathrm{d}\mathbf{z}'} \\
&= \frac{p_{\boldsymbol{\theta}}(\mathbf{z})\left(\frac{\partial \log p_{\boldsymbol{\theta}}(z_i)}{\partial \theta_i}\right)^2}{\int p_{\boldsymbol{\theta}}(z_i')\left(\frac{\partial \log p_{\boldsymbol{\theta}}(z_i')}{\partial \theta_i}\right)^2 \mathrm{d}z_i'}.
\end{aligned}$$

Which, for $p_{\boldsymbol{\theta}}(\cdot) = \prod_{i=1}^{d} \mathrm{Ber}[\theta_i]$, becomes:

$$\begin{aligned} q_i(\mathbf{z}; \boldsymbol{\theta}) &= \prod_{j \neq i} p_{\theta_j}(z_j) \frac{\frac{1}{p_{\theta_i}(z_i)}}{\frac{1}{p_{\theta_i}(z_i)} + \frac{1}{1 - p_{\theta_i}(z_i)}} \\ &= \left(\prod_{j \neq i} p_{\theta_j}(z_j) \right) (1 - p_{\theta_i}(z_i)). \end{aligned}$$

Plugging this in $\mathbb{E}_{\mathbf{z} \sim q_i(\cdot; \boldsymbol{\theta})}[J(\mathbf{z})]$ and using $\mathbf{z}_{\setminus i} \sim p_{\boldsymbol{\theta}}(\cdot)$ as shorthand for $z_j \sim p_{\theta_j}(\cdot)$ for $j \neq i$:

$$\begin{aligned} \beta_i^* &= \mathop{\mathbb{E}}_{\substack{\mathbf{z}_{\setminus i} \sim p_{\boldsymbol{\theta}}(\cdot) \\ z_i \sim 1 - p_{\theta_i}(\cdot)}} [J(\mathbf{z})] \\ &= \mathop{\mathbb{E}}_{\mathbf{z} \sim p_{\boldsymbol{\theta}}(\cdot)} \left[J(z_1, \ldots, z_{i-1}, 1 - z_i, z_{i+1}, \ldots) \right], \end{aligned}$$

where the last step follows from the fact that, in general, we can sample $b' \sim \mathrm{Ber}[1 - \theta]$ as $b' = 1 - b$, for $b \sim \mathrm{Ber}[\theta]$. ■

A.4 Generalization of stochastic formulation

Theorem 5. *For some arbitrary $\boldsymbol{z}^*$ in $\{0,1\}^d$, define $J : \{0,1\}^d \to \mathbb{R}$ as:*

$$J(\boldsymbol{z}) = \begin{cases} m & \text{if } \boldsymbol{z} = \boldsymbol{z}^* \\ M_0 - d_H(\boldsymbol{z}, \boldsymbol{z}^*) \dfrac{\Delta M}{d} & \text{otherwise} \end{cases},$$

where $M_0 \in \mathbb{R}$, $\Delta M \in \mathbb{R}_{>0}$ and $m = \min_{\boldsymbol{z} \in \{0,1\}^d} J(\boldsymbol{z})$ is unique. Particularly, this is only satisfied if:

$$m < M_0 - \Delta M,$$

where $J(\mathbf{1} - \boldsymbol{z}^*) = M_0 - \Delta M$ is the second lowest value. For any arbitrary dimension i and assuming $p_{\boldsymbol{\theta}}(\cdot)$ is a factorized Bernoulli with parameter $\boldsymbol{\theta}$, we have:

$$\begin{aligned} & \left(-\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot)} [J(\boldsymbol{z})] \right)_i (\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{z}^*))_i \geq 0 \qquad (4.20) \\ \iff \quad & m \leq M_0 - \frac{\Delta M}{d \prod_{j \neq i} p_{\theta_j}(z_j^*)}. \end{aligned}$$

Proof As mentioned in the main text, $J(\cdot)$ is bounded to the interval $[m, M_0)$, regardless of d . Since $p_{\theta_i}(z_i)$ is a factorized Bernoulli, we have:

$$p_{\theta_i}(z_i) = \begin{cases} \theta_i & \text{if } z_i = 1 \\ 1 - \theta_i & \text{if } z_i = 0 \end{cases} \quad \text{and} \quad p_{\boldsymbol{\theta}}(\mathbf{z}) = \prod_{i=1}^{d} p_{\theta_i}(z_i).$$

We also have:

$$\frac{\partial p_{\boldsymbol{\theta}}(\mathbf{z})}{\partial \theta_i} = (2z_i - 1) \prod_{j \neq i} p_{\theta_j}(z_j)$$

and the direction that increases $p_{\boldsymbol{\theta}}(\mathbf{z}^*)$ at the i-th coordinate is:

$$\frac{\partial p_{\boldsymbol{\theta}}(\mathbf{z}^*)}{\partial \theta_i} = (2z_i^* - 1) \prod_{j \neq i} p_{\theta_j}(z_j^*).$$

Whereas the negative of the true gradient at the i-th coordinate is:

$$\begin{aligned} -\frac{\partial}{\partial \theta_i} \mathop{\mathbb{E}}_{\mathbf{z} \sim p_{\boldsymbol{\theta}}(\cdot)} [J(\mathbf{z})] &= -\sum_{h=0}^{2^d-1} \frac{\partial p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_h)}{\partial \theta_i} J(\boldsymbol{\zeta}_h) \\ &= -\sum_{h=0}^{2^d-1} \left((2(\boldsymbol{\zeta}_h)_i - 1) \prod_{j \neq i} p_{\theta_j}((\boldsymbol{\zeta}_h)_j) \right) J(\boldsymbol{\zeta}_h). \end{aligned}$$

Therefore, the i-th coordinate of $-\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z} \sim p_{\boldsymbol{\theta}}(\cdot)} J(\mathbf{z})$ and that of $\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z}^*)$ will point in the same direction if the following is ≥ 0:

$$\frac{\partial p_{\boldsymbol{\theta}}(\mathbf{z}^*)}{\partial \theta_i} \left(-\frac{\partial}{\partial \theta_i} \mathop{\mathbb{E}}_{\mathbf{z} \sim p_{\boldsymbol{\theta}}(\cdot)} [J(\mathbf{z})] \right) = (2z_i^* - 1) \prod_{j \neq i} p_{\theta_j}(z_j^*) \left(-\sum_{h=0}^{2^d-1} \left((2(\boldsymbol{\zeta}_h)_i - 1) \prod_{j \neq i} p_{\theta_j}((\boldsymbol{\zeta}_h)_j) \right) J(\boldsymbol{\zeta}_h) \right).$$

Since $\prod_{j \neq i} p_{\theta_j}(z_j^*)$ does not change the sign of the expression, we will ignore it. The remaining terms can be written as:

$$\begin{aligned} & (2z_i^* - 1) \left(-\sum_{h=0}^{2^d-1} \left((2(\boldsymbol{\zeta}_h)_i - 1) \prod_{j \neq i} p_{\theta_j}((\boldsymbol{\zeta}_h)_j) \right) J(\boldsymbol{\zeta}_h) \right) \\ = & (2z_i^* - 1) \left(-\sum_{z_i=0}^{1} (2z_i - 1) \mathbb{E}_{\{z_j \sim p_{\theta_j}(\cdot)\}_{j \neq i}} [J(\mathbf{z})] \right) \\ = & -\sum_{z_i=0}^{1} (-1)^{\mathbb{1}[z_i \neq z_i^*]} \mathbb{E}_{\{z_j \sim p_{\theta_j}(\cdot)\}_{j \neq i}} [J(\mathbf{z})] \end{aligned}$$

$$
\begin{aligned}
&= -\Big(\mathbb{E}_{\{z_j\sim p_{\theta_j}(\cdot)\}_{j\neq i}}[J(z_1,\dots,z_{i-1},z_i^*,z_{i+1},\dots)] - \\
&\qquad\qquad \mathbb{E}_{\{z_j\sim p_{\theta_j}(\cdot)\}_{j\neq i}}[J(z_1,\dots,z_{i-1},1-z_i^*,z_{i+1},\dots)]\Big) \\
&= -\mathbb{E}_{\{z_j\sim p_{\theta_j}(\cdot)\}_{j\neq i}}\big[J(z_1,\dots,z_{i-1},z_i^*,z_{i+1},\dots) - \\
&\qquad\qquad J(z_1,\dots,z_{i-1},1-z_i^*,z_{i+1},\dots)\big].
\end{aligned}
\tag{A.8}
$$

Note that the following implication is true for the given $J(\cdot)$:

$$
\begin{aligned}
(d_H(\mathbf{z},\mathbf{z}')=1)\wedge(d_H(\mathbf{z},\mathbf{z}^*)-d_H(\mathbf{z}',\mathbf{z}^*)=-1)\Rightarrow \\
\left((\mathbf{z}=\mathbf{z}^*)\wedge\left(J(\mathbf{z})-J(\mathbf{z}')=m-\left(M_0-\frac{\Delta M}{d}\right)\right)\right)\vee \\
\left((\mathbf{z}\neq\mathbf{z}^*)\wedge\left(J(\mathbf{z})-J(\mathbf{z}')=\frac{\Delta M}{d}\right)\right)
\end{aligned}
\tag{A.9}
$$

and the left statement from Equation (A.9) is satisfied for all pairs inside the expectation from the RHS of Equation (A.8). Therefore Equation (A.8) becomes:

$$
\begin{aligned}
&-\mathbb{E}_{\{z_j\sim p_{\theta_j}(\cdot)\}_{j\neq i}}\big[J(z_1,\dots,z_{i-1},z_i^*,z_{i+1},\dots) - \\
&\qquad\qquad J(z_1,\dots,z_{i-1},1-z_i^*,z_{i+1},\dots)\big] \\
&= -\left(\prod_{j\neq i}p_{\theta_j}(z_j^*)\left(m-\left(M_0-\frac{\Delta M}{d}\right)\right)+\left(1-\prod_{j\neq i}p_{\theta_j}(z_j^*)\right)\frac{\Delta M}{d}\right).
\end{aligned}
$$

Finally, after some algebraic manipulation:

$$
\begin{aligned}
&-\left(\prod_{j\neq i}p_{\theta_j}(z_j^*)\left(m-\left(M_0-\frac{\Delta M}{d}\right)\right)+\left(1-\prod_{j\neq i}p_{\theta_j}(z_j^*)\right)\frac{\Delta M}{d}\right)\geq 0 \\
\iff\ & m\leq M_0-\frac{\Delta M}{d\prod_{j\neq i}p_{\theta_j}(z_j^*)}. \qquad \triangleright \text{Assuming } \prod_{j\neq i}p_{\theta_j}(z_j^*)\neq 0
\end{aligned}
$$

■

Appendix B

Experimental details

B.1 Microworlds

B.1.1 Variance experiments

We write the following in terms of $\boldsymbol{\theta}$ instead of $\mathbf{r}$ to avoid clutter. First, we note that none of the estimators used here require $\nabla_{\mathbf{z}} J(\mathbf{z})$, so we can simply compute all $J(\boldsymbol{\zeta}_h)$ values and store them in a table to speed up computations. The variance of the gradient estimators is given by:

$$\begin{aligned}\operatorname{Var}\left[\hat{\boldsymbol{g}}((\mathbf{z}^{(s)})_{s=1}^{n};\boldsymbol{\theta})\right] &= \mathbb{E}\left[\hat{\boldsymbol{g}}((\mathbf{z}^{(s)})_{s=1}^{n};\boldsymbol{\theta})^2\right] - \mathbb{E}\left[\hat{\boldsymbol{g}}(\mathbf{z}^{(s)})_{s=1}^{n};\boldsymbol{\theta})\right]^2 \\ &= \mathbb{E}\left[\hat{\boldsymbol{g}}((\mathbf{z}^{(s)})_{s=1}^{n};\boldsymbol{\theta})^2\right] - \left(\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\cdot)}\left[J(\mathbf{z})\right]\right)^2. \end{aligned} \tag{B.1}$$

For these smaller scale experiments, computing the right summand can simply be done by using Equation (4.19), which requires 2^d evaluations. Here, this is feasible to compute, but not so much for the left summand, which is a sum of 2^{dn} terms. To reduce its complexity, we leverage the structure of the expressions involved. Notably, $\hat{\boldsymbol{g}}(\cdot)$ is order invariant with respect to $\mathbf{z}^{(s)}$ in all the estimators we use. Therefore, we only consider the combinations of n elements from $\{\boldsymbol{\zeta}_h\}_{h=0}^{2^d-1}$ instead of all possible n-tuples. Since each of them is an n-combination of 2^d elements with repetition, the number of different values $\hat{\boldsymbol{g}}(\cdot)$ can take is given by:

$$\left(\!\binom{2^d}{n}\!\right) = \binom{2^d - 1 + n}{n}. \tag{B.2}$$

Therefore, to compute the left summand of Equation (B.1), we need to evaluate $\hat{\boldsymbol{g}}(\cdot)^2$ only in these combinations, then multiply each evaluation by the probability of the respective combination and sum all of these terms. The probability of a combination will be the sum of the probabilities of all the permutations corresponding to it.

Since, as mentioned before, these combinations may have repetition, each of them corresponds to a set that can have repeated elements, sometimes called multisets. The number of times each value appear in the multiset is called its multiplicity. Considering a multiset $\mathcal{M}$ and denoting as m_h the multiplicity of $\boldsymbol{\zeta}_h$, where $m_h \geq 0$ and $\sum_{h=0}^{2^d-1} m_h = n$, we have that the number of n-permutations of $\mathcal{M}$ is given by:

$$\binom{n}{m_0, \ldots, m_{2^d-1}} = \frac{n!}{m_0! \ldots m_{2^d-1}!}. \tag{B.3}$$

For REINFORCE, LOORF, β^*, the n samples are iid, therefore, the probability of an arbitrary set of n samples is simply:

$$p((\mathbf{z}^{(s)})_{s=1}^n; \boldsymbol{\theta}) = \prod_{s=1}^{n} p_{\boldsymbol{\theta}}(\mathbf{z}^{(s)}). \tag{B.4}$$

Which will be the same for all permutations corresponding to each multiset. To conclude, the left summand from Equation (B.1) can be computed by iterating the combinations from Equation (B.2) and, for each of them, computing the quantity from Equation (B.4), multiplying by the one in Equation (B.3) and by $\hat{\boldsymbol{g}}(\cdot)^2$, then summing all of these results. For $d = 4$ and $n = 4$, this reduces the number of per-step evaluations from $65,536$ to $3,876$ when calculating $\mathbb{E}[\hat{\boldsymbol{g}}(\cdot)^2]$.

For ARMS, however, sampling follows *Algorithm* 7, which introduces dependence between different $\mathbf{z}^{(s)}$, causing Equation (B.4) to be no longer valid. To show how we calculate the new probabilities, we use $d = 1$, $n = 2$ and $(\tilde{z}^{(1)}, \tilde{z}^{(2)}) = (1, 0)$ as an example. The algorithm has two cases: $\theta > 0.5$, where the Dirchlet copula $(\tilde{u}^{(s)})_{s=1}^n$ is used; $\theta \leq 0.5$, where $(1 - \tilde{u}^{(s)})_{s=1}^n$ is used. For the first case, we have:

$$\begin{aligned} p(\tilde{z}^{(1)} = 1, \tilde{z}^{(2)} = 0; \theta) &= p(\tilde{u}^{(1)} < \theta, \tilde{u}^{(2)} > \theta) \\ &= p(d^{(1)} < F^{-1}(\theta), d^{(2)} > F^{-1}(\theta)), \end{aligned} \tag{B.5}$$

where $F^{-1}(\cdot)$ is the inverse of the marginal CDF of the Dirichlet distribution used to get the copula in Algorithm 7:

$$F^{-1}(\theta) = 1 - (1-\theta)^{1/(n-1)}.$$

This function is monotonically increasing in $[0, 1]$. By the law of total probability, Equation (B.5) corresponds to:

$$\int p(d^{(1)}, d^{(2)}) \mathbb{1}[(d^{(1)} < F^{-1}(\theta))] \mathbb{1}[(d^{(2)} > F^{-1}(\theta))] \mathrm{d}d^{(1)} \mathrm{d}d^{(2)}. \tag{B.6}$$

We can input this integral to a symbolic equation solver, such as Mathematica, by using the PDF of the Dirichlet and multivariate integration, yielding a closed form expression in terms of θ.

To generalize the above (still on $d = 1$ for now), we note that these steps are also order invariant with respect to $\tilde{z}^{(s)}$. Therefore, the number of possible integrals such as the one in Equation (B.6) is simply the number of n-combinations of $\{0, 1\}$ with repetition, given by:

$$\left(\!\binom{2}{n}\!\right) = \binom{2-1+n}{n} = n + 1.$$

Once θ_t is known, we can compute these $n + 1$ values in closed-form and store them in a table to then be repeatedly consulted for each possible set $\{\tilde{z}^{(1)}, \ldots, \tilde{z}^{(n)}\}$ when calculating the expectation $\mathbb{E}[\hat{g}_{ARMS}(\cdot)^2]$ (taking the place of Equation (B.4)), since they will not change until θ_t changes. In the second case, where $\theta \leq 0.5$, ARMS uses a different copula, so the above steps can be slightly changed to:

$$\begin{aligned} p(\tilde{z}^{(1)} = 1, \tilde{z}^{(2)} = 0; \theta) &= p(1 - \tilde{u}^{(1)} < \theta, 1 - \tilde{u}^{(2)} > \theta) \\ &= p(\tilde{u}^{(1)} > 1 - \theta, \tilde{u}^{(2)} < 1 - \theta) \\ &= p(d^{(1)} > F^{-1}(1-\theta), d^{(2)} < F^{-1}(1-\theta)). \end{aligned}$$

θ is simply exchanged by $1 - \theta$ when computing the table and the inequalities change sides when consulting it. In other words, it is the same result as running the previous steps, but using $1 - \theta$ instead of θ and $(\tilde{z}^{(1)}, \tilde{z}^{(2)}) = (0, 1)$ instead of $(1, 0)$.

Multiset	$\int p((d^{(s)})_{s=1}^n) \prod_{s=1}^n \mathbb{1}[(d^{(s)} < F^{-1}(\theta)) \oplus (\tilde{z}^{(s)} = 0)] \mathrm{d}d^{(s)}$
$\{0,0,0,0\}$	$f(\cdot;\theta) = \begin{cases} -(4\theta-1)^3 & 0 \leq \theta < \frac{1}{4} \\ 0 & \text{otherwise} \end{cases}$
$\{0,0,0,1\}$	$f(\cdot;\theta) = \begin{cases} \theta\,(37\theta^2 - 21\theta + 3) & 0 \leq \theta < \frac{1}{4} \\ -(3\theta-1)^3 & \frac{1}{4} \leq \theta < \frac{1}{3} \\ 0 & \text{otherwise} \end{cases}$
$\{0,0,1,1\}$	$f(\cdot;\theta) = \begin{cases} 6(1-3\theta)\theta^2 & 0 \leq \theta < \frac{1}{4} \\ 46\theta^3 - 42\theta^2 + 12\theta - 1 & \frac{1}{4} \leq \theta < \frac{1}{3} \\ -(2\theta-1)^3 & \frac{1}{3} \leq \theta < \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$
$\{0,1,1,1\}$	$f(\cdot;\theta) = \begin{cases} 6\theta^3 & 0 \leq \theta < \frac{1}{4} \\ -58\theta^3 + 48\theta^2 - 12\theta + 1 & \frac{1}{4} \leq \theta < \frac{1}{3} \\ 23\theta^3 - 33\theta^2 + 15\theta - 2 & \frac{1}{3} \leq \theta < \frac{1}{2} \\ -(\theta-1)^3 & \frac{1}{2} \leq \theta \leq 1 \end{cases}$
$\{1,1,1,1\}$	$f(\cdot;\theta) = \begin{cases} 0 & 0 \leq \theta < \frac{1}{4} \\ (4\theta-1)^3 & \frac{1}{4} \leq \theta < \frac{1}{3} \\ -44\theta^3 + 60\theta^2 - 24\theta + 3 & \frac{1}{3} \leq \theta < \frac{1}{2} \\ 4\theta^3 - 12\theta^2 + 12\theta - 3 & \frac{1}{2} \leq \theta \leq 1 \end{cases}$

Table B.1: Closed form expressions for probabilities of $\{\tilde{z}^{(s)}\}_{s=1}^n$, for $n = 1$ and $d = 4$. If $\theta > 0.5$, $p(\{\tilde{z}^{(s)}\}_{s=1}^n;\theta) = f(\{\tilde{z}^{(s)}\}_{s=1}^n;\theta)$, otherwise $p(\{\tilde{z}^{(s)}\}_{s=1}^n;\theta) = f(\{1-\tilde{z}^{(s)}\}_{s=1}^n;1-\theta)$.

For $d > 1$, independence permits per-dimension parallelization of these steps. The closed-form integral solutions will be the same across different $i \in \{1,\ldots,d\}$, requiring only changing θ for θ_i. To allow reproducibility, Table B.1 gives closed-form expressions for integrals like the one in Equation (B.6) for the case where $n = 4$ and $d = 1$, where we denote $\oplus$ to be the exclusive disjunction (i.e. XOR).

For the other experiments, where $n = 10$ and $d = 10$, we use estimated variance rather than closed forms. This simply amounts to changing the expectation in the left summand from Equation (B.1) for a Monte Carlo estimate, where we computed the mean over $10,000$ evaluations of $\hat{\boldsymbol{g}}(\cdot)^2$ for each timestep, the right summand is still used in closed-form.

B.1.2 Comparing methods

Whenever possible, we follow the recommendations from the authors, either directly from the papers or at least from the provided code. For REBAR, we initialized $\log \tau$ to 0.5 and η to 1, whereas for RELAX we also initialized $\log \tau$ to 0.5, but the method does not use η. We train these parameters using Adam and the same learning rate as $\mathbf{r}$. Additionally, the RELAX auxiliary neural network had 1 hidden layer of size 10 and used tanh activations. We trained it using Adam, with learning rate 1 and weight decay 0.001. For CP, the temperature starts at one and follows an exponential schedule, being updated every 100 iterations until arriving at the final value of $1/200$. We naturally show the loss for $\mathbb{1}[\cdot]$ instead of $\sigma(\cdot/\tau)$ when reporting its results. Moreover, we ran CP without learning rate decay, as it hindered performance.

B.2 MaskedNNRegression

The backbone network contained four fully connected hidden layers of size 50 with the linear operator followed by "batch norm" and then LeakyReLU. We did not normalize outputs from the last layer. Weights used Xavier normal initialization.

The target network, on the other hand, had a more complex design, consisting of five fully connected hidden layers of size 500 with the same sequence of per-layer operations as above. Here, we normalized outputs from the final layer to $[0, 1]$ by using the corresponding maximum and minimum from the training dataset. We initialized weights to either -1 or 1 with 50% chance.

To avoid adding trainable variables, we used batch normalization without affine parameters. We also excluded moving statistics from the implementation. Although Ioffe and Szegedy (2015) mention that this form can reduce the expressiveness of the unnormalized layer, we observed major improvements when including it. The trained models had much lower error for the same targets and the target NN could generate much more complex mappings.[1] When

[1] We confirmed this by experimenting with one dimensional inputs and plotting the target maps.

constructing the datasets via forward-passes, we inputted each of them in its entirety, producing good average statistics for the normalization. Although we generated them separately, our comparisons to joint generation indicated this did not impact the results.

The training dataset consisted of $10,000$ samples and we used batch size 100, whereas the validation dataset consisted of $5,000$ samples, which were input at once when validating. Reported results correspond to validation data, as we saw no need to generate more data for testing. When validating MC methods, we sample a batch of 500 data points and compare the average loss over five random masks, selecting the best one.

B.3 Pruning

Similarly to H. Zhou *et al.* (2019) we apply dynamic weight rescaling to CP and MC, with division of the weights by the mean of corresponding layer masks during forward passes. This quotient is treated as a constant in the backward pass. For CP, we consider the mean of the soft-mask during training iterations, which will eventually correspond to the hard mask as $\tau \to 0$. Hinton *et al.* (2012) also used DWR in the Dropout paper. We noticed that methods tend to prune too aggressively without this addition. Although not using it also leads to good results, sensitivity to λ gets higher.

When training stochastic masks with SGD, H. Zhou *et al.* (2019) had to resort to unusual learning rates such as 20 or 100. We did not observe this issue with RMSprop.

Our implementation of CP is very similar to Savarese *et al.* (2020), the main difference being the initialization and the λ sweeps. They noted in their experiments that setting λ close to zero and relying on tuning the mask initialization instead seemed to yield better results. Particularly, all else maintained, a lower mask initialization leads to sparser final networks. Since this is an empirical trick, rather than a fundamental part of the method, we chose to initialize $\boldsymbol{\theta}_0 = [0.5, \ldots, 0.5]^\top$ and rely on tuning lambda instead. Although this led to worse CP results, we believe it is more fair to MC methods, which did

not benefit from similar pruning-focused algorithmic tuning. Moreover, this initialization is also more consistent with the PB optimization discussion

In all experiments, we divided datasets in training, validation and test, with the test sets following the default split from CIFAR-10 and MNIST, while the validation sets consisted of $5,000$ random samples.

B.3.1 Supermask

Method	Parameter	Values
Shared	Batch size	$\{128\}$
	Epochs	$\{200\}$
	Optimizer ($\mathbf{r}$)	{RMSprop}
	$\boldsymbol{\theta}_0 = \theta(\mathbf{r}_0)$	$\{[0.5, \ldots, 0.5]^\top\}$
	Random seed	$\{0, 1, 2, 3, 4\}$
MC	Parametrization	{Escort, Sigmoid}
	Estimator	{ARMS, LOORF}
	Learning rate ($\mathbf{r}$)	{0.1, 0.01, 0.001}
	Learning rate schedule ($\mathbf{r}$) (% of training)	$\{[60\%]\}$
	Learning rate schedule ($\mathbf{r}$) (multipliers)	$\{[0.5]\}$
	$L0$-regularization (λ)	$\{1\text{E}-1, 5\text{E}-2, \ldots, 1\text{E}-5\}$
	n	$\{2, 10, 100\}$
CP	Learning rate ($\mathbf{r}$)	{0.1, 0.01, 0.001}
	Learning rate schedule ($\mathbf{r}$) (% of training)	{[40%, 60%]}
	Learning rate schedule ($\mathbf{r}$) (multipliers)	{[0.1, 0.1]}
	$L0$-regularization (λ)	$\{1\text{E}-1, 5\text{E}-2, \ldots, 1\text{E}-5\}$

Table B.2: Supermask hyperparameter sweep.

Table B.3 summarizes the hyperparameter sweep used. As mentioned before, sigmoid and LOORF had similar, but slightly worse results than escort and ARMS, so we only include these last two in the results. Remaining details for architectures and datasets are as described in H. Zhou *et al.* (2019, section S1), where they refer to Lenet as MNIST-FC.

B.4 Hyperparameter generalization

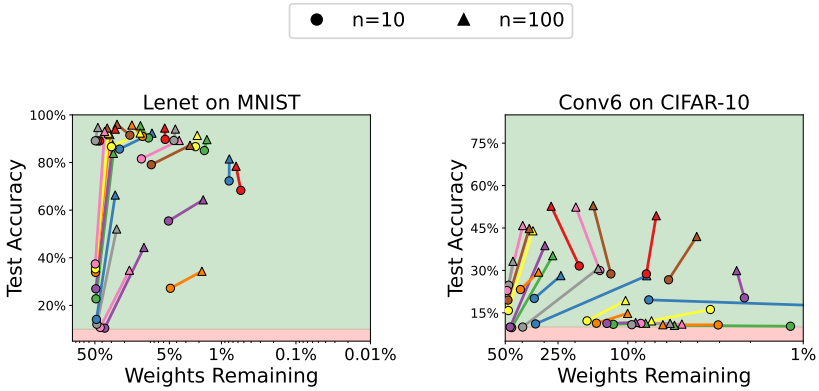
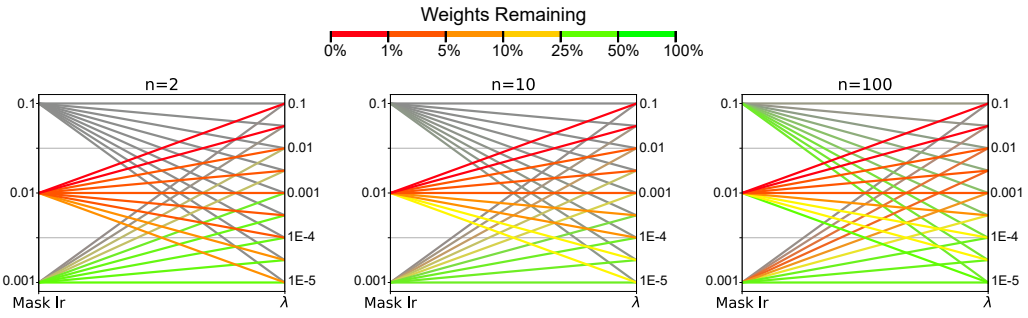


Figure B.1: Scatter plot for hyperparameter generalization on supermask results.

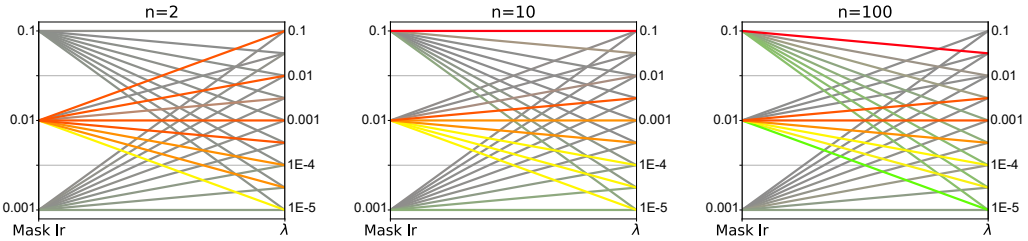
As mentioned before, we study the possibility of re-using the best hyperparameters for $n = 10$ on $n = 100$ by looking retroactively at the results from supermask experiments.[2] Figure B.1 shows a scatter plot, where each point is the average result across seeds. Red regions indicate accuracy akin to chance and runs with the same hyperparameter settings are linked and colored the same. On the majority of cases, runs became sparser or retained similar sparsity, but achieved higher accuracy. The main exception were the sparser runs on the red regions, but those are clearly failed runs.

Figure B.2 shows an alternative view of the same data, where each setting corresponds to a line. Lower saturation (i.e. regions closer to gray) are further from the Pareto front, whereas the more colored regions have better accuracies for their respective sparsity. Hue represents weights remaining, with red regions being sparse and green regions being dense. Overall, we note an agreement across different n, with similar hyperparameter combinations leading to similar sparsity and proximity to Pareto front

[2]For escort and ARMS only, although we saw similar results with sigmoid and LOORF.



(a) Lenet on MNIST



(b) Conv6 on CIFAR-10

Figure B.2: Parallel coordinates plot for hyperparameter generalization on supermask results. We omit failed runs.

B.4.1 Joint pruning

Differently from the supermask experiments, in joint pruning we use image augmentation on the training set in the form of random horizontal flips and random crops. Our backbone experimental settings mostly follow the Resnet-20 description from Frankle and Carbin (2018, figure 2) (they call it Resnet-18), except for the learning rate schedule and the total number of epochs, which are based on Savarese *et al.* (2020).

Inspired by Hoefler *et al.* (2021), MC trains the only the dense network in the first few epochs, the joint-training starts later. In the original GMP paper (Zhu and Gupta 2017), they also train this same way. Since CP uses smooth masks, joint training is already easier on the first epochs and there is no need for freezing masks. Similarly to Savarese *et al.* (2020) masks are frozen in the last epochs and only the main weights are fine-tuned.

As mentioned in the text, tentative hyperparameter values for $n = 100$ were based on results for $n = 10$. Our sweep was performed in two stages. First we used a broader range of hyperparameters, which we show in Table B.3. Then,

after analyzing its results, we ran a second "specialized" sweep (Table B.4) to better cover some sparsity ranges. On the main paper, our plots correspond to both sweeps combined.

Method	Parameter	Values
Shared	Batch size	{128}
	Epochs	{200}
	Finetune only (% of training)	{80%}
	Random seed	$\{0, 1, 2, 3, 4\}$
MC	Parametrization	{Escort}
	Estimator	{ARMS}
	Optimizer	{RMSprop}
	$\boldsymbol{\theta}_0 = \theta(\mathbf{r}_0)$	$\{[0.5, \ldots, 0.5]^\top\}$
	Learning rate ($\mathbf{r}$)	{0.1, 0.01, 0.001}
	Learning rate schedule ($\mathbf{r}$) (% of training)	{[60%]}
	Learning rate schedule ($\mathbf{r}$) (multipliers)	{[0.5]}
	$L0$-regularization (λ)	$\{1\text{E}-1, 5\text{E}-2, \ldots, 1\text{E}-5\}$
	n	$\{2, 10\}$
	Start training $\mathbf{r}$ (% of training)	{[10%]}
MC ($n = 100$)	Learning rate ($\mathbf{r}$)	{0.01}
	n	{100}
	(Rest is the same as MC)	
CP	Optimizer	{RMSprop}
	$\boldsymbol{\theta}_0 = \theta(\mathbf{r}_0)$	$\{[0.5, \ldots, 0.5]^\top\}$
	Learning rate ($\mathbf{r}$)	{0.1, 0.01, 0.001}
	Learning rate schedule ($\mathbf{r}$) (% of training)	{[40%, 60%]}
	Learning rate schedule ($\mathbf{r}$) (multipliers)	{[0.1, 0.1]}
	$L0$-regularization (λ)	$\{1\text{E}-1, 5\text{E}-2, \ldots, 1\text{E}-5\}$
GMP	Final weights remaining	{50%, 10%, ..., 0.1%, 0.05%}
	(Rest is the same as Zhu and Gupta (2017))	
MP	Final weights remaining	{50%, 10%, ..., 0.1%, 0.05%}
	Global prune	{True}

Table B.3: Joint pruning broad sweep.

Method	Parameter	Values
MC ($n = 100$)	$L0$-regularization (λ)	$\{0.075, 0.045, 0.04,$ $0.035, 0.03, 0.025,$ $0.02, 0.015, 0.0035,$ $0.0025\}$
CP	Learning rate ($\mathbf{r}$) $L0$-regularization (λ)	$\{0.01\}$ $\{0.7, 0.6, 0.5, 0.4$ $0.3, 0.2, 0.045, 0.035$ $0.025, 0.015\}$
CP	Learning rate ($\mathbf{r}$) $L0$-regularization (λ)	$\{0.1\}$ $\{1.0, 0.9, 0.8, 0.7, 0.6$ $0.5, 0.4, 0.3, 0.2\}$
GMP and MP	Final weights remaining	$\{32\%, 21\%, 16\%, 7.5\%$ $3\%, 2.6\%, 2\%, 1.4\%,$ $1.1\%\}$
GMP only	Final weights remaining	$\{0.45\%, 0.35\%, 0.3\%$ $0.25\%,\ 0.15\%\} \cup$ $\{0.04\%, 0.03\%, 0.02\%$ $0.01\%\}$

Table B.4: Joint pruning "specialized" sweep. Other parameters are the same as Table B.3.

B.5 Summary of results

Table B.5: Summary of all experiments performed

Experiment	Comparison	Summary
Microworlds, Variance ($d \leq 10$)	Estimators	• REINFORCE sometimes had lower variance than ARMS/LOORF • ARMS always had lower variance than LOORF • β^* had significantly lower variance than the others
Microworlds ($d = 10$)	Estimators	• REINFORCE performed the worst • LOORF/ARMS/β^* performed similarly • True gradient failed to reach the correct solution in NNLoss
	Parametrization	• Escort and direct were faster, but converged to worse final solutions • Sigmoid and cosine were slower, but converged to better final solutions
	Approaches	• CP and ST performed the worst • $\nabla_{\mathbf{z}} J(\mathbf{z})$ did not seem to help hybrid methods
NN regression ($d \approx 8,000$)	Estimators	• REINFORCE performed much worse than the others
	Parametrization	• Cosine was still slower, but converged to worse values • Direct was still fast and still converged to poor values • Escort and sigmoid performed relatively well

Continued on next page

Table B.5: Summary of all experiments performed (Continued)

	Approaches	• $\nabla_{\mathbf{z}} J(\mathbf{z})$ started becoming helpful for hybrid methods • ST and LOORF performed the best • CP was still unstable
Supermask ($d \approx 300,000$ and $2,000,000$)	Estimators	• ARMS performed marginally better than LOORF
	Parametrization	• Escort performed marginally better than sigmoid
	Approaches	• CP performed much better than MC, even when $n = 100$
Pruning ($d \approx 300,000$)	Approaches	• MC was highly benefited by joint training • Best method varied by sparsity range • Qualitatively, per-method solutions were very different